This manual describes the operators of the image analysis system HALCON, version 6.0, in COM syntax. It was generated on November 16, 2000.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

Copyright © 1997-2000 by MVTec Software GmbH, München, Germany

More information about HALCON can be found at: http://www.mvtec.com
## Contents

1 Classification .......................... 1
- ClearSampset .......................... 1
- CloseAllClassBox ....................... 1
- CloseClassBox ......................... 2
- CreateClassBox ......................... 2
- DescriptClassBox ...................... 2
- EnquireClassBox ....................... 3
- EnquireRejectClassBox ................. 4
- GetClassBoxParam ...................... 4
- LearnClassBox ......................... 5
- LearnSampsetBox ....................... 6
- ReadClassBox .......................... 7
- ReadSampset .......................... 7
- SetClassBoxParam ...................... 8
- TestSampsetBox ....................... 9
- WriteClassBox ......................... 10

2 File ........................................ 11

2.1 Images ................................... 11
- ReadImage .................................. 11
- ReadSequence ............................ 12
- WriteImage ................................ 14

2.2 Misc ....................................... 15
- FileExists .................................. 15
- ReadWorldFile ............................ 16

2.3 Region ..................................... 16
- ReadRegion ............................... 16
- WriteRegion .............................. 17

2.4 Text ....................................... 18
- CloseAllFiles ............................. 18
- CloseFile .................................. 18
- FnewLine .................................. 19
- FreadChar .................................. 19
- FreadString .............................. 20
- FwriteString ............................. 21
- OpenFile .................................. 21

2.5 Tuple ...................................... 22
- ReadTuple ................................. 22
- WriteTuple ............................... 23

2.6 XLD ....................................... 23
- ReadContourXldArcInfo ................. 23
- ReadPolygonXldArcInfo .................. 24
- WriteContourXldArcInfo ................. 25
- WritePolygonXldArcInfo ............... 26

3 Filter ...................................... 27
3.1 Affine-Transformations ........................................... 27
   AffineTransImage ........................................... 27
   MirrorImage ................................................ 29
   PolarTransImage ........................................... 29
   RotateImage ................................................ 31
   ZoomImageFactor ........................................... 32
   ZoomImageSize ............................................ 33
3.2 Arithmetic .................................................. 34
   AbsImage ................................................ 34
   AddImage ................................................ 34
   DivImage ................................................ 35
   InvertImage ............................................. 36
   MaxImage ................................................. 37
   MinImage ................................................. 38
   Multiply ................................................ 39
   ScaleImage .............................................. 40
   SubImage ................................................ 41
3.3 Bit .......................................................... 42
   BitAnd .................................................. 42
   BitLshift ............................................... 43
   BitMask ................................................ 44
   BitNot .................................................. 44
   BitOr .................................................. 45
   BitRshift ............................................... 46
   BitSlice ............................................... 46
   BitXor ................................................ 47
3.4 Color ........................................................ 48
   Rgb1ToGray .............................................. 48
   Rgb3ToGray .............................................. 49
   TransFromRgb ............................................ 49
   TransToRgb ............................................. 53
3.5 Edges ........................................................ 56
   CloseEdges ............................................ 56
   CloseEdgesLength ......................................... 57
   DerivateGauss ........................................... 58
   DiffOfGauss ............................................. 61
   EdgesImage ............................................. 62
   EdgesSubPix ............................................. 65
   FreiAmp ............................................... 67
   FreiDir ................................................ 68
   HighpassImage ......................................... 69
   InfoEdges .............................................. 70
   KirschAmp .............................................. 71
   KirschDir .............................................. 72
   Laplace ................................................ 73
   LaplaceOfGauss .......................................... 75
   PrewittAmp ............................................. 76
   PrewittDir ............................................. 76
   Roberts ................................................ 78
   RobinsonAmp .......................................... 79
   RobinsonDir ........................................... 80
   SobelAmp ............................................. 81
   SobelDir .............................................. 82
3.6 Enhancement ............................................... 84
   Emphasize ............................................. 84
   EquHistImage .......................................... 85
   Illuminate ............................................. 86
   ScaleImageMax ........................................ 87
### Contents

3.7 FFT ................................................. 88
   ConvolFft ........................................ 88
   ConvolGabor ....................................... 89
   EnergyGabor ....................................... 89
   FitGeneric ......................................... 90
   FitImage ........................................... 91
   FitImageInv ....................................... 92
   GenBandfilter ..................................... 93
   GenBandpass ....................................... 94
   GenFilterMask ..................................... 94
   GenGabor .......................................... 95
   GenHighpass ....................................... 97
   GenLowpass ........................................ 97
   GenSinBandpass ................................... 98
   GenStdBandpass ................................... 99
   PhaseDeg .......................................... 100
   PhaseRad .......................................... 100
   PowerByte ......................................... 101
   PowerLn ........................................... 102
   PowerReal ......................................... 102

3.8 Lines ................................................. 103
   BandpassImage ..................................... 103
   LinesFacet ......................................... 104
   LinesGauss ......................................... 105

3.9 Match ................................................ 108
   AdaptTemplate ...................................... 108
   BestMatch .......................................... 108
   BestMatchMg ....................................... 110
   BestMatchPreMg ................................... 111
   BestMatchRot ....................................... 112
   BestMatchRotMg .................................... 114
   ClearTemplate ....................................... 115
   CornerResponse ..................................... 115
   CreateTemplate ..................................... 117
   CreateTemplateRot .................................. 118
   ExhaustiveMatch .................................... 120
   ExhaustiveMatchMg ................................ 121
   FastMatch ......................................... 123
   FastMatchMg ....................................... 124
   FillDvf ............................................. 125
   GenGaussPyramid ................................... 125
   Monotony ............................................ 126
   OpticalFlowMatch .................................. 127
   ReadTemplate ....................................... 129
   SetOffsetTemplate .................................. 129
   SetReferenceTemplate ................................ 130
   WriteTemplate ...................................... 130

3.10 Misc ................................................. 131
   ConvolImage ........................................ 131
   ExpandDomainGray .................................. 132
   GrayInside ......................................... 133
   GraySkeleton ....................................... 134
   LutTrans ............................................ 134
   PrincipalComp ....................................... 135
   Symmetry ............................................ 136
   TopographicSketch .................................. 137

3.11 Noise ................................................. 138
   AddNoiseDistribution ................................ 138
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddNoiseWhite</td>
<td>139</td>
</tr>
<tr>
<td>GaussDistribution</td>
<td>139</td>
</tr>
<tr>
<td>NoiseDistributionMean</td>
<td>140</td>
</tr>
<tr>
<td>SpDistribution</td>
<td>141</td>
</tr>
<tr>
<td>3.12 Smoothing</td>
<td>142</td>
</tr>
<tr>
<td>AnisotropeDiff</td>
<td>142</td>
</tr>
<tr>
<td>EliminateMinMax</td>
<td>143</td>
</tr>
<tr>
<td>EliminateSp</td>
<td>145</td>
</tr>
<tr>
<td>FillInterface</td>
<td>146</td>
</tr>
<tr>
<td>GaussImage</td>
<td>147</td>
</tr>
<tr>
<td>InfoSmooth</td>
<td>148</td>
</tr>
<tr>
<td>MeanImage</td>
<td>149</td>
</tr>
<tr>
<td>MeanN</td>
<td>150</td>
</tr>
<tr>
<td>MeanSp</td>
<td>151</td>
</tr>
<tr>
<td>MedianImage</td>
<td>152</td>
</tr>
<tr>
<td>MedianSeparate</td>
<td>153</td>
</tr>
<tr>
<td>MedianWeighted</td>
<td>154</td>
</tr>
<tr>
<td>MidrangeImage</td>
<td>155</td>
</tr>
<tr>
<td>RankImage</td>
<td>156</td>
</tr>
<tr>
<td>SigmaImage</td>
<td>158</td>
</tr>
<tr>
<td>SmoothImage</td>
<td>159</td>
</tr>
<tr>
<td>TrimmedMean</td>
<td>160</td>
</tr>
<tr>
<td>3.13 Texture</td>
<td>162</td>
</tr>
<tr>
<td>DeviationImage</td>
<td>162</td>
</tr>
<tr>
<td>EntropyImage</td>
<td>162</td>
</tr>
<tr>
<td>TextureLaws</td>
<td>163</td>
</tr>
<tr>
<td>3.14 Wiener-Filter</td>
<td>165</td>
</tr>
<tr>
<td>GenPsfDefocus</td>
<td>165</td>
</tr>
<tr>
<td>GenPsfMotion</td>
<td>166</td>
</tr>
<tr>
<td>SimulateDefocus</td>
<td>168</td>
</tr>
<tr>
<td>SimulateMotion</td>
<td>169</td>
</tr>
<tr>
<td>WienerFilter</td>
<td>170</td>
</tr>
<tr>
<td>WienerFilterNi</td>
<td>172</td>
</tr>
<tr>
<td>4 Graphics</td>
<td>175</td>
</tr>
<tr>
<td>4.1 Drawing</td>
<td>175</td>
</tr>
<tr>
<td>DragRegion1</td>
<td>175</td>
</tr>
<tr>
<td>DragRegion2</td>
<td>176</td>
</tr>
<tr>
<td>DragRegion3</td>
<td>177</td>
</tr>
<tr>
<td>DrawCircle</td>
<td>178</td>
</tr>
<tr>
<td>DrawCircleMod</td>
<td>179</td>
</tr>
<tr>
<td>DrawEllipse</td>
<td>180</td>
</tr>
<tr>
<td>DrawEllipseMod</td>
<td>181</td>
</tr>
<tr>
<td>DrawLine</td>
<td>182</td>
</tr>
<tr>
<td>DrawLineMod</td>
<td>183</td>
</tr>
<tr>
<td>DrawPoint</td>
<td>185</td>
</tr>
<tr>
<td>DrawPointMod</td>
<td>185</td>
</tr>
<tr>
<td>DrawPolygon</td>
<td>186</td>
</tr>
<tr>
<td>DrawRectangle1</td>
<td>187</td>
</tr>
<tr>
<td>DrawRectangle1Mod</td>
<td>188</td>
</tr>
<tr>
<td>DrawRectangle2</td>
<td>190</td>
</tr>
<tr>
<td>DrawRectangle2Mod</td>
<td>191</td>
</tr>
<tr>
<td>DrawRegion</td>
<td>192</td>
</tr>
<tr>
<td>4.2 Gnuplot</td>
<td>193</td>
</tr>
<tr>
<td>GnuplotClose</td>
<td>193</td>
</tr>
<tr>
<td>GnuplotOpenFile</td>
<td>193</td>
</tr>
<tr>
<td>GnuplotOpenPipe</td>
<td>194</td>
</tr>
<tr>
<td>GnuplotPlotCtrl</td>
<td>194</td>
</tr>
</tbody>
</table>
GnuplotPlotFunct1D ......................................... 195
GnuplotPlotImage .......................................... 196

4.3 LUT ................................................. 197
DispLut ............................................... 197
DrawLut ............................................... 198
GetFixedLut ............................................. 199
GetLut ............................................... 199
GetLutStyle ............................................. 200
QueryLut ............................................... 200
SetFixedLut ............................................. 201
SetLut ............................................... 201
SetLutStyle ............................................. 204
WriteLut ............................................... 205

4.4 Mouse ................................................ 205
GetMbutton ............................................. 205
GetMposition ............................................ 206
GetMshape .............................................. 207
QueryMshape ............................................ 208
SetMshape .............................................. 208

4.5 Output ................................................ 209
DispArc ............................................... 209
DispArrow .............................................. 210
DispChannel ............................................. 211
DispCircle .............................................. 212
DispColor .............................................. 214
DispDistribution ....................................... 214
DispEllipse ............................................. 215
DispImage .............................................. 217
DispLine ............................................... 218
DispObj ............................................... 219
DispPolygon ............................................ 220
DispRectangle1 ........................................... 220
DispRectangle2 ........................................... 222
DispRegion ............................................. 223
DispXld ............................................... 224

4.6 Parameters.............................................. 224
GetComprise ............................................. 224
GetDraw ............................................... 225
GetFix ................................................ 225
GetHsi ............................................... 226
GetIcon ............................................... 227
GetInsert ............................................... 227
GetLineApprox ........................................... 228
GetLineStyle ............................................ 228
GetLineWidth ............................................ 229
GetPaint ............................................... 229
GetPart ................................................ 230
GetPartStyle ............................................ 230
GetPixel ............................................... 231
GetRgb ................................................ 232
GetShape .............................................. 232
QueryAllColors ........................................... 233
QueryColor ............................................. 233
QueryColored .......................................... 234
QueryGray ............................................. 235
QueryInsert ............................................ 235
QueryLineWidth ........................................... 236
QueryPaint ............................................. 237
### Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryShape</td>
<td>237</td>
</tr>
<tr>
<td>SetColor</td>
<td>238</td>
</tr>
<tr>
<td>SetColored</td>
<td>239</td>
</tr>
<tr>
<td>SetComprise</td>
<td>239</td>
</tr>
<tr>
<td>SetDraw</td>
<td>240</td>
</tr>
<tr>
<td>SetFix</td>
<td>241</td>
</tr>
<tr>
<td>SetGray</td>
<td>242</td>
</tr>
<tr>
<td>SetHsi</td>
<td>243</td>
</tr>
<tr>
<td>SetIcon</td>
<td>244</td>
</tr>
<tr>
<td>SetInsert</td>
<td>244</td>
</tr>
<tr>
<td>SetLineApprox</td>
<td>245</td>
</tr>
<tr>
<td>SetLineStyle</td>
<td>246</td>
</tr>
<tr>
<td>SetLineWidth</td>
<td>247</td>
</tr>
<tr>
<td>SetPaint</td>
<td>247</td>
</tr>
<tr>
<td>SetPart</td>
<td>251</td>
</tr>
<tr>
<td>SetPartStyle</td>
<td>252</td>
</tr>
<tr>
<td>SetPixel</td>
<td>253</td>
</tr>
<tr>
<td>SetRgb</td>
<td>254</td>
</tr>
<tr>
<td>SetShape</td>
<td>255</td>
</tr>
<tr>
<td>GetFont</td>
<td>256</td>
</tr>
<tr>
<td>GetStringExtents</td>
<td>256</td>
</tr>
<tr>
<td>GetTposition</td>
<td>257</td>
</tr>
<tr>
<td>GetTshape</td>
<td>258</td>
</tr>
<tr>
<td>NewLine</td>
<td>259</td>
</tr>
<tr>
<td>QueryFont</td>
<td>259</td>
</tr>
<tr>
<td>QueryTshape</td>
<td>260</td>
</tr>
<tr>
<td>ReadChar</td>
<td>260</td>
</tr>
<tr>
<td>ReadString</td>
<td>261</td>
</tr>
<tr>
<td>SetFont</td>
<td>262</td>
</tr>
<tr>
<td>SetTposition</td>
<td>263</td>
</tr>
<tr>
<td>SetTshape</td>
<td>264</td>
</tr>
<tr>
<td>WriteString</td>
<td>264</td>
</tr>
<tr>
<td>ClearRectangle</td>
<td>265</td>
</tr>
<tr>
<td>ClearWindow</td>
<td>266</td>
</tr>
<tr>
<td>CloseWindow</td>
<td>267</td>
</tr>
<tr>
<td>CopyRectangle</td>
<td>267</td>
</tr>
<tr>
<td>DumpWindow</td>
<td>269</td>
</tr>
<tr>
<td>GetWindowExtents</td>
<td>270</td>
</tr>
<tr>
<td>GetWindowPointer3</td>
<td>271</td>
</tr>
<tr>
<td>GetWindowType</td>
<td>272</td>
</tr>
<tr>
<td>MoveRectangle</td>
<td>273</td>
</tr>
<tr>
<td>NewExternWindow</td>
<td>274</td>
</tr>
<tr>
<td>OpenTextwindow</td>
<td>276</td>
</tr>
<tr>
<td>OpenWindow</td>
<td>279</td>
</tr>
<tr>
<td>QueryWindowType</td>
<td>283</td>
</tr>
<tr>
<td>SetWindowAttr</td>
<td>283</td>
</tr>
<tr>
<td>SetWindowDe</td>
<td>284</td>
</tr>
<tr>
<td>SetWindowExtents</td>
<td>285</td>
</tr>
<tr>
<td>SetWindowType</td>
<td>286</td>
</tr>
<tr>
<td>SlideImage</td>
<td>287</td>
</tr>
<tr>
<td>AccessChannel</td>
<td>289</td>
</tr>
<tr>
<td>AppendChannel</td>
<td>289</td>
</tr>
<tr>
<td>ChannelsToImage</td>
<td>290</td>
</tr>
<tr>
<td>Compose2</td>
<td>290</td>
</tr>
</tbody>
</table>

5 Image

5.1 Channel

### 4.7 Text

5.1 Channel

### 4.8 Window

### 5 Image

5.1 Channel
Contents

Compose3 .............................................. 291
Compose4 .............................................. 292
Compose5 .............................................. 292
Compose6 .............................................. 293
Compose7 .............................................. 294
CountChannels ............................................ 294
Decompose2 ............................................. 295
Decompose3 ............................................. 295
Decompose4 ............................................. 296
Decompose5 ............................................. 297
Decompose6 ............................................. 297
Decompose7 ............................................. 298
ImageToChannels .......................................... 299

5.2 Creation ............................................... 299
CopyImage .............................................. 299
GenImage1 .............................................. 300
GenImage1Extern .......................................... 301
GenImage1Rect ........................................... 302
GenImage3 .............................................. 304
GenImageConst ........................................... 305
GenImageGrayRamp ........................................ 306
GenImageProto ........................................... 307
GenImageSurfaceSecondOrder ...................................
GetImageTime ............................................ 310
RegionToBin ............................................. 311
RegionToLabel ............................................ 312
RegionToMean ............................................ 313

5.3 Domain ............................................... 314
AddChannels ............................................. 314
ChangeDomain ........................................... 314
FullDomain ............................................. 315
GetDomain .............................................. 315
Rectangle1Domain ......................................... 316
ReduceDomain ........................................... 317

5.4 Features ............................................... 317
AreaCenterGray ........................................... 317
CoocFeatureImage .......................................... 318
CoocFeatureMatrix ......................................... 319
EllipticAxisGray ........................................... 320
EntropyGray ............................................. 321
FitSurfaceFirstOrder ....................................... 322
FitSurfaceSecondOrder ..................................... 323
FuzzyEntropy ............................................ 325
FuzzyPerimeter ........................................... 326
GenCoocMatrix ........................................... 327
GrayHisto .............................................. 328
GrayProjections ........................................... 329
Histo2Dim .............................................. 330
Intensity ............................................... 330
MinMaxGray ............................................ 331
MomentsGrayPlane ......................................... 333
PlaneDeviation ........................................... 334
SelectGray .............................................. 334
ShapeHistoAll ........................................... 336
ShapeHistoPoint .......................................... 337

5.5 Format ............................................... 338
ChangeFormat ........................................... 338
CropDomain ............................................. 339

HALCON 6.0
CropPart .......................................................... 339
CropRectangle1 ..................................................... 340
TileChannels .......................................................... 341
TileImages ............................................................ 342
TileImagesOffset ..................................................... 344

5.6 Framegrabber ...................................................... 345
CloseAllFramegrabbers ............................................... 345
CloseFramegrabber .................................................... 346
GetFramegrabberLut .................................................. 346
GetFramegrabberParam ............................................... 347
GrabImage ............................................................ 348
GrabImageAsync ...................................................... 349
GrabImageStart ...................................................... 350
GrabRegion ........................................................... 351
GrabRegionAsync .................................................... 351
InfoFramegrabber ..................................................... 352
OpenFramegrabber .................................................... 354
SetFramegrabberLut ................................................... 356
SetFramegrabberParam ............................................... 357

5.7 Manipulation ...................................................... 357
GetGrayval ............................................................ 357
GetImagePointer1 ..................................................... 358
GetImagePointer1Rect ............................................... 359
GetImagePointer3 ..................................................... 360
PaintGray ............................................................. 361
PaintRegion ........................................................... 362
SetGrayval ............................................................. 363

5.8 Type-Conversion .................................................. 364
ComplexToReal ....................................................... 364
ConvertImageType .................................................... 364
DvfToInt1 .............................................................. 365
Int1ToDvf .............................................................. 365
RealToComplex ......................................................... 366

6 Lines ................................................................... 367
6.1 Access ............................................................... 367
ApproxChain .......................................................... 367
ApproxChainSimple .................................................... 370

6.2 Features ............................................................ 371
LineOrientation ......................................................... 371
LinePosition ............................................................ 372
PartitionLines .......................................................... 373
SelectLines ............................................................. 375
SelectLinesLongest .................................................... 376

7 Morphology .......................................................... 379
7.1 Gray-Values ......................................................... 379
DualRank ............................................................... 379
GenDiscSe .............................................................. 381
GrayBothat ............................................................. 381
GrayClosing ............................................................. 382
GrayDilation ............................................................ 383
GrayDilationRect ....................................................... 383
GrayErosion ............................................................ 384
GrayErosionRect ....................................................... 385
GrayOpening ............................................................ 386
GrayRangeRect ........................................................ 386
GrayTophat .............................................................. 387
ReadGraySe ......................................................... 388

7.2 Region .................................................................. 389
BottomHat .............................................................. 389
Boundary ............................................................... 390
Closing ................................................................. 390
ClosingCircle ........................................................... 391
ClosingGolay ........................................................... 392
ClosingRectangle1 ...................................................... 393
Dilation1 ................................................................. 394
Dilation2 ................................................................. 396
DilationCircle .......................................................... 397
DilationGolay ........................................................... 398
DilationRectangle1 .................................................... 399
DilationSeq ............................................................ 400
Erosion1 ................................................................. 401
Erosion2 ................................................................. 402
ErosionCircle .......................................................... 403
ErosionGolay ........................................................... 404
ErosionRectangle1 ..................................................... 406
ErosionSeq ............................................................. 407
Fitting ........................................................................ 408
GenStructElements ................................................... 408
GolayElements .......................................................... 409
HitOrMiss ............................................................... 413
HitOrMissGolay ........................................................ 414
HitOrMissSeq .......................................................... 415
MinkowskiAdd1 ....................................................... 416
MinkowskiAdd2 ....................................................... 417
MinkowskiSub1 ........................................................ 418
MinkowskiSub2 ........................................................ 419
MorphHat ............................................................... 421
MorphSkeleton ........................................................ 421
MorphSkiz ............................................................. 422
Opening ...................................................................... 423
OpeningCircle .......................................................... 424
OpeningGolay .......................................................... 425
OpeningRectangle1 .................................................... 426
OpeningSeg ............................................................ 427
Pruning ...................................................................... 428
Thickening .............................................................. 429
ThickeningGolay ........................................................ 431
ThickeningSeq .......................................................... 432
Thinning ................................................................. 433
ThinningGolay .......................................................... 434
ThinningSeq ............................................................ 435
TopHat ..................................................................... 436

8 Object .................................................................. 439

8.1 Information .......................................................... 439
CountObj ................................................................. 439
GetChannelInfo ........................................................ 439
GetObjClass ............................................................. 440
TestEqualObj ............................................................ 441
TestEqualRegion ....................................................... 441
TestObjDef ............................................................... 442

8.2 Manipulation .......................................................... 443
ClearObj ................................................................. 443
ConcatObj ............................................................... 444
CopyObj ................................................................. 444

HALCON 6.0
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenEmptyObj</td>
<td>445</td>
</tr>
<tr>
<td>IntegerToObj</td>
<td>445</td>
</tr>
<tr>
<td>ObjToInteger</td>
<td>446</td>
</tr>
<tr>
<td>SelectObj</td>
<td>447</td>
</tr>
</tbody>
</table>

### 9 Regions

#### 9.1 Access
- GetRegionChain 449
- GetRegionContour 450
- GetRegionConvex 450
- GetRegionPoints 451
- GetRegionPolygon 452
- GetRegionRuns 453

#### 9.2 Affine-Transformations
- AffineTransRegion 454
- MirrorRegion 455
- MoveRegion 456
- TransposeRegion 456
- ZoomRegion 457

#### 9.3 Creation
- GenCheckerRegion 458
- GenCircle 460
- GenEllipse 461
- GenEmptyRegion 462
- GenGridRegion 463
- GenRandomRegion 464
- GenRandomRegions 465
- GenRectangle1 467
- GenRectangle2 468
- GenRegionHisto 470
- GenRegionHline 470
- GenRegionLine 471
- GenRegionPoints 472
- GenRegionPolygon 473
- GenRegionPolygonFilled 474
- GenRegionRuns 475
- LabelToRegion 476

#### 9.4 Features
- AreaCenter 477
- Circularity 478
- Compactness 479
- ConnectAndHoles 479
- Contlength 480
- Convexity 481
- DiameterRegion 482
- Eccentricity 482
- EllipticAxis 483
- EulerNumber 485
- FindNeighbors 485
- GetRegionIndex 486
- GetRegionThickness 487
- HammingDistance 488
- HammingDistanceNorm 489
- InnerCircle 490
- MomentsRegion2Nd 491
- MomentsRegion2NdInvar 492
- MomentsRegion2NdRelInvar 493
- MomentsRegion3Rd 494
GetSystem .............................................. 603
SetSystem .............................................. 606

11.7 Serial ................................................. 610
ClearSerial .............................................. 610
CloseAllSerials ........................................... 611
CloseSerial .............................................. 611
GetSerialParam ........................................... 612
OpenSerial .............................................. 613
ReadSerial .............................................. 613
SetSerialParam ............................................ 614
WriteSerial .............................................. 615

11.8 Sockets ................................................ 616
CloseSocket ............................................. 616
GetNextSocketDataType ....................................... 616
OpenSocketAccept ......................................... 617
OpenSocketConnect ......................................... 618
ReceiveImage ............................................ 619
ReceiveRegion ............................................ 619
ReceiveTuple ............................................ 620
ReceiveXld .............................................. 620
SendImage .............................................. 621
SendRegion ............................................. 621
SendTuple .............................................. 622
SendXld ............................................... 622
SocketAcceptConnect ....................................... 623

12 Tools 625

12.1 Affine-Transformations ....................................... 625
AffineTransPoint2D ......................................... 625
AffineTransPoint3D ......................................... 626
DvrToHomMat2d .......................................... 627
HomMat2dCompose ......................................... 628
HomMat2dIdentity ......................................... 628
HomMat2dInvert ........................................... 629
HomMat2dRotate .......................................... 629
HomMat2dScale ........................................... 630
HomMat2dSlant ........................................... 631
HomMat2dToAffinePar ....................................... 632
HomMat2dTranslate ......................................... 633
HomMat3dCompose ......................................... 634
HomMat3dIdentity ......................................... 635
HomMat3dInvert ........................................... 636
HomMat3dRotate .......................................... 637
HomMat3dScale ........................................... 639
HomMat3dTranslate ......................................... 640
VectorAngleToRigid ......................................... 642
VectorToHomMat2d ......................................... 643
VectorToRigid ............................................. 644
VectorToSimilarity ......................................... 645

12.2 Background-Estimator ........................................ 645
CloseAllBgEsti ........................................... 645
CloseBgEsti .............................................. 646
CreateBgEsti .............................................. 647
GetBgEstiParams .......................................... 649
GiveBgEstiParams .......................................... 651
RunBgEsti .............................................. 652
SetBgEstiParams .......................................... 653
UpdateBgEsti .............................................. 655

HALCON 6.0
ReadFunct1D ............................................ 7 3 2
SampleFunct1D ........................................... 7 3 2
ScaleYFunct1D ........................................... 7 3 3
SmoothFunct1DGauss ........................................ 7 3 4
SmoothFunct1DMean ........................................ 7 3 4
TransformFunct1D .......................................... 7 3 5
WriteFunct1D ............................................ 7 3 6
XRangeFunct1D ........................................... 7 3 6
YRangeFunct1D ........................................... 7 3 7

12.7 Geometry .............................................. 7 3 7
AngleLI ............................................... 7 3 7
AngleLx ............................................... 7 3 8
DistanceLr .............................................. 7 3 9
DistancePl .............................................. 7 4 0
DistancePp .............................................. 7 4 1
DistancePr .............................................. 7 4 3
DistancePs .............................................. 7 4 4
DistanceRrMin ............................................ 7 4 5
DistanceRrMinDil .......................................... 7 4 6
DistanceSr .............................................. 7 4 8
DistanceSs .............................................. 7 4 9
GetPointsEllipse ........................................... 7 5 0
IntersectionLl ............................................ 7 5 1
ProjectionPl ............................................. 7 5 3

12.8 Hough ............................................... 7 5 4
HoughCircleTrans .......................................... 7 5 4
HoughCircles ............................................ 7 5 4
HoughLineTrans ........................................... 7 5 5
HoughLines ............................................. 7 5 6
SelectMatchingLines ........................................ 7 5 7

12.9 Kalman-Filter ........................................ 7 5 8
FilterKalman ............................................. 7 5 8
ReadKalman ............................................. 7 6 2
SensorKalman ............................................ 7 6 5
UpdateKalman ............................................ 7 6 6

12.10 Matching ............................................. 7 6 8
ClearShapeModel .......................................... 7 6 8
CreateShapeModel ......................................... 7 6 9
FindShapeModel .......................................... 7 7 1
InspectShapeModel ......................................... 7 7 3
ReadShapeModel .......................................... 7 7 4
WriteShapeModel .......................................... 7 7 4

12.11 Measure ............................................. 7 7 5
CloseMeasure ............................................. 7 7 5
GenMeasureArc .......................................... 7 7 5
GenMeasureRectangle2 ...................................... 7 7 7
MeasurePairs ............................................. 7 7 9
MeasurePos ............................................. 7 8 1
MeasureProjection ......................................... 7 8 3
MeasureThresh ........................................... 7 8 4

12.12 OCR ................................................. 7 8 5
AppendOcrTrainf .......................................... 7 8 5
CloseAllOcrs ........................................... 7 8 6
CloseOcr ............................................... 7 8 6
ConcatOcrTrainf ......................................... 7 8 7
CreateOcrClassBox ........................................ 7 8 7
DoOcrMulti ............................................. 7 9 0
### Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoOcrSingle</td>
<td>790</td>
</tr>
<tr>
<td>InfoOcrClassBox</td>
<td>791</td>
</tr>
<tr>
<td>OcrChangeChar</td>
<td>792</td>
</tr>
<tr>
<td>OcrGetFeatures</td>
<td>793</td>
</tr>
<tr>
<td>ReadOcr</td>
<td>794</td>
</tr>
<tr>
<td>ReadOcrTrainf</td>
<td>794</td>
</tr>
<tr>
<td>ReadOcrTrainfNames</td>
<td>795</td>
</tr>
<tr>
<td>ReadOcrTrainfSelect</td>
<td>795</td>
</tr>
<tr>
<td>TestdOcrClassBox</td>
<td>796</td>
</tr>
<tr>
<td>TraindOcrClassBox</td>
<td>797</td>
</tr>
<tr>
<td>TrainfOcrClassBox</td>
<td>798</td>
</tr>
<tr>
<td>WriteOcr</td>
<td>798</td>
</tr>
<tr>
<td>WriteOcrTrainf</td>
<td>799</td>
</tr>
<tr>
<td>WriteOcrTrainfImage</td>
<td>800</td>
</tr>
</tbody>
</table>

12.13 OCV

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloseAllOcvv</td>
<td>801</td>
</tr>
<tr>
<td>CloseOcvv</td>
<td>801</td>
</tr>
<tr>
<td>CreateOcvProj</td>
<td>801</td>
</tr>
<tr>
<td>DoOcvSimple</td>
<td>802</td>
</tr>
<tr>
<td>ReadOcv</td>
<td>803</td>
</tr>
<tr>
<td>TraindOcvProj</td>
<td>804</td>
</tr>
<tr>
<td>WriteOcv</td>
<td>805</td>
</tr>
</tbody>
</table>

12.14 Shape-from

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DepthFromFocus</td>
<td>806</td>
</tr>
<tr>
<td>Estimate4AlAm</td>
<td>806</td>
</tr>
<tr>
<td>Estimate4SLAI4Lr</td>
<td>807</td>
</tr>
<tr>
<td>Estimate4SLAI4Zc</td>
<td>808</td>
</tr>
<tr>
<td>Estimate4TiltLr</td>
<td>808</td>
</tr>
<tr>
<td>Estimate4TiltZc</td>
<td>809</td>
</tr>
<tr>
<td>PhotStereo</td>
<td>809</td>
</tr>
<tr>
<td>SelectGrayvaluesFromChannels</td>
<td>810</td>
</tr>
<tr>
<td>SfsModLr</td>
<td>811</td>
</tr>
<tr>
<td>SfsOrigLr</td>
<td>812</td>
</tr>
<tr>
<td>SfsPentland</td>
<td>813</td>
</tr>
<tr>
<td>ShadeHeightField</td>
<td>814</td>
</tr>
</tbody>
</table>

13 Tuple

13.1 Arithmetic

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TupleAbs</td>
<td>817</td>
</tr>
<tr>
<td>TupleAcos</td>
<td>817</td>
</tr>
<tr>
<td>TupleAdd</td>
<td>818</td>
</tr>
<tr>
<td>TupleAsin</td>
<td>818</td>
</tr>
<tr>
<td>TupleAtan</td>
<td>819</td>
</tr>
<tr>
<td>TupleAtan2</td>
<td>819</td>
</tr>
<tr>
<td>TupleCos</td>
<td>820</td>
</tr>
<tr>
<td>TupleCosh</td>
<td>820</td>
</tr>
<tr>
<td>TupleDiv</td>
<td>820</td>
</tr>
<tr>
<td>TupleExp</td>
<td>821</td>
</tr>
<tr>
<td>TupleFabs</td>
<td>821</td>
</tr>
<tr>
<td>TupleFmod</td>
<td>822</td>
</tr>
<tr>
<td>TupleLdexp</td>
<td>822</td>
</tr>
<tr>
<td>TupleLog</td>
<td>823</td>
</tr>
<tr>
<td>TupleLog10</td>
<td>823</td>
</tr>
<tr>
<td>TupleMult</td>
<td>824</td>
</tr>
<tr>
<td>TupleNeg</td>
<td>824</td>
</tr>
<tr>
<td>TuplePow</td>
<td>824</td>
</tr>
<tr>
<td>TupleRad</td>
<td>825</td>
</tr>
<tr>
<td>TupleSin</td>
<td>825</td>
</tr>
<tr>
<td>TupleSinh</td>
<td>826</td>
</tr>
</tbody>
</table>
### 13.2 Bit-Operations
- TupleBand .............................................. 828
- TupleBnot .............................................. 828
- TupleBor ............................................... 829
- TupleBxor .............................................. 829
- TupleLsh ............................................... 830
- TupleRsh ............................................... 830

### 13.3 Comparison
- TupleEqual .............................................. 831
- TupleGreater ............................................. 831
- TupleGreaterEqual .......................................... 832
- TupleLess .............................................. 832
- TupleLessEqual ........................................... 833
- TupleNotEqual ............................................ 834

### 13.4 Conversion
- TupleChr ............................................... 834
- TupleChrt .............................................. 834
- TupleDeg ............................................... 835
- TupleNumber ............................................ 835
- TupleOrd ............................................... 836
- TupleOrds .............................................. 836
- TupleReal .............................................. 837
- TupleRound ............................................. 837
- TupleString ............................................. 838

### 13.5 Creation
- TupleConcat ............................................. 839
- TupleGenConst ........................................... 840

### 13.6 Element-Order
- TupleInverse ............................................. 840
- TupleSort .............................................. 840
- TupleSortIndex ........................................... 841

### 13.7 Features
- TupleCeil .............................................. 841
- TupleDeviation ........................................... 842
- TupleFloor .............................................. 842
- TupleIsNumber ........................................... 843
- TupleLength ............................................. 843
- TupleMax .............................................. 843
- TupleMean .............................................. 844
- TupleMin .............................................. 844
- TupleSum .............................................. 845

### 13.8 Logical-Operations
- TupleAnd .............................................. 845
- TupleNot ............................................... 846
- TupleOr ............................................... 846
- TupleXor ............................................... 846

### 13.9 Selection
- TupleFirstN ............................................. 847
- TupleLastN ............................................. 848
- TupleSelect ............................................. 848
- TupleSelectRange ........................................ 849
- TupleStrBitSelect ........................................ 849

### 13.10 String-Operators ........................................... 850
## TupleEnvironment

TupleEnvironment

## TupleSplit

TupleSplit

## TupleStrFirstN

TupleStrFirstN

## TupleStrLastN

TupleStrLastN

## TupleStrchr

TupleStrchr

## TupleStrlen

TupleStrlen

## TupleStrrchr

TupleStrrchr

## TupleStrrstr

TupleStrrstr

## TupleStrstr

TupleStrstr

### 14 XLD

#### 14.1 Access

- GetContourXld
- GetLinesXld
- GetParallelsXld
- GetPolygonXld

#### 14.2 Creation

- GenContourPolygonXld
- GenContourRegionXld
- GenContoursSkeletonXld
- GenEllipseContourXld
- GenParallelsXld
- GenPolygonsXld
- ModParallelsXld

#### 14.3 Features

- AreaCenterXld
- ContourPointNumXld
- DistEllipseContourXld
- FitCircleContourXld
- FitEllipseContourXld
- FitLineContourXld
- GetContourAngleXld
- GetContourAttribXld
- GetContourGlobalAttribXld
- GetRegressParamsXld
- InfoParallelsXld
- LengthXld
- LocalMaxContoursXld
- MaxParallelsXld
- MomentsAnyXld
- MomentsXld
- QueryContourAttribsXld
- QueryContourGlobalAttribsXld
- SelectContoursXld

#### 14.4 Transformation

- AddNoiseWhiteContourXld
- AffineTransContourXld
- AffineTransPolygonXld
- ClipContoursXld
- CombineRoadsXld
- GenParallelContourXld
- MergeContLineScanXld
- RegressContoursXld
- SegmentContoursXld
- SmoothContoursXld
- SplitContoursXld
- UnionStraightContoursHistoXld
- UnionStraightContoursXld
<table>
<thead>
<tr>
<th>15 Class Hierarchy</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15.1 HBarcodex</td>
<td>901</td>
</tr>
<tr>
<td>15.2 HBgEstiX</td>
<td>902</td>
</tr>
<tr>
<td>15.3 HClassBoxX</td>
<td>902</td>
</tr>
<tr>
<td>15.4 HFeatureSetX</td>
<td>903</td>
</tr>
<tr>
<td>15.5 HFileX</td>
<td>903</td>
</tr>
<tr>
<td>15.6 HFramegrabberX</td>
<td>904</td>
</tr>
<tr>
<td>15.7 HFunction1dX</td>
<td>906</td>
</tr>
<tr>
<td>15.8 HGnuplotX</td>
<td>907</td>
</tr>
<tr>
<td>15.9 HHistogramX</td>
<td>907</td>
</tr>
<tr>
<td>15.10 HHomMat2dX</td>
<td>907</td>
</tr>
<tr>
<td>15.11 HHomMat3dX</td>
<td>909</td>
</tr>
<tr>
<td>15.12 HimageX</td>
<td>909</td>
</tr>
<tr>
<td>15.13 HInfoX</td>
<td>920</td>
</tr>
<tr>
<td>15.14 HMeasureX</td>
<td>921</td>
</tr>
<tr>
<td>15.15 HMiscX</td>
<td>921</td>
</tr>
<tr>
<td>15.16 HObjectX</td>
<td>924</td>
</tr>
<tr>
<td>15.17 HOCRX</td>
<td>925</td>
</tr>
<tr>
<td>15.18 HOevX</td>
<td>925</td>
</tr>
<tr>
<td>15.19 HOperatorSetX</td>
<td>926</td>
</tr>
<tr>
<td>15.20 HPoseX</td>
<td>962</td>
</tr>
<tr>
<td>15.21 HRegionX</td>
<td>963</td>
</tr>
<tr>
<td>15.22 HSerialX</td>
<td>971</td>
</tr>
<tr>
<td>15.23 HShapeModelX</td>
<td>971</td>
</tr>
<tr>
<td>15.24 HSocketX</td>
<td>972</td>
</tr>
<tr>
<td>15.25 HSystemX</td>
<td>973</td>
</tr>
<tr>
<td>15.26 HTemplateX</td>
<td>974</td>
</tr>
<tr>
<td>15.27 HTupleX</td>
<td>974</td>
</tr>
<tr>
<td>15.28 HUntypedObjectX</td>
<td>978</td>
</tr>
<tr>
<td>15.29 HWindowX</td>
<td>978</td>
</tr>
<tr>
<td>15.30 HXLDX</td>
<td>985</td>
</tr>
<tr>
<td>15.31 HXLDCntX</td>
<td>985</td>
</tr>
<tr>
<td>15.32 HXLDExtParaX</td>
<td>987</td>
</tr>
<tr>
<td>15.33 HXLDModParaX</td>
<td>987</td>
</tr>
<tr>
<td>15.34 HXLDParaX</td>
<td>988</td>
</tr>
<tr>
<td>15.35 HXLDPolyX</td>
<td>988</td>
</tr>
</tbody>
</table>

HALCON 6.0
Chapter 1

Classification

```c
void HOperatorSetX.ClearSampset ([in] long SampKey )
```

Free memory of a data set.

ClearSampset frees the memory which was used for training data set having read by ReadSampset. This memory is only reusable in combination with ReadSampset.

**Parameter**

- **SampKey** (input control) ........................................ feature_set ~ HFeatureSetX / long
  Number of the data set.

**Result**

ClearSampset returns TRUE. An exception handling is raised if the key SampKey does not exist.

**Parallelization Information**

ClearSampset is local and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

CreateClassBox, EnquireClassBox, LearnClassBox, WriteClassBox

**See Also**

TestSampsetBox, LearnSampsetBox, ReadSampset

**Module**

Tools

```c
void HMiscX.CloseAllClassBox ( )
void HOperatorSetX.CloseAllClassBox ( )
```

Destroy all classifiers.

CloseAllClassBox deletes all classifiers and frees the used memory space. All the trained data will be lost.

**Attention**

Since all classifiers are closed by CloseAllClassBox all handles become invalid.

**Result**

If it is possible to close the classifiers the operator CloseAllClassBox returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

CloseAllClassBox is local and processed completely exclusively without parallelization.

**Alternatives**

CloseClassBox

**Module**

Tools
CHAPTER 1. CLASSIFICATION

```c
void HOperatorSetX.CloseClassBox ([in] long ClassifHandle )
```

Destroy the classifier.

CloseClassBox deletes the classifier and frees the used memory space. All the trained data will be lost. For saving this trained data you should use WriteClassBox before.

Parameter

▷ ClassifHandle (input control) ............................................. class_box ~ HClassBoxX / long Classifier’s handle number.

Result

CloseClassBox returns TRUE.

Parallelization Information

CloseClassBox is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

CreateClassBox, EnquireClassBox, LearnClassBox, WriteClassBox

See Also

CreateClassBox, EnquireClassBox, LearnClassBox

Tools

```c
void HClassBoxX.CreateClassBox ()
void HOperatorSetX.CreateClassBox ([out] long ClassifHandle )
```

Create a new classifier.

CreateClassBox creates a new adaptive classifier. All procedures which are explained in chapter classification refer to such a initialized classifier (of type 2). See LearnClassBox for more details about the functionality of the classifier.

Parameter

▷ ClassifHandle (output control) ............................................. class_box ~ HClassBoxX / long Classifier’s handle number.

Result

CreateClassBox returns TRUE if the parameter is correct. An exception handling is raised if a classifier with this name already exists or there is not enough memory.

Parallelization Information

CreateClassBox is local and processed completely exclusively without parallelization.

Possible Successor Functions

LearnClassBox, EnquireClassBox, WriteClassBox, CloseClassBox, ClearSampset

See Also

LearnClassBox, EnquireClassBox, CloseClassBox

Tools

```c
void HClassBoxX.DescriptClassBox ([in] long Dimensions )
void HOperatorSetX.DescriptClassBox ([in] long ClassifHandle, [in] long Dimensions )
```

Description of the classifier.

A classifier uses a set of hyper cuboids for every class. With these hyper cuboids it is attempted to get the array attributes inside the class. DescriptClassBox returns for every class the expansion of every appropriate cuboid from dimension 1 up to Dimensions (to ’standard_output’).
Classify a tuple of attributes.

FeatureList is a tuple of any floating point- or integer numbers (attributes) which has to be assigned to a class with assistance of a previous trained (LearnClassBox) classifier. It is possible to specify attributes as unknown by indicating the symbol ‘?’ instead of a number. If you specify n values, then all following values, i.e. the attributes n+1 until ‘max’, are automatically supposed to be undefined.

See LearnClassBox for more details about the functionality of the classifier.

You may call the procedures LearnClassBox and EnquireClassBox alternately, so that it is possible to classify already in the phase of learning. This means you could see when a satisfying behavior had been reached.

Parameter

- **ClassifHandle** (input control) .......................... class_box ~ HClassBoxX / long  
  Classifier’s handle number.
- **FeatureList** (input control) .......................... real ~ VARIANT ( integer, real, string )  
  Array of attributes which has to be classified.
- **Class** (output control) .......................... integer ~ long  
  Number of the class to which the array of attributes had been assigned.

Result

EnquireClassBox returns TRUE.

Parallelization Information

EnquireClassBox is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

CreateClassBox, LearnClassBox, SetClassBoxParam

Possible Successor Functions

LearnClassBox, WriteClassBox, CloseClassBox

See Also

CreateClassBox, EnquireClassBox, LearnClassBox, ReadClassBox, WriteClassBox

Module

Tools

```plaintext
[out] long Class HClassBoxX.EnquireClassBox ([in] VARIANT FeatureList )
void HOperatorSetX.EnquireClassBox ([in] long ClassifHandle,  
  [in] VARIANT FeatureList, [out] long Class )
```

HALCON 6.0
Classify a tuple of attributes with rejection class.

FeatureList is a tuple of any floating point- or integer numbers (attributes) which has to be assigned to a class with assistance of a previous trained (LearnClassBox) classificator. It is possible to specify attributes as unknown by indicating the symbol 'A' instead of a number. If you specify n values, then all following values, i.e. the attributes n+1 until 'max', are automatically supposed to be undefined.

If the array of attributes cannot be assigned to a class, i.e. the array does not reside inside of one of the hyper boxes, then in contrary to EnquireClassBox not the next class is going to be returned, but the rejection class -1 as a result is going to be passed.

See LearnClassBox for more details about the functionality of the classificator.

You may call the procedures LearnClassBox and EnquireClassBox alternately, so that it is possible to classify already in the phase of learning. By this means you could see when a satisfying behavior had been reached.

Parameter

- **ClassifHandle** (input control) .................. class_box ~ HClassBoxX / long Classifier’s handle number.
- **FeatureList** (input control) ......................... real ~ VARIANT (integer, real, string) Array of attributes which has to be classified.
  Default Value: 1.0
- **Class** (output control) ................................. integer ~ long Number of the class, to which the array of attributes had been assigned or -1 for the rejection class.

Result

EnquireRejectClassBox returns TRUE.

Parallelization Information

EnquireRejectClassBox is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

CreateClassBox, LearnClassBox, SetClassBoxParam

Possible Successor Functions

LearnClassBox, WriteClassBox, CloseClassBox

See Also

TestSampsetBox, LearnClassBox, LearnSampsetBox

Alternatives

EnquireClassBox

Module

Tools

Get information about the current parameter.

GetClassBoxParam gets the parameter of the classificator. The meaning of the parameter is explained in SetClassBoxParam.
Default values:
'min_samples_for_split' = 80,
'split_error' = 0.1,
'prop_constant' = 0.25

Parameter

ClassifHandle (input control) ........................................... class_box ~ HClassBoxX / long
Classifier’s handle number.

Flag (input control) ....................................................... string ~ String
Name of the system parameter.

Default Value : 'split_error'
Value List : Flag ∈ {'split_error', 'prop_constant', 'used_memory', 'min_samples_for_split'}

Value (output control) ................................................. number ~ VARIANT( integer, real )
Value of the system parameter.

GetClassBoxParam returns TRUE. An exception handling is raised if Flag has been set with wrong values.

Parallelization Information
GetClassBoxParam is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
CreateClassBox, EnquireClassBox, LearnClassBox, WriteClassBox

Possible Successor Functions
SetClassBoxParam, LearnClassBox, EnquireClassBox, WriteClassBox, CloseClassBox, ClearSampset

See Also

CreateClassBox, SetClassBoxParam

Train the classificator.

Features is a tuple of any floating point numbers or integers (attributes) which has to be assigned to the class
Class. This class is specified by an integer. You may use procedure EnquireClassBox later to find the
most probable class for any array (=tupel). The algorithm tries to describe the set of arrays of one class by hyper
cuboids in the feature space. On demand you may even create several cuboids per class. Hence it is possible to
learn disjunct concepts, too. I.e such concepts which split in several ”cluster” of points in the feature space. The
data structure is hidden to the user and only accessible with such procedures which are described in this chapter.

It is possible to specify attributes as unknown by indicating the symbol 'A' instead of a number. If you specify n
values, then all following values, i.e. the attributes n+1 until 'max', are automatically supposed to be undefined.

You may call the procedures LearnClassBox and EnquireClassBox alternately, so that it is possible to
classify already in the phase of learning. By this means you could see when a satisfying behavior had been
reached.

The classificator is going to be bigger using further training. This means, that it is not advisable to continue training
after reaching a satisfactory behavior.

Parameter

ClassifHandle (input control) ........................................... class_box ~ HClassBoxX / long
Classifier’s handle number.

Features (input control) ................................................. number ~ VARIANT( integer, real, string )
Array of attributes to learn.

Default Value : [1.0,1.5,2.0]
CHAPTER 1. CLASSIFICATION

Class (input control) .............................................................. integer ~ long
Class to which the array has to be assigned.
Default Value : 1

Result

LearnClassBox returns TRUE for a normal case. An exception handling is raised if there are memory allocation problems. The number of classes is constrained. If this limit is passed, an exception handling is raised, too.

Parallelization Information

LearnClassBox is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

CreateClassBox, EnquireClassBox

Possible Successor Functions

TestSampsetBox, LearnClassBox, EnquireClassBox, WriteClassBox, CloseClassBox, ClearSampset

See Also

TestSampsetBox, CloseClassBox, CreateClassBox, EnquireClassBox, LearnSampsetBox

Module

Tools


Train the classifier with one data set.

LearnSampsetBox trains the classifier with data for the key SampKey (see ReadSampset). The training sequence is terminated at least after NSamples examples. If NSamples is bigger than the number of examples in SampKey, then a cyclic start at the beginning occurs. If the error underpasses the value StopError, then the training sequence is prematurely terminated. StopError is calculated with N / ErrorN. Whereby N signifies the number of examples which were wrong classified during the last ErrorN training examples. Typically ErrorN is the number of examples in SampKey and NSamples is a multiple of it. If you want a data set with 100 examples to run 5 times at most and if you want it to terminate with an error lower than 5%, then the corresponding values are NSamples = 500, ErrorN = 100 and StopError = 0.05. A protocol of the training activity is going to be written in file Outfile.

Parameter

- ClassifHandle (input control) ............................................ class_box ~ HClassBoxX / long
  Classifier’s handle number.
- SampKey (input control) ................................................ feature_set ~ HFeatureSetX / long
  Number of the data set to train.
- Outfile (input control) ................................................ filename ~ String
  Name of the protocol file.
  Default Value : ’training_prot’
- NSamples (input control) .............................................. integer ~ long
  Number of arrays of attributes to learn.
  Default Value : 500
- StopError (input control) ............................................. real ~ double
  Classification error for termination.
  Default Value : 0.05
LearnSampsetBox returns TRUE. An exception handling is raised if key SampKey does not exist or there are problems while opening the file.

```
void HClassBoxX.ReadClassBox ([in] String FileName )
void HOperatorSetX.ReadClassBox ([in] long ClassifHandle, [in] String FileName )
```

Read the classifier from a file.

ReadClassBox reads the saved classifier from the file FileName (see WriteClassBox). The values of the current classifier are overwritten.

```
Attention
```

All values of the classifier are going to be overwritten.

```
Parameter
```

- **ClassifHandle** (input control) .......................... class_box  HClassBoxX / long
  Classifier’s handle number.
- **FileName** (input control) .......................... filename  String
  Filename of the classifiers.

```
Default Value: ‘klassifikator1’
```

ReadClassBox returns TRUE. An exception handling is raised if it was not possible to open file FileName or the file has the wrong format.

```
Parallelization Information
```

ReadClassBox is local and processed completely exclusively without parallelization.

```
Possible Predecessor Functions
```

``` CreateClassBox ```
```
Possible Successor Functions
```

``` TestSampsetBox, EnquireClassBox, WriteClassBox, CloseClassBox, ClearSampset ```
```
See Also
```

``` TestSampsetBox, EnquireClassBox, LearnClassBox, ReadSampset ```
```
Module
```

```
Tools
```

```
void HFeatureSetX.ReadSampset ([in] String FileName )
void HOperatorSetX.ReadSampset ([in] String FileName, [out] long SampKey )
```

Read a training data set from a file.
The training examples are accessible with the key `SampKey` by calling procedures `ClearSampset` and `LearnSampsetBox`. You may edit the file using an editor. Every row contains an array of attributes with corresponding class. An example for a format might be:

\[(1.0, 25.3, *, 17 \mid 3)\]

This row specifies an array of attributes which belong to class 3. In this array the third attribute is unknown. Attributes upwards 5 are supposed to be unknown, too. You may insert comments like /* .. */ in any place.

### Parameter

- **FileName** (input control) ........................................ filename → String
  
  Filename of the data set to train.

  Default Value: 'sampset1'

- **SampKey** (output control) ................................ feature_set → HFeatureSetX / long
  
  Identification of the data set to train.

### Result

ReadSampset returns TRUE. An exception handling is raised if it is not possible to open the file or it contains syntax errors or there is not enough memory.

### Parallelization Information

ReadSampset is local and processed completely exclusively without parallelization.

### Possible Predecessor Functions

- `CreateClassBox`

### Possible Successor Functions

- `TestSampsetBox`, `EnquireClassBox`, `WriteClassBox`, `CloseClassBox`, `ClearSampset`

### See Also

- `TestSampsetBox`, `ClearSampset`, `LearnSampsetBox`

### Tools

```c
void HClassBoxX.SetClassBoxParam ([in] String Flag, [in] VARIANT Value )
void HOperatorSetX.SetClassBoxParam ([in] long ClassifHandle, [in] String Flag, [in] VARIANT Value )
```

Set system parameters for classification.

`SetClassBoxParam` modifies parameter which manipulate the training sequence while calling `LearnClassBox`. Only parameters of the classificator are modified, all other classifiers remain unmodified. 'min_samples_for_split' is the number of examples at least which have to train in one cuboid of this classificator, before the cuboid is allowed to divide itself. 'split_error' indicates the critical error. By its exceeding the cuboid divides itself, if there are more than 'min_samples_for_split' examples to train. 'prop_constant' manipulates the extension of the cuboids. It is proportional to the average distance of the training examples in this cuboid to the center of the cuboid. More detailed:

\[\text{extension} \times \text{prop} = \text{average distance of the expectation value}\]

This relation is valid in every dimension. Hence inside a cuboid the dimensions of the feature space are supposed to be independent.

The parameters are set with problem independent default values, which must not modified without any reason. Parameters are only important during a learning sequence. They do not influence on the behavior of `EnquireClassBox`.

Default setting:

- 'min_samples_for_split' = 80,
- 'split_error' = 0.1,
- 'prop_constant' = 0.25
Parameter

» ClassifHandle (input control) .......................... class_box  ~  HClassBoxX / long
  Classifier’s handle number.

» Flag (input control) .............................. string  ~  String
  Name of the wanted parameter.
  Default Value: ˈsplit_errorˈ
  Value Suggestions: Flag ∈ ˈmin_samples_for_split’, ˈsplit_error’, ˈprop_constantˈ

» Value (input control) ........................... number  ~  VARIANT ( integer, real )
  Value of the parameter.
  Default Value: 0.1

Result

Parallelization Information
SetClassBoxParam is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
CreateClassBox, EnquireClassBox

Possible Successor Functions
LearnClassBox, TestSampsetBox, WriteClassBox, CloseClassBox, ClearSampset

See Also
EnquireClassBox, GetClassBoxParam, LearnClassBox

Tools

[out] double Error  HClassBoxX.TestSampsetBox
  ([in] HFeatureSetX SampKey )

[out] double Error  HFeatureSetX.TestSampsetBox
  ([in] HClassBoxX ClassifHandle )

void  HOperatorSetX.TestSampsetBox ([in] long ClassifHandle,
  [in] long SampKey, [out] double Error )

Classify a set of arrays.
In contrast to LearnSampsetBox there is not a learning here. Typically you use TestSampsetBox to classify
independent test data. Error gives you information about the applicability of the learned training set on new
examples.

Parameter

» ClassifHandle (input control) .......................... class_box  ~  HClassBoxX / long
  Classifier’s handle number.

» SampKey (input control) .......................... feature_set  ~  HFeatureSetX / long
  Key of the test data.

» Error (output control) ........................... real  ~  double
  Error during the assignment.

Result

TestSampsetBox returns TRUE. An exception handling is raised, if if key SampKey does not exist or problems
occur while opening the file.

Parallelization Information
TestSampsetBox is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
CreateClassBox, LearnClassBox, SetClassBoxParam

Possible Successor Functions
EnquireClassBox, LearnClassBox, WriteClassBox, CloseClassBox, ClearSampset
Save the classificator in a file.

WriteClassBox saves the classificator in a file. You may read the data by calling ReadClassBox.

Attention

If a file with this name exists, it is overwritten without a warning. The file can not be edited.

Parameter

▷ ClassifHandle (input control) ....................................... class_box ~ HClassBoxX / long
  Classificator’s handle number.

▷ FileName (input control) ................................................ filename ~ String
  Name of the file which contains the written data.
  Default Value: ’klassifikator1’

Result

WriteClassBox returns TRUE. An exception handling is raised if it was not possible to open file FileName.

Parallelization Information

WriteClassBox is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

CreateClassBox, EnquireClassBox, LearnClassBox, TestSampsetBox, WriteClassBox

Possible Successor Functions

CloseClassBox, ClearSampset

See Also

CreateClassBox, ReadClassBox

Module
Chapter 2

File

2.1 Images

```c
void HImageX.ReadImage ([in] VARIANT FileName )

void HOperatorSetX.ReadImage ([out] HUntypedObjectX Image ,
[in] VARIANT FileName )
```

Read an image with different file formats.

The operator `ReadImage` reads the indicated image files from the background storage and generates the image. One or more files can be indicated. The region of the generated image object (= all pixels of the matrix) is maximal chosen.

All images files written by the operator `WriteImage` (format 'ima') have the extension '.ima'. A description file can be available for every image in HALCON format (same file name with extension '.exp'). The type of the pixel data (byte, int4, real) can also be taken from the description file. If this information is not available the type byte is used as presetting.

Besides the HALCON format TIFF, GIF, BMP, JPEG, PCX, SUN-Raster, PGM, PPM, PBM and XWD files can also be read. The gray values of PBM images are set at the values 0 and 255. The file formats are either recognized by the extension (if indicated) or because of the internal structure of the files. If the extension is indicated the image can be found faster. In case of PGM, PPM and PBM the corresponding extension (e.g. 'pgm') or the general value 'pnm' can be used. In case of TIFF 'tiff' and 'tif' are accepted. In case of colored images an image with three color channels (matrices) is created, the red channel being stored in the first, the blue channel in the second and the green channel in the third component (channel number).

Image files are searched in the current directory (determined by the environment variable) and in the image directory of HALCON. The image directory of HALCON is preset at '.' and '/usr/local/halcon/images' in a UNIX environment and can be set via the operator `SetSystem`. More than one image directory can be indicated. This is done by separating the individual directories by a colon.

Furthermore the search path can be set via the environment variable HALCONIMAGES (same structure as 'image_dir'). Example:

```bash
setenv HALCONIMAGES "/usr/images:/usr/local/halcon/images"
```

HALCON also searches images in the subdirectory ‘images’ (Images for the program examples). The environment variable HALCONROOT is used for the HALCON directory.

---

Parameter

- **Image** (output iconic) ......................... image ~ HImageX / HUntypedObjectX ( byte, int4, real )
  < no description available >
- **FileName** (input control) ......................... filename(-array) ~ VARIANT ( string )
  < no description available >

Default Value: `fabrik`

Value Suggestions: `FileName ∈ { 'monkey', 'fabrik', 'mreut' }`
Example

/* Reading an image: */
read_image(Image,'monkey').

/* Reading 3 images into an image object: */
read_image(Bildobjekt, ['house_red','house_green','house_blue']).

/* Setting of search path for images on '/mnt/images' and '/home/images': */
set_system('image_dir','/mnt/images:/home/images').

Result

If the parameters are correct the operator ReadImage returns the value TRUE. If the indicated files cannot be found ReadImage returns the value FAIL. Otherwise an exception handling is raised.

Parallelization Information

ReadImage is processed under mutual exclusion against itself and without parallelization.

Possible Successor Functions

DispImage, Threshold, RegionGrowing, CountChannels, Decompose3, ClassNdimNorm, GaussImage, FillInterlace, ZoomImageSize, ZoomImageFactor, CropPart, WriteImage, Rgb1ToGray

See Also

SetSystem, WriteImage

Alternatives

ReadSequence

Module

Image / region / XLD management

Read images.

The operator ReadSequence reads unformatted image data, from a file and returns a “suitable” image. The image data must be filled consecutively pixel by pixel and line by line.

Any file headers (with the length HeaderSize bytes) are skipped. The parameters SourceWidth and SourceHeight indicate the size of the filled image. DestWidth and DestHeight indicate the size of the image. In the simplest case these parameters are the same. However, areas can also be read. The upper left corner of the required image area can be determined via StartRow and StartColumn.

The pixel types 'bit', 'byte', 'short' (16 bits, unsigned), 'signed short' (16 bits, signed), 'long' (32 bits, signed) and 'swapped long' (32 bits, with swapped segments) are supported. Furthermore the operator ReadSequence enables the extraction of components of a RBG image, if a triple of three bytes (in the sequence “red”, “green”, “blue”) was filed in the image file. For the red component the pixel type 'r_byte' must be chosen, and correspondingly for the green and blue components 'g_byte' or 'b_byte', respectively.
2.1. IMAGES

'MSBFirst' (most significant bit first) or the inversion thereof ('LSBFirst') can be chosen for the bit order (BitOrder). The byte orders (ByteOrder) 'MSBFirst' (most significant byte first) or 'LSBFirst', respectively, are processed analogously. Finally an alignment (Pad) can be set at the end of the line: 'byte', 'short' or 'long'. If a whole image sequence is stored in the file a single image (beginning at Index 1) can be chosen via the parameter Index.

Image files are searched in the current directory (determined by the operating system environment) and in the image directory of HALCON. The image directory of HALCON is preset as '/bilder' and '/usr/local/halcon/images' in a UNIX environment and can be set via the operator SetSystem. More than one image directory can be indicated by separating the individual directories by blanks.

Furthermore the search path can be set via the environment variable HALCONIMAGES (same structure as in 'image dir'). Example:

```
setenv HALCONIMAGES "/usr/images:/usr/local/halcon/images"
```

HALCON also searches the image in the sub-directory "images" (images for program examples). The environment variable HALCONROOT is used for the HALCON directory.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
<th>Restriction</th>
<th>Value List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>(output iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image ~ HImageX / HUntypedObjectX</td>
<td>image</td>
<td></td>
<td></td>
<td>HImageX, HUntypedObjectX</td>
</tr>
<tr>
<td>HeaderSize</td>
<td>(input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . integer ~ long</td>
<td>integer</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SourceWidth</td>
<td>(input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . extent.x ~ long</td>
<td>extent.x</td>
<td>512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SourceHeight</td>
<td>(input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . extent.y ~ long</td>
<td>extent.y</td>
<td>512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>StartRow</td>
<td>(input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . point.y ~ long</td>
<td>point.y</td>
<td>0</td>
<td>(StartRow &lt; SourceHeight)</td>
<td></td>
</tr>
<tr>
<td>StartColumn</td>
<td>(input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . point.x ~ long</td>
<td>point.x</td>
<td>0</td>
<td>(StartColumn &lt; SourceWidth)</td>
<td></td>
</tr>
<tr>
<td>DestWidth</td>
<td>(input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . extent.x ~ long</td>
<td>extent.x</td>
<td>512</td>
<td>(DestWidth ≤ SourceWidth)</td>
<td></td>
</tr>
<tr>
<td>DestHeight</td>
<td>(input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . extent.y ~ long</td>
<td>extent.y</td>
<td>512</td>
<td>(DestHeight ≤ SourceHeight)</td>
<td></td>
</tr>
<tr>
<td>PixelType</td>
<td>(input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . string ~ String</td>
<td>string</td>
<td>'byte'</td>
<td></td>
<td>'bit', 'byte', 'r byte', 'g byte', 'b byte', 'short', 'signed short', 'long', 'swapped long'</td>
</tr>
<tr>
<td>BitOrder</td>
<td>(input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . string ~ String</td>
<td>string</td>
<td>'MSBFirst'</td>
<td></td>
<td>'MSBFirst', 'LSBFirst'</td>
</tr>
</tbody>
</table>
**ByteOrder** (input control) 

Sequence of bytes within one ‘short’ unit.

**Default Value:** 'MSBFirst'

**Value List:** ByteOrder ∈ { 'MSBFirst', 'LSBFirst' }

**Pad** (input control)

Data units within one image line (alignment).

**Default Value:** 'byte'

**Value List:** Pad ∈ { 'byte', 'short', 'long' }

**Index** (input control)

Number of images in the file.

**Default Value:** 1

**FileName** (input control)

Name of input file.

---

If the parameter values are correct the operator ReadSequence returns the value TRUE. If the file cannot be opened FAIL is returned. Otherwise an exception handling is raised.

---

**Parallelization Information**

ReadSequence is processed under **mutual exclusion** against itself and without parallelization.

---

**Possible Successor Functions**

DispImage, CountChannels, Decompose3, WriteImage, Rgb1ToGray

---

**See Also**

ReadImage

---

**Alternatives**

ReadImage

---

**Module**

Image / region / XLD management

---

```c

```

---

**Write images in graphic formats.**

The operator WriteImage returns the indicated image (Image) in different image formats in files. Pixels outside the region receive the color defined by FillColor. For gray value images a value between 0 (black) and 255 (white) must be passed, with RGB color images the RGB values can be passed directly as a hexadecimal value: e.g., 0xffff00 for a yellow background (red=255, green=255, blue=0).

The following formats are currently supported:

- **'tiff'**  
  TIFF format, 3-channel-images (RGB): 3 samples per pixel; other images (grayvalue images): 1 sample per pixel, 8 bits per sample, uncompressed, 72 dpi; file extension: *.tiff

- **'bmp'**  
  Windows-BMP format, 3-channel-images (RGB): 3 bytes per pixel; other images (gray value image): 1 byte per pixel; file extension: *.bmp

- **'jpeg'**  
  JPEG format, with lost of information; together with the format string the quality value determining the compression rate can be provided: e.g., 'jpeg 30'. Attention: images stored for being processed later should not be compressed with the jpeg format according to the lost of information.

- **'ima'**  
  The data is written binary line by line (without header or carriage return). The size of the image and the pixel type are stored in the description file "FileName.exp". **byte, int4 and real** images can be written. The file extension is: '.ima'

---

**Attention**
2.2 Misc

void HMiscX.FileExists ([in] String FileName )
void HOperatorSetX.FileExists ([in] String FileName )

Check whether file exists.

The operator FileExists checks whether the indicated file already exists. If this is the case, the operator FileExists returns TRUE, otherwise FALSE.

Parameter

- **FileName** (input control) ................................................ filename ~ String
  Name of file to be checked.
  Default Value : /bin/cc

Result

The operator FileExists returns the value TRUE (file exists) or FALSE (file does not exist).

Parallelization Information

FileExists is reentrant and processed without parallelization.

Possible Successor Functions

OpenFile

Alternatives

OpenFile

Module

Basic operators
Read the geo coding from an ARC/INFO world file.

ReadWorldFile reads a geocoding from an ARC/INFO world file with the file name FileName and returns it as a homogeneous 2D transformation matrix in WorldTransformation. To find the file FileName, all directories contained in the HALCON system variable 'image_dir' (usually this is the content of the environment variable HALCONIMAGES) are searched (see ReadImage). This transformation matrix can be used to transform XLD contours to the world coordinate system before writing them with WriteContourXldArcInfo. If the matrix WorldTransformation is inverted by calling HomMat2dInvert, the resulting matrix can be used to transform contours that have been read with ReadContourXldArcInfo to the image coordinate system.

Parameter

¬ FileName (input control) ......................... filename ~ String 
Name of the ARC/INFO world file.
¬ WorldTransformation (output control) .......... affine2d ~ HHomMat2dX / VARIANT ( real ) 
Transformation matrix from image to world coordinates.

Parameter Number : 6

Result

If the parameters are correct and the world file could be read, the operator ReadWorldFile returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

ReadWorldFile is reentrant and processed without parallelization.

Possible Successor Functions

HomMat2dInvert, AffineTransContourXld, AffineTransPolygonXld

See Also

WriteContourXldArcInfo, ReadContourXldArcInfo, WritePolygonXldArcInfo, ReadPolygonXldArcInfo

Module

Sub-pixel operators

2.3 Region

Read binary images or HALCON regions.

The operator ReadRegion reads regions from a binary file. The data is stored in packed form.

Tiff: Binary Tiff images with extension 'tiff' or 'tif'. The result is always one region. The color black is used as foreground.

BMP: Binary Windows bitmap images with extension 'bmp'. The result is always one region. The color black is used as foreground.

HALCON regions: File format of HALCON for regions. Several images can be stored (in one file) or read simultaneously via the operators WriteRegion and ReadRegion. All region files have the extension '.reg', which is not indicated when reading or writing the file.

A search path ('image_dir') can be defined analogous to the operator ReadImage.

Attention
2.3. REGION

Parameter

- **Region** (output iconic) ................................ region(-array) \(\sim\) HRegionX / HUtypedObjectX
  <no description available>
- **FileName** (input control) ........................................................ filename \(\sim\) String
  <no description available>

Example

/* Reading of regions and giving them gray values. */
read_image(Img,'bild_test5')
read_region(Regs,'reg_test5')
reduce_domain(Img,Regs,Res)

Result

If the parameter values are correct the operator ReadRegion returns the value TRUE. If the file cannot be opened FAIL is returned. Otherwise an exception handling is raised.

Parallelization Information

ReadRegion is reentrant and processed without parallelization.

Possible Predecessor Functions

ReadImage

Possible Successor Functions

ReduceDomain, DispRegion

See Also

WriteRegion, ReadImage

Module

Image / region / XLD management

void **HRegionX.WriteRegion** ([in] String FileName )

void **HOperatorSetX.WriteRegion** ([in] IObjectX Region, [in] String FileName )

Write regions on file.

The operator WriteRegion writes the regions of the input images (in runlength coding) to a binary file. The data is stored in packed form. The output data can be read via the operator ReadRegion. The file name is <FileName>.reg. The extension is not indicated.

Attention

Parameter

- **Region** (input iconic) ................................ region(-array) \(\sim\) HRegionX / HObjectX
  Region of the images which are returned.
- **FileName** (input control) ........................................................ filename \(\sim\) String
  Name of region file without extension.

Default Value: /tmp/region

Example

regiongrowing(Img, Segmente, 3, 3, 5, 10)
write_region(Segmente, 'result1').

Result

If the parameter values are correct the operator WriteRegion returns the value TRUE. If the file cannot be opened FAIL is returned. Otherwise an exception handling is raised.
WriteRegion is reentrant and processed without parallelization.

Possible Predecessor Functions

OpenWindow, ReadImage, ReadRegion, Threshold, RegionGrowing

See Also

ReadRegion

Module

Image / region / XLD management

2.4 Text

```c
void HMiscX.CloseAllFiles()
void HOperatorSetX.CloseAllFiles()
```

Close all open files.

CloseAllFiles closes all open files.

Attention

Since all files are closed by CloseAllFiles all handles become invalid.

Result

If it is possible to close the files the operator CloseAllFiles returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

CloseAllFiles is local and processed completely exclusively without parallelization.

Alternatives

CloseFile

Module

Basic operators

```c
void HOperatorSetX.CloseFile([in] long FileHandle)
```

Closing a text file.

The operator CloseFile closes a file which was opened via the operator OpenFile.

Parameter

- **FileHandle** (input control) .......................................................... file ~ HFileX / long File handle.

Example

```c
open_file('/tmp/data.txt','input',FileHandle)
// ....
close_file(FileHandle).
```

Result

If the file handle is correct CloseFile returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

CloseFile is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenFile

See Also

OpenFile

2.4. TEXT

Basic operators

```c
void HFileX.Fnewline ( )
void HOperatorSetX.Fnewline ( [in] long FileHandle )
```

Create a line feed.
The operator \texttt{Fnewline} puts out a line feed into the output file. At the same time the output buffer is cleaned.

\begin{itemize}
\item \textbf{Attention}
\end{itemize}

\begin{itemize}
\item \textbf{Parameter}
\end{itemize}

\begin{enumerate}
\item \texttt{FileHandle} (input control) \quad \texttt{HFileX / long}
\item \texttt{Char} (output control) \quad \texttt{String}
\end{enumerate}

\begin{itemize}
\item \textbf{Example}
\end{itemize}

\begin{enumerate}
\item \texttt{fwrite_string(FileHandle,’Good Morning’)}
\item \texttt{fnewline(FileHandle)}
\end{enumerate}

\begin{itemize}
\item \textbf{Result}
\end{itemize}

If an output file is open and it can be written to the file the operator \texttt{Fnewline} returns the value \texttt{TRUE}. Otherwise an exception handling is raised.

\begin{itemize}
\item \textbf{Parallelization Information}
\end{itemize}

\texttt{Fnewline} is \textit{reentrant} and processed \textit{without} parallelization.

\begin{itemize}
\item \textbf{Possible Predecessor Functions}
\end{itemize}

\texttt{FwriteString}

\begin{itemize}
\item \textbf{See Also}
\end{itemize}

\texttt{FwriteString}

\begin{itemize}
\item \textbf{Module}
\end{itemize}

Basic operators

```c
[out] String Char HFileX.FreadChar ( )
void HOperatorSetX.FreadChar ( [in] long FileHandle, [out] String Char )
```

Read a character from a text file.
The operator \texttt{FreadChar} reads a character from the current input file. \texttt{FreadChar} returns the character sequence \texttt{’eof’}. At the end of a line the value \texttt{’nl’} is returned.

\begin{itemize}
\item \textbf{Attention}
\end{itemize}

\begin{itemize}
\item \textbf{Parameter}
\end{itemize}

\begin{enumerate}
\item \texttt{FileHandle} (input control) \quad \texttt{HFileX / long}
\item \texttt{Char} (output control) \quad \texttt{String}
\end{enumerate}

\begin{itemize}
\item \textbf{Example}
\end{itemize}

\begin{enumerate}
\item \texttt{fread_char(FileHandle:Char)}
\item \texttt{(if(Char = ’nl’) \rightarrow fnewline(FileHandle)) |}
\item \texttt{(if(Char != ’nl’) \rightarrow fwrite_string(FileHandle,Char))}
\item until(Char = ’eof’).
\end{enumerate}
If an input file is open the operator `fread_char` returns `TRUE`. Otherwise an exception handling is raised.

**Parallelization Information**

`freadChar` is reentrant and processed without parallelization.

**Possible Predecessor Functions**

- `OpenFile`
- `CloseFile`

**Possible Successor Functions**

- `See Also`
  - `OpenFile`, `CloseFile`, `FreadChar`

**Alternatives**

- `FreadString`, `ReadString`

**Module**

Basic operators

---

### Examples

1. `fwrite_string(FileHandle,'Please enter text and return: ..')`
2. `fread_string(FileHandle,String,IsEOF) >`
3. `fwrite_string(FileHandle,['here it is again: ',String])`
4. `fnew_line(FileHandle)`

**Result**

If a file is open and a suitable string is read `FreadString` returns the value `TRUE`. Otherwise an exception handling is raised.

**Parallelization Information**

`FreadString` is reentrant and processed without parallelization.

**Possible Predecessor Functions**

- `OpenFile`

**Possible Successor Functions**

- `CloseFile`

**See Also**

- `OpenFile, CloseFile, FreadChar`

---

**HFileX.FreadString**

```cpp
[out] String OutString
[in] long FileHandle,
[out] long IsEOF
```

Read strings from a text file.

The operator `FreadString` reads a string from the current input file. A string begins with the first representable character: letters, numbers, additional characters (except blanks). A string ends when a blank or a line skip is reached. Several successive line skips are ignored. If the end of the file is reached `IsEOF` return the value `1`, otherwise `0`.

**Parameter**

- **FileHandle** (input control) .................. file \(\sim\) `HFileX`/`long` File handle.
- **OutString** (output control) .................. string \(\sim\) `String` Read character sequence.
- **IsEOF** (output control) .................. integer \(\sim\) `long` Reached end of file.

**Example**

```cpp
fwrite_string(FileHandle,'Please enter text and return: ..')
fread_string(FileHandle,String,IsEOF) >
fwrite_string(FileHandle,['here it is again: ',String])
fnew_line(FileHandle).
```

**Result**

If a file is open and a suitable string is read `FreadString` returns the value `TRUE`. Otherwise an exception handling is raised.
Write values in a text file.

The operator `FwriteString` puts out a string or numbers on the output file. The operator `OpenFile` opens a file. The call `SetSystem (::'flushFile', <boolean-value>;)` determines whether the output characters are put out directly on the output medium. If the value 'flush_file' is set to 'false' the characters (especially in case of screen output) show up only after the operator `FnewLine` is called.

Strings as well as whole numbers and floating point numbers can be used as arguments. If more than one value serves as input the values are put out consecutively without blanks.

```
void HFileX.FwriteString ([in] VARIANT String )
void HOperatorSetX.FwriteString ([in] long FileHandle, [in] VARIANT String )
```

### Example

```c
fwrite_string(FileHandle,['text with numbers:',5,' and ',1.0]).
/* results in the following output: */
/* 'text with numbers:5 and 1.00000' */
```

### Result

If the writing procedure was carried out successfully the operator `FwriteString` returns the value TRUE. Otherwise an exception handling is raised.

### Parallelization Information

`FwriteString` is reentrant and processed without parallelization.

### Possible Predecessor Functions

- `OpenFile`

### Possible Successor Functions

- `CloseFile`
- `OpenFile,CloseFile,SetSystem`

### Alternatives

- `WriteString`

### Module

```
void HFileX.OpenFile ([in] String FileName, [in] String FileType )
void HOperatorSetX.OpenFile ([in] String FileName, [in] String FileType, [out] long FileHandle )
```

Open text file.
The operator OpenFile opens a file. FileType determines whether this file is an input ('input') or output file ('output'). Text files which can be accessed either by reading ('input') or by writing ('output') are created. For terminal input and output the file names 'standard' ('input' and 'output') and 'error' (only 'output') are reserved.

### Attention

### Parameter

- **FileName** (input control)  
  Name of file to be opened.  
  **Default Value:** 'standard'  
  **Value Suggestions:** FileName ∈ {'standard', 'error', '/tmp/dat.dat'}

- **FileType** (input control)  
  Type of file.  
  **Default Value:** 'output'  
  **Value List:** FileType ∈ {'input', 'output'}

- **FileHandle** (output control)  
  File handle.  
  **Value:** file ∼ HFileX / long

### Example

```c
/* Creating of an output file with the name '/tmp/log.txt' and writing */
/* of one string: */
open_file('/tmp/log.txt','output',FileHandle)
fwrite_string(FileHandle,'these are the first and last lines')
fnew_line(FileHandle)
close_file(FileHandle).
```

### Result

If the parameters are correct the operator OpenFile returns the value TRUE. Otherwise an exception handling is raised.

### Parallelization Information

OpenFile is local and processed completely exclusively without parallelization.

### Possible Predecessor Functions

- ResetObjDb

### Possible Successor Functions

- FWriteString, FReadChar, FReadString, CloseFile

### See Also

- CloseFile, FWriteString, FReadChar, FReadString

### Module

Basic operators

### 2.5 Tuple

```c
[out] VARIANT Tuple HMiscX.ReadTuple ([in] String FileName )
void HOperatorSetX.ReadTuple ([in] String FileName, [out] VARIANT Tuple )
```

**Read a tuple from a file.**

The operator ReadTuple reads the contents of FileName and converts it into Tuple. The file has to be generated by WriteTuple.
2.6 XLD

Parameter

- **FileName** (input control) ........................................................ filename $\rightarrow$ String
  Name of the file to be read.
- **Tuple** (output control) ........................................ number(-array) $\rightarrow$ VARIANT ( real, integer, string )
  Tuple with any kind of data.

Result

If the parameters are correct the operator **ReadTuple** returns the value TRUE. If the file could not be opened **ReadTuple** returns FAIL. Otherwise an exception handling is raised.

Parallelization Information

ReadTuple is reentrant and processed without parallelization.

See Also

WriteTuple, GnuplotPlotCtrl, WriteImage, WriteRegion, OpenFile

Alternatives

** السنوات **

Module

Operators not requiring licensing

void **HMiscX.WriteTuple** ([in] VARIANT Tuple, [in] String FileName )

void **HOperatorSetX.WriteTuple** ([in] VARIANT Tuple, [in] String FileName )

Write a tuple to a file.

The operator **WriteTuple** writes the contents of Tuple to a file. The data is written in an ASCII format. Therefore, the file can be exchanged between different architectures. There is no specific extension for this kind of file.

Parameter

- **Tuple** (input control) ........................................ number(-array) $\rightarrow$ VARIANT ( real, integer, string )
  Tuple with any kind of data.
- **FileName** (input control) ................................ filename $\rightarrow$ String
  Name of the file to be written.

Result

If the parameters are correct the operator **WriteTuple** returns the value TRUE. If the file could not be opened **WriteTuple** returns FAIL. Otherwise an exception handling is raised.

Parallelization Information

WriteTuple is reentrant and processed without parallelization.

See Also

ReadTuple, WriteImage, WriteRegion, OpenFile

Alternatives

** السنوات **

Module

Operators not requiring licensing

2.6 XLD

void **HXLDContX.ReadContourXldArcInfo** ([in] String FileName )

void **HOperatorSetX.ReadContourXldArcInfo** ([out] HUntypedObjectX Contours, [in] String FileName )

Read XLD contours to a file in ARC/INFO generate format.
ReadContourXldArcInfo reads the lines stored in ARC/INFO generate format in the file FileName, and returns them as XLD contours in Contours. To find the file FileName, all directories contained in the HALCON system variable 'image_dir' (usually this is the content of the environment variable HALCONIMAGES) are searched (see ReadImage). The returned contours are in world coordinates. They can be transformed to the image coordinate system with the operator AffineTransContourXld. The necessary transformation matrix can be generated by using ReadWorldFile to read the transformation matrix from image to world coordinates, and inverting this matrix by calling HomMat2dInvert.

Parameter

- **Contours** (output iconic) ......................... xld_cont(-array) \(\sim\) HXLDContX / HUntypedObjectX
  Read XLD contours.
- **FileName** (input control) .......................... filename \(\sim\) String
  Name of the ARC/INFO file.

Example

```c++
/* Read the transformation and invert it */
read_world_file ('image.tfw', WorldTransformation)
hom_mat2d_invert (WorldTransformation, ImageTransformation)
/* Read the image */
read_image (Image, ‘image.tif’)
/* Read the line data */
read_contour_xld_arc_info (LinesWorld, ‘lines.gen’)
/* Transform the line data to image coordinates */
affine_trans_contour_xld (LinesWorld, Lines, ImageTransformation)
```

Result

If the parameters are correct and the file could be read, the operator ReadContourXldArcInfo returns the value TRUE. Otherwise an exception is raised.

Parallelization Information
ReadContourXldArcInfo is reentrant and processed without parallelization.

Possible Successor Functions
HomMat2dInvert, AffineTransContourXld

See Also
ReadWorldFile, WriteContourXldArcInfo, ReadPolygonXldArcInfo

Module
Sub-pixel operators

---

void HXLDPolyX.ReadPolygonXldArcInfo ([in] String FileName )
void HOperatorSetX.ReadPolygonXldArcInfo
([out] HUntypedObjectX Polygons, [in] String FileName )

Read XLD polygons from a file in ARC/INFO generate format.

ReadPolygonXldArcInfo reads the lines stored in ARC/INFO generate format in the file FileName, and returns them as XLD polygons in Polygons. To find the file FileName, all directories contained in the HALCON system variable "image_dir" (usually this is the content of the environment variable HALCONIMAGES) are searched (see ReadImage). The returned polygons are in world coordinates. They can be transformed to the image coordinate system with the operator AffineTransPolygonXld. The necessary transformation matrix can be generated by using ReadWorldFile to read the transformation matrix from image to world coordinates, and inverting this matrix by calling HomMat2dInvert.

Parameter

- **Polygons** (output iconic) ......................... xld_poly(-array) \(\sim\) HXLDPolyX / HUntypedObjectX
  Read XLD polygons.
- **FileName** (input control) .......................... filename \(\sim\) String
  Name of the ARC/INFO file.
/* Read the transformation and invert it */
read_world_file (’image.tfw’, WorldTransformation)
hom_mat2d_invert (WorldTransformation, Image Transformation)
/* Read the image */
read_image (Image, ’image.tif’)
/* Read the line data */
read_polygon_xld_arc_info (LinesWorld, ’lines.gen’)
/* Transform the line data to image coordinates */
affine_trans_polygon_xld (LinesWorld, Lines, ImageTransformation)

Result
If the parameters are correct and the file could be read, the operator ReadPolygonXldArcInfo returns the value TRUE. Otherwise an exception is raised.

Parallelization Information
ReadPolygonXldArcInfo is reentrant and processed without parallelization.

Possible Successor Functions
HomMat2dInvert, AffineTransPolygonXld

See Also
ReadWorldFile, WritePolygonXldArcInfo, ReadContourXldArcInfo

Module
Sub-pixel operators

void HXLDContX.WriteContourXldArcInfo ([in] String FileName )

void HOperatorSetX.WriteContourXldArcInfo ([in] IObjectX Contours, [in] String FileName )

Write XLD contours to a file in ARC/INFO generate format.

WriteContourXldArcInfo writes the XLD contours Contours to an ARC/INFO generate format file with name FileName. If no absolute path is given in FileName, the output file is created in the current directory of the HALCON process. The contours must have been transformed to the world coordinate system with AffineTransContourXld beforehand. The necessary transformation can be read from an ARC/INFO world file with ReadWorldFile.

Parameter

♦ Contours (input iconic) xld(-array) ~ HXLDContX / IObjectX
XLD contours to be written.

♦ FileName (input control) filename ~ String
Name of the ARC/INFO file.

Example

/* Read transformation and image */
read_world_file (’image.tfw’, WorldTransformation)
read_image (Image, ’image.tif’)
/* Segment image */
...
/* Write result */
affine_trans_contour_xld (Contours, ContoursWorld, WorldTransformation)
write_contour_xld_arc_info (ContoursWorld, ’result.gen’)

Result
If the parameters are correct and the file could be written, the operator WriteContourXldArcInfo returns the value TRUE. Otherwise an exception is raised.
WriteContourXldArcInfo is reentrant and processed without parallelization.

Possible Predecessor Functions

AffineTransContourXld

See Also

ReadWorldFile, ReadContourXldArcInfo, WritePolygonXldArcInfo

Sub-pixel operators

void HXLDPolyX.WritePolygonXldArcInfo ([in] String FileName )
void HOperatorSetX.WritePolygonXldArcInfo ([in] IOBjectX Polygons,
[in] String FileName )

Write XLD polygons to a file in ARC/INFO generate format.

WritePolygonXldArcInfo writes the XLD polygons Polygons to an ARC/INFO generate format file with name FileName. If no absolute path is given in FileName, the output file is created in the current directory of the HALCON process. The polygons must have been transformed to the world coordinate system with AffineTransPolygonXld beforehand. The necessary transformation can be read from an ARC/INFO world file with ReadWorldFile.

Attention

The XLD contours that are possibly referenced by Polygons are not stored in the ARC/INFO file, since this is not possible with the ARC/INFO generate file format. Therefore, when the polygons are read again using ReadPolygonXldArcInfo, this information is lost, and no references to contours are generated for the polygons. Hence, operators that access the contours associated with a polygon, e.g., SplitContoursXld will not work correctly.

Parameter

- Polygons (input iconic) xld_poly(-array) \( \rightarrow \) HXLDPolyX / IOBjectX
- FileName (input control) String

Example

```c
/* Read transformation and image */
read_world_file ('image.tfw', WorldTransformation)
read_image (Image, 'image.tif')
/* Segment image */
...
/* Write result */
affine_trans_polygon_xld (Polygons, PolygonsWorld, WorldTransformation)
write_polygon_xld_arc_info (PolygonsWorld, 'result.gen')
```

Result

If the parameters are correct and the file could be written, the operator WritePolygonXldArcInfo returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

WritePolygonXldArcInfo is reentrant and processed without parallelization.

Possible Predecessor Functions

AffineTransPolygonXld

See Also

ReadWorldFile, ReadPolygonXldArcInfo, WriteContourXldArcInfo

Sub-pixel operators
Chapter 3

Filter

3.1 Affine-Transformations

Apply an arbitrary affine transformation to an image.

\texttt{AffineTransImage} applies an arbitrary affine transformation, i.e., scaling, rotation, translation, and skewing, to an image. The affine map is described by the transformation matrix given in \texttt{HomMat2D}, which is built up by using \texttt{HomMat2dIdentity}, \texttt{HomMat2dScale}, \texttt{HomMat2dRotate}, and \texttt{HomMat2dTranslate}. The components of the homogeneous transformation matrix are interpreted as follows: the row coordinate of the image corresponds to the \texttt{x} coordinate of the matrix, while the \texttt{column} coordinate of the image corresponds to the \texttt{y} coordinate of the matrix. This is necessary to obtain a right-handed coordinate system, which is assumed for the homogeneous transformation matrices, also for the image. In particular, by doing so rotations are performed in the correct direction. Note that the \((x,y)\) order of the matrices quite naturally corresponds to the usual \((row,column)\) order for coordinates in the image.

The region of the input image is ignored, i.e., assumed to be the full rectangle of the image. The region of the resulting image is set to the transformed rectangle of the input image. If necessary, the resulting image is filled with zero (black) outside of the region of the original image.

Generally, transformed points will lie between pixel coordinates. Therefore, an appropriate interpolation scheme has to be used. The interpolation can also be used to avoid aliasing effects for scaled images. The quality and speed of the interpolation can be set by the parameter \texttt{Interpolation}:

- **none**: No interpolation: The gray value is determined from the nearest pixel's gray value (possibly low quality, very fast).
- **constant**: Use equally weighted interpolation between adjacent pixels (medium quality and run time).
- **weighted**: Use Gaussian interpolation between adjacent pixels (best quality, slow).

In addition, the system parameter \texttt{'int\_zooming'} (see \texttt{SetSystem}) affects the accuracy of the transformation. If \texttt{'int\_zooming'} is set to \texttt{'true'}, the transformation for byte and int2 images is carried out internally using fixed point arithmetic, leading to much shorter execution times. However, the accuracy of the transformed gray values is smaller in this case. For byte images, the differences to the more accurate calculation (using \texttt{'int\_zooming' = 'false'}) is typically less than two gray levels. Correspondingly, for int2 images, the gray value differences are less than 1/128 times the dynamic gray value range of the image, i.e., they can be as large as 512 gray levels if the entire dynamic range of 16 bit is used. For real images, the parameter \texttt{'int\_zooming'} does not affect the accuracy, since the internal calculations are always done using floating point arithmetic.
The size of the target image can be controlled by the parameter `AdaptImageSize`: With value `true` the size will be adapted so that no clipping occurs. With value `false` the target image has the same size as the input image.

The region of the input image is ignored.

The components of the homogeneous transformation matrix are interpreted as follows: the row coordinate of the image corresponds to the \( x \) coordinate of the matrix, while the column coordinate of the image corresponds to the \( y \) coordinate of the matrix. This is necessary to obtain a right-handed coordinate system, which is assumed for the homogeneous transformation matrices, also for the image. In particular, by doing so rotations are performed in the correct direction. Note that the \((x,y)\) order of the matrices quite naturally corresponds to the usual \((\text{row, column})\) order for coordinates in the image.

---

**Example**

/* Reduction of an image (512 x 512 Pixels) by 50%, rotation */
/* by 180 degrees and translation to the upper-left corner: */

```cpp
hom_mat2d_identity(Matrix1)
hom_mat2d_scale(Matrix1, 0.5, 0.5, 256.0, 256.0, Matrix2)
hom_mat2d_rotate(Matrix2, 3.14, 256.0, 256.0, Matrix3)
hom_mat2d_translate(Matrix3, -128.0, -128.0, Matrix4,)
affine_trans_image(Image, TransImage, Matrix4, 1).
```

/* Enlarging the part of an image in the interactively */
/* chosen rectangular window sector: */

draw_rectangle2(WindowHandle, L, C, Phi, L1, L2)
hom_mat2d_identity(Matrix1)
get_system(width, Width)
get_system(height, Height)
hom_mat2d_translate(Matrix1, Height/2.0-L, Width/2.0-C, Matrix2)
hom_mat2d_rotate(Matrix2, 3.14-Phi, Height/2.0, Width/2.0, Matrix3)
hom_mat2d_scale(Matrix3, Height/(2.0*L2), Width/(2.0*L1),
    Height/2.0, Width/2.0, Matrix4)
```

**Result**

`AffineTransImage` returns TRUE if all parameter values are correct. If the input is empty the behaviour can be set via `SetSystem(':noObjectResult:',<Result>:)`. If necessary, an exception handling is raised.

---

**Parallelization Information**

`AffineTransImage` is reentrant and automatically parallelized (on tuple level, channel level).

### Possible Predecessor Functions

- HomMat2dIdentity
- HomMat2dTranslate
- HomMat2dRotate
- HomMat2dScale

### See Also

- SetPartStyle

### Alternatives

- ZoomImageSize
- ZoomImageFactor
- MirrorImage
- RotateImage
- AffineTransRegion

### Module

**Image filters**

```c
[out] HImageX ImageMirror
HImageX.MirrorImage ([in] String Mode )
```

```c
void HOperatorSetX.MirrorImage ([in] IOBJECTX Image,
[out] HUNTYPEDOBJECTX ImageMirror, [in] String Mode )
```

**Mirror an image.**

MirrorImage reflects an image `Image` about one of three possible axes. If `Mode` is set to 'row', it is reflected about the horizontal axis, if `Mode` is set to 'column', about the vertical axis, and if `Mode` is set to 'main', about the main diagonal.

**Parameter**

- **Image** (input iconic) (multichannel-)image-array
  - `HImageX / IOBJECTX ( byte, int2 )`
  - Input image.

- **ImageMirror** (output iconic) (multichannel-)image-array
  - `HImageX / HUNTYPEDOBJECTX ( byte, int2 )`
  - Reflected image.

- **Mode** (input control) string
  - `String`
  - Axis of reflection.
  - **Default Value**: 'row'
  - **Value List**: `Mode ∈ { 'row', 'column', 'main' }`

**Example**

```c
color = read_image(Image,'affe')
disp_image(Image,WindowHandle)
mirror_image(Image,MirImage,'row').
disp_image(MirImage,WindowHandle)
```

**Parallelization Information**

MirrorImage is **reentrant** and automatically **parallelized** (on tuple level, channel level).

### See Also

- RotateImage
- HomMat2dRotate

### Alternatives

- HomMat2dRotate
- AffineTransImage
- RotateImage

### Module

**Image filters**

```c
[out] HImageX ImagePolar
HImageX.PolarTransImage ([in] long Row,
```

```c
void HOperatorSetX.PolarTransImage ([in] IOBJECTX ImageXY,
[out] HUNTYPEDOBJECTX ImagePolar, [in] long Row, [in] long Column,
[in] long Width, [in] long Height )
```

**Transform an image to polar coordinates**
PolarTransImage transforms an image in cartesian coordinates to an image in polar coordinates. The size of the resulting image is selected with \textit{Width} and \textit{Height}. \textit{Width} determines the angular resolution, while \textit{Height} determines the resolution of the radius. \textit{Row} and \textit{Column} determine the center of the polar coordinate system in the original image \textit{ImageXY}. This point is mapped to the upper row of \textit{ImagePolar}.

A point \((x',y')\) in the result image corresponds to the point \((x,y)\) in the original image in the following manner:

\[
x = y' \cos(2\pi (x' / \text{resultwidth})) + \text{Column} = y' \sin(2\pi (x' / \text{resultwidth})) + \text{Row}.
\]

\begin{itemize}
  \item \textbf{ImageXY} (input iconic) \dots (multichannel-image-array) \sim HImageX / IObjectX (byte, int2)
    Input image in cartesian coordinates.
  \item \textbf{ImagePolar} (output iconic) \dots (multichannel-image-array) \sim HImageX / HUntypedObjectX (byte, int2)
    Result image in polar coordinates.
  \item \textbf{Row} (input control) \dots \texttt{point.y} \sim \texttt{long}
    Row coordinate of the center of the coordinate system.
    \begin{itemize}
      \item Default Value: 100
      \item Value Suggestions: \( \text{Row} \in \{ 0, 10, 100, 200 \} \)
      \item Typical Range of Values: \( 0 \leq \text{Row} \leq 0 \)
      \item Minimal Value Step: 1
      \item Recommended Value Step: 1
    \end{itemize}
  \item \textbf{Column} (input control) \dots \texttt{point.x} \sim \texttt{long}
    Column coordinate of the center of the coordinate system.
    \begin{itemize}
      \item Default Value: 100
      \item Value Suggestions: \( \text{Column} \in \{ 0, 10, 100, 200 \} \)
      \item Typical Range of Values: \( 0 \leq \text{Column} \leq 0 \)
      \item Minimal Value Step: 1
      \item Recommended Value Step: 1
    \end{itemize}
  \item \textbf{Width} (input control) \dots \texttt{extent.x} \sim \texttt{long}
    Width of the result image.
    \begin{itemize}
      \item Default Value: 314
      \item Value Suggestions: \( \text{Width} \in \{ 100, 200, 157, 314, 512 \} \)
      \item Typical Range of Values: \( 2 \leq \text{Width} \leq 2 \)
      \item Minimal Value Step: 1
      \item Recommended Value Step: 10
    \end{itemize}
  \item \textbf{Height} (input control) \dots \texttt{extent.y} \sim \texttt{long}
    Height of the result image.
    \begin{itemize}
      \item Default Value: 200
      \item Value Suggestions: \( \text{Height} \in \{ 100, 128, 256, 512 \} \)
      \item Typical Range of Values: \( 2 \leq \text{Height} \leq 2 \)
      \item Minimal Value Step: 1
      \item Recommended Value Step: 10
    \end{itemize}
\end{itemize}

\textbf{Example}

\begin{verbatim}
read_image(Image,'affe')
disp_image(Image,WindowHandle)
polar_trans_image(Image,PolarImage,100,100,314,200).
disp_image(PolarImage,WindowHandle)
\end{verbatim}

\textbf{Parallelization Information}

PolarTransImage is \textit{reentrant} and automatically \textit{parallelized} (on tuple level, channel level).

\textbf{Alternatives}

\textit{AffineTransImage}

\textbf{Module}

Image filters
**3.1. AFFINE-TRANSFORMATIONS**

Rotate an image about its center.

RotateImage rotates the image Image counterclockwise by Phi degrees about its center. This operator is much faster if Phi is a multiple of 90 degrees than the general operator AffineTransImage. For rotations by 90, 180, and 270 degrees, the region is rotated accordingly. For all other rotations the region is set to the maximum region, i.e., to the extent of the resulting image. The effect of the parameter Interpolation is the same as in AffineTransImage. It is ignored for rotations by 90, 180, and 270 degrees. The size of the resulting image is the same as that of the input image, with the exception of rotations by 90 and 270 degrees, where the width and height will be exchanged.

---

**Attention**
The angle Phi is given in degrees, not in radians.

---

**Parameter**

- **Image** (input iconic) ............... (multichannel-)image(-array) → HImageX / IObjectX (byte, int2)
  Input image.

- **ImageRotate** (output iconic) (multichannel-)image(-array) → HImageX / HUntypedObjectX (byte, int2)
  Rotated image.

- **Phi** (input control) ............................................ angle.deg → VARIANT (integer, real)
  Rotation angle.
  Default Value: 90
  Value Suggestions: Phi ∈ {90, 180, 270}
  Typical Range of Values: 0 ≤ Phi ≤ 0
  Minimal Value Step: 0.001
  Recommended Value Step: 0.2

- **Interpolation** (input control) ................................................... string → String
  Type of interpolation.
  Default Value: ‘constant’
  Value List: Interpolation ∈ {‘none’, ‘constant’, ‘weighted’}

---

**Example**

```plaintext
read_image(Image,'affe')
disp_image(Image,WindowHandle)
rotate_image(ImageRotImage,270).
disp_image(RotImage,WindowHandle)
```

---

**Parallelization Information**

RotateImage is reentrant and automatically parallelized (on tuple level, channel level).

---

**See Also**

MirrorImage

**Alternatives**

HomMat2dRotate, AffineTransImage

---

Image filters

HALCON 6.0
Zoom an image by a given factor.

`ZoomImageFactor` scales the image `Image` by a factor of `ScaleWidth` in width and a factor `ScaleHeight` in height. The parameter `Interpolation` determines the type of interpolation used (see `AffineTransImage`).

**Parameter**

- **Image** (input iconic) .................................................. (multichannel-)image(-array)  ~  `HImageX / IObjectX` (byte, int2)
  - Input image.
- **ImageZoomed** (output iconic) (multichannel-)image(-array)  ~  `HImageX / HUntypedObjectX` (byte, int2)
  - Scaled image.
- **ScaleWidth** (input control) ............................................. extent.x  ~  `double`
  - Scale factor for the width of the image.
    - Default Value : 0.5
    - Value Suggestions : `ScaleWidth ∈ {0.25, 0.5, 1.5, 2.0}`
    - Typical Range of Values : `0.001 ≤ ScaleWidth ≤ 0.001`
    - Minimal Value Step : 0.001
    - Recommended Value Step : 0.1
- **ScaleHeight** (input control) .......................................... extent.y  ~  `double`
  - Scale factor for the height of the image.
    - Default Value : 0.5
    - Value Suggestions : `ScaleHeight ∈ {0.25, 0.5, 1.5, 2.0}`
    - Typical Range of Values : `0.001 ≤ ScaleHeight ≤ 0.001`
    - Minimal Value Step : 0.001
    - Recommended Value Step : 0.1
- **Interpolation** (input control) ....................................... string  ~  `String`
  - Type of interpolation.
    - Default Value : 'constant'
    - Value List : `Interpolation ∈ {'none', 'constant', 'weighted'}`

**Example**

```plaintext```
read_image(Image,'affe')
disp_image(Image,WindowHandle)
zoom_image_factor(Image,ZooImage,0,0.5,0.5).
disp_image(ZooImage,WindowHandle)
```

**Parallelization Information**

`ZoomImageFactor` is reentrant and automatically parallelized (on tuple level, channel level).

**See Also**

- `HomMat2dScale`, `AffineTransImage`

**Alternatives**

- `ZoomImageFactor`, `AffineTransImage`, `HomMat2dScale`

**Module**

Image filters
3.1. AFFINE-TRANSFORMATIONS


Zoom an image to a given size.

ZoomImageSize scales the image Image to the size given by Width and Height. The parameter Interpolation determines the type of interpolation used (see AffineTransImage).

Parameter

- **Image** (input iconic) .................. (multichannel-)image(-array) ~ HImageX / IHObjectX ( byte, int2 )
  Input image.

- **ImageZoom** (output iconic) ... (multichannel-)image(-array) ~ HImageX / HUntypedObjectX ( byte, int2 )
  Scaled image.

- **Width** (input control) ........................................................ extent.x ~ long
  Width of the resulting image.
  Default Value : 512
  Value Suggestions : Width ∈ {128, 256, 512}
  Typical Range of Values : 2 ≤ Width ≤ 2
  Minimal Value Step : 1
  Recommended Value Step : 10

- **Height** (input control) ........................................................ extent.y ~ long
  Height of the resulting image.
  Default Value : 512
  Value Suggestions : Height ∈ {128, 256, 512}
  Typical Range of Values : 2 ≤ Height ≤ 2
  Minimal Value Step : 1
  Recommended Value Step : 10

- **Interpolation** (input control) ................................................. string ~ String
  Type of interpolation.
  Default Value : 'constant'
  Value List : Interpolation ∈ {'none', 'constant', 'weighted'}

Example

read_image(Image,'affe')
disp_image(Image,WindowHandle)
zoom_image_size(Image,ZooImage,0,200,200).
disp_image(ZooImage,WindowHandle)

Parallelization Information

ZoomImageSize is **reentrant** and automatically **parallelized** (on tuple level, channel level).

See Also

HomMat2dScale, AffineTransImage

Alternatives

ZoomImageFactor, AffineTransImage, HomMat2dScale

Module

Image filters
3.2 Arithmetic

```c
void HOperatorSetX.AbsImage ([in] IObjectX Image,
[out] HUntypedObjectX ImageAbs )
```

Calculate the absolute value (modulus) of an image.

The operator AbsImage calculates the absolute gray values of images of any type and stores the result in ImageAbs. The power spectrum of complex images is calculated as a ‘real’ image. The operator AbsImage generates a logical copy of unsigned images.

Parameter

- **Image** (input iconic) ................. (multichannel-)image(array) → HImageX / IObjectX ( any ) Image(s) for which the absolute gray values are to be calculated.
- **ImageAbs** (output iconic) ........... (multichannel-)image(array) → HImageX / HUntypedObjectX ( any ) Result image(s).

The operator AbsImage returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem::<noObjectResult',<Result>::>

Parallelization Information

AbsImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See Also

ConvertImageType, PowerByte

Image filters

```c
void HOperatorSetX.AddImage ([in] IImageX Image2,
```

Add two images.

The operator AddImage adds two images. The gray values \((g_1, g_2)\) of the input images \((\text{Image1} \text{ and Image2})\) are transformed as follows:

\[
g' := (g_1 + g_2) \times \text{Mult} + \text{Add}
\]

If an overflow or an underflow occurs the values are clipped. This is not the case with int2 images if Mult is equal to 1 and Add is equal to 0. To reduce the runtime the underflow and overflow check is skipped. The resulting image is stored in ImageResult.

It is possible to add byte images with int2 and int4 images. In this case the result will be of type int2 or int4 respectively.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

Please note that the runtime of the operator varies with different control parameters. For frequently used combinations special optimizations were used.

Parameter

- **Image1** (input iconic) ..(multichannel-)image(array) → HImageX / IObjectX ( byte, int1, int2, int4, real, direction, cyclic, complex ) Image(s) 1.
- **Image2** (input iconic) ..(multichannel-)image(array) → HImageX / IObjectX ( byte, int1, int2, int4, real, direction, cyclic, complex ) Image(s) 2.
3.2. ARITHMETIC

Result image(s) by the addition.

\[ \text{Result} \]

\[ \text{Mult} \]

Factor for gray value adaption.

Default Value : 0.5

Value Suggestions : Mult \( \in \{ 0.2, 0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 5.0 \} \)

Typical Range of Values : \(-255.0 \leq \text{Mult} \leq 255.0\)

Minimal Value Step : 0.001

Recommended Value Step : 0.1

\[ \text{Add} \]

Value for gray value range adaption.

Default Value : 0

Value Suggestions : Add \( \in \{ 0, 64, 128, 255, 512 \} \)

Typical Range of Values : \(-512.0 \leq \text{Add} \leq 512.0\)

Minimal Value Step : 0.01

Recommended Value Step : 1.0

Example

```
read_image(Image0,"fabrik")
disp_image(Image0,WindowHandle)
read_image(Image1,"Affe")
disp_image(Image1,WindowHandle)
add_image(Image0,Image1,Result,2.0,10.0)
disp_image(Result,WindowHandle)
```

Result

The operator \texttt{AddImage} returns the value TRUE if the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem}(::'noObjectResult',<Result>:)

If necessary an exception handling is raised.

Parallelization Information

\texttt{AddImage} is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See Also

\texttt{SubImage, MultImage}

Alternatives

\texttt{SubImage, MultImage}

Module

\texttt{Image filters}

```
```

Divide two images.

The operator \texttt{DivImage} divides two images. The gray values \( (g1, g2) \) of the input images \( (\text{Image1}) \) are transformed as follows:

\[
g' := g1/g2 \times \text{Mult} + \text{Add}
\]

If an overflow or an underflow occurs the values are clipped.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.
### Parameter

- **Image1** (input iconic) \((\text{multichannel-image-array}) \Rightarrow HImageX / HObjectX (\text{byte, int1, int2, int4, real, complex})\)
  
  Image(s) 1.

- **Image2** (input iconic) \((\text{multichannel-image-array}) \Rightarrow HImageX / HObjectX (\text{byte, int1, int2, int4, real, complex})\)
  
  Image(s) 2.

- **ImageResult** (output iconic) \((\text{multichannel-image-array}) \Rightarrow HImageX / HUntypedObjectX (\text{byte, int1, int2, int4, real, complex})\)
  
  Result image(s) by the division.

- **Mult** (input control) \(~ \text{number} \Rightarrow \text{VARIANT (integer, real)}\)
  
  Factor for gray range adaption.
  
  **Default Value**: 255
  
  **Value Suggestions**: Mult \(\in \{0.1, 0.2, 0.5, 1.0, 2.0, 3.0, 10, 100, 500, 1000\}\)
  
  **Typical Range of Values**: \(-1000 \leq \text{Mult} \leq -1000\)
  
  **Minimal Value Step**: 0.001
  
  **Recommended Value Step**: 1

- **Add** (input control) \(~ \text{number} \Rightarrow \text{VARIANT (integer, real)}\)
  
  Value for gray range adaption.
  
  **Default Value**: 0
  
  **Value Suggestions**: Add \(\in \{0.0, 128.0, 256.0, 1025\}\)
  
  **Typical Range of Values**: \(-1000 \leq \text{Add} \leq -1000\)
  
  **Minimal Value Step**: 0.01
  
  **Recommended Value Step**: 1.0

### Example

```plaintext
read_image(Image0,"fabrik")
disp_image(Image0,WindowHandle)
read_image(Image1,"Affe")
disp_image(Image1,WindowHandle)
div_image(Image0,Image1,Result,2.0,10.0)
disp_image(Result,WindowHandle)
```

### Result

The operator `DivImage` returns the value `TRUE` if the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator `SetSystem(::'noObjectResult',<Result>:)` If necessary an exception handling is raised.

### Parallelization Information

`DivImage` is **reentrant** and automatically **parallelized** (on tuple level, channel level, domain level).

### See Also

AddImage, SubImage, MultImage

### Alternatives

AddImage, SubImage, MultImage

### Module

Image filters

---

```plaintext
[out] HImageX ImageInvert HImageX.InvertImage ()

void HOperatorSetX.InvertImage ([in] HObjectX Image,
[out] HUntypedObjectX ImageInvert )
```

Invert an image.

The operator `InvertImage` inverts the gray values of an image. For images of the 'byte' and 'cyclic' type the result is calculated as:

3.2. ARITHMETIC

\[ g' = 255 - g \]

Images of the 'direction' type are transformed by

\[ g' = (g + 90) \mod 180 \]

In the case of signed types the values are negated. The resulting image has the same pixel type as the input image. Several images can be processed in one call. An output image is generated for every input image.

**Parameter**

- **Image** (input iconic) . . (multichannel-)image(array) \( \sim HImageX / IHObjectX\) (byte, int1, int2, int4, real, cyclic, direction)
  - Input image(s).
- **ImageInvert** (output iconic) . (multichannel-)image(array) \( \sim HImageX / HUnTypedObjectX\) (byte, int1, int2, int4, real, cyclic, direction)
  - Image(s) with inverted gray values.

**Example**

```c
read_image(Orig, "fabrik")
invert_image(Orig, Invert)
disp_image(Invert, WindowHandle).
```

**Parallelization Information**

InvertImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Successor Functions**

ScaleImage, AddImage, SubImage

**See Also**

ScaleImage

**Alternatives**

ScaleImage

**Module**

Image filters

```
[out] HImageX ImageMax HImageX.MaxImage ([in] HImageX Image1, [in] HImageX Image2)

```

*Calculate the maximum of two images pixel by pixel.*

MaxImage calculates the maximum of the images Image1 and Image2 (pixel by pixel). The result is stored in the image ImageMax. The resulting image has the same pixel type as the input image. If several (pairs of) images are processed in one call, every i-th image from Image1 is compared to the i-th image from Image2. Thus the number of images in both input parameters must be the same. An output image is generated for every input pair.

**Attention**

The two input images must be of the same type and size.

**Parameter**

- **Image1** (input iconic) . . (multichannel-)image(array) \( \sim HImageX / IHObjectX\) (byte, int1, int2, int4, real, direction, cyclic)
  - Image(s) 1.
- **Image2** (input iconic) . . (multichannel-)image(array) \( \sim HImageX / IHObjectX\) (byte, int1, int2, int4, real, direction, cyclic)
  - Image(s) 2.
**ImageMax** (output iconic) (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX ( byte, int1, int2, int4, real, direction, cyclic )
Result image(s) by the maximization.

Example

```csharp
read_image(Bild1, "affe")
read_image(Bild2, "fabrik")
max_image(Bild1, Bild2, Max)
disp_image(Max, WindowHandle)
```

**Result**

If the parameter values are correct the operator `MaxImage` returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator `SetSystem(\(':\text{'noObjectResult'},<Result>:\) )`. If necessary an exception handling is raised.

**Parallelization Information**

`MaxImage` is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**See Also**

`MinImage`

**Alternatives**

`MaxImage`

**Module**

Image filters

```csharp
[out] HImageX ImageMin HImageX.MinImage ([in] HImageX Image1, [in] HImageX Image2 )
```

**Calculate the minimum of two images pixel by pixel.**

The operator `MinImage` determines the minimum (pixel by pixel) of the images `Image1` and `Image2`. The result is stored in the image `ImageMin`. The resulting image has the same pixel type as the input image. If several (pairs of) images are processed in one call, every i-th image from `Image1` is compared to the i-th image from `Image2`. Thus the number of images in both input parameters must be the same. An output image is generated for every input pair.

**Parameter**

- **Image1** (input iconic) (multichannel-)image(-array) \(\sim\) HImageX / IObjectX ( byte, int1, int2, int4, real, direction, cyclic )
  Image(s) 1.
- **Image2** (input iconic) (multichannel-)image(-array) \(\sim\) HImageX / IObjectX ( byte, int1, int2, int4, real, direction, cyclic )
  Image(s) 2.
- **ImageMin** (output iconic) (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX ( byte, int1, int2, int4, real, direction, cyclic )
  Result image(s) by the minimization.

**Result**

If the parameter values are correct the operator `MinImage` returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator `SetSystem(\(':\text{'noObjectResult'},<Result>:\) )`. If necessary an exception handling is raised.

**Parallelization Information**

`MinImage` is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**See Also**

`MaxImage`, `MinImage`
3.2. ARITHMETIC

Alternatives

Image filters

```halcon

```

Multiply two images.

MultImage multiplies two images. The gray values \((g_1, g_2)\) of the input images \((\text{Image1})\) are transformed as follows:

\[
g' := g_1 \times g_2 \times \text{Mult} + \text{Add}
\]

If an overflow or an underflow occurs the values are clipped.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

Parameter

- **Image1** (input iconic) . . . (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX ( byte, int1, int2, int4, real, direction, cyclic, complex )
  Image(s) 1.
- **Image2** (input iconic) . . . (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX ( byte, int1, int2, int4, real, direction, cyclic, complex )
  Image(s) 2.
- **ImageResult** (output iconic) . (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX ( byte, int1, int2, int4, real, direction, cyclic, complex )
  Result image(s) by the product.
- **Mult** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number \(\sim\) VARIANT ( integer, real )
  Factor for gray range adaption.
  **Default Value**: 0.005
  **Value Suggestions**: Mult \(\in\) \{0.001, 0.01, 0.5, 1.0, 2.0, 3.0, 5.0, 10.0\}
  **Typical Range of Values**: \(-255.0 \leq \text{Mult} \leq -255.0\)
  **Minimal Value Step**: 0.001
  **Recommended Value Step**: 0.1
- **Add** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number \(\sim\) VARIANT ( integer, real )
  Value for gray range adaption.
  **Default Value**: 0
  **Value Suggestions**: Add \(\in\) \{0.0, 128.0, 256.0\}
  **Typical Range of Values**: \(-512.0 \leq \text{Add} \leq -512.0\)
  **Minimal Value Step**: 0.01
  **Recommended Value Step**: 1.0

Example

```halcon
read_image(Image0,"fabrik")
disp_image(Image0,WindowHandle)
read_image(Image1,"Affe")
disp_image(Image1,WindowHandle)
mult_image(Image0,Image1,Result,2.0,10.0)
disp_image(Result,WindowHandle)
```

HALCON 6.0
The operator $\text{MultImage}$ returns the value TRUE if the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator $\text{SetSystem}(::'\text{noObjectResult}',<\text{Result}>::)$. If necessary an exception handling is raised.

$\textbf{Parallelization Information}$

$\text{MultImage}$ is reentrant and automatically parallelized (on tuple level, channel level, domain level).

$\textbf{See Also}$

$\text{AddImage, SubImage, DivImage}$

$\textbf{Alternatives}$

$\text{AddImage, SubImage, DivImage}$

$\textbf{Module}$

$\text{Image filters}$

```c
[out] HImageX ImageScaled $\text{HImageX.ScaleImage}$ ([in] VARIANT Mult, [in] VARIANT Add)

```

Scale the gray values of an image.

The operator $\text{ScaleImage}$ scales the input images ($\text{Image}$) by the following transformation:

$$g' := g \times \text{Mult} + \text{Add}$$

If an overflow or an underflow occurs the values are clipped.

$\textbf{Parameter}$

- $\textbf{Image}$ (input iconic) ... (multichannel-)image-array) $\sim$ HImageX / HObjectX (byte, int1, int2, int4, real, direction, cyclic, complex)
  Image(s) whose gray values are to be scaled.
- $\textbf{ImageScaled}$ (output iconic) ... (multichannel-)image-array) $\sim$ HImageX / HUntypedObjectX (byte, int1, int2, int4, real, direction, cyclic, complex)
  Result image(s) by the scale.
- $\textbf{Mult}$ (input control) ... number $\sim$ VARIANT (integer, real)
  Scale factor.
  Default Value: 0.01
  Value Suggestions: Mult $\in \{0.001, 0.003, 0.005, 0.008, 0.01, 0.02, 0.03, 0.05, 0.08, 0.1, 0.5, 1.0\}$
  Typical Range of Values: $-255.0 \leq \text{Mult} \leq -255.0$
  Minimal Value Step: 0.001
  Recommended Value Step: 0.1
- $\textbf{Add}$ (input control) ... number $\sim$ VARIANT (integer, real)
  Offset.
  Default Value: 0
  Value Suggestions: Add $\in \{0, 10, 50, 100, 200, 500\}$
  Typical Range of Values: $-512.0 \leq \text{Add} \leq -512.0$
  Minimal Value Step: 0.01
  Recommended Value Step: 1.0

$\textbf{Example}$

```c
/* Complement of the gray values: */
scale_image(Bild, Invert, -1.0, 255.0).
```

The operator $\text{ScaleImage}$ returns the value TRUE if the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator $\text{SetSystem}(::'\text{noObjectResult}',<\text{Result}>::)$.
input (no input images available) is set via the operator SetSystem('noObjectResult',<Result>):
Otherwise an exception treatment is carried out.

Parallelization Information

ScaleImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessor Functions

MinMaxGray

See Also

MinMaxGray

Alternatives

MultImage, AddImage, SubImage

Module

Image filters

```
[out] HImageX ImageSub HImageX.SubImage ([in] HImageX ImageSubtrahend,

void HOperatorSetX.SubImage ([in] IHObjectX ImageMinuend,
[in] IHObjectX ImageSubtrahend, [out] HUntypedObjectX ImageSub,
```

Subtract two images.

The operator SubImage subtracts two images. The gray values \( g_1, g_2 \) of the input images (ImageMinuend and ImageSubtrahend) are transformed as follows:

\[
g' := (g_1 - g_2) \ast \text{Mult} + \text{Add}
\]

If an overflow or an underflow occurs the values are clipped.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

Parameter

- **ImageMinuend** (input iconic) ... (multichannel-)image(-array) \( \sim \) HImageX / IHObjectX (byte, int1, int2, int4, real, direction, cyclic, complex) Minuend(s).
- **ImageSubtrahend** (input iconic) ... (multichannel-)image(-array) \( \sim \) HImageX / IHObjectX (byte, int1, int2, int4, real, direction, cyclic, complex) Subtrahend(s).
- **ImageSub** (output iconic) (multichannel-)image(-array) \( \sim \) HImageX / HUntypedObjectX (byte, int1, int2, int4, real, direction, cyclic, complex) Result image(s) by the subtraction.
- **Mult** (input control) ............................... number \( \sim \) VARIANT (integer, real) Correction factor.
  - Default Value: 1.0
  - Value Suggestions: Mult \( \in \{0.0, 1.0, 2.0, 3.0, 4.0\} \)
  - Typical Range of Values: \(-255.0 \leq \text{Mult} \leq -255.0\)
  - Minimal Value Step: 0.001
  - Recommended Value Step: 0.1
- **Add** (input control) ............................... number \( \sim \) VARIANT (integer, real) Correction value.
  - Default Value: 128.0
  - Value Suggestions: Add \( \in \{0.0, 128.0, 256.0\} \)
  - Typical Range of Values: \(-512.0 \leq \text{Add} \leq -512.0\)
  - Minimal Value Step: 0.01
  - Recommended Value Step: 1.0
Example

```plaintext
read_image(Image0,"fabrik")
disp_image(Image0,WindowHandle)
read_image(Image1,"Affe")
disp_image(Image1,WindowHandle)
sub_image(Image0,Image1,Result,2.0,10.0)
disp_image(Result,WindowHandle)
```

Result

The operator SubImage returns the value TRUE if the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator SetSystem(::'noObjectResult',<Result>);

Parallelization Information

SubImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successor Functions

DualThreshold

See Also

AddImage, MultImage, DynThreshold, CheckDifference

Alternatives

MultImage, AddImage, SubImage

Module

Image filters

3.3 Bit

```plaintext
[out] HImageX ImageAnd HImageX.BitAnd ([in] HImageX Image2 )
void HOperatorSetX.BitAnd ([in] IHObjectX Image1, [in] IHObjectX Image2, [out] HUntypedObjectX ImageAnd )
```

Bit-by-bit AND of all pixels of the input images.

The operator BitAnd calculates the “and” of all pixels of the input images bit by bit. The semantics of the “and” operation corresponds to that of C for the respective types (signed char, unsigned char, short, int/long). The images must have the same size and pixel type. The pixels within the definition range of the image in the first parameter are processed.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

Parameter

- **Image1** (input iconic) .... (multichannel-)image(-array) ~ HImageX / IHObjectX (byte, int1, int2, int4 )
  Input image(s) 1.
- **Image2** (input iconic) .... (multichannel-)image(-array) ~ HImageX / IHObjectX (byte, int1, int2, int4 )
  Input image(s) 2.
- **ImageAnd** (output iconic) (multichannel-)image(-array) ~ HImageX / HUntypedObjectX (byte, int1, int2, int4 )
  Result image(s) by AND-operation.

Example

```plaintext
read_image(Image0,'affe')
disp_image(Image0,WindowHandle)
read_image(Image1,'fabrik')
disp_image(Image1,WindowHandle)
bit_and(Image0,Image1,ImageBitA)
disp_image(ImageBitA,WindowHandle).
```
3.3. BIT

Result

If the images are correct (type and number) the operator BitAnd returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem::'noObjectResult',<Result>:: If necessary an exception handling is raised.

Parallelization Information

BitAnd is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See Also

BitMask, AddImage, MaxImage

Alternatives

BitMask, AddImage, MaxImage

Module

Image filters

Left shift of all pixels of the image.

The operator BitLshift calculates a “left shift” of all pixels of the input image bit by bit. The semantics of the “left shift” operation corresponds to that of C ("<"<<") for the respective types (signed char, unsigned char, short, int/long). If an overflow occurs the result is limited to the maximum value of the respective pixel type. Only the pixels within the definition range of the image are processed.

Several images can be processed in one call. An output image is generated for every input image.

Parameter

- **Image** (input iconic) ...................... (multichannel-)image(-array) ~ HImageX / IHObjectX
  - Input image(s).
- **ImageLShift** (output iconic). (multichannel-)image(-array) ~ HImageX / HUntypedObjectX (byte, int1, int2, int4 )
  - Result image(s) by shift operation.
- **Shift** (input control) .......................... integer ~ long
  - Shift value.
  - Default Value : 3
  - Value Suggestions : Shift ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 31}
  - Typical Range of Values : 0 ≤ Shift ≤ 0
  - Minimal Value Step : 1
  - Recommended Value Step : 1
  - Restriction : (Shift ≥ 1) ∧ (Shift ≤ 31)

Result

If the images are correct (type) and if Shift has a valid value the operator BitLshift returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem::'noObjectResult',<Result>:: If necessary an exception handling is raised.

Parallelization Information

BitLshift is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See Also

BitRshift

Alternatives

ScaleImage

Module

Image filters
Logical “AND” of each pixel using a bit mask.

The operator BitMask carries out an “and” operation of each pixel with a fixed mask. The semantics of the “and” operation corresponds to that of C for the respective types (signed char, unsigned char, short, int/long). Only the pixels within the definition range of the image are processed. Several images can be processed in one call. An output image is generated for every input image.

Parameter

- **Image** (input iconic) ………(multichannel-)image(-array) $$\sim$$ HIImageX / IHObjectX ( byte, int1, int2, int4 )
  - Input image(s).
- **ImageMask** (output iconic) … (multichannel-)image(-array) $$\sim$$ HIImageX / HUntypedObjectX ( byte, int1, int2, int4 )
  - Result image(s) by combination with mask.
- **BitMask** (input control) ............................................................integer $$\sim$$ long
  - Bit field
  - Default Value: 128
  - Value Suggestions: BitMask ∈ {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096}
  - Value List: BitMask ∈ {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096}

Result

If the images are correct (type) the operator BitMask returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem(:’noObjectResult’,<Result>:)

If necessary an exception handling is raised.

Parallelization Information

BitMask is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successor Functions

- Threshold, BitOr
- See Also
- BitAnd, BitLshift
- Alternatives
- BitSlice

Module

Image filters

Complement all bits of the pixels.

The operator BitNot calculates the “complement” of all pixels of the input image bit by bit. The semantics of the “complement” operation corresponds to that of C (“~”) for the respective types (signed char, unsigned char, short, int/long). Only the pixels within the definition range of the image are processed. Several images can be processed in one call. An output image is generated for every input image.

Parameter

- **Image** (input iconic) ………(multichannel-)image(-array) $$\sim$$ HIImageX / IHObjectX ( byte, int1, int2, int4 )
  - Input image(s).
- **ImageNot** (output iconic) (multichannel-)image(-array) $$\sim$$ HIImageX / HUntypedObjectX ( byte, int1, int2, int4 )
  - Result image(s) by complement operation.
### Example

```plaintext
read_image(Image0,'affe')
disp_image(Image0,WindowHandle)
bit_not(Image0,ImageBitN)
disp_image(ImageBitN,WindowHandle).
```

### Result

If the images are correct (type) the operator `BitNot` returns the value `TRUE`. The behavior in case of empty input (no input images available) is set via the operator `SetSystem(::'noObjectResult','<Result>::')` If necessary an exception handling is raised.

### Parallelization Information

`BitNot` is **reentrant** and automatically **parallelized** (on **tuple level**, **channel level**, **domain level**).

See Also

`BitSlice`, `BitMask`

Alternatives

`BitOr`, `BitAnd`, `AddImage`

Module

**Image filters**

```plaintext
[out] HImageX ImageOr HImageX.BitOr ([in] HImageX Image2 )
```

**Bit-by-bit OR of all pixels of the input images.**

The operator `BitOr` calculates the “or” of all pixels of the input images bit by bit. The semantics of the “or” operation corresponds to that of C for the respective types (signed char, unsigned char, short, int/long). The images must have the same size and pixel type. The pixels within the definition range of the image in the first parameter are processed.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

### Parameter

- **Image1** (input iconic) . . . . . . (multichannel-)image(-array) \(\sim HImageX / IHObjectX\) (byte, int1, int2, int4 )
  - Input image(s) 1.
- **Image2** (input iconic) . . . . . . (multichannel-)image(-array) \(\sim HImageX / IHObjectX\) (byte, int1, int2, int4 )
  - Input image(s) 2.
- **ImageOr** (output iconic) . (multichannel-)image(-array) \(\sim HImageX / HUntypedObjectX\) (byte, int1, int2, int4 )
  - Result image(s) by OR-operation.

### Example

```plaintext
read_image(Image0,'affe')
disp_image(Image0,WindowHandle)
read_image(Image1,'fabrik')
disp_image(Image1,WindowHandle)
bit_or(Image0,Image1,ImageBitO)
disp_image(ImageBitO,WindowHandle).
```

### Result

If the images are correct (type and number) the operator `BitOr` returns the value `TRUE`. The behavior in case of empty input (no input images available) is set via the operator `SetSystem(:: 'noObjectResult','<Result>::')` If necessary an exception handling is raised.

### Parallelization Information

`BitOr` is **reentrant** and automatically **parallelized** (on **tuple level**, **channel level**, **domain level**).
Right shift of all pixels of the image.

The operator BitRshift calculates a “right shift” of all pixels of the input image bit by bit. The semantics of the “right shift” operation corresponds to that of C (“>>”) for the respective types (signed char, unsigned char, short, int/long). Only the pixels within the definition range of the image are processed.

Several images can be processed in one call. An output image is generated for every input image.

Parameter

- **Image** (input iconic) ...............(multichannel-)image(-array) \(\sim\) HImageX / IObjectX (byte, int1, int2, int4)
  - Input image(s).
- **ImageRShift** (output iconic) .(multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX( byte, int1, int2, int4)
  - Result image(s) by shift operation.
- **Shift** (input control) ..................integer \(\sim\) long
  - shift value
  - Default Value : 3
  - Value Suggestions : Shift \(\in\) \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 31\}
  - Typical Range of Values : 0 \(\leq\) Shift \(\leq\) 0
  - Minimal Value Step : 1
  - Recommended Value Step : 1
  - Restriction : \((\text{Shift} \geq 1) \land (\text{Shift} \leq 31)\)

Result

If the images are correct (type) and Shift has a valid value the operator BitRshift returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem(:= 'noObjectResult',<Result>); If necessary an exception handling is raised.

Parallelization Information

BitRshift is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See Also

BitLshift, ScaleImage

Alternatives

BitAnd, AddImage

Module

Image filters

Extract a bit from the pixels.

The operator BitSlice extracts a bit level from the input image. The semantics of the “and” operation corresponds to that of C for the respective types (signed char, unsigned char, short, int/long). Only the pixels within the
definition range of the image are processed. Several images can be processed in one call. An output image is generated for every input image.

Parameter

▷ **Image** (input iconic) 
   ... (multichannel-)image(-array) \(\leadsto HImageX / IObjectX (\ \text{byte, int}1, \text{int}2, \text{int}4 )\)
   Input image(s).

▷ **ImageSlice** (output iconic) 
   .. (multichannel-)image(-array) \(\leadsto HImageX / HUntypedObjectX (\ \text{byte, int}1, \text{int}2, \text{int}4 )\)
   Ergebnis(se) der Extraktion.

▷ **Bit** (input control) 
   ... integer \(\leadsto \text{long}\)
   Bit to be selected.

Default Value: 8

Value Suggestions: \(\text{Bit} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 32\}\)

Typical Range of Values: \(1 \leq \text{Bit} \leq 1\)

Minimal Value Step: 1

Recommended Value Step: 1

Restriction: \((\text{Bit} \geq 1 \land \text{Bit} \leq 32)\)

Result

If the images are correct (type) and Bit has a valid value, the operator BitSlice returns the value TRUE.
The behavior in case of empty input (no input images available) is set via the operator SetSystem(::
\text{noObjectResult},<\text{Result}>): If necessary an exception handling is raised.

Parallelization Information

BitSlice is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successor Functions

Threshold, BitOr

See Also

BitAnd, BitLshift

Alternatives

BitMask

Module

Image filters

```
[out] HImageX ImageXor HImageX.BitXor ([in] HImageX Image2 )
```

void HOperatorSetX.BitXor ([in] IObjectX Image1, [in] IObjectX Image2, 
[out] HUntypedObjectX ImageXor )

Bit-by-bit XOR of all pixels of the input images.
The operator BitXor calculates the “xor” of all pixels of the input images bit by bit. The semantics of the “xor” operation corresponds to that of C for the respective types (signed char, unsigned char, short, int/long). The images must have the same size and pixel type. The pixels within the definition range of the image in the first parameter are processed.
Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

Parameter

▷ **Image1** (input iconic) 
   ... (multichannel-)image(-array) \(\leadsto HImageX / IObjectX (\ \text{byte, int}1, \text{int}2, \text{int}4 )\)
   Input image(s) 1.

▷ **Image2** (input iconic) 
   ... (multichannel-)image(-array) \(\leadsto HImageX / IObjectX (\ \text{byte, int}1, \text{int}2, \text{int}4 )\)
   Input image(s) 2.

▷ **ImageXor** (output iconic) 
   (multichannel-)image(-array) \(\leadsto HImageX / HUntypedObjectX (\ \text{byte, int}1, \text{int}2, \text{int}4 )\)
   Result image(s) by XOR-operation.

Example
read_image(Image0,'affe')
disp_image(Image0,WindowHandle)
read_image(Image1,'fabrik')
disp_image(Image1,WindowHandle)
bit_xor(Image0,Image1,ImageBitX)
disp_image(ImageBitX,WindowHandle).

Result

If the parameter values are correct the operator BitXor returns the value TRUE. The behavior in case of empty input (no input images available) can be determined by the operator SetSystem(::{'noObjectResult',<Result>:}) If necessary an exception handling is raised.

Parallelization Information

BitXor is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See Also

BitOr, BitAnd

Alternatives

BitOr, BitAnd, AddImage

Module

Image filters

3.4 Color

Transform an RGB image into a gray scale image.

Rgb1ToGray transforms an RGB image into a gray scale image. The three channels of the RGB image are passed as the first three channels of the input image. The image is transformed according to the following formula:

\[ k = 0.299r + 0.587g + 0.144b \]

Parameter

- RGBImage (input iconic) ......................... image(-array) ~ HImageX / IObjectX (byte, int2 )
  Three-channel RBG image.
- GrayImage (output iconic) ..................... image(-array) ~ HImageX / HUntypedObjectX (byte, int2 )
  Gray scale image.

Example

/* Tranformation from rgb to gray */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
rgb1_to_gray(Image,GrayImage)
disp_image(GrayImage,WindowHandle).

Parallelization Information

Rgb1ToGray is reentrant and automatically parallelized (on tuple level, domain level).

Possible Predecessor Functions

Compose3

Alternatives

TransFromRgb, Rgb3ToGray

Module

Image filters

Transform an RGB image to a gray scale image.

Rgb3ToGray transforms an RGB image into a gray scale image. The three channels of the RGB image are passed as three separate images. The image is transformed according to the following formula:

\[ k = 0.299r + 0.587g + 0.114b \]  

Parameter

- **ImageRed** (input iconic) ......................... image(-array) \( \sim HImageX / IHObjectX \) (byte, int2)
  Input image (red channel).
- **ImageGreen** (input iconic) ......................... image(-array) \( \sim HImageX / IHObjectX \) (byte, int2)
  Input image (green channel).
- **ImageBlue** (input iconic) ......................... image(-array) \( \sim HImageX / IHObjectX \) (byte, int2)
  Input image (blue channel).
- **ImageGray** (output iconic) ......................... image(-array) \( \sim HImageX / HUntypedObjectX \) (byte, int2)
  Gray scale image.

Example

```c
/* Transformation from rgb to gray */
read_image(Image,‘patras’)
disp_color(Image,WindowHandle)
decompose3(Image,Rimage,Gimage,Bimage)
rgb3_to_gray(Rimage,Gimage,Bimage,GrayImage)
disp_image(GrayImage,WindowHandle).
```

Parallelization Information

Rgb3ToGray is reentrant and automatically parallelized (on tuple level, domain level).

Possible Predecessor Functions

**Decompose3**

Alternatives

Rgb1ToGray, TransFromRgb

Module

Image filters

Transform an image from the RGB color space to an arbitrary color space.

TransFromRgb transforms an image from the RGB color space to an arbitrary color space (ColorSpace). The three channels of the image are passed as three separate images on input and output.

The following transformations are supported:
'yi6q'

\[
\begin{bmatrix}
Y \\
I \\
Q
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.144 \\
0.595 & -0.276 & -0.333 \\
0.209 & -0.522 & 0.287
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

'argyb'

\[
\begin{bmatrix}
A \\
R_g \\
Y_b
\end{bmatrix} =
\begin{bmatrix}
0.30 & 0.59 & 0.11 \\
0.50 & -0.50 & 0.00 \\
0.25 & 0.25 & -0.50
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

'ciexyz'

\[
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} =
\begin{bmatrix}
0.476 & 0.299 & 0.175 \\
0.262 & 0.656 & 0.082 \\
0.020 & 0.161 & 0.909
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

'hls'

\[
\begin{align*}
\text{min} &= \text{min}(R, G, B) \\
\text{max} &= \text{max}(R, G, B) \\
L &= (\text{min} + \text{max}) / 2 \\
\text{if} \ (\text{max} == \text{min}) \\
&\quad \text{H} = 0 \\
&\quad \text{S} = 0 \\
\text{else} \\
&\quad \text{if} \ (L > 0.5) \\
&\quad \quad \text{S} = (\text{max} - \text{min}) / (2 - \text{max} - \text{min}) \\
&\quad \text{else} \\
&\quad \quad \text{S} = (\text{max} - \text{min}) / (\text{max} + \text{min}) \\
&\quad \text{fi} \\
&\quad \text{if} \ (R == \text{max}) \\
&\quad \quad \text{H} = ((G - B) / (\text{max} - \text{min})) \times 60 \\
&\quad \text{elif} \ (G == \text{max}) \\
&\quad \quad \text{H} = (2 + (B - R) / (\text{max} - \text{min})) \times 60 \\
&\quad \text{elif} \ (B == \text{max}) \\
&\quad \quad \text{H} = (4 + (R - G) / (\text{max} - \text{min})) \times 60 \\
&\quad \text{fi} \\
&\quad \text{fi}
\end{align*}
\]

'hsl'

\[
\begin{bmatrix}
M1 \\
M2 \\
I1
\end{bmatrix} =
\begin{bmatrix}
\frac{2}{\sqrt{6}} & \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{6}} \\
0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}}
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

\[
\begin{bmatrix}
H \\
S \\
I
\end{bmatrix} =
\begin{bmatrix}
\arctan \frac{M1}{\sqrt{M1^2 + M2^2}} \\
\sqrt{M1^2 + M2^2} \\
I1 \times \sqrt{3}
\end{bmatrix}
\]

'hsv'

\[
\begin{align*}
\text{min} &= \text{min}(R, G, B) \\
\text{max} &= \text{max}(R, G, B) \\
V &= \text{max} \\
\text{if} \ (\text{max} == \text{min}) \\
&\quad \text{S} = 0 \\
&\quad \text{H} = 0 \\
\text{else} \\
&\quad \text{S} = (\text{max} - \text{min}) / \text{max} \\
&\quad \text{if} \ (R == \text{max}) \\
&\quad \quad \text{H} = ((G - B) / (\text{max} - \text{min})) \times 60 \\
&\quad \text{elif} \ (G == \text{max}) \\
&\quad \quad \text{H} = (2 + (B - R) / (\text{max} - \text{min})) \times 60 \\
&\quad \text{elif} \ (B == \text{max}) \\
&\quad \quad \text{H} = (4 + (R - G) / (\text{max} - \text{min})) \times 60 \\
&\quad \text{fi}
\end{align*}
\]
3.4. COLOR

\[ H = \left( 4 + \frac{R - G}{\max - \min} \right) \times 60 \]
fi
fi

\texttt{\textbf{\textquotesingle}ih\textquotesingle}:
\begin{verbatim}
min = \min(R,G,B)
max = \max(R,G,B)
I = \frac{(R + G + B)}{3}
if (I == 0)
    H = 0
    S = 1
else
    S = 1 - \min / I
    if (S == 0)
        H = 0
    else
        A = \frac{(R + R - G - B)}{2}
        B = (R - G) \times (R - G) + (R - B) \times (G - B)
        C = \sqrt{B}
        if (C == 0)
            H = 0
        else
            H = \arccos(A / C)
        fi
        if (B > G)
            H = 2 \times \pi - H
        fi
    fi
fi

\texttt{\textbf{\textquotesingle}isfeuklid\textquotesingle}:
\begin{verbatim}
min = \min(R,G,B)
max = \max(R,G,B)
I = \frac{\sqrt{R \times R + G \times G + B \times B}}{\sqrt{3}}
if (I == 0)
    S = 0
    F = 0
else
    S = \arccos(\max(1, (R + G + B) / (I \times 3))) / (\pi / 2)
    if (R == \min)
        F = \arccos(\max(1, B / \sqrt{G \times G + B \times B}))
            / (\pi \times 3 / 2) + 1 / 3
    elif (G == \min)
        F = \arccos(\max(1, R / \sqrt{B \times B + R \times R}))
            / (\pi \times 3 / 2) + 2 / 3
    else
        F = \arccos(\max(1, G / \sqrt{R \times R + G \times G}))
            / (\pi \times 3 / 2)
    fi
fi

\texttt{\textbf{\textquotesingle}isfdiff\textquotesingle}:
\begin{verbatim}
if (R >= B && B >= G)
    I = R
    S = R - G
    F = (R - B) / 6
elif (R >= G && G >= B)
    I = R
    S = R - B
    F = (2 - (R - G)) / 6
elif (G >= R && R >= B)
    I = G
    S = G - B
    F = (2 + (G - R)) / 6
\end{verbatim}

HALCON 6.0
elif \((G \geq B \&\& B \geq R)\)
  \(I = G\)
  \(S = G - R\)
  \(F = \frac{(4 - (G - B))}{6}\)
elif \((B \geq G \&\& G \geq R)\)
  \(I = B\)
  \(S = B - G\)
  \(F = \frac{(4 + (B - G))}{6}\)
else
  \(I = B\)
  \(S = B - G\)
  \(F = \frac{(6 + (B - R))}{6}\)
fi

'cielab'

\[
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix} = \begin{pmatrix}
0.476 & 0.299 & 0.175 \\
0.262 & 0.656 & 0.082 \\
0.020 & 0.161 & 0.909
\end{pmatrix} \begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

\[L = 116 \times Y^{\frac{1}{3}} - 16\]
\[a = 500 \times \left( \frac{X}{0.0031308} - Y^{\frac{1}{3}} \right)\]
\[b = 200 \times \left( Y^{\frac{1}{3}} - \frac{Z}{1.00} \right)\]

'ici2i3'

\[
\begin{pmatrix}
I1 \\
I2 \\
I3
\end{pmatrix} = \begin{pmatrix}
0.333 & 0.333 & 0.333 \\
1.0 & 0.0 & -1.0 \\
-0.5 & 1.0 & -0.5
\end{pmatrix} \begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

'ciexyz2'

\[
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix} = \begin{pmatrix}
0.620 & 0.170 & 0.180 \\
0.310 & 0.590 & 0.110 \\
0.000 & 0.066 & 1.020
\end{pmatrix} \begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

'ciexyz3'

\[
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix} = \begin{pmatrix}
0.618 & 0.177 & 0.205 \\
0.299 & 0.587 & 0.114 \\
0.000 & 0.056 & 0.944
\end{pmatrix} \begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

If necessary, certain scalings are performed, e.g., for byte-images [0..1] \(-\rightarrow\) [0..255]. In the explanation above all input and output values, including angles, are assumed to be in the range [0..1].
3.4. COLOR

- **ImageResult3** (output iconic)(-array) \( \rightarrow \) **HImageX** / **HUntypedObjectX** (byte, int1, int2, int4, real, complex)
  Color-transformed output image (channel 1).

- **ColorSpace** (input control) \( \rightarrow \) String
  Color space of the output image.
  **Default Value** : 'hsv'
  **Value List** : ColorSpace \( \in \{ \text{cielab}', '\text{hsv}', '\text{hsi}', '\text{yiq}', '\text{argyb}', '\text{ciexyz}', '\text{ciexyz2}', '\text{ciexyz3}', '\text{hls}', '\text{ihs}', '\text{isfeuklid}', '\text{isfdiff}', '\text{ii2i3}' \}

```c
/* Transformation from rgb to hsv and conversely */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
decompose3(Image,Rimage,Gimage,Bimage)
trans_from_rgb(Rimage,Gimage,Bimage,Image1,Image2,Image3,'hsv')
trans_to_rgb(Image1,Image2,Image3,ImageRed,ImageGreen,ImageBlue,'hsv')
compose3(ImageRed,ImageGreen,ImageBlue,Multichannel)
disp_color(Multichannel,WindowHandle).
```

---

**Result**

TransFromRgb returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via
SetSystem('noObjectResult',<Result>); If necessary, an exception handling is raised.

---

**Parallelization Information**

TransFromRgb is reentrant and automatically parallelized on tuple level, domain level.

---

**Possible Predecessor Functions**

Decompose3

---

**Possible Successor Functions**

Compose3

---

**See Also**

TransToRgb

---

**Alternatives**

Rgb1ToGray, Rgb3ToGray

---

**Module**

Image filters

```c
[out] HImageX ImageRed **HImageX.TransToRgb** ([in] HImageX ImageInput2,
[in] HImageX ImageInput3, [out] HImageX ImageGreen, [out] HImageX ImageBlue,
[in] String ColorSpace )

void **HOperatorSetX.TransToRgb** ([in] IObjectX ImageInput1,
[in] IObjectX ImageInput2, [in] IObjectX ImageInput3,
[out] HUntypedObjectX ImageRed, [out] HUntypedObjectX ImageGreen,
```

Transform an image from an arbitrary color space to the RGB color space.

TransToRgb transforms an image from an arbitrary color space (ColorSpace) to the RGB color space. The three channels of the image are passed as three separate images on input and output.

The following transformations are supported:

'yiq'

\[
\begin{pmatrix}
R \\
G \\
B \\
\end{pmatrix} = \begin{pmatrix}
0.999 & 0.962 & 0.615 \\
0.949 & -0.220 & -0.732 \\
0.999 & -1.101 & 1.706 \\
\end{pmatrix} \begin{pmatrix}
Y \\
I \\
Q \\
\end{pmatrix}
\]

HALCON 6.0
'argyb'

\[
\begin{bmatrix}
R \\ G \\ B
\end{bmatrix}
=
\begin{bmatrix}
1.00 & 1.29 & 0.22 \\
1.00 & -0.71 & 0.22 \\
1.00 & 0.29 & -1.78
\end{bmatrix}
\begin{bmatrix}
A \\ Rg \\ Yb
\end{bmatrix}
\]

'ciexyz'

\[
\begin{bmatrix}
R \\ G \\ B
\end{bmatrix}
=
\begin{bmatrix}
2.750 & -1.149 & -0.426 \\
-1.118 & 2.026 & 0.033 \\
0.138 & -0.333 & 1.104
\end{bmatrix}
\begin{bmatrix}
X \\ Y \\ Z
\end{bmatrix}
\]

'hls'

\[
\begin{align*}
\text{Hi} &= \text{integer}(H \times 6) \\
\text{Hf} &= \text{fraction}(H \times 6) \\
\text{if} \ (L <= 0.5) \\
\text{max} &= L \times (S + 1) \\
\text{else} \\
\text{max} &= L + S - (L \times S) \\
\text{fi} \\
\text{min} &= 2 \times L - \text{max} \\
\text{if} \ (S == 0) \\
R &= L \\
G &= L \\
B &= L \\
\text{else} \\
\text{if} \ (\text{Hi} == 0) \\
R &= \text{max} \\
G &= \text{min} + \text{Hf} \times (\text{max} - \text{min}) \\
B &= \text{min} \\
\text{elif} \ (\text{Hi} == 1) \\
R &= \text{min} + (1 - \text{Hf}) \times (\text{max} - \text{min}) \\
G &= \text{max} \\
B &= \text{min} \\
\text{elif} \ (\text{Hi} == 2) \\
R &= \text{min} \\
G &= \text{max} \\
B &= \text{min} + \text{Hf} \times (\text{max} - \text{min}) \\
\text{elif} \ (\text{Hi} == 3) \\
R &= \text{min} \\
G &= \text{min} + (1 - \text{Hf}) \times (\text{max} - \text{min}) \\
B &= \text{max} \\
\text{elif} \ (\text{Hi} == 4) \\
R &= \text{min} + \text{Hf} \times (\text{max} - \text{min}) \\
G &= \text{min} \\
B &= \text{max} \\
\text{elif} \ (\text{Hi} == 5) \\
R &= \text{max} \\
G &= \text{min} \\
B &= \text{min} + (1 - \text{Hf}) \times (\text{max} - \text{min}) \\
\text{fi} \\
\text{fi}
\end{align*}
\]

'hsi'

\[
\begin{align*}
M1 &= S \times \sin H \\
M2 &= S \times \cos H \\
I1 &= \frac{I}{\sqrt{3}} \\
\begin{bmatrix}
R \\ G \\ B
\end{bmatrix}
= \begin{bmatrix}
\frac{2}{\sqrt{6}} & \frac{0}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\
\frac{0}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\
\frac{2}{\sqrt{6}} & \frac{0}{\sqrt{2}} & \frac{1}{\sqrt{3}}
\end{bmatrix}
\begin{bmatrix}
M1 \\ M2 \\ I1
\end{bmatrix}
\end{align*}
\]
if (S == 0)
    if (H == 0)
        R = V
        G = V
        B = V
    else
        R = 0
        G = 0
        B = 0
    fi
else
    Hi = integer(H)
    Hf = fraction(H)
    if (Hi == 0)
        R = V
        G = V * (1 - (S * (1 - Hf)))
        B = V * (1 - S)
    elif (Hi == 1)
        R = V * (1 - (S * Hf))
        G = V
        B = V * (1 - S)
    elif (Hi == 2)
        R = V * (1 - S)
        G = V
        B = V * (1 - (S * (1 - Hf)))
    elif (Hi == 3)
        R = V * (1 - S)
        G = V * (1 - (S * Hf))
        B = V
    elif (Hi == 4)
        R = V * (1 - (S * (1 - Hf)))
        G = V * (1 - S)
        B = V
    elif (Hi == 5)
        R = V
        G = V * (1 - S)
        B = V * (1 - (S * Hf))
    fi
fi

If necessary, certain scalings are performed, e.g., for byte-images [0..1] → [0..255]. In the explanation above all input and output values, including angles, are assumed to be in the range [0..1].

<table>
<thead>
<tr>
<th>Parameter</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ImageInput1</strong> (input iconic) .....................image(-array) 瑁 HIImageX / IHObjectX ( byte, int4, real ) Input image (channel 1).</td>
<td></td>
</tr>
<tr>
<td><strong>ImageInput2</strong> (input iconic) .....................image(-array) 瑁 HIImageX / IHObjectX ( byte, int4, real ) Input image (channel 2).</td>
<td></td>
</tr>
<tr>
<td><strong>ImageInput3</strong> (input iconic) .....................image(-array) 瑁 HIImageX / IHObjectX ( byte, int4, real ) Input image (channel 3).</td>
<td></td>
</tr>
<tr>
<td><strong>ImageRed</strong> (output iconic) .....................image(-array) 瑁 HIImageX / HUntypedObjectX ( byte, int4, real ) Red channel.</td>
<td></td>
</tr>
<tr>
<td><strong>ImageGreen</strong> (output iconic) .....................image(-array) 瑁 HIImageX / HUntypedObjectX ( byte, int4, real ) Green channel.</td>
<td></td>
</tr>
<tr>
<td><strong>ImageBlue</strong> (output iconic) .....................image(-array) 瑁 HIImageX / HUntypedObjectX ( byte, int4, real ) Blue channel.</td>
<td></td>
</tr>
</tbody>
</table>
**ColorSpace** (input control) .................................................. string  ~ String
Color space of the input image.
**Default Value : 'hsv'**
**Value List :** ColorSpace ∈ {'hsi', 'yiq', 'argyb', 'ciexyz', 'hls', 'hsv'}

/* Transformation from rgb to hsv and conversely */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
decompose3(Image, Rimage, Gimage, Bimage)
trans_from_rgb(Rimage, Gimage, Bimage, Image1, Image2, Image3, 'hsv')
trans_to_rgb(Image1, Image2, Image3, ImageRed, ImageGreen, ImageBlue, 'hsv')
compose3(ImageRed, ImageGreen, ImageBlue, Multichannel)
disp_color(Multichannel, WindowHandle).

TransToRgb returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via
SetSystem(:'noObjectResult',<Result>:). If necessary, an exception handling is raised.

**Parallelization Information**
TransToRgb is reentrant and automatically parallelized (on tuple level, domain level).

**Possible Predecessor Functions**
Decompose3

**Possible Successor Functions**
Compose3, DispColor

**See Also**
Decompose3

**Module**
Image filters

### 3.5 Edges

```plaintext
[out] HRegionX RegionResult HRegionX.CloseEdges ([in] HImageX EdgeImage,
[in] long MinAmplitude )
```

```plaintext
void HOperatorSetX.CloseEdges ([in] IObjectX Edges,
[in] IObjectX EdgeImage, [out] HUntypedObjectX RegionResult,
[in] long MinAmplitude )
```

Close edge gaps using the edge amplitude image.

CloseEdges closes gaps in the output of an edge detector, and thus tries to produce complete object contours. This is done by examining the neighbors of each edge point to determine the point with maximum amplitude (i.e., maximum gradient), and adding the point to the edge if its amplitude is larger than the minimum amplitude passed in MinAmplitude. This operator expects as input the edges (Edges) and amplitude image (EdgeImage) returned by typical edge operators, such as EdgesImage or SobelAmp. CloseEdges does not take into account the edge directions that may be returned by an edge operator. Thus, in areas where the gradient is almost constant the edges may become rather "wiggly."

**Parameter**

- **Edges** (input iconic) ...............................region(-array)  ~ HRegionX / IObjectX
  Region containing one pixel thick edges.
- **EdgeImage** (input iconic) .........................image  ~ HImageX / IObjectX (byte)
  Edge amplitude (gradient) image.
- **RegionResult** (output iconic) .......................region(-array)  ~ HRegionX / HUntypedObjectX
  Region containing closed edges.
**MinAmplitude** (input control) .............................................................. integer ~ long
Minimum edge amplitude.
**Default Value**: 16

**Value Suggestions**: MinAmplitude ∈ {5, 8, 10, 12, 16, 20, 25, 30, 40, 50}

**Typical Range of Values**: 1 ≤ MinAmplitude ≤ 1

**Minimal Value Step**: 1
**Recommended Value Step**: 1

**Restriction**: (MinAmplitude ≥ 0)

---

**Result**
CloseEdges returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

---

**Parallelization Information**
CloseEdges is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**
EdgesImage, SobelAmp, Threshold, Skeleton

---

**Possible Successor Functions**
Skeleton

---

See Also
GraySkeleton

---

**Alternatives**
CloseEdgesLength, Dilation1, Closing

---

**Module**
Image filters

---

```plaintext
[out] HRegionX ClosedEdges HRegionX.CloseEdgesLength

void HOperatorSetX.CloseEdgesLength ([in] IHObjectX Edges,
[in] IHObjectX Gradient, [out] HUntypedObjectX ClosedEdges,
[in] long MinAmplitude, [in] long MaxGapLength )
```

---

**Close edge gaps using the edge amplitude image.**

CloseEdgesLength closes gaps in the output of an edge detector, and thus tries to produce complete object contours. This operator expects as input the edges (Edges) and amplitude image (Gradient) returned by typical edge operators, such as EdgesImage or SobelAmp.

Contours are closed in two steps: First, one pixel wide gaps in the input contours are closed, and isolated points are eliminated. After this, open contours are extended by up to MaxGapLength points by adding edge points until either the contour is closed or no more significant edge points can be found. A gradient is regarded as significant if it is larger than MinAmplitude. The neighboring points examined as possible new edge points are the point in the direction of the contour and its two adjacent points in an 8-neighborhood. For each of these points, the sum of its gradient and the maximum gradient of that points three possible neighbors is calculated (look ahead of length 1). The point with the maximum sum is then chosen as the new edge point.

---

**Attention**
In the current implementation MinAmplitude is ignored; any gradient value larger than zero is regarded as significant.

---

**Parameter**

- **Edges** (input iconic) .............................................. region(-array) ~ HRegionX / IHObjectX
  Region containing one pixel thick edges.
- **Gradient** (input iconic) ........................................... image ~ HImageX / IHObjectX (byte)
  Edge amplitude (gradient) image.
- **ClosedEdges** (output iconic) ............................... region(-array) ~ HRegionX / HUntypedObjectX
  Region containing closed edges.
**MinAmplitude** (input control) ................................................................. integer  \( \sim \) long
Minimum edge amplitude.

**Default Value:** 16

**Value Suggestions:** \( \text{MinAmplitude} \in \{5, 8, 10, 12, 16, 20, 25, 30, 40, 50\} \)

**Typical Range of Values:** \( 1 \leq \text{MinAmplitude} \leq 1 \)

**Minimal Value Step:** 1

**Recommended Value Step:** 1

**Restriction:** \( (\text{MinAmplitude} \geq 0) \)

**MaxGapLength** (input control) ................................................................. integer  \( \sim \) long
Maximal number of points by which edges are extended.

**Default Value:** 3

**Value Suggestions:** \( \text{MaxGapLength} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 30, 40, 50, 70, 100\} \)

**Typical Range of Values:** \( 1 \leq \text{MaxGapLength} \leq 1 \)

**Minimal Value Step:** 1

**Recommended Value Step:** 1

**Restriction:** \( (\text{MaxGapLength} > 0) \)

---

**Result**

CloseEdgesLength returns `TRUE` if all parameters are correct. If the input is empty the behaviour can be set via `SetSystem('noObjectResult',<Result>)`. If necessary, an exception handling is raised.

---

**Parallelization Information**

CloseEdgesLength is *reentrant* and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

Possibly Predecessor Functions

---

**Alternatives**

Available Alternatives

---

**Bibliography**


---

**Module**

Image filters

---

```plaintext
[out] HImageX DerivGauss HImageX.DerivateGauss ([in] double Sigma, [in] String Component )

```

---

**Convolve an image with derivatives of the Gaussian.**

DerivateGauss convolves an image with the derivatives of a Gaussian and calculates various features derived thereof. Possible values for Component are:

- **'none'** Smoothing only.
- **'x'** First derivative along x.
  \[
  g'(x, y) = \frac{\partial g(x, y)}{\partial x}
  \]
- **'y'** First derivative along y.
  \[
  g'(x, y) = \frac{\partial g(x, y)}{\partial y}
  \]
- **'gradient'** Absolute value of the gradient.
  \[
  g'(x, y) = \sqrt{\frac{\partial g(x, y)^2}{\partial x} \frac{\partial g(x, y)^2}{\partial y}}
  \]
- **'xx'** Second derivative along x.
  \[
  g'(x, y) = \frac{\partial^2 g(x, y)}{\partial x^2}
  \]

---

3.5. EDGES

'y^y' Second derivative along y.
\[ g'(x, y) = \frac{\partial^2 g(x, y)}{\partial y^2} \]

'x^y' Second derivative along x and y.
\[ g'(x, y) = \frac{\partial^2 g(x, y)}{\partial x \partial y} \]

'x^x' Third derivative along x.
\[ g'(x, y) = \frac{\partial^3 g(x, y)}{\partial x^3} \]

'y^y' Third derivative along y.
\[ g'(x, y) = \frac{\partial^3 g(x, y)}{\partial y^3} \]

'x^x' Third derivative along x, x and y.
\[ g'(x, y) = \frac{\partial^3 g(x, y)}{\partial x^2 \partial y} \]

'x^y' Third derivative along x, y and y.
\[ g'(x, y) = \frac{\partial^3 g(x, y)}{\partial x \partial y^2} \]

'det' Determinant of the Hessian matrix:
\[ DET = \frac{\partial^2 g(x, y)}{\partial x^2} \frac{\partial^2 g(x, y)}{\partial y^2} - \left( \frac{\partial g(x, y)}{\partial y \partial x} \right)^2 \]

'laplace' Laplace operator (trace of the Hessian matrix):
\[ TR = \frac{\partial^2 g(x, y)}{\partial x^2} + \frac{\partial^2 g(x, y)}{\partial y^2} \]

'mean_curvatue' Mean curvature \( H \)
\[
\begin{align*}
\alpha &= (1 + \frac{\partial g(x, y)^2}{\partial x}) \frac{\partial^2 g(x, y)}{\partial y^2} \\
\beta &= 2 \frac{\partial g(x, y)}{\partial x} \frac{\partial g(x, y)}{\partial y} \frac{\partial^2 g(x, y)}{\partial y \partial x} \\
\gamma &= (1 + \frac{\partial g(x, y)^2}{\partial y}) \frac{\partial^2 g(x, y)}{\partial x^2} \\
\delta &= (1 + \frac{\partial g(x, y)^2}{\partial x}) + \frac{\partial g(x, y)^2}{\partial y} \\
H &= \frac{\alpha - \beta + \gamma}{\delta} \\
H &= \frac{1}{2} (\kappa_{\text{min}} + \kappa_{\text{max}})
\end{align*}
\]

'gauss_curvatue' Gaussian curvature \( K \)
\[ K = \frac{DET}{(1 + \frac{\partial g(x, y)^2}{\partial x} + \frac{\partial g(x, y)^2}{\partial y})^2} \]

'eigenvalue1' First eigenvalue
\[
\begin{align*}
\alpha &= \frac{\partial^2 g(x, y)}{\partial x^2} + \frac{\partial^2 g(x, y)}{\partial y^2} \\
\lambda_1 &= \alpha + \sqrt{\alpha^2 - \left( \frac{\partial g(x, y)}{\partial x^2} \frac{\partial^2 g(x, y)}{\partial y^2} - \frac{\partial g(x, y)^2}{\partial y \partial x} \right)}
\end{align*}
\]
'eigenvalue2' Second eigenvalue

\[ \lambda_2 = \alpha - \sqrt{\alpha^2 - (\frac{\partial^2 g(x,y)}{\partial x^2} - \frac{\partial^2 g(x,y)}{\partial y^2})^2} \]

'main1_curvature' First principal curvature

\[ \kappa_{max} = H + \sqrt{H^2 - K} \]

'main2_curvature' Second principal curvature

\[ \kappa_{min} = H - \sqrt{H^2 - K} \]

'kitchen_rosenfeld' Second derivative perpendicular to the gradient

\[ k = \frac{\partial^2 g(x,y)}{\partial x^2} \frac{\partial g(y)}{\partial y} - \frac{\partial^2 g(x,y)}{\partial y^2} \frac{\partial g(x)}{\partial x} + \frac{\partial^2 g(x,y)}{\partial x^2} \frac{\partial g(y)}{\partial y} + \frac{\partial^2 g(x,y)}{\partial y^2} \frac{\partial g(x)}{\partial x} \]

'2nd_ddg' Second derivative along the gradient

\[ k = \frac{\partial^2 g(x,y)}{\partial x^2} \frac{\partial g(x)}{\partial x} + \frac{\partial^2 g(x,y)}{\partial y^2} \frac{\partial g(y)}{\partial y} - \frac{\partial^2 g(x,y)}{\partial x^2} \frac{\partial g(x)}{\partial x} - \frac{\partial^2 g(x,y)}{\partial y^2} \frac{\partial g(y)}{\partial y} \]

'de_saint_venant' Second derivative along and perpendicular to the gradient

\[ k = \frac{\partial g(y)}{\partial y} \frac{\partial g(x)}{\partial x} \frac{\partial^2 g(x,y)}{\partial x^2} - \frac{\partial g(y)}{\partial y} \frac{\partial^2 g(x,y)}{\partial y^2} + \frac{\partial g(x)}{\partial x} \frac{\partial g(y)}{\partial y} \frac{\partial^2 g(x,y)}{\partial x^2} - \frac{\partial g(x)}{\partial x} \frac{\partial^2 g(x,y)}{\partial y^2} \]

Parameter

- **Image** (input iconic) .. (multichannel-)image(-array) \( \sim \) HImageX / IHOBJECTX ( byte, int1, int2, int4, real )
  Input image.

- **DerivGauss** (output iconic) ...... (multichannel-)image(-array) \( \sim \) HImageX / HUnTypedObjectX ( real )
  Filtered result image.

- **Sigma** (input control) ................................................................. real \( \sim \) double
  Sigma of the Gaussian.

  **Default Value :** 1.0
  **Value Suggestions :** Sigma \( \in \{0.7, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0\} \)
  **Typical Range of Values :** 0.2 \( \leq \) Sigma \( \leq \) 0.2
  **Minimal Value Step :** 0.01
  **Recommended Value Step :** 0.1
  **Restriction :** (Sigma > 0.0)

- **Component** (input control) ......................................................... string \( \sim \) String
  Derivative or feature to be calculated.

  **Default Value :** 'x'


---

**Parallelization Information**

**Possible Successor Functions**

-ZEROCrossing, DualThreshold

---

**See Also**

-ZEROCrossing, DualThreshold
Approximate the LoG operator (Laplace of Gaussian).

DiffOfGauss approximates the Laplace-of-Gauss operator by a difference of Gaussians. The standard deviations of these Gaussians can be calculated, according to Marr, from the Parameter Sigma of the LoG and the ratio of the two standard deviations (SigFactor) as:

\[
\sigma_1 = \frac{\text{Sigma}}{\log(\frac{\text{Sigma}}{\text{SigFactor}})}
\]

\[
\sigma_2 = \frac{\sigma_1}{\text{SigFactor}}
\]

\[
\text{DiffOfGauss} = (\text{Image} \ast \text{gauss}(\sigma_1)) - (\text{Image} \ast \text{gauss}(\sigma_2))
\]

For a SigFactor = 1.6, according to Marr, an approximation to the Mexican-Hat-Operator results. The resulting image is stored in DiffOfGauss.

Parameter

- **Image** (input iconic) ................. (multichannel-)image(-array) \(\sim\) HImageX / IObjectX (byte) 
  Input image

- **DiffOfGauss** (output iconic) ..... (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (int2) 
  LoG image.

- **Sigma** (input control) ................. (multichannel-)image(-array) \(\sim\) HImageX / IObjectX (real) \(\sim\) double 
  <no description available>
  Default Value: 3.0
  Value Suggestions: Sigma \(\in\) \{2.0, 3.0, 4.0, 5.0\}
  Typical Range of Values: \(0.2 \leq \text{Sigma} \leq 0.2\)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.1
  Restriction: (Sigma > 0.0)

- **SigFactor** (input control) ................. (real) \(\sim\) double 
  Ratio of the standard deviations used (Marr recommends 1.6).
  Default Value: 1.6
  Typical Range of Values: 0.1 \(\leq\) SigFactor \(\leq 0.1\)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.1
  Restriction: (SigFactor > 0.0)

Example

read_image(Image,'fabrik')
diff_of_gauss(Image,Laplace,2.0,1.6)
zero_crossing(Laplace,ZeroCrossings).
DiffOfGauss returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

**DiffOfGauss is reentrant and automatically parallelized (on tuple level, channel level, domain level).**

**Possible Successor Functions**

ZeroCrossing, DualThreshold

**Alternatives**

Laplace, DerivateGauss

**Bibliography**


**Module**

Image filters

---

### Extract edges using Deriche, Lanser, Shen, or Canny filters.

`HImageX.EdgesImage` detects step edges using recursively implemented filters (according to Deriche, Lanser and Shen) or the conventionally implemented “derivative of Gaussian” filter (using filter masks) proposed by Canny. Thus, the following edge operators are available:

- 'deriche1', 'lanser1', 'deriche1_int4', 'deriche2', 'lanser2', 'lanser2_int4', 'shen', 'mshen', and 'canny' (parameter `Filter`).

The edge amplitudes (gradient magnitude) and directions are returned in `ImaAmp` and `ImaDir`, respectively. The edge operator 'lanser2' is also available for int4-images, and returns the signed filter response instead of its absolute value. This behavior can be obtained for byte-images as well by selecting 'lanser2_int4' as filter. This can be used to calculate the second derivative of an image by applying `EdgesImage` (with parameter 'lanser2') to the signed first derivative. Edge directions are stored in 2-degree steps, i.e., an edge direction of $\theta$ degrees with respect to the horizontal axis is stored as $\theta$ in the edge direction image. Furthermore, the direction of the change of intensity is taken into account. Let $[E_x, E_y]$ denote the image gradient. Then the following edge directions are returned as $r/2$:

<table>
<thead>
<tr>
<th>intensity increase</th>
<th>$E_x/E_y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>edge direction $r$</td>
<td></td>
</tr>
<tr>
<td>from bottom to top</td>
<td>$0/+0$</td>
</tr>
<tr>
<td>from lower right to upper left</td>
<td>$/+090$</td>
</tr>
<tr>
<td>from right to left</td>
<td>$+090$</td>
</tr>
<tr>
<td>from upper right to lower left</td>
<td>$+0[90,180]$</td>
</tr>
<tr>
<td>from top to bottom</td>
<td>$+0180$</td>
</tr>
<tr>
<td>from upper left to lower right</td>
<td>$-/+180, 270$</td>
</tr>
<tr>
<td>from left to right</td>
<td>$+0270$</td>
</tr>
<tr>
<td>from lower left to upper right</td>
<td>$-/+[270,360]$</td>
</tr>
</tbody>
</table>

Points with edge amplitude 0 are assigned the edge direction 255 (undefined direction).

The “filter width” (i.e., the amount of smoothing) can be chosen arbitrarily, and can be estimated by calling `InfoEdges` for concrete values of the parameter `Alpha`. It decreases for increasing `Alpha` for the Deriche, Lanser and Shen filters and increases for the Canny filter, where it is the standard deviation of the Gaussian on
which the Canny operator is based. “Wide” filters exhibit a larger invariance to noise, but also a decreased ability
to detect small details. Non-recursive filters, such as the Canny filter, are realized using filter masks, and thus the
execution time increases for increasing filter width. In contrast, the execution time for recursive filters does not
depend on the filter width. Thus, arbitrary filter widths are possible using the Deriche, Lanser and Shen filters
without increasing the run time of the operator. The resulting advantage in speed compared to the Canny operator
naturally increases for larger filter widths. As border treatment, the recursive operators assume that the images to
be zero outside of the image, while the Canny operator repeats the gray value at the image’s border. Comparable
filter widths can be obtained by the following choices of Alpha:

\[
\begin{align*}
\alpha('lanser1') & = \alpha('deriche1')/2 \\
\alpha('deriche2') & = \alpha('deriche1')/2 \\
\alpha('lanser2') & = \alpha('deriche1')/2 \\
\alpha('shen') & = \alpha('deriche1')/2 \\
\alpha('mshen') & = \alpha('deriche1')/2 \\
\alpha('gauss') & = 1.77/\alpha('deriche1')
\end{align*}
\]

The originally proposed recursive filters (‘deriche1’, ‘deriche2’, ‘shen’) return a biased estimate of the amplitude
of diagonal edges. This bias is removed in the corresponding modified version of the operators (‘lanser1’, ‘lanser2’
und ‘mshen’), while maintaining the same execution speed.

For relatively small filter widths (11 × 11), i.e., for Alpha (‘lanser2’ = 0.5), all filters yield similar results. Only for
“wider” filters differences begin to appear: the Shen filters begin to yield qualitatively inferior results. However,
they are the fastest of the implemented operators — closely followed by the Deriche operators.

EdgesImage optionally offers to apply a non-maximum-suppression (NMS = ‘nms’/‘inms’/‘hvnms’; ‘none’ if not desired) and hysteresis threshold operation (Low,High; at least one negative if not desired) to the resulting edge image. Conceptually, this corresponds to the following calls:

```plaintext
nonmax_suppression_dir(..., NMS, ...) ▷
hysteresis_threshold(..., Low, High, 999, ...)
```

However, this calling sequence would return appropriately modified regions, whereas EdgesImage marks sup-
pressed edge points by 0 in the amplitude and 255 in the direction image, while leaving the region of the image
unchanged.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>▷ <strong>Image</strong> (input iconic) ...................(multichannel-)image(-array) ▷ HImageX / HObjectX ( byte, int4, int2 ) Input image.</td>
</tr>
<tr>
<td>▷ <strong>I*maAmp</strong> (output iconic) ..(multichannel-)image(-array) ▷ HImageX / HUn*typedObjectX ( byte, int4, int2 ) Edge amplitude (gradient magnitude) image.</td>
</tr>
<tr>
<td>▷ <strong>I*maDir</strong> (output iconic) ................. image(-array) ▷ HImageX / HUn*typedObjectX ( direction ) Edge direction image.</td>
</tr>
<tr>
<td>▷ <strong>Filter</strong> (input control) ....................string ▷ String Edge operator to be applied.</td>
</tr>
<tr>
<td>Default Value : ‘lanser2’</td>
</tr>
<tr>
<td>▷ <strong>Alpha</strong> (input control) ....................real ▷ double Filter parameter: small values result in strong smoothing, and thus less detail (opposite for ‘canny’).</td>
</tr>
<tr>
<td>Default Value : 0.5</td>
</tr>
<tr>
<td>Value Suggestions : Alpha ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1.1}</td>
</tr>
<tr>
<td>Typical Range of Values : 0.2 ≤ Alpha ≤ 0.2</td>
</tr>
<tr>
<td>Minimal Value Step : 0.01</td>
</tr>
<tr>
<td>Recommended Value Step : 0.1</td>
</tr>
<tr>
<td>Restriction : (Alpha &gt; 0.0)</td>
</tr>
<tr>
<td>▷ <strong>NMS</strong> (input control) ....................string ▷ String Non-maximum suppression (‘none’, if not desired).</td>
</tr>
<tr>
<td>Default Value : ‘nms’</td>
</tr>
</tbody>
</table>

HALCON 6.0
Low (input control) ................................................................. integer ~ long
Lower threshold for the hysteresis threshold operation (negative, if no thresholding is desired).

Default Value : 20
Value Suggestions : Low ∈ {5, 10, 15, 20, 25, 30, 40}
Typical Range of Values : 1 ≤ Low ≤ 1
Minimal Value Step : 1
Recommended Value Step : 5
Restriction : ([Low > 1] ∨ ([Low < 0])

High (input control) ................................................................. integer ~ long
Upper threshold for the hysteresis threshold operation (negative, if no thresholding is desired).

Default Value : 40
Value Suggestions : High ∈ {10, 15, 20, 25, 30, 40, 50, 60, 70}
Typical Range of Values : 1 ≤ High ≤ 1
Minimal Value Step : 1
Recommended Value Step : 5
Restriction : ([High > 1] ∨ (High < 0]) ∧ (High ≥ Low)

Example

read_image(Image,'fabrik')
edges_image(Image,Amp,Dir,'lanser2',0.5,'none',-1,-1)
hysteresis_threshold(Amp,Margin,20,30,30).

Result
EdgesImage returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information
EdgesImage is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessor Functions
InfoEdges

Possible Successor Functions
Threshold,HysteresisThreshold,CloseEdgesLength

See Also
InfoEdges,NonmaxSuppressionAmp,HysteresisThreshold,BandpassImage

Alternatives
SobelDir,FreiDir,KirschDir,PrewittDir,RobinsonDir

Bibliography
S.Castan, J.Zhao und J.Shen: “Optimal Filter for Edge Detection Methods and Results”; Proc. of the First European Conference on Computer Vision, Antibes; Lecture Notes on computer Science; no. 427; S. 12-17; Springer-Verlag; 1990.
Image filters

Extract sub-pixel precise edges using Deriche, Lanser, Shen, or Canny filters.

EdgesSubPix detects step edges using recursively implemented filters (according to Deriche, Lanser and Shen) or the conventionally implemented “derivative of Gaussian” filter (using filter masks) proposed by Canny. Thus, the following edge operators are available:

- 'deriche1', 'lanser1', 'lanser1_int4', 'deriche2', 'lanser2', 'lanser2_int4', 'shen', 'mshen', and 'canny' (parameter Filter).

The extracted edges are returned as sub-pixel precise XLD contours in Edges. For each edge point the following attributes are defined (see GetContourAttribXld):

- 'angle' Edge direction (modulo π)
- 'response' Edge amplitude (gradient magnitude)

The “filter width” (i.e., the amount of smoothing) can be chosen arbitrarily, and can be estimated by calling InfoEdges for concrete values of the parameter Alpha. It decreases for increasing Alpha for the Deriche, Lanser and Shen filters and increases for the Canny filter, where it is the standard deviation of the Gaussian on which the Canny operator is based. “Wide” filters exhibit a larger invariance to noise, but also a decreased ability to detect small details. Non-recursive filters, such as the Canny filter, are realized using filter masks, and thus the execution time increases for increasing filter width. In contrast, the execution time for recursive filters does not depend on the filter width. Thus, arbitrary filter widths are possible using the Deriche, Lanser and Shen filters without increasing the run time of the operator. The resulting advantage in speed compared to the Canny operator naturally increases for larger filter widths. As border treatment, the recursive operators assume that the images to be zero outside of the image, while the Canny operator repeats the gray value at the image’s border. Comparable filter widths can be obtained by the following choices of Alpha:

\[
\begin{align*}
\text{Alpha('lanser1')} &= \text{Alpha('deriche1')}/2 \\
\text{Alpha('deriche2')} &= \text{Alpha('deriche1')}/2 \\
\text{Alpha('lanser2')} &= \text{Alpha('deriche1')}/2 \\
\text{Alpha('shen')} &= \text{Alpha('deriche1')}/2 \\
\text{Alpha('mshen')} &= \text{Alpha('deriche1')}/2 \\
\text{Alpha('gauss')} &= 1.77/\text{Alpha('deriche1')}
\end{align*}
\]

The originally proposed recursive filters ('deriche1', 'deriche2', 'shen') return a biased estimate of the amplitude of diagonal edges. This bias is removed in the corresponding modified version of the operators ('lanser1', 'lanser2' and 'mshen'), while maintaining the same execution speed.

For relatively small filter widths (11 × 11), i.e., for Alpha ('lanser2' = 0.5), all filters yield similar results. Only for “wider” filters differences begin to appear: the Shen filters begin to yield qualitatively inferior results. However, they are the fastest of the implemented operators — closely followed by the Deriche operators.

EdgesSubPix links the edge points into edges by using an algorithm similar to a hysteresis threshold operation, which is also used in LinesGauss. Points with an amplitude larger than high are immediately accepted as belonging to an edge, while points with an amplitude smaller than low are rejected. All other points are accepted as edges if they are connected to accepted edge points (see also LinesGauss and HysteresisThreshold).
Filter (input control) .................................................. string ~ String
   Edge operator to be applied.
   Default Value : 'lanser2'
   Value List: Filter ∈ { 'deriche1', 'lanser1', 'deriche2', 'lanser2', 'shen', 'mshen', 'canny', 'sobel' }

Alpha (input control) .................................................. real ~ double
   Filter parameter: small values result in strong smoothing, and thus less detail (opposite for 'canny').
   Default Value : 0.5
   Value Suggestios : Alpha ∈ { 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1.1 }
   Typical Range of Values : 0.2 ≤ Alpha ≤ 0.2
   Minimal Value Step : 0.01
   Recommended Value Step : 0.1
   Restriction : (Alpha > 0.0)

Low (input control) .................................................... integer ~ long
   Lower threshold for the hysteresis threshold operation.
   Default Value : 20
   Value Suggestions : Low ∈ { 5, 10, 15, 20, 25, 30, 40 }
   Typical Range of Values : 1 ≤ Low ≤ 1
   Minimal Value Step : 1
   Recommended Value Step : 5
   Restriction : (Low > 0)

High (input control) ................................................... integer ~ long
   Upper threshold for the hysteresis threshold operation.
   Default Value : 40
   Value Suggestions : High ∈ { 10, 15, 20, 25, 30, 40, 50, 60, 70 }
   Typical Range of Values : 1 ≤ High ≤ 1
   Minimal Value Step : 1
   Recommended Value Step : 5
   Restriction : ((High > 0) ∧ (High ≥ Low))

Example

read_image(Image,'fabrik')
edges_sub_pix(Image,Edges,'lanser2',0.5,20,40).

Result

EdgesSubPix returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behaviour can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

EdgesSubPix is reentrant and automatically parallelized (on tuple level).

See Also

InfoEdges, HysteresisThreshold, BandpassImage, LinesGauss, LinesFacet

Alternatives

SobelDir, FreiDir, KirschDir, PrewittDir, RobinsonDir, EdgesImage

Bibliography

3.5. EDGES


S. Castan, J. Zhao und J. Shen: “Optimal Filter for Edge Detection Methods and Results”; Proc. of the First European Conference on Computer Vision, Antibes; Lecture Notes on computer Science; no. 427; S. 12-17; Springer-Verlag; 1990.

Sub-pixel operators

```
out HImageX ImageEdgeAmp HImageX.FreiAmp ( )

void HOperatorSetX.FreiAmp ([in] IHObjectX Image,
[out] HUntypedObjectX ImageEdgeAmp )
```

Detect edges (amplitude) using the Frei-Chen operator.

FreiAmp calculates an approximation of the first derivative of the image data and is used as an edge detector. The filter is based on the following filter masks:

\[
A = \begin{bmatrix}
1 & 1 & 1 \\
0 & 0 & 0 \\
-1 & -1 & -1
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1
\end{bmatrix}
\]

The result image contains the maximum response of the masks \(A\) and \(B\).

Parameter

- **Image** (input iconic) .................. (multichannel-image-array) \(\sim HImageX / IHObjectX(\text{byte, int2})\)
  Input image.
- **ImageEdgeAmp** (output iconic) (multichannel-image-array) \(\sim HImageX / HUntypedObjectX(\text{byte, int2})\)
  Edge amplitude (gradient magnitude) image.

Example

```c
read_image(Image,’fabrik’)
frei_amp(Image,Frei_amp)
threshold(Frei_amp,Edges,128,255).
```

Result

FreiAmp always returns TRUE. If the input is empty the behaviour can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

FreiAmp is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessor Functions

- GaussImage, SigmaImage, MedianImage, SmoothImage

See Also

- BandpassImage, LaplaceOfGauss

Alternatives

- SobelAmp, KirschAmp, PrewittAmp, RobinsonAmp, Roberts

Image filters
Detect edges (amplitude and direction) using the Frei-Chen operator.

FreiDir calculates an approximation of the first derivative of the image data and is used as an edge detector. The filter is based on the following filter masks:

\[
\begin{bmatrix}
1 & \sqrt{2} & 1 \\
0 & 0 & 0 \\
-1 & -\sqrt{2} & -1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & -1 \\
\sqrt{2} & 0 & -\sqrt{2} \\
1 & 0 & -1 \\
\end{bmatrix}
\]

The result image contains the maximum response of the masks \(A\) and \(B\). The edge directions are returned in ImageEdgeDir, and are stored in 2-degree steps, i.e., an edge direction of \(x\) degrees with respect to the horizontal axis is stored as \(x/2\) in the edge direction image. Furthermore, the direction of the change of intensity is taken into account. Let \([E_x, E_y]\) denote the image gradient. Then the following edge directions are returned as \(r/2\):

<table>
<thead>
<tr>
<th>intensity increase</th>
<th>(E_x/E_y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>edge direction (r)</td>
<td></td>
</tr>
<tr>
<td>from bottom to top</td>
<td>0/+ 0</td>
</tr>
<tr>
<td>from lower right to upper left</td>
<td>+/− [0, 90]</td>
</tr>
<tr>
<td>from right to left</td>
<td>+/0 90</td>
</tr>
<tr>
<td>from upper right to lower left</td>
<td>+/+ [90, 180]</td>
</tr>
<tr>
<td>from top to bottom</td>
<td>0/+ 180</td>
</tr>
<tr>
<td>from upper left to lower right</td>
<td>−/+ [180, 270]</td>
</tr>
<tr>
<td>from left to right</td>
<td>+/0 270</td>
</tr>
<tr>
<td>from lower left to upper right</td>
<td>−/− [270, 360]</td>
</tr>
</tbody>
</table>

Points with edge amplitude 0 are assigned the edge direction 255 (undefined direction).

**Parameter**

- **Image** (input iconic) ................. (multichannel-)image(-array) \(\rightarrow\) HImageX / IHObjectX (byte)  
  Input image.

- **ImageEdgeAmp** (output iconic) ... (multichannel-)image(-array) \(\rightarrow\) HImageX / HUntypedObjectX (byte)  
  Edge amplitude (gradient magnitude) image.

- **ImageEdgeDir** (output iconic) (multichannel-)image(-array) \(\rightarrow\) HImageX / HUntypedObjectX (direction)  
  Edge direction image.

**Example**

```c
read_image(Image,’fabrik’)
frei_dir(Image,Frei_dirA,Frei_dirD)
threshold(Frei_dirA,Res,128,255).
```

**Result**

FreiDir always returns TRUE. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

**Parallelization Information**

FreiDir is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Predecessor Functions**

GaussImage, SigmaImage, MedianImage, SmoothImage

Possible Successor Functions
HysteresisThreshold, Threshold, GraySkeleton, NonmaxSuppressionDir, CloseEdges, CloseEdgesLength

See Also
BandpassImage, LaplaceOfGauss

Alternatives
EdgesImage, SobelDir, RobinsonDir, PrewittDir, KirschDir

Module
Image filters

```plaintext
[out] HImageX Highpass HImageX.HighpassImage ([in] long Width,
[in] long Height )

void HOperatorSetX.HighpassImage ([in] IObjectX Image,
[out] HUntypedObjectX Highpass, [in] long Width, [in] long Height )
```

Extract high frequency components from an image.

HighpassImage extracts high frequency components in an image by applying a linear filter with the following matrix (in case of a $7 \times 5$ matrix):

```
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 -35 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
```

This corresponds to applying a mean operator (MeanImage), and then subtracting the original gray value. A value of 128 is added to the result, i.e., zero crossings occur for 128.

This filter emphasizes high frequency components (edges and corners). The cutoff frequency is determined by the size ($\text{Height} \times \text{Width}$) of the filter matrix: the larger the matrix, the smaller the cutoff frequency is.

At the image borders the pixels’ gray values are mirrored. In case of over- or underflow the gray values are clipped (255 and 0, resp.).

Attention
If even values are passed for \text{Height} or \text{Width}, the operator uses the next larger odd value instead. Thus, the center of the filter mask is always uniquely determined.

Parameter

- **Image** (input iconic) ..................... (multichannel-)image(-array) $\sim$ HImageX / IObjectX (byte )
  Input image.

- **Highpass** (output iconic) ............ (multichannel-)image(-array) $\sim$ HImageX / HUntypedObjectX (byte )
  High-pass-filtered result image.

- **Width** (input control) ...................... extent.x $\sim$ long
  Width of the filter mask.
  Default Value : 9
  Value Suggestions : \text{Width} $\in \{3, 5, 7, 9, 11, 13, 17, 21, 29, 41, 51, 73, 101\}$
  Typical Range of Values : $3 \leq \text{Width} \leq 3$
  Minimal Value Step : 2
  Recommended Value Step : 2
  Restriction : ($\text{Width} \& \text{odd}$)

- **Height** (input control) ...................... extent.y $\sim$ long
  Height of the filter mask.
  Default Value : 9
  Value Suggestions : \text{Height} $\in \{3, 5, 7, 9, 11, 13, 17, 21, 29, 41, 51, 73, 101\}$
  Typical Range of Values : $3 \leq \text{Height} \leq 3$
  Minimal Value Step : 2
  Recommended Value Step : 2
  Restriction : ($\text{Height} \& \text{odd}$)

HALCON 6.0
HighpassImage returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information
HighpassImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successor Functions
Threshold, Skeleton

See Also
DynThreshold

Alternatives
MeanImage, SubImage, ConvolImage, BandpassImage

Image filters

Estimate the width of a filter in EdgesImage.
InfoEdges returns an estimate of the width of any of the filters used in EdgesImage. To do so, the corresponding continuous impulse responses of the filters are sampled until the first filter coefficient is smaller than five percent of the largest coefficient. Alpha is the filter parameter (see EdgesImage). Seven edge operators are supported (parameter Filter):

The parameter Mode (‘edge’/’smooth’) is used to determine whether the corresponding edge or smoothing operator is to be sampled. The Canny operator (which uses the Gaussian for smoothing) is implemented using conventional filter masks, while all other filters are implemented recursively. Therefore, for the Canny filter the coefficients of the one-dimensional impulse responses \( f(n) \) with \( n \geq 0 \) are returned in Coeffs in addition to the filter width.

Parameter

Filter (input control) ......................................................... string ~ String
Name of the edge operator.
Default Value : 'lanser2'
Value List : Filter ∈ {'deriche1', 'lanser1', 'deriche2', 'lanser2', 'shen', 'mshen', 'canny'}

Mode (input control) ........................................................ string ~ String
1D edge filter ('edge') or 1D smoothing filter ('smooth').
Default Value : 'edge'
Value List : Mode ∈ {'edge', 'smooth'}

Alpha (input control) ......................................................... real ~ double
Filter parameter: small values result in strong smoothing, and thus less detail (opposite for 'canny').
Default Value : 0.5
Typical Range of Values : 0.2 ≤ Alpha ≤ 0.2
Minimal Value Step : 0.01
Recommended Value Step : 0.1
Restriction : (Alpha > 0.0)

Size (output control) ............................................. integer ~ long
Filter width in pixels.

Coeffs (output control) ............................................ integer ~ VARIANT(integer)
For Canny filters: Coefficients of the “positive” half of the 1D impulse response.

Example

read_image(Image,’fabrik’)

info_edges(‘lanser2’,‘edge’,0.5,Size,Coeffs)
edges_image(Image,Amp,Dir,’lanser2’,0.5,’none’,-1,-1)
hysteresis_threshold(Amp,Margin,20,30,30).

Result
InfoEdges returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information
InfoEdges is reentrant and processed without parallelization.

Possible Successor Functions
EdgesImage,Threshold,Skeleton

See Also
EdgesImage

Module
Image filters

[out] HImageX ImageEdgeAmp HImageX.KirschAmp ( )
void HOperatorSetX.KirschAmp ([in] IHObjectX Image,
[out] HUntypedObjectX ImageEdgeAmp )

Detect edges (amplitude) using the Kirsch operator.
KirschAmp calculates an approximation of the first derivative of the image data and is used as an edge detector. The filter is based on the following filter masks:

\[
\begin{align*}
-3 & -3 & 5 \\
-3 & 0 & 5 \\
-3 & -3 & 5 \\
-3 & 5 & 5 \\
-3 & 0 & 5 \\
-3 & -3 & -3 \\
5 & 5 & 5 \\
-3 & 0 & -3 \\
-3 & -3 & -3 \\
5 & 5 & -3 \\
5 & 0 & -3 \\
-3 & -3 & -3 \\
5 & -3 & -3 \\
5 & 0 & -3 \\
5 & -3 & -3 \\
-3 & -3 & -3 \\
5 & 0 & -3 \\
5 & 5 & -3 \\
-3 & -3 & -3 \\
-3 & 0 & -3 \\
5 & 5 & 5 \\
-3 & -3 & -3 \\
-3 & 0 & 5 \\
-3 & 5 & 5 \\
-3 & 5 & 5
\end{align*}
\]

The result image contains the maximum response of all masks.

HALCON 6.0
ImageEdgeAmp (output iconic)................. image(-array) \sim HImageX / HUntypedObjectX (byte, int2)
Edge amplitude (gradient magnitude) image.

Example

read_image(Image,’fabrik’)
kirsch_amp(Image,Kirsch_amp)
threshold(Kirsch_amp,Edges,128,255).

Result

KirschAmp always returns TRUE. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

KirschAmp is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessor Functions

GaussImage, SigmaImage, MedianImage, SmoothImage

See Also

BandpassImage, LaplaceOfGauss

Alternatives

SobelAmp, FreiAmp, PrewittAmp, RobinsonAmp, Roberts

Module

Image filters

[out] HImageX ImageEdgeAmp HImageX.KirschDir
(out] HImageX ImageEdgeDir

void HOperatorSetX.KirschDir ([in] IHObjectX Image,
[out] HUntypedObjectX ImageEdgeAmp, [out] HUntypedObjectX ImageEdgeDir)

Detect edges (amplitude and direction) using the Kirsch operator.

KirschDir calculates an approximation of the first derivative of the image data and is used as an edge detector. The filter is based on the following filter masks:

\[
\begin{bmatrix}
-3 & -3 & 5 \\
-3 & 0 & 5 \\
-3 & -3 & 5 \\
\end{bmatrix},
\begin{bmatrix}
-3 & 5 & 5 \\
-3 & 0 & 5 \\
-3 & -3 & -3 \\
\end{bmatrix},
\begin{bmatrix}
5 & 5 & 5 \\
-3 & 0 & -3 \\
-3 & -3 & -3 \\
\end{bmatrix},
\begin{bmatrix}
5 & 5 & -3 \\
5 & 0 & -3 \\
-3 & -3 & -3 \\
\end{bmatrix},
\begin{bmatrix}
5 & -3 & -3 \\
5 & 0 & -3 \\
5 & -3 & -3 \\
\end{bmatrix}
\]
The result image contains the maximum response of all masks. The edge directions are returned in ImageEdgeDir, and are stored as \( x/2 \). They correspond to the direction of the mask yielding the maximum response.

### Parameter

- **Image** (input iconic) 
  (multichannel-)image-array \( \sim \) HImageX / HObjectX (byte) 
  Input image.
- **ImageEdgeAmp** (output iconic) 
  (multichannel-)image-array \( \sim \) HImageX / HUntypedObjectX (byte) 
  Edge amplitude (gradient magnitude) image.
- **ImageEdgeDir** (output iconic) 
  (multichannel-)image-array \( \sim \) HImageX / HUntypedObjectX (direction) 
  Edge direction image.

### Example

```plaintext
read_image(Image,'fabrik')
kirsch_dir(Image,Kirsch_dirA,Kirsch_dirD)
threshold(Kirsch_dirA,Res,128,255).
```

### Result

KirschDir always returns TRUE. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

### Parallelization Information

KirschDir is reentrant and automatically parallelized (on tuple level, channel level, domain level).

### Possible Predecessor Functions

GaussImage, SigmaImage, MedianImage, SmoothImage

### Possible Successor Functions

HysteresisThreshold, Threshold, GraySkeleton, NonmaxSuppressionDir, CloseEdges, CloseEdgesLength

### See Also

BandpassImage, LaplaceOfGauss

### Alternatives

EdgesImage, SobelDir, RobinsonDir, PrewittDir, FreiDir

### Module

**Image filters**

```plaintext

```

<no description available>

Laplace filters the input images Image using a Laplace operator. Depending on the parameter NeighbourhoodType the following simple approximations of the Laplace operator are used:
The filter 'n_8' corresponds to the filter mask used in HighpassImage(O:R:3,3:). For a Laplace operator with size $3 \times 3$, the corresponding filter is applied directly, while for larger filter sizes ($\text{Size} = 5,7,9$ and 11) the input image is first smoothed using a Gaussian filter of the selected size. Therefore,

\[
\text{laplace}(O:R:\text{int4},S,N:)
\]

for $\text{Size} >$ is equivalent to

\[
\text{gauss_image}(O:G:S: \triangleright \text{laplace}(G:R:\text{int4},3,N:)).
\]

Laplace either returns the absolute value of the Laplace filtered image ($\text{FilterType}$ 'abs') in a byte-image or the signed result ($\text{FilterType}$ 'int4') in an int4-image.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value List</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Image}$ (input iconic)</td>
<td>$\sim \text{HimageX}/\text{HObjectX}$ (byte) Input image.</td>
</tr>
<tr>
<td>$\text{ImageLaplace}$ (output iconic)</td>
<td>$(\text{multichannel-})\text{image-array} \sim \text{HimageX}/\text{HUntypedObjectX}$ (byte, int4) Laplace-filtered result images.</td>
</tr>
<tr>
<td>$\text{FilterType}$ (input control)</td>
<td>$\sim \text{String}$ Calculate the absolute value of the filter to a byte-image or the signed result to an int4-image.</td>
</tr>
<tr>
<td>$\text{Size}$ (input control)</td>
<td>$\sim 3, 5, 7, 9, 11$ Size of filter mask.</td>
</tr>
<tr>
<td>$\text{NeighbourhoodType}$ (input control)</td>
<td>$\sim \text{String}$ Neighborhood used in the Laplace operator</td>
</tr>
</tbody>
</table>

Laplace returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via $\text{SetSystem}$('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

Laplace is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successor Functions

ZeroCrossing, DualThreshold
3.5. EDGES

See Also

HighpassImage, EdgesImage

Alternatives

DiffOfGauss, LaplaceOfGauss

Module

Image filters

LoG-Operator (Laplace of Gaussian).

LaplaceOfGauss calculates the Laplace-of-Gaussian operator, i.e., the Laplace operator on a Gaussian smoothed image, for arbitrary smoothing parameters \( \Sigma \). The Laplace operator is given by:

\[
\Delta g(x, y) = \frac{\partial^2 g(x, y)}{\partial x^2} + \frac{\partial^2 g(x, y)}{\partial y^2}
\]

The derivatives in LaplaceOfGauss are calculated by appropriate derivatives of the Gaussian, resulting in the following formula for the convolution mask:

\[
\Delta G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \left( \frac{x^2 + y^2}{2\sigma^2} - 1 \right) \left[ \exp \left( -\frac{x^2 + y^2}{2\sigma^2} \right) \right]
\]

Parameter

\( \textbf{Image} \) (input iconic) ... (multichannel-)image(-array) \( \sim \) HImageX / IHObjectX (byte, int1, int2, int4, real)

Input image.

\( \textbf{ImageLaplace} \) (output iconic) .... (multichannel-)image(-array) \( \sim \) HImageX / HUntypedObjectX (int2)

Laplace filtered image.

\( \textbf{Sigma} \) (input control) .................................................. number \( \sim \) VARIANT (real, integer)

Smoothing parameter of the Gaussian.

Default Value : 2.0

Value Suggestions : \( \Sigma \in \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0\} \)

Typical Range of Values : \( 0.2 \leq \Sigma \leq 0.2 \)

Minimal Value Step : 0.01

Recommended Value Step : 0.1

Restriction : \( (\Sigma > 0.0) \)

Parallelogram Information

LaplaceOfGauss is \textit{reentrant} and automatically \textit{parallelized} (on tuple level, channel level, domain level).

Possible Successor Functions

ZeroCrossing, DualThreshold

See Also

DerivateGauss

Alternatives

Laplace, DiffOfGauss, DerivateGauss

Module

Image filters
Detect edges (amplitude) using the Prewitt operator.

PrewittAmp calculates an approximation of the first derivative of the image data and is used as an edge detector. The filter is based on the following filter masks:

$$
A = \begin{bmatrix}
1 & 1 & 1 \\
0 & 0 & 0 \\
-1 & -1 & -1 \\
\end{bmatrix}
$$

$$
B = \begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1 \\
\end{bmatrix}
$$

The result image contains the maximum response of the masks $A$ and $B$.

---

### Parameter

- **Input** (iconic) ................. (multichannel)-image(-array) $\sim HImageX / IHObjectX$ (byte, int2)
  - **Image**
  - **Input image.**

- **Output** (iconic) (multichannel)-image(-array) $\sim HImageX / HUntypedObjectX$ (byte, int2)
  - **ImageEdgeAmp**
  - **Edge amplitude (gradient magnitude) image.**

---

### Example

read_image(Image,‘fabrik’)  
prewitt_amp(Image,Prewitt)  
threshold(Prewitt,Edges,128,255).

---

**Result**

PrewittAmp always returns TRUE. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

---

**Parallelization Information**

PrewittAmp is reentrant and automatically parallelized (on tuple level, channel level, domain level).

---

**Possible Predecessor Functions**

GaussImage, SigmaImage, MedianImage, SmoothImage

---

**Possible Successor Functions**

Threshold, GraySkeleton, NonmaxSuppressionAmp, CloseEdges, CloseEdgesLength

---

**See Also**

BandpassImage, LaplaceOfGauss

---

**Alternatives**

SobelAmp, KirschAmp, FreiAmp, RobinsonAmp, Roberts

---

**Module**

Image filters

---

Detect edges (amplitude and direction) using the Prewitt operator.

---
PrewittDir calculates an approximation of the first derivative of the image data and is used as an edge detector. The filter is based on the following filter masks:

\[
A = \begin{bmatrix}
1 & 1 & 1 \\
0 & 0 & 0 \\
-1 & -1 & -1
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
1 & 0 & -1 \\
0 & 1 & -1 \\
1 & 0 & -1
\end{bmatrix}
\]

The result image contains the maximum response of the masks \(A\) and \(B\). The edge directions are returned in \(\text{ImageEdgeDir}\), and are stored in 2-degree steps, i.e., an edge direction of \(x\) degrees with respect to the horizontal axis is stored as \(x/2\) in the edge direction image. Furthermore, the direction of the change of intensity is taken into account. Let \([E_x, E_y]\) denote the image gradient. Then the following edge directions are returned as \(r/2\):

<table>
<thead>
<tr>
<th>intensity increase</th>
<th>(E_x/E_y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>edge direction (r) from bottom to top</td>
<td>(0/+ 0)</td>
</tr>
<tr>
<td>from lower right to upper left</td>
<td>(+/- [0, 90])</td>
</tr>
<tr>
<td>from right to left</td>
<td>(+/0 90)</td>
</tr>
<tr>
<td>from upper right to lower left</td>
<td>(+/+ [90, 180])</td>
</tr>
<tr>
<td>from top to bottom</td>
<td>(0/+ 180)</td>
</tr>
<tr>
<td>from left to right</td>
<td>(-/+ [180, 270])</td>
</tr>
<tr>
<td>from lower left to upper right</td>
<td>(+/- 270)</td>
</tr>
<tr>
<td>from upper left to lower right</td>
<td>(-/- [270, 360])</td>
</tr>
</tbody>
</table>

Points with edge amplitude 0 are assigned the edge direction 255 (undefined direction).

\[\text{read_image(Image,'fabrik')}\]
\[\text{prewitt_dir(Image,PrewittA,PrewittD)}\]
\[\text{threshold(PrewittA,Edges,128,255).}\]

\[\text{PrewittDir always returns TRUE. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.}\]

\[\text{PrewittDir is reentrant and automatically parallelized (on tuple level, channel level, domain level).}\]

\[\text{Possible Predecessor Functions} \quad \text{GaussImage, SigmaImage, MedianImage, SmoothImage}\]

\[\text{Possible Successor Functions} \quad \text{HysteresisThreshold, Threshold, GraySkeleton, NonmaxSuppressionDir, CloseEdges, CloseEdgesLength}\]

\[\text{See Also} \quad \text{BandpassImage, LaplaceOfGauss}\]
Detect edges using the Roberts filter.

Roberts calculates the first derivative of an image and is used as an edge operator. If the following mask describes a part of the image,

\[
\begin{pmatrix}
A & B \\
C & D
\end{pmatrix}
\]

the different filter types are defined as follows:

- `'roberts_max'`: \[\max(|A - D|, |B - C|)\]
- `'gradient_max'`: \[\max(|A + B - (C + D)|, |A + C - (B + D)|)\]
- `'gradient_sum'`: \[|A + B - (C + D)| + |A + C - (B + D)|\]

If an overflow occurs the result is clipped. The result of the operator is stored at the pixel with the coordinates of “D”.

---

**Parameter**

- **Image** (input iconic) .................. (multichannel-)image(-array) \(\sim HImageX / IObjectX\) (byte, int2)
  - Input image.
- **ImageRoberts** (output iconic) (multichannel-)image(-array) \(\sim HImageX / HUntypedObjectX\) (byte, int2)
  - Roberts-filtered result images.
- **FilterType** (input control) .............................. string \(\sim String\)
  - Filter type.
  - **Default Value**: `'gradient_sum'`
  - **Value List**: `FilterType \in \{'roberts_max', 'gradient_max', 'gradient_sum'\}`

**Example**

```plaintext
gread_image(Image,'fabrik')
roberts(Image,Roberts,'roberts_max')
threshold(Roberts,Margin,128,255).
```

---

**Result**

Roberts returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via `SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

---

**Parallelization Information**

Roberts is **reentrant** and automatically **parallelized** (on tuple level, channel level, domain level).

---

**Possible Predecessor Functions**

- GaussImage
- Threshold, Skeleton

**Possible Successor Functions**

- Threshold, Skeleton
- Laplace, HighpassImage, BandpassImage

**See Also**

Laplace, HighpassImage, BandpassImage
Detect edges (amplitude) using the Robinson operator.

RobinsonAmp calculates an approximation of the first derivative of the image data and is used as an edge detector. In RobinsonAmp the following four of the originally proposed eight $3 \times 3$ filter masks are convolved with the image. The other four masks are obtained by a multiplication by -1. All masks contain only the values 0, 1, -1, 2, -2.

\[
\begin{array}{ccc}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1 \\
2 & 1 & 0 \\
1 & 0 & -1 \\
0 & -1 & -2 \\
0 & 1 & 2 \\
-1 & 0 & 1 \\
-2 & -1 & 0 \\
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{array}
\]

The result image contains the maximum response of all masks.

Parameter

\[\textbf{Image} \quad \text{(input iconic)} \quad \text{..........} \quad \text{(multichannel-)image(-array)} \quad \sim \quad H\text{ImageX} / H\text{ObjectX} \quad \text{byte, int2} \]

\[\text{Input image.}\]

\[\textbf{ImageEdgeAmp} \quad \text{(output iconic)} \quad \text{(multichannel-)image(-array)} \quad \sim \quad H\text{ImageX} / H\text{UntypedObjectX} \quad \text{byte, int2} \]

\[\text{Edge amplitude (gradient magnitude) image.}\]

Example

read_image(Image,'fabrik')
robinson_amp(Image,Robinson_amp)
threshold(Robinson_amp,Edges,128,255).

Result

RobinsonAmp always returns TRUE. If the input is empty the behaviour can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

RobinsonAmp is \textit{reentrant} and automatically \textit{parallelized} (on tuple level, channel level, domain level).

Possible Predecessor Functions

GaussImage, SigmaImage, MedianImage, SmoothImage

See Also

BandpassImage, LaplaceOfGauss

Alternatives

SobelAmp, FreiAmp, PrewittAmp, RobinsonAmp, Roberts

Module

Image filters
Detect edges (amplitude and direction) using the Robinson operator.

RobinsonDir calculates an approximation of the first derivative of the image data and is used as an edge detector. In RobinsonAmp the following four of the originally proposed eight $3 \times 3$ filter masks are convolved with the image. The other four masks are obtained by a multiplication by -1. All masks contain only the values 0,1,-1,2,-2.

The result image contains the maximum response of all masks. The edge directions are returned in ImageEdgeDir, and are stored as $x/2$. They correspond to the direction of the mask yielding the maximum response.

```
Parameter

> Image (input iconic) ................. (multichannel-)image(-array) $\sim$ HImageX / IHObjectX ( byte )
< no description available>
> ImageEdgeAmp (output iconic) ...(multichannel-)image(-array) $\sim$ HImageX / HUntypedObjectX ( byte )
Edge amplitude (gradient magnitude) image.
> ImageEdgeDir (output iconic) (multichannel-)image(-array) $\sim$ HImageX / HUntypedObjectX ( direction )
Edge direction image.
```

```
Example

read_image(Image,‘fabrik’)
robinson_dir(Image,Robinson_dirA,Robinson_dirD)
```

```
Result

RobinsonDir always returns TRUE. If the input is empty the behaviour can be set via
SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.
```

```
Parallelization Information

RobinsonDir is reentrant and automatically parallelized (on tuple level, channel level, domain level).
```

```
Possible Predecessor Functions

GaussImage, SigmaImage, MedianImage, SmoothImage
```

```
Possible Successor Functions

HysteresisThreshold, Threshold, GraySkeleton, NonmaxSuppressionDir, CloseEdges, CloseEdgesLength
```

```
See Also

BandpassImage, LaplaceOfGauss
```

3.5. EDGES

Alternatives

EdgesImage, SobelDir, KirschDir, PrewittDir, FreiDir

Image filters

[out] HImageX EdgeAmplitude HImageX.SobelAmp ([in] String FilterType, [in] long Size )


Detect edges (amplitude) using the Sobel operator.

SobelAmp calculates first derivative of an image and is used as an edge detector. The filter is based on the following filter masks:

\[
A = \begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1
\end{bmatrix}
\]

These masks are used differently, according to the selected filter type. (In the following, \(a\) and \(b\) denote the results of convolving an image with \(A\) and \(B\) for one particular pixel.)

- 'sum_abs': \(\sqrt{a^2 + b^2}\)
- 'thin_sum_abs': \(|\text{thin}(a)| + |\text{thin}(b)|)/2\)
- 'thin_max_abs': \(\max(|\text{thin}(a)|, |\text{thin}(b)|)\)
- 'x': \(b\)
- 'y': \(a\)

Here, \(\text{thin}(x)\) is equal to \(x\) for a vertical maximum (mask A) and a horizontal maximum (mask B), respectively, and 0 otherwise. Thus, for 'thin_sum_abs' and 'thin_max_abs' the gradient image is thinned. In the case of byte images for 'x' and 'y' the value 127 is added to the result to avoid underflows. For a Sobel operator with size \(3 \times 3\), the corresponding filters \(A\) and \(B\) are applied directly, while for larger filter sizes (\(Size = 5, 7, 9\) and 11) the input image is first smoothed using a Gaussian filter of size \(Size-2\). Therefore,

\[
sobel\_amp(I:E,Dir:FilterTyp,S:)
\]

for \(Size > 3\) is equivalent to

\[
\text{gauss\_image}(I:G:S-2) \triangleright sobel\_amp(G:E:FilterType,3:).
\]

Parameter

- **Image** (input iconic) ................. (multichannel-)image(-array) \(\rightsquigarrow\) HImageX / IObjectX (byte, int2 ) Input image.
- **EdgeAmplitude** (output iconic) ....... (multichannel-)image(-array) \(\rightsquigarrow\) HImageX / HUntypedObjectX (byte, int2 ) Edge amplitude (gradient magnitude) image.
- **FilterType** (input control) .................. string \(\rightsquigarrow\) String Filter type.

Default Value : 'sum_abs'

Value List : FilterType \(\in\) \{'sum_abs', 'thin_sum_abs', 'thin_max_abs', 'sum_sqrt', 'x', 'y'\}
- **Size** (input control) .................................................................integer $\sim$ long
  Size of filter mask.
  **Default Value :** 3
  **Value List :** $\{3, 5, 7, 9, 11, 13\}$

---

Example

\[
\text{read\_image(Image,'fabrik')}
\]
\[
\text{sobel\_amp(Image,Amp,'sum\_abs',3)}
\]
\[
\text{threshold(Amp,Edg,128,255)}.
\]

---

Result

SobelAmp returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via
\[
\text{SetSystem('noObjectResult'<Result>)}.
\]
If necessary, an exception handling is raised.

---

Parallelization Information

SobelAmp is reentrant and automatically parallelized (on tuple level, channel level, domain level).

---

Possible Predecessor Functions

GaussImage, MeanImage, AnisotropeDiff, SigmaImage

---

Possible Successor Functions

Threshold, NonmaxSuppressionAmp, GraySkeleton

---

See Also

Laplace, HighpassImage, BandpassImage

---

Alternatives

FreiAmp, Roberts, KirschAmp, PrewittAmp, RobinsonAmp

---

Module

Image filters

---

```plaintext
[out] HImageX EdgeAmplitude HIMageX.SobelDir
[out] HImageX EdgeDirection, [in] String FilterType, [in] long Size

void HOperatorSetX.SobelDir ([in] IObjectX Image,
[out] HUntypedObjectX EdgeAmplitude, [out] HUntypedObjectX EdgeDirection,
[in] String FilterType, [in] long Size)
```

Detect edges (amplitude and direction) using the Sobel operator.

SobelDir calculates first derivative of an image and is used as an edge detector. The filter is based on the following filter masks:

\[
A = \begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{bmatrix}
\]
\[
B = \begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1
\end{bmatrix}
\]

These masks are used differently, according to the selected filter type. (In the following, $a$ und $b$ denote the results of convolving an image with $A$ und $B$ for one particular pixel.)

- 'sum_sqrt' $\sqrt{a^2 + b^2}$
- 'sum_abs' $(|a| + |b|)/2$

For a Sobel operator with size $3 \times 3$, the corresponding filters $A$ and $B$ are applied directly, while for larger filter sizes ($\text{Size} = 5, 7, 9$ and $11$) the input image is first smoothed using a Gaussian filter of size $\text{Size}-2$. Therefore,

\[
sobel\_dir(I:Amp,Dir:FilterTyp,S:)
\]
for $\text{Size} \geq 3$ is equivalent to

\begin{verbatim}
  gauss_image(I:G:S-2) \triangleright
  sobel_dir(G:Ampl,Dir:FilterType,3:).
\end{verbatim}

The edge directions are returned in EdgeDirection, and are stored in 2-degree steps, i.e., an edge direction of $x$ degrees with respect to the horizontal axis is stored as $x/2$ in the edge direction image. Furthermore, the direction of the change of intensity is taken into account. Let $[E_x, E_y]$ denote the image gradient. Then the following edge directions are returned as $r/2$:

\begin{align*}
\text{intensity increase} & \quad E_x / E_y \\
\text{edge direction } r & \\
\text{from bottom to top} & \quad 0/0 \\
\text{from lower right to upper left} & \quad +/ - [0, 90] \\
\text{from right to left} & \quad +/ 90 \\
\text{from upper right to lower left} & \quad +/+ [90, 180] \\
\text{from top to bottom} & \quad 0/+ 180 \\
\text{from upper left to lower right} & \quad +/- [180, 270] \\
\text{from left to right} & \quad +/0 270 \\
\text{from lower left to upper right} & \quad +/- [270, 360] \\
\end{align*}

Points with edge amplitude 0 are assigned the edge direction 255 (undefined direction).

Parameter

\begin{itemize}
  \item \textbf{Image} (input iconic) ................. (multichannel-)image(-array) $\sim HImageX / HObjectX (\text{byte, int2})$
  \hspace{1cm} Input image.
  \item \textbf{EdgeAmplitude} (output iconic) ... (multichannel-)image(-array) $\sim HImageX / HUntypedObjectX (\text{byte, int2})$
  \hspace{1cm} Edge amplitude (gradient magnitude) image.
  \item \textbf{EdgeDirection} (output iconic) ........... (multichannel-)image(-array) $\sim HImageX / HUntypedObjectX (\text{direction})$
  \hspace{1cm} Edge direction image.
  \item \textbf{FilterType} (input control) .......................... string $\sim \text{String}$
  \hspace{1cm} Filter type.
  \hspace{1cm} \textbf{Default Value} : `sum_abs`
  \hspace{1cm} \textbf{Value List} : FilterType $\in \{`sum_abs', `sum_sqrt'\}$
  \item \textbf{Size} (input control) .................................................. integer $\sim \text{long}$
  \hspace{1cm} Size of filter mask.
  \hspace{1cm} \textbf{Default Value} : 3
  \hspace{1cm} \textbf{Value List} : Size $\in \{3, 5, 7, 9, 11, 13\}$
\end{itemize}

Example

\begin{verbatim}
read_image(Image,`fabrik')
sobel_dir(Image,Amp,Dir,`sum_abs',3)
threshold(Amp,Edg,128,255).
\end{verbatim}

Result

\begin{verbatim}
SobelDir returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via 
SetSystem(`noObjectResult',<Result>). If necessary, an exception handling is raised.
\end{verbatim}

Parallelization Information

\begin{verbatim}
SobelDir is reentrant and automatically parallelized (on tuple level, channel level).
\end{verbatim}

Possible Predecessor Functions

\begin{verbatim}
GaussImage, MeanImage, AnisotropeDiff, SigmaImage
\end{verbatim}

Possible Successor Functions

\begin{verbatim}
NonmaxSuppressionDir, HysteresisThreshold, Threshold
\end{verbatim}
See Also

Roberts, Laplace, HighpassImage, BandpassImage

Alternatives

EdgesImage, FreiDir, KirschDir, PrewittDir, RobinsonDir

Module

Image filters

3.6 Enhancement


Enhance contrast of the image.

The operator Emphasize emphasizes high frequency areas of the image (edges and corners). The resulting images appears sharper.

First the procedure carries out a filtering with the low pass (MeanImage). The resulting gray values (res) are calculated from the obtained gray values (mean) and the original gray values (orig) as follows:

$$\text{res} := \text{round}((\text{orig} - \text{mean}) \times \text{Factor}) + \text{orig}$$

Factor serves as measurement of the increase in contrast. The division frequency is determined via the size of the filter matrix: The larger the matrix, the lower the division frequency.

As an edge treatment the gray values are mirrored at the edges of the image. Overflow and/or underflow of gray values is clipped (255 or 0, respectively).

Attention

Parameter

- **Image** (input iconic) ......................... (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX (byte)
  Image to be enhanced.

- **ImageEmphasize** (output iconic) (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte)
  contrast enhanced image.

- **MaskWidth** (input control) .........................extent.x \(\sim\) long
  Width of low pass mask.
  Default Value: 7
  Value Suggestions: MaskWidth \(\in\) \{3, 5, 7, 9, 11, 15, 21, 25, 31, 39\}
  Typical Range of Values: \(3 \leq\) MaskWidth \(\leq\) 3
  Minimal Value Step: 2
  Recommended Value Step: 2

- **MaskHeight** (input control) .........................extent.y \(\sim\) long
  Height of the low pass mask.
  Default Value: 7
  Value Suggestions: MaskHeight \(\in\) \{3, 5, 7, 9, 11, 15, 21, 25, 31, 39\}
  Typical Range of Values: \(3 \leq\) MaskHeight \(\leq\) 3
  Minimal Value Step: 2
  Recommended Value Step: 2


**Factor** (input control) .......................................................... real \sim double

Intensity of contrast emphasis.

**Default Value**: 1.0

**Value Suggestions**: Factor \in \{0.3, 0.5, 0.7, 1.0, 1.4, 1.8, 2.0\}

**Typical Range of Values**: 0.0 \leq \text{Factor} \leq 0.0(\sqrt{\text{sqrt}})

**Minimal Value Step**: 0.01

**Recommended Value Step**: 0.2

**Restriction**: \((0 < \text{Factor}) \land (\text{Factor} < 20)\)

---

**Example**

```c
read_image(Image,'meer_rot')
disp_image(Image,WindowHandle)
draw_region(Region,WindowHandle)
reduce_domain(Image,Region,Mask)
emphasize(Mask,Sharp,7,7,2.0)
disp_image(Sharp,WindowHandle).
```

**Result**

If the parameter values are correct the operator **Emphasize** returns the value \text{TRUE}

The behavior in case of empty input (no input images available) is set via the operator **SetSystem**(::’noObjectResult’,<Result>): . If necessary an exception handling is raised.

---

**Parallelization Information**

Emphasize is **reentrant** and automatically **parallelized** (on tuple level, channel level, domain level).

---

**Possible Successor Functions**

**DispImage**

---

**See Also**

**MeanImage, HighpassImage**

---

**Alternatives**

**MeanImage, SubImage, Laplace, AddImage**

---

**Module**

**Image filters**

---

```c
[out] HImageX ImageEquHisto HImageX.EquHistoImage ( )

void HOperatorSetX.EquHistoImage ([in] IObjectX Image,
[out] HUntypedObjectX ImageEquHisto )
```

---

**Histogram linearisation of images**

The operator **EquHistoImage** enhances the contrast. The starting point is the histogram of the input images.

The following simple gray value transformation \(f(g)\) is carried out:

\[
f(g) = 255 \sum_{x=0}^{g} h(x)
\]

\(h(x)\) describes the relative frequency of the occurrence of the gray value \(x\). This transformation linearises the cumulative histogram. Maxima in the original histogram are "spreaded" and thus the contrast in image regions with these frequently occurring gray values is increased. Supposedly homogenous regions receive more easily visible structures. On the other hand, of course, the noise in the image increases correspondingly. Minima in the original histogram are dually "compressed". The transformed histogram contains gaps, but the remaining gray values used occur approximately at the same frequency ("histogram equalization").

---

**Attention**

The operator **EquHistoImage** primarily serves for optical processing of images for a human viewer. For example, the (local) contrast spreading can lead to a detection of fictitious edges.
Parallelization Information

EquHistoImage is reentrant and automatically parallelized (on tuple level, channel level).

Possible Successor Functions

DispImage

ScaleImage

ScaleImage, ScaleImageMax, Illuminate

Bibliography


Module

Image filters

```
[out] HImageX ImageIlluminate \HImageX.Illuminate ([in] long MaskWidth, [in] long MaskHeight, [in] double Factor )

```

Illuminate image.

The operator Illuminate enhances contrast. Very dark parts of the image are "illuminated" more strongly, very light ones are "darkened". If \( orig \) is the original gray value and \( mean \) is the corresponding gray value of the low pass filtered image detected via the operators MeanImage and filter size \( \text{MaskHeight} \times \text{MaskWidth} \), the resulting gray value is \( \text{new} \):

\[
\text{new} := \text{round} ((127 - \text{mean}) \times \text{Factor} + \text{orig})
\]

The low pass should have rather large dimensions (30 x 30 to 200 x 200). Reasonable parameter combinations might be:

<table>
<thead>
<tr>
<th>MaskHeight</th>
<th>MaskWidth</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>40</td>
<td>0.55</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>0.7</td>
</tr>
<tr>
<td>150</td>
<td>150</td>
<td>0.8</td>
</tr>
</tbody>
</table>

i.e. the larger the low pass mask is chosen, the larger Factor should be as well.

The following "spotlight effect" should be noted: If, for example, a dark object is in front of a light wall the object as well as the wall, which is already light in the immediate proximity of the object contours, are lightened by the operator Illuminate. This corresponds roughly to the effect that is produced when the object is illuminated by a strong spotlight. The same applies to light objects in front of a darker background. In this case, however, the fictitious "spotlight" darkens objects.

Attention

Parameter

- **Image** (input iconic) \( \ldots \) (multichannel-)image(-array) \( \sim \) HI mageX / HIObjectX (byte)
  Image to be enhanced.
- **ImageIlluminate** (output iconic) (multichannel-)image(-array) \( \sim \) HI mageX / HUntypedObjectX (byte)
  "Illuminated" image.
3.6. ENHANCEMENT

**MaskWidth** (input control) ................................................................. \text{extent.x} \sim \text{long}
Width of low pass mask.
- **Default Value**: 101
- **Value Suggestions**: \( \text{MaskWidth} \in \{31, 41, 51, 71, 101, 121, 151, 201\} \)
- **Typical Range of Values**: \( 3 \leq \text{MaskWidth} \leq 3 \)
- **Minimal Value Step**: 2
- **Recommended Value Step**: 10

**MaskHeight** (input control) ................................................................. \text{extent.y} \sim \text{long}
Height of low pass mask.
- **Default Value**: 101
- **Value Suggestions**: \( \text{MaskHeight} \in \{31, 41, 51, 71, 101, 121, 151, 201\} \)
- **Typical Range of Values**: \( 3 \leq \text{MaskHeight} \leq 3 \)
- **Minimal Value Step**: 2
- **Recommended Value Step**: 10

**Factor** (input control) ................................................................. \text{real} \sim \text{double}
Scales the "correction gray value" added to the original gray values.
- **Default Value**: 0.7
- **Value Suggestions**: \( \text{Factor} \in \{0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0\} \)
- **Typical Range of Values**: \( 0.0 \leq \text{Factor} \leq 0.0 \)
- **Minimal Value Step**: 0.01
- **Recommended Value Step**: 0.2
- **Restriction**: \((0 < \text{Factor}) \land (\text{Factor} < 5)\)

---

**Example**

```plaintext
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
illuminate(Image,Better,40,40,0.55)
disp_image(Better,WindowHandle).
```

---

**Result**

If the parameter values are correct the operator **Illuminate** returns the value TRUE The behavior in case of empty input (no input images available) is set via the operator **SetSystem** \( \text{‘noObjectResult’,<Result>:} \). If necessary an exception handling is raised.

**Parallelization Information**

**Illuminate** is **reentrant** and automatically **parallelized** (on tuple level, channel level, domain level).

**Possible Successor Functions**

**DispImage**

**See Also**

**Emphasize, GrayHisto**

**Alternatives**

**ScaleImageMax, EquHistoImage, MeanImage, SubImage**

**Module**

**Image filters**

```plaintext
[out] HImageX ImageScaleMax HImageX.ScaleImageMax ( )
void HOperatorSetX.ScaleImageMax ([in] IObjectX Image,
[out] HUntypedObjectX ImageScaleMax )
```

**Maximum gray value spreading in the value range 0 bis 255.**

The operator **ScaleImageMax** calculates the minimum and maximum and scales the image to the maximum value range of a byte image. This way the dynamics (value range) is fully exploited. The number of different gray scales does not change, but in general the visual impression is enhanced. The gray values of images of the **real**, **int2** and **int4** type are scaled to the range 0 to 255 and returned as **byte** images.
The output always is an image of the type byte.

Parameter

- **Image** (input iconic) ............... (multichannel-)image(-array) \(\leadsto\) HImageX / HObjectX (byte, int2, int4, real) Image to be scaled.
- **ImageScaleMax** (output iconic) .............. image(-array) \(\leadsto\) HImageX / HUntypedObjectX (byte) contrast enhanced image.

Parallelization Information

ScaleImageMax is reentrant and automatically parallelized (on tuple level, channel level).

Possible Successor Functions

DispImage

See Also

MinMaxGray, GrayHisto

Alternatives

EquHistoImage, ScaleImage, Illuminate, ConvertImageType

Module

Image filters

### 3.7 FFT

Convolve an image with a byte-mask in the frequency domain.

ConvolFft convolves two (Fourier-transformed) images in the frequency domain, i.e., the pixels of the complex image ImageFFT are multiplied by the corresponding pixels of the filter ImageFilter. This image is a byte-image in which 0 corresponds to complete suppression of a frequency, and 255 corresponds to no suppression. The result image is of type 'complex'.

Attention

The filtering is always done on the entire image, i.e., the region of the image is ignored.

Parameter

- **ImageFFT** (input iconic) .................... image(-array) \(\leadsto\) HImageX / HObjectX (complex) Complex input image.
- **ImageFilter** (input iconic) .................... image \(\leadsto\) HImageX / HObjectX (byte) Filter in frequency domain.
- **ImageConvol** (output iconic) .................. image(-array) \(\leadsto\) HImageX / HUntypedObjectX (complex) Result of applying the filter.

Result

ConvolFft returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem (:::‘noObjectResult’,<Result>::). If necessary, an exception handling is raised.

Parallelization Information

ConvolFft is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

FftImage, FftGeneric, GenHighpass, GenLowpass, GenBandpass, GenBandfilter

Possible Successor Functions

PowerByte, PowerReal, PowerLn, FftImageInv, FftGeneric

See Also

GenGabor, GenHighpass, GenLowpass, GenBandpass, ConvolGabor, FftImageInv
Alternatives

ConvolGabor

Module

Image filters

[out] HImageX ImageResultGabor HImageX.ConvolGabor
([in] HImageX GaborFilter, [out] HImageX ImageResultHilbert )

void HOperatorSetX.ConvolGabor ([in] IHOBJECTX ImageFFT,
[in] IHOBJECTX GaborFilter, [out] HUNTIREDOBJECTX ImageResultGabor,
[out] HUNTIREDOBJECTX ImageResultHilbert )

Convolve an image with a Gabor filter in the frequency domain.

ConvolGabor convolves a Fourier-transformed image with a Gabor filter GaborFilter (see GenGabor) and its Hilbert transform in the frequency domain. The result image is of type 'complex'.

Attention
The filtering is always done on the entire image, i.e., the region of the image is ignored.

Parameter

▷ ImageFFT (input iconic) ................. image(-array) ~ HImageX/IHOBJECTX (complex)
Input image.

▷ GaborFilter (input iconic) ................. image ~ HImageX/IHOBJECTX (byte)
Gabor/Hilbert-Filter.

▷ ImageResultGabor (output iconic) ....... image(-array) ~ HImageX/HUNTIREDOBJECTX (complex)
Result of the Gabor filter.

▷ ImageResultHilbert (output iconic) ...... image(-array) ~ HImageX/HUNTIREDOBJECTX (complex)
Result of the Hilbert filter.

Result

ConvolGabor returns TRUE if all images are of correct type. If the input is empty the behaviour can be set via SetSystem(::'noObjectResult',<Result>:). If necessary, an exception handling is raised.

Parallelization Information
ConvolGabor is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
FftImage, FftGeneric, GenGabor

Possible Successor Functions
PowerByte, PowerReal, PowerLn, FftImageInv, FftGeneric

See Also

ConvolImage

Alternatives

ConvolFft

Module

Image filters


void HOperatorSetX.EnergyGabor ([in] IHOBJECTX ImageGabor,

Calculate the energy of a two-channel image.

EnergyGabor calculates the local contrast (Energy) of the two input images. The energy of the resulting image is then defined as

\[ Energy = \text{channel1}^2 + \text{channel2}^2 \]
Often the calculation of the energy is preceded by the convolution of an image with a Gabor filter and the Hilbert transform of the Gabor filter (see `ConvolGabor`). In this case, the first channel of the image passed to `EnergyGabor` is the Gabor-filtered image, transformed back into the spatial domain (see `FftImageInv`), and the second channel the result of the convolution with the Hilbert transform, also transformed back into the spatial domain. The local energy is a measure for the local contrast of structures (e.g., edges and lines) in the image.

**Parameter**

- **ImageGabor** (input iconic) .......................... image(-array) 1st channel of input image (usually: Gabor image).
- **ImageHilbert** (input iconic) .......................... image(-array) 2nd channel of input image (usually: Hilbert image).
- **Energy** (output iconic) .......................... image(-array) Image containing the local energy.

EnergyGabor returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via `SetSystem(::'noObjectResult',<Result>:)`.

**Parallelization Information**

EnergyGabor is reentrant and automatically parallelized (on tuple level, domain level).

**Possible Predecessor Functions**

GenGabor, ConvolGabor, FftImageInv

**Module**

Image filters


`FftGeneric` computes the fast Fourier transform of an image. Because several definitions of the forward and reverse transforms exist in the literature, this operator allows the user to select the most convenient definition.

The general definition of a Fourier transform is as follows:

\[
F(m, n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} e^{2\pi i (km + ln / N)} f(k, l)
\]

Opinions vary on whether the sign \( \ast \) in the exponent should be set to 1 or \(-1\) for the forward transform, i.e. the for going to the frequency domain. There is also disagreement on the magnitude of the normalizing factor \( c \). This is sometimes set to 1 for the forward transform, sometimes to \( M N \), and sometimes (in case of the unitary FFT) to \( \sqrt{M N} \). Especially in image processing applications the DC term is shifted to the center of the image.

`FftGeneric` allows to select these choices individually. The parameter `Direction` allows to select the logical direction of the FFT. (This parameter is not unnecessary; it is needed to discern how to shift the image if the DC term should rest in the center of the image.) Possible values are `'to_freq'` and `'from_freq'`. The parameter `Exponent` is used to determine the sign of the exponent. It can be set to 1 or \(-1\). The normalizing factor can be set with `Norm`, and can take on the values `'none'`, `'sqrt'` and `'n'`. The parameter `Mode` determines the location of the DC term of the FFT. It can be set to `'dc_center'` or `'dc_edge'`.

In any case, the user must ensure the consistent use of the parameters. This means that the normalizing factors used for the forward and backward transform must yield \( M N \) when multiplied, the exponents must be of opposite sign, and `Mode` must be equal for both transforms.
A consistent combination is, for example ("to_freq",-1,\'n','dc_edge") for the forward transform and ("from_freq",1,\'none','dc_edge") for the reverse transform. In this case, the FFT can be interpreted as interpolation with trigonometric basis functions. Another useful combination is ("to_freq",1,\'sqrt','dc_center") and ("from_freq",-1,\'sqrt','dc_center").

### Parameter

- **Image** (input iconic)  
  image(-array) \(\sim\) \(H\)ImageX / HObjectX (any)  
  Input image.

- **ImageFFT** (output iconic)  
  image(-array) \(\sim\) \(H\)ImageX / HUntypedObjectX (complex)  
  Fourier-transformed image.

- **Direction** (input control)  
  string \(\sim\) String  
  Calculate forward or reverse transform.  
  **Default Value:** \('to_freq'\)  
  **Value List:** \{'to_freq', 'from_freq'\}

- **Exponent** (input control)  
  integer \(\sim\) long  
  Sign of the exponent.  
  **Default Value:** 1  
  **Value List:** \{-1, 1\}

- **Norm** (input control)  
  string \(\sim\) String  
  Normalizing factor of the transform.  
  **Default Value:** \'sqrt'  
  **Value List:** \{'none', 'sqrt', 'n'\}

- **Mode** (input control)  
  string \(\sim\) String  
  Location of the DC term in the frequency domain.  
  **Default Value:** \'dc_center'  
  **Value List:** \{'dc_center', 'dc_edge'\}

### Result

FftGeneric returns TRUE if the input image is of correct type, its width and height are a power of 2, and all parameters are correct. If the input is empty the behaviour can be set via SetSystem(:,:,\'noObjectResult',<Result>):). If necessary, an exception handling is raised.

### Parallelization Information

FftGeneric is reentrant and automatically parallelized (on tuple level).

### Possible Successor Functions

ConvolFft, ConvolGabor, ConvertImageType, PowerByte, PowerReal, PowerLn, PhaseDeg, PhaseRad, EnergyGabor

### Alternatives

FftImage, FftImageInv

### Module

Image filters

```c
[out] HImageX ImageFFT HImageX.FftImage ( )

void HOperatorSetX.FftImage ([in] HObjectX Image,  
[out] HUntypedObjectX ImageFFT )
```

**Compute the fast Fourier transform of an image.**

FftImage calculates the Fourier transform of the input image (Image), i.e., it transforms the image into the frequency domain. The algorithm used is the fast Fourier transform. This corresponds to the call

```
FftGeneric(Image, ImageFFT,'toFreq',1,'n','dcCenter':)
```

The result image is of type \'complex\'.

### Attention

The filtering is always done on the entire image, i.e., the region of the image is ignored. The images must be quadratic and the width and height must be a power of 2.
### Parameter

- **Image** (input iconic) ................................. image(-array)  .getImageX / IObjectX (byte, real)
  
  Input image.

- **ImageFFT** (output iconic) .............................. image(-array)  .getImageX / IObjectX (complex)
  
  Fourier-transformed image.

### Result

FftImage returns TRUE if the input image is of correct type and its width and height are a power of 2. If the input is empty the behaviour can be set via SetSystem(  :'noObjectResult',<Result>:). If necessary, an exception handling is raised.

### Parallelization Information

FftImage is reentrant and automatically parallelized (on tuple level).

### Possible Successor Functions

ConvolFft, ConvolGabor, ConvertImageType, PowerByte, PowerReal, PowerLn, PhaseDeg, PhaseRad

### See Also

FftImageInv

### Alternatives

FftGeneric

### Module

Image filters

```c
[out] HImageX ImageFFTInv HImageX.FftImageInv ( )

void HOperatorSetX.FftImageInv ([in] IObjectX Image,
[out] IUntypedObjectX ImageFFTInv )
```

Compute the inverse fast Fourier transform of an image.

FftImageInv calculates the inverse Fourier transform of the input image (Image), i.e., it transforms the image back into the spatial domain. This corresponds to the call

```
FftGeneric(Image, ImageFFT, 'fromFreq', -1, 'sqrt', 'dcCenter')
ConvertImageType(ImageFFT, ImageFFTInv, 'byte') .
```

The result image is of type 'byte'.

### Attention

The filtering is always done on the entire image, i.e., the region of the image is ignored. The images must be quadratic and the width and height must be a power of 2.

### Parameter

- **Image** (input iconic) ................................. image(-array)   getImageX / IObjectX (complex)
  
  Input image.

- **ImageFFTInv** (output iconic) .............................. image(-array)   getImageX / IObjectX (byte)
  
  Inverse-Fourier-transformed image.

FftImageInv returns TRUE if the input image is of correct type and its width and height are a power of 2. If the input is empty the behaviour can be set via SetSystem(  :'noObjectResult',<Result>:). If necessary, an exception handling is raised.

### Parallelization Information

FftImageInv is reentrant and automatically parallelized (on tuple level).
3.7. FFT

See Also
FftImage, FftGeneric, EnergyGabor

Alternatives
FftGeneric

Module
Image filters

Generate an ideal band filter.

GenBandfilter generates an ideal band filter in the frequency domain. The DC term is assumed to lie in the center of the image. The parameters MinFrequency and MaxFrequency determine the cutoff frequencies of the filter (in pixels). The resulting image contains an annulus with the value 0, and the value 255 outside of this annulus.

Parameter

- **ImageFilter** (output iconic) ......................... image ~ HImageX / HUntypedObjectX (byte)
  Band filter in the frequency domain.
- **MinFrequency** (input control) .......................... real ~ double
  Minimum frequency.
  Default Value: 20
  Value Suggestions: MinFrequency ∈ {10, 20, 30, 40, 50, 60, 70, 100}
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: (MinFrequency > 0)
- **MaxFrequency** (input control) .......................... real ~ double
  Maximum frequency.
  Default Value: 40
  Value Suggestions: MaxFrequency ∈ {10, 20, 30, 40, 50, 60, 70, 100}
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: (MaxFrequency > 0)
- **Size** (input control) .......................... integer ~ long
  Size (dimension) of the image (filter).
  Default Value: 512
  Value List: Size ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Result

GenBandfilter returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

Parallelization Information

GenBandfilter is reentrant and processed without parallelization.

Possible Successor Functions

ConvolFft

See Also

GenHighpass, GenLowpass, GenBandpass

Alternatives

GenCircle, PaintRegion

Module

Image filters
Generate an ideal bandpass filter.

GenBandpass generates an ideal bandpass filter in the frequency domain. The DC term is assumed to lie in the center of the image. The parameters \( \text{MinFrequency} \) and \( \text{MaxFrequency} \) determine the cutoff frequencies of the filter (in pixels). The resulting image contains an annulus with the value 255, and the value 0 outside of this annulus.

\[
\begin{align*}
\text{ImageBandpass} & \quad \text{(output iconic)} \quad \text{image} \sim HImageX/HUntypedObjectX(\ \text{byte}) \\
\text{MinFrequency} & \quad \text{(input control)} \quad \text{real} \sim double \\
\text{MaxFrequency} & \quad \text{(input control)} \quad \text{real} \sim double \\
\text{Size} & \quad \text{(input control)} \quad \text{integer} \sim long
\end{align*}
\]

Result

GenBandpass returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

Parallelization Information

GenBandpass is \textit{reentrant} and processed \textit{without} parallelization.

Possible Successor Functions

ConvolFft

See Also

GenHighpass, GenLowpass, GenBandfilter

Alternatives

GenCircle, PaintRegion

Module

Image filters

\[
\begin{align*}
\text{void HImageX.GenFilterMask} & \quad \text{[in]} \ \text{String FileName, [in]} \ \text{double Scale, [in]} \ \text{long Size} \\
\text{void HOperatorSetX.GenFilterMask} & \quad \text{[out]} \ \text{HUntypedObjectX ImageFilter, [in]} \ \text{String FileName, [in]} \ \text{double Scale, [in]} \ \text{long Size}
\end{align*}
\]

Store a filter mask in the spatial domain as a real-image.
3.7. FFT

GenFilterMask stores a filter mask in the spatial domain as a real-image. The center of the filter mask lies in the center of the resulting image. The parameter Scale determines by which amount the values of the filter mask are multiplied (this results in larger values of the Fourier transform of the filter). The file format of the filter mask file is described with the operator ConvolImage. Example filter masks can be found in the directory “filter” in the HALCON home directory. This operator is useful for visualizing the frequency response of filter masks (by applying a Fourier transform to the result image of this operator).

Parameter

- **ImageFilter** (output iconic) image(-array) \(\sim HImageX / HUntypedObjectX(\text{real})\)
  - Filter in the spatial domain.

- **FileName** (input control) string \(\sim\) String
  - File name of the filter mask.
  - Default Value: 'sobel'
  - Value Suggestions: FileName: {'laplace4', 'laplace8', 'lowpass_3_3'}

- **Scale** (input control) real \(\sim\) double
  - Scaling factor.
  - Default Value: 1.0
  - Value Suggestions: Scale: {0.3, 0.5, 0.75, 1.0, 1.25, 1.5, 2.0}
  - Typical Range of Values: 0.001 \(\leq\) Scale \(\leq\) 0.001
  - Minimal Value Step: 0.001
  - Recommended Value Step: 0.1
  - Restriction: (Scale > 0.0)

- **Size** (input control) integer \(\sim\) long
  - Size (dimension) of the image (filter).
  - Default Value: 512
  - Value List: Size \(\in\) {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Parallelization Information

GenFilterMask is reentrant and processed without parallelization.

Possible Successor Functions

FftImage, FftGeneric

See Also

ConvolImage

Module

Image filters

```c
void HImageX.GenGabor([in] double Angle, [in] double Frequency,
```

```c
void HOperatorSetX.GenGabor([out] HUntypedObjectX ImageFilter,
[in] double Angle, [in] double Frequency, [in] double Bandwidth,
```

Generate a Gabor filter.

GenGabor generates a Gabor filter with a user-definable bandpass frequency range and sign for the Hilbert transformation. This is done by calculating a symmetrical filter in the frequency domain, which can be adapted by the parameters Angle, Frequency, Bandwidth and Orientation such that a certain frequency band and a certain direction range in the spatial domain is filtered out in the frequency domain.

The parameters Frequency (central frequency = distance from the DC term) and Orientation (direction) determine the center of the filter. Larger values of Frequency result in higher frequencies being passed. A value of 0 for Orientation generates a horizontally oriented “crescent” (the bulge of the crescent points upward). Higher values of Orientation result in the counterclockwise rotation of the crescent.

The parameters Angle and Bandwidth are used to determine the range of frequencies and angles being passed by the filter. The larger Angle is, the smaller the range of angles passed by the filter gets (because the “crescent” gets narrower). The larger Bandwidth is, the smaller the frequency band being passed gets (because the “crescent” gets thinner).
The resulting image is a byte-image, containing the Gabor filter in the upper half of the image (symmetry of the Gabor filter), and the corresponding sign values needed for the calculation of the Hilbert transform, which is needed to compute the local energy of the image, in the lower half of the image.

**Parameter**

- **ImageFilter** (output iconic) ........................................ image(-array) \(\sim\) HImageX / HUntypedObjectX (byte) 
  Gabor filter (and sign values).

- **Angle** (input control) .................................................. real \(\sim\) double 
  Angle range, inversely proportional to the range of orientations.
  
  Default Value : 1.4
  Value Suggestions : \(\text{Angle} \in \{1.0, 1.2, 1.4, 1.6, 2.0, 2.5, 3.0, 5.0, 6.0, 10.0, 20.0, 30.0, 50.0, 70.0, 100.0\}\)
  Typical Range of Values : \(1.0 \leq \text{Angle} \leq 1.0\)
  Minimal Value Step : 0.001
  Recommended Value Step : 0.1

- **Frequency** (input control) ........................................... real \(\sim\) double 
  Distance of the center of the filter to the DC term.
  
  Default Value : 0.4
  Value Suggestions : \(\text{Frequency} \in \{0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.699\}\)
  Typical Range of Values : \(0.0 \leq \text{Frequency} \leq 0.0\)
  Minimal Value Step : 0.00001
  Recommended Value Step : 0.005

- **Bandwidth** (input control) .......................................... real \(\sim\) double 
  Bandwith range, inversely proportional to the range of frequencies being passed.
  
  Default Value : 1.0
  Value Suggestions : \(\text{Bandwidth} \in \{0.1, 0.3, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0, 50.0\}\)
  Typical Range of Values : \(0.05 \leq \text{Bandwidth} \leq 0.05\)
  Minimal Value Step : 0.001
  Recommended Value Step : 0.1

- **Orientation** (input control) ....................................... real \(\sim\) double 
  Angle of the principal orientation being passed.
  
  Default Value : 1.5
  Value Suggestions : \(\text{Orientation} \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0, 3.14\}\)
  Typical Range of Values : \(0.0 \leq \text{Orientation} \leq 0.0\)
  Minimal Value Step : 0.0001
  Recommended Value Step : 0.05

- **Size** (input control) .................................................. integer \(\sim\) long 
  Size (dimension) of the image (filter).
  
  Default Value : 512
  Value List : \(\text{Size} \in \{8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192\}\)

**Result**

GenGabor returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

**Parallelization Information**

GenGabor is reentrant and processed without parallelization.

**Possible Predecessor Functions**

FftImage, FftGeneric

**Possible Successor Functions**

ConvolGabor

**See Also**

FftImageInv, EnergyGabor

**Alternatives**

GenBandpass, GenBandfilter, GenHighpass, GenLowpass

**Module**

Image filters
Generate an ideal highpass filter.

GenHighpass generates an ideal highpass filter in the frequency domain. The DC term is assumed to lie in the center of the image. The parameter Frequency determines the cutoff frequency of the filter (in pixels). The resulting image contains a circle of radius Frequency with the value 0, and the value 255 outside of this circle.

**Parameter**

- **ImageHighpass** (output iconic) ....................... image  \( \rightarrow \) **HImageX / HUntypedObjectX (byte)**
  Highpass filter in the frequency domain.

  - **Frequency** (input control) .................................................. real  \( \rightarrow \) **double**
    Cutoff frequency.
    - **Default Value**: 20
    - **Value Suggestions**: Frequency \( \in \{10, 20, 30, 40, 50, 60, 70, 100\} \)
    - **Typical Range of Values**: \(1 \leq \text{Frequency} \leq 1\)
    - **Minimal Value Step**: 1
    - **Recommended Value Step**: 1
    - **Restriction**: \( \text{Frequency} > 0 \)

- **Size** (input control) .................................................. integer  \( \rightarrow \) **long**
  Size (dimension) of the image (filter).
  - **Default Value**: 512
  - **Value List**: Size \( \in \{16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192\} \)

**Result**

GenHighpass returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

**Parallelization Information**

GenHighpass is reentrant and processed without parallelization.

**Possible Successor Functions**

- **ConvolFft**

**See Also**

ConvolFft, GenLowpass, GenBandpass, GenBandfilter

**Alternatives**

GenCircle, PaintRegion

**Module**

Image filters

Generate an ideal lowpass filter.

GenLowpass generates an ideal lowpass filter in the frequency domain. The DC term is assumed to lie in the center of the image. The parameter Frequency determines the cutoff frequency of the filter (in pixels). The resulting image contains a circle of radius Frequency with the value 255, and the value 0 outside of this circle.

**Parameter**

- **ImageLowpass** (output iconic) ....................... image  \( \rightarrow \) **HImageX / HUntypedObjectX (byte)**
  Highpass filter in the frequency domain.
**Frequency** (input control) ..............................<double>  
Cutoff frequency.  
**Default Value**: 20  
**Value Suggestions**: $\{10, 20, 30, 40, 50, 60, 70, 100\}$  
**Typical Range of Values**: $1 \leq \text{Frequency} \leq 1$  
**Minimal Value Step**: 1  
**Recommended Value Step**: 1

**Size** (input control) ......................................<long>  
Size (dimension) of the image (filter).  
**Default Value**: 512  
**Value List**: $\{16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192\}$

---

**ImageFilter** (output iconic) ......................<byte>  
Bandpass filter as image in the frequency domain.  
**Default Value**: 20  
**Value Suggestions**: $\{0.0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 80, 100, 150, 200\}$  
**Typical Range of Values**: $0.0 \leq \text{Frequency} \leq 0.0$  
**Minimal Value Step**: 1  
**Recommended Value Step**: 2

---

**Frequency** (input control) ..............................<double>  
Distance of the filter’s maximum from the DC term.  
**Default Value**: 20  
**Value Suggestions**: $\{0.0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 80, 100, 150, 200\}$  
**Typical Range of Values**: $0.0 \leq \text{Frequency} \leq 0.0$  
**Minimal Value Step**: 1  
**Recommended Value Step**: 2

---

**Size** (input control) ......................................<long>  
Size (dimension) of the image (filter).  
**Default Value**: 512  
**Value List**: $\{8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192\}$

---

GenLowpass returns TRUE if all parameters are correct. If necessary, an exception handling is raised.  
---

**Parallelization Information**  
GenLowpass is reentrant and processed without parallelization.  
---

**Possible Successor Functions**  
ConvolFft, GenHighpass, GenBandpass, GenBandfilter, GenCircle, PaintRegion  
---

**Alternatives**

---

**Module**  
Image filters

---

**Parameter**

---

Generate a bandpass filter with sinusoidal shape.  

**ImageFilter** (output iconic) ......................<byte>  
Bandpass filter as image in the frequency domain.  
**Default Value**: 20  
**Value Suggestions**: $\{0.0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 80, 100, 150, 200\}$  
**Typical Range of Values**: $0.0 \leq \text{Frequency} \leq 0.0$  
**Minimal Value Step**: 1  
**Recommended Value Step**: 2

---

**Frequency** (input control) ..............................<double>  
Distance of the filter’s maximum from the DC term.  
**Default Value**: 20  
**Value Suggestions**: $\{0.0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 80, 100, 150, 200\}$  
**Typical Range of Values**: $0.0 \leq \text{Frequency} \leq 0.0$  
**Minimal Value Step**: 1  
**Recommended Value Step**: 2

---

**Size** (input control) ......................................<long>  
Size (dimension) of the image (filter).  
**Default Value**: 512  
**Value List**: $\{8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192\}$

---

GenSinBandpass returns TRUE if all parameters are correct. If necessary, an exception handling is raised.  
---

**Parallelization Information**  
GenSinBandpass is reentrant and processed without parallelization.  
---

**Possible Predecessor Functions**  
FftImage, FftGeneric
Possible Successor Functions

ConvolFft

See Also

FftImageInv

Alternatives

GenStdBandpass, GenBandpass, GenBandfilter, GenHighpass, GenLowpass

Image filters

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
</table>

Generate a bandpass filter with Gaussian or sinusoidal shape.

GenStdBandpass generates a rotationally invariant bandpass filter with the response being determined by the parameters `Frequency` and `Sigma`: `Frequency` determines the location of the maximum response with respect to the DC term, while `Sigma` determines the width of the frequency band that passes the filter. For `Mode = 'gauss'`, a Gaussian response is generated with `Sigma` being the standard deviation. For `Mode = 'sin'`, a sine function is generated with the maximum at `Frequency` and the extent `Sigma`.

Parameter

- **ImageFilter** (output iconic) ........................................ image(-array)  \( \sim \) HImageX / HUntypedObjectX (byte)  
  Bandpass filter as image in the frequency domain.
- **Frequency** (input control) ..................................................real  \( \sim \) double  
  Distance of the filter’s maximum from the DC term.
  Default Value: 20  
  Value Suggestions: `Frequency \in \{0.0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 80, 100, 150, 200\}`  
  Typical Range of Values: \(0.0 \leq \text{Frequency} \leq 0.0\)  
  Minimal Value Step: 1  
  Recommended Value Step: 2
- **Sigma** (input control) .....................................................real  \( \sim \) double  
  Bandwidth of the filter (standard deviation).
  Default Value: 1.0  
  Value Suggestions: `Sigma \in \{0.1, 0.3, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0, 50.0\}`  
  Typical Range of Values: \(0.05 \leq \text{Sigma} \leq 0.05\)  
  Minimal Value Step: 0.001  
  Recommended Value Step: 0.1
- **Size** (input control) ...................................................integer  \( \sim \) long  
  Size (dimension) of the image (filter).
  Default Value: 512  
  Value List: `Size \in \{8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192\}`
- **Mode** (input control) ....................................................string  \( \sim \) String  
  Filter type.
  Default Value: 'sin'  
  Value List: `Mode \in \{\text{'sin'}, \text{'gauss'}\}`

Result

GenStdBandpass returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

Parallellization Information

GenStdBandpass is reentrant and processed without parallelization.

Possible Predecessor Functions

FftImage, FftGeneric

Possible Successor Functions

ConvolFft
Return the phase of a complex image in degrees.

PhaseDeg computes the phase of a complex image in degrees. The following formula is used:

$$\text{phase} = \frac{180\tan^{-1}(\text{imaginary part}/\text{real part})}{\pi}.$$ 

**Parameter**

- **ImageComplex** (input iconic) ...................... image(-array) \(\rightsquigarrow\) HImageX / IObjectX (complex)
  - Input image in frequency domain.
- **ImagePhase** (output iconic) ...................... image(-array) \(\rightsquigarrow\) HImageX / HUntypedObjectX (direction)
  - Phase of the image in degrees.

PhaseDeg returns TRUE if the image is of correct type. If the input is empty the behaviour can be set via SetSystem(::'noObjectResult',<Result>:). If necessary, an exception handling is raised.

**Parallelization Information**

PhaseDeg is reentrant and automatically parallelized (on tuple level, domain level).

**Possible Predecessor Functions**

FftImage, FftGeneric

**Possible Successor Functions**

DispImage

---

Return the phase of a complex image in radians.

PhaseRad computes the phase of a complex image in radians. The following formula is used:

$$\text{phase} = \tan^{-1}(\text{imaginary part}/\text{real part})$$.
### 3.7. FFT

**Parameter**

- **ImageComplex** (input iconic) ......................... image(-array) ~ HImageX / IObjectX ( complex )
  - Input image in frequency domain.

- **ImagePhase** (output iconic) ......................... image(-array) ~ HImageX / HUntypedObjectX ( real )
  - Phase of the image in radians.

**Result**

PhaseRad returns TRUE if the image is of correct type. If the input is empty the behaviour can be set via SetSystem(::’noObjectResult’,<Result>):. If necessary, an exception handling is raised.

**Parallelization Information**

PhaseRad is reentrant and automatically parallelized (on tuple level, domain level).

**Possible Predecessor Functions**

- FftImage, FftGeneric
- DispImage
- FftImageInv
- PhaseDeg

**Module**

Image filters

---

```c
[out] HImageX PowerByte HImageX.PowerByte ( )
void HOperatorSetX.PowerByte ([in] IObjectX Image,
[out] HUntypedObjectX PowerByte )
```

Return the power spectrum of a complex image.

PowerByte computes the power spectrum from the real and imaginary parts of a Fourier-transformed image (see FftImage), i.e., the modulus of the frequencies. The result image is of type 'byte'. The following formula is used:

\[
\sqrt{\text{real part}^2 + \text{imaginary part}^2}
\]

**Parameter**

- **Image** (input iconic) ......................... image(-array) ~ HImageX / IObjectX ( complex )
  - Input image in frequency domain.

- **PowerByte** (output iconic) ......................... image(-array) ~ HImageX / HUntypedObjectX ( byte )
  - Power spectrum of the input image.

**Result**

PowerByte returns TRUE if the image is of correct type. If the input is empty the behaviour can be set via SetSystem(::’noObjectResult’,<Result>):. If necessary, an exception handling is raised.

**Parallelization Information**

PowerByte is reentrant and automatically parallelized (on tuple level, domain level).

**Possible Predecessor Functions**

- FftImage, FftGeneric, ConvolFft, ConvolGabor
- DispImage
- FftImage

**Possible Successor Functions**

- See Also

**Alternatives**

- AbsImage, ConvertImageType, PowerReal, PowerLn

**Module**

Image filters
Return the power spectrum of a complex image.

PowerLn computes the power spectrum from the real and imaginary parts of a Fourier-transformed image (see FftImage), i.e., the modulus of the frequencies. Additionally, the natural logarithm is applied to the result. The result image is of type 'real'. The following formula is used:

\[
\ln \left( \sqrt{\text{real part}^2 + \text{imaginary part}^2} \right)
\]

Parameter

\[\text{Image}\] (input iconic) image(-array) \(\sim\) HImageX / IObjectX (complex)
Input image in frequency domain.

\[\text{ImageResult}\] (output iconic) image(-array) \(\sim\) HImageX / HUntypedObjectX (real)
Power spectrum of the input image.

Result

PowerLn returns TRUE if the image is of correct type. If the input is empty the behaviour can be set via SetSystem(::'noObjectResult',<Result>:). If necessary, an exception handling is raised.

Parallelization Information

PowerLn is reentrant and automatically parallelized (on tuple level, domain level).

Possible Predecessor Functions

FftImage, FftGeneric, ConvolFft, ConvolGabor

Possible Successor Functions

DispImage, ConvertImageType, ScaleImage

See Also

FftImage, FftGeneric

Alternatives

AbsImage, ConvertImageType, PowerReal, PowerByte

Module

Image filters

Return the power spectrum of a complex image.

PowerReal computes the power spectrum from the real and imaginary parts of a Fourier-transformed image (see FftImage), i.e., the modulus of the frequencies. The result image is of type 'real'. The following formula is used:

\[
\sqrt{\text{real part}^2 + \text{imaginary part}^2}
\]

Parameter

\[\text{Image}\] (input iconic) image(-array) \(\sim\) HImageX / IObjectX (complex)
Input image in frequency domain.

\[\text{ImageResult}\] (output iconic) image(-array) \(\sim\) HImageX / HUntypedObjectX (real)
Power spectrum of the input image.

Result

PowerReal returns TRUE if the image is of correct type. If the input is empty the behaviour can be set via SetSystem(::'noObjectResult',<Result>:). If necessary, an exception handling is raised.
### Parallelization Information

PowerReal is *reentrant* and automatically *parallelized* (on *tuple level*, *domain level*).

#### Possible Predecessor Functions

FftImage, FftGeneric, ConvolFft, ConvolGabor

#### Possible Successor Functions

DispImage, ConvertImageType, ScaleImage

#### See Also

FftImage

#### Alternatives

AbsImage, ConvertImageType, PowerByte, PowerLn

#### Module

Image filters

---

### 3.8 Lines

#### Function Signature

```cpp
[out] HImageX ImageBandpass HImageX::BandpassImage
([in] String FilterType )

void HOperatorSetX::BandpassImage ([in] IObjectX Image,
[out] HUntypedObjectX ImageBandpass, [in] String FilterType )
```

Edge extraction using bandpass filters.

BandpassImage serves as an edge filter. It applies a linear filter with the following convolution mask to Image:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>-2</td>
<td>3</td>
<td>12</td>
<td>3</td>
<td>-2</td>
</tr>
<tr>
<td>-2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
</tr>
</tbody>
</table>

At the border of the image the gray values are mirrored. Over- and underflows of gray values are clipped to the interval [0,255]. The resulting images are returned in ImageBandpass.

#### Attention

BandpassImage returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(::’noObjectResult’,<Result>). If necessary, an exception handling is raised.

#### Parameter

- **Image** (input iconic) ………………… (multichannel-)image(-array) ～ HImageX / IObjectX (byte)
  Input images.
- **ImageBandpass** (output iconic) … (multichannel-)image(-array) ～ HImageX / HUntypedObjectX (byte)
  Bandpass-filtered images.
- **FilterType** (input control) ………………………………………………………………… string ～ String
  Filter type: currently only ‘lines’ is supported.
  **Default Value**: ‘lines’
  **Value List**: FilterType ∈ {'lines'}

#### Result

BandpassImage is *reentrant* and automatically *parallelized* (on *tuple level*, *channel level*, *domain level*).
Detection of lines using the facet model.

The operator LinesFacet can be used to extract lines (curvilinear structures) from the image Image. The extracted lines are returned in Lines as sub-pixel precise XLD-contours. The parameter LightDark determines, whether bright or dark lines are extracted.

The extraction is done by using the facet model, i.e., a least squares fit, to determine the parameters of a quadratic polynomial in x and y for each point of the image. The parameter MaskSize determines the size of the window used for the least squares fit. Larger values of MaskSize lead to a larger smoothing of the image, but can lead to worse localization of the line. The parameters of the polynomial are used to calculate the line direction for each pixel. Pixels which exhibit a local maximum in the second directional derivative perpendicular to the line direction are marked as line points. The line points found in this manner are then linked to contours. This is done by immediately accepting line points that have a second derivative larger than High. Points that have a second derivative smaller than Low are rejected. All other line points are accepted if they are connected to accepted points by a connected path. This is similar to a hysteresis threshold operation with infinite path length (see HysteresisThreshold). However, this function is not used internally since it does not allow the extraction of sub-pixel precise contours.

The gist of how to select the thresholds in the description of LinesGauss also holds for this operator. A value of Sigma = 1.5 there roughly corresponds to a MaskSize of 5 here.

The extracted lines are returned in a topologically sound data structure in Lines. This means that lines are correctly split at junction points.

Attention

The smaller the filter size MaskSize is chosen, the more short, fragmented lines will be extracted. This can lead to considerably longer execution times.

Parameter

d Image (input iconic) .......................... image ⇝ HImageX / IObjectX (byte, int1, int2, int4, real) 
Input image.

d Lines (output iconic) .......................... xld_cont ⇝ HXLDContX / HUntypedObjectX 
Extracted lines.

d MaskSize (input control) ......................... integer ⇝ long 
Size of the facet model mask.

Default Value : 5
Value List : MaskSize ∈ {3, 5, 7, 9, 11}

Low (input control) .............................. number ⇝ VARIANT (real, integer) 
Lower threshold for the hysteresis threshold operation.

Default Value : 3
Value Suggestions : Low ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10}

Typical Range of Values : 0 ≤ Low ≤ 0

Recommended Value Step : 0.5
Restriction : (Low ≥ 0)
3.8. LINES

- **High** (input control) ......................................................... number → VARIANT (real, integer)
  
  Upper threshold for the hysteresis threshold operation.
  
  **Default Value**: 8
  
  **Value Suggestions**: High ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10, 12, 15, 18, 20, 25}
  
  **Typical Range of Values**: 0 ≤ High ≤ 0
  
  **Recommended Value Step**: 0.5
  
  **Restriction**: ((High ≥ 0) ∧ (High ≥ Low))

- **LightDark** (input control) .................................................. string → String
  
  Extract bright or dark lines.
  
  **Default Value**: 'light'
  
  **Value List**: LightDark ∈ {'dark', 'light'}

---

/* Detection of lines in an aerial image */
read_image(Image,’mreut4_3’)
lines_facet(Image,Lines,5,3,8,’light’)
disp_xld(Lines,WindowHandle).

---

Result

LinesFacet returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behaviour can be set via SetSystem(::’noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

LinesFacet is reentrant and automatically parallelized (on tuple level).

Possible Successor Functions

GenPolygonsXld

See Also

BandpassImage, DynThreshold, TopographicSketch

Alternatives

LinesGauss

Bibliography


Module

Sub-pixel operators

```plaintext
[out] HXLDContX Lines HImageX.LinesGauss ([in] VARIANT Sigma,
[in] String ExtractWidth, [in] String CorrectPositions,
[in] String CompleteJunctions )

void HOperatorSetX.LinesGauss ([in] IObjectX Image,
[out] HUntypedObjectX Lines, [in] VARIANT Sigma, [in] VARIANT Low,
```

Detect lines and their width.

The operator LinesGauss can be used to extract lines (curvilinear structures) from the image Image. The extracted lines are returned in Lines as sub-pixel precise XLD-contours. The parameter LightDark determines, whether bright or dark lines are extracted. If ExtractWidth is set to 'true' the line width is extracted for each line point. If CorrectPositions is set to 'true', LinesGauss compensates the effect of asymmetrical lines (lines having different contrast on each side of the line), and corrects the position and width of the line. This parameter is only meaningful if ExtractWidth='true'. Because the line extractor is unable to extract certain junctions
because of differential geometric reasons, it tries to extract these by different means if CompleteJunctions is set to 'true'.

The extraction is done by using partial derivatives of a Gaussian smoothing kernel to determine the parameters of a quadratic polynomial in x and y for each point of the image. The parameter Sigma determines the amount of smoothing to be performed. Larger values of Sigma lead to a larger smoothing of the image, but can lead to worse localization of the line. Generally, the localization will be much better than that of lines returned by LinesFacet with comparable parameters. The parameters of the polynomial are used to calculate the line direction for each pixel. Pixels which exhibit a local maximum in the second directional derivative perpendicular to the line direction are marked as line points. The line points found in this manner are then linked to contours. This is done by immediately accepting line points that have a second derivative larger than High. Points that have a second derivative smaller than Low are rejected. All other line points are accepted if they are connected to accepted points by a connected path. This is similar to a hysteresis threshold operation with infinite path length (see HysteresisThreshold). However, this function is not used internally since it does not allow the extraction of sub-pixel precise contours.

For the choice of the thresholds High and Low one has to keep in mind that the second directional derivative depends on the amplitude and width of the line as well as the choice of Sigma. The value of the second derivative depends linearly on the amplitude, i.e., the larger the amplitude, the larger the response. For the width of the line there is an approximately inverse exponential dependence: The wider the line is, the smaller the response gets. This holds analogously for the dependence on Sigma: The larger Sigma is chosen, the smaller the second derivative will be. This means that for larger smoothing correspondingly smaller values for High and Low have to be chosen. Two examples help to illustrate this: If 5 pixel wide lines with an amplitude larger than 100 are to be extracted from an image with a smoothing of Sigma = 1.5, High should be chosen larger than 14. If, on the other hand, 10 pixel wide lines with an amplitude larger than 100 and a Sigma = 3 are to be detected, High should be chosen larger than 3.5. For the choice of Low values between 0.25 High and 0.5 High are appropriate.

The extracted lines are returned in a topologically sound data structure in Lines. This means that lines are correctly split at junction points.

LinesGauss defines the following attributes for each line point if ExtractWidth was set to 'false':

- 'angle'  The angle of the direction perpendicular to the line
- 'response' The magnitude of the second derivative

If ExtractWidth was set to 'true' and CorrectPositions to 'false', the following attributes are defined in addition to the above ones:

- 'width_left'  The line width to the left of the line
- 'width_right' The line width to the right of the line

Finally, if CorrectPositions was set to 'true', additionally the following attributes are defined:

- 'asymmetry'  The asymmetry of the line point
- 'contrast'  The contrast of the line point

Here, the asymmetry is positive if the asymmetric part, i.e., the part with the weaker gradient, is on the right side of the line, while it is negative if the asymmetric part is on the left side of the line. All these attributes can be queried via the operator GetContourAttribXld.

Attention

In general, but in particular if the line width is to be extracted, \( \Sigma \geq w/\sqrt{3} \) should be selected, where \( w \) is the width (half the diameter) of the lines in the image. As the lowest allowable value \( \Sigma \geq w/2.5 \) must be selected. If, for example, lines with a width of 4 pixels (diameter 8 pixels) are to be extracted, \( \Sigma \geq 2.3 \) should be selected.

Parameter

- **Image** (input iconic) ............................. image \( \leadsto HImageX / HObjectX \) (byte, int1, int2, int4, real )  
  Input image.
- **Lines** (output iconic) .............................. xld_cont \( \leadsto HXLDContX / HUntypedObjectX \)  
  Extracted lines.
- **Sigma** (input control) ............................... number \( \leadsto \text{VARIANT} \) (real, integer )  
  Amount of Gaussian smoothing to be applied.
  Default Value : 1.5
  Value Suggestions : \( \Sigma \in \{ 1, 1.2, 1.5, 1.8, 2, 2.5, 3, 4, 5 \} \)
  Typical Range of Values : \( 0.7 \leq \Sigma \leq 0.7 \)
  Recommended Value Step : 0.1

3.8. LINES

- **Low** (input control) .......................... number  \( \rightarrow \) VARIANT (real, integer)
  Lower threshold for the hysteresis threshold operation.
  **Default Value**: 3
  **Value Suggestions**: \( \text{Low} \in \{0, 0.5, 1, 2, 3, 4, 5, 8, 10\}\)
  **Typical Range of Values**: \(0 \leq \text{Low} \leq 0\)
  **Recommended Value Step**: 0.5
  **Restriction**: \((\text{Low} \geq 0)\)

- **High** (input control) .......................... number  \( \rightarrow \) VARIANT (real, integer)
  Upper threshold for the hysteresis threshold operation.
  **Default Value**: 8
  **Value Suggestions**: \( \text{High} \in \{0, 0.5, 1, 2, 3, 4, 5, 8, 10, 12, 15, 18, 20, 25\}\)
  **Typical Range of Values**: \(0 \leq \text{High} \leq 0\)
  **Recommended Value Step**: 0.5
  **Restriction**: \((\text{High} \geq 0) \land (\text{High} \geq \text{Low})\)

- **LightDark** (input control) .................. string  \( \rightarrow \) String
  Extract bright or dark lines.
  **Default Value**: 'light'
  **Value List**: LightDark \( \in \{'\text{dark}', '\text{light}'\}\)

- **ExtractWidth** (input control) .............. string  \( \rightarrow \) String
  Should the line width be extracted?
  **Default Value**: 'true'
  **Value List**: ExtractWidth \( \in \{'\text{true}', '\text{false}'\}\)

- **CorrectPositions** (input control) ......... string  \( \rightarrow \) String
  Should the line position and width be corrected?
  **Default Value**: 'true'
  **Value List**: CorrectPositions \( \in \{'\text{true}', '\text{false}'\}\)

- **CompleteJunctions** (input control) ......... string  \( \rightarrow \) String
  Should junctions be added where they cannot be extracted?
  **Default Value**: 'true'
  **Value List**: CompleteJunctions \( \in \{'\text{true}', '\text{false}'\}\)

---

**Example**

```c
/* Detection of lines in an aerial image */
read_image(Image,'mreut4_3')
lines_gauss(Image,Lines,1.5,3,8,'light','true','true','true')
disp_xld(Lines,WindowHandle).
```

---

**Result**

LinesGauss returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behaviour can be set via \(\text{SetSystem}::'\text{noObjectResult}',<\text{Result}>::\). If necessary, an exception handling is raised.

---

**Parallelization Information**

LinesGauss is reentrant and automatically parallelized (on tuple level).

---

**Possible Successor Functions**

GenPolygonsXld

---

**See Also**

BandpassImage, DynThreshold, TopographicSketch

---

**Alternatives**

LinesFacet

---

**Bibliography**


Module

3.9 Match

| void HImageX.AdaptTemplate ([in] HTemplateX TemplateID ) |
| void HTemplateX.AdaptTemplate ([in] HImageX Image ) |
| void HOperatorSetX.AdaptTemplate ([in] IObjectX Image, [in] long TemplateID ) |

Adapting a template to the size of an image.

The operator AdaptTemplate serves to adapt a template which has been created by CreateTemplate to the size of an image. The operator AdaptTemplate can be called before the template is used with images of another size, or if the image used to create the template had another size. If it is not called explicitly it will be called internally each time another image size is used. The contents of the image is hereby irrelevant; only the width of Image will be considered.

| Parameter |
| ▼ Image (input iconic) ......................... image(-array) ~ HImageX / IObjectX (byte) Image which determines the size of the later matching. |
| ▼ TemplateID (input control) ...................... template ~ HTemplateX / long Template number. |

| Result |
| If the parameter values are correct, the operator AdaptTemplate returns the value TRUE. If necessary, an exception handling is raised. |

| Parallelization Information |
| AdaptTemplate is processed under mutual exclusion against itself and without parallelization. |

| Possible Predecessor Functions |
| CreateTemplate, CreateTemplateRot, ReadTemplate |

| Possible Successor Functions |
| SetReferenceTemplate, BestMatch, FastMatch, FastMatchMg, SetOffsetTemplate, BestMatchMg, BestMatchPreMg, BestMatchRot, BestMatchRotMg |

Template matching


Searching the best matching of a template and an image.

The operator BestMatch performs a matching of the template of TemplateID and Image. Hereby the template will be moved over the points of Image so that the template will lie always inside Image. BestMatch
works similar to FastMatch, with the exception, that each time a better match is found the value of MaxError
is internally updated to a lower value to reduce runtime.

With regard to the parameter SubPixel, the position will be indicated by subpixel accuracy. The matching
criterion (“displaced frame difference”) is defined as follows:

\[
\text{error}[\text{row}, \text{col}] = \sum_{u,v} \left| \text{Image}[\text{row} - u, \text{col} - v] - \text{TemplateID}[u, v] \right| \div \text{area(TemplateID)}
\]

The runtime of the operator depends on the size of the domain of Image. Therefore it is important to restrict the
domain as far as possible, i.e. to apply the operator only in a very confined “region of interest”. The parameter
MaxError determines the maximal error which the searched position is allowed to have at most. The lower this
value is, the faster the operator runs.

Row and Column return the position of the best match, whereby Error indicates the average difference of the
grayvalues. If no position with an error below MaxError was found the position (0, 0) and a matching result of
255 for Error are returned. In this case MaxError has to be set larger.

The maximum error of the position (without noise) is 0.1 pixel. The average error is 0.03 pixel.

---

**Parameter**

- **Image** (input iconic) 
  image(-array) \(\sim\) HImageX / HObjectX (byte) 
  Input image inside of which the pattern has to be found.

- **TemplateID** (input control) 
  template \(\sim\) HTemplateX / long 
  Template number.

- **MaxError** (input control) 
  Maximum average difference of the grayvalues.
  Default Value: 20
  Value Suggestions: \(\text{MaxError} \in \{0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 0, 70\}\)
  Typical Range of Values: \(0 \leq \text{MaxError} \leq 0\)
  Minimal Value Step: 1
  Recommended Value Step: 3

- **SubPixel** (input control) 
  Subpixel accuracy in case of ‘true’.
  Default Value: ‘false’
  Value List: SubPixel \(\in\) {‘true’, ‘false’}

- **Row** (output control) 
  Row position of the best match.

- **Column** (output control) 
  Column position of the best match.

- **Error** (output control) 
  Average divergence of the grayvalues of the best match.

---

**Result**

If the parameter values are correct, the operator BestMatch returns the value TRUE. If the input is empty (no
input images are available) the behaviour can be set via SetSystem (‘noObjectResult’, <Result>). If
necessary, an exception handling is raised.

---

**Parallelization Information**

BestMatch is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

CreateTemplate, ReadTemplate, SetOffsetTemplate, SetReferenceTemplate, 
AdaptTemplate, DrawRegion, DrawRectangle1, ReduceDomain

---

**Alternatives**

FastMatch, FastMatchMg, BestMatchMg, BestMatchPreMg, BestMatchRot, BestMatchRotMg, 
ExhaustiveMatch, ExhaustiveMatchMg

---

**Module**

Template matching
WhichLevels
lower. Check also if the illumination has changed (see returned. If the desired object is missed (no object found or wrong position) you have to set
MaxError
/BC The maximum error of the position (without noise) is
Error
in
more stable detection. The value of
NumLevels
Row
and a stable detection. A larger value for
WhichLevels
algorithm starts at the lowest resultion and searches for a position with the lowest matching error. At the next higher
BestMatch
in an operator similar to
BestMatchMg
applies gray value matching using an image pyramid. Such a
Row
Input image inside of which the pattern has to be found.

\begin{verbatim}
[out] double Row HImageX.BestMatchMg ([in] HTemplateX TemplateID, 
\end{verbatim}

\begin{verbatim}
[out] double Row HTemplateX.BestMatchMg ([in] HImageX Image, 
\end{verbatim}

\begin{verbatim}
void HOperatorSetX.BestMatchMg ([in] IObjectX Image, 
[in] long NumLevels, [in] VARIANT WhichLevels, [out] double Row, 
[out] double Column, [out] double Error )
\end{verbatim}

\textit{Suche nach dem besten Grauwert-Match in einer Pyramide.}

\textit{BestMatchMg} applies gray value matching using an image pyramid. \textit{BestMatchMg} works analogously to
\textit{BestMatch}, but it is faster due to the use of a pyramid. Input is an image with an optionally reduced domain. The parameter \textit{MaxError} specifies the maximum error for template matching. Using smaller values results in a reduced runtime but it is possible that the pattern might be missed. The value of \textit{MaxError} has to set larger compared with \textit{BestMatch}, because the error is at higher levels of the pyramid often increased.

\textit{SubPixel} specifies if the result is calculated with sub pixel accuracy or not. A value of \textit{1} for \textit{SubPixel} results in an operator similar to \textit{BestMatch}, i.e. only the original gray values are used. For values larger than \textit{1}, the algorithm starts at the lowest resolution and searches for a position with the lowest matching error. At the next higher resolution this position is refined. This is continued up to the maximum resolution (\textit{WhichLevels} = 'all'). As an alternative Method the mode \textit{WhichLevels} with value 'original' can be used. In this case not only the position with the lowest error but all points below \textit{MaxError} are analysed further on in the next higher resolution. This method is slower but it is more stable and the possibility to miss the correct position is very low. In this case it is often possible to start with a lower resolution (higher level in Pyramid, i.e. larger value for \textit{NumLevels}) which leads to a reduced runtime. Besides the values 'all' and 'original' for \textit{WhichLevels} you can specify the pyramid level explicitly where to switch between a “match all” and the ”best match”. Here \textit{0} corresponds to 'original' and \textit{NumLevels} - \textit{1} is equivalent to 'all'. A value in between is in most cases a good compromise between speed and a stable detection. A larger value for \textit{WhichLevels} results in a reduced runtime, a smaller value results in a more stable detection. The value of \textit{NumLevels} has to equal or smaller than the value used to create the template.

The position of the found matching position is returned in \textit{Row} and \textit{Column}. The corresponding error is given in \textit{Error}. If no point below \textit{MaxError} is found a value of \textit{255} for \textit{Error} and \textit{0} for \textit{Row} and \textit{Column} is returned. If the desired object is missed (no object found or wrong position) you have to set \textit{MaxError} higher or \textit{WhichLevels} lower. Check also if the illumination has changed (see \textit{SetOffsetTemplate}).

The maximum error of the position (without noise) is \textit{0.1} pixel. The average error is \textit{0.03} pixel.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
Parameter & Description \\
\hline
\textbf{Image} & Input image inside of which the pattern has to be found. \\
\hline
\textbf{TemplateID} & Template number. \\
\hline
\textbf{MaxError} & Maximal average difference of the grayvalues. \\
\hline
\textbf{SubPixel} & Exactness in subpixels in case of 'true'. \\
\hline
\textbf{NumLevels} & Number of the used resolution levels. \\
\hline
\end{tabular}
\end{table}

\textit{Image} (input iconic) \hspace{1cm} \textit{image(-array) } \rightsquigarrow \textit{HImageX / IObjectX ( byte )}
Input image inside of which the pattern has to be found.

\textit{TemplateID} (input control) \hspace{1cm} \textit{template } \rightsquigarrow \textit{HTemplateX / long}
Template number.

\textit{MaxError} (input control) \hspace{1cm} \textit{real } \rightsquigarrow \textit{double}
Maximal average difference of the grayvalues.
\textbf{Default Value} : \textit{30}
\textbf{Value Suggestions} : \textit{MaxError} \in \{0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70\}
\textbf{Typical Range of Values} : 0 \leq \textit{MaxError} \leq 0
\textbf{Minimal Value Step} : 1
\textbf{Recommended Value Step} : 3

\textit{SubPixel} (input control) \hspace{1cm} \textit{string } \rightsquigarrow \textit{String}
Exactness in subpixels in case of 'true'.
\textbf{Default Value} : 'false'
\textbf{Value List} : \textit{SubPixel} \in \{'true', 'false'\}

\textit{NumLevels} (input control) \hspace{1cm} \textit{integer } \rightsquigarrow \textit{long}
Number of the used resolution levels.
\textbf{Default Value} : \textit{4}
\textbf{Value List} : \textit{NumLevels} \in \{1, 2, 3, 4, 5, 6\}
3.9. MATCH

- **WhichLevels** (input control) ........................ integer ~ VARIANT (string, integer)
  Resolution level up to which the method “best match” is used.
- **Default Value:** 2
  **Value Suggestions:** WhichLevels ∈ {'all', 'original', 0, 1, 2, 3, 4, 5, 6}
- **Row** (output control) ................................. point.y ~ double
  Row position of the best match.
- **Column** (output control) .............................. point.x ~ double
  Column position of the best match.
- **Error** (output control) ............................... real ~ double
  Average divergence of the grayvalues in the best match.

**Result**

If the parameter values are correct, the operator BestMatchMg returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

**Parallelization Information**

BestMatchMg is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

CreateTemplate, ReadTemplate, AdaptTemplate, DrawRegion, DrawRectangle1, ReduceDomain, SetReferenceTemplate, SetOffsetTemplate

**Alternatives**

FastMatch, FastMatchMg, BestMatch, BestMatchPreMg, BestMatchRot, BestMatchRotMg, ExhaustiveMatch, ExhaustiveMatchMg

**Module**

Template matching

```halcon
[out] double Row HImageX.BestMatchPreMg ([in] HTemplateX TemplateID,  
```

```halcon
[out] double Row HTemplateX.BestMatchPreMg ([in] HImageX ImagePyramid,  
```

```halcon
void HOperatorSetX.BestMatchPreMg ([in] IObjectX ImagePyramid,  
[in] long NumLevels, [in] VARIANT WhichLevels, [out] double Row,  
[out] double Column, [out] double Error )
```

**Suche nach dem besten Grauwert-Match in einer vorherberechneten Pyramide.**

BestMatchPreMg applies gray value matching using an image pyramid. BestMatchPreMg works analogously to BestMatchMg, but it makes use of pre calculated pyramid which has to be generated beforehand using GenGaussPyramid. This reduces runtime if more than one match has to be done or the pyramid has been used otherwise. The pyramid has to be generated using the zooming factor 0.5 and the mode 'constant'.

**Parameter**

- **ImagePyramid** (input iconic) ........................ image ~ HImageX / IObjectX (byte)
  Image pyramid inside of which the pattern has to be found.
- **TemplateID** (input control) ........................... template ~ HTemplateX / long
  Template number.
- **MaxError** (input control) ............................ real ~ double
  Maximal average difference of the grayvalues.
  **Default Value:** 30
  **Value Suggestions:** MaxError ∈ {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70}
  **Typical Range of Values:** 0 ≤ MaxError ≤ 0
  **Minimal Value Step:** 1
  **Recommended Value Step:** 3
The operator \texttt{BestMatchPreMg} returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via \texttt{SetSystem(’noObjectResult’,<Result>).}

- \texttt{SubPixel} (input control) \hfill \texttt{string} \sim \texttt{String}
  - \textbf{Default Value}: ‘false’
  - \textbf{Value List}: \texttt{SubPixel} \in \{’true’,’false’\}

- \texttt{NumLevels} (input control) \hfill \texttt{integer} \sim \texttt{long}
  - \textbf{Default Value}: 3
  - \textbf{Value List}: \texttt{NumLevels} \in \{1, 2, 3, 4, 5, 6\}

- \texttt{WhichLevels} (input control) \hfill \texttt{integer} \sim \texttt{VARIANT}(\texttt{string}, \texttt{integer})
  - \textbf{Resolution level up to which the method “best match” is used.}
  - \textbf{Default Value}: ‘original’
  - \textbf{Value Suggestions}: \texttt{WhichLevels} \in \{’all’,’original’,0, 1, 2, 3, 4, 5, 6\}

- \texttt{Row} (output control) \hfill \texttt{point.y} \sim \texttt{double}
  - \textbf{Row position of the best match.}

- \texttt{Column} (output control) \hfill \texttt{point.x} \sim \texttt{double}
  - \textbf{Column position of the best match.}

- \texttt{Error} (output control) \hfill \texttt{real} \sim \texttt{double}
  - \textbf{Average divergence of the grayvalues in the best match.}

\textbf{Parallelization Information}

\texttt{BestMatchPreMg} is \texttt{reentrant} and automatically \texttt{parallelized} (on tuple level).

\textbf{Possible Predecessor Functions}

\texttt{GenGaussPyramid,CreateTemplate,ReadTemplate,AdaptTemplate,DrawRegion,}
\texttt{DrawRectangle1,ReduceDomain,SetReferenceTemplate}

\textbf{Alternatives}

\texttt{FastMatch,FastMatchMg,ExhaustiveMatch,ExhaustiveMatchMg}

\textbf{Module}

\begin{verbatim}
[out] VARIANT Row HImageX.BestMatchRot ([in] HTemplateX TemplateID,
[in] String SubPixel, [out] VARIANT Column, [out] VARIANT Angle,
[out] VARIANT Error )

[out] VARIANT Row HTemplateX.BestMatchRot ([in] HImageX Image,
[in] String SubPixel, [out] VARIANT Column, [out] VARIANT Angle,
[out] VARIANT Error )

void HOperatorSetX.BestMatchRot ([in] IObjectX Image,
[in] double MaxError, [in] String SubPixel, [out] VARIANT Row,
[out] VARIANT Column, [out] VARIANT Angle, [out] VARIANT Error )
\end{verbatim}

\textbf{Searching the best matching of a template and an image with rotation.}

The operator \texttt{BestMatchRot} performs a matching of the template of \texttt{TemplateID} and \texttt{Image}. It works similar to \texttt{BestMatch} with the extension that the pattern can be rotated. The parameters \texttt{AngleStart} and \texttt{AngleExtend} define the maximum rotation of the pattern: \texttt{AngleStart} specifies the maximum counter clockwise rotation and \texttt{AngleExtend} the maximum clockwise rotation relative to this angle. Both values have to smaller or equal to the values used for the creation of the pattern (see \texttt{CreateTemplateRot}). In addition to \texttt{BestMatch} \texttt{BestMatchRot} returns the rotation angle of the pattern in \texttt{Angle} (radian). The accuracy of this angle depends on the parameter \texttt{AngleStep} of \texttt{CreateTemplateRot}. In the case of \texttt{SubPixel} = ’true’ the position and the angle are calculated with “sub pixel” accuracy.
3.9. MATCH

Parameter

- **Image** (input iconic) .............................................. image(-array) \(\sim\) HImageX / IObjectX (byte)
  Input image inside of which the pattern has to be found.

- **TemplateID** (input control) ................................ template \(\sim\) HTemplateX / long
  Template number.

- **AngleStart** (input control) .............................. angle.rad \(\sim\) double
  Smallest Rotation auf the pattern.
  **Default Value**: -0.39
  **Value Suggestions**: \(\text{AngleStart} \in \{-3.14, -1.57, -0.79, -0.39, -0.20, 0.0\}\)

- **AngleExtend** (input control) .............................. angle.rad \(\sim\) double
  Maximum positive Extension of AngleStart.
  **Default Value**: 0.79
  **Value Suggestions**: \(\text{AngleExtend} \in \{6.28, 3.14, 1.57, 0.79, 0.39\}\)
  **Restriction**: \((\text{AngleExtend} > 0)\)

- **MaxError** (input control) .................................................. real \(\sim\) double
  Maximum average difference of the grayvalues.
  **Default Value**: 30
  **Value Suggestions**: \(\text{MaxError} \in \{0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70\}\)
  **Typical Range of Values**: \(0 \leq \text{MaxError} \leq 0\)
  **Minimal Value Step**: 1
  **Recommended Value Step**: 3

- **SubPixel** (input control) ............................................. string \(\sim\) String
  Subpixel accuracy in case of 'true'.
  **Default Value**: 'false'
  **Value List**: SubPixel \(\in \{\text{true}, \text{false}\}\)

- **Row** (output control) ................................................. point.y(-array) \(\sim\) VARIANT (real)
  Row position of the best match.

- **Column** (output control) ......................................... point.x(-array) \(\sim\) VARIANT (real)
  Column position of the best match.

- **Angle** (output control) ........................................... angle.rad(-array) \(\sim\) VARIANT (real)
  Rotation angle of pattern.

- **Error** (output control) .............................................. real(-array) \(\sim\) VARIANT (real)
  Average divergence of the grayvalues of the best match.

**Result**

If the parameter values are correct, the operator BestMatchRot returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

**Parallelization Information**

BestMatchRot is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

CreateTemplateRot, ReadTemplate, SetOffsetTemplate, SetReferenceTemplate, AdaptTemplate, DrawRegion, DrawRectangle1, ReduceDomain

**See Also**

BestMatch, BestMatchMg

**Alternatives**

BestMatchRotMg

**Module**

Template matching
The value of MaxError often increased.

The operator BestMatchRotMg performs a matching of the template of TemplateID and Image. It works similar to BestMatchMg with the extension that the pattern can be rotated analogously to BestMatchRot.

The parameters AngleStart and AngleExtend define the maximum rotation of the pattern: AngleStart specifies the maximum counter clockwise rotation and AngleExtend the maximum clockwise rotation relative to this angle. Both values have to be smaller or equal to the values used for the creation of the pattern (see CreateTemplateRot). In addition to BestMatchMg BestMatchRotMg returns the rotation angle of the pattern in Angle (radiant).

The value of MaxError has to be set larger compared with BestMatchRot, because the error is at higher levels of the pyramid often increased.

In the case of SubPixel = 'true' the position and the angle are calculated with “sub pixel” accuracy.

The value of NumLevels has to be equal or smaller than the value used to create the template.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image (input iconic)</td>
<td>Image(-array)</td>
</tr>
<tr>
<td>TemplateID (input control)</td>
<td>Template  ~ HTempalteX / IObjectX (byte)</td>
</tr>
<tr>
<td>AngleStart (input control)</td>
<td>Angle.ro ~ double</td>
</tr>
<tr>
<td>MaxError (input control)</td>
<td>MaxError ~ real</td>
</tr>
<tr>
<td>SubPixel (input control)</td>
<td>SubPixel accuracy in case of 'true'</td>
</tr>
<tr>
<td>NumLevels (input control)</td>
<td>Number of the used resolution levels</td>
</tr>
</tbody>
</table>

Searching the best matching of a template and a pyramid with rotation.

void HOperatorSetX.BestMatchRotMg ([in] IObjectX Image,
[out] VARIANT Row, [out] VARIANT Column, [out] VARIANT Angle,
[out] VARIANT Error )

[out] VARIANT Row HImageX.BestMatchRotMg ([in] HTempalteX TemplateID,
[in] String SubPixel, [in] long NumLevels, [out] VARIANT Column,
[out] VARIANT Angle, [out] VARIANT Error )
3.9. MATCH

- Row (output control) .............................................. point.y(-array) \rightarrow \text{VARIANT}(\text{real})
  Row position of the best match.
- Column (output control) .............................................. point.x(-array) \rightarrow \text{VARIANT}(\text{real})
  Column position of the best match.
- Angle (output control) .......................................... angle.rad(-array) \rightarrow \text{VARIANT}(\text{real})
  Rotation angle of pattern.
- Error (output control) ............................................... real(-array) \rightarrow \text{VARIANT}(\text{real})
  Average divergence of the gray values of the best match.

Result

If the parameter values are correct, the operator \text{BestMatchRotMg} returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via \text{SetSystem('noObjectResult',<Result>)}. If necessary, an exception handling is raised.

--- Parallelization Information ---

\text{BestMatchRotMg} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

--- Possible Predecessor Functions ---

\text{CreateTemplateRot}, \text{SetReferenceTemplate}, \text{SetOffsetTemplate}, \text{AdaptTemplate}, \text{DrawRegion}, \text{DrawRectangle}, \text{ReduceDomain}

--- See Also ---

\text{FastMatch}

--- Alternatives ---

\text{BestMatchRot}, \text{BestMatchMg}

--- Module ---

Template matching

---

```
void \text{HOperatorSetX.ClearTemplate} ([\text{in}] \text{long TemplateID})
```

Deallocation of the memory of a template.

The operator ClearTemplate deallocates the memory of a template which has been created by \text{CreateTemplate} or \text{CreateTemplateRot}. After execution of the operator ClearTemplate the template can no longer be used. The value of TemplateID is not valid. However, the number can be used again by further calls of \text{CreateTemplate} or \text{CreateTemplateRot}.

--- Parameter ---

- TemplateID (input control) .............................................. template \rightarrow \text{HTemplateX}/\text{long}
  Template number.

--- Result ---

If the number of the template is valid, the operator ClearTemplate returns the value TRUE. If necessary an exception handling will be raised.

--- Parallelization Information ---

ClearTemplate is \textit{local} and processed \textit{completely exclusively} without parallelization.

--- Possible Predecessor Functions ---

\text{CreateTemplate,CreateTemplateRot,ReadTemplate,WriteTemplate}

--- Module ---

Template matching

---

```
[out] \text{HImageX ImageCorner} \text{HImageX.CornerResponse} ([\text{in}] \text{long Size,}
\text{[in]} \text{double Weight})
```

Searching corners in images.

---

```
void \text{HOperatorSetX.CornerResponse} ([\text{in}] \text{IHObjectX Image,}
\text{[out]} \text{HUntypedObjectX ImageCorner, [in]} \text{long Size, [in]} \text{double Weight})
```

Searching corners in images.
The operator `CornerResponse` helps to extract grayvalue corners in an image. The formula for the calculation is as follows:

\[
R(x, y) = A(x, y) \cdot B(x, y) - C^2(x, y) - Weight \cdot (A(x, y) + B(x, y))^2 \\
A(x, y) = W(u, v) \cdot (\nabla_x I(x, y))^2 \\
B(x, y) = W(u, v) \cdot (\nabla_y I(x, y))^2 \\
C(c, y) = W(u, v) \cdot (\nabla_x I(x, y)\nabla_y I(x, y))
\]

whereby \(I\) is the input image and \(R\) the output image of the filtering. The operator `GaussImage` is used for smoothing \(W\), the operator `SobelAmp` is used for calculating the derivative \(\nabla\).

The corner response function is invariant with regard to rotation. In order to achieve a suitable dependency of the function \(R(x, y)\) of the local gradient, the parameter `Weight` has to be set to 0.04. Thereby only grayvalue corners will return positive values for \(R(x, y)\), while straight edges will receive negative values (due to clipping they are set to 0).

Parameter

\[\text{Parameter}\]

- **Image** (input iconic) \(\ldots\) (multichannel-)image-array \(\leadsto\) \(HImageX / HObjectX\) (byte)
  - Input image.
- **ImageCorner** (output iconic) \(\ldots\) multichannel-image-array \(\leadsto\) \(HImageX / HUntypedObjectX\) (byte)
  - Result of the filtering.
  - **Parameter Number**: \(\text{ImageCorner} = \text{Image}\)
- **Size** (input control) \(\ldots\) \(\ldots\) \(\text{integer} \leadsto\) \(\text{long}\)
  - Desired filtersize of the graymask.
  - **Default Value**: 3
  - **Value List**: \(\text{Size} \in \{3, 5, 7, 9, 11\}\)
- **Weight** (input control) \(\ldots\) \(\ldots\) \(\text{real} \leadsto\) \(\text{double}\)
  - Weighting.
  - **Default Value**: 0.04
  - **Typical Range of Values**: \(0.0 \leq \text{Weight} \leq 0.0\)
  - **Minimal Value Step**: 0.001
  - **Recommended Value Step**: 0.01

Parallelization Information

`CornerResponse` is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successor Functions

- **LocalMax**, **Threshold**

See Also

- `GaussImage`, `SobelAmp`

Bibliography


Image filters
Preparing a pattern for template matching.

The operator \texttt{CreateTemplate} preprocesses a pattern (\texttt{Template}), which is passed as an image, for the template matching. After the transformation, a number (\texttt{TemplateID}) is assigned to the template for being used in the further process. The shape and the size of \texttt{Template} can be chosen arbitrarily. You have to be aware, that the matching is only applied to that part of an image where \texttt{Template} fits completely into the image.

The template has been chosen such that it contains no pixels of the (changing) background. Here you can make use of the arbitrary shape of a template which is not restricted to a rectangle. To create a template you can use segmentation operators like \texttt{Threshold}. In the case of sub pixel accurate matching \texttt{Template} has in addition to be one pixel smaller than the pattern (i.e. one pixel border to the changing background). This can be done e.g. by applying the operator \texttt{ErosionCircle}.

The parameter \texttt{NumLevel} specifies the number of pyramid levels (\texttt{NumLevel} = 1 means only original gray values) which can be used for matching. The number of levels used later for matching will be below or equal this value. If the pattern becomes to small due to zooming, the maximum number of pyramid levels is automatically reduced (without error message).

The parameter \texttt{GrayValues} defines, whether the original gray values (\texttt{original}, \texttt{normalized}) or the edge amplitude (\texttt{gradient}, \texttt{sobel}) is used. With \texttt{original} the sum of the differences is used as feature which is very stable and fast if there is no change in illumination. \texttt{normalized} is used if the illumination changes. The method is a bit slower and not quite as stable. If there is no change in illumination the mode \texttt{original} should be used. The edge amplitude is another method to be invariant to changes in illumination. The disadvantage is the increased execution time and the higher sensitivity to changes in the shape of the pattern. The mode \texttt{gradient} is slightly faster but more sensitive to noise.

The maximum error for matching has typically to be chosen higher when using the edge amplitude. The mode chosen by \texttt{GrayValues} leads automatically to calling the appropriate filter during matching — if necessary.

As an alternative to the gradient approach the operator \texttt{SetOffsetTemplate} can be used, if the change in illumination is known.

The parameter \texttt{Optimize} specifies if the pattern has to optimized for runtime. This optimization results in a longer time to create the template but reduces the time for matching. In addition the optimization leads to a more stable matching, i.e., the possibility to miss good matches is reduced. The optimization process selects the most stable and significant gray values to be tested first during the matching process. Using this technique a wrong match can be eliminated very early.

The reference position for the template is its center of gravity. I.e. if you apply the template to the original image the center of gravity is returned. This default reference can be adapted using the operator \texttt{SetReferenceTemplate}.

In sub pixel mode a special position correction is calculated which is added after each matching: The template is applied to the original image and the difference between the found position and the center of gravity is used as a correction vector. This is important for patterns in a textured context or for asymmetric pattern. For most templates this correction vector is near null.

If the pattern is no longer used, it has to be freed by the operator \texttt{ClearTemplate} in order to deallocate the memory.

Before the use of the template, which is stored independently of the image size, it can be adapted explicitly to the size of a definite image size by using \texttt{AdaptTemplate}.
CHAPTER 3. FILTER

Parameter

- **Template** (input iconic) .............................................. image \( \sim \) HImageX / IHObjectX (byte)
  - Input image whose domain will be processed for the pattern matching.

- **FirstError** (input control) ........................................... integer \( \sim \) long
  - Not in use.
  - Default Value: 255
  - Value List: FirstError \( \in \{255\}\)

- **NumLevel** (input control) .......................................... integer \( \sim \) long
  - Maximal number of pyramid levels.
  - Default Value: 4
  - Value List: NumLevel \( \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}\)

- **Optimize** (input control) .......................................... string \( \sim \) String
  - Kind of optimizing.
  - Default Value: 'sort'
  - Value List: Optimize \( \in \{\text{"none"}, \text{"sort"}\}\)

- **GrayValues** (input control) ...................................... string \( \sim \) String
  - Kind of grayvalues.
  - Default Value: 'original'
  - Value List: GrayValues \( \in \{\text{"original"}, \text{"normalized"}, \text{"gradient"}, \text{"sobel"}\}\)

- **TemplateID** (output control) .................................. template \( \sim \) HTemplateX / long
  - Template number.

Result

If the parameters are valid, the operator **CreateTemplate** returns the value TRUE. If necessary an exception handling will be raised.

Parallelization Information

CreateTemplate is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

- DrawRegion, ReduceDomain, Threshold

Possible Successor Functions

- AdaptTemplate, SetReferenceTemplate, ClearTemplate, WriteTemplate,
  - SetOffsetTemplate, BestMatch, BestMatchMg, FastMatch, FastMatchMg

Alternatives

- CreateTemplateRot, ReadTemplate

Module

Template matching

```c
[out] HTemplateX TemplateID
HImageX.CreateTemplateRot
[in] long NumLevel, [in] double AngleStart, [in] double AngleExtend,
```

```c
void HTemplateX.CreateTemplateRot ([in] HImageX Template,
[in] long NumLevel, [in] double AngleStart, [in] double AngleExtend,
```

```c
void HOperatorSetX.CreateTemplateRot ([in] IHObjectX Template,
[in] long NumLevel, [in] double AngleStart, [in] double AngleExtend,
[out] long TemplateID )
```

Preparing a pattern for template matching with rotation.

The operator **CreateTemplateRot** preprocesses a pattern, which is passed as an image, for the template matching. An extension to **CreateTemplate** the matching can applied to rotated patterns. The parameters **AngleStart** and **AngleExtend** define the maximum rotation of the pattern: **AngleStart** specifies the maximum counter clockwise rotation and **AngleExtend** the maximum clockwise rotation relative to this angle. Therefore **AngleExtend** has to be smaller than \(2\pi\). With the parameter **AngleStep** the maximum angle resolution (on the highest resolution level) can be specified.
You have to be aware, that all possible rotations are calculated beforehand to reduce runtime during matching. This leads to a higher execution time for \texttt{CreateTemplateRot} and high memory requirements for the template. The amount of memory depends on the parameters \texttt{AngleExtend} and \texttt{AngleStep}. The number of pyramid levels can be neglected. If $A$ is the number of pixels of \texttt{Template}, the memory $M$ needed for the template in byte is about:

$$M = \frac{A \times 12 \times \text{AngleExtend}}{\text{AngleStep}}$$

After the transformation, a number (\texttt{TemplateID}) is assigned to the template for being used in the further process.

A description of the other parameters can be found at the operator \texttt{CreateTemplate}.

\begin{verbatim}
<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{Template} (input iconic) ........................................... image \sim HI mageX / IO bjectX (byte)</td>
</tr>
<tr>
<td>\textbf{NumLevel} (input control) ........................................... integer \sim long</td>
</tr>
<tr>
<td>\textbf{AngleStart} (input control) ........................................... angle.rad \sim double</td>
</tr>
<tr>
<td>\textbf{AngleExtend} (input control) ........................................... angle.rad \sim double</td>
</tr>
<tr>
<td>\textbf{AngleStep} (input control) ........................................... angle.rad \sim double</td>
</tr>
<tr>
<td>\textbf{Optimize} (input control) ........................................... string \sim String</td>
</tr>
<tr>
<td>\textbf{GrayValues} (input control) ........................................... string \sim String</td>
</tr>
<tr>
<td>\textbf{TemplateID} (output control) ........................................... template \sim HTemplateX / long</td>
</tr>
</tbody>
</table>
\end{verbatim}

If the parameters are valid, the operator \texttt{CreateTemplateRot} returns the value TRUE. If necessary an exception handling will be raised.

\begin{verbatim}
<table>
<thead>
<tr>
<th>Parallelization Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateTemplateRot is local and processed completely exclusively without parallelization.</td>
</tr>
</tbody>
</table>
\end{verbatim}

\begin{verbatim}
<table>
<thead>
<tr>
<th>Possible Predecessor Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrawRegion, ReduceDomain, Threshold</td>
</tr>
</tbody>
</table>
\end{verbatim}

\begin{verbatim}
<table>
<thead>
<tr>
<th>Possible Successor Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>BestMatchRot, BestMatchRotMg, AdaptTemplate, SetReferenceTemplate, ClearTemplate, SetOffsetTemplate, WriteTemplate</td>
</tr>
</tbody>
</table>
\end{verbatim}
Calculating the average "displaced frame difference":

\[
\text{AREA}(X) = \sum_{u,v} (\text{Image}[i-u][j-v] \cdot \text{ImageTemplate}[l-u][c-v])
\]

The terms are the same as in 'norm_correlation'. AREA (X) means the area of the region X. Range of value 0 (best fit) - 255.

To calculate the normalized correlation as well as the "displaced frame difference" is with regard to the input region (RegionOfInterest) very time consuming. Therefore it is important to restrict the input region (RegionOfInterest) if possible, i.e. to apply the filter only in a very confined "region of interest".

As far as quality is concerned, both modes return comparable results, whereby the mode 'dfd' is faster by a factor of about 3.5.

Module

Alternatives

CreateTemplate

Template matching

HImageX ImageMatch HImageX.ExhaustiveMatch
([in] HRegionX RegionOfInterest, [in] HImageX ImageTemplate,
[in] String Mode

void HOperatorSetX.ExhaustiveMatch ([in] HObjectX Image,
[in] HObjectX RegionOfInterest, [in] HObjectX ImageTemplate,
[out] HUntypedObjectX ImageMatch, [in] String Mode )

Matching of a template and an image.

The operator ExhaustiveMatch matches ImageTemplate and Image within the region of interest RegionOfInterest. Hereby the ImageTemplate will be moved over all points of Image which lie within the RegionOfInterest. With regard to the parameter Mode, a matching criterion will be calculated. The result values will be stored in ImageMatch.

The following matching criteria (Mode) are available:

'norm_correlation'

ImageMatch[i][j] = 255 \cdot \frac{\sum_{u,v} (\text{Image}[i-u][j-v] \cdot \text{ImageTemplate}[l-u][c-v])}{\sqrt{\sum_{u,v} (\text{Image}[i-u][j-v]^2) \cdot \sum_{u,v} (\text{ImageTemplate}[l-u][c-v]^2)}}

whereby X[i][j] indicates the grayvalue in the ith column and jth row of the image X. (l,c) is the center of the region of ImageTemplate. u and v are chosen so that all points of the template will be reached, i,j run across the RegionOfInterest. At the image frame only those parts of ImageTemplate will be considered which lie inside the image (i.e. u and v will be restricted correspondingly). Range of values: 0 - 255 (best fit).

'dfd' Calculating the average "displaced frame difference":

ImageMatch[i][j] = \frac{\sum_{u,v} |\text{Image}[i-u][j-v] - \text{ImageTemplate}[l-u][c-v]|}{\text{AREA(ImageTemplate)}}

The terms are the same as in 'norm_correlation'. AREA (X) means the area of the region X. Range of value 0 (best fit) - 255.

To calculate the normalized correlation as well as the "displaced frame difference" is with regard to the area of ImageTemplate very time consuming. Therefore it is important to restrict the input region (RegionOfInterest) if possible, i.e. to apply the filter only in a very confined "region of interest".

As far as quality is concerned, both modes return comparable results, whereby the mode 'dfd' is faster by a factor of about 3.5.

Parameter

- **Image** (input iconic) .......................... image $\sim$ HImageX / HObjectX (byte)  
  Input image.

- **RegionOfInterest** (input iconic) .............. region $\sim$ HRegionX / HObjectX  
  Area to be searched in the input image.

- **ImageTemplate** (input iconic) ................. image $\sim$ HImageX / HObjectX (byte)  
  This area will be "matched" by Image within the RegionOfInterest.

- **ImageMatch** (output iconic) ...................... image $\sim$ HImageX / HUntypedObjectX  
  Result image: values of the matching criterion.

- **Mode** (input control) .......................... string $\sim$ String  
  Desired matching criterion.

  **Default Value** : 'dfd'

  **Value List** : Mode $\in \{ \text{‘norm_correlation’}, \text{‘dfd’} \} $
3.9. MATCH

Example

read_image(Image,'monkey')
disp_image(Image,WindowHandle)
draw_rectangle2(WindowHandle,Row,Column,Phi,Length1,Length2)
gen_rectangle2(Rectangle,Row,Column,Phi,Length1,Length2)
reduce_domain(Image,Rectangle,Template)
exhaustive_match(Image,Image,Template,ImageMatch,'dfd')
invert_image(ImageMatch,ImageInvert)
local_max(Image,Maxima)
union1(Maxima,AllMaxima)
add_channels(AllMaxima,ImageInvert,FitMaxima)
threshold(FitMaxima,BestFit,230.0,255.0)
disp_region(BestFit,WindowHandle).

Result
If the parameter values are correct, the operator ExhaustiveMatch returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information
ExhaustiveMatch is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Possible Successor Functions

Alternatives

ExhaustiveMatchMg

Module

Image filters

Matching a template and an image in a resolution pyramid.

The operator ExhaustiveMatchMg is an additional option for the operator ExhaustiveMatch performing a matching of the image Image and the template ImageTemplate. Hereby ImageTemplate will be moved over all points of the region of Image, a matching criterion will be calculated with regard to the parameter Mode and the result values will be stored in ImageMatch.

Of images having been filtered this way, normally only those areas with good matching results are of interest. The size of the area to be searched, i.e. the region of the input image Image, determines decisively the runtime of the matching filter. Therefore it is recommendable to use at first ExhaustiveMatchMg with reduced image resolution in order to determine a “region of interest” in which good matching results can be expected; then in this restricted area only the real matching (see also ExhaustiveMatch) will be executed with normal resolution.

Hereby the Gauss-pyramids of Image and ImageTemplate will be composed (in particular the corresponding regions will be transformed as well). Then on each level of the resolution pyramids - starting with the startlevel Level - the matching inside the current “region of interest” will be executed. Whereby the “region of interest” on the startlevel is equivalent to the region of the input image Image. After the filtering, a new “region of interest” is determined with the help of a threshold operation and will be transformed on the next resolution level.
Threshold(0, Threshold), if Mode = 'dfd'
Threshold(Threshold, 255), if Mode = 'crosscorrelation'

The final matching in the determined “region of interest” will then be calculated with the highest resolution (Level 0). The output image ImageMatch includes the corresponding filter result and the final “region of interest”, which is determined on the result image with the help of a threshold operation.

The operator ExhaustiveMatchMg therefore is not simply a filter, but can also be considered as a member of the class of region transformations.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Input image.</td>
</tr>
<tr>
<td>ImageTemplate</td>
<td>The domain of this image will be matched with Image.</td>
</tr>
<tr>
<td>ImageMatch</td>
<td>Result image and result region: values of the matching criterion within the determined “region of interest”.</td>
</tr>
<tr>
<td>Mode</td>
<td>Desired matching criterion.</td>
</tr>
<tr>
<td>Value List</td>
<td>Mode ∈ {‘crosscorrelation’, ‘dfd’}</td>
</tr>
<tr>
<td>Level</td>
<td>Start level in the resolution pyramid (highest resolution: Level 0).</td>
</tr>
<tr>
<td>Value List</td>
<td>Level ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8}</td>
</tr>
<tr>
<td>Restriction</td>
<td>(Image ∧ (height &lt; ld))</td>
</tr>
<tr>
<td>Threshold</td>
<td>Threshold to determine the “region of interest”.</td>
</tr>
<tr>
<td>Value Suggestionsth</td>
<td>Threshold ∈ {5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190, 195, 200, 205, 210, 215, 220, 225, 230, 235, 240, 245, 250}</td>
</tr>
<tr>
<td>Typical Range of Values</td>
<td>0 ≤ Threshold ≤ 0</td>
</tr>
<tr>
<td>Minimal Value Step</td>
<td>1</td>
</tr>
<tr>
<td>Recommended Value Step</td>
<td>5</td>
</tr>
</tbody>
</table>

If the parameter values are correct, the operator ExhaustiveMatchMg returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

ExhaustiveMatchMg is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions:
- DrawRegion
- DrawRectangle

Possible Successor Functions:
- Threshold
- LocalMax

See Also:
- GenGaussPyramid

Alternatives:
- ExhaustiveMatch

Module:
- Image filters

3.9. MATCH

Searching all good matches of a template and an image.

The operator FastMatch performs a matching of the template of TemplateID and Image. Hereby the template will be moved over the points of Image so that the template always lies completely inside of Image. The matching criterion ("displaced frame difference") is defined as follows:

$$error[r, c] = \frac{\sum_{u,v} |Image[r, c] - TemplateID[u, v]|}{area(TemplateID)}$$

The difference between FastMatch and ExhaustiveMatch is that the matching for one position is stopped if the error is too high. This leads to a reduced runtime but one might miss correct matches. The runtime of the operator depends mainly on the size of the domain of Image. Therefore it is important to restrict the domain as far as possible, i.e. to apply the operator only in a very confined "region of interest". The parameter MaxError determines the maximal error which the searched position is allowed to show. The lower this value is, the faster the operator runs.

All points which show a matching error smaller than MaxError will be returned in the output region Matches. This region can be used for further processing. For example by using a connection and BestMatch to find all the matching objects. If no point has a match error below MaxError the empty region (i.e no points) is returned.

**Parameter**

- **Image** (input iconic) ............................................. image(-array) \( \sim \) HImageX / IHObjectX (byte)
  Input image inside of which the pattern has to be found.

- **Matches** (output iconic) ........................................... region(-array) \( \sim \) HRegionX / HUntypedObjectX
  All points whose error lies below a certain threshold.

- **TemplateID** (input control) ..................................... template \( \sim \) HTemplateX / long
  Template number.

- **MaxError** (input control) ........................................... real \( \sim \) double
  Maximal average difference of the grayvalues.

  **Default Value** : 20
  **Value Suggestions** : MaxError \( \in \{0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30\}$
  **Typical Range of Values** : 0 \( \leq \) MaxError \( \leq \) 0
  **Minimal Value Step** : 1
  **Recommended Value Step** : 1

**Result**

If the parameter values are correct, the operator FastMatch returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

**Parallelization Information**

FastMatch is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

CreateTemplate, ReadTemplate, AdaptTemplate, DrawRegion, DrawRectangle1, ReduceDomain

**Possible Successor Functions**

Connection, BestMatch

**Alternatives**

BestMatch, BestMatchMg, FastMatchMg, ExhaustiveMatch, ExhaustiveMatchMg

**Module**

Template matching
Searching all good grayvalue matches in a pyramid.

The operator FastMatchMg like the operator FastMatch performs a matching of the template of TemplateID and Image. In contrast to FastMatch, however, the search for good matches starts in scaled down images (pyramid). The number of levels of the pyramid will be determined by NumLevel. Hereby the value 1 indicates that no pyramid will be used. In this case the operator FastMatchMg is equivalent to the operator FastMatch. The value 2 triggers the search in the image with half the framesize. The search for all those points showing an error small enough in the scaled down image (error smaller than MaxError) will be refined at the corresponding positions in the original image (Image).

The runtime of matching dependends on the parameter MaxError: the larger the value the longer is the processing time, because more points of the pattern have to be tested. If MaxError is to low the pattern will not be found. The value has therefore to be optimized for every application.

NumLevel indicates the number of levels of the pyramid, including the original image. Optionally a second value can be given. This value specifies the number (0..n) of the lowest level which is used the the matching. The region found up to this level will then be zoomed to the size of the original level. This can used to increase the runtime in the case that the accuracy does not have to be so high.

Parameter

- **Image** (input iconic) ........................................ image(-array) ~ HImageX / IHObjectX (byte)
  Input image inside of which the pattern has to be found.
- **Matches** (output iconic) ................................. region(-array) ~ HRegionX / HUntypedObjectX
  All points which have an error below a certain threshold.
- **TemplateID** (input control) ................................. template ~ HTemplateX / long
  Template number.
- **MaxError** (input control) ........................................ real ~ double
  Maximal average difference of the grayvalues.
  Default Value : 30
  Value Suggestions : MaxError ∈ {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70}
  Typical Range of Values : 0 ≤ MaxError ≤ 0
  Minimal Value Step : 1
  Recommended Value Step : 3
- **NumLevel** (input control) ........................................ integer(-array) ~ VARIANT (integer)
  Number of levels in the pyramid.
  Default Value : 3
  Value List : NumLevel ∈ {1, 2, 3, 4, 5, 6, 7, 8}

Result

If the parameter values are correct, the operator FastMatchMg returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

FastMatchMg is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

CreateTemplate, ReadTemplate, AdaptTemplate, DrawRegion, DrawRectangle1, ReduceDomain

Alternatives

BestMatch, BestMatchMg, FastMatch, ExhaustiveMatch, ExhaustiveMatchMg

Module

Template matching

Expansion of a displacement vector field onto undefined areas.

The operator `FillDvf` calculates a value for each undefined point in a displacement vector field by averaging its neighbors. The dimension of the neighborhood will be determined by the parameters `Width` and `Height`. As the gaps may be larger than the masks, the filter may be used iteratively (`Iterations`). Correctly defined values in the input image will not be modified.

---

**Parameter**

- **Image** (input iconic) ........................................... image(-array) \(\sim\) `HImageX / IHOBJECTX (dvf)`
  - Input image.

- **ImageExpanded** (output iconic) ................................. image(-array) \(\sim\) `HImageX / HUNTYPEDOBJECTX (dvf)`
  - Result of adding the points which are missing.

- **Width** (input control) ...........................................\(\sim\) long
  - Width of the filter mask.
  - **Default Value**: 5
  - **Value Suggestions**: `Width \in \{3, 5, 7, 9, 11, 15\}`
  - **Typical Range of Values**: `3 \leq Width \leq 3`
  - **Minimal Value Step**: 2
  - **Recommended Value Step**: 2

- **Height** (input control) .........................................\(\sim\) long
  - Height of the filter mask.
  - **Default Value**: 5
  - **Value Suggestions**: `Height \in \{3, 5, 7, 9, 11, 13, 15\}`
  - **Typical Range of Values**: `3 \leq Height \leq 3`
  - **Minimal Value Step**: 2
  - **Recommended Value Step**: 2

- **Iterations** (input control) ...................................\(\sim\) long
  - Number of iterations.
  - **Default Value**: 3
  - **Value Suggestions**: `Iterations \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}`
  - **Typical Range of Values**: `1 \leq Iterations \leq 1`
  - **Minimal Value Step**: 1
  - **Recommended Value Step**: 1

---

**Parallelization Information**

`FillDvf` is reentrant and automatically parallelized (on tuple level, domain level).

---

**Possible Predecessor Functions**

**Module**

Image filters

---

**Calculating a Gauss pyramid.**

The operator `GenGaussPyramid` calculates a pyramid of scaled down images. The scale by which the next image will be reduced is determined by the parameter `Scale`. For instance, a value of 0.5 for `Scale` will shorten the edge length of `Image` by 50%. This is exactly equivalent to the "normal" pyramid.
The parameter `Mode` determines the way of averaging. For a more detailed description concerning this parameter see also `AffineTransImage`. In the case that `Scale` is equal 0.5 there are the additional modes `min` and `max` available. In this case the minimum or the maximum of the four neighboring pixels is selected.

Please note that each level will be returned as an individual image, i.e. as one iconic object, with one matrix and its own domain. If a multichannel image is needed as a result, the operator `ImageToChannels` has to be used. A single level or more than one level can be selected by using `SelectObj` respectively `CopyObj`.

### Calculating the monotony operation.

The operator `Monotony` calculates the monotony operator. Thereby the points which are strictly smaller than the current grayvalue will be counted in the 8 neighborhood. This number will be entered into the output imaged. If there is a strict maximum, the value 8 is returned; in case of a minimum or a plateau, the value 0 will be returned. A ridge or a slope will return the corresponding intermediate values.

The monotony operator is often used to prepare matching operations as it is invariant with regard to lightness modifications.

```plaintext
[out] HImageX ImageMonotony HImageX.Monotony ( )

```

Calculating the monotony operation.

The operator `Monotony` calculates the monotony operator. Thereby the points which are strictly smaller than the current grayvalue will be counted in the 8 neighborhood. This number will be entered into the output imaged. If there is a strict maximum, the value 8 is returned; in case of a minimum or a plateau, the value 0 will be returned. A ridge or a slope will return the corresponding intermediate values.

The monotony operator is often used to prepare matching operations as it is invariant with regard to lightness modifications.
### Parallelization Information

Monotony is reentrant and automatically parallelized (on tuple level, channel level, domain level).

### Possible Predecessor Functions

GaussImage, MedianImage, MeanImage, SmoothImage, InvertImage

### Possible Successor Functions

Threshold, ExhaustiveMatch, DispImage

### Alternatives

LocalMax, TopographicSketch, CornerResponse

### Module

Image filters

---

Calculating the Displacement vector field by correlation methods.

The operator `OpticalFlowMatch` determines a displacement vector field (DVF) with the help of correlation methods. Simple- and multichannel Images may be used. A window of the size `SizeWindow` will be moved inside a mask of the size `SizeSearch` over a second image and there the grayvalues will be compared with the ones from the first image. The correlation showing the slightest error will determine the vector. In case of two vectors having the same weighting, the shorter vector will be preferred. After having processed one image point, the next image point at a distance of `Step` will be selected. It is therefore not guaranteed to get a dense displacement vector field.

The parameter `Threshold` specifies the maximal divergence of previously estimated correlations and the best estimation of a displacement vector.

The method used by the operator is determined by the parameter `Mode`. The following values for `Mode` can be selected:

- `'dfd'` Calculation of displaced frame difference.
- `'dfd_norm'` Calculation of displaced frame difference normalized by the mean value of the matching windows.

If the image has more than one channel the parameter `Weights` indicates the weighting of the individual channels. For each channel one weight factor must be passed.

For the parameters `SizeWindow`, `SizeSearch` and `Step` optionally two values can be passed: The first value indicates hereby the width (respectively the row), the second value the height (respectively the column).

---

```plaintext
[out] HImageX VectorField HImageX.OpticalFlowMatch
([in] HImageX Image2, [in] String Mode, [in] long Threshold,
[in] VARIANT Weights )

void HOperatorSetX.OpticalFlowMatch ([in] IHObjectX Image1,
[in] IHObjectX Image2, [out] HUntypedObjectX VectorField, [in] String Mode,
```

---

---

Parameter

- **Image1** (input iconic) ......................... image(-array) \( \sim \) HImageX / IHObjectX (byte)
  First input image (optionally with more than one channel).
- **Image2** (input iconic) ......................... image(-array) \( \sim \) HImageX / IHObjectX (byte)
  Second input image (optionally with more than one channel).
- **Parameter Number** : \((Image2 = Image1)\)
- **VectorField** (output iconic) .................. image(-array) \( \sim \) HImageX / HUntypedObjectX (dvf)
  Displacement vector field to be calculated.
- **Parameter Number** : \((VectorField = Image1)\)
- **Mode** (input control) ......................... string \( \sim \) String
  Kind of correlation.
  Default Value : 'dfd'
- **Value List** : Mode \( \in \) \{'dfd', 'dfd_norm'\}
 Threshold (input control) .......................................................... integer \( \sim \) long
Maximal divergence of the previously estimated correlations and the best estimate of a displacement vector.

**Default Value:** 10

**Value Suggestions:** Threshold \( \in [1, 2, 3, 4, 5, 7, 10, 12, 15, 17, 20] \)

**Typical Range of Values:** \( 1 \leq \text{Threshold} \leq 1 \)

**Minimal Value Step:** 1

**Recommended Value Step:** 5

**Step** (input control) .......................................................... integer(-array) \( \sim \) VARIANT(integer)
Step width.

**Default Value:** 10

**Value Suggestions:** Step \( \in [1, 2, 3, 5, 7, 10, 15, 20, 30, 50] \)

**Typical Range of Values:** \( 1 \leq \text{Step} \leq 1 \)

**Minimal Value Step:** 1

**Recommended Value Step:** 5

**SizeWindow** (input control) ................................................ integer(-array) \( \sim \) VARIANT(integer)
Size of the correlation window.

**Default Value:** 11

**Value Suggestions:** SizeWindow \( \in [3, 5, 7, 9, 11, 15, 20, 30] \)

**Typical Range of Values:** \( 1 \leq \text{SizeWindow} \leq 1 \)

**Minimal Value Step:** 1

**Recommended Value Step:** 2

**SizeSearch** (input control) ................................................. integer(-array) \( \sim \) VARIANT(integer)
Size of the area to be searched.

**Default Value:** 20

**Value Suggestions:** SizeSearch \( \in [3, 5, 7, 10, 20, 30, 40, 50, 60] \)

**Typical Range of Values:** \( 1 \leq \text{SizeSearch} \leq 1 \)

**Minimal Value Step:** 1

**Recommended Value Step:** 2

**Weights** (input control) ..................................................... integer(-array) \( \sim \) VARIANT(integer)
Weighting of the channels.

**Default Value:** 1

**Value Suggestions:** Weights \( \in [1, 3, 5, 7, 10] \)

**Typical Range of Values:** \( 1 \leq \text{Weights} \leq 1 \)

**Minimal Value Step:** 1

**Recommended Value Step:** 1

---

**Example**

```plaintext
read_image(B1,'image1')
mean_image(B1,L1,16,16)
read_image(B2,'image2')
mean(B2,L2,16,16)
optical_flow_match(L1,L2,VVF,'dfd',10,10,11,20,1)
disp_image(B1,WindowHandle)
set_color(WindowHandle,'red')
disp_image(VVF,WindowHandle).
```

---

**Parallelization Information**

OpticalFlowMatch is *reentrant* and automatically *parallelized* (on tuple level).

**Possible Predecessor Functions**

MeanImage, GaussImage, SmoothImage

**Possible Successor Functions**

FillDvf

**See Also**

DvfToInt1, DvfToHomMat2d

---

Image filters

3.9. MATCH

| void HTemplateX.ReadTemplate ([in] String FileName ) |
| void HOperatorSetX.ReadTemplate ([in] String FileName, [out] long TemplateID ) |

Reading a template from file.
The operator ReadTemplate reads a matching template from file which has been written with WriteTemplate.

Parameter

- **FileName** (input control) ........................................ filename ~ String file name.
- **TemplateID** (output control) ......................................... template ~ HTemplateX / long Template number.

Result

If the file name is valid, the operator ReadTemplate returns the value TRUE. If necessary an exception handling will be raised.

Parallelization Information

ReadTemplate is local and processed completely exclusively without parallelization.

Possible Successor Functions

AdaptTemplate, SetReferenceTemplate, SetOffsetTemplate, BestMatch, FastMatch, BestMatchRot

Module

Template matching

| void HTemplateX.SetOffsetTemplate ([in] long GrayOffset ) |
| void HOperatorSetX.SetOffsetTemplate ([in] long TemplateID, [in] long GrayOffset ) |

Gray value offset for template.

SetOffsetTemplate adds a gray value offset to the template to eliminate gray value changes in the image. The parameter GrayOffset specifies a difference relative to the gray values of the pattern when it was created with CreateTemplate (not relative to the last call of SetOffsetTemplate). The values of GrayOffset has to be chosen according to the gray values of the image: A brighter image results in a positive value, a darker image results in a negative value. SetOffsetTemplate has to be called each time the gray values of the image changes. The gray values can be measured in a reference area using Intensity or MinMaxGray

Parameter

- **TemplateID** (input control) ......................................... template ~ HTemplateX / long Template number.
- **GrayOffset** (input control) .......................................... number ~ long Offset of gray values.

Default Value : 0

Value Suggestions : GrayOffset ∈ {-10, -5, -2, -1, 0, 1, 2, 5, 10}

Typical Range of Values : -255 ≤ GrayOffset ≤ -255

Minimal Value Step : 1

Recommended Value Step : 1

Result

If the parameter values are correct, the operator SetOffsetTemplate returns the value TRUE. If necessary, an exception handling is raised.

Parallelization Information

SetOffsetTemplate is processed under mutual exclusion against itself and without parallelization.

Possible Predecessor Functions

CreateTemplate, AdaptTemplate, ReadTemplate
Template matching

void HTemplateX.SetReferenceTemplate ([in] double Row,
[in] double Column )

void HOperatorSetX.SetReferenceTemplate ([in] long TemplateID,
[in] double Row, [in] double Column )

Define reference position for a matching template.

SetReferenceTemplate allows to define a new reference position for a template. As default after calling CreateTemplate or CreateTemplateRot the center of gravity of the template is used. Using SetReferenceTemplate the reference position can be redefined. In the case of the center of gravity as reference the vector (0, 0) is returned after matching for a null translation of the pattern relative to the image.

Parameter

- **TemplateID** (input control) . . . . . . . . . . . . . . . . . . . . template \~ HTemplateX / long Template number.
- **Row** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . point.y \~ double Reference position of template (row).
- **Column** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . point.x \~ double Reference position of template (column).

Result

If the parameter values are correct, the operator SetReferenceTemplate returns the value TRUE. If necessary, an exception handling is raised.

Parallelization Information

SetReferenceTemplate is processed under **mutual exclusion** against itself and without parallelization.

Possible Predecessor Functions

CreateTemplate, CreateTemplateRot, ReadTemplate, AdaptTemplate

Possible Successor Functions

BestMatch, BestMatchMg, BestMatchRot, FastMatch, FastMatchMg

Module

Template matching

void HTemplateX.WriteTemplate ([in] String FileName )

void HOperatorSetX.WriteTemplate ([in] long TemplateID,
[in] String FileName )

Writing a template to file.

The operator WriteTemplate writes a matching template to file which can be read again with ReadTemplate.

Parameter

- **TemplateID** (input control) . . . . . . . . . . . . . . . . . . . . template \~ HTemplateX / long Template number.
- **FileName** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . filename \~ String file name.

Result

If the file name is valid (permission to write), the operator WriteTemplate returns the value TRUE. If necessary an exception handling will be raised.
3.10. Misc

WriteTemplate is reentrant and processed without parallelization.

Possible Predecessor Functions

CreateTemplate, CreateTemplateRot

Parallelization Information

ConvolImage is reentrant and automatically parallelized (on tuple level, channel level).

Template matching

### ConvolImage

**ConvolImage** convolves the input image `Image` with an arbitrary linear filter. The corresponding filter matrix is stored in a file named `FileName`. Several options for the treatment at the image’s borders can be chosen (Margin):

- **0...255** Gray values are assumed constant outside of the image (with the given gray value).
- **-1** The border’s gray values are continued.
- **-2** The border’s gray values are continued cyclically.
- **-3** The gray values are mirrored along the image borders.

All image points are convolved with the filter mask. If an overflow or underflow occurs, the resulting gray value is clipped (to 0 or 255, respectively). The filter mask is contained in a file with the following structure:

- **Mask size**
- **Inverse weight of the mask**
- **Matrix**

The first line contains the size of the filter mask, given as two numbers separated by white space (e.g., 3 3 for 3 × 3). Here, the first number defines the height of the filter mask, while the second number defines its width. The next line contains the inverse weight of the mask, i.e., the number by which the convolution of a particular image point is divided. The remaining lines contain the filter mask as integer numbers (separated by white space), one line of the mask per line in the file. The file must have the extension “.fil”. This extension must not be passed to the operator.

**Parallelization Information**

**ConvolImage** is reentrant and automatically parallelized (on tuple level, channel level).

**Module**

Image filters
Expand the domain of an image and set the gray values in the expanded domain. The width of the expansion is set by the parameter ExpansionRange. All filters in HALCON use gray values of the pixels outside the domain depending on the filter width. This may lead to undesirable side effects especially in the border region of the domain. For example, if the foreground (domain) and the background of the image differ strongly in brightness, the result of a filter operation may lead to undesired darkening or brightening at the border of the domain. In order to avoid this drawback, the domain is expanded by ExpandDomainGray in a preliminary stage, copying the gray values of the border pixels to the outside of the domain. In addition, the domain itself is also expanded to reflect the newly set pixels. Therefore, in many cases it is reasonable to reduce the domain again (ReduceDomain or ChangeDomain) after using ExpandDomainGray and call the filter operation afterwards. ExpansionRange should be set to the half of the filter width.

**Parameter**

- **InputImage** (input iconic) .......................... image(-array)  $\sim$  HImageX / IObjectX (byte, int2)
  Input image with domain to be expanded.
- **ExpandedImage** (output iconic) ............... image(-array)  $\sim$  HImageX / HUntypedObjectX (byte, int2)
  Output image with new gray values in the expanded domain.
- **ExpansionRange** (input control) ...................... integer  $\sim$  long
  Radius of the gray value expansion, measured in pixels.
  Default Value: 2
  Value Suggestions: ExpansionRange $\in$ \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16\}
  Restriction: \(\text{ExpansionRange} \geq 1\)

**Example**

```c
read_image(Fabrik, 'fabrik.tif');
gen_rectangle2(Rectangle_Label,243,320,-1.55,62,28);
reduce_domain(Fabrik, Rectangle_Label, Fabrik_Label);
/* Character extraction without gray value expansion: */
mean_image(Fabrik_Label,Label_Mean_normal,31,31);
dyn_threshold(Fabrik_Label,Label_Mean_normal,Characters_normal,10,'dark');
dev_display(Fabrik);
dev_display(Characters_normal);
/* The characters in the border region are not extracted ! */
stop();
/* Character extraction with gray value expansion: */
expand_domain_gray(Fabrik_Label, Label_expanded,15);
reduce_domain(Label_expanded,Rectangle_Label, Label_expanded_reduced);
mean_image(Label_expanded_reduced,Label_Mean_expanded,31,31);
dyn_threshold(Fabrik_Label,Label_Mean_expanded,Characters_expanded,10,'dark');
dev_display(Fabrik);
dev_display(Characters_expanded);
/* Now, even in the border region the characters are recognized */
```

**Result**

ExpandDomainGray returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

**Parallelization Information**

ExpandDomainGray is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

ReduceDomain

**Possible Successor Functions**

ReduceDomain, MeanImage, DynThreshold
3.10. MISC

See Also
ReduceDomain, MeanImage

Module
Image filters

```c
[out] HImageX ImageDist HImageX.GrayInside ( )
void HOperatorSetX.GrayInside ([in] IHObjectX Image,
[out] HUntypedObjectX ImageDist )
```

Calculate the lowest possible gray value on an arbitrary path to the image border for each point in the image.

GrayInside determines the “cheapest” path to the image border for each point in the image, i.e., the path on which the lowest gray values have to be overcome. The resulting image contains the difference of the gray value of the particular point and the maximum gray value on the path. Bright areas in the result image therefore signify that these areas (which are typically dark in the original image) are surrounded by bright areas. Dark areas in the result image signify that there are only small gray value differences between them and the image border (which doesn’t mean that they are surrounded by dark areas; a small “gap” of dark values suffices). The value 0 (black) in the result image signifies that only darker or equally bright pixels exist on the path to the image border.

The operator is implemented by first segmenting into basins and watersheds the image using the Watersheds operator. If the image is regarded as a gray value mountain range, basins are the places where water accumulates and the mountain ridges are the watersheds. Then, the watersheds are distributed to adjacent basins, thus leaving only basins. The border of the domain (region) of the original image is now searched for the lowest gray value, and the region in which it resides is given its result values. If the lowest gray value resides on the image border, all result values can be calculated immediately using the gray value differences to the darkest point. If the smallest found gray value lies in the interior of a basin, the lowest possible gray value has to be determined from the already processed adjacent basins in order to compute the new values. An 8-neighborhood is used to determine adjacency. The found region is subtracted from the regions yet to process, and the whole process is repeated. Thus, the image is “stripped” from the outside.

Analogously to Watersheds, it is advisable to apply a smoothing operation before calling Watersheds, e.g., GaussImage, in order to reduce the amount of regions that result from the watershed algorithm, and thus to speed up the processing time.

Parameter

- **Image** (input iconic) ...........................(multichannel-)image(-array) ~ HImageX / IHObjectX ( byte )
  Image being processed.

- **ImageDist** (output iconic) .......(multichannel-)image(-array) ~ HImageX / HUntypedObjectX ( int2 )
  Result image.

Example

```c
read_image(Bild,'coin')
gauss_image (Bild,G_Bild,11)
open_window (0,0,512,512,0,'visible',' ',WindowHandle)
gray_inside(G_Bild,Ausgabebild)
disp_image (Ausgabebild,WindowHandle).
```

Result

GrayInside always returns TRUE.

Parallelization Information

GrayInside is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessor Functions

GaussImage, SmoothImage, MeanImage, MedianImage

Possible Successor Functions

SelectShape, AreaCenter, CountObj
See Also

Watersheds

Module

Image filters

```c
[out] HImageX GraySkeleton
HImageX.GraySkeleton( )

void HOperatorSetX.GraySkeleton ([in] IHObjectX Image,
[out] HUntypedObjectX GraySkeleton )
```

**Thinning of gray value images.**

GraySkeleton applies a gray value thinning operation to the input image Image. Figuratively, the gray value “mountain range” is reduced to its ridge lines by setting the gray value of “hillsides” to the gray value at the corresponding valley bottom. The resulting ridge lines are at most two pixels wide. This operator is especially useful for thinning edge images, and is thus an alternative to NonmaxSuppressionAmp. In contrast to NonmaxSuppressionAmp, GraySkeleton preserves contours, but is much slower. In contrast to Skeleton, this operator changes the gray values of an image while leaving its region unchanged.

**Parameter**

- **Image** (input iconic) .................. (multichannel-)image(-array) \(\rightarrow\) HImageX / IHObjectX ( byte )
  Image to be thinned.
- **GraySkeleton** (output iconic) ... (multichannel-)image(-array) \(\rightarrow\) HImageX / HUntypedObjectX ( byte )
  Thinned image.

**Example**

```c
/* Seeking leafs of a beech tree in an aerial picture: */
read_image(Image,'wald1')
gray_skeleton(Image,Skelett)
mean_image(Skelett,MeanSkelett,7,7)
dyn_threshold(Skelett,MeanSkelett,Leafs,3,'light').
```

**Result**

GraySkeleton returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception is raised.

**Parallelization Information**

GraySkeleton is reentrant and automatically parallelized (on tuple level, channel level).

**Possible Successor Functions**

MeanImage

**See Also**

Skeleton, GrayDilationRect

**Alternatives**

NonmaxSuppressionAmp, NonmaxSuppressionDir, LocalMax

**Module**

Image filters

```c
[out] HImageX ImageResult
HImageX.LutTrans ([in] VARIANT Lut )

void HOperatorSetX.LutTrans ([in] IHObjectX Image,
```

**Transform an image with a gray-value look-up-table**

LutTrans transforms an image Image by using a gray value look-up-table Lut. This table acts as a transformation function.

3.10. MISC

Parameter

- **Image** (input iconic) ........................................ (multichannel-)image-array \( \sim HImageX / HObjectX \) (byte)
  Image whose gray values are to be transformed.
- **ImageResult** (output iconic) .... (multichannel-)image-array \( \sim HImageX / HUntypedObjectX \) (byte)
  Transformed image.
- **Lut** (input control) ......................................................integer \( \sim \) VARIANT(integer)
  Table containing the transformation.

Example

/* To get the inverse of an image: */
read_image(Image,'waldl')
def_tab(Tab,0)
lut_trans(Image,Invers,Tab,1,1)

def_tab(Tab,I) :- I=255
  Tab = 0

def_tab([Tk|Ts],I) :-
  Tk is 255 - I
  Iw is I -1
  def_tab(Ts,Iw)

Result

The operator LutTrans returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

Parallelization Information

LutTrans is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Module

Image filters

Compute the principal components of multi-channel images.

PrincipalComp does a principal components analysis of multi-channel images. This is useful for images obtained, e.g., with the thematic mapper of the Landsat satellite. Because the spectral bands are highly correlated it is desirable to transform them to uncorrelated images. This can be used to save storage, since the bands containing little information can be discarded, and with respect to a later classification step.

The operator PrincipalComp takes a (multi-channel) image MultichannelImage and transforms it to the output image PCAImage, which contains the same number of channels, using the principal components analysis. The parameter InfoPerComp contains the relative information content of each output channel.

Parameter

- **MultichannelImage** (input iconic) .... multichannel-image-array \( \sim HImageX / HObjectX \) (byte, int1, int2, int4, real)
  Multi-channel input image.
- **PCAImage** (output iconic) .............. multichannel-image-array \( \sim HImageX / HUntypedObjectX \) (real)
  Multi-channel output image.
- **InfoPerComp** (output control) ..........................................real \( \sim \) VARIANT(real)
  Information content of each output channel.

Result

The operator PrincipalComp returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

HALCON 6.0
Parallelization Information

PrincipalComp is *reentrant* and processed *without* parallelization.

Image filters

```
[out] HImageX ImageSymmetry HImageX.Symmetry ([in] long MaskSize,
[in] double Direction, [in] double Exponent )
```

```
void HOperatorSetX.Symmetry ([in] IHObjectX Image,
[out] HUntypedObjectX ImageSymmetry, [in] long MaskSize,
[in] double Direction, [in] double Exponent )
```

*Symmetry of gray values along a row.*

Symmetry calculates the symmetry along a line. For each pixel the gray values of both sides of the line are compared: The absolute value of the differences of gray values with same distance to the pixel is computed. Each of these differences is weighted by the exponent (after division by 255) and the summed up.

$$
sym := 255 - \frac{255}{\text{MaskSize}} \sum_{i=1}^{\text{MaskSize}} \left( \frac{|g(i) - g(-i)|}{255} \right)^{\text{Exponent}}
$$

Pixels with a high symmetry have large gray values.

Currently only horizontal search lines are implemented

**Attention**

**Parameter**

- **Image** (input iconic) ...........................................(multichannel-)image(-array) $\sim$ HImageX / IHObjectX ( byte )
  - Input image.
- **ImageSymmetry** (output iconic) ..................................(multichannel-)image(-array) $\sim$ HImageX / HUntypedObjectX ( byte )
  - Symmetry image.
- **MaskSize** (input control) ......................................................number $\sim$ long
  - Extension of search area.
  - Default Value : 40
  - Value Suggestions : MaskSize $\in \{3, 5, 7, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 100, 120, 140, 180\}$
  - Typical Range of Values : $3 \leq \text{MaskSize} \leq 3$
  - Minimal Value Step : 1
  - Recommended Value Step : 2
- **Direction** (input control) ......................................................number $\sim$ double
  - Angle of test direction.
  - Default Value : 0.0
  - Value Suggestions : Direction $\in \{0.0\}$
  - Typical Range of Values : $0.0 \leq \text{Direction} \leq 0.0$
- **Exponent** (input control) ......................................................number $\sim$ double
  - Exponent for weighting.
  - Default Value : 0.5
  - Value Suggestions : Exponent $\in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1.0\}$
  - Typical Range of Values : $0.05 \leq \text{Exponent} \leq 0.05$
  - Minimal Value Step : 0.01
  - Recommended Value Step : 0.1
  - **Restriction** : $(0 < \text{Exponent} \land (\text{Exponent} \leq 1))$

**Example**

```
read_image(Image, 'monkey')
symmetry(Image, ImageSymmetry, 70, 0.0, 0.5)
threshold(ImageSymmetry, SymmPoints, 170, 255)
```
If the parameter values are correct the operator Symmetry returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem({'noObjectResult',<Result>:}). If necessary an exception handling is raised.

**Parallelization Information**

Symmetry is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Successor Functions**

Image filters

```
[out] HImageX Sketch HImageX.TopographicSketch ( )
void HOperatorSetX.TopographicSketch ([in] IHObjectX Image,
[out] HUntypedObjectX Sketch )
```

*Compute the topographic primal sketch of an image.*

TopographicSketch computes the topographic primal sketch of the input image Image. This is done by approximating the image locally by a bicubic polynomial (“facet model”). It serves to calculate the first and second partial derivatives of the image, and thus to classify the image into 11 classes. These classes are coded in the output image Sketch as numbers from 1 to 11. The classes are as follows:

- Peak: 1
- Pit: 2
- Ridge: 3
- Ravine: 4
- Saddle: 5
- Flat: 6
- Hillside Slope: 7
- Hillside Convex: 8
- Hillside Concave: 9
- Hillside Saddle: 10
- Hillside Inflection: 11

In order to obtain the separate classes as regions, a threshold operation has to be applied to the result image with the appropriate thresholds.

**Parameter**

- **Image** (input iconic) ................. (multichannel-)image(-array) ∼ HImageX / IHObjectX (byte)  
  Image for which the topographic primal sketch is to be computed.
- **Sketch** (output iconic) ............ (multichannel-)image(-array) ∼ HImageX / HUntypedObjectX (byte)  
  Label image containing the 11 classes.

**Example**

```c
/* To extract the Ridges from a Image */
read_image(Image,'sinus')
topographic_sketch(Image,Sketch)
threshold(Sketch,Ridges,3,3).
```

**Result**

TopographicSketch returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem({'noObjectResult',<Result>:}). If necessary, an exception is raised.

**Parallelization Information**

TopographicSketch is reentrant and automatically parallelized (on tuple level, channel level).

**Possible Successor Functions**

Threshold
Bibliography


Module

3.11 Noise

Add noise to an image.

AddNoiseDistribution adds noise distributed according to Distribution to the image Image. Resulting gray values are clipped to the range \([0,255]\).

Parameter

- **Image** (input iconic) ......................(multichannel-image-array) \(\sim\) HImageX / IObjectX( byte, int2 ) Input image.
- **ImageNoise** (output iconic) . (multichannel-image-array) \(\sim\) HImageX / HUntypedObjectX( byte, int2 ) Noisy image.
- **Distribution** (input control) ...................distribution.values \(\sim\) VARIANT( real ) Noise distribution.

Example

```plaintext
read_image(Image,'meer_rot')
disp_image(Image,WindowHandle)
sp_distribution(30,30,Dist)
add_noise_distribution(Image,ImageNoise,Dist)
disp_image(ImageNoise,WindowHandle).
```

Result

AddNoiseDistribution returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

AddNoiseDistribution is *reentrant* and automatically *parallelized* (on tuple level, channel level, domain level).

Possible Predecessor Functions

GaussDistribution, SpDistribution, NoiseDistributionMean

See Also

SpDistribution, GaussDistribution, NoiseDistributionMean, AddNoiseWhite

Alternatives

AddNoiseWhite

Module

Image filters
Add noise to an image.

AddNoiseWhite adds noise to the image Image. The noise is white noise, equally distributed in the interval \([-Amp, Amp]\), and is generated by using the C function “drand48” with an initial time dependent seed. Resulting gray values are clipped to the range [0,255].

**Parameter**

- **Image** (input iconic) ................. (multichannel-)image(-array) \(\sim\) HImageX / HObjectX (byte, int2) Input image.
- **ImageNoise** (output iconic) . (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, int2) Noisy image.

**Parameter Number:** (ImageNoise = Image)

- **Amp** (input control) .................................................. real \(\sim\) double Maximum noise amplitude.

**Default Value:** 60.0

**Value Suggestions:** \(Amp \in \{1.0, 2.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}\)

**Typical Range of Values:** \(1.0 \leq Amp \leq 1.0\)

**Minimal Value Step:** 1.0

**Recommended Value Step:** 10.0

**Restriction:** \((Amp > 0)\)

**Example**

```plaintext```
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
add_noise_white(Image,ImageNoise,90)
disp_image(ImageNoise,WindowHandle).
```

**Result**

AddNoiseWhite returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

**Parallelization Information**

AddNoiseWhite is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**See Also**

AddNoiseDistribution, NoiseDistributionMean, GaussDistribution, SpDistribution

**Alternatives**

AddNoiseDistribution

**Module**

Image filters

---

Generate a Gaussian noise distribution.

GaussDistribution generates a Gaussian noise distribution. The parameter Sigma determines the noise’s standard deviation. Usually, the result Distribution is used as input for the operator AddNoiseDistribution.
Parameter

**Sigma** (input control) ..................................................... real  \(\sim\) double
Standard deviation of the Gaussian noise distribution.

- **Default Value:** 2.0
- **Value Suggestions:** \(\Sigma \in \{1.5, 2.0, 3.0, 5.0, 10.0\}\)
- **Typical Range of Values:** \(0.0 \leq \Sigma \leq 0.0\)
- **Minimal Value Step:** 0.1
- **Recommended Value Step:** 1.0

**Distribution** (output control) .................................... distribution.values  \(\sim\) VARIANT (real)
Resulting Gaussian noise distribution.

Example

```c
read_image(Image,‘fabrik’)
disp_image(Image,WindowHandle)
 gauss_distribution(30,Dist)
 add_noise_distribution(Image,ImageNoise,Dist)
disp_image(ImageNoise,WindowHandle).
```

Parallelization Information

GaussDistribution is **reentrant** and processed **without** parallelization.

Possible Successor Functions

AddNoiseDistribution

See Also

SpDistribution, AddNoiseWhite, NoiseDistributionMean

Alternatives

SpDistribution, NoiseDistributionMean

Module

Image filters

```c
[out] VARIANT Distribution HRegionX.NoiseDistributionMean
([in] HImageX Image, [in] long FilterSize )

void HOperatorSetX.NoiseDistributionMean
([in] IHObjectX ConstRegion, [in] IHObjectX Image, [in] long FilterSize,
[out] VARIANT Distribution )
```

Determine the noise distribution of an image.

NoiseDistributionMean calculates the noise distribution in a region of the image Image. The parameter ConstRegion determines a region of the image with approximately constant gray values. Ideally, the changes in gray values should only be caused by noise in this region. From this region the noise distribution is determined by using the MeanImage operator to smooth the image, and to use the gray value differences in this area as an estimate for the noise distribution, which is returned in Distribution.

Attention

It is important to ensure that the region ConstRegion is not too close to a large gradient in the image, because the gradient values are then used for calculating the mean. This means the the distance of ConstRegion must be at least as large as the filter size FilterSize used for calculating the mean.

Parameter

- **ConstRegion** (input iconic) ............................. region(-array)  \(\sim\) HRegionX / IHObjectX
  Region from which the noise distribution is to be estimated.
- **Image** (input iconic) ......................................... image  \(\sim\) HImageX / IHObjectX (byte)
  Corresponding image.
3.11. NOISE

- **FilterSize** (input control) ............................................ integer ~ long
  Size of the mean filter.
  **Default Value**: 21
  **Value Suggestions**: FilterSize ∈ {5, 11, 15, 21, 31, 51, 101}
  **Typical Range of Values**: 3 ≤ FilterSize ≤ 3 (lin)
  **Minimal Value Step**: 2
  **Recommended Value Step**: 2

- **Distribution** (output control) .......................... distribution.values ~ VARIANT (real)
  Noise distribution of all input regions.

**Parallelization Information**

DrawRegion, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, Threshold,
ErosionCircle, GaussImage, SmoothImage, SubImage

**Possible Predecessor Functions**

AddNoiseDistribution, DispDistribution

**Possible Successor Functions**

See Also

MeanImage, GaussDistribution

**Image filters**

```
[out] VARIANT Distribution HMiscX.SpDistribution
([in] VARIANT PercentSalt, [in] VARIANT PercentPepper )

void HOperatorSetX.SpDistribution ([in] VARIANT PercentSalt,
[in] VARIANT PercentPepper, [out] VARIANT Distribution )
```

Generate a salt-and-pepper noise distribution.

SpDistribution generates a noise distribution with the values 0 and 255. The parameters PercentSalt and PercentPepper determine the percentage of white and black noise pixels, respectively. The sum of these parameters must be smaller than 100. Usually, the result Distribution is used as input for the operator AddNoiseDistribution.

**Parameter**

- **PercentSalt** (input control) ................................. number ~ VARIANT (real, integer)
  Percentage of salt (white noise pixels).
  **Default Value**: 5.0
  **Value Suggestions**: PercentSalt ∈ {1.0, 2.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0}
  **Typical Range of Values**: 0.0 ≤ PercentSalt ≤ 0.0
  **Minimal Value Step**: 0.1
  **Recommended Value Step**: 1.0
  **Restriction**: ((0.0 ≤ PercentSalt) ∧ (PercentSalt ≤ 100.0))

- **PercentPepper** (input control) ............................. number ~ VARIANT (real, integer)
  Percentage of pepper (black noise pixels).
  **Default Value**: 5.0
  **Value Suggestions**: PercentPepper ∈ {1.0, 2.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0}
  **Typical Range of Values**: 0.0 ≤ PercentPepper ≤ 0.0
  **Minimal Value Step**: 0.1
  **Recommended Value Step**: 1.0
  **Restriction**: ((0.0 ≤ PercentPepper) ∧ (PercentPepper ≤ 100.0))

- **Distribution** (output control) .......................... distribution.values ~ VARIANT (real)
  Resulting noise distribution.
  **Parameter Number**: 513

**Example**

HALCON 6.0
read_image(Image, 'fabrik')
disp_image(Image, WindowHandle)
sp_distribution(30, 30, Dist)
add_noise_distribution(Image, ImageNoise, Dist)
disp_image(ImageNoise, WindowHandle)

---

**Parallelization Information**

SpDistribution is reentrant and processed without parallelization.

---

**Possible Successor Functions**

AddNoiseDistribution

---

**See Also**

GaussDistribution, NoiseDistributionMean, AddNoiseWhite

---

**Alternatives**

GaussDistribution, NoiseDistributionMean

---

**Module**

Image filters

### 3.12 Smoothing

Smooth an image by edge-preserving anisotropic diffusion.

The operator AnisotropeDiff carries out an iterative, anisotropic smoothing process on the mathematical basis of physical diffusion. In analogy to the physical diffusion process describing the concentration balance between molecules dependent on the density gradient, the diffusion filter carries out a smoothing of the gray values dependent on the local gray value gradients.

For iterative calculation of the gray value of a pixel the gray value differences in relation to the four or eight neighbors, respectively, are used. These gray value differences, however, are evaluated differently, i.e., a non-linear diffusion process is carried out. The evaluation is carried out by using a diffusion function (two different functions were implemented, namely Mode = 1 and/or 2), which — depending on the gradient — ensures that within homogenous regions the smoothing is stronger than over the margins of regions so that the edges remain sharp. The diffusion function is adjusted to the noise ratio of the image by a histogram analysis in the gradient image (according to Canny). A high value for Percent increases the smoothing effect but blurs the edges a little more (values from 80 - 90 percent are typical).

The parameter Iteration determines the number of iterations (typically 3–7).

---

**Parameter**

- **Image** (input iconic) ................. (multichannel-)image(-array) ~ HImageX / IHObjectX (byte)
  Image to be smoothed.
- **ImageAniso** (output iconic) ........ (multichannel-)image(-array) ~ HImageX / HUntypedObjectX (byte)
  Smoothed image.
- **Percent** (input control) ............ (multichannel-)image(-array) ..........................integer ~ long
  For histogram analysis; higher values increase the smoothing effect, typically: 80-90.
  Default Value : 80
  Value Suggestions : Percent ∈ {65, 70, 75, 80, 85, 90}
  Typical Range of Values : 50 ≤ Percent ≤ 50
  Minimal Value Step : 1
  Recommended Value Step : 5
3.12. SMOOTHING

- **Mode** (input control)  
  Default Value: 1  
  Value List: Mode ∈ {1, 2}  
  No description available.

- **Iteration** (input control)  
  Default Value: 5  
  Value Suggestions: Iteration ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}  
  Typical Range of Values: 1 ≤ Iteration ≤ 1  
  Minimal Value Step: 1  
  Recommended Value Step: 1

- **neighborhoodType** (input control)  
  Default Value: 8  
  Value List: neighborhoodType ∈ {4, 8}

---

Example

```h
read_image(Image, 'fabrik')
anisotrope_diff(Image, Aniso, 80, 1, 5, 8)
sub_image(Image, Aniso, Sub, 2.0, 127)
disp_image(Sub, WindowHandle).
```

---

Result

If the parameter values are correct the operator `AnisotropeDiff` returns the value `TRUE`. The behavior in case of empty input (no input images available) is set via the operator `SetSystem('noObjectResult', <Result>)`. If necessary an exception handling is raised.

**Parallelization Information**

`AnisotropeDiff` is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Predecessor Functions**

ReadImage, GrabImage

**Possible Successor Functions**

RegionGrowing, Threshold, SubImage, DynThreshold, AutoThreshold

**See Also**

SmoothImage, GaussImage, SigmaImage, RankImage, EliminateMinMax

**Alternatives**

SigmaImage, RankImage

**Bibliography**


**Module**

Image filters

```h
```

```h
void HOperatorSetX.EliminateMinMax([in] IHObjectX Image,  
[out] HUntypedObjectX filteredImage, [in] long MaskWidth,  
```

Smooth an image in the spatial domain to suppress noise.

EliminateMinMax smooths an image by replacing gray values with neighboring mean values, or local minima/maxima. In order to prevent edges and lines from being smoothed, only those gray values that represent local minima or maxima are replaced (if there is a line or edge within an image there will be at least one neighboring
pixel with a comparable gray value). \( \text{Gap} \) controls the strictness of replacement: Only gray values that exceed all other values within their local neighborhood more than \( \text{Gap} \) and all values that fall below their neighboring more than \( \text{Gap} \) are replaced: \( E(x, y) \) represents a \( N \times M \) sized rectangular neighborhood of an pixel at position \( (x, y) \), containing all pixels within the neighborhood except the pixel itself:

- if \( \text{grayvalue}(x, y) \geq \text{Gap} + \max(E(x, y)) \) then replacement;
- else if \( \text{grayvalue}(x, y) + \text{Gap} \leq \min(E(x, y)) \) then replacement;
- else adopt \( \text{grayvalue}(x, y) \) without change;

\( \text{Mode} \) specifies how to perform the new value in case of a replacement.

\( \text{Mode} = 1 \rightarrow \) replace a local maximum with next minor local maximum and replace a local minimum with next bigger local minimum

\( \text{Mode} = 2 \rightarrow \) replace with mean value of all pixels within the local neighborhood (including the replaced pixel)

\( \text{Mode} = 3 \rightarrow \) replace with median value of all pixels within the local neighborhood (including the replaced pixel (this is default and used if \( \text{Mode} \) has got any other value than 1 or 2)

\( \text{MaskWidth} \) and \( \text{MaskHeight} \) specify the width and height of the rectangular neighborhood. Border treatment: Pixels outside the image border are not considered (e.g.: With a local \( 3 \times 3 \)-mask the neighborhood of a pixel at \( (0, 0) \) reduces to the pixels at \( (1, 0), (0, 1) \) and \( (1, 1) \)).

---

**Attention**

\( \text{EliminateMinMax} \) only can work on byte images (HALCON image type \( \text{BYTE\_IMAGE} \)). If \( \text{MaskWidth} \) or \( \text{MaskHeight} \) is an even number, it is replaced by the next higher odd number (this allows the unique extraction of the center of the filter mask). Width/height of the mask may not exceed the image width/height.

---

**Parameter**

- **Image** (input iconic) .................\( \text{(multichannel\_image)} \sim \text{HImageX / IOBJECTX (byte)} \) Image to smooth.
- **filteredImage** (output iconic) .................\( \text{(multichannel\_image)} \sim \text{HImageX / HUNTYPEOBJECTX (byte)} \) Smoothed image.
- **MaskWidth** (input control) ..................exten.x \sim \text{long}
  - Default Value : 3
  - Value Suggestions : \( \text{MaskWidth} \in \{3, 5, 7, 9\} \)
  - Typical Range of Values : \( 3 \leq \text{MaskWidth} \leq 3 \)
  - Minimal Value Step : 2
  - Recommended Value Step : 2
  - Restriction : \( \text{odd} \)
- **MaskHeight** (input control) ..................exten.y \sim \text{long}
  - Default Value : 3
  - Value Suggestions : \( \text{MaskHeight} \in \{3, 5, 7, 9\} \)
  - Typical Range of Values : \( 3 \leq \text{MaskHeight} \leq 3 \)
  - Minimal Value Step : 2
  - Recommended Value Step : 2
  - Restriction : \( \text{odd} \)
- **Gap** (input control) .............................................. number \( \sim \text{double} \)
  - Default Value : 1.0
  - Value Suggestions : \( \text{Gap} \in \{1.0, 2.0, 5.0, 10.0\} \)
- **Mode** (input control) .............................................. integer \( \sim \text{long} \)
  - Replacement rule (1 = next minimum/maximum, 2 = average, 3 = median).
  - Default Value : 3
  - Value List : \( \text{Mode} \in \{1, 2, 3\} \)

---

**Result**

\( \text{EliminateMinMax} \) returns \( \text{TRUE} \) if all parameters are correct. If the input is empty \( \text{EliminateMinMax} \) returns with an error message.
3.12. SMOOTHING

Parallelization Information
EliminateMinMax is **reentrant** and automatically **parallelized** (on tuple level, channel level, domain level).

Possible Successor Functions
WienerFilter, WienerFilterNi

See Also
MeanSp, MeanImage, MedianImage, MedianWeighted, GaussImage, SmoothImage

Bibliography
M. L”uckenhaus:“Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse”; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Image filters

```c
```

Replace values outside of thresholds with average value.

The operator EliminateSp replaces all gray values outside the indicated gray value intervals (MinThresh to MaxThresh) with the neighboring mean values. Only those neighboring pixels which also fall within the gray value interval are used for averaging. If no such pixel is present in the vicinity the original gray value is used. The gray values in the input image falling within the gray value interval are also adopted without change.

**Attention**
If even values instead of odd values are given for MaskHeight or MaskWidth, the routine uses the next larger odd values instead (this way the center of the filter mask is always explicitly determined).

**Parameter**

- **Image** (input iconic) ...................(multichannel-)image(-array) ~ HImageX / HObjectX ( byte )
  Input image.
- **ImageFillSP** (output iconic) .......(multichannel-)image(-array) ~ HImageX / HUntypedObjectX ( byte )
  Smoothed image.
- **MaskWidth** (input control) ..................extent.x ~ long
  Width of filter mask.
  **Default Value**: 3
  **Value Suggestions**: MaskWidth ∈ {3, 5, 7, 9, 11}
  **Typical Range of Values**: 3 ≤ MaskWidth ≤ 3(lin)
  **Minimal Value Step**: 2
  **Recommended Value Step**: 2
  **Restriction**: odd
- **MaskHeight** (input control) ..................extent.y ~ long
  Height of filter mask.
  **Default Value**: 3
  **Value Suggestions**: MaskHeight ∈ {3, 5, 7, 9, 11}
  **Typical Range of Values**: 3 ≤ MaskHeight ≤ 3(lin)
  **Minimal Value Step**: 2
  **Recommended Value Step**: 2
  **Restriction**: odd

HALCON 6.0
MinThresh (input control) .............................................................. integer \sim long
Minimum gray value.
Default Value : 1
Value Suggestions : MinThresh \in \{1, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101\}
Typical Range of Values : 0 \leq \text{MinThresh} \leq 0(lin)
Minimal Value Step : 1
Recommended Value Step : 1
MaxThresh (input control) .............................................................. integer \sim long
Maximum gray value.
Default Value : 254
Value Suggestions : MaxThresh \in \{5, 7, 9, 11, 15, 23, 31, 43, 61, 101, 200, 230, 250, 254\}
Typical Range of Values : 1 \leq \text{MaxThresh} \leq 1(lin)
Minimal Value Step : 1
Recommended Value Step : 1
Restriction : (\text{MinThresh} \leq \text{MaxThresh})

Example

read_image(Image,'meer_rot')
disp_image(Image,WindowHandle)
eliminate_sp(Image,ImageMeansp,3,3,101,201)
disp_image(ImageMeansp,WindowHandle).

Parallelization Information
EliminateSp is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successor Functions
DispImage

See Also
GaussImage, SmoothImage, AnisotropeDiff, SigmaImage, EliminateMinMax

Alternatives
MeanSp, MeanImage, MedianImage, EliminateMinMax

Module

Image filters

Interpolate 2 video half images.

The operator FillInterlace calculates an interpolated full image or removes odd/even lines from a video image composed of two half images. If an image is recorded with a video camera it consists of two half images recorded at different times but stored in one image in the digital form. This can lead to several errors in further processing. In order to reduce these errors the video image is modified. Every second line is re-calculated or removed. The parameter Mode determines whether this must be done for even (‘even’, ‘rmeven’) or odd (‘odd’, ‘rmodd’) line numbers. If you choose ‘even’ or ‘odd’ the gray values in the generated lines are calculated as mean values from the direct neighbors above or below the current pixel, respectively. If you choose ‘rmeven’ or ‘rmodd’ the even or odd lines numbers are removed (in that case the resulting image is only half as high as the input image). The value ‘switch’ for Mode cause the odd and even lines to be exchanged.

Parameter

\begin{itemize}
\item \textbf{ImageCamera} (input iconic) \ldots \ldots \ldots (multichannel-)image(-array) \sim \text{HImageX / IObjectX (byte)}
Gray image consisting of two half images.
\item \textbf{ImageFilled} (output iconic) \ldots \ldots (multichannel-)image(-array) \sim \text{HImageX / HUntypedObjectX (byte)}
Full image with interpolated/removed lines.
\end{itemize}
3.12. SMOOTHING

- **Mode** (input control) ................................................................. string → String  
  Instruction whether even or odd lines should be replaced/removed.
  
  **Default Value**: 'odd'
  
  **Value List**: Mode ∈ {'odd', 'even', 'rmodd', 'rmeven', 'switch'}

---

### Example

```plaintext
read_image(Image,'video_bild')
fill_interlace(Image,New,'odd')
sobel_amp(New,Sobel,'sum_abs',3).
```

---

### Result

If the parameter values are correct the operator FillInterlace returns the value TRUE.

The behavior in case of empty input (no input images available) is set via the operator SetSystem('noObjectResult',<Result>). If necessary an exception handling is raised.

---

### Parallelization Information

FillInterlace is reentrant and automatically parallelized (on tuple level, channel level, domain level).

---

### Possible Predecessor Functions

ReadImage, GrabImage

### Possible Successor Functions

SobelAmp, EdgesImage, Regiongrowing, DiffOfGauss, Threshold, DynThreshold, AutoThreshold, MeanImage, GaussImage, AnisotropeDiff, SigmaImage, MedianImage

---

### See Also

MedianImage, GaussImage, CropPart

---

### Module

Image filters

---

<table>
<thead>
<tr>
<th>[out] HImageX ImageGauss HImageX.GaussImage ([in] long Size )</th>
</tr>
</thead>
</table>
void HOperatorSetX.GaussImage ([in] IObjectX Image,  
[out] HUntypedObjectX ImageGauss, [in] long Size )

**Smooth using discrete gauss functions.**

The operator GaussImage smoothes images using the discrete Gaussian. The smoothing effect increases with increasing filter size. The following filter sizes (Size) are supported (the sigma value of the gauss function is indicated in brackets):

<table>
<thead>
<tr>
<th>Size</th>
<th>Sigma Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.81</td>
</tr>
<tr>
<td>5</td>
<td>0.93</td>
</tr>
<tr>
<td>7</td>
<td>1.50</td>
</tr>
<tr>
<td>9</td>
<td>2.00</td>
</tr>
<tr>
<td>11</td>
<td>2.45</td>
</tr>
</tbody>
</table>

For margin control the gray values of the images are reflected at the image borders.

---

### Parameter

- **Image** (input iconic) .................(multichannel-)image(-array) → HImageX / IObjectX ( byte, int4, int2 )  
  Image to be smoothed.

- **ImageGauss** (output iconic) .......(multichannel-)image(-array) → HImageX / HUntypedObjectX ( byte, int4, int4 )  
  Filtered image.

- **Size** (input control) ......................................................integer → long  
  Required filter size.
  
  **Default Value**: 5
  
  **Value List**: Size ∈ {3, 5, 7, 9, 11}
Example

gauss_image(Input,Gauss,7)
regiongrowing(Gauss,Segments,7,7,5,100).

Result
If the parameter values are correct the operator GaussImage returns the value TRUE.
The behavior in case of empty input (no input images available) is set via the operator
SetSystem('noObjectResult',<Result>). If necessary an exception handling is raised.

Parallelization Information
GaussImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessor Functions
ReadImage, GrabImage

Possible Successor Functions
Regiongrowing, Threshold, SubImage, DynThreshold, AutoThreshold

See Also
MeanImage, AnisotropeDiff, SigmaImage, GenLowpass

Alternatives
SmoothImage, DerivateGauss

Image filters

Information on smoothing filter SmoothImage.
The operator InfoSmooth returns an estimation of the width of the smoothing filters used in routine
SmoothImage. For this purpose the underlying continuous impulse answers of Filter are scanned until a
filter coefficient is smaller than five percent of the maximum coefficient (at the origin). Alpha is the filter param-
eter (see SmoothImage). Currently four filters are supported (parameter Filter):

'deriche1', 'deriche2', 'shen' und 'gauss'.

The gauss filter was conventionally implemented with filter masks (the other three are recursive filters). In the case
of the gauss filter the filter coefficients (of the one-dimensional impulse answer \( f(n) \) with \( n \geq 0 \)) are returned in
Coeffs in addition to the filter size.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td>string</td>
<td>Name of required filter. Default Value: 'deriche2'. Value List: Filter ∈ {'deriche1', 'deriche2', 'shen', 'gauss'}</td>
</tr>
<tr>
<td>Alpha</td>
<td>real</td>
<td>Filter parameter: small values effect strong smoothing (reversed in case of 'gauss'). Default Value: 0.5 Value Suggestions: Alpha ∈ [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0, 10.0] Typical Range of Values: 0.01 ≤ Alpha ≤ 0.01 Minimal Value Step: 0.01 Recommended Value Step: 0.1 Restriction: (Alpha &gt; 0.0)</td>
</tr>
<tr>
<td>Size</td>
<td>integer</td>
<td>Width of filter is approx. size x size pixels.</td>
</tr>
</tbody>
</table>

3.12. SMOOTHING

- **Coeffs** (output control) .................................................. integer ~ VARIANT( integer )
  In case of gauss filter: coefficients of the “positive” half of the 1D impulse answer.
  
  Example

  info_smooth('deriche2', 0.5, Size, Coeffs)
  smooth_image(Input, Smooth, 'deriche2', 7).

  Result

  If the parameter values are correct the operator InfoSmooth returns the value TRUE. Otherwise an exception handling is raised.

  Parallelization Information

  InfoSmooth is reentrant and processed without parallelization.

  Possible Predecessor Functions

  ReadImage

  Possible Successor Functions

  SmoothImage

  See Also

  SmoothImage

  Module

Image filters

Smooth by averaging.

The operator MeanImage carries out a linear smoothing with the gray values of all input images (Image). The filter matrix consists of ones (evaluated equally) and has the size MaskHeight \times MaskWidth. The result of the convolution is divided by MaskHeight \times MaskWidth. For margin control the gray values are reflected at the image edges.

Attention

If even values instead of odd values are given for MaskHeight or MaskWidth, the routine uses the next larger odd values instead (this way the center of the filter mask is always explicitly determined).

Parameter

- **Image** (input iconic) ...(multichannel-)image(-array) \sim HImageX / IObjectX ( byte, int2, int4, real, dvf )
  Image to be smoothed.

- **ImageMean** (output iconic) ...(multichannel-)image(-array) \sim HImageX / HUntypedObjectX ( byte, int2, int4, real, dvf )
  Smoothed image.

- **MaskWidth** (input control) .......................................................... extent.x \sim long
  Width of filter mask.

  Default Value : 9
  Value Suggestions : MaskWidth \in \{ 3, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101 \}
  Typical Range of Values : 3 \leq MaskWidth \leq 3
  Minimal Value Step : 2
  Recommended Value Step : 2
  Restriction : odd

HALCON 6.0
**MaskHeight** (input control) .......................................................... extent.y ~ long
Height of filter mask.

Default Value : 9

Value Suggestions : MaskHeight ∈ \{3, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101\}

Typical Range of Values : 3 ≤ MaskHeight ≤ 3

Minimal Value Step : 2

Recommended Value Step : 2

Restriction : add

Example

read_image(Image,’fabrik’)
mean_image(Image,Mean,3,3)
disp_image(Mean,WindowHandle).

If the parameter values are correct the operator MeanImage returns the value TRUE.
The behavior in case of empty input (no input images available) is set via the operator
SetSystem(’noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

MeanImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successor Functions

DynThreshold, RegionGrowing

See Also

AnisotropeDiff, SigmaImage, ConvolveImage, GenLowpass

Alternatives

GaussImage, SmoothImage

Module

Image filters

```c
[out] HImageX ImageMean HImageX.MeanN ()

void HOperatorSetX.MeanN ([in] IHObjectX Image,
[out] HUntypedObjectX ImageMean )
```

Average gray values over several channels.

The operator MeanN generates the pixel-by-pixel mean value of all channels. For each coordinate point the sum of all gray values at this coordinate is calculated. The result is the mean of the gray values (sum divided by the number of channels). The output image has one channel.

Parameter

- **Image** (input iconic) .......... multichannel-image-array ~ HImageX / IHObjectX (byte, real)
  Multichannel gray image.
- **ImageMean** (output iconic) ............... image(-array) ~ HImageX / HUntypedObjectX (byte, real)
  Result of averaging.

MeanN is reentrant and automatically parallelized (on tuple level, domain level).

Possible Predecessor Functions

Compose2, Compose3, Compose4, AddChannels

Possible Successor Functions

DispImage

See Also

CountChannels

Module

Image filters
Suppose salt and pepper noise.

The operator MeanSp carries out a smoothing by averaging the values. Only the gray values within the interval from MinThresh to MaxThresh are averaged. Gray values which are too light or too dark are ignored during summation. If no gray value lies within the default interval during summation the original gray value is adopted. If the thresholds are set at 0 or 255, respectively, the operator MeanSp behaves like MeanImage except for the running time.

The operator MeanSp is used to suppress extreme gray values (salt and pepper noise = white and black dots).

Attention

If even values instead of odd values are given for MaskHeight or MaskWidth, the routine uses the next larger odd values instead (this way the center of the filter mask is always explicitly determined).

Parameter

- **Image** (input iconic) ..................... (multichannel-)image(-array) \(\Rightarrow\) HImageX / IObjectX (byte)
  Input image.

- **ImageSPMean** (output iconic) ....(multichannel-)image(-array) \(\Rightarrow\) HImageX / HUntypedObjectX (byte)
  Smoothed image.

- **MaskWidth** (input control) ...................................................... extent.x \(\sim\) long
  Width of filter mask.

  Default Value: 3
  Value Suggestions: MaskWidth \(\in\) \{3, 5, 7, 9, 11\}
  Typical Range of Values: \(3 \leq\) MaskWidth \(\leq\) 3
  Minimal Value Step: 2
  Recommended Value Step: 2
  Restriction: odd

- **MaskHeight** (input control) ...................................................... extent.y \(\sim\) long
  Height of filter mask.

  Default Value: 3
  Value Suggestions: MaskHeight \(\in\) \{3, 5, 7, 9, 11\}
  Typical Range of Values: \(3 \leq\) MaskHeight \(\leq\) 3
  Minimal Value Step: 2
  Recommended Value Step: 2
  Restriction: odd

- **MinThresh** (input control) ...................................................... integer \(\sim\) long
  Minimum gray value.

  Default Value: 1
  Value Suggestions: MinThresh \(\in\) \{1, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101\}
  Typical Range of Values: \(0 \leq\) MinThresh \(\leq\) 0
  Minimal Value Step: 1
  Recommended Value Step: 1

- **MaxThresh** (input control) ...................................................... integer \(\sim\) long
  Maximum gray value.

  Default Value: 254
  Value Suggestions: MaxThresh \(\in\) \{5, 7, 9, 11, 15, 23, 31, 43, 61, 101, 200, 230, 250, 254\}
  Typical Range of Values: \(0 \leq\) MaxThresh \(\leq\) 0
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: (MinThresh \(\leq\) MaxThresh)

Example

```c
read_image(Image,’meer_rot’)
```

HALCON 6.0
disp_image(Image,WindowHandle)
mean_sp(Image,ImageMeansp,3,3,101,201)
disp_image(ImageMeansp,WindowHandle).

**Parallelization Information**
MeanSp is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Successor Functions**

**DispImage**
See Also
AnisotropeDiff, SigmaImage, GaussImage, SmoothImage, EliminateMinMax
Alternatives
MeanImage, MedianImage, MedianSeparate, EliminateMinMax

**Module**

```
[out] HImageX ImageMedian HImageX.MedianImage ([in] String MaskType,

void HOperatorSetX.MedianImage ([in] IHObjectX Image,
[out] HUntypedObjectX ImageMedian, [in] String MaskType, [in] long Radius,
[in] long Margin )
```

Median filtering with different rank masks.
The operator MedianImage carries out a non-linear smoothing of the gray values of all input images (Image).
The shift mask (MaskType) is transmitted in the form of an object (more precisely: its region). Several margin controls can be chosen for filtering (Margin):

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0...255</td>
<td>Pixels outside of the image edges are assumed constant (with the indicated gray value).</td>
</tr>
<tr>
<td>-1</td>
<td>Continuation of edge pixels.</td>
</tr>
<tr>
<td>-2</td>
<td>Cyclic continuation of image edges.</td>
</tr>
<tr>
<td>-3</td>
<td>Reflection of pixels at the image edges.</td>
</tr>
</tbody>
</table>

The indicated mask (= region of the mask image) is put over the image to be filtered in such a way that the center of the mask touches all pixels of the objects once. For each of these pixels all neighboring pixels covered by the mask are sorted in an ascending sequence according to their gray values. Thus, each of these sorted gray value sequences contains exactly as many gray values as the mask has pixels. From these sequences the median is selected and entered as resulting gray value at the corresponding output image.

**Parameter**

- **Image** (input iconic) .........................(multichannel-)image-array) \(\sim\) HImageX / IHObjectX (byte)
  Image to be filtered.
- **ImageMedian** (output iconic) ............(multichannel-)image-array) \(\sim\) HImageX / HUntypedObjectX (byte)
  Median filtered image.
- **MaskType** (input control) ..............string \(\sim\) String
  Type of median mask.
  Default Value : 'circle'
  Value List : MaskType \(\in\) {'circle', 'rectangle'}
- **Radius** (input control) ..................integer \(\sim\) long
  Radius of median mask.
  Default Value : 1
  Value Suggestions : Radius \(\in\) { 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 15, 19, 25, 31, 39, 47, 59 }
  Typical Range of Values : \[ 1 \leq \text{Radius} \leq 1 \]
  Minimal Value Step : 1
  Recommended Value Step : 2
3.12. SMOOTHING

Margin (input control) .............................................................. integer ≈ long
Margin control: 0...255 (constant), -1 (edge pixels continued), -2 (cyclic continuation), -3 (reflection).
Default Value: -3
Typical Range of Values: -3 ≤ Margin ≤ -3

Example

read_image(Image,’fabrik’)
median_image(Image,Median,’circle’,3,-1)
disp_image(MedianWeighted,WindowHandle).

Result
If the parameter values are correct the operator MedianImage returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem(‘noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information
MedianImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessor Functions
ReadImage

Possible Successor Functions
Threshold, DynThreshold, RegionGrowing

See Also
GenCircle, GenRectangle1, GrayErosionRect, GrayDilationRect

Alternatives
RankImage

Bibliography

Module

Image filters

Separated median filtering with rectangle masks.
The operator MedianSeparate carries out a variation of the median filtering: First two auxiliary images are created. The first one originates from a median filtering with a horizontal mask with a height of one pixel and the width MaskWidth followed by filtering with a mask with the height MaskHeight. The second auxiliary image is created by filtering with the same masks, but with a reversed sequence of the operation: first the vertical, then the horizontal mask. The output image results from averaging the two auxiliary images pixel by pixel.
The operator MedianSeparate is clearly faster than the normal operator MedianImage because both masks are one pixel wide, facilitating a very efficient processing. The runtime is practically independent of the size of the mask. For example, the operator MedianSeparate can be well used after texture filters, where large masks are needed.
The filter can also be used several times in a row in order to enhance the smoothing.

Parameter

Image (input iconic) ................. (multichannel-)image(-array) ≈ HImageX / IObjectX (byte)
Image to be filtered.

ImageSMedian (output iconic) ... (multichannel-)image(-array) ≈ HImageX / HUnTypedObjectX (byte)
Median filtered image.

HALCON 6.0
**MaskWidth** (input control) ..........................................................\( \text{extent.x} \sim \text{long} \\
\text{Width of rank mask.} \\
\text{Default Value:} 25 \\
\text{Value Suggestions:} \text{MaskWidth} \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 27, 43, 51, 67, 91, 121, 151\} \\
\text{Typical Range of Values:} 1 \leq \text{MaskWidth} \leq 1 \\
\text{Minimal Value Step:} 2 \\
\text{Recommended Value Step:} 2

**MaskHeight** (input control) ..........................................................\( \text{extent.y} \sim \text{long} \\
\text{Height of rank mask.} \\
\text{Default Value:} 25 \\
\text{Value Suggestions:} \text{MaskHeight} \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 27, 43, 51, 67, 91, 121, 151\} \\
\text{Typical Range of Values:} 1 \leq \text{MaskHeight} \leq 1 \\
\text{Minimal Value Step:} 2 \\
\text{Recommended Value Step:} 2

**Margin** (input control) ..........................................................\( \text{integer} \sim \text{long} \\
\text{<no description available>} \\
\text{Default Value:} -3 \\
\text{Typical Range of Values:} -3 \leq \text{Margin} \leq -3 \\
\text{Minimal Value Step:} 1 \\
\text{Recommended Value Step:} 1

---

**Example**

```python
read_image(Image,'fabrik')
median_separate(Image,MedianSeparate,5,5,3)
disp_image(MedianSeparate,WindowHandle).
```

---

**Parallelization Information**

MedianSeparate is *reentrant* and automatically *parallelized* (on *tuple level, channel level*).

---

**Possible Predecessor Functions**

TextureLaws, SobelAmp, DeviationImage

---

**Possible Successor Functions**

LearnNdimNorm, LearnNdimBox, MedianSeparate, RegionGrowing, AutoThreshold

---

**See Also**

RankImage

---

**Alternatives**

MedianImage

---

**Bibliography**


---

**Module**

Image filters

```c
[out] HImageX ImageWMedian HImageX.MedianWeighted ([in] String MaskType, [in] long MaskSize )
```

Weighted median filtering with different rank masks.

The operator MedianWeighted calculates the median of the gray values within a local environment. In contrast to MedianImage, which uses all gray values within the environment exactly once, the operator MedianWeighted weights all gray values several times depending on their position. A gray value is received into the field to be sorted several times according to its weighting. The following masks are available:
3.12. SMOOTHING

'gauss' (MaskSize = 3)

1 2 1
2 4 2
1 2 1

'inner' (MaskSize = 3)

1 1 1
1 3 1
1 1 1

The operator MedianWeighted means that, contrary to MedianImage, gray value corners remain.

Parameter

- **Image** (input iconic) ............... (multichannel-image-array) \( \sim HImageX / IHObjectX(\text{byte, int2}) \)
  Image to be filtered.

- **ImageWMedian** (output iconic) (multichannel-image-array) \( \sim HImageX / HUntypedObjectX(\text{byte, int2}) \)
  Median filtered image.

- **MaskType** (input control) .......................................................... string \( \sim \text{String} \)
  Type of median mask.
  Default Value: 'inner'
  Value List: MaskType \( \in \{ 'inner', 'gauss' \} \)

- **MaskSize** (input control) ......................................................... integer \( \sim \text{long} \)
  Mask size.
  Default Value: 3
  Value List: MaskSize \( \in \{ 3 \} \)

Example

read_image(Image,‘fabrik’)
median_weighted(Image,MedianWeighted,‘gauss’,3)
disp_image(MedianWeighted,WindowHandle).

Parallelization Information

MedianWeighted is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessor Functions

ReadImage

Possible Successor Functions

Threshold, DynThreshold, RegionGrowing

Alternatives

MedianImage, TrimmedMean, SigmaImage

Bibliography


Module

Image filters

```h
[out] HImageX ImageMidrange HImageX.MidrangeImage ([in] HRegionX Mask,
[in] long Margin)
```

```c
void HOperatorSetX.MidrangeImage ([in] IHObjectX Image,
```

Calculate the average of maximum and minimum inside any mask.

The operator MidrangeImage forms the average of maximum and minimum inside the indicated mask in the whole image. Several margin controls (Margin) can be chosen for filtering:
Pixels outside of the image edges are assumed constant (with the indicated gray value).
-1 Continuation of edge pixels.
-2 Cyclic continuation of image edges.
-3 Reflection of pixels at the image edges.

The indicated mask (= region of the mask image) is put over the image to be filtered in such a way that the center of the mask touches all pixels once.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Image (input iconic)</td>
<td>(multichannel-)image-array ( \leftarrow ) HImageX / IObjectX (byte)</td>
</tr>
<tr>
<td>▶ Mask (input iconic)</td>
<td>region ( \leftarrow ) HRegionX / IObjectX (byte) Region serving as filter mask.</td>
</tr>
<tr>
<td>▶ ImageMidrange (output iconic)</td>
<td>multichannel-image-array ( \leftarrow ) HImageX / HUntypedObjectX Filtered output image.</td>
</tr>
<tr>
<td>▶ Margin (input control)</td>
<td>integer ( \leftarrow ) long Margin control: 0...255 (constant), -1 (edge pixels continued), -2 (cyclic continuation), -3 (reflection).</td>
</tr>
</tbody>
</table>

Default Value: -3
Typical Range of Values: \(-3 \leq \text{Margin} \leq -3\)

Example

```plaintext
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
midrange_image(Image,Region,Midrange,-3)
disp_image(Midrange,WindowHandle).
```

If the parameter values are correct the operator MidrangeImage returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem('noObjectResult',<Result>). If necessary an exception handling is raised.

Parallelization Information
MidrangeImage is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessor Functions
ReadImage, DrawRegion, GenCircle, GenRectangle1

Possible Successor Functions
Threshold, DynThreshold, RegionGrowing

See Also
GenCircle, GenRectangle1, GrayErosionRect, GrayDilationRect, GrayRangeRect

Alternatives
SigmaImage

Bibliography
R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 319

Image filters

```plaintext
[out] HImageX ImageRank HImageX.RankImage ([in] HRegionX Mask,
[in] long Rank, [in] long Margin )

void HOperatorSetX.RankImage ([in] IObjectX Image,
[in] IObjectX Mask, [out] HUntypedObjectX ImageRank, [in] long Rank,
[in] long Margin )
```

Smooth an image with an arbitrary rank mask.
The operator \texttt{RankImage} carries out a non-linear smoothing of the gray values of all input images (\texttt{Image}). The shift mask (\texttt{Mask}) is transmitted in the form of a region. The rank gray value (\texttt{Rank}) within the shift mask is calculated for all pixels. Several margin controls can be chosen for filtering (\texttt{Margin}): 

- 0...255 Pixels outside of the image edges are assumed constant (with the indicated gray value).
- -1 Continuation of edge pixels.
- -2 Cyclic continuation of image edges.
- -3 Reflection of pixels at the image edges.

The indicated mask is put over the image to be filtered in such a way that the center of the mask touches all pixels once. For each of these pixels all neighboring pixels covered by the mask are sorted in an ascending sequence according to their gray values. Thus, each of these sorted gray value sequences contains exactly as many gray values as the mask has pixels. From these sequences the n-largest element (= \texttt{Rank}) is selected and entered as resulting gray value at the corresponding output image.

If \texttt{Rank} is set to the median the well known median filter is applied. For \texttt{Mask} images can be created e.g., via the operators \texttt{GenCircle} or \texttt{GenRectangle1}. If \texttt{Rank} is applied to 1 or the area of \texttt{Mask}, respectively, the effect is the same as in case of \texttt{GrayErosionRect} or \texttt{GrayDilationRect}.

\begin{verbatim}
| Parameter |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| \texttt{Image} \hspace{2cm} (input iconic) \hspace{2cm} multichannel-image-array \hspace{2cm} \sim \hspace{2cm} HImageX \hspace{2cm} / \hspace{2cm} IObjectX (byte) | \hspace{2cm} \hspace{2cm} Image to be filtered. |
| \texttt{Mask} \hspace{2cm} (input iconic) \hspace{2cm} \hspace{2cm} region \hspace{2cm} \sim \hspace{2cm} HRegionX \hspace{2cm} / \hspace{2cm} IObjectX (byte) | \hspace{2cm} \hspace{2cm} Region serving as filter mask. |
| \texttt{ImageRank} \hspace{2cm} (output iconic) \hspace{2cm} \hspace{2cm} multichannel-image-array \hspace{2cm} \sim \hspace{2cm} HImageX \hspace{2cm} / \hspace{2cm} HUntypedObjectX | \hspace{2cm} \hspace{2cm} Filtered image. |
| \texttt{Rank} \hspace{2cm} (input control) \hspace{2cm} \hspace{2cm} integer \hspace{2cm} \sim \hspace{2cm} \hspace{2cm} long \hspace{2cm} | \hspace{2cm} Rank of the output gray value in the sorted sequence of input gray values inside the filter mask. Typical value (median): area(mask) / 2. |
| \texttt{Margin} \hspace{2cm} (input control) \hspace{2cm} \hspace{2cm} integer \hspace{2cm} \sim \hspace{2cm} \hspace{2cm} long \hspace{2cm} | \hspace{2cm} Margin control: 0...255 (constant), -1 (edge pixels continued), -2 (cyclic continuation), -3 (reflection). |

\texttt{read_image(Image,'fabrik')} \hspace{2cm} \texttt{draw_region(Region,WindowHandle)} \hspace{2cm} \texttt{rank_image(Image,Region,ImageRank,5,-3)} \hspace{2cm} \texttt{disp_image(ImageRank,WindowHandle)}. 
\end{verbatim}

\begin{verbatim}
| Result |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| If the parameter values are correct the operator \texttt{RankImage} returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem('noObjectResult',<Result>).} If necessary an exception handling is raised. |

\begin{verbatim}
| Parallelization Information |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| \texttt{RankImage} is reentrant and automatically parallelized (on tuple level, channel level). |

| Possible Predecessor Functions |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| \texttt{ReadImage}, \texttt{DrawRegion}, \texttt{GenCircle}, \texttt{GenRectangle1} |

| Possible Successor Functions |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| \texttt{Threshold}, \texttt{DynThreshold}, \texttt{RegionGrowing} |
\end{verbatim}

HALCON 6.0
Non-linear smoothing with the sigma filter.

The operator SigmaImage carries out a non-linear smoothing of the gray values of all input images (Image). All pixels are checked in a rectangular window (MaskHeight × MaskWidth). All pixels of the window which differ from the current pixel by less than Sigma are used for calculating the new pixel, which is the average of the chosen pixels. If all differences are larger than Sigma the gray value is adapted unchanged.

Attention

The filter is implemented for images of the 'byte' type only. If even values instead of odd values are given for MaskHeight or MaskWidth, the routine uses the next larger odd values instead (this way the center of the filter mask is always explicitly determined).

Parameter

- **Image** (input iconic) .. (multichannel-)image(array) ~ HImageX / IObjectX (byte, int1, int2, int4, real)
  
  Image to be smoothed.

- **ImageSigma** (output iconic) .. (multichannel-)image(array) ~ HImageX / HUntypedObjectX (byte, int1, int2, int4, real)
  
  Smoothed image.

- **MaskHeight** (input control) .......................................................... extent.y ~ long
  
  Height of the mask (number of lines).

  Default Value : 5

  Value Suggestions : MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15}

  Typical Range of Values : 3 ≤ MaskHeight ≤ 3

  Minimal Value Step : 2

  Recommended Value Step : 2

  Restriction : odd

- **MaskWidth** (input control) .......................................................... extent.x ~ long
  
  Width of the mask (number of columns).

  Default Value : 5

  Value Suggestions : MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}

  Typical Range of Values : 3 ≤ MaskWidth ≤ 3

  Minimal Value Step : 2

  Recommended Value Step : 2

  Restriction : odd

- **Sigma** (input control) .......................................................... integer ~ long
  
  Max. deviation to the average.

  Default Value : 3

  Value Suggestions : Sigma ∈ {3, 5, 7, 9, 11, 20, 30, 50}

  Typical Range of Values : 0 ≤ Sigma ≤ 0

  Minimal Value Step : 1

  Recommended Value Step : 2
3.12. SMOOTHING

Example

read_image(Image, 'fabrik')
sigma_image(Image, ImageSigma, 5, 5, 3)
disp_image(ImageSigma, WindowHandle).

Result

If the parameter values are correct the operator SigmaImage returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem('noObjectResult', <Result>). If necessary an exception handling is raised.

Parallelization Information

SigmaImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessor Functions

ReadImage

Possible Successor Functions

Threshold, DynThreshold, RegionGrowing

See Also

SmoothImage, GaussImage, MeanImage

Alternatives

AnisotropeDiff, RankImage

Bibliography

R. Haralick, L. Shapiro; "Computer and Robot Vision"; Addison-Wesley, 1992, Seite 325

Module

Image filters

Smooth an image using recursive filters.

SmoothImage smooths gray images using recursive filters originally developed by Deriche and Shen and using the non-recursive Gaussian filter. The following filters can be choosen via the parameter Filter:

'deriche1', 'deriche2', 'shen' and 'gauss'.

The "filter width" (i.e., the range of the filter and thereby result of the filter) can be of any size. In the case that the Deriche or Shen is choosen it decreases by increasing the filter parameter Alpha and increases in the case of the Gauss filter (and Alpha corresponds to the standard deviation of the Gaussian function). An approximation of the appropriate size of the filterwidth Alpha is performed by the operator InfoSmooth.

Non-recursive filters like the Gaussian filter are often implemented using filter-masks. In this case the runtime of the operator increases with increasing size of the filter mask. The runtime of the recursive filters remains constant; except the margin control becomes a little bit more time consuming. The Gaussian filter becomes slow in comparison to the recursive ones but is in contrast to them isotropic (the filter 'deriche2' is only weakly direction sensitive). A comparable result of the smoothing is achieved by choosing the following values for the parameter:

\[
\begin{align*}
\text{Alpha('deriche2')} &= \frac{\text{Alpha('deriche1')}}{2} \\
\text{Alpha('shen')} &= \frac{\text{Alpha('deriche1')}}{2} \\
\text{Alpha('gauss')} &= \frac{1.77}{\text{Alpha('deriche1')}}
\end{align*}
\]
Filtering

- **Image**: (input iconic) Image to be smoothed.
- **ImageSmooth**: (output iconic) Smoothed image.
- **Filter**: (input control) Filter. Default Value: 'deriche2'. Value List: 'deriche1', 'deriche2', 'shen', 'gauss'.
- **Alpha**: (input control) Filter parameter: small values cause strong smoothing (vice versa by using 'gauss'). Default Value: 0.5. Value Suggestions: 0.1, 0.2, 0.3, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0, 10.0.

**Example**

```plaintext```
info_smooth('deriche2', 0.5, Size, Coeffs)
smooth_image(Input, Smooth, 'deriche2', 7)
```

**Result**

If the parameter values are correct, the operator `SmoothImage` returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator `SetSystem('noObjectResult',<Result>)`. If necessary an exception handling is raised.

**Parallelization Information**

`SmoothImage` is reentrant and automatically parallelized (on tuple level, channel level).

**Possible Predecessor Functions**

ReadImage, Threshold, DynThreshold, RegionGrowing

**Possible Successor Functions**

InfoSmooth, MedianImage, SigmaImage, AnisotropeDiff

**See Also**

GaussImage, MeanImage, DerivateGauss

**Alternatives**

**Bibliography**


**Module**

Image filters

```
[out] HImageX ImageTMean
HImageX.TrimmedMean ([in] HRegionX Mask,
[in] long Number, [in] long Margin )
```

```
void HOperatorSetX.TrimmedMean ([in] HObjectX Image,
[in] HObjectX Mask, [out] HUntypedObjectX ImageTMean, [in] long Number,
[in] long Margin )
```

Smooth an image with an arbitrary rank mask.

The operator `TrimmedMean` carries out a non-linear smoothing of the gray values of all input images (`Image`). The shift mask (`Mask`) is transmitted in the form of a region. The average of `Number` gray values located near the median is calculated. Several margin controls can be chosen for filtering (`Margin`).

3.12. SMOOTHING

Pixels outside of the image edges are assumed constant (with the indicated gray value).
-1 Continuation of edge pixels.
-2 Cyclic continuation of image edges.
-3 Reflection of pixels at the image edges.

The indicated mask (= region of the mask image) is put over the image to be filtered in such a way that the center of the mask touches all pixels once. For each of these pixels all neighboring pixels covered by the mask are sorted in an ascending sequence according to their gray values. Thus, each of these sorted gray value sequences contains exactly as many gray values as the mask has pixels. If \( F \) is the surface of the mask the average of these sequences is calculated as follows: The first \( (F - \text{Number})/2 \) gray values are ignored. Then the following \( \text{Number} \) gray values are summed up and divided by \( \text{Number} \). Again the remaining \( (F - \text{Number})/2 \) gray values are ignored.

Parameter

- **Image** (input iconic) …………………. multichannel-image-array \( \sim \) HImageX / HObjectX (byte)
  Image to be filtered.
- **Mask** (input iconic) ………………………………………………………………… region \( \sim \) HRegionX / HObjectX
  Image whose region serves as filter mask.
- **ImageMean** (output iconic) ……… multichannel-image-array \( \sim \) HImageX / HUntypedObjectX (byte)
  Filtered output image.
- **Number** (input control) …………………………………………………………………………… integer \( \sim \) long
  Number of averaged pixels. Typical value: Surface(Mask) / 2.
  Default Value: 5
  Value Suggestions: \( \text{Number} \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31\} \)
  Typical Range of Values: \( 1 \leq \text{Number} \leq 1 \)
  Minimal Value Step: 1
  Recommended Value Step: 2
- **Margin** (input control) …………………………………………………………………………… integer \( \sim \) long
  Margin control: 0…255 (constant), -1 (edge pixels continued), -2 (cyclic continuation), -3 (reflection).
  Default Value: -3
  Typical Range of Values: \( -3 \leq \text{Margin} \leq -3 \)

Example

```plaintext
read_image(Image,‘fabrik’)
draw_region(Region,WindowHandle)
trimmed_mean(Image,Region,TrimmedMean,5,-3)
disp_image(TrimmedMean,WindowHandle).
```

Result

If the parameter values are correct the operator TrimmedMean returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem(‘noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

TrimmedMean is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessor Functions

ReadImage, DrawRegion, GenCircle, GenRectangle1

Possible Successor Functions

Threshold, DynThreshold, RegionGrowing

See Also

GenCircle, GenRectangle1, GrayErosionRect, GrayDilationRect

Alternatives

SigmaImage, MedianWeighted, MedianImage

Bibliography

R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 320

Module

Image filters
### 3.13 Texture

Calculate the standard deviation of gray values within rectangular windows.

\[ \text{DeviationImage} \text{ calculates the standard deviation of gray values in the image } \text{Image} \text{ within a rectangular mask of size } (\text{Height}, \text{Width}). \text{ The resulting image is returned in } \text{ImageDeviation}. \text{ For byte images, the result is multiplied by 2. If the parameters } \text{Height} \text{ and } \text{Width} \text{ are even, they are changed to the next larger odd value. At the image borders the gray values are mirrored.} \]

**Parameter**

- **Image** (input iconic) ……… (multichannel-)image-array \(\sim HImageX / IHOBJECTX (byte, int4, real, int2)\)  
  Image for which the standard deviation is to be calculated.
- **ImageDeviation** (output iconic) .image-array \(\sim HImageX / HUntypedObjectX (byte, int4, real, int2)\)  
  Image containing the standard deviation.
- **Width** (input control) …………………………………………………………………………………….. extent.x \(\sim long\)  
  Width of the mask in which the standard deviation is calculated.  
  **Default Value**: 11  
  **Value List**: Width \(\in\) \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25\}  
  **Restriction**: (Width \& odd)  
- **Height** (input control) …………………………………………………………………………………….. extent.y \(\sim long\)  
  Height of the mask in which the standard deviation is calculated.  
  **Default Value**: 11  
  **Value List**: Height \(\in\) \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25\}  
  **Restriction**: (Height \& odd)

**Example**

```plaintext
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
deviation_image(Image,Deviation,9,9)
disp_image(Deviation,WindowHandle).
```

**Result**

\(\text{DeviationImage}\) returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via \(\text{SetSystem}('\text{noObjectResult'},<\text{Result}>).\) If necessary, an exception handling is raised.

**Parallelization Information**

\(\text{DeviationImage}\) is reentrant and automatically parallelized (on tuple level, channel level).

**Possible Successor Functions**

- DispImage
- ConvolutionImage, TextureLaws, Intensity
- EntropyImage, EntropyGray

**Module**

Image filters

Calculate the entropy of gray values within a rectangular window.
EntropyImage calculates the entropy of gray values in the image Image within a rectangular mask of size (Height, Width). The resulting image is returned in ImageEntropy, in which the entropy is multiplied by 32. If the parameters Height and Width are even, they are changed to the next larger odd value. At the image borders the gray values are mirrored.

Parameter

- **Image** (input iconic) ........................................... (multichannel-)image(-array) \(\sim\) HImageX / HObjectX (byte )
  Image for which the entropy is to be calculated.

- **ImageEntropy** (output iconic) ... (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte )
  Entropy image.

- **Width** (input control) .......................................................... extent.x \(\sim\) long
  Width of the mask in which the entropy is calculated.
  Default Value : 9
  Value Suggestions : Width \(\in\) \{3, 5, 7, 9, 11, 13, 15\}
  Value List : Height \(\in\) \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25\}
  Restriction : (Width \(\&\) odd)

- **Height** (input control) .......................................................... extent.y \(\sim\) long
  Height of the mask in which the entropy is calculated.
  Default Value : 9
  Value Suggestions : Height \(\in\) \{3, 5, 7, 9, 11, 13, 15\}
  Value List : Height \(\in\) \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25\}
  Restriction : (Height \(\&\) odd)

Example

read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
entropy_image(Image,Entropy1,9,9)
disp_image(Entropy1,WindowHandle).

Result

EntropyImage returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

EntropyImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successor Functions

DispImage

See Also

EnergyGabor, EntropyGray

Alternatives

EntropyGray

Module

Image filters

Filter an image using a Laws texture filter.

TextureLaws applies one or more texture transformations (according to Laws) to an image. This is done by convolving the input image with one (or more) filter masks. The filters are:

9 different 3x3 matrices obtainable from the following three vectors:
1 = \begin{bmatrix} 1 & 2 & 1 \\ -1 & 0 & 1 \\ -1 & 2 & -1 \end{bmatrix}

e = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ -1 & -2 & 0 & 2 & 1 \\ -1 & 0 & 2 & 0 & -1 \end{bmatrix}

25 different 5x5 matrices obtainable from the following five vectors:

1 = \begin{bmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ -1 & -4 & -5 & 0 & 5 & 4 & 1 \\ -1 & -2 & 1 & 4 & 1 & -2 & -1 \end{bmatrix}

e = \begin{bmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \\ -1 & -4 & -5 & 0 & 5 & 4 & 1 \\ -1 & -2 & 1 & 4 & 1 & -2 & -1 \end{bmatrix}

36 different 7x7 matrices obtainable from the following six vectors:

For most of the filters the resulting gray values must be modified by a Shift. This makes the different textures in the output image more comparable to each other, provided suitable filters are used.

The name of the filter is composed of the letters of the two vectors used, where the first letter denotes convolution in the column direction while the second letter denotes convolution in the row direction.

If more than one filter type is passed, a multi-channel image is returned for each (single-channel) input image, with each channel corresponding to a particular filter.

Parameter

- **Image** (input iconic) .......................... (multichannel-)image(-array)  \( HImageX / IHObjectX \) (byte, int2)
  Images to which the texture transformation is to be applied.
- **ImageTexture** (output iconic) ... (multichannel-)image(-array)  \( HImageX / HUntypedObjectX \) (byte)
  Texture images.
- **FilterTypes** (input control) .......................... string(-array)  \( \text{VARIANT} \) (string)
  Desired filters (name or number).
  Default Value : 'el'
  Value Suggestions : FilterTypes \( \in \{ 'll', 'le', 'ls', 'lr', 'lw', 'lo', 'el', 'ee', 'es', 'er', 'ew', 'eo', 'sl', 'se',
  'ss', 'sr', 'sw', 'so', 'rl', 're', 'rs', 'rr', 'rw', 'ro', 'wl', 'we', 'ws', 'wr', 'wo', 'ol', 'oe', 'os', 'or', 'ow',
  'oo' \}
- **Shift** (input control) .......................... integer  \( \text{long} \)
  Shift to reduce the gray value dynamics.
  Default Value : 2
  Value List : Shift \( \in \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \} \)
- **FilterSize** (input control) .......................... integer  \( \text{long} \)
  Size of the filter kernel.
  Default Value : 5
  Value List : FilterSize \( \in \{ 3, 5, 7 \} \)

Example

```c
/* 2 dimensional pixel classification */
read_image(Image,'combine')
open_window(0,0,-1,-1,'root','visible',',',WindowHandle)
disp_image(Image,WindowHandle)
texture_laws(Image,Texture1,'es',2,5)
texture_laws(Image,Texture2,'le',2,5)
```
3.14 Wiener-Filter


Generate an impulse response of an uniform out-of-focus blurring.

GenPsfDefocus generates an impulse response (spatial domain) of an uniform out-of-focus blurring and writes it into an image of HALCON image type 'real'. Blurring specifies the extent of blurring by defining the "blur radius" (out-of-focus blurring maps each image pixel on a small circle with a radius of Blurring - specified in "number of pixels"). If specified less than zero, the absolute value of Blurring is used. The result image of GenPsfDefocus encloses an spatial domain impulse response of the specified blurring. Its representation presumes the origin in the upper left corner. This results in the following disposition of an \(N \times M\) sized image:

- first rectangle ("upper left"): (image coordinates \(x_b = 0..(N/2) - 1, y_b = 0..(M/2) - 1\))
  - conforms to the fourth quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \(x = 0..N/2\) and \(y = 0..M/2\)
- second rectangle ("upper right"): (image coordinates \(x_b = N/2..N - 1, y_b = 0..(M/2) - 1\))
  - conforms to the third quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \(x = -N/2.. -1\) and \(y = -1.. - M/2\)
This representation conforms to that of the impulse response parameter of the HALCON-operator \texttt{WienerFilter}. So one can use \texttt{GenPsfDefocus} to generate an impulse response for Wiener filtering.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default Value</th>
<th>Value Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Psf}</td>
<td>image</td>
<td>\texttt{HImageX/HUntypedObjectX(,real,)}</td>
<td>\texttt{HImageX/HUntypedObjectX(,real,)}</td>
</tr>
<tr>
<td>\texttt{PSFwidth}</td>
<td>integer</td>
<td>\texttt{256}</td>
<td>{128, 256, 512, 1024}</td>
</tr>
<tr>
<td>\texttt{PSFheight}</td>
<td>integer</td>
<td>\texttt{256}</td>
<td>{128, 256, 512, 1024}</td>
</tr>
<tr>
<td>\texttt{Blurring}</td>
<td>real</td>
<td>\texttt{5.0}</td>
<td>{1.0, 5.0, 10.0, 15.0, 18.0}</td>
</tr>
</tbody>
</table>

\texttt{GenPsfDefocus} returns TRUE if all parameters are correct.

**Parallelization Information**
\texttt{GenPsfDefocus} is \textit{reentrant} and processed without parallelization.

**Possible Predecessor Functions**
\texttt{SimulateMotion, GenPsfMotion}

**Possible Successor Functions**
\texttt{SimulateDefocus, WienerFilter, WienerFilterNi}

**See Also**
\texttt{SimulateDefocus, GenPsfMotion, SimulateMotion, WienerFilter, WienerFilterNi}

**Bibliography**

M. L"uckenhaus: "Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse"; Diplomarbeit; Technische Universität Munich, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

**Module**

```c
void \texttt{HImageX.GenPsfMotion} (\,[\,in\,] \texttt{long PSFwidth}, \,[\,in\,] \texttt{long PSFheight},
\,[\,in\,] \texttt{double Blurring}, \,[\,in\,] \texttt{long Angle}, \,[\,in\,] \texttt{long Type})
```

Generate an impulse response of a (linearly) motion blurring.

\texttt{GenPsfMotion} generates an impulse response (spatial domain) of a blurring caused by a relative motion between the object and the camera during exposure. The generated impulse response is output into an image of HALCON image type 'real'. \texttt{PSFwidth} and \texttt{PSFheight} define the width and height of the output image. The blurring motion moves along an even. \texttt{Angle} fixes its direction by specifying the angle between the motion direction and
3.14. WIENER-FILTER

the x-axis (anticlockwise, measured in degrees). To specify different velocity behaviour five PSF prototypes can be generated. Type switches between the following prototypes:

1. reverse ramp (crude model for acceleration)
2. reverse trapezoid (crude model for high acceleration)
3. square pulse (exact model for constant velocity), this is default
4. forward trapezoid (crude model for deceleration)
5. forward ramp (crude model for high deceleration)

The blurring affects all part of the image uniformly. Blurring controls the extent of blurring. It specifies the number of pixels (lying one after another) that are affected by the blurring. This number is determined by velocity of the motion and exposure time. If Blurring is a negative number, an adequate blurring in reverse direction is simulated. If Angle is a negative number, it is interpreted clockwise. If Angle exceeds 360 or falls below -360, it is transformed modulo(360) in an adequate number between [0..360] resp. [-360..0]. The result image of GenPsfMotion encloses an spatial domain impulse response of the specified blurring. Its representation conforms to the second quadrant of the Cartesian coordinate system, encloses values of the impulse response

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Psf</td>
<td>(output iconic) image $\sim$ HImageX / HUntypedObjectX (real) Impulse response of motion-blur.</td>
</tr>
<tr>
<td>PSFWidth</td>
<td>(input control) Width of impulse response image. Default Value: 256 Value Suggestions: PSFWidth $\in$ {128, 256, 512, 1024}</td>
</tr>
<tr>
<td>PSFHeight</td>
<td>(input control) Height of impulse response image. Default Value: 256 Value Suggestions: PSFHeight $\in$ {128, 256, 512, 1024}</td>
</tr>
<tr>
<td>Blurring</td>
<td>(input control) Degree of motion-blur. Default Value: 20.0 Value Suggestions: Blurring $\in$ {5.0, 10.0, 20.0, 30.0, 40.0}</td>
</tr>
<tr>
<td>Angle</td>
<td>(input control) Angle between direction of motion and x-axis (anticlockwise). Default Value: 0 Value Suggestions: Angle $\in$ {0, 45, 90, 180, 270}</td>
</tr>
<tr>
<td>Type</td>
<td>(input control) PSF prototype resp. type of motion. Default Value: 3 Value List: Type $\in$ {1, 2, 3, 4, 5}</td>
</tr>
</tbody>
</table>

This representation conforms to that of the impulse response parameter of the HALCON-operator WienerFilter. So one can use GenPsfMotion to generate an impulse response for Wiener filtering a motion blurred image.
GenPsfMotion returns TRUE if all parameters are correct.

**Parallelization Information**
GenPsfMotion is *reentrant* and processed *without* parallelization.

**Possible Predecessor Functions**
GenPsfMotion, SimulateDefocus, GenPsfDefocus

**Possible Successor Functions**
SimulateMotion, WienerFilter, WienerFilterNi

**See Also**
SimulateMotion, SimulateDefocus, GenPsfDefocus, WienerFilter, WienerFilterNi

**Bibliography**

M. L"uckenhaus:"‘Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse’”; Diplomarbeit; Technische Universit"at M"unchen, Institut f"ur Informatik; Lehrstuhl Prof. Radig; 1995.


Wiener filter

```plaintext
[out] HImageX DefocusedImage
[HImageX.SimulateDefocus (in double Blurring )]

void HOperatorSetX.SimulateDefocus ([in] IHObjectX Image, 
[out] HUntypedObjectX DefocusedImage, [in] double Blurring )
```

Simulate an uniform out-of-focus blurring of an image.

SimulateDefocus simulates out-of-focus blurring of an image. All parts of the image are blurred uniformly. Blurring specifies the extent of blurring by defining the "blur radius" (out-of-focus blurring maps each image pixel on a small circle with a radius of Blurring - specified in "number of pixels"). If specified less than null, the absolute value of Blurring is used. Simulation of blurring is done by a convolution of the image with a blurring specific impulse response. The convolution is realized by multiplication in the Fourier domain.

**Attention**
As the simulation of the blurring is realized based on the Fourier transform, image height and width must be a power of 2, e.g. 128, 128, 256, 512,....

**Parameter**

- **Image** (input iconic) ..........................image ~  HImageX / IHObjectX ( any ) Image to blur.
- **DefocusedImage** (output iconic) .................image ~  HImageX / HUntypedObjectX ( real ) Blurred image.
- **Blurring** (input control) ........................real ~  double Degree of blurring.
  
**Default Value :** 5.0

**Value Suggestions :** Blurring ∈ {1.0, 5.0, 10.0, 15.0, 18.0}

**Result**
SimulateDefocus returns TRUE if all parameters are correct. If the input is empty SimulateDefocus returns with an error message.

**Parallelization Information**
SimulateDefocus is *reentrant* and processed *without* parallelization.

**Possible Predecessor Functions**
GenPsfDefocus, SimulateMotion, GenPsfMotion

**Possible Successor Functions**
WienerFilter, WienerFilterNi

3.14. WIENER-FILTER

See Also

GenPsfDefocus, SimulateMotion, GenPsfMotion

Bibliography


M. L"uckenhaus:’Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse’; Diplomarbeit; Technische Universit"at M"unchen, Institut f"ur Informatik; Lehrstuhl Prof. Radig; 1995.

Module

Simulation of (linearly) motion blur.

SimulateMotion simulates blurring caused by a relative motion between the object and the camera during exposure. The simulated motion moves along an even. Angle fixes its direction by specifying the angle between the motion direction and the x-axis (anticlockwise, measured in degrees). Simulation is done by a convolution of the image with a blurring specific impulse response. The convolution is realized by multiplication in the Fourier domain. SimulateMotion offers five prototypes of impulse responses conforming to different acceleration behaviours. Type allows to choose one of the following PSF prototypes:

1. reverse ramp (crude model for acceleration)
2. reverse trapezoid (crude model for high acceleration)
3. square pulse (exact model for constant velocity), this is default
4. forward trapezoid (crude model for deceleration)
5. forward ramp (crude model for high deceleration)

The simulated blurring affects all part of the image uniformly. Blurring controls the extent of blurring. It specifies the number of pixels (lying one after another) that are affected by the blurring. This number is determined by velocity of the motion and exposure time. If Blurring is a negative number, an adequate blurring in reverse direction is simulated. If Angle is a negative number, it is interpreted clockwise. If Angle exceeds 360 or falls below -360, it is transformed modulo(360) in an adequate number between [0, 360) resp. [-360, 0].

Attention

As the simulation of the blurring is realized based on the Fourier transform, image height and width must be a power of 2, e.g. 128, 128, 256, 512, ...

Parameter

- **Image** (input iconic) ........................................... image \( \rightarrow \) HImageX / IHObjectX (any) image to be blurred.
- **MovedImage** (output iconic) ............................. image \( \rightarrow \) HImageX / HUntypedObjectX (real) motion blurred image.
- **Blurring** (input control) ................................. real \( \rightarrow \) double extent of blurring.
  Default Value: 20.0
  Value Suggestions: Blurring \( \in \{ 5.0, 10.0, 20.0, 30.0, 40.0 \} \)
- **Angle** (input control) ........................................ integer \( \rightarrow \) long Angle between direction of motion and x-axis (anticlockwise).
  Default Value: 0
  Value Suggestions: Angle \( \in \{ 0, 45, 90, 180, 270 \} \)
**Type** (input control) ................................................................. integer → long
impulse response of motion blur.

Default Value : 3

Value List : Type ∈ \{1, 2, 3, 4, 5\}

SimulateMotion returns TRUE if all parameters are correct. If the input is empty SimulateMotion returns with an error message.

### Parallelization Information

SimulateMotion is reentrant and processed without parallelization.

### Possible Predecessor Functions

GenPsfMotion, GenPsfMotion

### Possible Successor Functions

SimulateDefocus, WienerFilter, WienerFilterNi

See Also

GenPsfMotion, SimulateDefocus, GenPsfDefocus

### Bibliography


M. L"uckenhaus: "Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse"; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.


### Module

**Wiener filter**

```c
[out] HImageX RestoredImage HImageX.WienerFilter ([in] HImageX Psf,
[in] HImageX FilteredImage )

void HOperatorSetX.WienerFilter ([in] IHOBJECTX Image,
[in] IHOBJECTX Psf, [in] IHOBJECTX FilteredImage,
[out] HUNTypedObjectX RestoredImage )
```

Image restoration by Wiener filtering.

WienerFilter produces an estimate of the original image (= image without noise and blurring) by minimizing the mean square error between estimated and original image. WienerFilter can be used to restore images corrupted by noise and/or blurring (e.g. motion blur, atmospheric turbulence or out-of-focus blur). Method and realisation of this restoration technique bases on the following model: The corrupted image is interpreted as the output of a (disturbed) linear system. Functionality of a linear system is determined by its specific impulse response. So the convolution of original image and impulse response results in the corrupted image. The specific impulse response describes image acquisition and the occurred degradations. In the presence of additive noise an additional noise term must be considered. So the corrupted image can be modeled as the result of

\[
\text{convolution}(\text{impulse response, original image}) + \text{noise term}
\]

The noise term encloses two different terms describing image-dependent and image-independent noise. According to this model, two terms must be known for restoration by Wiener filtering:

1. degradation-specific impulse response
2. noise term

So WienerFilter needs a smoothed version of the input image to estimate the power spectral density of noise and original image. One can use one of the smoothing HALCON-filters (e.g. EliminateMinMax) to get this version. WienerFilter needs further the impulse response that describes the specific degradation. This impulse response (represented in spatial domain) must fit into an image of HALCON image type ‘real’. There exist two HALCON-operators for generation of an impulse response for motion blur and out-of-focus (see GenPsfMotion, GenPsfDefocus). The representation of the impulse response presumes the origin in the upper left corner. This results in the following disposition of an \(N\times M\) sized image:
As the Wiener filter is realized using the Fourier transform, only images with height and width fitting a power of 2 can be processed. **WienerFilter** works as follows:

- estimation of the power spectrum density of the original image by using the smoothed version of the corrupted image,
- estimation of the power spectrum density of each pixel by subtracting smoothed version from unsmoothed version,
- building the Wiener filter kernel with the quotient of power spectrum densities of noise and original image and with the impulse response,
- processing the convolution of image and Wiener filter frequency response.

The result image has got image type 'real'.

**Attention**  
Psf must be of image type 'real' and conform to Image and FilteredImage in image width and height.

**Parameter**

- **Image** (input iconic)  
Corrupted image.
- **Psf** (input iconic)  
impulse response (PSF) of degradation (in spatial domain).
- **FilteredImage** (input iconic)  
Smoothed version of corrupted image.
- **RestoredImage** (output iconic)  
Restored image.

**Result**  
**WienerFilter** returns TRUE if all parameters are correct. If the input is empty **WienerFilter** returns with an error message.

**Parallelization Information**  
**WienerFilter** is reentrant and processed without parallelization.

**Possible Predecessor Functions**

GenPsfMotion, SimulateMotion, SimulateDefocus, GenPsfDefocus

**See Also**

SimulateMotion, GenPsfMotion, SimulateDefocus, GenPsfDefocus

**Alternatives**

**WienerFilterNi**

**Bibliography**

M. L"uckenhaus: "Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse"; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995

**Module**

Wiener filter
Image restoration by Wiener filtering.

WienerFilterNi (ni = noise-estimation integrated) produces an estimate of the original image (= image without noise and blurring) by minimizing the mean square error between estimated and original image. WienerFilter can be used to restore images corrupted by noise and/or blurring (e.g. motion blur, atmospheric turbulence or out-of-focus blur). Method and realisation of this restoration technique bases on the following model: The corrupted image is interpreted as the output of a (disturbed) linear system. Functionality of a linear system is determined by its specific impulse response. So the convolution of original image and impulse response results in the corrupted image. The specific impulse response describes image acquisition and the occured degradations. In the presence of additive noise an additional noise term must be considered. So the corrupted image can be modeled as the result of

\[ \text{convolution(impulse\_response, original\_image)} + \text{noise\_term} \]

The noise term encloses two different terms describing image-dependent and image-independent noise. According to this model, two terms must be known for restoration by Wiener filtering:

1. degradation-specific impulse response
2. noise term

WienerFilterNi estimates the noise term as follows: The user defines a region that is suitable for noise estimation within the image (homogeneous as possible, as edges or textures aggravate noise estimation). After smoothing within this region by an (unweighted) median filter and subtracting smoothed version from unsmoothed, the average gray value within the region allows estimating the quotient of the power spectral densities of noise and original image (in contrast to WienerFilterWienerFilterNi assumes a rather constant quotient within the whole image). The user can define width and height of the rectangular (median-)filter mask to influence the noise estimation. WienerFilterNi needs further the impulse response that describes the specific degradation. This impulse response (represented in spatial domain) must fit into an image of HALCON image type 'real'. There exist two HALCON-operators for generation of an impulse response for motion blur and out-of-focus (see GenPsfMotion, GenPsfDefocus). The representation of the impulse response presumes the origin in the upper left corner. This results in the following disposition of an \(N \times M\) sized image:

- first rectangle ("upper left"): \( (\text{image coordinates } x = 0..(N/2) - 1, y = 0..(M/2) - 1) \)
  - conforms to the fourth quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \( x = 0..N/2 \) and \( y = 0.. - M/2 \)
- second rectangle ("upper right"): \( (\text{image coordinates } x = N/2..N - 1, y = 0..(M/2) - 1) \)
  - conforms to the third quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \( x = -N/2.. - 1 \) and \( y = 0.. - M/2 \)
- third rectangle ("lower left"): \( (\text{image coordinates } x = 0..(N/2) - 1, y = M/2..M - 1) \)
  - conforms to the first quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \( x = 1..N/2 \) and \( y = M/2.. - 1) \)
- fourth rectangle ("lower right"): \( (\text{image coordinates } x = N/2..N - 1, y = M/2..M - 1) \)
  - conforms to the second quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \( x = -N/2.. - 1 \) and \( y = M/2..1 \)

As the Wiener filter is realized using the Fourier transform, only images with height and width fitting a power of 2 can be processed. WienerFilter works as follows:

- estimating the quotient of the power spectrum densities of noise and original image,
- building the Wiener filter kernel with the quotient of power spectrum densities of noise and original image and with the impulse response,
- processing the convolution of image and Wiener filter frequency response.
The result image has got image type 'real'.

\[ \text{Attention} \]

\( \text{Psf} \) must be of image type 'real' and conform to \( \text{Image} \) in width and height. The Region used for noise estimation must lie completely within the image. If \( \text{MaskWidth} \) or \( \text{MaskHeight} \) is an even number, it is replaced by the next higher odd number (this allows the unique extraction of the center of the filter mask). Width/height of the mask may not exceed the image width/height or be less than null.

\[ \text{Parameter} \]

- \( \text{Image} \) (input iconic) \( \sim H\text{ImageX} / H\text{ObjectX} \) (any)
  Corrupted image.
- \( \text{Psf} \) (input iconic) \( \sim H\text{ImageX} / H\text{ObjectX} \) (real)
  Impulse response (PSF) of degradation (in spatial domain).
- \( \text{NoiseRegion} \) (input iconic) \( \sim H\text{RegionX} / H\text{ObjectX} \)
  Region for noise estimation.
- \( \text{RestoredImage} \) (output iconic) \( \sim H\text{ImageX} / H\text{UntypedObjectX} \) (real)
  Restored image.
- \( \text{MaskWidth} \) (input control) \( \sim \text{long} \)
  Width of filter mask.
  \[ \text{Default Value} : 3 \]
  \[ \text{Value Suggestions} : \text{MaskWidth} \in \{3, 5, 7, 9\} \]
  \[ \text{Typical Range of Values} : 0 \leq \text{MaskWidth} \leq 0 \]
- \( \text{MaskHeight} \) (input control) \( \sim \text{long} \)
  Height of filter mask.
  \[ \text{Default Value} : 3 \]
  \[ \text{Value Suggestions} : \text{MaskHeight} \in \{3, 5, 7, 9\} \]
  \[ \text{Typical Range of Values} : 0 \leq \text{MaskHeight} \leq 0 \]

\[ \text{Result} \]

\( \text{WienerFilterNi} \) returns TRUE if all parameters are correct. If the input is empty \( \text{WienerFilterNi} \) returns with an error message.

\[ \text{Parallelization Information} \]

\( \text{WienerFilterNi} \) is \textit{reentrant} and processed \textit{without} parallelization.

\[ \text{Possible Predecessor Functions} \]

\( \text{GenPsfMotion, SimulateMotion, SimulateDefocus, GenPsfDefocus} \)

\[ \text{See Also} \]

\( \text{SimulateMotion, GenPsfMotion, SimulateDefocus, GenPsfDefocus} \)

\[ \text{Alternatives} \]

\( \text{WienerFilter} \)

\[ \text{Bibliography} \]

M. L"uckenhaus: "Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse"; Diplomarbeit; Technische Universität M"unchen, Institut f"ur Informatik; Lehrstuhl Prof. Radig; 1995


\[ \text{Module} \]

Wiener filter
Chapter 4

Graphics

4.1 Drawing

Interactive moving of a region.

DragRegion1 is used to move a region on the display by mouse. Calling DragRegion1 turns the region visible as soon as the left mouse button is pressed. Therefore the region’s edges are displayed only. As representation mode the mode 'not' (see SetDraw) is used during procedure’s permanence. During the movement the cursor resides in the region’s barycenter. If you move the mouse with pressed left mouse button, the depicted region follows - delayed - this movement. If you press the right mouse button you terminate DragRegion1. The depicted region disappears from the display. Output is a region which corresponds to the last position on the display. You may pass even several regions at once. Procedure AffineTransImage moves the gray values.

Attention
Gray values of regions are not moved. With moving the input region it is not sure whether the gray values of the output regions are filled reasonable. This may occur if the gray values of the input regions do not comprise the whole image.

Parameter

- **SourceRegion** (input iconic) .......................... region ∼ HRegionX / IObjectX Regions to move.
- **DestinationRegion** (output iconic) ..................... region ∼ HRegionX / HUntypedObjectX Moved Regions.
- **WindowHandle** (input control) .......................... window ∼ HWindowX / long Window id.

Example

draw_region(Obj,WindowHandle)
drag_region1(Obj,New,WindowHandle)
disp_region(New,WindowHandle)
position(Obj,...,Row1,Column1,...,...,...)
position(New,...,Row2,Column2,...,...,...)
disp_arrow(WindowHandle,Row1,Column1,Row2,Column2,1.0)
fwrite_string(’Transformation: (’,Row2-Row1,’’,Column2-Column1,’)’)
fnew_line().
Result

DragRegion1 returns TRUE, if a region is entered, the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised. You may determine the behavior after an empty input with SetSystem(\#:‘noObjectResult’,<Result>);

Parallelization Information

DragRegion1 is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow

Possible Successor Functions

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

See Also

SetInsert, SetDraw, AffineTransImage

Alternatives

GetMposition, MoveRegion

Module

System

Interactive movement of a region with fixpoint specification.

You use DragRegion2 to move a region on the display by mouse. It corresponds to the procedure DragRegion1 with the difference, that the position of the mouse cursor can be determined.

Attention

Gray values of the regions are not moved. With moving the input region it is not sure whether the gray values of the output regions are filled reasonable. This may occur if the gray values of the input regions do not comprise the whole image.

Parameter

- **SourceRegion** (input iconic) ........................................ region \(\sim\) HRegionX / IHObjectX
  
  Regions to move.

- **DestinationRegion** (output iconic) .............................. region \(\sim\) HRegionX / HUntypedObjectX
  
  Moved regions.

- **WindowHandle** (input control) .................................. window \(\sim\) HWindowX / long

  Window id.

- **Row** (input control) .............................................. point.y \(\sim\) long

  Row index of the reference point.

  Default Value : 100

  Value Suggestions : Row \(\in\) \(\{0, 64, 128, 256, 512\}\)

  Typical Range of Values : \(0 \leq \text{Row} \leq 0\)

- **Column** (input control) ........................................... point.x \(\sim\) long

  Column index of the reference point.

  Default Value : 100

  Value Suggestions : Column \(\in\) \(\{0, 64, 128, 256, 512\}\)

  Typical Range of Values : \(0 \leq \text{Column} \leq 0\)

Result

DragRegion2 returns TRUE, if a region is entered, the window is valid and the needed drawing mode (see
SetInsert) is available. If necessary, an exception handling is raised. You may determine the behavior after an empty input with SetSystem(::'noObjectResult',<Result>:).

---

Parallelization Information

DragRegion2 is local and processed completely exclusively without parallelization.

---

Possible Predecessor Functions

OpenWindow

---

Possible Successor Functions

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert, AffineTransImage

---

See Also

SetInsert, SetDraw, AffineTransImage

---

Alternatives

GetMposition, MoveRegion, DragRegion1, DragRegion3

---

Module

System

---

Interactive movement of a region with restriction of positions.

You use DragRegion3 to move a region on the display by mouse. It corresponds to the procedure DragRegion2 with the enhancement, that all points are specified which can be entered by mouse. If you move the mouse outside of this area (MaskRegion), the region on the point with the smallest distance inside MaskRegion will be displayed.

---

Attention

The region’s gray values are not moved. With moving the input region it is not sure whether the gray values of the output regions are filled reasonable. This may occur if the gray values of the input regions do not comprise the whole image.

---

Parameter

- **SourceRegion** (input iconic) .......................... region  ~/ HRegionX / IHOBJECTX Regions to move.
- **MaskRegion** (input iconic) ........................... region  ~/ HRegionX / IHOBJECTX Points on which it is allowed for a region to move.
- **DestinationRegion** (output iconic) .................. region  ~/ HRegionX / HUntypedObjectX Moved regions.
- **WindowHandle** (input control) .......................... window  ~/ HWindowX / long Window_id.
- **Row** (input control) .................................. point.y  ~/ long Row index of the reference point.

**Default Value**: 100

**Value Suggestions**: Row ∈ {0, 64, 128, 256, 512}  
**Typical Range of Values**: 0 ≤ Row ≤ 0

---

HALCON 6.0
DragRegion3 returns TRUE, if a region is entered, if the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised. You may determine the behavior after an empty input with SetSystem(‘noObjectResult’,<Result>).

DragRegion3 is local and processed completely exclusively without parallelization.

Parallelization Information

Possible Predecessor Functions

Possible Successor Functions

See Also

Alternatives

Interactive drawing of a circle.

DrawCircle produces the parameter for a circle created interactive by the user in the window.

To create a circle you have to press the mouse button at the location which is used as the center of that circle. While keeping the mouse button pressed, the Radius’s length can be modified through moving the mouse. After another mouse click in the created circle center you can move it. A clicking close to the circular arc you can modify the Radius of the circle. Pressing the right mouse button terminates the procedure. After terminating the procedure the circle is not visible in the window any longer.

Parameter

| WindowHandle (input control) | window ~ HWindowX/long Window_id. |
| Row (output control)         | circle.center.y ~ double Barycenter’s row index. |
| Column (output control)      | circle.center.x ~ double Barycenter’s column index. |
| Radius (output control)      | circle.radius ~ double Circle’s radius. |

Example

read_image(Image,’affe’)
draw_circle(WindowHandle,Row,Column,Radius)
gen_circle(Circle,Row,Column,Radius,)
reduce_domain(Image,Circle,GrayCircle)
disp_image(GrayCircle,WindowHandle).
### 4.1. DRAWING

**Result**

DrawCircle returns TRUE if the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised.

**Parallelization Information**

DrawCircle is local and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

OpenWindow

**Possible Successor Functions**

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

**See Also**

GenCircle, DrawRectangle1, DrawRectangle2, DrawPolygon, SetInsert

**Alternatives**

DrawCircleMod, DrawEllipse, DrawRegion

**Module**

System

---

```plaintext
```

**Interactive drawing of a circle.**

DrawCircleMod produces the parameter for a circle created interactive by the user in the window.

To create a circle are expected the coordinates RowIn and ColumnIn of the center of a circle with radius RadiusIn. After another mouse click in the created circle center you can move it. A clicking close to the circular arc you can modify the Radius of the circle. Pressing the right mousebutton terminates the procedure. After terminating the procedure the circle is not visible in the window any longer.

**Parameter**

- **WindowHandle** (input control) ................................. window ∼ HWindowX/long Window.jd.
- **RowIn** (input control) .............................. circle.center.y ∼ double Row index of the center.
- **ColumnIn** (input control) .............................. circle.center.x ∼ double Column index of the center.
- **RadiusIn** (input control) ............................ circle.radius1 ∼ double Radius of the circle.
- **Row** (output control) ............................. circle.center.y ∼ double Barycenter’s row index.
- **Column** (output control) ............................ circle.center.x ∼ double Barycenter’s column index.
- **Radius** (output control) ............................... circle.radius ∼ double Circle’s radius.

**Example**

```plaintext
read_image(Image,’affe’)
draw_circle_mod(WindowHandle,20,20,15,Row,Column,Radius)
gen_circle(Circle,Row,Column,Radius,)
reduce_domain(Image,Circle,GrayCircle)
disp_image(GrayCircle,WindowHandle).
```
DRAWCIRCLEMOD returns TRUE if the window is valid and the needed drawing mode (see SETINSERT) is available. If necessary, an exception handling is raised.

PARALLELIZATION INFORMATION

DRAWCIRCLEMOD is local and processed completely exclusively without parallelization.

POSSIBLE PREDECESSOR FUNCTIONS

OPENWINDOW

POSSIBLE SUCCESSOR FUNCTIONS

REDUCEDOMAIN, DISPREGION, SETCOLORED, SETLINEWIDTH, SETDRAW, SETINSERT

SEE ALSO

GENCIRCLE, DRAWRECTANGLE1, DRAWRECTANGLE2, DRAWPOLYGON, SETINSERT

ALTERNATIVES

DRAWCIRCLE, DRAWELLIPSE, DRAWREGION

MODULE

SYSTEM

[out] double Row HWINDOWX.DRAWELLIPSE ([out] double Column,
[out] double Phi, [out] double Radius1, [out] double Radius2 )

void HOPERATORSETX.DRAWELLIPSE ([in] long WindowHandle,
[out] double Row, [out] double Column, [out] double Phi, [out] double Radius1,
[out] double Radius2 )

Interactive drawing of an ellipse.

DrawEllipse returns the parameter for any orientated ellipse, which has been created interactively by the user in the window.

The created ellipse is described by its center, its two half axes and the angle between the first half axis and the horizontal coordinate axis.

To create an ellipse you have to determine the center of the ellipse with the left mouse button. Keeping the button pressed determines the length (Radius1) and the orientation (Phi) of the first half axis. In doing so a temporary default length for the second half axis is assumed, which may be modified afterwards on demand. After another mouse click in the center of the created ellipse you can move it. A mouse click close to a vertex "grips" it to modify the length of the appropriate half axis. You may modify the orientation only, if a vertex of the first half axis is gripped.

Pressing the right mouse button terminates the procedure. After terminating the procedure the ellipse is not visible in the window any longer.

Parameter

- WindowHandle (input control) window ~ HWINDOWX/long Window id.
- Row (output control) ellipse.center.y ~ double Row index of the center.
- Column (output control) ellipse.center.x ~ double Column index of the center.
- Phi (output control) ellipse.angle.rad ~ double Orientation of the first half axis in radians.
- Radius1 (output control) ellipse.radius1 ~ double First half axis.
- Radius2 (output control) ellipse.radius2 ~ double Second half axis.

Example

read_image(Image,’affe’)
draw_ellipse(WindowHandle, Row, Column, Phi, Radius1, Radius2)
gen_ellipse(Ellipse, Row, Column, Phi, Radius1, Radius2)
reduce_domain(Image, Ellipse, GrayEllipse)
sobel_amp(GrayEllipse, Sobel, ’sum_abs’, 3)
disp_image(Sobel, WindowHandle).

---

**Result**

DrawEllipse returns TRUE, if the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised.

---

**Parallelization Information**

DrawEllipse is local and processed completely exclusively without parallelization.

---

**Possible Predecessor Functions**

OpenWindow

---

**Possible Successor Functions**

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

---

**See Also**

GenEllipse, DrawRectangle1, DrawRectangle2, DrawPolygon, SetInsert

---

**Alternatives**

DrawEllipseMod, DrawCircle, DrawRegion

---

**Module**

System

---

```c
[out] double Row HWindowX.DrawEllipseMod ([in] double RowIn,
[in] double ColumnIn, [in] double PhiIn, [in] double Radius1In,
[in] double Radius2In, [out] double Column, [out] double Phi,
[out] double Radius1, [out] double Radius2 )
```

---

Interactive drawing of an ellipse.

DrawEllipseMod returns the parameter for any orientated ellipse, which has been created interactively by the user in the window.

The created ellipse is described by its center, its two half axes and the angle between the first half axis and the horizontal coordinate axis.

To create an ellipse are expected the parameters RowIn, ColumnIn, PhiIn, Radius1In, Radius2In. Keeping the button pressed determines the length (Radius1) and the orientation (Phi) of the first half axis. In doing so a temporary default length for the second half axis is assumed, which may be modified afterwards on demand. After another mouse click in the center of the created ellipse you can move it. A mouse click close to a vertex “grips” it to modify the length of the appropriate half axis. You may modify the orientation only, if a vertex of the first half axis is gripped.

Pressing the right mouse button terminates the procedure. After terminating the procedure the ellipse is not visible in the window any longer.

---

**Parameter**

- **WindowHandle** (input control) ................................. window ～ HWindowX / long Window_JD.
- **RowIn** (input control) ........................................ ellipse.center.y ～ double Row index of the barycenter.
- **ColumnIn** (input control) ................................. ellipse.center.x ～ double Column index of the barycenter.
ϕ<sub>In</sub> (input control) ................................................ ellipse.angle.rad  \(\sim\) double
Orientation of the bigger half axis in radians.

Radius1<sub>In</sub> (input control) ................................................ ellipse.radius1  \(\sim\) double
Bigger half axis.

Radius2<sub>In</sub> (input control) ................................................ ellipse.radius1  \(\sim\) double
Smaller half axis.

Row (output control) ................................................ ellipse.center.y  \(\sim\) double
Row index of the center.

Column (output control) ................................................ ellipse.center.x  \(\sim\) double
Column index of the center.

ϕ (output control) ................................................ ellipse.angle.rad  \(\sim\) double
Orientation of the first half axis in radians.

Radius1 (output control) ................................................ ellipse.radius1  \(\sim\) double
First half axis.

Radius2 (output control) ................................................ ellipse.radius2  \(\sim\) double
Second half axis.

Example

```
read_image(Image,’affe’)
draw_ellipse_mod(WindowHandle,RowIn,ColumnIn,ϕIn,Radius1In,Radius2In,Row,Column,ϕ,R
[0x0]
erase_ellipse(Ellipse,ϕ,R)
reduce_domain(Image,Ellipse,GrayEllipse)
sobel_amp(GrayEllipse,Sobel,’sum_abs’,3)
disp_image(Sobel,WindowHandle).
```

Result

DrawEllipseMod returns TRUE, if the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised.

Parallelization Information

DrawEllipseMod is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow

Possible Successor Functions

ReduceDomain,DispRegion,SetColored,SetLineWidth,SetDraw,SetInsert

See Also

GenEllipse,DrawRectangle1,DrawRectangle2,DrawPolygon,SetInsert

Alternatives

DrawEllipse,DrawCircle,DrawRegion

Module

System

```
[out] double Row1 HWindowX.DrawLine ([out] double Column1,
[out] double Row2, [out] double Column2 )

void HOperatorSetX.DrawLine ([in] long WindowHandle, [out] double Row1,
[out] double Column1, [out] double Row2, [out] double Column2 )
```

Draw a line.

DrawLine returns the parameter for a line, which has been created interactively by the user in the window.

To create a line you have to press the left mouse button determining a start point of the line. While keeping the button pressed you may “drag” the line in any direction. After another mouse click in the middle of the created line you can move it. If you click on one end point of the created line, you may move this point. Pressing the right mousebutton terminates the procedure.
After terminating the procedure the line is not visible in the window any longer.

---

**Parameter**

- **WindowHandle** (input control) .......................................................... window ~ HWindowX / long Window jd.
- **Row1** (output control) .......................................................... line.begin.y ~ double Row index of the first point of the line.
- **Column1** (output control) .......................................................... line.begin.x ~ double Column index of the first point of the line.
- **Row2** (output control) .......................................................... line.end.y ~ double Row index of the second point of the line.
- **Column2** (output control) .......................................................... line.end.x ~ double Column index of the second point of the line.

---

**Example**

get_system('width', Width)
get_system('height', Height)
set_part(WindowHandle, 0, 0, Width-1, Height-1)
read_image(Image, 'affe')
disp_image(Image, WindowHandle)
draw_line(WindowHandle, Row1, Column1, Row2, Column2)
set_part(WindowHandle, Row1, Column1, Row2, Column2)
disp_image(Image, WindowHandle)
fwrite_string([['Clipping = (', Row1, ',', Column1, ')'])
fwrite_string([',(', Row2, ',', Column2, ')'])
fnew_line().

---

**Result**

DrawLine returns TRUE, if the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised.

---

**Parallelization Information**

DrawLine is local and processed completely exclusively without parallelization.

---

**Possible Predecessor Functions**

OpenWindow

---

**Possible Successor Functions**

ReduceDomain, DispLine, SetColored, SetLineWidth, SetDraw, SetInsert

---

**See Also**

DrawLineMod, GenRectangle1, DrawCircle, DrawEllipse, SetInsert

---

**Module**

```c
[out] double Row1 HWindowX.DrawLineMod ([in] double Row1In,
[in] double Column1In, [in] double Row2In, [in] double Column2In,
[out] double Column1, [out] double Row2, [out] double Column2 )
```

```c
void HOperatorSetX.DrawLineMod ([in] long WindowHandle,
[in] double Row1In, [in] double Column1In, [in] double Row2In,
[in] double Column2In, [out] double Row1, [out] double Column1,
[out] double Row2, [out] double Column2 )
```

---

**Draw a line.**

DrawLineMod returns the parameter for a line, which has been created interactively by the user in the window.

To create a line are expected the coordinates of the start point Row1In, Column1In and of the end point Row2In, Column2In. If you click on one end point of the created line, you may move this point. After another mouse click in the middle of the created line you can move it.
Pressing the right mouse button terminates the procedure.

After terminating the procedure the line is not visible in the window any longer.

---

**Parameter**

- `WindowHandle` (input control) .................. `window ~ HWindowX / long`
  Window handle.
- `Row1In` (input control) .................. `line.begin.y ~ double`
  Row index of the first point of the line.
- `Column1In` (input control) .................. `line.begin.x ~ double`
  Column index of the first point of the line.
- `Row2In` (input control) .................. `line.end.y ~ double`
  Row index of the second point of the line.
- `Column2In` (input control) .................. `line.end.x ~ double`
  Column index of the second point of the line.
- `Row1` (output control) .................. `line.begin.y ~ double`
  Row index of the first point of the line.
- `Column1` (output control) .................. `line.begin.x ~ double`
  Column index of the first point of the line.
- `Row2` (output control) .................. `line.end.y ~ double`
  Row index of the second point of the line.
- `Column2` (output control) .................. `line.end.x ~ double`
  Column index of the second point of the line.

---

**Example**

```
get_system('width',Width)
get_system('height',Height)
set_part(WindowHandle,0,0,Width-1,Height-1)
read_image(Image,'affe')
disp_image(Image,WindowHandle)
draw_line_mod(WindowHandle,10,20,55,124,Row1,Column1,Row2,Column2)
set_part(WindowHandle,Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle)
fwrite_string([['Clipping = (',Row1,',',Column1,')']])
fwrite_string([',(',Row2,',',Column2,')'])
fnew_line().
```

---

**Result**

`DrawLineMod` returns `TRUE`, if the window is valid and the needed drawing mode is available. If necessary, an exception handling is raised.

---

**Parallelization Information**

`DrawLineMod` is `local` and processed `completely exclusively` without parallelization.

---

**Possible Predecessor Functions**

```
OpenWindow
```

---

**Possible Successor Functions**

```
ReduceDomain,DispLine,SetColored,SetLineWidth,SetDraw,SetInsert
```

---

**See Also**

```
GenCircle,DrawRectangle1,DrawRectangle2
```

---

**Alternatives**

```
DrawLine,DrawEllipse,DrawRegion
```

---

**Module**

```
System
```

---

### 4.1. DRAWING

**[out] double Row HWindowX.DrawPoint**

[out] double Column

**void HOperatorSetX.DrawPoint**

[in] long WindowHandle, [out] double Row,

[out] double Column

#### Draw a point.

DrawPoint returns the parameter for a point, which has been created interactively by the user in the window.

To create a point you have to press the left mouse button. While keeping the button pressed you may “drag” the point in any direction. Pressing the right mouse button terminates the procedure.

After terminating the procedure the point is not visible in the window any longer.

---

**Parameter**

- **WindowHandle** (input control) 
  - window \( \sim HWindowX / long \)
  - Window id.
- **Row** (output control) 
  - point.y \( \sim double \)
  - Row index of the point.
- **Column** (output control) 
  - point.x \( \sim double \)
  - Column index of the point.

---

#### Example

```plaintext
get_system('width',Width)
get_system('height',Height)
set_part(WindowHandle,0,0,Width-1,Height-1)
read_image(Image,'affe')
disp_image(Image,WindowHandle)
draw_point(WindowHandle,Row1,Column1)
disp_line(WindowHandle,Row1-2,Column1,Row1+2,Column1)
disp_line(WindowHandle,Row1,Column1-2,Row1,Column1+2)
disp_image(Image,WindowHandle)
fwrite_string(['Clipping = (',Row1,',',Column1,')]')
```

---

**Result**

DrawPoint returns TRUE, if the window is valid and the needed drawing mode is available. If necessary, an exception handling is raised.

---

**Parallelization Information**

DrawPoint is **local** and processed **completely exclusively** without parallelization.

---

**Possible Predecessor Functions**

OpenWindow

---

**Possible Successor Functions**

ReduceDomain, DispLine, SetColored, SetLineWidth, SetDraw, SetInsert

---

**See Also**

DrawPointMod, DrawCircle, DrawEllipse, SetInsert

---

**Module**

**System**

---

**Draw a point.**

DrawPointMod returns the parameter for a point, which has been created interactively by the user in the window.
To create a point are expected the coordinates RowIn and ColumnIn. While keeping the button pressed you may "drag" the point in any direction. Pressing the right mousebutton terminates the procedure.

After terminating the procedure the point is not visible in the window any longer.

```
Example
```

```
get_system('width',Width)
get_system('height',Height)
set_part(WindowHandle,0,0,Width-1,Height-1)
read_image(Image,'affe')
draw_point_mod(WindowHandle,Row1,Column1)
disp_line(WindowHandle,Row1-2,Column1,Row1+2,Column1)
disp_line(WindowHandle,Row1,Column1-2,Row1,Column1+2)
disp_image(Image,WindowHandle)
fwrite_string(['Clipping = (',Row1,',',Column1,')'])
```

```
Result
```

DrawPointMod returns TRUE, if the window is valid and the needed drawing mode is available. If necessary, an exception handling is raised.

```
Parallelization Information
```

DrawPointMod is local and processed completely exclusively without parallelization.

```
Possible Predecessor Functions
```

OpenWindow

```
Possible Successor Functions
```

ReduceDomain,DispLine,SetColored,SetLineWidth,SetDraw,SetInsert

```
See Also
```

DrawPoint,DrawCircle,DrawEllipse,SetInsert

```
Module
```

```
HRegionX PolygonRegion
```HWindowX.DrawPolygon ( )

```void
HRegionX.DrawPolygon ([in] HWindowX WindowHandle )
```

```void
HOperatorSetX.DrawPolygon ([out] HUntypedObjectX PolygonRegion, [in] long WindowHandle )
```

Interactive drawing of a polygon row.

DrawPolygon produces an image. The region of that image spans exactly the imagepoints entered interactively by mouse clicks (gray values remain undefined).

Painting in the output window happens while pressing the left mouse button. Releasing the left mouse button and repressing it at another position effects drawing a line between these two points. Pressing the right mouse button terminates the input. Painting uses that color which has been set by SetColor, SetRgb, etc. /
4.1. DRAWING

To put gray values on the created PolygonRegion for further processing, you may use the procedure ReduceDomain.

Attention
The painted contour is not closed automatically, in particular it is not “filled up” either.
Output object’s gray values are not defined.

Parameter

- **PolygonRegion** (output iconic) .............................region  $\sim HRegionX / \text{HU}ntypedObjectX$
  Region, which encompasses all painted points.
- **WindowHandle** (input control) ...............................window $\sim HWindowX / \text{long Window}_w$

Example

draw_polygon(Polygon,WindowHandle)
convex(Polygon,Filled)
disp_region(Filled,WindowHandle).

Result
If the window is valid, DrawPolygon returns TRUE. If necessary, an exception handling is raised.

Parallelization Information
DrawPolygon is *local* and processed *completely exclusively* without parallelization.

Possible Predecessor Functions
OpenWindow

Possible Successor Functions
ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw

See Also
ReduceDomain, FillUp, SetColor

Alternatives
DrawRegion, DrawCircle, DrawRectangle1, DrawRectangle2, Boundary

Module

System

```
[out] double Row1 HWindowX.DRAWRectangle1 ([out] double Column1,
[out] double Row2, [out] double Column2 )
void HOperatorSetX.DRAWRectangle1 ([in] long WindowHandle,
[out] double Row1, [out] double Column1, [out] double Row2,
[out] double Column2 )
```

Draw a rectangle parallel to the coordinate axis.

DrawRectangle1 returns the parameter for a rectangle parallel to the coordinate axes, which has been created interactively by the user in the window.

To create a rectangle you have to press the left mouse button determining a corner of the rectangle. While keeping the button pressed you may “drag” the rectangle in any direction. After another mouse click in the middle of the created rectangle you can move it. A click close to one side “grips” it to modify the rectangle’s dimension in perpendicular direction to this side. If you click on one corner of the created rectangle, you may move this corner.

Pressing the right mouse button terminates the procedure.

After terminating the procedure the rectangle is not visible in the window any longer.

Parameter

- **WindowHandle** (input control) .............................window $\sim HWindowX / \text{long Window}_w$
- **Row1** (output control) .................................rectangle.origin.y $\sim \text{double Row}$
  Row index of the left upper corner.
Get the system’s width:

```c
get_system('width', Width)
```

Get the system’s height:

```c
get_system('height', Height)
```

Set the part of the window:

```c
set_part(WindowHandle, 0, 0, Width-1, Height-1)
```

Read an image:

```c
read_image(Image, 'affe')
```

Display the image:

```c
disp_image(Image, WindowHandle)
```

Draw a rectangle:

```c
draw_rectangle1(WindowHandle, Row1, Column1, Row2, Column2)
```

Set the part of the window again:

```c
set_part(WindowHandle, Row1, Column1, Row2, Column2)
```

Redisplay the image:

```c
disp_image(Image, WindowHandle)
```

Write the clipping region to a file:

```c
fwrite_string(['Clipping = (', Row1, ', ', Column1, ')
'])
fwrite_string(['(', Row2, ', ', Column2, ')'])
```

End file writing:

```c
fnew_line().
```

**Example**

```c
get_system('width',Width)
get_system('height',Height)
set_part(WindowHandle,0,0,Width-1,Height-1)
read_image(Image,'affe')
disp_image(Image,WindowHandle)
draw_rectangle1(WindowHandle,Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle)
fwrite_string(['Clipping = (',Row1,',',Column1,')'])
fwrite_string(['(',Row2,',',Column2,')'])
fnew_line().
```

**Result**

`draw_rectangle1` returns `TRUE`, if the window is valid and the needed drawing mode (see `set_insert`) is available. If necessary, an exception handling is raised.

**Parallelization Information**

`draw_rectangle1` is `local` and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

`open_window`

**Possible Successor Functions**

`reduce_domain`, `disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`

**See Also**

`gen_rectangle1`, `draw_circle`, `draw_ellipse`, `set_insert`

**Alternatives**

`draw_rectangle1_mod`, `draw_rectangle2`, `draw_region`

**Module**

System

```c
[out] double Row1
```

**HWindowX**

```c
HWindowX.DrawRectangle1Mod([in] double Row1In,
[in] double Column1In, [in] double Row2In, [in] double Column2In,
[out] double Column1, [out] double Row2, [out] double Column2)
```

```c
void HOperatorSetX.DrawRectangle1Mod([in] long WindowHandle,
[in] double Row1In, [in] double Column1In, [in] double Row2In,
[in] double Column2In, [out] double Row1, [out] double Column1,
[out] double Row2, [out] double Column2)
```

**Draw a rectangle parallel to the coordinate axis.**

`draw_rectangle1_mod` returns the parameter for a rectangle parallel to the coordinate axes, which has been created interactively by the user in the window.

To create a rectangle are expected the parameters `Row1In`, `Column1In`, `Row2In` and `Column2In`. After a mouse click in the middle of the created rectangle you can move it. A click close to one side “grIPS” it to modify the rectangle’s dimension in perpendicular direction to this side. If you click on one corner of the created rectangle, you may move this corner. Pressing the right mouse button terminates the procedure.

After terminating the procedure the rectangle is not visible in the window any longer.
4.1. DRAWING

Parameter

- **WindowHandle** (input control) 
  - window \( \sim HWindowX / long \) Window.id.
- **Row1In** (input control) 
  - \( \text{rectangle.origin.y} \sim double \) Row index of the left upper corner.
- **Column1In** (input control) 
  - \( \text{rectangle.origin.x} \sim double \) Column index of the left upper corner.
- **Row2In** (input control) 
  - \( \text{rectangle.corner.y} \sim double \) Row index of the right lower corner.
- **Column2In** (input control) 
  - \( \text{rectangle.corner.x} \sim double \) Column index of the right lower corner.
- **Row1** (output control) 
  - \( \text{rectangle.origin.y} \sim double \) Row index of the left upper corner.
- **Column1** (output control) 
  - \( \text{rectangle.origin.x} \sim double \) Column index of the left upper corner.
- **Row2** (output control) 
  - \( \text{rectangle.corner.y} \sim double \) Row index of the right lower corner.
- **Column2** (output control) 
  - \( \text{rectangle.corner.x} \sim double \) Column index of the right lower corner.

Example

get_system('width', Width)
get_system('height', Height)
set_part(WindowHandle, 0, 0, Width-1, Height-1)
read_image(Image, 'affe')
disp_image(Image, WindowHandle)
draw_rectangle1_mod(WindowHandle, Row1In, Column1In, Row2In, Column2In, Row1, Column1, Row2, Column2)
set_part(WindowHandle, Row1, Column1, Row2, Column2)
disp_image(Image, WindowHandle)
fwrite_string(['Clipping = (', Row1, ',', Column1, ')'])
fwrite_string(['(', Row2, ',', Column2, ')'])
fnew_line().

Result

**DrawRectangle1Mod** returns TRUE, if the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised.

Parallelization Information

**DrawRectangle1Mod** is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow

Possible Successor Functions

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

See Also

GenRectangle1, DrawCircle, DrawEllipse, SetInsert

Alternatives

DrawRectangle1, DrawRectangle2, DrawRegion

Module

System

HALCON 6.0
Interactive drawing of any orientated rectangle.

DrawRectangle2 returns the parameter for any orientated rectangle, which has been created interactively by the user in the window.

The created rectangle is described by its center, its two half axes and the angle between the first half axis and the horizontal coordinate axis.

To create a rectangle you have to press the left mouse button for the center of the rectangle. While keeping the button pressed you may dimension the length ($Length1$) and the orientation ($Phi$) of the first half axis. In doing so a temporary default length for the second half axis is assumed, which may be modified afterwards on demand. After another mouse click in the middle of the created rectangle, you can move it. A click close to one side “grips” it to modify the rectangle’s dimension in perpendicular direction to this side. You only can modify the orientation, if you grip a side perpendicular to the first half axis. Pressing the right mouse button terminates the procedure.

After terminating the procedure the rectangle is not visible in the window any longer.

**Parameter**

- **WindowHandle** (input control) .................. window $\sim$ HWindowX/long Window\_id.
- **Row** (output control) .......................... rectangle2.center.y $\sim$ double Row index of the barycenter.
- **Column** (output control) .................... rectangle2.center.x $\sim$ double Column index of the barycenter.
- **Phi** (output control) .......................... rectangle2.angle.rad $\sim$ double Orientation of the bigger half axis in radians.
- **Length1** (output control) .................... rectangle2.hw\_width $\sim$ double Bigger half axis.
- **Length2** (output control) .................... rectangle2.hh\_height $\sim$ double Smaller half axis.

**Result**

DrawRectangle2 returns TRUE, if the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised.

**Parallelization Information**

DrawRectangle2 is local and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

OpenWindow

**Possible Successor Functions**

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

**See Also**

GenRectangle2, DrawCircle, DrawEllipse, SetInsert

**Alternatives**

DrawRectangle2Mod, DrawRectangle1, DrawRegion

**Module**

System
Interactive drawing of any orientated rectangle.

DrawRectangle2Mod returns the parameter for any orientated rectangle, which has been created interactively by the user in the window.

The created rectangle is described by its center, its two half axes and the angle between the first half axis and the horizontal coordinate axis.

To create a rectangle are expected the parameters RowIn, ColumnIn, PhiIn, Length1In, Length2In. A click close to one side “grips” it to modify the rectangle’s dimension in perpendicular direction (Length2) to this side. You only can modify the orientation (Phi), if you grip a side perpendicular to the first half axis. After another mouse click in the middle of the created rectangle, you can move it. Pressing the right mouse button terminates the procedure.

After terminating the procedure the rectangle is not visible in the window any longer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WindowHandle</td>
<td>(input control) window ~ HWindowX / long Window_id.</td>
</tr>
<tr>
<td>RowIn</td>
<td>(input control) rectangle2.center.y ~ double Row index of the barycenter.</td>
</tr>
<tr>
<td>ColumnIn</td>
<td>(input control) rectangle2.center.x ~ double Column index of the barycenter.</td>
</tr>
<tr>
<td>PhiIn</td>
<td>(input control) rectangle2.angle.rad ~ double Orientation of the bigger half axis in radians.</td>
</tr>
<tr>
<td>Length1In</td>
<td>(input control) rectangle2.hwidth ~ double Bigger half axis.</td>
</tr>
<tr>
<td>Length2In</td>
<td>(input control) rectangle2.hheight ~ double Smaller half axis.</td>
</tr>
<tr>
<td>Row</td>
<td>(output control) rectangle2.center.y ~ double Row index of the barycenter.</td>
</tr>
<tr>
<td>Column</td>
<td>(output control) rectangle2.center.x ~ double Column index of the barycenter.</td>
</tr>
<tr>
<td>Phi</td>
<td>(output control) rectangle2.angle.rad ~ double Orientation of the bigger half axis in radians.</td>
</tr>
<tr>
<td>Length1</td>
<td>(output control) rectangle2.hwidth ~ double Bigger half axis.</td>
</tr>
<tr>
<td>Length2</td>
<td>(output control) rectangle2.hheight ~ double Smaller half axis.</td>
</tr>
</tbody>
</table>

Result

DrawRectangle2Mod returns TRUE, if the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised.

Parallelization Information

DrawRectangle2Mod is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow

Possible Successor Functions

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

HALCON 6.0
Interactive drawing of a closed region.

DrawRegion produces an image. The region of that image spans exactly the image region entered interactively by mouse clicks (gray values remain undefined). Painting happens in the output window while keeping the left mouse button pressed. The left mouse button even operates by clicking in the output window; through this a line between the previous clicked points is drawn. Clicking the right mouse button terminates input and closes the outline. Subsequently the image is "filled up". Also it contains the whole image area enclosed by the mouse.

Painting uses that color which has been set by SetColor, SetRgb, etc.

Attention

The output object’s gray values are not defined.

Parameter

- **Region** (output iconic) ............................. region ~ HRegionX / HUntypedObjectX
  Interactive created region.
- **WindowHandle** (input control) ........................ window ~ HWindowX / long WindowId.

Example

read_image(Image,’fabrik’)
disp_image(Image,WindowHandle)
draw_region(Region,WindowHandle)
reduce_domain(Image,Region,New)
regiongrowing(New,Segmente,5,5,6,50)
set_colored(WindowHandle,12)
disp_region(Segmente,WindowHandle).

Result

If the window is valid, DrawRegion returns TRUE. If necessary, an exception handling is raised.

Parallelization Information

DrawRegion is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow

Possible Successor Functions

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw

See Also

DrawPolygon, ReduceDomain, FillUp, SetColor

Alternatives

DrawCircle, DrawEllipse, DrawRectangle1, DrawRectangle2

Module

System
Close all open gnuplot files or terminate an active gnuplot sub-process.

GnuplotClose closes all gnuplot files opened by GnuplotOpenFile or terminates the gnuplot sub-process created with GnuplotOpenPipe. In the latter case, all temporary files used to display images and control values are deleted. This means that GnuplotClose must be called after such a plot sequence. GnuplotFileID is the identifier of the corresponding gnuplot output stream.

**Parameter**

- **GnuplotFileID** (input control) | Identifier for the gnuplot output stream.

**Result**

GnuplotClose returns TRUE if GnuplotFileID is a valid gnuplot output stream. Otherwise, an exception handling is raised.

**Parallelization Information**

GnuplotClose is processed completely exclusively without parallelization.

**Possible Predecessor Functions**

GnuplotOpenPipe, GnuplotOpenFile, GnuplotPlotImage

**See Also**

GnuplotOpenPipe, GnuplotClose, GnuplotPlotImage

### GnuplotOpenFile

Open a gnuplot file for visualization of images and control values.

GnuplotOpenFile allows the output of images and control values in a format which can be later processed by gnuplot. The parameter FileName determines the base-name of the files to be created by calls to GnuplotPlotImage. GnuplotOpenFile generates a gnuplot control file with the name `<FileName>.gp`, in which the respective plot commands are written. Each image plotted by GnuplotPlotImage (or control values plotted by GnuplotPlotCtrl) creates a data file with the name `<FileName>.dat.<Number>`, where Number is the number of the plot in the current sequence. The generated control file can later be edited to create the desired effect. After the last plot GnuplotClose has to be called in order to close all open files. The corresponding identifier for the gnuplot output stream is returned in GnuplotFileID.

**Parameter**

- **FileName** (input control) | Base name for control and data files.
- **GnuplotFileID** (output control) | Identifier for the gnuplot output stream.

**Result**

GnuplotOpenFile returns the value TRUE if the control file could be opened. Otherwise, an exception handling is raised.

**Parallelization Information**

GnuplotOpenFile is processed completely exclusively without parallelization.

**Possible Successor Functions**

GnuplotPlotImage, GnuplotClose

**See Also**

GnuplotOpenPipe, GnuplotClose, GnuplotPlotImage
Open a pipe to a gnuplot process for visualization of images and control values.

GnuplotOpenPipe opens a pipe to a gnuplot sub-process with which subsequently images can be visualized as 3D-plots (GnuplotPlotImage) or control values can be visualized as 2D-plots (GnuplotPlotCtrl). The sub-process must be terminated after displaying the last plot by calling GnuplotClose. The corresponding identifier for the gnuplot output stream is returned in GnuplotFileID.

Attention

GnuplotOpenPipe is only implemented for Unix because gnuplot for Windows (wgnuplot) cannot be controlled by an external process.

Parameter

- **GnuplotFileID** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . gnuplot id ¬ HGnuplotX / long
  Identifier for the gnuplot output stream.

Result

GnuplotOpenPipe returns the value TRUE if the sub-process could be created. Otherwise, an exception handling is raised.

Parallelization Information

GnuplotOpenPipe is processed completely exclusively without parallelization.

Possible Successor Functions

GnuplotPlotImage, GnuplotPlotCtrl, GnuplotClose

Alternatives

GnuplotOpenFile

Plot control values using gnuplot.

GnuplotPlotCtrl displays a tuple of control values using gnuplot. If there is an active gnuplot sub-process (started with GnuplotOpenPipe), the image is displayed in a gnuplot window. Otherwise, the image is output to a file, which can be later read by gnuplot. In both cases the gnuplot output stream is identified by GnuplotFileID.

Parameter

- **GnuplotFileID** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . gnuplot_id ¬ HGnuplotX / long
  Identifier for the gnuplot output stream.

- **Values** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number ¬ VARIANT ( real, integer )
  Control values to be plotted (y-values).

Result

GnuplotPlotCtrl returns the value if GnuplotFileID is a valid gnuplot output stream, if the data file for the current plot could be opened, and if only integer or floating point values were passed. Otherwise, an exception handling is raised.
4.2. GNUPLOT

Parallelization Information
GnuplotPlotCtrl is processed completely exclusively without parallelization.

Possible Predecessor Functions
GnuplotOpenPipe, GnuplotOpenFile

Possible Successor Functions
GnuplotClose

See Also
GnuplotOpenPipe, GnuplotOpenFile, GnuplotClose

Module
System

void HGnuplotX.GnuplotPlotFunct1D ([in] HFunction1dX Function )
void HOperatorSetX.GnuplotPlotFunct1D ([in] long GnuplotFileID, [in] VARIANT Function )

Plot a function using gnuplot.

GnuplotPlotCtrl displays a function of control values using gnuplot. If there is an active gnuplot sub-
process (started with GnuplotOpenPipe), the image is displayed in a gnuplot window. Otherwise, the image
is output to a file, which can be later read by gnuplot. In both cases the gnuplot output stream is identified by
GnuplotFileID.

Parameter

▷ GnuplotFileID (input control) ........................................... gnuplot_id ~ HGnuplotX / long
  Identifier for the gnuplot output stream.
▷ Function (input control) ................................. function_id ~ HFunction1dX / VARIANT ( real, integer )
  Function to be plotted.

Result
GnuplotPlotCtrl returns TRUE if GnuplotFileID is a valid gnuplot output stream, and if the data file for
the current plot could be opened. Otherwise, an exception handling is raised.

Parallelization Information
GnuplotPlotFunct1D is processed completely exclusively without parallelization.

Possible Predecessor Functions
GnuplotOpenPipe, GnuplotOpenFile

Possible Successor Functions
GnuplotClose

See Also
GnuplotOpenPipe, GnuplotOpenFile, GnuplotClose

Alternatives
GnuplotPlotCtrl

Module
System
Visualize images using gnuplot.

GnuplotPlotImage displays an image as a 3D-plot using gnuplot. If there is an active gnuplot sub-process (started with GnuplotOpenPipe), the image is displayed in a gnuplot window. Otherwise, the image is output to a file, which can be later read by gnuplot. In both cases the gnuplot output stream is identified by GnuplotFileID. The parameters SamplesX and SamplesY determine the number of data points in the x- and y-direction, respectively, which gnuplot should use to display the image. They are the equivalent of the gnuplot variables samples and isosamples. The parameters ViewRotX and ViewRotZ determine the rotation of the plot with respect to the viewer. ViewRotX is the rotation of the coordinate system about the x-axis, while ViewRotZ is the rotation of the plot about the z-axis. These two parameters correspond directly to the first two parameters of the 'set view' command in gnuplot. The parameter Hidden3D determines whether hidden surfaces should be removed. This is equivalent to the 'set hidden3d' command in gnuplot. If a single image is passed to the operator, it is displayed in a separate plot. If multiple images are passed, they are displayed in the same plot.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image (input iconic)</td>
<td>Image to be plotted.</td>
<td>IHObjectX / HImageX</td>
</tr>
<tr>
<td>GnuplotFileID (input control)</td>
<td>Identifier for the gnuplot output stream.</td>
<td>long</td>
</tr>
<tr>
<td>SamplesX (input control)</td>
<td>Number of samples in the x-direction.</td>
<td>integer</td>
</tr>
<tr>
<td>Default Value: 64</td>
<td>Typical Range of Values: $2 \leq \text{SamplesX} \leq 2$</td>
<td></td>
</tr>
<tr>
<td>SamplesY (input control)</td>
<td>Number of samples in the y-direction.</td>
<td>integer</td>
</tr>
<tr>
<td>Default Value: 64</td>
<td>Typical Range of Values: $2 \leq \text{SamplesY} \leq 2$</td>
<td></td>
</tr>
<tr>
<td>ViewRotX (input control)</td>
<td>Rotation of the plot about the x-axis.</td>
<td>number</td>
</tr>
<tr>
<td>Default Value: 60</td>
<td>Typical Range of Values: $0 \leq \text{ViewRotX} \leq 0$</td>
<td></td>
</tr>
<tr>
<td>ViewRotZ (input control)</td>
<td>Rotation of the plot about the z-axis.</td>
<td>number</td>
</tr>
<tr>
<td>Default Value: 30</td>
<td>Typical Range of Values: $0 \leq \text{ViewRotZ} \leq 0$</td>
<td></td>
</tr>
<tr>
<td>Hidden3D (input control)</td>
<td>Plot the image with hidden surfaces removed.</td>
<td>string</td>
</tr>
<tr>
<td>Default Value: 'hidden3d'</td>
<td>Value List: Hidden3D $\in { \text{'hidden3d'}, \text{'nohidden3d'} }$</td>
<td></td>
</tr>
</tbody>
</table>
4.3 LUT

Result

GnuplotPlotImage returns the value if GnuplotFileID is a valid gnuplot output stream, and if the data file for the current plot could be opened. Otherwise, an exception handling is raised.

Parallelization Information

GnuplotPlotImage is processed completely exclusively without parallelization.

Possible Predecessor Functions

GnuplotOpenPipe, GnuplotOpenFile

Possible Successor Functions

GnuplotClose

See Also

GnuplotOpenPipe, GnuplotOpenFile, GnuplotClose

Module

System

4.3 LUT

Graphical view of the look-up-table (lut).

DispLut displays a graphical view of the look-up-table (lut) in the valid window. A look-up-table defines the transformation of image gray values to colors/gray levels on the screen. On most systems this can be modified. DispLut creates a graphical view of the table assigned to the output window with the logical window number WindowHandle and displays it for every basic color (red, green, blue). Row and Column define the position of the centre of the graphic. Scale allows scaling of the graphic, whereas 1 means displaying all 256 values, 2 means displaying 128 values, 3 means displaying only 64 values, etc. Tables for monochrome-representations are displayed in the currently set color (see SetColor, SetRgb, etc.). Tables for displaying "false colors" are viewed with red, green and blue for each color component.

Attention

DispLut can only be used on hardware supporting look-up-tables for the output.

Parameter

- **WindowHandle** (input control)  .window ~ HWindowX / long
  Window identifier.
- **Row** (input control)  .point.y ~ long
  Row of centre of the graphic.
  Default Value: 128
  Typical Range of Values: 0 ≤ Row ≤ 0
- **Column** (input control)  .point.x ~ long
  Column of centre of the graphic.
  Default Value: 128
  Typical Range of Values: 0 ≤ Column ≤ 0
- **Scale** (input control)  .integer ~ long
  Scaling of the graphic.
  Default Value: 1
  Value List: Scale ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  Typical Range of Values: 0 ≤ Scale ≤ 0

Example

```
set_lut(WindowHandle,'color')
disp_lut(WindowHandle,256,256,1)
get_mbutton(WindowHandle,_,_,_)
set_lut(WindowHandle,'sqrt')
disp_lut(WindowHandle,128,128,2).
```
Result
DispLut returns TRUE if the hardware supports a look-up-table, the window is valid and the parameters are correct. Otherwise an exception handling is raised.

Parallelization Information
DispLut is \textit{local} and processed \textit{completely exclusively} without parallelization.

Possible Predecessor Functions
SetLut

See Also
OpenWindow, OpenTextWindow, DrawLut, SetLut, SetFix, SetPixel, WriteLut, GetLut, SetColor

Module
System

\begin{verbatim}
void HWindowX.DrawLut ( )

void HOperatorSetX.DrawLut ([in] long WindowHandle )
\end{verbatim}

Manipulate look-up-table (lut) interactively.

\texttt{DrawLut} allows interactive manipulation of the look-up-table of the device currently displaying the output window.

By pressing and holding down the left mouse button one can change (from "left to right") the red-, green- and blue-intensity displayed in a 2 dimensional diagram with the gray values on the x-axis. The left mouse button also is used for choosing the color channel that should be changed. As an alternative, one can map pure gray levels (gray "color channel") to the gray values on the x-axis. The right mouse button is used for terminating the change-process. The modified look-up-table can be saved by \texttt{WriteLut} and reloaded later by \texttt{SetLut}. \texttt{GetLut} succeeding \texttt{DrawLut} returns directly the RGB tuple of the look-up-table. These are suitable as input of \texttt{SetLut}.

\texttt{Attention}
\texttt{DrawLut} can only be used on hardware supporting look-up-tables for the output and allow dynamic changing of the tables.

Parameter
\begin{itemize}
\item \texttt{WindowHandle} (input control) \hfill \texttt{window ~ HWindowX / long}
\end{itemize}

Example
\begin{verbatim}
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
draw_lut(WindowHandle)
write_lut(WindowHandle,'my_lut').
...
read_image(Image,'fabrik')
set_lut(WindowHandle,'my_lut').
\end{verbatim}

Result
\texttt{DrawLut} returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information
\texttt{DrawLut} is \textit{local} and processed \textit{completely exclusively} without parallelization.

Possible Successor Functions
SetLutStyle, SetLut, WriteLut, DispLut

See Also
WriteLut, SetLut, GetLut, DispLut

Alternatives
SetFix, SetRgb

Module
System
Get fixing of "'look-up-table'" (lut) for "'real color images'"

Parameter

- **WindowHandle** (input control) .......................... window \(\sim\) \texttt{HWindowX / long}
  Window identifier.

- **Mode** (output control) .......................... string \(\sim\) \texttt{String}
  Mode of fixing.
  **Default Value**: 'true'
  **Value List**: \(\text{Mode} \in \{ \text{true}, \text{false} \}\)

Parallelization Information

GetFixedLut is *local* and processed *completely exclusively* without parallelization.

Possible Successor Functions

System

Get current look-up-table (lut).

GetLut returns the name or the values of the look-up-table (lut) of the window, currently used by \texttt{DispImage} (or indirectly by \texttt{DispRegion}, etc.) for output. To set a look-up-table use \texttt{SetLut}. If the current table is a system table without any modification (by \texttt{SetFix}), the name of the table is returned. If it is a modified table, a table read from a file or a table for output with pseudo real colors, the RGB-values of the table are returned.

Attention

Parameter

- **WindowHandle** (input control) .......................... window \(\sim\) \texttt{HWindowX / long}
  Window identifier.

- **LookUpTable** (output control) .......................... string \(\sim\) \texttt{VARIANT ( integer, string )}
  Name of look-up-table or tuple of RGB-values.

Result

GetLut returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetLut is *local* and processed *completely exclusively* without parallelization.

Possible Successor Functions

DrawLut, \texttt{SetLut}

See Also

SetLut, DrawLut

Alternatives

SetFix, GetPixel

Module

System

HALCON 6.0
Get modification parameters of look-up-table (lut).

GetLutStyle returns the values that were set with SetLutStyle. Default is:

**Hue:** 0.0  
**Saturation:** 1.0  
**Intensity:** 1.0

**Parameter**

- **WindowHandle** (input control)              window  ~  HWindowX / long  
  Window identifier.
- **Hue** (output control)                      real  ~  double  
  Modification of color value.
- **Saturation** (output control)               real  ~  double  
  Modification of saturation.
- **Intensity** (output control)                real  ~  double  
  Modification of intensity.

**Result**

GetLutStyle returns TRUE if the window is valid and the parameter is correct. Otherwise an exception handling is raised.

**Parallelization Information**

GetLutStyle is local and processed completely exclusively without parallelization.

**Possible Successor Functions**

SetLutStyle, SetLut

**See Also**

SetLutStyle

System

**Module**

**HALCON/COM Reference Manual / 2000-11-16**
4.3. LUT

Possible Successor Functions
SetLutStyle, SetLut, WriteLut, DispLut

See Also
SetLut

Module
System

void HWindowX.SetFixedLut ([in] String Mode )
void HOperatorSetX.SetFixedLut ([in] long WindowHandle, [in] String Mode )

Fix "'look-up-table'" (lut) for "'real color images'".
<no description available>

Parameter

▷ WindowHandle (input control) .............................. window ~> HWindowX / long Window identifier.
▷ Mode (input control) ................................. string ~> String Mode of fixing.

Default Value : 'true'

Value List : Mode ∈ {'true', 'false'}

Parallelization Information

SetFixedLut is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

GetFixedLut

Module

System

void HWindowX.SetLut ([in] VARIANT LookUpTable )
void HOperatorSetX.SetLut ([in] long WindowHandle, [in] VARIANT LookUpTable )

Set "'look-up-table'" (lut).

SetLut sets look-up-table of the device (monitor) displaying the output window. A look-up-table defines the transformation of a "'gray value'" within an image into a gray value or color on the screen. It describes the screen gray value/color as a combination of red, green and blue for any image gray value (0..255) (so it is a 'table' to 'look up' the screen gray value/color for each image gray value: look-up-table). Transformation into screen-colors is performed in real-time at every time the screen is displayed new (typically this happens about 60 - 70 times per second). So it is possible to change the look-up-table to get a new look of images or regions. Please remind that not all machines support changing the look-up-table (e.g. monochrome resp. real color).

Look-up-tables within HALCON (and on a machine that supports 256 colors) are disposed into three areas:

S: system area resp. user area,
G: graphic colors,
B: image data.

Colors in S descend from applications that were active before starting HALCON and should not get lost. Graphic colors in G are used for operators such as DispRegion, DispCircle etc. and are set unique within all look-up-tables. An output in a graphic color has always got the same (color-)look, even if different look-up-tables are used. SetColor and SetRgb set graphic colors. Gray values resp. colors in B are used by DispImage to display an image. They can change according to the current look-up-table. There exist two exceptions to this concept:
• SetGray allows setting of colors of the area B for operators such as DispRegion,
• SetFix that allows modification of graphic colors.

For common monitors only one look-up-table can be loaded per screen. Whereas SetLut can be activated separately for each window. There is the following solution for this problem: It will always be activated the look-up-table that is assigned to the "active window" (a window is set into the state "active" by the window manager). Look-up-table can also be used with truecolor displays. In this case the look-up-table will be simulated in software. This means, that the look-up-table will be used each time an image is displayed.

WindowsNT specific: if the graphic card is used in mode different from truecolor, you must display the image after setting the look-up-table.

QueryLut lists the names of all look-up-tables. They differ from each other in the area used for gray values. Within this area the following behaviour is defined:

Gray value tables (1-7 image levels)

'default': Only the two basic colors (generally black and white) are used.

Color tables (Real color, static gray value steps)

'default': Table proposed by the hardware.

Gray value tables (256 colors)

'default': As 'linear'.

'linear': Linear increasing of gray values from 0 (black) to 255 (white).

'linear': Inverse function of 'linear'.

'sqr': Gray values increase according to square function.

'inv_sqr': Inverse function of 'sqr'.

'cube': Gray values increase according to cubic function.

'inv_cube': Inverse function of 'cube'.

'sqrt': Gray values increase according to square-root function.

'inv_sqrt': Inverse function of 'sqrt'.

'cubic_root': Gray values increase according to cubic-root function.

'inv_cubic_root': Inverse function of 'cubic_root'.

Color tables (256 colors)

'color1': Linear transition from red via green to blue.

'color2': Smooth transition from yellow via red, blue to green.

'color3': Smooth transition from yellow via red, blue, green, red to blue.

'color4': Smooth transition from yellow via red to blue.

'three': Displaying the three colors red, green and blue.

'six': Displaying the six basic colors yellow, red, magenta, blue, cyan and green.

'twelve': Displaying 12 colors.

'twenty_four': Displaying 24 colors.

'rainbow': Displaying the spectral colors from red via green to blue.

'temperature': Temperature table from black via red, yellow to white.

'change1': Color changement after every pixel within the table alternating the six basic colors.

'change2': Fivefold color changement from green via red to blue.

'change3': Threefold color changement from green via red to blue.
A look-up-table can be read from a file. Every line of such a file must contain three numbers in the range of 0 to 255, with the first number describing the amount of red, the second the amount of green and the third the amount of blue of the represented display color. The number of lines can vary. The first line contains information for the first gray value and the last line for the last value. If there are less lines than gray values, the available information values are distributed over the whole interval calculating the missing values by interpolation. If there are more lines than gray values, a number of (uniformly distributed) lines is ignored. The file-name must conform to "LookUpTable.lut". Within the parameter the name is specified without file extension. HALCON will search for the file in the current directory and after that in a specified directory (see SetSystem(":lutDir",<Pfad>)). It is also possible to call SetLut with a tuple of RGB-Values. These will be set directly. The number of parameter values must conform to the number of pixels currently used within the look-up-table.

SetLut can only be used with monitors supporting 256 gray levels/colors.

Parameter

- **WindowHandle** (input control) ........................ window ~ HWindowX/long
  Window identifier.
- **LookUpTable** (input control) ........................ string-array ~ VARIANT( integer, string )
  Name of look-up-table, values of look-up-table (RGB) or file name.

  Default Value : 'default'

  Value Suggestions : LookUpTable ∈ { 'default', 'linear', 'inverse', 'sqr', 'inv_sqr', 'cube', 'inv_cube',
  'sqrt', 'inv_sqrt', 'cubic_root', 'inv_cubic_root', 'color1', 'color2', 'color3', 'color4', 'three', 'six', 'twelve',
  'twenty_four', 'rainbow', 'temperature', 'cyclic_gray', 'cyclic_temperature', 'hsi', 'change1', 'change2',
  'change3'}

Example

```c
read_image(Image,'affe')
query_lut(WindowHandle,LUTs)
for(1,|LUTs|,i)
  set_lut(WindowHandle,LUTs[i])
  fwrite_string(["current table ",LUTs[i]])
  fnew_line()
get_mbutton(WindowHandle,_,_,_)
loop() .
```

Result

SetLut returns TRUE if the hardware supports a look-up-table and the parameter is correct. Otherwise an exception handling is raised.

Parallelization Information

SetLut is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

QueryLut, DrawLut, GetLut

Possible Successor Functions

WriteLut

See Also

GetLut, QueryLut, DrawLut, SetFix, SetColor, SetRgb, SetHsi, WriteLut

Alternatives

DrawLut, SetFix, SetPixel

Module

System
Changing the look-up-table (lut).

SetLutStyle changes the look-up-table (lut) of the device displaying the valid output window. It has got three parameters:

**Hue**: Rotation of color space. Hue = 1.9 conforms to a one-time rotation of the color space. No change: Hue = 0.0 Complement colors: Hue = 0.5

**Saturation**: Change of saturation. No change: Saturation = 1.0 Gray value image: Saturation = 0.0

**Intensity**: Change of intensity. No change: Intensity = 1.0 Black image: Intensity = 0.0

Change affects only the part of the look-up-table that is used for displaying images. The parameter of modification remain until the next call of SetLutStyle. Calling SetLut has got no effect on these parameters.

```c
void HWindowX.SetLutStyle ([in] double Hue, [in] double Saturation, [in] double Intensity )
```

---

**Example**

```c
read_image(Image,'affe')
set_lut(WindowHandle,'color')
repeat (:::) >
    get_mbutton(WindowHandle,Row,Column,Button)
    eval(Row/300.0,Saturation)
    eval(Column/512.0,Hue)
    set_lut_style(WindowHandle,Hue,Saturation,1.0)
until(Button = 1).
```

---

**Result**

SetLutStyle returns TRUE if the window is valid and the parameter is correct. Otherwise an exception handling is raised.

---

**Parallelization Information**

SetLutStyle is *local* and processed *completely exclusively* without parallelization.

---

**Possible Predecessor Functions**

GetLutStyle

---

**Possible Successor Functions**

SetLut

---

**See Also**

GetLutStyle

---

Write look-up-table (lut) as file.
WriteLut saves the look-up-table (resp. the part of it that is relevant for displaying image gray values) of the valid output window into a file named 'FileName.lut'. It can be read again later with SetLut.

Attention
WriteLut is only suitable for systems using 256 colors.

Parameter
- WindowHandle (input control) window ~ HWindowX / long Window identifier.
- FileName (input control) filename ~ String File name (of file containing the look-up-table).
  Default Value: /tmp/lut

Example
read_image(Image,'affe')
disp_image(Image,WindowHandle)
draw_lut(WindowHandle)
write_lut(WindowHandle,'test_lut').

Result
DispLut returns TRUE if the window with the demanded properties (256 colors) is valid and the parameter (file-name) is correct. It returns FAIL if the specified file can’t be opened. Otherwise an exception handling is raised.

Parallelization Information
WriteLut is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
DrawLut, SetLut

See Also
SetLut, DrawLut, SetPixel, GetPixel

4.4 Mouse

Wait until a mouse button is pressed.
GetMbutton returns the coordinates of the mouse pointer in the output window and the mouse button pressed (Button):
1: Left button,
2: Middle button,
4: Right button.

The operator waits until a button is pressed in the output window. If more than one button is pressed, the sum of the individual buttons’ values is returned. The origin of the coordinate system is located in the left upper corner of the window. The row coordinates increase towards the bottom, while the column coordinates increase towards the right. For graphics windows, the coordinates of the lower right corner are (image height-1, image width-1) (see OpenWindow, ResetObjDb), while for text windows they are (window height-1, window width-1) (see OpenTextwindow).

Attention
GetMbutton only returns if a mouse button is pressed in the window.

Parameter

- **WindowHandle** (input control)  
  Window identifier.

- **Row** (output control)  
  Row coordinate of the mouse position in the window.

- **Column** (output control)  
  Column coordinate of the mouse position in the window.

- **Button** (output control)  
  Mouse button(s) pressed.

Result
GetMbutton returns the value TRUE.

Parallelization Information
GetMbutton is *local* and processed *completely exclusively* without parallelization.

Possible Predecessor Functions
OpenWindow, OpenTextwindow

See Also
OpenWindow, OpenTextwindow

Alternatives
GetMposition

Module
System

```c
[out] long HWindowX.GetMposition ([out] long Column, [out] long Button )

```

Query the mouse position.

GetMposition returns the coordinates of the mouse pointer in the output window and the mouse button pressed. These values are returned regardless of the state of the mouse buttons (pressed or not pressed). If more than one button is pressed, the sum of the individual buttons’ values is returned. The possible values for Button are:

0: No button,
1: Left button,
2: Middle button,
4: Right button.

The origin of the coordinate system is located in the left upper corner of the window. The row coordinates increase towards the bottom, while the column coordinates increase towards the right. For graphics windows, the coordinates of the lower right corner are (image height-1, image width-1) (see OpenWindow, ResetObjDb), while for text windows they are (window height-1, window width-1) (see OpenTextwindow).
GetMposition fails (returns FAIL) if the mouse pointer is not located within the window. In this case, no values are returned.

- **WindowHandle** (input control)  window \( \sim \) HWindowX / long
  Window identifier.
- **Row** (output control)  point.y \( \sim \) long
  Row coordinate of the mouse position in the window.
- **Column** (output control)  point.x \( \sim \) long
  Column coordinate of the mouse position in the window.
- **Button** (output control)  integer \( \sim \) long
  Mouse button(s) pressed or 0.

GetMposition returns the value TRUE. If the mouse pointer is not located within the window, FAIL is returned.

**Parallelization Information**
GetMposition is local and processed completely exclusively without parallelization.

**Possible Predecessor Functions**
OpenWindow, OpenTextwindow

**Possible Successor Functions**
SetMshape

**See Also**
OpenWindow, OpenTextwindow, QueryMshape

**Alternatives**
GetMbutton

**Module**
System

```c
[out] String Cursor HWindowX.GetMshape ( )
void HOperatorSetX.GetMshape ([in] long WindowHandle,
[out] String Cursor )
```

Query the current mouse pointer shape.
GetMshape returns the name of the pointer shape set for the window. The mouse pointer shape can be used in the operator SetMshape.
QueryMshape returns the names of all available mouse pointer shapes for the window. These can be used in the operator SetMshape. If no mouse pointers are available, the empty tuple is returned.

**Parameter**

- **WindowHandle** (input control)  
  
  Window identifier.

- **ShapeNames** (output control)  
  
  Available mouse pointer names.

**Result**

QueryMshape returns the value TRUE.

**Parallelization Information**

QueryMshape is **local** and processed **completely exclusively** without parallelization.

**Possible Predecessor Functions**

OpenWindow, OpenTextwindow, GetMshape

**Possible Successor Functions**

SetMshape

**See Also**

SetMshape, GetMshape

**Module**

System

Set the current mouse pointer shape.

SetMshape sets the shape of the mouse pointer for the window. A list of the names of all available mouse pointer shapes can be obtained by calling QueryMshape. The mouse pointer shape given by Cursor is used if the mouse pointer enters the window, irrespective of which window is the output window at present.

**Parameter**

- **WindowHandle** (input control)  
  
  Window identifier.

- **Cursor** (input control)  
  
  Mouse pointer name.  
  
  **Default Value** : `arrow`

**Result**

SetMshape returns the value TRUE if the mouse pointer shape Cursor is defined for this window. Otherwise, an exception handling is raised.

**Parallelization Information**

SetMshape is **local** and processed **completely exclusively** without parallelization.

**Possible Predecessor Functions**

OpenWindow, OpenTextwindow, QueryMshape, GetMshape

**See Also**

GetMshape, QueryMshape

**Module**

System
4.5 Output


Displays circular arcs in a window.

DispArc displays one or several circular arcs in the output window. An arc is described by its center point (CenterRow,CenterCol), the angle between start and end of the arc (Angle in radians) and the first point of the arc (BeginRow,BeginCol). The arc is displayed in clockwise direction. The parameters for output can be determined - as with the output of regions - with the procedures SetColor, SetGray, SetDraw, etc. It is possible to draw several arcs with one call by using tuple parameters. For the use of colors with several arcs, see SetColor.

Attention

The center point has to be within the window. The radius of the arc has be at least 2 pixel.

Parameter

- **WindowHandle** (input control) ........................................ window \( \sim \) HWindowX / long Window identifier.
- **CenterRow** (input control) ........................................... arc.center.y \( \sim \) VARIANT ( integer, real ) Row coordinate of center point.
  - Default Value : 64
  - Value Suggestions: CenterRow \( \in \) \{ 0, 64, 128, 256 \}
  - Typical Range of Values : \( 0 \leq \) CenterRow \( \leq \) 0(lin)
  - Minimal Value Step : 1
  - Recommended Value Step : 1
- **CenterCol** (input control) ......................................... arc.center.x \( \sim \) VARIANT ( integer, real ) Column coordinate of center point.
  - Default Value : 64
  - Value Suggestions: CenterCol \( \in \) \{ 0, 64, 128, 256 \}
  - Typical Range of Values : \( 0 \leq \) CenterCol \( \leq \) 0(lin)
  - Minimal Value Step : 1
  - Recommended Value Step : 1
- **Angle** (input control) .............................................. arc.angle.rad \( \sim \) VARIANT ( integer, real ) Angle between start and end of the arc (in radians).
  - Default Value : 3.1415926
  - Value Suggestions: Angle \( \in \) \{ 0.0, 0.785398, 1.570796, 3.1415926, 6.283185 \}
  - Typical Range of Values : \( 0.0 \leq \) Angle \( \leq \) 0.0(lin)
  - Minimal Value Step : 0.01
  - Recommended Value Step : 0.1
  - Restriction : \( (\text{Angle} > 0.0) \)
- **BeginRow** (input control) ........................................ arc.begin.y(-array) \( \sim \) VARIANT ( integer, real ) Row coordinate of the start of the arc.
  - Default Value : 32
  - Value Suggestions: BeginRow \( \in \) \{ 0, 64, 128, 256 \}
  - Typical Range of Values : \( 0 \leq \) BeginRow \( \leq \) 0(lin)
  - Minimal Value Step : 1
  - Recommended Value Step : 1
- **BeginCol** (input control) .......................................... arc.begin.x(-array) \( \sim \) VARIANT ( integer, real ) Column coordinate of the start of the arc.
  - Default Value : 32
  - Value Suggestions: BeginCol \( \in \) \{ 0, 64, 128, 256 \}
  - Typical Range of Values : \( 0 \leq \) BeginCol \( \leq \) 0(lin)
  - Minimal Value Step : 1
  - Recommended Value Step : 1
Example

open_window(0,0,-1,-1,'root','visible', '', WindowHandle)
set_draw(WindowHandle, 'fill')
set_color(WindowHandle, 'white')
set_insert(WindowHandle, 'not')
Row = 100
Column = 100
disp_arc(WindowHandle, Row, Column, 3.14, Row + 10, Column + 10)
close_window(WindowHandle).

Result

DispArc returns TRUE.

Parallelization Information

DispArc is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow, SetDraw,SetColor, SetColored, SetLineWidth, SetRgb, SetHsi

See Also

OpenWindow, OpenTextwindow, SetColor, SetDraw, SetRgb, SetHsi

Alternatives

DispCircle, DispEllipse, DispRegion, GenCircle, GenEllipse

Module

System

void \texttt{HWindowX\_DispArrow} ([\texttt{in}] \texttt{VARINT Row1}, [\texttt{in}] \texttt{VARINT Column1},
[\texttt{in}] \texttt{VARINT Row2}, [\texttt{in}] \texttt{VARINT Column2}, [\texttt{in}] \texttt{VARINT Size})

void \texttt{HOperatorSetX\_DispArrow} ([\texttt{in}] \texttt{long WindowHandle},
[\texttt{in}] \texttt{VARINT Row1}, [\texttt{in}] \texttt{VARINT Column1}, [\texttt{in}] \texttt{VARINT Row2},
[\texttt{in}] \texttt{VARINT Column2}, [\texttt{in}] \texttt{VARINT Size})

Displays arrows in a window.

DispArrow displays one or several arrows in the output window. An arrow is described by the coordinates of the start (Row1, Column1) and the end (Row2, Column2). An arrowhead is displayed at the end of the arrow. The size of the arrowhead is specified by the parameter Size. If the arrow consists of just one point (start = end) nothing is displayed. The procedures used to control the display of regions (e.g. SetDraw, SetColor, SetLineWidth) can also be used with arrows. Several arrows can be displayed with one call by using tuple parameters. For the use of colors with several arcs, see SetColor.

Attention

The start and the end of the arrows must fall within the window.

Parameter

- \texttt{WindowHandle} (input control) \hspace{1cm} \texttt{Window identifier}
- \texttt{Row1} (input control) \hspace{1cm} \texttt{Row index of the start}
  - \textbf{Default Value:} 10.0
  - \textbf{Value Suggestions:} Row1 ∈ {0.0, 64.0, 128.0, 256.0}
  - \textbf{Typical Range of Values:} 0.0 ≤ Row1 ≤ 0.0(in)
  - \textbf{Minimal Value Step:} 1.0
  - \textbf{Recommended Value Step:} 1.0
4.5. OUTPUT

- **Column1** (input control) ................................. line.begin.x(-array) ~ VARIANT (integer, real)
  
  **Default Value**: 10.0
  
  **Value Suggestions**: Column1 ∈ {0.0, 64.0, 128.0, 256.0}
  
  **Typical Range of Values**: 0.0 ≤ Column1 ≤ 0.0 (lin)
  
  **Minimal Value Step**: 1.0
  
  **Recommended Value Step**: 1.0

- **Row2** (input control) ................................. line.end.y(-array) ~ VARIANT (integer, real)
  
  **Row index of the end.**
  
  **Default Value**: 118.0
  
  **Value Suggestions**: Row2 ∈ {0.0, 64.0, 128.0, 256.0}
  
  **Typical Range of Values**: 0.0 ≤ Row2 ≤ 0.0 (lin)
  
  **Minimal Value Step**: 1.0
  
  **Recommended Value Step**: 1.0

- **Column2** (input control) ................................. line.end.x(-array) ~ VARIANT (integer, real)
  
  **Column index of the end.**
  
  **Default Value**: 118.0
  
  **Value Suggestions**: Column2 ∈ {0.0, 64.0, 128.0, 256.0}
  
  **Typical Range of Values**: 0.0 ≤ Column2 ≤ 0.0 (lin)
  
  **Minimal Value Step**: 1.0
  
  **Recommended Value Step**: 1.0

- **Size** (input control) ................................. number ~ VARIANT (integer, real)
  
  **Size of the arrowhead.**
  
  **Default Value**: 1.0
  
  **Value Suggestions**: Size ∈ {1.0, 2.0, 3.0, 5.0}
  
  **Typical Range of Values**: 0.0 ≤ Size ≤ 0.0 (lin)
  
  **Minimal Value Step**: 1.0
  
  **Recommended Value Step**: 1.0
  
  **Restriction**: (Size > 0.0)

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| set_color(WindowHandle, ['red', 'green'])
| disp_arrow(WindowHandle, [10,10], [10,10], [118,110], [118,118], 1.0).
| DispArrow returns TRUE. |

- **Parallelization Information**
  
  DispArrow is *local* and processed *completely exclusively* without parallelization.

- **Possible Predecessor Functions**
  
  OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, SetRgb, SetHsi

- **See Also**
  
  OpenWindow, OpenTextwindow, SetColor, SetDraw, SetLineWidth

- **Alternatives**
  
  DispLine, GenRegionPolygon, DispRegion

- **Module**
  
  System

```c
void HImageX.DispChannel ([in] HWindowX WindowHandle,
[in] VARIANT Channel )

void HWindowX.DispChannel ([in] HImageX MultichannelImage,
[in] VARIANT Channel )

void HOperatorSetX.DispChannel ([in] IHObjectX MultichannelImage,
[in] long WindowHandle, [in] VARIANT Channel )
```

Displays images with several channels.
DispChannel displays an image in the output window. It is possible to display several images with one call. In this case the images are displayed one after another. If the definition domains of the images overlap only the last image is visible. The parameter \texttt{Channel} defines the number of the channel that is displayed. For RGB-images the three color channels have to be used within a tuple parameter. For more information see \texttt{DispImage}.

\begin{itemize}
\item \textbf{MultichannelImage} (input iconic) \( \cdots \) \( \text{image(-array)} \sim HImageX / IHObjectX \)
  Multichannel images to be displayed.
\item \textbf{WindowHandle} (input control) \( \cdots \) \( \text{window} \sim HWindowX / long \)
  Window identifier.
\item \textbf{Channel} (input control) \( \cdots \) \( \text{integer(-array)} \sim \text{VARIANT ( integer )} \)
  Number of channel or the numbers of the RGB-channels
\end{itemize}

\begin{enumerate}
\item Default Value : 1
\item Value List : \( \text{Channel} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \)
\end{enumerate}

\begin{verbatim}
/* Transformation from rgb to gray */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
rgb1_to_gray(Image,GrayImage)
disp_image(GrayImage,WindowHandle).
\end{verbatim}

\textbf{Example}

If the used images contain valid values and a correct output mode is set, \texttt{DispChannel} returns TRUE. Otherwise an exception handling is raised.

\textbf{Parallelization Information}
DispChannel is local and processed completely exclusively without parallelization.

\textbf{Possible Predecessor Functions}
OpenWindow, SetRgb, SetLut, SetHsi
OpenWindow, OpenTextwindow, ResetObjDb, SetLut, DrawLut, DumpWindow

\textbf{Alternatives}
DispImage, DispColor

\textbf{Module}

\begin{verbatim}
void HWindowX.DispCircle ([in] VARIANT Row, [in] VARIANT Column,
[in] VARIANT Radius )
void HOperatorSetX.DispCircle ([in] long WindowHandle,
\end{verbatim}

Displays circles in a window.

DispCircle displays one or several circles in the output window. A circle is described by the center (Row, Column) and the radius Radius. If the used coordinates are not within the window the circle is clipped accordingly.

The procedures used to control the display of regions (e.g. SetDraw, SetGray, SetDraw) can also be used with circles. Several circles can be displayed with one call by using tuple parameters. For the use of colors with several circles, see SetColor.

\begin{itemize}
\item Attention
\end{itemize}

The center of the circle must be within the window.

Parameter

- **WindowHandle** (input control)  
  Window identifier.

- **Row** (input control)  
  Row index of the center.
  Default Value: 64
  Value Suggestions: Row ∈ {0, 64, 128, 256}
  Typical Range of Values: 0 ≤ Row ≤ 0(lin)
  Minimal Value Step: 1
  Recommended Value Step: 1

- **Column** (input control)  
  Column index of the center.
  Default Value: 64
  Value Suggestions: Column ∈ {0, 64, 128, 256}
  Typical Range of Values: 0 ≤ Column ≤ 0(lin)
  Minimal Value Step: 1
  Recommended Value Step: 1

- **Radius** (input control)  
  Radius of the circle.
  Default Value: 64
  Value Suggestions: Radius ∈ {0, 64, 128, 256}
  Typical Range of Values: 0 ≤ Radius ≤ 0(lin)
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: (Radius > 0.0)

Example

```cpp
open_window(0,0,-1,-1,'root','visible',,,WindowHandle)
set_draw(WindowHandle,'fill')
set_color(WindowHandle,'white')
repeat()
  get_mbutton(WindowHandle,Row,Column,Button)
  disp_circle(WindowHandle,Row,Column,(Row + Column) mod 50)
until(Button = 1)
close_window(WindowHandle).
```

Result

DispCircle returns TRUE.

Parallelization Information

DispCircle is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, SetRgb, SetHsi

See Also

OpenWindow, OpenTextwindow, SetColor, SetDraw, SetRgb, SetHsi

Alternatives

DispEllipse, DispRegion, GenCircle, GenEllipse

Module

System
Displays a color (RGB) image

DispColor displays the three channels of a color image in the output window. The channels are ordered in the sequence (red,green,blue). DispColor can be simulated by DispChannel.

Attention
Due to the restricted number of available colors the color appearance is usually different from the original.

Parameter

- **ColorImage** (input iconic) .................................................. image ~ HImageX / IHObjectX Color image to display.
- **WindowHandle** (input control) ............................................. window ~ HWindowX / long Window identifier.

Result
If the used image contains valid values and a correct output mode is set, DispColor returns TRUE. Otherwise an exception handling is raised.

Parallelization Information
DispColor is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
OpenWindow, SetRgb, SetLut, SetHsi

See Also
DispImage, OpenWindow, OpenTextwindow, ResetObjDb, SetLut, DrawLut, DumpWindow

Alternatives
DispChannel, DispObj

Displays a noise distribution.

DispDistribution displays a distribution in the window. The parameters are the same as in SetPaint(WindowHandle,'histogram') or GenRegionHisto. Noise distributions can be generated with operations like GaussDistribution or NoiseDistributionMean.

Parameter

- **WindowHandle** (input control) ............................................. window ~ HWindowX / long Window identifier.
- **Distribution** (input control) ............................................. real ~ VARIANT (real) Gray value distribution (513 values).
- **Row** (input control) ....................................................... point.y ~ long Row index of center.
  - Default Value : 256
  - Value Suggestions : Row ∈ {0, 64, 128, 256}
  - Typical Range of Values : 0 ≤ Row ≤ 0(lin)
  - Minimal Value Step : 1
  - Recommended Value Step : 10
4.5. OUTPUT

- **Column** (input control)  
  Column index of center.
  
  **Default Value**: 256  
  **Value Suggestions**: Column ∈ {0, 64, 128, 256}  
  **Typical Range of Values**: 0 ≤ Column ≤ 0 (lin)  
  **Minimal Value Step**: 1  
  **Recommended Value Step**: 10

- **Scale** (input control)  
  Size of display.
  
  **Default Value**: 1  
  **Value Suggestions**: Scale ∈ {1, 2, 3, 4, 5, 6}

---

**Example**

```c
open_window(0,0,-1,-1,'root','visible','','WindowHandle)
set_draw(WindowHandle,'fill')
set_color(WindowHandle,'white')
set_insert(WindowHandle,'not')
read_image(Image,'affe')
draw_region(Region,WindowHandle)
note_distribution_mean(Region,Image,21,Distribution)
disp_distribution (WindowHandle,Distribution,100,100,3).
```

---

**Parallelization Information**

DispDistribution is *local* and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, SetRgb, SetHsi, NoiseDistributionMean, GaussDistribution

**See Also**

GenRegionHisto, SetPaint, GaussDistribution, NoiseDistributionMean

---

**Module**

System

---

```c
void HWindowX.DispEllipse ([in] VARIANT CenterRow,
[in] VARIANT CenterCol, [in] VARIANT Phi, [in] VARIANT Radius1,
[in] VARIANT Radius2 )

void HOperatorSetX.DispEllipse ([in] long WindowHandle,
[in] VARIANT CenterRow, [in] VARIANT CenterCol, [in] VARIANT Phi,
[in] VARIANT Radius1, [in] VARIANT Radius2 )
```

**Displays ellipses.**

DispEllipse displays one or several ellipses in the output window. An ellipse is described by the center (CenterRow, CenterCol), the orientation Phi (in radians) and the radii of the minor and the major axis (Radius1 and Radius2).

The procedures used to control the display of regions (e.g. SetDraw, SetGray, SetDraw) can also be used with ellipses. Several ellipses can be displayed with one call by using tuple parameters. For the use of colors with several ellipses, see SetColor.

**Attention**

The center of the ellipse must be within the window.

**Parameter**

- **WindowHandle** (input control)  
  Window identifier.
**CenterRow** (input control) .......................... ellipse.center.y(-array)  \(\sim\) VARIANT (integer)
  Row index of center.
  Default Value : 64
  Value Suggestions : CenterRow \(\in\) \{0, 64, 128, 256\}
  Typical Range of Values : \(0 \leq\) CenterRow \(\leq\) 0
  Minimal Value Step : 1
  Recommended Value Step : 10

**CenterCol** (input control) .......................... ellipse.center.x(-array)  \(\sim\) VARIANT (integer)
  Column index of center.
  Default Value : 64
  Value Suggestions : CenterCol \(\in\) \{0, 64, 128, 256\}
  Typical Range of Values : \(0 \leq\) CenterCol \(\leq\) 0
  Minimal Value Step : 1
  Recommended Value Step : 10

**Phi** (input control) .......................... ellipse.angle.rad(-array)  \(\sim\) VARIANT (integer, real)
  Orientation of the ellipse in radians
  Default Value : 0.0
  Value Suggestions : Phi \(\in\) \{0.0, 0.785398, 1.570796, 3.1415926, 6.283185\}
  Typical Range of Values : \(0.0 \leq\) Phi \(\leq\) 0.0
  Minimal Value Step : 0.01
  Recommended Value Step : 0.1

**Radius1** (input control) .......................... ellipse.radius1(-array)  \(\sim\) VARIANT (integer, real)
  Radius of major axis.
  Default Value : 24.0
  Value Suggestions : Radius1 \(\in\) \{0.0, 64.0, 128.0, 256.0\}
  Typical Range of Values : \(0.0 \leq\) Radius1 \(\leq\) 0.0
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0

**Radius2** (input control) .......................... ellipse.radius2(-array)  \(\sim\) VARIANT (integer, real)
  Radius of minor axis.
  Default Value : 14.0
  Value Suggestions : Radius2 \(\in\) \{0.0, 64.0, 128.0, 256.0\}
  Typical Range of Values : \(0.0 \leq\) Radius2 \(\leq\) 0.0
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0

---

**Example**

```plaintext
set_color(WindowHandle,’red’)
draw_region(MyRegion,WindowHandle)
elliptic_axis(MyRegionRa,Rb,Phi)
area_center(MyRegion,-,Row,Column)
disp_ellipse(WindowHandle,Row,Column,Phi,Ra,Rb).
```

---

**Result**

DispEllipse returns TRUE, if the parameters are correct. Otherwise an exception handling is raised.

---

**Parallelization Information**

DispEllipse is *local* and processed *completely exclusively* without parallelization.

---

**Possible Predecessor Functions**

- OpenWindow
- SetDraw
- SetColor
- SetColored
- SetLineWidth
- SetRgb
- SetHsi
- EllipticAxis
- AreaCenter

---

**See Also**

- OpenWindow
- OpenTextwindow
- SetColor
- SetRgb
- SetHsi
- SetDraw
- SetLineWidth

---

**Alternatives**

- DispCircle
- DispRegion
- GenEllipse
- GenCircle

---

**Module**

**System**

Displays gray value images.

DispImage displays the gray values of an image in the output window. The gray value pixels of the definition domain (SetComprise(:WindowHandle,'object':)) or of the whole image (SetComprise(:WindowHandle,'image':)) are used. Restriction to the definition domain is the default.

For the display of gray value images the number of gray values is usually reduced. This is due to the fact that colors have to be reserved for the display of graphics (e.g. SetColor) and the window manager. Also depending on the number of bitplanes on the used output device often less than 256 colors (eight bitplanes) are available. The number of "colors" actually reserved for the display of gray values can be queried by GetSystem. Before opening the first window this value can be modified by SetSystem. For instance for 8 bitplanes 200 real gray values are the default.

The reduction of the number of gray values does not pose problems as long as only gray value information is displayed, humans cannot distinguish 256 different shades of gray. If certain gray values are used for the representation of region information (which is not the style commonly used in HALCON), confusions might be the result, since different numerical values are displayed on the screen with the same gray value. The procedure LabelToRegion should be used on these images in order to transform the label data into HALCON objects.

If images of type 'int2', 'int4', 'real' or 'complex' are displayed, the smallest and largest gray value is computed. Afterwards the pixel data is rescaled according to the number of available gray values (depending on the output device, e.g. 200). It is possible that some pixels have a very different value than the other pixels. This might lead to the display of an (almost) completely white or black image. In order to decide if the current image is a binary image MinMaxGray can be used. If necessary the image can be transformed or converted by ScaleImage and ConvertImageType before it is displayed.

Attention

If a wrong output mode was set by SetPaint, the error will be reported when DispImage is used.

Parameter

- **Image** (input iconic) ........................................ image ~ HImageX / IHObjectX
  Gray value image to display.
- **WindowHandle** (input control) ............................... window ~ HWindowX / long
  Window identifier.

Example

/* Output of a gray image: */
read_image(Image1,'affe')
disp_image(Image1,WindowHandle).

Result

If the used image contains valid values and a correct output mode is set, DispImage returns TRUE. Otherwise an exception handling is raised.

Parallelization Information

DispImage is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow, SetRgb, SetLut, SetHsi, ScaleImage, ConvertImageType, MinMaxGray

See Also

OpenWindow, OpenTextwindow, ResetObjDb, SetComprise, SetPaint, SetLut, DrawLut, PaintGray, ScaleImage, ConvertImageType, DumpWindow

Alternatives

DispObj, DispColor

Module

System

HALCON 6.0


Draws lines in a window.

DispLine displays one or several lines in the output window. A line is described by the coordinates of the start (Row1,Column1) and the coordinates of the end (Row2,Column2). The procedures used to control the display of regions (e.g. SetColor, SetGray, SetDraw, SetLineWidth) can also be used with lines. Several lines can be displayed with one call by using tuple parameters. For the use of colors with several lines, see SetColor.

**Attention**
The starting points and the ending points of the lines must be in the window.

**Parameter**

- **WindowHandle** (input control) Window identifier.

- **Row1** (input control) Row index of the start.
  - Default Value: 32
  - Value Suggestions: Row1 ∈ {0, 64, 128, 256, 511}
  - Typical Range of Values: 0 ≤ Row1 ≤ 0(lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 10

- **Column1** (input control) Column index of the start.
  - Default Value: 32
  - Value Suggestions: Column1 ∈ {0, 64, 128, 256, 511}
  - Typical Range of Values: 0 ≤ Column1 ≤ 0(lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 10

- **Row2** (input control) Row index of end.
  - Default Value: 64
  - Value Suggestions: Row2 ∈ {0, 64, 128, 256, 511}
  - Typical Range of Values: 0 ≤ Row2 ≤ 0(lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 10

- **Column2** (input control) Column index of end.
  - Default Value: 64
  - Value Suggestions: Column2 ∈ {0, 64, 128, 256, 511}
  - Typical Range of Values: 0 ≤ Column2 ≤ 0(lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 10

**Example**

```c
/* Prozedur zur Ausgabe der Kontur eines Rechtecks: */

disp_rectangle1_margin(WindowHandle,Row1,Column1,Row2,Column2):
    disp_line(WindowHandle,Row1,Column1,Row1,Column2)
    disp_line(WindowHandle,Row1,Column2,Row2,Column2)
    disp_line(WindowHandle,Row2,Column2,Row2,Column1)
    disp_line(WindowHandle,Row2,Column1,Row1,Column1).
```

DispLine returns TRUE.
4.5. OUTPUT

Parallelization Information
DispLine is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
OpenWindow, SetRgb, SetLut, SetHsi, SetDraw, SetColor, SetColored, SetLineWidth

See Also
OpenWindow, OpenTextwindow, SetColor, SetRgb, SetHsi, SetInsert, SetLineWidth

Alternatives
DispArrow, DispRectangle1, DispRectangle2, DispRegion, GenRegionPolygon, GenRegionPoints

See Also
OpenWindow, OpenTextwindow, ResetObjDb, SetComprise, SetPaint, SetLut, DrawLut, PaintGray, ScaleImage, ConvertImageType, DumpWindow

Alternatives
DispColor, DispImage, DispXld, DispRegion

Module

System

void HObjectX::DispObj ([in] HWindowX WindowHandle )
void HWindowX::DispObj ([in] IHObjectX Object )
void HOperatorSetX::DispObj ([in] IHObjectX Object , [in] long WindowHandle )

Displays image objects (image, region, XLD).

DispObj displays objects depending of their kind. DispObj is equivalent to DispImage for one channel images, equivalent to DispColor for three channel images, equivalent to DispRegion for regions and equivalent to DispXld for XLDs.

Parameter

► Object (input iconic) .........................................................object(-array) ~ IHObjectX
  Image object to be displayed.
► WindowHandle (input control) ..............................................window ~ HWindowX / long
  Window identifier.

Example

/* Output of a gray image: */
read_image(Image1, 'affe')
disp_obj(Image1, WindowHandle)
threshold(Image, Region, 0, 128)
disp_obj(Region, WindowHandle)

Result

If the used object is valid and a correct output mode is set, DispObj returns TRUE. Otherwise an exception handling is raised.

Parallelization Information
DispObj is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
OpenWindow, SetRgb, SetLut, SetHsi, ScaleImage, ConvertImageType, MinMaxGray

See Also
OpenWindow, OpenTextwindow, ResetObjDb, SetComprise, SetPaint, SetLut, DrawLut, PaintGray, ScaleImage, ConvertImageType, DumpWindow

Alternatives
DispColor, DispImage, DispXld, DispRegion

Module

System
Displays a polyline.

DispPolygon displays a polyline with the row coordinates \( Row \) and the column coordinates \( Column \) in the output window. The parameters \( Row \) and \( Column \) have to be provided as tuples. Straight lines are drawn between the given points. The start and the end of the polyline are not connected.

The procedures used to control the display of regions (e.g. SetColor, SetGray, SetDraw, SetLineWidth) can also be used with polylines.

---

### Attention

The given coordinates must lie within the window.

---

### Parameter

- **WindowHandle** (input control)  
  window \( \sim HWindowX / long \)  
  Window identifier.

- **Row** (input control)  
  polygon.y \( \sim VARIANT ( integer, real ) \)  
  Row index  
  Default Value: [16,80,80]  
  Value Suggestions: \( Row \in \{0, 64, 128, 256, 511\} \)  
  Typical Range of Values: \( 0 \leq Row \leq 0\text{(lin)} \)  
  Minimal Value Step: 1  
  Recommended Value Step: 10

- **Column** (input control)  
  polygon.x \( \sim VARIANT ( integer, real ) \)  
  Column index  
  Default Value: [48,16,80]  
  Value Suggestions: \( Column \in \{0, 64, 128, 256, 511\} \)  
  Typical Range of Values: \( 0 \leq Column \leq 0\text{(lin)} \)  
  Minimal Value Step: 1  
  Recommended Value Step: 10

---

### Result

DispPolygon returns TRUE.

---

### Parallelization Information

DispPolygon is *local* and processed *completely exclusively* without parallelization.

---

### Possible Predecessor Functions

OpenWindow, SetRgb, SetLut, SetHsi, SetDraw, SetColor, SetColored, SetLineWidth

---

### See Also

OpenWindow, OpenTextwindow, SetColor, SetRgb, SetHsi, SetInsert, SetLineWidth

---

### Alternatives

DispLine, GenRegionPolygon, DispRegion

---

### Module

System

---

Display of rectangles aligned to the coordinate axes.

DispRectangle1 displays one or several rectangles in the output window. A rectangle is described by the upper left corner (\( Row1, Column1 \)) and the lower right corner (\( Row2, Column2 \)). If the given coordinates are not within the boundary of the window the rectangle is clipped accordingly. The procedures used to control the display
of regions (e.g., SetColor, SetGray, SetDraw, SetLineWidth) can also be used with rectangles. Several rectangles can be displayed with one call by using tuple parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WindowHandle</strong></td>
<td>(input control) window ( \sim ) HWindowX/long Window identifier.</td>
</tr>
<tr>
<td><strong>Row1</strong></td>
<td>(input control) rectangle.origin.y(-array) ( \sim ) VARIANT (integer, real) ( \sim ) Row index of the upper left corner.</td>
</tr>
<tr>
<td><strong>Column1</strong></td>
<td>(input control) rectangle.origin.x(-array) ( \sim ) VARIANT (integer, real) ( \sim ) Column index of the upper left corner.</td>
</tr>
<tr>
<td><strong>Row2</strong></td>
<td>(input control) rectangle.corner.y(-array) ( \sim ) VARIANT (integer, real) ( \sim ) Row index of the lower right corner.</td>
</tr>
<tr>
<td><strong>Column2</strong></td>
<td>(input control) rectangle.corner.x(-array) ( \sim ) VARIANT (integer, real) ( \sim ) Column index of the lower right corner.</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
set_color(WindowHandle,'green')
draw_region(MyRegion,WindowHandle)
smallest_rectangle1(MyRegion,R1,C1,R2,C2)
disp_rectangle1(WindowHandle,R1,C1,R2,C2).
```

**Result**

DispRectangle1 returns TRUE.

**Parallelization Information**

DispRectangle1 is *local* and processed *completely exclusively* without parallelization.

**Possible Predecessor Functions**

OpenWindow, SetRgb, SetLut, SetHsi, SetDraw, SetColor, SetColored, SetLineWidth

**See Also**

OpenWindow, OpenTextwindow, SetColor, SetDraw, SetLineWidth

**Alternatives**

DispRectangle2, GenRectangle1, DispRegion, DispLine, SetShape

**Module**

System

HALCON 6.0


Displays arbitrarily oriented rectangles.

DispRectangle2 draws one or several arbitrarily oriented rectangles in the output window. A rectangle is described by the center (CenterRow,CenterCol), the orientation Phi (in radians) and half the lengths of the edges Length1 and Length2. The procedures used to control the display of regions (e.g. SetDraw, SetGray, SetDraw) can also be used with rectangles. Several rectangles can be displayed with one call by using tuple parameters. For the use of colors with several rectangles, see SetColor.

Attention

The center must lie within the window boundaries.

Parameter

- **WindowHandle** (input control) ................................................. window \( \sim \) HWindowX / long Window identifier.
- **CenterRow** (input control) ............................................ rectangle2.center.y(-array) \( \sim \) VARIANT ( integer, real ) Row index of the center.
  - Default Value : 48
  - Value Suggestions : CenterRow \( \in \) \( \{ \) 0, 64, 128, 256, 511 \( \} \)
  - Typical Range of Values : \( 0 \leq \) CenterRow \( \leq 0 \) (lin)
  - Minimal Value Step : 1
  - Recommended Value Step : 10
- **CenterCol** (input control) ........................................... rectangle2.center.x(-array) \( \sim \) VARIANT ( integer, real ) Column index of the center.
  - Default Value : 64
  - Value Suggestions : CenterCol \( \in \) \( \{ \) 0, 64, 128, 256, 511 \( \} \)
  - Typical Range of Values : \( 0 \leq \) CenterCol \( \leq 0 \) (lin)
  - Minimal Value Step : 1
  - Recommended Value Step : 10
- **Phi** (input control) .................................................. rectangle2.angle.rad(-array) \( \sim \) VARIANT ( integer, real ) Orientation of rectangle in radians.
  - Default Value : 0.0
  - Value Suggestions : Phi \( \in \) \( \{ \) 0.0, 0.785398, 1.570796, 3.1415926, 6.283185 \( \} \)
  - Typical Range of Values : \( 0.0 \leq \) Phi \( \leq 0.0 \) (lin)
  - Minimal Value Step : 0.01
  - Recommended Value Step : 0.1
- **Length1** (input control) .............................................. rectangle2.hwidth(-array) \( \sim \) VARIANT ( integer, real ) Half of the length of the longer side.
  - Default Value : 48
  - Value Suggestions : Length1 \( \in \) \( \{ \) 0, 64, 128, 256, 511 \( \} \)
  - Typical Range of Values : \( 0 \leq \) Length1 \( \leq 0 \) (lin)
  - Minimal Value Step : 1
  - Recommended Value Step : 10
- **Length2** (input control) .............................................. rectangle2.hheight(-array) \( \sim \) VARIANT ( integer, real ) Half of the length of the shorter side.
  - Default Value : 32
  - Value Suggestions : Length2 \( \in \) \( \{ \) 0, 64, 128, 256, 511 \( \} \)
  - Typical Range of Values : \( 0 \leq \) Length2 \( \leq 0 \) (lin)
  - Minimal Value Step : 1
  - Recommended Value Step : 10
  - **Restriction** : \( Length2 \leq Length1 \)

Example
4.5. OUTPUT 223

set_color(WindowHandle,'green')
draw_region(MyRegion:WindowHandle)
elliptic_axis(MyRegion,Ra,Rb,Phi)
area_center(MyRegion,_,Row,Column)
disp_rectangle2(WindowHandle,Row,Column,Phi,Ra,Rb).

Result
DispRectangle2 returns TRUE, if the parameters are correct. Otherwise an exception handling is raised.

Parallelization Information
DispRectangle2 is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
OpenWindow, SetRgb, SetLut, SetHsi, SetDraw, SetColor, SetColored, SetLineWidth

See Also
OpenWindow, OpenTextwindow, DispRegion, SetColor, SetDraw, SetLineWidth

Alternatives
DispRegion, GenRectangle2, DispRectangle1, SetShape

Module
System

void HRegionX.DispRegion ([in] HWindowX WindowHandle )
void HWindowX.DispRegion ([in] HRegionX DispRegions )
void HOperatorSetX.DispRegion ([in] IObjectX DispRegions, [in] long WindowHandle )

Displays regions in a window.
DispRegion displays the regions in DispRegions in the output window. The parameters for output can be set with the procedures SetColor, SetGray, SetDraw, SetLineWidth, etc.
The color(s) for the display of the regions are determined with SetColor, SetRgb, SetGray or SetColored. If more than one region is displayed and more than one color is set, the colors are assigned in a cyclic way to the regions.
The form of the region for output can be modified by SetPaint (e.g. encompassing circle, convex hull). The command SetDraw determines if the region is filled or only the boundary is drawn. If only the boundary is drawn, the thickness of the boundary will be determined by SetLineWidth and the style by SetLineStyle.

Parameter

▷ DispRegions (input iconic) .................................. region(-array) 〜 HRegionX / IObjectX
Regions to display.
▷ WindowHandle (input control) .................................... window 〜 HWindowX / long
Window identifier.

Example

/* Output with 12 colors: */
set_colored(WindowHandle,12)
disp_region(SomeSegments,WindowHandle).

/* Symbolic representation: */
set_draw(WindowHandle,'margin')
set_color(WindowHandle,'red')
set_shape(WindowHandle,'ellipse')
disp_region(SomeSegmentsWindowHandle).

/* Representation of a margin with pattern: */
set_draw(WindowHandle,'margin')
set_color(WindowHandle,'blue')
set_line_style(WindowHandle,[12,3])
disp_region(Segments,WindowHandle).

---

**Result**

DispRegion returns TRUE.

---

**Parallelization Information**

DispRegion is local and processed completely exclusively without parallelization.

---

**Possible Predecessor Functions**

OpenWindow, SetRgb, SetLut, SetHsi, SetShape, SetLineStyle, SetInsert, SetFix,
SetDraw, SetColor, SetColored, SetLineWidth

---

**See Also**

OpenWindow, OpenTextwindow, SetColor, SetColored, SetDraw, SetShape, SetPaint,
SetGray, SetRgb, SetHsi, SetPixel, SetLineWidth, SetLineStyle, SetInsert, SetFix,
PaintRegion, DumpWindow

---

**Alternatives**

DispObj, DispArrow, DispLine, DispCircle, DispRectangle1, DispRectangle2,
DispEllipse

---

**Module**

Sub-pixel operators

---

**4.6 Parameters**

```
void HXLDX.DispXld ([in] HWindowX WindowHandle )
void HWindowX.DispXld ([in] HXLDX XLDObject )
void HOperatorSetX.DispXld ([in] IHObjectX XLDObject, [in] long WindowHandle )
```

Display an XLD object.

DispXld serves to display an XLD object of arbitrary type.

---

**Parameter**

- **XLDObject** (input iconic) ................................. xld \( \sim \) HXLDX / IHObjectX
  XLD object to display.
- **WindowHandle** (input control) .......................... window \( \sim \) HWindowX / long
  Window id.

---

**Parallelization Information**

DispXld is local and processed completely exclusively without parallelization.

---

**See Also**

DispImage, DispRegion, DispChannel, DispColor, DispLine, DispArc

---

**Module**

Sub-pixel operators

---

```
[out] String Mode HInfoX.GetComprise ([in] HWindowX WindowHandle )
void HOperatorSetX.GetComprise ([in] long WindowHandle, [out] String Mode )
```

Get the output treatment of an image matrix.

GetComprise returns the output mode of grayvalues in the window WindowHandle that is used by
DispImage and DispColor. The output mode defines whether only the grayvalues of objects are displayed or
the whole image is displayed. The query is used for temporary mode settings, i.e., the current mode is queried, then overwritten with \( \text{SetComprise} \) and finally reset.

\[ \text{Attention} \]

\[ \text{Parameter} \]

\[ \text{WindowHandle} \] (input control) \hspace{1em} \text{window} \sim \text{HWindowX} / \text{long WindowId}.
\[ \text{Mode} \] (output control) \hspace{1em} \text{string} \sim \text{String}

\[ \text{Display mode for images.} \]

\[ \text{Result} \]

GetComprise returns TRUE if the window is valid. Otherwise an exception handling is raised.

\[ \text{Parallelization Information} \]

GetComprise is processed completely exclusively without parallelization.

\[ \text{Possible Successor Functions} \]

SetComprise, \( \text{DispImage}, \text{DispImage} \)

\[ \text{See Also} \]

SetComprise, \( \text{DispImage}, \text{DispImage} \)

\[ \text{Module} \]

\[ \text{System} \]

\[
\text{[out]} \text{String Mode} \hspace{1em} \text{HWindowX.GetDraw ( )} \\
\text{void HOperatorSetX.GetDraw ([in] long WindowHandle, [out] String Mode )}
\]

Get the current region fill mode.

GetDraw returns the region fill mode of the output window. It is used by operators as \( \text{DispRegion}, \text{DispCircle}, \text{DispArrow}, \text{DispRectangle1}, \text{DispRectangle2} \) etc. The region fill mode is set with \( \text{SetDraw} \).

\[ \text{Attention} \]

\[ \text{Parameter} \]

\[ \text{WindowHandle} \] (input control) \hspace{1em} \text{window} \sim \text{HWindowX} / \text{long WindowId}.
\[ \text{Mode} \] (output control) \hspace{1em} \text{string} \sim \text{String}

\[ \text{Current region fill mode.} \]

\[ \text{Result} \]

GetDraw returns TRUE, if the window is valid. Otherwise an exception handling is raised.

\[ \text{Parallelization Information} \]

GetDraw is processed completely exclusively without parallelization.

\[ \text{Possible Successor Functions} \]

SetDraw, \( \text{DispRegion} \)

\[ \text{See Also} \]

SetDraw, \( \text{DispRegion}, \text{SetPaint} \)

\[ \text{Module} \]

\[ \text{System} \]

\[
\text{[out]} \text{String Mode} \hspace{1em} \text{HWindowX.GetFix ( )} \\
\text{void HOperatorSetX.GetFix ([in] long WindowHandle, [out] String Mode )}
\]

Get mode of fixing of current look-up-table (lut).
Use `GetFix` to get mode of fixing of current look-up-table (look-up-table of valid window) set before by `SetFix`.

- **Attention**

- **Parameter**

  - **WindowHandle** (input control) \[\text{window} \sim \text{HWindowX / long}\]
    Window identifier.

  - **Mode** (output control) \[\text{string} \sim \text{String}\]
    Current Mode of fixing.

- **Result**

  `GetFix` returns TRUE if the window is valid. Otherwise an exception handling is raised.

- **Parallelization Information**

  `GetFix` is processed `completely exclusively` without parallelization.

- **Possible Successor Functions**

  `SetFix, SetPixel, SetRgb`

  - **See Also**

  `SetFix`

  - **Module**

  `System`

```c
[out] VARIANT Hue HWindowX.GetHsi ([out] VARIANT Saturation,
[out] VARIANT Intensity )
```

Get the HSI coding of the current color.

`GetHsi` returns the output color or grayvalues, respectively, for the window, described in `Hue, Saturation` and `Intensity`. `GetHsi` corresponds to the procedure `GetPixel` but returns the entries of the color lookup table instead of its indices. The values returned by `GetHsi` can be set with `SetHsi`.

- **Attention**

  The values returned by `GetHsi` may be inaccurate due to rounding errors. They do not necessarily match the values set with `SetHsi` exactly (colors are stored in RGB internally).

- **Parameter**

  - **WindowHandle** (input control) \[\text{window} \sim \text{HWindowX / long}\]
    Window _id_.

  - **Hue** (output control) \[\text{integer} \sim \text{VARIANT( integer )}\]
    Hue (color value) of the current color.

  - **Saturation** (output control) \[\text{integer} \sim \text{VARIANT( integer )}\]
    Saturation of the current color.

  - **Intensity** (output control) \[\text{integer} \sim \text{VARIANT( integer )}\]
    Intensity of the current color.

- **Result**

  `GetHsi` returns TRUE, if the window is valid. Otherwise an exception handling is raised.

- **Parallelization Information**

  `GetHsi` is processed `completely exclusively` without parallelization.

- **Possible Successor Functions**

  `SetHsi, SetRgb, DispImage`

  - **See Also**

  `SetHsi, SetColor, SetRgb, TransToRgb, TransFromRgb`

  - **Module**

  `System`

void HRegionX.GetIcon ([in] HWindowX WindowHandle )

[out] HRegionX Icon HWindowX.GetIcon ( )

void HOperatorSetX.GetIcon ([out] HUntypedObjectX Icon, [in] long WindowHandle )

Query the icon for region output

GetIcon queries the icon that was set with SetIcon.

Parameter

- **Icon** (output iconic): region ~ HRegionX / HUntypedObjectX
- **WindowHandle** (input control): window ~ HWindowX / long

Result

GetIcon always returns TRUE.

Parallelization Information

GetIcon is processed *completely exclusively* without parallelization.

Possible Predecessor Functions

SetIcon

Possible Successor Functions

DispRegion

System

[out] String Mode HWindowX.GetInsert ( )

void HOperatorSetX.GetInsert ([in] long WindowHandle, [out] String Mode )

Get the current display mode.

GetInsert returns the display mode of the output window. It is used by procedures like DispRegion, DispLine, DispRectangle, etc. The mode is set with SetInsert. Possible values for Mode can be queried with the procedure QueryInsert.

Attention

Parameter

- **WindowHandle** (input control): window ~ HWindowX / long
- **Mode** (output control): string ~ String

Result

GetInsert returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetInsert is processed *completely exclusively* without parallelization.

Possible Predecessor Functions

QueryInsert

Possible Successor Functions

SetInsert, DispImage

See Also

SetInsert, QueryInsert, DispRegion, DispLine

Module

System
Get the current approximation error for contour display.

GetLineApprox returns a parameter that controls the approximation error for region contour display in the window. It is used by the procedure DispRegion. Approximation controls the polygon approximation for contour display (0 ↔ no approximation). Approximation is only important for displaying the contour of objects, especially if a line style was set with SetLineStyle.

\[
\text{Attention}
\]

Parameter

\begin{itemize}
  \item WindowHandle (input control) \( \text{window } \sim \text{HWindowX/long Window\_id}. \)
  \item Approximation (output control) \( \text{Current approximation error for contour display.} \)
\end{itemize}

Result

GetLineApprox returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetLineApprox is processed completely exclusively without parallelization.

Possible Successor Functions

SetLineApprox, SetLineStyle, DispRegion

See Also

GetRegionPolygon, SetLineApprox, SetLineStyle, DispRegion

Module

System

Get the current graphic mode for contours.

GetLineStyle returns the display mode for contours when displaying regions. It is used by procedures like DispRegion, DispLine, DispPolygon, etc. Style is set with the procedure SetLineStyle. Style is only important for displaying the contour of objects, especially if a line style was set with SetLineStyle.

\[
\text{Attention}
\]

Parameter

\begin{itemize}
  \item WindowHandle (input control) \( \text{window } \sim \text{HWindowX/long Window\_id}. \)
  \item Style (output control) \( \text{Template for contour display.} \)
\end{itemize}

Result

GetLineStyle returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetLineStyle is local and processed completely exclusively without parallelization.

See Also

SetLineStyle, DispRegion

Module

System
Get the current line width for contour display.

GetLineWidth returns the line width for region display in the window. It is used by procedures like DispRegion, DispLine, DispPolygon, etc. Width is set with the procedure SetLineWidth. Width is only important for displaying the contour of objects.

Attention

Parameter

WindowHandle (input control) window ~ HWindowX / long WindowId.

Width (output control) integer ~ long

Current line width for contour display.

Result

GetLineWidth returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetLineWidth is processed completely exclusively without parallelization.

Possible Successor Functions

SetLineWidth, SetLineStyle, DispRegion

SetLineWidth, DispRegion

See Also

Module

System

Get the current display mode for grayvalues.

GetPaint returns the display mode for grayvalues in the window. Mode is used by the procedure DispImage. GetPaint is used for temporary changes of the grayvalue display mode. The current value is queried, then changed (with procedure SetPaint) and finally the old value is written back. The available modes can be viewed with the procedure QueryPaint. Mode is the name of the display mode. If a mode can be customized with parameters, the parameter values are passed in a tuple after the mode name. The order of values is the same as in SetPaint.

Attention

Parameter

WindowHandle (input control) window ~ HWindowX / long WindowId.

Mode (output control) string ~ VARIANT ( string, integer )

Name and parameter values of the current display mode.

Result

GetPaint returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetPaint is processed completely exclusively without parallelization.

Possible Predecessor Functions

QueryPaint
Possible Successor Functions

SetPaint, DispRegion, DispImage

See Also

SetPaint, QueryPaint, DispImage

Module

System

```
[out] long Row1 HWindowX.GetPart ([out] long Column1, [out] long Row2,
[out] long Column2 )
```

```
void HOperatorSetX.GetPart ([in] long WindowHandle, [out] long Row1,
[out] long Column1, [out] long Row2, [out] long Column2 )
```

Get the image part.

GetPart returns the upper left and lower right corner of the image part shown in the window. The image part can be changed with the procedure SetPart (Default is the whole image).

Attention

Parameter

- WindowHandle (input control) .................................. window \sim HWindowX / long WindowId.
- Row1 (output control) ........................................... rectangle.origin.y \sim long Row index of the image part’s upper left corner.
- Column1 (output control) ....................................... rectangle.origin.x \sim long Column index of the image part’s upper left corner.
- Row2 (output control) ........................................... rectangle.corner.y \sim long Row index of the image part’s lower right corner.
- Column2 (output control) ..................................... rectangle.corner.x \sim long Column index of the image part’s lower right corner.

Result

GetPart returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetPart is processed completely exclusively without parallelization.

Possible Successor Functions

SetPart, DispRegion, DispImage

See Also

SetPart, DispImage, DispRegion, DispColor

Module

System

```
[out] long Style HWindowX.GetPartStyle ( )
```

```
void HOperatorSetX.GetPartStyle ([in] long WindowHandle, 
[out] long Style )
```

Get the current interpolation mode for grayvalue display.

GetPartStyle returns the interpolation mode used for displaying an image part in the window. An interpolation takes place if the output window is larger than the image format or the image output format (see SetPart).

HALCON supports three interpolation modes:

- 0 no interpolation (low quality, very fast).
- 1 unweighted interpolation (average quality and computation time)
2 weighted interpolation (high quality, slow)

The current mode can be changed with SetPartStyle.

Parameter

- **WindowHandle** (input control) window ~ HWindowX / long
- **Style** (output control) Interpolation mode for image display: 0 (fast, low quality) to 2 (slow, high quality).

Value List: Style ∈ {0, 1, 2}

Result

GetPartStyle returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetPartStyle is processed completely exclusively without parallelization.

Possible Successor Functions

SetPartStyle, DispRegion, DispImage

See Also

SetPartStyle, SetPart, DispImage, DispColor

Module

System

```
[out] VARIANT Pixel HWindowX.GetPixel ()

void HOperatorSetX.GetPixel ([in] long WindowHandle,
[out] VARIANT Pixel )
```

Get the current color lookup table index.

GetPixel returns the internal coding of the output grayvalue or color, respectively, for the window. If the output mode is set to color(s) or grayvalue(s) (see SetColor or SetGray), then the color- or grayvalues are transformed for internal use. The internal code is then used for (physical) screen display. The transformation depends on the mapping characteristics and the condition of the output device and can be different in different program runs. Don’t confuse the term “’pixel’” with the term “’pixel’” in image processing (the other procedure is GetGrayval). Here a pixel is meant to be the color lookup table index.

With GetPixel it is possible to save the output mode without knowing whether colors or grayvalues are used. Pixel is set with the procedure SetPixel.

Parameter

- **WindowHandle** (input control) window ~ HWindowX / long
- **Pixel** (output control) Index of the current color lookup table.

Result

GetPartStyle returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetPixel is processed completely exclusively without parallelization.

Possible Successor Functions

SetPixel, DispRegion, DispImage

See Also

SetPixel, SetFix

Module

System
Get the current color in RGB-coding.

GetRgb returns the output colors or grayvalues, respectively, for the output window. They are defined by the three color components red, green and blue.

GetRgb is like GetPixel but returns the entries of the color lookup table rather than the indices. The values returned by GetRgb can be set with SetRgb.

### Attention

#### Parameter

- **WindowHandle** (input control) ... window \( \sim \) HWindowX / long Window_id.
- **Red** (output control) ... integer \( \sim \) VARIANT ( integer )
  - The current color's red value.
- **Green** (output control) ... integer \( \sim \) VARIANT ( integer )
  - The current color's green value.
- **Blue** (output control) ... integer \( \sim \) VARIANT ( integer )
  - The current color's blue value.

#### Result

GetRgb returns TRUE if the window is valid. Otherwise an exception handling is raised.

### Parallelization Information

GetRgb is processed **completely exclusively** without parallelization.

### Possible Successor Functions

SetRgb, DispRegion, DispImage

---

Get the current region output shape.

GetShape returns the shape in which regions are displayed. The available shapes can be queried with QueryShape and then changed with SetShape.

### Attention

#### Parameter

- **WindowHandle** (input control) ... window \( \sim \) HWindowX / long Window_id.
- **DisplayShape** (output control) ... string \( \sim \) String
  - Current region output shape.

#### Result

GetShape returns TRUE if the window is valid. Otherwise an exception handling is raised.

### Parallelization Information

GetShape is processed **completely exclusively** without parallelization.
Possible Predecessor Functions

QueryShape

Possible Successor Functions

SetShape, DispRegion

See Also

SetShape, QueryShape, DispRegion

Module

System

---

**Query all color names.**

QueryAllColors returns the names of all colors that are known to HALCON. That doesn’t mean, that these colors are available for specific screens. On some screens there may only be a subset of colors available (see QueryColor). Before opening the first window, SetSystem can be used to define which and how many colors should be used. The HALCON colors are used to display regions (DispRegion, DispPolygon, DispCircle, etc.). They can be defined with SetColor.

Attention

Parameter

- **WindowHandle** (input control) 
- **Colors** (output control)

Example

```c
query_all_colors(WindowHandle, Colors)
set_system('graphic_colors', ActColors)
open_window(0,0,1,'root','invisible','root',WindowHandle)
query_color(WindowHandle,F)
close_window(WindowHandle)
fwrite_string(['Setting Colors: ',F]).
```

Result

QueryAllColors always returns TRUE

Parallelization Information

QueryAllColors is local and processed completely exclusively without parallelization.

Possible Successor Functions

SetSystem, SetColor, DispRegion

See Also

QueryColor, SetSystem, SetColor, DispRegion, OpenWindow, OpenTextWindow

Module

System

---

**Query all color names displayable in the window.**

```c
void HOperatorSetX.QueryAllColors ([in] long WindowHandle, [out] VARIANT Colors )
```

```
void HWindowX.QueryAllColors ()
```

HALCON 6.0
QueryColor returns the names of all colors that are usable for region output (DispRegion, DispPolygon, DispCircle, etc.). On a b/w screen QueryColor returns 'black' and 'white'. These two "colors" are displayable on any screen. In addition to 'black' and 'white' several grayvalues (e.g. 'dim gray') are returned on screens capable of grayvalues. A list of all displayable colors is returned for screens with color lookup table. The returned tuple of colors begins with b/w, followed by the three primaries ('red','green','blue') and several grayvalues. Before opening the first window it is furthermore possible to define the color list with SetSystem( : 'graphicColors',...:). QueryAllColors<::WindowHandle::Colors ) returns a list of all available colors for the SetSystem( :: 'graphicColors',...:) call. For screens with truecolor output the same list is returned by QueryColor. The list of available colors (to HALCON ) must not be confused with the list of displayable colors. For screens with truecolor output the available colors are only a small subset of the displayable colors. Colors that are not directly available to HALCON can be chosen manually with SetRgb or SetHsi. If colors are chosen that are known to HALCON but cannot be displayed, HALCON can choose a similar color. To use this feature, SetCheck( :: '˜color': ) must be set.

**Attention**

**Parameter**

- **WindowTitle** (input control) .............................. window ～ HWindow long WindowId.
- **Colors** (output control) ................................. string ～ VARIANT ( string )
  Color names.

**Example**

```c
open_window(0,0,-1,-1,'root','invisible',' ',WindowHandle)
query_color(WindowHandle,Colors)
close_window(WindowHandle)
fwrite_string(["Displayable colors: ',Farben"]).
```

**Result**

QueryColor returns TRUE, if the window is valid. Otherwise an exception handling is raised.

**Parallelization Information**

QueryColor is local and processed completely exclusively without parallelization.

**Possible Successor Functions**

SetColor, DispRegion

**See Also**

QueryAllColors, SetColor, DispRegion, OpenWindow, OpenTextLabel

**Module**

System

[out] VARIANT PossibleNumberOfColors HInfoX.QueryColored ( )

void HOperatorSetX.QueryColored ([out] VARIANT PossibleNumberOfColors )

Query the number of colors for color output.

QueryColored returns all possible parameter values for SetColored. SetColored defines how many colors are used for region or graphics output.

**Attention**

**Parameter**

- **PossibleNumberOfColors** (output control) ................ integer ～ VARIANT ( integer )
  Tuple of the possible numbers of colors.

Example

region_growing(Image, Seg, 5, 5, 6, 100)
query_colored(Colors)
set_colored(WindowHandle, Colors[1])
disp_region(Seg, WindowHandle).

Result

QueryColored always returns TRUE.

Parallelization Information

QueryColored is reentrant and processed without parallelization.

Possible Successor Functions

SetColored, SetColor, DispRegion

See Also

SetColored, SetColor

Alternatives

QueryColor

Module

System

[out] VARIANT Grayval HWindowX.QueryGray ( )
void HOperatorSetX.QueryGray ([in] long WindowHandle,
[out] VARIANT Grayval )

Query the displayable grayvalues.

QueryGray returns all grayvalues that are used for grayvalue output (DispImage) and that can be reproduced exactly in the window. They can be set with the SetGray call. The number of displayable grayvalues can be set with SetSystem (:numGray*, ...:) before opening the first window.

Attention

Parameter

- WindowHandle (input control) ........................ window ~ HWindowX/long Window_id.
- Grayval (output control) .............................. integer ~ VARIANT ( integer )
  Tuple of all displayable grayvalues.

Result

QueryGray returns TRUE, if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

QueryGray is local and processed completely exclusively without parallelization.

Possible Successor Functions

SetGray, DispRegion

See Also

SetGray, DispImage

Module

System

[out] VARIANT Mode HWindowX.QueryInsert ( )
void HOperatorSetX.QueryInsert ([in] long WindowHandle,
[out] VARIANT Mode )

Query the possible graphic modes.
QueryInsert returns the possible modes pixels can be displayed in the output window. New pixels may e.g. overwrite old ones. In most of the cases there is a functional relationship between old and new values.

Possible display functions:

'copy': overwrite displayed pixels
'xor': display old “xor” new pixels
'not': complement displayed pixels

"copy" is always available.

---

### Attention

---

### Parameter

- **WindowHandle** (input control)  
  - **Mode** (output control)

---

### Result

QueryInsert returns TRUE, if the window is valid. Otherwise an exception handling is raised.

---

### Parallelization Information

QueryInsert is local and processed completely exclusively without parallelization.

---

### Possible Successor Functions

SetInsert, DispRegion
SetsInsert, GetInsert

---

### See Also

Module

---

**[out] long Min HInfoX.QueryLineWidth ([out] long Max )**

void HOperatorSetX.QueryLineWidth ([out] long Min, [out] long Max )

---

Query the possible line widths.

QueryLineWidth returns the minimal (Min) and maximal (Max) values of widths of region border which can be displayed. Setting of the border width is done with SetLineWidth. Border width is used by operators like DispRegion, DispLine, DispCircle, DispRectangle1, DispRectangle2 etc. if the drawing mode is "margin" (SetDraw(‘WindowHandle,’margin’)).

---

### Attention

---

### Parameter

- **Min** (output control)
- **Max** (output control)

---

### Result

QueryLineWidth returns TRUE if the window is valid. Otherwise an exception handling is raised.

---

### Parallelization Information

QueryLineWidth is reentrant and processed without parallelization.

---

### Possible Successor Functions

GetLineWidth, SetLineWidth, SetLineStyle, DispLine

---

### See Also

DispCircle, DispLine, DispRectangle1, DispRectangle2, DispRegion, SetLineWidth, GetLineWidth, SetLineStyle
4.6. PARAMETERS

System

[out] VARIANT Mode HWindowX.QueryPaint ( )

void HOperatorSetX.QueryPaint ([in] long WindowHandle,
[out] VARIANT Mode )

Query the grayvalue display modes.

QueryPaint returns the names of all grayvalue display modes (e.g. 'gray', '3D-plot', 'contourline', etc.) for the output window. These modes are used by SetPaint. QueryPaint only returns the names of the display values, not the additional parameters that may be necessary for some modes.

Attention

Parameter

▷ WindowHandle (input control) ................................................... window ~ HWWindowX/long WindowId.
▷ Mode (output control) ...................................................... string ~ VARIANT( string )

Grayvalue display mode names.

Result

QueryPaint returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

QueryPaint is local and processed completely exclusively without parallelization.

Possible Successor Functions

GetPaint, SetPaint, DispImage

See Also

SetPaint, GetPaint, DispImage

System

[out] VARIANT DisplayShape HInfoX.QueryShape ( )

void HOperatorSetX.QueryShape ([out] VARIANT DisplayShape )

Query the region display modes.

QueryShape returns the names of all region display modes (e.g. 'original', 'circle', 'rectangle1', 'rectangle2', 'ellipse', etc.) for the window. They are used by SetShape.

Attention

Parameter

▷ DisplayShape (output control) ............................................. string ~ VARIANT( string )

Region display mode names.

Result

QueryShape returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

QueryShape is reentrant and processed without parallelization.

Possible Successor Functions

GetShape, SetShape, DispRegion

See Also

SetShape, GetShape, DispRegion

System
Set output color.

SetColor defines the colors for region output in the window. The available colors can be queried with the procedure QueryColor. The "colors" 'black' and 'white' are available for all screens. If colors are used that are not displayable on the screen, HALCON can choose a similar, displayable color of the output. For this, SetCheck(::{'color':}) must be called. Furthermore, the list of available colors can be set with the procedure SetSystem(::{'graphicColors',...:}). That must be done before opening the first output window. If only a single color is passed, all output is in this color. If a tuple of colors is passed, the output color of regions is modulo to the number of colors. In the example below, the first circle is displayed red, the second in green and the third in red again. HALCON always begins output with the first color passed. Note, that the number of output colors depends on the number of objects that are displayed in one procedure call. If only single objects are displayed, they always appear in the first color, even if the consist of more than one connected components.

The defined colors are used until SetColor, SetPixel, SetRgb, SetHsi or SetGray is called again.

Colors are defined separately for each window. They can only be changed for the valid window. Color is used in procedures with region output like DispRegion, DispLine, DispRectangle1, DispArrow etc. It is also used by procedures with grayvalue output in certain output modes (e.g. '3D-plot', 'histogram', 'contourline', etc. See SetPaint).

---

**Attention**

---

**Parameter**

- **WindowHandle** (input control) .............................................. window ~ HWindowX / long Window_id.
- **Color** (input control) ...................................................... string(-array) ~ VARIANT( string )
  
  Output color names.
  Default Value: 'white'

---

**Example**

```
set_color(WindowHandle, ['red', 'green'])
disp_circle(WindowHandle, [100, 200, 300], [200, 300, 100], [100, 100, 100]).
```

---

**Result**

SetColor returns TRUE if the window is valid and the passed colors are displayable on the screen. Otherwise an exception handling is raised.

---

**Parallelization Information**

SetColor is local and processed completely exclusively without parallelization.

---

**Possible Predecessor Functions**

QueryColor

---

**Possible Successor Functions**

DispRegion

---

See Also

GetRgb, DispRegion, SetFix, SetPaint

---

**Alternatives**

SetRgb, SetHsi

---

**Module**

System
Set multiple output colors.

SetColored is a shortcut for certain SetColor calls. It allows the user to display a region set in different colors. NumberOfColors defines the number of colors that are used. Valid values for NumberOfColors can be queried with QueryColored. Furthermore, the list of available colors can be set with the procedure SetSystem(‘graphicColors’,...:). That must be done before opening the first output window.

```c
void HWindowX.SetColored ([in] long NumberOfColors )
void HOperatorSetX.SetColored ([in] long WindowHandle, [in] long NumberOfColors )
```

SetColored returns TRUE if NumberOfColors is correct and the window is valid. Otherwise an exception handling is raised.

**Example**

```c
def set_colored(WindowHandle,Num):
    if(Num == 3) > set_color(WindowHandle,[red,green,blue]) |
    if(Num == 6) > set_color(WindowHandle,[red,green,blue,cyan,magenta,yellow]).
```

**Attention**

Parameter

▶ **WindowHandle** (input control) .............................................. window ~ HWindowX / long Window.id.
▶ **NumberOfColors** (input control) ................................................ integer ~ long Number of output colors.

Default Value : 12

**Parallelization Information**

SetColored is local and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

QueryColored, SetColor

**Possible Successor Functions**

DispRegion

**See Also**

QueryColored, SetColor, DispRegion

**Module**

System

Define the image matrix output clipping.

SetComprise defines the image matrix output clipping. If Mode is set to ‘object’, only grayvalues belonging to the output object are displayed. If set to ‘image’, the whole image matrix is displayed. Default is ‘object’.

```c
void HWindowX.SetComprise ([in] String Mode )
void HOperatorSetX.SetComprise ([in] long WindowHandle, [in] String Mode )
```

SetComprise returns TRUE if Mode is correct and the window is valid. Otherwise an exception handling is raised.

**Attention**

If Mode was set to ‘image’, undefined grayvalues may be displayed. Depending on the context they are black or can have random content. See the examples.
Parameter

- **WindowHandle** (input control)  
  WindowHandle \( \leadsto HWindowX / long \)

- **Mode** (input control)  
  Mode \( \leadsto string \)

  **Default Value:** 'object'

  **Value List:** Mode ∈ {`image`, `object`}

Example

```plaintext
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
read_image(Image,'fabrik')
threshold(Image,Seg,100,255)
set_system('init_new_image','false')
sobel_amp(Image,Sob,'sum_abs',3)
disp_image(Sob,WindowHandle)
get_comprise(Mode)
fwrite_string(['Current mode for gray values: ',Mode])
fnewline()
set_comprise(WindowHandle,'image')
get_mbutton(WindowHandle,_,_,_)
disp_image(Sob,WindowHandle)
fwrite_string(['Current mode for gray values: image'])
fnewline().
```

Result

SetComprise returns TRUE if `Mode` is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information

SetComprise is processed completely exclusively without parallelization.

Possible Predecessor Functions

- GetComprise

Possible Successor Functions

- DispImage

See Also

- GetComprise, DispImage, DispColor

Module

System

```plaintext
void HWindowX.SetDraw ([in] String Mode )

void HOperatorSetX.SetDraw ([in] long WindowHandle, [in] String Mode )
```

Define the region fill mode.

SetDraw defines the region fill mode. If `Mode` is set to 'fill', output regions are filled, if set to 'margin', only contours are displayed. Setting `Mode` only affects the valid window. It is used by procedures with region output like DispRegion, DispCircle, DispRectangle1, DispRectangle2, DispArrow etc. It is also used by procedures with grayvalue output for some grayvalue output modes (e.g. 'histogram', see SetPaint). If the mode is 'margin', the contour can be affected with SetLineWidth, SetLineApprox and SetLineStyle.

Attention

If the output mode is 'margin' and the line width is more than one, objects may not be displayed.
4.6. PARAMETERS

**Parameter**

- **WindowHandle** (input control)  
  Window identifier.
- **Mode** (input control)  
  Fill mode for region output.
  
  *Default Value*: ‘fill’
  *Value List*: \( \text{Mode} \in \{\text{‘fill’}, \text{‘margin’}\} \)

**Result**

SetFix returns TRUE if Mode is correct and the window is valid. Otherwise an exception handling is raised.

**Parallelization Information**

SetFix is *local* and processed *completely exclusively* without parallelization.

**Possible Predecessor Functions**

GetFix

**Possible Successor Functions**

SetPixel, SetRgb

**See Also**

GetFix, SetFix, SetPixel, SetRgb, SetColor, SetHsi, SetGray

---

```c
void HWindowX.SetFix ([in] String Mode )
void HOperatorSetX.SetFix ([in] long WindowHandle, [in] String Mode )
```

Set fixing of “’look-up-table’” (lut)

Behaviour for Mode = 'true': SetFix fixes that pixel lastly ascertained by one of the operators SetGray, SetColor, SetHsi or SetRgb (Remark: Here a pixel is the index within the current look-up-table). To assign a new color to a fixed pixel set a color or gray value by using SetColor, SetRgb, SetHsi or SetGray. This makes it possible to define any color (SetColor), any gray value (SetGray) and any color combination (SetRgb, SetHsi) at any position of the look-up-table.

Mode set to 'false' reset the fixing. To modify or create a look-up-table process SetPixel, SetFix (:::WindowHandle,'true';),SetRgb and SetFix (:::WindowHandle,'false';) one after another.

**Attention**

As a side effect SetFix can change colors of "’non-HALCON windows’".

---

**Parameter**

- **WindowHandle** (input control)  
  Window identifier.
- **Mode** (input control)  
  Mode of fixing.
  
  *Default Value*: ‘true’
  *Value List*: \( \text{Mode} \in \{\text{‘true’}, \text{‘false’}\} \)

**Result**

SetFix returns TRUE if the window is valid, the hardware supports a look-up-table and all parameters are correct. Otherwise an exception handling is raised.

**Parallelization Information**

SetFix is *local* and processed *completely exclusively* without parallelization.

**Possible Predecessor Functions**

GetFix

**Possible Successor Functions**

SetPixel, SetRgb

**See Also**

GetFix, SetFix, SetPixel, SetRgb, SetColor, SetHsi, SetGray
void HWindowX.SetGray ([in] VARIANT GrayValues )

void HOperatorX.SetGray ([in] long WindowHandle, [in] VARIANT GrayValues )

Define grayvalues for region output.

SetGray defines the grayvalues for region output. Grayvalues are defined as the range of the color lookup table that is used for grayvalue output with DispImage in conjunction with SetPaint( WindowHandle,'gray'). These entries can be modified by SetLut. So a 'grayvalue' is the color in which a pixel with the same value is displayed (not necessarily really gray). In general, when changing the color lookup table with SetLut, the colors of the displayed image will change too.

If a grayvalue is needed as a color for image output (i.e. no color changes with SetLut are possible), it can be set with SetColor( WindowHandle,'gray').

If only a single grayvalue is passed, all output will take place in that grayvalue. If a tuple of grayvalues is passed, all output will take place in grayvalues modulo the number of tuple elements. In the example below, the first circle is displayed with grayvalue 100, the second with 200 and the third with 100 again. Every output procedure starts with the first grayvalue. Note, that the number of output grayvalues depends on the number of objects that are displayed in one procedure call. If only single objects are displayed, they always appear in the first grayvalue, even if the consist of more than one connected components.

When the procedures SetGray, SetColor, SetRgb, SetHsi are called, the overwrite the existing values. If not all grayvalues are displayable on the output device, the number range of GrayValues (0..255) is dithered to the range of displayable grayvalues. In any case 0 is displayed as black and 255 as white. The displayable grayvalues can be queried with the procedure QueryGray. Furthermore, the number of actually displayed grayvalues can be changed with SetSystem( numGray*,...:).

This must be done before opening the first window. With SetCheck( ~color:) error messages can be suppressed if a grayvalue can’t be displayed on the screen. In that case, a similar grayvalue is displayed.

Attention

Parameter

WindowHandle (input control) ......................... window ~ HWindowX / long Window_id.

GrayValues (input control) ......................... integer(-array) ~ VARIANT( integer ) Grayvalues for region output.

Default Value : 255

Value Suggestions: GrayValues ∈ {0, 1, 2, 10, 16, 32, 64, 100, 120, 128, 250, 251, 252, 253, 254, 255}

Typical Range of Values : 0 ≤ GrayValues ≤ 0

Example

set_gray(WindowHandle,[100,200])
disp_circle(WindowHandle,[100,200,300],[200,300,100],[100,100,100]).

Parallelization Information

SetGray is local and processed completely exclusively without parallelization.

See Also

GetPixel, SetColor

Module
Define output colors (HSI-coded).

SetHsi sets the region output color(s)/grayvalue(s) for the valid window. Colors are passed as Hue, Saturation, and Intensity. Transformation from HSI to RGB is done with:

\[
\begin{align*}
H & = \left( \frac{2\pi \text{Hue}}{255} \right) \\
I & = \left( \frac{\sqrt{\text{Intensity}}}{255} \right) \\
M1 & = \left( \frac{\sin(H) \text{Saturation}}{(255\sqrt{3})} \right) \\
M2 & = \left( \frac{\cos(H) \text{Saturation}}{(255\sqrt{3})} \right) \\
R & = \frac{(2M1 + I)}{(4\sqrt{3})} \\
G & = \frac{(-M1 + M2 + I)}{(4\sqrt{6})} \\
B & = \frac{(-M1 - M2 + I)}{(4\sqrt{6})}
\end{align*}
\]

If only one combination is passed, all output will take place in that color. If a tuple of colors is passed, the output color of regions and geometric objects is modulo to the number of colors. HALCON always begins output with the first color passed. Note, that the number of output colors depends on the number of objects that are displayed in one procedure call. If only single objects are displayed, they always appear in the first color, even if the consist of more than one connected components.

Selected colors are used until the next call of SetColor, SetPixel, SetRgb or SetGray. Colors are relevant to windows, i.e. only the colors of the valid window can be set. Region output colors are used by operators like DispRegion, DispLine, DispRectangle1, DispRectangle2, DispArrow, etc. It is also used by procedures with grayvalue output in certain output modes (e.g. '3D-plot', 'histogram', 'contourline', etc. See SetPaint).

- **Attention**
  The selected intensities may not be available for the selected hues. In that case, the intensities will be lowered automatically.

---

### Parameter

- **WindowHandle** (input control)  
  window \( \sim \) HWindowX / long Window.id.

- **Hue** (input control)  
  integer \( \sim \) VARIANT( integer )  
  Hue for region output.  
  **Default Value:** 30  
  **Typical Range of Values:** \( 0 \leq \text{Hue} \leq 0 \)  
  **Restriction:** \( (0 \leq \text{Hue}) \land (\text{Hue} \leq 255) \)

- **Saturation** (input control)  
  integer \( \sim \) VARIANT( integer )  
  Saturation for region output.  
  **Default Value:** 255  
  **Typical Range of Values:** \( 0 \leq \text{Saturation} \leq 0 \)  
  **Restriction:** \( (0 \leq \text{Saturation}) \land (\text{Saturation} \leq 255) \)

- **Intensity** (input control)  
  integer \( \sim \) VARIANT( integer )  
  Intensity for region output.  
  **Default Value:** 84  
  **Typical Range of Values:** \( 0 \leq \text{Intensity} \leq 0 \)  
  **Restriction:** \( (0 \leq \text{Intensity}) \land (\text{Intensity} \leq 255) \)

---

### Result

SetHsi returns TRUE if the window is valid and the output colors are displayable. Otherwise an exception handling is raised.
Parallelization Information

SetHsi is *local* and processed *completely exclusively* without parallelization.

Possible Predecessor Functions

GetHsi

Possible Successor Functions

DispRegion

See Also

GetHsi, GetPixel, TransFromRgb, TransToRgb, DispRegion

Module

System

```c
void HRegionX.SetIcon ([in] HWindowX WindowHandle )
void HWindowX.SetIcon ([in] HRegionX Icon )
void HOperatorSetX.SetIcon ([in] IHObjectX Icon, [in] long WindowHandle )
```

.Icon definition for region output.

SetIcon defines an icon for region output (DispRegion). It is displayed in the regions center of gravity. The use of this icon is activated with SetShape.

Parameter

- **Icon** (input iconic) .......................... region ～ HRegionX / IHObjectX Icon for center of gravity.
- **WindowHandle** (input control) ................ window ～ HWindowX / long Window.id.

Parallelization Information

SetIcon is processed *completely exclusively* without parallelization.

Possible Predecessor Functions

GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion

Possible Successor Functions

SetShape, DispRegion

Module

System

```c
void HWindowX.SetInsert ([in] String Mode )
void HOperatorSetX.SetInsert ([in] long WindowHandle, [in] String Mode )
```

Define the pixel output function.

SetInsert defines the function, with which pixels are displayed in the output window. It is e.g. possible for a pixel to overwrite the old value. In most of the cases there is a functional relationship between old and new values.

The definition value is only valid for the valid window. Output procedures that honor Mode are e.g. DispRegion, DispPolygon, DispCircle.

Possible display functions are:

- **'copy'**: overwrite displayed pixels
- **'xor'**: display old "xor" new pixels
- **'not'**: complement displayed pixels
There may not be all functions available, depending on the physical display. However, "'copy'" is always available.

--- Attention Parameter ---

- **WindowHandle** (input control)  
  window ~ HWindowX / long WindowId.

- **Mode** (input control)  
  Name of the display function.  
  Default Value: 'copy'
  Value List: Mode ∈ {'copy', 'xor', 'not'}

--- Result ---

SetInsert returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

--- Parallelization Information ---

SetInsert is local and processed completely exclusively without parallelization.

--- Possible Predecessor Functions ---

QueryInsert, GetInsert

--- Possible Successor Functions ---

DispRegion

--- See Also ---

GetInsert, QueryInsert

--- Module ---

System

```c
void HWindowX.SetLineApprox ([in] long Approximation )

void HOperatorSetX.SetLineApprox ([in] long WindowHandle, [in] long Approximation )
```

**Define the approximation error for contour display.**

SetLineApprox defines the approximation error for region contour display in the window. Approximation values greater than zero cause a polygon approximation smoothing (with a maximum polygon/contour deviation of Approximation pixel). The approximation algorithm is the same as in GetRegionPolygon. SetLineApprox is important for contour output via SetLineStyle.

--- Attention ---

--- Parameter ---

- **WindowHandle** (input control)  
  window ~ HWindowX / long WindowId.

- **Approximation** (input control)  
  Maximum deviation from the original contour.  
  Default Value: 0
  Restriction: (Approximation ≥ 0)

--- Example ---

```c
/* Calling */
set_line_approx(WindowHandle,Approximation)
set_draw(WindowHandle,'margin')
disp_region(Obj,WindowHandle).

/* correspond with */
get_region_polygon(Obj,Approximation,Row,Col)
disp_polygon(WindowHandle,Row,Col).
```
SetLineApprox returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

SetLineApprox is processed completely exclusively without parallelization.

GetLineApprox

DispRegion

Possible Predecessor Functions

GetLineApprox, SetLineStyle, SetDraw, DispRegion

Possible Successor Functions

DispRegion, DispPolygon

Alternatives

GetRegionPolygon, DispPolygon

Module

System

```c
void HWindowX.SetLineStyle ([in] VARIANT Style )
void HOperatorSetX.SetLineStyle ([in] long WindowHandle, [in] VARIANT Style )
```

Define a contour output pattern.

SetLineStyle defines the output pattern of region contours. The information is used by procedures like DispRegion, DispLine, DispPolygon etc. The current value can be queried with GetLineStyle. Style contains up to five pairs of values. The first value is the length of the visible contour part, the second is the length of the invisible part. The value pairs are used cyclical for contour output.

Attention

SetLineStyle does an implicit polygon approximation (see SetLineApprox(::WindowHandle,3:)). It is only possible to enlarge it with SetLineApprox.

Parameter

- **WindowHandle** (input control) ................................. window  ~ HWindowX / long Window_id.
- **Style** (input control) ................................. integer  ~ VARIANT( integer ) Countour pattern.
  Default Value : []

Example

* stroke line: X-Windows
  set_line_style(WindowHandle,[20,7])
* point-stroke line: X-Windows
  set_line_style(WindowHandle,[20,7,3,7])
* passing line (standard)
  set_line_style(WindowHandle,[])
4.6. PARAMETERS

Possible Successor Functions

DispRegion

See Also

GetLineStyle, SetLineApprox, DispRegion

Module

System

void HWindowX.SetLineWidth ([in] long Width )
void HOperatorSetX.SetLineWidth ([in] long WindowHandle, [in] long Width )

Define the line width for region contour output.

SetLineWidth defines the line width (in pixel) in which a region contour is displayed (e.g. with DispRegion, DispLine, DispPolygon, etc.) The procedure GetLineWidth returns the current value for the window. Some output devices may not allow to change the contour width. If it is possible for the current device, it can be queried with QueryLineWidth.

Attention

The line width is important if the output mode was set to 'margin' (see SetDraw). If the line width is greater than one, regions may not always be displayed correctly.

Parameter

- WindowHandle (input control) 
  window \sim HWindow / long WindowHandle.
- Width (input control) 
  Line width for region output in contour mode.

Default Value : 1

Restriction : (Width \geq 1)

Result

SetLineWidth returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information

SetLineWidth is processed completely exclusively without parallelization.

Possible Predecessor Functions

QueryLineWidth, GetLineWidth

Possible Successor Functions

DispRegion

See Also

GetLineWidth, QueryLineWidth, SetDraw, DispRegion

Module

System

void HWindowX.SetPaint ([in] VARIANT Mode )
void HOperatorSetX.SetPaint ([in] long WindowHandle, [in] VARIANT Mode )

Define the grayvalue output mode.

SetPaint defines the output mode for grayvalue display (single- or multichannel) in the window. The mode is used by DispImage.

This page describes the different modes, that can be used for grayvalue output. It should be noted, that the mode 'default' is the most suitable.

The hardware characteristics determine how grayvalues can be displayed. On a screen with one to seven bit planes, only binary data can be displayed. On screens with at least eight bit planes, it is possible to display multiple...
grayvalues. For binary displays HALCON includes algorithms with dithering matrix (fast, but low resolution), minimal error (good, but slow) and thresholding. Using the thresholding algorithm, the threshold can be passed as a second parameter (a tuple with the string 'threshold' and the actual threshold, e.g.: ['threshold', 100]). Displays with eight bit planes use approximately 200 grayvalues for output. Of course it is still possible to use a binary display on those displays.

A different way to display grayvalues is the histogram (mode: 'histogram'). This mode has two additional parameter values: Row (second value) and column (third value). They denote row and column of the histogram center for positioning on the screen. The scale factor (fourth value) determines the histogram size: a scale factor of 1 distinguishes 256 grayvalues, 2 distinguishes 128 grevalues, and so on. The four values are passed as a tuple, e.g. ['histogram', 256,256,1]. If only the first value is passed ('histogram'), the other values are set to defaults or the last values, respectively. For histogram computation see GrayHisto. Histogram output honors the same parameters as procedures like DispRegion etc. (e.g. SetColor, SetDraw, etc.)

Yet another mode is the display of relative frequencies of the number of connection components ('component_histogram'). For informations on computing the component histogram see ShapeHistoAll. Positioning and resolution are exactly as in the mode 'histogram'.

In mode 'mean', all object regions are displayed in their mean grayvalue.

The modes 'line' and 'column' allow the display of lines or columns, respectively. The position (line- and columnindex) is passed with the second parameter value. The third parameter value is the scale factor in percent (100 means 1 pixel per grayvalue, 50 means one pixel per two grayvalues).

Gray images can also be interpreted as 3d data, depending on the grayvalue. To view these 3d plots, select the modes 'contourline', '3D-plot' or '3D-plot_hidden'.

Two-channel images can be viewed as 'vectorfield'.

Three-channel images are interpreted as RGB images. They can be displayed in three different modes. Two of them can be optimized by Floyd-Steinberg dithering.

Parameters for modes that need more than one parameter can be passed the following ways:

- Only the name of the mode is passed: the defaults or the last values are used, respectively. Example: SetPaint(::WindowHandle, 'contourline':)
- All values are passed: all output characteristics can be set. Example: SetPaint(::WindowHandle, ['contourline',10,1]:)
- Only the first n values are passed: only the passed values are changed. Example: SetPaint(::WindowHandle, ['contourline',10]:)
- Some of the values are replaced by an asterisk ('*'): The value of the replaced parameters is not changed. Example: SetPaint(::WindowHandle, ['contourline','*',1]:)

If the current mode is 'default', HALCON chooses a suitable algorithm for the output of 2- and 3-channel images. No SetPaint call is necessary in this case.

Apart from SetPaint there are other procedures that affect the output of grayvalues. The most important of them are SetPart, SetPartStyle, SetLut and SetLutStyle. Some output modes display grayvalues using region output (e.g. 'histogram', 'contourline', '3D-plot', etc.). In these modes, parameters set with SetColor, SetRgb, SetHsi, SetPixel, SetShape, SetLineWidth and SetInsert influence grayvalue output. This can lead to unexpected results when using SetShape(::'convex':) and SetPaint(::WindowHandle, 'histogram':). Here the convex hull of the histogram is displayed.

Modes:

- 'default' optimal display on given hardware
- 'gray' grayvalue output
- 'mean' mean grayvalue
- 'dither4_1' binary image, dithering matrix 4x4
- 'dither4_2' binary image, dithering matrix 4x4
- 'dither4_3' binary image, dithering matrix 4x4
- 'dither8_1' binary image, dithering matrix 8x8
- 'floyd_steinberg' binary image, optimal grayvalue simulation
4.6. PARAMETERS

[‘threshold’, Threshold ]
  ‘threshold’ binary image, threshold: 128 (default)
  [‘threshold’, 200 ] binary image, any threshold: (here: 200)

[‘histogram’, Line, Column, Scale ]
  ‘histogram’ grayvalue output as histogram.
  position default: max. size, in the window center
  [‘histogram’, 256, 256, 2 ] grayvalue output as histogram, any parameter values.
  position: window center (here (256, 256))
  size: (here 2, half the max. size)

[‘component_histogram’, Line, Column, Scale ]
  ‘component_histogram’ output as histogram of the connection components.
  Positioning: default
  [‘component_histogram’, 256, 256, 1 ] output as histogram of the connection components.
  Positioning: (here (256, 256))
  Scaling: (here 1, max. size)

[‘line’, Line, Scale ]
  ‘line’ output of the grayvalue profile along the given line.
  line: image center (default)
  Scaling: 50
  [‘line’, 100, 20 ] output of the grayvalue profile of line 100 with a scaling of 0.2 (20)

[‘column’, Column, Scale ]
  ‘column’ output of the grayvalue profile along the given column.
  column: image center (default)
  Scaling: 50
  [‘column’, 100, 20 ] output of the grayvalue profile of column 100 with a scaling of 0.2 (20)

[‘contourline’, Step, Colored ]
  ‘contourline’ grayvalue output as contour lines: the grayvalue difference per line is defined with the
  parameter ‘Step’ (default: 30, i.e. max. 8 lines for 256 grayvalues). The line can be displayed in
  a given color (see set_color) or in the grayvalue they represent. This behaviour is defined with the
  parameter ‘Colored’ (0 = color, 1 = grayvalues). Default is color.
  [‘contourline’, 15, 1 ] grayvalue output as contour lines with a step of 15 and gray output.

[‘3D-plot’, Step, Colored, EyeHeight, EyeDistance, ScaleGray, LinePos, ColumnPos ]
  ‘3D-plot’ grayvalues are interpreted as 3d data: the greater the value, the ‘higher’ the assumed moun-
  tain. Lines with step 2 (second parameter value) are drawn along the x- and y-axes. The third pa-
  rameter (Colored) determines, if the output should be in color (default) or grayvalues. To define the
  projection of the 3d data, use the parameters EyeHeight and EyeDistance. The projection parameters
  take values from 0 to 255. ScaleGray defines a factor, by which the grayvalues are multiplied for
  ‘height’ interpretation (given in percent. 100EyeHeight and EyeDistance the image can be shifted
  out of place. Use RowPos and ColumnPos to move the whole output. Values from -127 to 127 are
  possible.
  [‘3D-plot’, 5, 1, 110, 160, 150, 70, -10 ] line step: 5 pixel
    Colored: yes (1)
    EyeHeight: 110
    EyeDistance: 160
    ScaleGray: 1.5 (150)
    RowPos: 70 pixel down
    ColumnPos: 10 pixel right

[‘3D-plot_hidden’, Step, Colored, EyeHeight, EyeDistance, ScaleGray, LinePos, ColumnPos ]
  ‘3D-plot_hidden’ like ‘3D-plot’, but computes hidden lines.

Two-channel images:
  ‘default’ output as vector field.

[‘vectorfield’, Step, MinLength, ScaleLength ]
  ‘vectorfield’ output of the two channels as displacement vector field. Procedures like optical_flow*
  produce images of pixel displacements (first channel: x-displacement, second channel: y-
  displacement).
  In this mode, an arrow is drawn for each vector. The parameter Step has to be set in correspondence

HALCON 6.0
to optical flow*. Short vectors can be suppressed by the third parameter value (MinLength). The fourth parameter value scales the vector length. Vectors, that exceed the window boundaries are not drawn.

`[vectorfield',16,2,3 ]` Output of every 16. vector, that is longer than 2 pixel. Each vector is multiplied by 3 for output.

- Three-channel images:
  - `default` output as RGB image with `median_cut`.
  - `television` color addition algorithm for RGB images: (three components necessary for DispImage). Images are displayed via a fixed color lookup table. Fast, but non-optimal color resolution. Only recommended on bright screens.
  - `grid_scan_floyd_steinberg` grid-scan with Floyd-Steinberg dithering for smooth color boundaries.
  - `median_cut_floyd_steinberg` median-cut algorithm with Floyd-Steinberg dithering for smooth color boundaries.

---

**Attention**

- Display of color images (`television`, `grid_scan`, etc.) changes the color lookup tables.
- If a wrong color mode is set, the error message may appear not until the `DispImage` call.
- Grayvalue output may be influenced by region output parameters. This can yield unexpected results.

---

**Parameter**

- **WindowHandle** (input control) ......................................... window \( \sim HWindowX / long \) Window id.
- **Mode** (input control) .................................................. string \( \sim \) VARIANT ( integer, string )
  - Grayvalue output name. Additional parameters possible.
  - **Default Value** : `default`

---

**Example**

```plaintext
read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible',',WindowHandle)
query_paint(WindowHandleModi)
fwrite_string([['available gray value modes: ',Modi]])
newline()
disp_image(Image,WindowHandle)
get_mbutton(WindowHandle,_,_,_)
set_color(WindowHandle,'red')
set_draw(WindowHandle,'margin')
set_paint(WindowHandle,'histogram')
disp_image(Image,WindowHandle)
set_color(WindowHandle,'blue')
set_paint(WindowHandle,['histogram',100,100,3])
disp_image(Image,WindowHandle)
set_color(WindowHandle,'yellow')
set_paint(WindowHandle,['line',100])
disp_image(Image,WindowHandle)
get_mbutton(WindowHandle,_,_,_)
clear_window(WindowHandle)
set_paint(WindowHandle,['contourline',10,1])
disp_image(Image,WindowHandle)
```
set_lut(WindowHandle,'color')
get_mbutton(WindowHandle,_,_,_)
clear_window(WindowHandle)
set_part(WindowHandle,100,100,300,300)
set_paint(WindowHandle,'3D-plot')

disp_image(Image,WindowHandle).

Result
SetPaint returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information
SetPaint is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
QueryPaint,GetPaint

Possible Successor Functions
DispImage

See Also
GetPaint,QueryPaint,DispImage,SetShape,SetRgb,SetColor,SetGray

Module
System

```
```

```
```

Modify the displayed image part.
SetPart modifies the image part that is displayed in the window. (Row1,Column1) denotes the upper left corner and (Row2,Column2) the lower right corner of the image part to display. The changed values are used by grayvalue output operators (DispImage,DispColor) as well as region output operators (DispRegion)).

If only part of an image is displayed, it will be zoomed to full window size. The zooming interpolation method can be set with SetPartStyle. GetPart returns the values of the image part to display.
Beside setting the image part directly, the following special modes are supported:

Row1 = Column1 = Row2 = Column2 = -1: The window size is choosen as the image part, i.e. no zooming of the image will be performed.
Row1,Column1 > -1 and Row2 = Column2 = -1: The size of the last displayed image (in this window) is choosen as the image part, i.e. the image can completely be displayed in the image. For this the image will be zoomed if necessary.

Attention
After a call of SetPart, some operators like DrawRegion, DrawEllipse, etc. can no longer be used.

```
WindowHandle (input control) ................................. window ~/HWindowX/long Window_id.
Row1 (input control) ................................. rectangle.origin.y ~ long Row of the upper left corner of the chosen image part.
Default Value : 0
Column1 (input control) ................................. rectangle.origin.x ~ long Column of the upper left corner of the chosen image part.
Default Value : 0
```
**Row2** (input control) ..........................................................rectangle.corner.y \sim long  
Row of the lower right corner of the chosen image part.  
**Default Value** : -1  
**Restriction** : (Row2 \geq Row1) \lor (Row2 = -1)

**Column2** (input control) .....................................................rectangle.corner.x \sim long  
Column of the lower right corner of the chosen image part.  
**Default Value** : -1  
**Restriction** : (Column2 \geq Column1) \lor (Column2 = -1)

---

**Example**

```plaintext
get_system('width', Width)
get_system('height', Height)
set_part(WindowHandle, 0, 0, Height-1, Width-1)
disp_image(Image, WindowHandle)
draw_rectangle1(WindowHandle:Row1,Column1,Row2,Column2)
set_part(WindowHandle,Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle).
```

**Result**

SetPart returns TRUE if the window is valid. Otherwise an exception handling is raised.

---

**Parallelization Information**

SetPart is processed *completely exclusively* without parallelization.

---

**Possible Predecessor Functions**

GetPart

---

**Possible Successor Functions**

SetPartStyle, DispImage, DispRegion

---

**See Also**

GetPart, SetPartStyle, DispRegion, DispImage, DispColor

---

**Alternatives**

AffineTransImage

---

**Module**

System

```plaintext
void HWindowX.SetPartStyle ([in] long Style )

void HOperatorSetX.SetPartStyle ([in] long WindowHandle, [in] long Style )
```

*Define an interpolation method for grayvalue output.*

SetPartStyle defines the interpolation method to zoom an image part which is displayed in the window. Interpolation takes place, if the output window has different size than the image to display (e.g. after a call to SetPart or a window resize). Three modes are supported:

0 no interpolation (low quality, very fast).
1 unweighted interpolation (medium quality and run time)
2 weighted interpolation (high quality, slow)

The current value can be queried with GetPartStyle.

---

**Attention**
4.6. PARAMETERS

Parameter

- **WindowHandle** (input control)  
  window \( \sim \) HWindowX / long Window
- **Style** (input control)  
  Interpolation method for image output: 0 (fast, low quality) to 2 (slow, high quality).

  - Default Value: 0
  - Value List: 0, 1, 2

Result

SetPartStyle returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information

SetPartStyle is processed completely exclusively without parallelization.

Possible Predecessor Functions

- GetPartStyle

Possible Successor Functions

- SetPart, DispImage, DispRegion

See Also

- GetPartStyle, SetPart, DispImage, DispColor

Alternatives

- AffineTransImage

Module

System

---

```
void HWindowX.SetPixel ([in] VARIANT Pixel )
void HOperatorSetX.SetPixel ([in] long WindowHandle, [in] VARIANT Pixel )
```

**Define a color lookup table index.**

SetPixel sets pixel values: colors (SetColor, SetRGB, etc.) and grayvalues (SetGray) are coded together into a number, called pixel. This 'pixel' is an index in the color lookup table. It ranges from 0 to 1 in b/w images and 0 to 255 color images with 8 bit planes. It is different from the 'pixel' ("picture element") in image processing. Therefore HALCON distinguishes between pixel and image element (or grayvalue).

The current value can be queried with GetPixel.

---

Attention

---

Parameter

- **WindowHandle** (input control)  
  window \( \sim \) HWindowX / long Window
- **Pixel** (input control)  
  Color lookup table index.

  - Default Value: 128
  - Typical Range of Values: 0 \( \leq \) Pixel \( \leq \) 0

Result

SetPixel returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information

SetPixel is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

GetPixel

Possible Successor Functions

DispImage, DispRegion

HALCON 6.0
Set the color definition via RGB values.

SetRgb sets the output color(s) or the grayvalues, respectively, for region output for the window. The colors are defined with the red, green and blue components. If only one combination is passed, all output takes place in that color. If a tuple is passed, region output and output of geometric objects takes place modulo the passed colors.

For every call of an output procedure, output is started with the first color. If only one object is displayed per call, it will always be displayed in the first color. This is even true for objects with multiple connection components. If multiple objects are displayed per procedure call, multiple colors are used. The defined colors are used until SetColor, SetPixel, SetRgb or SetGray is called again. The values are used by procedures like DispRegion, DispLine, DispRectangle1, DispRectangle2, DispArrow, etc.

Attention

If a passed is not available, an exception handling is raised. If SetCheck(ː·‘\'\color\'\ː) was called before, HALCON uses a similar color and suppresses the error.

SetRgb sets the output color(s) or the grayvalues, respectively, for region output for the window. The colors are defined with the red, green and blue components. If only one combination is passed, all output takes place in that color. If a tuple is passed, region output and output of geometric objects takes place modulo the passed colors.

For every call of an output procedure, output is started with the first color. If only one object is displayed per call, it will always be displayed in the first color. This is even true for objects with multiple connection components. If multiple objects are displayed per procedure call, multiple colors are used. The defined colors are used until SetColor, SetPixel, SetRgb or SetGray is called again. The values are used by procedures like DispRegion, DispLine, DispRectangle1, DispRectangle2, DispArrow, etc.

Attention

If a passed is not available, an exception handling is raised. If SetCheck(ː·‘\'\color\'\ː) was called before, HALCON uses a similar color and suppresses the error.

SetRgb returns TRUE if the window is valid and all passed colors are available and displayable. Otherwise an exception handling is raised.

Parallelization Information

SetRgb is local and processed completely exclusively without parallelization.

Possible Successor Functions

DispImage, DispRegion

See Also

SetFix, DispRegion

Alternatives

SetHsi, SetColor, SetGray
Define the region output shape.

SetShape defines the shape for region output. It is only valid for the window with the logical window number WindowHandle. The output shape is used by DispRegion. The available shapes can be queried with QueryShape.

Available modes:

'original': The shape is displayed unchanged. Nevertheless modifications via parameters like set_line_width or set_line_approx can take place. This is also true for all other modes.

'outer_circle': Each region is displayed by the smallest surrounding circle. (See SmallestCircle.)

'inner_circle': Each region is displayed by the largest included circle. (See InnerCircle.)

'ellipse': Each region is displayed by an ellipse with the same moments and orientation (See EllipticAxis.)

'rectangle1': Each region is displayed by the smallest surrounding rectangle parallel to the coordinate axes. (See SmallestRectangle1.)

'rectangle2': Each region is displayed by the smallest surrounding rectangle. (See SmallestRectangle2.)

'convex': Each region is displayed by its convex hull (See Convexity.)

'icon' Each region is displayed by the icon set with SetIcon in the center of gravity.

Attention

Caution is advised for grayvalue output procedures with output parameter settings that use region output, e.g. DispImage with SetPaint(::WindowHandle,'histogram') and SetShape(::WindowHandle,'convex'). In that case the convex hull of the grayvalue histogram is displayed.

Parameter

- **WindowHandle** (input control) window ~ HWindowX / long Window._
- **Shape** (input control) string ~ String Region output mode.

  **Default Value**: 'original'

  **Value List**: Shape ∈ {'original', 'convex', 'outer_circle', 'inner_circle', 'rectangle1', 'rectangle2', 'ellipse', 'icon'}

Example

```
read_image(Image,'fabrik')
regiongrowing(Image,Seg,5,5,6,100)
set_colored(WindowHandle,12)
set_shape(WindowHandle,'rectangle2')
disp_region(Seg,WindowHandle).
```

Result

SetShape returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information

SetShape is processed completely exclusively without parallelization.

Possible Predecessor Functions

SetIcon, QueryShape, GetShape

Possible Successor Functions

DispRegion
4.7 Text

Get the current font.

GetFont queries the name of the font used in the output window. The font is used by the operators WriteString, ReadString etc. The font is set by the operator SetFont. Text windows as well as windows for image display use fonts. Both types of windows have a default font that can be modified with SetSystem (‘defaultFont’, Fontname) prior to opening the window. A list of all available fonts can be obtained using QueryFont.

GetFont returns TRUE.

Get the spatial size of a string.

See Also
GetShape, QueryShape, DispRegion

Module System

Get the current font.

GetFont queries the name of the font used in the output window. The font is used by the operators WriteString, ReadString etc. The font is set by the operator SetFont. Text windows as well as windows for image display use fonts. Both types of windows have a default font that can be modified with SetSystem (‘defaultFont’, Fontname) prior to opening the window. A list of all available fonts can be obtained using QueryFont.

GetFont returns TRUE.

Get the spatial size of a string.

See Also
SetFont, QueryFont, OpenWindow, OpenTextwindow, SetSystem

Module System
GetStringExtents queries width and height of the output size of a string using the font of the window. In addition the extension above and below the current baseline for writing is returned (Ascent bzw. Descent).

The sizes are measured in the coordinate system of the window (for text windows in pixels). Using GetStringExtents it is possible to determine text output and input independently from the used font. The conversion from integer numbers and floating point numbers to text strings is the same as in WriteString.

**Attention**

**Parameter**

- WindowHandle (input control) \( \Rightarrow \) window \( \sim \) HWindowX / long
  Window identifier.

- Values (input control) \( \Rightarrow \) string(-array) \( \sim \) VARIANT( string, real, integer )
  Values to consider.
  **Default Value:** “test_string”

- Ascent (output control) \( \Rightarrow \) integer \( \sim \) long
  Maximum height above baseline for writing.

- Descent (output control) \( \Rightarrow \) integer \( \sim \) long
  Maximum extension below baseline for writing.

- Width (output control) \( \Rightarrow \) integer \( \sim \) long
  Text width.

- Height (output control) \( \Rightarrow \) integer \( \sim \) long
  Text height.

**Result**

GetStringExtents returns TRUE if the window is valid. Otherwise an exception handling is raised.

**Parallelization Information**

GetStringExtents is local and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

OpenWindow, OpenTextwindow, SetFont

**Possible Successor Functions**

SetTposition, WriteString, ReadString, ReadChar

**See Also**

SetTposition, SetFont

**Module**

System

```c
[out] long Row HWindowX.GetTposition ([out] long Column )
```

Get cursor position.

GetTposition queries the current position of the text cursor in the output window. The position is measured in the coordinate system of the window (in pixels for text windows). The next output of text in this window starts at the cursor position. The left end of the baseline for writing the next string (not considering descenders) is placed on this position. The position is changed by the output or input of text (WriteString, ReadString) or by an explicit change of position by (SetTposition, NewLine).

**Attention**

If the output text does not fit completely into the window, an exception handling is raised. This can be avoided by SetCheck(‘~text’).
**Parameter**

- **WindowHandle** (input control) \( \text{window} \sim HWindowX / \text{long} \)
  Window identifier.

- **Row** (output control) \( \text{point.y} \sim \text{long} \)
  Row index of text cursor position.

- **Column** (output control) \( \text{point.x} \sim \text{long} \)
  Column index of text cursor position.

**Result**

GetTposition returns TRUE if the window is valid. Otherwise an exception handling is raised.

**Parallelization Information**

GetTposition is local and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

OpenWindow, OpenTextwindow, SetFont

**Possible Successor Functions**

SetTposition, WriteString, ReadString, ReadChar

**See Also**

NewLine, ReadString, SetTposition, WriteString, SetCheck

**Module**

System

```c
[out] String TextCursor HWindowX.GetTshape ( )

void HOperatorSetX.GetTshape ( [in] long WindowHandle, [out] String TextCursor )
```

Get the shape of the text cursor.

GetTshape queries the shape of the text cursor for the output window. A new cursor shape is set by the operator SetTshape.

A text cursor marks the current position for text output (which can also be invisible). It is different from the mouse cursor (although both will be called "cursor" if the context makes misconceptions impossible). The available shapes for the text cursor can be queried with QueryTshape.
Set the position of the text cursor to the beginning of the next line.

NewLine sets the position of the text cursor to the beginning of the next line. The new position depends on the current font. The left end of the baseline for writing the following text string (not considering descenders) is placed on this position.

If the next line does not fit into the window the content of the window is scrolled by the height of one line in the upper direction. In order to reach the correct new cursor position the font used in the next line must be set before NewLine is called. The position is changed by the output or input of text (WriteString, ReadString) or by an explicit change of position by (SetTposition).

Parameter

- **WindowHandle** (input control) .......................... window ~ HWindowX / long  
  Window identifier.

Result

NewLine returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

NewLine is processed **completely exclusively** without parallelization.

Possible Predecessor Functions

OpenWindow, OpenTextWindow, SetFont, WriteString

See Also

writestring, setfont

Alternatives

GetTposition, GetStringExtents, SetTposition, MoveRectangle

Module

System

Query the available fonts.

QueryFont queries the fonts available for text output in the output window. They can be set with the operator SetFont. Fonts are used by the operators WriteString, ReadChar, ReadString and NewLine.

Attention

For different machines the available fonts may differ a lot. Therefore QueryFont will return different fonts on different machines.

Parameter

- **WindowHandle** (input control) .......................... window ~ HWindowX / long  
  Window identifier.
- **Font** (output control) ................................. string ~ VARIANT ( string )  
  Tupel with available font names.

Example

```halcon
open_window(0,0,-1,-1,'root','visible',WindowHandle)
set_check('˜text')
query_font(WindowHandle,Fontlist)
set_color(WindowHandle,'white')
for i=0 to |Fontlist|-1 by 1
  set_font(WindowHandle,Fontlist[i])
```
write_string(WindowHandle, Fontlist[i])
new_line(WindowHandle)
endfor

---------- Result
QueryFont returns TRUE.

---------- Parallelization Information
QueryFont is local and processed completely exclusively without parallelization.

---------- Possible Predecessor Functions
OpenWindow, OpenTextWindow

---------- Possible Successor Functions
SetFont, WriteString, ReadString, ReadChar

---------- See Also
SetFont, WriteString, ReadString, ReadChar, NewLine

---------- Module
System

[out] VARIANT TextCursor HWindowX.QueryTshape ( )

void HOperatorSetX.QueryTshape ([in] long WindowHandle,
[out] VARIANT TextCursor )

Query all shapes available for text cursors.
QueryTshape queries the available shapes of text cursors for the output window. The retrieved shapes can be used by the operator SetTshape.

---------- Parameter

▷ WindowHandle (input control) .............................. window ⇆ HWindowX / long
Window identifier.
▷ TextCursor (output control) .............................. string ⇆ VARIANT ( string )
Names of the available text cursors.

---------- Result
QueryTshape returns TRUE.

---------- Parallelization Information
QueryTshape is local and processed completely exclusively without parallelization.

---------- Possible Predecessor Functions
OpenWindow, OpenTextWindow

---------- Possible Successor Functions
SetTshape, WriteString, ReadString

---------- See Also
SetTshape, GetShape, SetTposition, WriteString, ReadString

---------- Module
System

[out] String Char HWindowX.ReadChar ([out] String Code )

void HOperatorSetX.ReadChar ([in] long WindowHandle, [out] String Char,
[out] String Code )

Read a character from a text window.
ReadChar reads a character from the keyboard in the input window (= output window). If the character is printable it is returned in Char. If a control key has been pressed, this will be indicated by the value of Code. Some important keys are recognizable by this value. Possible values are:
4.7. TEXT

'character': printable character
'left': cursor left
'right': cursor right
'up': cursor up
'down': cursor down
'insert': insert
'none': none of these keys

Attention
The window has to be a text window.

Parameter
/BA WindowHandle (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . window
/BN HWindowX / long Window identifier.
/BA Char (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . string
/BN String Input character (if it is not a control character).
/BA Code (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . string
/BN String Code for input character.

Result
ReadChar returns TRUE if the text window is valid. Otherwise an exception handling is raised.

Parallelization Information
ReadChar is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
OpenTextWindow, SetFont

See Also
WriteString, SetFont

Alternatives
ReadString, ReadChar, ReadString

Module
System

[out] String OutString HWindowX.ReadString ([in] String InString,
[in] long Length )

void HOperatorSetX.ReadString ([in] long WindowHandle,

Read a string in a text window.

ReadString reads a string with a predeterminded maximum size (Length) from the keyboard in the input window (= output window). The string is read from the current position of the text cursor using the current font. The maximum size has to be small enough to keep the string within the right window boundary. A default string which can be edited or simply accepted by the user may be provided. After text input the text cursor is positioned at the end of the edited string. Commands for editing:

RETURN finish input
BACKSPACE delete the character on the left side of the cursor and move the cursor to this position.

Parameter

/BA WindowHandle (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . window
/BN HWindowX / long Window identifier.
/BA InString (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . string
/BN String Default string (visible before input).

Default Value: ”
Length (input control) ................................................................. integer $\rightarrow$ long
  Maximum number of characters.
  Default Value : 32
  Restriction : ($Length > 0$)

OutString (output control) ......................................................... string $\rightarrow$ String
  Read string.

Result

ReadStream returns TRUE if the text window is valid and a string of maximal length fits within the right window boundary. Otherwise an exception handling is raised.

Parallelization Information

ReadStream is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenTextwindow, SetFont

See Also

SetTposition, NewLine, OpenTextwindow, SetFont, SetColor

Alternatives

ReadChar, FreadString, FreadChar

Module

System

void HWindowX.SetFont ([in] String Font )
void HOperatorSetX.SetFont ([in] long WindowHandle, [in] String Font )

Set the font used for text output.

SetFont sets the font for the output window. The font is used by the operators WriteString, ReadString etc. Text windows as well as windows for image display use fonts. A default font is assigned when a window is opened. This font can be changed with SetFont. On UNIX systems all available fonts can be queried with QueryFont. The default font can be modified with SetSystem(‘defaultFont’, Fontname). Fonts are not used for file operations.

The syntax for the specification of a font (in Font) differs for UNIX and Windows NT environments: In Windows NT a string with the following components is used:

-FontName-Height-Width-Italic-Underlined-Strikeout-

where “Italic”, “Underlined”, and “Strikeout” can take the values 1 and 0 to activate or deactivate the corresponding feature. Please refer to the Windows NT documentation for a detailed discussion.

Attention

For different machines the available fonts may differ a lot. Therefore it is suggested to use wildcards, tables of fonts and/or the operator QueryFont.

Parameter

- WindowHandle (input control) ................................................. window $\rightarrow$ HWindowX / long
  Window identifier.

- Font (input control) ......................................................... string $\rightarrow$ String
  Name of new font.

Example

// UNIX
set_font(WindowHandle,’-*-courier-**-18-*-*-*-*-*’)
// Windows NT
set_font(WindowHandle,’-Arial-18-*-*-*-*’)
write_string(WindowHandle,’Text with Font size of 18 pixels’)
new_line(WindowHandle)
SetFont returns TRUE if the font name is correct. Otherwise an exception handling is raised.

SetFont is **local** and processed **completely exclusively** without parallelization.

---

### Possible Predecessor Functions

OpenWindow, OpenTextwindow

---

### Possible Successor Functions

QueryFont

---

### See Also

GetFont, QueryFont, OpenTextwindow, OpenWindow

---

## Module

**System**

---

### Example Code

```c
void HWindowX.SetTposition ([in] long Row, [in] long Column )

```

---

**Set the position of the text cursor.**

SetTposition sets the position of the text cursor in the output window. The reference position is the upper left corner of an upper case character.

The position is measured in the coordinate system of the window and has to be valid for the window size and font. The position of the text cursor can be marked e.g. by an underscore. The next text output in this window starts at the cursor position. The left end of the baseline for writing the following text string (not considering descenders) is placed on this position.

The position is changed by the output or input of text (WriteString, ReadString) or by an explicit change of position by (SetTposition, NewLine). In order to stop the display of the cursor, the operator SetTshape with the parameter ”invisible” can be used.

---

### Attention

If a string starting at the given position does not fit into the window an exception handling will be raised. This exception handling can be avoided by SetCheck(‘˜text’).

---

### Parameter

**WindowHandle** (input control) ...............................window ˜ HWindowX/long Window identifier.

**Row** (input control) ..................................................point.y ˜ long Row index of text cursor position.

**Column** (input control) .................................................point.x ˜ long Column index of text cursor position.

---

### Result

SetTposition returns TRUE if the window is valid and the values of the parameters are valid. Otherwise an exception handling is raised.

---

### Parallelization Information

SetTposition is **local** and processed **completely exclusively** without parallelization.

---

### Possible Predecessor Functions

OpenWindow, OpenTextwindow

---

### Possible Successor Functions

SetTshape, WriteString, ReadString

---

### See Also

ReadString, SetTshape, WriteString

HALCON 6.0
NewLine

Module

System

void HWindowX.SetTshape ([in] String TextCursor )
void HOperatorSetX.SetTshape ([in] long WindowHandle, [in] String TextCursor )

Set the shape of the text cursor.

SetTshape sets the shape and the display mode of the text cursor of the output window.

A text cursor marks the current position for text output. It is different from the mouse cursor (although both will be called 'cursor', if the context makes misconceptions impossible). The available shapes for the text cursor can be queried with QueryTshape.

Parameter

- WindowHandle (input control) .............................. window ~ HWindowX / long Window identifier.
- TextCursor (input control) ................................. string ~ String Name of cursor shape.
  Default Value : 'invisible'

Result

GetTshape returns TRUE if the window is valid and the given cursor shape is defined for this window. Otherwise an exception handling is raised.

Parallelization Information

SetTshape is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow, OpenTextwindow, QueryTshape, GetTshape

Possible Successor Functions

WriteString, ReadString

See Also

GetTshape, QueryTshape, WriteString, ReadString

System

void HWindowX.WriteString ([in] VARIANT String )
void HOperatorSetX.WriteString ([in] long WindowHandle, [in] VARIANT String )

Print text in a window.

WriteString prints String in the output window starting at the current cursor position. The output text has to fit within the right window boundary (the width of the string can be queried by GetStringExtents).

The font currently assigned to the window will used. The text cursor is positioned at the end of the text.

WriteString can output all three types of data used in HALCON . The conversion to a string is guided by the following rules:

- strings are not converted.
- integer numbers are converted without any spaces before or after the number.
- floating numbers are printed (if possible) with a floating point and without an exponent.
- the resulting strings are concatenated without spaces.
For buffering of texts see SetSystem with the flag 'flush_graphic'.

Attention

If clipping at the window boundary is desired, exceptions can be switched off by SetCheck (’˜text’).

Parameter

▶ WindowHandle (input control) .......................... window ˜ HWindowX / long
Window identifier.
▶ String (input control) ................................. string ˜ VARIANT ( integer, real, string )
Tuple of output values (all types).
  Default Value : 'hello'

Result

WriteString returns TRUE if the window is valid and the output text fits within the current line (see SetCheck). Otherwise an exception handling is raised.

Parallelization Information

WriteString is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow, OpenTextwindow, SetFont, GetStringExtents

See Also

SetTposition, GetStringExtents, OpenTextwindow, SetFont, SetSystem, SetCheck

Alternatives

FwriteString

Module

System

4.8  Window

void HWindowX.ClearRectangle ([in] VARIANT Row1, [in] VARIANT Column1,
[in] VARIANT Row2, [in] VARIANT Column2 )

void HOperatorSetX.ClearRectangle ([in] long WindowHandle,
[in] VARIANT Row1, [in] VARIANT Column1, [in] VARIANT Row2,
[in] VARIANT Column2 )

Delete a rectangle on the output window.

ClearRectangle deletes all entries in the rectangle which is defined through the upper left corner (Row1,Column1) and the lower right corner (Row2,Column2). Deletion signifies that the specified rectangle is set to the background color (see OpenWindow, OpenTextwindow).

If you want to delete more than one rectangle, you may pass several rectangles, i.e. the parameters Row1, Column1,Row2 and Column2 are tupel.

Parameter

▶ WindowHandle (input control) .......................... window ˜ HWindowX / long
Window identifier.
▶ Row1 (input control) ................................. rectangle.origin.y(-array) ˜ VARIANT ( integer )
  Line index of upper left corner.
  Default Value : 10
  Typical Range of Values : 0 ≤ Row1 ≤ 0(lin)
  Minimal Value Step : 1
  Recommended Value Step : 1
▶ Column1 (input control) ................................. rectangle.origin.x(-array) ˜ VARIANT ( integer )
  Column index of upper left corner.
  Default Value : 10
  Typical Range of Values : 0 ≤ Column1 ≤ 0(lin)
  Minimal Value Step : 1
  Recommended Value Step : 1
Row2 (input control) .......................... rectangle.corner.y(-array) ~ VARIANT (integer)
            Row index of lower right corner.
            Default Value : 118
            Typical Range of Values : 0 \leq \text{Row2} \leq 0 (\text{lin})
            Minimal Value Step : 1
            Recommended Value Step : 1
            Restriction : (\text{Row2} > \text{Row1})

Column2 (input control) .......................... rectangle.corner.x(-array) ~ VARIANT (integer)
            Column index of lower right corner.
            Default Value : 118
            Typical Range of Values : 0 \leq \text{Column2} \leq 0 (\text{lin})
            Minimal Value Step : 1
            Recommended Value Step : 1
            Restriction : (\text{Column2} \geq \text{Column1})

Example

/* "Interaktives", Loeschen eines Rechtecks im Ausgabefenster: */
draw_rectangle1(WindowHandle,L1,C1,L2,C2)
clear_rectangle(WindowHandle,L1,C1,L2,C2).

Result

If an output window exists and the specified parameters are correct ClearRectangle returns TRUE. If necessary an exception handling is raised.

Parallelization Information

ClearRectangle is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, SetRgb, SetHsi, DrawRectangle1

See Also

OpenWindow, OpenTextwindow

Alternatives

ClearWindow, DispRectangle1

Module

System

void HWindowX.ClearWindow ( )
void HOperatorSetX.ClearWindow ([in] long WindowHandle )

Delete an output window.

ClearWindow deletes all entries in the output window. The window (background and edge) is reset to its original state. Parameters assigned to this window (e.g. with SetColor, SetPaint, etc.) remain unmodified.

Parameter

WindowHandle (input control) .......................... window ~ HWindowX / long
            Window identifier.

Example

clear_window(WindowHandle).

Result

If the output window is valid ClearWindow returns TRUE. If necessary an exception handling is raised.

Parallelization Information

ClearWindow is local and processed completely exclusively without parallelization.
4.8. WINDOW

Possible Predecessor Functions

OpenWindow, OpenTextwindow

See Also

OpenWindow, OpenTextwindow

Alternatives

ClearRectangle, DispRectangle1

Module

System

**void HOperatorSetX.CloseWindow ([in] long WindowHandle )**

Close an output window.

CloseWindow closes a window which have been opened by OpenWindow or by OpenTextwindow. Afterwards the output device or the window area, respectively, is ready to accept new calls of OpenWindow or OpenTextwindow.

**Parameter**

► **WindowHandle** (input control) .......................... window ~ HWindowX / long Window identifier.

**Result**

If the output window is valid CloseWindow returns TRUE. If necessary an exception handling is raised.

Parallelization Information

CloseWindow is **local** and processed **completely exclusively** without parallelization.

Possible Predecessor Functions

OpenWindow, OpenTextwindow

See Also

OpenWindow, OpenTextwindow

Module

System


Copy all pixels within rectangles between output windows.

CopyRectangle copies all pixels from the specified window with the logical window number WindowHandleSource in the specified window with the logical window number WindowHandleDestination. It copies pixels which reside inside a rectangle which is specified by parameters Row1, Column1, Row2 and Column2. The target position is specified through the upper left corner of the rectangle (DestRow, DestColumn).

If you want to move more than one rectangle, you may pass them at once (in form of the tuple mode).

You may use CopyRectangle to copy edited graphics from an "invisible" window in a visible window. Therefore a window with the option "buffer" is opened. The graphics is then displayed in this window and is copied in a visible window afterwards. The advantage of this strategy is, that CopyRectangle is much more rapid than output procedures as e.g. DispChannel. This means a particular advantage while using demo programs. You could even realise short "clips": you have to create for every image of a sequence a window of a "buffer" type
and pass the data into it. Output is the image sequence whereat all buffers are copied one after another in a visible window.

Attention
Both windows have to reside on the same computer.

Parameter

- **WindowHandleSource** (input control)  
  Number of the source window.

- **WindowHandleDestination** (input control)  
  Number of the destination window.

- **Row1** (input control)  
  Row index of upper left corner in the source window.
  - Default Value: 0
  - Typical Range of Values: \(0 \leq \text{Row1} \leq 0\) (lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 1

- **Column1** (input control)  
  Column index of upper left corner in the source window.
  - Default Value: 0
  - Typical Range of Values: \(0 \leq \text{Column1} \leq 0\) (lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 1

- **Row2** (input control)  
  Row index of lower right corner in the source window.
  - Default Value: 128
  - Typical Range of Values: \(0 \leq \text{Row2} \leq 0\) (lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 1
  - Restriction: \(\text{Row2} \geq \text{Row1}\)

- **Column2** (input control)  
  Column index of lower right corner in the source window.
  - Default Value: 128
  - Typical Range of Values: \(0 \leq \text{Column2} \leq 0\) (lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 1
  - Restriction: \(\text{Column2} \geq \text{Column1}\)

- **DestRow** (input control)  
  Row index of upper left corner in the target window.
  - Default Value: 0
  - Typical Range of Values: \(0 \leq \text{DestRow} \leq 0\) (lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 1

- **DestColumn** (input control)  
  Column index of upper left corner in the target window.
  - Default Value: 0
  - Typical Range of Values: \(0 \leq \text{DestColumn} \leq 0\) (lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 1

Example

```c
read_image(Image,'affe')
open_window(0,0,-1,-1,'root','buffer', '',WindowHandle)
disp_image(Image,WindowHandle)
open_window(0,0,-1,-1,'root','visible', '',WindowHandleDestination)
repeat()
  get_mbutton(WindowHandleDestination, Row, Column, Button)
  copy_rectangle(BufferID, WindowHandleDestination, 20, 90, 120, 390, Row, Column)
```
4.8. WINDOW

until(Button = 1)
close_window(WindowHandleDestination)
close_window(WindowHandle)
clear(Image).

Result
If the output window is valid and if the specified parameters are correct CloseWindow returns TRUE. If necessary an exception handling is raised.

Parallelization Information
CopyRectangle is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
OpenWindow, OpenTextwindow

Possible Successor Functions
CloseWindow

Possible Predecessor Functions
OpenWindow, OpenTextwindow

Possible Successor Functions
CloseWindow

See Also
OpenWindow, OpenTextwindow

Alternatives
MoveRectangle, SlideImage

Module
System

```
void HWindowX.DumpWindow ([in] VARIANT Device, [in] String FileName )
```

Write the window content to a file.

DumpWindow writes the content of the window to a file. You may continue to process this file by convenient printers or other programs. The content of a display is prepared for each special device (Device), i.e. it is formatted in a manner, that you may print this file directly or it can be processed furthermore by a graphical program.

To transform gray values the current color table of the window is used, i.e. the values of SetLutStyle remain unconsidered.

The gray values of the display have in general a poorer resolution then the original gray values of the output images. This results in a reduced resolution (default setting are e.g. 200 gray levels with 8 bit planes) for the representation of gray values (see also DispChannel). You may request the amount of actual disposable gray levels by GetSystem. To modify them before opening the first window you may use SetSystem.

Possible values for Device

'postscript': PostScript - file. Filetermination: <FileName>.ps
'postscript',Width,Height: PostScript - file with specification of the output size. Width and Height refer to the size. In this case a tuple with three values as Device is passed. Filetermination: <FileName>.ps
'tiff': TIFF - file, 1 byte per pixel incl. current color table or 3 bytes per sample (dependent on VGA card), uncompressed. Filetermination: <FileName>.tiff
'bmp': Windows-BMP format, RGB image, 3 bytes per pixel. The color resolution pends on the VGA card. File extension: '.bmp'
'jpeg' JPEG format, with lost of information; together with the format string the quality value determining the compression rate can be provided: e.g., 'jpeg 30'.

Attention
Under X Windows, the HALCON window must be completely visible on the root window, because otherwise the contents of the window cannot be read due to limitations in X Windows. If larger graphical displays are to be written to a file, the window type 'pixmap' can be used.

HALCON 6.0
Parameter

- **WindowHandle** (input control) window \(\sim HWindowX/\text{long}
Window identifier.
- **Device** (input control) string(-array) \(\sim\) VARIANT (string, integer)
Name of the target device or of the graphic format.
  - **Default Value**: 'postscript'
- **FileName** (input control) filename \(\sim\) String
File name (without extension).
  - **Default Value**: 'halcon_dump'

Example

```c
/* PostScript - Dump von Bild und eingeblendeten Regionen: */
disp_image(Image,WindowHandle)
set_colored(WindowHandle,12)
disp_region(Regions,WindowHandle)
dump_window(WindowHandle,’postscript’,’/tmp/halcon_dump’)
system_call(’lp -d ps /tmp/halcon_dump.ps’).
```

Result

If the appropriate window is valid and the specified parameters are correct
DumpWindow returns TRUE. If necessary an exception handling is raised.

Parallelization Information

DumpWindow is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, OpenTextWindow, DispRegion

Possible Successor Functions

SystemCall

See Also

OpenWindow, OpenTextWindow, SetSystem

Module

System

```
[out] long Row HWindowX.GetWindowExtents ([out] long Column,
[out] long Width, [out] long Height )
```

Information about a window’s size and position.

GetWindowExtents returns the position of the upper left corner, as well as width and height of the output window.

Attention

Size and position of a window may be modified by the window manager, without explicit instruction in the program. Therefore the values which are returned by GetWindowExtents may change cause of side effects.

- **WindowHandle** (input control) window \(\sim HWindowX/\text{long}
Window identifier.
- **Row** (output control) rectangle.origin.y \(\sim\) long
  Row index of upper left corner of the window.
- **Column** (output control) rectangle.origin.x \(\sim\) long
  Column index of upper left corner of the window.
4.8. WINDOW

- **Width** (output control) ....................................................... \( \text{rectangle.extent.x} \sim \text{long} \)
  Window width.

- **Height** (output control) ....................................................... \( \text{rectangle.extent.y} \sim \text{long} \)
  Window height.

**Example**

```c
open_window(100,100,200,200,'root','visible','','WindowHandle')
fwrite_string('Move the window with the mouse!')
fnew_line()
repeat()
get_mbutton(WindowHandle,_,_,Button)
get_window_extents(WindowHandle,Row,Column,Width,Height)
fwrite(['('Row,',Column,)'])
fnew_line()
until(Button = 4).
```

**Result**

If the window is valid `GetWindowExtents` returns `TRUE`. If necessary an exception handling is raised.

**Parallelization Information**

`GetWindowExtents` is **local** and processed **completely exclusively** without parallelization.

**Possible Predecessor Functions**

`OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, OpenTextwindow`

**See Also**

`SetWindowExtents, OpenWindow, OpenTextwindow`

**Module**

```c
[out] long ImageRed  HWindowX.GetWindowPointer3 {[out] long ImageGreen,
[out] long ImageBlue, [out] long Width, [out] long Height }

void HOperatorSetX.GetWindowPointer3 ([in] long WindowHandle,
[out] long ImageRed, [out] long ImageGreen, [out] long ImageBlue,
[out] long Width, [out] long Height )
```

Access to a window’s pixel data.

`GetWindowPointer3` enables (in some window systems) the direct access to the bitmap. Result values are the three pointers on the color extracts of a 24-bit window (ImageRed, ImageGreen, ImageBlue), as well as the window size (Width, Height). In the language C the type of the image points is unsigned char.

**Attention**

`GetWindowPointer3` is usable only for window type ‘pixmap’.

**Parameter**

- **WindowHandle** (input control) .............................................. \( \text{window} \sim \text{HWindowX/long} \)
  Window identifier.

- **ImageRed** (output control) .................................................. \( \text{integer} \sim \text{long} \)
  Pointer on red channel of pixel data.

- **ImageGreen** (output control) ................................................ \( \text{integer} \sim \text{long} \)
  Pointer on green channel of pixel data.

- **ImageBlue** (output control) ................................................ \( \text{integer} \sim \text{long} \)
  Pointer on blue channel of pixel data.

- **Width** (output control) ...................................................... \( \text{extent.x} \sim \text{long} \)
  Length of an image line.

- **Height** (output control) .................................................... \( \text{extent.y} \sim \text{long} \)
  Number of image lines.
CHAPTER 4. GRAPHICS

Result

If a window of type ‘pixmap’ exists and it is valid GetWindowPointer3 returns TRUE. If necessary an exception handling is raised.

Parallelization Information

GetWindowPointer3 is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow, OpenTextwindow

See Also

OpenWindow, SetWindowType

System

Get the window type.

GetWindowType determines the type or the graphical software, respectively, of the output device for the window. You may query the available types of output devices with procedure QueryWindowType. A reasonable use for GetWindowType might be in the field of the development of machine independent software. Possible values are:

‘X-Window’ X-Window Version 11.

‘pixmap’ Windows are not shown, but managed in memory. By this means Halcon programs can be ported on computers without a graphical display.

‘PostScript’ Objects are output to a PostScript File.

Parameter

* WindowHandle (input control) ... window ~ HWindowX / long Window identifier.
* WindowType (output control) ... string ~ String Window type

Example

open_window(100,100,200,200,’root’,’visible’,’’,WindowHandle)
get_window_type(WindowHandle,WindowType)
fwrite_string([’Window type: ’,WindowType])
fnew_line().

Result

If the window is valid GetWindowType returns TRUE. If necessary an exception handling is raised.

Parallelization Information

GetWindowType is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

OpenWindow, OpenTextwindow

See Also

QueryWindowType, SetWindowType, GetWindowPointer3, OpenWindow, OpenTextwindow

Module

System
Copy inside an output window.

MoveRectangle copies all entries in the rectangle (Row1,Column1), (Row2,Column2) of the output window to a new position inside the same window. This position is determined by the upper left corner (DestRow, DestColumn). Regions of the window, which are ”uncovered” through moving the rectangle, are set to the color of the background.

If you want to move several rectangles at once, you may pass parameters in form of tuples.

Parameter

- **WindowHandle** (input control) ................................. window ~ HWindowX/long
  Window identifier.

- **Row1** (input control) ................................. rectangle.origin.y(-array) ~ VARIANT (integer)
  Row index of upper left corner of the source rectangle.
  Default Value : 0
  Typical Range of Values : 0 ≤ Row1 ≤ 0(lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

- **Column1** (input control) ................................. rectangle.origin.x(-array) ~ VARIANT (integer)
  Column index of upper left corner of the source rectangle.
  Default Value : 0
  Typical Range of Values : 0 ≤ Column1 ≤ 0(lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

- **Row2** (input control) ................................. rectangle.corner.y(-array) ~ VARIANT (integer)
  Row index of lower right corner of the source rectangle.
  Default Value : 64
  Typical Range of Values : 0 ≤ Row2 ≤ 0(lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

- **Column2** (input control) ................................. rectangle.corner.x(-array) ~ VARIANT (integer)
  Column index of lower right corner of the source rectangle.
  Default Value : 64
  Typical Range of Values : 0 ≤ Column2 ≤ 0(lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

- **DestRow** (input control) ................................. point.y(-array) ~ VARIANT (integer)
  Row index of upper left corner of the target position.
  Default Value : 64
  Typical Range of Values : 0 ≤ DestRow ≤ 0(lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

- **DestColumn** (input control) ................................. point.x(-array) ~ VARIANT (integer)
  Column index of upper left corner of the target position.
  Default Value : 64
  Typical Range of Values : 0 ≤ DestColumn ≤ 0(lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

Example

/* "Interaktives" Kopieren eines Rechtecks im Ausgabefenster: */
draw_rectangle1(WindowHandle,L1,C1,L2,C2)
get_mbutton(WindowHandle,LN,CN,Button)
move_rectangle(WindowHandle,L1,C1,L2,C2,LN,CN).

**Result**

If the window is valid and the specified parameters are correct, MoveRectangle returns TRUE. If necessary an exception handling is raised.

**Parallelization Information**

MoveRectangle is *local* and processed *completely exclusively* without parallelization.

**Possible Predecessor Functions**

OpenWindow, OpenTextwindow

**See Also**

OpenWindow, OpenTextwindow

**Alternatives**

CopyRectangle

**Module**

System

```c
void HWindowX.NewExternWindow ([in] long WINHWnd, [in] long WINHDC,

void HOperatorSetX.NewExternWindow ([in] long WINHWnd,
   [in] long Height, [out] long WindowHandle)
```

Create a virtual graphics window under Windows NT.

NewExternWindow opens a new virtual window. Virtual means that a new window will not be created, but the window whose Windows NT handle is given in the parameter WINHWnd is used to perform output of gray value data, regions, graphics as well as to perform textual output. Visualization parameters for the output of data can be done either using Halcon commands or by the appropriate Windows NT commands.

Example: setting of the drawing color:

Halcon:

```halcon
set_color(WindowHandle,"green");
disp_region(WindowHandle,region);
```

Windows NT:

```c
HPEN* penold;
HPEN penGreen = CreatePen(PS_SOLID,1,RGB(0,255,0));
pen = (HPEN*)SelectObject(WINHDC,penGreen);
disp_region(WindowHandle,region);
```

Interactive operators, for example DrawRegion, DrawCircle or GetMbutton cannot be used in this window. The following operators can be used:

- **Output of gray values**: SetPaint, SetComprise, (SetLut and SetLutStyle after output)
- **Regions**: SetColor, SetRgb, SetHsi, SetGray, SetPixel, SetShape, SetLineWidth, SetInsert, SetLineStyle, SetDraw
- **Image part**: SetPart
- **Text**: SetFont

You may query current set values by calling procedures like GetShape. As some parameters are specified through the hardware (Resolution/Colors), you may query current available resources by calling operators like QueryColor.
The parameter WINHWnd is used to pass the window handle of the Windows NT window, in which output should be done. The parameter WINHDC is used to pass the device context of the window WINHWnd. This device context is used in the output routines of Halcon.

The origin of the coordinate system of the window resides in the upper left corner (coordinates: (0,0)). The row index grows downward (maximum: Height-1), the column index grows to the right (maximal: Width-1).

You may use the value -1 for parameters Width and Height. This means, that the corresponding value is chosen automatically. In particular, this is important if the aspect ratio of the pixels is not 1.0 (see SetSystem). If one of the two parameters is set to -1, it will be chosen through the size which results out of the aspect ratio of the pixels. If both parameters are set to -1, they will be set to the current image format.

The position and size of a window may change during runtime of a program. This may be achieved by calling SetWindowExtents, but also through external influences (window manager). For the latter case the procedure SetWindowExtents is provided.

Opening a window causes the assignment of a default font. It is used in connection with procedures like WriteString and you may change it by performing SetFont after calling OpenWindow. On the other hand, you have the possibility to specify a default font by calling SetSystem(‘defaultFont’,<Fontname>: ) before opening a window (and all following windows; see also QueryFont).

You may set the color of graphics and font, which is used for output procedures like DispRegion or DispCircle, by calling SetRgb, SetHsi, SetGray or SetPixel. Calling SetInsert specifies how graphics is combined with the content of the image repeat memory. Thereto you may achieve by calling e.g. SetInsert(‘:not:’) to eliminate the font after writing text twice at the same position.

The content of the window is not saved, if other windows overlap the window. This must be done in the program code that handles the Windows NT window in the calling program.

For graphical output (DispImage, DispRegion, etc.) you may adjust the window by calling procedure SetPart in order to represent a logical clipping of the image format. In particular this implies that only this part (appropriately scaled) of images and regions is displayed.

Steps to use new extern window:

Creation:  
- Create a Windows-window.
- Call new extern window whit the WINHWnd of the above created window.

Use:  
- Before drawing in the window you have to call the method set window dc. This ensures that the halcon drawing routines use the right DC.

Destroy:  
- Call close window.

Note that parameters as Row, Column, Width and Height are constrained through the output device, i.e., the size of the Windows NT desktop.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WINHWnd</td>
<td>Windows windowhandle of a previously created window.</td>
</tr>
<tr>
<td>WINHDC</td>
<td>Device context of WINHWnd.</td>
</tr>
<tr>
<td>Row</td>
<td>Row coordinate of upper left corner.</td>
</tr>
<tr>
<td>Column</td>
<td>Column coordinate of upper left corner.</td>
</tr>
<tr>
<td>Width</td>
<td>Width of the window.</td>
</tr>
</tbody>
</table>

HALCON 6.0
Height (input control) ............................................. rectangle.extent.y " long
Height of the window.
Default Value : 512
Restriction : \((\text{Height} > 0) \lor (\text{Height} = -1)\)

WindowHandle (output control) .................................. window " HWindowX / long
Window identifier.

Result
If the values of the specified parameters are correct NewExternWindow returns TRUE. If necessary, an exception is raised.

Parallelization Information
NewExternWindow is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
ResetObjDb

Possible Successor Functions
SetColor, QueryWindowType, GetWindowType, SetWindowType, GetMposition, SetTposition, SetTshape, SetWindowExtents, GetWindowExtents, QueryColor, SetCheck, SetSystem

See Also
OpenWindow, DispRegion, DispImage, DispColor, SetLut, QueryColor, SetColor, SetRgb, SetHsi, SetPixel, SetGray, SetPart, SetPartStyle, QueryWindowType, GetWindowType, SetWindowType, GetMposition, SetTposition, SetWindowExtents, GetWindowExtents, SetWindowAttr, SetCheck, SetSystem

Alternatives
OpenWindow, OpenTextwindow

Module
System

```c
void HWindowX.OpenTextwindow ([in] long Row, [in] long Column,
[in] long Width, [in] long Height, [in] long BorderWidth,
in String BorderColor, [in] String BackgroundColor,
```

```c
void HOperatorSetX.OpenTextwindow ([in] long Row, [in] long Column,
[in] long Width, [in] long Height, [in] long BorderWidth,
in String BorderColor, [in] String BackgroundColor,
in VARIANT FatherWindow, [in] String Mode, [in] String Machine,
[out] long WindowHandle )
```

Open a textual window.

OpenTextwindow opens a new textual window, which can be used to perform textual input and output, as well as to perform output of images. All output (WriteString, ReadString, DispRegion, etc.) is redirected to this window, if the same logical window number WindowHandle is used.

Besides the mouse cursor textual windows possess also a textual cursor which indicates the current writing position (more exactly: the lower left corner of the output string without consideration of descenders). Its position is indicated through an underscore or another type (the indication of this position may also be disabled (= default setting); cf. SetTshape). You may set or query the position by calling the procedures SetTposition or GetTposition.

After you opened a textual window the position of the cursor is set to \((H,0)\). Whereby \(H\) signifies the height of the default font less the descenders. But the cursor is not shown. Hence the output starts for writing in the upper left corner of the window.

You may query the colors of the background and the image edges by calling QueryColor. In the same way you may use QueryColor in a window of type ‘invisible’. During output (WriteString) you may set the clipping of text out of the window edges by calling SetCheck("~text"). This disables the creation of error messages, if text passes over the edge of the window.
The origin of the coordinate system of the window resides in the upper left corner (coordinates: (0,0)). The row index grows downward (maximal: Height-1), the column index grows to the right (maximal: Width-1).

The parameter Machine indicates the name of the computer, which has to open the window. In case of a X-window, TCP-IP only sets the name, DEC-Net sets in addition a colon behind the name. The "'server'" or the "'screen'", respectively, are not specified. If the empty string is passed the environment variable DISPLAY is used. It indicates the target computer. At this the name is indicated in common syntax

<Host>:0.0

Position and size of a window may change during runtime of a program. This may be achieved by calling SetWindowExtents, but also through external interferences (window manager). In the latter case the procedure SetWindowExtents is provided.

Opening a window causes the assignment of a called default font. It is used in connection with procedures like WriteString and you may overwrite it by performing SetFont after calling OpenTextwindow. On the other hand you have the possibility to specify a default font by calling SetSystem(:: 'defaultFont',<Fontname>:) before opening a window (and all following windows; see also QueryFont).

You may set the color of the font (WriteString, ReadString) by calling SetColor, SetRgb, SetHsi, SetGray or SetPixel. Calling SetInsert specifies how the text or the graphics, respectively, is combined with the content of the image repeat memory. So you may achieve by calling e.g. SetInsert(::__'not':) to eliminate the font after writing text twice at the same position.

Normally every output (e.g. WriteString, DispRegion, DispCircle, etc.) in a window is terminated by a "'flush'". This causes the data to be fully visible on the display after termination of the output procedure. But this is not necessary in all cases, in particular if there are permanently output tasks or there is a mouse procedure active. Therefore it is more favorable (i.e. more rapid) to store the data until sufficient data is available. You may stop this behavior by calling SetSystem(::__'flushGraphic','false':).

The content of windows is saved (in case it is supported by special driver software); i.e. it is preserved, also if the window is hidden by other windows. But this is not necessary in all cases: If you use a textual window e.g. as a parent window for other windows, you may suppress the security mechanism for it and save the necessary memory at the same moment. You achieve this before opening the window by calling SetSystem(::__'backingStore','false':).

Difference: graphical window - textual window

- In contrast to graphical windows (OpenWindow) you may specify more parameters (color, edge) for a textual window while opening it.
- You may use textual windows only for input of user data (ReadString).
- Using textual windows, the output of images, regions and graphics is "'clipped'" at the edges. Whereas during the use of graphical windows the edges are "'zoomed'".
- The coordinate system (e.g. with GetMbutton or GetMposition) consists of display coordinates independently of image size. The maximum coordinates are equal to the size of the window minus 1. In contrast to this, graphical windows (OpenWindow) use always a coordinate system, which corresponds to the image format.

The parameter Mode specifies the mode of the window. It can have following values:

'visible': Normal mode for textual windows: The window is created according to the parameters and all inputs and outputs are possible.

'invisible': Invisible windows are not displayed in the display. Parameters like Row, Column, BorderWidth, BorderColor, BackgroundColor and FatherWindow do not have any meaning. Output to these windows has no effect. Input (ReadString, mouse, etc.) is not possible. You may use these windows to query representation parameter for an output device without opening a (visible) window. General queries are e.g. QueryColor and GetStringExtents.

'transparent': These windows are transparent: the window itself is not visible (edge and background), but all the other operations are possible and all output is displayed. Parameters like BorderColor and BackgroundColor do not have any meaning. A common use for this mode is the creation of mouse sensitive regions.

HALCON 6.0
buffer*: These are also not visible windows. The output of images, regions and graphics is not visible on
the display, but is stored in memory. Parameters like Row, Column, BorderWidth, BorderColor,
BackgroundColor and FatherWindow do not have any meaning. You may use buffer windows, if you
prepare output (in the background) and copy it finally with CopyRectangle in a visible window. Another
usage might be the rapid processing of image regions during interactive manipulations. Textual input and
mouse interaction are not possible in this mode.

Attention: You have to keep in mind that parameters like Row, Column, Width and Height are restricted by the output
device. Is a father window (FatherWindow <> 'root') specified, then the coordinates are relative to this window.

Parameter

- **Row** (input control) .......................................................... rectangle.origin.y ~ long
  Row index of upper left corner.
  Default Value: 0
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: (Row ≥ 0)

- **Column** (input control) ................................................. rectangle.origin.x ~ long
  Column index of upper left corner.
  Default Value: 0
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: (Column ≥ 0)

- **Width** (input control) .................................................. rectangle.extent.x ~ long
  Window’s width.
  Default Value: 256
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: (Width > 0)

- **Height** (input control) ................................................ rectangle.extent.y ~ long
  Window’s height.
  Default Value: 256
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: (Height > 0)

- **BorderWidth** (input control) ..................................... integer ~ long
  Window border’s width.
  Default Value: 2
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: (BorderWidth ≥ 0)

- **BorderColor** (input control) ................................. string ~ String
  Window border’s color.
  Default Value: ’white’

- **BackgroundColor** (input control) ............................. string ~ String
  Background color.
  Default Value: ’black’

- **FatherWindow** (input control) .................. integer ~ VARIANT (integer, string)
  Logical number of the father window. For the display as father you may specify ’root’ or 0.
  Default Value: 0
  Restriction: (FatherWindow ≥ 0)

- **Mode** (input control) ................................. string ~ String
  Window mode.
  Default Value: ’visible’
  Value List: Mode ∈ {’visible’, ’invisible’, ’transparent’, ’buffer’}

- **Machine** (input control) ................... string ~ String
  Computer name, where the window has to be opened or empty string.
  Default Value: ”
4.8. WINDOW

**Example**

```c
open_textwindow(0,0,900,600,1,'black','slate blue','root','visible',
               'WindowHandle)
open_textwindow(10,10,300,580,3,'red','blue',Father,'visible',
               'WindowHandle)
open_window(10,320,570,580,Father,'visible','WindowHandle)
set_color(WindowHandle,'red')
read_image(Image,'affe')
disp_image(Image,WindowHandle)
repeat()
get_mposition(WindowHandle,Row,Column,Button)
get_grayval(Image,Row,Column,1,Gray)
write_string(WindowHandle,[ ' Position (',Row,',',Column,') '])
write_string(WindowHandle,[ 'Gray value (',Gray,') '])
new_line(WindowHandle)
until(Button = 4)
close_window(WindowHandle)
clear_obj(Image).
```

**Result**

If the values of the specified parameters are correct `OpenTextwindow` returns TRUE. If necessary an exception handling is raised.

**Parallelization Information**

`OpenTextwindow` is **local** and processed **completely exclusively** without parallelization.

**Possible Predecessor Functions**

`ResetObjDb`

**Possible Successor Functions**

`SetColor, QueryWindowType, GetWindowType, GetMposition, SetTposition, SetTshape, SetWindowExtents, GetWindowExtents, QueryColor, SetCheck, SetSystem`

See Also

`WriteString, ReadString, NewLine, GetStringExtents, GetTposition, GetTshape, GetWindowExtents, QueryColor, SetCheck, SetSystem`

**Alternatives**

`OpenWindow`

**Module**

**System**

```c
void HWindowX.OpenWindow ([in] long Row, [in] long Column,
                          [in] long Width, [in] long Height, [in] VARIANT FatherWindow,

void HOperatorSetX.OpenWindow ([in] long Row, [in] long Column,
                               [in] long Width, [in] long Height, [in] VARIANT FatherWindow,
```

**OpenWindow**

`OpenWindow` opens a new window, which can be used to perform output of gray value data, regions, graphics as well as to perform textual output. All output (`DispRegion, DispImage, etc.`) is redirected to this window, if the same logical window number `WindowHandle` is used.
The background of the created window is set to black in advance and it has a white border, which is 2 pixels wide (see also setWindowAttr('borderWidth',<Breite>):).

Certain parameters for the editing of output data are assigned to a window. These parameters are considered during the output itself (e.g. with DispImage or DispRegion). They are not specified by an output procedure, but by "configuration procedures". If you want to set e.g. the color red for the output of regions, you have to call SetColor(::<WindowHandle,'red':) before calling DispRegion. These parameters are always set for the window with the logical window number WindowHandle and remain assigned to a window as long as they will be overwritten. You may use the following configuration procedures:

- Output of gray values: SetPaint, SetComprise,(SetLut and SetLutStyle after output)
- Regions: SetColor, SetRgb, SetHsi, SetGray, SetPixel, SetShape, SetLineWidth, SetInsert, SetLineStyle, SetDraw
- Image clipping: SetPart
- Text: SetFont

You may query current set values by calling procedures like GetShape. As some parameters are specified through the hardware (Resolution/Colors), you may query current available resources by calling QueryColor.

The origin of the coordinate system of the window resides in the upper left corner (coordinates: (0,0)). The row index grows downward (maximal: Height-1), the column index grows to the right (maximal: Width-1). You have to keep in mind, that the range of the coordinate system is independent of the window size. It is specified only through the image format (see ResetObjDb).

The parameter Machine indicates the name of the computer, which has to open the window. In case of a X-window, TCP-IP only sets the name, DEC-Net sets in addition a colon behind the name. The ""server"" resp. the ""screen"" are not specified. If the empty string is passed the environment variable DISPLAY is used. It indicates the target computer. At this the name is indicated in common syntax

<Host>:0.0

You may use the value "'-1'" for parameters Width and Height. This means, that the according value has to be specified automatically. In particular this is of importance, if the proportion of pixels is not 1.0 (see SetSystem): Is one of the two parameters set to "'-1'", it will be specified through the size which results out of the proportion of pixels. Are both parameters set to "'-1'", they will be set to the maximum image format, which is currently used (further information about the currently used maximum image format can be found in the description of GetSystem using "width" or "height").

Position and size of a window may change during runtime of a program. This may be achieved by calling SetWindowExtents, but also through external interferences (window manager). In the latter case the procedure SetWindowExtents is provided.

Opening a window causes the assignment of a called default font. It is used in connection with procedures like WriteString and you may overwrite it by performing SetFont after calling OpenWindow. On the other hand you have the possibility to specify a default font by calling SetSystem('defaultFont',<Fontname>:) before opening a window (and all following windows; see also QueryFont).

You may set the color of graphics and font, which is used for output procedures like DispRegion or DispCircle, by calling SetRgb, SetHsi, SetGray or SetPixel. Calling SetInsert specifies how graphics is combined with the content of the image repeat memory. Thereto you may achieve by calling e.g. SetInsert('not:') to eliminate the font after writing text twice at the same position.

Normally every output (e.g. DispImage, DispRegion, DispCircle, etc.) in a window is terminated by a called "flush". This causes the data to be fully visible on the display after termination of the output procedure. But this is not necessary in all cases, in particular if there are permanently output tasks or if there is a mouse procedure active. Therefore it is more favorable (i.e. more rapid) to store the data until sufficient data is available. You may stop this behavior by calling SetSystem('flushGraphic','false').

The content of windows is saved (in case it is supported by special driver software); i.e. it is preserved, also if the window is hidden by other windows. But this is not necessary in all cases: If the content of a window is built up permanently new (CopyRectangle), you may suppress the security mechanism for that and hence you can save the necessary memory. This is done by calling SetSystem('backingStore','false') before opening a window. In doing so you save not only memory but also time to compute. This is significant for the output of video clips (see CopyRectangle).
4.8. WINDOW

For graphical output (DispImage, DispRegion, etc.) you may adjust the window by calling procedure SetPart in order to represent a logical clipping of the image format. In particular this implicates that you obtain this clipping (with appropriate enlargement) of images and regions only.

Difference: graphical window - textual window

- Using graphical windows the layout is not as variable as concerned to textual windows.
- You may use textual windows for the input of user data only (ReadString).
- During the output of images, regions and graphics a "zooming" is performed using graphical windows: Independent on size and side ratio of the window images are transformed in that way, that they are displayed in the window by filling it completely. On the opposite side using textual windows the output does not care about the size of the window (only if clipping is necessary).
- Using graphical windows the coordinate system of the window corresponds to the coordinate system of the image format. Using textual windows, its coordinate system is always equal to the display coordinates independent on image size.

The parameter Mode determines the mode of the window. It may have following values:

'visible': Normal mode for graphical windows: The window is created according to the parameters and all input and output are possible.

'invisible': Invisible windows are not displayed in the display. Parameters like Row, Column and FatherWindow do not have any meaning. Output to these windows has no effect. Input (ReadString, mouse, etc.) is not possible. You may use these windows to query representation parameter for an output device without opening a (visible) window. Common queries are e.g. QueryColor and GetStringExtents.

'transparent': These windows are transparent: the window itself is not visible (edge and background), but all the other operations are possible and all output is displayed. A common use for this mode is the creation of mouse sensitive regions.

'buffer': These are also not visible windows. The output of images, regions and graphics is not visible on the display, but is stored in memory. Parameters like Row, Column and FatherWindow do not have any meaning. You may use buffer windows, if you prepare output (in the background) and copy it finally with CopyRectangle in a visible window. Another usage might be the rapid processing of image regions during interactive manipulations. Textual input and mouse interaction are not possible in this mode.

---

Attention

You may keep in mind that parameters as Row, Column, Width and Height are constrained by the output device. If you specify a father window (FatherWindow <> 'root') the coordinates are relative to this window.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| Row       | 0     | Row index of upper left corner. Default Value: 0 (lin)
|           |       | Minimal Value Step: 1 |
|           |       | Recommended Value Step: 1 |
|           |       | Restriction: (Row >= 0) |
| Column    | 0     | Column index of upper left corner. Default Value: 0 (lin)
|           |       | Minimal Value Step: 1 |
|           |       | Recommended Value Step: 1 |
|           |       | Restriction: (Column >= 0) |
| Width     | 256   | Width of the window. Default Value: 256 (lin) Minimal Value Step: 1 |
|           |       | Recommended Value Step: 1 |
|           |       | Restriction: ((Width > 0) ∨ (Width = -1)) |
**Height** (input control) ........................................... rectangle.extent.y $\sim$ long
Height of the window.

**Default Value** : 256

Minimal Value Step : 1

**Recommended Value Step** : 1

**Restriction** : $((Height > 0) \lor (Height = 0))$

**FatherWindow** (input control) ...................................... integer $\sim$ VARIANT ( integer, string )
Logical number of the father window. To specify the display as father you may enter 'root' or 0.

**Default Value** : 0

**Restriction** : $\mathbf{FatherWindow} \geq 0$

**Mode** (input control) .................................................. string $\sim$ String
Window mode.

**Default Value** : 'visible'

**Value List** : Mode $\in$ { 'visible', 'invisible', 'transparent', 'buffer' }

**Machine** (input control) ........................................... string $\sim$ String
Name of the computer on which you want to open the window. Otherwise the empty string.

**Default Value** : "

**WindowHandle** (output control) .................................... window $\sim$ HWindowX / long
Window identifier.

---

**Example**

```halcon
set_system('pixel_ratio', 1.53)
open_window(0, 0, 400, 400, 'root', 'visible', '', WindowHandle)
read_image(Image, 'fabrik')
disp_image(Image, WindowHandle)
write_string(WindowHandle, 'File, fabrik.ima')
ew_line(WindowHandle)
get_mbutton(WindowHandle, -, -, -)
set_lut(WindowHandle, 'temperature')
set_color(WindowHandle, 'blue')
write_string(WindowHandle, 'temperature')
ew_line(WindowHandle)
write_string(WindowHandle, 'Draw Rectangle')
ew_line(WindowHandle)
draw_rectangle1(WindowHandle, Row1, Column1, Row2, Column2)
set_part(Row1, Column1, Row2, Column2)
disp_image(Image, WindowHandle)
ew_line(WindowHandle).
```

---

**Result**

If the values of the specified parameters are correct `OpenWindow` returns TRUE. If necessary an exception handling is raised.

---

**Parallelization Information**

`OpenWindow` is local and processed completely exclusively without parallelization.

---

**Possible Predecessor Functions**

`ResetObjDb`

---

**Possible Successor Functions**

`SetColor, QueryWindowType, GetWindowType, GetMposition, SetPosition, SetTshape, SetWindowExtents, GetWindowExtents, QueryColor, SetCheck, SetSystem`

---

**See Also**

`DispRegion, DispImage, DispColor, SetLut, QueryColor, SetColor, SetRgb, SetHsi, SetPixel, SetGray, SetPart, SetPartStyle, QueryWindowType, GetWindowType, SetWindowType, GetMposition, SetPosition, SetWindowExtents, GetWindowExtents, SetWindowAttr, SetCheck, SetSystem`

---

**Alternatives**

`OpenTextwindow`

---

Module

4.8. WINDOW

System

[out] VARIANT WindowTypes HInfoX.QueryWindowType ( )
[out] VARIANT WindowTypes HSystemX.QueryWindowType ( )
void HOperatorSetX.QueryWindowType ([out] VARIANT WindowTypes )

Query all available window types.

QueryWindowType returns a tuple which contains all devices or software systems, respectively, which are used
to display image objects. You may use QueryWindowType usefully while developing machine independent
programs. Possible values are:

'X-Window' X-Window Version 11.
'pixmap' Windows are not displayed, but managed in memory. In this manner it is possible to port Halcon
programs to computers without graphical display.
'PostScript' Objects are output to a PostScript File.

Parameter

WindowTypes (output control) ..........................................string ~ VARIANT( string )
Names of available window types.

Result

QueryWindowType always returns TRUE.

Parallelization Information

QueryWindowType is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

ResetObjDb

Module

void HSystemX.SetWindowAttr ([in] String AttributeName,
[in] VARIANT AttributeValue )
void HOperatorSetX.SetWindowAttr ([in] String AttributeName,
[in] VARIANT AttributeValue )

Set window's characteristics.

You may use SetWindowAttr to set specific characteristics of graphical windows. With it you may modify
following default parameters of a window:

'border_width' Width of the window border in pixels.
'border_color' Color of the window border.
'background_color' Background color of the window.
>window_title' Name of the window in the titlebar.

Attention

You have to call SetWindowAttr before calling OpenWindow.
 CHAPTER 4. GRAPHICS

Parameter

- **AttributeName** (input control)  String
  Name of the attribut, which has to be modified.
  **Value List**: AttributeName ∈ {'border_width', 'border_color', 'background_color', 'window_title'}

- **AttributeValue** (input control)  VARIANT(string, integer)
  Value of the attribut, which has to be set.
  **Value List**: AttributeValue ∈ {0, 1, 2, 'white', 'black', 'MyName', 'default'}

---

Result

If the parameters are correct SetWindowAttr returns TRUE. If necessary an exception handling is raised.

---

Parallelization Information

SetWindowAttr is reentrant, local, and processed without parallelization.

---

Possible Predecessor Functions

OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, OpenTextwindow

---

See Also

OpenWindow

---

Module

System

```c
void HWindowX.SetWindowDc ([in] long WINHDC )

void HOperatorSetX.SetWindowDc ([in] long WindowHandle,
[in] long WINHDC )
```

Set the device context of a virtual graphics window (Windows NT).

SetWindowDc sets the device context of a window previously opened with NewExternWindow. All output (DispRegion, DispImage, etc.) is done in the window with this device context.

The parameter WINHDC contains the device context of the window in which Halcon should output its data. This device context is used in all output routines of Halcon.

---

Attention

The window WindowHandle has to created with NewExternWindow beforehand.

---

Parameter

- **WindowHandle** (input control)  window
  Window identifier.

- **WINHDC** (input control)  integer
  devicecontext of WINHWnd.
  **Restriction**: (WINHDC ≠ 0)

---

Example

```c
set_system('pixel_ratio',1.53)
hWnd = createWINDOW(...)
newExternWindow(hwnd, hdc, 0,0,400,-1,WindowHandle)
set_device_context(WindowHandle, hdc)
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
write_string(WindowHandle,'File, fabrik.ima')
new_line(WindowHandle)
get_mbutton(WindowHandle,_,_,_)
set_lut(WindowHandle,'temperature')
set_color(WindowHandle,'blue')
write_string(WindowHandle,'temperature')
new_line(WindowHandle)
write_string(WindowHandle,'Draw Rectangle')
new_line(WindowHandle)
```

4.8. WINDOW

If the values of the specified parameters are correct, SetWindowDc returns TRUE. If necessary, an exception is raised.

Parallelization Information

SetWindowDc is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

NewExternWindow, DispImage, DispRegion

Possible Successor Functions

DispImage, DispRegion

See Also

NewExternWindow, DispRegion, DispImage, DispColor, SetLut, QueryColor, SetColor, SetRgb, SetHsi, SetPixel, SetGray, SetPart, SetPartStyle, QueryWindowType, GetWindowType, SetWindowType, GetMposition, SetTposition, SetWindowExtents, GetWindowExtents, SetWindowAttr, SetCheck, SetSystem

Module

System

Modify position and size of a window.

SetWindowExtents positions the upper left corner of the output window at (Row, Column) and changes the size of the window to Width and Height at the same time.

Attention

If you modify the size of a window an adaptation of the displayed data to the new format is not processed automatically. This has to be done by the program in performing another output of these data.

Parameter

- **WindowHandle** (input control) window \( \sim \) HWindowX / long Window identifier.
- **Row** (input control) rectangle.origin.y \( \sim \) long Row index of upper left corner in target position.
  - Default Value: 0
  - Minimal Value Step: 1
  - Recommended Value Step: 1
- **Column** (input control) rectangle.origin.x \( \sim \) long Column index of upper left corner in target position.
  - Default Value: 0
  - Minimal Value Step: 1
  - Recommended Value Step: 1
- **Width** (input control) rectangle.extent.x \( \sim \) long Width of the window.
  - Default Value: 512
  - Minimal Value Step: 1
  - Recommended Value Step: 1
**Height** (input control) 

- **rectangle.extent.y** ~ long
- **Height of the window.**
- **Default Value:** 512
- **(lin)** **Minimal Value Step:** 1
- **Recommended Value Step:** 1

**Result**

If the window is valid and the parameters are correct `SetWindowExtents` returns TRUE. If necessary an exception handling is raised.

**Parallelization Information**

`SetWindowExtents` is *local* and processed *completely exclusively* without parallelization.

**Possible Predecessor Functions**

- `OpenWindow`, `OpenTextwindow`

**See Also**

- `GetWindowExtents`, `OpenWindow`, `OpenTextwindow`

**Module**

- `System`

---

```c
void HSystemX.SetWindowType ([in] String WindowType )
void HOperatorSetX.SetWindowType ([in] String WindowType )
```

**Specify a window type.**

`SetWindowType` determines on which type of output device the output is going to be displayed. This specification is going to be used by procedure `OpenWindow` while opening the windows. You may open different windows on different types of output devices. Therefore you have to specify the wanted type before opening. You may request the available types of output devices by calling procedure `QueryWindowType`. Possible values are:

- **‘X-Window’** X-Window Version 11.
- **‘WIN32-Window’** Microsoft Windows.
- **‘pixmap’** Windows are not displayed, but managed in memory only. In this manner you may port Halcon programs to computers without graphical display.
- **‘PostScript’** Objects are output to a PostScript File.

A useful usage of `SetWindowType` could be the development of machine independent programs.

**Parameter**

- **WindowType** (input control) 
- **string** ~ **String**
- **Name of the window type which has to be set.**
- **Default Value: ‘X-Window’**
- **Value List:** WindowType ∈ {'X-Window’, ‘WIN32-Window’, ‘pixmap’, ‘PostScript’}

**Result**

If the type of the output device is available, then `SetWindowType` returns TRUE. If necessary an exception handling is raised.

**Parallelization Information**

`SetWindowType` is *local* and processed *completely exclusively* without parallelization.

**Possible Predecessor Functions**

- `OpenWindow`, `OpenTextwindow`

**See Also**

- `OpenWindow`, `OpenTextwindow`, `QueryWindowType`, `GetWindowType`

**Module**

- `System`

---

Interactive output from two window buffers.

`SlideImage` divides the window horizontally into two logical areas depending on the mouse position. The content of the first indicated window is copied into the upper area, the content of the second window is copied into the lower area. When you press the left mouse button you may scroll the delimitation between the two areas (you may move the mouse outside the window, too). In doing so the position of the mouse relative to the window determines the borderline.

Pressing the right mouse button in the window terminates the procedure `SlideImage`.

A useful application of procedure `SlideImage` might be the visualization of the effect of a filtering operation for an image. The output is directed to the current set window (`WindowHandle`).

```
read_image(Image,'fabrik')
sobel_amp(Image,Amp,'sum_abs',3)
on_open_window(0,0,-1,-1,'root','buffer','',WindowHandle)
disp_image(Amp,WindowHandle)
sobel_dir(Image,Dir,'sum_abs',3)
on_open_window(0,0,-1,-1,'root','buffer','',WindowHandle)
disp_image(Dir,WindowHandle)
on_open_window(0,0,-1,-1,'root','visible','',WindowHandle)
slide_image(Puffer1,Puffer2,WindowHandle).
```

If the both windows exist and one of these windows is valid `SlideImage` returns TRUE. If necessary an exception handling is raised.

**Attention**

The three windows must have the same size and have to reside on the same computer.

**Parameter**

- `WindowHandleSource1` (input control) ......................... window ~ `HWindowX`/`long`
  Logical window number of the "upper window".
- `WindowHandleSource2` (input control) ......................... window ~ `HWindowX`/`long`
  Logical window number of the "lower window".
- `WindowHandle` (input control) ................................. window ~ `HWindowX`/`long`
  Window identifier.

**Example**

**Result**

If the both windows exist and one of these windows is valid `SlideImage` returns TRUE. If necessary an exception handling is raised.

**Parallelization Information**

`SlideImage` is local and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

- `OpenWindow`, `OpenTextWindow`

**See Also**

- `OpenWindow`, `OpenTextWindow`, `MoveRectangle`

**Alternatives**

- `CopyRectangle`, `GetMPosition`

**Module**

System
Chapter 5

Image

5.1 Channel

Access a channel of a multichannel image.

The operator AccessChannel accesses a channel of the (multichannel) input image. The result is a one-channel image. The definition domain of the input is adopted. The channels are numbered from 1 to n. The number of channels can be determined via the operator CountChannels.

Parameter

- **MultiChannelImage** (input iconic) multichannel-image-array
- **Image** (output iconic) One channel of MultiChannelImage.
- **Channel** (input control) Index of channel to be accessed.

Default Value: 1

Value Suggestions: Channel ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

Parallelization Information

AccessChannel is reentrant and processed without parallelization.

Possible Predecessor Functions

CountChannels

Possible Successor Functions

DispImage

See Also

CountChannels

Alternatives

Decompose2, Decompose3, Decompose4, Decompose5

Module

Image / region / XLD management

Append additional matrices (channels) to the image.
The operator **AppendChannel** appends the matrices of the image **Image** to the matrices of **MultiChannelImage**. The result is an image containing as many matrices (channels) as **MultiChannelImage** and **Image** combined. The definition domain of the output image is calculated as the average of the definition domains of both input images.

**Parameter**

- **MultiChannelImage** (input iconic) → **HImageX** → **HObjectX**
  Multichannel image.
- **Image** (input iconic) → **HImageX** → **HObjectX**
  Image to be appended.
- **ImageExtended** (output iconic) → **HImageX** → **HUntypedObjectX**
  Image appended by **Image**.

**Parallelization Information**

AppendChannel is **reentrant** and processed **without parallelization**.

**Possible Successor Functions**

- **DispImage**
- **Compose2, Compose3, Compose4, Compose5**

**Module**

Image / region / XLD management

```c
[out] HImageX MultiChannelImage HImageX.ChannelsToImage ( )

void HOperatorSetX.ChannelsToImage ([in] HObjectX Images,
[out] HUntypedObjectX MultiChannelImage )
```

Convert one-channel images into a multichannel image

The operator **ChannelsToImage** converts several one-channel images into a multichannel image. The definition domain of the output image is the average of the definition domains of the input images.

**Parameter**

- **Images** (input iconic) → **HImageX** → **HObjectX**
  One-channel images to be combined into a one-channel image.
- **MultiChannelImage** (output iconic) → **HImageX** → **HUntypedObjectX**
  Multichannel image.

**Parallelization Information**

ChannelsToImage is **reentrant, local**, and processed **without parallelization**.

**Possible Successor Functions**

- **CountChannels, DispImage**

**Module**

Image / region / XLD management

```c
[out] HImageX MultiChannelImage HImageX.Compose2 ([in] HImageX Image2 )

void HOperatorSetX.Compose2 ([in] HObjectX Image1,
[in] HObjectX Image2, [out] HUntypedObjectX MultiChannelImage )
```

Convert two images into a two-channel image.

The operator **Compose2** converts 2 one-channel images into a 2-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.
5.1. CHANNEL

Parameter

▷ **Image1** (input iconic) ............................................. image(-array) \(\sim\) HImageX / IObjectX
Input image 1.

▷ **Image2** (input iconic) ............................................. image(-array) \(\sim\) HImageX / IObjectX
Input image 2.

▷ **MultiChannelImage** (output iconic) ...... multichannel-image-array \(\sim\) HImageX / HUntypedObjectX
Multichannel image.

---

Parallelization Information

Compose2 is *reentrant* and automatically *parallelized* (on *tuple level*).

Possible Successor Functions

DispImage

See Also

Decompose2

Alternatives

AppendChannel

Module

Image / region / XLD management

---

```c
[out] HImageX MultiChannelImage HImageX.Compose3 ([in] HImageX Image2,
[in] HImageX Image3 )
```

```c
void HOperatorSetX.Compose3 ([in] IObjectX Image1,
[in] IObjectX Image2, [in] IObjectX Image3,
[out] HUntypedObjectX MultiChannelImage )
```

Convert 3 images into a three-channel image.

The operator `Compose3` converts 3 one-channel images into a 3-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

Parameter

▷ **Image1** (input iconic) ............................................. image(-array) \(\sim\) HImageX / IObjectX
Input image 1.

▷ **Image2** (input iconic) ............................................. image(-array) \(\sim\) HImageX / IObjectX
Input image 2.

▷ **Image3** (input iconic) ............................................. image(-array) \(\sim\) HImageX / IObjectX
Input image 3.

▷ **MultiChannelImage** (output iconic) ...... multichannel-image-array \(\sim\) HImageX / HUntypedObjectX
Multichannel image.

---

Parallelization Information

Compose3 is *reentrant* and automatically *parallelized* (on *tuple level*).

Possible Successor Functions

DispImage

See Also

Decompose3

Alternatives

AppendChannel

Module

Image / region / XLD management

---

HALCON 6.0
Convert 4 images into a four-channel image.

The operator `Compose4` converts 4 one-channel images into a 4-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

**Parameter**

- **Image1** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) \(\sim\) HImageX / IHObjectX
  - Input image 1.
- **Image2** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) \(\sim\) HImageX / IHObjectX
  - Input image 2.
- **Image3** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) \(\sim\) HImageX / IHObjectX
  - Input image 3.
- **Image4** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) \(\sim\) HImageX / IHObjectX
  - Input image 4.
- **MultiChannelImage** (output iconic) . . . . . . multichannel-image-array \(\sim\) HImageX / HUntypedObjectX
  - Multichannel image.

**Parallelization Information**

`Compose4` is reentrant and automatically parallelized (on tuple level).

**Possible Successor Functions**

- DispImage
- Decompose4
- AppendChannel

**See Also**

- Alternatives
- Module

Image / region / XLD management

Convert 5 images into a five-channel image.

The operator `Compose5` converts 5 one-channel images into a 5-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

**Parameter**

- **Image1** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) \(\sim\) HImageX / IHObjectX
  - Input image 1.
- **Image2** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) \(\sim\) HImageX / IHObjectX
  - Input image 2.
- **Image3** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) \(\sim\) HImageX / IHObjectX
  - Input image 3.
- **Image4** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) \(\sim\) HImageX / IHObjectX
  - Input image 4.
- **Image5** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) \(\sim\) HImageX / IHObjectX
  - Input image 5.
5.1. CHANNEL 293

- **MultiChannelImage** (output iconic) .... multichannel-image-array \( \sim \) HImageX / HUntypedObjectX
  Multichannel image.

  **Parallelization Information**
  Compose5 is reentrant and automatically parallelized (on tuple level).

  **Possible Successor Functions**
  DispImage
  Decompose5
  AppendChannel

  **Alternatives**
  **Module**
  Image / region / XLD management

```c
[out] HImageX MultiChannelImage HImageX.Compose6 ([in] HImageX Image2,
[in] HImageX Image6 )
```

Convert 6 images into a six-channel image.

The operator Compose6 converts 6 one-channel images into a 6-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

**Parameter**

- **Image1** (input iconic) ........................................ image(-array) \( \sim \) HImageX / IHObjectX
  Input image 1.
- **Image2** (input iconic) ........................................ image(-array) \( \sim \) HImageX / IHObjectX
  Input image 2.
- **Image3** (input iconic) ........................................ image(-array) \( \sim \) HImageX / IHObjectX
  Input image 3.
- **Image4** (input iconic) ........................................ image(-array) \( \sim \) HImageX / IHObjectX
  Input image 4.
- **Image5** (input iconic) ........................................ image(-array) \( \sim \) HImageX / IHObjectX
  Input image 5.
- **Image6** (input iconic) ........................................ image(-array) \( \sim \) HImageX / IHObjectX
  Input image 6.
- **MultiChannelImage** (output iconic) .... multichannel-image-array \( \sim \) HImageX / HUntypedObjectX
  Multichannel image.

  **Parallelization Information**
  Compose6 is reentrant and automatically parallelized (on tuple level).

  **Possible Successor Functions**
  DispImage
  Decompose6
  AppendChannel

  **See Also**

  **Alternatives**

  **Module**
  Image / region / XLD management

HALCON 6.0
Convert 7 images into a seven-channel image.

The operator `Compose7` converts 7 one-channel images into a 7-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

### Parameter

- **Image1** (input iconic) .............................................. image-array \( \sim \) HImageX / IHObjectX
  Input image 1.
- **Image2** (input iconic) .............................................. image-array \( \sim \) HImageX / IHObjectX
  Input image 2.
- **Image3** (input iconic) .............................................. image-array \( \sim \) HImageX / IHObjectX
  Input image 3.
- **Image4** (input iconic) .............................................. image-array \( \sim \) HImageX / IHObjectX
  Input image 4.
- **Image5** (input iconic) .............................................. image-array \( \sim \) HImageX / IHObjectX
  Input image 5.
- **Image6** (input iconic) .............................................. image-array \( \sim \) HImageX / IHObjectX
  Input image 6.
- **Image7** (input iconic) .............................................. image-array \( \sim \) HImageX / IHObjectX
  Input image 7.
- **MultiChannelImage** (output iconic) ....................... multichannel-image-array \( \sim \) HImageX / HUntypedObjectX
  Multichannel image.

### Parallelization Information

`Compose7` is reentrant and automatically parallelized (on tuple level).

### Possible Successor Functions

- **DispImage**

### See Also

- **Decompose7**

### Alternatives

- **AppendChannel**

### Module

Image / region / XLD management

Count channels of image.

The operator `CountChannels` counts the number of channels of all input images.

### Parameter

- **MultiChannelImage** (input iconic) ....................... (multichannel-)image-array \( \sim \) HImageX / IHObjectX
  One- or multichannel image.
- **Channels** (output control) ...................................... integer-array \( \sim \) VARIANT (integer)
  Number of channels.
5.1. CHANNEL

Parallelization Information
CountChannels is reentrant and processed without parallelization.

Possible Successor Functions
AccessChannel, AppendChannel, DispImage

Possible Predecessor Functions
AppendChannel, AccessChannel

See Also
Module

Image / region / XLD management

```
[out] HImageX Image1 HImageX.Decompose2 ([out] HImageX Image2 )

void HOperatorSetX.Decompose2 ([in] IHObjectX MultiChannelImage,
[out] HUntypedObjectX Image1, [out] HUntypedObjectX Image2 )
```

Convert a two-channel image into two images.
The operator Decompose2 converts a 2-channel image into two one-channel images with the same definition domain.

Parameter

- **MultiChannelImage** (input iconic) .......... multichannel-image-array & HImageX / IHObjectX Multichannel image.
- **Image1** (output iconic) ......................... image-array & HImageX / HUntypedObjectX Output image 1.
- **Image2** (output iconic) ......................... image-array & HImageX / HUntypedObjectX Output image 2.

Parallelization Information
Decompose2 is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
CountChannels
DispImage

Possible Successor Functions
Compose2

See Also
Alternatives

AccessChannel, ImageToChannels

Module

Image / region / XLD management

```
[out] HImageX Image1 HImageX.Decompose3 ([out] HImageX Image2,
[out] HImageX Image3 )

void HOperatorSetX.Decompose3 ([in] IHObjectX MultiChannelImage,
[out] HUntypedObjectX Image1, [out] HUntypedObjectX Image2,
[out] HUntypedObjectX Image3 )
```

Convert a three-channel image into three images.
The operator Decompose3 converts a 3-channel image into three one-channel images with the same definition domain.

Parameter

- **MultiChannelImage** (input iconic) .......... multichannel-image-array & HImageX / IHObjectX Multichannel image.
Decompose4 is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

CountChannels

Possible Successor Functions

DispImage

See Also

Compose4

Alternatives

AccessChannel, ImageToChannels

Module

Image / region / XLD management

Convert a four-channel image into four images.
The operator Decompose4 converts a 4-channel image into four one-channel images with the same definition domain.

Parameter

- **MultiChannelImage** (input iconic) multichannel-image-array \(\rightarrow\) HImageX / IOBJECTX Multichannel image.
- **Image1** (output iconic) \(\rightarrow\) HImageX / HUntypedObjectX Output image 1.
- **Image2** (output iconic) \(\rightarrow\) HImageX / HUntypedObjectX Output image 2.
- **Image3** (output iconic) \(\rightarrow\) HImageX / HUntypedObjectX Output image 3.
- **Image4** (output iconic) \(\rightarrow\) HImageX / HUntypedObjectX Output image 4.

Parallelization Information

Decompose4 is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

CountChannels

Possible Successor Functions

DispImage

See Also

Compose4

Alternatives

AccessChannel, ImageToChannels

Module

Image / region / XLD management

5.1. CHANNEL

Convert a five-channel image into five images.

The operator Decompose5 converts a 5-channel image into five one-channel images with the same definition domain.

Parameter

- **MultiChannelImage** (input iconic) ........................................... multichannel-image-array $\sim$ HImageX / IHObjectX
  Multichannel image.
- **Image1** (output iconic) ................................................................. image(-array) $\sim$ HImageX / HUntypedObjectX
  Output image 1.
- **Image2** (output iconic) ................................................................. image(-array) $\sim$ HImageX / HUntypedObjectX
  Output image 2.
- **Image3** (output iconic) ................................................................. image(-array) $\sim$ HImageX / HUntypedObjectX
  Output image 3.
- **Image4** (output iconic) ................................................................. image(-array) $\sim$ HImageX / HUntypedObjectX
  Output image 4.
- **Image5** (output iconic) ................................................................. image(-array) $\sim$ HImageX / HUntypedObjectX
  Output image 5.

Parallelization Information

Decompose5 is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

CountChannels

Possible Successor Functions

DispImage

See Also

Compose5

Alternatives

AccessChannel, ImageToChannels

Module

Image / region / XLD management

Convert a six-channel image into six images.

The operator Decompose6 converts a 6-channel image into six one-channel images with the same definition domain.

Parameter

- **MultiChannelImage** (input iconic) ........................................... multichannel-image-array $\sim$ HImageX / IHObjectX
  Multichannel image.
Parallelization Information

Decompose6 is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

CountChannels

Possible Successor Functions

DispImage

See Also

Alternatives

Module

Image / region / XLD management

Convert a seven-channel image into seven images.
The operator Decompose7 converts a 7-channel image into seven one-channel images with the same definition domain.

Parameter

- **MultiChannelImage** (input iconic) . . . . . . . multichannel-image-array \( \Rightarrow \) HImageX / IHObjectX Multichannel image.
- **Image1** (output iconic) . . . . . . . image(-array) \( \Rightarrow \) HImageX / HUntypedObjectX Output image 1.
- **Image2** (output iconic) . . . . . . . image(-array) \( \Rightarrow \) HImageX / HUntypedObjectX Output image 2.
- **Image3** (output iconic) . . . . . . . image(-array) \( \Rightarrow \) HImageX / HUntypedObjectX Output image 3.
- **Image4** (output iconic) . . . . . . . image(-array) \( \Rightarrow \) HImageX / HUntypedObjectX Output image 4.
- **Image5** (output iconic) . . . . . . . image(-array) \( \Rightarrow \) HImageX / HUntypedObjectX Output image 5.
- **Image6** (output iconic) . . . . . . . image(-array) \( \Rightarrow \) HImageX / HUntypedObjectX Output image 6.
ImageToChannels generates a one-channel image for each channel of the multichannel image in MultiChannelImage. The definition domains are adopted from the input image. As many images are created as MultiChannelImage has channels.

Parameter

- **MultiChannelImage**: multichannel-image-array
  - (input iconic) multichannel image to be decomposed.
  - (output iconic) Generated one-channel images.

Parallelization Information

ImageToChannels is reentrant and processed without parallelization.

Possible Predecessor Functions

CountChannels

Possible Successor Functions

DispImage

Alternatives

AccessChannel, Decompose2, Decompose3, Decompose4, Decompose5

Module

Image / region / XLD management

5.2 Creation

CopyImage copies the first channel of the input image into a new (single-channel) image with the same domain as the input image. In contrast to HALCON operators such as CopyObj, the image is copied into a newly
allocated memory segment. This can be used to modify the gray values of the new image, for example (see GetImagePointer1).

Parameter

- **Image** (input iconic)  
  Image to be copied.

- **DupImage** (output iconic)  
  Copied image.

Parallelization Information

CopyImage is **reentrant** and processed **without** parallelization.

Possible Predecessor Functions

ReadImage, GenImageConst

Possible Successor Functions

SetGrayval, GetImagePointer1

See Also

GetImagePointer1

Alternatives

SetGrayval, PaintGray, GenImageConst, GenImageProto

Module

Basic operators

```c
```

Create an image from a pointer to the pixels.

The operator GenImage1 creates an image of the size Width x Height. The pixels in PixelPointer are stored line-sequentially. The type of the given pixels (PixelPointer) must correspond to Type. The storage for the new image is newly created by HALCON. Thus, the storage on the PixelPointer can be released after the call. Since the type of the parameter PixelPointer is generic (long) a cast has to be used for the call.

Attention

Parameter

- **Image** (output iconic)  
  Created image with new image matrix.

- **Type** (input control)  
  Pixel type.
  **Default Value**: 'byte'
  **Value List**: Type ∈ {'int1', 'int2', 'int4', 'byte', 'real', 'direction', 'cyclic'}

- **Width** (input control)  
  Width of image.
  **Default Value**: 512
  **Value Suggestions**: Width ∈ {128, 256, 512, 1024}
  **Typical Range of Values**: 1 ≤ Width ≤ 1(lin)
  **Minimal Value Step**: 1
  **Recommended Value Step**: 10
  **Restriction**: (Width ≥ 1)
5.2. CREATION

Height (input control) ..............................................
Height of image.

Default Value : 512
Value Suggestions : Height ∈ {128, 256, 512, 1024}
Typical Range of Values : 1 ≤ Height ≤ 1(lin)
Minimal Value Step : 1
Recommended Value Step : 10
Restriction : (Height ≥ 1)

PixelPointer (input control) ..........................................
Pointer to first gray value.

Result
If the parameter values are correct, the operator GenImage1 returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information
GenImage1 is reentrant and processed without parallelization.

Possible Predecessor Functions
GenImageConst, GetImagePointer1

See Also
ReduceDomain, PaintGray, PaintRegion, SetGrayval

Alternatives
GenImage3, GenImageConst, GetImagePointer1

Module
Image / region / XLD management

Create an image from a pointer on the pixels with storage management.

The operator GenImage1Extern creates an image of the size Width × Height. The pixels in PixelPointer are stored line-sequentially. The type of the given pixels (PixelPointer) must correspond to Type. Since the type of the parameter PixelPointer is generic (long) a cast must be used for the call.

The memory for the new image is not newly allocated by HALCON, contrary to GenImage1, and thus is not copied either. This means that the memory space that PixelPointer points to must be released by deleting the object Image. This is done by the procedure ClearProc provided by the caller. This procedure must have the following signature:

```c
void ClearProc(void* ptr);
```

It is called when deleting Image. If the memory shall not be released (in the case of frame grabbers or static memory) a procedure “without trunk” or the NULL-Pointer can be passed. Analogous to the parameter PixelPointer the pointer has to be passed to the procedure by casting it to long.

Parameter

Image (output iconic) ..............................................
Created HALCON image.

Type (input control) ..............................................
Pixel type.

Default Value : 'byte'
Value List : Type ∈ {'int1', 'int2', 'int4', 'byte', 'real', 'direction', 'cyclic'}
The operator GenImage1Extern returns the value TRUE if the parameter values are correct. Otherwise an exception handling is raised.

---

**Parallelization Information**

GenImage1Extern is reentrant and processed without parallelization.

---

**See Also**

ReduceDomain, PaintGray, PaintRegion, SetGrayval

---

**Alternatives**

GenImage1, GenImageConst, GetImagePointer1

---

**Module**

Image / region / XLD management

---

```c

```

Create an image with a rectangular domain from a pointer on the pixels (with storage management).

The operator GenImage1Rect creates an image of size (Width + VerticalPitch) * Height. The pixels pointed to by PixelPointer are stored line by line. Since the type of the parameter PixelPointer is generic (long) a cast must be used for the call. VerticalPitch determines the distance in pixels between the last pixel in row n and the first pixel in row n+1 inside of memory. All rows of the ‘input image’ have the same vertical pitch. The width of the output image equals Width + VerticalPitch. The height of input and output image are equal. The domain of the output image Image is a rectangle of the size Width * Height. The parameters HorizontalBitPitch and BitsPerPixel are not supported at the time and have to be passed as default values.

If DoCopy is set 'true', the image data pointed to by PixelPointer is copied and memory for the new image is newly allocated by HALCON. Else the image data is not duplicated and the memory space that PixelPointer points to must be released when deleting the object Image. This is done by the procedure ClearProc provided by the caller. This procedure must have the following signature:
void ClearProc(void* ptr);

It is called when deleting \texttt{Image}. If the memory shall not be released (in the case of frame grabbers or static memory) a procedure “without trunk” or the NULL-pointer can be passed. Analogously to the parameter \texttt{PixelPointer} the pointer has to be passed to the procedure by casting it to long. If \texttt{DoCopy} is 'true' then \texttt{ClearProc} is irrelevant. The operator \texttt{GenImage1Rect} is approximately symmetrical to \texttt{GetImagePointer1Rect}.

- **Image** (output iconic) \texttt{image} \sim \texttt{HImageX / HUntypedObjectX}
  - Created HALCON image.
- **PixelPointer** (input control) \texttt{integer}\sim\texttt{long}
  - Pointer to the first pixel.
- **Width** (input control) \texttt{extent.x} \sim \texttt{long}
  - Width of the image.
  - Default Value: 512
  - Value Suggestions: \texttt{Width} \in \{128, 256, 512, 1024\}
  - Typical Range of Values: \( 1 \leq \texttt{Width} \leq 1 \texttt{(lin)} \)
  - Minimal Value Step: 1
  - Recommended Value Step: 10
  - Restriction: \((\texttt{Width} \geq 1)\)
- **Height** (input control) \texttt{extent.y} \sim \texttt{long}
  - Height of the image.
  - Default Value: 512
  - Value Suggestions: \texttt{Height} \in \{128, 256, 512, 1024\}
  - Typical Range of Values: \( 1 \leq \texttt{Height} \leq 1 \texttt{(lin)} \)
  - Minimal Value Step: 1
  - Recommended Value Step: 10
  - Restriction: \((\texttt{Height} \geq 1)\)
- **VerticalPitch** (input control) \texttt{integer}\sim\texttt{long}
  - Distance between last pixel in row \( n \) and first pixel in row \( n+1 \) of the 'input image'.
  - Restriction: \((\texttt{VerticalPitch} \geq 0)\)
- **HorizontalBitPitch** (input control) \texttt{integer}\sim\texttt{long}
  - Distance between two neighbouring pixels in bits.
  - Default Value: 8
  - Restriction: \((\texttt{HorizontalBitPitch} = 8)\)
- **BitsPerPixel** (input control) \texttt{integer}\sim\texttt{long}
  - Number of used bits per pixel.
  - Default Value: 8
  - Restriction: \((\texttt{BitsPerPixel} = 8)\)
- **DoCopy** (input control) \texttt{string}\sim\texttt{String}
  - Copy image data.
  - Default Value: 'false'
  - Value Suggestions: \texttt{DoCopy} \in \{'true', 'false'\}
- **ClearProc** (input control) \texttt{integer}\sim\texttt{long}
  - Pointer to the procedure releasing the memory of the image when deleting the object.
  - Default Value: 0

---

The operator \texttt{GenImage1Rect} returns the value \texttt{TRUE} if the parameter values are correct. Otherwise an exception handling is raised.

---

**Parallelization Information**

\texttt{GenImage1Rect} is \textit{reentrant} and processed \textit{without} parallelization.

See Also

**GetImagePointer1Rect**

**Alternatives**

\texttt{GenImage1}, \texttt{GenImage1Extern}

**Module**

Image / region / XLD management

HALCON 6.0
Create an image from three pointers to the pixels (red/green/blue).

The operator `GenImage3` creates a three-channel image of the size $Width \times Height$. The pixels in `PixelPointerRed`, `PixelPointerGreen` and `PixelPointerBlue` are stored line-sequentially. The type of the given pixels (`PixelPointerRed` etc.) must correspond to the name of the pixels (`Type`). The storage for the new image is newly created by HALCON. Thus, it can be released after the call. Since the type of the parameters (`PixelPointerRed` etc.) is generic (long) a “cast” must be used for the call.

---

**Parameter**

- **ImageRGB** (output iconic) ................. image $\sim$ `HImageX/ HUntypedObjectX`  
  Created image with new image matrix.

- **Type** (input control) ............................. string $\sim$ `String`  
  Pixel type.  
  Default Value : 'byte'  
  Value List : `Type` $\in \{ \text{'int1', 'int2', 'int4', 'byte', 'real', 'direction', 'cyclic'} \}$

- **Width** (input control) .......................... extent.x $\sim$ `long`  
  Width of image.  
  Default Value : 512  
  Value Suggestions : $Width \in \{128, 256, 512, 1024\}$  
  Typical Range of Values : $1 \leq Width \leq 1($lin$)$  
  Minimal Value Step : 1  
  Recommended Value Step : 10

- **Height** (input control) ......................... extent.y $\sim$ `long`  
  Height of image.  
  Default Value : 512  
  Value Suggestions : $Height \in \{128, 256, 512, 1024\}$  
  Typical Range of Values : $1 \leq Height \leq 1($lin$)$  
  Minimal Value Step : 1  
  Recommended Value Step : 10

- **PixelPointerRed** (input control) ............... `integer` $\sim$ `long`  
  Pointer to first red value (channel 1).

- **PixelPointerGreen** (input control) ............... `integer` $\sim$ `long`  
  Pointer to first green value (channel 2).

- **PixelPointerBlue** (input control) ............... `integer` $\sim$ `long`  
  Pointer to first blue value (channel 3).

---

**Result**

If the parameter values are correct, the operator `GenImage3` returns the value TRUE. Otherwise an exception handling is raised.

---

**Parallelization Information**

`GenImage3` is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

- `GenImageConst`, `GetImagePointer1`

---

**Possible Successor Functions**

- `DispColor`

---

See Also

- `ReduceDomain`, `PaintGray`, `PaintRegion`, `SetGrayval`, `GetImagePointer1`, `Decompose3`
5.2. CREATION

Alternatives
GenImage1, Compose3, GenImageConst

Module
Image / region / XLD management

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>void HImageX.GenImageConst</strong></td>
<td>(in) String Type, (in) long Width, (in) long Height</td>
</tr>
<tr>
<td><strong>void HOperatorSetX.GenImageConst</strong></td>
<td>(out) HUntypedObjectX Image, (in) String Type, (in) long Width, (in) long Height</td>
</tr>
</tbody>
</table>

Create an image with constant gray value.

The operator GenImageConst creates an image of the indicated size. The height and width of the image are determined by Height and Width. HALCON supports the following image types:

- **'byte'** 1 byte per pixel (0..255)
- **'int1'** 1 byte per pixel (-127..127)
- **'int2'** 2 bytes per pixel (-32767..32767)
- **'int4'** 4 bytes per pixel (-2147483647..2147483647)
- **'real'** 4 bytes per pixel, floating point
- **'complex'** two matrices of the type real
- **'dvf'** two matrices of the type int1
- **'dir'** 1 byte per pixel (0..180)
- **'cyclic'** 1 byte per pixel; cyclic arithmetics (0..255).

The default value 0 is set via the operator SetSystem('initNewImage',<true/false>).

Parameter

- **Image** (output iconic) image ~ HImageX / HUntypedObjectX
  Created image with new image matrix.
- **Type** (input control) string ~ String
  Pixel type.
  Default Value: 'byte'
  Value List: Type ∈ {'int1', 'int2', 'int4', 'byte', 'real', 'direction', 'cyclic', 'complex', 'dvf', 'lut'}
- **Width** (input control) extent.x ~ long
  Width of image.
  Default Value: 512
  Value Suggestions: Width ∈ {128, 256, 512, 1024}
  Typical Range of Values: \(1 \leq \text{Width} \leq 1\text{(lin)}\)
  Minimal Value Step: 1
  Recommended Value Step: 10
  Restriction: (Width ≥ 1)
- **Height** (input control) extent.y ~ long
  Height of image.
  Default Value: 512
  Value Suggestions: Height ∈ {128, 256, 512, 1024}
  Typical Range of Values: \(1 \leq \text{Height} \leq 1\text{(lin)}\)
  Minimal Value Step: 1
  Recommended Value Step: 10
  Restriction: (Height ≥ 1)
If the parameter values are correct, the operator GenImageConst returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

GenImageConst is *reentrant* and processed *without* parallelization.

**Possible Successor Functions**

PaintRegion, ReduceDomain, GetImagePointer1, CopyObj

**See Also**

ReduceDomain, PaintGray, PaintRegion, SetGrayval, GetImagePointer1

**Alternatives**

GenImage1, GenImage3

---

**Module**

Image / region / XLD management

```c

```

Create a gray value ramp.

The operator GenImageGrayRamp creates a gray value ramp according to the following equation:

\[ \text{ImageGrayRamp}(r,c) = \text{Alpha}(r - \text{Row}) + \text{Beta}(c - \text{Column}) + \text{Mean} \]

The size of the image is determined by Width and Height. The gray values are of the type byte. Gray values outside the valid area are clipped.

**Attention**

- **ImageGrayRamp** (output iconic) ................. image ∼ HImageX / HUntypedObjectX (byte)
  - Created image with new image matrix.
- **Alpha** (input control) .......................................................... number ∼ double
  - Gradient in line direction.
  - Default Value : 1.0
  - Value Suggestions : Alpha ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
  - Minimal Value Step : 0.000001
  - Recommended Value Step : -0.005
- **Beta** (input control) .......................................................... number ∼ double
  - Gradient in column direction.
  - Default Value : 1.0
  - Value Suggestions : Beta ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
  - Minimal Value Step : 0.000001
  - Recommended Value Step : -0.005
- **Mean** (input control) ....................................................... number ∼ double
  - Mean gray value.
  - Default Value : 128
  - Value Suggestions : Mean ∈ {0, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 255}
  - Minimal Value Step : 1
  - Recommended Value Step : 10
5.2. CREATION

- **Row** (input control) ................................................................. point.y ∼ long
  Line index of reference point.
  **Default Value**: 256
  **Value Suggestions**: Row ∈ {128, 256, 512, 1024}
  **Minimal Value Step**: 1
  **Recommended Value Step**: 10

- **Column** (input control) .............................................................. point.x ∼ long
  Column index of reference point.
  **Default Value**: 256
  **Value Suggestions**: Column ∈ {128, 256, 512, 1024}
  **Minimal Value Step**: 1
  **Recommended Value Step**: 10

- **Width** (input control) ............................................................... extent.x ∼ long
  Width of image.
  **Default Value**: 512
  **Value Suggestions**: Width ∈ {128, 256, 512, 1024}
  **Typical Range of Values**: 1 ≤ Width ≤ 1(lin)
  **Minimal Value Step**: 1
  **Recommended Value Step**: 10
  **Restriction**: (Width ≥ 1)

- **Height** (input control) ............................................................... extent.y ∼ long
  Height of image.
  **Default Value**: 512
  **Value Suggestions**: Height ∈ {128, 256, 512, 1024}
  **Typical Range of Values**: 1 ≤ Height ≤ 1(lin)
  **Minimal Value Step**: 1
  **Recommended Value Step**: 10
  **Restriction**: (Height ≥ 1)

---

**Result**

If the parameter values are correct, GenImageGrayRamp returns the value TRUE. Otherwise an exception handling is raised.

---

**Parallelization Information**

GenImageGrayRamp is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

MomentsGrayPlane

---

**Possible Successor Functions**

PaintRegion, ReduceDomain, GetImagePointer1, CopyObj

---

**See Also**

ReduceDomain, PaintGray

---

**Alternatives**

GenImage1

---

**Module**

Image / region / XLD management

---

[out] HImageX ImageCleared

HImageX.GenImageProto ([in] VARIANT Grayval )

void HOperatorSetX.GenImageProto ([in] IObjectX Image,

Set the gray values of an image to a specified value.

GenImageProto sets the gray values of the image ImageCleared to the value Grayval. The input image is not modified. It is merely used to determine the size of the output image.
Parameter

- **Image**: (input iconic) Image, whose gray values are to be cleared.
- **ImageCleared**: (output iconic) Image with constant gray value.
- **Grayval**: (input control) Gray value to be used for the output image.
  
  **Default Value**: 0
  
  **Value Suggestions**: Grayval ∈ {0, 1, 2, 5, 10, 16, 32, 64, 128, 253, 254, 255}

**Result**

GenImageProto returns TRUE if all parameters are correct. If necessary, an exception is raised.

**Parallelization Information**

GenImageProto is **reentrant** and processed without parallelization.

**Possible Predecessor Functions**

TestObjDef

**See Also**

GetImagePointer1

**Alternatives**

SetGrayval, PaintGray, GenImageConst, CopyImage

**Module**

Basic operators

```c
void HImageX.GenImageSurfaceSecondOrder
    ([in] String Type,
     [in] long Width, [in] long Height )
```

```c
void HOperatorSetX.GenImageSurfaceSecondOrder
    ([out] HUntypedObjectX ImageSurface, [in] String Type, [in] double Alpha,
     [in] long Height )
```

Create a curved gray surface with second order polynomial.

The operator GenImageSurfaceSecondOrder creates a curved gray value surface according to the following equation:

\[
\text{ImageSurface}(r, c) = \text{Alpha}(r - \text{Row})^2 + \text{Beta}(c - \text{Col})^2 + \text{Gamma}(r - \text{Row})(c - \text{Col}) + \text{Delta}(r - \text{Row}) + \text{Delta}(c - \text{Col}) + \text{Zeta}
\]

The size of the image is determined by Width and Height. The gray values are of the type Type. Gray values outside the valid area are clipped.

Parameter

- **ImageSurface**: (output iconic) Created image with new image matrix.
- **Type**: (input control) Pixel type.
  
  **Default Value**: 'byte'
  
  **Value List**: Type ∈ {'byte', 'real'}

- **Alpha**: (input control) Second order coefficient in line direction.
  
  **Default Value**: 1.0
  
  **Value Suggestions**: Alpha ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
  
  **Minimal Value Step**: 0.000001
  
  **Recommended Value Step**: 0.005
5.2. CREATION

- **Beta** (input control) .......................... number \(\sim\) double
  Second order coefficient in column direction.
  Default Value : 1.0
  Value Suggestions : \(\text{Beta} \in \{-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0\}\)
  Minimal Value Step : 0.000001
  Recommended Value Step : -0.005

- **Gamma** (input control) .......................... number \(\sim\) double
  Mixed second order coefficient.
  Default Value : 1.0
  Value Suggestions : \(\text{Gamma} \in \{-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0\}\)
  Minimal Value Step : 0.000001
  Recommended Value Step : -0.005

- **Delta** (input control) .......................... number \(\sim\) double
  First order coefficient in line direction.
  Default Value : 1.0
  Value Suggestions : \(\text{Delta} \in \{-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0\}\)
  Minimal Value Step : 0.000001
  Recommended Value Step : -0.005

- **Epsilon** (input control) .......................... number \(\sim\) double
  First order coefficient in column direction.
  Default Value : 1.0
  Value Suggestions : \(\text{Epsilon} \in \{-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0\}\)
  Minimal Value Step : 0.000001
  Recommended Value Step : -0.005

- **Zeta** (input control) .......................... number \(\sim\) double
  Zero order coefficient.
  Default Value : 1.0
  Value Suggestions : \(\text{Zeta} \in \{-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0\}\)
  Minimal Value Step : 0.000001
  Recommended Value Step : -0.005

- **Row** (input control) .......................... number \(\sim\) double
  Line coordinate of the apex of the surface.
  Default Value : 256.0
  Value Suggestions : \(\text{Row} \in \{0.0, 128.0, 256.0, 512.0\}\)
  Minimal Value Step : 0.000001
  Recommended Value Step : -0.005

- **Col** (input control) .......................... number \(\sim\) double
  Column coordinate of the apex of the surface.
  Default Value : 256.0
  Value Suggestions : \(\text{Col} \in \{0.0, 128.0, 256.0, 512.0\}\)
  Minimal Value Step : 0.000001
  Recommended Value Step : -0.005

- **Width** (input control) .......................... extent.x \(\sim\) long
  Width of image.
  Default Value : 512
  Value Suggestions : \(\text{Width} \in \{128, 256, 512, 1024\}\)
  Typical Range of Values : \(1 \leq \text{Width} \leq 1\text{lin}\)
  Minimal Value Step : 1
  Recommended Value Step : 10
  Restriction : \(\text{Width} \geq 1\)

- **Height** (input control) .......................... extent.y \(\sim\) long
  Height of image.
  Default Value : 512
  Value Suggestions : \(\text{Height} \in \{128, 256, 512, 1024\}\)
  Typical Range of Values : \(1 \leq \text{Height} \leq 1\text{lin}\)
  Minimal Value Step : 1
  Recommended Value Step : 10
  Restriction : \(\text{Height} \geq 1\)
If the parameter values are correct \texttt{GenImageSurfaceSecondOrder} returns the value \texttt{TRUE}. Otherwise an exception handling is raised.

\textbf{Parallelization Information}

\texttt{GenImageSurfaceSecondOrder} is \textit{reentrant}, \textit{local}, and processed \textit{without} parallelization.

\textbf{See Also}

\texttt{GenImageGrayRamp}

\textbf{Module}

Image / region / XLD management

---

### Parameter

- \textbf{Image} (input iconic) \textit{image} \sim \texttt{HImageX} / \texttt{IHObjectX} 
  
  \texttt{Input image.}

- \textbf{MSecond} (output control) \textit{integer} \sim \texttt{long} 
  
  \texttt{Milliseconds (0..999).}

- \textbf{Second} (output control) \textit{integer} \sim \texttt{long} 
  
  \texttt{Seconds (0..59).}

- \textbf{Minute} (output control) \textit{integer} \sim \texttt{long} 
  
  \texttt{Minutes (0..59).}

- \textbf{Hour} (output control) \textit{integer} \sim \texttt{long} 
  
  \texttt{Hours (0..11).}

- \textbf{Day} (output control) \textit{integer} \sim \texttt{long} 
  
  \texttt{Day of the month (1..31).}

- \textbf{YDay} (output control) \textit{integer} \sim \texttt{long} 
  
  \texttt{Day of the year (1..365).}

- \textbf{Month} (output control) \textit{integer} \sim \texttt{long} 
  
  \texttt{Month (1..12).}

- \textbf{Year} (output control) \textit{integer} \sim \texttt{long} 
  
  \texttt{Year (xxxx).}

### Result

The operator \texttt{GetImageTime} returns the value \texttt{TRUE} if exactly one image was passed. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem(‘noObjectResult’,	extless Result	extgreater)}. If necessary an exception handling is raised.

\textbf{Parallelization Information}

\texttt{GetImageTime} is \textit{reentrant}, \textit{local}, and processed \textit{without} parallelization.

\textbf{Possible Predecessor Functions}

\texttt{ReadImage}, \texttt{GrabImage}

\textbf{See Also}

\texttt{CountSeconds}

\textbf{Module}

Image / region / XLD management
5.2. CREATION

应付图

Convert a region into a binary byte-image.
RegionToBin converts the input region given in Region into a byte-image and assigns a gray value of ForegroundGray to all pixels in the region. If the input region is larger than the generated image, it is clipped at the image borders. The background is set to BackgroundGray.

Parameter

- **Region** (input iconic) region(-array) ~ HRegionX / IHObjectX
  - Regions to be converted.
- **BinImage** (output iconic) image ~ HImageX / HUntypedObjectX (byte)
  - Result image of dimension Width x Height containing the converted regions.
- **ForegroundGray** (input control) integer ~ long
  - Gray value in which the regions are displayed.
    - **Default Value**: 255
    - **Value Suggestions**: ForegroundGray ∈ {0, 1, 50, 100, 128, 150, 200, 254, 255}
    - **Typical Range of Values**: 0 ≤ ForegroundGray ≤ 0(lin)
    - **Recommended Value Step**: 1
- **BackgroundGray** (input control) integer ~ long
  - Gray value in which the background is displayed.
    - **Default Value**: 0
    - **Value Suggestions**: BackgroundGray ∈ {0, 1, 50, 100, 128, 150, 200, 254, 255}
    - **Typical Range of Values**: 0 ≤ BackgroundGray ≤ 0(lin)
    - **Recommended Value Step**: 1
- **Width** (input control) extent.x ~ long
  - Width of the image to be generated.
    - **Default Value**: 512
    - **Value Suggestions**: Width ∈ {256, 512, 1024}
    - **Typical Range of Values**: 1 ≤ Width ≤ 1(lin)
    - **Minimal Value Step**: 1
    - **Recommended Value Step**: 16
    - **Restriction**: (Width ≥ 1)
- **Height** (input control) extent.y ~ long
  - Height of the image to be generated.
    - **Default Value**: 512
    - **Value Suggestions**: Height ∈ {256, 512, 1024}
    - **Typical Range of Values**: 1 ≤ Height ≤ 1(lin)
    - **Minimal Value Step**: 1
    - **Recommended Value Step**: 16
    - **Restriction**: (Height ≥ 1)

Result

RegionToBin always returns TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem(‘noObjectResult’,<Result>) and the behavior in case of an empty input region via SetSystem(‘emptyRegionResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

RegionToBin is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Connection, Regiongrowing, Pouring

Possible Successor Functions

GetGrayval

See Also

GenImageProto, PaintGray

HALCON 6.0
Convert regions to a label image.

RegionToLabel converts the input regions into a label image according to their index (1..n), i.e., the first region is painted with the gray value 1, the second the gray value 2, etc. Only positive gray values are used. For byte-images the index is entered modulo 256.

Regions larger than the generated image are clipped appropriately. If regions overlap the regions with the higher image are entered (i.e., they are painted in the order in which they are contained in the input regions). If so desired, the regions can be made non-overlapping by calling ExpandRegion.

The background, i.e., the area not covered by any regions, is set to 0. This can be used to test in which image range no region is present.

**Parameter**

- Region (input iconic) .......................... region(-array) ~ HRegionX / IHOBJECTX Regions to be converted.
- ImageLabel (output iconic) .................... image ~ HImageX / HUntypedObjectX (byte, int2, int4) Result image of dimension Width x Height containing the converted regions.
- Type (input control) .......................... string ~ String Pixel type of the result image.
  - Default Value: 'int2'
  - Value List: Type ∈ {'byte', 'int2', 'int4'}
- Width (input control) .......................... extent.y ~ long Width of the image to be generated.
  - Default Value: 512
  - Value Suggestions: Width ∈ {64, 128, 256, 512, 1024}
  - Typical Range of Values: 1 ≤ Width ≤ 1(lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 16
  - Restriction: (Width ≥ 1)
- Height (input control) .......................... extent.x ~ long Height of the image to be generated.
  - Default Value: 512
  - Value Suggestions: Height ∈ {64, 128, 256, 512, 1024}
  - Typical Range of Values: 1 ≤ Height ≤ 1(lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 16
  - Restriction: (Height ≥ 1)

**Result**

RegionToLabel always returns TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

**Parallelization Information**

RegionToLabel is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection, ExpandRegion
5.2. CREATION

Possible Successor Functions

GetGrayval, GetImagePointer1

See Also

LabelToRegion

Alternatives

RegionToBin, PaintRegion

Module

Image / region / XLD management

```plaintext
[out] HImageX ImageMean HRegionX.RegionToMean ([in] HImageX Image )
void HOperatorSetX.RegionToMean ([in] IHObjectX Regions, [in] IHObjectX Image, [out] HUntypedObjectX ImageMean )
```

Paint regions with their average gray value.

RegionToMean returns an image in which the regions Regions are painted with their average gray value based on the image Image. This operator is mainly intended to visualize segmentation results.

Parameter

- **Regions** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . region(-array) \(\sim\) HRegionX / IHObjectX
  Input regions.
- **Image** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image \(\sim\) HImageX / IHObjectX (byte)
  Original gray-value image.
- **ImageMean** (output iconic) . . . . . . . . . . . . . . . . . . . . . . image \(\sim\) HImageX / HUntypedObjectX (byte)
  Result image with painted regions.

Example

```plaintext
read_image(Image,'fabrik')
region_growing(Image,Regions,3,3,6,100)
region_to_mean(Regions,Image,Disp)
disp_image(Disp,WindowHandle)
set_draw(WindowsHandle,'margin')
set_color(WindowsHandle,'black')
disp_region(Regions,WindowHandle).
```

Result

RegionToMean returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception is raised.

Parallelization Information

RegionToMean is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessor Functions

Regiongrowing, Connection

Possible Successor Functions

DispImage

Alternatives

PaintRegion, Intensity

Module

Image filters

HALCON 6.0
5.3 Domain

```c
[out] HImageX GrayRegions HRegionX.AddChannels ([in] HImageX Image )
```
```c
void HOperatorSetX.AddChannels ([in] IHObjectX Regions,
```

Add gray values to regions.

The operator `AddChannels` adds the gray values from `Image` to the regions in `Regions`. All channels of `Image` are adopted. The definition domain is calculated as the average of the definition domain of the image with the region. Thus the new definition domain can be a subset of the input region. The size of the matrix is not changed.

Parameter

- **Regions** (input iconic) region(-array)  
  Input regions (without gray values).
- **Image** (input iconic) (multichannel-)image  
  Gray image for regions.
- **GrayRegions** (output iconic) image(-array)  
  Regions with gray values (also gray images).

Parameter Number : `Regions = GrayRegions`

Parallelization Information

- `AddChannels` is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

- `Threshold`, `Regiongrowing`, `GenCircle`, `DrawRegion`

Possible Successor Functions

- `Threshold`, `Regiongrowing`, `GetDomain`

See Also

- `FullDomain`, `GetDomain`, `Intersection`

Alternatives

- `ChangeDomain`, `ReduceDomain`

Module

- Image / region / XLD management

```c
[out] HImageX ImageNew HImageX.ChangeDomain ([in] HRegionX NewDomain )
```
```c
void HOperatorSetX.ChangeDomain ([in] IHObjectX Image,
```

Change definition domain of an image.

The operator `ChangeDomain` uses the indicated region as new definition domain. Unlike the operator `ReduceDomain` it does not form the intersection of the previous definition domain. This can lead to errors particularly when the region is larger than the image matrix. The size of the matrix is not changed.

Attention

Due to running time the transferred region is not checked for consistency (i.e., whether it fits with the image matrix). Incorrect regions lead to system hang-ups during subsequent operations.

Parameter

- **Image** (input iconic) image(-array)  
  Input image.
- **NewDomain** (input iconic) region(-array)  
  New definition domain.
- **ImageNew** (output iconic) image(-array)  
  Image with new definition domain.
5.3. DOMAIN

Parallelization Information
ChangeDomain is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
GetDomain

See Also
FullDomain, GetDomain, Intersection

Alternatives
ReduceDomain

Module
Image / region / XLD management

function

[out] HImageX ImageFull HImageX.FullDomain ( )

void HOperatorSetX.FullDomain ([in] IHObjectX Image,
[out] HUntypedObjectX ImageFull )

Expand the domain of an image to maximum.
The operator FullDomain enters a rectangle with the edge length of the image as new definition domain. This means that all pixels of the matrix are included in further operations. Thus the same definition domain is obtained as by reading or generating an image. The size of the matrix is not changed.

Parameter

▷ Image (input iconic) .............................................. image(-array) ~ HImageX / IHObjectX
Input image.
▷ ImageFull (output iconic) ................................. image(-array) ~ HImageX / HUntypedObjectX
Image with maximum definition domain.

Parallelization Information
FullDomain is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
GetDomain

See Also
GetDomain, GenRectangle1

Alternatives
ChangeDomain, ReduceDomain

Module
Image / region / XLD management

function

[out] HRegionX Domain HImageX.GetDomain ( )

void HOperatorSetX.GetDomain ([in] IHObjectX Image,
[out] HUntypedObjectX Domain )

Get the domain of an image.
The operator GetDomain returns the definition domains of all input images as a region.

Parameter

▷ Image (input iconic) .............................................. image(-array) ~ HImageX / IHObjectX
Input images.
▷ Domain (output iconic) ................................. region(-array) ~ HRegionX / HUntypedObjectX
Definition domains of input images.

Parallelization Information
GetDomain is reentrant and automatically parallelized (on tuple level).
Reduce the domain of an image to a rectangle.
The operator \texttt{Rectangle1Domain} reduces the definition domain of the given image to the specified rectangle. The old domain of the input image is ignored. The size of the matrix is not changed.

\begin{itemize}
\item \textbf{Image} (input iconic) \hspace{1cm} \textbf{ImageReduced} (output iconic)
\item \textbf{Row1} (input control)
\item \textbf{Column1} (input control)
\item \textbf{Row2} (input control)
\item \textbf{Column2} (input control)
\end{itemize}

\begin{itemize}
\item Default Value: \texttt{100}
\item Value Suggestions: \texttt{Row1 \in \{10, 20, 50, 100, 200, 300, 500\}}
\item Typical Range of Values: \texttt{0 \leq Row1 \leq 0}
\item Default Value: \texttt{100}
\item Value Suggestions: \texttt{Column1 \in \{10, 20, 50, 100, 200, 300, 500\}}
\item Typical Range of Values: \texttt{0 \leq Column1 \leq 0}
\item Default Value: \texttt{200}
\item Value Suggestions: \texttt{Row2 \in \{10, 20, 50, 100, 200, 300, 500\}}
\item Typical Range of Values: \texttt{0 \leq Row2 \leq 0}
\item Default Value: \texttt{200}
\item Value Suggestions: \texttt{Column2 \in \{10, 20, 50, 100, 200, 300, 500\}}
\item Typical Range of Values: \texttt{0 \leq Column2 \leq 0}
\end{itemize}

\textbf{Parallelization Information}

\texttt{Rectangle1Domain} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

\textbf{Possible Predecessor Functions}

\begin{itemize}
\item \texttt{GetDomain}
\end{itemize}

\textbf{See Also}

\begin{itemize}
\item \texttt{GetDomain, ChangeDomain, ReduceDomain, FullDomain}
\end{itemize}

\textbf{Module}

\textit{Image / region / XLD management}
5.4 Features

Reduce the domain of an image.

The operator ReduceDomain reduces the definition domain of the given image to the indicated region. The new definition domain is calculated as the intersection of the old definition domain with the region. Thus, the new definition domain can be a subset of the region. The size of the matrix is not changed.

Parameter

- **Image** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array)
- **Region** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . region
- **ImageReduced** (output iconic) ........................ image(-array)

Parallelization Information

ReduceDomain is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

- GetDomain

See Also

- FullDomain, GetDomain, Intersection

Alternatives

- ChangeDomain, Rectangle1Domain, AddChannels

Module

Image / region / XLD management

5.4 Features

Compute the area and center of gravity of a region in a gray value image.

AreaCenterGray computes the area and center of gravity of the regions Regions that have gray values which are defined by the image Image. This operator is similar to AreaCenter, but in contrast to that operator, the gray values of the image are taken into account while computing the area and center of gravity.

The area $A$ of a region $R$ in the image with the gray values $g(r, c)$ is defined as

$$A = \sum_{(r,c) \in R} g(r, c).$$

This means that the area is defined by the volume of the gray value function $g(r, c)$. The center of gravity is defined by the first two normalized moments of the gray values $g(r, c)$, i.e., by $(m_{1,0}, m_{0,1})$, where

$$m_{p,q} = \frac{1}{A} \sum_{(r,c) \in R} r^p c^q g(r, c).$$
Parameter

- **Regions** (input iconic)  
  Region(s) to be examined.
- **Image** (input iconic)  
  Gray value image.
- **Area** (output control)  
  Gray value volume of the region.
- **Row** (output control)  
  Row coordinate of the gray value center of gravity.
- **Column** (output control)  
  Column coordinate of the gray value center of gravity.

Result

AreaCenterGray returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behavior can be set via `SetSystem(::'noObjectResult',<Result>):`. If necessary, an exception handling is raised.

Parallelization Information

AreaCenterGray is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Regiongrowing, Connection

See Also

AreaCenterXld, EllipticAxisGray

Alternatives

AreaCenter

Module

Image filters

```plaintext
```

Calculate a co-occurrence matrix and derive gray value features thereof.

The call of CoocFeatureImage corresponds to the consecutive execution of the operators GenCoocMatrix and CoocFeatureMatrix. If several direction matrices of the co-occurrence matrix are to be evaluated consecutively, it is more efficient to generate the matrix via GenCoocMatrix and then call the operator CoocFeatureMatrix for the resulting matrix. The parameter `Direction` transfers the direction of the neighborhood in angle or 'mean'. In the case of 'mean' the mean value is calculated in all four directions.
The operator CoocFeatureImage returns the value TRUE if an image with defined gray values (byte) is entered and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator SetSystem(‘noObjectResult’, <Result>:), the behavior in case of empty region is set via SetSystem(‘emptyRegionResult’, <Result>:). If necessary an exception handling is raised.

--- Parallelization Information ---
CoocFeatureImage is reentrant and automatically parallelized (on tuple level).

--- Possible Predecessor Functions ---
GenCoocMatrix

--- See Also ---
Intensity, MinMaxGray, EntropyGray, SelectGray

--- Alternatives ---
CoocFeatureMatrix

--- Module ---
Image filters

Calculate gray value features from a co-occurrence matrix.

The procedure calculates from a co-occurrence matrix (CoocMatrix) the energy (Energy), correlation (Correlation), local homogeneity (Homogenity) and contrast (Contrast).

The operator CoocFeatureMatrix calculates the gray value features from the part of the input matrix generated by GenCoocMatrix corresponding to the direction matrix indicated by the parameters LeftGray and Direction according to the following formulae:

Energy:

\[
Energy = \sum_{i,j=0}^{width} c_{ij}^2
\]

(Measure for image homogeneity)

Correlation:

\[
Correlation = \sum_{i,j=0}^{width} \frac{(i - u_x)(j - u_y)c_{ij}}{\sqrt{x} \sqrt{y}}
\]

(Measure for gray value dependencies)

Local homogeneity:

\[
Homogenity = \sum_{i,j=0}^{width} \frac{1}{1 + (i - j)^2} c_{ij}
\]

Contrast:

\[
Contrast = \sum_{i,j=0}^{width} (i - j)^2 c_{ij}
\]
(Measure for the size of the intensity differences)

where

\[ width = \text{Width of CoocMatrix} \]
\[ c_{ij} = \text{Entry of co-occurrence matrix} \]
\[ u_x = \sum_{i,j=0}^{width} i \cdot c_{ij} \]
\[ u_y = \sum_{i,j=0}^{width} j \cdot c_{ij} \]
\[ s_x^2 = \sum_{i,j=0}^{width} (i - u_x)^2 \cdot c_{ij} \]
\[ s_y^2 = \sum_{i,j=0}^{width} (i - u_y)^2 \cdot c_{ij} \]

The region of the input image is disregarded.

**Parameter**

- **CoocMatrix** (input iconic) ................................. image \( \rightsquigarrow \) HImageX / IHObjectX (int4)
  Co-occurrence matrix.

- **Energy** (output control) .................................. real \( \rightsquigarrow \) double
  Homogeneity of the gray values.

- **Correlation** (output control) ............................. real \( \rightsquigarrow \) double
  Correlation of gray values.

- **Homogeneity** (output control) ........................... real \( \rightsquigarrow \) double
  Local homogeneity of gray values.

- **Contrast** (output control) ................................. real \( \rightsquigarrow \) double
  Gray value contrast.

**Result**

The operator CoocFeatureMatrix returns the value TRUE if an image with defined gray values is passed and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator SetSystem(::’noObjectResult’,<Result>:). If necessary an exception handling is raised.

**Parallelization Information**

CoocFeatureMatrix is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

GenCoocMatrix

**See Also**

Intensity, MinMaxGray, EntropyGray, SelectGray

**Alternatives**

CoocFeatureImage

**Module**

Image filters

```java
[out] VARIANT Ra HRegionX.EllipticAxisGray ([in] HImageX Image,
[out] VARIANT Rb, [out] VARIANT Phi )
```

```java
void HOperatorSetX.EllipticAxisGray ([in] IHObjectX Regions,
```

Compute the orientation and major axes of a region in a gray value image.

The operator EllipticAxisGray calculates the length of the axes and the orientation of the ellipse having the “same orientation” and the “aspect ratio” as the input region. Several input regions can be passed in Regions as tuples. The length of the major axis Ra and the minor axis Rb as well as the orientation of the major axis with regard to the x-axis (Phi) are determined. The angle is returned in radians. The calculation is done analogously to EllipticAxis. The only difference is that in EllipticAxisGray the gray value moments are used instead of the region moments. The gray value moments are derived from the input image Image. For the definition of the gray value moments, see AreaCenterGray.
5.4. FEATURES

Parameter

- **Regions** (input iconic) .......................... region(-array) $\sim$ HRegionX / IObjectX
  Region(s) to be examined.
- **Image** (input iconic) ............. image $\sim$ HImageX / IObjectX (byte, cyclic, direction, int1, int2, int4, real)
  Gray value image.
- **Ra** (output control) .......................... real(-array) $\sim$ VARIANT (real)
  Major axis of the region.
- **Rb** (output control) .......................... real(-array) $\sim$ VARIANT (real)
  Minor axis of the region.
- **Phi** (output control) ...................... angle.rad(-array) $\sim$ VARIANT (real)
  Angle enclosed by the major axis and the x-axis.

Result

EllipticAxisGray returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behavior can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

EllipticAxisGray is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

- Threshold, RegionGrowing, Connection

Possible Successor Functions

- GenEllipse

See Also

Alternatives

EllipticAxis

Module

Image filters


Determine the entropy and anisotropy of images.

The operator EntropyGray creates the histogram of relative frequencies of the gray values in the input image and calculates from these frequencies the entropy and the anisotropy coefficient for each region from Regions according to the following formulae:

**Entropy:**

$$Entrop_{y} = - \sum_{0}^{255} rel[i] \times \log_{2}(rel[i])$$

**Anisotropy coefficient:**

$$Anisotropy = \frac{\sum_{0}^{k} rel[i] \times \log_{2}(rel[i])}{Entrop_{y}}$$

where

- $rel[i] = \text{Histogram of relative gray value frequencies}$
- $i = \text{Gray value of input image (0..255)}$
- $k = \text{Smallest possible gray value with } \sum_{0}^{k} rel[i] \geq 0.5$
Parameter

- **Regions** (input iconic) .................................................. region(-array) \( \sim \) HRegionX / IObjectX
  Regions where the features are to be determined.

- **Image** (input iconic) .................................................. image \( \sim \) HImageX / IObjectX (byte)
  Gray value image.

- **Entropy** (output control) ........................................ real(-array) \( \sim \) VARIANT (real)
  Information content (entropy) of the gray values.
  **Restriction:** \((0 \leq \text{Entropy}) \land (\text{Entropy} \leq 8)\)

- **Anisotropy** (output control) ..................................... real(-array) \( \sim \) VARIANT (real)
  Measure of the symmetry of gray value distribution.

**Result**

The operator EntropyGray returns the value TRUE if an image with defined gray values is entered and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator SetSystem(:'noObjectResult',<Result>), the behavior in case of empty region is set via SetSystem(:'emptyRegionResult',<Result>). If necessary an exception handling is raised.

**Parallelization Information**

EntropyGray is reentrant and automatically parallelized (on tuple level).

**See Also**

EntropyImage, GrayHisto, FuzzyEntropy, FuzzyPerimeter

**Alternatives**

SelectGray

**Module**

Image filters

```plaintext
```

```
```

Calculate gray value moments and approximation by a first order surface (plane).

The operator FitSurfaceFirstOrder calculates the gray value moments and the parameters of the approximation of the gray values by a first order surface. The calculation is done by minimizing the distance between the gray values and the surface. A first order surface is described by the following formula:

\[
\text{Image}(r,c) = \text{Alpha}(r - r_{center}) + \text{Beta}(c - c_{center}) + \text{Gamma}
\]

\(r_{center}\) and \(c_{center}\) are the coordinates of the center of the input region. By the minimization process the parameters from Alpha to Gamma is calculated.

The algorithm used for the fitting can be selected via Algorithm:

- **regression**  Standard 'least squares' line fitting.

- **huber**  Weighted 'least squares' fitting, where the impact of outliers is decreased based on the approach of Huber.

- **tukey**  Weighted 'least squares' fitting, where the impact of outliers is decreased based on the approach of Tukey.

The parameter ClippingFactor (a scaling factor for the standard deviation) controls the amount of damping outliers: The smaller the value chosen for ClippingFactor the more outliers are detected. The detection of outliers is repeated. The parameter Iterations specifies the number of iterations. In the modus 'regression' this value is ignored.
The operator \( \text{FitSurfaceSecondOrder} \) calculates the gray value moments and the parameters of the approximation of the gray values by a second order surface. The calculation is done by minimizing the distance between the gray values and the surface. A second order surface is described by the following formula:

\[
\text{Image}(r,c) = \text{Alpha}(r-r_{\text{center}})^2 + \text{Beta}(c-c_{\text{center}})^2 + \text{Gamma}(r-r_{\text{center}})(c-c_{\text{center}}) + \text{Delta}(r-r_{\text{center}}) + \text{Epsilon} + \text{Zeta}
\]

\( r_{\text{center}} \) and \( c_{\text{center}} \) are the coordinates of the center of the input region. By the minimization process the parameters from Alpha to Zeta is calculated.

The algorithm used for the fitting can be selected via Algorithm:

\[\text{Algorithm} \in \{\text{regression}, \text{huber}, \text{tukey}\}\]

The fit can be performed parallelized (on tuple level).
'regression' Standard 'least squares' fitting.
'huber' Weighted 'least squares' fitting, where the impact of outliers is decreased based on the approach of Huber.
'tukey' Weighted 'least squares' fitting, where the impact of outliers is decreased based on the approach of Tukey.

The parameter ClippingFactor (a scaling factor for the standard deviation) controls the amount of damping outliers: The smaller the value chosen for ClippingFactor the more outliers are detected. The detection of outliers is repeated. The parameter Iterations specifies the number of iterations. In the modus 'regression' this value is ignored.
Determine the fuzzy entropy of regions.

FuzzyEntropy calculates the fuzzy entropy of a fuzzy set. To do so, the image is regarded as a fuzzy set. The entropy then is a measure of how well the image approximates a white or black image. It is defined as follows:

\[
H(X) = \frac{1}{MN} \sum_{i} T_e(l) h(l)
\]

where \( M \times N \) is the size of the image, and \( h(l) \) is the histogram of the image. Furthermore,

\[
T_e(l) = -\mu(l) \ln \mu(l) - (1 - \mu(l)) \ln (1 - \mu(l))
\]

Here, \( u(x(m,n)) \) is a fuzzy membership function defining the fuzzy set (see FuzzyPerimeter). The same restrictions hold as in FuzzyPerimeter.

---

**Parameter**

- **Regions** (input iconic) ... region(-array) ~ HRegionX / IObjectX
  
  Regions for which the fuzzy entropy is to be calculated.

- **Image** (input iconic) ... image ~ HImageX / IObjectX (byte)
  
  Input image containing the fuzzy membership values.

- **Apar** (input control) ... integer ~ long
  
  Start of the fuzzy function.

  **Default Value**: 0
  
  **Value Suggestions**: Apar \( \in \{0, 5, 10, 20, 50, 100\} \)
  
  **Typical Range of Values**: \( 0 \leq Apar \leq 0(lin) \)
  
  **Minimal Value Step**: 1
  
  **Recommended Value Step**: 5

- **Cpar** (input control) ... integer ~ long
  
  End of the fuzzy function.

  **Default Value**: 255
  
  **Value Suggestions**: Cpar \( \in \{50, 100, 150, 200, 220, 255\} \)
  
  **Typical Range of Values**: \( 0 \leq Cpar \leq 0(lin) \)
  
  **Minimal Value Step**: 1
  
  **Recommended Value Step**: 5
  
  **Restriction**: \( Apar \leq Cpar \)

- **Entropy** (output control) ... real(-array) ~ VARIANT (real)
  
  Fuzzy entropy of a region.

---

**Example**

```c
/* To find a Fuzzy Entropy from an Image */
read_image(Image,'affe')
fuzzy_entropy(Trans,Trans,0,255,Entro).
```

**Result**

The operator FuzzyEntropy returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

---

**Parallelization Information**

FuzzyEntropy is reentrant and automatically parallelized (on tuple level).

---

**See Also**

FuzzyPerimeter
The operator FuzzyPerimeter is used to determine the differences of fuzzy membership between an image point and its neighbor points. The right and lower neighbor are taken into account. The fuzzy perimeter is then defined as follows:

\[ p(X) = \sum_{m=1}^{M-1} \sum_{n=1}^{N-1} |\mu_X(x_{m,n}) - \mu_X(x_{m,n+1})| + \sum_{m=1}^{M-1} \sum_{n=1}^{N-1} |\mu_X(x_{m,n}) - \mu_X(x_{m+1,n})| \]

where \( M \times N \) is the size of the image, and \( u(x(m, n)) \) is the fuzzy membership function (i.e., the input image).

This implementation uses Zadeh’s Standard-S function, which is defined as follows:

\[
\mu_X(x) = \begin{cases} 
0, & x \leq a \\
2 \left( \frac{x-a}{c-a} \right)^2, & a < x \leq b \\
1 - 2 \left( \frac{x-a}{c-a} \right)^2, & b < x \leq c \\
1, & c \leq x
\end{cases}
\]

The parameters \( a, b, c \) obey the following restrictions: \( b = \frac{\Delta a + c}{2} \) is the inflection point of the function, \( \Delta b = b - a = c - b \) is the bandwidth, and for \( x = b \), \( \mu(x) = 0.5 \) holds. In FuzzyPerimeter, the parameters Apar and Cpar are defined as follows: \( b = \frac{a + c}{2} \).

**Regions** (input iconic) ................................................. region(-array) \( \sim \) HRegionX / IHObjectX

Regions for which the fuzzy perimeter is to be calculated.

**Image** (input iconic) ................................................. image \( \sim \) HImageX / IHObjectX ( byte )

Input image containing the fuzzy membership values.

**Apar** (input control) .................................................. integer \( \sim \) long

Start of the fuzzy function.

Default Value : 0

Value Suggestions : Apar \( \in \{0, 5, 10, 20, 50, 100\} \)

Typical Range of Values : 0 \( \leq \) Apar \( \leq 0\) (lin)

Minimal Value Step : 1

Recommended Value Step : 5

**Cpar** (input control) .................................................. integer \( \sim \) long

End of the fuzzy function.

Default Value : 255

Value Suggestions : Cpar \( \in \{50, 100, 150, 200, 220, 255\} \)

Typical Range of Values : 0 \( \leq \) Cpar \( \leq 0\) (lin)

Minimal Value Step : 1

Recommended Value Step : 5

Restriction : \( Apar \leq Cpar \)

**Perimeter** (output control) ........................................... real(-array) \( \sim \) VARIANT ( real )

Fuzzy perimeter of a region.
5.4. FEATURES

Example

/* To find a Fuzzy Entropy from an Image */
read_image(Image,’affe’)
fuzzy_perimeter(Trans,Trans,0,255,Per).

Result

The operator FuzzyPerimeter returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

Parallelization Information

FuzzyPerimeter is reentrant and automatically parallelized (on tuple level).

See Also

FuzzyEntropy

Bibliography


Module

Image filters

```
[out] HImageX Matrix HRegionX.GenCoocMatrix ([in] HImageX Image,
```

```
void HOperatorSetX.GenCoocMatrix ([in] IObjectX Regions,
[in] long Direction )
```

Calculate the co-occurrence matrix of a region in an image.

The operator GenCoocMatrix determines from the input regions how often the gray values $i$ and $j$ are located next to each other in a certain direction ($0, 45, 90, 135$ degrees), stores this number in the co-occurrence matrix at the locations $(i,j)$ and $(j,i)$ (the matrix is symmetrical), and finally scales the matrix with the number of entries. LdGray indicates the number of gray values to be distinguished (namely $2^{LdGray}$).

Example ($LdGray = 2$, i.e. 4 gray values are distinguished):

```
Input image Co-occurrence matrix
with gray values: (not scaled)

0 0 3 2 0 0 1 0 1 1 0
1 1 2 0 2 2 0 1 0 1 1
1 2 3 0 2 0 1 1 0 0
    1 0 1 0 0 1 0 0

0 2 0 0 0 1 0 0
2 2 1 0 1 2 0 1
0 1 0 2 0 0 2 0
0 0 2 0 0 1 0 0
```

Parameter

- **Regions** (input iconic) ......................................... region(-array) $\leadsto$ HRegionX / IObjectX
  Region to be checked.
- **Image** (input iconic) ............................................. image $\leadsto$ HImageX / IObjectX (byte)
  Image providing the gray values.
- **Matrix** (output iconic) ......................................... image(-array) $\leadsto$ HImageX / HUntypedObjectX (int4)
  Co-occurrence matrix (matrices).

HALCON 6.0
**LdGray** (input control) .......................................................... integer \( \sim \) long  
Number of gray values to be distinguished \( (2^{\text{LdGray}}) \).

**Default Value** : 6  
**Value List** : \( \text{LdGray} \in \{ 1, 2, 3, 4, 5, 6, 7, 8 \} \)  
**Typical Range of Values** : \( 1 \leq \text{LdGray} \leq 1 \) (lin)  
**Minimal Value Step** : 1  
**Recommended Value Step** : 1

**Direction** (input control) ....................................................... integer \( \sim \) long  
Direction of neighbor relation.  
**Default Value** : 0  
**Value List** : \( \text{Direction} \in \{ 0, 45, 90, 135 \} \)

The operator GenCoocMatrix returns the value TRUE if an image with defined gray values is entered and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator SetSystem (: : \text{noObjectResult'}, <Result> : ), the behavior in case of empty region is set via SetSystem (: : \text{emptyRegionResult'}, <Result> : ). If necessary an exception handling is raised.

**Parallelization Information**  
GenCoocMatrix is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**  
DrawRegion, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, Threshold, ErosionCircle, GaussImage, SmoothImage, SubImage

**See Also**  
CoocFeatureMatrix  
CoocFeatureImage  
Module  
Image filters

```plaintext
[out] <none>X AbsoluteHisto HRegionX.GrayHisto ([in] HImageX Image,  
[out] <none>X RelativeHisto )

void HOperatorSetX.GrayHisto ([in] IObjectX Regions,  
[in] IObjectX Image, [out] VARIANT AbsoluteHisto,  
[out] VARIANT RelativeHisto )
```

Calculate the gray value distribution.

The operator GrayHisto calculates for the image (Image) within Regions the absolute (AbsoluteHisto) and relative (RelativeHisto) histogram of the gray values. Both histograms are tupels of 256 values, which — beginning at 0 — contain the frequencies of the individual gray values of the image.

AbsoluteHisto indicates the absolute frequencies of the gray values in integers, and RelativeHisto indicates the relative, i.e. the absolute frequencies divided by the area of the image as floating point numbers.

real-, int2- and int4-images are transformed into byte-images (first the largest and smallest gray value in the image are determined, and then the original gray values are mapped linearly into the area 0..255) and then processed as mentioned above. The histogram can also be returned directly as a graphic via the operators SetPaint (: : WindowHandle,'histogram') and DispImage.

**Attention**  
Real, int2 and int4 images are reduced to 256 gray values.

**Parameter**  

**Regions** (input iconic) ..................................................... region(-array) \( \sim \) HRegionX / IObjectX  
Region in which the histogram is to be calculated.

**Image** (input iconic) ........... image \( \sim \) HImageX / IObjectX (byte, cyclic, direction, int1, int2, int4, real )  
Image the gray value distribution of which is to be calculated.
5.4. FEATURES

- **AbsoluteHisto** (output control) ................................. histogram \( \sim \) none \( X \)/VARIANT (integer)
  Absolute frequencies of the gray values.

- **RelativeHisto** (output control) ................................. histogram \( \sim \) none \( X \)/VARIANT (real)
  Frequencies, normalized to the area of the region.

The operator GrayHisto returns the value TRUE if the image has defined gray values and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator SetSystem(: :’noObjectResult’,<Result>:), the behavior in case of empty region is set via SetSystem(: :’emptyRegionResult’,<Result>:). If necessary an exception handling is raised.

Parallelization Information
GrayHisto is reentrant and processed without parallelization.

Possible Successor Functions
HistoToThresh, GenRegionHisto

See Also
SetPaint, DispImage, Histo2Dim, ScaleImageMax, EntropyGray

Alternatives
MinMaxGray, Intensity

Image filters

```
[out] VARIANT HorProjection HRegionX.GrayProjections

[out] VARIANT HorProjection HImageX.GrayProjections
([in] HRegionX Region, [in] String Mode, [out] VARIANT VertProjection )

void HOperatorSetX.GrayProjections ([in] IHObjectX Region,
[in] IHObjectX Image, [in] String Mode, [out] VARIANT HorProjection,
[out] VARIANT VertProjection )
```

Calculate horizontal and vertical gray-value projections.

GrayProjections calculates the horizontal and vertical gray-value projections.

Parameter

- **Region** (input iconic) ................................. region \( \sim \) HRegionX / IHObjectX
  Region to be processed.

- **Image** (input iconic) ................................. image \( \sim \) HImageX / IHObjectX
  Grayvalues for projections.

- **Mode** (input control) ................................. string \( \sim \) String
  Type of invariance.
  Default Value : 'simple'
  Value List : Mode \( \in \) { 'simple', 'rectangle', 'wave' }

- **HorProjection** (output control) ............................ real \( \sim \) VARIANT (real)
  Horizontal projections.

- **VertProjection** (output control) ............................ real \( \sim \) VARIANT (real)
  Vertical projections.

Parallelization Information
GrayProjections is reentrant and processed without parallelization.

Module

Image filters

HALCON 6.0
CHAPTER 5. IMAGE

void HOperatorSetX.Histo2Dim ([in] IHObjectX Regions,  
[in] IHObjectX ImageCol, [in] IHObjectX ImageRow,  
[out] HUntypedObjectX Histo2Dim)

Calculate the histogram of two-channel gray value images.

The operator Histo2dim calculates the 2-dimensional histogram of two images within Regions. The gray values of channel 1 (ImageCol) are interpreted as row index, those of channel 2 (ImageRow) as column index. The gray value at one point \( P(g_1, g_2) \) in the output image Histo2Dim indicates the frequency of the gray value combination \((g_1, g_2)\) with \( g_1 \) indicating the line index and \( g_2 \) the column index.

Parameter

- **Regions** (input iconic) ………………. region(-array) \( \sim \) HRegionX / IHObjectX Region in which the histogram is to be calculated.
- **ImageCol** (input iconic) …………. image \( \sim \) HImageX / IHObjectX (byte, direction, cyclic, int1) Channel 1.
- **ImageRow** (input iconic) ……………. image \( \sim \) HImageX / IHObjectX (byte) Channel 2.
- **Histo2Dim** (output iconic) ……………image \( \sim \) HImageX / HUntypedObjectX (int4) Histogram to be calculated.

Example

```plaintext
read_image(Image,’affe’)  
texture_laws(Image,Texture,’el’,1,5)  
draw_region(Region,WindowHandle)  
histo_2dim(Region,Texture,Image,Histo2Dim)  
disp_image(Histo2Dim,WindowHandle).
```

Result

The operator Histo2dim returns the value TRUE if both images have defined gray values. The behavior in case of empty input (no input images available) is set via the operator SetSystem(:: ‘noObjectResult’,<Result>:), the behavior in case of empty region is set via SetSystem(:: ‘emptyRegionResult’,<Result>:). If necessary an exception handling is raised.

Parallelization Information

Histo2Dim is reentrant and processed without parallelization.

Possible Predecessor Functions

Decompose3, Decompose2, DrawRegion

Possible Successor Functions

Threshold, Class2DimSup, Pouring, LocalMax, GraySkeleton

See Also

GetGrayval

Alternatives

GrayHisto

Module

Image filters

[out] VARIANT Mean HRegionX.Intensity ([in] HImageX Image,  
[out] VARIANT Deviation )

Calculate the mean and deviation of gray values.
The operator \texttt{Intensity} calculates the mean and the deviation of the gray values in the input image within \texttt{Regions}. If \( R \) is a region, \( p \) a pixel from \( R \) with the gray value \( g(p) \) and \( F \) the plane \( (F = |R|) \), the features are defined by:

\[
\begin{align*}
\text{Mean} & := \frac{\sum_{p \in R} g(p)}{F} \\
\text{Deviation} & := \sqrt{\frac{\sum_{p \in R} (g(p) - \text{Mean})^2}{F}}
\end{align*}
\]

\textbf{Attention}

The calculation of \texttt{Deviation} does not follow the usual definition if the region of the image contains only one pixel. In this case 0.0 is returned.

\textbf{Parameter}

- **Regions** (input iconic) ........................................... \texttt{region(-array)} \sim \texttt{HRegionX / IHObjectX} Regions the features of which are to be calculated.
- **Image** (input iconic) ........................................... \texttt{image} \sim \texttt{HImageX / IHObjectX ( byte, int1, int2, int4, real )} Gray value image.
- **Mean** (output control) ........................................... \texttt{real(-array)} \sim \texttt{VARIANT( real )} Mean gray value of a region.
- **Deviation** (output control) .................................... \texttt{real(-array)} \sim \texttt{VARIANT( real )} Deviation of gray values within a region.

The operator \texttt{Intensity} returns the value \texttt{TRUE}. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem(::‘noObjectResult’,<Result>:)}; the behavior in case of empty region is set via \texttt{SetSystem(::‘emptyRegionResult’,<Result>:)}. If necessary an exception handling is raised.

\textbf{Parallelization Information}

\texttt{Intensity} is \texttt{reentrant} and automatically \texttt{parallelized} (on tuple level).

\textbf{Possible Successor Functions}

- **Threshold**
- **MeanImage, MeanImage, GrayHisto**
- **SelectGray, MinMaxGray**

\textbf{Module}

\textbf{Image filters}

\begin{verbatim}
[out] VARIANT Min \texttt{HRegionX.MinMaxGray ([in] HImageX Image,}
[in] VARIANT Percent, [out] VARIANT Max, [out] VARIANT Range )

void \texttt{HOperatorSetX.MinMaxGray ([in] IHObjectX Regions,}
[in] IHObjectX Image, [in] VARIANT Percent, [out] VARIANT Min,
[out] VARIANT Max, [out] VARIANT Range )
\end{verbatim}

\textit{Determine the minimum and maximum gray values within regions.}

The operator \texttt{MinMaxGray} creates the histogram of the absolute frequencies of the gray values within \texttt{Regions} in the input image \texttt{Image} (see \texttt{GrayHisto}) and calculates the number of pixels \texttt{Percent} corresponding to the area of the input image. Then it goes inwards on both sides of the histogram by this number of pixels and determines the smallest and the largest gray value:

\textbf{e.g.}:

\begin{enumerate}
\item Area = 60, percent = 5, i.e. 3 pixels
\item histogram = [2,8,0,7,13,0,0,...,0,10,10,5,3,1,1]
\item \Rightarrow Maximum = 255, Minimum = 0, Range = 255
\end{enumerate}
MinMaxGray returns: Maximum = 253, Minimum = 1, Range = 252
If Percent is set at 50, Min = Max = Median. If Percent is 0 no histogram is calculated in order to enhance the runtime.

Attention
In case of int2, int4- and real-images Percent has to be 0.

Parameter

- **Regions** (input iconic) region(-array) ~ HRegionX / IHObjectX
  Regions, the features of which are to be calculated.

- **Image** (input iconic) image ~ HImageX / IHObjectX (byte, int1, int2, int4, real)
  Gray value image.

- **Percent** (input control) number ~ VARIANT (integer, real)
  Percentage below (above) the absolute maximum (minimum).
  **Default Value:** 0
  **Value Suggestions:** Percent ∈ {0, 1, 2, 5, 7, 10, 15, 20, 30, 40, 50}
  **Restriction:** (0 ≤ Percent) ∧ (Percent ≤ 50)

- **Min** (output control) real(-array) ~ VARIANT (real)
  “Minimum” gray value.

- **Max** (output control) real(-array) ~ VARIANT (real)
  “Maximum” gray value.
  **Restriction:** (Max ≥ Min)

- **Range** (output control) real(-array) ~ VARIANT (real)
  Difference between Max and Min.
  **Restriction:** (Range ≥ 0)

Example

```c
/* Threshold segmentation with training region: */
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
min_max_gray(Region,Image,5,Min,Max,)
threshold(Bild,Seg,Min,Max)
disp_region(Seg,WindowHandle).
```

Result

The operator MinMaxGray returns the value TRUE if the input image has the defined gray values and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator SetSystem(::‘noObjectResult’,<Result>:).

Parallelization Information

MinMaxGray is reentrant and processed without parallelization.

Possible Predecessor Functions

DrawRegion, GenCircle, GenEllipse, GenRectangle1, Threshold, Regiongrowing

Possible Successor Functions

Threshold

See Also

GrayHisto, ScaleImage, ScaleImageMax, LearnNdimNorm

Alternatives

SelectGray, Intensity

Module

Image filters
Calculate gray value moments and approximation by a plane.

The operator MomentsGrayPlane calculates the gray value moments and the parameters of the approximation of the gray values by a plane. The calculation is carried out according to the following formula:

\[
\begin{align*}
\text{MRow} &= \frac{1}{F^2} \sum_{(r,c) \in \text{Regions}} (r - \bar{r})(\text{Image}(r,c) - \text{Mean}) \\
\text{MCol} &= \frac{1}{F^2} \sum_{(r,c) \in \text{Regions}} (c - \bar{c})(\text{Image}(r,c) - \text{Mean}) \\
\text{Alpha} &= \frac{\text{MRow}m_{02} - m_{11}\text{MCol}}{F} - \frac{m_{20}m_{02} - m_{11}^2}{F} \\
\text{Beta} &= \frac{\text{MRow}m_{02} - m_{11}\text{MCol}}{m_{20}m_{02} - m_{11}^2}
\end{align*}
\]

where \( F \) is the plane, \( \bar{r}, \bar{c} \) the center, and \( m_{11}, m_{20}, \) and \( m_{02} \) the scaled moments of Regions.

The parameters \( \text{Alpha}, \text{Beta} \) describe a plane above the region:

\[\text{Image}(r,c) = \text{Alpha}(r - \bar{r}) + \text{Beta}(c - \bar{c}) + \text{Mean}\]

Thus \( \text{Alpha} \) indicates the gradient in the direction of the line axis (“down”), \( \text{Beta} \) the gradient in the direction of the column axis (to the “right”).

---

**Parameter**

- **Regions** (input iconic) ... region(-array) \( \rightarrow \) HRegionX / IObjectX Regions to be checked.
- **Image** (input iconic) ... image \( \rightarrow \) HImageX / IObjectX (byte, real) Corresponding gray values.
- **MRow** (output control) ... real(-array) \( \rightarrow \) VARIANT (real) Mixed moments along a line.
- **MCol** (output control) ... real(-array) \( \rightarrow \) VARIANT (real) Mixed moments along a column.
- **Alpha** (output control) ... real(-array) \( \rightarrow \) VARIANT (real) Parameter Alpha of the approximating plane.
- **Beta** (output control) ... real(-array) \( \rightarrow \) VARIANT (real) Parameter Beta of the approximating plane.
- **Mean** (output control) ... real(-array) \( \rightarrow \) VARIANT (real) Mean gray value.

---

**Result**

The operator MomentsGrayPlane returns the value TRUE if an image with the defined gray values (byte) is entered and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator SetSystem(’noObjectResult’,<Result>:), the behavior in case of empty region is set via SetSystem(’emptyRegionResult’,<Result>:). If necessary an exception handling is raised.

---

**Parallelization Information**

MomentsGrayPlane is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

DrawRegion, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, Threshold, RegionGrowing

---

**See Also**

Intensity, MomentsRegion2Nd

---

**Bibliography**


---

**Module**

Image filters
Calculate the deviation of the gray values from the approximating image plane.

The operator `PlaneDeviation` calculates the deviation of the gray values in Image from the approximation of the gray values through a plane. Contrary to the standard deviation in case of `Intensity` planes also receive the value zero. The gray value plane is calculated according to `GenImageGrayRamp`.

If \( P \) is the plane, \( \alpha, \beta, \mu \) the parameters of the image plane and \( (r', c') \) the center, \( \text{Deviation} \) is defined by:

\[
\text{Deviation} = \frac{\sum_{(r,c) \in \text{Regions}} \left( (\alpha(r - r') + \beta(c - c') + \mu - \text{Image}(r,c)) \right)^2}{F}
\]

Attention

It should be noted that the calculation of \( \text{Deviation} \) does not follow the usual definition. It is defined to return the value 0.0 for an image with only one pixel.

Parameter

\( \textbf{Regions} \) (input iconic) ........................................... region(-array) \( \rightarrow \) `HRegionX` / `IHObjectX`
Regions, of which the plane deviation is to be calculated.

\( \textbf{Image} \) (input iconic) ............................................. image \( \rightarrow \) `HImageX` / `IHObjectX (byte)`
Gray value image.

\( \textbf{Deviation} \) (output control) ......................................... real(-array) \( \rightarrow \) `VARIANT` (real)
Deviation of the gray values within a region.

Result

The operator `PlaneDeviation` returns the value TRUE if Image is of the type `byte`. The behavior in case of empty input (no input images available) is set via the operator `SetSystem`::

- `noObjectResult',<Result>:`
- `emptyRegionResult',<Result>:`
If necessary an exception handling is raised.

Parallelization Information

`PlaneDeviation` is reentrant and automatically parallelized (on tuple level).

See Also

`MomentsGrayPlane`

Alternatives

`Intensity`, `GenImageGrayRamp`, `SubImage`

Module

Image filters
Possible values for Features:

- **'area'**
  - Gray value volume of region (see AreaCenterGray)
- **'row'**
  - Row index of the center of gravity (see AreaCenterGray)
- **'column'**
  - Column index of the center of gravity (see AreaCenterGray)
- **'ra'**
  - Major axis of equivalent ellipse (see EllipticAxisGray)
- **'rb'**
  - Minor axis of equivalent ellipse (see EllipticAxisGray)
- **'phi'**
  - Orientation of equivalent ellipse (see EllipticAxisGray)
- **'min'**
  - Minimum gray value (see MinMaxGray)
- **'max'**
  - Maximum gray value (see MinMaxGray)
- **'mean'**
  - Mean gray value (see Intensity)
- **'deviation'**
  - Deviation of gray values (see Intensity)
- **'plane_deviation'**
  - Deviation from the approximating plane (see PlaneDeviation)
- **'anisotropy'**
  - Anisotropy (see EntropyGray)
- **'entropy'**
  - Entropy (see EntropyGray)
- **'fuzzy_entropy'**
  - Fuzzy entropy of region (see FuzzyEntropy, with a fuzzy function from Apar=0 to Cpar=255)
- **'fuzzy_perimeter'**
  - Fuzzy perimeter of region (see FuzzyPerimeter, with a fuzzy function from Apar=0 to Cpar=255)
- **'moments_row'**
  - Mixed moments along a row (see MomentsGrayPlane)
- **'moments_column'**
  - Mixed moments along a column (see MomentsGrayPlane)
- **'alpha'**
  - Approximating plane, parameter Alpha (see MomentsGrayPlane)
- **'beta'**
  - Approximating plane, parameter Beta (see MomentsGrayPlane)

---

**Attention**

If only one feature is used the value of Operation is meaningless. Several features are processed in the order in which they are entered.

---

**Parameter**

- **Regions** (input iconic) .......................... `region`  \(\sim\) `HRegionX / IHObjectX`
  - Regions to be examined.
- **Image** (input iconic) .......................... `image`  \(\sim\) `HImageX / IHObjectX (byte, int2, int4, real)`
  - Gray value image.
- **SelectedRegions** (output iconic) .......................... `region`  \(\sim\) `HRegionX / HUntypedObjectX`
  - Regions having features within the limits.
- **Features** (input control) .......................... `string(-array)`  \(\sim\) `VARIANT (string)`
  - Names of the features.
  - **Default Value:** 'mean'
  - **Value List:** `Features \in \{ 'area', 'row', 'column', 'ra', 'rb', 'phi', 'min', 'max', 'mean', 'deviation', 'plane_deviation', 'anisotropy', 'entropy', 'fuzzy_entropy', 'fuzzy_perimeter', 'moments_row', 'moments_column', 'alpha', 'beta' \}`
- **Operation** (input control) .......................... `string`  \(\sim\) `String`
  - Logical connection of features.
  - **Default Value:** 'and'
  - **Value List:** `Operation \in \{ 'and', 'or' \}`
- **Min** (input control) .......................... `number(-array)`  \(\sim\) `VARIANT (integer, real)`
  - Lower limit(s) of features.
  - **Default Value:** 128.0
  - **Value Suggestions:** `Min \in \{ 0.5, 1.0, 10.0, 20.0, 50.0, 128.0, 255.0, 1000.0 \}`
- **Max** (input control) .......................... `number(-array)`  \(\sim\) `VARIANT (integer, real)`
  - Upper limit(s) of features.
  - **Default Value:** 255.0
  - **Value Suggestions:** `Max \in \{ 0.5, 1.0, 10.0, 20.0, 50.0, 128.0, 255.0, 1000.0 \}`

---

**Result**

The operator SelectGray returns the value TRUE if the input image has the defined gray values and the parameters are correct. The behavior in case of empty input (no input images available) is set via the oper-
ator SetSystem(::’noObjectResult’,<Result>:), the behavior in case of empty region is set via
SetSystem(::’emptyRegionResult’,<Result>:). If necessary an exception handling is raised.

Parallelization Information
SelectGray is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
Connection, MeanImage, EntropyImage, SobelAmp, MedianSeparate

Possible Successor Functions
SelectShape, SelectGray, ShapeTrans, ReduceDomain, CountObj

See Also
DeviationImage, EntropyGray, Intensity, MeanImage, MinMaxGray, SelectObj

Image filters

Determine a histogram of features along all threshold values.

The operator ShapeHistoAll carries out 255 threshold operations within Region with the gray values of
Image. The entry \( i \) in the histogram corresponds to the number of connected components/holes of this image
segmented with the threshold \( i \) (Feature = ’connected_components’, ’holes’) or the mean value of the feature
values of the regions segmented in this way (Feature = ’convexity’, ’compactness’, ’ansisometry’), respec-
tively.

The histogram can also be displayed directly as a graphic via the operators SetPaint(::
WindowHandle,’componentHistogram’:) and DispImage.

Attention
The operator ShapeHistoAll expects a region and exactly one gray value image as input. Because of the power
of this operator the runtime of ShapeHistoAll is relatively large!

Parameter

\( \textbf{Region} \text{ (input iconic) } \rightarrow \text{ HRegionX / IObjectX} \)
Region in which the features are to be examined.

\( \textbf{Image} \text{ (input iconic) } \rightarrow \text{ HImageX / IObjectX (byte)} \)
Gray value image.

\( \textbf{Feature} \text{ (input control) } \rightarrow \text{ String} \)
Feature to be examined.

\( \textbf{AbsoluteHisto} \text{ (output control) } \rightarrow \text{ VARIANT} \)
Absolute distribution of the feature.

\( \textbf{RelativeHisto} \text{ (output control) } \rightarrow \text{ VARIANT} \)
Relative distribution of the feature.

Example

/* Simulation von shape_histo_all mit Merkmal ’connected_components’: */
my_shape_histo_all(Region, Image, AbsHisto, RelHisto):
reduce_domain(Region, Image, RegionGray)
for(0, 255, i):
    threshold(RegionGray, Seg, i, 255)
    connect_and_holes(Seg, AbsHisto[i], _)
clear_obj([Seg, H])
end_for()
eval(0,Sum)
for(0,255,i)
   eval(Sum+AbsHisto[i],Sum)
end_for()
for(0,255,i)
   eval(AbsHisto[i]/Sum,RelHisto[i])
end_for()

Result

The operator ShapeHistoAll returns the value TRUE if an image with the defined gray values is entered. The behavior in case of empty input (no input images) is set via the operator SetSystem(:: ‘noObjectResult’,<Result>);, the behavior in case of empty region is set via SetSystem(:: ‘emptyRegionResult’,<Result>);. If necessary an exception handling is raised.

Parallelization Information
ShapeHistoAll is reentrant and processed without parallelization.

Possible Successor Functions
HistToThresh, Threshold, GenRegionHisto
See Also
Connection, Convexity, Compactness, ConnectAndHoles, EntropyGray, GrayHisto, SetPaint, CountObj

Alternatives
ShapeHistoPoint

Image filters

[out] <none>X AbsoluteHisto HRegionX. ShapeHistoPoint
[out] <none>X RelativeHisto )
void HOperatorSetX. ShapeHistoPoint ([in] IHObjectX Region,
[out] VARIANT AbsoluteHisto, [out] VARIANT RelativeHisto )

Determine a histogram of features along all threshold values.
Like ShapeHistoAll the operator ShapeHistoPoint carries out 255 threshold value operations within Region with the gray values of Image. Contrary to ShapeHistoAll only the segmented region containing the pixel (Row, Column) is taken into account here. The entry $i$ in the histogram then corresponds to the number of holes of this region segmented with the threshold $i$ (Feature = ‘holes’) or the feature value of the region (Feature = ‘convexity’, ‘compactness’, ‘anisometry’), respectively.

The histogram can also be displayed directly as a graphic via the operators SetPaint(:: WindowHandle,‘componentHistogram’: ) and DispImage.

Parameter

▷ Region (input iconic) ......................................................region \( \sim \) HRegionX / IHObjectX
Region in which the features are to be examined.
▷ Image (input iconic) ......................................................image \( \sim \) HImageX / IHObjectX ( byte )
Gray value image.
▷ Feature (input control) ..................................................string \( \sim \) String
Feature to be examined.
 Default Value : ‘convexity’
 Value List : Feature \( \in \{ ‘\text{convexity}', ‘\text{compactness}', ‘\text{anisometry}', ‘\text{holes}' \} \)
Row (input control) ................................................. point.y \sim \text{long}
Row of the pixel which the region must contain.
Default Value: 256
Value Suggestions: Row \in \{10, 50, 100, 200, 300, 400\}

Column (input control) ............................................. point.x \sim \text{long}
Column of the pixel which the region must contain.
Default Value: 256
Value Suggestions: Column \in \{10, 50, 100, 200, 300, 400\}

AbsoluteHisto (output control) .................. histogram \sim \text{name}_X / \text{VARIANT}(\text{integer, real})
Absolute distribution of the feature.

RelativeHisto (output control) .................. histogram \sim \text{name}_X / \text{VARIANT}(\text{real})
Relative distribution of the feature.

---

The operator ShapeHistoPoint returns the value TRUE if an image with defined gray values is entered. The behavior in case of empty input (no input images available) is set via the operator SetSystem(:
\text{`}noObjectResult','<Result>\text{'); the behavior in case of empty region is set via SetSystem(\text{(':}
\text{`}emptyRegionResult','<Result>\text{'). If necessary an exception handling is raised.

---

Parallelization Information
ShapeHistoPoint is reentrant and processed without parallelization.

---

Possible Predecessor Functions
GetMbutton, AreaCenter

Possible Successor Functions
HistoToThresh, Threshold, GenRegionHisto

See Also
Connection, ConnectAndHoles, Convexity, Compactness, SetPaint

Alternatives
ShapeHistoAll

Module
Image filters

5.5 Format

<table>
<thead>
<tr>
<th>[out]</th>
<th>HImageX ImagePart</th>
<th>HImageX.ChangeFormat ([in] long Width, [in] long Height )</th>
</tr>
</thead>
</table>

Change image size.

The operator ChangeFormat increases or decreases the size of the input images to the indicated height or width, respectively. If the image is reduced parts are cut off at the “right” or “lower” edge of the image, respectively. If the image is enlarged the additional areas are set to 0. The definition domain includes all pixels of the new image. No zooming is carried out.

---

Parameter

- Image (input iconic) ............................................. image(-array) \sim HImageX / HObjectX
Input image.

- ImagePart (output iconic) ..................................... image(-array) \sim HImageX / HUntypedObjectX
Image with new format.

- Width (input control) ............................................ extent.x \sim \text{long}
Width of new image.
Default Value: 512
Value Suggestions: Width \in \{32, 64, 128, 256, 512, 768, 1024\}
### Height (input control)

- **Default Value:** 512
- **Value Suggestions:** Height ∈ {32, 64, 128, 256, 512, 525, 1024}

**Parallelization Information**

ChangeFormat is reentrant and automatically parallelized (on tuple level).

**Possible Successor Functions**

DispImage

**See Also**

ZoomImageSize, ZoomImageFactor

**Alternatives**

CropPart

**Module**

Image / region / XLD management

```c
[out] HImageX ImagePart HImageX.CropDomain ( )
void HOperatorSetX.CropDomain ([in] IHObjectX Image,
[out] HUntypedObjectX ImagePart )
```

*Cut out of defined gray values.*

The operator CropDomain cuts a rectangular area from the input images. This rectangle is the smallest surrounding rectangle of the domain of the input image. The new definition domain includes all pixels of the new image. The new image matrix has the size of the rectangle.

**Parameter**

- **Image** (input iconic) ► (multichannel-)image(-array) ↦ HImageX / IHObjectX
  - Input image.
- **ImagePart** (output iconic) ◄ (multichannel-)image(-array) ↦ HImageX / HUntypedObjectX
  - Image area.

**Parallelization Information**

CropDomain is reentrant and automatically parallelized (on tuple level).

**Possible Successor Functions**

DispImage

**See Also**

ZoomImageSize, ZoomImageFactor

**Alternatives**

CropPart, CropRectangle1, ChangeFormat, ReduceDomain

**Module**

Image / region / XLD management

```c
[out] HImageX ImagePart HImageX.CropPart ([in] long Row, [in] long Column,
[in] long Width, [in] long Height )
void HOperatorSetX.CropPart ([in] IHObjectX Image,
[in] long Width, [in] long Height )
```

*Cut out a rectangular image area.*

The operator CropPart cuts a rectangular area from the input images. The area is indicated by a rectangle (upper left corner and size). The area must be within the image. The definition domain includes all pixels of the new image. The new image matrix has the size of a rectangle.
Parameter

- **Image** (input iconic) .......................... (multichannel-)image-array)  HImageX / IHObjectX  
  Input image.

- **ImagePart** (output iconic) .................. (multichannel-)image-array)  HImageX / HUntypedObjectX  
  Image area.

- **Row** (input control) .......................... rectangle.origin.y  long  
  Line index of upper left corner of image area.  
  Default Value : 100  
  Value Suggestions : Row ∈ { 10, 20, 50, 100, 200, 300, 500 }  
  Typical Range of Values : 0 ≤ Row ≤ 0

- **Column** (input control) ...................... rectangle.origin.x  long  
  Column index of upper left corner of image area.  
  Default Value : 100  
  Value Suggestions : Column ∈ { 10, 20, 50, 100, 200, 300, 500 }  
  Typical Range of Values : 0 ≤ Column ≤ 0

- **Width** (input control) ...................... rectangle.extent.x  long  
  Width of new image.  
  Default Value : 128  
  Value Suggestions : Width ∈ { 32, 64, 128, 256, 512, 768 }  
  Typical Range of Values : 0 ≤ Width ≤ 0

- **Height** (input control) ...................... rectangle.extent.y  long  
  Height of new image.  
  Default Value : 128  
  Value Suggestions : Height ∈ { 32, 64, 128, 256, 512, 525 }  
  Typical Range of Values : 0 ≤ Height ≤ 0

Parallelization Information

CropPart is reentrant and automatically parallelized (on tuple level).

Possible Successor Functions

DispImage

ZoomImageSize, ZoomImageFactor

Alternatives

CropRectangle1, CropDomain, ChangeFormat, ReduceDomain

Module

Image / region / XLD management

```

```

Cut out a rectangular image area.

The operator CropRectangle1 cuts a rectangular area from the input images. The area is indicated by a rectangle (upper left and lower right corner). The area must be within the image. The definition domain includes all pixels of the new image. The new image matrix has the size of a rectangle.

Parameter

- **Image** (input iconic) .......................... (multichannel-)image-array)  HImageX / IHObjectX  
  Input image.

- **ImagePart** (output iconic) .................. (multichannel-)image-array)  HImageX / HUntypedObjectX  
  Image area.
5.5. FORMAT

- **Row1** (input control) ........................................... rectangle.origin.y ~ long
  - Line index of upper left corner of image area.
  - Default Value: 100
  - Value Suggestions: Row1 ∈ {10, 20, 50, 100, 200, 300, 500}
  - Typical Range of Values: 0 ≤ Row1 ≤ 0

- **Column1** (input control) ........................................... rectangle.origin.x ~ long
  - Column index of upper left corner of image area.
  - Default Value: 100
  - Value Suggestions: Column1 ∈ {10, 20, 50, 100, 200, 300, 500}
  - Typical Range of Values: 0 ≤ Column1 ≤ 0

- **Row2** (input control) ........................................... rectangle.corner.y ~ long
  - Line index of lower right corner of image area.
  - Default Value: 200
  - Value Suggestions: Row2 ∈ {10, 20, 50, 100, 200, 300, 500}
  - Typical Range of Values: 0 ≤ Row2 ≤ 0

- **Column2** (input control) ........................................... rectangle.corner.x ~ long
  - Column index of lower right corner of image area.
  - Default Value: 200
  - Value Suggestions: Column2 ∈ {10, 20, 50, 100, 200, 300, 500}
  - Typical Range of Values: 0 ≤ Column2 ≤ 0

---

**Parallelization Information**

CropRectangle1 is reentrant and automatically parallelized (on tuple level).

---

**Possible Successor Functions**

DispImage

---

**See Also**

ZoomImageSize, ZoomImageFactor

---

**Alternatives**

CropPart, CropDomain, ChangeFormat, ReduceDomain

---

**Module**

Image / region / XLD management

---

```
[out] HImageX TiledImage HImageX.TileChannels ([in] long NumColumns,
[in] String TileOrder )

void HOperatorSetX.TileChannels ([in] IObjectX Image,
[out] HUntypedObjectX TiledImage, [in] long NumColumns,
[in] String TileOrder )
```

---

**Tile multiple images into a large image.**

TileChannels tiles an image consisting of multiple channels into a large single-channel image. The input image Image contains Num images of the same size, which are stored in the individual channels. The output image TiledImage contains a single channel image, where the Num input channels have been tiled into NumColumns columns. In particular, this means that TileChannels cannot tile color images. For this purpose, TileImages can be used. The parameter TileOrder determines the order in which the images are copied into the output in the cases in which this is not already determined by NumColumns (i.e., if NumColumns ≠ 1 and NumColumns ≠ Num). If TileOrder = 'horizontal' the images are copied in the horizontal direction, i.e., the second channel of Image will be to the right of the first channel. If TileOrder = 'vertical' the images are copied in the vertical direction, i.e., the second channel of Image will be below the first channel. The domain of TiledImage is obtained by copying the domain of Image to the corresponding locations in the output image. If Num is not a multiple of NumColumns the output image will have undefined gray values in the lower right corner of the image. The output domain will reflect this.

---

**Attention**
Parameter

- **Image** (input iconic) multichannel-image-array \(\sim HImageX / IObjectX\) (byte, int1, cyclic, direction, int2, int4, real)
  - Input image.
- **TiledImage** (output iconic) image-array \(\sim HIImageX / HIObjectX\)
  - Tiled output image.
- **NumColumns** (input control) \(\sim long\)
  - Number of columns to use for the output image.
    - **Default Value** : 1
    - **Value Suggestions** : \(\{1, 2, 3, 4, 5, 6, 7\}\)
- **TileOrder** (input control) \(\sim String\)
  - Order of the input images in the output image.
    - **Default Value** : 'vertical'
    - **Value List** : \{'horizontal', 'vertical'\}

Example

```c
/* Grab 5 single-channel images and stack them vertically. */
gen_rectangle1 (Image, 0, 0, Height-1, Width-1)
for I := 1 to 5 by 1
   grab_image_async (ImageGrabbed, FGHandle, -1)
   append_channel (Image, ImageGrabbed, Image)
endfor
tile_channels (Image, TiledImage, 1, 'vertical')
```

Result

TileChannels returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behavior can be set via `SetSystem(‹:‘noObjectResult’,<Result>;)`. If necessary, an exception handling is raised.

Parallelization Information

TileChannels is *reentrant* and automatically *parallelized* (on tuple level).

Possible Predecessor Functions

- **AppendChannel**
- **ChangeFormat, CropPart, CropRectangle1**

Alternatives

- **TileImages, TileImagesOffset**

Module

Image / region / XLD management

---

**Tile multiple image objects into a large image.**

TileImages tiles multiple input image objects, which must contain the same number of channels, into a large image. The input image object Images contains \(Num\) images, which may be of different size. The output image TiledImage contains as many channels as the input images. In the output image the \(Num\) input images have been tiled into NumColumns columns. Each tile has the same size, which is determined by the maximum width and height of all input images. If an input image is smaller than the tile size it is copied to the center of the respective tile. The parameter TileOrder determines the order in which the images are copied into the output in the cases
in which this is not already determined by NumColumns (i.e., if NumColumns \(!= 1 \text{ and } \text{NumColumns} \neq \text{Num}\)).

If TileOrder = 'horizontal' the images are copied in the horizontal direction, i.e., the second image of Images will be to the right of the first image. If TileOrder = 'vertical' the images are copied in the vertical direction, i.e., the second image of Images will be below the first image. The domain of TiledImage is obtained by copying the domains of Images to the corresponding locations in the output image. If Num is not a multiple of NumColumns the output image will have undefined gray values in the lower right corner of the image. The output domain will reflect this.

Attention

Parameter

- \texttt{Images} (input iconic) \(\ldots\) (multichannel-)image \(\sim\) \texttt{HImageX, HObjectX} (byte, int1, cyclic, direction, int2, int4, real)
  - Input images.

- \texttt{TiledImage} (output iconic) \(\ldots\) (multichannel-)image \(\sim\) \texttt{HImageX, HUntypedObjectX}
  - Tiled output image.

- \texttt{NumColumns} (input control) \(\ldots\) integer \(\sim\) long
  - Number of columns to use for the output image.
  - Default Value: 1
  - Value Suggestions: \texttt{NumColumns} \(\in\) \{1, 2, 3, 4, 5, 6, 7\}

- \texttt{TileOrder} (input control) \(\ldots\) string \(\sim\) String
  - Order of the input images in the output image.
  - Default Value: 'vertical'
  - Value List: \texttt{TileOrder} \(\in\) \{'horizontal', 'vertical'\}

Example

/* Grab 5 (multi-channel) images and stack them vertically. */
gen_empty_obj (Images)
for I := 1 to 5 by 1
  grab_image_async (ImageGrabbed, FGHandle, -1)
  concat_obj (Images, ImageGrabbed, Images)
endfor
tile_images (Images, TiledImage, 1, 'vertical')

Result

TileImages returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behavior can be set via \texttt{SetSystem}(::'noObjectResult',<Result>::). If necessary, an exception handling is raised.

Parallelization Information

TileImages is \textit{reentrant} and automatically \textit{parallelized} (on channel level).

Possible Predecessor Functions

- \texttt{AppendChannel}

See Also

- \texttt{ChangeFormat, CropPart, CropRectangle1}

Alternatives

- \texttt{TileChannels, TileImagesOffset}

Module

- Image / region / XLD management
Tile multiple image objects into a large image with explicit positioning information.

TileImagesOffset tiles multiple input image objects, which must contain the same number of channels, into a large image. The input image object Images contains Num images, which may be of different size. The output image TiledImage contains as many channels as the input images. The size of the output image is determined by the parameters Width and Height. The position of the upper left corner of the input images in the output images is determined by the parameters OffsetRow and OffsetCol. Both parameters must contain exactly Num values. Optionally, each input image can be cropped to an arbitrary rectangle that is smaller than the input image. To do so, the parameters Row1, Col1, Row2, and Col2 must be set accordingly. If any of these four parameters is set to -1, the corresponding input image is not cropped. In any case, all four parameters must contain Num values. If the input images are cropped the position parameters OffsetRow and OffsetCol refer to the upper left corner of the cropped image. If the input images overlap each other in the output image (while taking into account their respective domains), the image with the higher index in Images overwrites the image data of the image with the lower index. The domain of TiledImage is obtained by copying the domains of Images to the corresponding locations in the output image.

Attention
If the input images all have the same size and tile the output image exactly, the operator TileImages usually will be slightly faster.

Parameter

- **Images** (input iconic) .. (multichannel-)image \(\sim\) HImageX / IHObjectX (byte, int1, cyclic, direction, int2, int4, real)
  Input images.
- **TiledImage** (output iconic) .................. (multichannel-)image \(\sim\) HImageX / HUntypedObjectX
  Tiled output image.
- **OffsetRow** (input control) .................. point.y \(\sim\) VARIANT (integer)
  Row coordinate of the upper left corner of the input images in the output image.
  Default Value : 0
  Value Suggestions : OffsetRow \(\in\) \{0, 50, 100, 150, 200, 250\}
- **OffsetCol** (input control) .................. point.x \(\sim\) VARIANT (integer)
  Column coordinate of the upper left corner of the input images in the output image.
  Default Value : 0
  Value Suggestions : OffsetCol \(\in\) \{0, 50, 100, 150, 200, 250\}
- **Row1** (input control) ................................ rectangle.origin.y \(\sim\) VARIANT (integer)
  Row coordinate of the upper left corner of the copied part of the respective input image.
  Default Value : -1
  Value Suggestions : Row1 \(\in\) \{-1, 0, 10, 20, 50, 100, 200, 300, 500\}
- **Col1** (input control) ................................ rectangle.origin.x \(\sim\) VARIANT (integer)
  Column coordinate of the upper left corner of the copied part of the respective input image.
  Default Value : -1
  Value Suggestions : Col1 \(\in\) \{-1, 0, 10, 20, 50, 100, 200, 300, 500\}
- **Row2** (input control) ................................ rectangle.corner.y \(\sim\) VARIANT (integer)
  Row coordinate of the lower right corner of the copied part of the respective input image.
  Default Value : -1
  Value Suggestions : Row2 \(\in\) \{-1, 0, 10, 20, 50, 100, 200, 300, 500\}
- **Col2** (input control) ................................ rectangle.corner.x \(\sim\) VARIANT (integer)
  Column coordinate of the lower right corner of the copied part of the respective input image.
  Default Value : -1
  Value Suggestions : Col2 \(\in\) \{-1, 0, 10, 20, 50, 100, 200, 300, 500\}
Width (input control) .............................................................. extent.x ~ long
Width of the output image.
Default Value : 512
Value Suggestions : Width ∈ {32, 64, 128, 256, 512, 768, 1024, 2048, 4096}

Height (input control) .............................................................. extent.y ~ long
Height of the output image.
Default Value : 512
Value Suggestions : Height ∈ {32, 64, 128, 256, 512, 525, 1024, 2048, 4096}

Example
/* Grab 2 (multi-channel) NTSC images, crop the bottom 5 lines off */
/* of each image, the right 5 columns off of the first image, and */
/* the left five lines off of the second image, and put the cropped */
/* images side-by-side. */
gen_empty_obj (Images)
for I := 1 to 2 by 1
  grab_image_async (ImageGrabbed, FGHandle, -1)
  concat_obj (Images, ImageGrabbed, Images)
endfor
tile_images_offset (Images, TiledImage, [0,635], [0,0], [0,0],
                      [0,5], [474,474], [634,639])

Result
TileImagesOffset returns TRUE if all parameters are correct and no error occurs during execution. If the
input is empty the behavior can be set via SetSystem(::’noObjectResult’,<Result>:).
If necessary, an exception handling is raised.

Parallelization Information
TileImagesOffset is reentrant and automatically parallelized (on channel level).

Possible Predecessor Functions
AppendChannel

See Also
ChangeFormat, CropPart, CropRectangle1

Alternatives
TileChannels, TileImages

Module
Image / region / XLD management

5.6 Framegrabber

void HMiscX.CloseAllFramegrabbers ( )
void HOperatorSetX.CloseAllFramegrabbers ( )

Close all frame grabbers.
The operator CloseAllFramegrabbers closes all frame grabbers. It is used to cope with deadlocks resulting
from damaged frame grabber handles (in that case the use of CloseFramegrabber is impossible).

Attention
Since all frame grabbers are closed by CloseAllFramegrabbers all frame grabber handles become invalid.

Result
If it is possible to close the frame grabbers the operator CloseAllFramegrabbers returns the value TRUE.
Otherwise an exception handling is raised.

Parallelization Information
CloseAllFramegrabbers is local and processed completely exclusively without parallelization.
void HOperatorSetX.CloseFramegrabber ([in] long FGHandle )

Close specified frame grabber.
The operator CloseFramegrabber closes the frame grabber specified by FGHandle. In particular possible storage space for data buffers is released and the frame grabber is made available for other processes.

Parameter

▷ FGHandle (input control) ............................................framegrabber ~ HFramegrabberX / long Handle of the frame grabber to be closed.

Result

If the frame grabber is open the operator CloseFramegrabber returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

CloseFramegrabber is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

GrabImage

See Also

OpenFramegrabber

Module

Image / region / XLD management

[out] VARIANT ImageRed HFramegrabberX.GetFramegrabberLut
([out] VARIANT ImageGreen, [out] VARIANT ImageBlue )

void HOperatorSetX.GetFramegrabberLut ([in] long FGHandle,
[out] VARIANT ImageRed, [out] VARIANT ImageGreen, [out] VARIANT ImageBlue )

Query frame grabber lut.
The operator GetFramegrabberLut queries the lut of the frame grabber specified by FGHandle. This operation is not supported for all kinds of frame grabbers.

Parameter

▷ FGHandle (input control) ............................................framegrabber ~ HFramegrabberX / long Handle of the desired frame grabber.
▷ ImageRed (output control) ............................................integer ~ VARIANT( integer ) Red level of the lut entries.
▷ ImageGreen (output control) ............................................integer ~ VARIANT( integer ) Green level of the lut entries.
▷ ImageBlue (output control) ............................................integer ~ VARIANT( integer ) Blue level of the lut entries.

Result

The operator GetFramegrabberLut returns the value TRUE if the frame grabber is open.
Parallelization Information

GetFramegrabberLut is reentrant and processed without parallelization.

Possible Predecessor Functions

OpenFramegrabber

Possible Successor Functions

SetFramegrabberLut

See Also

SetFramegrabberLut, OpenFramegrabber

Module

Image / region / XLD management

Get specific parameters for a frame grabber.

The operator GetFramegrabberParam returns specific parameter values for the frame grabber specified by FGHandle. The standard parameters listed below are available for all frame grabbers. Additional parameters may be supported by specific frame grabbers. A list of those can be obtained with the query 'parameter' via InfoFramegrabber.

Standard values for Param, see OpenFramegrabber:

'name' Name of the frame grabber.
'horizontal_resolution' Horizontal resolution of the frame grabber.
'vertical_resolution' Vertical resolution of the frame grabber.
'image_width' Width of the image area.
'image_height' Height of the image area.
'start_row' Line number of upper left corner of the image area.
'start_column' Column number of upper left corner of the image area.
'field' Selected half image or for full image.
'bits_per_channel' Number of transferred bits per pixel and image channel.
'color_space' Color images: color space.
'gain' Amplification factor for video amplifier.
'external_trigger' External triggering ('true' / 'false').
'camera_type' Type of used camera (frame grabber specific).
'device' Device the frame grabber is linked to.
'port' Port of the frame grabber the video signal is linked to.
'line_in' Camera input line of multiplexer.

Attention

Parameter

- **FGHandle** (input control) ............................... framegrabber ~ HFramegrabberX / long Handle of the frame grabber to be used.
- **Param** (input control) ............................... string(-array) ~ VARIANT ( string ) Parameter of interest.
  
  Default Value: 'name'
  
  Value Suggestions: Param ∈ {'name', 'horizontal_resolution', 'vertical_resolution', 'image_width', 'image_height', 'start_row', 'start_column', 'field', 'bits_per_channel', 'color_space', 'gain', 'external_trigger', 'camera_type', 'device', 'port', 'line_in'}
  
- **Value** (output control) ............................... string(-array) ~ VARIANT ( string, real, integer ) Parameter value.

HALCON 6.0
If the frame grabber is open and the specified parameter is supported by the frame grabber, the operator GetFramegrabberParam returns the value TRUE. Otherwise an exception handling is raised.

--- Parallelization Information ---
GetFramegrabberParam is reentrant and processed without parallelization.

--- Possible Predecessor Functions ---
OpenFramegrabber

--- Possible Successor Functions ---
GrabImage, GrabRegion, GrabImageStart, GrabImageAsync, GrabRegionAsync, CloseFramegrabber

See Also
OpenFramegrabber, InfoFramegrabber, SetFramegrabberParam

Module
Image / region / XLD management

--- Parameter ---

> **Image** (output iconic) ................................. image  \( \sim \) HImageX / HUntypedObjectX
Grabbed image.

> **FGHandle** (input control) ............................... framegrabber  \( \sim \) HFramegrabberX / long
Handle of the frame grabber to be used.

--- Example ---

```cpp
// Select a suitable frame grabber FgName
info_framegrabber(FgName,'ports',Information,Val)
// Choose the port P and the input line L your camera is connected to
open_framegrabber(FgName,1,1,0,0,0,0,'default',-1,'default','-1.0,'default','default','default',P,L,FGHandle)
grab_image(Img,FGHandle)
close_framegrabber(FGHandle).
```

--- Result ---

If the frame grabber is open the operator GrabImage returns the value TRUE. Otherwise an exception handling is raised.

--- Parallelization Information ---
GrabImage is reentrant and processed without parallelization.

--- Possible Predecessor Functions ---
OpenFramegrabber, GrabImageStart

--- Possible Successor Functions ---
GrabImage, GrabImageStart, GrabImageAsync, CloseFramegrabber

--- See Also ---
OpenFramegrabber, InfoFramegrabber, SetFramegrabberParam

---

Module

Image / region / XLD management

```c
void HImageX.GrabImageAsync ([in] HFramegrabberX FGHandle, 
[in] double MaxDelay )
```

```
[out] HImageX Image = HFramegrabberX.GrabImageAsync 
([in] double MaxDelay )
```

```c
void HOperatorSetX.GrabImageAsync ([out] HUntypedObjectX Image, 
[in] long FGHandle, [in] double MaxDelay )
```

Grab of an image from the specified frame grabber and start of the asynchronous grab of the next image.

The operator `GrabImageAsync` grabs an image via the frame grabber specified by `FGHandle` and starts the asynchronous grab of the next image. The desired operational mode of the frame grabber as well as a suitable image area can be adjusted via the operator `OpenFramegrabber`. Additional (frame grabber specific) settings might be possible via `SetFramegrabberParam`. The grab of the next image is finished via `GrabImage` or again via `GrabImageAsync`. If more than `MaxDelay` ms have passed since the asynchronous grab was started, a new image is grabbed (the asynchronously grabbed image is considered as too old). If a negative value is assigned to `MaxDelay` this control mechanism is deactivated.

**Attention**

**Parameter**

- **Image** (output iconic) .............................................. image \(\sim\) `HImageX / HUntypedObjectX`
  - Grabbed image.
- **FGHandle** (input control) ......................................................... `HFramegrabberX / long`
  - Handle of the frame grabber to be used.
- **MaxDelay** (input control) ...................................................... number \(\sim\) `double`
  - Maximum tolerated delay between the start of the asynchronous grab and the delivery of the image [ms].
  - **Default Value** : -1.0
  - **Value Suggestions** : `MaxDelay \in \{-1.0, 20.0, 33.3, 40.0, 66.6, 80.0, 99.9\}`

**Example**

```c
// Select a suitable frame grabber FgName
info_framegrabber(FgName,’ports’,Information,Val)
// Choose the port P and the input line L your camera is connected to
open_framegrabber(FgName,1,1,0,0,0,’default’,-1,’default’,-1.0, 
’default’,’default’,’default’,P,L,FgHandle)
// grab image + start next grab
grab_image_async(Img,FgHandle,-1.0)
// Process Img ...
// Finish asynchronous grab + start next grab
grab_image_async(Img,FgHandle,-1.0)
close_framegrabber(FgHandle).
```

**Result**

If the frame grabber is open and supports asynchronous grabbing the operator `GrabImageAsync` returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

`GrabImageAsync` is reentrant and processed without parallelization.

**Possible Predecessor Functions**

- `OpenFramegrabber`

**Possible Successor Functions**

- `GrabImage`, `GrabImageAsync`, `CloseFramegrabber`
void **[in] double MaxDelay)**

Start an asynchronous grab of an image from the specified frame grabber.

The operator `GrabImageStart` starts an asynchronous grab of an image via the frame grabber specified by `FGHandle`. The desired operational mode of the frame grabber as well as a suitable image area can be adjusted via the operator `OpenFramegrabber`. Additional (frame grabber specific) settings might be possible via `SetFramegrabberParam`. The grab is finished via `GrabImage`, `GrabImageAsync`, `GrabRegion`, or `GrabRegionAsync`. If one of those operators is called more than `MaxDelay` ms later, a new image is grabbed (the asynchronously grabbed image is considered as to old). If a negative value is assigned to `MaxDelay` this control mechanism is deactivated.

**Attention**

**Parameter**

- **FGHandle** (input control) .................framegrabber \(~=\) **HFramegrabberX** / long Handle of the frame grabber to be used.
- **MaxDelay** (input control) ......................number \(~=\) double Maximum tolerated delay between the start of the asynchronous grab and the delivery of the image [ms].

**Default Value** : -1.0

**Value Suggestions** : `MaxDelay \in \{20.0, 33.3, 40.0, 66.6, 80.0, 99.9\}`

**Example**

```c
// Select a suitable frame grabber FgName
info_framegrabber(FgName,'ports',Information,Val)
// Choose the port P and the input line L your camera is connected to
open_framegrabber(FgName,1,1,0,0,0,'default','-1','default','-1.0',
                      'default','default','default',P,L,FGHandle)
grab_image(Img,FGHandle)
// Start next grab
grab_image_start(FGHandle,-1.0)
// Process Img ...
// Finish asynchronous grab
grab_image(Img,FGHandle)
close_framegrabber(FGHandle).
```

**Result**

If the frame grabber is open and supports asynchronous grabbing the operator `GrabImageStart` returns the value `TRUE`. Otherwise an exception handling is raised.

**Parallelization Information**

`GrabImageStart` is **reentrant** and processed **without** parallelization.

**Possible Predecessor Functions**

- `OpenFramegrabber`

**Possible Successor Functions**

- `GrabImage`, `GrabRegion`, `GrabImageAsync`, `GrabRegionAsync`, `CloseFramegrabber`

**See Also**

- `OpenFramegrabber`, `InfoFramegrabber`, `SetFramegrabberParam`

**Module**

Image / region / XLD management
5.6. FRAMEGRABBER

Grab and segment an image from the specified frame grabber.

The operator `GrabRegion` grabs an image via the frame grabber specified by `FGHandle` and segments it. The desired operational mode of the frame grabber as well as a suitable image area can be adjusted via the operator `OpenFramegrabber`. Additional (frame grabber specific) settings might be possible via `SetFramegrabberParam`.

```
void HRegionX.GrabRegion ([in] HFramegrabberX FGHandle )
[out] HRegionX Region HFramegrabberX.GrabRegion ( )
```

```
void HOperatorSetX.GrabRegion ([out] HUntypedObjectX Region, [in] long FGHandle )
```

Grab and segment an image from the specified frame grabber.

```
Grab and segment an image from the specified frame grabber and start the asynchronous grab of the next image.
```

```
void HRegionX.GrabRegionAsync ([in] HFramegrabberX FGHandle, [in] double MaxDelay )
[out] HRegionX Region HFramegrabberX.GrabRegionAsync ([in] double MaxDelay )
```

```
```

HALCON 6.0
The operator \texttt{GrabRegionAsync} grabs an image via the frame grabber specified by \texttt{FGHandle}, segments it, and starts the asynchronous grab of the next image. The desired operational mode of the frame grabber as well as a suitable image area can be adjusted via the operator \texttt{OpenFramegrabber}. Additional (frame grabber specific) settings might be possible via \texttt{SetFramegrabberParam}. The grab of the next image is finished via \texttt{GrabRegion} or again via \texttt{GrabRegionAsync}. If more than \texttt{MaxDelay} ms have passed since the asynchronous grab was started, a new image is grabbed (the asynchronously grabbed image is considered as too old). If a negative value is assigned to \texttt{MaxDelay} this control mechanism is deactivated.

\begin{itemize}
\item \textbf{Attention}
\end{itemize}

\begin{itemize}
\item \underline{Parameter}
\end{itemize}

- \textbf{Region} (output iconic) \hspace{1cm} \texttt{Region(-array)} \rightarrow \texttt{HRegionX/HUntypedObjectX}
  
 Grabbed and segmented image: Region(s).

- \textbf{FGHandle} (input control) \hspace{1cm} \texttt{framegrabber} \rightarrow \texttt{HFramegrabberX/long}
  
  Handle of the frame grabber to be used.

- \textbf{MaxDelay} (input control) \hspace{1cm} \texttt{number} \rightarrow \texttt{double}
  
  Maximum tolerated delay between the start of the asynchronous grab and the delivery of the image [ms].

  \textbf{Default Value} : -1.0

  \textbf{Value Suggestions} : \texttt{MaxDelay} \in \{-1.0, 20.0, 33.3, 40.0, 66.6, 80.0, 99.9\}

\begin{itemize}
\item \underline{Example}
\end{itemize}

```c
// Select a suitable frame grabber FgName
info_framegrabber(FgName,'ports',Information,Val)
// Choose the port P and the input line L your camera is connected to
open_framegrabber(FgName,1,1,0,0,0,'default',-1,'default',-1.0,
                   'default','default',P,L,FGHandle)
// grab image, segment it, and start next grab
grab_region_async(Region,FGHandle,-1.0)
// Process Region ...
// Finish asynchronous grab, segment this image, and start next grab
grab_region_async(Region,FGHandle,-1.0)
close_framegrabber(FGHandle).
```

\begin{itemize}
\item \underline{Result}
\end{itemize}

If the frame grabber is open the operator \texttt{GrabRegionAsync} returns the value TRUE. Otherwise an exception handling is raised.

\begin{itemize}
\item \underline{Parallelization Information}
\end{itemize}

\texttt{GrabRegionAsync} is \texttt{reentrant} and processed \texttt{without} parallelization.

\begin{itemize}
\item \underline{Possible Predecessor Functions}
\end{itemize}

\texttt{OpenFramegrabber,GrabImageStart}

\begin{itemize}
\item \underline{Possible Successor Functions}
\end{itemize}

\texttt{GrabRegion,GrabRegionAsync,GrabImageStart,GrabImage,GrabImageAsync,CloseFramegrabber}

\begin{itemize}
\item \underline{See Also}
\end{itemize}

\texttt{OpenFramegrabber,InfoFramegrabber,SetFramegrabberParam}

\begin{itemize}
\item \underline{Module}
\end{itemize}

\texttt{Image / region / XLD management}

\begin{verbatim}
[out] String Information HInfoX.InfoFramegrabber ([in] String Name,
[in] String Query, [out] VARIANT ValueList )
void HOperatorSetX.InfoFramegrabber ([in] String Name,
\end{verbatim}

\textit{Information about the specified frame grabber.}
The operator InfoFramegrabber returns information about the frame grabber Name. The desired information is specified via Query. A textual description according to the selected topic is returned in Information. If applicable, ValueList contains a list of supported values. Up to now, the following queries are possible:

'ports': Description of the ports (signal, connectors etc.) in Information and the port numbers in ValueList.
'camera_types': Description of the frame grabber specific parameter 'CameraType', see OpenFramegrabber.
'general': General information (in Information).
'defaults': Framegrabber specific default values in ValueList, see OpenFramegrabber.
'parameters': Frame grabber specific parameters accessible via SetFramegrabberParam and GetFramegrabberParam.

Please check also the directory doc/html/manuals for files describing selected frame grabbers.

**Attention**

| Parameter |
|-----------------|-----------------|
| Name (input control) .................................. string | String |
| Frame grabber of interest. |
| Default Value : 'File' |
| Value Suggestions : Name ∈ {'BitFlow', 'DFG-BW1', 'DFG-LC', 'DT315x', 'File', 'Fire-i', 'FlashBus', 'GINGA', 'IDS', 'Inspecta2', 'MatrixVision', 'Meteor1', 'Opteon', 'PCEye', 'PicPort', 'PX', 'PXC', 'PXD', 'Ramses1', 'TWAIN', 'VideoPort'} |
| Query (input control) ......................... string | String |
| Name of the chosen query. |
| Default Value : 'info_boards' |
| Value List : Query ∈ {'info_boards', 'ports', 'camera_types', 'general', 'defaults', 'parameters'} |
| Information (output control) ..................... string | String |
| Textual information (according to Query). |
| ValueList (output control) ....................... string(-array) | VARIANT ( string, integer, real ) |
| If applicable, a list of values (according to Query). |

**Example**

// Select a suitable frame grabber FgName
info_framegrabber(FgName,'ports',Information,Val)

// Choose the port P and the input line L your camera is connected to
open_framegrabber(FgName,1,1,0,0,0,'default',-1,'default',-1.0,
                  'default','default','default',P,L,FgHandle)
grab_image(Img,FgHandle)
close_framegrabber(FgHandle).

**Result**

If the parameter values are correct and the desired frame grabber is available at call time, InfoFramegrabber returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

InfoFramegrabber is local and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

OpenFramegrabber

**Possible Successor Functions**

OpenFramegrabber

**See Also**

OpenFramegrabber

**Module**

Image / region / XLD management
Open and configure a frame grabber.

The operator OpenFramegrabber opens and configures the chosen frame grabber. During this process the connection to the frame grabber is tested, the frame grabber is (normally) locked for other processes, and if necessary memory is reserved as data buffer. The image is actually grabbed via the operators GrabImage, GrabRegion, GrabImageAsync, or GrabRegionAsync. If the frame grabber is not needed anymore, it should be closed via the operator CloseFramegrabber, releasing it for other processes. This automatically happens when a frame grabber is reopened. Some frame grabbers allow to open several instances of the same frame grabber class (up to 5).

For some parameters frame grabber specific default values can be chosen explicitly (see the parameter description below). Additional information for a specific frame grabber is available via InfoFramegrabber. Please check also the directory doc/html/manuals for files describing selected frame grabbers.

The meaning of the particular parameters is as follows:

- **HorizontalResolution, VerticalResolution** Desired resolution of the frame grabber.
- **ImageWidth, ImageHeight** Size of the image area to be returned by GrabImage etc.
- **StartRow, StartColumn** StartRow, StartColumn upper left corner of the desired image area.
- **Field** Desired half image (‘first’, ‘second’, or ‘next’) or selection of a full image.
- **BitsPerChannel** Number of bits, which are transferred from the frame grabber board per pixel and image channel (typically 5, 8, 10, 12, or 16 Bits).
- **ColorSpace** Specify to grab single-channel (‘gray’) or three-channel images (‘rgb’, ‘yu’, ‘...’).
- **Gain** Amplification factor for the video amplifier (if available).
- **ExternalTrigger** Activation of external triggering (if available).
- **CameraType** Frame grabber specific parameter (string type) to specify the type of the used camera, which can be inquired via InfoFramegrabber.
- **Device** Device name of the frame grabber card.
- **Port** Port of the frame grabber the video signal is connected to.
- **LineIn** Camera input line of multiplexer (if available).

The operator OpenFramegrabber returns a handle (FGHandle) to the opened frame grabber.

---

**Attention**

Due to the multitude of supported frame grabbers a large number of parameters is necessary for the operator OpenFramegrabber. However, not all parameters are needed for a specific frame grabber.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>File</td>
</tr>
</tbody>
</table>

HALCON frame grabber interface, i.e. name of the corresponding DLL (WindowsNT) or shared library (UNIX).

**Default Value :** File

5.6. FRAMEGRABBER

- **HorizontalResolution** (input control) .................................................. `extent.x` ~ `long`
  Desired horizontal resolution of frame grabber (1: full resolution, 2: half resolution, 4: quarter resolution).
  Default Value: 1
  Value Suggestions: `HorizontalResolution` ∈ {1, 2, 4, 768, 720, 640, 384, 320, 192, 160}

- **VerticalResolution** (input control) .................................................. `extent.y` ~ `long`
  Desired vertical resolution of frame grabber (1: full resolution, 2: half resolution, 4: quarter resolution).
  Default Value: 1
  Value Suggestions: `VerticalResolution` ∈ {1, 2, 4, 576, 480, 288, 240, 144, 120}

- **ImageWidth** (input control) .......................................................... `rectangle.extent.x` ~ `long`
  Width of desired image part (0: `HorizontalResolution` - 2*StartColumn).
  Default Value: 0
  Value Suggestions: `ImageWidth` ∈ {-1, 0}

- **ImageHeight** (input control) .................................................. `rectangle.extent.y` ~ `long`
  Height of desired image part (0: `VerticalResolution` - 2*StartRow).
  Default Value: 0
  Value Suggestions: `ImageHeight` ∈ {-1, 0}

- **StartRow** (input control) .......................................................... `rectangle.origin.y` ~ `long`
  Line number of upper left corner of desired image part, for `ImageHeight = 0`: Height of a border.
  Default Value: 0
  Value Suggestions: `StartRow` ∈ {-1, 0}

- **StartColumn** (input control) .................................................. `rectangle.origin.x` ~ `long`
  Column number of upper left corner of desired image part, for `ImageWidth = 0`: Width of a border.
  Default Value: 0
  Value Suggestions: `StartColumn` ∈ {-1, 0}

- **Field** (input control) .......................................................... `string` ~ `String`
  Desired half image or for full image.
  Default Value: 'default'
  Value Suggestions: `Field` ∈ {'first', 'second', 'next', 'interlaced', 'progressive', 'default'}

- **BitsPerChannel** (input control) .................................................. `integer(-array)` ~ `VARIANT (integer)`
  Number of transferred bits per pixel and image channel (-1: frame grabber specific default value).
  Default Value: -1
  Value Suggestions: `BitsPerChannel` ∈ {5, 8, 10, 12, 16, -1}

- **ColorSpace** (input control) .................................................. `string(-array)` ~ `VARIANT (string)`
  Specify to grab single-channel ('gray') or three-channel ('rgb', 'yuv', ...) images ('default': frame grabber specific default value, mostly 'rgb').
  Default Value: 'default'
  Value Suggestions: `ColorSpace` ∈ {'gray', 'rgb', 'yuv', 'default'}

- **Gain** (input control) .......................................................... `real` ~ `double`
  Amplification factor for video amplifier (-1.0: frame grabber specific default value).
  Default Value: -1.0
  Value Suggestions: `Gain` ∈ {0.25, 0.5, 0.75, 1.0, -1.0}

- **ExternalTrigger** (input control) .................................................. `string` ~ `String`
  External triggering.
  Default Value: 'default'
  Value List: `ExternalTrigger` ∈ {'true', 'false', 'default'}

- **CameraType** (input control) .................................................. `string(-array)` ~ `VARIANT (string)`
  Type of used camera (frame grabber specific) ('default': frame grabber specific default value).
  Default Value: 'default'
  Value Suggestions: `CameraType` ∈ {'ntsc', 'pal', 'auto', 'default'}

- **Device** (input control) .................................................. `string(-array)` ~ `VARIANT (string)`
  Device the frame grabber is linked to ('default': frame grabber specific default value).
  Default Value: 'default'
  Value Suggestions: `Device` ∈ {'-1', '0', '1', '2', '3', 'default'}

- **Port** (input control) .................................................. `integer(-array)` ~ `VARIANT (integer)`
  Port of the frame grabber the video signal is linked to (-1: frame grabber specific default value).
  Default Value: -1
  Value Suggestions: `Port` ∈ {0, 1, 2, 3, -1}

HALCON 6.0
LineIn (input control) ................................................. integer(-array)  \sim  VARIANT ( integer )
Camera input line of multiplexer.

**Default Value:** -1

**Value Suggestions:** LineIn \in \{1, 2, 3, 4, -1\}

FGHandle (output control) ........................................ framegrabber  \sim  HFramegrabberX / long
Handle of the opened frame grabber.

--- Example ---

```c
// Select a suitable frame grabber FgName
info_framegrabber(FgName,'port',Information,Val)
// Choose the port P and the input line L your camera is connected to
open_framegrabber(FgName,1,1,0,0,0,0,'default',-1,'default',-1.0,
   'default','default','default',P,L,FGHandle)
grab_image(Img,FGHandle)
close_framegrabber(FGHandle).
```

--- Result ---

If the parameter values are correct and the desired frame grabber is available at call time, OpenFramegrabber returns the value TRUE. Otherwise an exception handling is raised.

--- Parallelization Information ---

OpenFramegrabber is *local* and processed *completely exclusively* without parallelization.

--- Possible Predecessor Functions ---

InfoFramegrabber

--- Possible Successor Functions ---

GrabImage, GrabRegion, GrabImageStart, GrabImageAsync, GrabRegionAsync, SetFramegrabberParam

--- See Also ---

InfoFramegrabber, CloseFramegrabber, GrabImage

--- Module ---

Image / region / XLD management

---

void HFramegrabberX.SetFramegrabberLut ([in] VARIANT ImageRed,
   [in] VARIANT ImageGreen, [in] VARIANT ImageBlue )

void HOperatorSetX.SetFramegrabberLut ([in] long FGHandle,

Set frame grabber lut.
The operator SetFramegrabberLut sets the lut of the frame grabbers specified by FGHandle. This operation is not supported for all kinds of frame grabbers.

--- Parameter ---

FGHandle (input control) ........................................ Handle of the desired frame grabber.

ImageRed (input control) ......................................... integer  \sim  VARIANT ( integer )
Red level of the lut entries.

ImageGreen (input control) ........................................ integer  \sim  VARIANT ( integer )
Green level of the lut entries.

ImageBlue (input control) ......................................... integer  \sim  VARIANT ( integer )
Blue level of the lut entries.

--- Result ---

The operator SetFramegrabberLut returns the value TRUE if the transferred lut is correct and the frame grabber is open.

--- Parallelization Information ---

SetFramegrabberLut is *reentrant* and processed *without* parallelization.
5.7. MANIPULATION

### Possible Predecessor Functions

OpenFramegrabber, GetFramegrabberLut

### Possible Successor Functions

GrabImage, GrabRegion, GrabImageStart, GrabImageAsync, GrabRegionAsync

See Also

GetFramegrabberLut, OpenFramegrabber

Module

Image / region / XLD management

```c
void HFramegrabberX.SetFramegrabberParam ([in] VARIANT Param,
[in] VARIANT Value )

void HOperatorSetX.SetFramegrabberParam ([in] long FGHandle,
```

Set specific parameters for a frame grabber.

The operator `SetFramegrabberParam` sets specific parameters for the frame grabber specified by `FGHandle`.

#### Attention

- **FGHandle** (input control) .............................................................. framegrabber ~ HFramegrabberX / long Handle of the frame grabber to be used.
- **Param** (input control) ......................................................... string(-array) ~ VARIANT ( string ) Parameter to be set.
- **Value** (input control) ............................................................ string(-array) ~ VARIANT ( string, real, integer ) Parameter value.

#### Result

If the frame grabber is open and the specified parameter / parameter value is supported by the frame grabber, the operator `SetFramegrabberParam` returns the value TRUE. Otherwise an exception handling is raised.

#### Parallelization Information

`SetFramegrabberParam` is reentrant and processed without parallelization.

### Possible Predecessor Functions

OpenFramegrabber

### Possible Successor Functions

GrabImage, GrabRegion, GrabImageStart, GrabImageAsync, GrabRegionAsync, CloseFramegrabber

See Also

OpenFramegrabber, InfoFramegrabber, GetFramegrabberParam

Module

Image / region / XLD management

---

5.7 Manipulation

```c
[out] VARIANT Grayval HImageX.GetGrayval ([in] VARIANT Row,
[in] VARIANT Column )

void HOperatorSetX.GetGrayval ([in] IObjectX Image, [in] VARIANT Row,
[in] VARIANT Column, [out] VARIANT Grayval )
```

Access the gray values of an image object.

HALCON 6.0
The operator Grayval is a tuple of floating point numbers, integer respectively, which returns the gray values of several pixels of Image. The line coordinates of the pixels are in the tuple Row, the columns in Column.

Attention

The type of the values of Grayval depends on the type of the gray values.

Gray values which do not belong to the image can also be accessed. The state of these gray values is not ascertained.

The operator GetGrayval involves a lot of work. It is not suitable for programming image processing operations such as filters. In this case it is more useful to use the procedure GetImagePointer1 or to directly use the C interface for integrating own procedures.

Parameter

- **Image** (input iconic)  
  Image whose gray value is to be accessed.

- **Row** (input control)  
  Line numbers of pixels to be viewed.
  
  Default Value: 0  
  Value Suggestions: Row ∈ {0, 64, 128, 256, 512, 1024}  
  Typical Range of Values: 0 ≤ Row ≤ 0(lin)  
  Minimal Value Step: 1  
  Recommended Value Step: 1  
  Restriction: (Row ∧ (Image < height))

- **Column** (input control)  
  Column numbers of pixels to be viewed.
  
  Default Value: 0  
  Value Suggestions: Column ∈ {0, 64, 128, 256, 512, 1024}  
  Typical Range of Values: 0 ≤ Column ≤ 0(lin)  
  Minimal Value Step: 1  
  Recommended Value Step: 1  
  Restriction: (Column ∧ (Image < width))  
  Parameter Number: (Column = Row)

- **Grayval** (output control)  
  Gray values of indicated pixels.
  
  Parameter Number: (Grayval = Row)

Result

If the state of the parameters is correct the operator GetGrayval returns the value TRUE.

The behavior in case of empty input (no input images available) is set via the operator SetSystem('noObjectResult',<Result>). If necessary an exception handling is raised.

Parallelization Information

GetGrayval is reentrant and processed without parallelization.

Possible Predecessor Functions

ReadImage

See Also

SetGrayval

Alternatives

GetImagePointer1

Module

Image / region / XLD management

The operator GetImagePointer1 returns a C pointer to the first channel of the image Image. Additionally the image type (Type = ´byte´, ´integer´, ´float´ etc.) and the image size (width and height) are returned. Consequently
a direct access to the image data in the HALCON databank from the HALCON host language via the pointer is possible. An image is stored in HALCON as a vector of image lines. The operator GetImagePointer1 is only of interest for HALCON/C.

Attention

The operator GetImagePointer1 should only be used for entry into newly created images, since otherwise the gray values of other images might be overwritten (see relational structure).

Parameter

- **Image** (input iconic) ................................................................. image $\sim$ HImageX / IHopeX
  Input image.
- **Pointer** (output control) .......................................................... pointer $\sim$ long
  Pointer to the image data in the HALCON databank.
- **Type** (output control) .......................................................... string $\sim$ long
  Type of image.
  Value List: Type $\in$ \{'int1', 'int2', 'int4', 'byte', 'real', 'direction', 'cyclic', 'complex', 'dvf', 'lut'\}
- **Width** (output control) .......................................................... extent.x $\sim$ long
  Width of image.
- **Height** (output control) ....................................................... extent.y $\sim$ long
  Height of image.

Result

The operator GetImagePointer1 returns the value TRUE if exactly one image was passed. The behavior in case of empty input (no input images available) is set via the operator SetSystem('noObjectResult',<Result>). If necessary an exception handling is raised.

Parallelization Information

GetImagePointer1 is reentrant and processed without parallelization.

Possible Predecessor Functions

ReadImage

See Also

PaintRegion, PaintGray

Alternatives

SetGrayval, GetGrayval, GetImagePointer3

Module

Image / region / XLD management

```
[out] long PixelPointer HImageX.GetImagePointer1Rect
([out] long Width, [out] long Height, [out] long VerticalPitch )

void HOperatorSetX.GetImagePointer1Rect ([in] IHopeX Image,
[out] long PixelPointer, [out] long Width, [out] long Height,
[out] long VerticalPitch )
```

Access to the image data pointer and the image data inside the smallest rectangle of the domain of the input image. The operator GetImagePointer1Rect returns the pointer PixelPointer which points to the beginning of the image data inside the smallest rectangle of the domain of Image. The difference between the width of Image and the width of the smallest rectangle is returned by VerticalPitch. Width and Height correspond to the size of the smallest rectangle of the input region. GetImagePointer1Rect is approximately symmetrical to GenImage1Rect.

Attention

The operator GetImagePointer1Rect should only be used for entry into newly created images, since otherwise the gray values of other images might be overwritten (see relational structure).

Parameter

- **Image** (input iconic) ................................................................. image $\sim$ HImageX / IHopeX
  Input image (Himage).
PixelPointer (output control) ..................................................... pointer ~ long
Pointer to the image data.

Width (output control) .......................................................... extent.x ~ long
Width of the output image.

Height (output control) ....................................................... extent.y ~ long
Height of the output image.

VerticalPitch (output control) ............................................ integer ~ long
Difference between the width of the smallest rectangle of the domain of the input image and the width of the input image.

The operator GetImagePointer1Rect returns the value TRUE if exactly one image (byte) was passed. The behavior in case of empty input (no input images available) is set via the operator SetSystem('noObjectResult',<Result>). If necessary an exception handling is raised.

Parallelization Information
GetImagePointer1Rect is reentrant and processed without parallelization.

Possible Predecessor Functions
ReadImage

See Also
PaintRegion, PaintGray, GenImage1Rect

Alternatives
SetGrayval, GetGrayval, GetImagePointer3, GetImagePointer1

Module
Image / region / XLD management

Access the pointers of a colored image.
The operator GetImagePointer3 returns a C pointer to the three channels of a colored image (ImageRGB). Additionally the image type (Type = 'byte', 'int2', 'int4', 'float' etc.) and the image size (Width and Height) are returned. Consequently a direct access to the image data in the HALCON databank from the HALCON host language via the pointer is possible. An image is stored in HALCON as a vector of image lines. The three channels must have the same pixel type and the same size. The operator GetImagePointer3 is only of interest for HALCON/C.

Attention
Only one image can be passed. The operator GetImagePointer3 should only be used for entry into newly created images, since otherwise the gray values of other images might be overwritten (see relational structure).

Parameter

- ImageRGB (input iconic) ................................................. image ~ HImageX / IHObjectX
Input image.
- PointerRed (output control) ........................................... pointer ~ long
Pointer to the pixels of the first channel.
- PointerGreen (output control) ....................................... pointer ~ long
Pointer to the pixels of the second channel.
- PointerBlue (output control) .......................................... pointer ~ long
Pointer to the pixels of the third channel.
- Type (output control) .................................................. string ~ String
Type of image.

Value List: Type ∈ {'int1', 'int2', 'int4', 'byte', 'real', 'direction', 'cyclic', 'complex', 'dvf', 'lut'}
5.7. MANIPULATION

The operator GetImagePointer3 returns the value TRUE if exactly one image is passed. The behavior in case of empty input (no input images available) is set via the operator SetSystem(‘noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information
GetImagePointer3 is reentrant and processed without parallelization.

Possible Predecessor Functions

ReadImage

PaintRegion, PaintGray

Alternatives
SetGrayval, GetGrayval, GetImagePointer1

Module
Image / region / XLD management

Copy the gray values of an image into another image.

PaintGray copies the gray values of the images given in ImageSource into the image in ImageDestination. Only the gray values of the domain of ImageSource are copied (see ReduceDomain).

Parameter

- **ImageSource** (input iconic) ................................................... image \( \sim \) HImageX / IHObjectX
  Input image containing the desired gray values.
- **ImageDestination** (input iconic) ........................................... image \( \sim \) HImageX / IHObjectX
  Input image to be painted over.
- **MixedImage** (output iconic) .............................................. image \( \sim \) HImageX / HUntypedObjectX
  Result image.

Example

/* Copy of a segment (circle) of the image ‘affe’ into a new image (New): */

read_image(Image,’affe’)
gen_circle(Circle,200,200,150)
reduce_domain(Image,Circle,Mask)
/* New image with black (0) background */
gen_image_proto(Image,New1,0.0)
/* Copy of a segment of the image ‘affe’ into New1 */
paint_gray(Mask,New1,New).

Result

PaintGray returns TRUE if all parameters are correct. If necessary, an exception is raised.

Parallelization Information

PaintGray is reentrant and processed without parallelization.
CHAPTER 5. IMAGE

Possible Predecessor Functions

ReadImage, GenImageConst, GenImageProto

See Also

PaintRegion

Alternatives

GetImagePointer1, SetGrayval, CopyImage

Module

Basic operators

```
```

Paint a region in an image with a constant gray value.

PaintRegion paints the regions given in Region with a constant gray value into the image given in Image. The parameter Type determines whether the region should be painted filled ("fill") or whether only its boundary should be painted ("margin"). The resulting image is returned in ImageResult.

Attention

PaintRegion should only be used to paint into newly created images (GenImageConst), because otherwise the gray values of other existing images may be overwritten.

Parameter

- **Region** (input iconic) ................. region(-array) \( \sim \) HRegionX / IObjectX Regions to be painted into the input image.
- **Image** (input iconic) .................. image \( \sim \) HImageX / IObjectX Image in which the regions are to be painted.
- **ImageResult** (output iconic) .......... image \( \sim \) HImageX / HUntypedObjectX Image containing the result.
- **Grayval** (input control) ............... number \( \sim \) VARIANT (integer, real) Desired gray value of the region.
  - Default Value: 255.0
  - Value Suggestions: Grayval \( \in \{ 0.0, 1.0, 2.0, 5.0, 10.0, 16.0, 32.0, 64.0, 128.0, 253.0, 254.0, 255.0 \} \)
  - Typical Range of Values: \( 0.0 \leq \) Grayval \( \leq 0.0 \)
- **Type** (input control) ................. string \( \sim \) String Paint regions filled or as boundaries.
  - Default Value: 'fill'
  - Value List: Type \( \in \{ 'fill', 'margin' \} \)

Example

```c
/* Copy of a rectangle in a new image (New) */

read_image(Image,'affe')
gen_rectangle1(Rectangle,100.0,100.0,300.0,300.0)
reduce_domain(Image,Rectangle,Mask)
/* generate a black image */
gen_image_proto(Image,New1,0.0)
/* copy a white rectangle */
paint_region(Mask,New1,New,255.0,'fill',).
```

Result

PaintRegion returns TRUE if all parameters are correct. If the input is empty the behavior can be set via SetSystem(:'noObjectResult',<Result>). If necessary, an exception is raised.
5.7. MANIPULATION

Parallelization Information

PaintRegion is reentrant and processed without parallelization.

Possible Predecessor Functions

ReadImage, GenImageConst, GenImageProto, ReduceDomain

See Also

ReduceDomain, SetDraw, GenImageConst

Alternatives

SetGrayval, PaintGray

Module

Basic operators

```c
```

Set single gray values in an image.

SetGrayval sets the gray values of the input image Image at the positions (Row,Column) to the values specified by Grayval. The number of values in Grayval must match the number of points passed to the operator.

Parameter

- **Image** (input iconic) ................................. image ↦ HImageX/IHObjectX Image to be modified.
- **Row** (input control) ................................. point.y(-array) ↦ VARIANT (integer) Row coordinates of the pixels to be modified.
  - Default Value : 0
  - Value Suggestions : Row ∈ {0, 10, 50, 127, 255, 511}
  - Restriction : (Row ∧ (Image < height))
- **Column** (input control) ............................... point.x(-array) ↦ VARIANT (integer) Column coordinates of the pixels to be modified.
  - Default Value : 0
  - Value Suggestions : Column ∈ {0, 10, 50, 127, 255, 511}
  - Restriction : (Column ∧ (Image < width))
- **Grayval** (input control) ............................... grayval(-array) ↦ VARIANT (integer, real) Gray values to be used.
  - Default Value : 255.0
  - Value Suggestions : Grayval ∈ {0.0, 1.0, 10.0, 128.0, 255.0}

Result

SetGrayval returns TRUE if all parameters are correct. If the input is empty the behavior can be set via SetSystem(ː‘noObjectResult’,<Result>:). If necessary, an exception is raised.

Parallelization Information

SetGrayval is reentrant and processed without parallelization.

Possible Predecessor Functions

ReadImage, GetImagePointer1, GenImageProto, GenImage1

See Also

GetGrayval, GenImageConst, GenImage1, GenImageProto

Alternatives

GetImagePointer1, PaintGray, PaintRegion

Module

Basic operators
5.8 Type-Conversion

Convert a complex image into two real images.

ComplexToReal converts a complex image \texttt{ImageComplex} into two real images \texttt{ImageReal} and \texttt{ImageImaginary}, which contain the real and imaginary part of the complex image.

---

**Parameter**

- \texttt{ImageComplex} (input iconic) \hspace{1em} \texttt{IHImageX} / \texttt{IHObjectX} (complex) Complex image.
- \texttt{ImageReal} (output iconic) \hspace{1em} \texttt{IHImageX} / \texttt{HUntypedObjectX} (real) Real part.
- \texttt{ImageImaginary} (output iconic) \hspace{1em} \texttt{IHImageX} / \texttt{HUntypedObjectX} (real) Imaginary part.

---

**Parallelization Information**

\texttt{ComplexToReal} is reentrant and automatically parallelized (on tuple level).

---

**See Also**

RealToComplex

---

**Module**

Image / region / XLD management

---

Convert the type of an image.

ConvertImageType converts images of an arbitrary type into an arbitrary new image type. If the conversion is done from a larger to a smaller gray value range (e.g., from ‘int4’ to ‘byte’), too large or too small values are simply “clipped.” It is therefore advisable to adapt the range of gray values by calling \texttt{ScaleImage} before calling this operator.

---

**Attention**

If the source and destination image type are identical, no new image matrix is allocated.

---

**Parameter**

- \texttt{Image} (input iconic) \hspace{1em} \texttt{IHImageX} / \texttt{IHObjectX} (any) Image whose image type is to be changed.
- \texttt{ImageConverted} (output iconic) \hspace{1em} \texttt{IHImageX} / \texttt{HUntypedObjectX} (any) Converted image.
- \texttt{NewType} (input control) \hspace{1em} \texttt{String} Desired image type (i.e., type of the gray values).

**Default Value**: ‘byte’

**Value List**: \texttt{NewType} $\in \{ \text{'int1'}, \text{'int2'}, \text{'int4'}, \text{'byte'}, \text{'real'}, \text{'direction'}, \text{'cyclic'}, \text{'complex'}, \text{'dvf'}, \text{'lut'} \}$

---

**Result**

ConvertImageType returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via \texttt{SetSystem('noObjectResult',<Result>)}. If necessary, an exception is raised.

---

**Parallelization Information**

ConvertImageType is reentrant and automatically parallelized (on tuple level, channel level, domain level).
5.8. TYPE-CONVERSION

---

**Possible Predecessor Functions**

ScaleImage

---

**See Also**

ScaleImage, AbsImage

---

**Module**

Image / region / XLD management

---

```c
[out] HImageX Row HImageX.DvfToInt1 ([out] HImageX Col )
```

```c
void HOperatorSetX.DvfToInt1 ([in] IHObjectX VectorField, [out] HUntypedObjectX Row, [out] HUntypedObjectX Col )
```

*Convert a displacement vector field into two int1-images.*

DvfToInt1 converts the displacement vector field `VectorField` into two int1-images `Row` and `Col`. The output images contain the displacements in `x`- and `y`-direction, respectively.

---

**Parameter**

- `VectorField` (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) ∼ `HImageX / IHObjectX (dvf)` Displacement vector field.
- `Row` (output iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) ∼ `HImageX / HUntypedObjectX (int1)` Displacement along `y`.
- `Col` (output iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) ∼ `HImageX / HUntypedObjectX (int1)` Displacement along `x`.

---

**Parallelization Information**

DvfToInt1 is *reentrant* and automatically *parallelized* (on tuple level).

---

**Possible Predecessor Functions**

MeanImage, OpticalFlowMatch, FillDvf

---

**Module**

Image / region / XLD management

---

```c
[out] HImageX VectorField HImageX.Int1ToDvf ([in] HImageX Col )
```

```c
void HOperatorSetX.Int1ToDvf ([in] IHObjectX Row, [in] IHObjectX Col, [out] HUntypedObjectX VectorField )
```

*Convert two int1-images into a displacement vector field.*

Int1ToDvf converts two int1-images `Row` and `Col` into a displacement vector field `VectorField`. The input images contain the displacements in `x`- and `y`-direction, respectively.

---

**Parameter**

- `Row` (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) ∼ `HImageX / IHObjectX (int1)` Displacement along `y`.
- `Col` (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) ∼ `HImageX / IHObjectX (int1)` Displacement along `x`.
- `VectorField` (output iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) ∼ `HImageX / HUntypedObjectX (dvf)` Displacement vector field.

---

**Parallelization Information**

Int1ToDvf is *reentrant* and automatically *parallelized* (on tuple level).

---

**Possible Predecessor Functions**

DvfToInt1

---

HALCON 6.0
Possible Successor Functions

OpticalFlowMatch, FillDvf

Module

Image / region / XLD management

```
[out] HImageX ImageComplex HImageX.RealToComplex
([in] HImageX ImageImaginary )

void HOperatorSetX.RealToComplex ([in] IHObjectX ImageReal,
```

Convert two real images into a complex image.

RealToComplex converts two real images ImageReal and ImageImaginary, which contain the real and imaginary part of a complex image, into a complex image ImageComplex.

Parameter

- **ImageReal** (input iconic) image(-array) $\sim$ HImageX / IHObjectX (real )
  Real part.
- **ImageImaginary** (input iconic) image(-array) $\sim$ HImageX / IHObjectX (real )
  Imaginary part.
- **ImageComplex** (output iconic) image(-array) $\sim$ HImageX / HUntypedObjectX (complex )
  Complex image.

Parallelization Information

RealToComplex is reentrant and automatically parallelized (on tuple level).

See Also

ComplexToReal

Module

Image / region / XLD management
Chapter 6

Lines

6.1 Access

Approximate a contour by arcs and lines.

The coordinates of a curve are approximated by a row of lines and arcs. The procedure tries values from a user-definable range for certain parameters. The limits of these ranges are explicitly stated in the parameter list of the function (MinWidthCoord ... MaxWidthCoord, ThreshStart ... ThreshEnd, MinWidthSmooth ... MaxWidthSmooth, MinWidthCurve ... MaxWidthCurve). Additionally, the step width for the parameter area of the threshold value for pointed corners has to be indicated (ThreshStep). By narrowing the covered areas the runtime of the calculation can be shortened, but the result may deteriorate.

The parameters Weight1, Weight2 and Weight3 indicate whether the desired weighting is placed more on precision of the approximation, obtaining as much large segments as possible or as few small segments as possible. Thus, for (Weight1,Weight2,Weight3) (1,0,0) creates a very precise approximation and (0,1,1) an approximation with as few large segments as possible.

The result of the procedure is returned separately as arcs and lines. If one is interested in the sequence of the segments the individual resulting elements can be read successively from the resulting tuples; the sequence can be taken from the return parameter order (0: next element is next line segment, 1: next element is next arc segment).

Attention

Contours which can possibly consist of only one segment should also be examined with a threshold maximum (ThreshEnd) > 1.0, because otherwise at least one “corner point” is determined in any case.
Parameter

- **Row** (input control) ................................. \( point.y \sim \text{long} \)
  Row of the contour.
  Default Value: 32

- **Column** (input control) ................................. \( point.x \sim \text{long} \)
  Column of the contour.
  Default Value: 32

- **MinWidthCoord** (input control) .......................... \( \text{real} \sim \text{double} \)
  Minimum width of Gauss operator for coordinate smoothing (> 0.4).
  Default Value: 0.5
  Value Suggestions: \( \text{MinWidthCoord} \in \{0.5, 0.7, 1.0, 1.2, 1.5, 1.7\} \)
  Typical Range of Values: \( 0.4 \leq \text{MinWidthCoord} \leq 0.4(\text{lin}) \)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.1

- **MaxWidthCoord** (input control) .......................... \( \text{real} \sim \text{double} \)
  Maximum width of Gauss operator for coordinate smoothing (> 0.4).
  Default Value: 2.4
  Value Suggestions: \( \text{MaxWidthCoord} \in \{0.5, 0.7, 1.0, 1.2, 1.5, 1.7\} \)
  Typical Range of Values: \( 0.4 \leq \text{MaxWidthCoord} \leq 0.4(\text{lin}) \)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.1

- **ThreshStart** (input control) ............................ \( \text{real} \sim \text{double} \)
  Minimum threshold value of the curvature for accepting a corner (relative to the largest curvature present).
  Default Value: 0.3
  Value Suggestions: \( \text{ThreshStart} \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\} \)
  Typical Range of Values: \( 0.1 \leq \text{ThreshStart} \leq 0.1(\text{lin}) \)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.1

- **ThreshEnd** (input control) ............................. \( \text{real} \sim \text{double} \)
  Maximum threshold value of the curvature for accepting a corner (relative to the largest curvature present).
  Default Value: 0.9
  Value Suggestions: \( \text{ThreshEnd} \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\} \)
  Typical Range of Values: \( 0.1 \leq \text{ThreshEnd} \leq 0.1(\text{lin}) \)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.1

- **ThreshStep** (input control) ............................ \( \text{real} \sim \text{double} \)
  Step width for threshold increase.
  Default Value: 0.2
  Value Suggestions: \( \text{ThreshStep} \in \{0.3, 0.4, 0.5\} \)
  Typical Range of Values: \( 0.1 \leq \text{ThreshStep} \leq 0.1(\text{lin}) \)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.1

- **MinWidthSmooth** (input control) ........................ \( \text{real} \sim \text{double} \)
  Minimum width of Gauss operator for smoothing the curvature function (> 0.4).
  Default Value: 0.5
  Value Suggestions: \( \text{MinWidthSmooth} \in \{0.5, 0.7, 1.0, 1.2, 1.5, 1.7\} \)
  Typical Range of Values: \( 0.4 \leq \text{MinWidthSmooth} \leq 0.4(\text{lin}) \)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.1

- **MaxWidthSmooth** (input control) ........................ \( \text{real} \sim \text{double} \)
  Maximum width of Gauss operator for smoothing the curvature function.
  Default Value: 2.4
  Value Suggestions: \( \text{MaxWidthSmooth} \in \{0.5, 0.7, 1.0, 1.2, 1.5, 1.7\} \)
  Typical Range of Values: \( 0.4 \leq \text{MaxWidthSmooth} \leq 0.4(\text{lin}) \)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.1
6.1. ACCESS

- **MinWidthCurve** (input control) .......................................................... integer \( \sim \) long
  Minimum width of curve area for curvature determination (\( \geq 0.4 \)).
  
  Default Value: 2
  Value Suggestions: MinWidthCurve \( \in \{2, 5, 7\}\)
  Typical Range of Values: \( 1 \leq \text{MinWidthCurve} \leq 1\text{lin} \)
  Minimal Value Step: 1
  Recommended Value Step: 2

- **MaxWidthCurve** (input control) .......................................................... integer \( \sim \) long
  Maximum width of curve area for curvature determination.
  
  Default Value: 12
  Value Suggestions: MaxWidthCurve \( \in \{2, 5, 7\}\)
  Typical Range of Values: \( 1 \leq \text{MaxWidthCurve} \leq 1\text{lin} \)
  Minimal Value Step: 1
  Recommended Value Step: 2

- **Weight1** (input control) ................................................................. real \( \sim \) double
  Weighting factor for approximation precision.
  
  Default Value: 1.0
  Value Suggestions: Weight1 \( \in \{0.0, 0.5, 1.0\}\)
  Typical Range of Values: \( 0.0 \leq \text{Weight1} \leq 0.0\text{lin} \)
  Minimal Value Step: 0.1
  Recommended Value Step: 0.5

- **Weight2** (input control) ................................................................. real \( \sim \) double
  Weighting factor for large segments.
  
  Default Value: 1.0
  Value Suggestions: Weight2 \( \in \{0.0, 0.5, 1.0\}\)
  Typical Range of Values: \( 0.0 \leq \text{Weight2} \leq 0.0\text{lin} \)
  Minimal Value Step: 0.1
  Recommended Value Step: 0.5

- **Weight3** (input control) ................................................................. real \( \sim \) double
  Weighting factor for small segments.
  
  Default Value: 1.0
  Value Suggestions: Weight3 \( \in \{0.0, 0.5, 1.0\}\)
  Typical Range of Values: \( 0.0 \leq \text{Weight3} \leq 0.0\text{lin} \)
  Minimal Value Step: 0.1
  Recommended Value Step: 0.5

- **ArcCenterRow** (output control) ....................................................... arc.center.y \( \sim \) VARIANT( integer )
  Row of the center of an arc.

- **ArcCenterCol** (output control) ....................................................... arc.center.x \( \sim \) VARIANT( integer )
  Column of the center of an arc.

- **ArcAngle** (output control) ............................................................. arc.angle.rad \( \sim \) VARIANT( real )
  Angle of an arc.

- **ArcBeginRow** (output control) ....................................................... arc.begin.y \( \sim \) VARIANT( integer )
  Row of the starting point of an arc.

- **ArcBeginCol** (output control) ....................................................... arc.begin.x \( \sim \) VARIANT( integer )
  Column of the starting point of an arc.

- **LineBeginRow** (output control) ....................................................... line.begin.y \( \sim \) VARIANT( integer )
  Row of the starting point of a line segment.

- **LineBeginCol** (output control) ....................................................... line.begin.x \( \sim \) VARIANT( integer )
  Column of the starting point of a line segment.

- **LineEndRow** (output control) ......................................................... line.end.y \( \sim \) VARIANT( integer )
  Row of the ending point of a line segment.

- **LineEndCol** (output control) .......................................................... line.end.x \( \sim \) VARIANT( integer )
  Column of the ending point of a line segment.

- **Order** (output control) ................................................................. integer \( \sim \) VARIANT( integer )
  Sequence of line (value 0) and arc segments (value 1).

---

Result

The operator ApproxChain returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

HALCON 6.0
ApproxChain is reentrant and processed without parallelization.

Possible Predecessor Functions

SobelAmp, EdgesImage, GetRegionContour, Threshold, HysteresisThreshold

Possible Successor Functions

SetLineWidth, DispArc, DispLine

See Also

GetRegionChain, SmallestCircle, DispCircle, DispLine

Alternatives

GetRegionPolygon, ApproxChainSimple

Module

Region processing

Approximate a contour by arcs and lines.

The contour of a curve is approximated by a sequence of lines and arcs. If one is interested in the sequence of the segments the individual resulting elements can be read successively from the resulting tuple. The sequence can be taken from the return parameter order (0: next element is next line segment, 1: next element is next arc segment).

The operator ApproxChainSimple behaves similarly as ApproxChain except that in the case of ApproxChainSimple the missing parameters are internally allocated as follows: MinWidthCoord = 1.0, MaxWidthCoord = 3.0, ThreshStart = 0.5, ThreshEnd = 0.9, ThreshStep = 0.3, MinWidthSmooth = 1.0, MaxWidthSmooth = 3.0, MinWidthCurve = 2, MaxWidthCurve = 9, Weight1 = 1.0, Weight2 = 1.0, Weight3 = 1.0.

Parameter

- Row (input control) ................................................................. point.y ∼ long
  Row of the contour.
  Default Value: 32

- Column (input control) ........................................................... point.x ∼ long
  Column of the contour.
  Default Value: 32

- ArcCenterRow (output control) .......................................... arc.center.y ∼ VARIANT (integer)
  Row of the center of an arc.

- ArcCenterCol (output control) ........................................... arc.center.x ∼ VARIANT (integer)
  Column of the center of an arc.

- ArcAngle (output control) ................................................. arc.angle.rad ∼ VARIANT (real)
  Angle of an arc.

- ArcBeginRow (output control) ........................................... arc.begin.y ∼ VARIANT (integer)
  Row of the starting point of an arc.

- ArcBeginCol (output control) .......................................... arc.begin.x ∼ VARIANT (integer)
  Column of the starting point of an arc.

- LineBeginRow (output control) ..................................... line.begin.y ∼ VARIANT (integer)
  Row of the starting point of a line segment.
6.2. FEATURES

The operator **ApproxChainSimple** returns the value **TRUE** if the parameters are correct. Otherwise an exception is raised.

---

**Parallelization Information**
**ApproxChainSimple** is **reentrant** and processed **without parallelization**.

---

**Possible Predecessor Functions**
SobelAmp, EdgesImage, GetRegionContour, Threshold, HysteresisThreshold

---

**Possible Successor Functions**
SetLineWidth, DispArc, DispLine

---

**See Also**
GetRegionChain, SmallestCircle, DispCircle, DispLine

---

**Alternatives**
GetRegionPolygon, ApproxChain

---

**Module**
Region processing

6.2 Features

```haskell

```

Calculate the orientation of lines.

The operator **LineOrientation** returns the orientation \((-\pi/2 < \Phi \leq \pi/2\)) of the given lines. If more than one line is to be treated the line and column indices can be passed as tuples. In this case **Phi** is, of course, also a tuple and contains the corresponding orientations.

The procedure is typically applied to model lines in order to select parallel image lines, which were found, e.g., by **DetectEdgeSegments**, via the operator **SelectLines**.

---

**Parameter**

- **RowBegin** (input control) \(\text{line.begin.y(-array)} \sim \text{VARIANT (integer, real)}\)
  Row coordinates of the starting points of the input lines.

- **ColBegin** (input control) \(\text{line.begin.x(-array)} \sim \text{VARIANT (integer, real)}\)
  Column coordinates of the starting points of the input lines.

- **RowEnd** (input control) \(\text{line.end.y(-array)} \sim \text{VARIANT (integer, real)}\)
  Row coordinates of the ending points of the input lines.

- **ColEnd** (input control) \(\text{line.end.x(-array)} \sim \text{VARIANT (integer, real)}\)
  Column coordinates of the ending points of the input lines.

- **Phi** (output control) \(\text{angle.rad(-array)} \sim \text{VARIANT (real)}\)
  Orientation of the input lines.
Result

LineOrientation always returns the value TRUE.

Parallelization Information

LineOrientation is reentrant and processed without parallelization.

Possible Predecessor Functions

SobelAmp, EdgesImage, Threshold, HysteresisThreshold, SplitSkeletonRegion, SplitSkeletonLines

Possible Successor Functions

SetLineWidth, DispLine

See Also

LinePosition, SelectLines, PartitionLines, DetectEdgeSegments

Alternatives

LinePosition, SelectLines, PartitionLines

Module

Region processing

```

```

Calculate the center of gravity, length, and orientation of a line.

The operator LinePosition returns the center (RowCenter, ColCenter), the (Euclidean) length (Length) and the orientation (-π/2 < Phi ≤ π/2) of the given lines. If more than one line is to be treated the line and column indices can be passed as tuples. In this case the output parameters, of course, are also tuples.

The routine is applied, for example, to model lines in order to determine search regions for the edge detection (DetectEdgeSegments).

Parameter

- **RowBegin** (input control) .................. line.begin.y(-array) ~ VARIANT( integer ) Row coordinates of the starting points of the input lines.
- **ColBegin** (input control) .................. line.begin.x(-array) ~ VARIANT( integer ) Column coordinates of the starting points of the input lines.
- **RowEnd** (input control) .................. line.end.y(-array) ~ VARIANT( integer ) Row coordinates of the ending points of the input lines.
- **ColEnd** (input control) .................. line.end.x(-array) ~ VARIANT( integer ) Column coordinates of the ending points of the input lines.
- **RowCenter** (output control) .................. point.y(-array) ~ VARIANT( real ) Row coordinates of the centers of gravity of the input lines.
- **ColCenter** (output control) .................. point.x(-array) ~ VARIANT( real ) Column coordinates of the centers of gravity of the input lines.
- **Length** (output control) .................. real(-array) ~ VARIANT( real ) Euclidean length of the input lines.
- **Phi** (output control) .................. angle.rad(-array) ~ VARIANT( real ) Orientation of the input lines.

Result

LinePosition always returns the value TRUE.

Parallelization Information

LinePosition is reentrant and processed without parallelization.
Possible Predecessor Functions
SobelAmp, EdgesImage, Threshold, HysteresisThreshold, SplitSkeletonRegion, SplitSkeletonLines

Possible Successor Functions
SetLineWidth, DispLine

See Also
LineOrientation, SelectLines, PartitionLines, DetectEdgeSegments

Alternatives
LineOrientation, SelectLines, PartitionLines

Module
Region processing

HALCON 6.0
Feature (input control) ................................. string(-array)  \rightarrow  VARIANT ( string )
Features to be used for selection.
Value List: Feature \in \{ ’length’, ’row’, ’column’, ’phi’ \}

Operation (input control) ................................. string  \rightarrow  String
Desired combination of the features.
Value List: Operation \in \{ ’and’, ’or’ \}

Min (input control) ................................. string(-array)  \rightarrow  VARIANT ( integer, real, string )
Lower limits of the features or ’min’.
Default Value: ’min’

Max (input control) ................................. string(-array)  \rightarrow  VARIANT ( integer, real, string )
Upper limits of the features or ’max’.
Default Value: ’max’

RowBeginOut (output control) ............................. line.begin.y  \rightarrow  VARIANT ( integer )
Row coordinates of the starting points of the lines fulfilling the conditions.

ColBeginOut (output control) ............................. line.begin.x  \rightarrow  VARIANT ( integer )
Column coordinates of the starting points of the lines fulfilling the conditions.

RowEndOut (output control) ............................. line.end.y  \rightarrow  VARIANT ( integer )
Row coordinates of the ending points of the lines fulfilling the conditions.

ColEndOut (output control) ............................. line.begin.x  \rightarrow  VARIANT ( integer )
Column coordinates of the ending points of the lines fulfilling the conditions.

FailRowBOut (output control) ............................. line.begin.y  \rightarrow  VARIANT ( integer )
Row coordinates of the starting points of the lines not fulfilling the conditions.

FailColBOut (output control) ............................. line.begin.x  \rightarrow  VARIANT ( integer )
Column coordinates of the starting points of the lines not fulfilling the conditions.

FailRowEOut (output control) ............................. line.end.x  \rightarrow  VARIANT ( integer )
Row coordinates of the ending points of the lines not fulfilling the conditions.

FailColEOut (output control) ............................. line.end.x  \rightarrow  VARIANT ( integer )
Column coordinates of the ending points of the lines not fulfilling the conditions.

The operator PartitionLines returns the value TRUE if the parameter values are correct. Otherwise an exception is raised.

--- Parallelization Information ---
PartitionLines is reentrant and processed without parallelization.

--- Possible Predecessor Functions ---
SobelAmp, EdgesImage, Threshold, HysteresisThreshold, SplitSkeletonRegion,
SplitSkeletonLines

--- Possible Successor Functions ---
SetLineWidth, DispLine

See Also
SelectLines, SelectLinesLongest, DetectEdgeSegments, SelectShape

--- Alternatives ---
LineOrientation, LinePosition, SelectLines, SelectLinesLongest

--- Module ---
Region processing
6.2. FEATURES

Select lines according to various criteria.

The operator SelectLines chooses lines according to various criteria. For every input line the indicated features (Feature) are calculated. If each (Operation = 'and') or at least one (Operation = 'or') of the calculated features is within the given limits (Min, Max) the line is transferred into the output.

Condition: \[ Min_i \leq Feature_i(\text{Line}) \leq Max_i \]

Possible values for Feature:

'length' (Euclidean) length of the line
'row' Line index of the center
'column' Column index of the center
'phi' Orientation of the line \((-\pi < \varphi \leq \pi)\)

Attention
If only one feature is used the value of Operation is meaningless. Several features are processed according to the sequence in which they are passed.

Parameter

- **RowBeginIn** (input control) ................................................. \text{line.begin.y} \sim \text{VARIANT (integer)}
  Row coordinates of the starting points of the input lines.

- **ColBeginIn** (input control) ................................................. \text{line.begin.x} \sim \text{VARIANT (integer)}
  Column coordinates of the starting points of the input lines.

- **RowEndIn** (input control) ................................................. \text{line.end.y} \sim \text{VARIANT (integer)}
  Row coordinates of the ending points of the input lines.

- **ColEndIn** (input control) ................................................. \text{line.end.x} \sim \text{VARIANT (integer)}
  Column coordinates of the ending points of the input lines.

- **Feature** (input control) .................................................. \text{string(-array)} \sim \text{VARIANT (string)}
  Features to be used for selection.
  Default Value: 'length'
  Value List: Feature \in \{ 'length', 'row', 'column', 'phi' \}

- **Operation** (input control) .................................................. \text{string} \sim \text{String}
  Desired combination of the features.
  Default Value: 'and'
  Value List: Operation \in \{ 'and', 'or' \}

- **Min** (input control) .................................................. \text{string(-array)} \sim \text{VARIANT (integer, real, string)}
  Lower limits of the features or 'min'.
  Default Value: 'min'

- **Max** (input control) .................................................. \text{string(-array)} \sim \text{VARIANT (integer, real, string)}
  Upper limits of the features or 'max'.
  Default Value: 'max'

- **RowBeginOut** (output control) ................................................. \text{line.begin.y} \sim \text{VARIANT (integer)}
  Row coordinates of the starting points of the output lines.

- **ColBeginOut** (output control) ................................................. \text{line.begin.x} \sim \text{VARIANT (integer)}
  Column coordinates of the starting points of the output lines.
The operator SelectLines returns the value TRUE if the parameter values are correct. Otherwise an exception is raised.

### Parallelization Information
SelectLines is reentrant and processed without parallelization.

### Possible Predecessor Functions
SobelAmp, EdgesImage, Threshold, HysteresisThreshold, SplitSkeletonRegion, SplitSkeletonLines

### Possible Successor Functions
SetLineWidth, DispLine

### See Also
PartitionLines, SelectLinesLongest, DetectEdgeSegments, SelectShape

### Alternatives
LineOrientation, LinePosition, PartitionLines

### Module
Region processing

The operator SelectLinesLongest selects the `Num` longest input lines from the input lines described by the tuples `RowBeginIn`, `ColBeginIn`, `RowEndIn` and `ColEndIn`.

### Parameter

- **RowBeginIn** (input control) ........................................line.begin.y ~ VARIANT (integer)
  Row coordinates of the starting points of the input lines.

- **ColBeginIn** (input control) ........................................line.begin.x ~ VARIANT (integer)
  Column coordinates of the starting points of the input lines.

- **RowEndIn** (input control) ........................................line.end.y ~ VARIANT (integer)
  Row coordinates of the ending points of the input lines.

- **ColEndIn** (input control) ........................................line.end.x ~ VARIANT (integer)
  Column coordinates of the ending points of the input lines.

- **Num** (input control) ..............................................integer ~ long
  (Maximum) desired number of output lines.

  **Default Value**: 10

- **RowBeginOut** (output control) .................................line.begin.y ~ VARIANT (integer)
  Row coordinates of the starting points of the output lines.

- **ColBeginOut** (output control) .................................line.begin.x ~ VARIANT (integer)
  Column coordinates of the starting points of the output lines.

- **RowEndOut** (output control) .................................line.end.y ~ VARIANT (integer)
  Row coordinates of the ending points of the output lines.

- **ColEndOut** (output control) .................................line.end.x ~ VARIANT (integer)
  Column coordinates of the ending points of the output lines.
\begin{itemize}
\item \textbf{ColEndOut} (output control) \hspace{5cm} line.end.x \sim \text{VARIANT (integer)}
\end{itemize}

Column coordinates of the ending points of the output lines.

\begin{tabular}{|l|}
\hline
\textbf{Result} \hline
The operator \texttt{SelectLinesLongest} returns the value \texttt{TRUE} if the parameter values are correct. Otherwise an exception is raised. \hline
\end{tabular}

\begin{tabular}{|l|}
\hline
\textbf{Parallelization Information} \hline
\texttt{SelectLinesLongest} is \textit{reentrant} and processed \textit{without} parallelization. \hline
\end{tabular}

\begin{tabular}{|l|}
\hline
\textbf{Possible Predecessor Functions} \hline
\texttt{SobelAmp, EdgesImage, Threshold, HysteresisThreshold, SplitSkeletonRegion, SplitSkeletonLines} \hline
\end{tabular}

\begin{tabular}{|l|}
\hline
\textbf{Possible Successor Functions} \hline
\texttt{SetLineWidth, DispLine} \hline
\end{tabular}

\begin{tabular}{|l|}
\hline
\textbf{See Also} \hline
\texttt{SelectLines, PartitionLines, DetectEdgeSegments, SelectShape} \hline
\end{tabular}

\begin{tabular}{|l|}
\hline
\textbf{Alternatives} \hline
\texttt{LineOrientation, LinePosition, SelectLines, PartitionLines} \hline
\end{tabular}

\begin{tabular}{|l|}
\hline
\textbf{Module} \hline
\texttt{Region processing} \hline
\end{tabular}
Chapter 7

Morphology

7.1 Gray-Values

```c
```

Opening, Median and Closing with circle or rectangle mask.

The operator DualRank carries out a non-linear transformation of the gray values of all input images (Image). Circles or squares can be used as structuring elements. The operator DualRank effects two consecutive calls of RankImage. At the first call the range gray value is calculated with the indicated range (ModePercent). The result of this calculation is the input of a further call of RankImage, this time using the range value 100—ModePercent.

When filtering different parameters for margin control (Margin) can be chosen:

<table>
<thead>
<tr>
<th>Margin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0...255</td>
<td>Pixels outside the image edges are assumed to be constant (with the indicated gray value).</td>
</tr>
<tr>
<td>-1</td>
<td>Continuation of edge pixels.</td>
</tr>
<tr>
<td>-2</td>
<td>Cyclic continuation of image edges.</td>
</tr>
<tr>
<td>-3</td>
<td>Reflection of pixels at image edges.</td>
</tr>
</tbody>
</table>

A range filtering is calculated according to the following scheme: The indicated mask is put over the image to be filtered in such a way that the center of the mask touches all pixels once. For each of these pixels all neighboring pixels covered by the mask are sorted in an ascending sequence corresponding to their gray values. Each sorted sequence of gray values contains the same number of gray values like the mask has image points. The n-th highest element, (= ModePercent, rank values between 0...100 in percent) is selected and set as result gray value in the corresponding result image.

If ModePercent is 0, then the operator equals to the gray value opening (GrayOpening). If ModePercent is 50, the operator results in the median filter, which is applied twice (MedianImage). The ModePercent 100 in DualRank means that it calculates the gray value closing (GrayClosing). Choosing parameter values inside this range results in a smooth transformation of these operators.

Parameter

<table>
<thead>
<tr>
<th>Image (input iconic)</th>
<th>(multichannel-)image-array</th>
<th>HImageX / IObjectX (byte)</th>
<th>Image to be filtered.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageRank (output iconic)</td>
<td>multichannel-image-array</td>
<td>HImageX / HUntypedObjectX</td>
<td>Filtered Image.</td>
</tr>
</tbody>
</table>
\textbf{MaskType} (input control) \hspace{1cm} string $\sim$ \textit{String}
Shape of the mask.
\textbf{Default Value} : 'circle'
\textbf{Value List} : \texttt{[\{circle’, rectangle\]}

\textbf{Radius} (input control) \hspace{1cm} integer $\sim$ \textit{long}
Radius of the filter mask.
\textbf{Default Value} : 1
\textbf{Value Suggestions} : \texttt{[\{1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 15, 19, 25, 31, 39, 47, 59\]}
\textbf{Typical Range of Values} : \texttt{[\{1 $\leq$ Radius $\leq$ 1\]}
\textbf{Minimal Value Step} : 1
\textbf{Recommended Value Step} : 2

\textbf{ModePercent} (input control) \hspace{1cm} integer $\sim$ \textit{long}
Filter Mode: 0 corresponds to a gray value opening, 50 corresponds to a median and 100 to a gray values closing.
\textbf{Default Value} : 10
\textbf{Value Suggestions} : \texttt{[\{0, 2, 5, 10, 15, 20, 40, 50, 60, 80, 85, 90, 95, 98, 100\]}
\textbf{Typical Range of Values} : \texttt{[\{0 $\leq$ ModePercent $\leq$ 0\]}
\textbf{Minimal Value Step} : 1
\textbf{Recommended Value Step} : 2

\textbf{Margin} (input control) \hspace{1cm} integer $\sim$ \textit{long}
Margin control: 0...255 (constant), -1 (margin points continued), -2 (cyclic continuation), -3 (mirroring).
\textbf{Default Value} : -3
\textbf{Typical Range of Values} : \texttt{[-3 $\leq$ Margin $\leq$ -3\]}

---

\textbf{Example}

\begin{verbatim}
read_image(Image,'fabrik')
dual_rank(Image,ImageOpening,'circle',10,10,-3)
disp_image(ImageOpening,WindowHandle).
\end{verbatim}

---

\textbf{Result}
If the parameter values are correct the operator \texttt{DualRank} returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem(’noObjectResult’,<Result>).\ If necessary an exception handling is raised.}

\textbf{Parallelization Information}
\texttt{DualRank} is \textit{reentrant} and automatically \textit{parallelized} (on \textit{tuple level, channel level}).

\textbf{Possible Predecessor Functions}
\texttt{ReadImage}

\textbf{Possible Successor Functions}
\texttt{Threshold,DynThreshold,SubImage,Regiongrow}

---

\textbf{See Also}
\texttt{GenCircle,GenRectangle1,GrayErosionRect,GrayDilationRect,\SigmaImage}

---

\textbf{Alternatives}
\texttt{RankImage,GrayClosing,GrayOpening,MedianImage}

---

\textbf{Bibliography}

---

\textbf{Module}
Image filters
Generate ellipsoidal structuring elements for gray morphology.

GenDiscSe generates an ellipsoidal structuring element (SE) for gray morphology of images. The parameters Width and Height determine the length of the two major axes of the ellipse. The value of Smax determines the maximum gray value of the structuring element. For the generation of arbitrary structuring elements, see ReadGraySe.

**Parameter**

- **SE** (output iconic) .................................................... image → HImageX / HUntypedObjectX (byte)  
  Generated structuring element.

- **Width** (input control) .............................................. integer → long  
  Width of the structuring element.  
  Default Value: 5  
  Value Suggestions: Width ∈ {0, 1, 2, 3, 4, 5, 10, 15, 20}  
  Typical Range of Values: 0 ≤ Width ≤ 0(lin)  
  Minimal Value Step: 1  
  Recommended Value Step: 1

- **Height** (input control) ............................................. integer → long  
  Height of the structuring element.  
  Default Value: 5  
  Value Suggestions: Height ∈ {0, 1, 2, 3, 4, 5, 10, 15, 20}  
  Typical Range of Values: 0 ≤ Height ≤ 0(lin)  
  Minimal Value Step: 1  
  Recommended Value Step: 1

- **Smax** (input control) .............................................. integer → long  
  Maximum gray value of the structuring element.  
  Default Value: 0  
  Value Suggestions: Smax ∈ {0, 1, 2, 5, 10, 20, 30, 40}  
  Typical Range of Values: 0 ≤ Smax ≤ 0(lin)  
  Minimal Value Step: 1  
  Recommended Value Step: 1

**Result**

GenDiscSe returns TRUE if all parameters are correct. If necessary, an exception is raised.

**Parallelization Information**

GenDiscSe is reentrant and processed without parallelization.

**Possible Successor Functions**

GrayErosion, GrayDilation, GrayOpening, GrayClosing, GrayTophat, GrayBothat

**See Also**

ReadImage, PaintRegion, PaintGray, CropPart

**Alternatives**

ReadGraySe

**Module**

Image filters

```c
void HImageX.GenDiscSe ( [in] long Width, [in] long Height, [in] long Smax )

```

Perform a gray value bottom hat transformation on an image.

HALCON 6.0
GrayBothat applies a gray value bottom hat transformation to the input image \( \text{Image} \) with the structuring element \( SE \). The gray value bottom hat transformation of an image \( i \) with a structuring element \( s \) is defined as

\[
\text{bothat}(i, s) = (i \ast s) - i,
\]

i.e., the difference of the closing of the image with \( s \) and the image (see \text{GrayClosing}). For the generation of structuring elements, see \text{ReadGraySe}.

Parameter

\[\begin{align*}
\textbf{Image} & \text{ (input iconic) } \quad \text{image(-array)} \sim \text{HImageX / IObjectX (byte)} \\
\text{SE} & \text{ (input iconic) } \quad \text{image} \sim \text{HImageX / IObjectX (byte)} \\
\text{ImageBotHat} & \text{ (output iconic) } \quad \text{image(-array)} \sim \text{HImageX / HUntypedObjectX (byte)}
\end{align*}\]

Result

\( \text{GrayBothat} \) returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.

Parallelization Information

\( \text{GrayBothat} \) is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

\text{ReadGraySe, GenDiscSe}

Possible Successor Functions

\text{Threshold}

See Also

\text{GrayTophat, TopHat, GrayErosionRect, SubImage}

Alternatives

\text{GrayClosing}

Module

\text{Image filters}

```
[out] HImageX ImageClosing HImageX.GrayClosing ([in] HImageX SE )
```

Perform a gray value closing on an image.

\text{GrayClosing} applies a gray value closing to the input image \( \text{Image} \) with the structuring element \( SE \). The gray value closing of an image \( i \) with a structuring element \( s \) is defined as

\[
i \ast s = (i \oplus s) \ominus s,
\]

i.e., a dilation of the image with \( s \) followed by an erosion with \( s \) (see \text{GrayDilation} and \text{GrayErosion}). For the generation of structuring elements, see \text{ReadGraySe}.

Parameter

\[\begin{align*}
\textbf{Image} & \text{ (input iconic) } \quad \text{image(-array)} \sim \text{HImageX / IObjectX (byte)} \\
\text{SE} & \text{ (input iconic) } \quad \text{image} \sim \text{HImageX / IObjectX (byte)} \\
\text{ImageClosing} & \text{ (output iconic) } \quad \text{image(-array)} \sim \text{HImageX / HUntypedObjectX (byte)}
\end{align*}\]

Result

\( \text{GrayClosing} \) returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.

GrayClosing is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

ReadGraySe

See Also

Closing, GrayDilation, GrayErosion

Possible Successor Functions

SubImage, GrayErosion

See Also

GrayOpening, GrayClosing, Dilation1, GraySkeleton

Alternatives

GrayDilationRect

Module

Image filters

---

[perform a gray value dilation on an image.]

GrayDilation applies a gray value dilation to the input image Image with the structuring element SE. The gray value dilation of an image \( i \) with a structuring element \( s \) at the pixel position \( x \) is defined as:

\[
(i \odot s)(x) = \max\{f(x - z) + s(z)|z \in S\}
\]

Here, \( S \) is the domain of the structuring element \( s \), i.e., the pixels \( z \) where \( s(z) > 0 \) (see ReadGraySe).

Parameter

- **Image** (input iconic) image-array
  - Input image.
- **SE** (input iconic) image-array
  - Structuring element.
- **ImageDilation** (output iconic) image-array
  - Gray-dilated image.

Result

GrayDilation returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.

Parallelization Information

GrayDilation is reentrant and automatically parallelized (on tuple level).

---

[out] HImageX ImageDilation [in] HImageX SE

void HOperatorSetX.GrayDilation ([in] IHObjectX Image,
[in] IHObjectX SE, [out] HUntypedObjectX ImageDilation )

Determine the maximum gray value within a rectangle.

---

[out] HImageX ImageMax

void HOperatorSetX.GrayDilationRect ([in] long MaskHeight,
[in] long MaskWidth )

Determine the maximum gray value within a rectangle.
GrayDilationRect calculates the maximum gray value of the input image Image within a rectangular mask of size (MaskHeight, MaskWidth) for each image point. The resulting image is returned in ImageMax. If the parameters MaskHeight or MaskWidth are even, they are changed to the next larger odd value. At the border of the image the gray values are mirrored.

Parameter

- **Image** (input iconic) ................. image(-array) \(\sim\) HImageX / IHObjectX (byte, direction, int4, real)
  
  Image for which the maximum gray values are to be calculated.

- **ImageMax** (output iconic) ........ image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, direction, int4, real)
  
  Image containing the maximum gray values.

- **MaskHeight** (input control) ........................................................ extent.y \(\sim\) long
  
  Height of the filter mask.

  Default Value: 11

  Value Suggestions: MaskHeight \(\in\) \{3, 5, 7, 9, 11, 13, 15\}

  Typical Range of Values: \(3 \leq\) MaskHeight \(\leq 3\)

  Minimal Value Step: 2

  Recommended Value Step: 2

  Restriction: odd

- **MaskWidth** (input control) ........................................................ extent.x \(\sim\) long
  
  Width of the filter mask.

  Default Value: 11

  Value Suggestions: MaskWidth \(\in\) \{3, 5, 7, 9, 11, 13, 15\}

  Typical Range of Values: \(3 \leq\) MaskWidth \(\leq 3\)

  Minimal Value Step: 2

  Recommended Value Step: 2

  Restriction: odd

Result

GrayDilationRect returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception is raised.

Parallelization Information

GrayDilationRect is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See Also

GraySkeleton

Module

Image filters

```
```

Perform a gray value erosion on an image.

GrayErosion applies a gray value erosion to the input image Image with the structuring element SE. The gray value erosion of an image \(i\) with a structuring element \(s\) at the pixel position \(x\) is defined as:

\[
(i \ominus s)(x) = \min \{ f(x + z) - s(z) | z \in S \}
\]

Here, \(S\) is the domain of the structuring element \(s\), i.e., the pixels \(z\) where \(s(z) > 0\) (see ReadGraySe).

Parameter

- **Image** (input iconic) ................. image(-array) \(\sim\) HImageX / IHObjectX (byte)
  
  Input image.

- **SE** (input iconic) .............................. image \(\sim\) HImageX / IHObjectX (byte)
  
  Structuring element.

- **ImageErosion** (output iconic) ........ image(-array) \(\sim\) HImageX / HUntypedObjectX (byte)
  
  Gray-eroded image.
7.1. GRAY-VALUES 385

GrayErosion returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.

Parallelization Information

GrayErosion is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

ReadGraySe

Possible Successor Functions

GrayDilation, SubImage

See Also

GrayOpening, GrayClosing, Erosion1, GraySkeleton

Alternatives

GrayErosionRect

Module

Image filters

---

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong></td>
<td>(input iconic) ... image-array ( \sim ) ( \text{HImageX} / \text{HObjectX} ) (byte, direction, int4, real) Image for which the minimum gray values are to be calculated.</td>
</tr>
<tr>
<td><strong>ImageMin</strong></td>
<td>(output iconic) ... image-array ( \sim ) ( \text{HImageX} / \text{HUntypedObjectX} ) (byte, direction, int4, real) Image containing the minimum gray values.</td>
</tr>
<tr>
<td><strong>MaskHeight</strong></td>
<td>(input control) ... extent.y ( \sim ) long Height of the filter mask. Default Value: 11 Value Suggestions: ( \text{MaskHeight} \in {3, 5, 7, 9, 11, 13, 15} ) Typical Range of Values: ( 3 \leq \text{MaskHeight} \leq 3(\text{lin}) ) Minimal Value Step: 2 Recommended Value Step: 2 Restriction: ( \text{odd} )</td>
</tr>
<tr>
<td><strong>MaskWidth</strong></td>
<td>(input control) ... extent.x ( \sim ) long Width of the filter mask. Default Value: 11 Value Suggestions: ( \text{MaskWidth} \in {3, 5, 7, 9, 11, 13, 15} ) Typical Range of Values: ( 3 \leq \text{MaskWidth} \leq 3(\text{lin}) ) Minimal Value Step: 2 Recommended Value Step: 2 Restriction: ( \text{odd} )</td>
</tr>
</tbody>
</table>

---

Result

GrayErosionRect returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via \text{SetSystem} ("noObjectResult", <Result>). If necessary, an exception is raised.

Parallelization Information

GrayErosionRect is reentrant and automatically parallelized (on tuple level, channel level, domain level).
Perform a gray value opening on an image.

GrayOpening applies a gray value opening to the input image Image with the structuring element SE. The gray value opening of an image $i$ with a structuring element $s$ is defined as

$$i \circ s = (i \ominus s) \oplus s,$$

i.e., an erosion of the image with $s$ followed by a dilation with $s$ (see GrayErosion and GrayDilation). For the generation of structuring elements, see ReadGraySe.

Parameter

- **Image** (input iconic)  
  Input image.

- **SE** (input iconic)  
  Structuring element.

- **ImageOpening** (output iconic)  
  Gray-opened image.

Result

GrayOpening returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.

Parallelization Information

GrayOpening is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Opening, GrayDilation, GrayErosion

See Also

DualRank

Alternatives

Image filters

Determine the gray value range within a rectangle.

GrayRangeRect calculates the gray value range, i.e., the difference (max – min) of the maximum and minimum gray values, of the input image Image within a rectangular mask of size (MaskHeight, MaskWidth) for each image point. The resulting image is returned in ImageResult. If the parameters MaskHeight or MaskWidth are even, they are changed to the next larger odd value. At the border of the image the gray values are mirrored.
### 7.1. GRAY-VALUES

**Parameter**

- **Image** (input iconic) .......................... image(-array) \(\sim\) HImageX / IHObjectX (byte, direction, int4, real)
  Image for which the gray value range is to be calculated.
- **ImageResult** (output iconic) image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, direction, int4, real)
  Image containing the gray value range.
- **MaskHeight** (input control) .......................................................... extent.y \(\sim\) long
  Height of the filter mask.
  - Default Value: 11
  - Value Suggestions: \(\text{MaskHeight} \in \{3, 5, 7, 9, 11, 13, 15\}\)
  - Typical Range of Values: \(3 \leq \text{MaskHeight} \leq 3\) (lin)
  - Minimal Value Step: 2
  - Recommended Value Step: 2
  - Restriction: odd

- **MaskWidth** (input control) .......................................................... extent.x \(\sim\) long
  Width of the filter mask.
  - Default Value: 11
  - Value Suggestions: \(\text{MaskWidth} \in \{3, 5, 7, 9, 11, 13, 15\}\)
  - Typical Range of Values: \(3 \leq \text{MaskWidth} \leq 3\) (lin)
  - Minimal Value Step: 2
  - Recommended Value Step: 2
  - Restriction: odd

**Result**

GrayRangeRect returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception is raised.

**Parallelization Information**

GrayRangeRect is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Alternatives**

GrayDilationRect, GrayErosionRect, SubImage

**Module**

Image filters

```plaintext
[out] HImageX ImageTopHat HImageX.GrayTophat ([in] HImageX SE )

```

Perform a gray value top hat transformation on an image.

GrayTophat applies a gray value top hat transformation to the input image Image with the structuring element SE. The gray value top hat transformation of an image \(i\) with a structuring element \(s\) is defined as

\[
\text{tophat}(i, s) = i - (i \circ s),
\]

i.e., the difference of the image and its opening with \(s\) (see GrayOpening). For the generation of structuring elements, see ReadGraySe.
GrayTophat is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

ReadGraySe, GenDiscSe

Possible Successor Functions

Threshold

See Also

GrayBothat, TopHat, GrayErosionRect, SubImage

Alternatives

GrayOpening

Module

Image filters

--- Parallelization Information ---

GrayTophat is reentrant and automatically parallelized (on tuple level).

--- Possible Predecessor Functions ---

ReadGraySe, GenDiscSe

--- Possible Successor Functions ---

Threshold

See Also

GrayBothat, TopHat, GrayErosionRect, SubImage

Alternatives

GrayOpening

Module

Load a structuring element for gray morphology.

ReadGraySe loads a structuring element for gray morphology from a file. The file names of these structuring elements must end in `.gse` (for gray-scale structuring element). This suffix is automatically appended by ReadGraySe to the passed file name, and thus must not be passed. The structuring element’s data must be contained in the file in the following format: The first two numbers in the file determine the width and height of the structuring element, and determine a rectangle enclosing the structuring element. Both values must be greater than 0. Then, Width*Height integer numbers follow, with the following interpretation: Values smaller than 0 are regarded as not belonging to the region of the structuring element, i.e., they are not considered in morphological operations. This allows the creation of irregularly shaped, not connected structuring elements. All other values are regarded as the corresponding values for gray morphology. Structuring elements are stored internally as byte-images, with negative values being mapped to 0, and all other values increased by 1. Thus, normal byte-images can also be used as structuring elements. However, care should be taken not to use too large images, since the runtime is proportional to the area of the image times the area of the structuring element.

--- Parameter ---

- **SE** (output iconic) ... image \sim HImageX / HUntypedObjectX ( byte )
  Generated structuring element.
- **FileName** (input control) ... filename \sim String
  Name of the file containing the structuring element.

--- Result ---

ReadGraySe returns TRUE if all parameters are correct. If the file cannot be opened, FAIL is returned. Otherwise, an exception is raised.

--- Parallelization Information ---

GrayTophat is reentrant and processed without parallelization.

--- Possible Successor Functions ---

GrayErosion, GrayDilation, GrayOpening, GrayClosing, GrayTophat, GrayBothat

--- See Also ---

ReadImage, PaintRegion, PaintGray, CropPart

--- Alternatives ---

GenDiscSe

--- Module ---

Image filters

--- Code ---

```c
void HImageX.ReadGraySe ([in] String FileName )

void HOperatorSetX.ReadGraySe ([out] HUntypedObjectX SE, [in] String FileName )
```

--- Notes ---

ReadGraySe loads a structuring element for gray morphology from a file. The file names of these structuring elements must end in `.gse` (for gray-scale structuring element). This suffix is automatically appended by ReadGraySe to the passed file name, and thus must not be passed. The structuring element’s data must be contained in the file in the following format: The first two numbers in the file determine the width and height of the structuring element, and determine a rectangle enclosing the structuring element. Both values must be greater than 0. Then, Width*Height integer numbers follow, with the following interpretation: Values smaller than 0 are regarded as not belonging to the region of the structuring element, i.e., they are not considered in morphological operations. This allows the creation of irregularly shaped, not connected structuring elements. All other values are regarded as the corresponding values for gray morphology. Structuring elements are stored internally as byte-images, with negative values being mapped to 0, and all other values increased by 1. Thus, normal byte-images can also be used as structuring elements. However, care should be taken not to use too large images, since the runtime is proportional to the area of the image times the area of the structuring element.

--- Parameter ---

- **SE** (output iconic) ... image \sim HImageX / HUntypedObjectX ( byte )
  Generated structuring element.
- **FileName** (input control) ... filename \sim String
  Name of the file containing the structuring element.

--- Result ---

ReadGraySe returns TRUE if all parameters are correct. If the file cannot be opened, FAIL is returned. Otherwise, an exception is raised.

--- Parallelization Information ---

GrayTophat is reentrant and processed without parallelization.

--- Possible Successor Functions ---

GrayErosion, GrayDilation, GrayOpening, GrayClosing, GrayTophat, GrayBothat

--- See Also ---

ReadImage, PaintRegion, PaintGray, CropPart

--- Alternatives ---

GenDiscSe

--- Module ---

Image filters

--- Code ---

```c
void HImageX.ReadGraySe ([in] String FileName )

void HOperatorSetX.ReadGraySe ([out] HUntypedObjectX SE, [in] String FileName )
```

--- Notes ---

ReadGraySe loads a structuring element for gray morphology from a file. The file names of these structuring elements must end in `.gse` (for gray-scale structuring element). This suffix is automatically appended by ReadGraySe to the passed file name, and thus must not be passed. The structuring element’s data must be contained in the file in the following format: The first two numbers in the file determine the width and height of the structuring element, and determine a rectangle enclosing the structuring element. Both values must be greater than 0. Then, Width*Height integer numbers follow, with the following interpretation: Values smaller than 0 are regarded as not belonging to the region of the structuring element, i.e., they are not considered in morphological operations. This allows the creation of irregularly shaped, not connected structuring elements. All other values are regarded as the corresponding values for gray morphology. Structuring elements are stored internally as byte-images, with negative values being mapped to 0, and all other values increased by 1. Thus, normal byte-images can also be used as structuring elements. However, care should be taken not to use too large images, since the runtime is proportional to the area of the image times the area of the structuring element.
7.2 Region

Compute the bottom hat of regions.

BottomHat computes the Closing of Region with StructElement. The difference between the result of the closing and the original region is called the bottom hat. In contrast to Closing, which merges regions under certain circumstances, BottomHat computes the regions generated by such a merge.

The position of StructElement is meaningless, since a closing operation is invariant with respect to the choice of the reference point.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

Example

read_image (Image,’/bilder/name.ext’)
threshold (Image,Regions,128,255)
gen_circle (Circle,0,0,16)
bottom_hat (Regions,Circle,RegionBottomHat).

Result

BottomHat returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

BottomHat is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

TopHat, MorphHat, GrayBothat, Opening

Alternatives

Closing, Difference

Module

Morphology

HALCON 6.0
Reduce a region to its boundary.

Boundary computes the boundary of a region by using morphological operations. The parameter BoundaryType determines the type of boundary to compute: ‘inner’, ‘inner_filled’ and ‘outer’.

Boundary computes the contour of each input region. The resulting regions consist only of the minimal border of the input regions. If BoundaryType is set to ‘inner’, the contour lies within the original region, if it is set to ‘outer’, it is one pixel outside of the original region. If BoundaryType is set to ‘inner_filled’, holes in the interior of the input region are suppressed.

**Parameter**

- **Region** (input iconic) 
  - region(-array) 
  - HRegionX / IHObjectX
  - Regions for which the boundary is to be computed.

- **RegionBorder** (output iconic) 
  - region(-array) 
  - HRegionX / HUntypedObjectX
  - Resulting boundaries.

- **BoundaryType** (input control) 
  - string
  - Boundary type.
  - **Default Value**: ‘inner’
  - **Value List**: BoundaryType ∈ {‘inner’, ‘outer’, ‘inner_filled’}

**Result**

Boundary returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

**Parallelization Information**

Boundary is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

**Possible Successor Functions**

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

FillUp

Alternatives

DilationCircle, ErosionCircle, Difference

Module

Close a region.

A Closing operation is defined as a dilation followed by a Minkowski subtraction. By applying Closing to a region, larger structures remain mostly intact, while small gaps between adjacent regions and holes smaller than StructElement are closed, and the regions’ boundaries are smoothed. All Closing variants share the
property that separate regions are not merged, but remain separate objects. The position of StructElement is meaningless, since a closing operation is invariant with respect to the choice of the reference point.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc. Closing is applied to each input region separately. If gaps between different regions are to be closed, Union1 or Union2 has to be called first.

Parameter

- **Region** (input iconic) .......................... region(-array) ~ HRegionX / IHObjectX Regions to be closed.
- **StructElement** (input iconic) .............. region ~ HRegionX / IHObjectX Structuring element (position-invariant).
- **RegionClosing** (output iconic) ............. region(-array) ~ HRegionX / HUntypedObjectX Closed regions.

Result

Closing returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

Closing is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

Dilation1, Erosion1, Opening, MinkowskiSub1

Alternatives

ClosingCircle, ClosingGolay

Module

Morphology

```plaintext
[out] HRegionX RegionClosing HRegionX.ClosingCircle
([in] VARIANT Radius )

void HOperatorSetX.ClosingCircle ([in] IHObjectX Region,
[out] HUntypedObjectX RegionClosing, [in] VARIANT Radius )
```

Close a region with a circular structuring element.

ClosingCircle behaves analogously to Closing, i.e., the regions’ boundaries are smoothed and holes within a region which are smaller than the circular structuring element of radius Radius are closed. The ClosingCircle operation is defined as a dilation followed by a Minkowski subtraction, both with the same circular structuring element.

Attention

ClosingCircle is applied to each input region separately. If gaps between different regions are to be closed, Union1 or Union2 has to be called first.
### Parameter

- **Region** (input iconic)  
  Region( - array )  
  Regions to be closed.

- **RegionClosing** (output iconic)  
  Region( - array )  
  Closed regions.

- **Radius** (input control)  
  Radius of the circular structuring element.  
  **Default Value:** 3.5  
  **Value Suggestions:** Radius ∈ {1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5}  
  **Typical Range of Values:** 0.5 ≤ Radius ≤ 0.5(lin)  
  **Minimal Value Step:** 1.0  
  **Recommended Value Step:** 1.0

### Result

ClosingCircle returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: `SetSystem('noObjectResult',<RegionResult>)`
- empty region: `SetSystem('emptyRegionResult',<RegionResult>)`

Otherwise, an exception is raised.

### Parallelization Information

ClosingCircle is reentrant and automatically parallelized (on tuple level).

### Possible Predecessor Functions

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

### Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

### See Also

Dilation1, MinkowskiSub1, Erosion1, Opening

### Alternatives

RankRegion, FillUp, Closing, ClosingCircle, ClosingGolay

### Module

Morphology

```c
[out] HRegionX RegionClosing HRegionX.ClosingGolay
([in] String GolayElement, [in] long Rotation )

void HOperatorSetX.ClosingGolay ([in] IObjectX Region,
[out] HUntypedObjectX RegionClosing, [in] String GolayElement,
[in] long Rotation )
```

Close a region with an element from the Golay alphabet.

ClosingGolay is defined as a Minkowski addition followed by a Minkowski subtraction. First the Minkowski addition of the input region (Region) with the structuring element from the Golay alphabet defined by GolayElement and Rotation is computed. Then the Minkowski subtraction of the result and the structuring element rotated by 180° is performed.

The following structuring elements are available:

- 'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The rotation number Rotation determines which rotation of the element should be used, and whether the foreground (even) or background version (odd) of the selected element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements.

ClosingGolay serves to close holes smaller than the structuring element, and to smooth regions' boundaries.
Not all values of Rotation are valid for any Golay element. For some of the values of Rotation, the resulting regions are identical to the input regions.

Attention

ClosingGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(’noObjectResult’,<RegionResult>)
- empty region: SetSystem(’emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

ClosingGolay is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Regiongrowing, Connection, Union1, Watersheds, ClassNdimNorm

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

ErosionGolay, DilationGolay, OpeningGolay, HitOrMissGolay, ThickeningGolay, ThinningGolay, GolayElements

Alternatives

Morphology

Close a region with a rectangular structuring element.

ClosingRectangle1 performs a DilationRectangle1 followed by an ErosionRectangle1 on the input region Region. The size of the rectangular structuring element is determined by the parameters Width and Height. As is the case for all Closing variants, regions’ boundaries are smoothed and holes within a region which are smaller than the rectangular structuring element are closed.

Attention

ClosingRectangle1 is applied to each input region separately. If gaps between different regions are to be closed, Union1 or Union2 has to be called first.
**Parameter**

- **Region** (input iconic) .......................... region(-array) \( \sim \) HRegionX / IHObjectX
  - Regions to be closed.

- **RegionClosing** (output iconic) ................. region(-array) \( \sim \) HRegionX / HUntypedObjectX
  - Closed regions.

- **Width** (input control) ........................... extent.x \( \sim \) VARIANT ( integer, real )
  - Width of the structuring rectangle.
    - **Default Value**: 10
    - **Value Suggestions**: \( \text{Width} \in \{1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200\} \)
    - **Typical Range of Values**: \( 1 \leq \text{Width} \leq 1(\text{lin}) \)
    - **Minimal Value Step**: 1
    - **Recommended Value Step**: 1

- **Height** (input control) .......................... extent.y \( \sim \) VARIANT ( integer, real )
  - Height of the structuring rectangle.
    - **Default Value**: 10
    - **Value Suggestions**: \( \text{Height} \in \{1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200\} \)
    - **Typical Range of Values**: \( 1 \leq \text{Height} \leq 1(\text{lin}) \)
    - **Minimal Value Step**: 1
    - **Recommended Value Step**: 1

**Result**

ClosingRectangle1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’, <RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’, <RegionResult>)

Otherwise, an exception is raised.

**Parallelization Information**

ClosingRectangle1 is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNDimNorm

**Possible Successor Functions**

ReduceDomain, SelectShape, AreaCenter, Connection

**See Also**

DilationRectangle1, ErosionRectangle1, OpeningRectangle1, GenRectangle1

**Alternatives**

Closing

**Module**

Morphology

```plaintext
[out] HRegionX RegionDilation HRegionX.Dilation1
([in] HRegionX StructElement, [in] long Iterations )

void HOperatorSetX.Dilation1 ([in] IHObjectX Region,
[in] IHObjectX StructElement, [out] HUntypedObjectX RegionDilation,
[in] long Iterations )
```

*Dilate a region.*

Dilation1 dilates the input regions with a structuring element. By applying Dilation1 to a region, its boundary gets smoothed. In the process, the area of the region is enlarged. Furthermore, disconnected regions may be merged. Such regions, however, remain logically distinct region. The dilation is a set-theoretic region operation. It uses the union operation.

Let \( M \) (StructElement) and \( R \) (Region) be two regions, where \( M \) is the structuring element and \( R \) is the region to be processed. Furthermore, let \( m \) be a point in \( M \). Then the displacement vector \( \overrightarrow{\delta_m} = (dx, dy) \) is
defined as the difference of the center of gravity of \( M \) and the vector \( \vec{m} \). Let \( t_{\vec{m}}(R) \) denote the translation of a region \( R \) by a vector \( \vec{m} \). Then

\[
Dilation1(R, M) := \bigcup_{m \in M} t_{-\vec{m}}(R)
\]

For each point \( m \) in \( M \) a translation of the region \( R \) is performed. The union of all these translations is the dilation of \( R \) with \( M \). \( Dilation1 \) is similar to the operator \( MinkowskiAdd1 \), the difference is that in \( Dilation1 \) the structuring element is mirrored at the origin. The position of \( StructElement \) is meaningless, since the displacement vectors are determined with respect to the center of gravity of \( M \).

The parameter \( Iterations \) determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \). From the above definition it follows that an empty region is generated in case of an empty structuring element.

Structuring elements (\( StructElement \)) can be generated with operators such as \( GenCircle \), \( GenRectangle1 \), \( GenRectangle2 \), \( GenEllipse \), \( DrawRegion \), \( GenRegionPolygon \), \( GenRegionPoints \), etc.

---

**Attention**

A dilation always results in enlarged regions. Closely spaced regions which may touch or overlap as a result of the dilation are still treated as two separate regions. If the desired behavior is to merge them into one region, the operator \( Union1 \) has to be called first.

---

**Parameter**

- **Region** (input iconic) .............................................. \( \text{region(-array)} \sim \text{HRegionX} / \text{IObjectX} \)
  - Regions to be dilated.

- **StructElement** (input iconic) ................................. \( \text{region} \sim \text{HRegionX} / \text{IObjectX} \)
  - Structuring element.

- **RegionDilation** (output iconic) .............................. \( \text{region(-array)} \sim \text{HRegionX} / \text{HUntypedObjectX} \)
  - Dilated regions.

- **Iterations** (input control) ..................................... integer \( \sim \text{long} \)
  - Number of iterations.
  - **Default Value:** 1
  - **Value Suggestions:** Iterations \( \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\} \)
  - **(lin)Minimal Value Step:** 1
  - **Recommended Value Step:** 1

---

**Result**

\( Dilation1 \) returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: \( \text{SetSystem('noObjectResult','<RegionResult>')} \)
- empty region: \( \text{SetSystem('emptyRegionResult','<RegionResult>')} \)

Otherwise, an exception is raised.

---

**Parallelization Information**

\( Dilation1 \) is **reentrant** and automatically **parallelized** (on tuple level).

**Possible Predecessor Functions**

\( \text{Threshold}, \text{RegionGrowing}, \text{Connection}, \text{Union1}, \text{Watersheds}, \text{ClassNdimNorm}, \text{GenCircle}, \text{GenEllipse}, \text{GenRectangle1}, \text{GenRectangle2}, \text{DrawRegion}, \text{GenRegionPoints}, \text{GenStructElements}, \text{GenRegionPolygonFilled} \)

**Possible Successor Functions**

\( \text{ReduceDomain}, \text{AddChannels}, \text{SelectShape}, \text{AreaCenter}, \text{Connection} \)

---

**See Also**

\( \text{Erosion1}, \text{Erosion2}, \text{Opening}, \text{Closing} \)

---

**Alternatives**

\( \text{MinkowskiAdd1}, \text{MinkowskiAdd2}, \text{Dilation2}, \text{DilationGolay}, \text{DilationSeq} \)

---

**Module**

\( \text{Morphology} \)

HALCON 6.0
**Dilate a region (using a reference point).**

Dilation2 dilates the input regions with a structuring element (StructElement) having the reference point (Row,Column). Dilation2 has a similar effect as Dilation1, the difference is that the reference point of the structuring element can be chosen arbitrarily. The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \).

An empty region is generated in case of an empty structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

**Attention**

A dilation always results in enlarged regions. Closely spaced regions which may touch or overlap as a result of the dilation are still treated as two separate regions. If the desired behavior is to merge them into one region, the operator Union1 has to be called first.

**Parameter**

- **Region** (input iconic) ......................... region(-array) \( \sim \) HRegionX / IHObjectX
  Regions to be dilated.
- **StructElement** (input iconic) .................. region \( \sim \) HRegionX / IHObjectX
  Structuring element.
- **RegionDilation** (output iconic) ............... region(-array) \( \sim \) HRegionX / HUntypedObjectX
  Dilated regions.
- **Row** (input control) ............................ point.y \( \sim \) long
  Row coordinate of the reference point.
  Default Value : 0
- **Column** (input control) ........................ point.x \( \sim \) long
  Column coordinate of the reference point.
  Default Value : 0
- **Iterations** (input control) ..................... integer \( \sim \) long
  Number of iterations.
  Default Value : 1
  Value Suggestions: Iterations \( \in \{1, 2, 3, 4, 5, 7, 11, 17, 25, 32, 64, 128\} \)
  (lin)Minimal Value Step : 1
  Recommended Value Step : 1

**Result**

Dilation2 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

**Parallelization Information**

Dilation2 is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled
7.2. REGION

Possible Successor Functions
ReduceDomain, AddChannels, SelectShape, AreaCenter, Connection

See Also
Erosion1, Erosion2, Opening, Closing

Alternatives
MinkowskiAdd1, MinkowskiAdd2, Dilation1, DilationGolay, DilationSeq

Module
Morphology

```
[out] HRegionX RegionDilation  HRegionX.DilationCircle
([in] VARIANT Radius )
```

```
void HOperatorSetX.DilationCircle ([in] IHObjectX Region,
[out] HUntypedObjectX RegionDilation, [in] VARIANT Radius )
```

Dilate a region with a circular structuring element.

DilationCircle applies a Minkowski addition with a circular structuring element to the input regions Region. Because the circular mask is symmetrical, this is identical to a dilation. The size of the circle used as structuring element is determined by Radius.

The operator results in enlarged regions, smoothed region boundaries, and the holes smaller than the circular mask in the interior of the region are closed. It is useful to select only values like \( \frac{3.5}{5.5}, \frac{2.5}{4.5}, \) etc. for Radius in order to avoid a translation of a region, because integer radii result in the circle having a non-integer center of gravity which is rounded to the next integer.

Attention
DilationCircle is applied to each input region separately. If gaps between different regions are to be closed, Union1 or Union2 has to be called first.

Parameter

- **Region** (input iconic) . . . . . . . . . . . . . . . . . . . region(-array) \( \rightarrow \) HRegionX / IHObjectX
  - Regions to be dilated.
- **RegionDilation** (output iconic) . . . . . . . . . . . . . . . . . region(-array) \( \rightarrow \) HRegionX / HUntypedObjectX
  - Dilated regions.
- **Radius** (input control) . . . . . . . . . . . . . . . . . . . . real \( \rightarrow \) VARIANT ( real, integer )
  - Radius of the circular structuring element.
  - Default Value : 3.5
  - Value Suggestions : Radius \( \in \{ 1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5 \} \)
  - Typical Range of Values : \( 0.5 \leq \text{Radius} \leq 0.5 \) (lin)
  - Minimal Value Step : 1.0
  - Recommended Value Step : 1.0

Result
DilationCircle returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(’noObjectResult’,<RegionResult>)
- empty region: SetSystem(’emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information
DilationCircle is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNDimNorm

Possible Successor Functions
ReduceDomain, SelectShape, AreaCenter, Connection
Dilate a region with an element from the Golay alphabet.

DilationGolay dilates a region with the selected element GolayElement from the Golay alphabet. The following structuring elements are available:


The rotation number Rotation determines which rotation of the element should be used, and whether the foreground (even) or background version (odd) of the selected element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements. The operator works by shifting the structuring element over the region to be processed (Region). For all positions of the structuring element that intersect the region, the corresponding reference point (relative to the structuring element) is added to the output region. This means that the union of all translations of the structuring element within the region is computed.

The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration \( D_2 \) is used as input for iteration \( D_2 \).

Not all values of Rotation are valid for any Golay element. For some of the values of Rotation, the resulting regions are identical to the input regions.

**Attention**

DilationGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(’noObjectResult’,<RegionResult>)
• empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

---

**Parallelization Information**

DilationGolay is *reentrant* and automatically *parallelized* (on *tuple level*).

---

**Possible Predecessor Functions**

Threshold, Regiongrowing, Connection, Union1, Watersheds, ClassNdimNorm

---

**Possible Successor Functions**

ReduceDomain, SelectShape, AreaCenter, Connection

---

**See Also**

ErosionGolay, OpeningGolay, ClosingGolay, HitOrMissGolay, ThinningGolay, ThickeningGolay, GolayElements

---

**Alternatives**

Dilation1, Dilation2, DilationSeq

---

**Module**

Morphology

---

```cpp
[out] HRegionX RegionDilation HRegionX.DilationRectangle1([in] long Width, [in] long Height )

```

*Dilate a region with a rectangular structuring element.*

DilationRectangle1 applies a dilation with a rectangular structuring element to the input regions Region. The size of the structuring rectangle is Width \times Height. The operator results in enlarged regions, and the holes smaller than the rectangular mask in the interior of the regions are closed.

DilationRectangle1 is a very fast operation because the height of the rectangle enters only logarithmically into the runtime complexity, while the width does not enter at all. This leads to excellent runtime efficiency, even in the case of very large rectangles (edge length \gt 100).

---

**Attention**

DilationRectangle1 is applied to each input region separately. If gaps between different regions are to be closed, Union1 or Union2 has to be called first.

---

**Parameter**

- **Region** (input iconic) .............................................. region(-array) \sim HRegionX / IHObjectX
  
  Regions to be dilated.

- **RegionDilation** (output iconic) ............................... region(-array) \sim HRegionX / HUntypedObjectX
  
  Dilated regions.

- **Width** (input control) ........................................... extent.x \sim long
  
  Width of the structuring rectangle.

  **Default Value:** 10  
  **Value Suggestions:** Width \in \{1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100, 150, 200\}  
  **Typical Range of Values:** 1 \leq Width \leq 1(lin)  
  **Minimal Value Step:** 1  
  **Recommended Value Step:** 10

- **Height** (input control) ......................................... extent.y \sim long
  
  Height of the structuring rectangle.

  **Default Value:** 10  
  **Value Suggestions:** Height \in \{1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100, 150, 200\}  
  **Typical Range of Values:** 1 \leq Height \leq 1(lin)  
  **Minimal Value Step:** 1  
  **Recommended Value Step:** 10

HALCON 6.0
DilationRectangle1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

--- Parallelization Information ---
DilationRectangle1 is reentrant and automatically parallelized (on tuple level).

--- Possible Predecessor Functions ---
Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

--- Possible Successor Functions ---
ReduceDomain, SelectShape, AreaCenter, Connection

--- See Also ---
GenRectangle1, GenRegionPolygonFilled

--- Alternatives ---
MinkowskiAdd1, MinkowskiAdd2, ExpandRegion, Dilation1, Dilation2, DilationCircle

--- Module ---
Morphology

Dilate a region sequentially.

DilationSeq computes the sequential dilation of the input region Region with the selected structuring element GolayElement from the Golay alphabet. This is done by executing the operator DilationGolay with all rotations of the structuring element Iterations times. The following structuring elements can be selected: ‘l’, ‘d’, ‘c’, ‘f’, ‘h’, ‘k’.

In order to compute the skeleton of a region, usually the elements ‘l’ and ‘m’ are used. Only the “foreground elements” (even rotation numbers) are used. The elements ‘l’ and ‘e’ result in unchanged output regions. The elements ‘l’, ‘m’ and ‘f2’ are identical for the foreground. The Golay elements, together with all possible rotations, are described with the operator GolayElements.

--- Parameter ---

- **Region** (input iconic) ................................. region(-array) \(\rightsquigarrow\) HRegionX / IHObjectX
  Regions to be dilated.
- **RegionDilation** (output iconic) .................... region(-array) \(\rightsquigarrow\) HRegionX / HUntypedObjectX
  Dilated regions.
- **GolayElement** (input control) ........................ string \(\rightsquigarrow\) String
  Structuring element from the Golay alphabet.
  Default Value: ‘h’
- **Iterations** (input control) ............................ integer \(\rightsquigarrow\) long
  Number of iterations.
  Default Value: 1
  Value Suggestions: Iterations \(\in\) \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}
  (lin) Minimal Value Step: 1
  Recommended Value Step: 1
DilationSeq returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

---

**Parallelization Information**

DilationSeq is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

---

**Possible Successor Functions**

ReduceDomain, SelectShape, AreaCenter, Connection

---

**See Also**

ErosionSeq, HitOrMissSeq, ThinningSeq

---

**Alternatives**

Dilation1, Dilation2, DilationGolay

---

**Module**

Morphology

---

```plaintext
[out] HRegionX RegionErosion HRegionX.Erosion1
([in] HRegionX StructElement, [in] long Iterations )

void HOperatorSetX.Erosion1 ([in] IHObjectX Region,
[in] IHObjectX StructElement, [out] HUntypedObjectX RegionErosion,
[in] long Iterations )
```

---

Erode a region.

Erosion1 erodes the input regions with a structuring element. By applying Erosion1 to a region, its boundary gets smoothed. In the process, the area of the region is reduced. Furthermore, connected regions may be split. Such regions, however, remain logically one region. The erosion is a set-theoretic region operation. It uses the intersection operation.

Let \( M \) (StructElement) and \( R \) (Region) be two regions, where \( M \) is the structuring element and \( R \) is the region to be processed. Furthermore, let \( m \) be a point in \( M \). Then the displacement vector \( \vec{v}_m = (dx, dy) \) is defined as the difference of the center of gravity of \( M \) and the vector \( \vec{r} \). Let \( t_{\vec{v}_m}(R) \) denote the translation of a region \( R \) by a vector \( \vec{v} \). Then

\[
\text{Erosion1}(R, M) := \bigcap_{m \in M} t_{-\vec{v}_m}(R).
\]

For each point \( m \) in \( M \) a translation of the region \( R \) is performed. The intersection of all these translations is the erosion of \( R \) with \( M \). Erosion1 is similar to the operator MinkowskiSub1, the difference is that in Erosion1 the structuring element is mirrored at the origin. The position of StructElement is meaningless, since the displacement vectors are determined with respect to the center of gravity of \( M \).

The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \). From the above definition it follows that the maximum region is generated in case of an empty structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.
Parameter

- **Region** (input iconic) .......................... region(-array) ~ HRegionX / IHObjectX
  Regions to be eroded.
- **StructElement** (input iconic) .......................... region ~ HRegionX / IHObjectX
  Structuring element.
- **RegionErosion** (output iconic) ...................... region(-array) ~ HRegionX / HUntypedObjectX
  Eroded regions.
- **Iterations** (input control) ...................... integer ~ long
  Number of iterations.
  Default Value : 1
  Value Suggestions : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  (in)Minimal Value Step : 1
  Recommended Value Step : 1

Result

Erosion1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

Erosion1 is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenRegionPolygonFilled

Possible Successor Functions

Connection, ReduceDomain, SelectShape, AreaCenter

See Also

TransposeRegion

Alternatives

MinkowskiSub1, MinkowskiSub2, Erosion2, ErosionGolay, ErosionSeq

Module

Morphology

```
[out] HRegionX RegionErosion HRegionX.Erosion2
  ([in] HRegionX StructElement, [in] long Row, [in] long Column,
  [in] long Iterations )

void HOperatorSetX.Erosion2 ([in] IHObjectX Region,
  [in] IHObjectX StructElement, [out] HUntypedObjectX RegionErosion,
```

Erode a region (using a reference point).

Erosion2 erodes the input regions with a structuring element (StructElement) having the reference point (Row,Column). Erosion2 has a similar effect as Erosion1, the difference is that the reference point of the structuring element can be chosen arbitrarily. The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration n – 1 is used as input for iteration n.

A maximum region is generated in case of an empty structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.
7.2. REGION

Parameter

- **Region** (input iconic) ................................. region(-array) \( \sim HRegionX / IObjectX \)
  Regions to be eroded.

- **StructElement** (input iconic) ........................ region \( \sim HRegionX / IObjectX \)
  Structuring element.

- **RegionErosion** (output iconic) ....................... region(-array) \( \sim HRegionX /HUntypedObjectX \)
  Eroded regions.

- **Row** (input control) ..................................... point.y \( \sim \) long
  Row coordinate of the reference point.
  Default Value : 0
  Typical Range of Values : \( 0 \leq \text{Row} \leq 0 \) (lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

- **Column** (input control) ................................. point.x \( \sim \) long
  Column coordinate of the reference point.
  Default Value : 0
  Typical Range of Values : \( 0 \leq \text{Column} \leq 0 \) (lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

- **Iterations** (input control) ............................ integer \( \sim \) long
  Number of iterations.
  Default Value : 1
  Value Suggestions : Iterations \( \in \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50 \} \)
  (lin) Minimal Value Step : 1
  Recommended Value Step : 1

Result

Erosion2 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Paralleization Information

Erosion2 is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Regiongrowing, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

TransposeRegion, GenCircle, GenRectangle2, GenRegionPolygon

Alternatives

MinkowskiSub2, MinkowskiSub1, Erosion1, ErosionGolay, ErosionSeq

Module

Morphology

```c
[out] HRegionX RegionErosion HRegionX.ErosionCircle
([in] VARIANT Radius )

void HOperatorSetX.ErosionCircle ([in] IObjectX Region,
[out] HUntypedObjectX RegionErosion, [in] VARIANT Radius )
```

Erode a region with a circular structuring element.
ErosionCircle applies a Minkowski subtraction with a circular structuring element to the input regions Region. Because the circular mask is symmetrical, this is identical to an erosion. The size of the circle used as structuring element is determined by Radius.

The operator results in reduced regions, smoothed region boundaries, and the regions smaller than the circular mask are eliminated. It is useful to select only values like 3.5, 5.5, etc. for Radius in order to avoid a translation of a region, because integer radii result in a circle having a non-integer center of gravity which is rounded to the next integer.

Parameter

- **Region** (input iconic) \( \rightarrow \) HRegionX / IHObjectX
  Regions to be eroded.

- **RegionErosion** (output iconic) \( \rightarrow \) HRegionX / HUntypedObjectX
  Eroded regions.

- **Radius** (input control) \( \rightarrow \) VARIANT (real, integer)
  Radius of the circular structuring element.
  - Default Value: 3.5
  - Value Suggestions: \( \text{Radius} \in \{1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5\} \)
  - Typical Range of Values: \( 0.5 \leq \text{Radius} \leq 0.5(\text{lin}) \)
  - Minimal Value Step: 1.0
  - Recommended Value Step: 1.0

Result

ErosionCircle returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

ErosionCircle is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Watersheds, ClassNdimNorm

Possible Successor Functions

Connection, ReduceDomain, SelectShape, AreaCenter

See Also

GenCircle, DilationCircle, ClosingCircle, OpeningCircle

Alternatives

MinkowskiSub1

Module

Morphology

```
[out] HRegionX RegionErosion HRegionX.ErosionGolay

void HOperatorSetX.ErosionGolay ([in] IHObjectX Region,
[out] HUntypedObjectX RegionErosion, [in] String GolayElement,
[in] long Iterations, [in] long Rotation )
```

Erode a region with an element from the Golay alphabet.

ErosionGolay erodes a region with the selected element GolayElement from the Golay alphabet. The following structuring elements are available:

'1', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The rotation number Rotation determines which rotation of the element should be used, and whether the foreground (even) or background version (odd) of the selected element should be used. The Golay elements, together
with all possible rotations, are described with the operator \texttt{GolayElements}. The operator works by shifting the structuring element over the region to be processed (Region). For all positions of the structuring element fully contained in the region, the corresponding reference point (relative to the structuring element) is added to the output region. This means that the intersection of all translations of the structuring element within the region is computed.

The parameter \texttt{Iterations} determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \).

\textbf{Attention}

Not all values of \texttt{Rotation} are valid for any Golay element. For some of the values of \texttt{Rotation}, the resulting regions are identical to the input regions.

\begin{itemize}
  \item \texttt{Region} (input iconic) \begin{itemize}
    \item \texttt{region} (array) \begin{itemize}
      \item \texttt{HRegionX} / \texttt{IObjectX}
    \end{itemize}
  \end{itemize}
  \item \texttt{RegionErosion} (output iconic) \begin{itemize}
    \item \texttt{region} (array) \begin{itemize}
      \item \texttt{HRegionX} / \texttt{HUntypedObjectX}
    \end{itemize}
  \end{itemize}
  \item \texttt{GolayElement} (input control) \begin{itemize}
    \item \texttt{String} \begin{itemize}
      \item \texttt{Structuring element from the Golay alphabet.}
    \end{itemize}
  \end{itemize}
  \item \texttt{Iterations} (input control) \begin{itemize}
    \item \texttt{integer} \begin{itemize}
      \item \texttt{Number of iterations.}
    \end{itemize}
  \end{itemize}
  \item \texttt{Rotation} (input control) \begin{itemize}
    \item \texttt{integer} \begin{itemize}
      \item \texttt{Rotation of the Golay element} Depending on the element, not all rotations are valid.
    \end{itemize}
  \end{itemize}
\end{itemize}

\textbf{Result}

\texttt{ErosionGolay} returns \texttt{TRUE} if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: \texttt{SetSystem(’noObjectResult’,<RegionResult>)}
- empty region: \texttt{SetSystem(’emptyRegionResult’,<RegionResult>)}

Otherwise, an exception is raised.

\textbf{Parallelization Information}

\texttt{ErosionGolay} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

\textbf{Possible Predecessor Functions}

Threshold, RegionGrowing, Watersheds, ClassNDimNorm

\textbf{Possible Successor Functions}

ReduceDomain, SelectShape, AreaCenter, Connection

\textbf{See Also}

DilationGolay, OpeningGolay, ClosingGolay, HitOrMissGolay, ThinningGolay,
ThickeningGolay, GolayElements

\textbf{Alternatives}

ErosionSeq, Erosion1, Erosion2

\textbf{Module}

\texttt{Morphology}
Erode a region with a rectangular structuring element.

ErosionRectangle1 applies an erosion with a rectangular structuring element to the input regions Region. The size of the structuring rectangle is Width × Height. The operator results in reduced regions, and the areas smaller than the rectangular mask are eliminated.

ErosionRectangle1 is a very fast operation because the height of the rectangle enters only logarithmically into the runtime complexity, while the width does not enter at all. This leads to excellent runtime efficiency, even in the case of very large rectangles (edge length > 100).

Regions containing small connecting strips between large areas are separated only seemingly. They remain logically one region.

### Parameter

- **Region** (input iconic) .............................. region(-array) \( \sim \) HRegionX / IHObjectX
  
  Regions to be eroded.

- **RegionErosion** (output iconic) ...................... region(-array) \( \sim \) HRegionX / HUntypedObjectX
  
  Eroded regions.

- **Width** (input control) ................................ extent.x \( \sim \) long
  
  Width of the structuring rectangle.

  Default Value : 10

  Value Suggestions : Width \( \in \) \{1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100\}

  Typical Range of Values : \(1 \leq \text{Width} \leq 1\text{(lin)}\)

  Minimal Value Step : 1

  Recommended Value Step : 1

- **Height** (input control) .............................. extent.y \( \sim \) long
  
  Height of the structuring rectangle.

  Default Value : 10

  Value Suggestions : Height \( \in \) \{1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100\}

  Typical Range of Values : \(1 \leq \text{Height} \leq 1\text{(lin)}\)

  Minimal Value Step : 1

  Recommended Value Step : 1

### Result

ErosionRectangle1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’, <RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’, <RegionResult>)

Otherwise, an exception is raised.

### Parallelization Information

ErosionRectangle1 is **reentrant** and automatically parallelized (on tuple level).

### Possible Predecessor Functions

Threshold, RegionGrowing, Watersheds, ClassNdimNorm

### Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

### See Also

GenRectangle1

### Alternatives

Erosion1, MinkowskiSub1

### Module

Morphology
Erode a region sequentially.

ErosionSeq computes the sequential erosion of the input region Region with the selected structuring element GolayElement from the Golay alphabet. This is done by executing the operator ErosionGolay with all rotations of the structuring element Iterations times. The following structuring elements can be selected:

'1', 'd', 'c', 'f', 'h', 'k'.

Only the “foreground elements” (even rotation numbers) are used. The elements 'i' and 'e' result in unchanged output regions. The elements '1', 'm' and 'f2' are identical for the foreground. The Golay elements, together with all possible rotations, are described with the operator GolayElements.

Parameter

- **Region** (input iconic) region(-array) `HRegionX` / `IHObjectX` Regions to be eroded.
- **RegionErosion** (output iconic) region(-array) `HRegionX` / `HUntypedObjectX` Eroded regions.
- **GolayElement** (input control) `String` Structuring element from the Golay alphabet.
  - Default Value: 'h'
  - Value List: `GolayElement` = {'1', 'd', 'c', 'f', 'h', 'k'}
- **Iterations** (input control) integer Number of iterations.
  - Default Value: 1
  - Value Suggestions: `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  - (lin)Minimal Value Step: 1
  - Recommended Value Step: 1

Result

ErosionSeq returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: `SetSystem('noObjectResult',<RegionResult>)`
- empty region: `SetSystem('emptyRegionResult',<RegionResult>)`

Otherwise, an exception is raised.

Parallelization Information

ErosionSeq is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Watersheds, ClassNdimNorm

Possible Successor Functions

Connection, ReduceDomain, SelectShape, AreaCenter

See Also

DilationSeq, HitOrMissSeq, ThinningSeq

Alternatives

ErosionGolay, Erosion1, Erosion2

Module

Morphology
Perform a closing after an opening with multiple structuring elements.

Fitting performs an Opening and a Closing successively on the input regions. The eight structuring elements normally used for this operation can be generated with the operator GenStructElements. However, other user-defined structuring elements can also be used. Let \( R \) be the input region(s) and let \( M_i \) denote the structuring elements. Furthermore, let \( P \) be the result of the opening and \( Q \) be the final result. Then the operator can be formalized as follows:

\[
P = \bigcup_{i=1}^{n}(R \circ M_i)
\]
\[
Q = \bigcap_{i=1}^{n}(P \bullet M_i)
\]

Regions larger than the structuring elements are preserved, while small gaps are closed.

Parameter

- \textbf{Region} (input iconic) \text{:} \textit{region(-array)} \rightarrow HRegionX / IHObjectX
  Regions to be processed.
- \textbf{StructElements} (input iconic) \text{:} \textit{region(-array)} \rightarrow HRegionX / IHObjectX
  Structuring elements.
- \textbf{RegionFitted} (output iconic) \text{:} \textit{region(-array)} \rightarrow HRegionX / HUntypedObjectX
  Fitted regions.

Result

Fitting returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(’noObjectResult’,<RegionResult>)
- empty region: SetSystem(’emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

Fitting is reentrant and processed without parallelization.

Possible Predecessor Functions

GenStructElements, GenRegionPoints

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

Alternatives

Opening, Closing, Connection, SelectShape

Module

Morphology

Generate standard structuring elements.
GenStructElements serves to generate eight structuring elements normally used in the operator Fitting. The default value 'noise' of the parameter Type generates elements especially suited for the elimination of noise.

![Structuring Elements Diagram]

- **StructElements** (output iconic) region(-array) → HRegionX/HUntypedObjectX
  Generated structuring elements.

- **Type** (input control) string → String
  Type of structuring element to generate.
  **Default Value**: 'noise'
  **Value List**: 'noise'

- **Row** (input control) point.y → long
  Row coordinate of the reference point.
  **Default Value**: 1
  **Value Suggestions**: 0, 1, 10, 50, 100, 200, 300, 400

- **Column** (input control) point.x → long
  Column coordinate of the reference point.
  **Default Value**: 1
  **Value Suggestions**: 0, 1, 10, 50, 100, 200, 300, 400

**Result**

GenStructElements returns TRUE if all parameters are correct. Otherwise, an exception is raised.

**Parallelization Information**

GenStructElements is reentrant and processed without parallelization.

**Possible Successor Functions**

Fitting, HitOrMiss, Opening, Closing, Erosion2, Dilation2

**See Also**

GolayElements

**Module**

Morphology

```c
void HRegionX.GolayElements ([out] HRegionX StructElement2,
[in] String GolayElement, [in] long Rotation, [in] long Row,
[in] long Column )

void HOperatorSetX.GolayElements
([out] HUntypedObjectX StructElement1, [out] HUntypedObjectX StructElement2,
[in] String GolayElement, [in] long Rotation, [in] long Row,
[in] long Column )
```

Generate the structuring elements of the Golay alphabet.
GolayElements generates the structuring elements from the Golay alphabet. The parameter GolayElement determines the name of the structuring element, while Rotation determines its rotation. The structuring elements are intended for use in HitOrMiss: In StructElement1 the structuring element for the foreground is returned, while in StructElement2 the structuring element for the background is returned. Row and Column determine the reference point of the structuring element.

The rotations are numbered from 0 to 15. This does not mean, however, that there are 16 different rotations: Even values denote rotations of the foreground elements, while odd values denote rotations of the background elements.

For GolayElements only even values are accepted, and determine the Golay element for StructElement1. The next larger odd value is used for StructElement2. There are no rotations for the Golay elements 'h' and 'i'. Therefore, only the values 0 and 1 are possible as “rotations” (and hence only 0 for GolayElements). The element 'e' has only four possible rotations, and hence the rotation must be between 0 and 7 (for GolayElements the values 0, 2, 4, or 6 must be used).

The tables below show the elements of the Golay alphabet with all possible rotations. The characters used have the following meaning:
- Foreground pixel
- Background pixel
- Don’t care pixel

The names of the elements and their rotation numbers are displayed below the respective element. The elements with even number contain the foreground pixels, while the elements with odd numbers contain the background pixels.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{h}(0,1)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{i}(0,1)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{e}(0,1)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{e}(2,3)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{e}(4,5)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{e}(6,7)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{l}(0,1)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{l}(2,3)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{l}(4,5)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{l}(6,7)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{l}(8,9)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{l}(10,11)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{l}(12,13)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{l}(14,15)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{m}(0,1)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{m}(2,3)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{m}(4,5)\]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\text{m}(6,7)\]
**StructElement1** (output iconic) .................................. region ~ HRegionX / HUntypedObjectX
Structuring element for the foreground.

**StructElement2** (output iconic) .................................. region ~ HRegionX / HUntypedObjectX
Structuring element for the background.

**GolayElement** (input control) ................................... string ~ String
Name of the structuring element.

Default Value : 'l'
Value List : GolayElement ∈ {'l', 'm', 'd', 'c', 'i', 'f', 'f2', 'h', 'k'}

**Rotation** (input control) ........................................ integer ~ long
Rotation of the Golay element Depending on the element, not all rotations are valid.

Default Value : 0
Value List : Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14}

**Row** (input control) ........................................... point.y ~ long
Row coordinate of the reference point.

Default Value : 16
Value Suggestions : Row ∈ {0, 16, 32, 128, 256}
Typical Range of Values : 0 ≤ Row ≤ 0(lin)
Minimal Value Step : 1
Recommended Value Step : 1

**Column** (input control) ......................................... point.x ~ long
Column coordinate of the reference point.

Default Value : 16
Value Suggestions : Column ∈ {0, 16, 32, 128, 256}
Typical Range of Values : 0 ≤ Column ≤ 0(lin)
Minimal Value Step : 1
Recommended Value Step : 1

---

Result

GolayElements returns TRUE if all parameters are correct. Otherwise, an exception is raised.

---

Parallelization Information

GolayElements is reentrant and processed without parallelization.

---

Possible Successor Functions

HitOrMiss

---

See Also

DilationGolay, ErosionGolay, OpeningGolay, ClosingGolay, HitOrMissGolay, ThickeningGolay

---

Alternatives

GenRegionPoints, GenStructElements, GenRegionPolygonFilled

---

Bibliography

Hit-or-miss operation for regions.

HitOrMiss performs the hit-or-miss-transformation. First, an erosion with the structuring element StructElement1 is done on the input region Region. Then an erosion with the structuring element StructElement2 is performed on the complement of the input region. The intersection of the two resulting regions is the result RegionHitMiss of HitOrMiss.

The hit-or-miss-transformation selects precisely the points for which the conditions given by the structuring elements StructElement1 and StructElement2 are fulfilled. StructElement1 determines the condition for the foreground pixels, while StructElement2 determines the condition for the background pixels. In order to obtain sensible results, StructElement1 and StructElement2 must fit like key and lock. In any case, StructElement1 and StructElement2 must be disjunct.

Row and Column determine the reference point of the structuring elements.

Structuring elements (StructElement1, StructElement2) can be generated by calling operators like GenStructElements, GenRegionPoints, etc.

Parameter

- **Region** (input iconic) ........................................ region(-array) ː HRegionX / IHObjectX
  Regions to be processed.
- **StructElement1** (input iconic) ............................ region ː HRegionX / IHObjectX
  Erosion mask for the input regions.
- **StructElement2** (input iconic) ............................ region ː HRegionX / IHObjectX
  Erosion mask for the complements of the input regions.
- **RegionHitMiss** (output iconic) .......................... region(-array) ː HRegionX / HUntypedObjectX
  Result of the hit-or-miss operation.
- **Row** (input control) ......................................... point.y ː long
  Row coordinate of the reference point.
- **Column** (input control) ..................................... point.x ː long
  Column coordinate of the reference point.

Result

HitOrMiss returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

HitOrMiss is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

GolayElements, GenStructElements, Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm
Possible Successor Functions

Difference, ReduceDomain, SelectShape, AreaCenter, Connection

See Also

Thinning, Thickening, GenRegionPoints, GenRegionPolygonFilled

Alternatives

HitOrMissGolay, HitOrMissSeq, Erosion2, Dilation2

Module  Morphology

```plaintext
[out]  HRegionX  RegionHitMiss  HRegionX.HitOrMissGolay
([in]  String  GolayElement, [in]  long  Rotation )

void  HOperatorSetX.HitOrMissGolay  ([in]  IObjectX  Region,
[out]  HUntypedObjectX  RegionHitMiss, [in]  String  GolayElement,
[in]  long  Rotation )
```

Hit-or-miss operation for regions using the Golay alphabet.

HitOrMissGolay performs the hit-or-miss-transformation for the input regions Region (using structuring elements from the Golay alphabet). First, an erosion with the foreground of the structuring element GolayElement is done on the input region Region. Then an erosion with the background of the structuring element GolayElement is performed on the complement of the input region. The intersection of the two resulting regions is the result RegionHitMiss of HitOrMissGolay. The following structuring elements are available:

' l ', ' m ', ' d ', ' c ', ' e ', ' i ', ' f ', ' f2 ', ' h ', ' k '.

The rotation number Rotation determines which rotation of the element should be used. The hit-or-miss-transformation selects precisely the points for which the conditions given by the selected Golay element are fulfilled.

Not all values of Rotation are valid for any Golay element.

Parameter

- **Region** (input iconic)  
  regions-array  
  Regions to be processed.

- **RegionHitMiss** (output iconic)  
  regions-array  
  Result of the hit-or-miss operation.

- **GolayElement** (input control)  
  string  
  Structuring element from the Golay alphabet.

  Default Value: ' h '  
  Value List: GolayElement ∈ { ' l ', ' m ', ' d ', ' c ', ' e ', ' i ', ' f ', ' f2 ', ' h ', ' k ' }

- **Rotation** (input control)  
  integer  
  Rotation of the Golay element. Depending on the element, not all rotations are valid.

  Default Value: 0  
  Value List: Rotation ∈ { 0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15 }

HitOrMissGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem ( ' noObjectResult ', <RegionResult> )
- empty region: SetSystem ( ' emptyRegionResult ', <RegionResult> )

Otherwise, an exception is raised.

Parallelization Information

HitOrMissGolay is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Regiongrowing, Connection, Union1, Watersheds, ClassNdimNorm
Possible Successor Functions
ReduceDomain, SelectShape, AreaCenter, Connection

See Also
ErosionGolay, DilationGolay, OpeningGolay, ClosingGolay, ThinningGolay, ThickeningGolay, GolayElements

Alternatives
HitOrMissSeq, HitOrMiss

Module
Morphology

Hit-or-miss operation for regions using the Golay alphabet (sequential).
HitOrMissGolay performs the hit-or-miss-transformation for the input regions Region using all rotations of a structuring element from the Golay alphabet. The result of the operator is the union of all intermediate results of the respective rotations. The following structuring elements are available: 'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.
The Golay elements, together with all possible rotations, are described with the operator GolayElements.

Parameter

- **Region** (input iconic) .........................................................region(-array) ¬ HRegionX / IHObjectX
  Regions to be processed.
- **RegionHitMiss** (output iconic) .................................region(-array) ¬ HRegionX / HUntypedObjectX
  Result of the hit-or-miss operation.
- **GolayElement** (input control) .........................................................string ¬ String
  Structuring element from the Golay alphabet.
  Default Value : 'h'
  Value List: GolayElement ∈ { 'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k' }

Result
HitOrMissSeq returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information
HitOrMissSeq is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

Possible Successor Functions
ReduceDomain, SelectShape, AreaCenter, Connection

See Also

Alternatives
HitOrMissGolay, HitOrMiss

Module
Morphology
Perform a Minkowski addition on a region.

MinkowskiAdd1 dilates the input regions with a structuring element. By applying MinkowskiAdd1 to a region, its boundary gets smoothed. In the process, the area of the region is enlarged. Furthermore, disconnected regions may be merged. Such regions, however, remain logically distinct region. The Minkowski addition is a set-theoretic region operation. It is based on translations and union operations.

Let $\mathbf{C}$ ($\text{StructElement}$) and $\mathbf{A}$ ($\text{Region}$) be two regions, where $\mathbf{C}$ is the structuring element and $\mathbf{A}$ is the region to be processed. Furthermore, let $m$ be a point in $\mathbf{C}$. Then the displacement vector $\vec{v}_m = (dx, dy)$ is defined as the difference of the center of gravity of $\mathbf{C}$ and the vector $\vec{m}$. Let $t_{v_m}(\mathbf{R})$ denote the translation of a region $\mathbf{R}$ by a vector $\vec{v}$. Then

$$\text{MinkowskiAdd1}(\mathbf{R}, \mathbf{M}) := \bigcup_{m \in \mathbf{M}} t_{v_m}(\mathbf{R})$$

For each point $m$ in $\mathbf{M}$ a translation of the region $\mathbf{R}$ is performed. The union of all these translations is the Minkowski addition of $\mathbf{R}$ with $\mathbf{M}$. MinkowskiAdd1 is similar to the operator Dilation1, the difference is that in Dilation1 the structuring element is mirrored at the origin. The position of StructElement is meaningless, since the displacement vectors are determined with respect to the center of gravity of $\mathbf{C}$.

The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration $n - 1$ is used as input for iteration $n$. From the above definition it follows that an empty region is generated in case of an empty structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

**Attention**

A Minkowski addition always results in enlarged regions. Closely spaced regions which may touch or overlap as a result of the dilation are still treated as two separate regions. If the desired behavior is to merge them into one region, the operator Union1 has to be called first.

<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Region</strong> (input iconic) .................. region(-array) $\rightsquigarrow$ HRegionX / IHOBJECTX</td>
</tr>
<tr>
<td><strong>StructElement</strong> (input iconic) .................. region $\rightsquigarrow$ HRegionX / IHOBJECTX</td>
</tr>
<tr>
<td><strong>RegionMinkAdd</strong> (output iconic) ................. region(-array) $\rightsquigarrow$ HRegionX / HUntypedObjectX</td>
</tr>
<tr>
<td><strong>Iterations</strong> (input control) .................. integer $\rightsquigarrow$ long</td>
</tr>
<tr>
<td><strong>Default Value</strong> : 1</td>
</tr>
<tr>
<td><strong>Value Suggestions</strong> : Iterations $\in {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}$ (lin)</td>
</tr>
<tr>
<td><strong>Minimal Value Step</strong> : 1</td>
</tr>
<tr>
<td><strong>Recommended Value Step</strong> : 1</td>
</tr>
</tbody>
</table>

**Result**

MinkowskiAdd1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.
### Parallelization Information

*MinkowskiAdd1* is reentrant and automatically parallelized (on tuple level).

### Possible Predecessor Functions

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNDimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

### Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

### See Also

TransposeRegion, MinkowskiSub1

### Alternatives

*MinkowskiAdd2*, *Dilation1*

## Module

Morphology

```c
[out] HRegionX RegionMinkAdd HRegionX.MinkowskiAdd2
([in] HRegionX StructElement, [in] long Row, [in] long Column,
[in] long Iterations )

void HOperatorSetX.MinkowskiAdd2 ([in] IQObjectX Region,
[in] IQObjectX StructElement, [out] HUntypedObjectX RegionMinkAdd,
```

Dilate a region (using a reference point).

*MinkowskiAdd2* computes the Minkowski addition of the input regions with a structuring element (*StructElement*) having the reference point (*Row*, *Column*). *MinkowskiAdd2* has a similar effect as *MinkowskiAdd1*, the difference is that the reference point of the structuring element can be chosen arbitrarily. The parameter *Iterations* determines the number of iterations which are to be performed with the structuring element. The result of iteration $D_2$ is used as input for iteration $D_2$.

An empty region is generated in case of an empty structuring element.

Structuring elements (*StructElement*) can be generated with operators such as *GenCircle*, *GenRectangle1*, *GenRectangle2*, *GenEllipse*, *GenRegionPolygon*, *GenRegionPoints*, etc.

### Attention

A Minkowski addition always results in enlarged regions. Closely spaced regions which may touch or overlap as a result of the dilation are still treated as two separate regions. If the desired behavior is to merge them into one region, the operator *Union1* has to be called first.

### Parameter

- **Region** (input iconic) ..................... region(-array) $\rightarrow$ HRegionX / IQObjectX
  
  Regions to be dilated.

- **StructElement** (input iconic) .................. region $\rightarrow$ HRegionX / IQObjectX
  
  Structuring element.

- **RegionMinkAdd** (output iconic) .............. region(-array) $\rightarrow$ HRegionX / HUntypedObjectX
  
  Dilated regions.

- **Row** (input control) .......................... point.y $\sim$ long
  
  Row coordinate of the reference point.

  **Typical Range of Values**: $1 \leq \text{Row} \leq 1(\text{lin})$

  **Minimal Value Step**: 1

  **Recommended Value Step**: 1

- **Column** (input control) ........................ point.x $\sim$ long
  
  Column coordinate of the reference point.

  **Typical Range of Values**: $1 \leq \text{Column} \leq 1(\text{lin})$

  **Minimal Value Step**: 1

  **Recommended Value Step**: 1
**Iterations** (input control) .................................................. integer ~ long
Number of iterations.

**Default Value :** 1

**Value Suggestions :** Iterations \( \in \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50 \} \)

**Result**

MinkowskiAdd2 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

**Parallelization Information**

MinkowskiAdd2 is **reentrant** and automatically **parallelized** (on tuple level).

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

**Possible Successor Functions**

ReduceDomain, SelectShape, AreaCenter, Connection

**See Also**

TransposeRegion

**Alternatives**

MinkowskiAdd1, Dilation1

**Module**

Morphology

---

### Erode a region.

MinkowskiSub1 computes the Minkowski subtraction of the input regions with a structuring element. By applying MinkowskiSub1 to a region, its boundary gets smoothed. In the process, the area of the region is reduced. Furthermore, connected regions may be split. Such regions, however, remain logically one region. The Minkowski subtraction is a set-theoretic region operation. It uses the intersection operation.

Let \( M \) (StructElement) and \( R \) (Region) be two regions, where \( M \) is the structuring element and \( R \) is the region to be processed. Furthermore, let \( m \) be a point in \( M \). Then the displacement vector \( \vec{v}_m = (dx, dy) \) is defined as the difference of the center of gravity of \( M \) and the vector \( \vec{m} \). Let \( t_{v_m}(R) \) denote the translation of a region \( R \) by a vector \( \vec{v} \). Then

\[
\text{MinkowskiSub1}(R, M) := \bigcap_{m \in M} t_{v_m}(R)
\]

For each point \( m \) in \( M \) a translation of the region \( R \) is performed. The intersection of all these translations is the Minkowski subtraction of \( R \) with \( M \). MinkowskiSub1 is similar to the operator Erosion1, the difference is that in Erosion1 the structuring element is mirrored at the origin. The position of StructElement is meaningless, since the displacement vectors are determined with respect to the center of gravity of \( M \).

The parameter **Iterations** determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \). From the above definition it follows that the maximum region is generated in case of an empty structuring element.
Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

Parameter

- **Region** (input iconic) .......... region(-array) ~ HRegionX / HObjectX
  Regions to be eroded.

- **StructElement** (input iconic) .......... region ~ HRegionX / HObjectX
  Structuring element.

- **RegionMinkSub** (output iconic) .......... region(-array) ~ HRegionX / HUntypedObjectX
  Eroded regions.

- **Iterations** (input control) .......... integer ~ long
  Number of iterations.
  Default Value : 1
  Value Suggestions : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  (lin) Minimal Value Step : 1
  Recommended Value Step : 1

Result

MinkowskiSub1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

MinkowskiSub1 is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Regiongrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

TransposeRegion

Alternatives

MinkowskiSub2, Erosion1

Module

Morphology

```c
[out] HRegionX RegionMinkSub HRegionX.MinkowskiSub2
([in] HRegionX StructElement, [in] long Row, [in] long Column,
[in] long Iterations )
```

```c
void HOperatorSetX.MinkowskiSub2 ([in] HObjectX Region,
[in] HObjectX StructElement, [out] HUntypedObjectX RegionMinkSub,
```

Erode a region (using a reference point).

MinkowskiSub2 computes the Minkowski subtraction of the input regions with a structuring element (StructElement) having the reference point (Row,Column). MinkowskiSub2 has a similar effect as MinkowskiSub1, the difference is that the reference point of the structuring element can be chosen arbitrarily. The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \).

A maximum region is generated in case of an empty structuring element.
Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

Parameter

- **Region** (input iconic)  
  region(-array)  
  Regions to be eroded.

- **StructElement** (input iconic)  
  region  
  Structuring element.

- **RegionMinkSub** (output iconic)  
  Eroded regions.

- **Row** (input control)  
  Row coordinate of the reference point.
  
  Default Value: 0
  Value Suggestions: Row ∈ {0, 10, 16, 32, 64, 100, 128}
  Typical Range of Values: 0 ≤ Row ≤ 0(lin)
  Minimal Value Step: 1
  Recommended Value Step: 1

- **Column** (input control)  
  Column coordinate of the reference point.
  
  Default Value: 0
  Value Suggestions: Column ∈ {0, 10, 16, 32, 64, 100, 128}
  Typical Range of Values: 0 ≤ Column ≤ 0(lin)
  Minimal Value Step: 1
  Recommended Value Step: 1

- **Iterations** (input control)  
  Number of iterations.
  
  Default Value: 1
  Value Suggestions: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  (lin)Minimal Value Step: 1
  Recommended Value Step: 1

---

Result

MinkowskiSub2 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

---

Parallelization Information

MinkowskiSub2 is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Watersheds, ClassNDimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

GenCircle, GenRectangle2, GenRegionPolygon

Alternatives

MinkowskiSub1, Erosion1, Erosion2, ErosionGolay, ErosionSeq

Module

Morphology
**Compute the union of BottomHat and TopHat.**

MorphHat computes the union of the regions that are removed by an Opening operation with the regions that are added by a Closing operation. Hence this is the union of the results of TopHat and BottomHat. The position of StructElement does not influence the result.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

---

**Attention**

The individual regions are processed separately.

---

**Parameter**

- **Region** (input iconic) ........................................... region(-array) \(\rightarrow\) HRegionX / IHObjectX
  - Regions to be processed.

- **StructElement** (input iconic) ................................. region \(\rightarrow\) HRegionX / IHObjectX
  - Structuring element (position-invariant).

- **RegionTopHat** (output iconic) ............................... region(-array) \(\rightarrow\) HRegionX / HUntypedObjectX
  - Union of top hat and bottom hat.

---

**Result**

MorphHat returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

---

**Parallelization Information**

MorphHat is **reentrant** and automatically **parallelized** (on tuple level).

---

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

---

**Possible Successor Functions**

ReduceDomain, SelectShape, AreaCenter

---

**See Also**

Opening, Closing

---

**Alternatives**

TopHat, BottomHat, Union2

---

**Module**

Morphology

---

**Compute the morphological skeleton of a region.**

MorphSkeleton computes the skeleton of the input regions (Region) using morphological transformations. The computation yields a disconnected skeleton (gaps in the diagonals) having a width of one or two pixels. The calculation uses the Golay element ‘h’, i.e., an 8-neighborhood. This is equivalent to the maximum-norm.
Parameter

▷ **Region** (input iconic) ........................................... region(-array)  \(\sim\) HRegionX / IHObjectX
Regions to be processed.

▷ **RegionSkiz** (output iconic) ................................... region(-array)  \(\sim\) HRegionX / HUntypedObjectX
Resulting morphological skeleton.

Result

MorphSkeleton returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

MorphSkeleton is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

Possible Successor Functions

Skeleton, ReduceDomain, SelectShape, AreaCenter, Connection

See Also

ThinningSeq, MorphSkiz

Alternatives

Skeleton, Thinning

Module

Morphology

```plaintext
[out] HRegionX RegionSkiz HRegionX.MorphSkiz ([in] VARIANT Iterations1,
[in] VARIANT Iterations2 )

void HOperatorSetX.MorphSkiz ([in] IHObjectX Region,
[out] HUntypedObjectX RegionSkiz, [in] VARIANT Iterations1,
[in] VARIANT Iterations2 )
```

**Thinning of a region.**

MorphSkiz first performs a sequential thinning (ThinningSeq) of the input region with the element ‘l’ of the Golay alphabet. The number of iterations is determined by the parameter Iterations1. Then a sequential thinning of the resulting region with the element ‘e’ of the Golay alphabet is carried out. The number of iterations for this step is determined by the parameter Iterations2. The skiz operation serves to compute a kind of skeleton of the input regions, and to prune the branches of the resulting skeleton. If the skiz operation is applied to the complement of the region, the region and the resulting skeleton are separated.

If very large values or ‘maximal’ are passed for Iterations1 or Iterations2, the processing stops if no more changes occur.

Parameter

▷ **Region** (input iconic) ........................................... region(-array)  \(\sim\) HRegionX / IHObjectX
Regions to be thinned.

▷ **RegionSkiz** (output iconic) ................................... region(-array)  \(\sim\) HRegionX / HUntypedObjectX
Result of the skiz operator.

▷ **Iterations1** (input control) ................................... integer  \(\sim\) VARIANT (integer, string )
Number of iterations for the sequential thinning with the element ‘l’ of the Golay alphabet.

Default Value: 100

Value Suggestions: Iterations1 \(\in\) \{'maximal’, 0, 1, 2, 3, 5, 7, 10, 15, 20, 30, 40, 50, 70, 100, 150, 200, 300, 400\}

(in)Minimal Value Step: 1

Recommended Value Step: 1
7.2. REGION

- **Iterations2** (input control) .......................................................... integer ~ VARIANT (integer, string)
  Number of iterations for the sequential thinning with the element ’e’ of the Golay alphabet.

  **Value Suggestion:** Iterations2 ∈ \{’maximal’, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}
  (lin) **Minimal Value Step:** 1
  **Recommended Value Step:** 1

**Result**

MorphSkiz returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

**Parallelization Information**

MorphSkiz is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

**Possible Successor Functions**

Pruning, ReduceDomain, SelectShape, AreaCenter, Connection, BackgroundSeg, Complement

**See Also**

Thinning, HitOrMissSeq, Difference

**Alternatives**

Skeleton, ThinningSeq, MorphSkeleton, Interjacent

**Module**

Morphology

```c
[out] HRegionX RegionOpening HRegionX.Opening
([in] HRegionX StructElement)
void HOperatorSetX.Opening ([in] IObjectX Region,
[in] IObjectX StructElement, [out] HUntypedObjectX RegionOpening)
```

**Open a region.**

An Opening operation is defined as an erosion followed by a Minkowski addition. By applying Opening to a region, larger structures remain mostly intact, while small structures like lines or points are eliminated. In contrast, a Closing operation results in small gaps being retained or filled up (see Closing).

Opening serves to eliminate small regions (smaller than StructElement) and to smooth the boundaries of a region. The position of StructElement is meaningless, since an opening operation is invariant with respect to the choice of the reference point.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

**Parameter**

- **Region** (input iconic) .............................................................. region(-array) ~ HRegionX/IObjectX
  Regions to be opened.

- **StructElement** (input iconic) ...................................................... region ~ HRegionX/IObjectX
  Structuring element (position-invariant).

- **RegionOpening** (output iconic) ............................................. region(-array) ~ HRegionX/HUntypedObjectX
  Opened regions.
Example

/* Large regions in an aerial picture (beech trees or meadows): */
read_image(Image,'wald1')
threshold(Image,Light,80,255)
gen_circle(StructElement1,100,100,2)
gen_circle(StructElement2,100,100,20)
/* close the small gap */
closing(Light,StructElement1,H)
/* selecting the large regions */
opening(H,StructElement2,Large).

/* Selecting of edges with certain orientation: */
read_image(Image,'fabrik')
sobel_amp(Image,Sobel,'sum_abs',3)
threshold(Sobel,Edges,10,255)
gen_rectangle2(StructElement,100,100,3.07819,20,1)
opening(Edges,StructElement,Direction).

Result
Opening returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information
Opening is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
Threshold,RegionGrow,Connection,Union,Union1,Watersheds,ClassNdimNorm,GenCircle,
GenEllipse,GenRectangle1,GenRectangle2,DrawRegion,GenRegionPoints,
GenStructElements,GenRegionPolygonFilled

Possible Successor Functions
ReduceDomain,SelectShape,AreaCenter,Connection

See Also
GenCircle,GenRectangle2,GenRegionPolygon

Alternatives
MinkowskiAdd1,Erosion1,OpeningCircle

Module

```
[out] HRegionX RegionOpening HRegionX.OpeningCircle
([in] VARIANT Radius )

void HOperatorSetX.OpeningCircle ([in] IHObjectX Region,
[out] HUntypedObjectX RegionOpening, [in] VARIANT Radius )
```

Open a region with a circular structuring element.

OpeningCircle is defined as an erosion followed by a Minkowski addition with a circular structuring element (see example). Opening serves to eliminate small regions (smaller than the circular structuring element) and to smooth the boundaries of a region.
7.2. REGION

Parameter

- **Region** (input iconic) .............................. region(-array) \(\rightarrow\) HRegionX / IHObjectX
  Regions to be opened.

- **RegionOpening** (output iconic) ................. region(-array) \(\rightarrow\) HRegionX / HUntypedObjectX
  Opened regions.

- **Radius** (input control) .............................. real \(\rightarrow\) VARIANT (real, integer)
  Radius of the circular structuring element.

  Default Value: 3.5
  Value Suggestions: Radius \(\in\) \{1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5\}
  Typical Range of Values: 0.5 \(\leq\) Radius \(\leq\) 0.5(lin)
  Minimal Value Step: 1.0
  Recommended Value Step: 1.0

Example

/* Large regions in an aerial picture (beech trees or meadows): */
read_image(Image, 'wald1')
threshold(Image, Light, 80, 255)
/* close the small gap */
closing_circle(LightH, 2)
/* selecting the large regions */
opening_circle(H, Large, 20).

Result

OpeningCircle returns TRUE if all parameters are correct. The behavior in case of empty or no input region
may be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

OpeningCircle is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Regiongrowing, Connection, Union1, Watersheds, ClassNdimNorm

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

TransposeRegion

Alternatives

Opening, Dilation1, MinkowskiAdd1, GenCircle

Module

```
[out] HRegionX RegionOpening HRegionX.OpeningGolay

void HOperatorSetX.OpeningGolay ([in] IHObjectX Region,
```

Open a region with an element from the Golay alphabet.

OpeningGolay is defined as a Minkowski subtraction followed by a Minkowski addition. First the Minkowski
subtraction of the input region (Region) with the structuring element from the Golay alphabet defined by
GolayElement and Rotation is computed. Then the Minkowski addition of the result and the structuring element rotated by 180° is performed.

The following structuring elements are available:
'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The rotation number Rotation determines which rotation of the element should be used, and whether the foreground (even) or background version (odd) of the selected element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements.

OpeningGolay serves to eliminate regions smaller than the structuring element, and to smooth regions’ boundaries.

---

**Attention**

Not all values of Rotation are valid for any Golay element. For some of the values of Rotation, the resulting regions are identical to the input regions.

---

**Parameter**

- **Region** (input iconic) ............................ region(-array) ◄ \(\text{HRegionX/\text{IObjectX}}\) Regions to be opened.
- **RegionOpening** (output iconic) ...................... region(-array) ◄ \(\text{HRegionX/\text{HUntypedObjectX}}\) Opened regions.
- **GolayElement** (input control) ...................... string ◄ String Structuring element from the Golay alphabet.
  - Default Value : 'h'
  - Value List : GolayElement ∈ \{'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'\}
- **Rotation** (input control) ............................ integer ◄ long Rotation of the Golay element Depending on the element, not all rotations are valid.
  - Default Value : 0
  - Value List : Rotation ∈ \{0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15\}

---

OpeningGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

---

**Parallelization Information**

OpeningGolay is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

---

**Possible Successor Functions**

ReduceDomain, SelectShape, AreaCenter, Connection

---

See Also

ErosionGolay, DilationGolay, ClosingGolay, HitOrMissGolay, ThinningGolay, ThickeningGolay, GolayElements

---

Alternatives

OpeningSeg, Opening

---

Module

---

Morphology

---

```plaintext
[out] HRegionX RegionOpening \ HRegionX.OpeningRectangle1
([in] VARIANT Width, [in] VARIANT Height )

void HOperatorSetX.OpeningRectangle1 ([in] IHObjectX Region, 
[out] HUntypedObjectX RegionOpening, [in] VARIANT Width, 
[in] VARIANT Height )
```

Open a region with a rectangular structuring element.
OpeningRectangle1 performs an ErosionRectangle1 followed by a DilationRectangle1 on the input region Region. The size of the rectangular structuring element is determined by the parameters Width and Height. As is the case for all Opening variants, larger structures are preserved, while small regions like lines or points are eliminated.

### Parameter

- **Region** (input iconic) 
  - region(-array) \(\rightsquigarrow\) HRegionX / IObjectX
  - Regions to be opened.

- **RegionOpening** (output iconic) 
  - region(-array) \(\rightsquigarrow\) HRegionX / HUntypedObjectX
  - Opened regions.

- **Width** (input control) 
  - extent.x \(\rightsquigarrow\) VARIANT (integer, real)
    - Width of the structuring rectangle.
    - Default Value: 10
    - Value Suggestions: Width \(\in\) \{1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200\}
    - Typical Range of Values: \(1 \leq \text{Width} \leq 1\) (lin)
    - Minimal Value Step: 1
    - Recommended Value Step: 1

- **Height** (input control) 
  - extent.y \(\rightsquigarrow\) VARIANT (integer, real)
    - Height of the structuring rectangle.
    - Default Value: 10
    - Value Suggestions: Height \(\in\) \{1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200\}
    - Typical Range of Values: \(1 \leq \text{Height} \leq 1\) (lin)
    - Minimal Value Step: 1
    - Recommended Value Step: 1

### Result

OpeningRectangle1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(’noObjectResult’,<RegionResult>)
- empty region: SetSystem(’emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

### Parallelization Information

OpeningRectangle1 is reentrant and automatically parallelized (on tuple level).

### Possible Predecessor Functions

- Threshold, Regiongrowing, Watersheds, ClassNdimNorm

### Possible Successor Functions

- ReduceDomain, SelectShape, AreaCenter, Connection

### See Also

- OpeningSeg, OpeningGolay

### Alternatives

- Opening, GenRectangle1, DilationRectangle1, ErosionRectangle1

### Module

**Morphology**

```plaintext
[out] HRegionX RegionOpening HRegionX.OpeningSeg
([in] HRegionX StructElement )

void HOperatorSetX.OpeningSeg ([in] IObjectX Region,
[in] IObjectX StructElement, [out] HUntypedObjectX RegionOpening )
```

Separate overlapping regions.

The OpeningSeg operation is defined as a sequence of the following operators: Erosion1, Connection and Dilation1 (see example). Only one iteration is done in Erosion1 and Dilation1.
OpeningSeg serves to separate overlapping regions whose area of overlap is smaller than StructElement. It should be noted that the resulting regions can overlap without actually merging (see ExpandRegion).

OpeningSeg uses the center of gravity as the reference point of the structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

### Parameter

- **Region** (input iconic) .............................. region(-array)  \( \sim HRegionX / IHObjectX \)
  Regions to be opened.

- **StructElement** (input iconic) ............................. region  \( \sim HRegionX / IHObjectX \)
  Structuring element (position-invariant).

- **RegionOpening** (output iconic) .............................. region  \( \sim HRegionX / HUntypedObjectX \)
  Opened regions.

### Example

```c
/* Simulation of opening_seg */
opening_seg(Region,StructElement,RegionOpening):
  erosion1(Region,StructElement,H1,1) >
  connection(H1,H2)
  dilation1(H2,StructElement,RegionOpening,1)
  clear_obj([H1,H2]).
```

### Result

OpeningSeg returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

### Parallelization Information

OpeningSeg is reentrant and automatically parallelized (on tuple level).

### Possible Predecessor Functions

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

### Possible Successor Functions

ExpandRegion, ReduceDomain, SelectShape, AreaCenter, Connection

### Alternatives

Erosion1, Connection, Dilation1

### Module

Morphology

```c
[out] HRegionX RegionPrune HRegionX.Pruning ([in] long Length )

void HOperatorSetX.Pruning ([in] IHObjectX Region, [out] HUntypedObjectX RegionPrune, [in] long Length )
```

Prune the branches of a region.

Pruning removes branches from a skeleton (Region) having a length less than Length. All other branches are preserved.
7.2. REGION 429

Parameter

- **Region** (input iconic) .................................. region(-array) ~ HRegionX / IObjectX
  Regions to be processed.

- **RegionPrune** (output iconic) .........................region(-array) ~ HRegionX / HUntypedObjectX
  Result of the pruning operation.

- **Length** (input control) .................................... integer ~ long
  Length of the branches to be removed.
  Default Value : 2
  Value Suggestions : Length ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typical Range of Values : 1 ≤ Length ≤ 1(lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

Result

Pruning returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

Pruning is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

MorphSkiz, Skeleton, ThinningSeq

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

MorphSkeleton, JunctionsSkeleton

Module

Morphology

```plaintext
[out] HRegionX RegionThick HRegionX.Thickening
([in] HRegionX StructElement1, [in] HRegionX StructElement2, [in] long Row,
[in] long Column, [in] long Iterations )

void HOperatorSetX.Thickening ([in] IObjectX Region,
[in] IObjectX StructElement1, [in] IObjectX StructElement2,
[out] HUntypedObjectX RegionThick, [in] long Row, [in] long Column,
[in] long Iterations )
```

Add the result of a hit-or-miss operation to a region.

Thickening performs a thickening of the input regions using morphological operations. The operator first applies a hit-or-miss-transformation to Region (cf. HitOrMiss), and then adds the detected points to the input region. The parameter Iterations determines the number of iterations performed.

For the choice of the structuring elements StructElement1 and StructElement2, as well as for Row and Column, the same restrictions described under HitOrMiss apply.

The structuring elements (StructElement1 and StructElement2) can be generated by calling GolayElements, for example.

Attention

If the reference point is contained in StructElement1 the input region remains unchanged.
**Parameter**

- **Region** (input iconic) : region(-array) \(\sim HRegionX / IHObjectX\)
  - Regions to be processed.

- **StructElement1** (input iconic) : region \(\sim HRegionX / IHObjectX\)
  - Structuring element for the foreground.

- **StructElement2** (input iconic) : region \(\sim HRegionX / IHObjectX\)
  - Structuring element for the background.

- **RegionThick** (output iconic) : region(-array) \(\sim HRegionX / HUntypedObjectX\)
  - Result of the thickening operator.

- **Row** (input control) : point.y \(\sim long\)
  - Row coordinate of the reference point.
  - **Default Value** : 16
  - **Value Suggestions** : Row \(\in\{0, 2, 4, 8, 16, 32, 128\}\)
  - **Typical Range of Values** : \(0 \leq \text{Row} \leq 0\) (lin)
  - **Minimal Value Step** : 1
  - **Recommended Value Step** : 1

- **Column** (input control) : point.x \(\sim long\)
  - Column coordinate of the reference point.
  - **Default Value** : 16
  - **Value Suggestions** : Column \(\in\{0, 2, 4, 8, 16, 32, 128\}\)
  - **Typical Range of Values** : \(0 \leq \text{Column} \leq 0\) (lin)
  - **Minimal Value Step** : 1
  - **Recommended Value Step** : 1

- **Iterations** (input control) : integer \(\sim long\)
  - Number of iterations.
  - **Default Value** : 1
  - **Value Suggestions** : Iterations \(\in\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 30, 40, 50, 70, 100, 200, 400\}\)
  - **(lin)Minimal Value Step** : 1
  - **Recommended Value Step** : 1

**Result**

Thickening returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:
- no region: SetSystem(’noObjectResult’,<RegionResult>)
- empty region: SetSystem(’emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

**Parallelization Information**

Thickening is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

GolayElements, Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

**Possible Successor Functions**

ReduceDomain, SelectShape, AreaCenter, Connection

**See Also**

HitOrMiss

**Alternatives**

ThickeningGolay, ThickeningSeq

**Module**

Morphology
Add the result of a hit-or-miss operation to a region (using a Golay structuring element).

ThickeningGolay performs a thickening of the input regions using morphological operations and structuring elements from the Golay alphabet. The operator first applies a hit-or-miss-transformation to Region (cf. HitOrMissGolay), and then adds the detected points to the input region. The following structuring elements are available:

'1', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The rotation number Rotation determines which rotation of the element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements.

Attention

Not all values of Rotation are valid for any Golay element.

Parameter

▷ Region (input iconic) .............................................. region(-array) \( \sim \) HRegionX / IObjectX
Regions to be processed.

▷ RegionThick (output iconic) ................................. region(-array) \( \sim \) HRegionX / HUntypedObjectX
Result of the thickening operator.

▷ GolayElement (input control) ................................. string \( \sim \) String
Structuring element from the Golay alphabet.

Value List: GolayElement \( \in \) \{'1', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'\}

▷ Rotation (input control) ................................. integer \( \sim \) long
Rotation of the Golay element Depending on the element, not all rotations are valid.

Value List: Rotation \( \in \) \{0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15\}

Result

ThickeningGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

ThickeningGolay is reentrant and automatically parallelized (on tuple level).

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

ErosionGolay, HitOrMissGolay

Alternatives

Thickening, ThickeningSeq

Module

Morphology
Add the result of a hit-or-miss operation to a region (sequential).

ThickeningSeq calculates the sequential thickening of the input regions with a structuring element from the Golay alphabet (GolayElement). To do so, ThickeningSeq calls the operator ThickeningGolay with all possible rotations of the structuring element Iterations times. The following structuring elements are available:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The Golay elements, together with all possible rotations, are described with the operator GolayElements. For all elements of the Golay alphabet, except for 'c', the foreground and background masks are exchanged in order to have an effect for them on the outer boundary of the region. The element 'c' can be used to generate the convex hull of the input region if enough iterations are performed.

Parameter
- Region (input iconic) .................................. region(-array) ~ HRegionX / IHObjectX
  Regions to be processed.
- RegionThick (output iconic) .......................... region(-array) ~ HRegionX / HUntypedObjectX
  Result of the thickening operator.
- GolayElement (input control) ............................ string ~ String
  Structuring element from the Golay alphabet.
  Default Value : 'h'
  Value List: GolayElement ∈ { 'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k' }
- Iterations (input control) ................................ integer ~ long
  Number of iterations.
  Default Value : 1
  Value Suggestions: Iterations ∈ { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50, 70, 100, 200 } (lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

Result
ThickeningSeq returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information
ThickeningSeq is reentrant and automatically parallelized (on tuple level).

Possible Successor Functions
ReduceDomain, SelectShape, AreaCenter, Connection

See Also
ErosionGolay, ThinningSeq

Alternatives
ThickeningGolay, Thickening

Module
Morphology
Remove the result of a hit-or-miss operation from a region.

Thinning performs a thinning of the input regions using morphological operations. The operator first applies a hit-or-miss-transformation to Region (cf. HitOrMiss), and then removes the detected points from the input region. The parameter Iterations determines the number of iterations performed.

For the choice of the structuring elements StructElement1 and StructElement2, as well as for Row and Column, the same restrictions described under HitOrMiss apply.

Structuring elements (StructElement1, StructElement2) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

--- Parameter 

- **Region** (input iconic) .......................... region(-array)  \( \sim \) HRegionX / IHObjectX
  
  Regions to be processed.

- **StructElement1** (input iconic) .......................... region  \( \sim \) HRegionX / IHObjectX
  
  Structuring element for the foreground.

- **StructElement2** (input iconic) .......................... region  \( \sim \) HRegionX / IHObjectX
  
  Structuring element for the background.

- **RegionThin** (output iconic) .......................... region(-array)  \( \sim \) HRegionX / HUntypedObjectX
  
  Result of the thinning operator.

- **Row** (input control) .......................... point.y  \( \sim \) long
  
  Row coordinate of the reference point.

  Default Value : 0
  
  Minimal Value Step : 1
  
  Recommended Value Step : 1

- **Column** (input control) .......................... point.x  \( \sim \) long
  
  Column coordinate of the reference point.

  Default Value : 0
  
  Minimal Value Step : 1
  
  Recommended Value Step : 1

- **Iterations** (input control) .......................... integer  \( \sim \) long
  
  Number of iterations.

  Default Value : 1
  
  Value Suggestions: Iterations \( \in \) \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}

  (lin) Minimal Value Step : 1
  
  Recommended Value Step : 1

--- Result 

Thinning returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

--- Parallelization Information 

Thinning is reentrant and automatically parallelized (on tuple level).
Possible Predecessor Functions

Threshold, RegionGrowing, Connection, Union, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successor Functions

ReduceDomain, SelectShape, AreaCenter, Connection

See Also

HitOrMiss

Alternatives

ThinningGolay, ThinningSeq

Module

Morphology

Remove the result of a hit-or-miss operation from a region (using a Golay structuring element).

ThinningGolay performs a thinning of the input regions using morphological operations and structuring elements from the Golay alphabet. The operator first applies a hit-or-miss-transformation to Region (cf. HitOrMissGolay), and then removes the detected points from the input region. The following structuring elements are available:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The rotation number Rotation determines which rotation of the element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements.

Attention

Not all values of Rotation are valid for any Golay element.

Parameter

Region (input iconic) ............................... region(-array)  \( \Rightarrow \) HRegionX / IHObjectX

Regions to be processed.

RegionThin (output iconic) ............................... region(-array)  \( \Rightarrow \) HRegionX / HUntypedObjectX

Result of the thinning operator.

GolayElement (input control) ............................... string  \( \Rightarrow \) String

Structuring element from the Golay alphabet.

Default Value : 'h'

Value List : GolayElement \( \in \) \{'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'\}

Rotation (input control) ............................... integer  \( \Rightarrow \) long

Rotation of the Golay element Depending on the element, not all rotations are valid.

Default Value : 0

Value List : Rotation \( \in \) \{0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15\}

ThinningGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

ThinningGolay is reentrant and automatically parallelized (on tuple level).
7.2. REGION

Possible Successor Functions
ReduceDomain, SelectShape, AreaCenter, Connection

See Also
ErosionGolay, HitOrMissGolay

Alternatives
ThinningSeq, Thinning

Module
Morphology

Remove the result of a hit-or-miss operation from a region (sequential).

ThinningSeq calculates the sequential thinning of the input regions with a structuring element from the Golay alphabet (GolayElement). To do so, ThinningSeq calls the operator ThinningGolay with all possible rotations of the structuring element Iterations times. If Iterations is chosen large enough, the operator calculates the skeleton of a region if the structuring elements 'l' or 'm' are used. For the element 'c' the background and foreground are exchanged in order to have an effect on the interior boundary of a region. If a very large value or 'maximal' is passed for Iterations the iteration stops if no more changes occur. The following structuring elements are available:

'l'  Skeleton, similar to Skeleton. This structuring element is also used in MorphSkiz.
'm'  A skeleton with many "hairs" and multiple (parallel) branches.
'd'  A skeleton without multiple branches, but with many gaps, similar to MorphSkeleton.
'e'  Uniform erosion of the region.
'e'  One pixel wide lines are shortened. This structuring element is also used in MorphSkiz.
'l'  Y-junctions are eliminated. (Only Iterations = 1 is useful.)
'l'  Y-junctions are eliminated. (Only Iterations = 1 is useful.)
'l'  One pixel long branches and corners are removed. (Only Iterations = 1 is useful.)
'h'  A kind of inner boundary, which, however, is thicker than the result of Boundary, is generated. (Only Iterations = 1 is useful.)
'k'  Junction points are eliminated, but also new ones are generated.

The Golay elements, together with all possible rotations, are described with the operator GolayElements.

Parameter

Region (input iconic) .......................... region(-array) \( \rightsquigarrow \) HRegionX / IObjectX
Regions to be processed.

RegionThin (output iconic) ................. region(-array) \( \rightsquigarrow \) HRegionX / HUntypedObjectX
Result of the thinning operator.

GolayElement (input control) .................. string \( \rightsquigarrow \) String
Structuring element from the Golay alphabet.

Default Value : 'l'
Value List : GolayElement \( \in \) \{'l', 'm', 'd', 'e', 'c', 'i', 'l', 'f2', 'h', 'k'\}

Iterations (input control) ...................... integer \( \rightsquigarrow \) VARIANT ( integer, string )
Number of iterations. For 'l', 'f2', 'h' and 'i' the only useful value is 1.

Default Value : 20
Value Suggestions : Iterations \( \in \) \{'maximal', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50, 70, 100, 150, 200\}
(linear) Minimal Value Step : 1
Recommended Value Step : 1
ThinningSeq returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

ThinningSeq is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
- Threshold, Regiongrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successor Functions
- Pruning, ReduceDomain, SelectShape, AreaCenter, Connection, Complement

See Also
- HitOrMissSeq, ErosionGolay, Difference, ThinningGolay, Thinning, ThickeningSeq

Alternatives
- Skeleton, MorphSkiz, ExpandRegion

Module
- Morphology

```c
[out] HRegionX RegionTopHat HRegionX.TopHat
([in] HRegionX StructElement )

void HOperatorSetX.TopHat ([in] IHObjectX Region,
[in] IHObjectX StructElement, [out] HUntypedObjectX RegionTopHat )
```

Compute the top hat of regions.

TopHat computes the Opening of Region with StructElement. The difference between the original region and the result of the opening is called the top hat. In contrast to Opening, which splits regions under certain circumstances, TopHat computes the regions removed by such a splitting.

The position of StructElement is meaningless, since an opening operation is invariant with respect to the choice of the reference point.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

### Parameter

- **Region** (input iconic) . . . . . . . . . . . . . region(-array) ∼ HRegionX / IHObjectX
  - Regions to be processed.
- **StructElement** (input iconic) . . . . . . . . . . . . . region ∼ HRegionX / IHObjectX
  - Structuring element (position independent).
- **RegionTopHat** (output iconic) . . . . . . . . region(-array) ∼ HRegionX / HUntypedObjectX
  - Result of the top hat operator.

### Result

TopHat returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

**Parallelization Information**

TopHat is *reentrant* and automatically *parallelized* (on tuple level).

**Possible Predecessor Functions**

Threshold, Regiongrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

**Possible Successor Functions**

ReduceDomain, SelectShape, AreaCenter, Connection

**See Also**

BottomHat, MorphHat, GrayTophat, Opening

**Alternatives**

Opening, Difference

**Module**

Morphology
Chapter 8

Object

8.1 Information

[out] long Number HObjectX.CountObj ()
void HOperatorSetX.CountObj ([in] IHObjectX Objects, [out] long Number )

Number of objects in a tuple.
The operator CountObj determines for the object parameter Objects the number of objects it contains. In this connection it should be noted that object is not the same as connection component (see Connection). For example, the number of objects of a region not consisting of three connected parts is 1.

Attention
In Prolog and Lisp the length of the list is not necessarily identical with the number of objects. This is the case when object keys are contained which were created in the compact mode (keys from compact and normal mode can be used as a mixture). See in this connection SetSystem('compactObject',<true/false>:).

Parameter
Objects (input iconic) object ~ IHObjectX
Objects to be examined.
Number (output control) integer ~ long
Number of objects in the tuple Objects.

Result
If the surrogates are correct, i.e. all objects are present in the HALCON operator data base, the operator CountObj returns the value TRUE. The behavior in case of empty input (no input objects available) is set via the operator SetSystem('noObjectResult',<Result>:).

Parallelization Information
CountObj is reentrant and processed without parallelization.

See Also
CopyObj, ObjToInteger, Connection, SetSystem

Basic operators

[out] VARIANT Information HObjectX.GetChannelInfo ([in] String Request, [in] VARIANT Channel )

Informations about the components of an image object.
The operator GetChannelInfo gives information about the components of an image object. The following requests (Request) are currently possible:
'creator' Output of the names of the procedures which initially created the image components (not the object).

'type' Output of the type of image component ('byte', 'int1', 'int2', 'int4', 'real', 'direction', 'cyclic', 'complex', 'dvf', 'lut'). The component 0 is of type 'region' or 'xld'.

In the tuple Channel the numbers of the components about which information is required are stated. After carrying out GetChannelInfo, Information contains a tuple of strings (one string per entry in Channel) with the required information.

```
<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Object (input iconic) ..................</td>
</tr>
<tr>
<td>▶ Request (input control) ...............</td>
</tr>
<tr>
<td>Default Value: 'creator'</td>
</tr>
<tr>
<td>Value List: Request ∈ {'creator', 'type'}</td>
</tr>
<tr>
<td>◄ Channel (input control) ...............</td>
</tr>
<tr>
<td>Default Value: 0</td>
</tr>
<tr>
<td>Value Suggestions: Channel ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8}</td>
</tr>
<tr>
<td>◄ Information (output control) ..........</td>
</tr>
</tbody>
</table>
```

If the parameters are correct the operator GetChannelInfo returns the value TRUE. Otherwise an exception is raised.

```
Parallelization Information
GetChannelInfo is reentrant and processed without parallelization.
```

Possible Predecessor Functions

```
CountRelation
```

See Also

```
CountRelation
```

Module

```
Basic operators
```

```
[out] VARIANT Class HObjectX.GetObjClass()  
void HOperatorSetX.GetObjClass([in] IObjectX Object,  
[out] VARIANT Class)
```

Name of the class of an image object.

GetObjClass returns the name of the corresponding class to each object. The following classes are possible:

'object' Object with region (definition domain) and at least one channel.

'region' Object with a region without gray values.

'xld_cont' XLD object as contour

'xld_poly' XLD object as polygon

'xld_parallel' XLD object with parallel polygons

```
<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Object (input iconic) ..................</td>
</tr>
<tr>
<td>▶ Class (output control) ...............</td>
</tr>
</tbody>
</table>
```

If the parameter values are correct the operator \texttt{GetObjClass} returns the value TRUE. Otherwise an exception is raised.

\textbf{Parallelization Information}\n
\texttt{GetObjClass} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

\textbf{Possible Successor Functions}\n
\texttt{DispImage,DispRegion,DispXld}\n
\textbf{See Also}\n
\texttt{GetChannelInfo,CountRelation}\n
\textbf{Module}\n
\textbf{Basic operators}\n
\begin{verbatim}
void \texttt{HObjectX.TestEqualObj} ([in] \texttt{IObjectX Objects2} )
void \texttt{HOperatorSetX.TestEqualObj} ([in] \texttt{IObjectX Objects1,}
\texttt{[in] \texttt{IObjectX Objects2} )
\end{verbatim}

\textit{Compare image objects regarding equality.}\n
The operator \texttt{TestEqualObj} compares the regions and gray value components of all objects of the two input parameters. The \textit{n}-th object in \texttt{Objects1} is compared to the \textit{n}-th object in \texttt{Objects2} (for all \textit{n}). If all corresponding regions are equal and the number of regions is also identical the operator \texttt{TestEqualObj} returns the value TRUE, otherwise FALSE.

\textbf{Attention}\n
Image matrices are not compared regarding their contents. Thus, two images are “equal” if they are at the same place in the store. If the input parameters are empty and the behavior was set via the operator \texttt{SetSystem(\{::'noObjectResult',<Result>:\}),} the operator \texttt{TestEqualObj} returns TRUE, since all input (= empty set) is equal.

\textbf{Parameter}\n
\begin{itemize}
  \item \texttt{Objects1} (input iconic) \hspace{1cm} \texttt{object} \sim \texttt{IObjectX} Test objects.
  \item \texttt{Objects2} (input iconic) \hspace{1cm} \texttt{object} \sim \texttt{IObjectX} Comparative objects.
\end{itemize}

\textbf{Result}\n
The operator \texttt{TestEqualObj} returns the value TRUE if both object tuples are identical. If the tuples differ in at least one place \texttt{TestEqualObj} returns FALSE. The behavior in case of empty input (no input objects available) is set via the operator \texttt{SetSystem(\{::'noObjectResult',<Result>:\}). If the number of objects differs an exception is raised.

\textbf{Parallelization Information}\n
\texttt{TestEqualObj} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{See Also}\n
\texttt{TestEqualRegion}\n
\textbf{Module}\n
\textbf{Basic operators}\n
\begin{verbatim}
void \texttt{HRegionX.TestEqualRegion} ([in] \texttt{RegionX Regions2} )
void \texttt{HOperatorSetX.TestEqualRegion} ([in] \texttt{IObjectX Regions1,}
\texttt{[in] \texttt{IObjectX Regions2} )
\end{verbatim}

\textit{Test whether the regions of two objects are identical.}\n
The operator \texttt{TestEqualRegion} compares the regions of the two input parameters. The \textit{n}-th element in \texttt{Regions1} is compared to the \textit{n}-th object in \texttt{Regions2} (for all \textit{n}). If all regions are equal and the number of regions is identical the operator \texttt{TestEqualRegion} returns the value TRUE, otherwise FALSE.
The operator \texttt{TestEqualRegion} returns the value \texttt{TRUE} if both object tuples are identical. If the tuples differ in at least one place \texttt{TestEqualRegion} returns \texttt{FALSE}. The behavior in case of empty input (no input objects available) is set via the operator \texttt{SetSystem(\texttt{::'noObjectResult',<Result>:})}. If the number of objects differs an exception is raised.

\textbf{Parallelization Information}

\texttt{TestEqualRegion} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{See Also}

\texttt{TestEqualObj}

\textbf{Alternatives}

\texttt{Intersection, Complement, AreaCenter}

\textbf{Module}

\textbf{Basic operators}

\begin{verbatim}
void \texttt{HObjectX.TestObjDef} ( )
void \texttt{HOperatorSetX.TestObjDef} ([in] \texttt{IHObjectX Object })
\end{verbatim}

Test whether an object is already deleted.

The operator \texttt{TestObjDef} checks whether the object still exists in the HALCON operator data base (i.e. whether the surrogate is still valid). This check especially makes sense before deleting an object if it is not sure that the object has already been deleted by a prior deleting operator (\texttt{ClearObj}).

\textbf{Attention}

The operator \texttt{TestObjDef} can return \texttt{TRUE} even if the object was already deleted because the surrogates of deleted objects are re-used for new objects. In this context see the example.

\textbf{Parameter}

\begin{itemize}
  \item \textbf{Object} (input iconic) \texttt{Object to be checked.}
\end{itemize}

\textbf{Result}

The operator \texttt{TestObjDef} returns the value \texttt{TRUE} if an object with this surrogate is present in the HALCON operator data base, otherwise the value \texttt{FALSE} is returned. The behavior in case of empty input (no input objects available) is set via the operator \texttt{SetSystem(\texttt{::'noObjectResult',<Result>:})}.

\textbf{Parallelization Information}

\texttt{TestObjDef} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{Possible Predecessor Functions}

\texttt{ClearObj, GenCircle, GenRectangle1}

\textbf{See Also}

\texttt{SetCheck, ClearObj, ResetObjDb}

\textbf{Module}

\textbf{Basic operators}
8.2 Manipulation

```c
void HOperatorSetX.ClearObj ([in] IHObjectX Objects )
```

Delete an iconic object from the HALCON database.

ClearObj deletes iconic objects, which are no longer needed, from the HALCON database. It should be noted that ClearObj is the only way to delete objects from the database, and hence to reclaim their memory, in all host languages except Smalltalk and C++.

Images and regions are normally used by several iconic objects at the same time (uses less memory!). This has the consequence that a region or an image is only deleted if all objects using it have been deleted.

The operator ResetObjDb can be used to reset the system and clear all remaining iconic objects.

---

**Attention**

Regarding the use of local variables: Because only local variables are deleted on exit of a subroutine (or a rule in Prolog), while the HALCON database is not updated, it is necessary to clear local objects before exiting the subroutine. Special care has to be taken if backtracking is possible in Prolog (before reaching the ClearObj statements).

---

**Parameter**

- `Objects` (input iconic) ........................................ object(-array) ~> IHObjectX
  
  Objects to be deleted.

---

**Result**

ClearObj returns TRUE if all objects are contained in the HALCON database. If not all objects are valid (e.g., already cleared), an exception is raised, which also clears all valid objects. The operator SetCheck (:: 'clear') can be used to suppress the raising of this exception. If the input is empty the behavior can be set via SetSystem (:: 'noObjectResult', <Result>). If necessary, an exception is raised.

---

**Parallelization Information**

ClearObj is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

- TestObjDef

---

**See Also**

- TestObjDef, SetCheck

---

**Alternatives**

- ResetObjDb

---

**Module**

Basic operators

```c
[out] HUntypedObjectX ObjectsConcat HOperatorX.ConcatObj
([in] IHObjectX Objects )
```

Concatenate two iconic object tuples.

ConcatObj concatenates the two tuples of iconic objects Objects1 and Objects2 into a new object tuple ObjectsConcat. Hence, this tuple contains all the objects of the two input tuples:

ObjectsConcat = [Objects1, Objects2]

In ObjectsConcat the objects of Objects1 are stored first, followed by the objects of Objects2. The order of the objects is preserved. As usual, only the objects are copied, and not the corresponding images and regions, i.e., no new memory is allocated. ConcatObj is designed especially for HALCON/C. In languages like Smalltalk, Prolog, and C++ it is not needed.

ConcatObj should not be confused with Union1 or Union2, in which regions are merged, i.e., in which the number of objects is modified.
ConcatObj can be used to concatenate objects of different image object types (e.g., images and XLD contours) into a single object. This is only recommended if it is necessary to accumulate in a single object variable, for example, the results of an image processing sequence. It should be noted that the only operators that can handle such object tuples of mixed type are ConcatObj, CopyObj, SelectObj, and DispObj. For technical reasons, object tuples of mixed type must not be created in HDevelop.

Parameter

- **Objects1** (input iconic) ....................................................... object(-array) \( \sim \) IHObjectX
  Object tuple 1.
- **Objects2** (input iconic) ....................................................... object(-array) \( \sim \) IHObjectX
  Object tuple 2.
- **ObjectsConcat** (output iconic) .............................................. object \( \sim \) HUntypedObjectX
  Concatenated objects.

Result

ConcatObj returns TRUE if all objects are contained in the HALCON database. If the input is empty the behavior can be set via SetSystem('noObjectResult',<Result>:]. If necessary, an exception is raised.

Parallelization Information

ConcatObj is reentrant and processed without parallelization.

See Also

CountObj, CopyObj, SelectObj, DispObj

Module

Basic operators

```
[out] HUntypedObjectX ObjectsSelected HObjectX.CopyObj ([in] long Index, [in] long NumObj)

```

Copy an iconic object in the HALCON database.

CopyObj copies NumObj iconic objects beginning with index Index (starting with 1) from the iconic input object tuple Objects to the output object ObjectsSelected. If -1 is passed for NumObj all objects beginning with Index are copied. No new storage is allocated for the regions and images. Instead, new objects containing references to the existing objects are created. The number of objects in an object tuple can be queried with the operator CountObj.

Parameter

- **Objects** (input iconic) ....................................................... object(-array) \( \sim \) IHObjectX
  Objects to be copied.
- **ObjectsSelected** (output iconic) ............................................ object(-array) \( \sim \) HUntypedObjectX
  Copied objects.
- **Index** (input control) ........................................................ integer \( \sim \) long
  Starting index of the objects to be copied.
  Default Value : 1
  Value Suggestions : Index \( \in \{1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}
  Restriction : (Objects \( \leq \) number)
- **NumObj** (input control) ....................................................... integer \( \sim \) long
  Number of objects to be copied or -1.
  Default Value : 1
  Value Suggestions : NumObj \( \in \{-1, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}\)
  Restriction : ((Objects \( \leq \) number) \( \land \) (NumObj \( \neq \) 0))

Example

```c
/* Access all regions */
```

count_obj(Regions, Num)
for(1, Num, i)
    copy_obj(Regions, Single, i, 1)
get_region_polygon(Single, 5.0, Line, Column)
disp_polygon(WindowHandle, Line, Column)
clear_obj(Single)
loop().

CopyObj returns TRUE if all objects are contained in the HALCON database and all parameters are correct. If the input is empty the behavior can be set via SetSystem(:::‘noObjectResult’,<Result>::). If necessary, an exception is raised.

CopyObj is reentrant and processed without parallelization.

Basic operators

void HObjectX.GenEmptyObj ()
void HOperatorSetX.GenEmptyObj ([out] HUntypedObjectX EmptyObject )

Create an empty object tuple.
The operator GenEmptyObj creates an empty tuple. This means that the output parameter does not contain any objects. Thus, the operator CountObj returns 0. However, ClearObj can be called for the output. It should be noted that no objects must not be confused with an empty region. In case of an empty region, i.e. a region with 0 pixels CountObj returns the value 1.

Parameter

\textbf{EmptyObject} (output iconic) \hspace{1cm} object \sim HUntypedObjectX

No objects.

GenEmptyObj is reentrant and processed without parallelization.

Image / region / XLD management

void HObjectX.IntegerToObj ([in] VARIANT SurrogateTuple )
void HOperatorSetX.IntegerToObj ([out] HUntypedObjectX Objects,
[in] VARIANT SurrogateTuple )

Convert an “integer number” into an iconic object.

IntegerToObj is the inverse operator to ObjToInteger. All surrogates of objects passed in SurrogateTuple are stored as objects. In contrast to ObjToInteger, the objects are duplicated. IntegerToObj is intended especially for use in HALCON/C, because iconic objects and control parameters are treated differently in C.

The objects are duplicated in the database.
Objects (output iconic) ................................. object ～ HUnTypedObjectX
Published objects.

SurrogateTuple (input control) .............................. integer ～ VARIANT ( integer )
Tuple of object surrogates.
Default Value : 1

Result

IntegerToObj returns TRUE if all parameters are correct, i.e., if they are valid object keys. If the input is empty the behavior can be set via SetSystem(::’noObjectResult’,<Result>:). If necessary, an exception is raised.

Parallelization Information

IntegerToObj is reentrant and processed without parallelization.

See Also

Module

Basic operators

[out] VARIANT SurrogateTuple HObjectX.ObjToInteger ([in] long Index, [in] long Number )


Convert an iconic object into an “integer number.”

ObjToInteger stores Number, starting at index Index, of the database keys of the input object Objects as integer numbers in the output parameter SurrogateTuple. This facilitates a direct access to an arbitrary element of Objects. In conjunction with CountObj (returns the number of objects in Objects) the elements of Objects can be processed successively. The objects are not duplicated by ObjToInteger and thus must not be cleared by ClearObj.

Attention

The objects’ data is not duplicated.

Parameter

Objects (input iconic) ................................. object ～ IHObjectX
Objects for which the surrogates are to be returned.

Index (input control) ................................. integer ～ long
Starting index of the surrogates to be returned.
Default Value : 1

Number (input control) ................................. integer ～ long
Number of surrogates to be returned.
Default Value : 1
Restriction : (Objects ≤ number)

SurrogateTuple (output control) .............................. integer ～ VARIANT ( integer )
Tuple containing the surrogates.

Example

/* Access the i-th element: */
obj_to_integer(Objects, i, 1, Surrogat).

Result

ObjToInteger returns TRUE if all parameters are correct. If the input is empty the behavior can be set via SetSystem(::’noObjectResult’,<Result>:). If necessary, an exception is raised.

Parallelization Information

ObjToInteger is reentrant and processed without parallelization.
8.2. MANIPULATION

Possible Predecessor Functions

TestObjDef

See Also

IntegerToObj, CountObj

Alternatives

CopyObj, SelectObj, CopyImage, GenImageProto

Module

Basic operators

```
[out] HUntypedObjectX ObjectSelected
[in] long Index

void HOperatorSetX.SelectObj ([in] IHObjectX Objects,
[out] HUntypedObjectX ObjectSelected, [in] long Index)
```

Select an object from an object tuple.

SelectObj copies the iconic object with index Index (starting with 1) from the iconic input object tuple Objects to the output object ObjectSelected. No new storage is allocated for the regions and images. Instead, new objects containing references to the existing objects are created. The number of objects in an object tuple can be queried with the operator CountObj.

Parameter

- **Objects** (input iconic) .......................... object(-array) \(\sim\) IHObjectX Objects, of which one is to be selected.
- **ObjectSelected** (output iconic) .................. object \(\sim\) HUntypedObjectX Selected object.
- **Index** (input control) ............................ integer \(\sim\) long Index of the object to be selected.

Default Value: 1

Value Suggestions: Index \(\in\) \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 50, 100, 200, 500, 1000, 2000, 5000\}

Result

SelectObj returns TRUE if all objects are contained in the HALCON database and all parameters are correct. If the input is empty the behavior can be set via SetSystem(::’noObjectResult’,<Result>:). If necessary, an exception is raised.

Parallelization Information

SelectObj is reentrant and processed without parallelization.

Possible Predecessor Functions

CountObj

See Also

CountObj, ConcatObj, ObjToInteger

Alternatives

CopyObj

Module

Basic operators
Contour of an object as chain code.

The operator GetRegionChain returns the contour of a region. A contour is a series of pixels describing the outline of the region. The contour “lies on” the region. It starts at the smallest line number; in that line at the pixel with the largest column index. The rotation occurs clockwise. Holes of the region are ignored. The direction code (chain code) is defined as follows:

```
3 2 1
4 6 7
```

The operator GetRegionChain returns the code in the form of a tuple. In case of an empty region the parameters Row and Column are zero and Chain is the empty tuple.

---
**Attention**
---

Holes of the region are ignored. Only one region may be passed, and it must have exactly one connection component.

---
**Parameter**
---

- **Region** (input iconic) ........................................................... region ~ HRegionX / IHObjectX
- **Row** (output control) ............................................................ chain.begin.y ~ long
- **Column** (output control) ...................................................... chain.begin.x ~ long
- **Chain** (output control) ....................................................... chain.code ~ VARIANT (integer)

**Typical Range of Values**: 0 ≤ Chain ≤ 0

---
**Result**
---

The operator GetRegionChain normally returns the value TRUE. If more than one connection component is passed an exception handling is caused. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via the operator SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.
GetRegionChain is reentrant and processed without parallelization.

Possible Predecessor Functions
SobelAmp, Threshold, Skeleton, EdgesImage, GenRectangle1, GenCircle

Possible Successor Functions
ApproxChain, ApproxChainSimple

See Also
CopyObj, GetRegionChain, GetRegionPolygon

Module
Region processing

[out] VARIANT Rows HRegionX.GetRegionContour ([out] VARIANT Columns )
void HOperatorSetX.GetRegionContour ([in] IHObjectX Region,
[out] VARIANT Rows, [out] VARIANT Columns )

Access the contour of an object.

The operator GetRegionContour returns the contour of a region. A contour is a result of line (Rows) and column coordinates (Columns), describing the boundary of the region. The contour lies on the region. It starts at the smallest line number in that line at the pixel with the largest column index. The rotation direction is clockwise. The first pixel of the contour is identical with the last. Holes of the region are ignored. The operator GetRegionContour returns the coordinates in the form of tuples. An empty region is passed as empty tuple.

Attention
Holes of the region are ignored. Only one region may be passed, and this region must have exactly one connection component.

Parameter

- **Region** (input iconic) ......................................region \( \sim \) HRegionX / IHObjectX Output region.
- **Rows** (output control) ......................................contour.y \( \sim \) VARIANT (integer) Line numbers of the contour pixels.
- **Columns** (output control) ...................................contour.x \( \sim \) VARIANT (integer) Column numbers of the contour pixels.

Parameter Number: \( (Columns \geq Rows) \)

Result

The operator GetRegionContour normally returns the value TRUE. If more than one connection component is passed an exception handling is caused. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>).

Parallelization Information
GetRegionContour is reentrant and processed without parallelization.

Possible Predecessor Functions
SobelAmp, Threshold, Skeleton, EdgesImage, GenRectangle1, GenCircle

See Also
CopyObj, GetRegionChain, GetRegionPolygon

Module
Region processing

[out] VARIANT Rows HRegionX.GetRegionConvex ([out] VARIANT Columns )
void HOperatorSetX.GetRegionConvex ([in] IHObjectX Region,
[out] VARIANT Rows, [out] VARIANT Columns )

Access convex hull as contour.
The operator GetRegionConvex returns the convex hull of a region as polygon. The polygon is the minimum result of line (Rows) and column coordinates (Columns) describing the hull of the region. The polygon pixels lie on the region. The polygon starts at the smallest line number; in this line at the pixel with the largest column index. The rotation direction is clockwise. The first pixel of the polygon is identical with the last. The operator GetRegionConvex returns the coordinates in the form of tuples. An empty region is passed as empty tuple.

**Parameter**

- **Region** (input iconic) ...............................region  \( \sim \) HRegionX / IHObjectX
  - Output region.
- **Rows** (output control) .............................contour.y  \( \sim \) VARIANT (integer)
  - Line numbers of contour pixels.
- **Columns** (output control) ..........................contour.x  \( \sim \) VARIANT (integer)
  - Column numbers of the contour pixels.

**Parameter Number** :  \( (C\text{olumns} = R\text{ows}) \)

**Result**
The operator GetRegionConvex returns the value TRUE.

**Parallelization Information**
GetRegionConvex is reentrant and processed without parallelization.

**Possible Predecessor Functions**
Threshold, Skeleton, DynThreshold

**Possible Successor Functions**
DispPolygon

**See Also**
SelectObj, GetRegionContour

**Alternatives**
ShapeTrans

**Module**
Region processing

---

**Access the pixels of a region.**

The operator GetRegionPoints returns the region data in the form of coordinate lists. The coordinates are sorted in the following order:

\[ (r_1, c_1) \leq (r_2, c_2) := (r_1 < r_2) \lor (r_1 = r_2) \land (c_1 \leq c_2) \]

GetRegionPoints returns the coordinates in the form of tuples. An empty region is passed as empty tuple.

**Attention**
Only one region may be passed.

**Parameter**

- **Region** (input iconic) ...............................region  \( \sim \) HRegionX / IHObjectX
  - This region is accessed.
- **Rows** (output control) .............................coordinates.y  \( \sim \) VARIANT (integer)
  - Line numbers of the pixels in the region
- **Columns** (output control) ..........................coordinates.x  \( \sim \) VARIANT (integer)
  - Column numbers of the pixels in the region.

**Parameter Number** :  \( (C\text{olumns} = R\text{ows}) \)
The operator \texttt{GetRegionPoints} normally returns the value TRUE. If more than one connection component is passed an exception handling is caused. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem(’noObjectResult’,\langle\texttt{Result}\rangle)}.

\textbf{Parallelization Information} \texttt{GetRegionPoints} is reentrant and processed without parallelization.

\textbf{Possible Predecessor Functions} \texttt{SobelAmp, Threshold, Connection}

\textbf{See Also} \texttt{CopyObj, GenRegionPoints}

\textbf{Alternatives} \texttt{GetRegionRuns}

\textbf{Module} Region processing

\begin{verbatim}
[out] VARIANT Rows HRegionX.GetRegionPolygon ([in] VARIANT Tolerance, 
[out] VARIANT Columns )
void HOperatorSetX.GetRegionPolygon ([in] IObjectX Region, 
\end{verbatim}

\section*{Polygon approximation of a region.}
The operator \texttt{GetRegionPolygon} calculates a polygon to approximate the edge of a region. A polygon is a sequence of line (\texttt{Rows}) and column coordinates (\texttt{Columns}). It describes the contour of the region. Only the base points of the polygon are returned. The parameter \texttt{Tolerance} indicates how large the maximum distance between the polygon and the edge of the region may be. Holes of the region are ignored. The operator \texttt{GetRegionPolygon} returns the coordinates in the form of tuples.

\begin{itemize}
  \item \textbf{Attention} Holes of the region are ignored. Only one region may be passed, and this region must have exactly one connection component.
\end{itemize}

\begin{itemize}
  \item \textbf{Region} (input iconic) Region to be approximated.
  \item \textbf{Tolerance} (input control) \texttt{number \sim \textsc{VARIANT}(\text{integer, real})} Maximum distance between the polygon and the edge of the region.
    \begin{itemize}
      \item \textbf{Default Value} : 5.0
      \item \textbf{Value Suggestions} : \texttt{Tolerance \in \{0.0, 2.0, 5.0, 10.0\}}
      \item \texttt{(lin)Minimal Value Step} : 0.01
      \item \texttt{Recommended Value Step} : 1.0
    \end{itemize}
  \item \textbf{Rows} (output control) \texttt{polygon.y \sim \textsc{VARIANT}(\text{integer})} Line numbers of the base points of the contour.
  \item \textbf{Columns} (output control) \texttt{polygon.x \sim \textsc{VARIANT}(\text{integer})} Column numbers of the base points of the contour.
\end{itemize}

\textbf{Parameter Number} : (\texttt{Columns = Rows})

\textbf{Result} The operator \texttt{GetRegionPolygon} normally returns the value TRUE. If more than one connection component is passed an exception handling is caused. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem(’noObjectResult’,\langle\texttt{Result}\rangle)}.

\textbf{Parallelization Information} \texttt{GetRegionPolygon} is reentrant and processed without parallelization.

\textbf{Possible Predecessor Functions} \texttt{SobelAmp, Threshold, Skeleton, EdgesImage}
Access the runlength coding of a region.

The operator GetRegionRuns returns the region data in the form of chord tuples. The chord representation is caused by examining a region line by line with ascending line number (= from “top” to “bottom”). Every line is passed from left to right (ascending column number); storing all starting and ending points of region segments (= chords). Thus a region can be described by a sequence of chords, a chord being defined by line number, starting and ending points (column number). The operator GetRegionRuns returns the three components of the chords in the form of tuples. In case of an empty region three empty tuples are returned.

Only one region may be passed.

Parameter

\( \textbf{Region} \) (input iconic) .............................. region \( \sim \) HRegionX / IHObjectX
Output region.

\( \textbf{Row} \) (output control) .............................. chord.y \( \sim \) VARIANT (integer)
Line numbers of the chords.

\( \textbf{ColumnBegin} \) (output control) ......................... chord.x1 \( \sim \) VARIANT (integer)
Column numbers of the starting points of the chords.

Parameter Number : (\( \textbf{ColumnBegin} = \textbf{Row} \))

\( \textbf{ColumnEnd} \) (output control) ............................ chord.x2 \( \sim \) VARIANT (integer)
Column numbers of the ending points of the chords.

Parameter Number : (\( \textbf{ColumnEnd} = \textbf{Row} \))

Result

The operator GetRegionRuns normally returns the value TRUE. If more than one region is passed an exception handling is caused. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>).

Parallelization Information

GetRegionRuns is reentrant and processed without parallelization.

Possible Predecessor Functions

Threshold, Connection

See Also

CopyObj, GenRegionRuns

Alternatives

GetRegionPoints

Module

Region processing
9.2 Affine-Transformations

Transform a region using an affine transformation.

AffineTransRegion applies an arbitrary affine transformation, i.e., scaling, rotation, translation, and skewing, to a region. The affine map is described by the transformation matrix given in HomMat2D, which is built up by using HomMat2dIdentity, HomMat2dScale, HomMat2dRotate, and HomMat2dTranslate. The components of the homogeneous transformation matrix are interpreted as follows: the row coordinate of the image corresponds to the \( x \) coordinate of the matrix, while the column coordinate of the image corresponds to the \( y \) coordinate of the matrix. This is necessary to obtain a right-handed coordinate system, which is assumed for the homogeneous transformation matrices, also for the image. In particular, by doing so rotations are performed in the correct direction. Note that the \((x,y)\) order of the matrices quite naturally corresponds to the usual \((\text{row},\text{column})\) order for coordinates in the image.

The parameter Interpolate determines whether the transformation is to be done by using interpolation internally. This can lead to smoother region boundaries, especially if regions are enlarged. However, the runtime increases drastically.

**Attention**

AffineTransRegion in general is not reversible (clipping and discretization during rotation and scaling).

The components of the homogeneous transformation matrix are interpreted as follows: the row coordinate of the image corresponds to the \( x \) coordinate of the matrix, while the column coordinate of the image corresponds to the \( y \) coordinate of the matrix. This is necessary to obtain a right-handed coordinate system, which is assumed for the homogeneous transformation matrices, also for the image. In particular, by doing so rotations are performed in the correct direction. Note that the \((x,y)\) order of the matrices quite naturally corresponds to the usual \((\text{row},\text{column})\) order for coordinates in the image.

**Parameter**

- **Region** (input iconic) .............................................. region(-array) \( \rightsquigarrow \) HRegionX / IHObjectX
  Region(s) to be rotated and scaled.

- **RegionAffineTrans** (output iconic) .............................. region(-array) \( \rightsquigarrow \) HRegionX / HUntypedObjectX
  Transformed output region(s).
  **Parameter Number :** \( \text{Region.AffineTrans} = \text{Region} \)

- **HomMat2D** (input control) ........................................ affine2d \( \rightsquigarrow \) VARIANT / HHomMat2dX (real)
  Input transformation matrix.
  **Parameter Number :** 6

- **Interpolate** (input control) ...................................... string \( \rightsquigarrow \) String
  Should the transformation be done using interpolation?
  **Default Value :** ‘false’
  **Value List :** Interpolate \( \in \{ \text{‘true’}, \text{‘false’} \} \)

**Result**

AffineTransRegion returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem(‘noObjectResult’,<Result>), the behavior in case of an empty input region via SetSystem(‘emptyRegionResult’,<Result>), and the behavior in case of an empty result region via SetSystem(‘storeEmptyRegion’,<true/false>). If necessary, an exception handling is raised.

**Parallelization Information**

AffineTransRegion is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

HomMat2dIdentity, HomMat2dScale, HomMat2dTranslate, HomMat2dInvert
Reflect a region about the x- or y-axis.

MirrorRegion reflects a region about the x- or y-axis (parameter RowColumn). The parameter WidthHeight specifies the corresponding image width or height respectively, i.e., it is two times the coordinate of the axis of symmetry.

Parameter

- **Region** (input iconic) ........................................ region(-array) \(\sim\) HRegionX / IHObjectX
  Region(s) to be reflected.
- **RegionMirror** (output iconic) .............................. region(-array) \(\sim\) HRegionX / HUntypedObjectX
  Reflected region(s).

**Parameter Number**: \(\text{RegionMirror} = \text{Region}\)

- **RowColumn** (input control) ..................................... string \(\sim\) String
  Coordinate of the axis of symmetry.
  **Default Value**: 'row'
  **Value List**: RowColumn \(\in\) \{'column', 'row'\}

- **WidthHeight** (input control) ................................. integer \(\sim\) long
  Height or width of the corresponding image.
  **Default Value**: 512
  **Value Suggestions**: WidthHeight \(\in\) \{128, 256, 512, 525, 768, 1024\}
  **Typical Range of Values**: \(1 \leq \text{WidthHeight} \leq 1\text{lin}\)
  **Minimal Value Step**: 1
  **Recommended Value Step**: 1
  **Restriction**: \(\text{WidthHeight} > 0\)

Parallelization Information

MirrorRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Connection, RegionGrowing, Pouring

Possible Successor Functions

SelectShape, DispRegion

See Also

ZoomRegion

Alternatives

AffineTransRegion

Module

Region processing
Translate a region.

MoveRegion translates the input regions by the vector given by (Row, Column). If necessary, the resulting regions are clipped with the current image format.

**Parameter**

- **Region** (input iconic) region(-array)  \( \sim \) HRegionX / IHObjectX Region(s) to be moved.
- **RegionMoved** (output iconic) region(-array)  \( \sim \) HRegionX / HUntypedObjectX Translated region(s).

**Parameter Number** 
\( \text{RegionMoved} = \text{Region} \)

- **Row** (input control) point.y  \( \sim \) long Row coordinate of the vector by which the region is to be moved.
  - Default Value: 30
  - Value Suggestions: Row \( \in \{ -128, -64, -32, -16, -8, -4, -2, -1, 0, 1, 2, 4, 5, 8, 10, 16, 32, 64, 128 \} \)
  - Typical Range of Values: \(-512 \leq \text{Row} \leq -512\) (lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 10

- **Column** (input control) point.x  \( \sim \) long Row coordinate of the vector by which the region is to be moved.
  - Default Value: 30
  - Value Suggestions: Column \( \in \{ -128, -64, -32, -16, -8, -4, -2, -1, 0, 1, 2, 4, 5, 8, 10, 16, 32, 64, 128 \} \)
  - Typical Range of Values: \(-512 \leq \text{Column} \leq -512\) (lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 10

**Result**

MoveRegion always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem(‘noObjectResult’,<Result>), the behavior in case of an empty input region via SetSystem(‘emptyRegionResult’,<Result>), and the behavior in case of an empty result region via SetSystem(‘storeEmptyRegion’,<true/false>). If necessary, an exception handling is raised.

**Parallelization Information**

MoveRegion is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

Threshold, Connection, RegionGrowing, Pouring

**Possible Successor Functions**

SelectShape, DispRegion

**See Also**

AffineTransImage, MirrorRegion, ZoomRegion

**Module**

Region processing

Reflect a region about a point.

TransposeRegion reflects a region about a point. The fixed point is given by Column and Row. The image \( P' \) of a point \( P \) is determined by the following requirement:
If \( P = S \), then \( P' = S \), i.e., the point \( S \) is the fixed point of the mapping. If \( P \neq S \), \( S \) is the center point of a line segment connecting \( P \) and \( P' \). Therefore, the following equations result:

\[
\begin{align*}
\text{Column} & = \frac{x + x'}{2} \\
\text{Row} & = \frac{y + y'}{2}
\end{align*}
\]

If \( \text{Row} \) and \( \text{Column} \) are set to the origin, the in morphology often used transposition results. Hence \( \text{TransposeRegion} \) is often used to reflect (transpose) a structuring element.

---

**Parameter**

- **Region** (input iconic) ............................................. \( \text{region}(-\text{array}) \) \( \rightarrow \text{HRegionX} / \text{IableObjectX} \)
  - Region to be reflected.
- **Transposed** (output iconic) ............................... \( \text{region}(-\text{array}) \) \( \rightarrow \text{HRegionX} / \text{HUntypedObjectX} \)
  - Transposed region.
- **Row** (input control) ...................................................... \( \text{point.y} \) \( \rightarrow \text{long} \)
  - Row coordinate of the reference point.
  - **Default Value** : 0
  - **Value Suggestions** : \( \text{Row} \in \{0, 64, 128, 256, 512\} \)
  - **Typical Range of Values** : \( 0 \leq \text{Row} \leq 0(\text{lin}) \)
  - **Minimal Value Step** : 1
  - **Recommended Value Step** : 1
- **Column** (input control) .................................................. \( \text{point.x} \) \( \rightarrow \text{long} \)
  - Column coordinate of the reference point.
  - **Default Value** : 0
  - **Value Suggestions** : \( \text{Column} \in \{0, 64, 128, 256, 512\} \)
  - **Typical Range of Values** : \( 0 \leq \text{Column} \leq 0(\text{lin}) \)
  - **Minimal Value Step** : 1
  - **Recommended Value Step** : 1

---

**Result**

\( \text{TransposeRegion} \) returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- **no region** : \( \text{SetSystem}('\text{noObjectResult}',<\text{RegionResult}>) \)
- **empty region** : \( \text{SetSystem}('\text{emptyRegionResult}',<\text{RegionResult}>) \)

Otherwise, an exception is raised.

---

**Parallelization Information**

\( \text{TransposeRegion} \) is reentrant and automatically parallelized (on tuple level).

---

**Possible Successor Functions**

\( \text{ReduceDomain}, \text{SelectShape}, \text{AreaCenter}, \text{Connection} \)

---

**See Also**

\( \text{Dilation1}, \text{Opening}, \text{Closing} \)

---

**Module**

- **Morphology**

---

```halcon
[out] HRegionX RegionZoom HRegionX.ZoomRegion ([in] double ScaleWidth, [in] double ScaleHeight )
void HOperatorSetX.ZoomRegion ([in] IObjectX Region, [out] HUntypedObjectX RegionZoom, [in] double ScaleWidth, [in] double ScaleHeight )
```

**Zoom a region.**
ZoomRegion enlarges or reduces the regions given in Region in the x- and y-direction by the given scale factors ScaleWidth and ScaleHeight.

Parameter

- **Region** (input iconic) ...................................................... region(-array)  \(\sim HRegionX / HObjectX\)
  Region(s) to be zoomed.
- **RegionZoom** (output iconic) .......................... region(-array)  \(\sim HRegionX / HUntypedObjectX\)
  Zoomed region(s).

Parameter Number: \((RegionZoom = Region)\)

- **ScaleWidth** (input control) ...................................................... extent.x  \(\sim double\)
  Scale factor in x-direction.
  Default Value: 2.0
  Value Suggestions: ScaleWidth \(\in\{0.25, 0.5, 1.0, 2.0, 3.0\}\)
  Typical Range of Values: \(0.0 \leq \text{ScaleWidth} \leq 0.0(\text{lin})\)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.5

- **ScaleHeight** (input control) ...................................................... extent.y  \(\sim double\)
  Scale factor in y-direction.
  Default Value: 2.0
  Value Suggestions: ScaleHeight \(\in\{0.25, 0.5, 1.0, 2.0, 3.0\}\)
  Typical Range of Values: \(0.0 \leq \text{ScaleHeight} \leq 0.0(\text{lin})\)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.5

Parallelization Information

ZoomRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Connection, RegionGrowing, Pouring

Possible Successor Functions

SelectShape, DispRegion

See Also

ZoomImageSize, ZoomImageFactor

Module

Region processing

### 9.3 Creation

```c
```

```c
```

Create a checkered region.

The operator GenCheckerRegion returns a checkered region. Every black field of the checkerboard belongs to the region. The horizontal and vertical expansion of the region is limited by WidthRegion, HeightRegion respectively, the size of the fields of the checkerboard by WidthPattern \(\times\) HeightPattern.

Attention

If a very small pattern is chosen (WidthPattern \(<\) 4) the created region requires much storage.

Parameter

- **RegionChecker** (output iconic) .................................. region  \(\sim HRegionX / HUntypedObjectX\)
  Created checkerboard region.
9.3. CREAT ION

- **WidthRegion** (input control) .............................................................. extent.x \( \sim \) long
  Largest occurring \( x \) value of the region.
  - Default Value: 511
  - Value Suggestions: WidthRegion \( \in \{10, 20, 31, 63, 127, 255, 300, 400, 511\} \)
  - Typical Range of Values: \( 1 \leq \text{WidthRegion} \leq 1(\text{lin}) \)
  - Minimal Value Step: 1
  - Recommended Value Step: 10
  - Restriction: \((\text{WidthRegion} \geq 1)\)

- **HeightRegion** (input control) .............................................................. extent.y \( \sim \) long
  Largest occurring \( y \) value of the region.
  - Default Value: 511
  - Value Suggestions: HeightRegion \( \in \{10, 20, 31, 63, 127, 255, 300, 400, 511\} \)
  - Typical Range of Values: \( 1 \leq \text{HeightRegion} \leq 1(\text{lin}) \)
  - Minimal Value Step: 1
  - Recommended Value Step: 10
  - Restriction: \((\text{HeightRegion} \geq 1)\)

- **WidthPattern** (input control) .............................................................. extent.y \( \sim \) long
  Width of a field of the checkerboard.
  - Default Value: 64
  - Value Suggestions: WidthPattern \( \in \{1, 2, 4, 8, 16, 20, 32, 64, 100, 128, 200, 300, 500\} \)
  - Typical Range of Values: \( 1 \leq \text{WidthPattern} \leq 1(\text{lin}) \)
  - Minimal Value Step: 1
  - Recommended Value Step: 10
  - Restriction: \( (\text{WidthPattern} > 0) \land (\text{WidthPattern} < \text{WidthRegion}) \)

- **HeightPattern** (input control) .............................................................. extent.y \( \sim \) long
  Height of a field of the checkerboard.
  - Default Value: 64
  - Value Suggestions: HeightPattern \( \in \{1, 2, 4, 8, 16, 20, 32, 64, 100, 128, 200, 300, 500\} \)
  - Typical Range of Values: \( 1 \leq \text{HeightPattern} \leq 1(\text{lin}) \)
  - Minimal Value Step: 1
  - Recommended Value Step: 10
  - Restriction: \( (\text{HeightPattern} > 0) \land (\text{HeightPattern} < \text{HeightRegion}) \)

---

**Example**

gen_checker_region(Checker, 512, 512, 32, 64):
set_draw(WindowHandle, 'fill')
set_part(WindowHandle, 0, 0, 511, 511)
disp_region(Checker, WindowHandle)

---

**Result**

The operator GenCheckerRegion returns the value TRUE if the parameter values are correct. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem(‘clipRegion’, ‘true’ / ‘false’).

---

**Parallelization Information**

GenCheckerRegion is **reentrant** and processed **without** parallelization.

---

**Possible Successor Functions**

- PaintRegion
- HammingChangeRegion, ReduceDomain

---

**See Also**

- GenGridRegion, GenRegionPolygonFilled, GenRegionPoints, GenRegionRuns,
- GenRectangle1, ConcatObj, GenRandomRegion, GenRandomRegions

---

**Alternatives**

---

**Module**

Region processing
Create a circle.

The operator GenCircle generates one or more circles described by the center and Radius. If several circles shall be generated the coordinates must be passed in the form of tuples.

The coordinate system runs from (0,0) (upper left corner) to (Width-1,Height-1). See GetSystem and ResetObjDb in this context.

If an integer value (1,2,3...) is given for the radius the result is an even-numbered diameter and thus an asymmetrical circle. In case of an odd-numbered diameter (radius = 1.5,2.5,3.5...) a symmetrical circle is obtained.

If the circle extends beyond the image edge it is clipped to the current image format according to the value of the system flag ‘clip_region’ (SetSystem).

Parameter

- **Circle** (output iconic) .................................. region(-array) ～ HRegionX / HUntypedObjectX
  Generated circle.

- **Row** (input control) ................................. circle.center.y(-array) ～ VARIANT ( integer, real )
  Line index of center.
  Default Value : 200.0
  Value Suggestions : Row ∈ {0.0, 10.0, 50.0, 100.0, 200.0, 300.0}
  Typical Range of Values : 1.0 ≤ Row ≤ 1.0(lin)
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0

- **Column** (input control) ............................. circle.center.x(-array) ～ VARIANT ( integer, real )
  Column index of center.
  Default Value : 200.0
  Value Suggestions : Column ∈ {0.0, 10.0, 50.0, 100.0, 200.0, 300.0}
  Typical Range of Values : 1.0 ≤ Column ≤ 1.0(lin)
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0

- **Radius** (input control) ............................. circle.radius(-array) ～ VARIANT ( integer, real )
  Radius of circle.
  Default Value : 100.5
  Value Suggestions : Radius ∈ {1.0, 1.5, 2.0, 2.5, 3, 3.5, 4, 4.5, 5.5, 6.5, 7.5, 9.5, 11.5, 15.5, 20.5, 25.5, 31.5, 50.5}
  Typical Range of Values : 1.0 ≤ Radius ≤ 1.0(lin)
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0
  Restriction : (Radius > 0.0)

Example

```plaintext
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
read_image(Image,'meer')
gen_circle(Circle,300.0,200.0,150.5)
reduce_domain(Image,Circle,Mask)
disp_color(Mask,WindowHandle).
```

Result

If the parameter values are correct, the operator GenCircle returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem(‘clipRegion’,<‘true’/’false’>). If an empty region is created by clipping (the circle is completely outside of the image format) the operator SetSystem(‘storeGenEmptyRegion’,<true/false>) determines whether the empty region is put out.

Parallelization Information

GenCircle is reentrant and processed without parallelization.

Possible Successor Functions

PaintRegion, ReduceDomain

See Also

DispCircle, SetShape, SmallestCircle, ReduceDomain

Alternatives

GenEllipse, GenRegionPolygonFilled, GenRegionPoints, GenRegionRuns, DrawCircle

Region processing

Create an ellipse.

The operator GenEllipse generates one or more ellipses with the center (Row, Column), the orientation Phi and the half-radii Radius1 and Radius2. The angle is indicated in arc measure according to the \( \pi \) axis in mathematically positive direction. More than one region can be created by passing tuples of parameter values.

The center must be located within the image coordinates. The coordinate system runs from (0.0) (upper left corner) to (Width-1,Height-1). See GetSystem and ResetObjDb in this context. If the ellipse reaches beyond the edge of the image it is clipped to the current image format according to the value of the system flag 'clip region' (SetSystem).

Parameter

- **Ellipse** (output iconic) .................................................. region(-array) \( \sim \) HRegionX / HUntypedObjectX
  Created ellipse(s).

- **Row** (input control) .................................................. ellipse.center.y(-array) \( \sim \) VARIANT ( integer, real )
  Line index of center.
  **Default Value:** 200.0
  **Value Suggestions:** Row \( \in \) \{ 0.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0 \}
  **Typical Range of Values:** 1.0 \( \leq \) Row \( \leq \) 1.0(lin)
  **Minimal Value Step:** 1.0
  **Recommended Value Step:** 10.0

- **Column** (input control) .................................................. ellipse.center.x(-array) \( \sim \) VARIANT ( integer, real )
  Column index of center.
  **Default Value:** 200.0
  **Value Suggestions:** Column \( \in \) \{ 0.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0 \}
  **Typical Range of Values:** 1.0 \( \leq \) Column \( \leq \) 1.0(lin)
  **Minimal Value Step:** 1.0
  **Recommended Value Step:** 10.0

- **Phi** (input control) .................................................. ellipse.angle.rad(-array) \( \sim \) VARIANT ( integer, real )
  Orientation of the longer radius (Radius1).
  **Default Value:** 0.0
  **Value Suggestions:** Phi \( \in \) \{ -1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097 \}
  **Typical Range of Values:** -1.178097 \( \leq \) Phi \( \leq \) -1.178097(lin)
  **Minimal Value Step:** 0.01
  **Recommended Value Step:** 0.1
CHAPTER 9. REGIONS

**Radius1** (input control) .................. ellipse.radius1(-array) ~ VARIANT ( integer, real )
   Longer radius.
   **Default Value :** 100.0
   **Value Suggestions:** Radius1 ∈ \{2.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0\}
   **Typical Range of Values :** 1.0 ≤ Radius1 ≤ 1.0(lin)
   **Minimal Value Step :** 1.0
   **Recommended Value Step :** 10.0
   **Restriction :** (Radius1 > 0)

**Radius2** (input control) .................. ellipse.radius2(-array) ~ VARIANT ( integer, real )
   Shorter radius.
   **Default Value :** 60.0
   **Value Suggestions:** Radius2 ∈ \{1.0, 2.0, 4.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0\}
   **Typical Range of Values :** 1.0 ≤ Radius2 ≤ 1.0(lin)
   **Minimal Value Step :** 1.0
   **Recommended Value Step :** 10.0
   **Restriction :** ((Radius2 > 0) ∧ (Radius2 ≤ Radius1))

---

**Example**

```plaintext
open_window(0,0,-1,-1,'root','visible','
set_insert(WindowHandle,'xor')
repeat()
   get_mbutton(WindowHandle,Row,Column,Button)
   gen_ellipse(Ellipse,Row,Column,(Column / 300.0),(Row mod 100)+1,(Column mod 50) + 1)
   disp_region(Ellipse,WindowHandle)
   clear_obj(Ellipse)
until(Button = 1).
```

---

**Result**

If the parameter values are correct, the operator GenEllipse returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem(‘clipRegion’,<‘true’/’false’>).

---

**Parallelization Information**

GenEllipse is reentrant and processed without parallelization.

---

**Possible Successor Functions**

PaintRegion, ReduceDomain

---

**See Also**

DispEllipse, SetShape, SmallestCircle, ReduceDomain

---

**Alternatives**

GenCircle, GenRegionPolygonFilled, DrawEllipse

---

**Module**

Region processing

---

```plaintext
void HRegionX.GenEmptyRegion ( )
void HOperatorSetX.GenEmptyRegion
   ([out] HUntypedObjectX EmptyRegion )
```

Create an empty region.

The operator GenEmptyRegion creates an empty region. This means that the output parameter contains an object. Thus, CountObj returns 1. The area of the region is 0. Most of the shape features are undefined (0). It should be noted that an empty region must not be confused with the empty tuple.
Parameter

- **EmptyRegion** (output iconic)  
  Empty region (no pixels).

Parallelization Information

GenEmptyRegion is reentrant and processed without parallelization.

Module

Region processing

```cpp
void HRegionX.GenGridRegion ([in] VARIANT RowSteps,
   [in] VARIANT ColumnSteps, [in] String Type, [in] long Width,
   [in] long Height )

void HOperatorSetX.GenGridRegion ([out] HUntypedObjectX RegionGrid,
   [in] VARIANT RowSteps, [in] VARIANT ColumnSteps, [in] String Type,
   [in] long Width, [in] long Height )
```

Create a region from lines or pixels.

The operator GenGridRegion creates a grid constructed of lines (`Type = ’lines’`) or pixels (`Type = ’points’`). In case of ’lines’ continuous lines are returned, in case of ’points’ only the intersections of the lines. Starting from the pixel (0,0) to the pixel (`Height`, `Width`) the grid is built up at stepping width `RowSteps` in line direction and `ColumnSteps` in column direction. In the ’lines’ mode `RowSteps`, `ColumnSteps` respectively, can be set to zero. In this case only columns, lines respectively, are created.

**Attention**

If a very small pattern is chosen (`RowSteps < 4` oder `ColumnSteps < 4`) the created region requires much storage.

In the ’points’ mode `RowSteps` and `ColumnSteps` must not be set to zero.

Parameter

- **RegionGrid** (output iconic)  
  Created lines/pixel region.

- **RowSteps** (input control)  
  Step width in line direction or zero.
  **Default Value**: 10
  **Value Suggestions**: `RowSteps ∈ {0, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 100}`
  **Typical Range of Values**: `0 ≤ RowSteps ≤ 0(lin)`
  **Minimal Value Step**: 1
  **Recommended Value Step**: 10
  **Restriction**: `((RowSteps > 1) ≡ (RowSteps = 0))`

- **ColumnSteps** (input control)  
  Step width in column direction or zero.
  **Default Value**: 10
  **Value Suggestions**: `ColumnSteps ∈ {0, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 100}`
  **Typical Range of Values**: `0 ≤ ColumnSteps ≤ 0(lin)`
  **Minimal Value Step**: 1
  **Recommended Value Step**: 10
  **Restriction**: `((ColumnSteps > 1) ≡ (ColumnSteps = 0))`

- **Type** (input control)  
  Type of created pattern.
  **Default Value**: ’lines’
  **Value List**: `Type ∈ {'lines', 'points'}`
**Width** (input control) ............................................................... \text{extent.x} \sim \text{long}
- Maximum width of pattern.
  - Default Value: 512
  - Value Suggestions: \( \text{Width} \in \{128, 256, 512, 1024\} \)
  - Typical Range of Values: \( 1 \leq \text{Width} \leq 1\text{(lin)} \)
  - Minimal Value Step: 1
  - Recommended Value Step: 10
  - Restriction: \((\text{Width} \geq 1)\)

**Height** (input control) ............................................................... \text{extent.y} \sim \text{long}
- Maximum height of pattern.
  - Default Value: 512
  - Value Suggestions: \( \text{Height} \in \{128, 256, 512, 1024\} \)
  - Typical Range of Values: \( 1 \leq \text{Height} \leq 1\text{(lin)} \)
  - Minimal Value Step: 1
  - Recommended Value Step: 10
  - Restriction: \((\text{Height} \geq 1)\)

---

**Example**

read_image(Image,’fabrik’)
gen_grid_region(Raster,10,10,’lines’,512,512)
reduce_domain(Image,Raster,Mask)
sobel_amp(Mask,GridSobel,’sum_abs’,3)
disp_image(GridSobel,WindowHandle).

---

**Result**

If the parameter values are correct the operator GenGridRegion returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem(‘clipRegion’,<’true’/’false’>).

---

**Parallelization Information**

GenGridRegion is reentrant and processed without parallelization.

---

**Possible Successor Functions**

ReduceDomain, PaintRegion

---

**See Also**

GenCheckerRegion, ReduceDomain

---

**Alternatives**

GenRegionLine, GenRegionPolygon, GenRegionPoints, GenRegionRuns

---

**Module**

Region processing

---

```c
void HRegionX.GenRandomRegion ([in] long Width, [in] long Height )

void HOperatorSetX.GenRandomRegion
([out] HUntypedObjectX RegionRandom, [in] long Width, [in] long Height )
```

Create a random region.

The operator GenRandomRegion returns a random region. During this process every pixel in the image area \([0 \ldots \text{Width} - 1][0 \ldots \text{Height} - 1]\) is adapted into the region with the probability 0.5. The created region can be imagined as the threshold formation in an image with noise.

This procedure is particularly important for the creation of uncorrelated binary patterns. The random pattern is created by the C function “nrand48()”.

---

**Attention**

If Width and Height are chosen large (> 100) the created region may require much storage space due to the internally used runlength coding. The gray values of the output region are undefined.
9.3. CREATION

**Parameter**

- **RegionRandom** (output iconic) .............................. region \( \sim HRegionX / HUntypedObjectX \)

  Created random region with expansion Width x Height.

- **Width** (input control) .............................................. extent.x \( \sim \) long

  Maximum horizontal expansion of random region.

  Default Value: 128
  
  Value Suggestions: Width \( \in \{16, 32, 50, 64, 100, 128, 256, 300, 400, 512\} \)
  
  Typical Range of Values: \( 1 \leq \text{Width} \leq 1(\text{lin}) \)
  
  Minimal Value Step: 1
  
  Recommended Value Step: 10
  
  Restriction: \((\text{Width} \geq 0)\)

- **Height** (input control) .............................................. extent.y \( \sim \) long

  Maximum vertical expansion of random region.

  Default Value: 128
  
  Value Suggestions: Height \( \in \{16, 32, 50, 64, 100, 128, 256, 300, 400, 512\} \)
  
  Typical Range of Values: \( 1 \leq \text{Height} \leq 1(\text{lin}) \)
  
  Minimal Value Step: 1
  
  Recommended Value Step: 10
  
  Restriction: \((\text{Height} \geq 0)\)

**Result**

If the parameter values are correct, the operator GenRandomRegion returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem('clipRegion',<true'/false'>).

**Parallelization Information**

GenRandomRegion is **reentrant** and processed **without** parallelization.

**Possible Successor Functions**

PaintRegion, ReduceDomain

**See Also**

GenCheckerRegion, HammingChangeRegion, AddNoiseDistribution, AddNoiseWhite, ReduceDomain

**Module**

Region processing

```c
void HRegionX.GenRandomRegions ([in] String Type,  
[in] VARIANT WidthMin, [in] VARIANT WidthMax, [in] VARIANT HeightMin,  
[in] VARIANT HeightMax, [in] VARIANT PhiMin, [in] VARIANT PhiMax,  

void HOperatorSetX.GenRandomRegions ([out] HUntypedObjectX Regions,  
[in] String Type, [in] VARIANT WidthMin, [in] VARIANT WidthMax,  
[in] VARIANT HeightMin, [in] VARIANT HeightMax, [in] VARIANT PhiMin,  
```

Create random regions like circles, rectangles and ellipses.

The operator GenRandomRegion generates circles, rectangles and ellipses whose parameters are determined at random. In each case only one lower, upper limit respectively, is given. The position is always random and cannot be determined by parameters. The parameter NumRegions indicates how many regions shall be created.

**Parameter**

- **Regions** (output iconic) .............................. region \( \sim HRegionX / HUntypedObjectX \)

  Created regions.

- **Type** (input control) .............................. string \( \sim \) String

  Type of regions to be created.

  Default Value: 'circle'

  Value List: Type \( \in \{\text{‘circle’, ‘ring’, ‘ellipse’, ‘rectangle1’, ‘rectangle2’}\} \)
**WidthMin** (input control) ................................. number ~ VARIANT (real, integer)
Minimum width of the region.
  Default Value: 10.0
  Value Suggestions: WidthMin ∈ \{1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0\}
  Typical Range of Values: 1.0 ≤ WidthMin ≤ 1.0 (lin)
  Minimal Value Step: 1.0
  Recommended Value Step: 10.0
  Restriction: (WidthMin > 0)

**WidthMax** (input control) ................................. number ~ VARIANT (real, integer)
Maximum width of the region.
  Default Value: 20.0
  Value Suggestions: WidthMax ∈ \{1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0\}
  Typical Range of Values: 1.0 ≤ WidthMax ≤ 1.0 (lin)
  Minimal Value Step: 1.0
  Recommended Value Step: 10.0
  Restriction: (WidthMax > 0)

**HeightMin** (input control) ................................. number ~ VARIANT (real, integer)
Minimum height of the region.
  Default Value: 10.0
  Value Suggestions: HeightMin ∈ \{1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0\}
  Typical Range of Values: 1.0 ≤ HeightMin ≤ 1.0 (lin)
  Minimal Value Step: 1.0
  Recommended Value Step: 10.0
  Restriction: (HeightMin > 0)

**HeightMax** (input control) ................................. number ~ VARIANT (real, integer)
Maximum height of the region.
  Default Value: 30.0
  Value Suggestions: HeightMax ∈ \{1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0\}
  Typical Range of Values: 1.0 ≤ HeightMax ≤ 1.0 (lin)
  Minimal Value Step: 1.0
  Recommended Value Step: 10.0
  Restriction: (HeightMax > 0)

**PhiMin** (input control) ................................. number ~ VARIANT (real, integer)
Minimum rotation angle of the region.
  Default Value: -0.7854
  Value Suggestions: PhiMin ∈ \{0.0, 0.1, 0.3, 0.6, 0.9, 1.2, 1.5\}
  Typical Range of Values: 0.0 ≤ PhiMin ≤ 0.0 (lin)
  Minimal Value Step: 0.001
  Recommended Value Step: 0.10
  Restriction: (PhiMin > 0)

**PhiMax** (input control) ................................. number ~ VARIANT (real, integer)
Maximum rotation angle of the region.
  Default Value: 0.7854
  Value Suggestions: PhiMax ∈ \{0.0, 0.1, 0.3, 0.6, 0.9, 1.2, 1.5\}
  Typical Range of Values: 0.0 ≤ PhiMax ≤ 0.0 (lin)
  Minimal Value Step: 0.001
  Recommended Value Step: 0.10
  Restriction: (PhiMax > 0)

**NumRegions** (input control) ................................. integer ~ long
Number of regions.
  Default Value: 100
  Value Suggestions: NumRegions ∈ \{1, 5, 20, 100, 200, 500, 1000, 2000\}
  Typical Range of Values: 1 ≤ NumRegions ≤ 1 (lin)
  Minimal Value Step: 1
  Recommended Value Step: 10
  Restriction: (NumRegions > 0)
9.3. CREAT I ON

- **Width** (input control) .................................................. integer \( \sim \) long
  
  Maximum horizontal expansion.

  **Default Value**: 512
  
  **Value Suggestions**: Width \( \in \{128, 256, 512, 1024\} \)
  
  **Typical Range of Values**: \( 1 \leq \text{Width} \leq \text{(lin)} \)
  
  **Minimal Value Step**: 1
  
  **Recommended Value Step**: 10
  
  **Restriction**: (\( \text{Width} > 0 \))

- **Height** (input control) .................................................. integer \( \sim \) long
  
  Maximum vertical expansion.

  **Default Value**: 512
  
  **Value Suggestions**: Height \( \in \{128, 256, 512, 1024\} \)
  
  **Typical Range of Values**: \( 1 \leq \text{Height} \leq \text{(lin)} \)
  
  **Minimal Value Step**: 1
  
  **Recommended Value Step**: 10
  
  **Restriction**: (\( \text{Height} > 0 \))

---

**Result**

If the parameter values are correct GenRandomRegions returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is determined by the operator SetSystem('clipRegion',<true'/false'>).

---

**Parallelization Information**

GenRandomRegions is *reentrant* and processed *without* parallelization.

---

**Possible Successor Functions**

PaintRegion

---

**Module**

Region processing

---

```c
void HRegionX.GenRectangle1 ([in] VARIANT Row1, [in] VARIANT Column1,
[in] VARIANT Row2, [in] VARIANT Column2 )

void HOperatorSetX.GenRectangle1 ([out] HUntypedObjectX Rectangle,
[in] VARIANT Row1, [in] VARIANT Column1, [in] VARIANT Row2,
[in] VARIANT Column2 )
```

Create a rectangle parallel to the coordinate axes.

The operator GenRectangle1 generates one or more rectangles parallel to the coordinate axes which are described by the upper left corner (Row1,Column1) and the lower right corner (Row2,Column2). More than one region can be created by passing a tuple of corner points. The coordinate system runs from (0,0) (upper left corner) to (Width-1,Height-1). See GetSystem and ResetObjDb in this context.

---

**Parameter**

- **Rectangle** (output iconic) ......................... region(-array) \( \sim \) HRegionX/HUntypedObjectX
  
  Created rectangle.

- **Row1** (input control) ......................... rectangle\(\).origin.y(-array) \( \sim \) VARIANT (integer, real)
  
  Line of upper left corner point.

  **Default Value**: 30.0
  
  **Value Suggestions**: Row1 \( \in \{0.0, 10.0, 20.0, 50.0, 100.0, 200.0\} \)
  
  (lin) **Minimal Value Step**: 1.0
  
  **Recommended Value Step**: 10.0

- **Column1** (input control) ......................... rectangle\(\).origin.x(-array) \( \sim \) VARIANT (integer, real)
  
  Column of upper left corner point.

  **Default Value**: 20.0
  
  **Value Suggestions**: Column1 \( \in \{0.0, 10.0, 20.0, 50.0, 100.0, 200.0\} \)
  
  (lin) **Minimal Value Step**: 1.0
  
  **Recommended Value Step**: 10.0
**Row2** (input control) .......................... rectangle.corner.y(-array)  \( \rightarrow \) VARIANT (integer, real)

Line of lower right corner point.

Default Value: 100.0

Value Suggestions: \( \text{Row2} \in \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0, 511.0\} \)

(line) Minimal Value Step: 1.0

Recommended Value Step: 10.0

Restriction: \((\text{Row2} \geq \text{Row1})\)

**Column2** (input control) .......................... rectangle.corner.x(-array)  \( \rightarrow \) VARIANT (integer, real)

Column of lower right corner point.

Default Value: 200.0

Value Suggestions: \( \text{Column2} \in \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0, 511.0\} \)

(line) Minimal Value Step: 1.0

Recommended Value Step: 10.0

Restriction: \((\text{Column2} \geq \text{Column1})\)

--- Example ---

/* Contrast improvement in a rectangular region of interest */

read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
disp_image(Image,WindowHandle)
draw_rectangle1(WindowHandle,Row1,Column1,Row2,Column2)
gen_rectangle1(Rectangle,Row1,Column1,Row2,Column2)
reduce_domain(Image,Rectangle,Mask)
emphasize(Mask,Emphasize,9,9,1.0)
disp_image(Emphasize,WindowHandle).

--- Result ---

If the parameter values are correct, the operator GenRectangle1 returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem(‘clipRegion’,<‘true’/’false’)).

--- Parallelization Information ---

GenRectangle1 is reentrant and processed without parallelization.

--- Possible Successor Functions ---

PaintRegion, ReduceDomain

--- See Also ---

DrawRectangle1, ReduceDomain, SmallestRectangle1

--- Alternatives ---

GenRectangle2, GenRegionPolygon, FillUp, GenRegionRuns, GenRegionPoints, GenRegionLine

--- Module ---

Region processing

```c
void HRegionX.GenRectangle2 ( [in] VARIANT Row, [in] VARIANT Column,
void HOperatorSetX.GenRectangle2 ( [out] HUntypedObjectX Rectangle,
                                   [in] VARIANT Length2 )
```

Create a rectangle of any orientation.

The operator GenRectangle2 generates one or more rectangles with the center \((\text{Row}, \text{Column})\), the orientation \(\Phi\) and the half edge lengths \(\text{Length1}\) and \(\text{Length2}\). The orientation is given in arc measure and indicates the angle between the horizontal axis and \(\text{Length1}\) (mathematically positive). The coordinate system runs from \((0,0)\) (upper left corner) to \((\text{Width}-1,\text{Height}-1)\). See GetSystem and ResetObjDb in this context. More than one region can be created by passing one tuple of corner points.
The gray values of the output objects are undefined.

**Parameter**

- **Rectangle** (output iconic) ...................... region(-array)  \(\sim\) HRegionX / HUntypedObjectX
  Created rectangle.

- **Row** (input control) ........................... rectangle2.center.y(-array)  \(\sim\) VARIANT (integer, real)
  Line index of the center.
  Default Value : 50.0
  Value Suggestions : Row \(\in\) \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0\}
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0

- **Column** (input control) ......................... rectangle2.center.x(-array)  \(\sim\) VARIANT (integer, real)
  Column index of the center.
  Default Value : 100.0
  Value Suggestions : Column \(\in\) \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0\}
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0

- **Phi** (input control) ........................... rectangle2.angle.rad(-array)  \(\sim\) VARIANT (integer, real)
  Angle of longitudinal axis to horizontal (arc measure).
  Default Value : 0.0
  Value Suggestions : Phi \(\in\) \{-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097\}
  Typical Range of Values : -1.178097 \(\leq\) Phi \(\leq\) -1.178097 (lin)
  Minimal Value Step : 0.001
  Recommended Value Step : 0.1
  Restriction : \((\frac{-\pi}{2} < Phi \land (Phi \leq \frac{-\pi}{2}))\)

- **Length1** (input control) ....................... rectangle2.hwidth(-array)  \(\sim\) VARIANT (integer, real)
  Half width.
  Default Value : 200.0
  Value Suggestions : Length1 \(\in\) \{3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0, 300.0, 500.0\}
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0

- **Length2** (input control) ....................... rectangle2.hheight(-array)  \(\sim\) VARIANT (integer, real)
  Half height.
  Default Value : 100.0
  Value Suggestions : Length2 \(\in\) \{1.0, 2.0, 3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0\}
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0
  Restriction : (Length2 \(\leq\) Length1)

**Result**

If the parameter values are correct the operator GenRectangle2 returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem(’clipRegion’,<’true’/’false’>).

**Parallelization Information**

GenRectangle2 is reentrant and processed without parallelization.

**Possible Successor Functions**

PaintRegion, ReduceDomain

**See Also**

DrawRectangle2, ReduceDomain, SmallestRectangle2, GenEllipse

**Alternatives**

GenRectangle1, GenRegionPolygonFilled, GenRegionPolygon, GenRegionPoints, FillUp

**Module**

Region processing


Convert a histogram into a region.

GenRegionHisto converts a histogram created with GrayHisto into a region. The effect of the three control parameters is the same as in DispImage and SetPaint.

Parameter

► Region (output iconic) .................................. region \(\sim\) HRegionX / HUntypedObjectX Region containing the histogram.

► Histogramm (input control) .................................. histogram \(\sim\) <none>X / VARIANT (integer) Input histogram.

► Row (input control) ........................................................ point.y \(\sim\) long Row coordinate of the center of the histogram.

  Default Value : 255
  Value Suggestions : Row \(\in\) \{100, 200, 255, 300, 400\}
  Typical Range of Values : \(0 \leq\) Row \(\leq 0\)

► Column (input control) ................................................ point.x \(\sim\) long Column coordinate of the center of the histogram.

  Default Value : 255
  Value Suggestions : Column \(\in\) \{100, 200, 255, 300, 400\}
  Typical Range of Values : \(0 \leq\) Column \(\leq 0\)

► Scale (input control) ........................................... integer \(\sim\) long Scale factor for the histogram.

  Default Value : 1
  Value List : Scale \(\in\) \{1, 2, 3, 4, 5, 6, 7\}
  Typical Range of Values : \(1 \leq\) Scale \(\leq 1\) (lin)
  Minimal Value Step : 1
  Recommended Value Step : 1

Result

GenRegionHisto returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

Parallelization Information

GenRegionHisto is reentrant and processed without parallelization.

Possible Predecessor Functions

GrayHisto

DispChannel, SetPaint

Module

Region processing

Store input lines described in Hesse normal shape as regions.

The operator GenRegionHline stores the lines described in Hesse normal shape as regions. A line is determined by the distance from the line to the origin (Distance, corresponds to the length of the normal vector) and the direction of the normal vector (Orientation, corresponds to the orientation of the line \(\pm \pi/2\)). The directions
were defined in such a way that at \( \text{Orientation} = 0 \) the normal vector lies in the direction of the X axis, which corresponds to a vertical line. At \( \text{Orientation} = \pi/2 \) the normal vector points in the direction of the Y axis, i.e. a horizontal line is described.

\[ \text{Attention} \]

The lines are clipped to the current maximum image format.

\[ \text{Parameter} \]

\( \text{Regions} \) (output iconic) \( \Rightarrow \) region(-array) \( \sim \) \( \text{HRegionX} / \text{HUtypedObjectX} \)

Created regions (one for every line), clipped to maximum image format.

\( \text{Parameter Number} : (\text{Regions} = \text{Distance}) \)

\( \text{Orientation} \) (input control) \( \Rightarrow \) hesseline.angle.rad(-array) \( \sim \) VARIANT (integer, real)

Orientation of the normal vector in radians.

Default Value : 0.0

Value Suggestions : Orientation \( \in \{-0.78, 0.0, 0.78, 1.57\} \)

Recommended Value Step : 0.02

\( \text{Distance} \) (input control) \( \Rightarrow \) hesseline.distance(-array) \( \sim \) VARIANT (integer, real)

Distance from the line to the coordinate origin (0.0).

Default Value : 200

Value Suggestions : Distance \( \in \{10, 50, 100, 200, 300, 400\} \)

Recommended Value Step : 1

\[ \text{Result} \]

The operator GenRegionHline always returns the value TRUE.

\[ \text{Parallelization Information} \]

GenRegionHline is reentrant and processed without parallelization.

\[ \text{See Also} \]

HoughLines

\[ \text{Alternatives} \]

GenRegionLine

\[ \text{Module} \]

Region processing


Store input lines as regions.

The operator GenRegionLine stores the given lines (with starting point [BeginRow,BeginCol] and ending point [EndRow,EndCol]) as region.

\[ \text{Attention} \]

\[ \text{Parameter} \]

\( \text{RegionLines} \) (output iconic) \( \Rightarrow \) region(-array) \( \sim \) \( \text{HRegionX} / \text{HUtypedObjectX} \)

Created regions.

\( \text{BeginRow} \) (input control) \( \Rightarrow \) line.begin.y(-array) \( \sim \) VARIANT (integer)

Line coordinates of the starting points of the input lines.

Default Value : 100

Value Suggestions : BeginRow \( \in \{10, 50, 100, 200, 300, 400\} \)

Minimal Value Step : 1

Recommended Value Step : 1
The operator \texttt{GenRegionLine} always returns the value \texttt{TRUE}. The clipping according to the current image format is determined by the operator \texttt{SetSystem(’clipRegion’,<’true’/’false’)}.

\textbf{Parallelization Information}

\texttt{GenRegionLine} is \textit{reentrant} and processed \textit{without parallelization}.

\textbf{Possible Predecessor Functions}

\texttt{SplitSkeletonLines}

\textbf{Alternatives}

\texttt{GenRegionHline}

\textbf{Module}

\textbf{Region processing}

\begin{verbatim}
void HRegionX.GenRegionPoints ([in] VARIANT Rows,
[in] VARIANT Columns )
void HOperatorSetX.GenRegionPoints ([out] HUntypedObjectX Region,
\end{verbatim}

\textit{Store individual pixels as image region.}

The operator \texttt{GenRegionPoints} creates a region described by a number of pixels. The pixels do not have to be stored in a fixed order, but the best runtime behavior is obtained when the pixels are stored in ascending order. The order is as follows:

\[ (l_1, c_1) \leq (l_2, c_2) := (l_1 < l_2) \lor (l_1 = l_2) \land (c_1 \leq c_2) \]

The indicated coordinates stand for two consecutive pixels in the tuple.

\textbf{Attention}

The gray values of the output regions are undefined. All pixels must be located within the image format. If no pixels are passed an empty region is created.

\textbf{Parameter}

\textbf{Region} (output iconic) ...................................... region \textit{\textasciitilde} HRegionX / HUntypedObjectX

\textit{Created region.}

\textbf{Rows} (input control) ...................................... coordinates.y(-array) \textit{\textasciitilde} VARIANT( integer )

\textit{Lines of the pixels in the region.}

\textbf{Default Value} : 100

\textbf{Value Suggestions} : Rows \in \{0, 10, 30, 50, 100, 200, 300, 500\}

\textbf{(lin)Minimal Value Step} : 1

\textbf{Recommended Value Step} : 1
9.3. CREATION 473

Columns (input control) .................................. coordinates.x(-array) \( \sim \) VARIANT (integer)
Columns of the pixels in the region.

Default Value : 100
Value Suggestions : Columns \( \in \{0, 10, 30, 50, 100, 200, 300, 500\}\)

(rows) Min Value Step : 1
Recommended Value Step : 1

Parameter Number : \((Columns = \text{Rows})\)

The operator GenRegionPoints returns the value TRUE if the pixels are located within the image format.

Otherwise an exception handling is raised. The clipping according to the current image format is set via the
operator SetSystem('clipRegion','<true'/'false'>). If an empty region is created (empty input)
the operator SetSystem('storeGenEmptyRegion','<true/false>') determines whether the region is
returned.

Parallelization Information
GenRegionPoints is reentrant, local, and processed without parallelization.

Possible Predecessor Functions
GetRegionPoints

Possible Successor Functions
PaintRegion, ReduceDomain

See Also
ReduceDomain

Alternatives
GenRegionPolygon, GenRegionRuns, GenRegionLine

Module
Region processing

Store a polygon as an image object.

The operator GenRegionPolygon creates a region from a polygon row described by a series of line and col-
umn coordinates. The created region consists of the pixels of the routes defined thereby, wherein it is linearily
interpolated between the base points.

Attention
The region is automatically closed and not filled. The gray values of the output regions are undefined. All base
points must be located within the image format. If no pixels are passed an empty region is created.

Parameter

Region (output iconic) ................................. region \( \sim \) HRegionX/HUntypedObjectX
Created region.

Rows (input control) .................................. polygon.y \( \sim \) VARIANT (integer)
Line indices of the base points of the region contour.

Default Value : 100
Value Suggestions : Rows \( \in \{0, 10, 30, 50, 100, 200, 300, 500\}\)

(rows) Minimal Value Step : 1
Recommended Value Step : 1

Columns (input control) ................................. polygon.x \( \sim \) VARIANT (integer)
Column indices of the base points of the region contour.

Default Value : 100
Value Suggestions : Columns \( \in \{0, 10, 30, 50, 100, 200, 300, 500\}\)

(rows) Minimal Value Step : 1
Recommended Value Step : 1

Parameter Number : \((Columns = \text{Rows})\)
/* Polygon-approximation*/
get_region_polygon(Region,7,Row,Column)
/* store it as a region */
gen_region_polygon(Pol,Row,Column)
/* fill up the hole */
fill_up(Pol,Filled).

Result
If the base points are correct the operator GenRegionPolygon returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem(‘clipRegion’,<‘true’/’false’>). If an empty region is created the operator SetSystem(‘storeGenEmptyRegion’,<true/false>) determines whether the region is returned.

Parallelization Information
GenRegionPolygon is reentrant, local, and processed without parallelization.

Possible Predecessor Functions
GetRegionPolygon, DrawPolygon

See Also
FillUp, ReduceDomain, GetRegionPolygon, DrawPolygon

Alternatives
GenRegionPolygonFilled, GenRegionPoints, GenRegionRuns

Module
Region processing

void HRegionX.GenRegionPolygonFilled ([in] VARIANT Rows, [in] VARIANT Columns)
void HOperatorSetX.GenRegionPolygonFilled ([out] HUntypedObjectX Region, [in] VARIANT Rows, [in] VARIANT Columns)

Store a polygon as a “filled” region.
The operator GenRegionPolygonFilled creates a region from a polygon containing the corner points of the region (line and column coordinates) either clockwise or anti-clockwise. Contrary to GenRegionPolygon a “filled” region is returned here.

Attention
All base points must be located within the image format. If no pixels are passed an empty region is created.

Parameter

» Region (output iconic) ..................................... region ~ HRegionX / HUntypedObjectX
Created region.

» Rows (input control) ...................................... polygon.y ~ VARIANT( integer )
Line indices of the base points of the region contour.
Default Value : 100
Value Suggestions : Rows ∈ {0, 10, 30, 50, 100, 200, 300, 500}
(lin)Minimal Value Step : 1
Recommended Value Step : 1

» Columns (input control) ..................................... polygon.x ~ VARIANT( integer )
Column indices of the base points of the region contour.
Default Value : 100
Value Suggestions : Columns ∈ {0, 10, 30, 50, 100, 200, 300, 500}
(lin)Minimal Value Step : 1
Recommended Value Step : 1
Parameter Number : (Columns = Rows)
If the base points are correct the operator GenRegionPolygonFilled returns the value TRUE. Otherwise an exception handling is raised. If an empty region is created the operator SetSystem(‘storeGenEmptyRegion’,<true/false>) determines whether the region is returned.

GenRegionPolygonFilled is reentrant, local, and processed without parallelization.

Possible Predecessor Functions

See Also

Alternatives

Module

Region processing

Create an image region from a runlength coding.

The operator GenRegionRuns creates a region described by the input runlength structure. The runlength representation is created by examining a region line by line with ascending line number (= from “top” to “bottom”). Every line runs through from left to right (ascending column number) All starting and ending points being stored by region segments (=runs). Thus a region can be described by a sequence of runs, a run being defined by line number as well as starting and ending points (column number).

The storing is fastest when the runs are sorted. The order is as follows:

\[(l_1, b_1, e_1) \leq (l_2, b_2, e_2) \iff (l_1 < l_2) \lor (l_1 = l_2) \land (b_1 \leq b_2)\]

Attention

All pixels must be located within the image format. If no runs are passed an empty region is created.

Parameter

- **Region** (output iconic) .................................................. region \( \sim HRegionX / HUntypedObjectX\)
  
  Created region.

- **Row** (input control) .................................................. chord.y(-array) \( \sim \) VARIANT(integer)
  
  Lines of the runs.
  
  **Default Value**: 100
  
  **Value Suggestions**: Row \( \in \{0, 50, 100, 200, 300, 500\}\)
  
  (lin) **Minimal Value Step**: 1
  
  **Recommended Value Step**: 10

- **ColumnBegin** (input control) .................................... chord.x1(-array) \( \sim \) VARIANT(integer)
  
  Columns of the starting points of the runs.
  
  **Default Value**: 50
  
  **Value Suggestions**: ColumnBegin \( \in \{0, 50, 100, 200, 300, 500\}\)
  
  (lin) **Minimal Value Step**: 1
  
  **Recommended Value Step**: 10
  
  **Parameter Number**: \( (ColumnBegin = Row) \)
\textbf{ColumnEnd} (input control) \hspace{1cm} \textbf{chord.x2(-array)} \Rightarrow \textbf{VARIANT ( integer )}

Columns of the ending points of the runs.

\textbf{Default Value :} 200
\textbf{Value Suggestions :} \texttt{ColumnEnd \in \{50, 100, 200, 300, 500\}}
\textbf{Minimal Value :} 1
\textbf{Recommended Value Step :} 10
\textbf{Restriction :} \hspace{1cm} \texttt{(ColumnEnd \geq ColumnBegin)}
\textbf{Parameter Number :} \hspace{1cm} \texttt{(ColumnEnd = Row)}

\textbf{Result}

If the data is correct the operator \texttt{GenRegionRuns} returns the value \texttt{TRUE}, otherwise an exception handling is raised. The clipping according to the current image format is set via the operator \texttt{SetSystem(‘clipRegion’,<’true’/’false’>}). If an empty region is created (= empty input) the operator \texttt{SetSystem(‘storeGenEmptyRegion’,<true/false>)} determines whether the region is returned.

\textbf{Parallelization Information}

\texttt{GenRegionRuns} is \textit{reentrant, local}, and processed \textit{without} parallelization.

\textbf{Possible Predecessor Functions}

\texttt{GetRegionRuns}

\textbf{See Also}

\texttt{ReduceDomain}

\textbf{Alternatives}

\texttt{GenRegionPoints, GenRegionPolygon, GenRegionLine, GenRegionPolygonFilled}

\textbf{Module}

\textit{Region processing}

\begin{verbatim}
[out] HRegionX Regions HImageX.LabelToRegion ( )

void HOperatorSetX.LabelToRegion ([in] IObjectX LabelImage,
[out] HUntypedObjectX Regions )
\end{verbatim}

Extract regions with equal gray values from an image.

\texttt{LabelToRegion} segments an image into regions of equal gray value. One output region is generated for each gray value occurring in the image. This is similar to calling \texttt{Threshold} multiple times, and accumulating the results with \texttt{ConcatObj}. Another related operator is \texttt{Regiongrowing}. However, \texttt{LabelToRegion} does not perform a \texttt{Connection} operation on the resulting regions, i.e., they may be disconnected. A typical application of \texttt{LabelToRegion} is the segmentation of label images, hence its name.

The number of output regions is limited by the system parameter ‘\texttt{max} outp \texttt{obj} \texttt{par}’, which can be read via \texttt{GetSystem(‘:max} outp \texttt{obj} \texttt{par’:<Anzahl>}).

\textbf{Attention}

\texttt{LabelToRegion} is not implemented for images of type ‘real’. The input images must not contain negative gray values.

\textbf{Parameter}

\begin{itemize}
  \item \textbf{LabelImage} (input iconic) \hspace{1cm} \texttt{image(-array)} \Rightarrow \texttt{HImageX/IObjectX ( byte, int2, int4 )}
  Label image.
  \item \textbf{Regions} (output iconic) \hspace{1cm} \texttt{region \Rightarrow HRegionX/HUntypedObjectX}
  Regions having a constant gray value.
\end{itemize}

\textbf{Result}

\texttt{LabelToRegion} returns \texttt{TRUE} if the gray values lie within a correct range. The behavior with respect to the input images and output regions can be determined by setting the values of the flags ‘\texttt{no} object \texttt{result}’, ‘\texttt{empty} region \texttt{result}’, and ‘\texttt{store} empty \texttt{region}’ with \texttt{SetSystem}. If necessary, an exception is raised.

\textbf{Parallelization Information}

\texttt{LabelToRegion} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).
### Possible Predecessor Functions

MinMaxGray, SobelAmp, GaussImage, ReduceDomain, DiffOfGauss

### Possible Successor Functions

Connection, Dilation1, Erosion1, Opening, Closing, RankRegion, ShapeTrans, Skeleton

See Also

ConcatObj, RegionGrowing, RegionToLabel

### Module

Region processing

## 9.4 Features

```c
[out] VARIANT Area HRegionX.AreaCenter ([out] VARIANT Row,
[out] VARIANT Column )
```

### Area and center of regions.

The operator AreaCenter calculates the area and the center of the input regions. The area is defined as the number of pixels of a region. The center is calculated as the mean value of the line or column coordinates, respectively, of all pixels.

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of the input region. In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

#### Parameter

- **Regions** (input iconic) region(-array) → HRegionX / IObjectX Region(s) to be examined.
- **Area** (output control) integer(-array) → VARIANT (integer) Area of the region.
- **Row** (output control) point.y(-array) → VARIANT (real) Line index of the center.
- **Column** (output control) point.x(-array) → VARIANT (real) Column index of the center.

#### Result

The operator AreaCenter returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

#### Parallelization Information

AreaCenter is reentrant and automatically parallelized (on tuple level).

### Possible Predecessor Functions

### See Also

SelectShape

### Module

Region processing
Shape factor for the circularity (similarity to a circle) of a region.

The operator \texttt{Circularity} calculates the similarity of the input region with a circle.

\textbf{Calculation:} If $F$ is the area of the region and $\text{max}$ is the maximum distance from the center to all contour pixels, the shape factor $C$ is defined as:

\[
C = \frac{F}{(\text{max}^2 / \pi)}
\]

The shape factor $C$ of a circle is 1. If the region is long or has holes $C$ is smaller than 1. The operator \texttt{Circularity} especially responds to large bulges, holes and unconnected regions.

In case of an empty region the operator \texttt{Circularity} returns the value 0 (if no other behavior was set (see \texttt{SetSystem})). If more than one region is passed the numerical values of the shape factor are stored in a tuple, the position of a value in the tuple corresponding to the position of the region in the input tuple.

\begin{verbatim}
/* Comparison between shape factors of rectangle, circle and ellipse: */
gen_rectangle1(R1,10,10,20,20)
gen_rectangle2(R2,100,100,0.0,100,20)
gen_ellipse(E100,100,0.0,100,20)
gen_circle(C,100,100,20)
circularity([R1,R2,E,C],M)
fwrite_string(FileId,[\{'quadrate: ',M[1]\}])
fnnew_line(FileId)
fwrite_string(FileId,[\{'rectangle: ',M[2]\}])
fnnew_line(FileId)
fwrite_string(FileId,[\{'ellipse: ',M[3]\}])
fnnew_line(FileId)
fwrite_string(FileId,[\{'circle: ',M[4]\}])
fnnew_line(FileId)
\end{verbatim}

\textbf{Result} The operator \texttt{Circularity} returns the value \texttt{TRUE} if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem(\texttt{'noObjectResult'},<Result>)}. The behavior in case of empty region (the region is the empty set) is set via \texttt{SetSystem(\texttt{'emptyRegionResult'},<Result>)}. If necessary an exception handling is raised.

\textbf{Parallelization Information} \texttt{Circularity} is \texttt{reentrant} and automatically \texttt{parallelized} (on tuple level).

\textbf{Possible Predecessor Functions} \texttt{Threshold}, \texttt{Region-growing}, \texttt{Connection}

\textbf{See Also} \texttt{AreaCenter}, \texttt{SelectShape}

\textbf{Alternatives} \texttt{Roundness}, \texttt{Compactness}, \texttt{Convexity}, \texttt{Eccentricity}

\textbf{Module} Region processing

Shape factor for the compactness of a region.

The operator Compactness calculates the compactness of the input regions.

**Calculation:** If \( L \) is the length of the contour (see `Contlength`) and \( F \) the area of the region the shape factor \( C \) is defined as:

\[
C = \frac{L^2}{4F\pi}
\]

The shape factor \( C \) of a circle is 1. If the region is long or has holes \( C \) is larger than 1. The operator Compactness responds to the course of the contour (roughness) and to holes. In case of an empty region the operator Compactness returns the value 0 if no other behavior was set (see `SetSystem`). If more than one region is passed the numerical values of the shape factor are stored in a tuple, the position of a value in the tuple corresponding to the position of the region in the input tuple.

---

**Parameter**

- **Regions** (input iconic) .............................. region(-array) \( \sim \) HRegionX / IObjectX Region(s) to be examined.
- **Compactness** (output control) ........................ real(-array) \( \sim \) VARIANT( real ) Compactness of the input region(s).

**Restriction:** \((\text{Compactness} \geq 1.0) \lor (\text{Compactness} = 0)\)

---

**Result**

The operator Compactness returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

---

**Parallelization Information**

Compactness is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

Threshold, RegionGrow, Connection

---

**See Also**

Contlength, AreaCenter, SelectShape

---

**Alternatives**

Compactness, Convexity, Eccentricity

---

**Module**

Region processing

---

Number of connection components and holes

The operator ConnectAndHoles calculates the number of connection components and the number of holes of each region of Regions.

If more than one region is passed the numerical values of the output control parameters NumConnected and NumHoles are each stored in a tuple, the position of a value in the tuple corresponding to the position of the region in the input tuple.
### ConnectAndHoles

The operator `ConnectAndHoles` returns the value `TRUE` if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via `SetSystem(‘emptyRegionResult’,<Result>.

**Parallelization Information**

`ConnectAndHoles` is reentrant and automatically parallelized (on tuple level).

### Possible Predecessor Functions

Threshold, RegionGrowing, Connection

### See Also

Connection, FillUp, FillUpShape, Union1

### Alternatives

EulerNumber

### Module

Region processing

---

**ContLength**

The contour length of a region is calculated as the total length of the contour (sum of all connection components of the region) for each region of `Regions`. The distance between two neighboring contour points parallel to the coordinate axes is rated 1, the distance in the diagonal is rated $\sqrt{2}$. If more than one region is passed the numerical values of the contour length are stored in a tuple, the position of a value in the tuple corresponding to the position of the region in the input tuple. In case of an empty region the operator `ContLength` returns the value 0.

**Attention**

The contour of holes is not calculated.

**Parameter**

- **Regions** (input iconic) ................................. region(-array) $\rightarrow$ `HRegionX / IObjectX
  Region(s) to be examined.

- **ContLength** (output control) ................................. real(-array) $\rightarrow$ `VARIANT (real
  Contour length of the input region(s).

**Restriction**

$\text{ContLength} \geq 0$

**Result**

The operator `ContLength` returns the value `TRUE` if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem(‘noObjectResult’,<Result>). If necessary an exception handling is raised.

**Parallelization Information**

`ContLength` is reentrant and automatically parallelized (on tuple level).

### Possible Predecessor Functions

Threshold, RegionGrowing, Connection

---
Shape factor for the convexity of a region.

The operator \texttt{Convexity} calculates the convexity of each input region of \texttt{Regions}.

\textbf{Calculation:} If $F_c$ is the area of the convex hull and $F_o$ the original area of the region the shape factor $C$ is defined as:

$$C = \frac{F_o}{F_c}$$

The shape factor $C$ is 1 if the region is convex (e.g. rectangle, circle etc.). If there are indentations or holes $C$ is smaller than 1.

In case of an empty region the operator \texttt{Convexity} returns the value 0 (if no other behavior was set (see SetSystem)). If more than one region is passed the numerical values of the contour length are stored in a tuple, the position of a value in the tuple corresponding to the position of the region in the input tuple.

\textbf{Attention}

\textbf{Parameter}

\begin{itemize}
  \item \texttt{Regions} (input iconic) : region(-array) \rightsquigarrow HRegionX / IHObjectX Region(s) to be examined.
  \item \texttt{Convexity} (output control) : real(-array) \rightsquigarrow VARIANT (real) Convexity of the input region(s).
\end{itemize}

\textbf{Restriction}: \texttt{(Convexity \leq 1)}

\textbf{Result}

The operator \texttt{Convexity} returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem('noObjectResult',<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem('emptyRegionResult',<Result>). If necessary an exception handling is raised.

\textbf{Parallelization Information}

\texttt{Convexity} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

\textbf{Possible Predecessor Functions}

Threshold, RegionGrowing, Connection

\textbf{See Also}

SelectShape, AreaCenter, ShapeTrans

\textbf{Module}

Region processing
Maximal distance between two boundary points of a region.

The operator DiameterRegion calculates the maximal distance between two boundary points of a region. The coordinates of these two extremes and the distance between them will be returned.

Attention

If the region is empty, the results of Row1, Column1, Row2 and Column2 (all of them = 0) may lead to confusion.

Parameter

- **Regions** (input iconic) .................. region(-array) ~ HRegionX / IHObjectX Regions to be examined.
- **Row1** (output control) .................. line.begin.y(-array) ~ VARIANT (integer) Row index of the first extreme point.
- **Column1** (output control) ............. line.begin.x(-array) ~ VARIANT (integer) Column index of the first extreme point.
- **Row2** (output control) .................. line.end.y(-array) ~ VARIANT (integer) Row index of the second extreme point.
- **Column2** (output control) .......... line.end.x(-array) ~ VARIANT (integer) Column index of the second extreme point.
- **Diameter** (output control) .......... number(-array) ~ VARIANT (real) Distance of the two extreme points.

Result

The operator DiameterRegion returns the value TRUE, if the input is not empty. The reaction to empty input (no input regions are available) may be determined with the help of the operator SetSystem(‘noObjectResult’,<Result>). The reaction concerning an empty region (region is the empty set) will be determined by the operator SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

DiameterRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Connection, RunlengthFeatures

Possible Successor Functions

DispLine

Alternatives

SmallestRectangle2

Module

Region processing

Shape features derived from the ellipse parameters.

The operator Eccentricity calculates three shape features derived from the geometric moments.
**Definition:** If the parameters $I_a$, $I_b$ and the area $A$ of the region are given (see MomentsRegion2nd, EllipticAxis), the following applies:

\[
\begin{align*}
\text{Anisometry} &= \frac{I_a}{I_b} \\
\text{Bulkiness} &= 4 \cdot \pi \cdot I_a \cdot I_b \\
\text{StructureFactor} &= \text{Anisometry} \cdot \text{Bulkiness} - 1
\end{align*}
\]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

---

**Attention**

Parameter

- **Regions** (input iconic) ...................................... region(-array) \sim HRegionX / IHObjectX Region(s) to be examined.
- **Anisometry** (output control) .............................. real(-array) \sim VARIANT( real )
  - Shape feature (in case of a circle = 1.0).
  - **Restriction:** ($\text{Anisometry} \geq 1.0$)
- **Bulkiness** (output control) ............................... real(-array) \sim VARIANT( real )
  - Calculated shape feature.
- **StructureFactor** (output control) ...................... real(-array) \sim VARIANT( real )
  - Calculated shape feature.

---

**Result**

The operator Eccentricity returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

---

**Parallelization Information**

Eccentricity is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

Threshold, Regiongrowing, Connection

---

**See Also**

EllipticAxis, MomentsRegion2Nd, SelectShape, AreaCenter

---

**Module**

Region processing

```halcon
[out] VARIANT Ra HRegionX.EllipticAxis ([out] VARIANT Rb,
[out] VARIANT Phi )
```

```halcon
void HOperatorSetX.EllipticAxis ([in] IHObjectX Regions,
[out] VARIANT Ra, [out] VARIANT Rb, [out] VARIANT Phi )
```

**Parameters of the equivalent ellipse.**

The operator EllipticAxis calculates the radii and the orientation of the ellipse having the “same orientation” and the “same side relation” as the input region. Several input regions can be passed in Regions as tuples. The length of the main radius $R_a$ and the secondary radius $R_b$ as well as the orientation of the main axis with regard to the horizontal ($\Phi$) are determined. The angle is indicated in arc measure.

**Calculation:**

If the moments $M_{20}$, $M_{02}$ and $M_{11}$ are normalized and passed to the area (see MomentsRegion2nd), the radii $R_a$ and $R_b$ are calculated as:

\[
R_a = \sqrt{\frac{8(M_{20} + M_{02} + \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}{2}}
\]
The orientation $\Phi$ is defined by:

$$\Phi = -0.5 \tan \left(2\left(M_{11}, M_{02} - M_{20}\right)\right)$$

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see `SetSystem('noObjectResult',<Result>).`)

---

### Example

```plaintext
read_image(Image, 'fabrik')
open_window(0, 0, -1, -1, 'root', 'visible', '', WindowHandle)
regiongrowing(Image, Seg, 5, 5, 6, 100)
elliptic_axis(Seg, Ra, Rb, Phi)
area_center(Seg, _, Row, Column)
gen_ellipse(Ellipses, Row, Column, Phi, Ra, Rb)
set_draw(WindowHandle, 'margin')
disp_region(Ellipses, WindowHandle)
```

---

### Result

The operator `EllipticAxis` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>).`. The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>).`. If necessary an exception handling is raised.

---

### Parallelization Information

`EllipticAxis` is **reentrant** and automatically **parallelized** (on tuple level).

---

### Possible Predecessor Functions

`Threshold`, `Regiongrowing`, `Connection`

---

### Possible Successor Functions

`GenEllipse`

---

### See Also

`MomentsRegion2Nd`, `SelectShape`, `SetShape`

---

### Alternatives

`SmallestRectangle2`, `OrientationRegion`

---

### Bibliography

R. Haralick, L. Shapiro “Computer and Robot Vision” Addison-Wesley, 1992, pp. 73-75
Calculate the Euler number.

The procedure EulerNumber calculates the Euler number, i.e. the difference between the number of connection components and the number of holes.

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

**Parameter**

- **Regions** (input iconic) ................. region(-array) ~ HRegionX / IHObjectX
  Region(s) to be examined.
- **EulerNumber** (output control) ........... integer(-array) ~ VARIANT(integer)
  Calculated Euler number.

**Result**

The operator EulerNumber returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem('noObjectResult','<Result>'). The behavior in case of empty region (the region is the empty set) is set via SetSystem('emptyRegionResult','<Result>'). If necessary an exception handling is raised.

**Parallelization Information**

EulerNumber is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection

**Alternatives**

ConnectAndHoles

**Module**

Region processing

---

Search direct neighbors.

The operator FindNeighbors determines neighboring regions with Regions1 and Regions2 containing the regions to be examined. Regions1 can have three different states:

- **Regions1 is empty**: In this case all regions in Regions2 are permutatively checked for neighborhood.
- **Regions1 consists of one region**: The regions of Regions1 are compared to all regions in Regions2.
- **Regions1 consists of the same number of regions as Regions2**: Here all regions at the n-th position in Regions1 and Regions2 are checked for the neighboring relation.

The operator FindNeighbors uses the city block distance between neighboring regions. It can be specified by the parameter MaxDistance. neighboring regions are located at the n-th position in RegionIndex1 and RegionIndex2, i.e. the region with index RegionIndex1[n] from Regions1 is the neighbor of the region with index RegionIndex2[n] from Regions2.

**Attention**

Covered regions are not found!
Chapter 9. Regions

Parameter

- **Regions1** (input iconic) ........................................... region(-array) \(\leadsto\) HRegionX / IHObjectX
  
  Starting regions.

- **Regions2** (input iconic) ........................................... region(-array) \(\leadsto\) HRegionX / IHObjectX
  
  Comparative regions.

- **MaxDistance** (input control) .............................................................. integer \(\leadsto\) long
  
  Maximal distance of regions.
  
  **Default Value:** 1
  
  **Value Suggestions:** MaxDistance \(\in\) \{1, 2, 3, 4, 5, 6, 7, 8, 10, 15, 20, 50\}
  
  **Typical Range of Values:** \(1 \leq \text{MaxDistance} \leq 1\)
  
  **Minimal Value Step:** 1
  
  **Recommended Value Step:** 1

- **RegionIndex1** (output control) .............................................. integer \(\leadsto\) VARIANT (integer)
  
  Indices of the found regions from Regions1.

- **RegionIndex2** (output control) .............................................. integer \(\leadsto\) VARIANT (integer)
  
  Indices of the found regions from Regions2.

Result

The operator FindNeighbors returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

FindNeighbors is *reentrant* and processed without parallelization.

Possible Predecessor Functions

Threshold, RegionGrowing, Connection

See Also

SpatialRelation, SelectRegionSpatial, ExpandRegion, Interjacent, Boundary

Module

Region processing

Index of all regions containing a given pixel.

The operator GetRegionIndex returns the index of all regions in Regions (value range 0 to n-1) containing the test pixel (Row,Column), i.e.:

\[
|\text{Regions}_{n} \cap \{\text{Row}, \text{Column}\}| \neq 0
\]

Attention

If the regions overlap more than one region might contain the pixel. In this case all these regions are returned. If no region contains the indicated pixel the empty tuple (= no region) is returned.

Parameter

- **Regions** (input iconic) ...................................................... region \(\leadsto\) HRegionX / IHObjectX
  
  Regions to be examined.

- **Row** (input control) ................................................................. point.y \(\leadsto\) long
  
  Line index of the test pixel.
  
  **Default Value:** 100
  
  (lin)
9.4. FEATURES

Column (input control) ........................................................ point.x \sim long
Column index of the test pixel.
Default Value: 100

Index (output control) ................................................ integer(-array) \sim VARIANT ( integer )
Index of the regions containing the test pixel.

The operator GetRegionIndex returns the value TRUE if the parameters are correct.
The behavior in case of empty input (no input regions available) is set via the operator
SetSystem('noObjectResult',<Result>). If necessary an exception handling is raised.

Parallelization Information
GetRegionIndex is reentrant and processed without parallelization.

Possible Predecessor Functions
Threshold, RegionGrowing, Connection
GetMButton, GetMPosition, TestRegionPoint
SelectRegionPoint

Module
Region processing

[out] VARIANT Thickness HRegionX.GetRegionThickness
(void) VARIANT Histogramm )

void HOperatorSetX.GetRegionThickness ([in] IObjectX Region,
[out] VARIANT Thickness, [out] VARIANT Histogramm )

Access the thickness of a region along the main axis.
The operator GetRegionThickness calculates the thickness of the regions along the main axis (see
EllipticAxis) for each pixel of the section. The thickness at one point on the main axis is defined as the
distance between the intersections of the contour with the plumb on the main axis in the respective point which
are the furthest apart. Additionally the operator GetRegionThickness returns the Histogramm of the thick-
esses of the region. The length of the histogram corresponds to the largest occurring thickness in the observed
region.

Attention
Only one region may be passed. If the region has several connection components, only the first one is investigated.
All other components are ignored.

Parameter

Region (input iconic) ...................................... region \sim HRegionX / IObjectX
Region to be analysed.

Thickness (output control) ................................ integer \sim VARIANT ( integer )
Thickness of the region along its main axis.

Histogramm (output control) ......................... integer \sim VARIANT ( integer )
Histogram of the thickness of the region along its main axis.

Result
The operator GetRegionThickness returns the value TRUE if exactly one region is passed.
The behavior in case of empty input (no input regions available) is set via the operator
SetSystem('noObjectResult',<Result>).

Parallelization Information
GetRegionThickness is reentrant and processed without parallelization.

Possible Predecessor Functions
SobelAmp, Threshold, Connection, SelectShape, SelectObj
Hamming distance between two regions.

The operator HammingDistance returns the hamming distance between two regions, i.e. the number of pixels of the regions which are different (\( \text{Distance} \)), i.e. the number of pixels contained in one region but not in the other:

\[
\text{Distance} = |\text{Regions}_1 \cap \text{Regions}_2| + |\text{Regions}_2 \cap \text{Regions}_1|
\]

The parameter \( \text{Similarity} \) describes the similarity between the two regions based on the hamming distance \( \text{Distance} \):

\[
\text{Similarity} = 1 - \frac{\text{Distance}}{|\text{Regions}_1| + |\text{Regions}_2|}
\]

If both regions are empty \( \text{Similarity} \) is set to 0. The regions with the same index from both input parameters are always compared.

Attention

In both input parameters the same number of regions must be passed.

Parameter

- \( \text{Regions}_1 \) (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . region(-array) \( \sim \) HRegionX / IHObjectX
  Regions to be examined.
- \( \text{Regions}_2 \) (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . region(-array) \( \sim \) HRegionX / IHObjectX
  Comparative regions.
- \( \text{Distance} \) (output control) . . . . . . . . . . . . . . . . . . . . . . . . . integer(-array) \( \sim \) VARIANT(integer)
  Hamming distance of two regions.
  Restriction: \( \text{Distance} \geq 0 \)
- \( \text{Similarity} \) (output control) . . . . . . . . . . . . . . . . . . . . . . . . . real(-array) \( \sim \) VARIANT(real)
  Similarity of two regions.
  Restriction: \( (0 \leq \text{Similarity} \land \text{Similarity} \leq 1) \)

Result

hamming_distance returns the value TRUE if the number of objects in both parameters is the same and is not 0. The behavior in case of empty input (no input objects available) is set via the operator \( \text{SetSystem('noObjectResult',<Result>)} \). The behavior in case of empty region (the region is the empty set) is set via \( \text{SetSystem('emptyRegionResult',<Result>)} \). If necessary an exception handling is raised.

Parallelization Information

HammingDistance is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Connection

See Also

HammingChangeRegion

Alternatives

Intersection, Complement, AreaCenter

Module

Region processing
Hamming distance between two regions using normalization.

The operator `HammingDistanceNorm` returns the hamming distance between two regions, i.e. the number of pixels of the regions which are different (Distance). Before calculating the different the region in `Regions1` are normalized according to the regions in `Regions2`. The result is the number of pixels contained in one region but not in the other:

\[
\text{Distance} = |\text{Norm}(\text{Regions1}) \cap \text{Regions2}| + |\text{Regions2} \cap \text{Norm}(\text{Regions1})|
\]

The parameter `Similarity` describes the similarity between the two regions based on the hamming distance `Distance`:

\[
\text{Similarity} = 1 - \frac{\text{Distance}}{|\text{Norm}(\text{Regions1})| + |\text{Regions2}|}
\]

The following types of normalization are available:

- **'center'**: The region is moved so that both regions have the same center of gravity.

If both regions are empty `Similarity` is set to 0. The regions with the same index from both input parameters are always compared.

---

**Attention**

In both input parameters the same number of regions must be passed.

**Parameter**

- **Regions1** (input iconic) ............................... region(-array) \( \sim \) `HRegionX / IObjectX`
  Regions to be examined.
- **Regions2** (input iconic) ............................... region(-array) \( \sim \) `HRegionX / IObjectX`
  Comparative regions.
- **Norm** (input control) ................................. string(-array) \( \sim \) VARIANT ( string )
  Type of normalization.
  Default Value: 'center'
  Value List: `Norm \in \{ \text{`center'} \}`
- **Distance** (output control) ............................ integer(-array) \( \sim \) VARIANT ( integer )
  Hamming distance of two regions.
  Restriction: (Distance \( \geq 0 \))
- **Similarity** (output control) ............................. real(-array) \( \sim \) VARIANT ( real )
  Similarity of two regions.
  Restriction: (0 \( \leq \) Similarity) \& (Similarity \( \leq 1 \))

**Result**

`HammingDistanceNorm` returns the value TRUE if the number of objects in both parameters is the same and is not 0. The behavior in case of empty input (no input objects available) is set via the operator `SetSystem('noObjectResult',<Result>)`. The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>)`. If necessary an exception handling is raised.

**Parallelization Information**

`HammingDistanceNorm` is reentrant and processed without parallelization.

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection

---

**See Also**

HammingChangeRegion

---

HALCON 6.0
Largest inner circle of a region.

The operator InnerCircle determines the largest inner circle of a region, i.e. the circle with the largest area of all circles that fit into the region. For this circle the center (Row,Column) and the radius (Radius) are calculated. The output of the procedure is chosen in such a way that it can be used as input for the HALCON procedures DispCircle and GenCircle.

If several regions are passed in Regions corresponding tuples are returned as output parameters. In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

Attention

If several inner circles are present at a region only one solution is returned.

Restriction: \( \text{Radius} \geq 0 \)

Example

```
read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
regiongrowing(Image,Seg,5,5,6,100)
select_shape(Seg,H,'area','and',100,2000)
inner_circle(H,Row,Column,Radius)
gen_circle(Circles,Row,Column,Radius:)
set_draw(WindowHandle,'margin')
disp_region(Circles,WindowHandle)
```

Result

The operator InnerCircle returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem('noObjectResult',<Result>), the behavior in case of empty region is set via SetSystem('emptyRegionResult',<Result>). If necessary an exception handling is raised.

Parallelization Information

InnerCircle is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Connection, RunLengthFeatures

Possible Successor Functions

GenCircle, DispCircle
9.4. FEATURES

See Also
SetShape, SelectShape, SmallestCircle

Alternatives
ErosionCircle

Module
Region processing

Geometric moments of regions.
The operator MomentsRegion2nd calculates the moments (M20, M02) and the product of inertia of the axes through the center parallel to the coordinate axes (M11). Furthermore the main axes of inertia (Ia, Ib) are calculated.

Calculation: Z₀ and S₀ are the coordinates of the center of a region R with the area F. Then the moments Mᵢⱼ are defined by:

\[ M_{i,j} = \sum_{(Z,S) \in R} (Z - Z₀)^i(S - S₀)^j \]

wherein Z and S run through all pixels of the region R.
Furthermore,

\[ h = \frac{M20 + M02}{2} \]

then Ia and Ib are defined by:

\[ Ia = h + \sqrt{h^2 - M20 \cdot M02 + M11^2} \]
\[ Ib = h - \sqrt{h^2 - M20 \cdot M02 + M11^2} \]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.
In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

Attention

Parameter

▷ Regions (input iconic) region(-array) \( \rightarrow \) HRegionX/ IObjectX
Regions to be examined.

▷ M11 (output control) real(-array) \( \rightarrow \) VARIANT (real)
Product of inertia of the axes through the center parallel to the coordinate axes.

▷ M20 (output control) real(-array) \( \rightarrow \) VARIANT (real)
Moment of 2nd order (line-dependent).

▷ M02 (output control) real(-array) \( \rightarrow \) VARIANT (real)
Moment of 2nd order (column-dependent).

▷ Ia (output control) real(-array) \( \rightarrow \) VARIANT (real)
The one main axis of inertia.

▷ Ib (output control) real(-array) \( \rightarrow \) VARIANT (real)
The other main axis of inertia.
CHAPTER 9. REGIONS

The operator \texttt{MomentsRegion2nd} returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem('noObjectResult',<Result>)}. The behavior in case of empty region (region is the empty set) is set via \texttt{SetSystem('emptyRegionResult',<Result>)}. If necessary an exception handling is raised.

**Parallelization Information**

\texttt{MomentsRegion2nd} is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

\texttt{Threshold}, \texttt{RegionGrowing}, \texttt{Connection}

**See Also**

\texttt{EllipticAxis}

**Alternatives**

\texttt{MomentsRegion2ndInvar}

**Module**

Region processing

\begin{verbatim}
[out] VARIANT M11 HRegionX.MomentsRegion2ndInvar ([out] VARIANT M20,
[out] VARIANT M02 )
void HOperatorSetX.MomentsRegion2ndInvar ([in] IHObjectX Regions,
[out] VARIANT M11, [out] VARIANT M20, [out] VARIANT M02 )
\end{verbatim}

Geometric moments of regions.

The operator \texttt{MomentsRegion2ndInvar} calculates the scaled moments (M20, M02) and the product of inertia of the axes through the center parallel to the coordinate axes (M11).

**Calculation:** \( Z_0 \) and \( S_0 \) are the coordinates of the center of a region \( R \) with the area \( F \). Then the moments \( M_{ij} \) are defined by:

\[
M_{ij} = \frac{1}{F^2} \sum_{(Z,S) \in R} (Z - Z_0)^i (S - S_0)^j
\]

wherein \( Z \) and \( S \) run through all pixels of the region \( R \).

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see \texttt{SetSystem}).

**Parameter**

\begin{itemize}
  \item **Regions** (input iconic) region(-array) \~ HRegionX / IHObjectX Regions to be examined.
  \item **M11** (output control) real(-array) \~ VARIANT( real ) Product of inertia of the axes through the center parallel to the coordinate axes.
  \item **M20** (output control) real(-array) \~ VARIANT( real ) Moment of 2nd order (line-dependent).
  \item **M02** (output control) real(-array) \~ VARIANT( real ) Moment of 2nd order (column-dependent).
\end{itemize}

**Result**

The operator \texttt{MomentsRegion2ndInvar} returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem('noObjectResult',<Result>)}. The behavior in case of empty region (the region is the empty set) is set via \texttt{SetSystem('emptyRegionResult',<Result>)}. If necessary an exception handling is raised.
Geometric moments of regions.

The operator `MomentsRegion2NdRelInvar` calculates the scaled relative moments (PHI1, PHI2).

**Calculation:** The moments PHI1 and PHI2 are defined by:

\[
\begin{align*}
PHI1 &= V_{20} + V_{02} \\
PHI2 &= (V_{20} + V_{02})^2 + V_{11}^2 
\end{align*}
\]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see `SetSystem`).

**Attention**

The operator `MomentsRegion2ndRelInvar` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>)`. The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>)`. If necessary an exception handling is raised.

**Parallelization Information**

`MomentsRegion2NdRelInvar` is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection

**See Also**

EllipticAxis

**Alternatives**

`MomentsRegion2Nd` Module

Region processing
Geometric moments of regions.

The operator \texttt{MomentsRegion3rd} calculates the translation-invariant central moments \((M_{21}, M_{12}, M_{03}, M_{30})\) of order \((p + q)\).

**Calculation:** \(x\) and \(y\) are the coordinates of the center of a region \(R\) with the area \(Z\). Then the moments \(M_{pq}\) are defined by:

\[
    M_{pq} = \sum_{i=1}^{n} M Z(x_i, y_i)(x_i - x)^p(y_i - y)^q
\]

wherein \(x = \frac{m_{10}}{m_{00}}\) and \(y = \frac{m_{01}}{m_{00}}\).

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see \texttt{SetSystem}).

----------
**Attention**

----------
**Parameter**

- \texttt{Regions} (input iconic) region(-array) \(\rightsquigarrow\) \texttt{HRegionX / IObjectX}

- \texttt{M21} (output control) real(-array) \(\rightsquigarrow\) \texttt{VARIANT ( real )}
  Moment of 3nd order (line-dependent).

- \texttt{M12} (output control) real(-array) \(\rightsquigarrow\) \texttt{VARIANT ( real )}
  Moment of 3nd order (column-dependent).

- \texttt{M03} (output control) real(-array) \(\rightsquigarrow\) \texttt{VARIANT ( real )}
  Moment of 3nd order (column-dependent).

- \texttt{M30} (output control) real(-array) \(\rightsquigarrow\) \texttt{VARIANT ( real )}
  Moment of 3nd order (line-dependent).

----------
**Result**

The operator \texttt{MomentsRegion3rd} returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem('noObjectResult',<Result>)}.

The behavior in case of empty region (the region is the empty set) is set via \texttt{SetSystem('emptyRegionResult',<Result>)}.

If necessary an exception handling is raised.

----------
**Parallelization Information**

\texttt{MomentsRegion3Rd} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

----------
**Possible Predecessor Functions**

\texttt{Threshold}, \texttt{Regiongrowing}, \texttt{Connection}

----------
**See Also**

\texttt{EllipticAxis}

----------
**Alternatives**

\texttt{MomentsRegion2Nd}

----------
**Module**

Region processing
Geometric moments of regions.
The operator MomentsRegion3rdInvar calculates the scale-invariant moments (M21, M12, M03, M30).

**Calculation:** Then the moments \( M_{pq} \) are defined by:

\[
M_{pq} = \frac{\mu_{pq}}{\mu^d}
\]

wobei \( p + q \geq 2 \) und \( \mu = \mu_{00} = m_{00} \) sind.

wherein are \( p + q \geq 2 \) and \( \mu = \mu_{00} = m_{00} \).

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to
the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

---

**Parameter**

- **Regions** (input iconic) region(-array) \( \rightsquigarrow \) HRegionX / IHObjectX Regions to be examined.
- **M21** (output control) real(-array) \( \rightsquigarrow \) VARIANT( real ) Moment of 3rd order (line-dependent).
- **M12** (output control) real(-array) \( \rightsquigarrow \) VARIANT( real ) Moment of 3rd order (column-dependent).
- **M03** (output control) real(-array) \( \rightsquigarrow \) VARIANT( real ) Moment of 3rd order (column-dependent).
- **M30** (output control) real(-array) \( \rightsquigarrow \) VARIANT( real ) Moment of 3rd order (line-dependent).

---

**Result**

The operator MomentsRegion3rdInvar returns the value TRUE if the input is not empty.
The behavior in case of empty input (no input regions available) is set via the operator
SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via
SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

---

**Parallelization Information**

MomentsRegion3rdInvar is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

Threshold, RegionGrowing, Connection

---

**See Also**

EllipticAxis

---

**Alternatives**

MomentsRegion2Nd

---

**Module**

Region processing
Calculation: Then the moments $I_i$ are defined by:

\[ I_1 = \mu_{20} \mu_{02} - \mu_{11}^2 \]
\[ I_2 = (\mu_{30} \mu_{03} - \mu_{21} \mu_{12})^2 - 4(\mu_{30} \mu_{12} - \mu_{21} \mu_{21})(\mu_{20} \mu_{02} - \mu_{11}^2) \]
\[ I_3 = \mu_{20} (\mu_{11}^2 - \mu_{12}^2) - \mu_{11} (\mu_{30} \mu_{03} - \mu_{21} \mu_{12}) + \mu_{02} (\mu_{30} \mu_{12} - \mu_{21} \mu_{21}) \]
\[ I_4 = \mu_{30} \mu_{03}^2 - 6 \mu_{30} \mu_{12} \mu_{21} \mu_{02} + 6 \mu_{30} \mu_{03} \mu_{12} \mu_{21} + (2 \mu_{11}^2 - \mu_{20} \mu_{02}) \]
\[ + \mu_{20} \mu_{30} + 6 \mu_{21} \mu_{12} \mu_{20} \mu_{02} - 8 \mu_{21} \mu_{12} \mu_{21} + 18 \mu_{21} \mu_{12} \mu_{20} \mu_{11} \mu_{02} \]
\[ + 6 \mu_{21} \mu_{03} \mu_{30} (2 \mu_{11}^2 - \mu_{20} \mu_{02}) + 9 \mu_{12} \mu_{30} \mu_{03} \mu_{12} \mu_{21} \mu_{02} - 6 \mu_{12} \mu_{30} \mu_{11} \mu_{20}^2 + \mu_{30} \mu_{12} \mu_{20}^2 \]
\[ + \mu_{30} \mu_{12} \mu_{20} \mu_{02} \]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

Geometric moments of regions.

The operator MomentsRegionCentral calculates the central moments ($I_1, I_2, I_3, I_4$).

\[ \text{out} \] VARIANT I1 HRegionX.MomentsRegionCentral ([out] VARIANT I2, [out] VARIANT I3, [out] VARIANT I4 )

\[ \text{void HOperatorSetX.MomentsRegionCentral ([in] IHObjectX Regions, [out] VARIANT I1, [out] VARIANT I2, [out] VARIANT I3, [out] VARIANT I4 )} \]
### Geometric moments of regions.

The operator `MomentsRegionCentralInvar` calculates the moments (\( \psi_1, \psi_2, \psi_3, \psi_4 \)) that are invariant under translation and general linear transformations.

**Calculation:**

Then the moments \( \psi_i \) are defined by:

\[
\begin{align*}
\psi_1 &= \frac{I_0}{\mu_0} \\
\psi_2 &= \frac{I_0}{\mu} \\
\psi_3 &= \frac{I_0}{\mu_1} \\
\psi_4 &= \frac{I_0}{\mu^{1.5}}
\end{align*}
\]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see `SetSystem`).

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attention</strong></td>
</tr>
<tr>
<td><strong>Regions</strong> (input iconic) region(-array) ( \sim ) HRegionX / IHObjectX Regions to be examined.</td>
</tr>
<tr>
<td><strong>PSI1</strong> (output control) real(-array) ( \sim ) VARIANT ( real ) Moment of 2nd order.</td>
</tr>
<tr>
<td><strong>PSI2</strong> (output control) real(-array) ( \sim ) VARIANT ( real ) Moment of 2nd order.</td>
</tr>
<tr>
<td><strong>PSI3</strong> (output control) real(-array) ( \sim ) VARIANT ( real ) Moment of 2nd order.</td>
</tr>
<tr>
<td><strong>PSI4</strong> (output control) real(-array) ( \sim ) VARIANT ( real ) Moment of 2nd order.</td>
</tr>
<tr>
<td><strong>Result</strong></td>
</tr>
</tbody>
</table>

The operator `MomentsRegionCentralInvar` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>)`. The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>)`. If necessary an exception handling is raised.

**Parallelization Information**

`MomentsRegionCentralInvar` is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

`Threshold, RegionGrowing, Connection`

**See Also**

`EllipticAxis`

**Alternatives**

`MomentsRegion2Nd`

**Module**

Region processing
Orientation of a region.

The operator OrientationRegion calculates the orientation of the region. The operator is based on EllipticAxis. In addition the point on the contour with maximal distance to the center of gravity is calculated. If the column coordinate of this point is less than the column coordinate of the center of gravity the value of $\pi$ is added to the angle.

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem(‘noObjectResult’,<Result>)).

Attention Parameter

Parameter

Regions (input iconic) region(-array) $\Rightarrow$ HRegionX / IHObjectX Region(s) to be examined.

Phi (output control) real(-array) $\Rightarrow$ VARIANT(real)
Orientation of region (arc measure).

Restriction: $\left((-(\pi)/2) < Phi \right) \land \left(Phi \leq ((3 \cdot \pi)/2)\right)$

Result

The operator OrientationRegion returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information
OrientationRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
Threshold, RegionGrowing, Connection

Possible Successor Functions
DispArrow

See Also
MomentsRegion2Nd, LineOrientation

Alternatives
EllipticAxis, SmallestRectangle2

Module
Region processing

Shape factors from contour.

The operator Roundness examines the distance between the contour and the center of the area. In particular the mean distance (Distance), the deviation from the mean distance (Sigma) and two shape features derived therefrom are determined. Roundness is the relation between mean value and standard deviation, and Sides indicates the number of polygon pieces if a regular polygon is concerned.
The contour for calculating the features is determined depending on the global neighborhood (see SetSystem).

Calculation:
If \( p \) is the center of the area, \( p_j \) the pixels and \( F \) the area of the contour.

\[
\text{Distance} = \frac{1}{F} \sum \| p - p_j \|
\]

\[
\Sigma^2 = \frac{1}{F} \sum (\| p - p_j \| - \text{Distance})^2
\]

\[
\text{Roundness} = 1 - \frac{\Sigma}{\text{Distance}}
\]

\[
\text{Sides} = 1.4111 \left( \frac{\text{Distance}}{\Sigma} \right)^{0.4724}
\]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

### Attention

- **Regions** (input iconic) .............................................. \( \text{region(array)} \) \( \rightarrow \) \( \text{HRegionX / IObjectX} \)
  - Region(s) to be examined.

- **Distance** (output control) .............................................. \( \text{real(array)} \) \( \rightarrow \) \( \text{VARIANT(real)} \)
  - Mean distance from the center.
  - **Restriction**: \( (\text{Distance} \geq 0.0) \)

- **Sigma** (output control) .............................................. \( \text{real(array)} \) \( \rightarrow \) \( \text{VARIANT(real)} \)
  - Standard deviation of Distance.
  - **Restriction**: \( (\Sigma > 0.0) \)

- **Roundness** (output control) .............................................. \( \text{real(array)} \) \( \rightarrow \) \( \text{VARIANT(real)} \)
  - Shape factor for roundness.
  - **Restriction**: \( (\text{Roundness} \leq 1.0) \)

- **Sides** (output control) .............................................. \( \text{real(array)} \) \( \rightarrow \) \( \text{VARIANT(real)} \)
  - Number of polygon sides.
  - **Restriction**: \( (\text{Sides} \geq 0) \)

The operator **Roundness** returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>), the behavior in case of empty region is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

### Parallelization Information

Roundness is **reentrant** and automatically **parallelized** (on tuple level).

### Possible Predecessor Functions

- Threshold, RegionGrowing, Connection

### See Also

- Contlength

### Alternatives

- Compactness

### Bibliography

Distribution of runs needed for runlength encoding of a region.

The operator `RunlengthDistribution` calculates the distribution of the runs of a region of the fore- and background. The frequency of the occurrence of a certain length is calculated. Runs of infinite length are not counted. Therefore the background are the holes of the region. As many values are passed as set by the maximum length of fore- or background, respectively. The length of both tuples usually differs. The first entry of the tuples is always 0 (no runs of the length 0). If there are no blanks the empty tuple is passed at `Background`. Analogously the empty tuple is passed in case of an empty region at `Foreground`.

**Parameter**

- **Region** (input iconic)  
  Region to be examined.
- **Foreground** (output control)  
  Length distribution of the region (foreground).
- **Background** (output control)  
  Length distribution of the background.

**Result**

The operator `RunlengthDistribution` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>)`. If more than one region is passed an exception handling is raised.

**Parallelization Information**

`RunlengthDistribution` is **reentrant** and processed **without** parallelization.

**Possible Predecessor Functions**

- `Threshold`, `SelectObj`

**See Also**

- `RunlengthFeatures`

**Alternatives**

- `RunlengthFeatures`

**Module**

Region processing

Characteristic values for runlength coding of regions.

The operator `RunlengthFeatures` calculates for every input region from `Regions` the number of runs necessary for storing this region with the aid of runlength coding. Furthermore the so-called "K-factor" is determined, which indicates by how much the number of runs differs from the ideal of the square in which this value is 1.0.

The K-factor (`KFactor`) is calculated according to the formula:

\[
KFactor = \frac{NumRuns}{\sqrt{Area}}
\]

wherein `Area` indicates the area of the region. It should be noted that the K-factor can be smaller than 1.0 (in case of long horizontal regions).
The L-factor (\(L_{\text{Factor}}\)) indicates the mean number of runs for each line index occurring in the region. MeanLength indicates the mean length of the runs. The parameter Bytes indicates how many bytes are necessary for coding the region with runlengths.

\[
\text{Attention}
\]

All features calculated by the operator RunlengthFeatures are not rotation invariant because the runlength coding depends on the direction. The operator RunlengthFeatures does not serve for calculating shape features but for controlling and analysing the efficiency of the runlength coding.

\[
\text{Parameter}
\]

- **Regions** (input iconic) region(-array) \(\sim\) HRegionX / IHObjectX
  
  Regions to be examined.

- **NumRuns** (output control) integer(-array) \(\sim\) VARIANT (integer)
  
  Number of runs.

  \[
  \text{Restriction:} \ 0 \leq \text{NumRuns}
  \]

- **KFactor** (output control) real(-array) \(\sim\) VARIANT (real)
  
  Storing factor in relation to a square.

  \[
  \text{Restriction:} \ 0 \leq \text{KFactor}
  \]

- **LFactor** (output control) real(-array) \(\sim\) VARIANT (real)
  
  Mean number of runs per line.

  \[
  \text{Restriction:} \ 0 \leq \text{LFactor}
  \]

- **MeanLength** (output control) real(-array) \(\sim\) VARIANT (real)
  
  Mean length of runs.

  \[
  \text{Restriction:} \ 0 \leq \text{MeanLength}
  \]

- **Bytes** (output control) integer(-array) \(\sim\) VARIANT (integer)
  
  Number of bytes necessary for coding the region.

  \[
  \text{Restriction:} \ 0 \leq \text{Bytes}
  \]

\[
\text{Result}
\]

The operator RunlengthFeatures returns the value TRUE if the input is not empty. If necessary an exception handling is raised.

\[
\text{Parallelization Information}
\]

RunlengthFeatures is \textit{reentrant} and automatically \textit{parallelized} (on \textit{tuple level}).

\[
\text{Possible Predecessor Functions}
\]

Threshold, RegionGrowing, Connection, RunlengthFeatures

\[
\text{See Also}
\]

RunlengthFeatures, RunlengthDistribution

\[
\text{Module}
\]

Region processing

\[
\begin{align*}
\text{[out]} \ & \ HRegionX \ DestRegions \ HRegionX.\text{SelectRegionPoint} \ (\text{[in]} \ \text{long Row, } \\
& \ \text{[in]} \ \text{long Column}) \\
\text{void} \ & \ HOperatorSetX.\text{SelectRegionPoint} \ (\text{[in]} \ \text{IHObjectX Regions, } \\
& \ \text{[out]} \ \text{HUntypedObjectX DestRegions, } \text{[in]} \ \text{long Row, } \text{[in]} \ \text{long Column})
\end{align*}
\]

\[
\text{Choose all regions containing a given pixel.}
\]

The operator SelectRegionPoint selects all regions from Regions containing the test pixel (Row,Column), i.e.:

\[
\left| \{ \text{Regions}[n] \cap \{(\text{Row}, \text{Column})\} \right| = 1
\]

\[
\text{Attention}
\]

If the regions overlap more than one region might contain the pixel. In this case all these regions are returned. If no region contains the indicated pixel the empty tuple (= no region) is returned.
Parameter

- **Regions** (input iconic) ........................................... region \( \sim HRegionX / IHObjectX \)
  Regions to be examined.

- **DestRegions** (output iconic) ............................... region \( \sim HRegionX / HUntypedObjectX \)
  All regions containing the test pixel.

- **Row** (input control) ............................................. point.y \( \sim \) long
  Line index of the test pixel.
  Default Value : 100 (lin)

- **Column** (input control) ......................................... point.x \( \sim \) long
  Column index of the test pixel.
  Default Value : 100 (lin)

Example

```plaintext
read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible','','WindowHandle)
disp_image(Image)
regiongrowing(Image,Seg,3,3,5,0)
set_color(WindowHandle,'red')
set_draw(WindowHandle,'margin')
Button := 1
while (Button = 1)
    fwrite_string(FileId,'Select the region with the mouse (End right button)')
    fnew_line(FileId)
    get_mbutton(WindowHandle,Row,Column,Button)
    select_region_point(Seg,Single,Row,Column)
    disp_region(Single,WindowHandle)
 endwhile
```

Result

The operator **SelectRegionPoint** returns the value TRUE if the parameters are correct. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>). If necessary an exception handling is raised.

Parallelization Information

**SelectRegionPoint** is **reentrant** and processed **without** parallelization.

Possible Predecessor Functions

**Threshold**, **RegionGrowing**, **Connection**

See Also

**GetMbutton**, **GetMposition**

Alternatives

**TestRegionPoint**

Module

**Region processing**

```plaintext
[out] VARIANT RegionIndex1 HRegionX.SelectRegionSpatial 
([in] HRegionX Regions2, [in] String Direction, [out] VARIANT RegionIndex2 )

void HOperatorSetX.SelectRegionSpatial ([in] IHObjectX Regions1, 
[in] IHObjectX Regions2, [in] String Direction, [out] VARIANT RegionIndex1, 
[out] VARIANT RegionIndex2 )
```

Pose relation of regions.
The operator `SelectRegionSpatial` chooses the regions from `Regions2` which are sufficient for the neighboring relation `Direction`. The regions to be examined have to be passed in `Regions1` or `Regions2`, respectively. `Regions1` can have three different states:

- **Regions1 is empty:**
  In this case all regions in `Regions2` are permutatively checked for neighborhood.
- **Regions1 consists of one region:**
  The regions of `Regions1` are compared to all regions in `Regions2`.
- **Regions1 consists of the same number of regions as `Regions2`:**
  The regions at the n-th position in `Regions1` and `Regions2` are each checked for a neighboring relation.

Possible values for `Direction` are:

- `'left'`: `Regions2` is left of `Regions1`
- `'right'`: `Regions2` is right of `Regions1`
- `'above'`: `Regions2` is above `Regions1`
- `'below'`: `Regions2` is below `Regions1`

The operator `SelectRegionSpatial` calculates the centers of the regions to be compared and decides according to the angle between the center straight lines and the x axis whether the direction relation is fulfilled. The relation is fulfilled within the area of -45 degree to +45 degree around the coordinate axes. Thus, the direction relation can be understood in such a way that the center of the second region must be located left (or right, above, below) of the center of the first region. The indices of the regions fulfilling the direction relation are located at the n-th position in `RegionIndex1` and `RegionIndex2`, i.e. the region with the index `RegionIndex2[n]` has the indicated relation with the region with the index `RegionIndex1[n]`. Access to regions via the index can be obtained via the operator `CopyObj`.

```

# Attention

Parameter

- **Regions1** (input iconic)  region(-array) \( \leadsto \) HRegionX / IHObjectX
  Starting regions
- **Regions2** (input iconic)  region(-array) \( \leadsto \) HRegionX / IHObjectX
  Comparative regions
- **Direction** (input control)  string \( \leadsto \) String
  Desired neighboring relation.
  Default Value: 'left'
  Value List: Direction \( \in \) {'left', 'right', 'above', 'below'}
- **RegionIndex1** (output control)  integer \( \leadsto \) VARIANT (integer)
  Indices in the input tuples (Regions1 or Regions2), respectively.
- **RegionIndex2** (output control)  integer \( \leadsto \) VARIANT (integer)
  Indices in the input tuples (Regions1 or Regions2), respectively.

Result

The operator `SelectRegionSpatial` returns the value TRUE if `Regions2` is not empty. The behavior in case of empty parameter `Regions2` (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>)`. The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>)`. If necessary an exception handling is raised.

Parallelization Information

- **Parallelization Information**
  SelectRegionSpatial is reentrant and processed without parallelization.

Possible Predecessor Functions

- **Possible Predecessor Functions**
  Threshold, RegionGrowing, Connection

See Also

- **See Also**
  SpatialRelation, FindNeighbors, CopyObj, ObjToInteger

Alternatives

- **Alternatives**
  AreaCenter, Intersection

Module

- **Module**
  Region processing

HALCON 6.0
```
Choose regions with the aid of shape features.

The operator SelectShape chooses regions according to shape. For each input region from Regions the indicated features (Features) are calculated. If each \(\text{Operation} = \text{'and'}\) or at least one \(\text{Operation} = \text{'or'}\) of the calculated features is within the default limits \(\text{Min}, \text{Max}\) the region is adapted into the output (duplicated). Condition: \(\text{Min} \leq \text{Feature}_i(\text{Object}) \leq \text{Max}\)

Possible values for Features:

- **'area'**: Area of the object
- **'row'**: Row index of the center
- **'column'**: Column index of the center
- **'width'**: Width of the region
- **'height'**: Height of the region
- **'row1'**: Row index of upper left corner
- **'column1'**: Column index of upper left corner
- **'row2'**: Row index of lower right corner
- **'column2'**: Column index of lower right corner
- **'circularity'**: Circularity (see Circularity)
- **'compactness'**: Compactness (see Compactness)
- **'contlength'**: Total length of contour (see operator Contlength)
- **'convexity'**: Convexity (see Convexity)
- **'ra'**: Main radius of the equivalent ellipse (see EllipticAxis)
- **'rb'**: Secondary radius of the equivalent ellipse (see EllipticAxis)
- **'phi'**: Orientation of the equivalent ellipse (see EllipticAxis)
- **'anisometry'**: Anisometry (see Eccentricity)
- **'bulkiness'**: Bulkiness (see operator Eccentricity)
- **'struct_factor'**: Structur Factor (see operator Eccentricity)
- **'outer_radius'**: Radius of smallest surrounding circle (see SmallestCircle)
- **'inner_radius'**: Radius of largest inner circle (see InnerCircle)
- **'dist_mean'**: Mean distance from the region border to the center (see operator Roundness)
- **'dist_deviation'**: Deviation of the distance from the region border from the center (see operator Roundness)
- **'roundness'**: Roundness (see operator Roundness)
- **'num_sides'**: Number of polygon sides (see operator Roundness)
- **'connect_num'**: Number of connection components (see operator ConnectAndHoles)
- **'holes_num'**: Number of holes (see operator ConnectAndHoles)
- **'max_diameter'**: Maximum diameter of the region (see operator DiameterRegion)
- **'orientation'**: Orientation of the region (see operator OrientationRegion)
- **'euler_number'**: Euler number (see operator EulerNumber)
- **'rect2_phi'**: Orientation of the smallest surrounding rectangle (see operator SmallestRectangle2)
- **'rect2_len1'**: Half the length of the smallest surrounding rectangle (see operator SmallestRectangle2)
- **'rect2_len2'**: Half the width of the smallest surrounding rectangle (see operator SmallestRectangle2)
- **'moments_m11'**: Geometric moments of the region (see operator MomentsRegion2nd)
Moments of the region (see operator MomentsRegion2nd)
Moments of the region (see operator MomentsRegion2ndInvar)
Moments of the region (see operator MomentsRegion2ndRelInvar)
Moments of the region (see operator MomentsRegion3rd)
Moments of the region (see operator MomentsRegion3rdInvar)
Moments of the region (see operator MomentsRegionCentral)
Moments of the region (see operator MomentsRegionCentralInvar)
Moments of the region (see operator MomentsRegionCentralInvar)
Moments of the region (see operator MomentsRegionCentralInvar)
Moments of the region (see operator MomentsRegionCentralInvar)

If only one feature (Features) is used the value of Operation is meaningless. Several features are processed in the sequence in which they are entered.

Attention Parameter

- **Regions** (input iconic) ... region ⇒ HRegionX / HObjectX Regions to be examined.
- **SelectedRegions** (output iconic) ... region ⇒ HRegionX / HUntypedObjectX Regions fulfilling the condition.
- **Features** (input control) ... string(-array) ⇒ VARIANT (string) Shape features to be checked.
  - **Default Value** : 'area'
  - **Value List** : Features ∈ {'area', 'row', 'column', 'width', 'height', 'row1', 'column1', 'row2', 'column2', 'circularity', 'compactness', 'contlength', 'convexity', 'ra', 'rb', 'phi', 'anisometry', 'bulkiness', 'struct_factor', 'outer_radius', 'inner_radius', 'max_diameter', 'dist_mean', 'dist_elevation', 'roundness', 'num_sides', 'orientation', 'connect_num', 'holes_num', 'euler_number', 'rect2_phi', 'rect2_len1', 'rect2_len2', 'moments_m11', 'moments_m20', 'moments_m02', 'moments_m11_invar', 'moments_m20_invar', 'moments_m02_invar', 'moments_psi1', 'moments_psi2', 'moments_psi3', 'moments_psi4'}
- **Operation** (input control) ... string ⇒ String Linkage type of the individual features.
  - **Default Value** : 'and'
  - **Value List** : Operation ∈ {'and', 'or'}

HALCON 6.0
Min (input control) real(-array) \rightsquigarrow \text{VARIANT ( integer, real, string )}
Lower limits of the features or 'min'.
Default Value : 150.0
Typical Range of Values : 0.0 \leq \text{Min} \leq 0.0
Minimal Value Step : 0.001
Recommended Value Step : 1.0

Max (input control) real(-array) \rightsquigarrow \text{VARIANT ( integer, real, string )}
Upper limits of the features or 'max'.
Default Value : 99999.0
Typical Range of Values : 0.0 \leq \text{Max} \leq 0.0
Minimal Value Step : 0.001
Recommended Value Step : 1.0
Restriction : (Max \geq \text{Min})

Example

/* where are the eyes of the ape ? */
read_image(Image’affe’)
threshold(Image,S1,160,255)
connection(S1,S2)
select_shape(S2,Eyes,['area','anisometry'],'and',[500,1.0],[50000,1.7])
disp_region(Eyes,WindowHandle)

Result

The operator SelectShape returns the value TRUE if the input is not empty. The behavior in case of empty input (no input objects available) is set via the operator SetSystem(’noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(’emptyRegionResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

SelectShape is \textit{reentrant} and processed \textit{without} parallelization.

Possible Predecessor Functions

Threshold, Regiongrowing, Connection, RunlengthFeatures

Possible Successor Functions

SelectShape, SelectGray, ShapeTrans, ReduceDomain, CountObj

See Also

AreaCenter, Circularity, Compactness, Contlength, Convexity, EllipticAxis, Eccentricity, InnerCircle, SmallestCircle, SmallestRectangle1, SmallestRectangle2, Roundness, ConnectAndHoles, DiameterRegion, OrientationRegion, MomentsRegion2Nd, MomentsRegion2NdInvar, MomentsRegion2NdRelInvar, MomentsRegion3Rd, MomentsRegion3RdInvar, MomentsRegionCentral, MomentsRegionCentralInvar, SelectObj

Region processing

Choose regions having a certain relation to each other.

The operator SelectShapeProto puts two regions in relation to each other. Every i-th region from Regions is compared to the combination of regions from Pattern. The limits (Min and Max) are indicated absolutely or in percent (0..100) depending on the feature. Possible values for Feature are:
'distance_dilate'  The minimum distance in the maximum norm from the edge of Pattern to the edge of every region from Regions is determined (see DistanceRrMinDil).

'distance_contour'  The minimum Euclidean distance from the edge of Pattern to the edge of every region from Regions is determined. (see DistanceRrMin).

'distance_center'  The Euclidean distance from the center of Pattern to the center of every region from Regions is determined.

covers'  It is examined how well the region Pattern fits into the regions from Regions. If there is no shift so that Pattern is a subset of Regions the overlap is 0. If Pattern corresponds to the region after a corresponding shift the overlap is 100. Otherwise the area of the opening of Regions with Pattern is put into relation with the area of Regions (in percent).

'fits'  It is examined whether Pattern can be shifted in such a way that it fits in Regions. If this is possible the corresponding region is copied from Regions. The parameters Min and Max have no meaning here.

Attention

Parameter

▷ Regions (input iconic)  region(-array)  ~  HRegionX / HObjectX
Regions to be examined.

▷ Pattern (input iconic)  region(-array)  ~  HRegionX / HObjectX
Region compared to Regions.

▷ SelectedRegions (output iconic)  region(-array)  ~  HRegionX / HUntypedObjectX
Regions fulfilling the condition.

▷ Feature (input control)  string(-array)  ~  VARIANT ( string )
Shape features to be checked.

Default Value:  'covers'

Value List:  Feature ∈ { 'distance_center', 'distance_dilate', 'distance_contour', 'covers', 'fits' }

▷ Min (input control)  number  ~  VARIANT ( integer, real )
Lower border of feature.

Default Value:  50.0

Value Suggestions:  Min ∈ { 0.0, 1.0, 5.0, 10.0, 20.0, 30.0, 50.0, 60.0, 70.0, 80.0, 90.0, 95.0, 99.0, 100.0, 200.0, 400.0 }

Minimal Value Step:  0.001
Recommended Value Step:  5.0

▷ Max (input control)  number  ~  VARIANT ( integer, real )
Upper border of the feature.

Default Value:  100.0

Value Suggestions:  Max ∈ { 0.0, 10.0, 20.0, 30.0, 50.0, 60.0, 70.0, 80.0, 90.0, 95.0, 99.0, 100.0, 200.0, 300.0, 400.0 }

Minimal Value Step:  0.001
Recommended Value Step:  5.0

Result

The operator SelectShapeProto returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem ('noObjectResult',<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem ('emptyRegionResult',<Result>). If necessary an exception handling is raised.

Parallelization Information

SelectShapeProto is reentrant and processed without parallelization.

Possible Predecessor Functions

Connection, DrawRegion, GenCircle, GenRectangle1, GenRectangle2, GenEllipse

Possible Successor Functions

SelectGray, ShapeTrans, ReduceDomain, CountObj

See Also

Opening, Erosion1, DistanceRrMinDil, DistanceRrMin
SelectShape

Region processing

```
[out]  HRegionX  SelectedRegions  HRegionX.SelectShapeStd
([in]  String  Shape,  [in]  double  Percent  )
```

```
void  HOperatorSetX.SelectShapeStd  ([in]  IHOBJECTX  Regions,
[out]  HUntypedObjectX  SelectedRegions,  [in]  String  Shape,
[in]  double  Percent  )
```

Select regions of a given shape.

The operator SelectShapeStd compares the shape of the given regions with default shapes. If the region has a similar shape it is adopted into the output. Possible values for `Shape` are:

- **'max_area'** The largest region is selected.
- **'rectangle1'** The surrounding rectangle parallel to the coordinate axes is determined via the operator SmallestRectangle1. If the area difference in percent is larger than `Percent` the region is adopted.
- **'rectangle2'** The smallest surrounding rectangle with any orientation is determined via the operator SmallestRectangle2. If the area difference in percent is larger than `Percent` the region is adopted.

### Parameter

- **Regions** (input iconic) ............................................. region(-array) ~ HRegionX / IHOBJECTX
  Input regions to be selected.
- **SelectedRegions** (output iconic) ............................ region(-array) ~ HRegionX / HUntypedObjectX
  Regions with desired shape.
- **Shape** (input control) .......................................... string ~ String
  Shape features to be checked.
  - **Default Value**: 'max_area'
  - **Value List**: Shape ∈ {'max_area', 'rectangle1', 'rectangle2'}
- **Percent** (input control) ........................................... real ~ double
  Similarity measure.
  - **Default Value**: 70.0
  - **Value Suggestions**: Percent ∈ {10.0, 30.0, 50.0, 60.0, 70.0, 80.0, 90.0, 95.0, 100.0}
  - **Typical Range of Values**: 0.0 ≤ Percent ≤ 0.0(lin)
  - **Minimal Value Step**: 0.1
  - **Recommended Value Step**: 10.0

### Parallelization Information

SelectShapeStd is **reentrant** and processed **without** parallelization.

### Possible Predecessor Functions

Threshold, RegionGrowing, Connection, SmallestRectangle1, SmallestRectangle2

### See Also

SmallestRectangle1, SmallestRectangle2

### Alternatives

Intersection, Complement, AreaCenter

Region processing
9.4. FEATURES

Smallest surrounding circle of a region.

The operator SmallestCircle determines the smallest surrounding circle of a region, i.e. the circle with the smallest area of all circles containing the region. For this circle the center (Row,Column) and the radius (Radius) are calculated. The procedure is applied when, for example, the location and size of circular objects (e.g. coins) which, however, are not homogeneous inside or have broken edges due to bad segmentation, has to be determined. The output of the procedure is selected in such a way that it can be used as input for the HALCON procedures DispCircle and GenCircle.

If several regions are passed in Regions corresponding tuples are returned as output parameter. In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

Attention

Parameter

- **Regions** (input iconic) region(-array) \( \sim \) HRegionX / IHObjectX

  - **Row** (output control) circle.center.y(-array) \( \sim \) VARIANT (real)

  - **Column** (output control) circle.center.x(-array) \( \sim \) VARIANT (real)

  - **Radius** (output control) circle.radius(-array) \( \sim \) VARIANT (real)

  **Restriction**: \( Radius \geq 0 \)

Example

```
read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible',''='',WindowHandle)
regiongrowing(Image,Seg,5,5,6,100:)
select_shape(Seg,H,'area','and',100,2000)
smallest_circle(H,Row,Column,Radius)
gen_circle(Circles,Row,Column,Radius)
set_draw(WindowHandle,'margin')
disp_region(Circles,WindowHandle)
```

Result

The operator SmallestCircle returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

SmallestCircle is **reentrant** and **automatically parallelized (on tuple level)**.

Possible Predecessor Functions

Threshold, RegionGrowing, Connection, RunLengthFeatures

Possible Successor Functions

GenCircle, DispCircle

See Also

SetShape, SelectShape, InnerCircle

Alternatives

EllipticAxis, SmallestRectangle1, SmallestRectangle2

HALCON 6.0
Surrounding rectangle parallel to the coordinate axes.

The operator SmallestRectangle1 calculates the surrounding rectangle of all input regions (parallel to the coordinate axes). The surrounding rectangle is described by the coordinates of the corner pixels (Row1, Column1, Row2, Column2)

If more than one region is passed in Regions, the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input. In case of empty region all parameters have the value 0 if no other behavior was set (see SetSystem).

Attention
In case of empty region the result of Row1, Column1, Row2 and Column2 (all are 0) can lead to confusion.

Parameter

- **Regions** (input iconic) ........................................ region(-array) \(\sim\) HRegionX / IObjectX
  Regions to be examined.
- **Row1** (output control) ...................................\(\sim\) rectangle.origin.y(-array) \(\sim\) VARIANT (integer)
  Line index of upper left corner point.
- **Column1** (output control) ...................................\(\sim\) rectangle.origin.x(-array) \(\sim\) VARIANT (integer)
  Column index of upper left corner point.
- **Row2** (output control) ...................................\(\sim\) rectangle.corner.y(-array) \(\sim\) VARIANT (integer)
  Line index of lower right corner point.
- **Column2** (output control) ...................................\(\sim\) rectangle.corner.x(-array) \(\sim\) VARIANT (integer)
  Column index of lower right corner point.

Result

The operator SmallestRectangle1 returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

SmallestRectangle1 is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, RegionGrowing, Connection, RunlengthFeatures

Possible Successor Functions

DispRectangle1, GenRectangle1

See Also

SelectShape

Alternatives

SmallestRectangle2, AreaCenter

Module

Region processing
9.4. FEATURES

The operator **SmallestRectangle2** determines the smallest surrounding rectangle of a region, i.e., the rectangle with the smallest area of all rectangles containing the region. For this rectangle the center, the inclination and the two radii are calculated.

The procedure is applied when, for example, the location of a scenery of several regions (e.g., printed text on a rectangular paper or in rectangular print (justified lines)) must be found. The parameters of **SmallestRectangle2** are chosen in such a way that they can be used directly as input for the HALCON-procedures **DispRectangle2** and **GenRectangle2**.

If more than one region is passed in **Regions** the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input. In case of empty region all parameters have the value 0.0 if no other behavior was set (see **SetSystem**).

---

**Attention**

**Parameter**

- **Regions** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . region(-array) ∼ **HRegionX / IObjectX** Regions to be examined.
- **Row** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . rectangle2.center.y(-array) ∼ VARIANT (real) Line index of the center.
- **Column** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . rectangle2.center.x(-array) ∼ VARIANT (real) Column index of the center.
- **Phi** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . rectangle2.angle.rad(-array) ∼ VARIANT (real) Orientation of the surrounding rectangle (arc measure)
  **Restriction**: \((-(\pi/2) < Phi) \land (Phi \leq (\pi/2))\)
- **Length1** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . rectangle2.hwidth(-array) ∼ VARIANT (real) First radius (half length) of the surrounding rectangle.
  **Restriction**: \((Length1 \geq 0.0)\)
- **Length2** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . rectangle2.hheight(-array) ∼ VARIANT (real) Second radius (half width) of the surrounding rectangle.
  **Restriction**: \((Length2 \geq 0.0) \land (Length2 \leq Length1)\)

---

**Example**

```plaintext
read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible',WindowHandle)
regiongrowing(Image,Seg,5,5,6,100)
smallest_rectangle2(Seg,Row,Column,Phi,Length1,Length2)
gen_rectangle2(Rectangle,Row,Column,Phi,Length1,Length2)
set_draw(WindowHandle,'margin')
disp_region(Rectangle,WindowHandle)
```

---

**Result**

The operator **SmallestRectangle2** returns the value **TRUE** if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator **SetSystem** (‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via **SetSystem** (‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

---

**Parallelization Information**

**SmallestRectangle2** is **reentrant** and automatically **parallelized** (on tuple level).

HALCON 6.0
Possible Predecessor Functions

Threshold, RegionGrowing, Connection, RunLengthFeatures

Possible Successor Functions

DispRectangle2, GenRectangle2

See Also

SmallestCircle, SetShape

Alternatives

EllipticAxis, SmallestRectangle1

Module

Region processing

```plaintext
[out] VARIANT RegionIndex1
HRegionX.SpatialRelation
([in] HRegionX Regions2, [in] long Percent, [out] VARIANT RegionIndex2,
[out] VARIANT Relation1, [out] VARIANT Relation2 )

void HOperatorSetX.SpatialRelation ([in] IHObjectX Regions1,
[in] IHObjectX Regions2, [in] long Percent, [out] VARIANT RegionIndex1,
[out] VARIANT RegionIndex2, [out] VARIANT Relation1,
[out] VARIANT Relation2 )
```

Pose relation of regions with regard to the coordinate axes.

The operator SpatialRelation selects regions located by Percent percent “left”, “right”, “above” or “below” other regions. Regions1 and Regions2 contain the regions to be compared. Regions1 can have three states:

- Regions1 is empty:
  In this case all regions in Regions2 are permutatively checked for neighborhood.
- Regions1 consists of one region:
  The regions of Regions1 are compared to all regions in Regions2.
- Regions1 consists of the same number of regions as Regions2:
  Regions1 and Regions2 are checked for a neighboring relation.

The percentage Percent is interpreted in such a way that the area of the second region has to be located really left/right or above/below the region margins of the first region by at least Percent percent. The indices of the regions that fulfill at least one of these conditions are then located at the n-th position in the output parameters RegionIndex1 and RegionIndex2. Additionally the output parameters Relation1 and Relation2 contain at the n-th position the type of relation of the region pair (RegionIndex1[n], RegionIndex2[n]), i.e. region with index RegionIndex2[n] has the relation Relation1[n] and Relation2[n] with region with index RegionIndex1[n].

Possible values for Relation1 and Relation2 are:

**Relation1:** 'left', 'right' or ''

**Relation2:** 'above', 'below' or ''

In RegionIndex1 and RegionIndex2 the indices of the regions in the tuples of the input regions (Regions1 or Regions2), respectively, are entered as image identifiers. Access to chosen regions via the index can be obtained by the operator CopyObj.

**Attention**

- **Regions1** (input iconic) ........................................... region(-array) \(\rightarrow\) HRegionX / IHObjectX
  Starting regions.
- **Regions2** (input iconic) ........................................... region(-array) \(\rightarrow\) HRegionX / IHObjectX
  Comparative regions.

**Feature 9.4:**

### Percent (input control)

- **Type:** integer
- **Range:** long
- **Description:** Percentage of the area of the comparative region which must be located left/right or above/below the region margins of the starting region.

**Default Value:** 50

**Value Suggestions:** Percent ∈ \{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}

**Typical Range of Values:** 0 ≤ Percent ≤ 0\(\text{lin}\)

**Minimal Value Step:** 1

**Recommended Value Step:** 10

**Restriction:** \((0 \leq \text{Percent}) \land (\text{Percent} \leq 100)\)

### RegionIndex1 (output control)

- **Type:** integer
- **Variant:** VARIANT (integer)
- **Description:** Indices of the regions in the tuple of the input regions which fulfill the pose relation.

### RegionIndex2 (output control)

- **Type:** integer
- **Variant:** VARIANT (integer)
- **Description:** Indices of the regions in the tuple of the input regions which fulfill the pose relation.

### Relation1 (output control)

- **Type:** string
- **Variant:** VARIANT (string)
- **Description:** Horizontal pose relation in which RegionIndex2\[n\] stands with RegionIndex1\[n\].

### Relation2 (output control)

- **Type:** string
- **Variant:** VARIANT (string)
- **Description:** Vertical pose relation in which RegionIndex2\[n\] stands with RegionIndex1\[n\].

---

**Result**

The operator **SpatialRelation** returns the value TRUE if Regions2 is not empty and Percent is correctly chosen. The behavior in case of empty parameter Regions2 (no input regions available) is set via the operator **SetSystem(‘noObjectResult’,<Result>).** The behavior in case of empty region (the region is the empty set) is set via **SetSystem(‘emptyRegionResult’,<Result>).** If necessary an exception handling is raised.

---

**Parallelization Information**

**SpatialRelation** is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

**See Also**

Threshold, Regiongrowing, Connection, SelectRegionSpatial, FindNeighbors, CopyObj, ObjToInteger

---

**Alternatives**

AreaCenter, Intersection

---

**Module**

Region processing

---

```c
void HRegionX.TestRegionPoint ( [in] long Row, [in] long Column )
```

**Test if the region consists of the given point.**

**TestRegionPoint** tests if at least one input region of Regions consists of the test point (Row,Column).

---

**Attention**

In case of empty input (= no region) and SetSystem(‘noObjectResult’,’true’) FALSE is returned (no region contains the pixel).

The test pixel is not contained in an empty region (no pixel of the region corresponds to the pixel). If all regions are empty FALSE is also returned, i.e. an empty region behaves as if it did not exist.

---

**Parameter**

- **Regions** (input iconic)
- **Type:** region(-array)
- **Variant:** HRegionX / IHObjectX

- **Row** (input control)
- **Type:** point.y
- **Range:** long

- **Default Value:** 100

---

**Module**

Region processing

---

**HALCON 6.0**
Column (input control)........................................................point.x  \sim long
Column index of the test pixel.

Default Value: 100
Typical Range of Values: 0 \leq Column \leq 0(lin)
Minimal Value Step: 1
Recommended Value Step: 1

\begin{verbatim}
Result
\end{verbatim}

The operator TestRegionPoint returns the value TRUE if a region contains the test pixel. If this is not the case TestRegionPoint returns FALSE. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(’noObjectResult’,<Result>). If necessary an exception handling is raised.

\begin{verbatim}
Parallelization Information
\end{verbatim}

TestRegionPoint is reentrant and processed without parallelization.

\begin{verbatim}
Possible Predecessor Functions
\end{verbatim}

Threshold, RegionGrowing, Connection

\begin{verbatim}
See Also
\end{verbatim}

SelectRegionPoint

\begin{verbatim}
Alternatives
\end{verbatim}

Union1, Intersection, AreaCenter

\begin{verbatim}
Module
\end{verbatim}

Region processing

\section*{9.5 Sets}

\begin{verbatim}
[out] HRegionX RegionComplement HRegionX.Complement ( )
void HOperatorSetX.Complement ([in] IObjectX Region,
[out] HUntypedObjectX RegionComplement )
\end{verbatim}

Return the complement of a region.

Complement determines the complement of the input region(s).

If the system flag ‘clip_region’ is ‘false’ (see SetSystem) the complement is done virtually by setting the complement flag of Region to TRUE. For succeeding operations the de Morgan laws are applied while calculating results.

If the system flag ‘clip_region’ is ‘true’ the difference of the largest image processed so far (see ResetObjDb) and the input region is returned.

\begin{verbatim}
Parameter
\end{verbatim}

\begin{verbatim}
\end{verbatim}

\begin{verbatim}
Region (input iconic) ..............................................region(-array) \leadsto HRegionX / IObjectX
Input region(s).
\end{verbatim}

\begin{verbatim}
RegionComplement (output iconic) ................................region(-array) \leadsto HRegionX / HUntypedObjectX
Complemented regions.
\end{verbatim}

Parameter Number: (RegionComplement = Region)

\begin{verbatim}
Result
\end{verbatim}

Complement always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem(’noObjectResult’,<Result>) and the behavior in case of an empty input region via SetSystem(’emptyRegionResult’,<Result>). If necessary, an exception handling is raised.

\begin{verbatim}
Parallelization Information
\end{verbatim}

Complement is reentrant and automatically parallelized (on tuple level).

\begin{verbatim}
Possible Predecessor Functions
\end{verbatim}

Threshold, Connection, RegionGrowing, Pouring, ClassNdimNorm

\begin{verbatim}
Possible Successor Functions
\end{verbatim}

SelectShape
Difference calculates the set-theoretic difference of two regions:

\[(\text{Regions in Region}) - (\text{Regions in Sub})\]

The resulting region is defined as the input region (Region) with all points from Sub removed. The following two special regions can also be used as input:

- 'full' i.e., the full region (FULL_REGION in HALCON/C) or
- 'empty' i.e., the empty region (EMPTY_REGION in HALCON/C).

Attention

Empty regions are valid for both parameters. On output, empty regions may result. The value of the system flag 'store_empty_region' determines the behavior in this case.

Parameter

- **Region** (input iconic) \(\rightarrow \text{HRegionX / IObjectX}\)
  - Regions to be processed.
- **Sub** (input iconic) \(\rightarrow \text{HRegionX / IObjectX}\)
  - The union of these regions is subtracted from Region.
- **RegionDifference** (output iconic) \(\rightarrow \text{HRegionX / HUntypedObjectX}\)
  - Resulting region.

Example

```c
/* provides the region X without the points in Y */
difference(X,Y,RegionDifference)
/* provides the region. */
difference(Region,[],RegionDifference)
```

Result

Difference always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

Difference is reentrant and processed without parallelization.

Possible Predecessor Functions

Threshold, Connection, RegionGrowing, Pouring, ClassNdimNorm

Possible Successor Functions

SelectShape, DispRegion

See Also

Intersection, Union1, Union2, Complement

Module

Region processing
Calculate the intersection of two regions.

Intersection calculates the intersection of the regions in Region1 with the regions in Region2. Each region in Region1 is intersected with all regions in Region2. The order of regions in RegionIntersection is identical to the order of regions in Region1.

Empty input regions are permitted. Because empty result regions are possible, the system flag 'store_empty_region' should be set appropriately.

Return the union of all input regions.

Union1 computes the union of all input regions and returns the result in RegionUnion.
threshold(Image,Seg1,128,255)
dyn_threshold(Image,Mean,Seg2,5,’light’)
union1([Seg1,Seg2],Seg).

Result

Union1 always returns TRUE. The behavior in case of empty input (no regions given) can be set via
SetSystem(’noObjectResult’,<Result>) and the behavior in case of an empty input region via
SetSystem(’emptyRegionResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

Union1 is reentrant and processed without parallelization.

Possible Predecessor Functions
Threshold, Connection, Regiongrowing, Pouring

Possible Successor Functions
SelectShape, DispRegion

See Also
Intersection, Complement

Alternatives

Region processing

Union2

Module

HRegionX RegionUnion HRegionX.Union2 ([in] HRegionX Region1, [in] HRegionX Region2, [out] HUntypedObjectX RegionUnion)

Return the union of two regions.

Union2 computes the union of the region in Region1 with all regions in Region2. This means that Union2
is not commutative!

Parameter

▷ Region1 (input iconic) region(-array) HRegionX / IObjectX
Region for which the union with all regions in Region2 is to be computed.
▷ Region2 (input iconic) region(-array) HRegionX / IObjectX
Regions which should be added to Region1.
▷ RegionUnion (output iconic) region(-array) HRegionX / HUntypedObjectX
Resulting regions.

Parameter Number: (RegionUnion = Region1)

Result

Union2 always returns TRUE. The behavior in case of empty input (no regions given) can be set via
SetSystem(’noObjectResult’,<Result>) and the behavior in case of an empty input region via
SetSystem(’emptyRegionResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

Union2 is reentrant and processed without parallelization.

Possible Predecessor Functions
Threshold, Connection, Regiongrowing, Pouring

Possible Successor Functions
SelectShape, DispRegion

See Also
Intersection, Complement

Alternatives

Region processing

HALCON 6.0
9.6 Transformation

```c
[out] HRegionX BackgroundRegions HRegionX.BackgroundSeg ( )
void HOperatorSetX.BackgroundSeg ([in] IHObjectX Foreground,
    [out] HUntypedObjectX BackgroundRegions )
```

Determine the connected components of the background of given regions.

BackgroundSeg determines connected components of the background of the foreground regions given in Foreground. This operator is normally used after an edge operator in order to determine the regions enclosed by the extracted edges. The connected components are determined using 4-connectivity.

Parameter

- **Foreground** (input iconic) .......... region(-array) ~ HRegionX / IHObjectX
  - Input regions.

- **BackgroundRegions** (output iconic) .......... region ~ HRegionX / HUntypedObjectX
  - Connected components of the background.

Example

```c
/* Simulation of background_seg: */
background_seg(Foreground,BackgroundRegions):
    complement(Foreground,Background)
    get_system(‘neighborhood’, Save)
    set_system(‘neighborhood’, 4)
    connection(Background,BackgroundRegions)
    clear_obj(Background)
    set_system(‘neighborhood’, Save).

/* Segmentation with edge filter: */
read_image(Image,’fabrik’)
sobel_dir(Image,Sobel,Dir,’sum_sqrt’,3)
threshold(Sobel,Edges,20,255)
skeleton(Edges,Margins)
background_seg(Margins,Regions).
```

Result

BackgroundSeg always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem(‘noObjectResult’,<Result>) and the behavior in case of an empty input region via SetSystem(‘emptyRegionResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

BackgroundSeg is reentrant and processed without parallelization.

Possible Predecessor Functions

Threshold, Connection, RegionGrowing, Pouring, ClassNdimNorm

Possible Successor Functions

SelectShape

See Also

Threshold, HysteresisThreshold, Skeleton, ExpandRegion, SetSystem, SobelAmp, EdgesImage, Roberts, BandpassImage

Alternatives

Complement, Connection

Module

Region processing
Clip a region to a rectangle.

ClipRegion clips the input regions to the rectangle given by the four control parameters. ClipRegion is more efficient than calling Intersection with a rectangle generated by GenRectangle1.

Parameter

- **Region** (input iconic)  
  Region to be clipped.
- **RegionClipped** (output iconic)  
  Clipped regions.
- **Row1** (input control)  
  Row coordinate of the upper left corner of the rectangle.
  Default Value: 0
  Value Suggestions: Row1 ∈ {0, 128, 200, 256}
  Typical Range of Values: 0 ≤ Row1 ≤ 256
  Minimal Value Step: 1
  Recommended Value Step: 10

- **Column1** (input control)  
  Column coordinate of the upper left corner of the rectangle.
  Default Value: 0
  Value Suggestions: Column1 ∈ {0, 128, 200, 256}
  Typical Range of Values: 0 ≤ Column1 ≤ 256
  Minimal Value Step: 1
  Recommended Value Step: 10

- **Row2** (input control)  
  Row coordinate of the lower right corner of the rectangle.
  Default Value: 256
  Value Suggestions: Row2 ∈ {128, 200, 256, 512}
  Typical Range of Values: 0 ≤ Row2 ≤ 256
  Minimal Value Step: 1
  Recommended Value Step: 10

- **Column2** (input control)  
  Column coordinate of the lower right corner of the rectangle.
  Default Value: 256
  Value Suggestions: Column2 ∈ {128, 200, 256, 512}
  Typical Range of Values: 0 ≤ Column2 ≤ 256
  Minimal Value Step: 1
  Recommended Value Step: 10

Result

ClipRegion returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem(’noObjectResult’,<Result>) and the behavior in case of an empty input region via SetSystem(’emptyRegionResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

ClipRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Connection, RegionGrowing, Pouring

Possible Successor Functions

SelectShape, DispRegion

Alternatives

Intersection, GenRectangle1, ClipRegionRel

Module

Region processing
Clip a region relative to its size.

ClipRegionRel clips a region to a rectangle lying within the region. The size of the rectangle is determined by the enclosing rectangle of the region, which is reduced by the values given in the four control parameters. All four parameters must contain numbers larger or equal to zero, and determine by which amount the rectangle is reduced at the top (Top), at the bottom (Bottom), at the left (Left), and at the right (Right). If all parameters are set to zero, the region remains unchanged.

**Parameter**

- **Region** (input iconic) .......................................................... region(-array)  \( \leftarrow \) HRegionX / IHObjectX
  Regions to be clipped.

- **RegionClipped** (output iconic) ......................................... region(-array)  \( \leftarrow \) HRegionX / HUntypedObjectX
  Clipped regions.

- **Top** (input control) .................................................... integer  \( \leftarrow \) long
  Number of rows clipped at the top.
  Default Value: 1
  Value Suggestions: \( \text{Top} \in \{0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50\} \)
  (lin)Minimal Step: 1
  Recommended Step: 1

- **Bottom** (input control) ............................................. integer  \( \leftarrow \) long
  Number of rows clipped at the bottom.
  Default Value: 1
  Value Suggestions: \( \text{Bottom} \in \{0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50\} \)
  (lin)Minimal Step: 1
  Recommended Step: 1

- **Left** (input control) ................................................ integer  \( \leftarrow \) long
  Number of columns clipped at the left.
  Default Value: 1
  Value Suggestions: \( \text{Left} \in \{0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50\} \)
  (lin)Minimal Step: 1
  Recommended Step: 1

- **Right** (input control) .............................................. integer  \( \leftarrow \) long
  Number of columns clipped at the right.
  Default Value: 1
  Value Suggestions: \( \text{Right} \in \{0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50\} \)
  (lin)Minimal Step: 1
  Recommended Step: 1

**Result**

ClipRegionRel returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult',<Result>)` and the behavior in case of an empty input region via `SetSystem('emptyRegionResult',<Result>)`. If necessary, an exception handling is raised.

---

**Parallelization Information**

ClipRegionRel is *reentrant and automatically parallelized (on tuple level).*

**Possible Predecessor Functions**

Threshold, Connection, RegionGrowing, Pouring

**Possible Successor Functions**

SelectShape, DispRegion

**Alternatives**

SmallestRectangle1, Intersection, GenRectangle1, ClipRegion
**Module**

Region processing

```cpp
[out] HRegionX ConnectedRegions HRegionX.Connection ( )

void HOperatorSetX.Connection ([in] IHObjectX Region,
[out] HUntypedObjectX ConnectedRegions )
```

*Compute connected components of a region.*

Connection determines the connected components of the input regions given in `Region`. The neighborhood used for this can be set via `SetSystem('neighborhood',<4/8>)`. The default is 8-neighborhood, which is useful for determining the connected components of the foreground. The inverse operator of `Connection` is `Union1`.

**Parameter**

- Region (input iconic) ........................................... region(-array) ⇒ HRegionX / IHObjectX
  Input region.
- ConnectedRegions (output iconic) ......................... region ⇒ HRegionX / HUntypedObjectX
  Connected components.

**Example**

```cpp
read_image (Image,’affe’)
set_colored (WindowHandle,12)
threshold (Image,Light,150.0,255.0)
count_obj (Light,Number1)
fwrite_string (’Number of regions after threshold = ’ + Number1)
fnew_line ()
disp_region (Light,WindowHandle)
connection (Light,Many)
count_obj (Many,Number2)
fwrite_string (’Number of regions after threshold = ’ + Number2)
fnew_line ()
disp_region (Many,WindowHandle).
```

**Result**

Connection always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via `SetSystem(’noObjectResult’,<Result>)` and the behavior in case of an empty input region via `SetSystem(’emptyRegionResult’,<Result>)`. If necessary, an exception handling is raised.

**Parallelization Information**

Connection is reentrant and processed without parallelization.

**Possible Predecessor Functions**

AutoThreshold, Threshold, DynThreshold, Erosion1

**Possible Successor Functions**

SelectShape, SelectGray, ShapeTrans, SetColored, Dilation1, CountObj, ReduceDomain, AddChannels

**See Also**

SetSystem, Union1

**Alternatives**

BackgroundSeg
### CropDomainRel

Cut out an image area relative to the domain.

CropDomainRel cuts a rectangular area from the input images. The area is determined by the surrounding rectangle of the domain of the input image. The rectangle can be influenced by the control parameters to modify at the top (`Top`), at the bottom (`Bottom`), at the left (`Left`), and at the right (`Right`). Positive values result in a smaller, negative values in a larger size. If all parameters are set to zero, the region remains unchanged. The new image matrix has the size of a rectangle.

Cut out an image area relative to the domain.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Image</code></td>
<td>Input image.</td>
</tr>
<tr>
<td><code>ImagePart</code></td>
<td>Image area.</td>
</tr>
<tr>
<td><code>Top</code></td>
<td>Number of rows clipped at the top.</td>
</tr>
<tr>
<td><code>Bottom</code></td>
<td>Number of rows clipped at the bottom.</td>
</tr>
<tr>
<td><code>Left</code></td>
<td>Number of columns clipped at the left.</td>
</tr>
<tr>
<td><code>Right</code></td>
<td>Number of columns clipped at the right.</td>
</tr>
</tbody>
</table>

#### Result

CropDomainRel returns `TRUE` if all parameters are correct. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult',<Result>)` and the behavior in case of an empty input region via `SetSystem('emptyRegionResult',<Result>)`. If necessary, an exception handling is raised.

#### Parallelization Information

CropDomainRel is `reentrant` and automatically `parallelized` (on `tuple level`, `channel level`).

#### Possible Predecessor Functions

ReduceDomain, Threshold, Connection, RegionGrowing, Pouring

#### See Also

SmallestRectangle1, Intersection, GenRectangle1, ClipRegion

#### Alternatives

CropDomain, CropRectangle1

#### Module

Image / region / XLD management
Compute the distance transformation of a region.

DistanceTransform computes for every point of the input region Region (or its complement, respectively) the distance of the point to the border of the region. The parameter Foreground determines whether the distances are calculated for all points within the region (Foreground = 'true') or for all points outside the region (Foreground = 'false'). The distance is computed for every point of the output image DistanceImage, which has the specified dimensions Width and Height. The input region is always clipped to the extent of the output image. If it is important that the distances within the entire region should be computed, the region should be moved (see MoveRegion) so that it has only positive coordinates and the width and height of the output image should be large enough to contain the region. The extent of the input region can be obtained with SmallestRectangle1.

The parameter Metric determines which metric is used for the calculation of the distances. If Metric = 'city-block', the distance is calculated from the shortest path from the point to the border of the region, where only horizontal and vertical "movements" are allowed. They are weighted with a distance of 1. If Metric = 'chessboard', the distance is calculated from the shortest path to the border, where horizontal, vertical, and diagonal "movements" are allowed. They are weighted with a distance of 1. If Metric = 'octagonal', a combination of these approaches is used, which leads to diagonal paths getting a higher weight. If Metric = 'chamfer-3-4', horizontal and vertical movements are weighted with a weight of 3, while diagonal movements are weighted with a weight of 4. To normalize the distances, the resulting distance image is divided by 3. Since this normalization step takes some time, and one usually is interested in the relative distances of the points, the normalization can be suppressed with Metric = 'chamfer-3-4-unnormalized'. Finally, if Metric = 'euclidean', the computed distance is approximately Euclidean.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value List</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Region</strong> (input iconic)</td>
<td>region(-array) (\sim) HRegionX / HObjectX</td>
</tr>
<tr>
<td></td>
<td>Region for which the distance to the border is computed.</td>
</tr>
<tr>
<td><strong>DistanceImage</strong> (output iconic)</td>
<td>Image containing the distance information.</td>
</tr>
<tr>
<td><strong>Metric</strong> (input control)</td>
<td>string (\sim) String</td>
</tr>
<tr>
<td></td>
<td>Type of metric to be used for the distance transformation.</td>
</tr>
<tr>
<td><strong>Foreground</strong> (input control)</td>
<td>string (\sim) String</td>
</tr>
<tr>
<td></td>
<td>Compute the distance for pixels inside (true) or outside (false) the input region.</td>
</tr>
<tr>
<td><strong>Width</strong> (input control)</td>
<td>extent.x (\sim) long</td>
</tr>
<tr>
<td></td>
<td>Width of the output image.</td>
</tr>
<tr>
<td><strong>Height</strong> (input control)</td>
<td>extent.y (\sim) long</td>
</tr>
<tr>
<td></td>
<td>Height of the output image.</td>
</tr>
</tbody>
</table>

/* Step towards extracting the medial axis of a shape: */
gen_rectangle1 (Rectangle1, 0, 0, 200, 400)
gen_rectangle1 (Rectangle2, 200, 0, 400, 200)
union2 (Rectangle1, Rectangle2, Shape)
distance_transform (Shape, DistanceImage, 'chessboard', 'true', 640, 480)
DistanceTransform returns H_MSG_TRUE if all parameters are correct.

Parallelization Information

DistanceTransform is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
Threshold, DynThreshold, RegionGrowing

Possible Successor Functions
Threshold

See Also
Skeleton

Bibliography


Region processing

Eliminate runs of a given length.
EliminateRuns eliminates all runs of the run length encoding of the input regions which are shorter than ElimShorter or longer as ElimLonger.

Parameter

- **Region** (input iconic) ................................. region(-array) \( \leadsto \) HRegionX / IObjectX
  
  Region to be clipped.

- **RegionClipped** (output iconic) ....................... region(-array) \( \leadsto \) HRegionX / HUntypedObjectX
  
  Clipped regions.

- **ElimShorter** (input control) ......................... integer \( \leadsto \) long
  
  All runs which are shorter are eliminated.

  Default Value : 3

  Value Suggestions : ElimShorter \( \in \{ 2, 3, 4, 5, 6, 8, 10, 12, 15 \} \)

  Typical Range of Values : \( 1 \leq \text{ElimShorter} \leq 1 \text{(lin)} \)

  Minimal Value Step : 1

  Recommended Value Step : 1

- **ElimLonger** (input control) ............................... integer \( \leadsto \) long
  
  All runs which are longer are eliminated.

  Default Value : 1000

  Value Suggestions : ElimLonger \( \in \{ 50, 100, 200, 500, 1000, 2000 \} \)

  Typical Range of Values : \( 1 \leq \text{ElimLonger} \leq 1 \text{(lin)} \)

  Minimal Value Step : 1

  Recommended Value Step : 10

Result

EliminateRuns returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.
Fill gaps between regions or split overlapping regions.

ExpandRegion closes gaps between the input regions, which resulted from the suppression of small regions in a segmentation operator, for example, (mode ‘image’), or to separate overlapping regions (mode ‘region’). Both uses result from the expansion of regions. The operator works by adding or removing a one pixel wide “strip” to a region.

The expansion takes place only in regions, which are designated as not “forbidden” (parameter ForbiddenArea). The number of iterations is determined by the parameter Iterations. By passing ‘maximal’, ExpandRegion iterates until convergence, i.e., until no more changes occur. By passing 0 for this parameter, all non-overlapping regions are returned. The two modes of operation (‘image’ and ‘region’) are different in the following ways:

‘image’ The input regions are expanded iteratively until they touch another region or the image border. Because ExpandRegion processes all regions simultaneously, gaps between regions are distributed evenly to all regions. Overlapping regions are split by distributing the area of overlap evenly to both regions.

‘region’ No expansion of the input regions is performed. Instead, only overlapping regions are split by distributing the area of overlap evenly to the respective regions. Because the intersection with the original region is computed after the shrinking operation gaps in the output regions may result, i.e., the segmentation is not complete. This can be prevented by calling ExpandRegion a second time with the complement of the original regions as “forbidden area.”

Parameter

- **Regions** (input iconic) region(-array) ~ HRegionX / IHObjectX
  Regions for which the gaps are to be closed, or which are to be separated.
- **ForbiddenArea** (input iconic) region ~ HRegionX / IHObjectX
  Regions in which no expansion takes place.
- **RegionExpanded** (output iconic) region(-array) ~ HRegionX / HUntypedObjectX
  Expanded or separated regions.
- **Iterations** (input control) integer ~ VARIANT (integer, string)
  Number of iterations.
  Default Value: ‘maximal’
  Value Suggestions: Iterations ∈ {‘maximal’, 0, 1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100, 200}
  Typical Range of Values: 0 ≤ Iterations ≤ 50(lin)
  Minimal Value Step: 1
  Recommended Value Step: 1
- **Mode** (input control) string ~ String
  Expansion mode.
  Default Value: ‘image’
  Value List: Mode ∈ {‘image’, ‘region’}
read_image(Image,'fabrik')
threshold(Image,Light,100,255)
disp_region(Light,WindowHandle)
connection(Light,Seg)
expand_region(Seg,[],Exp1,'maximal','image')
set_colored(WindowHandle,12)
set_draw(WindowHandle,'margin')
disp_region(Exp1,WindowHandle)

Result
ExpandRegion always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>). The behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>), and the behavior in case of an empty result region via SetSystem('storeEmptyRegion',<true/false>). If necessary, an exception handling is raised.

Parallelization Information
ExpandRegion is reentrant and processed without parallelization.

Possible Predecessor Functions
Pouring,Threshold,DynThreshold,RegionGrowing

See Also
ExpandGray,Interjacent,Skeleton

Alternatives
Dilation

Module
Region processing

[out] HRegionX RegionFillUp HRegionX.FillUp ()

void HOperatorSetX.FillUp ([in] IHObjectX Region,
[out] HUntypedObjectX RegionFillUp)

Fill up holes in regions.
FillUp fills up holes in regions. The number of regions remains unchanged. The neighborhood type is set via SetSystem('neighborhood',<4/8>) (default: 8-neighborhood).

Parameter
>* Region (input iconic) .................................. region(-array) ↷ HRegionX / IHObjectX Input regions containing holes.
>* RegionFillUp (output iconic) ......................... region(-array) ↷ HRegionX / HUntypedObjectX Regions without holes.

Result
FillUp returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information
FillUp is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
Threshold,Connection,RegionGrowing,Pouring

Possible Successor Functions
SelectShape,DispRegion

See Also
Boundary

FillUpShape

Alternatives

Module

Region processing

Fill up holes in regions having given shape features.

FillUpShape fills up those holes in the input region Region having given shape features. The parameter Feature determines the shape feature to be used, while Min and Max determine the range the shape feature has to lie in in order for the hole to be filled up.

Parameter

\[\text{Region} \quad \text{(input iconic)} \quad \text{RegionFillUp} \quad \text{RegionFillUp} \quad \text{String Feature,} \quad \text{VARIANT Min,} \quad \text{VARIANT Max} \]

\[\text{HRegionX } \text{RegionFillUp} \Rightarrow \text{HRegionX.FillUpShape} \quad \text{[in]} \quad \text{String Feature,} \quad \text{[in]} \quad \text{VARIANT Min,} \quad \text{[in]} \quad \text{VARIANT Max} \]

\[\text{void } \text{HOperatorSetX.FillUpShape} \quad \text{[in]} \quad \text{IHObjectX Region,} \quad \text{[out]} \quad \text{HUntypedObjectX RegionFillUp,} \quad \text{[in]} \quad \text{String Feature,} \quad \text{[in]} \quad \text{VARIANT Min,} \quad \text{[in]} \quad \text{VARIANT Max} \]

\[\text{void } \text{HOperatorSetX.FillUpShape} \quad \text{[in]} \quad \text{IHObjectX Region,} \quad \text{[out]} \quad \text{HUntypedObjectX RegionFillUp,} \quad \text{[in]} \quad \text{String Feature,} \quad \text{[in]} \quad \text{VARIANT Min,} \quad \text{[in]} \quad \text{VARIANT Max} \]

Result

FillUpShape returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem(‘noObjectResult’,<Result>) and the behavior in case of an empty input region via SetSystem(‘emptyRegionResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

FillUpShape is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Connection, Regiongrowing, Pouring

Possible Successor Functions

SelectShape, DispRegion

SelectShape, Connection, AreaCenter

See Also

Alternatives

FillUp

Module

Region processing
Generate a region having a given Hamming distance.

HammingChangeRegion changes the region in the left upper part of the image given by Width and Height such that the resulting regions have a Hamming distance of Distance to the input regions. This is done by adding or removing Distance points from the input region.

Attention

If Width and Height are chosen too large the resulting region requires a lot of memory.

**Parameter**

- **InputRegion** (input iconic) ....................... region(-array)  
  HRegionX / IHObjectX  
  Region to be modified.

- **OutputRegion** (output iconic) ....................... region(-array)  
  HRegionX / HUntypedObjectX  
  Regions having the required Hamming distance.

- **Width** (input control) .................................. extent.x  
  Width of the region to be changed.  
  Default Value: 100  
  Value Suggestions: Width ∈ {64, 128, 256, 512}  
  Typical Range of Values: 1 ≤ Width ≤ 1000  
  Minimal Value Step: 1  
  Recommended Value Step: 10  
  Restriction: (Width > 0)

- **Height** (input control) .................................. extent.y  
  Height of the region to be changed.  
  Default Value: 100  
  Value Suggestions: Height ∈ {64, 128, 256, 512}  
  Typical Range of Values: 1 ≤ Height ≤ 1000  
  Minimal Value Step: 1  
  Recommended Value Step: 10  
  Restriction: (Height > 0)

- **Distance** (input control) ............................... integer  
  Hamming distance between the old and new regions.  
  Default Value: 1000  
  Value Suggestions: Distance ∈ {100, 500, 1000, 5000, 10000}  
  Typical Range of Values: 0 ≤ Distance ≤ 10000  
  Minimal Value Step: 1  
  Recommended Value Step: 10  
  Restriction: (Distance ≥ 0) ∧ (Distance ≤ (Width · Height))

HammingChangeRegion returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

HammingChangeRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Connection, RegionGrowing, Pouring, ClassNdimNorm

Possible Successor Functions

SelectShape

See Also

HammingDistance

Module

Region processing
Partition the image plane using given regions.

Interjacent partitions the image plane using the regions given in Region. The result is a region containing the extracted separating lines. The following modes of operation can be used:

- **'medial_axis'** This mode is used for regions that do not touch or overlap. The operator will find separating lines between the regions which partition the background evenly between the input regions. This corresponds to the following calls:
  
  \[
  \text{complement('full',Region,Tmp) skeleton(Tmp,Result)},
  \]

- **'border'** If the input regions do not touch or overlap this mode is equivalent to \text{Boundary(Region,Result)}, i.e., it replaces each region by its boundary. If regions are touching they are aggregated into one region. The corresponding output region then contains the boundary of the aggregated region, as well as the one pixel wide separating line between the original regions. This corresponds to the following calls:
  
  \[
  \text{boundary(Region,Tmpl,'inner') union1(Tmpl1,Tmp2) skeleton(Tmp2,Result)},
  \]

- **'mixed'** In this mode the operator behaves like the mode 'medial_axis' for non-overlapping regions. If regions touch or overlap, again separating lines between the input regions are generated on output, but this time including the “touching line” between regions, i.e., touching regions are separated by a line in the output region. This corresponds to the following calls:
  
  \[
  \text{erosion1(Region,Mask,Tmpl1,1) union1(Tmpl1,Tmp2) complement(full,Tmp2,Tmp3) skeleton(Tmp3,Result)},
  \]

where Mask denotes the following “cross mask”:

\[
\begin{array}{cccc}
  & & x & \\
  & x & x & x \\
  & x & & \\
\end{array}
\]

**Parameter**

- **Region** (input iconic) \(\rightarrow\) \text{HRegionX / IObjectX}
  
  Regions for which the separating lines are to be determined.

- **RegionInterjacent** (output iconic) \(\rightarrow\) \text{HRegionX / HUntypedObjectX}
  
  Output region containing the separating lines.

- **Mode** (input control) \(\rightarrow\) \text{String}
  
  Mode of operation.

  **Default Value:** 'mixed'

  **Value List:** \(\text{Mode} \in \{\text{'medial_axis'}, \text{'border'}, \text{'mixed'}\}\)

**Example**

```helloworld
read_image(Image,'wald1_rot')
mean(Image,Mean,31,31)
dyn_threshold(Mean,Seg,20)
interjacent(Seg,Graph,'medial_axis')
disp_region(Graph,WindowHandle)
```

**Result**

Interjacent always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via \text{SetSystem('noObjectResult',<Result>)}, the behavior in case of an empty input region via \text{SetSystem('emptyRegionResult',<Result>)}, and the behavior in case of an empty result region via \text{SetSystem('storeEmptyRegion',<true/false>)}. If necessary, an exception handling is raised.

**Parallelization Information**

Interjacent is reentrant and processed without parallelization.
JunctionsSkeleton detects junctions and end points in a skeleton (see Skeleton). The junctions in the input region Region are output as a region in JuncPoints, while the end points are output as a region in EndPoints.

**Parameter**

- **Region** (input iconic) region(-array) \( \rightsquigarrow \) HRegionX / IHObjectX Input skeletons.
- **EndPoints** (output iconic) region(-array) \( \rightsquigarrow \) HRegionX / HUntypedObjectX Extracted end points.
- **JuncPoints** (output iconic) region(-array) \( \rightsquigarrow \) HRegionX / HUntypedObjectX Extracted junctions.

**Example**

```c
/* non-connected branches of a skeleton */
skeleton(Region, Skeleton)
junctions_skeleton(Skeleton, EPoints, JPoints)
difference(S, JPoints, Rows)
set_system(‘neighbourhood’, 4)
connection(Rows, Parts).
```

**Result**

JunctionsSkeleton always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem(‘noObjectResult’, <Result>), the behavior in case of an empty input region via SetSystem(‘emptyRegionResult’, <Result>), and the behavior in case of an empty result region via SetSystem(‘storeEmptyRegion’, <true/false>). If necessary, an exception handling is raised.

**Parallelization Information**

JunctionsSkeleton is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

Threshold, Connection, RegionGrowing, Pouring

**Possible Successor Functions**

SelectShape, DispRegion

**See Also**

ExpandRegion, JunctionsSkeleton, Boundary

**Module**

Region processing

---

```
[out] HRegionX EndPoints HRegionX.JunctionsSkeleton
[[out] HRegionX JuncPoints ]

void HOperatorSetX.JunctionsSkeleton ([in] IHObjectX Region,
[[out] HUntypedObjectX EndPoints, [out] HUntypedObjectX JuncPoints ]

Find junctions and end points in a skeleton.
```

/* non-connected branches of a skeleton */
skeleton(Region, Skeleton)
junctions_skeleton(Skeleton, EPoints, JPoints)
difference(S, JPoints, Rows)
set_system(‘neighbourhood’, 4)
connection(Rows, Parts).

JunctionsSkeleton always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem(‘noObjectResult’, <Result>), the behavior in case of an empty input region via SetSystem(‘emptyRegionResult’, <Result>), and the behavior in case of an empty result region via SetSystem(‘storeEmptyRegion’, <true/false>). If necessary, an exception handling is raised.

JunctionsSkeleton is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

Threshold, Connection, RegionGrowing, Pouring

**Possible Successor Functions**

SelectShape, DispRegion

**See Also**

ExpandRegion, JunctionsSkeleton, Boundary

**Module**

Region processing

### Merge regions from line scan images.

The operator `MergeRegionsLineScan` connects adjacent regions, which were segmented from adjacent images with the height `ImageHeight`. This operator was especially designed to process regions that were extracted from images grabbed by a line scan camera. `CurrRegions` contains the regions from the current image and `PrevRegions` the regions from the previous one.

With the help of the parameter `MergeBorder` two cases can be distinguished: If the top (first) line of the current image touches the bottom (last) line of the previous image, `MergeBorder` must be set to 'top', otherwise set to 'bottom'.

If the operator `MergeRegionsLineScan` is used recursively, the parameter `MaxImagesRegion` determines the maximum number of images which are covered by a merged region. All older region parts are removed.

The operator `MergeRegionsLineScan` returns two region arrays. `PrevMergedRegions` contains all those regions from the previous input regions `PrevRegions`, which could not be merged with a current region. `CurrMergedRegions` collects all current regions together with the merged parts from the previous images. Merged regions will exceed the original image, because the previous regions are moved upward (`MergeBorder='top'`) or downward (`MergeBorder='bottom'`) according to the image height. For this the system parameter 'clip.region' (see also `SetSystem`) will internally be set to 'false'.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CurrRegions</strong> (input iconic)</td>
<td>Current input regions.</td>
</tr>
<tr>
<td><strong>PrevRegions</strong> (input iconic)</td>
<td>Merged regions from the previous iteration.</td>
</tr>
<tr>
<td><strong>CurrMergedRegions</strong> (output iconic)</td>
<td>Current regions, merged with old ones where applicable.</td>
</tr>
<tr>
<td><strong>PrevMergedRegions</strong> (output iconic)</td>
<td>Regions from the previous iteration which could not be merged with the current ones.</td>
</tr>
<tr>
<td><strong>ImageHeight</strong> (input control)</td>
<td>Height of the line scan images. Default Value: 512 Value List: <code>ImageHeight ∈ {240, 480, 512}</code></td>
</tr>
<tr>
<td><strong>MergeBorder</strong> (input control)</td>
<td>Image line of the current image, which touches the previous image. Default Value: 'top' Value List: <code>MergeBorder ∈ {'top', 'bottom'}</code></td>
</tr>
<tr>
<td><strong>MaxImagesRegion</strong> (input control)</td>
<td>Maximum number of images for a single region. Default Value: 3 Value Suggestions: <code>MaxImagesRegion ∈ {1, 2, 3, 4, 5}</code></td>
</tr>
</tbody>
</table>

#### Result

The operator `MergeRegionsLineScan` returns the value TRUE if the given parameters are correct. Otherwise, an exception will be raised.

#### Parallelization Information

`MergeRegionsLineScan` is *reentrant* and processed without parallelization.

#### Module

Region processing
Partition a region horizontally into rectangles.

PartitionDynamic partitions the input region into rectangles having an average width of Distance. The region is not split exactly into \( n \) parts, but rather starting from the calculated splitting point positions having a minimum number of set pixels in the vertical directions are extracted. The search area is at most plus/minus the width of a rectangle. If Percent is passed as 100, this search area is used fully. If it is passed as 0, no search is done.

If the region is smaller than the given size its output remains unchanged. A partition is only done if the size of the region is at least 1.5 times the size of the rectangle given by the parameters.

\[
\text{Region (input iconic)} \rightarrow \text{Region (-array)} \sim \text{HRegionX / IHObjectX Region to be partitioned.}
\]

\[
\text{Partitioned (output iconic)} \rightarrow \text{region} \sim \text{HRegionX / HUntypedObjectX Partitioned region.}
\]

\[
\text{Distance (input control)} \rightarrow \text{real } \sim \text{double Width of the individual rectangles.}
\]

\[
\text{Percent (input control)} \rightarrow \text{real } \sim \text{double Maximum shift of the partition point. Default Value : 20 Value Suggestions : Percent } \in \{0, 10, 20, 30, 40, 50, 70, 90, 100\} \text{ Typical Range of Values : } 0 \leq \text{Percent} \leq 0
\]

PartitionDynamic returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>), the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>), and the behavior in case of an empty result region via SetSystem('storeEmptyRegion',<true/false>). If necessary, an exception handling is raised.

PartitionDynamic is reentrant and automatically parallelized (on tuple level).

Threshold, Connection

See Also

Intersection, SmallestRectangle1, ShapeTrans, ClipRegion

Alternatives

Region processing

Partition a region into rectangles of equal size.

PartitionRectangle partitions the input region into rectangles having an extent of Width times Height. The region is always split into rectangles of equal size. Therefore, Width and Height are adapted to the actual
size of the region. If the region is smaller than the given size its output remains unchanged. A partition is only done if the size of the region is at least 1.5 times the size of the rectangle given by the parameters.

### Parameter

- **Region** (input iconic) region(-array) \(\sim\) HRegionX / IObjectX
  Region to be partitioned.
- **Partitioned** (output iconic) region(-array) \(\sim\) HRegionX / HUntypedObjectX
  Partioned region.
- **Height** (input control) \(\sim\) double
  Height of the individual rectangles.
- **Width** (input control) \(\sim\) double
  Width of the individual rectangles.

### Result

PartitionRectangle returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>), the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>), and the behavior in case of an empty result region via SetSystem('storeEmptyRegion',<true/false>). If necessary, an exception handling is raised.

### Parallelization Information

PartitionRectangle is reentrant and automatically parallelized (on tuple level).

### Possible Predecessor Functions

Threshold, Connection

### See Also

Intersection, SmallestRectangle1, ShapeTrans, ClipRegion

### Alternatives

PartitionDynamic

### Module

Region processing

```plaintext
[out] HRegionX RegionCount HRegionX.RankRegion ([in] long Width,
[in] long Height, [in] long Number )

void HOperatorSetX.RankRegion ([in] IObjectX Region,
[out] HUntypedObjectX RegionCount, [in] long Width, [in] long Height,
[in] long Number )
```

**Rank operator for regions.**

RankRegion calculates the binary rank operator. A filter mask of size \(Height \times Width\) is used. In the process, for each point in the region the number of points of Region lying within the filter mask are counted. If this number is greater or equal to Number, the current point is added to the output region. If

\[
Number = \frac{Height \times Width}{2},
\]

is chosen, the median operator is obtained.

### Attention

For Height and Width only odd values \(> 3\) are valid. If invalid parameters are chosen they are converted automatically (without raising an exception handling) to the next larger even values.

### Parameter

- **Region** (input iconic) region(-array) \(\sim\) HRegionX / IObjectX
  Region(s) to be transformed.
- **RegionCount** (output iconic) region(-array) \(\sim\) HRegionX / HUntypedObjectX
  Resulting region(s).
Width (input control) ................................................................. extent.x ~ long
  Width of the filter mask.
  Default Value : 15
  Value Suggestions : Width ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21}
  Typical Range of Values : 3 ≤ Width ≤ 3(lin)
  Minimal Value Step : 2
  Recommended Value Step : 2
  Restriction : (Width ∧ odd)

Height (input control) ................................................................. extent.y ~ long
  Height of the filter mask.
  Default Value : 15
  Value Suggestions : Height ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21}
  Typical Range of Values : 3 ≤ Height ≤ 3(lin)
  Minimal Value Step : 2
  Recommended Value Step : 2
  Restriction : (Height ∧ odd)

Number (input control) .............................................................. integer ~ long
  Minimum number of points lying within the filter mask.
  Default Value : 70
  Value Suggestions : Number ∈ {5, 10, 20, 40, 60, 80, 90, 120, 150, 200}
  Typical Range of Values : 1 ≤ Number ≤ 1(lin)
  Minimal Value Step : 1
  Recommended Value Step : 10
  Restriction : (Number > 0)

Example

read_image(Image, ’affe’)
mean_image(Image, Mean, 5, 5)
dyn_threshold(Mean, Points, 25)
rank_region(Points, Textur, 15, 15, 30)
gen_circle(Mask, 10, 10, 3)
opening1(Textur, Mask, Seg).

Result

RankRegion returns TRUE if all parameters are correct. The behavior in case of empty input (no regions
given) can be set via SetSystem(’noObjectResult’, <Result>) and the behavior in case of an empty
input region via SetSystem(’emptyRegionResult’, <Result>). If necessary, an exception handling is
raised.

Parallelization Information

RankRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Threshold, Connection, RegionGrowing, Pouring, ClassNdimNorm

Possible Successor Functions

SelectShape, DispRegion

See Also

RankImage, MeanImage

Alternatives

ClosingRectangle1, ExpandRegion

Module

Region processing
Remove noise from a region.

RemoveNoiseRegion removes noise from a region. In mode 'n₄', a structuring element consisting of the four neighbors of a point is generated. A dilation with this structuring element is performed, and the intersection of the result and the input region is calculated. Thus all pixels having no 4-connected neighbor are removed.

Parameter

- **InputRegion** (input iconic) ......................... region(-array)  ~  HRegionX / IHObjectX
  Regions to be modified.
- **OutputRegion** (output iconic) ....................... region(-array)  ~  HRegionX / HUntypedObjectX
  Less noisy regions.
- **Type** (input control) ................................................... string  ~  String
  Mode of noise removal.
  Default Value : 'n₄'
  Value List : Type ∈ { 'n₄', 'n₈', 'n₄₈' }

Result

RemoveNoiseRegion returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

RemoveNoiseRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Connection, RegionGrowing, Pouring, ClassNdimNorm

Possible Successor Functions

SelectShape

See Also

Dilation1, Intersection, GenRegionPoints

Module

Region processing

Transform the shape of a region.

ShapeTrans transforms the shape of the input regions depending on the parameter Type:

- 'convex' Convex hull.
- 'ellipse' Ellipse with the same moments and area as the input region.
- 'outer_circle' Smallest enclosing circle.
- 'inner_circle' Largest circle fitting into the region.
- 'rectangle1' Smallest enclosing rectangle parallel to the coordinate axes.
- 'rectangle2' Smallest enclosing rectangle.
- 'inner_center' The point on the skeleton of the input region having the smallest distance to the center of gravity of the input region.
**Parameter**

- **Region** (input iconic) .................................................. `HRegionX / IObjectX`
  Regions to be transformed.

- **RegionTrans** (output iconic) ........................................... `HRegionX / HUntypedObjectX`
  Transformed regions.

- **Type** (input control) .................................................... `string`
  Default Value: 'convex'
  Value List: Type ∈ { 'convex', 'ellipse', 'outer_circle', 'inner_circle', 'rectangle1', 'rectangle2', 'inner_center' }

**Result**

ShapeTrans returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult', <Result>)`. If necessary, an exception handling is raised.

**Parallelization Information**

ShapeTrans is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

- Connection, RegionGrowing

**Possible Successor Functions**

- DispRegion, RegionGrowingMean, AreaCenter

See Also

- Convexity, EllipticAxis, AreaCenter, SmallestRectangle1, SmallestRectangle2, SetShape, SelectShape, InnerCircle

Module

Region processing

```
[out] HRegionX Skeleton HRegionX.Skeleton ()
void HOperatorSetX.Skeleton ([in] IObjectX Region,
[out] HUntypedObjectX Skeleton )
```

Compute the skeleton of a region.

Skeleton computes the skeleton of the input regions.

**Parameter**

- **Region** (input iconic) .................................................. `HRegionX / IObjectX`
  Region to be thinned.

- **Skeleton** (output iconic) ............................................. `HRegionX / HUntypedObjectX`
  Resulting skeleton.

**Parameter Number** (`Skeleton = Region`)

**Result**

Skeleton returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult', <Result>)` and the behavior in case of an empty input region via `SetSystem('emptyRegionResult', <Result>)`. If necessary, an exception handling is raised.

**Parallelization Information**

Skeleton is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

- SobelAmp, EdgesImage, BandpassImage, Threshold, HysteresisThreshold

**Possible Successor Functions**

- JunctionsSkeleton, Pruning

See Also

- GraySkeleton, SobelAmp, EdgesImage, Roberts, BandpassImage, Threshold

Alternatives

- MorphSkeleton, Thinning

Sorting of regions with respect to their relative position.

The operator SortRegion sorts the regions with respect to their relative position. All sorting methods with the exception of 'character' use one point of the region. With the help of the parameter RowOrCol = 'row' these points will be sorted according to their row and then according to their column. By using 'column', the column value will be used first. The following values are available for the parameter SortMode:

'character' The regions will be treated like characters in a row and will be sorted according to their order in the line. If two regions overlap horizontally, they will be sorted with respect to their column values, otherwise they will be sorted with regard to their row values.

'first_point' The point with the lowest column value in the first row of the region.

'last_point' The point with the highest column value in the last row of the region.

'upper_left' Upper left corner of the surrounding rectangle.

'upper_right' Upper right corner of the surrounding rectangle.

'lower_left' Lower left corner of the surrounding rectangle.

'lower_right' Lower right corner of the surrounding rectangle.

The parameter Order determines whether the sorting order is increasing or decreasing: using 'true' the order will be increasing, using 'false' the order will be decreasing.

Attention

Parameter

- **Regions** (input iconic) ........................................... region \(\Rightarrow\) HRegionX / HObjectX
  Regions to be sorted.

- **SortedRegions** (output iconic) ................................. region \(\Rightarrow\) HRegionX / HUntypedObjectX
  Sorted regions.

- **SortMode** (input control) ........................................... string \(\Rightarrow\) String
  Kind of sorting.
  Default Value : 'first_point'
  Value List : SortMode \(\in\) {'character', 'first_point', 'last_point', 'upper_left', 'lower_left', 'upper_right', 'lower_right'}

- **Order** (input control) ........................................... string \(\Rightarrow\) String
  Increasing or decreasing sorting order.
  Default Value : 'true'
  Value List : Order \(\in\) {'true', 'false'}

- **RowOrCol** (input control) ........................................... string \(\Rightarrow\) String
  Sorting first with respect to row, then to column.
  Default Value : 'row'
  Value List : RowOrCol \(\in\) {'row', 'column'}

Example
If the parameters are correct, the operator SortRegion returns the value TRUE. Otherwise an exception will be raised.

**Parallelization Information**

SortRegion is reentrant and processed without parallelization.

**Possible Successor Functions**

DoOcrMulti, DoOcrSingle

**Module**

Region processing

---

**Result**

Split lines represented by one pixel wide, non-branching lines.

SplitSkeletonLines splits lines represented by one pixel wide, non-branching regions into shorter lines based on their curvature. A line is split if the maximum distance of a point on the line to the line segment connecting its end points is larger than MaxDistance (split & merge algorithm). The start and end points of the approximating line segments are returned in BeginRow, BeginCol, EndRow, and EndCol.

**Attention**

The input regions must represent non-branching lines, that is single branches of the skeleton.

**Parameter**

- **SkeletonRegion** (input iconic) region \( \sim \) HRegionX / IHObjectX

  Input lines (represented by 1 pixel wide, non-branching regions).

- **MaxDistance** (input control) integer \( \sim \) long

  Maximum distance of the line points to the line segment connecting both end points.

  **Default Value**: 3
  **Value Suggestions**: MaxDistance \( \in \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \} \)
  **Typical Range of Values**: \( 1 \leq \text{MaxDistance} \leq 1(\text{lin}) \)
  **Minimal Value Step** : 1
  **Recommended Value Step** : 1

- **BeginRow** (output control) line.begin.y \( \sim \) VARIANT (integer)

  Row coordinates of the start points of the output lines.

- **BeginCol** (output control) line.begin.x \( \sim \) VARIANT (integer)

  Column coordinates of the start points of the output lines.

- **EndRow** (output control) line.end.y \( \sim \) VARIANT (integer)

  Row coordinates of the end points of the output lines.

- **EndCol** (output control) line.end.x \( \sim \) VARIANT (integer)

  Column coordinates of the end points of the output lines.

**Example**

read_image(Image,'fabrik')
edges_image (Image, ImaAmp, ImaDir, 'lanser2', 0.5, 'nms', 8, 16)
threshold (ImaAmp, RawEdges, 8, 255)
skeleton (RawEdges, Skeleton)
junctions_skeleton (Skeleton, EndPoints, JuncPoints)
difference (RawEdges, Skeleton, SkelWithoutJunc)
connection (SkelWithoutJunc, SingleBranches)
select_shape (SingleBranches, SelectedBranches, 'area', 'and', 16, 99999)
split_skeleton_lines (SelectedBranches, 3, BeginRow, BeginCol, EndRow,
SplitSkeletonLines always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult’,<Result>), the behavior in case of an empty input region via SetSystem('emptyRegionResult’,<Result>), and the behavior in case of an empty result region via SetSystem('storeEmptyRegion’,<true/false>). If necessary, an exception handling is raised.

Parallelization Information

SplitSkeletonLines is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Connection, SelectShape, Skeleton, JunctionsSkeleton, Difference

Possible Successor Functions

SelectLines, PartitionLines, DispLine

See Also

SplitSkeletonRegion, DetectEdgeSegments

Module

Region processing

| Split lines represented by one pixel wide, non-branching regions. |
|---|---|
| SplitSkeletonRegion splits lines represented by one pixel wide, non-branching regions into shorter lines based on their curvature. A line is split if the maximum distance of a point on the line to the line segment connecting its end points is larger than MaxDistance (split & merge algorithm). However, not the approximating lines are returned, but rather the original lines split into several output regions. |
| Attention |
| The input regions must represent non-branching lines, that is single branches of the skeleton. |
| Parameter |
| ▶ SkeletonRegion (input iconic) ......................region(-array) ↦ HRegionX / IObjectX |
| Input lines (represented by 1 pixel wide, non-branching regions). |
| ▶ RegionLines (output iconic) ......................region ↦ HRegionX / HUntypedObjectX |
| Split lines. |
| ▶ MaxDistance (input control) ...........................integer ↦ long |
| Maximum distance of the line points to the line segment connecting both end points. |
| Default Value : 3 |
| Value Suggestions : MaxDistance ∈ \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} |
| Typical Range of Values : 1 ≤ MaxDistance ≤ 1(lin) |
| Minimal Value Step : 1 |
| Recommended Value Step : 1 |

Example

read_image(Image,’fabrik’)
edges_image (Image, ImaAmp, ImaDir, ’lanser2’, 0.5, ’nms’, 8, 16)
threshold (ImaAmp, RawEdges, 8, 255)
skeleton (RawEdges, Skeleton)
junctions_skeleton (Skeleton, EndPoints, JuncPoints)
difference (Skeleton, JuncPoints, SkelWithoutJunc)
connection (SkelWithoutJunc, SingleBranches)
select_shape (SingleBranches, SelectedBranches, 'area', 'and', 16, 99999)
split_skeleton_region (SelectedBranches, Lines, 3)

---

**Result**

SplitSkeletonRegion always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>), the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>), and the behavior in case of an empty result region via SetSystem('storeEmptyRegion',<true/false>). If necessary, an exception handling is raised.

---

**Parallelization Information**

SplitSkeletonRegion is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

Connection, SelectShape, Skeleton, JunctionsSkeleton, Difference

---

**Possible Successor Functions**

CountObj, SelectShape, SelectObj, AreaCenter, EllipticAxis, SmallestRectangle2, GetRegionPolygon, GetRegionContour

---

**See Also**

SplitSkeletonLines, GetRegionPolygon, GenPolygonsXld

---

**Module**

Region processing
Chapter 10

Segmentation

Segment an image using thresholds determined from its histogram.

AutoThreshold segments a single-channel image using multiple thresholding. First the relative histogram of the gray values is determined. Then relevant minima are extracted from the histogram, which are used successively as parameters for a thresholding operation. The thresholds used are 0, 255, and all minima extracted from the histogram (after the histogram has been smoothed). For each gray value interval one region is generated. Thus, the number of regions is the number of minima + 1. The larger the value of Sigma is chosen, the less regions will be extracted. This operator is particularly suited if the regions to be extracted exhibit similar gray values (homogeneous regions).

Parameter

- **Image** (input iconic).................image(-array) $\leadsto$ HImageX / IObjectX (byte)
  Image to be segmented.
- **Regions** (output iconic)......................region $\leadsto$ HRegionX / HUntypedObjectX
  Regions with gray values within the automatically determined intervals.
- **Sigma** (input control) ......................number $\leadsto$ VARIANT (integer, real)
  Sigma for the Gaussian smoothing of the histogram.

Default Value: 2.0

Value Suggestions: $\Sigma \in \{0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0\}$

Typical Range of Values: $0.0 \leq \Sigma \leq 0.0$ (lin)

Minimal Value Step: 0.01

Recommended Value Step: 0.3

Restriction: $(\Sigma > 0.0)$

Parallelization Information

AutoThreshold is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
AnisotropeDiff, MedianImage, Illuminate

Possible Successor Functions
Connection, SelectShape, SelectGray

Alternatives
GrayHisto, SmoothFunct1DGauss, Threshold

Module

Region processing

---

[out] HRegionX Regions HImageX.AutoThreshold ([in] VARIANT Sigma )

void HOperatorSetX.AutoThreshold ([in] IObjectX Image,
[out] HUntypedObjectX Regions, [in] VARIANT Sigma )
Segment a black-and-white image using an automatically determined threshold.

BinThreshold segments a single-channel gray value image using an automatically determined threshold. First the relative histogram of the gray values is determined. Then relevant minima are extracted from the histogram, which are used as parameters for a thresholding operation. In order to reduce the number of minima, the histogram is smoothed with a Gaussian, as in AutoThreshold. The mask size is enlarged until there is only one minimum in the smoothed histogram. The selected region contains the pixels with gray values from 0 to the minimum. This operator is particularly suited for the segmentation of dark characters on a light paper.

Parameter

- **Image** (input iconic) ......................... image(-array) $\sim$ HImageX / IHObjectX (byte)
  Image to be segmented.
- **Region** (output iconic) ....................... region(-array) $\sim$ HRegionX / HUntypedObjectX
  Dark regions of the image.

Parallelization Information

BinThreshold is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

AnisotropeDiff, MedianImage, Illuminate

Possible Successor Functions

Connection, SelectShape, SelectGray

Alternatives

AutoThreshold, GrayHisto, SmoothFunct1DGauss, Threshold

Module

Region processing

Perform a threshold segmentation for extracting characters.

CharThreshold segments a single-channel image using an automatically determined threshold. It is assumed that the background of the image is bright and the foreground (the characters) are dark. First a histogram of the gray values in the image Image is computed in the points given by the region HistoRegion. Then the maximum in the histogram is computed (most frequently occurring gray value = bright paper). Finally, the first gray value to the “left” of the maximum is located, which is at least Percent less frequently occurring than the maximum. In addition a Gaussian smoothing can be applied to suppress gaps in the histogram.

Parameter

- **Image** (input iconic) ......................... image(-array) $\sim$ HImageX / IHObjectX (byte)
  Image to be segmented.
- **HistoRegion** (input iconic) ...................... region $\sim$ HRegionX / IHObjectX
  Region in which the histogram is computed.
- **Characters** (output iconic) ...................... region(-array) $\sim$ HRegionX / HUntypedObjectX
  Dark regions (characters).
**Sigma** (input control) ...................................................... number $\sim$ double
Sigma for the Gaussian smoothing of the histogram.
- **Default Value**: 2.0
- **Value Suggestions**: $\Sigma \in \{0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0\}$
- **Typical Range of Values**: $0.0 \leq \Sigma \leq 0.0$ (lin)
- **Minimal Value Step**: 0.01
- **Recommended Value Step**: 0.2

**Percent** (input control) .................................................... number $\sim$ VARIANT (integer, real)
Percentage for the gray value difference.
- **Default Value**: 95
- **Value Suggestions**: $\% \in \{90, 92, 95, 96, 97, 98, 99, 99.5, 100\}$
- **Typical Range of Values**: $0.0 \leq \% \leq 0.0$ (lin)
- **Minimal Value Step**: 0.1
- **Recommended Value Step**: 0.5

**Threshold** (output control) .............................................. integer(-array) $\sim$ VARIANT (integer)
Calculated threshold.
- **Parallelization Information**
  CharThreshold is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**
- AnisotropeDiff, MedianImage, Illuminate

---

**Possible Successor Functions**
- Connection, SelectShape, SelectGray

---

**Alternatives**
- BinThreshold, AutoThreshold, GrayHisto, SmoothFunct1DGauss, Threshold

**Module**

Region processing

- **CheckDifference**
  
  ```
  ```

  Compare two images pixel by pixel.

  CheckDifference selects from the input image Image those pixels ($g_i = g_{\text{image}}$), whose gray value difference to the corresponding pixels in Pattern are inside (outside) of the interval $[\text{DiffLowerBound}, \text{DiffLowerBound}]$. The pixels of Pattern are translated by (AddRow, AddCol) with respect to Image. Let $g_p$ be the gray value from Pattern translated by (AddRow, AddCol) w.r.t. $g_i$.

  If the selected mode Mode is 'diff inside', a pixel $g_i$ is selected if

  $$g_i - g_p - \text{GrayOffset} > \text{DiffLowerBound} \quad \text{and} \quad g_i - g_p - \text{GrayOffset} < \text{DiffUpperBound}.$$ 

  If the mode is set to 'diff outside', a pixel $g_i$ is selected if

  $$g_i - g_p - \text{GrayOffset} \leq \text{DiffLowerBound} \quad \text{or} \quad g_i - g_p - \text{GrayOffset} \geq \text{DiffUpperBound}.$$ 

  This test is performed for all points of the domain (region) of Image, intersected with the domain of the translated Pattern. All points fulfilling the above condition are aggregated in the output region. The two images may be of different size. Typically, Pattern is smaller than Image.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
<th>Default Value</th>
<th>Value Suggestions</th>
<th>Typical Range of Values</th>
<th>Minimal Value Step</th>
<th>Recommended Value Step</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong></td>
<td>Image to be examined.</td>
<td>IHImageX / IHObjectX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Pattern</strong></td>
<td>Comparison image.</td>
<td>IHImageX / IHObjectX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Selected</strong></td>
<td>Points in which the two images are similar/different.</td>
<td>HRegionX / HUntypedObjectX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mode</strong></td>
<td>Mode: return similar or different pixels.</td>
<td>string</td>
<td>'diff_outside'</td>
<td>['diff_inside', 'diff_outside']</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DiffLowerBound</strong></td>
<td>Lower bound of the tolerated gray value difference.</td>
<td>number</td>
<td>-5</td>
<td>[-255, -1, -2, -3, -5, -7, -10, -12, -15, -17, -20, -25, -30]</td>
<td></td>
<td>1</td>
<td>2</td>
<td>(-255 ≤ DiffLowerBound) ∧ (DiffLowerBound ≤ 255)</td>
</tr>
<tr>
<td><strong>DiffUpperBound</strong></td>
<td>Upper bound of the tolerated gray value difference.</td>
<td>number</td>
<td>5</td>
<td>[0, 1, 2, 3, 5, 7, 10, 12, 15, 17, 20, 25, 30]</td>
<td></td>
<td>1</td>
<td>2</td>
<td>(-255 ≤ DiffUpperBound) ∧ (DiffUpperBound ≤ 255)</td>
</tr>
<tr>
<td><strong>GrayOffset</strong></td>
<td>Offset gray value subtracted from Image.</td>
<td>number</td>
<td>0</td>
<td>[-30, -25, -20, -17, -15, -12, -10, -7, -5, -3, -2, -1, 0, 1, 2, 3, 5, 7, 10, 12, 15, 17, 20, 25, 30]</td>
<td></td>
<td>1</td>
<td>2</td>
<td>(-255 ≤ GrayOffset) ∧ (GrayOffset ≤ 255)</td>
</tr>
<tr>
<td><strong>AddRow</strong></td>
<td>Row coordinate, by which Pattern is translated.</td>
<td>point.y</td>
<td>0</td>
<td>[200, -100, -20, -10, 0, 10, 20, 100, 200]</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>AddCol</strong></td>
<td>Column coordinate, by which Pattern is translated.</td>
<td>point.x</td>
<td>0</td>
<td>[200, -100, -20, -10, 0, 10, 20, 100, 200]</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**Result**

CheckDifference returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

**Parallelization Information**

CheckDifference is reentrant and automatically parallelized (on tuple level).
Possible Successor Functions

Connection, SelectShape, ReduceDomain, SelectGray, RankRegion, Dilation1, Opening

Alternatives

SubImage, DynThreshold

Module

Region processing

Segment an image using two-dimensional pixel classification.

Class2dimSup classifies the points in two-channel images using a two-dimensional feature space. For each point, two gray values (one from each image) are used as features. The feature space is represented by the input region. The classification is done as follows:

A point from the input region of an image is accepted if the point \((g_r, g_c)\), which is determined by the respective gray values, is contained in the region \(\text{FeatureSpace}\). \(g_r\) is here a gray value from the image \(\text{ImageRow}\), while \(g_c\) is the corresponding gray value from \(\text{ImageCol}\).

Let \(P\) be a point with the coordinates \(P = (L, C)\), \(g_r\) be the gray value at position \((L, C)\) in the image \(\text{ImageRow}\), and \(g_c\) be the gray value at position \((L, C)\) in the image \(\text{ImageCol}\). Then the point \(P\) is aggregated into the output region if

\[(g_r, g_c) \in \text{FeatureSpace}\]

\(g_r\) is interpreted as row coordinate and \(g_c\) as column coordinate.

For the generation of \(\text{FeatureSpace}\), see \(\text{Histo2dim}\). The feature space can be modified by applying region transformation operators, such as \(\text{RankRegion}\), \(\text{Dilation1}\), \(\text{ShapeTrans}\), \(\text{EllipticAxis}\), etc., before calling \(\text{Class2dimSup}\).

The parameters \(\text{ImageCol}\) and \(\text{ImageRow}\) must contain an equal number of images with the same respective size. The image points are taken from the intersection of the domains of both images (see \(\text{ReduceDomain}\)).

Parameter

- \(\text{ImageCol}\) (input iconic) ............... image(-array) \sim HImageX / HObjectX (byte, int1, cyclic, direction) 
  Input image (first channel).
- \(\text{ImageRow}\) (input iconic) .................. image(-array) \sim HImageX / HObjectX (byte) 
  Input image (second channel).
- \(\text{FeatureSpace}\) (input iconic) ................. region(-array) \sim HRegionX / HObjectX 
  Region defining the feature space.
- \(\text{RegionClass2Dim}\) (output iconic) .......... region(-array) \sim HRegionX / HUntypedObjectX 
  Classified regions.

Result

\(\text{Class2dimSup}\) always returns TRUE. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with \(\text{SetSystem}\). If necessary, an exception is raised.

Parallelization Information

\(\text{Class2dimSup}\) is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

\(\text{Histo2Dim}, \text{Threshold}, \text{DrawRegion}, \text{Dilation1}, \text{Opening}, \text{ShapeTrans}\)

Possible Successor Functions

\(\text{Connection}, \text{SelectShape}, \text{SelectGray}\)
Segment two images by clustering.

Class2dimUnsup performs a classification with two single-channel images. First a two-dimensional histogram of the two images is computed (Histo2Dim). In this histogram the first maximum is extracted; it serves as the first cluster center. The histogram is computed with the intersection of the domains of both images (see ReduceDomain). After this, all pixels in the images, which are at most Threshold pixels from the cluster center in the maximum norm, are determined. These pixels form one output region. Next, the pixels thus classified are deleted from the histogram so that they are not taken into account for the next class. In this modified histogram again the maximum is extracted; it again serves as a cluster center. The above steps are repeated NumClasses times; thus, NumClasses output regions result. Only pixels defined in both images are returned.

Attention

Both input images must have the same size.

Parameter

- **Image1** (input iconic) ................................. image ~IHImageX / IHObjectX (byte)
  First input image.
- **Image2** (input iconic) ................................. image ~IHImageX / IHObjectX (byte)
  Second input image.
- **Classes** (output iconic) .............................. region ~HRegionX / HUntypedObjectX
  Segmentation result.
- **Threshold** (input control) .......................... integer ~long
  Threshold (maximum distance to the cluster’s center).
  Default Value: 15
  Value Suggestions: Threshold ∈ {0, 2, 5, 8, 12, 17, 20, 30, 50, 70}
- **NumClasses** (input control) ......................... integer ~long
  Number of classes (cluster centers).
  Default Value: 5
  Value Suggestions: NumClasses ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 30, 40, 50}

Result

Class2dimUnsup returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information

Class2DimUnsup is reentrant and processed without parallelization.

Possible Predecessor Functions

Decompose2, Decompose3, MedianImage, AnisotropeDiff, ReduceDomain

Possible Successor Functions

SelectShape, SelectGray, Connection

Alternatives

Threshold, Histo2Dim, Class2DimSup, ClassNdimNorm, ClassNdimBox

Module

Region processing
Classify pixels using hyper-cuboids.

ClassNdimBox classifies the pixels of the multi-channel image given in MultiChannelImage. To do so, the classificator ClassifHandle created with CreateClassBox is used. The classificator can be trained using ClassNdimBox or as described with CreateClassBox. More information on the structure of the classificator can be found also under that operator.

MultiChannelImage is a multi channel image. Its pixel values are used for the classification.

--- Parameter ---

- **MultiChannelImage** (input iconic) .... multichannel-image-array \(\rightsquigarrow\) HImageX / IHObjectX (byte, int1, int2, int4, real, direction, cyclic)
  - Multi-channel input image.

- **Regions** (output iconic) .................... region \(\rightsquigarrow\) HRegionX / HUntypedObjectX
  - Segmentation result.

- **ClassifHandle** (input control) .................. class_box \(\rightsquigarrow\) HClassBoxX / long
  - Classificator’s handle number.

--- Example ---

```
read_image(Bild,'meer')
disp_image(Image,WindowHandle)
set_color(WindowHandle,'green')
fwrite_string('Draw the learning region')
fnew_line()
draw_region(Reg1,WindowHandle)
reduce_domain(Image,Reg1,Foreground)
set_color(WindowHandle,'red')
fwrite_string('Draw Background')
fnew_line()
draw_region(Reg2,WindowHandle)
reduce_domain(Image,Reg2,Background)
fwrite_string('Start to learn')
fnew_line()
create_classif(ClassifHandle)
class_ndim_box(Foreground,Background,Image,ClassifHandle)
fwrite_string('start to classificate')
fnew_line()
class_ndim_box(Image,Res,ClassifHandle)
set_draw(WindowHandle,'fill')
disp_region(Res,WindowHandle)
free_classif(ClassifHandle).
```

--- Result ---

ClassNdimBox returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags ‘no_object_result’, ‘empty_region_result’, and ‘store_empty_region’ with SetSystem. If necessary, an exception is raised.

--- Parallelization Information ---

ClassNdimBox is local and processed completely exclusively without parallelization.

--- Possible Predecessor Functions ---

CreateClassBox, LearnClassBox, MedianImage, Compose2, Compose3, Compose4
ClassNdimNorm classifies the pixels of the multi-channel image given in MultiChannelImage. The result is returned in Regions as one region per classification object. The metric used ('euclid' or 'maximum') is determined by Metric. This parameter must be set to the same value used in LearnNdimNorm. The parameter is used to determine whether one region ('single') or multiple regions ('multiple') have to be generated for each cluster. Radius determines the radii or half edge length of the clusters, respectively. Center determines their centers.

ClassNdimNorm returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information
ClassNdimNorm is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
LearnNdimNorm, Compose2, Compose3, Compose4

Possible Successor Functions
Connection, SelectShape, ReduceDomain, SelectGray

See Also
DispCircle, DispRectangle

Alternatives
ClassNdimBox, Class2DimSup
Detect straight edge segments.  

DetectEdgeSegments detects straight edge segments in the gray image Image. The extracted edge segments are returned as line segments with start point (BeginRow, BeginCol) and end point (EndRow, EndCol). Edge detection is based on the Sobel filter, using 'sum_abs' as parameter and SobelSize as the filter mask size (see SobelAmp). Only pixels with a filter response larger than MinAmplitude are used as candidates for edge points. These thresholded edge points are thinned and split into straight segments. Due to technical reasons, edge points in which several edges meet are lost. Therefore, DetectEdgeSegments usually does not return closed object contours. The parameter MaxDistance controls the maximum allowed distance of an edge point to its approximating line. For efficiency reasons, the sum of the absolute values of the coordinate differences is used instead of the Euclidean distance. MinLength controls the minimum length of the line segments. Lines shorter than MinLength are not returned.

**Parameter**

- **Image** (input iconic) .............................. (multichannel-)image(-array)  \(\sim\)  HImageX / IObjectX (byte)  
  Input image.

- **SobelSize** (input control) .............................................................. integer  \(\sim\)  long  
  Mask size of the Sobel operator.  
  Default Value : 5  
  Value List : SobelSize \(\in\) \{3, 5, 7, 9, 11, 13\}

- **MinAmplitude** (input control) ........................................................ integer  \(\sim\)  long  
  Minimum edge strength.  
  Default Value : 32  
  Value Suggestions : MinAmplitude \(\in\) \{10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 80, 90, 100, 110\}  
  Typical Range of Values : 1 \(\leq\) MinAmplitude \(\leq\) 1  
  Minimal Value Step : 1  
  Recommended Value Step : 1  
  Restriction : (MinAmplitude \(\geq\) 0)

- **MaxDistance** (input control) ........................................................ integer  \(\sim\)  long  
  Maximum distance of the approximating line to its original edge.  
  Default Value : 3  
  Value Suggestions : MaxDistance \(\in\) \{2, 3, 4, 5, 6, 7, 8\}  
  Typical Range of Values : 1 \(\leq\) MaxDistance \(\leq\) 1  
  Minimal Value Step : 1  
  Recommended Value Step : 1  
  Restriction : (MaxDistance \(\geq\) 0)

- **MinLength** (input control) ........................................................ integer  \(\sim\)  long  
  Minimum length of to resulting line segments.  
  Default Value : 10  
  Value Suggestions : MinLength \(\in\) \{3, 5, 7, 9, 11, 13, 16, 20\}  
  Typical Range of Values : 1 \(\leq\) MinLength \(\leq\) 1  
  Minimal Value Step : 1  
  Recommended Value Step : 1  
  Restriction : (MinLength \(\geq\) 0)

- **BeginRow** (output control) .......................... line.begin.y  \(\sim\)  VARIANT( integer )  
  Row coordinate of the line segments' start points.
DetectEdgeSegments returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information
DetectEdgeSegments is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessor Functions
SigmaImage, MedianImage

Possible Successor Functions
SelectLines, PartitionLines, SelectLinesLongest, LinePosition, LineOrientation

Alternatives
SobelAmp, Threshold, Skeleton

Module

```
```

Threshold operator for signed images.

DualThreshold segments the input image into a region with gray values \( \geq \) Threshold (“positive” regions) and a region with gray values \( \leq -\)Threshold (“negative” regions). “Positive” or “negative” regions having a size of less than MinSize are suppressed, as well as regions whose maximum gray value is less than MinGray in absolute value.

The segmentation performed is not complete, i.e., the “positive” and “negative” regions together do not necessarily cover the entire image: Areas with a gray value between \(-\)Threshold and Threshold, \(-\)MinGray and MinGray respectively, are not taken into account.

DualThreshold is usually called after applying a Laplace operator (Laplace, DerivateGauss or DiffOfGauss) or the difference of two images (SubImage) to an image.

The zero crossings of a Laplace image correspond to edges in an image, and are the separating regions of the “positive” and “negative” regions in the Laplace image. They can be determined by calling DualThreshold with Threshold = 1. The parameter MinGray controls the noise invariance, while MinSize controls the resolution of the edge detection.

Using byte images only the positive part of the operator is applied. Therefore DualThreshold behaves like a standard threshold operator (Threshold) with successive Connection and SelectGray.

Parameter

- **Image** (input iconic) ............... image(-array)  \( \sim \) HImageX / IHObjectX (byte, int2, int4, real ) 
  Input Laplace image.

- **RegionCrossings** (output iconic) ...................... region  \( \sim \) HRegionX / HUntypedObjectX
  “Positive” and “negative” regions.
**MinSize** (input control) .......................................................... integer \(\sim\) long
Regions smaller than MinSize are suppressed.
Default Value : 20
Value Suggestions : \(\text{MinSize} \in \{0, 10, 20, 50, 100, 200, 500, 1000\}\)
Typical Range of Values : \(0 \leq \text{MinSize} \leq 0\) (lin)
Minimal Value Step : 1
Recommended Value Step : 10

**MinGray** (input control) .......................................................... real \(\sim\) double
Regions whose maximum absolute gray value is smaller than MinGray are suppressed.
Default Value : 5.0
Value Suggestions : \(\text{MinGray} \in \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 20.0\}\)
Typical Range of Values : \(0.001 \leq \text{MinGray} \leq 0.001\) (lin)
Minimal Value Step : 1.0
Recommended Value Step : 10.0
Restriction : \((\text{MinGray} > 0)\)

**Threshold** (input control) .................................................... real \(\sim\) double
Regions which have a gray value larger than Threshold (or smaller than \(-\text{Threshold}\)) are suppressed.
Default Value : 2.0
Value Suggestions : \(\text{Threshold} \in \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 20.0\}\)
Typical Range of Values : \(0.001 \leq \text{Threshold} \leq 0.001\) (lin)
Minimal Value Step : 1.0
Recommended Value Step : 10.0
Restriction : \((\text{Threshold} \geq 1) \land (\text{Threshold} \leq \text{MinGray})\)

---

**Example**

```c
/* Edge detection with the Laplace operator (and edge thinning) */
diff_of_gauss(Image, Laplace, 2.0, 1.6)
/* finde "positive" und "negative" Bildbereiche: */
dual_threshold(Laplace, Region, 20, 2, 1)
/*The zero runnings are the complement to these image section: */
complement('full', Region, Nulldurchgaenge).

/* Simulation of \OpRef{dual_threshold} */
dual_threshold(Laplace, Result, MinS, MinG, Threshold):
  threshold(Laplace, Tmp1, Threshold, 999999)
  connection(Tmp1, Tmp2)
  select_shape(Tmp2, Tmp3, 'area', 'and', MinS, 999999)
  select_gray(Laplace, Tmp3, Tmp4, 'max', 'and', MinG, 999999)
  threshold(Laplace, Tmp5, -999999, -Threshold)
  connection(Tmp5, Tmp6)
  select_shape(Tmp6, Tmp7, 'area', 'and', MinG, 999999)
  select_gray(Laplace, Tmp7, Tmp8, 'min', 'and', -999999, -MinG)
  concat_obj(Tmp4, Tmp8, Result).
```

---

**Result**

DualThreshold returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags \texttt{'no object result'}, \texttt{'empty region result'}, and \texttt{'store empty region'} with \texttt{SetSystem}. If necessary, an exception is raised.

---

**Parallelization Information**

DualThreshold is \texttt{reentrant} and automatically \texttt{parallelized} (on tuple level).

---

Possible Predecessor Functions

MinMaxGray, SobelAmp, GaussImage, ReduceDomain, DiffOfGauss, SubImage, DerivateGauss

---

Possible Successor Functions

Connection, Dilation1, Erosion1, Opening, Closing, RankRegion, ShapeTrans, Skeleton

---

See Also

Laplace, DiffOfGauss, ExpandRegion

HALCON 6.0
Segment an image using a local threshold.

DynThreshold selects from the input image those regions in which the pixels fulfill a threshold condition. Let \( g_o = g_{\text{OrigImage}} \) and \( g_m = g_{\text{ThresholdImage}} \). Then the condition for \( \text{LightDark} = \text{‘light’} \) is:

\[
g_o \geq g_m + \text{Offset}
\]

For \( \text{LightDark} = \text{‘dark’} \) the condition is:

\[
g_o \leq g_m - \text{Offset}
\]

For \( \text{LightDark} = \text{‘equal’} \) it is:

\[
g_m - \text{Offset} \leq g_o \leq g_m + \text{Offset}
\]

Finally, for \( \text{LightDark} = \text{‘not equal’} \) it is:

\[
g_m - \text{Offset} > g_o \lor g_o > g_m + \text{Offset}
\]

This means that all points in \( \text{OrigImage} \) whose gray value is larger than or equal to the gray value in \( \text{ThresholdImage} \) plus an offset are aggregated into the resulting region.

Typically, the threshold images are smoothed versions of the original image (e.g., by applying MeanImage, GaussImage, etc.). Then the effect of DynThreshold is similar to applying Threshold to a highpass-filtered version of the original image (see HighpassImage).

With DynThreshold contours of an object can be extracted, where the objects’ size (diameter) is determined by the mask size of the lowpass filter and the amplitude of the objects’ edges:

The larger the mask size is chosen, the larger the found regions get. As a rule of thumb, the mask size should be about twice the diameter of the objects to be extracted. It is important not to set the parameter Offset to zero because in this case too many small regions will be found (noise). Values between 5 and 40 are a sensible choice. The larger Offset is chosen, the smaller the extracted regions get.

All points of the input image fulfilling the above condition are stored jointly in one region. If necessary, the connected components can be obtained by calling Connection.

Attention

If Offset is chosen from \(-1\) to \(1\) usually a very noisy region is generated, requiring large storage. If Offset is chosen too large (> 60, say) it may happen that no points fulfill the threshold condition (i.e., an empty region is returned). If Offset is chosen too small (< -60, say) it may happen that all points fulfill the threshold condition (i.e., a full region is returned).

Parameter

- **OrigImage** (input iconic) image(-array) \( \rightarrow \text{HImageX} / \text{IOBJECTX} \)  
  Image to be segmented.

- **ThresholdImage** (input iconic) image(-array) \( \rightarrow \text{HImageX} / \text{IOBJECTX} \)  
  Image containing the local thresholds.

- **RegionDynThresh** (output iconic) region(-array) \( \rightarrow \text{HREGIONX} / \text{HUNTYPEOBJECTX} \)  
  Segmented regions.
Offset (input control) ........................ number  ~ VARIANT ( integer, real )
Offset added to ThresholdImage.

Default Value : 5.0
Value Suggestions : Offset ∈ {1.0, 3.0, 5.0, 7.0, 10.0, 20.0, 30.0}
Typical Range of Values : -255.0 ≤ Offset ≤ -255.0 (lin)
Minimal Value Step : 0.01
Recommended Value Step : 5
Restriction : ((-255 < Offset) ∧ (Offset < 255))

LightDark (input control) ........................ string  ~ String
Extract light, dark or similar areas?
Default Value : 'light'
Value List : LightDark ∈ {'dark', 'light', 'equal', 'not_equal'}

/* Looking for regions with the diameter D */
mean_image(Image,Mean,D*2+1,D*2+1)
dyn_threshold(Image,Mean,Seg:5,'light')
connection(Seg,Regions).

DynThreshold returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information
DynThreshold is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
MeanImage, SmoothImage, GaussImage

Possible Successor Functions
Connection, SelectShape, ReduceDomain, SelectGray, RankRegion, Dilation1, Opening

See Also
MeanImage, SmoothImage, GaussImage, Connection, RankRegion, Dilation1

Alternatives
CheckDifference, HighpassImage, SubImage, Threshold

Module
Region processing

[out] HRegionX RegionExpand HRegionX.ExpandGray ([in] HImageX Image,
[in] HRegionX ForbiddenArea, [in] VARIANT Iterations, [in] String Mode,
[in] VARIANT Threshold )

void HOperatorSetX.ExpandGray ([in] IObjectX Regions,
[in] IObjectX Image, [in] IObjectX ForbiddenArea,
[out] HUntypedObjectX RegionExpand, [in] VARIANT Iterations,

Fill gaps between regions (depending on gray value or color) or split overlapping regions.

ExpandGray closes gaps between the input regions, which resulted from the suppression of small regions in a segmentation operator, (mode 'image'), for example, or separates overlapping regions 'region'). Both uses result from the expansion of regions. The operator works by adding a one pixel wide “strip” to a region, in which the gray values or color are different from the gray values or color of neighboring pixels on the region’s border by at most Threshold (in each channel). For images of type 'cyclic' (e.g., direction images), also points with a gray value difference of at least 255 – Threshold are added to the output region.

The expansion takes place only in regions, which are designated as not “forbidden” (parameter ForbiddenArea). The number of iterations is determined by the parameter Iterations. By passing 'maximal', ExpandGray iterates until convergence, i.e., until no more changes occur. By passing 0 for this parameter,
all non-overlapping regions are returned. The two modes of operation ('image' and 'region') are different in the following ways:

'image' The input regions are expanded iteratively until they touch another region or the image border, or the expansion stops because of too high gray value differences. Because ExpandGray processes all regions simultaneously, gaps between regions are distributed evenly to all regions with a similar gray value. Overlapping regions are split by distributing the area of overlap evenly to both regions.

'region' No expansion of the input regions is performed. Instead, only overlapping regions are split by distributing the area of overlap evenly to regions having a matching gray value or color.

---

**Attention**

Because regions are only expanded into areas having a matching gray value or color, usually gaps will remain between the output regions, i.e., the segmentation is not complete.

---

### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Regions**     | (input iconic) region(-array) → HRegionX / IOBJECTX  
Regions for which the gaps are to be closed, or which are to be separated.                  |
| **Image**       | (input iconic) image → HImageX / IOBJECTX  
Image (possibly multi-channel) for gray value or color comparison.                             |
| **ForbiddenArea** | (input iconic) region(-array) → HRegionX / IOBJECTX  
Regions in which no expansion takes place.                                                  |
| **RegionExpand** | (output iconic) region(-array) → HRegionX / HUntypedOBJECTX  
Expanded or separated regions.                                                              |
| **Iterations**  | (input control) string → VARIANT (integer, string)  
Number of iterations.                                                                      |
| **Mode**        | (input control) string → String  
Expansion mode.                                                                           |
| **Threshold**   | (input control) integer(-array) → VARIANT (integer)  
Maximum difference between the gray value or color at the region’s border and a candidate for expansion. |

**Default Value**: 
- **Iterations**: 'maximal'
- **Mode**: 'image'
- **Threshold**: 32

**Value Suggestions**: 
- **Iterations**: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'maximal'
- **Mode**: 'image', 'region'
- **Threshold**: 5, 10, 15, 20, 25, 30, 40, 50

**Typical Range of Values**: 
- **Iterations**: 1 ≤ Iterations ≤ 1(lin)
- **Mode**: \{image\}
- **Threshold**: 1 ≤ Threshold ≤ 1(lin)

**Minimal Value Step**: 
- **Iterations**: 1
- **Mode**: 1
- **Threshold**: 1

**Recommended Value Step**: 
- **Iterations**: 1
- **Mode**: 1
- **Threshold**: 5

---

**Result**

ExpandGray always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>), the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>), and the behavior in case of an empty result region via SetSystem('storeEmptyRegion',<true/false>). If necessary, an exception handling is raised.

---

**Parallelization Information**

ExpandGray is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**: Connection, RegionGrowing, Pouring, ClassNdimNorm

**Possible Successor Functions**: SelectShape, See Also

---

**See Also**: ExpandGrayRef, ExpandRegion

---

Region processing
Fill gaps between regions (depending on gray value or color) or split overlapping regions.

ExpandGrayRef closes gaps between the input regions, which resulted from the suppression of small regions in a segmentation operator, (mode 'image'), for example, or separates overlapping regions 'region'). Both uses result from the expansion of regions. The operator works by adding a one pixel wide “strip” to a region, in which the gray values or color are different from a reference gray value or color by at most \( \text{Threshold} \) (in each channel). For images of type 'cyclic' (e.g., direction images), also points with a gray value difference of at least \( 255 - \text{Threshold} \) are added to the output region.

The expansion takes place only in regions, which are designated as not “forbidden” (parameter ForbiddenArea). The number of iterations is determined by the parameter Iterations. By passing 'maximal', ExpandGrayRef iterates until convergence, i.e., until no more changes occur. By passing 0 for this parameter, all non-overlapping regions are returned. The two modes of operation ('image' and 'region') are different in the following ways:

- **'image'** The input regions are expanded iteratively until they touch another region or the image border, or the expansion stops because of too high gray value differences. Because ExpandGrayRef processes all regions simultaneously, gaps between regions are distributed evenly to all regions with a similar gray value. Overlapping regions are split by distributing the area of overlap evenly to both regions.

- **'region'** No expansion of the input regions is performed. Instead, only overlapping regions are split by distributing the area of overlap evenly to regions having a matching gray value or color.

**Attention**
Because regions are only expanded into areas having a matching gray value or color, usually gaps will remain between the output regions, i.e., the segmentation is not complete.

---

**Parameter**

- **Regions** (input iconic) ........................................ region(-array) \( \sim HRegionX / IHObjectX \) Regions for which the gaps are to be closed, or which are to be separated.
- **Image** (input iconic) ........................................ image \( \sim HImageX / IHObjectX \) Image (possibly multi-channel) for gray value or color comparison.
- **ForbiddenArea** (input iconic) ........................... region \( \sim HRegionX / IHObjectX \) Regions in which no expansion takes place.
- **RegionExpand** (output iconic) ....................... region(-array) \( \sim HRegionX / HUntypedObjectX \) Expanded or separated regions.
- **Iterations** (input control) ................................. string \( \sim \text{VARIANT} \) (integer, string) Number of iterations.
  - **Default Value**: 'maximal'
  - **Value Suggestions**: \( \text{Iterations} \in \{ \text{maximal}, 1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 70, 100, 150, 200, 300, 500 \} \)
  - **Typical Range of Values**: \( 1 \leq \text{Iterations} \leq 1(\text{lin}) \)
  - **Minimal Value Step**: 1
  - **Recommended Value Step**: 1
- **Mode** (input control) ........................................ string \( \sim \text{String} \) Expansion mode.
  - **Default Value**: 'image'
  - **Value List**: Mode \( \in \{ \text{‘image’}, \text{‘region’} \} \)
**RefGray** (input control) .................................................. integer(-array) ~ VARIANT ( integer )
Reference gray value or color for comparison.

**Default Value :** 128

**Value Suggestions :** RefGray ∈ {1, 10, 20, 50, 100, 128, 200, 255}

**Typical Range of Values :** 1 ≤ RefGray ≤ 1(lin)

**Minimal Value Step :** 1

**Recommended Value Step :** 10

**Threshold** (input control) .................................................. integer(-array) ~ VARIANT ( integer )
Maximum difference between the reference gray value or color and a candidate for expansion.

**Default Value :** 32

**Value Suggestions :** Threshold ∈ {4, 10, 15, 20, 25, 30, 40}

**Typical Range of Values :** 1 ≤ Threshold ≤ 1(lin)

**Minimal Value Step :** 1

**Recommended Value Step :** 5

---

**ExpandGrayRef** always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem(‘noObjectResult’,<Result>), the behavior in case of an empty input region via SetSystem(‘emptyRegionResult’,<Result>), and the behavior in case of an empty result region via SetSystem(‘storeEmptyRegion’,<true/false>). If necessary, an exception handling is raised.

---

**Parallelization Information**

**ExpandGrayRef** is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**
Connection, RegionGrowing, Pouring, ClassNdimNorm

---

**Possible Successor Functions**
SelectShape

---

**See Also**
ExpandGray, ExpandRegion

---

**Region processing**

```c
[out] HRegionX RegionExpand
HImageX.ExpandLine ([in] long Index,

void HOperatorSetX.ExpandLine ([in] IHObjectX Image,
[out] HUntypedObjectX RegionExpand, [in] long Index, [in] String ExpandType,
```

Expand a region starting at a given line.

ExpandLine generates a region by expansion, starting at a given line (row or column).

---

**Parameter**

- **Image** (input iconic) .................................................. image(-array) ~ HImageX / IHObjectX ( byte )
Image to be segmented.

- **RegionExpand** (output iconic) ...................................... region(-array) ~ HRegionX / HUntypedObjectX
Extracted segments.

- **Index** (input control) ................................................ integer ~ long
Row or column index.

**Default Value :** 256

**Value Suggestions :** Index ∈ {16, 64, 128, 200, 256, 300, 400, 511}

**Restriction :** (Index ≥ 0)

- **ExpandType** (input control) ........................................... string ~ String
Stopping criterion.

**Default Value :** ‘gradient’

**Value List :** ExpandType ∈ {‘gradient’, ‘mean’}

---

RowColumn (input control) .......................................................... String
  Segmentation mode (row or column).
  Default Value : 'row'
  Value List: RowColumn ∈ { 'row', 'column' }  

Threshold (input control) ................................. number  ~  VARIANT ( integer, real )
  Threshold for the expansion.
  Default Value : 3.0
  Value Suggestions: Threshold∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 13.0, 17.0, 20.0, 30.0}  
  Typical Range of Values : 1.0 ≤ Threshold ≤ 1.0(lin)
  Minimal Value Step : 1.0
  Recommended Value Step : 1.0
  Restriction : ((Threshold ≥ 0.0) ∧ (Threshold ≤ 255.0))  

Parallelization Information
  ExpandLine is reentrant and automatically parallelized (on tuple level).
  Possible Predecessor Functions
    GaussImage, SmoothImage, AnisotropeDiff, MedianImage, AffineTransImage,
    RotateImage
  Possible Successor Functions
    Intersection, Opening, Closing
  Alternatives
    RegionGrowingMean, ExpandGray, ExpandGrayRef

Region processing

Fast selection of gray values within a given gray interval.
FastThreshold selects the pixels from the input image whose gray values \( g \) fulfill the following condition:

\[ \text{MinGray} \leq g \leq \text{MaxGray} \]

To reduce processing time, the selection is done in two steps: At first all pixels along rows with distances \( \text{MinHeight} \) are processed. In the next step the neighborhood (size \( \text{MinHeight} \times \text{MinHeight} \)) of all previously selected points are processed.

Parameter

Image (input iconic) ......................... image(-array)  ~  HImageX / IObjectX ( byte, direction, cyclic )
  Image to be thresholded.

Region (output iconic) .......................... region(-array)  ~  HRegionX / HUntypedObjectX
  Regions with gray values lying in the specified interval.

MinGray (input control) ............................ number  ~  VARIANT ( integer, real )
  Lower threshold for the gray values.
  Default Value : 128
  Value Suggestions: MinGray ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0}
  Typical Range of Values : 0.0 ≤ MinGray ≤ 0.0(lin)
  Minimal Value Step : 1
  Recommended Value Step : 5.0
MaxGray (input control) .............................................. number \sim \text{VARIANT ( integer, real )}
Upper threshold for the gray values.
Default Value : 255.0
Value Suggestions : MaxGray \in \{0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0\}
Typical Range of Values : 0.0 \leq \text{MaxGray} \leq 0.0(\text{lin})
Minimal Value Step : 1
Recommended Value Step : 5.0

MinHeight (input control) .......................................... number \sim \text{long}
Minimum height of objects to be extracted.
Default Value : 20
Value Suggestions : MinHeight \in \{5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 100\}
Typical Range of Values : 2 \leq \text{MinHeight} \leq 2(\text{lin})
Minimal Value Step : 1
Recommended Value Step : 2

FastThreshold returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no\_object\_result', 'empty\_region\_result', and 'store\_empty\_region' with SetSystem. If necessary, an exception is raised.

FastThreshold is \text{reentrant} and automatically \text{parallelized} (on \text{tuple level}).

Possible Predecessor Functions
HistoToThresh, MinMaxGray, SobelAmp, GaussImage, ReduceDomain, FillInterlace

Possible Successor Functions
Connection, Dilation1, Erosion1, Opening, Closing, RankRegion, ShapeTrans, Skeleton

See Also
Class2DimSup, HysteresisThreshold, DynThreshold

Alternatives
Threshold

Module

Region processing

Determine gray value thresholds from a histogram.
HistoToThresh determines gray value thresholds from a histogram for a segmentation of an image using Threshold. The thresholds returned are 0, 255, and all minima extracted from the histogram. Before the thresholds are determined the histogram is smoothed with a Gaussian.

Parameter

Histogramm (input control) ......................... histogram \sim \text{none}X / \text{VARIANT ( integer, real )}
Gray value histogram.

Sigma (input control) ........................................ number \sim \text{double}
Sigma for the Gaussian smoothing of the histogram.
Default Value : 2.0
Value Suggestions : Sigma \in \{0.5, 1.0, 2.0, 3.0, 4.0, 5.0\}
Typical Range of Values : 0.1 \leq \text{Sigma} \leq 0.1(\text{lin})
Minimal Value Step : 0.01
Recommended Value Step : 0.2

MinThreshold (output control) ..................... integer \sim \text{VARIANT ( integer )}
Minimum thresholds.

MaxThreshold (output control) ..................... integer \sim \text{VARIANT ( integer )}
Maximum thresholds.
**Parallelization Information**

HistoToThresh is reentrant and processed without parallelization.

**Possible Predecessor Functions**

GrayHisto

**Possible Successor Functions**

Threshold

**Alternatives**

AutoThreshold

**Module**

Region processing

```python
[out] HRegionX RegionHysteresis HImageX.HysteresisThreshold
([in] long Low, [in] long High, [in] long MaxLength )

void HOperatorSetX.HysteresisThreshold ([in] IHObjectX Image,
[out] HUntypedObjectX RegionHysteresis, [in] long Low, [in] long High,
[in] long MaxLength )
```

Perform a hysteresis threshold operation on an image.

HysteresisThreshold performs a hysteresis threshold operation (due to Canny) on an image. All points in the input image Image having a gray value larger than or equal to High are immediately accepted (“secure” points). Conversely, all points with gray values less than Low are immediately rejected. “Potential” points with gray values between both thresholds are accepted if they are connected to “secure” points by a path of “potential” points having a length of at most MaxLength points. This means that “secure” points influence their surroundings (hysteresis). The gray values of the input images remain unchanged. Only the regions of the image may get smaller.

**Parameter**

- **Image** (input iconic) . . . . . . . . . . . . . . . . . . . . . . image(-array) \(\sim\) HImageX / IHObjectX ( byte )
  Image to be segmented.

- **RegionHysteresis** (output iconic) . . . . . . . . . . . . . . . . region(-array) \(\sim\) HRegionX / HUntypedObjectX
  Segmentation result.

- **Low** (input control) . . . . . . . . . . . . . . . . . . . . . . integer \(\sim\) long
  Lower threshold for the gray values.
  Default Value : 30
  Value Suggestions : Low \(\in\) \{5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100\}
  Typical Range of Values : 0 \(\leq\) Low \(\leq\) 0 (lin)
  Minimal Value Step : 1
  Recommended Value Step : 5
  Restriction : \(((0 < \text{Low}) \land (\text{Low} < 255))\)

- **High** (input control) . . . . . . . . . . . . . . . . . . . . . . integer \(\sim\) long
  Lower threshold for the gray values.
  Default Value : 60
  Value Suggestions : High \(\in\) \{5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130\}
  Typical Range of Values : 0 \(\leq\) High \(\leq\) 0 (lin)
  Minimal Value Step : 1
  Recommended Value Step : 5
  Restriction : \(((0 < \text{High}) \land (\text{High} < 255)) \land (\text{High} > \text{Low})\)

- **MaxLength** (input control) . . . . . . . . . . . . . . . . . integer \(\sim\) long
  Maximum length of a path of “potential” points to reach a “secure” point.
  Default Value : 10
  Value Suggestions : MaxLength \(\in\) \{1, 2, 3, 5, 7, 10, 12, 14, 17, 20, 25, 30, 35, 40, 50\}
  Typical Range of Values : 1 \(\leq\) MaxLength \(\leq\) 1 (lin)
  Minimal Value Step : 1
  Recommended Value Step : 5
  Restriction : \((\text{MaxLength} > 1)\)
HysteresisThreshold returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information
HysteresisThreshold is reentrant and automatically parallelized (on tuple level).

See Also
EdgesImage, SobelAmp, BackgroundSeg

Alternatives
DynThreshold, Threshold, Class2DimSup

Bibliography

Module
Region processing

void HRegionX.LearnNdimBox ([in] HRegionX Background, [in] HImageX MultiChannelImage, [in] HClassBoxX ClassifHandle )

void HClassBoxX.LearnNdimBox ([in] HRegionX Foreground, [in] HRegionX Background, [in] HImageX MultiChannelImage )


Train the current classifier using a multi-channel image.
LearnNdimBox trains the classifier ClassifHandle with the gray values of MultiChannelImage using the points in Foreground as training sample. The points in Background are to be rejected by the classifier. The classifier trained thus can be used in LearnNdimBox to segment multi-channel images. Foreground are the points that have to be found, Background contains the points which shall not be found.
Each pixel is trained once during the training process. For points in Foreground the class "0" is used, while for Background "1" is used. Pixels are trained by alternating points from Foreground with points from Background. If one region is smaller than the other, pixels are taken cyclically from the smaller region until the larger region is exhausted. LearnNdimBox later accepts only points which can be classified into class "0".

Attention
All channels must be of the same type and have the same size.

Parameter

- Foreground (input iconic) . . . . . . . . . . . . . . . . region(-array) ~ HRegionX / IOBJECT
  Foreground pixels to be trained.
- Background (input iconic) . . . . . . . . . . . . . . . . region(-array) ~ HRegionX / IOBJECT
  Background pixels to be trained (rejection class).
- MultiChannelImage (input iconic) . . . . multichannel-image-array ~ HImageX / IOBJECT (byte, int1, int2, int4, real, direction, cyclic)
  Multi-channel training image.
- ClassifHandle (input control) . . . . . . . . . . . . . . . . . . . . . . class_box ~ HClassBoxX / long
  Classifier’s handle number.

Result
ClassNdimBox returns TRUE if all parameters are correct and there is an active classifier. The behavior with respect to the input images can be determined by setting the values of the flags 'no_object_result' and 'empty_region_result' with SetSystem. If necessary, an exception is raised.

Parallelization Information
LearnNdimBox is local and processed completely exclusively without parallelization.
Construct clusters for \texttt{ClassNdimNorm}.

\texttt{LearnNdimNorm} generates classification clusters from the region \texttt{Foreground} and the corresponding gray values in the multi-channel image \texttt{Image}, which can be used in \texttt{ClassNdimNorm}. \texttt{Background} determines a class of pixels not to be found in \texttt{ClassNdimNorm}. This parameter may be empty (empty tuple).

The parameter \texttt{Distance} determines the maximum distance \texttt{Radius} of the clusters. It describes the minimum distance between two cluster centers. If the parameter \texttt{Distance} is small the (small) hyper-cubes or hyperspheres can approximate the feature space well. Simultaneously the runtime during classification increases.

The ratio of the number of pixels in a cluster to the total number of pixels (in percent) must be larger than the value of \texttt{MinNumberPercent}, otherwise the cluster is not returned. \texttt{MinNumberPercent} serves to eliminate outliers in the training set. If it is chosen too large many clusters are suppressed.

Two different clustering procedures can be selected: The minimum distance algorithm (n-dimensional hyper-spheres) and the maximum algorithm (n-dimensional hyper-cubes) for describing the pixels of the image to classify in the n-dimensional histogram (parameter \texttt{Metric}). The Euclidean metric usually yields the better results, but takes longer to compute. The parameter \texttt{Quality} returns the quality of the clustering. It is a measure of overlap between the rejection class and the classificator classes. Values larger than 0 denote the corresponding ratio of overlap. If no rejection region is given, its value is set to 1. The regions in \texttt{Background} do not influence on the clustering. They are merely used to check the results that can be expected.

### Parameters

- **\texttt{Foreground}** (input iconic) \texttt{region-array} \texttt{HRegionX / IObjectX} 
  Foreground pixels to be trained.
- **\texttt{Background}** (input iconic) \texttt{region-array} \texttt{HRegionX / IObjectX} 
  Background pixels to be trained (rejection class).
- **\texttt{Image}** (input iconic) \texttt{image-array} \texttt{HImageX / IObjectX \(\text{byte, int1, int2, int4, real}\)} 
  Multi-channel training image.
- **\texttt{Metric}** (input control) \texttt{string} 
  Metric to be used.
  Default Value : 'euclid'
  Value List : ['euclid', 'maximum']
- **\texttt{Distance}** (input control) \texttt{number} \texttt{VARIANT \{ integer, real \}} 
  Maximum cluster radius.
  Default Value : 10.0
  Value Suggestions : \texttt{Distance} \texttt{\(\in\)} \{1.0, 2.0, 3.0, 4.0, 6.0, 8.0, 10.0, 13.0, 17.0, 24.0, 30.0, 40.0\}
  Typical Range of Values : \(0.0 \leq \texttt{Distance} \leq 0.0\) (lin)
  Minimal Value Step : 0.01
  Recommended Value Step : 1.0
  Restriction : (\texttt{Radius} > 0.0)

---

**Possible Predecessor Functions**
- \texttt{CreateClassBox, DrawRegion}

**Possible Successor Functions**
- \texttt{ClassNdimBox, DescriptClassBox}

**Alternatives**
- \texttt{LearnClassBox, Module}

**Module**
- Region processing

```plaintext

```
**MinNumberPercent** (input control) ................................................. number \( \sim \) VARIANT (integer, real)

The ratio of the number of pixels in a cluster to the total number of pixels (in percent) must be larger than MinNumberPercent (otherwise the cluster is not output).

**Default Value:** 0.01

**Value Suggestions:** MinNumberPercent \( \in \{0.001, 0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}\)

**Typical Range of Values:** 0.0 \( \leq \) MinNumberPercent \( \leq 0.0\) (lin)

**Minimal Value Step:** 0.01

**Recommended Value Step:** 0.1

**Restriction:** \((0 \leq \text{MinNumberPercent}) \land (\text{MinNumberPercent} \leq 100)\)

**Radius** (output control) ......................................................... real \( \sim \) VARIANT (real)

Cluster radii of half edge lengths.

**Center** (output control) .......................................................... real \( \sim \) VARIANT (real)

Coordinates of all cluster centers.

**Quality** (output control) ......................................................... real \( \sim \) double

Overlap of the rejection class with the classified objects (1: no overlapping).

**Restriction:** \((0 \leq \text{Quality}) \land (\text{Quality} \leq 1)\)

---

LearnNdimNorm returns TRUE if all parameters are correct. The behavior with respect to the input images can be determined by setting the values of the flags **'no object result'** and **'empty region result'** with SetSystem. If necessary, an exception is raised.

---

**Parallelization Information**

LearnNdimNorm is local and processed completely exclusively without parallelization.

---

**Possible Predecessor Functions**

MinMaxGray, SobelAmp, GaussImage, ReduceDomain, DiffOfGauss

---

**Possible Successor Functions**

ClassNdimNorm, Connection, Dilation1, Erosion1, Opening, Closing, RankRegion, ShapeTrans, Skeleton

---

**See Also**

ClassNdimNorm, ClassNdimBox, Histo2Dim

---

**Bibliography**

P. Haberacker, "Digitale Bildverarbeitung", Hanser-Studienbücher, München, Wien, 1987

---

**Module**

Region processing

---

```c
[out] HRegionX LocalMaxima HImageX.LocalMax ( )

void HOperatorSetX.LocalMax ([in] IObjectX Image, [out] HUntypedObjectX LocalMaxima )
```

Detect all local maxima in an image.

LocalMax extracts all points in an image having a gray value larger than the gray value of all its neighbors. The neighborhood used can be set by SetSystem (::’neighborhood’,<4/8>).

---

**Parameter**

- **Image** (input iconic) ......................... image(-array) \( \sim \) HImageX / IObjectX (byte, int1, int2, int4, real)
  
  Image to be processed.

- **LocalMaxima** (output iconic) ......................... region(-array) \( \sim \) HRegionX / HUntypedObjectX
  
  Extracted local maxima as regions.

**Parameter Number:** **LocalMaxima = Image**

---

**Parallelization Information**

LocalMax is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

GaussImage, SmoothImage

---

**Possible Successor Functions**

GetRegionPoints, Connection
Suppress non-maximum points on an edge.

NonmaxSuppressionAmp suppresses in the regions of the image ImgAmp all points whose gray values are not local (directed) maxima. In contrast to NonmaxSuppressionDir, a direction image is not needed. Two modes of operation can be selected:

'hvms' A point is labeled as a local maximum if its gray value is larger than or equal to the gray values within a search space of ±5 pixels, either horizontally or vertically. Non-maximum points are removed from the region, gray values remain unchanged.

'loc_max' A point is labeled as a local maximum if its gray value is larger than or equal to the gray values of its eight neighbors.

Parameter

- **ImgAmp** (input iconic) image(-array) ~ HImageX / IObjectX (byte)
  Amplitude (gradient magnitude) image.
- **ImageResult** (output iconic) image(-array) ~ HImageX / HUntypedObjectX (byte)
  Image with thinned edge regions.
- **Mode** (input control) string ~ String
  Select horizontal/vertical or undirected NMS.
  Default Value: 'hvms'
  Value List: Mode ∈ {'hvms', 'loc_max'}

Result

NonmaxSuppressionAmp returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information

NonmaxSuppressionAmp is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessor Functions

SobelAmp

Possible Successor Functions

Threshold, HysteresisThreshold

See Also

Region processing

Bibliography


Suppress non-maximum points on an edge.

NonmaxSuppressionDir suppresses in the regions of the image ImgAmp all points whose gray values are not local (directed) maxima. ImgDir is a direction image giving the direction perpendicular for the local maximum (Unit: 2 degrees, i.e., 50 degrees are coded as 25 in the image). Such images are returned, for example, by EdgesImage. Two modes of operation can be selected:

'nms' Each point in the image is tested whether its gray value is a local maximum perpendicular to its direction. In this mode only the two neighbors closest to the given direction are examined. If one of the two gray values is greater than the gray of the point to be tested, it is suppressed (i.e., removed from the input region. The corresponding gray value remains unchanged).

'inms' Like 'nms' However, the two gray values for the test are obtained by interpolation from four adjacent points.

Parameter

- **ImgAmp** (input iconic)..........................image(-array) ~ HImageX / IHObjectX Amplitude (gradient magnitude) image.
- **ImgDir** (input iconic)..........................image(-array) ~ HImageX / IHObjectX Direction image.
- **ImageResult** (output iconic)....................image(-array) ~ HImageX / HUntypedObjectX Image with thinned edge regions.
- **Mode** (input control)............................string ~ String Select non-maximum-suppression or interpolating NMS.
  
  Default Value: 'nms'
  
  Value List: Mode ∈ {'nms', 'inms'}

Result

NonmaxSuppressionDir returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information

NonmaxSuppressionDir is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessor Functions

EdgesImage, SobelDir

Possible Successor Functions

Threshold, HysteresisThreshold

See Also

Skeleton

Alternatives

NonmaxSuppressionAmp, GraySkeleton, GrayDilationRect

Bibliography


Module

Region processing
Detect all gray value plateaus.

Plateaus extracts all points with a gray value greater or equal to the gray value of its neighbors (8-connectivity). Each maximum is returned as a separate region.

Parameter

- **Image** (input iconic) .......................... image(-array) \(\sim\) HImageX / IHObjectX (byte, int1, int2, int4, real)
  Image to be processed.
- **Plateaus** (output iconic) ............................ region \(\sim\) HRegionX / HUntypedObjectX
  Extracted plateaus as regions (one region for each plateau).

Parallelization Information

Plateaus is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

GaussImage, SmoothImage

Possible Successor Functions

AreaCenter, GetRegionPoints, SelectShape

See Also

Monotony, TopographicSketch, CornerResponse, TextureLaws

Alternatives

PlateausCenter, GraySkeleton, NonmaxSuppressionAmp, LocalMax

Module

Region processing

Detect the centers of all gray value plateaus.

PlateausCenter extracts all points with a gray value greater or equal to the gray value of its neighbors (8-connectivity). If more than one of these points are connected (plateau), their center of gravity is returned. Each maximum is returned as a separate region.

Parameter

- **Image** (input iconic) .......................... image(-array) \(\sim\) HImageX / IHObjectX (byte, int1, int2, int4, real)
  Image to be processed.
- **Plateaus** (output iconic) ............................ region \(\sim\) HRegionX / HUntypedObjectX
  Centers of gravity of the extracted plateaus as regions (one region for each plateau).

Parallelization Information

PlateausCenter is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

GaussImage, SmoothImage

Possible Successor Functions

AreaCenter, GetRegionPoints, SelectShape

See Also

Monotony, TopographicSketch, CornerResponse, TextureLaws

Alternatives

Plateaus, GraySkeleton, NonmaxSuppressionAmp, LocalMax

Module

Region processing
Segment an image by “pouring water” over it.

Pouring regards the input image as a “mountain range.” Larger gray values correspond to mountain peaks, while smaller gray values correspond to valley bottoms. Pouring segments the input image in several steps. First the local maxima are extracted, i.e., pixels which either alone or in the form of an extended plateau have larger gray values than their immediate neighbors (in 4-connectivity). In the next step, the maxima thus found are the starting points for an expansion until “valley bottoms” are reached. The expansion is done as long as there are chains of pixels in which the gray value gets smaller (like water running downhill from the maxima in all directions). Again, 4-connectivity is used, but with a weaker condition (smaller or equal). This means that points at valley bottoms may belong to more than one maximum. These areas are at first not assigned to a region, but rather are split among all competing segments in the last step. The split is done by a uniform expansion of all involved segments, until all ambiguous pixels were assigned. The parameter Mode determines which steps are executed. The following values are possible:

'all' This is the normal mode of operation. All steps of the segmentation are performed. The regions are assigned to maxima, and overlapping regions are split.

'maxima' The segmentation only extracts the local maxima of the input image. No corresponding regions are extracted.

'regions' The segmentation extracts the local maxima of the input image and the corresponding regions, which are uniquely determined. Areas which could be assigned to more than one maximum are not split.

In order to prevent the algorithm from splitting a uniform background, which is different from the rest of the image, the parameters MinGray and MaxGray determine gray value thresholds for regions in the image which should be regarded as background. All parts of the image having a gray value smaller than MinGray or larger than MaxGray are disregarded for the extraction of the maxima as well as for the assignment of regions. For a complete segmentation of the image, MinGray = 0 and MaxGray = 255 should be selected. In any case, MinGray ≤ MaxGray must be observed.

**Parameter**

- **Image** (input iconic) ......................... image ~ HImageX / IObjectX (byte) Image to be segmented.
- **Regions** (output iconic) .................... region ~ HRegionX / HUntypedObjectX Segmentation result.
- **Mode** (input control) ......................... string ~ String Mode of operation.
  - Default Value : 'all'
  - Value List : Mode ∈ \{ 'all', 'maxima', 'regions' \}
- **MinGray** (input control) ....................... integer ~ long All gray values smaller than this threshold are disregarded.
  - Default Value : 0
  - Value Suggestions : MinGray ∈ \{ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110 \}
  - Typical Range of Values : 0 ≤ MinGray ≤ 0(\text{lin})
  - Minimal Value Step : 1
  - Recommended Value Step : 10
  - Restriction : (MinGray ≥ 0)
- **MaxGray** (input control) ....................... integer ~ long All gray values larger than this threshold are disregarded.
  - Default Value : 255
  - Value Suggestions : MaxGray ∈ \{ 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, ... }

```plaintext

```


Typical Range of Values: $0 \leq \text{MaxGray} \leq 0(\text{lin})$
Minimal Value Step: 1
Recommended Value Step: 10
Restriction: $(\text{MaxGray} \leq 255) \land (\text{MaxGray} > \text{MinGray})$

Example

```c
/* Segmentation of a filtering Image */
read_image(Image, 'br2')
mean_image(Image, Mean, 11, 11)
pouring(Mean, Seg, 'all', 0, 255)
disp_image(Mean, WindowHandle)
set_colored(WindowHandle, 12)
disp_region(Seg, WindowHandle).

/* Segmentation of a Image with masking of a dark backround */
read_image(Image, 'hand')
mean_image(ImageMean, 15, 15)
pouring(Mean, Seg, 'all', 40, 255)
disp_image(Mean, WindowHandle)
set_colored(WindowHandle, 12)
disp_region(Seg, WindowHandle).

/* Segmentation of a histogram */
read_image(Image, 'affe')
texture_laws(Image, Texture, 'el', 2, 5)
draw_region(Region, draw_region)
reduce_domain(Texture, Region, Testreg)
histo_2dim(Testreg, Texture, Region, Histo)
pouring(Histo, Seg, 'all', 0, 255).
```

Result
Pouring usually returns the value TRUE. If necessary, an exception is raised.

Parallelization Information
Pouring is processed under mutual exclusion against itself and without parallelization.

Possible Predecessor Functions
GaussImage, SmoothImage

See Also
Histo2Dim, ExpandRegion, ExpandGray, ExpandGrayRef

Alternatives
Watersheds, LocalMax

Module
Region processing

```
[out] HRegionX Regions HImageX.Regiongrowing ([in] long Row,

void HOperatorSetX.Regiongrowing ([in] IHObjectX Image,
[out] HUntypedObjectX Regions, [in] long Row, [in] long Column,
```

Segment an image using regiongrowing.
Regiongrowing segments images into regions of the same intensity — rastered into rectangles of size $Row \times Column$. In order to decide whether two adjacent rectangles belong to the same region only the gray value of their
center points is used. If the gray value difference is less than or equal to Tolerance the rectangles are merged into one region.

If \( g_1 \) und \( g_2 \) are two gray values to be examined, they are merged into the same region if:

\[
|g_1 - g_2| < \text{Tolerance}
\]

For images of type 'cyclic', the following formulas are used:

\[
(|g_1 - g_2| < \text{Tolerance}) \land (|g_1 - g_2| \leq 127)
\]

\[
(256 - |g_1 - g_2| < \text{Tolerance}) \land (|g_1 - g_2| > 127)
\]

For rectangles larger than one pixel, usually the images should be smoothed with a lowpass filter with a size of at least Row \times Column before calling RegionGrowing (so that the gray values at the centers of the rectangles are “representative” for the whole rectangle). If the image contains little noise and the rectangles are small, the smoothing can be omitted in many cases. This, of course, makes the whole procedure faster.

The resulting regions are collections of rectangles of the chosen size Row \times Column. Only regions containing at least MinSize points are returned.

RegionGrowing is a very fast operation, and thus suited for time-critical applications.

### Parameter

- **Image** (input iconic) ...................... image(-array) \( \sim \) HImageX / IHObjectX (byte, int1, int2, int4, cyclic, real)
  
  Image to be segmented.

- **Regions** (output iconic) ....................... region \( \sim \) HRegionX / HUntypedObjectX
  
  Extracted segments.

- **Row** (input control) .......................... extent.y \( \sim \) long
  
  Vertical distance between tested pixels (height of the raster).
  
  **Default Value:** 3
  
  **Value Suggestions:** Row \( \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21\}
  
  **Typical Range of Values:** 1 \( \leq \) Row \( \leq \) l(lin)
  
  **Minimal Value Step:** 2
  
  **Recommended Value Step:** 2
  
  **Restriction:** (Row \( \land \) odd)

- **Column** (input control) .......................... extent.x \( \sim \) long
  
  Horizontal distance between tested pixels (height of the raster).
  
  **Default Value:** 3
  
  **Value Suggestions:** Column \( \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21\}
  
  **Typical Range of Values:** 1 \( \leq \) Column \( \leq \) l(lin)
  
  **Minimal Value Step:** 2
  
  **Recommended Value Step:** 2
  
  **Restriction:** (Column \( \land \) odd)

- **Tolerance** (input control) .......................... number \( \sim \) VARIANT (integer, real)
  
  Points with a gray value difference less than or equal to tolerance are accumulated into the same object.
  
  **Default Value:** 6.0
  
  **Value Suggestions:** Tolerance \( \in \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 14.0, 18.0, 25.0\}
  
  **Typical Range of Values:** 1.0 \( \leq \) Tolerance \( \leq \) 1.0(lin)
  
  **Minimal Value Step:** 0.01
  
  **Recommended Value Step:** 1.0
  
  **Restriction:** ((0 \( \leq \) Tolerance) \( \land \) (Tolerance \( < \) 127))

- **MinSize** (input control) .......................... integer \( \sim \) long
  
  Minimum size of the output regions.
  
  **Default Value:** 100
  
  **Value Suggestions:** MinSize \( \in \{1, 5, 10, 20, 50, 100, 200, 500, 1000\}
  
  **Typical Range of Values:** 1 \( \leq \) MinSize \( \leq \) l(lin)
  
  **Minimal Value Step:** 1
  
  **Recommended Value Step:** 5
  
  **Restriction:** (MinSize \( \geq \) 1)
Example

read_image(Image, 'fabrik')
mean_image(Image, Mean, Row, Column)
regiongrowing(Mean, Result, Row, Column, 6.0, 100).

Result
Regiongrowing returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information
Regiongrowing is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
MeanImage, GaussImage, SmoothImage, MedianImage, AnisotropeDiff

Possible Successor Functions
SelectShape, ReduceDomain, SelectGray

Alternatives
RegiongrowingN, RegiongrowingMean, LabelToRegion

Module
Region processing

Regiongrowing using mean gray values.
RegiongrowingMean performs a regiongrowing using mean gray values of a region, starting from points given by StartRows and StartColumns. At any point in the process the mean gray value of the current region is calculated. Gray values at the boundary of the region are added to the region if they differ from the current mean by less than Tolerance. Regions smaller than MinSize are suppressed.

If no starting points are given (empty tuples), the expansion process starts at the upper leftmost point, and is continued with the first unprocessed point after a region has been created.

Parameter

- **Image** (input iconic) .............................. image(-array) \(\rightsquigarrow\) HImageX / IOBJECTX (byte, int4)
  Image to be segmented.

- **Regions** (output iconic) ........................... region \(\rightsquigarrow\) HRegionX / HUntypedObjectX
  Extracted segments.

- **StartRows** (input control) ........................ point.y(-array) \(\rightsquigarrow\) VARIANT (integer)
  Row coordinates of the starting points.

  Default Value: []
  Typical Range of Values: \(0 \leq \text{StartRows} \leq 0\) (lin)
  Minimal Value Step: 1
  Recommended Value Step: 1

- **StartColumns** (input control) ........................ point.x(-array) \(\rightsquigarrow\) VARIANT (integer)
  Column coordinates of the starting points.

  Default Value: []
  Typical Range of Values: \(0 \leq \text{StartColumns} \leq 0\) (lin)
  Minimal Value Step: 1
  Recommended Value Step: 1
**Tolerance** (input control) .......................................................... number ~ double  
Maximum deviation from the mean.  
Default Value: 5.0  
Value Suggestions: Tolerance ∈ {0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 15.0, 17.0, 20.0, 25.0, 30.0, 40.0}  
Typical Range of Values: 0.1 ≤ Tolerance ≤ 0.1(lin)  
Minimal Value Step: 0.01  
Recommended Value Step: 1.0  
Restriction: (Tolerance > 0.0)  

**MinSize** (input control) .......................................................... integer ~ long  
Minimum size of a region.  
Default Value: 100  
Value Suggestions: MinSize ∈ {0, 10, 30, 50, 100, 500, 1000, 2000}  
Typical Range of Values: 0 ≤ MinSize ≤ 0(lin)  
Minimal Value Step: 1  
Recommended Value Step: 100  
Restriction: (MinSize ≥ 0)  

Region growing Mean returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.  

**Parallelization Information**  
Region growing Mean is reentrant and automatically parallelized (on tuple level).  

**Possible Predecessor Functions**  
GaussImage, SigmaImage, AnisotropeDiff, MedianImage  

**Possible Successor Functions**  
SelectShape, ReduceDomain, Opening, ExpandRegion  

**Alternatives**  
Region growing, Region growing N  

Region processing  

```c
```

```c
```

Region growing for multi-channel images.  

Region growing N performs a multi-channel region growing. The n channels give rise to an n-dimensional feature vector. Neighboring points are aggregated into the same region if the difference of their feature vectors with respect to the given metric lies in the interval [MinTolerance, MaxTolerance]. Only 4-connected neighbors are examined. The following metrics can be used:  

Let $g_A$ denote the gray value in the feature vector A at point a of the image, and likewise be $g_B$ the gray value in the feature vector B at point a neighboring point b. Let $g(d)$ be the gray value with index d. Furthermore, let MinT denote MinTolerance and MaxT denote MaxTolerance.

**'1-norm':** Sum of absolute values  
\[
MinT \leq \frac{1}{n} \sum |g_A - g_B| \leq MaxT
\]

**'2-norm':** Euclidian distance  
\[
MinT \leq \sqrt{\frac{\sum (g_A - g_B)^2}{n}} \leq MaxT
\]
'3-norm': p - Norm with p = 3
\[ MinT \leq \sqrt[3]{\frac{\sum (g_A - g_B)^3}{n}} \leq MaxT \]

'4-norm': p - Norm with p = 4
\[ MinT \leq \sqrt[4]{\frac{\sum (g_A - g_B)^4}{n}} \leq MaxT \]

'n-norm': Minkowski distance
\[ MinT \leq \sqrt[n]{\frac{\sum (g_A - g_B)^n}{n}} \leq MaxT \]

'max-diff': Supremum distance
\[ MinT \leq \max \{ |g_A - g_B| \} \leq MaxT \]

'min-diff': Infimum distance
\[ MinT \leq \min \{ |g_A - g_B| \} \leq MaxT \]

'variance': Variance of gray value differences
\[ MinT \leq Var(g_A - g_B) \leq MaxT \]

'dot-product': Dot product
\[ MinT \leq \frac{1}{n} \sqrt{\sum (g_A g_B)} \leq MaxT \]

'correlation': Correlation
\[ m_A = \frac{1}{n} \sum g_A \]
\[ Var_A = \frac{1}{n} \sqrt{\sum (g_A - m_A)^2} \]
\[ m_B = \frac{1}{n} \sum g_B \]
\[ Var_B = \frac{1}{n} \sqrt{\sum (g_B - m_B)^2} \]
\[ MinT \leq \frac{1}{n^2} \sum \frac{(g_A - m_A)(g_B - m_B)}{(Var_A Var_B)} \leq MaxT \]

'mean-diff': Difference of arithmetic means
\[ a = \frac{1}{n} \sum g_A \]
\[ b = \frac{1}{n} \sum g_B \]
\[ MinT \leq |a - b| \leq MaxT \]

'mean-ratio': Ratio of arithmetic means
\[ a = \frac{1}{n} \sum g_A \]
\[ b = \frac{1}{n} \sum g_B \]
\[ MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT \]

'length-diff': Difference of the vector lengths
\[ a = \sqrt{\frac{\sum g_A^2}{n}} \]
\[ b = \sqrt{\frac{\sum g_B^2}{n}} \]
\[ MinT \leq |a - b| \leq MaxT \]

HALCON 6.0
'length-ratio': Ratio of the vector lengths

\[ a = \sqrt{\frac{\sum g_A^2}{n}} \]
\[ b = \sqrt{\frac{\sum g_B^2}{n}} \]
\[ \text{MinT} \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq \text{MaxT} \]

'n-norm-ratio': Ratio of the vector lengths w.r.t the p-norm with \( p = n \)

\[ a = \sqrt[n]{\frac{\sum g_A^p}{n}} \]
\[ b = \sqrt[n]{\frac{\sum g_B^p}{n}} \]
\[ \text{MinT} \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq \text{MaxT} \]

'gray-max-diff': Difference of the maximum gray values

\[ a = \max \{ |g_A| \} \]
\[ b = \max \{ |g_B| \} \]
\[ \text{MinT} \leq |a - b| \leq \text{MaxT} \]

'gray-max-ratio': Ratio of the maximum gray values

\[ a = \max \{ |g_A| \} \]
\[ b = \max \{ |g_B| \} \]
\[ \text{MinT} \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq \text{MaxT} \]

'gray-min-diff': Difference of the minimum gray values

\[ a = \min \{ |g_A| \} \]
\[ b = \min \{ |g_B| \} \]
\[ \text{MinT} \leq |a - b| \leq \text{MaxT} \]

'gray-min-ratio': Ratio of the minimum gray values

\[ a = \min \{ |g_A| \} \]
\[ b = \min \{ |g_B| \} \]
\[ \text{MinT} \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq \text{MaxT} \]

'variance-diff': Difference of the variances over all gray values (channels)

\[ \text{MinT} \leq |\text{Var}(g_A) - \text{Var}(g_B)| \leq \text{MaxT} \]

'variance-ratio': Ratio of the variances over all gray values (channels)

\[ \text{MinT} \leq \frac{\text{Var}(g_B)}{\text{Var}(g_A)} \leq \text{MaxT} \]
'mean-abs-diff': Difference of the sum of absolute values over all gray values (channels)

\[ a = \sum_{d,k,k<d} |g_A(d) - g_A(k)| \]

\[ b = \sum_{d,k,k<d} |g_B(d) - g_B(k)| \]

\[ \text{MinT} \leq \frac{|a - b|}{\text{Anzahl der Summen}} \leq \text{MaxT} \]

'mean-abs-ratio': Ratio of the sum of absolute values over all gray values (channels)

\[ a = \sum_{d,k,k<d} |g_A(d) - g_A(k)| \]

\[ b = \sum_{d,k,k<d} |g_B(d) - g_B(k)| \]

\[ \text{MinT} \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq \text{MaxT} \]

'max-abs-diff': Difference of the maximum distance of the components

\[ a = \max \{ g_A(d), g_A(k) \} \]

\[ b = \max \{ g_B(d), g_B(k) \} \]

\[ \text{MinT} \leq |a - b| \leq \text{MaxT} \]

'max-abs-ratio': Ratio of the maximum distance of the components

\[ a = \max \{ g_A(d), g_A(k) \} \]

\[ b = \max \{ g_B(d), g_B(k) \} \]

\[ \text{MinT} \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq \text{MaxT} \]

'min-abs-diff': Difference of the minimum distance of the components

\[ a = \min \{ g_A(d), g_A(k) \}, k < d \]

\[ b = \min \{ g_B(d), g_B(k) \}, k < d \]

\[ \text{MinT} \leq |a - b| \leq \text{MaxT} \]

'min-abs-ratio': Ratio of the minimum distance of the components

\[ a = \min \{ g_A(d), g_A(k) \}, k < d \]

\[ b = \min \{ g_B(d), g_B(k) \}, k < d \]

\[ \text{MinT} \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq \text{MaxT} \]

'plane': The following has to hold for all \( d_1, d_2 \in [1, n] \):

\[ g_A(d_1) > g_A(d_2) \Rightarrow g_B(d_1) > g_B(d_2) \]

\[ g_A(d_1) < g_A(d_2) \Rightarrow g_B(d_1) < g_B(d_2) \]

Regions with less than MinSize are suppressed.
Parameter

- **MultiChannelImage** (input iconic) image(-array) \sim HImageX / HObjectX (byte)
  Multi-channel image to be segmented.

- **Regions** (output iconic) region \sim HRegionX / HUntypedObjectX
  Segmented regions.

- **Metric** (input control) string \sim String
  Metric for the distance of the feature vectors.
  Default Value: 2-norm

- **MinTolerance** (input control) number \sim VARIANT (integer, real)
  Lower threshold for the features' distance.
  Default Value: 0.0
  Value Suggestions: MinTolerance \in \{ 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0, 25.0, 30.0 \}
  Typical Range of Values: 0.0 \leq \text{MinTolerance} \leq 0.0(\text{lin})
  Minimal Value Step: 0.01
  Recommended Value Step: 1.0

- **MaxTolerance** (input control) number \sim VARIANT (integer, real)
  Upper threshold for the features' distance.
  Default Value: 20.0
  Value Suggestions: MaxTolerance \in \{ 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0, 25.0, 30.0 \}
  Typical Range of Values: 0.0 \leq \text{MaxTolerance} \leq 0.0(\text{lin})
  Minimal Value Step: 0.01
  Recommended Value Step: 1.0

- **MinSize** (input control) integer \sim long
  Minimum size of the output regions.
  Default Value: 30
  Value Suggestions: MinSize \in \{ 1, 10, 25, 50, 100, 200, 500, 1000 \}
  Typical Range of Values: 1 \leq \text{MinSize} \leq 1(\text{lin})
  Minimal Value Step: 1
  Recommended Value Step: 5

Result

RegiongrowingN returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information

RegiongrowingN is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

- Compose2, Compose3

See Also

- Regiongrowing

Alternatives

- Class2DimSup, ClassNdimNorm, ClassNdimBox

Module

Region processing
Select gray values lying within an interval.
Threshold selects the pixels from the input image whose gray values \( g \) fulfill the following condition:

\[
\text{MinGray} \leq g \leq \text{MaxGray}.
\]

All points of an image fulfilling the condition are returned as one region. If more than one gray value interval is passed (tuples for \( \text{MinGray} \) and \( \text{MaxGray} \)), one separate region is returned for each interval.

---

**Parameter**

- **Image** (input iconic) ..... image(-array) \( \leadsto \) HImageX / IObjectX (byte, direction, cyclic, int2, int4, real)
  Image to be thresholded.
- **Region** (output iconic) ......................region(-array) \( \leadsto \) HRegionX / HUntypedObjectX
  Regions with gray values lying in the specified interval.
- **MinGray** (input control) ................... number(-array) \( \leadsto \) VARIANT (integer, real)
  Lower threshold for the gray values.
  **Default Value:** 128.0
  **Value Suggestions:** MinGray \( \in \{ 0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0 \} \)

  **Typical Range of Values:** \( 0.0 \leq \text{MinGray} \leq 0.0(\text{lin}) \)
  **Minimal Value Step:** 0.01
  **Recommended Value Step:** 5.0

- **MaxGray** (input control) ................... number(-array) \( \leadsto \) VARIANT (integer, real)
  Upper threshold for the gray values.
  **Default Value:** 255.0
  **Value Suggestions:** MaxGray \( \in \{ 0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0 \} \)

  **Typical Range of Values:** \( 0.0 \leq \text{MaxGray} \leq 0.0(\text{lin}) \)
  **Minimal Value Step:** 0.01
  **Recommended Value Step:** 5.0
  **Restriction:** \( \text{MaxGray} \geq \text{MinGray} \)

---

**Example**

```haskell
read_image(Image,'fabrik')
sobel_dir(Image,EdgeAmp,EdgeDir,'sum_abs',3)
threshold(EdgeAmp,Seg,50,255,2)
skeleton(Seg,Rand)
connection(Rand,Lines)
select_shape(Lines,Edges,'area','and',10,1000000).
```

---

**Result**

Threshold returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

---

**Parallelization Information**

Threshold is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

HistoToThresh, MinMaxGray, SobelAmp, GaussImage, ReduceDomain, FillInterlace

---

**Possible Successor Functions**

Connection, Dilation1, Erosion1, Opening, Closing, RankRegion, ShapeTrans, Skeleton

---

**See Also**

DualThreshold, ZeroCrossing, BackgroundSeg, RegionGrowing

---

**Alternatives**

Class2DimSup, HysteresisThreshold, DynThreshold

---

HALCON 6.0
Module

Region processing

```c
[out] HXLDContX Border HImageX.ThresholdSubPix
([in] VARIANT Threshold )
```

```c
void HOperatorSetX.ThresholdSubPix ([in] IHObjectX Image,
[out] HUntypedObjectX Border, [in] VARIANT Threshold )
```

Extract level crossings from an image with subpixel accuracy.

ThresholdSubPix extracts the level crossings at the level Threshold of the input image Image with subpixel accuracy. The extracted level crossings are returned as XLD-contours in Border. In contrast to the operator Threshold, ThresholdSubPix does not return regions, but the lines which separate regions with a gray value less than Threshold from regions with a gray value greater than Threshold.

For the extraction, the input image is regarded as a surface, in which the gray values are interpolated bilinearly between the centers of the individual pixels. Consistent with the surface thus defined, level crossing lines are extracted for each pixel and linked into topologically sound contours. This means that the level crossing contours are correctly split at junction points. If the image contains extended areas of constant gray value Threshold, only the border of such areas is returned as level crossings.

Parameter

- **Image** (input iconic) ................. image ~ HImageX / IHObjectX ( byte, int1, int2, int4, real )
  - Input image.

- **Border** (output iconic) .............. xld_cont ~ HXLDContX / HUntypedObjectX
  - Extracted level crossings.

- **Threshold** (input control) ............ number ~ VARIANT ( real, integer )
  - Threshold for the level crossings.

  **Default Value**: 128

  **Value Suggestions**: Threshold ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0}

Example

```c
read_image(Image,'fabrik')
threshold_sub_pix(Image,Border,35)
disp_xld(Border,WindowHandle)
```

Result

ThresholdSubPix usually returns the value TRUE. If necessary, an exception is raised.

Parallelization Information

ThresholdSubPix is reentrant and processed without parallelization.

See Also

ZeroCrossingSubPix

Alternatives

Threshold

Module

Sub-pixel operators

```c
[out] HRegionX Basins HImageX.Watersheds ([out] HRegionX Watersheds )
```

```c
void HOperatorSetX.Watersheds ([in] IHObjectX GrayImage,
[out] HUntypedObjectX Basins, [out] HUntypedObjectX Watersheds )
```

Extract watersheds and basins from an image.
Watersheds segments an image based on the topology of the gray values. The image is interpreted as a "mountain range." Higher gray values correspond to "mountains," while lower gray values correspond to "valleys." In the resulting mountain range watersheds are extracted. These correspond to the bright ridges between dark basins. On output, the parameter Basins contains these basins, while Watersheds contains the watersheds, which are at least one pixel wide (points on the ridge which form a plateau). Watersheds always is a single region per input image, while Basins contains a separate region for each basin. It is advisable to apply a smoothing operator (e.g., GaussImage) to the input image before calling Watersheds in order to reduce the number of output regions.

Attention
If the image contains many fine structures or is noisy, many output regions result, and thus the runtime increases considerably.

Parameter
- **GrayImage** (input iconic) .......... image(-array) \( \rightarrow \) HImageX / IHObjectX (byte)
  Images to be segmented.
- **Basins** (output iconic) .............. region \( \rightarrow \) HRegionX / HUntypedObjectX
  Segments found (dark basins).
- **Watersheds** (output iconic) ......... region(-array) \( \rightarrow \) HRegionX / HUntypedObjectX
  Watersheds between the basins.

Result
Watersheds always returns TRUE. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information
Watersheds is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
- GaussImage, SmoothImage, InvertImage

Possible Successor Functions
- ExpandRegion, SelectShape, ReduceDomain, Opening

Alternatives
- Pouring

Module
Region processing

[out] HRegionX RegionCrossing HImageX.ZeroCrossing ( )

void HOperatorSetX.ZeroCrossing ([in] IHObjectX Image,
[out] HUntypedObjectX RegionCrossing )

Detect zero crossings in an image.

ZeroCrossing returns the zero crossings of the input image as a region. A pixel is accepted as a zero crossing if its gray value (in Image) is zero, or if at least one of its 4-connected neighbors has a different sign.

This operator is intended to be used after edge operators returning the second derivative of the image (e.g., LaplaceOfGauss), which were possibly followed by a smoothing operator. In this case, the zero crossings are (candidates for) edges.

Parameter
- **Image** (input iconic) ................. image(-array) \( \rightarrow \) HImageX / IHObjectX (int2, int4, real)
  Input image.
- **RegionCrossing** (output iconic) ......... region(-array) \( \rightarrow \) HRegionX / HUntypedObjectX
  Zero crossings (as region).

Result
ZeroCrossing usually returns the value TRUE. If necessary, an exception is raised.

Parallelization Information
ZeroCrossing is reentrant and automatically parallelized (on tuple level).
 Possible Predecessor Functions
Laplace, LaplaceOfGauss, DerivateGauss

 Possible Successor Functions
Connection, Skeleton, Boundary, SelectShape, FillUp

 Alternatives
Threshold, DualThreshold

 Module
Region processing

 Extract zero crossings from an image with subpixel accuracy.
ZeroCrossingSubPix extracts the zero crossings of the input image Image with subpixel accuracy. The extracted zero crossings are returned as XLD-contours in ZeroCrossings. Thus, ZeroCrossingSubPix can be used as a sub-pixel precise edge extractor if the input image is a Laplace-filtered image (see Laplace, LaplaceOfGauss, DerivateGauss).
For the extraction, the input image is regarded as a surface, in which the gray values are interpolated bilinearly between the centers of the individual pixels. Consistent with the surface thus defined, zero crossing lines are extracted for each pixel and linked into topologically sound contours. This means that the zero crossing contours are correctly split at junction points. If the image contains extended areas of constant gray value 0, only the border of such areas is returned as zero crossings.

 Parameter

\- **Image** (input iconic) ......................... image \sim HImageX / IHObjectX (int1, int2, int4, real )
Input image.

\- **ZeroCrossings** (output iconic) ...................... xld_cont \sim HXLDContX / HUntypedObjectX
Extracted zero crossings.

 Example

/* Detection zero crossings of the Laplacian-of-Gaussian of an aerial image */
read_image(Image,’mreut’)
derivate_gauss(Image,Laplace,3,’laplace’)
zero_crossing_sub_pix(Laplace,ZeroCrossings)
disp_xld(ZeroCrossings,WindowHandle)

/* Detection of edges, i.e, zero crossings of the Laplacian-of-Gaussian that have a large gradient magnitude, in an aerial image */
read_image(Image,’mreut’)
Sigma := 1.5
/* Compensate the threshold for the fact that derivate_gauss(...,’gradient’) calculates a Gaussian-smoothed gradient, in which the edge amplitudes are too small because of the Gaussian smoothing, to correspond to a true edge amplitude of 20. */
Threshold := 20/(Sigma*sqrt(2*3.1415926))
derivate_gauss(Image,Gradient,Sigma,’gradient’)
threshold(Gradient,Region,Threshold,255)
reduce_domain(Image,Region,ImageReduced)
derivate_gauss(ImageReduced,Laplace,Sigma,’laplace’)
zero_crossing_sub_pix(Laplace,Edges)
disp_xld(Edges,WindowHandle)
### Result

ZeroCrossingSubPix usually returns the value TRUE. If necessary, an exception is raised.

### Parallelization Information

ZeroCrossingSubPix is reentrant and automatically parallelized (on tuple level).

### Possible Predecessor Functions

Laplace, LaplaceOfGauss, DiffOfGauss, DerivateGauss

### Alternatives

ZeroCrossing

### Module

Sub-pixel operators
Chapter 11

System

11.1 Database

Number of entries in the HALCON database.

The operator CountRelation counts the number of entries in one of the four relations of the HALCON database. The HALCON database is organized as follows:

There are two basic relations for region — data and image — matrices. The HALCON objects region and image are constructed from elements from these two relations: a region consists of a pointer to a tuplet in the region-data relation. An image consists also of a pointer to a tuplet in the region-data relation (like a region) and additionally of one or more pointers to tuplets in the matrix relation. If there is more than one matrix pointer, the image is called a multi — channel image.

Both regions and images are called objects. A region can be considered as the special case of an iconic object having no image matrices. For reasons of an efficient memory management, the tuplets of the region-data relation and the image-matrix relation will be used by different objects together. Therefore there may be for example more images than image matrices. Only the two lowlevel relations are of relevance to the memory consumption. Image objects (regions as well as images) consist only of references on region and matrix data and therefore only need a couple of bytes of memory.

Possible values for RelationName:

'image': Image matrices. One matrix may also be the component of more than one image object (no redundant storage).

'region': Regions (the full and the empty region are always available). One region may of course also be the component of more than one image object (no redundant storage).

'XLD': eXtended Line Description: Contours, Polygons, parallels, lines, etc. XLD data types don’t have gray values and are stored with subpixel accuracy.

'object': Iconic objects. Composed of a region (called region) and optionally image matrices (called image).

'tuple': In the compact mode, tuplets of iconic objects are stored as a surrogate in this relation. Instead of working with the individual object keys, only this tuplet key is used. It depends on the host language, whether the objects are passed individually (Prolog and C++) or as tuplets (C, Smalltalk, Lisp, OPS-5).

Certain database objects will be created already by the operator ResetObjDb and therefore have to be available all the time (the undefined gray value component, the objects 'full’ (FULL_REGION in HALCON/C) and 'empty’ (EMPTY_REGION in HALCON/C) as well as the herein included empty and full region). By calling GetChannelInfo, the operator therefore appears correspondingly also as 'creator’ of the full and empty region. The procedure can be used for example to check the completeness of the ClearObj operation.

<table>
<thead>
<tr>
<th>[out] long NumOfTuples</th>
<th>HSystemX.CountRelation ([in] String RelationName )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void HOperatorSetX.CountRelation ([in] String RelationName, [out] long NumOfTuples )</td>
<td></td>
</tr>
</tbody>
</table>

Attention
Parameter

Parameter

RelationName (input control) ................................................................. string ~ String

Relation of interest of the HALCON database.

Default Value : 'object'

Value List : RelationName ∈ { 'image', 'region', 'XLD', 'object', 'tuple' }

NumOfTuples (output control) ............................................................... integer ~ long

Number of tuplets in the relation.

Example

reset_obj_db(512,512,3)
count_relation('image',I1)
count_relation('region',R1)
count_relation('XLD',X1)
count_relation('object',O1)
count_relation('tuple',T1)
read_image(X,'monkey')
count_relation('image',I2)
count_relation('region',R2)
count_relation('XLD',X2)
count_relation('object',O2)
count_relation('tuple',T2)

/*

Result: I1 = 1 (undefined image)
R1 = 2 (full and empty region)
X1 = 0 (no XLD data)
O1 = 2 (full and empty objects)
T1 = 0 (always 0 in the normal mode)

I2 = 2 (additionally the image 'monkey')
R2 = 2 (read_image uses the full region)
X2 = 0 (no XLD data)
O2 = 3 (additionally the image object X)
T2 = 0.

*/

Result

If the parameter is correct, the operator CountRelation returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

CountRelation is reentrant and processed without parallelization.

Possible Predecessor Functions

ResetObjDb

Possible Predecessor Functions

See Also

Module

System

[out] VARIANT UsedModules HSystemX.GetModules ([out] long ModuleKey )

void HOperatorSetX.GetModules ([out] VARIANT UsedModules,
[out] long ModuleKey )

Query of used modules and the module key.

GetModules returns the module numbers of all operators used up to this point. Each operator belongs to one module (maximum 32). Each module has a name, which is returned in UsedModules. Based on the used
11.1. DATABASE 583

modules, a key is generated that is needed for the licence manager. GetModules is normally called at the end of
a programm to check the used modules.

Parameter

- **UsedModules** (output control) ...........................................string \(\sim\) VARIANT( string )
  
  Names of used modules.

- **ModuleKey** (output control) ..................................................integer \(\sim\) long
  
  Key for licence manager.

Parallelization Information

GetModules is reentrant and processed without parallelization.

Operators not requiring licensing

```cpp
void HSystemX.ResetObjDb ([in] long DefaultImageWidth,
                         [in] long DefaultImageHeight, [in] long DefaultChannels )

void HOperatorSetX.ResetObjDb ([in] long DefaultImageWidth,
                                [in] long DefaultImageHeight, [in] long DefaultChannels )
```

Initialization of the HALCON system.

The operator ResetObjDb initializes the HALCON system. With this procedure the four relations (grayvalue
data, region data, iconic objects and object tuplets) which are necessary for image processing with HALCON will
be installed (see also CountRelation). In case the relations already exist, all tuplets in the relations will be
deallocated!

The parameters DefaultImageWidth and DefaultImageHeight provide the initial values for the global
maximum image size. If the first created object is an image, (e.g. ReadImage), the set values will be overruled
in Standard-HALCON by the size of this picture. Instead of this, in Parallel HALCON the set values will only be
changed, if they are smaller than the size of the created object. If on the other hand the first object to be created
is a region, both in Standard- and in Parallel HALCON the values will only be adjusted in case the new image is
larger than the set values. This is not only the case for the first image which is created or read: the global image
size will always be enlarged, if larger images are created.

The global image size is of consequence for the opening of windows (OpenWindow) and the clipping of
regions. Whenever the clip mode is activated, regions will be clipped according to the global image size
(SetSystem(‘clipRegion’,true)). This can lead to problems if images of various sizes are used.
In this case only the fact that a region is smaller or of the same size as the largest image can be guaranteed.

The parameter DefaultChannels returns the most frequent number of channels of an image object. This value
can be set to 0 if for the most part regions are used. If more channels than those having been set at the initialization
are necessary for one image, the number will be enlarged dynamically for this image. If less channels than those
having been set at the initialization are necessary for the image, the superfluous channels will be set as undefined.
For the user this will seem as if they were non existent, however, memory is allocated unnecessarily.

The parameter values can be queried using the operator GetSystem.

Attention

If the operator ResetObjDb is not called at the beginning of a HALCON session, HALCON will be initialized
automatically by the operator ResetObjDb(128,128,0). In case the operator ResetObjDb is called again,
all image objects in the database will be deallocated.

Parameter

- **DefaultImageWidth** (input control) ...........................................integer \(\sim\) long
  
  Default image width (in pixels).

  **Default Value** : 128

  **Value Suggestions**: DefaultImageWidth \(\in\) \{64, 128, 256, 512, 525, 1024\}

- **DefaultImageHeight** (input control) ...........................................integer \(\sim\) long
  
  Default image height (in pixels).

  **Default Value** : 128

  **Value Suggestions**: DefaultImageHeight \(\in\) \{64, 128, 256, 512, 768, 1024\}

HALCON 6.0
 CHAPTER 11. SYSTEM

DefaultChannels (input control) ............................................................integer ⇆ long
Usual number of channels by which the system constant 'max_channels' is limited.

Value Suggestions: DefaultChannels ∈ {0, 1, 2, 3, 4, 5, 6, 7}

Result
The operator ResetObjDb returns the value TRUE if the parameter values are correct. Otherwise an exception will be raised.

Parallelization Information
ResetObjDb is reentrant and processed without parallelization.

See Also
GetChannelInfo, CountRelation

Module
Operators not requiring licensing

11.2 Error-Handling

State of the HALCON control modes.
Executing the operator GetCheck the user can inquire what kind of control modes are currently activated and which are not. Check gives the tuplet containing the names of the control modes (see also SetCheck) which are preceded by a tilde ('', e.g. ''data'), if the corresponding control is deactivated.

Attention

Parameter

Check (output control) .................................................................string ⇆ VARIANT(string)
Tuplet of the currently activated control modes.

Result
GetCheck always returns the value TRUE.

Parallelization Information
GetCheck is reentrant and processed without parallelization.

Possible Predecessor Functions
SetCheck

See Also
SetCheck

Module
System

Inquiry after the error text of a HALCON error number.
The operator GetErrorText returns the error text for the corresponding HALCON error number. This is indeed the same text which will be given during an exception. The operator GetErrorText is especially useful if the error treatment is programmed by the users themselves (see also SetCheck('give error')).

Attention
Unknown error numbers will trigger a standard message.
11.2. ERROR-HANDLING

Parameter

- **ErrorNumber** (input control) ................................................ integer \( \sim \) long
  Number of the HALCON error.
  **Restriction:** \( (1 \leq \text{ErrorNumber}) \land (\text{ErrorNumber} \leq 30000) \)
- **ErrorText** (output control) ................................................ string \( \sim \) String
  Corresponding error text.

Result

The operator GetErrorText always returns the value TRUE.

Parallelization Information

GetErrorText is **reentrant** and processed without parallelization.

Possible Predecessor Functions

SetCheck

See Also

SetCheck

Module

System

```c
[out] VARIANT Value HSystemX.GetSpy ([in] String Class )
void HOperatorSetX.GetSpy ([in] String Class, [out] VARIANT Value )
```

Current configuration of the HALCON debugging-tool.

The operator GetSpy returns the current configuration of spy, the HALCON debugging tool. The available control modes (possible choices for Class) as well as the corresponding tuning possibilities (possible values for Value) can be called up by using the operator QuerySpy. You will find a more detailed description under SetSpy.

Attention

- **Class** (input control) ................................................ string \( \sim \) String
  Control mode
  **Default Value:** 'mode'
  **Value List:** Class \( \in \) { 'mode', 'procedure', 'input_control', 'output_control', 'parameter_values', 'input_gray_window', 'output_gray_window', 'input_region_window', 'db', 'output_region_window', 'halt', 'timeout', 'button_window', 'button_window', 'button_click', 'button_notify', 'log_file', 'error', 'internal' }
- **Value** (output control) ................................................ string \( \sim \) VARIANT ( string, integer, real )
  State of the control mode.

Result

The operator GetSpy returns the value TRUE if the parameter Class is correct. Otherwise an exception is raised.

Parallelization Information

GetSpy is **reentrant** and processed without parallelization.

Possible Predecessor Functions

ResetObjDb

See Also

SetSpy, QuerySpy

Module

System

```c
[out] VARIANT Classes HSystemX.QuerySpy ([out] VARIANT Values )
void HOperatorSetX.QuerySpy ([out] VARIANT Classes, [out] VARIANT Values )
```

Inquiring for possible settings of the HALCON debugging tool.
The operator QuerySpy returns all possible settings of spy, the HALCON debugging tool, i.e. all the available control modes (Classes) as well as the corresponding possible ways of setting (Values). For a more detailed description of spy see operator SetSpy.

Attention

The values of Values cannot be used as direct input for SetSpy, as they are transmitted as a symbolic constant.

Parameter

▷ Classes (output control) .................................................. string ~ VARIANT( string )
   Available control modes (see also set_spy).
▷ Values (output control) ................................................... string ~ VARIANT( string )
   Corresponding state of the control modes.

query_spy always returns the value TRUE.

Result

QuerySpy is reentrant, local, and processed without parallelization.

Parallelization Information

ResetObjDb

Possible Predecessor Functions

See Also

SetSpy, GetSpy

Module

System

void HSystemX.SetCheck ([in] VARIANT Check )

void HOperatorSetX.SetCheck ([in] VARIANT Check )

Activating and deactivating of HALCON control modes.

With the help of the operator SetCheck different control modes of the HALCON system can be activated or deactivated. If a certain control mode is activated, parameters etc. will be checked at runtime. Whenever an inconsistency is hereby detected, the program will be interrupted by an exception.

It is recommendable to activate the control modes during the development of a program and to deactivate them only after a successfully concluded testrun. For if the control mode is deactivated and an error occurs, the system may react in an unpredictable way.

The HALCON system provides various possible control modes which can be activated and deactivated independently. By calling the operator SetCheck with the name (Check) of the desired control mode, this control mode is activated; the control mode is deactivated by passing its name prefixed with a tilde (˜, z.B. ‘˜data’).

Available control modes:

'color': If this control mode is activated, only colors may be used which are supported by the display for the currently active window. Otherwise an error message is displayed.

In case of deactivated control mode and non existent colors, the nearest color is used (see also SetColor, SetGray, SetRgb).

'text': If this control mode is activated, it will check the coordinates during the setting of the text cursor as well as during the display of strings (WriteString) to the effect whether a part of a sign would lie outside the windowframe (a fact which is not forbidden in principle by the system).

If the control mode is deactivated, the text will be clipped at the windowframe.

'parameter': (For HALCON/PRO only)

If this control mode is activated, output parameter may not be instantiated by calling a procedure.

Otherwise a normal unification mechanism is used.

'data': (For program development)

Checks the consistency of image objects (regions and grayvalue components).

'interface': If this control mode is activated, the interface between the host language and the HALCON procedures will be checked in course (e.g. typifying and counting of the values).
'database': This is a consistency check of the database (e.g. checks whether an object which shall be canceled does indeed exist or not.)

'give_error': Determines whether errors shall trigger exceptions or not. If this control mode is deactivated, the application program must provide a suitable error treatment itself. Please note that errors which are not reported usually lead to undefined output parameters which may cause an unpredictable reaction of the program.

'father': If this control mode is activated by calling the operator OpenWindow, HALCON allows only the usage of the number of another HALCON window as the ’father’ of the new window; otherwise it allows also other X Window IDs.

'region': For program development)
Checks the consistency of chords (this may lead to a notable speed reduction of routines).

'clear': Normally, if a list of objects shall be canceled by using ClearObj, an exception will be raised, in case individual objects do not or no longer exist. If the 'clear' mode is activated, such objects will be ignored.

'all': Activates all control modes.

'none': Deactivates all control modes.

'default': Original setting.

Attention
Parameter

Check (input control) \[\text{string-array} \] V ARIANT ( string )
Desired control mode.

Default Value : 'default'
Value List : Check \in \{ 'color', 'text', 'database', 'data', 'interface', 'give_error', 'father', 'region', 'clear', 'all', 'none', 'default' \}

Result

The operator SetCheck returns the value TRUE, if the parameters are correct. Otherwise an exception will be raised.

Parallelization Information
SetCheck is reentrant and processed without parallelization.

See Also
GetCheck, SetColor, SetRgb, SetHsi, WriteString

Module
System

void HSystemX.SetSpy ([in] String Class, [in] VARIANT Value )
void HOperatorSetX.SetSpy ([in] String Class, [in] VARIANT Value )

Control of the HALCON Debugging Tools.
The operator SetSpy is the HALCON debugging tool. This tool allows the flexible control of the input and output data of HALCON-operators - in graphical as well as in textual form. The datacontrol is activated by using

SetSpy(:'mode','on'),

and deactivated by using

SetSpy(:'mode','off').

The debugging tool can further be activated with the help of the environment variable HALCONSPY. The definition of this variable corresponds to calling up 'mode' and 'on'.

The following control modes can be tuned (in any desired combination of course) with the help of Class/Value:

Class Meaning / Value
'operator' When a routine is called, its name and the names of its parameters will be given (in TRIAS notation).
Value: 'on' or 'off'
default: 'off'

'input_control' When a routine is called, the names and values of the input control parameters will be given.
Value: 'on' or 'off'
default: 'off'

'output_control' When a routine is called, the names and values of the output control parameters are given.
Value: 'on' or 'off'
default: 'off'

'parameter_values' Additional information on 'input_control' and 'output_control': indicates how many values per parameter shall be displayed at most (maximum tuplet length of the output).
Value: tuplet length (integer)
default: 4

'db' Information concerning the 4 relations in the HALCON-database. This is especially valuable in looking for forgotten ClearObj.
Value: 'on' or 'off'
default: 'off'

'input_gray_window' Any reading access of the gray-value component of an (input) image object will cause the gray-value component to be shown in the indicated window (Window-ID; 'none' will deactivate this control).
Value: Window-ID (integer) or 'none'
default: 'none'

'output_gray_window' As soon as the gray-value component of an (output) image object is set, spy will show this gray-value component in the indicated window (Window-ID; 'none' will deactivate this control).
Value: Window-ID (integer) or 'none'
default: 'none'

'input_region_window' Any reading access of the region of an (input) iconic object will cause this region to be shown in the indicated window (Window-ID; 'none' will deactivate this control).
Value: Window-ID (integer) or 'none'
default: 'none'

'output_region_window' As soon as the region of an (output) iconic object is set, spy will show this region in the indicated window (Window-ID; 'none' will deactivate this control).
Value: Window-ID (integer) or 'none'
default: 'none'

'time' Processing time of the operator
Value: 'on' or 'off'
default: 'off'

'halt' Determines whether there is a halt after every individual action ('multiple') or only at the end of each operator ('single'). The parameter is only effective if the halt has been activated by 'timeout' or 'button_window'.
Value: 'single' or 'multiple'
default: 'multiple'

'timeout' After every output there will be a halt of the indicated number of seconds.
Value: seconds (real)
default: 0.0

'button_window' Alternative to 'timeout': after every output spy waits until the cursor indicates ('button_click' = 'false') or clicks into ('button_click' = 'true') the indicated window. (Window-ID; 'none' will deactivate this control).
Value: Window-ID (integer) or 'none'
default: 'none'

'button_click' Additional option for 'button_window': determines whether or not a mouse-click has to be waited for after an output.
Value: 'on' or 'off'
default: 'off'

'button_notify' If 'button_notify' is activated, spy generates a beep after every output. This is useful in combination with 'button_window'.
11.3 Information

Value: ‘on’ or ‘off’
default: ‘off’

'log_file' Spy can hereby divert the text output into a file having been opened with open_file.
Value: a file handle (see OpenFile)

'error' If ‘error’ is activated and an internal error occurs, spy will show the internal procedures (file/line) concerned.
Value: ‘on’ or ‘off’
default: ‘off’

'internal' If ‘internal’ is activated, spy will display the internal procedures and their parameters (file/line) while an HALCON-operator is processed.
Value: ‘on’ or ‘off’
default: ‘off’

Attention

Parameter

Class (input control) ................................................................. string  \( \rightarrow \) String
Control mode
Default Value: ‘mode’
\input\_gray\_window’, ‘db’, ‘time’, \output\_gray\_window’, ‘\input\_region\_window’,

Value (input control) ......................................................... string  \( \rightarrow \) VARIANT ( string, integer, real )
State of the control mode to be set.
Default Value: ‘on’
Value Suggestions: Value \[\{ ‘on’, ‘off’, 1, 2, 3, 4, 5, 10, 50, 0.0, 1.0, 2.0, 5.0, 10.0 \}\]

Result

The operator SetSpy returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

Parallelization Information

ResetObjDb

Possible Predecessor Functions

GetSpy, QuerySpy

See Also

Module

System

11.3 Information

Display the manual information of a procedure on the screen.

The operator DispInfo together with the previewer DispProgram shows the manual entry of the indicated procedure on the screen. The individual files (\<Name: >.Extension) are located in the HALCON-Directory "\doc/ps/reference/LANGUAGE/*", whereby the value for LANGUAGE will be determined by the currently used host language. The directory can also be generated with the help of the operator SetSystem (:: ‘referenceDir’, ‘Pfad’).
Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcName</td>
<td>Name of the sought procedure.</td>
</tr>
<tr>
<td>Default Value</td>
<td>'read_image'</td>
</tr>
<tr>
<td>Machine</td>
<td>Name of the computer to which the data shall be transmitted or empty string.</td>
</tr>
<tr>
<td>Default Value</td>
<td>''</td>
</tr>
<tr>
<td>DispProgram</td>
<td>Name of the program which shall display the help text.</td>
</tr>
<tr>
<td>Default Value</td>
<td>'ghostview'</td>
</tr>
<tr>
<td>Value Suggestions: DispProgram ∈ {'gs', 'ghostview'}</td>
<td></td>
</tr>
<tr>
<td>Extension</td>
<td>Extension of the helpfile.</td>
</tr>
<tr>
<td>Default Value</td>
<td>'ps'</td>
</tr>
</tbody>
</table>

Result

If the parameter values are correct and the file and the program are available, the operator DispInfo returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

DispInfo is reentrant and processed without parallelization.

See Also

OpenWindow

Module

Image / region / XLD management

Get information concerning the chapters on procedures.

The operator GetChapterInfo gives information concerning the chapters on procedures. If instead of Chapter the empty string is transmitted, the routine will provide in Info the names of all chapters. If on the other hand a certain chapter or a chapter and its subchapter(s) are indicated (by a tuple of names), the corresponding subchapters or - in case there are no further subchapters - the names of the corresponding procedures will be given. The organization of the chapters on procedures is the same as the organization of chapters and subchapters in the HALCON-manual. Please note: The chapters on procedures respectively the subchapters concerning an individual procedure can be called by using the operator GetOperatorInfo(:<Name>, 'chapter', Info: ). The Online-texts will be taken from the files english.hlp, english.sta, english.num and english.idx, which will be searched by HALCON in the currently used directory or the directory 'help_dir' (see also GetSystem and SetSystem).

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter</td>
<td>Procedure class or subclass of interest.</td>
</tr>
<tr>
<td>Default Value</td>
<td>''</td>
</tr>
<tr>
<td>Info</td>
<td>Procedure classes (Chapter = '') or procedure subclasses respectively procedures.</td>
</tr>
</tbody>
</table>

Result

If the parameter values are correct and the helpfile is available, the operator GetChapterInfo returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

GetChapterInfo is processed completely exclusively without parallelization.

Possible Predecessor Functions

GetSystem, SetSystem
GetOperatorInfo, GetSystem, SetSystem

See Also

Module

Image / region / XLD management

[out] VARIANT Keywords HInfoX.GetKeywords ([in] String ProcName )

void HOperatorSetX.GetKeywords ([in] String ProcName,
[out] VARIANT Keywords )

Get keywords which are assigned to procedures.

The operator GetKeywords returns all the keywords in the online-texts corresponding to those procedures which have the indicated substring ProcName in their name. If instead of ProcName the empty string is transmitted, the operator GetKeywords returns all keywords. The keywords of an individual procedure can also be called by using the operator GetOperatorInfo. The online-texts will be taken from the files english.hlp, english.sta, english.num, english.key and english.idx, which are searched by HALCON in the currently used directory and in the directory 'help_dir' (see also GetSystem and SetSystem).

Attention

Parameter

.proc_name

Default Value : 'get_keywords'

.string

Result

The operator GetKeywords returns the value TRUE if the parameters are correct and the helpfiles are available. Otherwise an exception handling is raised.

Parallelization Information

GetKeywords is processed completely exclusively without parallelization.

Possible Predecessor Functions

GetChapterInfo

See Also

GetOperatorName, SearchOperator, GetParamInfo

Alternatives

GetOperatorInfo

Module

Image / region / XLD management

[out] VARIANT Information HInfoX.GetOperatorInfo ([in] String ProcName,
[in] String Slot )

void HOperatorSetX.GetOperatorInfo ([in] String ProcName,
[in] String Slot, [out] VARIANT Information )

Get information concerning a HALCON-procedure.

With the help of the operator GetOperatorInfo the online-texts concerning a certain procedure can be called (see also GetOperatorName). The form of information available for all procedures (Slot) can be called using the operator QueryOperatorInfo. For the time being the following slots are available:

'short': Short description of the procedure.

'abstract': Description of the procedure.
'procedure_class': Name(s) of the chapter(s) in the procedure hierarchy (chapter, subchapter in the HALCON manual).

'functionality': Functionality is equivalent to the object class to which the procedure can be assigned.

'keywords': Keywords of the procedure (optional).

'example': Example for the use of the procedure (optional). The operator 'example.LANGUAGE' (LANGUAGE ∈ \{c,c++,smalltalk,triias\}) calls up examples for a certain language if available. If the language is not indicated or if no example is available in this language, the TRIAS-example will be returned.

'complexity': Complexity of the procedure (optional).

'effect': Not in use so far.

'alternatives': Alternative procedures (optional).

'see_also': Procedures containing further information (optional).

'predecessor': Possible and sensible predecessor

'successor': Possible and sensible successor

'result_state': Return value of the procedure (TRUE, FALSE, FAIL, VOID or EXCEPTION).

'attention': Restrictions and advice concerning the correct use of the procedure (optional).

'parameter': Names of the parameter of the procedure (see also GetParamInfo).

'references': Literary references (optional).

'module': The module to which the operator is assigned.

'html_path': The directory where the HTML documentation of the operator resides.

'parameter_relations': Assurances concerning the parameters (optional).

The texts will be taken from the files english.hlp, english.sta, english.key, english.num und english.idx which will be searched by HALCON in the currently used directory or in the directory 'help_dir' (respectively 'user_help_dir') (see also GetSystem and SetSystem). By adding '.latex' after the slotname, the text of slots containing textual information can be made available in LATEX notation.

---

**Parameter**

- **ProcName** (input control) .............................................. proc_name ~> String
  
  Name of the procedure on which more information is needed.

  **Default Value**: 'get_operator_info'

  - **Slot** (input control) ............................................. string ~> String
    
    Desired information.

    **Default Value**: 'abstract'

    **Value List**: Slot ∈ \{'short', 'abstract', 'procedure_class', 'functionality', 'effect', 'complexity', 'predecessor', 'successor', 'alternatives', 'see_also', 'keywords', 'example', 'attention', 'result_state', 'return_value', 'references', 'sourcefiles', 'deffile', 'module', 'html_path'\}

  - **Information** (output control) ................................ string ~> VARIANT( string )
    
    Information (empty if no information is available)

---

**Result**

The operator GetOperatorInfo returns the value TRUE if the parameters are correct and the helpfiles are available. Otherwise an exception handling is raised.

---

**Parallelization Information**

GetOperatorInfo is processed completely exclusively without parallelization.

**Possible Predecessor Functions**

GetKeywords, SearchOperator, GetOperatorName, QueryOperatorInfo, QueryParamInfo, GetParamInfo

**Possible Successor Functions**

GetParamNames, GetParamNum, GetParamTypes

**See Also**

QueryOperatorInfo, GetParamInfo, GetOperatorName, GetParamNum, GetParamTypes

**Alternatives**

GetParamNames

---

**Module**

Image / region / XLD management
11.3. INFORMATION

Get procedures with the given string as a substring of their name.

The operator GetOperatorName takes a string (Pattern) as input and searches all HALCON-procedures having this string as a substring in their name. If an empty string is entered, the names of all procedures available will be returned.

<table>
<thead>
<tr>
<th><strong>Attention</strong></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
</tr>
</thead>
</table>

▷ **Pattern** (input control) ........................................................................ string ⇄ String Substring of the sought names (empty ⇣ all names).  
Default Value: 'info'

▷ **ProcNames** (output control) ............................................................... string ⇄ VARIANT ( string ) Detected procedure names.

<table>
<thead>
<tr>
<th><strong>Result</strong></th>
</tr>
</thead>
</table>

The operator GetOperatorName returns the value TRUE if the helpfiles are available. Otherwise an exception handling is raised.

<table>
<thead>
<tr>
<th><strong>Parallelization Information</strong></th>
</tr>
</thead>
</table>

GetOperatorName is reentrant, local, and processed without parallelization.

<table>
<thead>
<tr>
<th><strong>Possible Successor Functions</strong></th>
</tr>
</thead>
</table>

GetOperatorInfo, GetParamNames, GetParamNum, GetParamTypes

<table>
<thead>
<tr>
<th><strong>See Also</strong></th>
</tr>
</thead>
</table>

GetOperatorInfo, GetParamNames, GetParamNum, GetParamTypes

<table>
<thead>
<tr>
<th><strong>Alternatives</strong></th>
</tr>
</thead>
</table>

SearchOperator

Image / region / XLD management

Get information concerning the procedure parameters.

The operator GetParamInfo is used for calling up the online-texts assigned to a parameter of an indicated procedure. The form of information available for each parameter (Slot), can be called up by using the operator QueryParamInfo. At the moment the following slots are available:

'**description**': Description of the parameter.

'**description.latex**': Description of the parameter in LATEX notation.

'**parameter_class**': Parameter classes: 'input_object', 'output_object', 'input_control' oder 'output_control'.

'**type_list**': Permitted type(s) of data for parameter values (for control parameters only). Value: 'real', 'integer' oder 'string'.

'**default_type**': Default-type for parameter values (for control parameters only). This type of parameter is the one HALCON/C uses in the "simple mode". If 'none' is indicated, the "tuple mode" must be used. Value: 'real', 'integer', 'string' oder 'none'.

'**sem_type**': Semantic type of the parameter. This is important to allow the assignment of the parameters to object classes in object-oriented languages (C++, Smalltalk). If more than one parameter belongs semantically to one type, this fact is indicated as well. So far the the following objects are supported:
object, image, region, xld,
xld_cont, xld_cont, xld_poly, xld_ext_cont, xld_mod_cont,
integer, real, number, string,
channel, grayval, window,
histogram, distribution,
point(.x, .y), extent(.x, .y),
angle(.rad oder .deg),
circle(.center.x, .center.y, .radius),
arc(.center.x, .center.y, .angle.rad, .begin.x, .begin.y),
ellipse(.center.x, .center.y, .angle.rad, .radius1, .radius2),
line(.begin.x, .begin.y, .end.x, .end.y)
rectangle(.origin.x, .origin.y, .corner.x, .corner.y
bzw. .extent.x, .extent.y),
polygon(.x, .y), contour(.x, .y),
coordinates(.x, .y), chord(.x1, .x2, .y),
chain(.begin.x, .begin.y, .code).

'default_value': Default-value for the parameter (for input-control parameters only). It is the question of mere
information only (the parameter value must be transmitted explicitly, even if the default-value is used). This
entry serves only as a notice, a point of departure for own experiments. The values have been selected so that
they normally do not cause any errors but generate something that makes sense.

'multi_value': 'true', if more than one value is permitted in this parameter position, otherwise 'false'.

'multichannel': 'true', in case the input image object may be multichannel.

'mixed_type': For control parameters exclusively and only if value tuples ('multi-value'-true') and various types
of data are permitted for the parameter values ('type_list' having more than one value). In this case $Slot$
indicates, whether values of various types may be mixed in one tuple ('true' or 'false').

'values': Selection of values (optional).

'value_list': In case a parameter can take only a limited number of values, this fact will be indicated explicitly
(possible).

'value_min': Minimum value of a value interval.

'value_max': Maximum value of a value interval.

'value_function': Function describing the course of the values for a series of tests (lin, log, quadr, ...).

'step_rec': Recommended step width for the parameter values in a series of tests.

'step_min': Minimum step width of the parameter values in a series of tests.

'value_number': Expression describing the number of parameters as such or in relation to other parameters.

'assertion': Expression describing the parameter values as such or in relation to other parameters.

The online-texts will be taken from the files english.hlp, english.sta, english.key, english.num and english.idx
which will be searched by HALCON in the currently used directory or the directory 'help_dir' (see also
GetSystem and SetSystem).

<table>
<thead>
<tr>
<th>Attention</th>
<th>Parameter</th>
</tr>
</thead>
</table>

- **ProcName** (input control) .................................................. proc_name $\rightarrow$ String
  Name of the procedure on whose parameter more information is needed.
  **Default Value**: 'get_param_info'

- **ParamName** (input control) .................................................. string $\rightarrow$ String
  Name of the parameter on which more information is needed.
  **Default Value**: 'Slot'

- **Slot** (input control) .................................................. string $\rightarrow$ String
  Desired information.
  **Default Value**: 'description'
  **Value List**: Slot $\in$ \{ 'description', 'type_list', 'default_type', 'sem_type', 'default_value', 'values',
  'value_list', 'valuenum', 'valuemax', 'valuefunction', 'valuenumber', 'assertion', 'steprec', 'stepmin',
  'mixed_type', 'multi-value', 'multichannel'\}
11.3. INFORMATION

Information (output control) ................................................. string  \( \rightarrow \) VARIANT ( string )
Information (empty in case there is no information available).

--- Parallelization Information

GetParamInfo is processed completely exclusively without parallelization.

--- Possible Predecessor Functions

GetKeywords, SearchOperator

--- See Also

QueryParamInfo, GetOperatorInfo, GetOperatorName

--- Alternatives

GetParamNames, GetParamNum, GetParamTypes

--- Module

Image / region / XLD management

---

Get the names of the parameters of a HALCON-procedure.

For the HALCON-procedure indicated in ProcName the operator GetParamNames returns the names of the input objects, the output objects, of the input control parameters and the output control parameters.

Parameter

- **ProcName** (input control) ................................................. proc_name  \( \rightarrow \) String
  Name of the procedure.
  Default Value: 'get_param_names'

- **InpObjPar** (output control) ................................................. string  \( \rightarrow \) VARIANT ( string )
  Names of the input objects.

- **OutpObjPar** (output control) ................................................. string  \( \rightarrow \) VARIANT ( string )
  Names of the output objects.

- **InpCtrlPar** (output control) ................................................. string  \( \rightarrow \) VARIANT ( string )
  Names of the input control parameters.

- **OutpCtrlPar** (output control) ................................................. string  \( \rightarrow \) VARIANT ( string )
  Names of the output control parameters.

--- Result

The operator GetParamNames returns the value TRUE if the name of the procedure exists and the helpfiles are available. Otherwise an exception handling is raised.

--- Parallelization Information

GetParamNames is reentrant and processed without parallelization.

--- Possible Predecessor Functions

GetKeywords, SearchOperator, GetOperatorName, GetOperatorInfo

--- Possible Successor Functions

GetParamNum, GetParamTypes

--- See Also

GetParamNum, GetParamTypes, GetOperatorName

--- Alternatives

GetOperatorInfo, GetParamInfo

--- Module

Image / region / XLD management
Get number of the different parameter classes of a HALCON-procedure.

The operator GetParamNum returns the number of the input and output object parameters, as well as the input and output control parameters for the indicated HALCON-procedure. Further, you will receive the name of the C-function (CName) called by the procedure. The output parameter Type indicates, whether the procedure is a system procedure or an user procedure.

Parameter

- **ProcName** (input control) .......................................................... **proc_name** ~ String
  Name of the procedure.
  Default Value: 'get_param_num'

- **CName** (output control) .......................................................... **string** ~ String
  Name of the called C-function.

- **InpObjPar** (output control) ...................................................... **integer** ~ **long**
  Number of the input object parameters.

- **OutpObjPar** (output control) ..................................................... **integer** ~ **long**
  Number of the output object parameters.

- **InpCtrlPar** (output control) .................................................... **integer** ~ **long**
  Number of the input control parameters.

- **OutpCtrlPar** (output control) ................................................... **integer** ~ **long**
  Number of the output control parameters.

- **Type** (output control) ............................................................ **string** ~ String
  System procedure or user procedure.
  Value Suggestions: **Type** ∈ {'system', 'user'}

Result

The operator GetParamNum returns the value TRUE if the name of the procedure exists. Otherwise an exception handling is raised.

Parallelization Information

GetParamNum is reentrant and processed without parallelization.

Possible Predecessor Functions

GetKeywords, SearchOperator, GetOperatorName, GetOperatorInfo

Possible Successor Functions

GetParamTypes

See Also

GetParamNames, GetParamTypes, GetOperatorName

Alternatives

GetOperatorInfo, GetParamInfo

Module

Image / region / XLD management

Get default data type for the control parameters of a HALCON-procedure.
The operator GetParamTypes returns the default data type for each input and output control parameter. The default type of a parameter is the type used in "simple mode" in HALCON/C. This concerns parameters which allow more than one type as for example WriteString. Hereby the types of input parameters are combined in the variable InpCtrlParType, whereas the types of output parameters are combined in the variable OutpCtrlParType. The following types are possible:

'integer': an integer.
'integer tuple': an integer or a tuple of integers.
'real': a floating point number.
'real tuple': a floating point number or a tuple of floating point numbers.
'string': a string.
'string tuple': a string or a tuple of strings.
'no_default': individual value of which the type cannot be determined.
'no_default tuple': individual value or tuple of values of which the type cannot be determined.
'default': individual value of unknown type, whereby the system assumes it to be an 'integer'.

--- Parameter ---

▷ **ProcName** (input control) .............................................. proc_name ~ Variant ( string )
   Name of the procedure.
   Default Value: 'get_param_types'

▷ **InpCtrlParType** (output control) ........................................ string ~ Variant ( string )
   Default type of the input control parameters.

▷ **OutpCtrlParType** (output control) ........................................ string ~ Variant ( string )
   Default type of the output control parameters.

--- Parallelization Information ---

GetParamTypes is reentrant and processed without parallelization.

--- Possible Predecessor Functions ---

GetKeywords, SearchOperator, GetOperatorName, GetOperatorInfo

--- See Also ---

GetParamNames, GetParamNum, GetOperatorInfo, GetOperatorName

--- Alternatives ---

GetParamInfo

--- Module ---

Image / region / XLD management

--- Code ---

```cpp
[out] VARIANT Slots HInfoX.QueryOperatorInfo ()
void HOperatorSetX.QueryOperatorInfo ([out] VARIANT Slots )
```

Query slots concerning information with relation to the operator GetOperatorInfo.
The operator QueryOperatorInfo returns the names of those online texts (Slots) which are available online for each procedure. The information itself can be called up using

GetOperatorInfo(::<ProcName>,<Slot>::<Information>).

--- Attention ---

▷ **Slots** (output control) .............................................. string ~ Variant ( string )
   Slotnames of the operator GetOperatorInfo.

--- Result ---

The operator QueryOperatorInfo always returns the value TRUE.

--- Parallelization Information ---

QueryOperatorInfo is local and processed completely exclusively without parallelization.
Possible Successor Functions

GetOperatorInfo

See Also

GetOperatorInfo

Module

Image / region / XLD management

[out] VARIANT Slots HInfoX.QueryParamInfo ( )

void HOperatorSetX.QueryParamInfo ([out] VARIANT Slots )

Query slots of the online-information concerning the operator GetParamInfo.

The operator QueryParamInfo returns the names of those pieces of information (Slots) which are available online for each parameter (online texts). The online texts themselves can be called up using

GetParamInfo(::<Procedurname>,<Parametername>,<Slot>:<Information>).

Attention

Parameter

 Slots (output control) .................................................. string ⇆ VARIANT ( string )
 Slotnames for the operator GetParamInfo.

Result

QueryParamInfo always returns the value TRUE.

Parallelization Information

QueryParamInfo is reentrant, local, and processed without parallelization.

Possible Successor Functions

GetParamInfo

See Also

GetParamInfo

Module

Image / region / XLD management

[out] VARIANT ProcNames HInfoX.SearchOperator ([in] String Keyword )

void HOperatorSetX.SearchOperator ([in] String Keyword, [out] VARIANT ProcNames )

Search names of all procedures assigned to one keyword.

The operator SearchOperator returns the names of all procedures whose online-texts include the keyword Keyword (see also GetOperatorInfo). All available keywords are called by using the operator GetKeywords(::<’’ : <keywords>). The online-texts are taken from the files english.hlp, english.sta, english.key, english.num and Halcon.idx, which are searched by HALCON in the currently used directory or the directory 'help_dir' (see also GetSystem and GetSystem).

Attention

Parameter

 Keyword (input control) .................................................. string ⇆ String
 Keyword for which corresponding procedures are searched.
 Default Value : 'Information'

ProcNames (output control) .................................................. string ⇆ VARIANT ( string )
 Procedures whose slot 'keyword' contains the keyword.
The operator **SearchOperator** returns the value TRUE if the parameters are correct and the helpfiles are available. Otherwise an exception handling is raised.

---

**Parallelization Information**

SearchOperator is processed **completely exclusively** without parallelization.

---

**Possible Predecessor Functions**

- GetKeywords
- See Also: GetKeywords, GetOperatorInfo, GetParamInfo

---

**Module**

Image / region / XLD management

### 11.4 Operating-System

<table>
<thead>
<tr>
<th>[out] double Seconds</th>
<th><strong>HSystemX.CountSeconds</strong> ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void <strong>HOperatorSetX.CountSeconds</strong> ([out] double Seconds )</td>
<td></td>
</tr>
</tbody>
</table>

**Elapsed processing time since the last call of CountSeconds.**

The operator CountSeconds helps to measure the runtime. The first call of the procedure normally returns a zero. Each further call returns the processing time which has elapsed in the meantime in seconds. The C-function ”’clock’” is used for the measurement.

---

**Attention**

The time measurement is not exact and depends on the load of the computer.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Seconds</strong> (output control) ..................real ~ double</td>
</tr>
</tbody>
</table>

Processtime since the program start.

---

**Example**

```c
count_seconds(:::Start) >
/* aktion to be measured */
count_seconds(:::End) >
eval(:::End - Start:RunTime).
```

---

**Result**

The operator CountSeconds always returns the value TRUE.

---

**Parallelization Information**

CountSeconds is **reentrant** and processed **without** parallelization.

---

**Module**

System

<table>
<thead>
<tr>
<th>void <strong>HSystemX.SystemCall</strong> ([in] String Command )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void <strong>HOperatorSetX.SystemCall</strong> ([in] String Command )</td>
</tr>
</tbody>
</table>

**Executes a system command.**

The operator SystemCall executes the system command specified by the string pointed to by Command (C-procedure ”’system’”). If the string is empty, an interactive shell will be started (’csh -i’).
Parameter

- **Command** (input control) .......................................................... string \( \Rightarrow \) String
  Command to be called by the system.
  
  **Default Value:** ‘ls’

**Result**

If the entered operator can be executed by the system, the operator `SystemCall` returns the value TRUE. Otherwise an exception will be raised.

**Parallelization Information**

`SystemCall` is reentrant and processed without parallelization.

**Possible Predecessor Functions**

- `CountSeconds`

**See Also**

- `CountSeconds`, `WaitSeconds`

**Module**

- `System`

```cpp
void HSystemX.WaitSeconds ([in] double Seconds )
void HOperatorSetX.WaitSeconds ([in] double Seconds )
```

Delaying the execution of the program.

The operator `WaitSeconds` delays the execution by the number of seconds indicated in `Seconds`.

**Attention**

- **Seconds** (input control) .......................................................... real \( \Rightarrow \) double
  Number of seconds by which the execution of the program will be delayed.
  
  **Default Value:** 10
  
  **Restriction:** \((Seconds \geq 0)\)

**Result**

The operator `WaitSeconds` always returns the value TRUE.

**Parallelization Information**

`WaitSeconds` is reentrant and processed without parallelization.

**Possible Successor Functions**

- `SystemCall`

**See Also**

- `SystemCall`, `CountSeconds`

**Module**

- `System`

### 11.5 Parallelization

```cpp
void HSystemX.CheckParHwPotential ([in] long AllInpPars )
void HOperatorSetX.CheckParHwPotential ([in] long AllInpPars )
```

Check hardware regarding its potential for parallel processing.

`CheckParHwPotential` is necessary for an efficient automatic parallelization, which is used by HALCON to better utilize multiprocessor hardware in order to speed up the processing of operators. As the parallelization of operators is done automatically, there is no need for the user to explicitly prepare or change programs for
their parallelization. Thus, all HALCON-based programs can be used unchanged on multiprocessor hardware and nevertheless utilize the potential of parallel hardware. CheckParHwPotential checks a given hardware with respect to a parallel processing of HALCON operators. At this, it examines every operator, which can be sped up in principle by an automatic parallelization. Each examined operator is processed several times - both sequentially and in parallel - with a changing set of input parameter values/images. The latter helps to evaluate dependencies between an operator’s input parameter characteristics (e.g. the size of an input image) and the efficiency of its parallel processing. At this, AllInpPars is used in the following way: In the normal case, i.e. if AllInpPars contains the default value 0 (“false”), only those input parameters are examined which are supposed to show influence on the processing time. Other parameters are not examined so that the whole process is sped up. However, in some rare cases, the internal implementation of a HALCON operator might change from one HALCON release to another. Then, a parameter which did not show any direct influence on the processing time in former releases, may now show such an influence. In this case it is necessary to set AllInpPars to 1 (“true”) in order to force the examination of all input parameters. If this happens, the HALCON release notes will most likely contain an appropriate note about this fact. Overall, CheckParHwPotential performs several test loops and collects a lot of hardware-specific informations, which enable HALCON to optimize the automatic parallelization for a given hardware. The hardware information is stored so that it can be used again in future HALCON sessions. Thus, it is sufficient, to start CheckParHwPotential once on each multiprocessor machine that is used for parallel processing. Of course, it should be started again, if the hardware of the machine changes, for example, by installing a new cpu, or if the operating system of the machine changes, or if the machine gets a new host name. The latter is necessary, because HALCON identifies the machine-specific parallelization information by the machine’s host name. If the same multiprocessor machine is used with different operating systems, such as Windows and Linux, it is necessary to start CheckParHwPotential once for each operating system in order to correctly measure the rather strong influence of the operating system on the potential of exploiting multiprocessor hardware. Under Windows, HALCON stores the parallelization knowledge, which belongs to a specific machine, in the machine’s registry. At this, it uses a machine-specific registry key, which can be used by different users simultaneously. In the normal case, this key can be written or changed by any user under Windows NT. However, under Windows 2000 the key may only be changed by users with administrator privileges or by users which at least belong to the “power user” group. For all other users CheckParHwPotential shows no effect (but does not return an error). Under Linux/UNIX the parallelization information is stored in a file in the HALCON installation directory (HALCONROOT). Again this means that CheckParHwPotential must be called by users with the appropriate privileges, here by users which have write access to the HALCON directory. If HALCON is used within a network under Linux/UNIX, the denoted file contains the information about every computer in the network for which the hardware check has been successfully completed.

During its test loops CheckParHwPotential has to start every examined operator several times. Thus, the processing of CheckParHwPotential can take rather a long time. CheckParHwPotential bases on the automatic parallelization of operators which is exclusively supported by Parallel HALCON. Thus, CheckParHwPotential always returns an appropriate error, if it used with a non-parallel HALCON version. CheckParHwPotential must be called by users with the appropriate privileges for storing the parallelization information permanently (see the operator’s description above for more details about this subject).

**Parameter**

- **AllInpPars** (input control) 
  Check every input parameter?
  Default Value: 0
  Value List: AllInpPars ∈ {0, 1}

**Result**

CheckParHwPotential returns TRUE if all parameters are correct.

**Parallelization Information**

CheckParHwPotential is local and processed completely exclusively without parallelization.

**Possible Successor Functions**

StoreParKnowledge

**See Also**

StoreParKnowledge, LoadParKnowledge

**Module**

System
Load knowledge about automatic parallelization from file.

LoadParKnowledge supports the automatic parallelization of HALCON operators, which is used to better utilize multiprocessor hardware in order to speed up the processing of operators. To parallelize the processing of operators automatically HALCON needs some specific knowledge about the used hardware. This hardware-specific knowledge can be obtained by using the operator CheckParHwPotential. In the normal case, HALCON stores this knowledge in a specific file in the HALCON installation directory (Linux/UNIX) or within the “registry” (Windows). This enables HALCON to use the knowledge again later on. With LoadParKnowledge it is possible to load this knowledge explicitly from an ASCII file. At this, FileName denotes the name of this file (incl. path and file extension). The file must conform to a specific syntax and must have been stored beforehand by using StoreParKnowledge. While reading the file LoadParKnowledge checks whether its content was written for the currently used computer and whether the contained parallelization information regards the currently used HALCON version (and revision). If this is the case, LoadParKnowledge adopts the information so that it will also be used with further HALCON sessions. Otherwise, the information is ignored and LoadParKnowledge returns an appropriate error message.

Parameter

- **FileName** (input control) ........................................ filename.named ~ String
  Name of parallelization knowledge file.

  Default Value: ”

Result

LoadParKnowledge returns TRUE if all parameters are correct.

Parallelization Information

LoadParKnowledge is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

StoreParKnowledge

See Also

StoreParKnowledge, CheckParHwPotential

Module

System

---

Store knowledge about automatic parallelization in file.

StoreParKnowledge supports the automatic parallelization of HALCON operators, which is used to better utilize multiprocessor hardware in order to speed up the processing of operators. To parallelize the processing of operators automatically HALCON needs some specific knowledge about the used hardware. This hardware-specific knowledge can be obtained by calling the operator CheckParHwPotential. There, HALCON stores the knowledge in a specific file in the HALCON installation directory (Linux/UNIX) or within the “registry” (Windows). This enables HALCON to use the knowledge again later on. With StoreParKnowledge it is possible to store this knowledge explicitly as an ASCII file. At this, FileName denotes the name of this file (incl. path and file extension). The stored knowledge can be read again later on by using LoadParKnowledge.

Parameter

- **FileName** (input control) ........................................ filename.named ~ String
  Name of parallelization knowledge file.

  Default Value: ”

Result

StoreParKnowledge returns TRUE if all parameters are correct.

Parallelization Information

StoreParKnowledge is local and processed completely exclusively without parallelization.
11.6 Parameters

Information concerning the currently used HALCON system parameter.

The operator GetSystem returns information concerning the currently activated HALCON system parameters. Some of these parameters can be changed dynamically by using the operator SetSystem. They are marked by a + in the list below. By passing the string '?' as the parameter Query, the names of all system parameters are provided with Information.

The following system parameters can be queried:

**Versions**
- 'version': HALCON version number, e.g.: 6.0
- 'last_update': Date of creation of the HALCON library
- 'revision': Revision number of the HALCON library, e.g.: 1

**Upper Limits**
- 'max_contour_length': Maximum number of contour respectively polygone control points of a region.
- 'max_images': Maximum total of images.
- 'max_channels': Maximum number of channels of an image.
- 'max_obj_per_par': Maximum number of image objects which may be used during one call up per parameter
- 'max_inp_obj_par': Maximum number of input parameters.
- 'max_outp_obj_par': Maximum number of output parameters.
- 'max_inp_ctrl_par': Maximum number of input control parameters.
- 'max_outp_ctrl_par': Maximum number of output control parameters.
- 'max_window': Maximum number of windows.
- 'max_window_types': Maximum number of window systems.
- 'max_proc': Maximum number of HALCON procedures (system defined + user defined).

**Graphic**
- '+flush_graphic': Determines, whether the flush operation is called or not after each HALCON procedure.
- '+int2_bits': Number of significant bits of int2 images. This number is used when scaling the gray values. If the values is -1 the gray values will be automatically scaled (default).
- '+int_zooming': Determines if the zooming of images is done with integer arithmetic or with floating point arithmetic.
- '+draw_mode': (no description available)
- '+ignore_colormap': (no description available)
- '+backing_store': Storage of the window contents in case of overlaps.
- '+icon_name': (no description available)
- '+window_name': (no description available)
+`default_font'`: Name of the font to set at opening the window.
+`single_lut'`: (no description available)
+`update_lut'`: (no description available)
+`x_package'`: Number of bytes which are sent to the X server during each transfer of data.
+`num_gray_4'`: Number of colors reserved under X Windows concerning the output of graylevels (DispChannel) on a machine with 4 bitplanes (16 colors).
+`num_gray_6'`: Number of colors reserved under X Window concerning the output of graylevels (DispChannel) on a machine with 6 bitplanes (64 colors).
+`num_gray_8'`: Number of colors reserved under X Window concerning the output of graylevels (DispChannel) on a machine with 8 bitplanes (256 colors).
+`num_gray_percentage'`: HALCON reserves a certain amount of the available colors under X Window for the representation of graylevels (DispImage). This shall interfere with other X applications as little as possible. However, if HALCON does not succeed in reserving a miminum percentage of `num_gray_percentage' of the necessary colors on the X server, a certain amount of the lookup-table will be claimed for the HALCON graylevels regardless of the consequences for other applications. This may result in undesired shifts of color when switching between HALCON windows and windows of other applications, or if (outside HALCON) a window-dump is generated. The number of the real graylevels to be reserved depends on the number of available bitplanes on the outputmachine (see also `num_gray_*`). Naturally no colors will be reserved on monochrome machines - the graylevels will instead be dithered when displayed. If graylevel displays are used, only different shades of gray will be applied (`black`, `white`, `gray`, etc.). Machines with 2 or 3 bitplanes will be considered monochrome machines, machines with 5 (7) bitplanes like machines with 4 (6) bitplanes, and machines having more than 8 bitplanes like machines with 8 bitplanes. A special case are machines providing a 24 bit display (true color machines). Naturally no colors are reserved for the display of graylevels in this case. Note: Before the first window on a machine with x bitplanes is opened, num_gray_x indicates the number of colors which have to be reserved for the display of graylevels, afterwards, however, it will indicate the number of colors which actually have been reserved.
+`num_graphic_percentage'`: (no description available)
+`num_graphic_2'`: Number of the HALCON graphic colors reserved under X Window (for DispRegion etc.) on a machine with 2 bitplanes (4 colors).
+`num_graphic_4'`: Number of the HALCON graphic colors reserved under X Window (for DispRegion etc.) on a machine with 4 bitplanes (16 colors).
+`num_graphic_6'`: Number of the HALCON graphic colors reserved under X Window (for DispRegion etc.) on a machine with 6 bitplanes (64 colors).
+`num_graphic_8'`: Number of the HALCON graphic colors reserved under X Window (for DispRegion etc.) on a machine with 8 bitplanes (256 colors).

Image Processing
+`neighborhood'`: Using the 4 or 8 neighborhood.
+`only_lines'`: (no description available)
+`init_new_image'`: Initialization of images before applying grayvalue transformations.
+`no_object_result'`: Behavior for an empty object lists.
+`empty_region_result'`: Reaction of procedures concerning input objects with empty regions which actually are not useful for such objects (e.g. certain region features, segmentation, etc.). Possible return values:
  - `true`: the error will be ignored if possible
  - `false`: the procedure returns FALSE
  - `fail`: the procedure returns FAIL
  - `void`: the procedure returns VOID
  - `exception`: an exception is raised
+`store_empty_region'`: Storing of objects with empty regions.
+`clip_region'`: Clipping of output regions so that they fit the global image size.
+`width'`: Global maximum image width - in Standard-HALCON this value contains the maximum image width of all HALCON image objects which are currently stored in memory. In Parallel HALCON this value contains the maximum image width of all HALCON image objects which are or were in memory since the start of the current HALCON session (this also includes objects which may be deleted meanwhile).
11.6. PARAMETERS

'height': Global maximum image height - in Standard-HALCON this value contains the maximum image height of all HALCON image objects which are currently stored in memory. In Parallel HALCON this value contains the maximum image height of all HALCON image objects which are or were in memory since the start of the current HALCON session (this also includes objects which may be deleted meanwhile).

'obj_images': Current number of grayvalue components per image object.

'current_runlength_number': Currently used number of chords which can be used for the encoding of regions.

Parallelization

'parallelize_operators': Determines whether Parallel HALCON uses an automatic parallelization to speed up the processing of operators on multiprocessor machines.

'reentrant': Denotes whether Parallel HALCON currently supports reentrancy (default case), or whether this feature has been switched off. Reentrancy is necessary for the automatic parallelization of Parallel HALCON and for calling and processing multiple HALCON operators in parallel within multithreaded applications.

'processor_num': Returns the number of processors which Parallel HALCON has found on the hardware it is running on. This also indicates the number of processors which is used by Parallel HALCON for the automatic parallelization of operators.

File

'flush_file': Buffering of file output.

Directories

'image_dir': Path which will searched for the image file after the default directory (see also: ReadImage).

'lut_dir': Path for the default directory for color tables (see also: SetLut).

'reference_dir': Path for the default directory concerning the postscript HALCON documentation.

'help_dir': Path for the default help directory for the online help files:
{ german,english }. { hlp,sta,idx,num,key }.

Other

'do_low_error': Flag, if low level error should be printed.

'num_proc': Total number of the available HALCON procedures ('num_sys_proc' + 'num_user_proc').

'num_sys_proc': Number of the system procedures (supported procedures).

'num_user_proc': Number of the user defined procedures (see also 'Extension Packages' manual).

'byte_order': Byte order of the processor ('msb_first' or 'lsb_first').

'operating_system': Name of the operating system of the computer on which the HALCON process is being executed.

'operating_system_version': Version number of the operating system of the computer on which the HALCON process is being executed.

'alloctmp_single_block': Flag for kind of temporary memory management.

'temp_mem': Amount of temporary memory used by the last operator. This feature is only supported by Standard-HALCON. Instead of this, Parallel HALCON returns the amount of temporary memory used by GetSystem itself.

'language': Language used for error messages ('english' or 'german').

Attention

Parameter

Desired system parameter.

Default Value: 'width'

Value List: Query ∈ {'?', 'version', 'last_update', 'revision', 'max_images', 'max_channels', 'max_obj_per_par', 'max_inp_obj_par', 'max_outp_obj_par', 'max_inp_ctrl_par', 'max_outp_ctrl_par', 'max_window', 'max_window_types', 'max_proc', 'flush_graphic', 'int2_bits', 'intzooming', 'draw_mode', 'ignore_colormap', 'backing_store', 'icon_name', 'window_name', 'default_font', 'single_lut', 'update_lut', 'x_package', 'num_gray_4', 'num_gray_6', 'num_gray_8', 'num_gray_percentage', 'num_graphic_2', 'num_graphic_4', 'num_graphic_6', 'num_graphic_8', 'num_graphic_percentage', 'neighborhood',

HALCON 6.0
The operator `GetSystem` returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

### Parallelization Information

`GetSystem` is local and processed completely exclusively without parallelization.

### Possible Predecessor Functions

- `ResetObjDb`

### Possible Successor Functions

- `SetSystem`

### See Also

- `SetSystem`

### Module

- `System`

---

**Setting of HALCON system parameters.**

The operator `SetSystem` allows to change different system parameters with relation to the runlength.

Available system parameters:

- **'neighborhood'**: This parameter is used with all procedures which examine neighborhood relations: `Connection`, `GetRegionContour`, `GetRegionChain`, `GetRegionPolygon`, `GetRegionThickness`, `Boundary`, `PaintRegion`, `DispRegion`, `FillUp`, `Contlength`, `ShapeHistoAll`.
  Value: 4 or 8
  default: 8

- **'default_font'**: Whenever a window is opened, a font will be set for the text output, whereby the 'default_font' will be used. If the preset font cannot be found, another fontname can be set before opening the window.
  Value: Filename of the fonts
  default: fixed

- **'single_lut'**: (no description available) default: 'false'

- **'update_lut'** Determines whether the HALCON color tables are adapted according to their environment or not.
  Value: 'true' or 'false'
  default: 'false'

- **'image_dir'**: Image files (e.g. `ReadImage` and `ReadSequence`) will be looked for in the currently used directory and in 'image_dir' (if no absolute paths are indicated). More than one directory name can be indicated (searchpaths), seperated by semicolons (Windows NT) or colons (Unix). The path can also be determined using the environment variable HALCONIMAGES.
  Value: Name of the filepath
  default: '$HALCONROOT/images' bzw. '%HALCONROOT%/images'

- **'lut_dir'**: Color tables (SetLut) which are realized as an ASCII-file will be looked for in the currently used directory and in 'lut_dir' (if no absolute paths are indicated). If HALCONROOT is set, HALCON will search the color tables in the sub-directory "lut".
  Value: Name of the filepath
  default: '$HALCONROOT/lut' bzw. '%HALCONROOT%/lut'
'reference_dir': The HTML and postscript sources of the HALCON documentation (especially the HALCON manual itself) will be looked for in the currently used directory or in 'reference_dir'. This system parameter is necessary for example using the operator DispInfo. This parameter can also be set by the environment variable HALCONROOT before initializing HALCON. In this case the variable must indicate the directory above the help directories (that is the HALCON-Homedirectory): e.g.: '/usr/local/halcon'
Value: Name of the filepath
default: '$HALCONROOT/doc/ps/reference/c' bzw. '%HALCONROOT%/doc/ps/reference/c'

'help_dir': The online text files german or english.hlp, .sta, .key .num and .idx will be looked for in the currently used directory or in 'help_dir'. This system parameter is necessary for instance using the operators GetOperatorInfo and GetParamInfo. This parameter can also be set by the environment variable HALCONROOT before initializing HALCON. In this case the variable must indicate the directory above the help directories (that is the HALCON-Homedirectory): e.g.: '/usr/local/halcon'
Value: Name of the filepath
default: '$HALCONROOT/help' bzw. '%HALCONROOT%/help'

'only_lines': (no description available) default: 'true'

'init_new_image': Determines whether new images shall be set to 0 before using filters. This is not necessary if always the whole image is filtered if the data of not filtered image areas are unimportant.
Value: 'true' or 'false'
default: 'true'

'no_object_result': Determines how operations processing iconic objects shall react if the object tuplet is empty (= no objects). Available values for Value:
'true': the error will be ignored
'false': the procedure returns FALSE
'fail': the procedure returns FAIL
'void': the procedure returns VOID
'exception': an exception is raised
default: 'true'

'empty_region_result': Controls the reaction of procedures concerning input objects with empty regions which actually are not useful for such objects (e.g. certain region features, segmentation, etc.). Available values for Value:
'true': the error will be ignored if possible
'false': the procedure returns FALSE
'fail': the procedure returns FAIL
'void': the procedure returns VOID
'exception': an exception is raised
default: 'true'

'store_empty_region': Quite a number of operations will lead to the creation of objects with an empty region (= no image points) (e.g. Intersection,Threshold, etc.). This parameter determines whether the object with an empty region will be returned as a result ('true') or whether it will be ignored ('false') that is no result will be returned.
Value: 'true' or 'false'
default: 'true'

'backing_store': Determines whether the window content will be refreshed in case of overlapping of the windows. Some implementations of X Window are faulty; in order to avoid these errors, the storing of contents can be deactivated. It may be recommendable in some cases to deactivate the security mechanism, if e.g. performance / memory is what matters.
Value: true or false
default: true

'flush_graphic': After each HALCON procedure which creates a graphic output, a flush operation will be executed in order to display the data immediately on screen. This is not necessary with all programs (e.g. if everything is done with the help of the mouse). In this case 'flush_graphic' can be set to 'false' to improve the runlength.
Value: 'true' or 'false'
default: 'true'

'flush_file': This parameter determines whether the output into a file (also to the terminal) shall be buffered or not. If the output is to be buffered, in general the data will be displayed on the terminal only after entering
the operator FnewLine.
Value: 'true' or 'false'
default: 'true'

'x_package': The output of image data via the network may cause errors owing to the heavy load on the computer
or on the network. In order to avoid this, the data are transmitted in small packages. If the computer is used
locally, these units can be enlarged at will. This can lead to a notably improved output performance.
Value: package size (in bytes)
default: 20480

'int2_bits': Number of significant bits of int2 images. This number is used when scaling the gray values. If the
values is -1 the gray values will be automatically scaled (default).
Value: -1 or 9..15
default: -1

'num_gray_4': Number of colors to be reserved under X Window to allow the output of graylevels disp_channel
on a machine with 4 bitplanes (16 colors).
Attention! This value may only be changed before the first window has been opened on the machine.
Value: 2 - 12
default: 8

'num_gray_6': Number of colors to be reserved under X Window to allow the output of graylevels disp_channel
on a machine with 6 bitplanes (64 colors).
Attention! This value may only be changed before the first window has been opened on the machine.
Value: 2 - 62
default: 50

'num_gray_8': Number of colors to be reserved under X Window to allow the output of graylevels disp_channel
on a machine with 8 bitplanes (256 colors).
Attention! This value may only be changed before the first window has been opened on the machine.
Value: 2 - 254
default: 140

'num_gray_percentage': Under X Window HALCON reserves a part of the available colors for the representation
of gray values (DispChannel). This shall interfere with other X applications as little as possible.
However, if HALCON does not succeed in reserving a minimum percentage of 'num_gray_percentage' of
the necessary colors on the X server, a certain amount of the lookup table will be claimed for the HALCON
graylevels regardless of the consequences. This may result in undesired shifts of color when switching
between HALCON windows and windows of other applications, or (outside HALCON) if a window-dump is
generated. The number of the real graylevels to be reserved depends on the number of available bitplanes on
the output machine (see also 'num_gray_*'). Naturally no colors will be reserved on monochrome machines -
the graylevels will instead be dithered when displayed. If grayscale-displays are used, only different shades
of gray will be applied ('black', 'white', 'gray', etc.). Machines with 2 or 3 bitplanes will be considered
monochrome machines, machines with 5 (7) bitplanes like machines with 4 (6) bitplanes, and machines hav-
ing more than 8 bitplanes like machines with 8 bitplanes. A special case are machines providing a 24 bit
display (true color machines). Naturally no colors are reserved for the display of graylevels in this case.
Note: This value may only be changed before the first window has been opened on the machine. For before
opening the first window on a machine with x bitplanes, num_gray_x indicates the number of colors which
have to be reserved for the display of graylevels, afterwards, however, it will indicate the number of colors
which actually have been reserved.
Value: 0 - 100
default: 30

'num_graphic_percentage': (no description available) default: 60

draw_mode': (no description available) default: 'complement'

'num_gray_2': Number of the graphic colors to be reserved by HALCON under X Window (concerning the
operators DispRegion etc.) on a machine with 2 bitplanes (4 colors).
Attention: This value may only be changed before the first window has been opened on the machine.
Value: 0 - 2
default: 2
'num_graphic_4': Number of the graphic colors to be reserved by HALCON under X Window (concerning the operators DispRegion etc.) on a machine with 4 bitplanes (16 colors).
Attention: This value may only be changed before the first window has been opened on the machine.
Value: 0 - 14
default: 5

'num_graphic_6': Number of the graphic colors to be reserved by HALCON under X Window (concerning the operators DispRegion etc.) on a machine with 6 bitplanes (64 colors).
Attention: This value may only be changed before the first window has been opened on the machine.
Value: 0 - 62
default: 10

'num_graphic_8': Number of the graphic colors to be reserved by HALCON under X Window (concerning the operators DispRegion etc.) on a machine with 8 bitplanes (256 colors).
Attention: This value may only be changed before the first window has been opened on the machine.
Value: 0 - 64
default: 20

'graphic_colors' HALCON reserves the first num_graphic_x colors form this list of color names as graphic colors. As a default HALCON uses this same list which is also returned by using QueryAllColors. However, the list can be changed individually: hereby a tuplet of color names will be returned as value. It is recommendable that such a tuplet always includes the colors 'black' and 'white', and optionally also 'red', 'green' and 'blue'. If 'default' is set as Value, HALCON returns to the initial setting. Note: On graylevel machines not the first x colors will be reserved, but the first x shades of gray from the list.
Attention: This value may only be changed before the first window has been opened on the machine.
Value: Tuplets of X Window color names
default: see also query_all_colors

'current_runlength_number': Regions will be stored internally in a certain runlengthcode. This parameter can determine the maximum number of chords which may be used for representing a region. Please note that some procedures raise the number on their own if necessary. The value can be enlarged as well as reduced.
Value: maximum number of chords
default: 50000

'clip_region': Determines whether the regions of iconic objects of the HALCON database will be clipped to the currently used image size or not. This is the case for example in procedures like GenCircle, GenRectangle1 or Dilation1.
See also: ResetObjDb
Value: 'true' or 'false'
default: 'true'

'do_low_error' Determines whether the HALCON should print low level error or not.
Value: 'true' or 'false'
default: 'false'

'reentrant' Determines whether HALCON must be reentrant for being used within a parallel programming environment (e.g. a multithreaded application). This parameter is only of importance for Parallel HALCON, which can process several operators concurrently. Thus, the parameter is ignored by the sequentially working HALCON-Version. If it is set to 'true', Parallel HALCON internally uses synchronization mechanisms to protect shared data objects from concurrent accesses. Though this is inevitable with any effectively parallel working application, it may cause undesired overhead, if used within an application which works purely sequentially. The latter case can be signalled by setting 'reentrant' to 'false'. This switches off all internal synchronization mechanisms and thus reduces overhead. Of course, Parallel HALCON then is no longer thread-safe, which causes another side-effect: Parallel HALCON will then no longer use the internal parallelization of operators, because this needs reentrancy. Setting 'reentrant' to 'true' resets Parallel HALCON to its default state, i.e. it is reentrant (and thread-safe) and it uses the automatic parallelization to speed up the processing of operators on multiprocessor machines.
Value: 'true' or 'false'
default: Parallel HALCON: 'true', otherwise: 'false'

'parallelize_operators' Determines whether Parallel HALCON uses an automatic parallelization to speed up the processing of operators on multiprocessor machines. This feature can be switched off by setting 'parallelize_operators' to 'false'. Even then Parallel HALCON will remain reentrant (and thread-safe), unless the parameter 'reentrant' is changed via SetSystem accordingly. Changing 'parallelize_operators' can be
helpful, for example, if HALCON-operators are called by a multithreaded application, which also does the scheduling and load-balancing of operators and data by itself. Then it may be undesired that HALCON performs additional parallelization steps, which may disturb the application’s scheduling and load-balancing concepts. The parameter 'parallelize operators' is only supported by Parallel HALCON and thus ignored by the sequentially working HALCON-Version.

Value: 'true' or 'false'

default: Parallel HALCON: 'true', otherwise: 'false'

'affectmp single block' Kind of temporary memory management. Value: 'true' or 'false'

default: 'false'

'language' Language used for error messages. Value: 'english' or 'german'. default: 'english'

Attention

Parameter

- **Systemparameter** (input control) ..................string(-array) ~ VARIANT ( string )
  Name of the system parameter to be changed.
  
  **Default Value:** 'image dir'
  
  **Value List:** Systemparameter ∈ {'neighborhood', 'default font', 'single lut', 'update lut', 'image dir', 'lut dir', 'reference dir', 'help dir', 'only lines', 'init new image', 'no object result', 'empty region result', 'store empty region', 'backing store', 'flush graphic', 'flush file', 'x package', 'int2 bits', 'icon name', 'num gray 4', 'num gray 6', 'num gray 8', 'num gray percentage', 'num graphic 2', 'num graphic 4', 'num graphic 6', 'num graphic 8', 'num graphic percentage', 'draw mode', 'int zooming', 'ignore colormap', 'graphic colors', 'current runlength number', 'clip region', 'update lut', 'do low error', 'reentrant', 'parallelize operators', 'affectmp single block', 'language'}

- **Value** (input control) ..................string(-array) ~ VARIANT ( integer, real, string )
  New value of the system parameter.
  
  **Default Value:** 'true'
  
  **Value Suggestions:** Value ∈ {'true', 'false', 0, 4, 8, 100, 140, 255}

Result

The operator SetSystem returns the value TRUE if the parameters are correct. Otherwise an exception will be raised.

Parallelization Information

SetSystem is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

ResetObjDb, GetSystem, SetCheck

See Also

GetSystem, SetCheck

Module

System

### 11.7 Serial

```c
void HSerialX.ClearSerial ( [in] String Channel )
void HOperatorSetX.ClearSerial ( [in] long SerialHandle,
                                [in] String Channel )
```

Clear the buffer of a serial connection.

ClearSerial discards data written to the serial device referred to by SerialHandle, but not transmitted (Channel = "output"), or data received, but not read (Channel = "input"), or performs both these operations at once (Channel = "in_out").
11.7. SERIAL

Parameter

- **SerialHandle** (input control) Serial interface handle.
- **Channel** (input control) Buffer to be cleared.
  
  **Default Value:** 'input'
  
  **Value List:** Channel ∈ {'input', 'output', 'inout'}

Result

If the parameters are correct and the buffers of the serial device could be cleared, the operator `ClearSerial` returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

`ClearSerial` is reentrant and processed without parallelization.

Possible Predecessor Functions

OpenSerial

Possible Successor Functions

ReadSerial, WriteSerial

See Also

ReadSerial

Module

System

```c
void HMiscX.CloseAllSerials ( )
void HOperatorSetX.CloseAllSerials ( )
```

Close all serial devices.

`CloseAllSerials` closes all serial devices that have been opened with `OpenSerial`.

Result

`CloseAllSerials` returns always TRUE.

Parallelization Information

`CloseAllSerials` is reentrant and processed without parallelization.

Possible Predecessor Functions

OpenSerial

See Also

OpenSerial, CloseFile

Alternatives

CloseSerial

Module

System

```c
void HOperatorSetX.CloseSerial ([in] long SerialHandle )
```

Close a serial device.

`CloseSerial` closes a serial device that was opened with `OpenSerial`.

Parameter

- **SerialHandle** (input control) Serial interface handle.

Result

If the parameters are correct and the device could be closed, the operator `CloseSerial` returns the value TRUE. Otherwise an exception is raised.
Parallelization Information

CloseSerial is *reentrant* and processed *without* parallelization.

Possible Predecessor Functions

OpenSerial

See Also

OpenSerial, CloseFile

Module

System

Get the parameters of a serial device.

GetSerialParam returns the current parameter settings of the serial device passed in SerialHandle. For a description of the parameters of a serial device, see SetSerialParam.

Parameter

- **SerialHandle** (input control) ................................. `serial_id`  
  Serial interface handle.
- **BaudRate** (output control) ................................. `integer`  
  Speed of the serial interface.
- **DataBits** (output control) ................................. `integer`  
  Number of data bits of the serial interface.
- **FlowControl** (output control) ............................... `string`  
  Type of flow control of the serial interface.
- **Parity** (output control) ................................. `string`  
  Parity of the serial interface.
- **StopBits** (output control) ................................. `integer`  
  Number of stop bits of the serial interface.
- **TotalTimeOut** (output control) ......................... `integer`  
  Total timeout of the serial interface in ms.
- **InterCharTimeOut** (output control) ...................... `integer`  
  Inter-character timeout of the serial interface in ms.

Result

If the parameters are correct and the parameters of the device could be read, the operator GetSerialParam returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

GetSerialParam is *reentrant* and processed *without* parallelization.

Possible Predecessor Functions

OpenSerial

Possible Successor Functions

GetSerialParam, ReadSerial, WriteSerial

See Also

SetSerialParam

Module

System
null
If the parameters are correct and the read from the device was successful, the operator ReadSerial returns
the value TRUE. Otherwise an exception is raised.

ReadSerial is reentrant and processed without parallelization.

Set the parameters of a serial device.

SetSerialParam can be used to set the parameters of a serial device. The parameter BaudRate determines
the input and output speed of the device. It should be noted that not all devices support all possible speeds.
The number of sent and received data bits is set with DataBits. The parameter FlowControl determines if
and what kind of data flow control should be used. In the latter case, a choice between software control
('xon_xoff') and hardware control ('cts_rts', 'dtr_dsr') can be made. If and what kind of parity check of the transmitted
data should be performed can be determined by Parity. The number of stop bits sent is set with StopBits.
Finally, two timeout for reading from the serial device can be set. The parameter TotalTimeOut determines the
maximum time, which may pass in ReadSerial until the first character arrives, independent of the actual number
of characters requested. The parameter InterCharTimeOut determines the time which may pass between the
reading of individual characters, if multiple characters are requested with ReadSerial. If one of the timeouts is
set to -1, a read waits an arbitrary amount of time for the arrival of characters. Thus, on Windows NT systems,
a total timeout of TotalTimeOut + n*InterCharTimeOut results if n characters are to be read. On Unix systems,
only one of the two timeouts can be set. Thus, if both timeouts are passed larger than -1, only the total timeout
is used. The unit of both timeouts is milliseconds. For each parameter, the current values can be left in effect by
passing 'unchanged'.

- **SerialHandle** (input control) .................................................. serial_id ~ HSerialX/long
  Serial interface handle.
- **BaudRate** (input control) .................................................. integer ~ VARIANT( integer, string )
  Speed of the serial interface.
  Default Value : 'unchanged'
  Value List : BaudRate ∈ {50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200,
  38400, 57600, 76800, 115200, 153600, 230400, 307200, 460800, 'unchanged'}
- **DataBits** (input control) .................................................. integer ~ VARIANT( integer, string )
  Number of data bits of the serial interface.
  Default Value : 'unchanged'
  Value List : DataBits ∈ {5, 6, 7, 8, 'unchanged'}
- **FlowControl** (input control) ............................................... string ~ String
  Type of flow control of the serial interface.
  Default Value : 'unchanged'
  Value List : FlowControl ∈ {'none', 'xon_xoff', 'cts_rts', 'dtr_dsr', 'unchanged'}
Parity (input control) string ~ String
Parity of the serial interface.
Default Value: 'unchanged'
Value List: Parity ∈ { 'none', 'odd', 'even', 'unchanged' }

StopBits (input control) integer ~ VARIANT(integer, string)
Number of stop bits of the serial interface.
Default Value: 'unchanged'
Value List: StopBits ∈ {1, 2, 'unchanged' }

TotalTimeOut (input control) integer ~ VARIANT(integer, string)
Total timeout of the serial interface in ms.
Default Value: 'unchanged'
Value Suggestions: TotalTimeOut ∈ {-1, 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 'unchanged'

InterCharTimeOut (input control) integer ~ VARIANT(integer, string)
Inter-character timeout of the serial interface in ms.
Default Value: 'unchanged'
Value Suggestions: InterCharTimeOut ∈ {-1, 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 'unchanged'

If the parameters are correct and the parameters of the device could be set, the operator SetSerialParam returns the value TRUE. Otherwise an exception is raised.

SetSerialParam is reentrant and processed without parallelization.

OpenSerial, GetSerialParam

Possible Predecessor Functions

Possible Successor Functions

ReadSerial, WriteSerial

See Also

GetSerialParam

Module

System

void HSerialX.WriteSerial ([in] VARIANT Data )
void HOperatorSetX.WriteSerial ([in] long SerialHandle, [in] VARIANT Data )

Write to a serial connection.

WriteSerial writes the characters given in Data to the serial device given by SerialHandle. The data to be written is passed as a tuple of integers. This allows to write NUL characters, which would otherwise be interpreted as the end of a string. WriteSerial always waits until all data has been transmitted, i.e., a timeout for writing cannot be set.

Parameter

SerialHandle (input control) serial_id ~ HSerialX/long
Serial interface handle.

Data (input control) integer(-array) ~ VARIANT(integer)
Characters to write (as tuple of integers).

Result

If the parameters are correct and the write to the device was successful, the operator WriteSerial returns the value TRUE. Otherwise an exception is raised.

WriteSerial is reentrant and processed without parallelization.

Possible Predecessor Functions

OpenSerial
11.8 Sockets

```c
void HOperatorSetX.CloseSocket ([in] long Socket )
```

Close a socket.

CloseSocket closes a socket that was previously opened with OpenSocketAccept, OpenSocketConnect, or SocketAcceptConnect. For a detailed example, see OpenSocketAccept.

**Parameter**

- `Socket` (input control) Socket number.

**Parallelization Information**

CloseSocket is reentrant and processed without parallelization.

**See Also**

OpenSocketAccept, OpenSocketConnect, SocketAcceptConnect

---

```c
[out] String DataType HSocketX.GetNextSocketDataType ( )
```

Determine the HALCON data type of the next socket data.

GetNextSocketDataType returns the data type of the next data that are present on the socket `Socket` and returns it in `DataType`. The possible values for `DataType` are:

- `'no_data'`: No data are present.
- `'no_halcon_data'`: Some data are present, but they are not HALCON data.
- `'tuple'`: The next data is a tuple.
- `'region'`: The next data is a region object.
- `'image'`: The next data is an image object.
- `'xld_cont'`: The next data is an XLD contour.
- `'xld_poly'`: The next data is an XLD polygon.
- `'xld_para'`: The next data is an XLD parallel.
- `'xld_mod_para'`: The next data is a modified XLD parallel.
- `'xld_ext_para'`: The next data is an extended XLD parallel.

**Parameter**

- `Socket` (input control) Socket number.
- `DataType` (output control) Data type of next HALCON data.
Open a socket that accepts connection requests.

OpenSocketAccept opens a socket that accepts incoming connection requests by other HALCON processes. This operator is the necessary first step in the establishment of a communication channel between two HALCON processes. The socket listens for incoming connection requests on the port number given by Port. The accepting socket is returned in AcceptingSocket. OpenSocketAccept returns immediately without waiting for a request from another process, done by calling OpenSocketConnect in the other process. This allows multiple other processes to connect to the particular HALCON process that calls OpenSocketAccept. To accept an incoming connection request, SocketAcceptConnect must be called after another process has called OpenSocketAccept.

**Parameter**

- **Port** (input control) ................................................................. integer \(\sim\) long
  - Port number.
  - Default Value : 3000
  - Typical Range of Values : \(1024 \leq \text{Port} \leq 1024\)
  - Minimal Value Step : 1
  - Recommended Value Step : 1
- **AcceptingSocket** (output control) ........................................... socket id \(\sim\) HSocketX / long
  - Socket number.

**Example**

```c
/* Process 1 */
dev_set_colored (12)
open_socket_accept (3000, AcceptingSocket)
/* Busy wait for an incoming connection */
dev_error_var (Error, 1)
dev_set_check ('˜give_error')
OpenStatus := 5
while (OpenStatus # 2)
    socket_accept_connect (AcceptingSocket, 'false', Socket)
    OpenStatus := Error
    wait_seconds (0.2)
endwhile
dev_set_check ('give_error')
/* Connection established */
receive_image (Image, Socket)
threshold (Image, Region, 0, 63)
send_region (Region, Socket)
receive_region (ConnectedRegions, Socket)
area_center (ConnectedRegions, Area, Row, Column)
send_tuple (Socket, Area)
send_tuple (Socket, Row)
send_tuple (Socket, Column)
close_socket (Socket)
```

HALCON 6.0
close_socket (AcceptingSocket)

/* Process 2 */
dev_set_colored (12)
open_socket_connect (’localhost’, 3000, Socket)
read_image (Image, ’fabrik’)
send_image (Image, Socket)
receive_region (Region, Socket)
connection (Region, ConnectedRegions)
send_region (ConnectedRegions, Socket)
receive_tuple (Socket, Area)
receive_tuple (Socket, Row)
receive_tuple (Socket, Column)
close_socket (Socket)

--- Parallelization Information ---
OpenSocketAccept is reentrant and processed without parallelization.
--- Possible Successor Functions ---
SocketAcceptConnect
--- See Also ---
OpenSocketAccept, CloseSocket, SendImage, ReceiveImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple
--- Module ---
System

void HSocketX.OpenSocketConnect ([in] String HostName, [in] long Port )
void HOperatorSetX.OpenSocketConnect ([in] String HostName, [in] long Port, [out] long Socket )

Open a socket to an existing socket.
OpenSocketConnect opens a connection to an accepting socket on the computer HostName, which listens on port Port. The listening socket in the other HALCON process must have been created earlier with the operator OpenSocketAccept. The socket thus created is returned in Socket. To establish the connection, the HALCON process, in which the accepting socket resides, must call SocketAcceptConnect. For a detailed example, see OpenSocketAccept.

--- Parameter ---
.bmp
- **HostName** (input control) .................................................. string  ~ String
  Hostname of the computer to connect to.
- **Port** (input control) .................................................. integer  ~ long
  Port number.
  Default Value : 3000
  Typical Range of Values : 1024 ≤ Port ≤ 1024
  Minimal Value Step : 1
  Recommended Value Step : 1
- **Socket** (output control) .................................................. socket_id  ~ HSocketX / long
  Socket number.

--- Parallelization Information ---
OpenSocketConnect is reentrant and processed without parallelization.
--- Possible Successor Functions ---
SendImage, ReceiveImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple
--- See Also ---
OpenSocketAccept, SocketAcceptConnect, CloseSocket

Receive an image over a socket connection.

ReceiveImage reads an image object that was sent over the socket connection determined by Socket by another HALCON process using the operator SendImage. If no image has been sent, the HALCON process calling ReceiveImage blocks until enough data arrives. For a detailed example, see OpenSocketAccept.

Parameter

- **Image** (output iconic) .................................. image(-array)  \sim HImageX / HUntypedObjectX
  Received image.
- **Socket** (input control) .................................. socket_id  \sim long / HSocketX
  Socket number.

Parallelization Information

ReceiveImage is **reentrant** and processed **without** parallelization.

Possible Predecessor Functions

OpenSocketConnect, SocketAcceptConnect

See Also

See Also

SendImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple,
GetNextSocketDataType

Receive regions over a socket connection.

ReceiveRegion reads a region object that was sent over the socket connection determined by Socket by another HALCON process using the operator SendRegion. If no regions have been sent, the HALCON process calling ReceiveRegion blocks until enough data arrives. For a detailed example, see OpenSocketAccept.

Parameter

- **Region** (output iconic) .................................. region(-array)  \sim HRegionX / HUntypedObjectX
  Received regions.
- **Socket** (input control) .................................. socket_id  \sim long / HSocketX
  Socket number.

Parallelization Information

ReceiveRegion is **reentrant** and processed **without** parallelization.

Possible Predecessor Functions

OpenSocketConnect, SocketAcceptConnect

See Also

SendRegion, SendImage, ReceiveImage, SendTuple, ReceiveTuple,
GetNextSocketDataType
**ReceiveTuple**

Receive a tuple over a socket connection.

ReceiveTuple reads a tuple that was sent over the socket connection determined by **Socket** by another HALCON process using the operator **SendTuple**. If no tuple has been sent, the HALCON process calling **ReceiveTuple** blocks until enough data arrives. For a detailed example, see **OpenSocketAccept**.

**Parameters**

- **Socket** (input control) : socket id ~ HSocketX / long  
  Socket number.
- **Tuple** (output control) : string ~ String  
  Received tuple.

**Parallelization Information**

**ReceiveTuple** is **reentrant** and processed **without** parallelization.

**Possible Predecessor Functions**

**OpenSocketConnect**, **SocketAcceptConnect**

**See Also**

**SendTuple**, **SendImage**, **ReceiveImage**, **SendRegion**, **ReceiveRegion**, **GetNextSocketDataType**

**Module**

**System**

---

**ReceiveXld**

Receive an XLD object over a socket connection.

ReceiveXld reads an XLD object that was sent over the socket connection determined by **Socket** by another HALCON process using the operator **SendXld**. If no XLD object has been sent, the HALCON process calling **ReceiveXld** blocks until enough data arrives. For a detailed example, see **SendXld**.

**Parameters**

- **XLD** (output iconic) : xld(-array) ~ HXLDX / HUntypedObjectX  
  Received XLD object.
- **Socket** (input control) : socket_id ~ long / HSocketX  
  Socket number.

**Parallelization Information**

**ReceiveXld** is **reentrant** and processed **without** parallelization.

**Possible Predecessor Functions**

**OpenSocketConnect**, **SocketAcceptConnect**

**See Also**

**SendXld**, **SendImage**, **ReceiveImage**, **SendRegion**, **ReceiveRegion**, **SendTuple**, **ReceiveTuple**, **GetNextSocketDataType**

**Module**

**System**
**11.8. SOCKETS**

```
void HImageX.SendImage ([in] long Socket )
void HSocketX.SendImage ([in] HImageX Image )
void HOperatorSetX.SendImage ([in] IHObjectX Image, [in] long Socket )
```

Send an image over a socket connection.

SendImage sends an image object over the socket connection determined by `Socket`. The receiving HAL-CON process must call *ReceiveImage* to read the image from the socket. For a detailed example, see *OpenSocketAccept*.

Parameter

- **Image** (input iconic) ...................................... image(-array) ∼ HImageX / IHObjectX Image to be sent.
- **Socket** (input control) ........................................... socket_id ∼ long / HSocketX Socket number.

Parallelization Information

SendImage is *reentrant* and processed *without* parallelization.

Possible Predecessor Functions

OpenSocketConnect, SocketAcceptConnect

See Also

ReceiveImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple, GetNextSocketDataType

Module

System

```
void HRegionX.SendRegion ([in] long Socket )
void HSocketX.SendRegion ([in] HRegionX Region )
void HOperatorSetX.SendRegion ([in] IHObjectX Region, [in] long Socket )
```

Send regions over a socket connection.

SendRegion sends a region object over the socket connection determined by `Socket`. The receiving HAL-CON process must call *ReceiveRegion* to read the regions from the socket. For a detailed example, see *OpenSocketAccept*.

Parameter

- **Region** (input iconic) ...................................... region(-array) ∼ HRegionX / IHObjectX Regions to be sent.
- **Socket** (input control) ........................................... socket_id ∼ long / HSocketX Socket number.

Parallelization Information

SendRegion is *reentrant* and processed *without* parallelization.

Possible Predecessor Functions

OpenSocketConnect, SocketAcceptConnect

See Also

ReceiveRegion, SendImage, ReceiveImage, SendTuple, ReceiveTuple, GetNextSocketDataType

Module

System
Send a tuple over a socket connection.

SendTuple sends a tuple over the socket connection determined by Socket. The receiving HALCON process must call ReceiveTuple to read the tuple from the socket. For a detailed example, see OpenSocketAccept.

Parameter

- **Socket** (input control) .................................................. socket_id ~ HSocketX / long
  
  Socket number.

- **Tuple** (input control) .................................................. string ~ String
  
  Tuple to be sent.

Parallelization Information

SendTuple is reentrant and processed without parallelization.

Possible Predecessor Functions

OpenSocketConnect, SocketAcceptConnect

See Also

ReceiveTuple, SendImage, ReceiveImage, ReceiveRegion, GetNextSocketDataType

Module

System

Send an XLD object over a socket connection.

SendXld sends an XLD object over the socket connection determined by Socket. The receiving HALCON process must call ReceiveXld to read the XLD object from the socket.

Parameter

- **XLD** (input iconic) .................................................. xld(-array) ~ HXLDX / IHObjectX
  
  XLD object to be sent.

- **Socket** (input control) .................................................. socket_id ~ long / HSocketX
  
  Socket number.

Example

```cpp
/* Process 1 */
dev_set_colored (12)
open_socket_accept (3000, AcceptingSocket)
socket_accept_connect (AcceptingSocket, 'true', Socket)
receive_image (Image, Socket)
edges_sub_pix (Image, Edges, 'canny', 1.5, 20, 40)
send_xld (Edges, Socket)
receive_xld (Polygons, Socket)
split_contours_xld (Polygons, Contours, 'polygon', 1, 5)
gen_parallels_xld (Polygons, Parallels, 10, 30, 0.15, 'true')
send_xld (Parallels, Socket)
receive_xld (ModParallels, Socket)
receive_xld (ExtParallels, Socket)
stop ()
close_socket (Socket)
close_socket (AcceptingSocket)
```
/* Process 2 */
dev_set_colored (12)
open_socket_connect (‘localhost’, 3000, Socket)
read_image (Image, ‘mreut’)
send_image (Image, Socket)
receive_xld (Edges, Socket)
gen_polygons_xld (Edges, Polygons, ‘ramer’, 2)
send_xld (Polygons, Socket)
split_contours_xld (Polygons, Contours, ‘polygon’, 1, 5)
receive_xld (Edges, Socket)
mod_parallels_xld (Parallels, Image, ModParallels, ExtParallels,
0.4, 160, 220, 10)
send_xld (ModParallels, Socket)
send_xld (ExtParallels, Socket)
stop ()
close_socket (Socket)

Parallelization Information
SendXld is reentrant and processed without parallelization.

Possible Predecessor Functions
OpenSocketConnect, SocketAcceptConnect

See Also
ReceiveXld, SendImage, ReceiveImage, SendRegion, ReceiveRegion, SendTuple,
ReceiveTuple, GetNextSocketDataType

Module
System

[out] long Socket HSocketX.SocketAcceptConnect ([in] String Wait )
void HOperatorSetX.SocketAcceptConnect ([in] long AcceptingSocket,
[in] String Wait, [out] long Socket )

Accept a connection request on a listening socket.

SocketAcceptConnect accepts an incoming connection request, generated by OpenSocketConnect in another HALCON process, on the listening socket AcceptingSocket. The listening socket must have been created earlier with OpenSocketAccept. If Wait=’true’, SocketAcceptConnect waits until a connection request from another HALCON process arrives. If Wait=’false’, SocketAcceptConnect returns with the error FAIL, if currently there are no connection requests from other HALCON processes. The result of SocketAcceptConnect is another socket Socket, which is used for a two-way communication with another HALCON process. After this connection has been established, data can be exchanged between the two processes by calling the appropriate send or receive operators. For a detailed example, see OpenSocketAccept.

Parameter

- **AcceptingSocket** (input control) .......................... socket_id ~ HSocketX / long
  Socket number of the accepting socket.
- **Wait** (input control) .................................................. string ~ String
  Should the operator wait until a connection request arrives?
  **Value List**: Wait ∈ {’true’, ’false’}
- **Socket** (output control) ............................................. socket_id ~ long
  Socket number.

Parallelization Information
SocketAcceptConnect is reentrant and processed without parallelization.

Possible Predecessor Functions
OpenSocketAccept

HALCON 6.0
Possible Successor Functions
SendImage, ReceiveImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple

See Also
OpenSocketConnect, CloseSocket

Module
System
Chapter 12

Tools

12.1 Affine-Transformations

Apply a homogeneous 2D transformation matrix to points.

AffineTransPoint2D calculates the result of applying the affine transformation, given by the homogeneous 2D transformation matrix HomMat2D, to the input points (Px,Py). The resulting points are returned in (Qx,Qy).

If the coordinates (Px,Py) are image coordinates, it should be noted that the components of the homogeneous transformation matrix are to be interpreted as follows: the row coordinate of the image corresponds to the x coordinate of the matrix, while the column coordinate of the image corresponds to the y coordinate of the matrix. This is necessary to obtain a right-handed coordinate system, which is assumed for the homogeneous transformation matrices, also for the image. In particular, by doing so rotations are performed in the correct direction. Note that the (x,y) order of the matrices quite naturally corresponds to the usual (row,column) order for coordinates in the image.

Parameter

- **HomMat2D** (input control) .......... affine2d ↔ HHomMat2dX / VARIANT ( real )
  - Input transformation matrix.
  - Parameter Number : 6

- **Px** (input control) ................. point.x(-array) ↔ VARIANT ( real, integer )
  - Input point (x coordinate).
  - Default Value : 64
  - Value Suggestions : Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  - Typical Range of Values : 0 ≤ Px ≤ 0
  - Minimal Value Step : 1
  - Recommended Value Step : 10

- **Py** (input control) .................. point.y(-array) ↔ VARIANT ( real, integer )
  - Input point (y coordinate).
  - Default Value : 64
  - Value Suggestions : Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  - Typical Range of Values : 0 ≤ Py ≤ 0
  - Minimal Value Step : 1
  - Recommended Value Step : 10

- **Qx** (output control) ............... point.x(-array) ↔ VARIANT ( real )
  - Output point (x coordinate).

- **Qy** (output control) ............... point.y(-array) ↔ VARIANT ( real )
  - Output point (y coordinate).
AffineTransPoint2d always returns TRUE.

AffineTransPoint2d is reentrant and processed without parallelization.

Possible Predecessor Functions
HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Possible Successor Functions
HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Apply a homogeneous 3D transformation matrix to points.

AffineTransPoint3d calculates the result of applying the affine transformation, given by the homogeneous 3D transformation matrix homMat3D, to the input points (Px, Py, Pz). The resulting points are returned in (Qx, Qy, Qz).

AffineTransPoint3d is used to transform a 3D point (Px, Py, Pz) into a new coordinate system. The necessary translation and rotation is given by the 4x3 transformation matrix HomMat3D. The values are given as a tuple in the following order: The first three values of each row describe the three rows of the rotation matrix and the last values of each row describe the translation T. During the transformation the 3D point is first rotated and subsequently translated:

\[
\begin{pmatrix}
  x_2 \\
  y_2 \\
  z_2
\end{pmatrix} =
\begin{pmatrix}
  R & t \\
  0 & 1
\end{pmatrix}
\begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix}
\]

Parameter

- **HomMat3D** (input control) .................................. affine3d \( \rightarrow \) HHomMat3dX/VARIANT( real )
  Input transformation matrix.
  Parameter Number : 12
- **Px** (input control) ........................................... point3d.x(-array) \( \rightarrow \) VARIANT( real, integer )
  Input point (x coordinate).
  Default Value : 64
  Value Suggestions : Px \( \in \) {0, 16, 32, 64, 128, 256, 512, 1024}
  Typical Range of Values : 0 \( \leq \) Px \( \leq \) 0
  Minimal Value Step : 1
  Recommended Value Step : 10
- **Py** (input control) ........................................... point3d.y(-array) \( \rightarrow \) VARIANT( real, integer )
  Input point (y coordinate).
  Default Value : 64
  Value Suggestions : Py \( \in \) {0, 16, 32, 64, 128, 256, 512, 1024}
  Typical Range of Values : 0 \( \leq \) Py \( \leq \) 0
  Minimal Value Step : 1
  Recommended Value Step : 10
12.1. AFFINE-TRANSFORMATIONS

Point3D.z(-array) \(\rightarrow\) VARIANT (real, integer)

**Input point (z coordinate).**

**Default Value:** 64

**Value Suggestions:** \(P_z \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)

**Typical Range of Values:** \(0 \leq P_z \leq 0\)

**Minimal Value Step:** 1

**Recommended Value Step:** 10

Point3D.x(-array) \(\rightarrow\) VARIANT (real)

**Output point (x coordinate).**

Point3D.y(-array) \(\rightarrow\) VARIANT (real)

**Output point (y coordinate).**

Point3D.z(-array) \(\rightarrow\) VARIANT (real)

**Output point (z coordinate).**

---

**Result**

AffineTransPoint3d always returns TRUE.

---

**Parallelization Information**

AffineTransPoint3D is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

HomMat3dTranslate, HomMat3dScale, HomMat3dRotate, PoseToHomMat3d

---

**Possible Successor Functions**

HomMat3dTranslate, HomMat3dScale, HomMat3dRotate, Project3DPoint

---

**Module**

Basic operators

---

[out] VARIANT HomMat2D HImageX.DvfToHomMat2d ( )

void HHomMat2dX.DvfToHomMat2d ([in] HImageX VectorField )

void HOperatorSetX.DvfToHomMat2d ([in] IObjectX VectorField, [out] VARIANT HomMat2D )

---

Approximate an affine map from a displacement vector field.

DvfToHomMat2d approximates an affine map from the displacement vector field VectorField. The affine map is returned in HomMat2D.

If the displacement vector field has been computed from the original image \(I_{\text{orig}}\) and the second image \(I_{\text{res}}\), the internally stored transformation matrix (see AffineTransImage) contains a map that describes how to transform the first image \(I_{\text{orig}}\) to the second image \(I_{\text{res}}\).

---

**Parameter**

**Image**

DvfToHomMat2d is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

OpticalFlowMatch

---

**Possible Successor Functions**

AffineTransImage

---

**Alternatives**

VectorToHomMat2d

---

**Module**

Basic operators
Compoze two homogeneous 2D transformation matrices.

HomMat3dCompose composes a new 3D transformation matrix from the two input matrices. The transformation given by HomMat2DFirst is applied first, followed by the transformation given by HomMat2DSecond, i.e., with HomMat2DFirst = H1, HomMat2DSecond = H2, and HomMat2DCompose = Hc is:

\[ H_c = H_2 \cdot H_1 \]

Parameter

HomMat2DSecond (input control) ........................................... affine2d \( \rightarrow \) VARIANT( real )

Second input transformation matrix.

Parameter Number : 6

HomMat2DFirst (input control) ........................................... affine2d \( \rightarrow \) HHomMat2dX / VARIANT( real )

First input transformation matrix.

Parameter Number : 6

HomMat2DCompose (output control) ...................................... affine2d \( \rightarrow \) VARIANT( real )

Output transformation matrix.

Parameter Number : 6

Result

HomMat2dCompose always returns TRUE.

Parallelization Information

HomMat2dCompose is reentrant and processed without parallelization.

Possible Predecessor Functions

HomMat2dIdentity, HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Possible Successor Functions

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Module

Basic operators

Generate the homogeneous transformation matrix of the identical 2D transformation.

HomMat2dIdentity generates the homogeneous 2 \( \times \) 3 transformation matrix HomMat2dIdentity describing the identical 2D transformation.

Parameter

HomMat2dIdentity (output control) ................................ affine2d \( \rightarrow \) HHomMat2dX / VARIANT( real )

Transformation matrix.

Parameter Number : 6

Result

HomMat2dIdentity always returns TRUE.

Parallelization Information

HomMat2dIdentity is reentrant and processed without parallelization.

Possible Successor Functions

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Module

Basic operators
12.1. AFFINE-TRANSFORMATIONS

Invert a homogeneous 2D transformation matrix.

HomMat2dInvert inverts the homogeneous 2D transformation matrix given by HomMat2D. The resulting matrix is returned in HomMat2dInvert.

Parameter

HomMat2D (input control) .......... affine2d ~ HHomMat2dX/VARIANT( real )
Input transformation matrix.
Paramter Number : 6

HomMat2dInvert (output control) .......... affine2d ~ VARIANT( real )
Output transformation matrix.
Paramter Number : 6

Result

HomMat2dInvert returns TRUE if the input matrix is invertible.

Parallelization Information

HomMat2dInvert is reentrant and processed without parallelization.

Possible Predecessor Functions

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Possible Successor Functions

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Module

Add a rotation to a homogeneous 2D transformation matrix.

HomMat2dRotate adds a rotation by the angle Phi to a homogeneous 2D transformation matrix HomMat2D. The point (Px,Py) is the fixed point of the transformation. The resulting matrix is returned in HomMat2dRotate.

Parameter

HomMat2D (input control) .......... affine2d ~ HHomMat2dX/VARIANT( real )
Input transformation matrix.
Paramter Number : 6

Phi (input control) .......... angle.rad ~ VARIANT( real, integer )
Rotation angle.
Default Value : 0.78
Value Suggestions : Phi ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
Typical Range of Values : 0 ≤ Phi ≤ 0
Minimal Value Step : 0.01
Recommended Value Step : 0.1

Px (input control) .......... point.x ~ VARIANT( real, integer )
Fixed point of the transformation (x coordinate).
Default Value : 0
Value Suggestions : Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typical Range of Values : 0 ≤ Px ≤ 0
Minimal Value Step : 1
Recommended Value Step : 10

HALCON 6.0
HomMat2dScale adds a scaling by the scale factors \( S_x \) and \( S_y \) to a homogeneous 2D transformation matrix \( \text{HomMat2D} \). The point \((P_x, P_y)\) is the fixed point of the transformation. The resulting matrix is returned in \( \text{HomMat2DScale} \).

**Parameter**

- **\( \text{HomMat2D} \) (input control)**: affine2d \( \sim \) \( \text{HHomMat2dX} / \text{VARIANT}( \text{real} ) \)
  
  Input transformation matrix.

- **\( S_x \) (input control)**: number \( \sim \) \( \text{VARIANT}( \text{real, integer} ) \)
  
  Scale factor along the x-axis.

  **Default Value:** 2
  
  **Value Suggestions:** \( S_x \in \{0.125, 0.25, 0.5, 1, 2, 4, 8, 16\} \)

  **Typical Range of Values:** \( 0 \leq S_x \leq 0 \)

  **Minimal Value Step:** 0.001

  **Recommended Value Step:** 0.125

  **Restriction:** \( (S_x \neq 0) \)

- **\( S_y \) (input control)**: number \( \sim \) \( \text{VARIANT}( \text{real, integer} ) \)
  
  Scale factor along the y-axis.

  **Default Value:** 2

  **Value Suggestions:** \( S_y \in \{0.125, 0.25, 0.5, 1, 2, 4, 8, 16\} \)

  **Typical Range of Values:** \( 0 \leq S_y \leq 0 \)

  **Minimal Value Step:** 0.001

  **Recommended Value Step:** 0.125

  ** Restriction:** \( (S_y \neq 0) \)
12.1. AFFINE-TRANSFORMATIONS

**Px** (input control) ................................. point.x ～ VARIANT (real, integer)
Fixed point of the transformation (x coordinate).
  Default Value : 0
  Value Suggestions : Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typical Range of Values : 0 ≤ Px ≤ 0
  Minimal Value Step : 1
  Recommended Value Step : 10

**Py** (input control) ................................. point.y ～ VARIANT (real, integer)
Fixed point of the transformation (y coordinate).
  Default Value : 0
  Value Suggestions : Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typical Range of Values : 0 ≤ Py ≤ 0
  Minimal Value Step : 1
  Recommended Value Step : 10

HomMat2DScale (output control) .............................. affine2d ～ VARIANT (real)
Output transformation matrix.
  Parameter Number : 6

---

**Result**

HomMat2dScale returns TRUE if both scale factors are not 0.

---

**Parallelization Information**

HomMat2dScale is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

HomMat2dIdentity, HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

---

**Possible Successor Functions**

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

---

**Module**

Basic operators

```
[out] VARIANT HomMat2DSlant HHomMat2dX.HomMat2dSlant

void HOperatorSetX.HomMat2dSlant ([in] VARIANT HomMat2D,
[out] VARIANT HomMat2dSlant )
```

Add a slant to a homogeneous 2D transformation matrix.

HomMat2dSlant adds a slant by the angle Theta to a homogeneous 2D transformation matrix HomMat2D.
A slant is an affine transformation in which one coordinate axis remains fixed, while the other coordinate axis is rotated counterclockwise by an angle Theta. The parameter Axis determines which coordinate axis is slanted.
For Axis = 'x', the x-axis is slanted, while for Axis = 'y' the y-axis is slanted. The point (Px,Py) is the fixed point of the transformation. The resulting matrix is returned in HomMat2DSlant.

---

**Parameter**

**HomMat2D** (input control) ................................. affine2d ～ HHomMat2dX / VARIANT (real)
Input transformation matrix.
  Parameter Number : 6

**Theta** (input control) ................................. angle.rad ～ VARIANT (real, integer)
Slant angle.
  Default Value : 0.78
  Value Suggestions : Theta ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
  Typical Range of Values : 0 ≤ Theta ≤ 0
  Minimal Value Step : 0.01
  Recommended Value Step : 0.1
**Axis** (input control)  string → String
Coordinate axis that is slanted.
- Default Value: 'x'
- Value List: Axis ∈ {'x', 'y'}

**Px** (input control)  point.x → VARIANT (real, integer)
Fixed point of the transformation (x coordinate).
- Default Value: 0
- Value Suggestions: Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- Typical Range of Values: 0 ≤ Px ≤ 0
- Minimal Value Step: 1
- Recommended Value Step: 10

**Py** (input control)  point.y → VARIANT (real, integer)
Fixed point of the transformation (y coordinate).
- Default Value: 0
- Value Suggestions: Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
- Typical Range of Values: 0 ≤ Py ≤ 0
- Minimal Value Step: 1
- Recommended Value Step: 10

**HomMat2DSlant** (output control)  affine2d → VARIANT (real)
Output transformation matrix.
- Parameter Number: 6

---

**Result**

**Parallelization Information**

**HomMat2DSlant** is reentrant and processed without parallelization.

**Possible Predecessor Functions**

HomMat2dIdentity, HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2DSlant

**Possible Successor Functions**

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2DSlant

---

**Module**

Basic operators

```c
[out] double Sx HHomMat2dX.HomMat2dToAffinePar ([out] double Sy,
[out] double Phi, [out] double Theta, [out] double Tx, [out] double Ty)
```

```c
void HOperatorSetX.HomMat2dToAffinePar ([in] VARIANT HomMat2D,
[out] double Sx, [out] double Sy, [out] double Phi, [out] double Theta,
[out] double Tx, [out] double Ty)
```

**Compute the affine transformation parameters from a homogeneous 2D transformation matrix.**

HomMat2dToAffinePar computes the affine transformation parameters corresponding to the homogeneous 2D transformation matrix HomMat2D. The parameters Sx and Sy determine how the transformation scales the original x- and y-axes, respectively. The two scaling factors are always positive. The angle Theta determines how the transformed coordinate axes deviate from orthogonality. For Theta = 0, the two coordinate axes are orthogonal. If Theta > π/2 the transformation contains a reflection. The angle Phi determines the rotation of the transformed x-axis with respect to the original x-axis. The parameters Tx and Ty determine the translation of the two coordinate systems. The Matrix HomMat2D can be constructed from the six transformation parameters by the following operator sequence:

```c
hom\_mat2d\_identity (HomMat2DIdentity)
hom\_mat2d\_scale (HomMat2DIdentity, Sx, Sy, 0, 0, HomMat2DScale)
hom\_mat2d\_slant (HomMat2DScale, Theta, 'y', 0, 0, HomMat2DSlant)
hom\_mat2d\_rotate (HomMat2DSlant, Phi, 0, 0, HomMat2DRotate)
hom\_mat2d\_translate (HomMat2DRotate, Tx, Ty, HomMat2D)
```
12.1. AFFINE-TRANSFORMATIONS

Parameter

- **HomMat2D** (input control) .................. affine2d ~ HHomMat2dX / VARIANT( real )
  Input transformation matrix.

Parameter Number : 6

- **Sx** (output control) .......................... real ~ double
  Scaling factor along the x direction.

- **Sy** (output control) .......................... real ~ double
  Scaling factor along the y direction.

- **Phi** (output control) .......................... angle.rad ~ double
  Rotation angle.

- **Theta** (output control) ...................... angle.rad ~ double
  Slant angle.

- **Tx** (output control) .......................... point.x ~ double
  Translation along the x direction.

- **Ty** (output control) .......................... point.y ~ double
  Translation along the y direction.

Result

If the matrix HomMat2D is non-degenerate, HomMat2dToAffinePar returns TRUE. Otherwise, an exception is raised.

Parallelization Information

HomMat2dToAffinePar is reentrant and processed without parallelization.

Possible Predecessor Functions

VectorToHomMat2d, VectorToRigid, VectorToSimilarity

Possible Successor Functions

HomMat2dIdentity, HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Module

Basic operators

```
[out] VARIANT HomMat2DTranslate HHomMat2dX.HomMat2dTranslate
([in] VARIANT Tx, [in] VARIANT Ty )
void HOperatorSetX.HomMat2dTranslate ([in] VARIANT HomMat2D,
```

Add a translation to a homogeneous 2D transformation matrix.

HomMat2dTranslate adds a translation by the vector (Tx,Ty) to a homogeneous 2D transformation matrix HomMat2D. The resulting matrix is returned in HomMat2DTranslate.

Parameter

- **HomMat2D** (input control) .................. affine2d ~ HHomMat2dX / VARIANT( real )
  Input transformation matrix.

Parameter Number : 6

- **Tx** (input control) .......................... point.x ~ VARIANT( real, integer )
  Translation along the x-axis.

  Default Value : 64

  Value Suggestions : Tx ∈ {0, 16, 32, 64, 128, 256, 512, 1024}

  Typical Range of Values : 0 ≤ Tx ≤ 0

  Minimal Value Step : 1

  Recommended Value Step : 10
634  CHAPTER 12.  TOOLS

Ty (input control) .......................................................... point.y ▸ VARIANT (real, integer)
Translation along the y-axis.

Default Value : 64
Value Suggestions : Ty ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typical Range of Values : 0 ≤ Ty ≤ 0
Minimal Value Step : 1
Recommended Value Step : 10

HomMat2DTranslate (output control) .............................. affine2d ▸ VARIANT (real)
Output transformation matrix.

Parameter Number : 6

Result

HomMat2dIdentity always returns TRUE.

Parallelization Information

HomMat2dTranslate is reentrant and processed without parallelization.

Possible Predecessor Functions

HomMat2dIdentity, HomMat2dTranslate, HomMat2dScale, HomMat2dRotate,
HomMat2dSlant

Possible Successor Functions

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Module

Basic operators

HOperatorSetX.HomMat3dCompose ([in] VARIANT HomMat3DSecond )

HHomMat3dX.HomMat3dCompose ([out] VARIANT HomMat3DCompose)

void HOperatorSetX.HomMat3dCompose ([in] VARIANT HomMat3DSecond, [in] VARIANT HomMat3DFirst, [out] VARIANT HomMat3DCompose )

Compose two homogeneous 3D transformation matrices.

HomMat3dCompose composes a new 3D transformation matrix from the two input matrices. The transformation
given by HomMat3DFirst is applied first, followed by the transformation given by HomMat3DSecond, i.e.,
with HomMat3DFirst = H1, HomMat3DSecond = H2, and HomMat3DCompose = Hc is:

Hc = H2 · H1

A homogeneous 4x3 transformation matrix describes the transformation of a 3D point from the source coordinate
system into the destination coordinate system. In this sight the following holds are usable:

\[
\begin{pmatrix}
  x_3 \\
  y_3 \\
  z_3
\end{pmatrix}
= H_c \cdot
\begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix}
= H_2 \cdot H_1 \cdot
\begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix}
\]

with:

\[
\begin{pmatrix}
  x_2 \\
  y_2 \\
  z_2
\end{pmatrix}
= H_1 \cdot
\begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix}
= R \cdot
\begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix} + \mathcal{T}
\]

therefore:
\[
\begin{pmatrix}
  x_3 \\
  y_3 \\
  z_3
\end{pmatrix}
= \mathcal{R}_2 \cdot \begin{pmatrix}
  x_2 \\
  y_2 \\
  z_2
\end{pmatrix} + \mathcal{T}_2 = \mathcal{R}_2 \cdot \begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix} + \mathcal{T}_1 + \mathcal{T}_2
\]

\[
= \mathcal{R}_2 \cdot \mathcal{R}_1 \cdot \begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix} + \mathcal{R}_2 \cdot \mathcal{T}_1 + \mathcal{T}_2 = \mathcal{R}_e \cdot \begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix} + \mathcal{T}_e
\]

with:

Rotation: \( \mathcal{R}_e = \mathcal{R}_2 \cdot \mathcal{R}_1 \)
Translation: \( \mathcal{T}_e = \mathcal{R}_2 \cdot \mathcal{T}_1 + \mathcal{T}_2 \)

---

**HomMat3DSecond** (input control) .......................... affine3d \( \sim \) \text{VARIANT (real)}
Second input transformation matrix.
**Parameter Number**: 12

**HomMat3DFirst** (input control) .......................... affine3d \( \sim \) \text{HHomMat3dX / VARIANT (real)}
First input transformation matrix.
**Parameter Number**: 12

**HomMat3DCompose** (output control) .......................... affine3d \( \sim \) \text{VARIANT (real)}
Output transformation matrix.
**Parameter Number**: 12

---

/* read camera pose */
read_pose('campose.dat',CamPose)
/* transform pose to transformation matrix */
pose_to_hom_mat3d(CamPose,HomTransMat1)
/* multiply transformation matrices */
hom_mat3d_compose(HomTransMat2,HomTransMat1,HomTransMatMult).

---

HomMat3dCompose always returns TRUE.

---

**Parallelization Information**
HomMat3dCompose is \textit{reentrant} and processed \textit{without} parallelization.

---

**Possible Predecessor Functions**
HomMat3dIdentity, HomMat3dTranslate, HomMat3dScale, HomMat3dRotate, PoseToHomMat3d

---

**Possible Successor Functions**
HomMat3dTranslate, HomMat3dScale, HomMat3dRotate

---

**See Also**
AffineTransPoint3D, HomMat3dIdentity, HomMat3dTranslate, HomMat3dRotate, PoseToHomMat3d, HomMat3dToPose

---

**Module**
Basic operators

---

```c
void HHomMat3dX.HomMat3dIdentity ( )
void HOperatorSetX.HomMat3dIdentity ([out] VARIANT HomMat3DIdentity )
```

Generate the homogeneous transformation matrix of the identical 3D transformation.

HomMat3dIdentity generates the homogeneous \(3 \times 4\) transformation matrix HomMat3DIdentity describing the identical 3D transformation.
Parameter

- **HomMat3DIdentity** (output control) ................. affine3d \(\sim HHomMat3dX / \text{VARIANT( real )}

Transformation matrix.

**Parameter Number:** 12

Result

HomMat3dIdentity always returns TRUE.

Parallelization Information

HomMat3dIdentity is *reentrant* and processed *without* parallelization.

Possible Successor Functions

HomMat3dTranslate, HomMat3dScale, HomMat3dRotate

Alternatives

PoseToHomMat3d

Module

Basic operators

---

```haskell
[out] VARIANT HomMat3DInvert HHomMat3dX.HomMat3dInvert ()
void HOperatorSetX.HomMat3dInvert ([in] VARIANT HomMat3D,
[out] VARIANT HomMat3DInvert )
```

Invert a homogeneous 3D transformation matrix.

HomMat3dInvert inverts the homogeneous 3D transformation matrix given by HomMat3D. The resulting matrix is returned in HomMat3DInvert.

A homogeneous 4x3 transformation matrix describes the transformation of a 3D point from the source coordinate system into the destination coordinate system. In this sight the inverted transformation matrix HomMat3DInvert describes the transformation from the original destination coordinate system to the original source coordinate system. The transformation of a 3D point (given in the source coordinate system) into a destination coordinate system is given by a rotation \(R\) and a translation \(T\) (see AffineTransPoint3d). Thus, the following holds:

\[
\begin{pmatrix}
  x_2 \\
  y_2 \\
  z_2
\end{pmatrix}
= R_{org} \cdot \begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix} + T_{org}
\]

\[
\begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix}
= R_{org}^{-1} \cdot \begin{pmatrix}
  x_2 \\
  y_2 \\
  z_2
\end{pmatrix} - T_{org}
\]

with \(R_{inv} = R_{org}^{-1}\) and \(T_{inv} = -T_{org}\):

\[
\begin{pmatrix}
  x_1 \\
  y_1 \\
  z_1
\end{pmatrix}
= R_{inv} \cdot \begin{pmatrix}
  x_2 \\
  y_2 \\
  z_2
\end{pmatrix} + T_{inv}
\]
Example

/* read camera pose */
read_pose('campose.dat',CamPose)
/* transform pose to transformation matrix */
pose_to_hom_mat3d(CamPose,HomTransMat)
/* invert transformation matrix */
hom_mat3d_invert(HomTransMat,HomTransMatInv).

Result
HomMat3dInvert returns TRUE if the input matrix is invertible.

Parallelization Information
HomMat3dInvert is reentrant and processed without parallelization.

Possible Predecessor Functions
HomMat3dTranslate, HomMat3dScale, HomMat3dRotate, PoseToHomMat3d

Possible Successor Functions
HomMat3dTranslate, HomMat3dScale, HomMat3dRotate, HomMat3dToPose

See Also
AffineTransPoint3D, HomMat3dIdentity, HomMat3dRotate, HomMat3dTranslate, PoseToHomMat3d, HomMat3dCompose

Module
Basic operators

[out] VARIANT HomMat3DRotate HHomMat3dX.HomMat3dRotate
[in] VARIANT Pz )

void HOperatorSetX.HomMat3dRotate ([in] VARIANT HomMat3D,
[in] VARIANT Pz, [out] VARIANT HomMat3dRotate )

Add a rotation to a homogeneous 3D transformation matrix.

HomMat3dRotate adds a rotation around the Axis-axis by the angle Phi to a homogeneous 3D transformation matrix HomMat3D. The point (Px,Py,Pz) is the fixed point of the transformation. The resulting matrix is returned in HomMat3dRotate.

The homogeneous 4x3 transformation matrix HomMat3D describes the transformation of a 3D point from the source coordinate system into the destination coordinate system (Without Attention to the fixed point).

HomMat3dRotate is used to generate a new 4x3 transformation matrix HomMat3dRotate by rotating the Axis-axis of HomMat3D by the angle Phi (in mathematical positive direction).

If you want to rotate the source coordinate system, you have to invert the transformation matrix using HomMat3dInvert, rotate it using HomMat3dRotate, and invert it back again.

The transformation of a 3D point (given in the source coordinate system) into a destination coordinate system is given by a rotation \( \mathcal{R} \) and a translation \( \mathcal{T} \) (see AffineTransPoint3d). Thus, the following holds:

\[
\begin{pmatrix}
 x_2 \\
 y_2 \\
 z_2 \\
\end{pmatrix} = \mathcal{R}_{rot} \cdot \begin{pmatrix}
 x_1 \\
 y_1 \\
 z_1 \\
\end{pmatrix} + \mathcal{T}_{rot}
\]

\[
= (\mathcal{R}_\phi \cdot \mathcal{R}_{org}) \cdot \begin{pmatrix}
 x_2 \\
 y_2 \\
 z_2 \\
\end{pmatrix} + \mathcal{T}_{org}
\]

with
\[
\begin{align*}
R_x^\phi &= \begin{pmatrix}
1 & 0 & 0 \\
0 & \cos \phi & -\sin \phi \\
0 & \sin \phi & \cos \phi
\end{pmatrix},
R_y^\phi &= \begin{pmatrix}
\cos \phi & 0 & \sin \phi \\
0 & 1 & 0 \\
-\sin \phi & 0 & \cos \phi
\end{pmatrix},
R_z^\phi &= \begin{pmatrix}
\cos \phi & -\sin \phi & 0 \\
\sin \phi & \cos \phi & 0 \\
0 & 0 & 1
\end{pmatrix}
\end{align*}
\]

Parameter

\begin{itemize}
\item \textbf{HomMat3D} (input control) \hspace{1cm} affine3d \sim HHomMat3dX / VARIANT (real)
  
  Input transformation matrix.
  
  Parameter Number: 12

\item \textbf{Phi} (input control) \hspace{1cm} angle.rad \sim VARIANT (real, integer)
  
  Rotation angle.
  
  Default Value: 0.78
  
  Value Suggestions: \(\Phi \in \{0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14\}\)
  
  Typical Range of Values: \(0 \leq \Phi \leq 0\)
  
  Minimal Value Step: 0.01
  
  Recommended Value Step: 0.1

\item \textbf{Axis} (input control) \hspace{1cm} string \sim String
  
  Axis, to be rotated around.
  
  Default Value: ‘x’
  
  Value Suggestions: \(\text{Axis} \in \{‘x’, ‘y’, ‘z’\}\)

\item \textbf{Px} (input control) \hspace{1cm} point3d.x \sim VARIANT (real, integer)
  
  Fixed point of the transformation (x coordinate).
  
  Default Value: 0
  
  Value Suggestions: \(\text{Px} \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)
  
  Typical Range of Values: \(0 \leq \text{Px} \leq 0\)
  
  Minimal Value Step: 1
  
  Recommended Value Step: 10

\item \textbf{Py} (input control) \hspace{1cm} point3d.y \sim VARIANT (real, integer)
  
  Fixed point of the transformation (y coordinate).
  
  Default Value: 0
  
  Value Suggestions: \(\text{Py} \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)
  
  Typical Range of Values: \(0 \leq \text{Py} \leq 0\)
  
  Minimal Value Step: 1
  
  Recommended Value Step: 10

\item \textbf{Pz} (input control) \hspace{1cm} point3d.z \sim VARIANT (real, integer)
  
  Fixed point of the transformation (z coordinate).
  
  Default Value: 0
  
  Value Suggestions: \(\text{Pz} \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)
  
  Typical Range of Values: \(0 \leq \text{Pz} \leq 0\)
  
  Minimal Value Step: 1
  
  Recommended Value Step: 10

\item \textbf{HomMat3DRotate} (output control) \hspace{1cm} affine3d \sim VARIANT (real)
  
  Output transformation matrix.
  
  Parameter Number: 12
\end{itemize}

Example

```c
/* read camera pose */
read_pose(’campose.dat’,CamPose)
/* transform pose to transformation matrix */
pose_to_hom_mat3d(CamPose,HomTransMat)
/* rotate destination coordinate system around x-axis by 10 degree */
hom_mat3d_rotate(HomTransMat,10,’x’,0,0,0,HomTransMatRot)
/* rotate source coordinate system around y-axis by 270 degree */
hom_mat3d_rotate(HomTransMat,HomTransMatRot)
hom_mat3d_invert(HomTransMat,HomTransMatInv)
hom_mat3d_rotate(HomTransMatInv,270,’y’,0,0,0,HomTransMatRot2)
hom_mat3d_invert(HomTransMatRot2,HomTransMat2).
```

Result

HomMat3DRotate always returns TRUE.

12.1. AFFINE-TRANSFORMATIONS

**Parallelization Information**
HomMat3dRotate is reentrant and processed without parallelization.

**Possible Predecessor Functions**
HomMat3dIdentity, HomMat3dTranslate, HomMat3dScale, HomMat3dRotate

**Possible Successor Functions**
HomMat3dTranslate, HomMat3dScale, HomMat3dRotate

**See Also**
HomMat3dInvert, HomMat3dIdentity, HomMat3dRotate, PoseToHomMat3d, HomMat3dToPose, HomMat3dCompose

**Basic operators**

```c
```

Add a scaling to a homogeneous 3D transformation matrix.

HomMat3dScale adds a scaling by the scale factors Sx and Sy to a homogeneous 3D transformation matrix HomMat3D. The point (Px,Py,Pz) is the fixed point of the transformation. The resulting matrix is returned in HomMat3DScale.

**Parameter**

- **HomMat3D** (input control) .................. affine3d ~/ HHomMat3dX / VARIANT( real )
  Input transformation matrix.

- **Parameter Number :** 12

- **Sx** (input control) .......................... number ~/ VARIANT( real, integer )
  Scale factor along the x-axis.

  - **Default Value :** 2
  - **Value Suggestions :** Sx ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
  - **Typical Range of Values :** 0 ≤ Sx ≤ 0
  - **Minimal Value Step :** 0.001
  - **Recommended Value Step :** 0.125
  - **Restriction :** (Sx ≠ 0)

- **Sy** (input control) .......................... number ~/ VARIANT( real, integer )
  Scale factor along the y-axis.

  - **Default Value :** 2
  - **Value Suggestions :** Sy ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
  - **Typical Range of Values :** 0 ≤ Sy ≤ 0
  - **Minimal Value Step :** 0.001
  - **Recommended Value Step :** 0.125
  - **Restriction :** (Sy ≠ 0)

- **Sz** (input control) .......................... number ~/ VARIANT( real, integer )
  Scale factor along the z-axis.

  - **Default Value :** 2
  - **Value Suggestions :** Sz ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
  - **Typical Range of Values :** 0 ≤ Sz ≤ 0
  - **Minimal Value Step :** 0.001
  - **Recommended Value Step :** 0.125
  - **Restriction :** (Sz ≠ 0)
px (input control) ........................................ point3d.x ~ VARIANT (real, integer)
Fixed point of the transformation (x coordinate).
Default Value : 0
Value Suggestions : px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typical Range of Values : 0 ≤ px ≤ 0
Minimal Value Step : 1
Recommended Value Step : 10

py (input control) ........................................ point3d.y ~ VARIANT (real, integer)
Fixed point of the transformation (y coordinate).
Default Value : 0
Value Suggestions : py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typical Range of Values : 0 ≤ py ≤ 0
Minimal Value Step : 1
Recommended Value Step : 10

pz (input control) ........................................ point3d.z ~ VARIANT (real, integer)
Fixed point of the transformation (z coordinate).
Default Value : 0
Value Suggestions : pz ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typical Range of Values : 0 ≤ pz ≤ 0
Minimal Value Step : 1
Recommended Value Step : 10

HomMat3dScale (output control) ........................................ affine3d ~ VARIANT (real)
Output transformation matrix.
Parameter Number : 12

HomMat3dScale returns TRUE if all three scale factors are not 0.

Parallelization Information
HomMat3dScale is reentrant and processed without parallelization.

Possible Predecessor Functions
HomMat3dIdentity, HomMat3dTranslate, HomMat3dScale, HomMat3dRotate

Possible Successor Functions
HomMat3dTranslate, HomMat3dScale, HomMat3dRotate

Module
Basic operators

[out] VARIANT HomMat3dTranslate HHomMat3dX.HomMat3dTranslate

void HOperatorSetX.HomMat3dTranslate ([in] VARIANT HomMat3D,
[in] VARIANT Tx, [in] VARIANT Ty, [in] VARIANT Tz,
[out] VARIANT HomMat3dTranslate )

Add a translation to a homogeneous 3D transformation matrix.

HomMat3dTranslate adds a translation by the vector (Tx,Ty,Tz) to a homogeneous 3D transformation matrix HomMat3D. The resulting matrix is returned in HomMat3dTranslate.

The homogeneous 4x3 transformation matrix HomMat3D describes the transformation of a 3D point from the source coordinate system into the destination coordinate system. In this sight HomMat3dTranslate is used to generate a new 4x3 transformation matrix HomMat3dTranslate by translating the origin of HomMat3D along the negative Axis by the amount (in meter) of the according to the vector (Tx,Ty,Tz).

If you want to translate the source coordinate system, you have to invert the transformation matrix using HomMat3dInvert, translate it using HomMat3dTranslate, and invert it back again.

The transformation of a 3D point (given in the source coordinate system) into a destination coordinate system ist given by a rotation R and a translation T (see AffineTransPoint3d). Thus, the following holds:
\[
\begin{pmatrix}
   x_2 \\
   y_2 \\
   z_2
\end{pmatrix}
= \mathcal{R}_{trans} \cdot 
\begin{pmatrix}
   x_1 \\
   y_1 \\
   z_1
\end{pmatrix} + \mathcal{T}_{trans}
\]
\[
= \mathcal{R}_{org} \cdot 
\begin{pmatrix}
   x_2 \\
   y_2 \\
   z_2
\end{pmatrix} + (\mathcal{T}_{org} + \mathcal{T}_\theta)
\]

with
\[
\mathcal{T}_\theta = \begin{pmatrix}
   t_x \\
   t_y \\
   t_z
\end{pmatrix}
\]

**Parameter**

- **HomMat3D** (input control) ............... affine3d \(\sim HHomMat3dX/\text{VARIANT}(\text{real})\)
  Input transformation matrix.
  **Parameter Number : 12**
  - **Tx** (input control) ....................... point3d.x \(\sim \text{VARIANT}(\text{real, integer})\)
    Translation along the x-axis.
    **Default Value : 64**
    **Value Suggestions :** \(Tx \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)
    **Typical Range of Values :** \(0 \leq Tx \leq 0\)
    **Minimal Value Step : 1**
    **Recommended Value Step : 10**
  - **Ty** (input control) ....................... point3d.y \(\sim \text{VARIANT}(\text{real, integer})\)
    Translation along the y-axis.
    **Default Value : 64**
    **Value Suggestions :** \(Ty \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)
    **Typical Range of Values :** \(0 \leq Ty \leq 0\)
    **Minimal Value Step : 1**
    **Recommended Value Step : 10**
  - **Tz** (input control) ....................... point3d.z \(\sim \text{VARIANT}(\text{real, integer})\)
    Translation along the z-axis.
    **Default Value : 64**
    **Value Suggestions :** \(Tz \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)
    **Typical Range of Values :** \(0 \leq Tz \leq 0\)
    **Minimal Value Step : 1**
    **Recommended Value Step : 10**

- **HomMat3DTranslate** (output control) ................. affine3d \(\sim \text{VARIANT}(\text{real})\)
  Output transformation matrix.
  **Parameter Number : 12**

**Example**

```c
/* read camera pose */
read_pose('campose.dat', CamPose)
/* transform pose to transformation matrix */
pose_to_hom_mat3d(CamPose, HomTransMat)
/* translate destination coordinate along x-axis by 0.4 m */
hom_mat3d_translate(HomTransMat, -0.4, 0, 0, HomTransMatTransl)
/* translate source coordinate along y-axis by -0.03 m */
hom_mat3d_inverse(HomTransMat, HomTransMatInv)
hom_mat3d_translate(HomTransMatInv, 0, 0.03, 0, HomTransMatTransl2)
hom_mat3d_inverse(HomTransMatTransl2, HomTransMat2).
```

**Result**

HomMat3dIdentity always returns TRUE.
Parallelization Information

HomMat3dTranslate is reentrant and processed without parallelization.

Possible Predecessor Functions

HomMat3dIdentity, HomMat3dTranslate, HomMat3dScale, HomMat3dRotate

Possible Successor Functions

HomMat3dTranslate, HomMat3dScale, HomMat3dRotate

See Also

HomMat3dInvert, HomMat3dIdentity, HomMat3dRotate, HomMat3dTranslate, PoseToHomMat3d, HomMat3dToPose, HomMat3dCompose

Basic operators

---

**void HHomMat2dX.VectorAngleToRigid** ([in] VARIANT Row1,
[in] VARIANT Column1, [in] VARIANT Angle1, [in] VARIANT Row2,
[in] VARIANT Column2)

**void HOperatorSetX.VectorAngleToRigid** ([in] VARIANT Row1,
[in] VARIANT Column1, [in] VARIANT Angle1, [in] VARIANT Row2,

---

Compute a rigid affine transformation from points and angles.

VectorAngleToRigid computes a rigid affine transformation from a point correspondence and two corresponding angles. The coordinates of the original point are passed in (Row1, Column1), while the corresponding angle is passed in Angle1. The coordinates of the transformed point are passed in (Row2, Column2), while the corresponding angle is passed in Angle2. In particular, the operator VectorAngleToRigid is useful to construct a rigid affine transformation from the results of the pattern matching operators (BestMatchRot and BestMatchRotMg), which transforms a reference image to the current image or (if the parameters are passed in reverse order) from the current image to the reference image. The computed transformation is returned in HomMat2D and can be used directly with operators that transform data using affine transformations, e.g., AffineTransImage.

**Parameter**

- **Row1** (input control) .................................................. point.y ~ VARIANT (real, integer)
  Row coordinate of the original point.
- **Column1** (input control) ............................................. point.x ~ VARIANT (real, integer)
  Column coordinate of the original point.
- **Angle1** (input control) ............................................. angle.rad ~ VARIANT (real, integer)
  Angle of the original point.
- **Row2** (input control) .................................................. point.y ~ VARIANT (real, integer)
  Row coordinate of the transformed point.
- **Column2** (input control) ............................................. point.x ~ VARIANT (real, integer)
  Column coordinate of the transformed point.
- **Angle2** (input control) ............................................. angle.rad ~ VARIANT (real, integer)
  Angle of the transformed point.
- **HomMat2D** (output control) ....................................... affine2d ~ HHomMat2dX / VARIANT (real)
  Output transformation matrix.

**Parameter Number : 6**

---

**Example**

```cpp
draw_rectangle2 (WindowID, RowTempl, ColumnTempl, PhiTempl, Length1, Length2)
gen_rectangle2 (Rectangle, RowTempl, ColumnTempl, PhiTempl, Length1, Length2)
reduce_domain (ImageTempl, Rectangle, ImageReduced)
create_template_rot (ImageReduced, 4, 0, rad(360), rad(1), 'sort',
  'original', TemplateID)
```

best_match_rot_mg (Image, TemplateID, 0, rad(360), 30, 'true', 4, Row, Column, Angle, ErrMatch)
if (ErrMatch<255)
    vector_angle_to_rigid (Row, Column, Angle, RowTempl, ColumnTempl, 0, HomMat2D)
    affine_trans_image (Image, ImageAffinTrans, HomMat2D, 'constant', 'false')
endif
endwhile
clear_template (TemplateID)

__Parallelization Information__
VectorAngleToRigid is reentrant and processed without parallelization.

Possible Predecessor Functions
BestMatchRotMg, BestMatchRot

Possible Successor Functions
HomMat2dInvert, AffineTransImage, AffineTransRegion, AffineTransContourXld, AffineTransPolygonXld, AffineTransPoint2D

__See Also__
DvftToHomMat2d

__Alternatives__
VectorToRigid

Module Basic operators

```c

```

Approximate an affine transformation from point correspondences.
VectorToHomMat2d approximates an affine transformation from at least three point correspondences. The point correspondences are passed in the tuples (Rows1, Columns1) and (Rows2, Columns2), where corresponding points must be at the same index positions in the tuples. If more than three point correspondences are passed the transformation is overdetermined. In this case, the returned transformation is the transformation that minimizes the distances between the input points (Rows1, Columns1) and the transformed points (Rows2, Columns2). The calculated transformation is returned in HomMat2D and can be used directly with operators that transform data using affine transformations, e.g., AffineTransImage.

__Parameter__
- **Rows1** (input control) ........................................ point.y ~ VARIANT (real) Row coordinates of the starting points.
- **Columns1** (input control) ..................................... point.x ~ VARIANT (real) Column coordinates of the starting points.
- **Rows2** (input control) ........................................ point.y ~ VARIANT (real) Row coordinates of the end points.
- **Columns2** (input control) ..................................... point.x ~ VARIANT (real) Column coordinates of the end points.
- **HomMat2D** (output control) ................................. affine2d ~ HHomMat2dX / VARIANT (real) Output transformation matrix.

__Parameter Number :6__

__Parallelization Information__
VectorToHomMat2d is reentrant and processed without parallelization.
Approximate a rigid affine transformation from point correspondences.
VectorToRigid approximates a rigid affine transformation from at least two point correspondences. The point correspondences are passed in the tuples (Rows1, Columns1) and (Rows2, Columns2), where corresponding points must be at the same index positions in the tuples. The transformation is always overdetermined. Therefore, the returned transformation is the transformation that minimizes the distances between the original points (Rows1, Columns1) and the transformed points (Rows2, Columns2). The calculated transformation is returned in HomMat2D and can be used directly with operators that transform data using affine transformations, e.g., AffineTransImage.

Parameter

- **Rows1** (input control) .......................................................... point.y ← VARIANT (real)  
  Row coordinates of the original points.
- **Columns1** (input control) ...................................................... point.x ← VARIANT (real)  
  Column coordinates of the original points.
- **Rows2** (input control) .......................................................... point.y ← VARIANT (real)  
  Row coordinates of the transformed points.
- **Columns2** (input control) ...................................................... point.x ← VARIANT (real)  
  Column coordinates of the transformed points.
- **HomMat2D** (output control) .................................................... affine2d ← HHomMat2dX / VARIANT (real)  
  Output transformation matrix.

Parameter Number: 6

Parallelization Information
VectorToRigid is reentrant and processed without parallelization.
Approximate an similarity transformation from point correspondences.

VectorToSimilarity approximates a similarity transformation from at least two point correspondences. The point correspondences are passed in the tuples (Rows1, Columns1) and (Rows2,Columns2), where corresponding points must be at the same index positions in the tuples. If more than two point correspondences are passed the transformation is overdetermined. In this case, the returned transformation is the transformation that minimizes the distances between the original points (Rows1,Columns1) and the transformed points (Rows2,Columns2). The calculated transformation is returned in HomMat2D and can be used directly with operators that transform data using affine transformations, e.g., AffineTransImage.

Parameter

- Rows1 (input control) ....................................................... point.y ~ VARIANT (real)
  Row coordinates of the original points.
- Columns1 (input control) ................................................. point.x ~ VARIANT (real)
  Column coordinates of the original points.
- Rows2 (input control) ....................................................... point.y ~ VARIANT (real)
  Row coordinates of the transformed points.
- Columns2 (input control) ................................................. point.x ~ VARIANT (real)
  Column coordinates of the transformed points.
- HomMat2D (output control) .............................................. affine2d ~ HHomMat2dX / VARIANT (real)
  Output transformation matrix.

Parallelization Information

VectorToSimilarity is reentrant and processed without parallelization.

Possible Successor Functions

AffineTransImage, AffineTransRegion, AffineTransContourXld,
AffineTransPolygonXld, AffineTransPoint2D

See Also

DvfToHomMat2d

Alternatives

VectorToHomMat2d, VectorToRigid

Module

Basic operators

12.2 Background-Estimator

Delete all background estimation data sets.

CloseAllBgEsti deletes the background estimation data sets and releases all used memory.

Attention

Since all estimaters are closed by CloseAllBgEsti, all handles become invalid.

Result

If it is possible to close the background estimation data sets the operator CloseAllBgEsti returns the value TRUE. Otherwise an exception handling is raised.
Parallelization Information

CloseAllBgEsti is local and processed completely exclusively without parallelization.

See Also

CreateBgEsti

Alternatives

CloseBgEsti

Module

Background estimation

---

**void HOperatorSetX.CloseBgEsti ([in] long BgEstiHandle)**

Delete the background estimation data set.

CloseBgEsti deletes the background estimation data set and releases all used memory.

**Parameter**

- **BgEstiHandle** (input control) ........ bg estimation \nobreak HBgEstiX / long ID of the BgEsti data set.

**Example**

```c
/* read Init-Image: */
read_image(InitImage,'Init_Image')
/* initialize BgEsti-Dataset with fixed gains and threshold adaption: */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02,
'on',7,10,3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,'Image_1')
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* display the foreground region: */
disp_region(Region1,WindowHandle)
/* read the next image in sequence: */
read_image(Image2,'Image_2')
/* estimate the Background: */
run_bg_esti(Image2,Region2,BgEstiHandle)
/* display the foreground region: */
disp_region(Region2,WindowHandle)
/* etc. */
/* - end of background estimation - */
/* close the dataset: */
close_bg_est(BgEstiHandle).
```

**Result**

CloseBgEsti returns TRUE if all parameters are correct.

Parallelization Information

CloseBgEsti is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

RunBgEsti

See Also

CreateBgEsti

Module

Background estimation
Generate and initialize a data set for the background estimation.

HCreateBgEsti creates a new data set for the background estimation and initializes it with the appropriate parameters. The estimated background image is part of this data set. The newly created set automatically becomes the current set.

InitializeImage is used as an initial prediction for the background image. For a good prediction an image of the observed scene without moving objects should be passed in InitializeImage. That way the foreground adaptation rate can be held low. If there is no empty scene image available, a homogenous gray image can be used instead. In that case the adaptation rate for the foreground image must be raised, because initially most of the image will be detected as foreground. The initialization image must be of type byte or real. Because of processing single-channel images, data sets must be created for every channel. Size and region of InitializeImage determines size and region for all background estimations (RunBgEsti) that are performed with this data set.

Syspar1 and Syspar2 are the parameters of the Kalman system matrix. The system matrix describes the system of the gray value changes according to Kalman filter theory. The background estimator implements a different system for each pixel.

GainMode defines whether a fixed Kalman gain should be used for the estimation or whether the gain should adapt itself depending on the difference between estimation and actual value. If GainMode is set to 'fixed', then Gain1 is used as Kalman gain for pixels predicted as foreground and Gain2 as gain for pixels predicted as background. Gain1 should be smaller than Gain2, because adaptation of the foreground should be slower than adaptation of the background. Both Gain1 and Gain2 should be smaller than 1.0.

If GainMode is set to 'frame', then tables for foreground and background estimation are computed containing Kalman gains for all the 256 possible grayvalue changes. Gain1 and Gain2 then denote the number of frames necessary to adapt the difference between estimated value and actual value. So with a fixed time for adaptation (i.e. number of frames) the needed Kalman gain grows with the grayvalue difference. Gain1 should therefore be larger than Gain2. Different gains for different grayvalue differences are useful if the background estimator is used for generating an 'empty' scene assuming that there are always moving objects in the observed area. In that case the adaptation time for foreground adaptation (Gain1) must not be too big. Gain1 and Gain2 should be bigger than 1.0.

AdaptMode denotes, whether the foreground/background decision threshold applied to the grayvalue difference between estimation and actual value is fixed or whether it adapts itself depending on the grayvalue deviation of the background pixels.

If AdaptMode is set to 'off', the parameter MinDiff denotes a fixed threshold. The parameters StatNum, ConfidenceC and TimeC are meaningless in this case.

If AdaptMode is set to 'on', then MinDiff is interpreted as a base threshold. For each pixel an offset is added to this threshold depending on the statistical evaluation of the pixel value over time. StatNum holds the number of data sets (past frames) that are used for computing the grayvalue variance (FIR-Filter). ConfidenceC is used to determine the confidence interval.

The confidence interval determines the values of the background statistics if background pixels are hidden by a foreground object and thus are detected as foreground. According to the student t-distribution the confidence constant is 4.30 (3.25, 2.82, 2.26) for a confidence interval of 99.8% (99.0%, 98.0%, 95.0%). TimeC holds a time constant for the exp-function that raises the threshold in case of a foreground estimation of the pixel. That
means, the threshold is raised in regions where movement is detected in the foreground. That way larger changes in
illumination are tolerated if the background becomes visible again. The main reason for increasing this tolerance is
the impossibility for a prediction of illumination changes while the background is hidden. Therefore no adaptation
of the estimated background image is possible.

Attention

If GainMode was set to 'frame', the run-time can be extremely long for large values of Gain1 or Gain2, because
the values for the gains’ table are determined by a simple binary search.

Parameter

- **InitializeImage** (input iconic) ............................................. image ～ HimageX / IObjectX (byte, real )
  initialization image.
- **Syspar1** (input control) .................................................. real ～ double
  1. system matrix parameter.
  
  Default Value : 0.7
  
  Value Suggestion : Syspar1 ∈ {0.65, 0.7, 0.75}
  
  Typical Range of Values : 0.05 ≤ Syspar1 ≤ 0.05
  
  Recommended Value Step : 0.05

- **Syspar2** (input control) .................................................. real ～ double
  2. system matrix parameter.
  
  Default Value : 0.7
  
  Value Suggestion : Syspar2 ∈ {0.65, 0.7, 0.75}
  
  Typical Range of Values : 0.05 ≤ Syspar2 ≤ 0.05
  
  Recommended Value Step : 0.05

- **GainMode** (input control) ............................................. string ～ String
  Gain type.
  
  Default Value : ’fixed’
  
  Value List : GainMode ∈ {’fixed’, ’frame’}

- **Gain1** (input control) .................................................. real ～ double
  Kalman gain / foreground adaptation time.
  
  Default Value : 0.002
  
  Value Suggestion : Gain1 ∈ {10.0, 20.0, 50.0, 0.1, 0.05, 0.01, 0.005, 0.001}
  
  Restriction : (0.0 ≤ Gain1)

- **Gain2** (input control) .................................................. real ～ double
  Kalman gain / background adaptation time.
  
  Default Value : 0.02
  
  Value Suggestion : Gain2 ∈ {2.0, 4.0, 8.0, 0.5, 0.1, 0.05, 0.01}
  
  Restriction : (0.0 ≤ Gain2)

- **AdaptMode** (input control) ............................................. string ～ String
  Threshold adaptation.
  
  Default Value : ’on’
  
  Value List : AdaptMode ∈ {’on’, ’off’}

- **MinDiff** (input control) ............................................. real ～ double
  Foreground/background threshold.
  
  Default Value : 7.0
  
  Value Suggestion : MinDiff ∈ {3.0, 5.0, 7.0, 9.0, 11.0}
  
  Recommended Value Step : 0.2

- **StatNum** (input control) ............................................. integer ～ long
  Number of statistic data sets.
  
  Default Value : 10
  
  Value Suggestion : StatNum ∈ {5, 10, 20, 30}
  
  Recommended Value Step : 5

- **ConfidenceC** (input control) ........................................... real ～ double
  Confidence constant.
  
  Default Value : 3.25
  
  Value Suggestion : ConfidenceC ∈ {4.30, 3.25, 2.82, 2.62}
  
  Recommended Value Step : 0.01
  
  Restriction : (0.0 < ConfidenceC)
**BACKGROUND-ESTIMATOR**

- **TimeC** (input control) ........................................... real \(\sim\) double
  - Constant for decay time.
  - **Default Value**: 15.0
  - **Value Suggestions**: \(\text{TimeC} \in \{10.0, 15.0, 20.0\}\)
  - **Recommended Value Step**: 5.0
  - **Restriction**: \(0.0 < \text{TimeC}\)

- **BgEstiHandle** (output control) .................................. bg_estimation \(\sim\) long / HBgEstiX
  - ID of the BgEsti data set.

---

```c
/* read Init-Image: */
read_image(InitImage,'Init_Image')
/* initialize 1. BgEsti-Dataset with 
   fixed gains and threshold adaption: */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02,
   'on',7.0,10,3.25,15.0,BgEstiHandle1)
/* initialize 2. BgEsti-Dataset with 
   frame orientated gains and fixed threshold */
create_bg_esti(InitImage,0.7,0.7,'frame',30.0,4.0,
   'off',9.0,10,3.25,15.0,BgEstiHandle2).
```

---

**GetBgEstiParams**

- **Return the parameters of the data set.**
- **GetBgEstiParams** returns the parameters of the data set. The returned parameters are the same as in **CreateBgEsti** and **SetBgEstiParams** (see these for an explanation).

---

**Parameter**

- **BgEstiHandle** (input control) ............................... bg_estimation \(\sim\) HBgEstiX / long
  - ID of the BgEsti data set.
- **Syspar1** (output control) ................................. real \(\sim\) double
  - 1. system matrix parameter.
- **Syspar2** (output control) ................................. real \(\sim\) double
  - 2. system matrix parameter.
- **GainMode** (output control) .............................. string \(\sim\) String
  - Gain type.

---

**HALCON 6.0**
Gain1 (output control) ...................................................... real  \rightarrow  double  
Kalman gain / foreground adaptation time.

Gain2 (output control) ...................................................... real  \rightarrow  double  
Kalman gain / background adaptation time.

AdaptMode (output control) .............................................. string  \rightarrow  String  
Threshold adaptation.

MinDiff (output control) ................................................... real  \rightarrow  double  
Foreground / background threshold.

StatNum (output control) ................................................... integer  \rightarrow  long  
Number of statistic data sets.

ConfidenceC (output control) ............................................. real  \rightarrow  double  
Confidence constant.

TimeC (output control) ................................................... real  \rightarrow  double  
Constant for decay time.

--- Example ---

/* read Init-Image:*/
read_image(InitImage,'Init_Image')
/* initialize BgEsti-Dataset with fixed gains and threshold adaptation: */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02,
'on',7.0,10,3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,'Image_1')
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* display the foreground region: */
disp_region(Region1,WindowHandle)
/* read the next image in sequence: */
read_image(Image2,'Image_2')
/* estimate the Background: */
run_bg_esti(Image2,Region2,BgEstiHandle)
/* display the foreground region: */
disp_region(Region2,WindowHandle)
/* etc. */
/* change only the gain parameter in dataset: */
get_bg_esti_params(BgEstiHandle,par1,par2,par3,par4,
par5,par6,par7,par8,par9,par10)
set_bg_esti_params(BgEstiHandle,par1,par2,par3,0.004,
0.08,par6,par7,par8,par9,par10)
/* read the next image in sequence: */
read_image(Image3,'Image_3')
/* estimate the Background: */
run_bg_esti(Image3,Region3,BgEstiHandle)
/* display the foreground region: */
disp_region(Region3,WindowHandle)
/* etc. */

--- Result ---

GetBgEstiParams returns TRUE if all parameters are correct.

--- Parallelization Information ---

GetBgEstiParams is reentrant and processed without parallelization.

--- Possible Predecessor Functions ---

CreateBgEsti

--- Possible Successor Functions ---

RunBgEsti

Return the estimated background image.

GiveBgEsti returns the estimated background image of the current BgEsti data set. The background image has the same type and size as the initialization image passed in CreateBgEsti.

### Parameter

- **BackgroundImage** (output iconic) ................image HImageX / HUntypedObjectX (byte, real)
  Estimated background image of the current data set.
- **BgEstiHandle** (input control) ......................bg_estimation long / HBgEstiX
  ID of the BgEsti data set.

### Example

```c
/* read Init-Image: */
read_image(InitImage,'Init_Image')
/* initialize BgEsti-Dataset with
 fixed gains and threshold adaption: */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02,
 'on',7,10,3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,'Image_1')
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* give the background image from the aktive dataset: */
give_bg_esti(BgImage,BgEstiHandle)
/* display the background image: */
disp_image(BgImage,WindowHandle).
```

### Result

GiveBgEsti returns TRUE if all parameters are correct.

### Parallelization Information

GiveBgEsti is local and processed completely exclusively without parallelization.

### Possible Predecessor Functions

RunBgEsti

### Possible Successor Functions

RunBgEsti, CreateBgEsti, UpdateBgEsti

### See Also

RunBgEsti, CreateBgEsti, UpdateBgEsti

### Module

Background estimation
Estimate the background and return the foreground region.

RunBgEsti adapts the background image stored in the BgEsti data set using a Kalman filter on each pixel and returns a region of the foreground (detected moving objects).

For every pixel an estimation of its grayvalue is computed using the values of the current data set and its stored background image and the current image (PresentImage). By comparison to the threshold (fixed or adaptive, see CreateBgEsti) the pixels are classified as either foreground or background.

The background estimation processes only single-channel images. Therefore the background has to be adapted separately for every channel.

The background estimation should be used on half- or even quarter-sized images. For this, the input images (and the initialization image!) has to be reduced using ZoomImageFactor. The advantage is a shorter run-time on one hand and a low-band filtering on the other. The filtering eliminates high frequency noise and results in a more reliable estimation. As a result the threshold (see CreateBgEsti) can be lowered. The foreground region returned by RunBgEsti then has to be enlarged again for further processing.

Attention

The passed image (PresentImage) must have the same type and size as the background image of the current data set (initialized with CreateBgEsti).

Parameter

- **PresentImage** (input iconic) .........................image ← HImageX / IObjectX ( byte, real )
  Current image.

- **ForegroundRegion** (output iconic) ......................region ← HRegionX / HUntypedObjectX
  Region of the detected foreground.

- **BgEstiHandle** (input control) .............................. bg_estimation ← long / HBgEstiX
  ID of the BgEsti data set.

Example

```c
/* read Init-Image: */
read_image(InitImage,'Init_Image')
/* initialize BgEsti-Dataset with
   fixed gains and threshold adaption */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02,
                 'on',7,10,3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,'Image_1')
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* display the foreground region: */
disp_region(Region1,WindowHandle)
/* read the next image in sequence: */
read_image(Image2,'Image_2')
/* estimate the Background: */
run_bg_esti(Image2,Region2,BgEstiHandle)
/* display the foreground region: */
disp_region(Region2,WindowHandle)
/* etc. */
```
12.2. BACKGROUND-ESTIMATOR

RunBgEsti returns TRUE if all parameters are correct.

RunBgEsti is reentrant and processed without parallelization.

Possible Predecessor Functions

Possible Successor Functions

See Also

SetBgEstiParams, CreateBgEsti, UpdateBgEsti, GiveBgEsti

Module

Background estimation

### Result

### Parallelization Information

### Possible Predecessor Functions

### Possible Successor Functions

### See Also

### Module

Background estimation

```c
#define HBGESTIX_SETBGESTIPARAMS (in) double Syspar1,
(in) double Syspar2, (in) String GainMode, (in) double Gain1,
[in] double Gain2, (in) String AdaptMode, (in) double MinDiff,
[in] long StatNum, (in) double ConfidenceC, (in) double TimeC )

#define HOPERATORSETX_SETBGESTIPARAMS (in) long BgEstiHandle,
[in] double Syspar1, (in) double Syspar2, (in) String GainMode,
[in] double Gain1, (in) double Gain2, (in) String AdaptMode,
[in] double MinDiff, (in) long StatNum, (in) double ConfidenceC,
[in] double TimeC )
```

Change the parameters of the data set.

SetBgEstiParams is used to change the parameters of the data set. The parameters passed by SetBgEstiParams are the same as in CreateBgEsti (see there for an explanation). The image format cannot be changed! To do this, a new data set with an initialization image of the appropriate format has to be created.

To exchange the background image completely, use UpdateBgEsti. The current image then has to be passed for both the input image and the update region.

Attention

If GainMode was set to 'frame', the run-time can be extremely long for large values of Gain1 or Gain2, because the values for the gains' table are determined by a simple binary search.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BgEstiHandle</td>
<td>(input control) ID of the BgEsti data set.</td>
</tr>
<tr>
<td>Syspar1</td>
<td>(input control) 1. system matrix parameter. Default Value: 0.7 Value Suggestions: Syspar1 ∈ {0.65, 0.7, 0.75} Typical Range of Values: 0.05 ≤ Syspar1 ≤ 0.05 Recommended Value Step: 0.05</td>
</tr>
<tr>
<td>Syspar2</td>
<td>(input control) 2. system matrix parameter. Default Value: 0.7 Value Suggestions: Syspar2 ∈ {0.65, 0.7, 0.75} Typical Range of Values: 0.05 ≤ Syspar2 ≤ 0.05 Recommended Value Step: 0.05</td>
</tr>
<tr>
<td>GainMode</td>
<td>(input control) Gain type. Default Value: 'fixed' Value List: GainMode ∈ {'fixed', 'frame'}</td>
</tr>
</tbody>
</table>
**Gain1** (input control) .................................................. real \( \sim \) double
Kalman gain / foreground adaptation time.
Default Value: 0.002
Value Suggestions: \( \text{Gain1} \in \{10.0, 20.0, 50.0, 0.1, 0.05, 0.01, 0.005, 0.001\} \)
Restriction: \( 0.0 \leq \text{Gain1} \)

**Gain2** (input control) .................................................. real \( \sim \) double
Kalman gain / background adaptation time.
Default Value: 0.02
Value Suggestions: \( \text{Gain2} \in \{2.0, 4.0, 8.0, 0.5, 0.1, 0.05, 0.01\} \)
Restriction: \( 0.0 \leq \text{Gain2} \)

**AdaptMode** (input control) ......................................... string \( \sim \) String
Threshold adaptation.
Default Value: ‘on’
Value List: AdaptMode \( \in \{\text{‘on’}, \text{‘off’}\} \)

**MinDiff** (input control) .............................................. real \( \sim \) double
Foreground/background threshold.
Default Value: 7.0
Value Suggestions: MinDiff \( \in \{3.0, 5.0, 7.0, 9.0, 11.0\} \)
Recommended Value Step: 0.2

**StatNum** (input control) ............................................ integer \( \sim \) long
Number of statistic data sets.
Default Value: 10
Value Suggestions: StatNum \( \in \{5, 10, 20, 30\} \)
Recommended Value Step: 5

**ConfidenceC** (input control) ....................................... real \( \sim \) double
Confidence constant.
Default Value: 3.25
Value Suggestions: ConfidenceC \( \in \{4.30, 3.25, 2.82, 2.62\} \)
Recommended Value Step: 0.01
Restriction: \( 0.0 < \text{ConfidenceC} \)

**TimeC** (input control) ............................................... real \( \sim \) double
Constant for decay time.
Default Value: 15.0
Value Suggestions: TimeC \( \in \{10.0, 15.0, 20.0\} \)
Recommended Value Step: 5.0
Restriction: \( 0.0 < \text{TimeC} \)

---

**Example**

```c
/* read Init-Image:*/
read_image(InitImage,’Init_Imgae’) /* initialize BgEsti-Dataset with
fixed gains and threshold adaption: */
create_bg_esti(InitImage,0.7,0.7,’fixed’,0.002,0.02,
’on’,7.0,10.3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,’Image_1’)
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* display the foreground region: */
disp_region(Region1,WindowHandle)
/* read the next image in sequence: */
read_image(Image2,’Image_2’)
/* estimate the Background: */
run_bg_esti(Image2,Region2,BgEstiHandle)
/* display the foreground region: */
disp_region(Region2,WindowHandle)
/* etc. */
/* change parameter in dataset: */
```

```c
set_bg_esti_params(BgEstiHandle,0.7,0.7,'fixed',
                    0.004,0.08,'on',9.0,10,3.25,20.0)
/* read the next image in sequence: */
read_image(Image3,'Image_3')
/* estimate the Background: */
run_bg_esti(Image3,Region3,BgEstiHandle)
/* display the foreground region: */
disp_region(Region3,WindowHandle)
/* etc. */
```

Result
---
SetBgEstiParams returns TRUE if all parameters are correct.

Parallelization Information
---
SetBgEstiParams is reentrant and processed without parallelization.

Possible Predecessor Functions
---
CreateBgEsti

Possible Successor Functions
---
RunBgEsti

See Also
---
UpdateBgEsti

Module
---
Background estimation

```c
void HImageX.UpdateBgEsti ([in] HRegionX UpdateRegion,
                            [in] long BgEstiHandle )

void HBgEstiX.UpdateBgEsti ([in] HImageX PresentImage,
                             [in] HRegionX UpdateRegion )

void HOperatorSetX.UpdateBgEsti ([in] HOBJECTX PresentImage,
                                  [in] HOBJECTX UpdateRegion, [in] long BgEstiHandle )
```

Change the estimated background image.

UpdateBgEsti overwrites the image stored in the current BgEsti data set with the grayvalues of PresentImage within the bounds of UpdateRegion. This can be used for a “hard” adaptation: Image regions with a sudden change in (known) background can be adapted very fast this way.

Attention
---
The passed image (PresentImage) must have the same type and size as the background image of the current data set (initialized with CreateBgEsti).

Parameter
---
- **PresentImage** (input iconic) ......................... image ※ HImageX / HOBJECTX ( byte, real )
  Current image.
- **UpdateRegion** (input iconic) .......................... region ※ HRegionX / HOBJECTX
  Region describing areas to change.
- **BgEstiHandle** (input control) .......................... bg_estimation ※ long / HBgEstiX
  ID of the BgEsti data set.

Example
---
/* read Init-Image: */
read_image(InitImage,'Init_Image')
/* initialize BgEsti-Dataset with
   fixed gains and threshold adaption */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02,
               'on',7,10,3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,’Image_1’)
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* use the Region and the information of a knowledge base */
/* to calculate the UpdateRegion */
update_bg_esti(Image1,UpdateRegion,BgEstiHandle)
/* then read the next image in sequence: */
read_image(Image2,’Image_2’)
/* estimate the Background: */
run_bg_esti(Image2,Region2,BgEstiHandle)
/* etc. */

Result
UpdateBgEsti returns TRUE if all parameters are correct.

Parallelization Information
UpdateBgEsti is reentrant and processed without parallelization.

Possible Predecessor Functions
RunBgEsti

Possible Successor Functions
RunBgEsti,GiveBgEsti

See Also
RunBgEsti, GiveBgEsti

Module
Background estimation

12.3 Barcode

Decoding of a sequence of elements of a barcode.

Decode1dBarCode decodes a sequence of elements which have been extracted by Find1dBarCode or Get1dBarCode into a sequence of characters. As input the widths of the elements (in pixels) are used. The discrete form as it is returned from Discrete1dBarCode can optionally be used. Otherwise Decode1dBarCode creates the discrete form automatically.

The result of Decode1dBarCode is a sequence of Characters and the corresponding reference numbers (Reference). In addition a parity check is applied, the result of which is returned in IsCorrect. If all characters are used to store user data (i.e. no parity is used) the value of this parameter has to be ignored.

Parameter

- **BarCodeElements** (input control) ...................... number → VARIANT ( real )
  Widths of the elements of the barcode.
- **BarCodeDescr** (input control) ...................... string → VARIANT ( string, integer, real )
  Description of a bar code class.
- **Characters** (output control) ...................... string → VARIANT ( string )
  Decoded characters in standard interpretation.
- **Reference** (output control) ...................... string → VARIANT ( integer )
  Decoded characters as numbers.
12.3. BARCODE

[72x796]12.3. BARCODE 657

\[ \text{IsCorrect} \] (output control) \hspace{-1em} \leftarrow \text{integer} \sim \text{long} \\
\text{Information whether the barcode is correct.} \\
\text{Value List: } \text{IsCorrect} \in \{0, 1\}

\[ \leftarrow \text{Result} \hspace{-1em} \]

The operator Decode1dBarCode returns the value TRUE if the transferred barcode description is correct and the element list can be decoded.

\[ Parallelization Information \hspace{-1em} \]

Decode1dBarCode is \textit{reentrant} and processed \textit{without} parallelization.

\[ Possible Predecessor Functions \hspace{-1em} \]

Find1dBarCode, Get1dBarCode

\[ Module \hspace{-1em} \]

Barcode reader

\[ \begin{array}{ll}
\text{[out]} & \text{VARIANT SymbolCharacters } \\
\text{[in]} & \text{VARIANT BarCodeDescr, [in]} \text{VARIANT BarCodeDimension,} \\
\text{[in]} & \text{VARIANT BarCodeData, [out]} \text{VARIANT CorrSymbolData,} \\
\text{[out]} & \text{VARIANT DecodedData, [out]} \text{long DecodingError,} \\
\text{[out]} & \text{VARIANT StructuredAppend} \\
\end{array} \]

\[ \text{void HOperatorSetX.Decode2dBarCode } \hspace{-1em} \leftarrow \]

\[ \begin{array}{ll}
\text{[in]} & \text{VARIANT BarCodeDescr,} \\
\text{[in]} & \text{VARIANT BarCodeDimension, [in]} \text{VARIANT BarCodeData,} \\
\text{[out]} & \text{VARIANT SymbolCharacters, [out]} \text{VARIANT CorrSymbolData,} \\
\text{[out]} & \text{VARIANT DecodedData, [out]} \text{long DecodingError,} \\
\text{[out]} & \text{VARIANT StructuredAppend} \\
\end{array} \]

Decode 2D bar code data.

Decode2dBarCode decodes (binary) data of a 2D bar code that has been extracted with the help of Get2dBarCode or Get2dBarCodePos. The parameter BarCodeData contains the data values, BarCodeDimension describes the size of the data matrix (see operator Get2dBarCode), and BarCodeDescr describes the 2D bar code class (see operator Gen2dBarCodeDescr).

First, the binary data are converted into a stream of 8-bit characters (SymbolCharacters). This stream consists of the actual, possibly erroneous data and of additional characters used by the subsequent error correcting routine. If all errors could be corrected, the resulting, still undecoded data is returned in the parameter CorrSymbolData. The parameter DecodingError contains the number of corrected errors, or a negative number if the correction failed.

Then, the data is decoded and returned as a tuple of ASCII characters in the parameter DecodedData. If the symbol is part of a group of symbols (ECC 200: “Structured Append”), the parameter StructuredAppend contains the number (place) of the symbol in the group, the overall number of symbols in the group, and a group identifier. If the symbol is not part of a group, the first two entries of StructuredAppend contain a 1.

\[ Parameter \hspace{-1em} \]

\[ \begin{array}{ll}
\text{BarCodeDescr} \hspace{-1em} \leftarrow \text{string} & \sim \text{VARIANT( string, integer, real )} \\
\text{Description of the bar code class.} \\
\text{BarCodeDimension} \hspace{-1em} \leftarrow \text{integer} & \sim \text{VARIANT( integer )} \\
\text{Tuple with the dimension of the examined symbol. In the case of ECC 200: width, height, symbol code.} \\
\text{BarCodeData} \hspace{-1em} \leftarrow \text{integer} & \sim \text{VARIANT( integer )} \\
\text{Tuple with the data values of the examined symbol.} \\
\text{SymbolCharacters} \hspace{-1em} \leftarrow \text{string} & \sim \text{VARIANT( string )} \\
\text{Data and error codewords of the symbol.} \\
\text{CorrSymbolData} \hspace{-1em} \leftarrow \text{integer} & \sim \text{VARIANT( integer )} \\
\text{Corrected data codewords of the symbol.} \\
\text{DecodedData} \hspace{-1em} \leftarrow \text{integer} & \sim \text{VARIANT( integer )} \\
\text{Decoded data characters as numbers.} \\
\text{DecodingError} \hspace{-1em} \leftarrow \text{integer} & \sim \text{long} \\
\text{Number of errors during the decoding process.} \\
\end{array} \]
StructuredAppend (output control) ........................................... integer ~ VARIANT ( integer )

If the symbol belongs to a group (“structured append”): position in the group, total symbol number, group
(“file”) id.

Result

The return value signals incorrect parameters as well as a failure in the decoding procedure. The error code 8812,
e.g., signals that the decoded data stream contained an invalid data word. Up to now, user-defined control words
can not be handled by the operator. If such control words are detected in the decoded data stream, the error code
8813 is returned.

Parallelization Information

Decode2DBarCode is reentrant and processed without parallelization.

Possible Predecessor Functions

Get2DBarCode

Module

Barcode reader

Generate a discrete barcode from the elements widths.

Discrete1DBarCode converts the list of element widths (output from Find1DBarCode or
Get1DBarCode) into a discrete barcode. Thus every element is then represented by its number of mod-
ules (1,2,...) and no longer as its width in pixels.

This operator is used if the barcode type is not available so that Decode1DBarCode cannot be applied, thus
the user wants to find the barcode with the help of HALCON operators and then himself decode the barcode.
To create the barcode description the operator Gen1DBarCodeDescrGen is used and with Find1DBarCode
the element widths are extracted. Then Discrete1DBarCode is used to create the list of the multiple of the
modules.

Parameter

• Elements (input control) .............................................. number ~ VARIANT ( real )
  List of elements widths of the barcode.

• BarCodeDescr (input control) ................................. string ~ VARIANT ( string, integer, real )
  Description of a bar code class.

• DiscreteBarCode (output control) ............................. number ~ VARIANT ( integer )
  Widths of elements as multiple of modules.

Result

The operator Discrete1DBarCode returns the value TRUE if the transferred barcode description is correct and
the the element list can be discretized.

Parallelization Information

Discrete1DBarCode is reentrant and processed without parallelization.

Possible Predecessor Functions

Find1DBarCode, Get1DBarCode

See Also

Decode1DBarCode

Module

Barcode reader
Look for one barcode in an image.

Find1dBarCode looks for a bar code in an image. As input it needs the description of the barcode as generated by Gen1dBarCodeDescr or Gen1dBarCodeDescrGen. With this description the kind of barcode to be extracted is specified.

The result of the operator are the widths of the elements and the barcode region with its orientation. If a barcode was found BarcodeFound returns the value 1. Otherwise it returns 0.

To control the internal image processing the parameters GenericName and GenericValue are used. This is done by passing the names of the control parameters to be changed in GenericName as a list of strings. In GenericValue the values are passed at the corresponding index positions.

Normally none of the values have to be changed because the operator can adapt automatically to changing situations. Only in the case of difficult capturing conditions or with special barcodes a change should be applied. Here is the list of all parameters which can be changed:

- **amplitude_sobel**: Minimum amplitude for edge extraction: The first step of the search for the barcode is based on the sobel filter. In contrast to thresholding this has the advantage of being independent of different illumination conditions. In the case of a low contrast (difference between bright and dark elements) this value can be chosen lower to find all parts of the barcode. If the contrast is very good the value can be set to a higher value. This results in a better runtime because less pixels have to be evaluated. The effect of this parameter can be checked with the operator SobelDir with edge type 'sum_abs'.
  
  Default value: 70

  Used for: Search for bar code region

- **min_size_element**: During the analysis of elements (or parts of them) this parameter is used to eliminate very small objects which do not belong to the barcode. You have to be aware that with a low image quality elements can be divided into smaller pieces. The parameter has to be larger than the smallest piece. With good image quality the value can be larger to reduce runtime. In any case the value may not be larger than the length of one element (in pixels). The area calculated includes only the border of the elements and is thus smaller. If the value is too large parts of the barcode may be missing.

  Default value: 30

  Used for: Search for bar code region

- **max_size_element**: Similar to the minimal size this parameter is used to suppress large objects. You have to know that with a low resolution multiple elements can be extracted as one intermediate region. The value has thus to be larger than the size of this region. This can be the size of many elements. The area internally calculated includes only the border of the elements. Furthermore the value should be smaller than the total size of the barcode.

  Default value: 15000

  Used for: Search for bar code region
'angle_range' This parameter has the same function as the parameter 'sum_angles' but here it is used during the initial search for elements of the barcode. It has influence on the estimation of the orientation of the elements. For low quality images this value has to be large. A small value results in a reduced runtime. The value is given in degrees. There is no need to change the parameter.

Default value: 24

Used for: Search for bar code region

'correct_angle' This parameter is used to decide if a region can be part of a barcode. For this the orientation of all boundary points are calculated. If enough points have the same orientation the region is accepted for further processing. Increasing this value reduces the amount of accepted regions and improves speed. For low image quality this value has to be low.

Default value: 0.5 (corresponds 50 percent)

Used for: Search for bar code region

'dilation_factor' After the initial step of finding elements these are combined into a barcode region (CodeRegion). This is implemented using the closing operator. The mask size is automatically estimated. If this estimation is wrong it can be corrected using this parameter. If the barcode elements have large gaps the value has to be larger than one. If the barcode is very close to other objects and is wrongly connected to this object the value can be smaller than 1. The parameter is independent of the image quality and runtime.

Default value: 1

Used for: Search for bar code region

'sum_angles' This parameter is used for the final estimation of the orientation of the barcode region (value of Orientation). For this at first the orientation of the boundaries of all elements are calculated. The most frequent orientation is calculated. Using the neighboring orientations the mean orientation is calculated. The parameter determines the amount of neighboring angles used for summing up. The value is given in degrees.

Default value: 40

Used for: Search for bar code region

'min_area_bar_code' This value is used for the final decision if the region is a barcode. The parameter specifies the minimum area of the barcode. This area used here consists only of the boundary pixels of the elements and is thus smaller than the area of CodeRegion. If areas were found which are too small the value has to be increased. The parameter can also be used if small areas with the same orientation as the barcode elements were found. The parameter is independent of the image quality and runtime.

Default value: 1000

Used for: Search for bar code region

'sigma_project' For the extraction of the element widths the gray values of the barcode region are projected in the direction of the elements. The data calculated this way is smoothed to reduce noise. For barcodes with large gaps between the elements this value can be increased if extra elements were detected. The value should be larger than 0.5.

Default value: 0.7

Used for: Extraction of element widths

'amplitude_project' For the calculation of the element widths from the gray projections this parameter controls the minimum amplitude (gray value difference) between dark and light elements. With a lot of noise (e.g. scratches) this value can be increased. For low contrast images the value has to be low.
Default value: 3

Used for: Extraction of element widths

'width_project' The gray value projections are calculated only in the center part of the barcode region. The parameter specifies the half width of this area. If the orientation of the region is not estimated correctly this value has to be small. If there are distortions (e.g. scratches) this value can be increased. For very small barcodes the value can be decreased.

Default value: 25

Used for: Extraction of element widths

'add_length_project' Because the length of the barcode is sometimes to short to get get elements at the beginning and at the end the region is extended based on this parameter. If the barcode lies near to another object resulting in a connection of both areas this value can be decreased.

Default value: 5

Used for: Extraction of element widths

'interpolation_project' With this parameter the type of interpolation used for the gray projection is determined. With a value of 1 a bilinear interpolation is used. With 0 no interpolation is used. In this case the runtime is shorter but it might result in a wrong extraction of the elements.

Default value: 1

Used for: Extraction of element widths

Please note that these parameters should normally not be changed.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image (input iconic)</td>
</tr>
<tr>
<td>CodeRegion (output iconic)</td>
</tr>
<tr>
<td>BarCodeDescr (input control)</td>
</tr>
<tr>
<td>GenericName (input control)</td>
</tr>
<tr>
<td>GenericValue (input control)</td>
</tr>
</tbody>
</table>

- BarcodeFound (output control) Information whether the barcode was found.
- Orientation (output control) Orientation of bar code.

The operator Find1dBarCode returns the value TRUE if the parameters are correct.

Parallelization Information

Find1DBarCode is reentrant and processed without parallelization.
Looking for multiple barcode regions in an image.

Find1DBarCodeRegion looks for multiple barcodes in the image. In contrast to Find1dBarCode, this operator is used if an image contains more than one barcode. Here only the regions but not the widths of the elements are extracted. For every region the orientation in radians is calculated.

The control of the image processing is identical to Find1dBarCode. The description of the parameters GenericName and GenericValue can be found at this operator.

Parameter

- **Image** (input iconic) ................................. image \( \rightarrow \) HImageX / IHObjectX Image with barcodes inside.
- **CodeRegion** (output iconic) ....................... region(-array) \( \rightarrow\) HRegionX / HUntypedObjectX Regions of barcodes.
- **BarCodeDescr** (input control) .................... string \( \rightarrow\) VARIANT (string, integer, real) Description of a bar code class.
- **GenericName** (input control) ....................... string(-array) \( \rightarrow\) VARIANT (string) Names of optional parameters.
  - **Default Value**: []
  - **Value List**: GenericName \( \in \) \{ ’amplitude_sobel’, ’min_size_element’, ’max_size_element’, ’angle_range’, ’correct_angle’, ’dilation_factor’, ’sum_angles’ \}
- **GenericValue** (input control) ..................... number(-array) \( \rightarrow\) VARIANT (integer, real, string) Values of optional parameters.
  - **Default Value**: []
- **Orientation** (output control) ...................... real(-array) \( \rightarrow\) VARIANT (real) Orientation of bar code.

Result

The operator Find1DBarCodeRegion returns the value TRUE if the parameters are correct and at least one barcode is found.

Parallelization Information

Find1DBarCodeRegion is reentrant and processed without parallelization.

Possible Predecessor Functions

Gen1DBarCodeDescr, Gen1DBarCodeDescrGen

Possible Successor Functions

Decode1DBarCode, Discrete1DBarCode

See Also

SobelDir, GrayProjections, GenMeasureRectangle2

Alternatives

Find1DBarCodeRegion

Barcode reader
12.3. BARCODE

Possible Successor Functions

Get1DBarCode, CountObj, SelectObj, ReduceDomain

See Also

SobelDir

Alternatives

Find1DBarCode

Module

Barcode reader

Find regions that might contain a 2D bar code.

Find2dBarCode searches in the image Image for regions that might contain a 2D bar code. Candidate regions are returned in the array of regions CodeRegion. Whether such a region really contains a (readable) 2D bar code can only be determined with the help of the operator Get2dBarCode (or Get2dBarCodePos).

Besides regions that might contain a 2D bar code, the operator passes further, internal information about the regions to the operator for extracting the data from a symbol (Get2dBarCode or Get2dBarCodePos). This information is stored in a region descriptor and returned in the parameter CodeRegDescr.

Depending on the method for printing the bar code ('mode'), different image processing steps are performed. Together with other information about the 2D bar code, the actual printing method is part of a descriptor created with the help of the operator Gen2dBarCodeDescr and passed in the parameter BarCodeDescr.

In the case of difficult conditions, additional control parameters can be passed to the operator with the help of the (optional) generic parameters GenParamNames and GenParamValues, in the form of descriptor-value pairs. One group of parameters describes the appearance of the symbols in the actual images, e.g., the expected size of data elements in pixels. The operator will then check candidate regions against the specified criteria. Suitable values can be determined from the real images, e.g., by manually selecting a region with a bar code and then applying the corresponding operator. If the specification of such parameters does not lead to successfully extracted regions, the user can influence the underlying image processing operators directly with the help of a second group of parameters. As the image processing steps taken depend on the printing method, different parameters are to be applied.

- Parameter that will be passed on in CodeRegDescr:
  - 'module_width' Mean size of a module (data element) in pixels.
    From this quantity, a set of other parameters will be derived. A module should have a size of 4 to 16 pixels in order to be recognized successfully.
    GenParamValues:  > 0
    default:  10

- Parameters for the printing method 'printed':
  The following parameters are used to limit the set of candidate regions with the help of region features. To prevent the use of a certain feature, the corresponding parameter must be set to -1.
  - 'anisometry_max' Maximum anisometry (see operator Eccentricity).
    GenParamValues:  > 1  (-1: feature not used)
    default:  1.45

[out] HRegionX CodeRegion  HImageX.Find2DBarCode
([in] VARIANT BarCodeDescr, [in] VARIANT GenParamNames,
 [in] VARIANT GenParamValues, [out] VARIANT CodeRegDescr )

[out] HRegionX CodeRegion  HBarcodeX.Find2DBarCode ([in] HImageX Image,
 [in] VARIANT BarCodeDescr, [in] VARIANT GenParamNames,
 [in] VARIANT GenParamValues, [out] VARIANT CodeRegDescr )

void  HOperatorSetX.Find2DBarCode ([in] IObjectX Image,
 [out] HUntypedObjectX CodeRegion, [in] VARIANT BarCodeDescr,
 [in] VARIANT GenParamNames, [in] VARIANT GenParamValues,
 [out] VARIANT CodeRegDescr )

HALCON 6.0
'compactness_min' Minimum compactness (see operator Compactness).
GenParamValues: \( \geq 1 \) (-1: feature not used)
default: 1.2

'compactness_max' Maximum compactness.
GenParamValues: \( \geq 'compactness_min' \) (-1: feature not used)
default: 3.0

'circularity_max' Maximum circularity (see operator Circularity).
GenParamValues: 0.0 ... 1.0 (-1: feature not used)
default: 0.8

'circularity_min' Minimum circularity.
GenParamValues: \( \leq 'circularity_max' \) (-1: feature not used)
default: 0.45

'deviation_min' Minimum deviation of gray values (see operator Intensity).
GenParamValues: \( \geq 1.0 \) (-1: feature not used)
default: 20.0

The following parameters are automatically derived from the size of a module, 'module_width', and from parameters of the bar code descriptor BarCodeDescr (see operator Gen2dBarCodeDescr). However, they can be adjusted individually, too.
ATTENTION: If you want to set 'module_width' together with other parameters, make sure set 'module_width' BEFORE the others!

'mean_mask_size_1' Mask size for the first smoothing of the image.
GenParamValues: \( \geq 3 \)
default: \( 2.5 \times 'module_width' \)

'mean_mask_size_2' Mask size for the second smoothing of the image.
GenParamValues: \( \geq 5 \)
default: \( 10.0 \times 'module_width' \)

'area_min' Minimum size of the symbol in pixels squared.
GenParamValues: \( \geq 'area_min' \)
default: \( 0.25 \times 'module_width'^2 \times 'columns_min' \times 'rows_min' \)

'area_max' Maximum size of the symbol in pixels squared.
GenParamValues: \( \geq 'area_min' \)
default: \( 4.00 \times 'module_width'^2 \times 'columns_min' \times 'rows_min' \)

'closing_mask_rad' Mask size for calling the operator ClosingCircle with a test region.
GenParamValues: \( \geq 'module_width' \)
default: 'module_width'

- Parameters for the printing method 'engraved_darkfield':

'compactness_min' Minimum compactness (see operator Compactness).
GenParamValues: \( \geq 1.0 \) (-1: feature not used)
default: 1.2

'compactness_max' Maximum compactness.
GenParamValues: \( \geq 'compactness_min' \) (-1: feature not used)
default: 3.0

'edge_thresh' Maximum compactness.
GenParamValues: 0 ... 255
default: 120
'region_rect2_rel' Relation between the areas of the region and the smallest surrounding rectangle.
GenParamValues: < 1.0
default: 0.7

'median_mask_rad' Mask size for the initial median filtering (see operator MedianImage).
GenParamValues: > 0
default: 0.1 * 'module_width' + 0.5

- Parameters for the printing method 'engraved_lightfield':

'anisometry_max' Maximum anisometry (see operator Eccentricity).
GenParamValues: ≥ 1 (-1: feature not used)
default: 1.45

'compactness_min' Minimum compactness (see operator Compactness).
GenParamValues: ≥ 'compactness_min' (-1: feature not used)
default: 1.2

'compactness_max' Maximum compactness.
GenParamValues: ≥ 'compactness_min' (-1: feature not used)
default: 3.0

'circularity_max' Maximum circularity (see operator Circularity).
GenParamValues: 0...1.0 (-1: feature not used)
default: 0.8

'circularity_min' Minimum circularity.
GenParamValues: ≤ 'circularity_max' (-1: feature not used)
default: 0.45

'deviation_min' Minimum deviation of gray values (see operator Intensity).
GenParamValues: > 1.0 (-1: feature not used)
default: 20.0

'mean_mask_size' Mask size for smoothing the image.
GenParamValues: ≥ 3
default: 10 * 'module_width'

'area_min' Minimum size of the symbol in pixels squared.
GenParamValues: > 0
default: 0.25 * 'module_width'^2 * 'columns_min' * 'rows_min'

'area_max' Maximum size of the symbol in pixels squared.
GenParamValues: ≥ 'area_min'
default: 4.00 * 'module_width'^2 * 'columns_min' * 'rows_min'

'closing_mask_rad' Mask size for calling the operator ClosingCircle with a test region..
GenParamValues: > 0
default: 'module_width'

'opening_mask_rad' Mask size for calling the operator OpeningCircle with a test region..
GenParamValues: > 0
default: 1.5 * 'module_width'

Parameter

- **Image** (input iconic) .............................. image ˜ HImageX / HObjectX
  Image of one or more bar codes.

- **CodeRegion** (output iconic) .......................... region(-array) ˜ HRegionX / HUntypedObjectX
  Regions that might contain a bar code.

- **BarcodeDescr** (input control) .......................... string ˜ VARIANT ( string, integer, real )
  Description of a 2D bar code class to look for.
The operator Find2dBarCode returns the value TRUE if the given parameters are correct. Otherwise, an exception will be raised.

**Parallelization Information**

Find2DBarCode is reentrant and processed without parallelization.

**Possible Predecessor Functions**

Gen2DBarCodeDescr

**Possible Successor Functions**

Get2DBarCode, Get2DBarCodePos

**Module**

Barcode reader

---

**Code Reg Descr** (output control) string

Additional parameters describing the bar code regions. They can be used for extracting the data (see Decode2dBarCode).

---

**Parameters**

**GenParamNames** (input control) string(-array) ~ VARIANT (string)

List of names of (optional) generic parameters controlling the image processing.

Default Value: []

**GenParamValues** (input control) number(-array) ~ VARIANT (integer, real, string)

List of values of generic parameters controlling the image processing.

Default Value: []

**CodeRegDescr** (output control) string ~ VARIANT (string, integer, real)

Additional parameters describing the bar code regions. They can be used for extracting the data (see Decode2dBarCode).

---

**Parameter**

**CodeName** (input control) string ~ String

Name of bar code.

Default Value: 'EAN 13'

Value List: CodeName ∈ {'2/5 industrial', '2/5 interleaved', 'code 39', 'EAN 13', 'EAN 8', 'Codabar'}

**MinCharacters** (input control) integer ~ long

Minimum number of characters (if not fixed).

Default Value: 6

Value Suggestions: MinCharacters ∈ {-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 25, 30}

**MaxCharacters** (input control) integer ~ long

Maximum number of characters (if not fixed).

Default Value: 10

Value Suggestions: MaxCharacters ∈ {-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 25, 30, 35, 40, 50}

**BarCodeDescr** (output control) string ~ VARIANT (string, integer, real)

Description of a bar code class.

---

Generate a description of a 1D bar code.

Gen1dBarCodeDescr generates a description of a one dimensional bar code. This description is used for the search (Find1dBarCode or Find1dBarCodeRegion) and the decoding (Decode1dBarCode) of the barcode. Gen1dBarCodeDescr is therefore the first operator in a program sequence for barcode processing. Gen1dBarCodeDescr has only to be called once at the beginning of a program. The descriptor can be used multiple times. In the case of different types of barcodes the operator has to be called once for each type.

You have to be aware that this description contains only basic informations about the barcode. Thus an arbitrary description can be used to extract almost every type of barcode. On the other hand a specific description is important for the decoding of a barcode type.

---

**CodeName** (input control) string ~ String

Name of bar code.

Default Value: 'EAN 13'

Value List: CodeName ∈ {'2/5 industrial', '2/5 interleaved', 'code 39', 'EAN 13', 'EAN 8', 'Codabar'}

**MinCharacters** (input control) integer ~ long

Minimum number of characters (if not fixed).

Default Value: 6

Value Suggestions: MinCharacters ∈ {-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 25, 30}

**MaxCharacters** (input control) integer ~ long

Maximum number of characters (if not fixed).

Default Value: 10

Value Suggestions: MaxCharacters ∈ {-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 25, 30, 35, 40, 50}

**BarCodeDescr** (output control) string ~ VARIANT (string, integer, real)

Description of a bar code class.
The operator Gen1dBarCodeDescr returns the value TRUE if the transferred barcode name is correct.

**Parallelization Information**

Gen1dBarCodeDescr is reentrant and processed without parallelization.

**Possible Successor Functions**

Find1DBarCode, Find1DBarCodeRegion

**Alternatives**

Gen1DBarCodeDescrGen

**Module**

Barcode reader

```haskell
[out] VARIANT BarCodeDescr HBarcodex.Gen1dBarCodeDescrGen
[in] VARIANT StartElement, [in] VARIANT StopElement,
```

```haskell
void HOperatorSetX.Gen1dBarCodeDescrGen ([in] long MinCodeLength,
[out] VARIANT BarCodeDescr )
```

Generate a generic description of a 1D bar code.

Gen1dBarCodeDescrGen generates a generic description of a one dimensional bar code. This operator is used if the barcode description cannot be generated by Gen1dBarCodeDescr but nonetheless has to be found by using the operators Find1DBarCode or Find1DBarCodeRegion. Note that in this case the barcode cannot be decoded by Decode1dBarCode. Thus in this case the decoding has to be done by the user.

The values for DiscreteCode and ElementSizes have to be set correctly in any case. The other values are not so important for the extraction because they can be estimated by the system automatically.

**Parameter**

- **MinCodeLength** (input control) ................................................ integer ~ long
  Minimum length of the code in modules (including start and stop elements).
  Default Value: 30
  Value List: MinCodeLength ∈ {-1, 30, 60, 90, 110, 130, 150, 200}

- **MaxCodeLength** (input control) ................................................ integer ~ long
  Maximum length of the code in modules (including start and stop elements).
  Default Value: 30
  Value List: MaxCodeLength ∈ {-1, 30, 60, 90, 110, 130, 150, 200}

- **ElementSizes** (input control) ................................................. integer ~ long
  Number of different element sizes.
  Default Value: 2
  Value List: ElementSizes ∈ {1, 2, 3, 4, 5, 6, 7}

- **StartElement** (input control) ............................................... integer(-array) ~ VARIANT ( integer )
  List of elements of the start sequence. The width of an element is given as the number of modules. Gaps are given as negative values.
  Default Value: [1,-1]

- **StopElement** (input control) ............................................... integer(-array) ~ VARIANT ( integer )
  List of elements of the stop sequence. The width of an element is given as the number of modules. Gaps are given as negative values.
  Default Value: [1,-1]

- **MaxSizeRatio** (input control) .............................................. number ~ double
  Maximum ratio length to height.
  Default Value: 2.5
  Value List: MaxSizeRatio ∈ {-1, 2, 3, 4, 5, 6}
**DiscreteCode** (input control) ............................... string $\rightarrow$ String
Discrete code (ignore white elements).

**Default Value:** 'false'
**Value List:** DiscreteCode $\in \{ 'true', 'false' \}

**BarCodeDescr** (output control) .............................. string $\rightarrow$ VARIANT (string, integer, real)
Description of a bar code class.

---

The operator GenldBarCodeDescrGen returns the value TRUE if the transferred values are correct.

---

**Parallelization Information**
GenIDBarCodeDescrGen is reentrant and processed without parallelization.

---

**Possible Successor Functions**
Find1DBarCode, Find1DBarCodeRegion

---

**Alternatives**
Gen1DBarCodeDescr

---

**Module**
Barcode reader

---

[out] VARIANT BarCodeDescr HBbarcodeX.Gen2DBarCodeDescr
([in] String CodeType, [in] VARIANT GenParamNames,
[in] VARIANT GenParamValues )

void HOperatorSetX.Gen2DBarCodeDescr ([in] String CodeType,
[in] VARIANT GenParamNames, [in] VARIANT GenParamValues,
[out] VARIANT BarCodeDescr )

Generate a generic description of a 2D bar code class.

Gen2dBarCodeDescr creates a generic description of a 2D bar code. This description consists of the type of the bar code and further characteristic features of the code.

The parameter CodeType specifies the type of the 2D bar code. Currently, the following types are supported:

'Data Matrix ECC 200'  Data Matrix type ECC 200.

The bar code can be described more closely using the parameters GenParamNames and GenParamValues, which form pairs of descriptors and values. The following parameters describe the size and the form of the symbols:

'columns' Exact number of columns of the symbol.
GenParamValues: 10, 12, 14, 16, 18, 20, 22, 24, 26, 32, 36, 40,
44, 48, 52, 64, 72, 80, 88, 96, 104, 120, 132, 144
default: -

'columns_min' Minimum number of columns.
GenParamValues: 10...144 (even)
default: 10

'columns_max' Maximum number of columns.
GenParamValues: 10...144 (even)

'rows' Exact number of rows of the symbol.
GenParamValues: 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 32, 36, 40,
44, 48, 52, 64, 72, 80, 88, 96, 104, 120, 132, 144
default: -

'rows_min' Minimum number of rows.
GenParamValues: 8...144 (even)
default: 8
**'rows_max'** Maximum number of rows.

```
GenParamValues:  8...144 (even)
default:        144
```

**'shape'** Form of the symbol.

```
GenParamValues:  'square', 'rectangle', 'any'
default:        'any'
```

The following parameters describe the printing method and the appearance of the bar code:

**'foreground'** Appearance of bits corresponding to a logical 1 (foreground).

2D bar codes can be printed dark on light or vice versa. If the appearance of bits corresponding to a logical 1 can be specified in advance, computing time will be significantly reduced, especially when looking for regions that might contain a bar code (operator `Find2dBarCode`).

```
GenParamValues:  'any': unknown appearance
                 'dark','black': symbols are printed dark on light
                 'light','white': symbols are printed light on dark
default:        'any'
```

**'mode'** Method used for printing the 2D bar code.

By specifying this parameter, image processing can be optimized.

```
GenParamValues:  'printed': Standard black-and-white printing. Modules appear as square regions; neighboring modules with the same value form one region.
                 'engraved_darkfield': Modules with the value 1 are engraved into the surface, e.g. by a laser. Using dark field lighting these modules appear as dark round dots with a white corona; neighboring modules with the value 1 are therefore separated by gaps.
                 'engraved_lightfield': Similar to 'engraved_darkfield', but using light field lighting. Modules with the value 1 therefore appear as light dots with a dark corona; neighboring modules with the value 1 are again separated by gaps.

default:        'printed'
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CodeType</strong></td>
<td>(input control) String</td>
</tr>
<tr>
<td>Default Value:</td>
<td>'Data Matrix ECC 200'</td>
</tr>
<tr>
<td>Value List:</td>
<td>CodeType ∈ ['Data Matrix ECC 200']</td>
</tr>
<tr>
<td><strong>GenParamNames</strong></td>
<td>(input control) String</td>
</tr>
<tr>
<td>Default Value:</td>
<td></td>
</tr>
<tr>
<td>Value List:</td>
<td>GenParamNames ∈ ['columns', 'columns_min', 'columns_max', 'rows', 'rows_min', 'rows_max', 'foreground', 'shape', 'mode']</td>
</tr>
<tr>
<td><strong>GenParamValues</strong></td>
<td>(input control) Integer</td>
</tr>
<tr>
<td>Default Value:</td>
<td>[]</td>
</tr>
<tr>
<td>Value List:</td>
<td>GenParamValues ∈ {8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 32, 36, 40, 44, 48, 52, 64, 72, 80, 88, 96, 104, 120, 132, 144, 'dark', 'light', 'square', 'rectangle', 'printed', 'engraved_darkfield', 'engraved_lightfield', 'any'}</td>
</tr>
<tr>
<td><strong>BarCodeDescr</strong></td>
<td>(output control) String</td>
</tr>
<tr>
<td>Result:</td>
<td>BarCodeDescr ∈ String</td>
</tr>
</tbody>
</table>

The operator `Gen2dBarCodeDescr` returns the value TRUE if the given parameters are correct. Otherwise, an exception will be raised.

**Parallelization Information**

`Gen2DBarCodeDescr` is reentrant and processed without parallelization.

HALCON 6.0
### Possible Successor Functions

Find2DBarCode

Barcode reader

<table>
<thead>
<tr>
<th>Possible Successor Functions</th>
</tr>
</thead>
</table>

### Module

Possible Predecessor Functions

Find1DBarCodeRegion, SelectObj, ReduceDomain

### Possible Successor Functions

Decode1DBarCode

### Alternatives

Find1DBarCode

Barcode reader

---

```c
[out] VARIANT BarCodeElements HImageX.Get1DBarCode
([in] VARIANT BarCodeDescr, [in] VARIANT GenericName,
```

```c
[out] VARIANT BarCodeElements HBarcodeX.Get1DBarCode
([in] HImageX BarCodeRegion, [in] VARIANT BarCodeDescr,
[in] VARIANT GenericName, [in] VARIANT GenericValue,
[in] double Orientation )
```

```c
void HOperatorSetX.Get1DBarCode ([in] IHObjectX BarCodeRegion,
[in] VARIANT BarCodeDescr, [in] VARIANT GenericName,
[in] VARIANT GenericValue, [in] double Orientation,
[out] VARIANT BarCodeElements )
```

---

**Extract the widths of the elements inside a barcode region.**

Get1dBarCode extracts the widths of the elements of a barcode inside the specified region.

The control of the processing is identical to Find1dBarCode. The description of the parameters GenericName and GenericValue can be found at this operator.

---

**Parameter**

- **BarCodeRegion** (input iconic) ........................................ image ~ HImageX / IHObjectX Region of bar code.
- **BarCodeDescr** (input control) ................................. string ~ VARIANT ( string, integer, real ) Description of a bar code class.
- **GenericName** (input control) .............................. string(-array) ~ VARIANT ( string ) Names of optional parameters.
  - **Default Value:** []
  - **Value List:** GenericName ∈ { 'sigma_project', 'amplitude_project', 'width_project', 'add_length_project', 'interpolation_project' }
- **GenericValue** (input control) ............................. number(-array) ~ VARIANT ( integer, real ) Values of optional parameters.
  - **Default Value:** []
- **Orientation** (input control) .............................. angle.rad ~ double Orientation of bar code.
- **BarCodeElements** (output control) ....................... number ~ VARIANT ( real ) Widths of elements.

---

**Result**

The operator Get1dBarCode returns the value TRUE if the parameters are correct.

---

**Parallelization Information**

Get1BarCode is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

Find1DBarCodeRegion, SelectObj, ReduceDomain

**Possible Successor Functions**

Decode1DBarCode

---

**Alternatives**

Find1DBarCode

Barcode reader
Extract the values of the data elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”). Get2dBarCode extracts the 2D bar code from a candidate symbol region given in BarCodeRegion and the corresponding image given in Image. BarCodeDescr contains a descriptor of the expected bar code class, which is created with the help of the operator Gen2dBarCodeDescr. CodeRegDescr contains further information regarding the candidate region. This descriptor is created by the operator Find2dBarCode.

If BarCodeRegion contains a 2D bar code, its data, i.e. the bits of the matrix, are written line by line into a tuple and returned in the parameter BarCodeData. Positive values stand for a logical one, negative values for a logical zero. A value of zero signals that the data element could not be classified unambiguously.

The size of the data field is returned in the parameter BarCodeDimension. The first two entries specify the width and the height of the data field, the third contains a so-called symbol index which also describes its size. The size of the data field is not to be confused with the size of the symbol which in turn encompasses additional finder and alignment patterns.

If no 2D bar code could be extracted the operator returns a corresponding error code.

In difficult cases, the user can influence image processing with the help of the (optional) generic parameters GenParamNames and GenParamValues. The width of a module in pixels ("module_width") does not need to be specified, it is part of the region descriptor CodeRegDescr. As in the case of the operator Find2dBarCode, for the different printing methods different image processing steps are applied, leading to different parameters.

As a short overview, image processing falls into the following parts: First, the region BarCodeRegion is divided into a dark and a light region by applying a thresholding operator. Because there must be a “silent” area around each 2D bar code, this area can be examined to determine whether the bar code is printed black-on-white or vice versa. With this information, the dark and the light region can be assigned to the logical data values. Next, BarCodeRegion is approximated by a rectangle. Then, the special finder patterns that each symbol must contain are searched for in the border areas of the rectangle. From the finder pattern, the size of the data matrix and the size and the position of the modules (data elements) can be determined. Finally, the value of the modules is determined.

- **Parameters controlling the classification of modules as being dark or light:**
  - "module_rad_ratio" Part of the module that is to be used for its classification.
    - This parameter controls which part of the module area (in percent, measured from the center) is used to determine its value. With a ‘module_rad_ratio’ of 1.0, the whole area of the module will be examined. With very small ‘module_rad_ratio’, only one point in the middle will be examined.
      - GenParamValues: 0 < ‘module_rad_ratio’ ≤ 1
      - default: 0.7
      - printing method 'printed': 0.3
      - printing method 'engraved_darkfield': 0.7
      - printing method 'engraved_lightfield': 0.3

- ‘use_grayvals’ Method for the classification of modules (Selectable only for the printing methods 'printed' and 'engraved_lightfield': for the printing method 'engraved_darkfield', the gray-value-based method is always used!).
  - The parameter ‘use_grayvals’ controls by which method the modules are classified. If ‘use_grayvals’ is
set to 'no', the module's value is determined by examining whether the part of the module specified by 'module_rad_ratio' is part of the region that has been assigned the value 1. If more than half of the module is part of this region, the module is interpreted as a logical one.

In the case 'use_grayvals' is set to 'yes', the gray values inside 'module_rad_ratio' are examined. For this, the known border parts of the symbol are used to create classifiers with the features “minimum gray value”, “maximum gray value”, and “range of gray values”. A feature will be automatically excluded if it cannot separate the known dark and light regions. It can also be excluded manually with the help of the parameters below. The value of the modules is then determined by applying the trained classifiers. In case of ambiguous results, the majority decision is adopted.

If none of the gray value features can separate the test regions, the operator switches automatically to the region-based method in case the printing method is set to 'printed'. This can be detected by examining the data values returned in BarCodeData. The region-based method yields multiples of 4, the gray-value-based method values between -3 and +3. If separation fails when the printing method is 'engraved_darkfield', an error code is returned. Note, that classification performance can be enhanced by choosing a suitable value for 'module_rad_ratio'.

GenParamValues: 'yes', 'no'
default: 'yes'

'use_grayval_min' Use the feature “gray value minimum” for classifying module values.
GenParamValues: 'yes', 'no'
default: 'yes'

'use_grayval_max' Use the feature “gray value maximum” for classifying module values.
GenParamValues: 'yes', 'no'
default: 'printed', 'engraved_lightfield': 'yes'
'engraved_darkfield': 'no'

'use_grayval_range' Use the feature “gray value range” for classifying module values.
GenParamValues: 'yes', 'no'
default: 'yes'

• Parameters for the printing methods 'printed' and 'engraved_lightfield':

'enlarge_region_rad' Enlarge the symbol region.
GenParamValues: > 0 (≤ 0: enlargement prohibited)
default: 2.5

'thresh_percent' Area of the histogram in which to look for an optimal threshold.

The threshold for separating dark and light regions will be searched for in the middle 'thresh_percent' percent of the gray value histogram. If 'thresh_percent' is set to 0, a threshold exactly in the middle of the histogram is selected. If 'thresh_percent' is set to 100, the whole histogram will be searched. Note, that the threshold influences not only the size of the data region, but also the position of the surrounding rectangle and with it, slightly, the position of the modules.
GenParamValues: 0...100
default: 50

'thresh_step_width' Step width for the search of the optimal threshold.

The histogram area specified in the parameter 'thresh_percent' will be stepped through using 'thresh_step_width' until an optimal separation of dark and light regions is reached. The optimum is declared to be found when the number of individual regions is at its maximum. The smaller 'thresh_step_width' is, the better a threshold can be determined, however at the cost of rising computing time, of course.
GenParamValues: > 0
default: 10

'smooth_cont' Parameter for approximating the region by a polygon (rectangle).

The parameter 'smooth_cont' controls to which degree the contour is smoothed before approximating it with a polygon (see operator SegmentContoursXld).
GenParamValues: 0,3,5,7,9,...
default: 5
'\texttt{max\_line\_dist1}', '\texttt{max\_line\_dist2}' Distance between the contour and its approximation.

The parameters '\texttt{max\_line\_dist1}' and '\texttt{max\_line\_dist2}' control the maximal distance between the contour and its approximation (see operator SegmentContoursXld).

\begin{itemize}
  \item \texttt{GenParamValues}: ≥ 0
  \item default: 4
\end{itemize}

'\texttt{min\_cont\_len}' Minimum length of the contour elements in pixel.

Contour elements shorter than '\texttt{min\_cont\_len}' will be ignored when approximating the region by a rectangle. In the case of small symbols or low resolution, it can be helpful to decrease this parameter.

\begin{itemize}
  \item \texttt{GenParamValues}: > 0
  \item default: 50
\end{itemize}

'\texttt{border\_width\_min}' Minimum width of the border search area.

The border area of the symbol region must contain the so-called finder patterns (ECC 200: two lines with the value 1, two with alternating values). To find these patterns, a search region is constructed along the sides of the rectangle. Starting with '\texttt{border\_width\_min}', the width of the search region is increased until the finder patterns have been detected, or until '\texttt{border\_width\_max}' has been reached.

\begin{itemize}
  \item \texttt{GenParamValues}: ≥ '\texttt{border\_width\_min}'
  \item default: 2
\end{itemize}

'\texttt{border\_width\_max}' Maximum width of the border search area.

\begin{itemize}
  \item \texttt{GenParamValues}: ≥ '\texttt{border\_width\_min}'
  \item default: 5
\end{itemize}

'\texttt{border\_width}' Exact width of the border search area.

Alternatively, the exact width of this search area can be specified in the parameter '\texttt{border\_width}'. This automatically sets '\texttt{border\_width\_min}' and '\texttt{border\_width\_max}'.

\begin{itemize}
  \item \texttt{GenParamValues}: > 0
  \item default: -
\end{itemize}

Parameters for the printing method '\texttt{engraved\_darkfield}':

'\texttt{median\_mask\_rad}' Mask size for the initial median filtering (see operator MedianImage).

\begin{itemize}
  \item \texttt{GenParamValues}: 3, 5, 7, ..., 501
  \item default: '\texttt{module\_width}'*0.1 + 0.5
\end{itemize}

'\texttt{gray\_erosion\_size}' Mask size for the subsequent gray value erosion (see operator GrayErosionRect).

\begin{itemize}
  \item \texttt{GenParamValues}: 3, 5, 7, ..., 511
  \item default: '\texttt{module\_width}'*0.18 + 3.5
\end{itemize}

'\texttt{measure\_sigma}' Parameter for the extraction of linear edge segment pairs (see operator MeasurePairs).

\begin{itemize}
  \item \texttt{GenParamValues}: 0.4...100
  \item default: '\texttt{module\_width}' < 20: 0.7
    \texttt{module\_width}' ≥ 20: 1.4
\end{itemize}

'\texttt{measure\_thresh}' Parameter for the extraction of linear edge segment pairs (see operator MeasurePairs).

\begin{itemize}
  \item \texttt{GenParamValues}: 1...255
  \item default: 2.0
\end{itemize}

\underline{Attention}

The operator Get2dBarCode returns error codes to signal that the candidate region did not contain a valid 2D bar code. As the region candidates extracted by the operator Find2dBarCode can perfectly well encompass regions without a bar code, every call to Get2dBarCode should be surrounded by error handling code (see example).

\underline{Parameter}

\begin{itemize}
  \item \texttt{BarCodeRegion} (input iconic) \hspace{1cm} \texttt{region} \rightsquigarrow \texttt{HRegionX} / \texttt{IHObjectX}
  \item Region that might contain a 2D bar code.
\end{itemize}
\textbf{Image} (input iconic) \textit{HimageX / IObjectX}  
Corresponding image.

\textbf{BarCodeDescr} (input control) \textit{string} \textit{\sim} \textit{VARIANT (string, integer, real)}  
Description of the bar code class.

\textbf{CodeRegDescr} (input control) \textit{string} \textit{\sim} \textit{VARIANT (string, integer, real)}  
Additional parameters describing the bar code region. They can be used for extracting the data.

\textbf{GenParamNames} (input control) \textit{string} \textit{\sim} \textit{VARIANT (string)}  
List of names of (optional) generic control parameters.

\textbf{GenParamValues} (input control) \textit{number} \textit{\sim} \textit{VARIANT (integer, real)}  
List of values of the generic parameters.

\textbf{BarCodeDimension} (output control) \textit{integer} \textit{\sim} \textit{VARIANT (integer)}  
Tuple with the dimension of the extracted symbol. In the case of ECC 200: data field width, height, symbol index.

\textbf{BarCodeData} (output control) \textit{integer} \textit{\sim} \textit{VARIANT (integer)}  
Tuple with the data values of the extracted symbol. value \(>0\): logical 1, value \(<0\): logical 0, value \(=0\): module could not be classified.

\textbf{Result}  
The return value can signal incorrect parameters as well as the failure to extract a 2D bar code. Such a failure can be due to different causes: If no surrounding rectangle was found, the operator returns the error code 8808. Error code 8807 signals that the rectangle does not lie completely inside the image. If no finder pattern was detected inside the border search area, the error code 8810 is returned. If the two continuous lines forming the ‘L’ of the finder pattern could not be found, the error code 8809 is returned. In the case of the printing method ‘engraved darkfield’, the error code 8811 signals that none of the gray value features can separate the test regions, i.e. that modules cannot be classified.

\textbf{Parallelization Information}  
Get2DBarCode is reentrant and processed without parallelization.

\textbf{Possible Predecessor Functions}  
Find2DBarCode

\textbf{Possible Successor Functions}  
Decode2DBarCode

\textbf{Alternatives}  
Get2DBarCodePos

\textbf{Module}  
Barcode reader
Get2dBarCodePos extracts the 2D bar code data from a candidate symbol region given in BarCodeRegion and the corresponding image given in Image elements. In contrast to Get2dBarCode, this operator also returns the positions of the data elements in the image in the parameters DataElementRow and DataElementCol. This can be used to check the result of the operator, e.g., by displaying the data in the real image. For a description of the other parameters, see Get2dBarCode.

Attention

The operator Get2dBarCodePos returns error codes to signal that the candidate region did not contain a valid 2D bar code. As the region candidates extracted by the operator Find2dBarCode can perfectly well encompass regions without a bar code, every call to Get2dBarCodePos should be surrounded by error handling code (see example).

Parameter

- **BarCodeRegion** (input iconic) ...........................................region  \( \sim \) HRegionX / IHOBJECTX
  Region that might contain a bar code.
- **Image** (input iconic) ............................................image  \( \sim \) HImageX / IHOBJECTX
  Corresponding image.
- **BarCodeDescr** (input control) .................................string  \( \sim \) VARIANT ( string, integer, real )
  Description of the 2D bar code class.
- **CodeRegDescr** (input control) .................................string  \( \sim \) VARIANT ( string, integer, real )
  Additional parameters describing the bar code region. They can be used for extracting the data.
- **GenParamNames** (input control) ...............................string  \( \sim \) VARIANT ( string )
  List of names of (optional) generic control parameters.
  **Default Value**: []
- **GenParamValues** (input control) ...............................number  \( \sim \) VARIANT ( integer, real )
  List of values of the generic parameters.
  **Default Value**: []
- **BarCodeDimension** (output control) ............................integer  \( \sim \) VARIANT ( integer )
  Tuple with the dimension of the extracted symbol. In the case of ECC 200: data field width, height, symbol index.
- **BarCodeData** (output control) .................................integer  \( \sim \) VARIANT ( integer )
  Tuple with the data values of the extracted symbol. value > 0: logical 1, value < 0: logical 0, value = 0: module could not be classified.

12.3. BARCODE
The return value can signal incorrect parameters as well as the failure to extract a 2D bar code. Such a failure can be due to different causes: If no surrounding rectangle was found, the operator returns the error code 8808. Error code 8807 signals that the rectangle does not lie completely inside the image. If no finder pattern was detected inside the border search area, the error code 8810 is returned. If the two continuous lines forming the ‘L’ of the finder pattern could not be found, the error code 8809 is returned. In the case of the printing method 'engraved darkfield', the error code 8811 signals that none of the gray value features can separate the test regions, i.e., that modules cannot be classified.

Parallelization Information

Get2DBarCodePos is reentrant and processed without parallelization.

Possible Predecessor Functions

Find2DBarCode

Possible Successor Functions

Decode2DBarCode

See Also

Gen2DBarCodeDescr

Alternatives

Get2DBarCode

Module

Barcode reader

12.4 Calibration

Read the mark center points from the calibration table description file.

CaltabPoints is used to read the mark center points from the calibration table description file CalTabDescrFile. The mark center points are 3D coordinates in the calibration table coordinate system and describe the 3D model of the calibration table. The application of the operator CameraCalibration projects these model points into the image. By minimizing the distance between the projected model points and the observed 2D coordinates in the image (see FindMarksAndPose) the exact values for the internal and external camera parameters are computed.

Parameter

**CalTabDescrFile** (input control) ............................... string  \( \sim \) String  
File name of the calibration table description.

**Default Value**: 'caltab.descr'

**X** (output control) ............................... real  \( \sim \) VARIANT ( real )  
X-coordinates of the mark center points.

**Y** (output control) ............................... real  \( \sim \) VARIANT ( real )  
Y-coordinates of the mark center points.

**Z** (output control) ............................... real  \( \sim \) VARIANT ( real )  
Z-coordinates of the mark center points.

Example
12.4. Calibration

```
// read_image(Image1,'calib-01.tiff')
// find calibration pattern */
find_caltab(Image1,Caltab1,'caltab.descr',3,112,5)
/* find calibration marks and start poses */
find_marks_and_pose(Image1,Caltab1,'caltab.descr',[0.008,0.0,0.000011,0.000011,384,288,768,576],128,10,18,0.9,15.0,100.0,RCoord1,CCoord1,StartPose1)
// read 3D positions of calibration marks */
caltab_points('caltab.descr',NX,NY,NZ) >
// camera calibration
camera_calibration(NX,NY,NZ,RCoord1,CCoord1,StartPose1,11,CamParam,FinalPose,Errors)
// visualize calibration result
disp_image(Image1,WindowHandle)
set_color(WindowHandle,'red')
disp_caltab('caltab.descr',CamParam,FinalPose,1.0).
```

Result

CaltabPoints returns TRUE if all parameter values are correct and the file CalTabDescrFile has been read successfully. If necessary, an exception handling is raised.

Parallelization Information

CaltabPoints is reentrant and processed without parallelization.

Possible Successor Functions

See Also

FindCaltab,FindMarksAndPose,CameraCalibration,DispCaltab,SimCaltab,Project3DPoint,GetLineOfSight,CreateCaltab

Module

Camera calibration

```

```

Determine all camera parameters by a simultaneous minimization process.

CameraCalibration performs the calibration of a video camera. Thus, known 3D model points (with coordinates NX, NY, NZ) are projected in the image and the sum of the squared distance between these projections and the corresponding image points (with coordinates NRow, NCol) is minimized.

In case of converging, the exact internal (CamParam) and external (NFinalPose) camera parameters are determined by this minimization algorithm. The parameters StartCamParam and NStartPose are used as initial values for the minimization process. Since this algorithm simultaneously handles correspondences between image and model points from different images, it is also called multi image calibration.

In general camera calibration means the exact determination of the parameters, which model the (optical) projection of any 3D world point \( P_W \) into a (sub-)pixel \([r,c]\) on the CCD sensor of the camera. This is of importance, if the original 3D pose of an object has to be computed using an image (e.g., measuring of industrial pieces).

The underlying camera model is a pinhole camera with radial distortions if the focal length passed in StartCamParam is greater than 0. It describes the transform of a 3D world point \( P_W \) into a (sub-)pixel \([r,c]\) of the video image by the following equations:

---

HALCON 6.0
The use of CameraCalibration

External camera parameters: These 7 parameters describe the position and orientation of the camera and are also called as camera pose. The relative position of the camera with regard of a world or object coordinate system is defined by a 3D translational vector (parameters 1, 2, and 3). The orientation of the camera is described by the three rotation angles around the three coordinate axes (parameters 4, 5, and 6).

The last value within the camera pose encodes the representation type of the described 3D transform: E.g., the value ‘0’ is the code of the representation type 1 and says, that the transform of a point is described, the 3D rotation is performed before the 3D translation, that the three rotations are specified as angles (in degree), and that the rotation order is $\gamma\beta\alpha$, i.e., the first rotation is around the Z-axis, the second one around the new Y-axis, and the third one around the new X-axis, which is generated after the second rotation.

CameraCalibration operates with all types of 3D transform for NStartPose. The sense of the other representation types are explained at CreatePose.

Note, that in contrast to the external camera parameters the internal ones are the same for all positions and orientations of the camera.

The use of CameraCalibration leads to some questions, which are dealt with in the following sections:
**How to generate a appropriate calibration table?** The simplest method to determine the internal parameter of a CCD camera is the use of the planar calibration table generated by the operator CreateCaltab. In case of small distances between object and lens it may be sufficient to print the calibration pattern by a laser printer and to mount it on a cardboard. Otherwise – especially by using a wide-angle lens – it is possible to print the PostScript file on a large inkjet printer and to mount it on an aluminum table. It is very important, that the mark coordinates in the calibration table description file correspond to the real ones on the calibration table with high accuracy. Thus, the calibration table description file has to be modified in accordance with the measurement of the calibration table!

**How to take a set of suitable images?** If you use the planar calibration table, you can proceed in the following way: With the combination of lens (fixed distance!), camera, and frame grabber to be calibrated a set of images of the calibration table has to be taken, see OpenFramegrabber and GrabImage. The following items have to be considered:

- At least a total of 10 to 20 images should be taken into account.
- The calibration table has to be completely visible (incl. border!).
- Reflections etc. on the calibration table should be avoided.
- Within the set of images the calibration table should appear in different positions and orientations: Once left in the image, once right, once (left and right) at the bottom, once (left or right) at the top, from different distances etc. At this, the calibration table should be rotated a little around its X- and/or Y-axis, so the perspective distortions of the calibration pattern are clearly visible. Thus, the external camera parameters (camera pose with regard of the calibration table) should be set to a large variety of different values!
- The calibration table should fill at least a quarter of the whole image to ensure the robust detection of the marks.

**How to extract the calibration marks in the images?** If a planar calibration table is used, for each image the operators FindCaltab and FindMarksAndPose can be used to determine the coordinates of the calibration marks and to compute a rough estimate for the external camera parameters. The concatenation of these values can directly be used as initial values for the external camera parameters (NStartPose) in CameraCalibration.

Obviously, images, on which the segmentation of the calibration table (FindCaltab) has failed or the calibration marks haven’t been determined successfully by FindMarksAndPose, should not be used.

**How to find suitable initial values for the internal camera parameters?** The operators FindMarksAndPose (determination of initial values for the external camera parameters) and CameraCalibration require initial values for the internal camera parameters. These parameters can be provided by a appropriate text file (see ReadCamPar), which can be generated by WriteCamPar or can be edited manually.

The following should be considered for the initial values of the single parameters:

- **Focus**: The initial value is the nominal focal length of the used lens, e.g., 0.008 m.
- **Kappa**: Use 0.0 as initial value. The calibrated value normally lies between -1000.0 and -50000.0 \(1/m^2\) depending on the used lens.
- **Sx**: The initial value for the horizontal distance between two neighboring CCD cells depends on the dimension of the used CCD chip of the camera (see technical specifications of the camera). Generally, common CCD chips are either 1/3"-Chips (e.g., SONY XC-73, SONY XC-777), 1/2"-Chips (e.g., SONY XC-999, Panasonic WV-CD50), or 2/3"-Chips (e.g., SONY DXC-151, SONY XC-77). Notice: The value of Sx increases, if the image is subsampled! Appropriate initial values are:

<table>
<thead>
<tr>
<th>Chip</th>
<th>Full image (768*576)</th>
<th>Subsampling (384*288)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3&quot;-Chip</td>
<td>0.0000055 m</td>
<td>0.0000110 m</td>
</tr>
<tr>
<td>1/2&quot;-Chip</td>
<td>0.0000086 m</td>
<td>0.0000172 m</td>
</tr>
<tr>
<td>2/3&quot;-Chip</td>
<td>0.0000110 m</td>
<td>0.0000220 m</td>
</tr>
</tbody>
</table>

The value for Sx is calibrated, since the video signal of a CCD camera normally isn’t sampled pixel-synchronously.

- **Sy**: Since most off-the-shelf cameras have quadratic pixels, the same values for Sy are valid as for Sx. In contrast to Sx the value for Sy will NOT be calibrated for pinhole cameras, because the video signal of a CCD camera normally is sampled line-synchronously. Thus, the initial value is equal to the final value. Appropriate initial values are:...
Full image (768*576) Subsampling (384*288)

1/3"-Chip 0.0000055 m 0.0000110 m
1/2"-Chip 0.0000086 m 0.0000172 m
2/3"-Chip 0.0000110 m 0.0000220 m

Cx and Cy: Initial values for the coordinates of the image center is the half image width and half image height. Notice: The values of Cx and Cy decrease, if the image is subsampled! Appropriate initial values are:

Full image (768*576) Subsampling (384*288)

Cx 384.0 192.0
Cy 288.0 144.0

ImageWidth and ImageHeight: These two parameters are set by the the used frame grabber and therefore are not calibrated. Appropriate initial values are:

Full image (768*576) Subsampling (384*288)

ImageWidth 768 384
ImageHeight 576 288

Which camera parameters have to be estimated? The input parameter EstimateParams is used to select which camera parameters to estimate. Usually this parameter is set to "all", i.e., all 6 external camera parameters (translation and rotation) and all internal camera parameters have to be determined. Otherwise, EstimateParams contains a tuple of strings indicating the combination of parameters to estimate.

What is the order within the single parameters? The length of the tuple NStartPose corresponds to the number of calibration images, e.g., using 15 images leads to a length of the tuple NStartPose equal to 15 \cdot 7 = 105 (15 times the 7 external camera parameters). The first 7 values correspond to the camera pose of the first image, the next 7 values to the pose of the second one, etc.

This fixed number of calibration images has to be considered within the tuples with the coordinates of the 3D model marks and the extracted 2D marks. If 15 images are used, the length of the tuples NRow and NCol is 15 times the length of the tuples with the coordinates of the 3D model marks (NX, NY, and NZ). If every image consists 49 marks, the order of the values in NRow and NCol is "image after image", i.e., using 49 marks the first 3D model point corresponds to the 1st, 50th, 99th, 148th, 197th, 246th, etc. extracted 2D mark.

The 3D model points can be read from a calibration table description file using the operator CaltabPoints. Initial values for the camera pose can be determined by applying FindMarksAndPose for each image. The tuple NStartPose is set by the concatenation of all these camera poses.

What is the meaning of the output parameters? If the camera calibration process is finished successfully, i.e., the minimization process is converged, the output parameters CamParam and NFinalPose contain the computed exact values for the internal and external camera parameters. The length of the tuple NFinalPose corresponds to the length of the tuple NStartPose.

The representation types of NFinalPose correspond to the representation type of the first tuple of NStartPose. You can convert the representation type by ConvertPoseType. The sense of all kind of representation types for the 3D transform are explained at CreatePose.

The computed average errors (Errors) give an impression of the accuracy of the calibration. The error values (deviations in x- and y-coordinates) are measured in pixels.

Must I use a planar calibration table? No. The operator CameraCalibration is designed in a way, that the input tuples NX, NY, NZ, NRow, and NCol can contain any 3D/2D correspondences, see the above paragraph explaining the order of the single parameters.

Thus, it makes no difference, how the required 3D model marks and the corresponding extracted 2D marks are determined. On the one hand it is possible to use a 3D calibration pattern, on the other hand you also can use any characteristic points (natural landmarks) with known position in the world. By setting EstimateParams to 6, it is possible to compute the world position of the camera! Hereby at least three 3D/2D-correspondences are necessary as input. Homogeneous transformation matrices are useful to generate NStartPose, see program example in HomMat3dToPose and/or program example in CreatePose for generating NStartPose directly.

---

Attention

The minimization process of the calibration depends on the initial values of the internal (StartCamParam) and

external (NStartPose) camera parameters. The computed average errors Errors give an impression of the accuracy of the calibration. The errors (deviations in x- and y-coordinates) are measured in pixels.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NX</strong> (input control)</td>
<td>Ordered Tuple with all X-coordinates of the calibration marks (in meters).</td>
</tr>
<tr>
<td><strong>NY</strong> (input control)</td>
<td>Ordered Tuple with all Y-coordinates of the calibration marks (in meters).</td>
</tr>
<tr>
<td><strong>NZ</strong> (input control)</td>
<td>Ordered Tuple with all Z-coordinates of the calibration marks (in meters).</td>
</tr>
<tr>
<td><strong>NRow</strong> (input control)</td>
<td>Ordered Tuple with all row-coordinates of the extracted calibration marks (in pixels).</td>
</tr>
<tr>
<td><strong>NCol</strong> (input control)</td>
<td>Ordered Tuple with all column-coordinates of the extracted calibration marks (in pixels).</td>
</tr>
<tr>
<td><strong>StartCamParam</strong> (input control)</td>
<td>Initial values for the internal camera parameters.</td>
</tr>
<tr>
<td><strong>NStartPose</strong> (input control)</td>
<td>Ordered tuple with all initial values for the external camera parameters.</td>
</tr>
</tbody>
</table>
| **EstimateParams** (input control) | Camera parameters to be estimated. Default Value: "all"
Value List: EstimateParams ∈ {"all", 'alpha', 'beta', 'gamma', 'transx', 'transy', 'transz', 'focus', 'kappa', 'cx', 'cy', 'sx', 'sy'} |
| **CamParam** (output control) | Internal camera parameters. |
| **NFinalPose** (output control) | Ordered tuple with all external camera parameters. |
| **Errors** (output control) | Average error distances in pixels. |

Example

```csharp
// read calibration images
read_image(Image1,'calib-01.tiff')
read_image(Image2,'calib-02.tiff')
read_image(Image3,'calib-03.tiff')
// find calibration pattern
find_caltab(Image1,Caltab1,'caltab.descr',3,112,5)
find_caltab(Image2,Caltab2,'caltab.descr',3,112,5)
find_caltab(Image3,Caltab3,'caltab.descr',3,112,5)
// find calibration marks and start poses
find_marks_and_pose(Image1,Caltab1,'caltab.descr',[0.008,0.0,0.000011,0.000011,384,288,768,576],128,10,18,0.9,15.0,100.0,RCoord1,CCoord1,StartPose1)
find_marks_and_pose(Image2,Caltab2,'caltab.descr',[0.008,0.0,0.000011,0.000011,384,288,768,576],128,10,18,0.9,15.0,100.0,RCoord2,CCoord2,StartPose2)
find_marks_and_pose(Image3,Caltab3,'caltab.descr',[0.008,0.0,0.000011,0.000011,384,288,768,576],128,10,18,0.9,15.0,100.0,RCoord3,CCoord3,StartPose3)
// read 3D positions of calibration marks
caltab_points('caltab.descr',NX,NY,NZ)
// camera calibration */
camera_calibration(NX,NY,NZ,[RCoord1,RCoord2,RCoord3],
[CCoord1,CCoord2,CCoord3],[0.008,0.0,0.000011,0.000011,384,288,768,576],
[StartPose1,StartPose2,StartPose3],11,CamParam,NFinalPose,Errors)
// write internal camera parameters to file
write_cam_par(CamParam,'campar.dat')
```

Result

CameraCalibration returns TRUE if all parameter values are correct and the desired camera parameters have been determined by the minimization algorithm. If necessary, an exception handling is raised.
Camera calibration

Parallelization Information

CameraCalibration is reentrant and processed without parallelization.

Possible Predecessor Functions

FindMarksAndPose, CaltabPoints, ReadCamPar

Possible Successor Functions

WritePose, PoseToHomMat3d, DispCaltab, SimCaltab

See Also

FindCaltab, FindMarksAndPose, DispCaltab, SimCaltab, WriteCamPar, ReadCamPar, CreatePose, ConvertPoseType, WritePose, ReadPose, PoseToHomMat3d, HomMat3dToPose, CaltabPoints, CreateCaltab

Module

Camera calibration

HMiscX.ChangeRadialDistortionCamPar


Determine new camera parameters in accordance to the specified radial distortion.

ChangeRadialDistortionCamPar modifies the internal camera parameters in accordance to the specified radial distortion Kappa. Via Mode one of the following modes can be selected:

- **'fixed'**: Only Kappa is modified, the other internal camera parameters remain unchanged. In general this leads to a change of the visible part of the scene.
- **'fullsize'**: The scale factors $S_x$ and $S_y$ and the image center point $[C_x, C_y]^T$ are modified in order to preserve the visible part of the scene. Thus, all points visible in the original video image are also visible in the modified (rectified) image. In general this leads to undefined pixels in the modified image.
- **'adaptive'**: A trade off between the other modes: The visible part of the scene is slightly reduced to prevent undefined pixels in the modified image. Similar to 'fullsize' the scale factors and the image center point are modified.

In all modes the radial distortion coefficient $\kappa$ in CamParOut is set to Kappa. The transformation of a pixel in the modified image into the image plane using CamParOut results in the same point as the transformation of a pixel in the original image via CamParIn.

**Parameter**

- **Mode** (input control) .......................................................... string ≈ String Mode
  - Default Value: 'adaptive'
  - Value Suggestions: Mode ∈ { 'fullsize', 'adaptive', 'fixed' }

- **CamParIn** (input control) .............................................. real ≈ VARIANT( real )
  - Internal camera parameters (original).

- **Kappa** (input control) .................................................. real ≈ double
  - Desired radial distortion.
  - Default Value: 0.0

- **CamParOut** (output control) ........................................... real ≈ VARIANT( real )
  - Internal camera parameters (modified).

**Result**

ChangeRadialDistortionCamPar returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information

ChangeRadialDistortionCamPar is reentrant and processed without parallelization.

Possible Predecessor Functions

CameraCalibration, ReadCamPar
12.4. CALIBRATION  

**Possible Successor Functions**

ChangeRadialDistortionImage, ChangeRadialDistortionContoursXld

**See Also**

CameraCalibration, ReadCamPar, ChangeRadialDistortionImage,
ChangeRadialDistortionContoursXld

**Module**

Camera calibration

```
[out] HXLDCtx ContoursRectified
HXLDCtx.ChangeRadialDistortionContoursXld
([in] VARIANT CamParIn, [in] VARIANT CamParOut)

void HOperatorSetX.ChangeRadialDistortionContoursXld
([in] IHObjectX Contours, [out] HUntypedObjectX ContoursRectified,
 [in] VARIANT CamParIn, [in] VARIANT CamParOut)
```

Change the radial distortion of contours.

ChangeRadialDistortionContoursXld changes the radial distortion of the input contours Contours in accordance to the internal camera parameters CamParIn and CamParOut. Each subpixel of an input contour is transformed into the image plane using CamParIn and subsequently projected into a subpixel of the corresponding contour in ContoursRectified using CamParOut.

If CamParOut was computed via ChangeRadialDistortionCamPar, the contours ContoursRectified are equivalent to Contours obtained with a lens with a modified radial distortion. If \( \kappa \) is 0, the contours are rectified. A subsequent pose estimation (determination of the external camera parameters) is not affected by this operation.

**Parameter**

- **Contours** (input iconic) : xld_cont(-array) \( \sim \) HXLDCtx / IHObjectX
  - Original contours.

- **ContoursRectified** (output iconic) : xld_cont(-array) \( \sim \) HXLDCtx / HUntypedObjectX
  - Resulting contours with modified radial distortion.

- **CamParIn** (input control) : real \( \sim \) VARIANT ( real )
  - Internal camera parameter for Contours.

- **CamParOut** (input control) : real \( \sim \) VARIANT ( real )
  - Internal camera parameter for ContoursRectified.

**Parallelization Information**

ChangeRadialDistortionContoursXld is reentrant and processed without parallelization.

**Possible Predecessor Functions**

ChangeRadialDistortionCamPar, GenContoursSkeletonXld, EdgesSubPix,
SmoothContoursXld

**Possible Successor Functions**

GenPolygonsXld, SmoothContoursXld

**See Also**

ChangeRadialDistortionCamPar, CameraCalibration, ReadCamPar,
ChangeRadialDistortionImage

**Module**

Camera calibration
Change the radial distortion of an image.

ChangeRadialDistortionImage changes the radial distortion of the input image Image in accordance to the internal camera parameters CamParIn and CamParOut. The image size remains unchanged. Each pixel of the output image is transformed into the image plane using CamParOut and subsequently projected into a subpixel of Image using CamParIn. The resulting grayvalue is determined by bilinear interpolation. If the subpixel is outside of Image, the corresponding pixel in ImageRectified is set to ‘black’ and eliminated from the image domain.

If CamParOut was computed via ChangeRadialDistortionCamPar, ImageRectified is equivalent to Image obtained with a lense with a modified radial distortion. If \( \kappa = 0 \), the image is rectified. A subsequent pose estimation (determination of the external camera parameters) is not affected by this operation.

---

**Parameter**

- **Image** (input iconic) ......................... (multichannel-)image(-array) \( \sim \) HImageX / IHObjectX
  - Original image.
- **Region** (input iconic) ................................ region \( \sim \) HRegionX / IHObjectX
  - Region of interest in Image.
- **ImageRectified** (output iconic) .......... (multichannel-)image(-array) \( \sim \) HImageX / HUntypedObjectX
  - Resulting image with modified radial distortion.
- **CamParIn** (input control) ........................................ real \( \sim \) VARIANT ( real )
  - Internal camera parameter for Image.
- **CamParOut** (input control) ........................................ real \( \sim \) VARIANT ( real )
  - Internal camera parameter for Image.

---

**Result**

ChangeRadialDistortionImage returns TRUE if all parameter values are correct and the input is not empty. If necessary, an exception handling is raised.

---

**Parallelization Information**

ChangeRadialDistortionImage is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

- ChangeRadialDistortionCamPar
- ReadImage
- GrabImage

---

**Possible Successor Functions**

- EdgesImage
- Threshold

---

**See Also**

- ChangeRadialDistortionCamPar
- CameraCalibration
- ReadCamPar
- ChangeRadialDistortionContoursXld

---

**Module**

Camera calibration

---

Transform an XLD contour to a world plane represented by external camera parameters.

The operator ContourToWorldPlaneXld transforms contour points given in Contours into a plane in the world coordinate system. The plane is either equal to the plane of the calibration table or parallel to it if the...
operator SetOriginPose has been applied previously to correct the thickness of the calibration table. In the first step the line of sight between the projection center and the image point is computed in the camera system, taking into account the radial distortion using the internal camera parameters CamParam. The line of sight is then transformed into the world coordinate system using the external camera parameters CamPose. By intersecting the plane (Z=0) with the line of sight the pseudo 3d coordinates are obtained, of which the X- and the Y-coordinates are stored in the transformed contour ContoursTrans. It is possible to scale the coordinates with the parameter Scale. This is particularly useful when displaying the coordinates in an image. The parameter Scale must be specified in [unit/pixel]. The original unit is determined by the coordinates of the calibration targets. In many cases this unit is introduced into the calibration as ‘meters’, i.e., one meter in the world coordinate system corresponds to one pixel in the image. In this case, it is possible to set the pixel size directly by selecting ‘m’, ‘cm’, ‘mm’ or ‘microns’ for the parameter Scale.

ContourToWorldPlane returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information
ContourToWorldPlane is reentrant and processed without parallelization.

Possible Predecessor Functions
CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration, SetOriginPose

See Also
ImagePointsToWorldPlane

Camera calibration

Change the representation type of 3D pose parameters.

ConvertPoseType converts the 3D pose PoseIn into a 3D pose PoseOut with a different representation type.

A 3D pose defines a 3D transformation consisting of a translation and a rotation. Halcon supports different representation types for such a transformation. This can be, for example, external camera parameters, given by a 3D translation vector (in meters), three rotation angles, and the code of the representation type of the 3D transformation. For example, the value '0' is the code of representation type 1, and signifies that the transformation of a point
is described, that the 3D rotation is applied before the translation, that the rotation is given by angles (in degrees), and that the order of the rotations is $\gamma \cdot \beta \cdot \alpha$, i.e., the first rotation is around the z-axis, the second rotation around the y-axis, and the third rotation around the x-axis. The meaning of the other representation types is described with the operator CreatePose.

The parameter ViewOfTransform determines whether the new transformation PoseOut describes the transformation of a point ('point') or the transformation of a coordinate system ('coordinate_system'). OrderOfTransform determines whether the rotation ('Rp+T') or the translation ('R(p-T)') is applied first. The meaning of the three rotation values is determined by OrderOfRotation. The values 'gba' and 'abg' signify that the rotation parameters describe the three rotation angles $\alpha$ (around the x-axis), $\beta$ (around the y-axis), and $\gamma$ (around the z-axis) in the 3D transformation. For 'gba', the rotation order is $\gamma \cdot \beta \cdot \alpha$, and for 'abg' it is $\alpha \cdot \beta \cdot \gamma$. If OrderOfRotation is passed as 'rodriguez', the rotation parameters are interpreted as Rodriguez rotation vector.

---

**Parameter**

- **PoseIn** (input control) ........................................pose $\sim$ :none$\gamma$X / VARIANT ( integer, real ) 3D transformation.
  
  **Parameter Number : 7**

- **OrderOfTransform** (input control) ........................................string $\sim$ String Order of rotation and translation.
  
  **Default Value : 'Rp+T'**
  
  **Value Suggestions :** OrderOfTransform $\in \{ 'Rp+T', 'R(p-T)' \}$

- **OrderOfRotation** (input control) ........................................string $\sim$ String Meaning of the rotation values.
  
  **Default Value : 'gba'**
  
  **Value Suggestions :** OrderOfRotation $\in \{ 'gba', 'abg', 'rodriguez' \}$

- **ViewOfTransform** (input control) ........................................string $\sim$ String View of transformation.
  
  **Default Value : 'point'**
  
  **Value Suggestions :** ViewOfTransform $\in \{ 'point', 'coordinate_system' \}$

- **PoseOut** (output control) .........................................pose $\sim$ :none$\gamma$X / VARIANT ( integer, real ) 3D transformation.
  
  **Parameter Number : 7**

---

**Example**

```c
// get pose (external camera parameters):
read_pose ('campose.dat', Pose)
// convert pose to a pose with desired semantic
cconvert_pose_type (Pose, 'Rp+T', 'abg', 'coordinate_system', Pose2).
```

**Result**

ConvertPoseType returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

---

**Parallelization Information**

ConvertPoseType is *reentrant* and processed *without* parallelization.

---

**Possible Predecessor Functions**

CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration

---

**Possible Successor Functions**

WritePose

---

**See Also**

CreatePose, GetPoseType, WritePose, ReadPose

---

**Module**

Camera calibration
Generate calibration table description file and corresponding PostScript file.

CreateCaltab generates the description of a plane calibration table. This calibration table consists of 49 black circular marks on a white plane, which are surrounded by a black frame. The parameter Width sets the width (equal to the height) of the whole calibration table in meters. Using a width of 0.8 m the distance between two neighboring marks becomes 10 cm, and the mark radius and the frame width are set to 2.5 cm.

The file CalTabDescrFile contains the calibration table description, e.g., the number of rows and columns of the calibration table, the geometry of the surrounding frame (see FindCaltab), and the coordinates and the radius of all calibration table marks given in the calibration table coordinate system. A file generated by CreateCaltab looks like the following (comments are marked by a '#' at the beginning of a line):

```plaintext
# Description of the standard calibration table
# used for the CCD-camera calibration in Halcon
# (generated by create_caltab())
#
# Halcon version 4.11 -- Fri Feb 7 16:13:56 1997
#
# 7 rows X 7 columns
# Distance between mark centers [meter]: 0.1
#
# Number of marks per row
# 7
# Number of marks per column
# 7
#
# Quadratic frame (with outer and inner border) around calibration table
# w 0.025
# o -0.41 0.41 0.41 -0.41
# i -0.4 0.4 0.4 -0.4
#
# calibration marks: x y radius [Meter]
#
# calibration marks at y = -0.3 m
# -0.3 -0.3 0.025
# -0.2 -0.3 0.025
# -0.1 -0.3 0.025
# 0 -0.3 0.025
# 0.1 -0.3 0.025
# 0.2 -0.3 0.025
# 0.3 -0.3 0.025
#
# calibration marks at y = -0.2 m
# -0.3 -0.2 0.025
# -0.2 -0.2 0.025
# -0.1 -0.2 0.025
# 0 -0.2 0.025
# 0.1 -0.2 0.025
# 0.2 -0.2 0.025
# 0.3 -0.2 0.025
#
# calibration marks at y = -0.1 m
```

HALCON 6.0
The file `CalTabFile` contains the corresponding PostScript description of the calibration table.

**Attention**

Dependent on the accuracy of the used output device (e.g. laser printer) the printed calibration table may not match the values in the calibration table description file `CalTabDescrFile` exactly. Thus, the coordinates of the calibration marks in the calibration table description file have to be corrected!

**Parameter**

- **Width** (input control) ................................................................. real → double
  
  Width of the calibration table in meters.

  **Default Value**: 0.8
  
  **Value Suggestions**: `Width ∈ {1.2, 0.8, 0.6, 0.4, 0.2, 0.1}`
  
  **Recommended Value Step**: 0.1
  
  **Restriction**: `(0.0 < Width)`
CalTabDescrFile (input control) ........................................ string \rightarrow String
File name of the calibration table description.
Default Value: 'caltab.descr'

CalTabFile (input control) ........................................ string \rightarrow String
File name of the PostScript file.
Default Value: 'caltab.ps'

---

Example

// create calibration table with width = 80 cm
create_caltab(0.8, 'caltab.descr', 'caltab.ps').

---

Result

CreateCaltab returns TRUE if all parameter values are correct and both files have been written successfully. If necessary, an exception handling is raised.

Parallelization Information

CreateCaltab is processed completely exclusively without parallelization.

Possible Successor Functions

ReadCamPar, CaltabPoints

See Also

FindCaltab, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab

Module

Camera calibration

[out] <none> X Pose HPoseX.CreatePose ([in] double TransX,
[in] String ViewOfTransform )

void HOperatorSetX.CreatePose ([in] double TransX, [in] double TransY,
[in] String OrderOfTransform, [in] String OrderOfRotation,
[in] String ViewOfTransform, [out] VARIANT Pose )

Create 3D pose parameters.

CreatePose creates 3D pose parameters Pose, which define a 3D transformation. The translation and rotation of the 3D pose are described by parameters (TransX, TransY, TransZ) and (Rot1, Rot2, Rot3), respectively. The parameter ViewOfTransform determines whether the transformation describes the transformation of a point ('point') or of a coordinate system ('coordinate system'). OrderOfTransform determines whether during the transformation the rotation is applied first ('Rp+T') or whether the translation is applied first ('R(p-T)'). The meaning of the three rotation parameters is determined by OrderOfRotation. The values 'gba' and 'abg' specify that Rot1 is the rotation angle \( \alpha \) (around the x-axis), Rot2 is the rotation angle \( \beta \) (around the y-axis), and Rot3 is the rotation angle \( \gamma \) (around the z-axis). For 'gba', the order is \( \gamma, \beta, \alpha \), while for 'abg' it is \( \alpha, \beta, \gamma \). If OrderOfRotation is passed as 'rodriguez', the rotation parameters are interpreted as Rodriguez rotation vector.

Pose is a tuple of length seven. The first three values hold the translation vector [TransX, TransY, TransZ], the following three parameters hold the rotations Rot1, Rot2, and Rot3, and the last value holds the so-called representation type of the 3D pose parameters, and thus the transformation.

The possible combinations of ViewOfTransform, OrderOfTransform, and OrderOfRotation lead to 12 different representation types for the 3D transformations. In particular, they are:
<table>
<thead>
<tr>
<th>No.</th>
<th>OrderOfTransform</th>
<th>OrderOfRotation</th>
<th>ViewOfTransform</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'Rp+T'</td>
<td>'gba'</td>
<td>'point'</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>'Rp+T'</td>
<td>'abg'</td>
<td>'point'</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>'Rp+T'</td>
<td>'rodriguez'</td>
<td>'point'</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>'Rp+T'</td>
<td>'gba'</td>
<td>'coordinate_system'</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>'Rp+T'</td>
<td>'abg'</td>
<td>'coordinate_system'</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>'Rp+T'</td>
<td>'rodriguez'</td>
<td>'coordinate_system'</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>'R(p-T)'</td>
<td>'gba'</td>
<td>'point'</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>'R(p-T)'</td>
<td>'abg'</td>
<td>'point'</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>'R(p-T)'</td>
<td>'rodriguez'</td>
<td>'point'</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>'R(p-T)'</td>
<td>'gba'</td>
<td>'coordinate_system'</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>'R(p-T)'</td>
<td>'abg'</td>
<td>'coordinate_system'</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>'R(p-T)'</td>
<td>'rodriguez'</td>
<td>'coordinate_system'</td>
<td>13</td>
</tr>
</tbody>
</table>

Note, that in general the symbols \( R \) and \( T \) in the parameter `OrderOfTransform` are not the same as within the common notation of a homogeneous transformation matrix.

The type of a 3D transformation can be queried with `GetPoseType`. The output of this operator, however, is not the code of the number of the transformation type, but the sight and order of the transformation, and the order of the rotations, corresponding to the parameters of `CreatePose`. Different representation types can be converted into one another with `ConvertPoseType`. The description of these conversions is given below.

The representation type no. 1 with code '0' corresponds to a transformation using homogeneous transformation matrices, see `PoseToHomMat3d`, because when multiplying a 3D vector by a homogeneous transformation matrix, the rotation and translation as also determined first, a point is transformed, and the rotation order is set to /AD-

\[
\begin{align*}
R_h &= \begin{bmatrix} x & y & z \end{bmatrix}^T = R^{(1)} \cdot P_a + T^{(1)} \\
&= R_x(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma) \cdot P_a + T^{(1)}
\end{align*}
\]

where:

\[
R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \quad \text{rotation around the x-axis}
\]

\[
R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \quad \text{rotation around the y-axis}
\]

\[
R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{rotation around the z-axis}
\]

\[
R^{(1)} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}
\]

where:

\[
\begin{align*}
r_{11} &= \cos(\beta) \cos(\gamma) \\
r_{12} &= -\cos(\beta) \sin(\gamma) \\
r_{13} &= \sin(\beta) \\
r_{21} &= \sin(\alpha) \sin(\beta) \cos(\gamma) + \cos(\alpha) \sin(\gamma) \\
r_{22} &= -\sin(\alpha) \sin(\beta) \sin(\gamma) + \cos(\alpha) \cos(\gamma) \\
r_{23} &= -\sin(\alpha) \cos(\beta) \\
r_{31} &= -\cos(\alpha) \sin(\beta) \cos(\gamma) + \sin(\alpha) \sin(\gamma) \\
r_{32} &= \cos(\alpha) \sin(\beta) \sin(\gamma) + \sin(\alpha) \cos(\gamma) \\
r_{33} &= \cos(\alpha) \cos(\beta)
\end{align*}
\]

The representation type no. 1 is the reference type for all calibration operators in HALCON. Therefore, the conversions from all other representation types into type 1 will be given below. The inverse conversions can be...
obtained by inverting the calculations below. For the conversions, several basic conversion types can be identified. These are the conversions from the representation types 2, 3, 4, and 7. All other conversions are derived from these basic conversions. The individual conversions are given by:

**Type 2: (\text{"Rp+T", \text{"abg", \text{"point}})\)** With respect to type 1, the order of rotations is reversed. First, a rotation around the x-axis, then around the new y-axis, and finally around the z-axis created by the 2nd rotation is performed. As for type 1, the parameters Rot1, Rot2, Rot3 determine the rotation angles $\alpha$, $\beta$, and $\gamma$. The translation vector $[\text{TransX, TransY, TransZ}]$ is identical for types 1 and 2. For the conversion we have:

$$P_2 = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^T = R \cdot P_1 + T$$

$$= R_x(\gamma(2)) \cdot R_y(\beta(2)) \cdot R_z(\alpha(2)) \cdot P_1 + T(2)$$

$$= R_x(\alpha(1)) \cdot R_y(\beta(1)) \cdot R_z(\gamma(1)) \cdot P_1 + T(1)$$

$$\Rightarrow$$

$$R(1) = R(2) = R_x(\alpha(1)) \cdot R_y(\beta(1)) \cdot R_z(\gamma(1)) \cdot P_1 + T(1)$$

$$= R_z(\gamma(2)) \cdot R_y(\beta(2)) \cdot R_x(\alpha(2)) \cdot P_1 + T(2)$$

where:

$$R(2) = \begin{pmatrix} r_{11}^{(2)} & r_{12}^{(2)} & r_{13}^{(2)} \\ r_{21}^{(2)} & r_{22}^{(2)} & r_{23}^{(2)} \\ r_{31}^{(2)} & r_{32}^{(2)} & r_{33}^{(2)} \end{pmatrix}$$

$$r_{11}^{(2)} = \cos(\gamma(2)) \cos(\beta(2))$$

$$r_{12}^{(2)} = \cos(\gamma(2)) \sin(\beta(2)) \sin(\alpha(2)) - \sin(\gamma(2)) \cos(\alpha(2))$$

$$r_{13}^{(2)} = \cos(\gamma(2)) \sin(\beta(2)) \cos(\alpha(2)) + \sin(\gamma(2)) \sin(\alpha(2))$$

$$r_{21}^{(2)} = \sin(\gamma(2)) \cos(\beta(2))$$

$$r_{22}^{(2)} = \sin(\gamma(2)) \sin(\beta(2)) \sin(\alpha(2)) + \cos(\gamma(2)) \cos(\alpha(2))$$

$$r_{23}^{(2)} = \sin(\gamma(2)) \sin(\beta(2)) \cos(\alpha(2)) - \cos(\gamma(2)) \sin(\alpha(2))$$

$$r_{31}^{(2)} = -\sin(\beta(2))$$

$$r_{32}^{(2)} = \cos(\beta(2)) \sin(\alpha(2))$$

$$r_{33}^{(2)} = \cos(\beta(2)) \cos(\alpha(2))$$

$$\cos(\beta(1)) = \sqrt{r_{11}^{(2)}^2 + r_{12}^{(2)}^2}$$

$$\Rightarrow$$

$$\gamma(1) = \arctan 2 \left( \frac{R(2)_{13}}{\cos(\beta(1))} - \frac{R(2)_{13}}{\cos(\beta(1))} \right)$$

$$\beta(1) = \arctan 2 \left( R(13)_{13}, \cos(\beta(1)) \right)$$

$$\alpha(1) = \arctan 2 \left( \frac{R(2)_{13}}{\cos(\beta(1))} - \frac{R(2)_{13}}{\cos(\beta(1))} \right)$$

$$T(1) = T(2)$$
Here, \( \arctan 2(y, x) \) denotes \( \arctan \left( \frac{2y}{x} \right) \) while taking into account the signs of \( x \) and \( y \), in order to determine the correct quadrant. For the pathological case \( \cos(\beta^{(1)}) = 0 \), the problem is unsolvable because the representation of a rotation by three parameters exhibits a singularity. In this case, the rotation matrix is perturbed minimally, thus avoiding the singularity. (The angle \( \beta \) is modified in steps of 0.00001 degrees.)

**Type 3:** ("Rp+T", 'rodriguez', 'point') The Rodriguez vector is an alternative representation of a rotation in space. The direction of the vector defines the axis of rotation. The length of the vector usually defines the rotation angle with positive orientation. Here, a variation of the Rodriguez vector is used, where the length of the vector defines the tangent of half the rotation angle.

Thus, the parameters Rot1, Rot2, and Rot 3 do not describe rotation angles, but correspond to the three values of the vector \([r_x, r_y, r_z]\). The translation vector \([\text{TransX}, \text{TransY}, \text{TransZ}]\) is identical for types 1 and 3. The conversion of the rotation part of the transformation from type 3 to type 1 is given by converting the Rodriguez vector to the rotation matrix \( R^{(1)} \):

\[
\begin{align*}
    h &= 2 \sqrt{h_r^2 + r_x^2 + r_y^2} \\
    r_{11} &= 1 - h (r_y^2 + r_z^2) \\
    r_{12} &= h (r_x r_y - r_z) \\
    r_{13} &= h (r_x r_z + r_y) \\
    r_{21} &= h (r_y r_x + r_z) \\
    r_{22} &= 1 - h (r_x^2 + r_y^2) \\
    r_{23} &= h (r_y r_z - r_x) \\
    r_{31} &= h (r_z r_x - r_y) \\
    r_{32} &= h (r_z r_y + r_x) \\
    r_{33} &= 1 - h (r_x^2 + r_z^2)
\end{align*}
\]

**Type 4:** ("Rp+T", 'gba', 'coordinate_system') In contrast to type 1, type 4 describes the transformation of a coordinate system instead of the transformation of a point. As for type 1, the parameters Rot1, Rot2, Rot3 define the rotation angles \( \alpha, \beta, \) and \( \gamma \). The translation vector \([\text{TransX}, \text{TransY}, \text{TransZ}]\) is identical for types 1 and 4. The only difference is that the rotation angles are negated. For the conversion we have:

\[
P_2 = [x \quad y \quad z]^T = R \cdot P_1 + T
\]

\[
\begin{align*}
    P &= R_x(\alpha^{(1)}) \cdot R_y(\beta^{(1)}) \cdot R_z(\gamma^{(1)}) \cdot P_1 + T^{(1)} \\
    \text{where:} \\
    T^{(1)} &= T^{(4)} \\
    \alpha^{(1)} &= -\alpha^{(4)} \\
    \beta^{(1)} &= -\beta^{(4)} \\
    \gamma^{(1)} &= -\gamma^{(4)}
\end{align*}
\]

**Type 5:** ("Rp+T", 'abg', 'coordinate_system') For the conversion of type 5 into type 1, a conversion into type 2 can be done first, followed by a conversion from type 2 to type 1; see the explanations there. Types 5 and 2 differ in the sense of the transformation (ViewOfTransform). Thus, as was the case for the conversion from type 4 to type 1, the signs of the rotation angles have to be changed. Again, the parameters Rot1, Rot2, Rot3 denote the rotation angles \( \alpha, \beta, \) and \( \gamma \). The translation vector \([\text{TransX}, \text{TransY}, \text{TransZ}]\) is identical for types 1 and 5. For the conversion to type 2 we have:

\[
P_2 = [x \quad y \quad z]^T = R \cdot P_1 + T
\]

\[
\begin{align*}
    P &= R_z(\gamma^{(5)}) \cdot R_y(\beta^{(5)}) \cdot R_x(\alpha^{(5)}) \cdot P_1 + T^{(5)} \\
    \text{where:} \\
    T^{(2)} &= T^{(5)} \\
    \alpha^{(2)} &= -\alpha^{(5)} \\
    \beta^{(2)} &= -\beta^{(5)} \\
    \gamma^{(2)} &= -\gamma^{(5)}
\end{align*}
\]

**Type 6:** ("Rp+T", 'rodriguez', 'coordinate_system') As for type 3, the rotation for type 6 is given by a Rodrigues vector. The parameters Rot1, Rot2, and Rot3 do not describe angles, but correspond to the three values of the vector \([r_x, r_y, r_z]\). The translation vector \([\text{TransX}, \text{TransY}, \text{TransZ}]\) is identical to type 1. As above, the conversion of the rotation part to type 1 is done in two steps: First, the vector \([r_x, r_y, r_z]\) is converted to
a rotation matrix, resulting in a representation of type 4. See the explanations on the conversion from type 3 to type 1.

The second step is the conversion from type 4 to type 1, where additionally the rotation angles have to be negated. See the explanations on the conversion from type 4 to type 1.

Type 7: ('R(p-T)', 'gba', 'point') In contrast to type 1, type 7 describes a transformation in which the translation is applied first, followed by the rotation. As for type 1, the parameters Rot1, Rot2, and Rot3 define the rotation angles α, β, and γ. For the transformation we have:

\[
\begin{align*}
P_2 &= \begin{bmatrix} x & y & z \end{bmatrix}^T = R^{(1)} \cdot P_1 + T^{(1)} \\
&= R^{(7)} \cdot (P_1 - T^{(7)}) \\
&= R^{(7)} \cdot P_1 - R^{(7)} \cdot T^{(7)} \\
\Rightarrow \quad R^{(1)} &= R^{(7)} \\
T^{(1)} &= -R^{(7)} \cdot T^{(7)}
\end{align*}
\]

Type 8: ('R(p-T)', 'abg', 'point') The conversion from type 8 to type 1 is again carried out in two steps: First, a conversion to type 7 is done. See the explanations on the conversion from type 2 to type 1. In the second step, a conversion from type 7 to type 1 is done. See the explanations there. The parameters Rot1, Rot2, and Rot3 again denote the rotation angles α, β, and γ.

Type 9: ('R(p-T)', 'rodriguez', 'point') The conversion from type 9 to type 1 is again carried out in two steps: First, a conversion to type 7 is done. See the explanations on the conversion from type 3 to type 1. In the second step, a conversion from type 7 to type 1 is done. See the explanations there. The parameters Rot1, Rot2, and Rot3 do not describe rotation angles, but correspond to the three values of the Rodriguez vector \( [r_x, r_y, r_z] \).

Type 10: ('R(p-T)', 'gba', 'coordinate_system') The conversion from type 10 to type 1 is again carried out in two steps: First, a conversion to type 7 is done. See the explanations on the conversion from type 4 to type 1. In the second step, a conversion from type 7 to type 1 is done. See the explanations there. The parameters Rot1, Rot2, and Rot3 again denote the rotation angles α, β, and γ.

Type 11: ('R(p-T)', 'abg', 'coordinate_system') The conversion from type 11 to type 1 is carried out in three steps: First, a conversion to type 8 is done. See the explanations on the conversion from type 5 to type 1. In the second step, a conversion from type 8 to type 7 is done. See the explanations on the conversion from type 2 to type 1. In the third step, the conversion from type 7 to type 1 is done. See the explanations there. The parameters Rot1, Rot2, and Rot3 again denote the rotation angles α, β, and γ.

Type 12: ('R(p-T)', 'rodriguez', 'coordinate_system') The conversion from type 11 to type 1 is also carried out in three steps: First, a conversion to type 10 is done. See the explanations on the conversion from type 6 to type 1. In the second step, a conversion from type 10 to Type 7 is done. See the explanations on the conversion from type 4 to type 1. In the third step, the conversion from type 7 to type 1 is done. See the explanations there. The parameters Rot1, Rot2, and Rot3 do not describe rotation angles, but correspond to the three values of the Rodriguez vector \( [r_x, r_y, r_z] \).

Useful hint: For the description of the pose of a new coordinate system within a base coordinate system, it is usually easiest to start by visualizing both coordinate systems to be identical, i.e., to lie exactly on top of one another. After this, the new coordinate system is translated until its origin rests in the correct position. The vector describing this translation is given by \([\text{TransX}, \text{TransY}, \text{TransZ}]\). After this, the rotations are applied until the desired orientation of the new coordinate system is achieved within the base coordinate system. If the rotation order is chosen as \( \gamma - \beta - \alpha \), first a rotation around the z-axis in positive orientation is done. Then, a rotation around the new y-axis, and finally a rotation around the x-axis resulting from the second rotation is done. The rotation angle around the x-axis corresponds to \( \alpha \), and must be passed in Rot1, the rotation angle around the y-axis corresponds to \( \beta \), and must be passed in Rot2, while the rotation angle around the z-axis corresponds to \( \gamma \) and has to be passed in Rot3. This description of a 3D transformation corresponds to type 10 with code '9'. Thus, \( \text{ViewOfTransform} = '\text{coordinate_system}', \text{OrderOfTransform} = 'R(p-T)', \text{OrderOfRotation} = 'gba' \) have to be passed to the operator.

In the programming example below, it is shown how the initial camera pose for a subsequent call to \( \text{CameraCalibration} \) can be determined, if the position of the camera is coarsely measured in the world coordinate system. This is of special interest if the standard planar calibration target (see \( \text{CreateCalTab} \)) is not used, and thus the operator \( \text{FindMarksAndPose} \) cannot be applied. Instead, natural or artificial landmarks with known world coordinates can, of course, be used for calibration.
Let, for example, the relative camera position with respect to the origin of the world coordinate system be given by $x = 1.08 \, \text{m}$, $y = 0.25 \, \text{m}$, $z = 0.62 \, \text{m}$, i.e., the origin of the camera coordinate system has the world coordinates $[1.08, 0.25, 0.62]$. Let the camera coordinate system be oriented such that the line of sight of the camera corresponds to the positive $z$-axis. The orientation of the camera coordinate system with respect to the world coordinate system can be described by three rotation angles. In the example, the camera coordinate system is first rotated around the $z$-axis of the world coordinate system by 100 degrees, followed by a rotation around the new $x$-axis by -120 degrees. See also the example for `HomMat3dToPose`.

**Parameter**

- **TransX** (input control) .......................................................... real $\rightarrow$ double
  Translation along the $x$-axis.
  
  **Default Value**: 0.1
  
  **Value Suggestions**: $\text{TransX} \in \{-1.0, -0.75, -0.5, -0.25, -0.1, -0.5, -0.25, -0.125, -0.01, 0, 0.01, 0.125, 0.25, 0.5, 0.1, 0.2, 0.25, 0.5, 0.75, 1.0\}$
  
  **Recommended Value Step**: 0.05

- **TransY** (input control) .......................................................... real $\rightarrow$ double
  Translation along the $y$-axis.
  
  **Default Value**: 0.1
  
  **Value Suggestions**: $\text{TransY} \in \{-1.0, -0.75, -0.5, -0.25, -0.1, -0.5, -0.25, -0.125, -0.01, 0, 0.01, 0.125, 0.25, 0.5, 0.1, 0.2, 0.25, 0.5, 0.75, 1.0\}$
  
  **Recommended Value Step**: 0.05

- **TransZ** (input control) .......................................................... real $\rightarrow$ double
  Translation along the $z$-axis.
  
  **Default Value**: 0.1
  
  **Value Suggestions**: $\text{TransZ} \in \{-1.0, -0.75, -0.5, -0.25, -0.1, -0.5, -0.25, -0.125, -0.01, 0, 0.01, 0.125, 0.25, 0.5, 0.1, 0.2, 0.25, 0.5, 0.75, 1.0\}$
  
  **Recommended Value Step**: 0.05

- **Rot1** (input control) .......................................................... real $\rightarrow$ double
  First rotation value.
  
  **Default Value**: 90
  
  **Value Suggestions**: $\text{Rot1} \in \{90, 180, 270\}$
  
  **Typical Range of Values**: $0 \leq \text{Rot1} \leq 0$
  
  **Minimal Value Step**: 0.001
  
  **Recommended Value Step**: 0.2

- **Rot2** (input control) .......................................................... real $\rightarrow$ double
  Second rotation value.
  
  **Default Value**: 90
  
  **Value Suggestions**: $\text{Rot2} \in \{90, 180, 270\}$
  
  **Typical Range of Values**: $0 \leq \text{Rot2} \leq 0$
  
  **Minimal Value Step**: 0.001
  
  **Recommended Value Step**: 0.2

- **Rot3** (input control) .......................................................... real $\rightarrow$ double
  Third rotation value.
  
  **Default Value**: 90
  
  **Value Suggestions**: $\text{Rot3} \in \{90, 180, 270\}$
  
  **Typical Range of Values**: $0 \leq \text{Rot3} \leq 0$
  
  **Minimal Value Step**: 0.001
  
  **Recommended Value Step**: 0.2

- **OrderOfTransform** (input control) ............................................ string $\rightarrow$ String
  Order of rotation and translation.
  
  **Default Value**: ’Rp+T’
  
  **Value Suggestions**: OrderOfTransform $\in \{’Rp+T’, ’R(p-T)’\}$

- **OrderOfRotation** (input control) ............................................ string $\rightarrow$ String
  Meaning of the rotation values.
  
  **Default Value**: ’gba’
  
  **Value Suggestions**: OrderOfRotation $\in \{’gba’, ’abg’, ’rodriguez’\}$
ViewOfTransform (input control)  
View of transformation.

- **Default Value**: 'point'
- **Value Suggestions**: `ViewOfTransform ∈ { 'point', 'coordinate_system' }

Pose (output control)  
3D-Transformation.

- **Parameter Number**: 7

---

Example

// get internal camera parameters:
read_cam_par ('campar.dat', CamParam)
// read 3D world points [WorldPointsX, WorldPointsY, WorldPointsZ]
// extract correspondent 2D image points [PixelsRow, PixelsColumn]
// get rough camera start pose from relative camera pose:
create_pose (1.08, 0.25, 0.62, -120, 0, 100, "R(p-T)", "gba",
"coordinate_system", StartPose)
// calibration of external camera params:
camera_calibration (WorldPointsX, WorldPointsY, WorldPointsZ,
PixelsRow, PixelsColumn, CamParam, StartPose, 6,
Dummy, FinalPose, Errors).

---

Result

CreatePose returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information

CreatePose is reentrant and processed without parallelization.

Possible Successor Functions

PoseToHomMat3d, WritePose, CameraCalibration, HandEyeCalibration

See Also

ConvertPoseType, GetPoseType, HomMat3dToPose, PoseToHomMat3d, WritePose, ReadPose

Alternatives

ReadPose, HomMat3dToPose

Module

Camera calibration

---

```
// Project and visualize the 3D model of the calibration table in the image.

DispCaltab is used to visualize the calibration marks and the connecting lines between the marks of the used calibration table (CalTabDescrFile) in the actual output window. Thus, the 3D model of the calibration table is projected into the image plane using the internal camera parameters (CamParam) and the camera pose (external camera parameters (CamPose). The underlying camera model (pinhole camera with radial distortion) is described in WriteCamPar.

Typically DispCaltab is used to verificate the result of the camera calibration (see CameraCalibration) by superimposing it onto the original image. The actual linewidth can be set by SetLineWidth, the actual color can be set by SetColor.

The parameter ScaleFac influences the number of supporting points to approximate the elliptic contours of the calibration marks. You should increase the number of supporting points, if the image part in the actual output window is displayed with magnification (see SetPart).
```

HALCON 6.0
Parameter

- **WindowHandle** (input control) - window \( \sim HWindowX / long \) Window.jd.
- **CalTabDescrFile** (input control) - string \( \sim String \) File name of the calibration table description.
  
  Default Value: `caltab.descr`
- **CamParam** (input control) - number \( \sim \) VARIANT ( integer, real ) Internal camera parameters.
  
  Parameter Number: 8
- **CamPose** (input control) - pose \( \sim \) none;X \( \sim \) VARIANT ( integer, real ) External camera parameters.
  
  Parameter Number: 7
- **ScaleFac** (input control) - real \( \sim double \) Scaling factor for the visualization.
  
  Default Value: 1.0
  
  Value Suggestions: ScaleFac \( \in \{0.5, 1.0, 2.0, 3.0\}\)
  
  Recommended Value Step: 0.05
  
  Restriction: \(0.0 < ScaleFac\)

Example

```c
// read calibration image
read_image(Image1,'calib-01.tiff')
// find calibration pattern
find_caltab(Image1,Caltab1,'caltab.descr',3,112,5)
// find calibration marks and start poses
find_marks_and_pose(Image1,Caltab1,'caltab.descr',[0.008,0.0, 0.000011,0.000011,384,288,768,576],128,10,18,0.9,15.0,100.0,RCoord1,CCoord1,StartPose1)
// read 3D positions of calibration marks
caltab_points('caltab.descr',NX,NY,NZ)
// camera calibration
camera_calibration(NX,NY,NZ,RCoord1,CCoord1, [0.008,0.0,0.000011,0.000011,384,288,768,576], StartPose1,11,CamParam,FinalPose,Errors)
// visualize calibration result
disp_image(Image1,WindowHandle)
set_color(WindowHandle,'red')
disp_caltab('caltab.descr',CamParam,FinalPose,1.0).
```

Result

DispCaltab returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information

DispCaltab is *local* and processed *completely exclusively* without parallelization.

Possible Predecessor Functions

CameraCalibration, ReadCamPar, ReadPose

See Also

FindMarksAndPose, CameraCalibration, SimCaltab, WriteCamPar, ReadCamPar, CreatePose, WritePose, ReadPose, Project3DPoint, GetLineOfSight

Module

Camera calibration
Segment the calibration table region in the image.

FindCaltab is used to determine the region of a plane calibration table with circular marks in the input image Image. First the input image is smoothed (see GaussImage); the size of the used filter mask is given by SizeGauss. Afterwards a thresholding operator (see Threshold) with minimum gray value MarkThresh and maximum gray value 255 is applied. Among the extracted connected regions the most convex region with almost correct number of holes (corresponding to the dark marks of the calibration table) is selected. Holes with a diameter smaller than the expected size of the marks MinDiamMarks are eliminated to reduce the impact of noise. The number of marks is read from the calibration table description file CalTabDescrFile. The complete explanation of this file can be found within the description of CreateCaltab.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Input image.</td>
</tr>
<tr>
<td>Caltab</td>
<td>Output region.</td>
</tr>
<tr>
<td>CalTabDescrFile</td>
<td>File name of the calibration table description. Default Value: 'caltab.descr'</td>
</tr>
<tr>
<td>SizeGauss</td>
<td>Filter size of the Gaussian. Default Value: 3 Value List: SizeGauss ∈ {0, 3, 5, 7, 9, 11, 13}</td>
</tr>
<tr>
<td>MarkThresh</td>
<td>Threshold value for mark extraction. Default Value: 112 Value List: MarkThresh ∈ {48, 64, 80, 96, 112, 128, 144, 160}</td>
</tr>
<tr>
<td>MinDiamMarks</td>
<td>Expected minimal diameter of the marks on the calibration table. Default Value: 5 Value List: MinDiamMarks ∈ {3, 5, 9, 15, 30, 50, 70}</td>
</tr>
</tbody>
</table>

Example

```java
// read calibration image
read_image(Image,'calib-01.tiff')
// find calibration pattern
find_caltab(Image,Caltab,'caltab.descr',3,112,5).
```

FindCaltab returns TRUE if all parameter values are correct and an image region is found. The behavior in case of empty input (no image given) can be set via SetSystem(:='noObjectResult',<Result>:) and the behavior in case of an empty result region via SetSystem(:='storeEmptyRegion',<true/false>:). If necessary, an exception handling is raised.

Parallelization Information

FindCaltab is reentrant and processed without parallelization.

Possible Predecessor Functions

ReadImage

Possible Successor Functions

FindMarksAndPose

See Also

FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab, CaltabPoints, CreateCaltab

HALCON 6.0
Camera calibration

FindMarksAndPose is used to determine the necessary input data for the subsequent camera calibration (see CameraCalibration). On the one hand the 2D center points \( RCoord \) and the contours have to be elliptically shaped. Contours shorter than \( \text{MinContLength} \) are discarded, just as contours enclosing regions with a diameter larger than \( \text{MaxDiamMarks} \) (e.g., the border of the calibration table).

For the detection of contours a threshold operator is applied to the amplitude of the edge detector. All points with a high amplitude (i.e., borders of marks) are selected. First, the threshold value is set to \( \text{StartThresh} \). If the search for the closed contours or the successive pose estimate fails, this threshold value is successively decreased by \( \text{DeltaThresh} \) down to a minimum value of \( \text{MinThresh} \).

Each of the found contours is refined with subpixel accuracy (see EdgesSubPix) and subsequently approximated by an ellipse. The center points of these ellipses represent a good approximation of the desired 2D image coordinates \( \{RCoord,CCoord\} \) of the calibration mark center points. The order of the values within these two tuples is in row-major order beginning at the upper left in the image. This order must correspond to the order of the calibration marks as described in the calibration table description file \( \text{CalTabDescrFile} \). In the edge image closed contours are searched for: The number of closed contours must correspond to the number of calibration marks as described in the calibration table description file \( \text{CalTabDescrFile} \) and the contours have to be elliptically shaped. Contours shorter than \( \text{MinContLength} \) are discarded, just as contours enclosing regions with a diameter larger than \( \text{MaxDiamMarks} \) (e.g., the border of the calibration table).

Based on the ellipse parameters for each calibration mark a rough estimate for the external camera parameters is computed finally. For that purpose the fixed correspondences between extracted image marks and known model marks are used. The estimate \( \text{StartPose} \) describes the pose of the camera in the calibration table coordinate system (see CreatePose) as desired by the operator CameraCalibration.

**Parameter**

- **Image** (input iconic) .......................... \( \text{image} \sim HImageX / IHObjectX \) (byte)
  Input image.
- **CalTabRegion** (input iconic) ...................... \( \text{region} \sim HRegionX / IHObjectX \)
  Region of the calibration table.
- **CalTabDescrFile** (input control) ................. \( \text{string} \sim \text{String} \)
  File name of the calibration table description.
  **Default Value:** 'caltab.descr'
- **StartCamParam** (input control) .................... \( \text{number} \sim \text{VARIANT} \) (integer, real)
  Initial values for the internal camera parameters.

Extract the 2D calibration marks from the video image and calculate initial values for the external camera parameters.
12.4. CALIBRATION

- **StartThresh** (input control) .................................................. number \(\sim\) long
  Initial threshold value for contour detection.
  Default Value : 128
  Value List : StartThresh \(\in\) \{80, 96, 112, 128, 144, 160\}

- **DeltaThresh** (input control) .................................................. number \(\sim\) long
  Loop value for successive reduction of StartThresh.
  Default Value : 10
  Value List : DeltaThresh \(\in\) \{6, 8, 10, 12, 14, 16, 18, 20, 22\}

- **MinThresh** (input control) .................................................. number \(\sim\) long
  Minimum threshold for contour detection.
  Default Value : 18
  Value List : MinThresh \(\in\) \{8, 10, 12, 14, 16, 18, 20, 22\}

- **Alpha** (input control) .................................................. real \(\sim\) double
  Filter parameter for contour detection, see EdgesImage.
  Default Value : 0.9
  Value Suggestions : Alpha \(\in\) \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1\}
  Typical Range of Values : \(0.2 \leq \text{Alpha} \leq 0.2\)
  Minimal Value Step : 0.01
  Recommended Value Step : 0.1
  Restriction : (Alpha > 0.0)

- **MinContLength** (input control) ............................................. real \(\sim\) double
  Minimum length of the contours of the marks.
  Default Value : 15.0
  Value Suggestions : MinContLength \(\in\) \{10.0, 15.0, 20.0, 30.0, 40.0, 100.0\}
  Restriction : (MinContLength > 0.0)

- **MaxDiamMarks** (input control) ............................................. real \(\sim\) double
  Maximum expected diameter of the marks.
  Default Value : 100.0
  Value Suggestions : MaxDiamMarks \(\in\) \{50.0, 100.0, 150.0, 200.0, 300.0\}
  Restriction : (MaxDiamMarks > 0.0)

- **RCoord** (output control) .................................................. real \(\sim\) VARIANT
  Tuple with row-coordinates of the detected marks.

- **CCoord** (output control) .................................................. real \(\sim\) VARIANT
  Tuple with column-coordinates of the detected marks.

- **StartPose** (output control) .............................................. pose \(\sim\) \{none\} / VARIANT (real, integer)
  Estimation for the external camera parameters.

  Parameter Number : 7

**Example**

```c
// read calibration image
read_image(Image,'calib-01.tiff')

// find calibration pattern
find_caltab(Image,Caltab,'caltab.descr',3,112,5)

// find calibration marks and start pose
find_marks_and_pose(Image,Caltab,'caltab.descr',[0.008,0.0,0.000011,0.000011,384,288,768,576],128,10,18,0.9,15.0,100.0,RCoord,CCoord,StartPose).
```

**Result**

FindMarksAndPose returns TRUE if all parameter values are correct and an estimation for the external camera parameters has been determined successfully. If necessary, an exception handling is raised.

**Parallelization Information**

FindMarksAndPose is reentrant and processed without parallelization.

**Possible Predecessor Functions**

FindCaltab

**Possible Successor Functions**

CameraCalibration
See Also
FindCaltab, CameraCalibration, DispCaltab, ReadCamPar, ReadPose, CreatePose, PoseToHomMat3d, CaltabPoints, CreateCaltab, EdgesSubPix, EdgesImage

Module

Camera calibration

Compute the line of sight corresponding to a point in the image.
GetLineOfSight computes the line of sight corresponding to a pixel \((\text{Row}, \text{Column})\) in the image plane. The line of sight is a (straight) line in the camera coordinate system, which is described by two points \((\text{PX}, \text{PY}, \text{PZ})\) and \((\text{QX}, \text{QY}, \text{QZ})\) on the line. A pinhole or telecentric camera model with radial distortions described by the internal camera parameters \text{CamParam} is used (see CameraCalibration). If a pinhole camera is used, the second point lies on the focal plane, i.e., the output parameter \text{QZ} is equivalent to the focal length of the camera. The equation of the line of sight is given by

\[
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
= \begin{pmatrix}
p_x \\
p_y \\
p_z
\end{pmatrix} + t \begin{pmatrix}
q_x - p_x \\
q_y - p_y \\
q_z - p_z
\end{pmatrix}.
\]

The advantage of representing the line of sight as two points is that with this, it is easier to transform the line in 3D. To do so, all that is necessary is to apply the operator AffineTransPoint3d to the two points.

Parameter

- **Row** (input control) ..................................................... real \(\sim\) VARIANT (real)
  Row coordinate of the pixel.
- **Column** (input control) ..................................................... real \(\sim\) VARIANT (real)
  Column coordinate of the pixel.
- **CamParam** (input control) .................................................number \(\sim\) VARIANT (integer, real)
  Internal camera parameters.

Parameter Number: 8

- **PX** (output control) .....................................................real \(\sim\) VARIANT (real)
  X coordinate of the first point on the line of sight
- **PY** (output control) .....................................................real \(\sim\) VARIANT (real)
  Y coordinate of the first point on the line of sight
- **PZ** (output control) .....................................................real \(\sim\) VARIANT (real)
  Z coordinate of the first point on the line of sight
- **QX** (output control) .....................................................real \(\sim\) VARIANT (real)
  X coordinate of the second point on the line of sight
- **QY** (output control) .....................................................real \(\sim\) VARIANT (real)
  Y coordinate of the second point on the line of sight
- **QZ** (output control) .....................................................real \(\sim\) VARIANT (real)
  Z coordinate of the second point on the line of sight

Example

```c
// get internal camera parameters
read_cam_par('campar.dat', CamParam)
// inverse projection
get_line_of_sight([50,100],[100,200],CamParam:PX,PY,PZ,QX,QY,QZ).
```

GetLineOfSight returns TRUE if all parameter values are correct. If necessary, an exception handling is raised. GetLineOfSight is reentrant and processed without parallelization.

Possible Predecessor Functions

ReadCamPar, CameraCalibration

Possible Successor Functions

AffineTransPoint3D

See Also

CameraCalibration, DispCaltab, ReadCamPar, Project3DPoint, AffineTransPoint3D

Module

Camera calibration

Get the representation type of 3D pose parameters.

With GetPoseType, the representation type of the 3D pose Pose can be queried.

A 3D pose defines a 3D transformation consisting of a translation and a rotation. Halcon supports different representation types for such a transformation. This can be, for example, external camera parameters, given by a 3D translation vector (in meters), three rotation angles, and the code of the representation type of the 3D transformation. For example, the value '0' is the code of representation type 1, and signifies that the transformation of a point is described, that the 3D rotation is applied before the translation, that the rotations are given as angles (in degrees), and that the order of the rotations is $\gamma\beta\alpha$, i.e., the first rotation is around the z-axis, the second rotation around the new y-axis, and the third rotation around the x-axis created by the previous two rotations. The meaning of the other representation types is described with the operator CreatePose.

The returned parameter ViewOfTransform determines, whether the transformation Pose describes the transformation of a point ("point") or the transformation of a coordinate system ("coordinate_system"). OrderOfTransform determines whether the rotation ("Rp\_T") or the translation ("R(p\_T)") is applied first. The meaning of the three rotation values is determined by OrderOfRotation. The values 'gba' and 'abg' signify that the rotation values describe the three rotation angles $\alpha$ (around the x-axis), $\beta$ (around the y-axis), and $\gamma$ (around the z-axis) in the 3D transformation. For 'gba', the rotation order is $\alpha\beta\gamma$, and for 'abg' it is $\alpha\gamma\beta$. If OrderOfRotation is 'rodriguez', the rotation values are interpreted as Rodriguez rotation vector.

Example

```csharp
// get pose (external camera parameters) */
read_pose ('campose.dat', CamPose)
// get semantic type of pose
get_pose_type (CamPose, OrderT, OrderR, SightT).
```

HALCON 6.0
CreatePose returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

GetPoseType is reentrant and processed without parallelization.

Possible Predecessor Functions

CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration

Possible Successor Functions

ConvertPoseType

See Also

CreatePose, ConvertPoseType, WritePose, ReadPose

Module

Camera calibration

```
[in] VARIANT MPointsOfImage, [in] <none>X MRelPoses,
in VARIANT ToEstimate, [in] String StopCriterion, [in] long MaxIterations,
in double MinError, [out] <none>X CamFinalPose, [out] VARIANT NumErrors)
```

Perform a hand-eye-calibration.

The operator HandEyeCalibration can be used to determine the 3D pose of the origin of the camera coordinate system with respect to the origin of the manipulator coordinate system (tool center point). This is typically used for robots equipped with a camera, where the pose of the camera (the “eye”) relative to the manipulator (the “hand”) has to be determined, in order to enable a video-based gripping of objects.

The assumption for the following description is, that the camera is fixed to the end-effector of the manipulator (hand-in-eye-configuration). In this case the calibration determines the (constant) pose of the camera in respect to the tool coordination system of the end-effector. In the other case (the camera is stationary and looks to the moving manipulator) the (constant) pose of the base coordination system of the manipulator in respect to the camera is computed. This can be performed by using HandEyeCalibration, too.

HandEyeCalibration works in a similar manner to the conventional calibration of the exterior camera parameters (camera pose); see CameraCalibration. However, HandEyeCalibration not only determines the camera pose as a simple transformation, but as a chain of transformations from the world coordinate system to the base coordinate system of the active system (robot or vehicle) to the manipulator coordinate system, and finally to the camera coordinate system. Thus, the transformation of a point $P_W$ in world coordinates to the camera coordinate system can be described by:

$$
P_C = [x \ y \ z]^T = R_W \cdot P_W + T_W
$$

where

- $R_W, T_W$: Rotation and translation from the world coordinate system to the base coordinate system of the active system (vehicle)
- $R_m, T_m$: Rotation and translation from the base coordinate system to the manipulator coordinate system
- $R_c, T_c$: Rotation and translation from the manipulator coordinate system to the camera coordinate system
The goal of the hand-eye-calibration is to determine \((R_c, T_c)\). In order to avoid having to construct an exactly measured set-up, a least-squares error adjustment is used, which estimates \((R_e, T_e)\) as well. Therefore, measurements referring to \(k\) different positions of the manipulator are used. Hence, it must be possible to determine the different positions of the manipulator with sufficient accuracy. For a robot arm, they can be determined from the angle positions of the individual joints of the arm. Analogously to the determination of the interior camera parameters (see CameraCalibration), a large number of different manipulator positions is used to achieve more robust results.

A Newton-type algorithm is used to minimize an error function based on normal equations. Therefore, suitable starting values have to be passed for \((R_c, T_c)\), and \((R_e, T_e)\) by the user.

Starting values for \((R_c, T_c)\) can be obtained by a coarse measurement of the set-up. Starting values for \((R_e, T_e)\) can be obtained from the starting values for \((R_c, T_c)\), the exterior camera parameters \((R_m, T_m)\) (pose of the camera in the entire transformation chain; see CameraCalibration), and the corresponding pose \((R_e, T_e)\) of the manipulator.

We have:

\[
P_C = R_a \cdot P_W + T_a = R_c (R_m (R_e \cdot P_W + T_e) + T_m) + T_c
\]

where

\(R_a, T_a\): Rotation and translation from the world coordinate system to the camera coordinate system (camera pose).

The following sections discuss individual questions arising from the use of HandEyeCalibration and are intended to be a guideline for using the operator in an application, as well as to aid the understanding of the operator.

**How do I get 3D model points?** 3D model points, given in the world coordinate system \((NX, NY, NZ)\), and their associated projections to the camera coordinate system \((NRow, NCol)\) form the basis of the hand-eye-calibration. In order to be able to perform a successful hand-eye-calibration, 3D model points must be used, which were obtained from sufficiently many different positions of the manipulator.

Arbitrary known points in the world coordinate system along with their associated projections to the camera coordinate system can be used for the calibration. However, it is usually most convenient to use a standard calibration table, e.g., the one that can be generated with CreateCaltab. By using this calibration table, the calibration table and position of the calibration marks can be extracted by using FindCaltab and FindMarksAndPose, respectively.

Please refer to CameraCalibration for a description of the extraction of the calibration table and its associated 3D model points.

The parameter MPointsOfImage determines the number of 3D model points used for each pose of the manipulator, i.e., for each image. With this, the 3D model points, which are stored in a linearized fashion in \(NX, NY, NZ\), and their corresponding projections \((NRow, NCol)\) can be associated with the corresponding position of the manipulator \((MRelPoses)\).

The 3D model points can be read from a calibration table description file for the standard calibration table with the operator CaltabPoints.

**How do I acquire a suitable set of images?** If a planar standard calibration table is used, the following procedure should be used: Images of the calibration table are taken with the active system to be calibrated, i.e., the combination of the base of the manipulator (the vehicle, in case of mobile systems, which is used as a fixed point for the manipulator), the manipulator (robot arm, pan-tilt-head, etc.), and the camera system; see OpenFramegrabber and GrabImage. For the image acquisition, the following points should be taken into account:

- At least 10 to 20 images from different positions should be taken in which the position of the camera with respect to the calibration table is sufficiently different. The position of the calibration table must not be changed between images.
- In each image, the calibration table must be completely visible (including its border).
- No reflections or other disturbances should be visible on the calibration table.
- The set of images must show the calibration table from very different positions of the manipulator. The calibration table can and should be visible in different parts of the images. Furthermore, it should be slightly to moderately rotated around its x- or y-axis, in order to clearly exhibit distortions of the calibration marks. In other words, the corresponding exterior camera parameters (pose of the camera
with respect to the calibration table) should take on many different values. Consider that the manipulator only has six degrees of freedom (three for rotation and three for translation), while twelve degrees of freedom have to be estimated for BaseFinalPose and CamFinalPose. This means that the space of the degrees of freedom for the manipulator has to be used as exhaustively as possible.

- In each image, the calibration table should fill at least one quarter of the entire image, in order to ensure the robust detection of the calibration marks.
- The interior camera parameters of the camera to be used must have been determined earlier and must be passed in CamParam; see CameraCalibration. Note that changes of the image size, the focal length, the aperture, or the focus effect a change of the interior camera parameters.
- The camera must not be modified between the acquisition of the individual images, i.e., neither focal length, nor aperture, nor focus must be changed, because all calibration images use the same interior camera parameters. Please make sure that the focus is sufficient for the expected changes of the distance the camera from the calibration table. Therefore, bright lighting conditions for the calibration table are important because smaller apertures result in larger depth of focus.

**How do I obtain suitable starting values?** The starting values for \((R_c, T_c)\) and \((R_w, T_w)\) are passed as 3D poses in the parameters CamStartPose and BaseStartPose, just like the exterior camera parameters. CamStartPose can be obtained by measuring the pose of the camera in the manipulator coordinate system. This is the same procedure as the determination of starting values for the calibration of the exterior camera parameters without using the standard calibration table. Use CreatePose to create a pose from your measurements. For the measurement, it is usually easier to determine the translation of the origin of the camera coordinate system within the manipulator coordinate system first, followed by the determination of the orientation.

The camera coordinate system is oriented such that the line of sight of the camera corresponds to the positive z-axis. The orientation of the camera coordinate system with respect to the world coordinate system can be described by three rotation angles. In order to determine the orientation, one can, for example, first “rotate” around the z-axis of the manipulator coordinate system, then around the new y-axis, and finally around the new x-axis created by the previous two rotations. In this case, the rotation angles around the x-, y-, and z-axis correspond directly to the rotation parameters of the manipulator coordinate system. In order to determine \(\theta\) and \(\phi\), one can rotate the coordinate system first around the z-axis, then around the new y-axis, and finally around the new x-axis created by the previous two rotations. In this case, the rotation angles around the x-, y-, and z-axis correspond directly to the rotation parameters of the manipulator coordinate system.

**How do I obtain the poses of the manipulator?** The parameter MRelPoses determines the positions of the manipulator. If \(m\) positions were obtained by positioning the manipulator, these \(m\) positions must be passed linearized. The representation of these 3D poses is the same as for the exterior camera parameters and
the starting parameters CamStartPose and BaseStartPose. 3D poses are tuples of length 7, see CreatePose.

It is extremely important for the algorithm of the hand-eye-calibration that the poses MRelPoses can be obtained exactly from the manipulator. The transformation of the angles of a robot arm to the representation of a pose (rotation and translation) can be performed with CreatePose. It is advisable to use the same procedure as for the determination of the starting value CamStartPose of the camera pose in the manipulator coordinate system. The manipulator coordinate system must be oriented such that it is identical or at least parallel to the base coordinate system in the initial position of the manipulator. In order to determine the poses of the individual manipulator positions, first the three translation parameters between the origin of the base coordinate system and the origin of the manipulator coordinate system should be determined. The translations should be determined along the axes of the base coordinate system when passing them to CreatePose. For the orientation of the coordinate systems with respect to each other, one first “rotates” around the z-axis of the translated coordinate system, then around the new y-axis, and finally around the new x-axis created by the previous two rotations. The three rotation angles are used as parameters to CreatePose. Hence, according to this description of the pose, the following settings must be passed to CreatePose: ’coordinate_system’ for the sight of the transformation, ’R(p-T)’ for the order of the transformations, and ’gba’ for the order of rotations. This corresponds to representation type 10. See also the description of CreatePose.

**How can I exclude individual pose parameters from the estimation?** HandEyeCalibration estimates a maximum of 12 pose parameters. They are three rotation and three translation parameters each for the pose of the base coordinate system in the world coordinate system and for the pose of the camera coordinate system in the manipulator coordinate system. However, it is possible to exclude some of these pose parameters from the estimation. This means that the starting values of the poses remain unchanged and are assumed constant for the estimation of all other pose parameters. The parameter ToEstimate is used to determine which pose parameters should be estimated. In ToEstimate, a list of keywords for the parameters to be estimated is passed. They are:

For the pose of the base coordinate system in the world coordinate system:

- ’baseTx’ = translation along the x-axis
- ’baseTy’ = translation along the y-axis
- ’baseTz’ = translation along the z-axis
- ’baseRa’ = rotation around the x-axis
- ’baseRb’ = rotation around the y-axis
- ’baseRg’ = rotation around the z-axis

For the pose of the camera coordinate system in the manipulator coordinate system:

- ’camTx’ = translation along the x-axis
- ’camTy’ = translation along the y-axis
- ’camTz’ = translation along the z-axis
- ’camRa’ = rotation around the x-axis
- ’camRb’ = rotation around the y-axis
- ’camRg’ = rotation around the z-axis

In order to estimate all 12 of the pose parameters, all 12 keywords can be passed. Alternatively, the keyword ‘all’ can be used.

It is useful to exclude individual parameters from the estimation if some of the pose parameters have been measured exactly. On the other hand, fixing some parameters reduces the degrees of freedom in the estimation, and thus increases the robustness to the estimated parameters, if the manipulator only has limited movement abilities, e.g., a pan-tilt head.

**Which terminating criteria can be used for the error minimization?** The error minimization terminates either after a fixed number of iterations or if the error falls below a given minimum error. The parameter StopCriterion is used to choose between these two alternatives. If ’CountIterations’ is passed, the algorithm terminates after MaxIterations iterations. If StopCriterion is passed as ’MinError’, the algorithm runs until the error falls below the error threshold given in MinError. If, however, the number of iterations reaches the number given in MaxIterations, the algorithm terminates with an error message.

**What is the order of the individual parameters?** The length of the tuple MPointsOfImage corresponds to the number of different positions of the manipulator. The parameter MPointsOfImage determines the number of model points used in the individual positions. If the standard calibration table is used, this means...
49 points per position (image). If 15 images were acquired, \texttt{MPointsOfImage} is a tuple of length 15, where all elements of the tuple have the value 49.

The number of calibration images, which is determined by the length of \texttt{MPointsOfImage}, must also be taken into account for the 3D model point tuples and the extracted 2D marks tuples, respectively. Hence, for 15 calibration images with 49 model points each, the tuples \texttt{NX}, \texttt{NY}, \texttt{NZ}, \texttt{NRow}, and \texttt{NCol} must contain $15 \cdot 49 = 735$ values each. These tuples are ordered according to the image the respective points lie in, i.e., the first 49 values correspond to the 49 model points in the first image. The order of the 3D model points and the extracted 2D model points must be the same in each image.

The length of the tuple \texttt{MRelPoses} corresponds to the number of calibration images. If, for example, 15 images from different positions were acquired, the length of the tuple \texttt{MRelPoses} is $15 \cdot 7 = 105$ (15 times 7 pose parameters). The first seven parameters thus determine the pose of the manipulator in the first image, and so on.

**What do the output parameters mean?** If \texttt{StopCriterion} was set to \texttt{'CountIterations'}, the output parameters are returned, even if the algorithm didn’t converge. If, however, \texttt{StopCriterion} was set to \texttt{'MinError'}, the error must fall below \texttt{'MinError'} in order for output parameters to be returned.

The output parameters are the pose of the base coordinate system in the world coordinate system (\texttt{BaseFinalPose}) and the pose of the camera coordinate system in the manipulator coordinate system (\texttt{CamFinalPose}).

The representation type of \texttt{BaseFinalPose} and \texttt{CamFinalPose} is the same as the corresponding starting values. It can be changed with the operator \texttt{ConvertPoseType}. The description of the different representation types and of their conversion can be found with the documentation of the operator \texttt{CreatePose}.

The parameter \texttt{NumErrors} contains a list of (numerical) errors from the individual iterations of the algorithm. Based on the evolution of the errors, it can be decided whether the algorithm has converged for the given starting values. The error values are returned as 3D deviations in meters. Thus, the last entry of the error list corresponds to an estimate of the accuracy of the returned pose parameters.

---

**Attention**

The quality of the calibration depends on the accuracy of the input parameters (position of the calibration marks and the starting positions \texttt{BaseStartPose}, \texttt{CamStartPose}). Based on the returned error measures \texttt{NumErrors}, it can be decided, whether the algorithm has converged. Furthermore, the accuracy of the returned pose can be estimated. The error measures are 3D differences in meters.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{NX} (input control)</td>
<td>Linear list containing all the X-coordinates of the calibration points (in the order of the image).</td>
</tr>
<tr>
<td>\texttt{NY} (input control)</td>
<td>Linear list containing all the Y-coordinates of the calibration points (in the order of the image).</td>
</tr>
<tr>
<td>\texttt{NZ} (input control)</td>
<td>Linear list containing all the Z-coordinates of the calibration points (in the order of the image).</td>
</tr>
<tr>
<td>\texttt{NRow} (input control)</td>
<td>Linear list containing all the labelled abscissas (in the order of the image).</td>
</tr>
<tr>
<td>\texttt{NCol} (input control)</td>
<td>Linear list containing all the labelled ordinates (in the order of the image).</td>
</tr>
<tr>
<td>\texttt{MPointsOfImage} (input control)</td>
<td>Number of the calibration points for each image.</td>
</tr>
<tr>
<td>\texttt{MRelPoses} (input control)</td>
<td>Measured values of relative pose of gripping device for each image.</td>
</tr>
<tr>
<td>\texttt{BaseStartPose} (input control)</td>
<td>Initial value for robot pose in world coordinate system.</td>
</tr>
<tr>
<td>\texttt{CamStartPose} (input control)</td>
<td>Initial value for camera pose relative to end-effector.</td>
</tr>
<tr>
<td>\texttt{CamParam} (input control)</td>
<td>Internal camera parameters.</td>
</tr>
<tr>
<td>\texttt{ToEstimate} (input control)</td>
<td>Parameters to be estimated (max. 12 free degrees).</td>
</tr>
</tbody>
</table>

**Default Value:** \texttt{'all'}

**Value List:** \texttt{ToEstimate} $\in \{ 'all', 'baseTx', 'baseTy', 'baseTz', 'baseRa', 'baseRb', 'baseRg', 'camTx', 'camTy', 'camTz', 'camRa', 'camRb', 'camRg' \}$
12.4. CALIBRATION

- **StopCriterion** (input control) ........................................... string $\sim$ String
  Type of stopping criterion.
  Default Value: `CountIterations`
  Value List: `StopCriterion` $\in \{`CountIterations', `MinError'\}

- **MaxIterations** (input control) ........................................... integer $\sim$ long
  Maximum number of iterations to be executed.
  Default Value: 15
  Value Suggestions: `MaxIterations` $\in \{10, 15, 20, 25, 30\}

- **MinError** (input control) ............................................... real $\sim$ double
  Minimum error used as the stopping criterion.
  Default Value: 0.0005
  Value Suggestions: `MinError` $\in \{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}

- **BaseFinalPose** (output control) .............................. pose $\sim$ `none` X/VARIANT (integer, real)
  Computed pose of base system to world coordinate system.

- **CamFinalPose** (output control) .............................. pose $\sim$ `none` X/VARIANT (integer, real)
  Computed pose of the camera relative to the coordinate system of the gripping device.

- **NumErrors** (output control) ................................. real(-array) $\sim$ VARIANT
  Error measures for each iteration.

--- Result

HandEyeCalibration returns TRUE if all parameter values are correct and the method converges with an error less than the specified minimum error (if StopCriterion = `MinError`). If necessary, an exception handling is raised.

--- Parallelization Information

HandEyeCalibration is reentrant and processed without parallelization.

--- Possible Predecessor Functions

FindMarksAndPose

--- Possible Successor Functions

WritePose, ConvertPoseType, PoseToHomMat3d, DispCaltab, SimCaltab

--- See Also

FindCaltab, FindMarksAndPose, DispCaltab, SimCaltab, WriteCamPar, ReadCamPar, CreatePose, ConvertPoseType, WritePose, ReadPose, PoseToHomMat3d, HomMat3dToPose, CaltabPoints, CreateCaltab

--- Module

```
[out] <none>X Pose HHomMat3dX.HomMat3dToPose ( )

void HOperatorSetX.HomMat3dToPose ([in] VARIANT HomMat3D,
[out] VARIANT Pose )
```

Convert homogeneous transformation matrix into 3D pose parameter.

HomMat3dToPose is used to convert a homogeneous transformation matrix into the correspondent representation of the transformation as translational vector and rotational angles.

The homogeneous 4x3 transformation matrix is given as a tuple `HomMat3D` in the following order: The first four values describe the first row of the rotation matrix and x-value of the translation vector. The next four values describe the second row of the rotation matrix and the y-value of the translation vector. These are followed by the values of the third row of the rotation matrix and the z-value of the translation vector.

The output tuple `Pose` describes the external camera parameter by the 3D translational vector (in meters), the three rotation angles and the representation type of the 3D transform. The representation type of a 3D transform generated by `HomMat3dToPose` has Type 1 with code '0'. This means, that the transform of a point is described, hereby the 3D rotation is performed before the 3D translation, that the three rotations are specified as angles (in degrees), and that the rotation order is $\gamma$-$\beta$-$\alpha$, i.e., the first rotation is around the Z-axis, the second one around the new Y-axis, and the third one around the new X-axis, which is generated after the second rotation. This corresponds to the transformation by homogenous matrices directly. This can be explained by the following holds:
\[ P_C = [x \ y \ z]^T = H \cdot P_W = R^{(1)} \cdot P_W + T^{(1)} \]

The sense of the other representation types are explained at CreatePose. With ConvertPoseType you can convert Pose to every other kind of representation type.

The subsequent program example shows how to generate an appropriate initial camera pose for CameraCalibration, if the camera pose is roughly measured in the world coordinate system. This is especially important if no planar calibration table (see CreateCaltab) is used and therefore the operator FindMarksAndPose cannot be applied. Instead of this natural or artificial landmarks with known world position can be used.

Let for example the relative camera position in the world coordinate system be the following: \( x = 1.08 \text{ m}, y = 0.25 \text{ m}, z = 0.62 \text{ m} \). This means, that the origin of the camera coordinate system in world coordinates has the value \([1.08, 0.25, 0.62]\). The camera coordinate system is oriented that way, that the line of sight of the camera is along the positive Z-axis. The orientation of the camera coordinate system is given by three rotational angles. In the program example the camera coordinate system is first rotated by 100 degrees around the Z-axis. Subsequently it is rotated by -120 degree around the new (!!!) X-axis. Note that the parameter with the rotation angle in HomMat3dRotate is negated. This is because the description of a point transformation and the description of a coordinate system transformation differs in the sign of the rotational angles. See also explanation and example at CreatePose.

---

**Parameter**

- **HomMat3D** (input control) .......... affine3d \( \rightarrow \) HHomMat3dX / VARIANT( real )
  Homogeneous transformation matrix.
  **Parameter Number** : 12

- **Pose** (output control) ................. pose \( \rightarrow \) noneX / VARIANT( integer, real )
  External camera parameters.
  **Parameter Number** : 7

---

```
// read internal camera parameters:
read_cam_par('campar.dat',CamParam)
// read 3D world points [WorldPointsX,WorldPointsY,WorldPointsZ]
// get rough camera pose from relative camera pose:
hom_mat3d_identity(HomMat3DIdent)
hom_mat3d_translate(HomMat3DIdent,-1.08,-0.25,-0.62,HomMat3DTrans)
hom_mat3d_rotate(HomMat3DTrans,-100*3.14159/180,'z',0,0,0,
                   HomMat3DRotZ)
hom_mat3d_rotate(HomMat3DRotZ,+120*3.14159/180,'x',0,0,0,
                   HomMat3DWorldStart)
// convert transformation matrix to pose (rotation and translation)
hom_mat3d_to_pose(HomMat3DWorldStart,StartPose)
// calibration of external camera params:
camera_calibration(WorldPointsX,WorldPointsY,WorldPointsZ,
                   PixelsRow,PixelsColumn,CamParam,StartPose,6,
                   Dummy,FinalPose,Errors).
```

---

**Result**

HomMat3dToPose returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

---

**Parallelization Information**

HomMat3dToPose is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

- HomMat3dRotate
- HomMat3dTranslate
- HomMat3dInvert

---

**Possible Successor Functions**

- CameraCalibration
- WritePose
- DispCaltab
- SimCaltab

---

**See Also**

- CreatePose
- CameraCalibration
- DispCaltab
- SimCaltab
- WritePose
- ReadPose

12.4. CALIBRATION 709

PoseToHomMat3d, Project3DPoint, GetLineOfSight, HomMat3dRotate, HomMat3dTranslate, HomMat3dInvert, AffineTransPoint3D

Camera calibration

Transform image points to a world plane represented by external camera parameters.

The operator ImagePointsToWorldPlane transforms image points which are given in Rows and Cols into a plane in the world coordinate system. The plane is either equal to the plane of the calibration plate or parallel to it if the operator SetOriginPose has been applied previously to correct the thickness of the calibration plate. In the first step the line of sight between the projection center and the image point is computed in the camera coordinate system, taking into account the radial distortions using the internal camera parameters CamParam. The line of sight is then transformed into the world coordinate system using the external camera parameters CamPose. By intersecting the plane (Z=0) with the line of sight the pseudo 3d coordinates X and Y are obtained. It is possible to scale the coordinates with the parameter Scale. This is particularly useful when displaying the coordinates in an image. The parameter Scale must be specified in \([\text{unit/pixel}]\). The original unit is determined by the coordinates of the calibration targets. In many cases this unit is introduced into the calibration as 'meters', i.e. one meter in the world coordinate system corresponds to one pixel in the image. In this case, it is possible to set the pixel size directly by selecting 'm', 'cm', 'mm' or 'm' for the parameter Scale.

Parameter

- **CamParam** (input control) ........................................... number ~ VARIANT (integer, real)
  Internal camera parameters.
- **CamPose** (input control) ......................................... pose ~ X / VARIANT (integer, real)
  External camera parameters.
- **Rows** (input control) ........................................... coordinates.y ~ VARIANT (integer, real)
  line indices of the points to be transformed
  Default Value: 100.0
- **Cols** (input control) ........................................... coordinates.x ~ VARIANT (integer, real)
  column indices of the points to be transformed
  Default Value: 100.0
- **Scale** (input control) .......................................... number ~ VARIANT (integer, real, string)
  Scale or dimension
  Default Value: 'm'
  Value Suggestions: Scale ∈ \{'m', 'cm', 'mm', 'microns', m, 1.0, 0.01, 0.001, 1e-6, 0.0254, 0.3048, 0.9144\}
- **X** (output control) .............................................. coordinates.x ~ VARIANT (real)
  x-coordinate in the world coordinate system.
- **Y** (output control) .............................................. coordinates.y ~ VARIANT (real)
  y-coordinate in the world coordinate system.

Result

ImagePointsToWorldPlane returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information

ImagePointsToWorldPlane is reentrant and processed without parallelization.

Possible Predecessor Functions

CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration, SetOriginPose
### See Also

**ContourToWorldPlaneXld**

Camera calibration

---

<table>
<thead>
<tr>
<th>[out] VARIANT HomMat3D</th>
<th>HPoseX.PoseToHomMat3d ([in] &lt;none&gt;X Pose )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void HOperatorSetX.PoseToHomMat3d ([in] VARIANT Pose,</td>
<td></td>
</tr>
<tr>
<td>[out] VARIANT HomMat3D )</td>
<td></td>
</tr>
</tbody>
</table>

### Convert 3d pose parameters into homogeneous transformation matrix.

**PoseToHomMat3d** is used to convert 3D pose parameters, like external camera parameter (camera pose `Pose`) into the equivalent homogeneous 4x3 transformation matrix `HomMat3D`.

This transformation matrix consists of a 3x3 rotation matrix and a translational vector. The values are given as a tuple `HomMat3D` in the following order: The first four values describe the first row of the rotation matrix and x-value of the translation vector. The next four values describe the second row of the rotation matrix and the y-value of the translation vector. These are followed by the values of the third row of the rotation matrix and the z-value of the translation vector.

The input tuple `Pose` describes 3D pose parameters which describe a 3D transform, for example the external camera parameter. These are given by the 3D translational vector (in meters), the three rotational angles and the code of the representation type of the 3D transform: E.g., the value '0' is the code of the representation type 1 and says, that the transform of a point is described, hereby the 3D rotation is performed before the 3D translation, that the three rotations are specified as angles (in degrees), and that the rotation order is $\gamma-\beta-\alpha$, i.e., the first rotation is around the Z-axis, the second one around the new Y-axis, and the third one around the new X-axis, which is generated after the second rotation. **PoseToHomMat3d** operates with all types of 3D transform. The sense of the other representation types are explained at CreatePose. The 3D transform with type 1 directly corresponds to the transformation by homogenous matrices.

This can be explained by the following holds:

$$
P_C = [x \ y \ z]^T = R^{(1)} \cdot P_W + T^{(1)} = H \cdot P_W
$$

### Parameter

- **Pose** (input control) ............................................. `Pose` ~ <none>X / VARIANT ( integer, real )
  - External camera parameters.
  - **Parameter Number**: 7
- **HomMat3D** (output control) ............................................. `affine3d` ~ VARIANT ( real )
  - Homogeneous transformation matrix.
  - **Parameter Number**: 12

### Example

```c
// read internal camera parameters
read_cam_par('campar.dat'CamParam)
// read initial camera pose
read_pose('campose.initial',StartCamPose)
// read 3D world points [WorldPointsX,WorldPointsY,WorldPointsZ]
// extract correspondent 2D image points [PixelsRow,PixelsColumn]
// calibration of external camera parameters:
camera_calibration(WorldPointsX,WorldPointsY,WorldPointsZ, PixelsRow,PixelsColumn,CamParam,StartCamPose,6, Dummy,FinalCamPose,Errors)
// transform FinalCamPose to 4x3 transformation matrix
pose_to_hom_mat3d(FinalCamPose, HomMat3D WorldCam).
```

### Result

**PoseToHomMat3d** returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.
Parallelization Information

PoseToHomMat3d is reentrant and processed without parallelization.

Possible Predecessor Functions

CameraCalibration, ReadPose

Possible Successor Functions

AffineTransPoint3D, HomMat3dInvert, HomMat3dTranslate, HomMat3dRotate, HomMat3dToPose

See Also

CreatePose, CameraCalibration, WritePose, ReadPose, HomMat3dToPose, Project3DPoint, GetLineOfSight, HomMat3dRotate, HomMat3dTranslate, HomMat3dInvert, AffineTransPoint3D

Module

Camera calibration

```c
```

```c
```

Project 3D points into (sub-)pixels.

Project3DPoint is used to project one or more 3D points (with coordinates X, Y, and Z) into the image plane (in pixels). The coordinates X, Y, and Z are given in the camera coordinate system, i.e., they describe the position of the point relative to the camera.

The internal camera parameters CamParam (Focus, Sx, Sy, Cx, Cy, Kappa (κ), ImageWidth and ImageHeight) describe the projection characteristics of the camera. The underlying camera model is a pinhole camera with radial distortions if the focal length passed in CamParam is greater than 0. It describes the transform of a 3D point \( P_C \) into a (sub-)pixel \([r,c]\) of the video image by the following equations:

\[
P_C = \begin{bmatrix} x & y & z \end{bmatrix}^T
\]

\[
u = \text{Focus} \cdot \frac{2u}{1 + \sqrt{1 - 4\kappa (u^2 + v^2)}} \quad \text{and} \quad v = \text{Focus} \cdot \frac{2v}{1 + \sqrt{1 - 4\kappa (u^2 + v^2)}}
\]

\[
c = \frac{\hat{v}}{\hat{u}} + C_x \quad \text{and} \quad r = \frac{\hat{u}}{\hat{v}} + C_y
\]

If the focal length is passed as 0 in CamParam, the camera model of a telecentric camera with radial distortions is used, i.e., it is assumed that the optics of the lens of the camera performs a parallel projection. In this case, the corresponding equations are:

\[
P_C = \begin{bmatrix} x & y & z \end{bmatrix}^T = \mathcal{R} \cdot P_W + \mathcal{T}
\]

\[
u = x \quad \text{and} \quad v = y
\]

\[
c = \frac{\hat{v}}{\hat{u}} + C_x \quad \text{bzw.} \quad r = \frac{\hat{u}}{\hat{v}} + C_y
\]

These equations consist of a coordinate transform from the world coordinate system into the camera coordinate system, a perspective or parallel projection into the image plane, a radial distortion of the projected point, and finally a sampling and an image center displacement.
Parameter

- **X** (input control) ........................................................real ↢ VARIANT ( real )
  X-coordinates of the 3D points to be projected.
- **Y** (input control) ........................................................real ↢ VARIANT ( real )
  Y-coordinates of the 3D points to be projected.
- **Z** (input control) ........................................................real ↢ VARIANT ( real )
  Z-coordinates of the 3D points to be projected.
- **CamParam** (input control) ..........................................number ↢ VARIANT ( integer, real )
  Internal camera parameters.
  **Parameter Number**: 8
- **Row** (output control) ..................................................real ↢ VARIANT ( real )
  Row-coordinates of pixels.
  **Default Value**: 'ProjectedRow'
- **Column** (output control) ..........................................real ↢ VARIANT ( real )
  Column-coordinates of pixels.
  **Default Value**: 'ProjectedCol'

Example

```c
// read camera pose
read_pose(campose.dat’,CamPose)
// transform camera pose to transformation matrix
pose_to_hom_mat3d(CamPose,HomMat3D)
// transform 3D points from source into destination coordinate system
affine_trans_point_3d([3.0,3.2],[4.5,4.5],[5.8,6.2],HomMat3D,X,Y,Z)
// read internal camera parameters
read_cam_par(’campar.dat’,CamParam)
// project 3D points into image
project_3d_point(X,Y,Z,CamParam,Row,Column).
```

Result

Project3dPoint returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information

- **ReadCamPar, AffineTransPoint3D**
  reentrant and processed without parallelization.

Possible Predecessor Functions

- GenRegionPoints, GenRegionPolygon, DispPolygon

Possible Successor Functions

- CameraCalibration, DispCaltab, ReadCamPar, GetLineOfSight, AffineTransPoint3D

See Also

Camera calibration

```
[out] VARIANT CamParam HMiscX.ReadCamPar ([in] String CamParFile )
void HOperatorSetX.ReadCamPar ([in] String CamParFile,
[out] VARIANT CamParam )
```

Read the internal camera parameters from text file.

ReadCamPar is used to read the internal camera parameters CamParam from a text file with name CamParFile.

The format of the text file is a (Halcon-independent) generic parameter description. Thus, arbitrary sets of parameters can be grouped together in a hierarchical way. The description of a single parameter within a parameter group consists of the following 3 lines:
12.4. CALIBRATION

The Halcon operator WriteCamPar expects in the file CamParFile the parameter group Camera:Parameter. This parameter group consists of the 8 parameters Focus, Sx, Sy, Cx, Cy, Kappa ($\kappa$), ImageWidth and ImageHeight. Comments are marked by a '#' at the beginning of a line. A suitable file can look like the following:

```
# INTERNAL CAMERA PARAMETERS
ParGroup: Camera: Parameter;
   "Internal CCD-camera parameters"

Focus:foc: 0.00806039;
   DOUBLE:0.0:;
   "Focal length of the lens [meter]"

Sx:sx: 1.0629e-05;
   DOUBLE:0.0:;
   "Width of a cell on the CCD-chip [meter]"

Sy:sy: 1.1e-05;
   DOUBLE:0.0:;
   "Height of a cell on the CCD-chip [meter]"

Cx:cx: 378.236;
   DOUBLE:0.0:;
   "X-coordinate of the image center [pixel]"

Cy:cy: 297.587;
   DOUBLE:0.0:;
   "Y-coordinate of the image center [pixel]"

Kappa:kappa: -2253.5;
   DOUBLE:0:;
   "Radial distortion coefficient [1/(meter*meter)]"

ImageWidth:imgw: 768;
   INT:0:2048;
   "Width of the used calibration images [pixel]"

ImageHeight:imgh: 576;
   INT:0:2048;
   "Height of the used calibration images [pixel]"
```

The underlying camera model is a **pinhole or telecentric camera with radial distortions**. A detailed description of these camera models can be found with description of WriteCamPar.

---

**Parameter**

- **CamParFile** (input control) ........................................string ~> String
  File name of internal camera parameters.
  Default Value : 'campar.dat'
  Value List : CamParFile $\in$ ['campar.dat', 'campar.initial', 'campar.final']

- **CamParam** (output control) ......................................number ~> VARIANT ( integer, real )
  Internal camera parameters.
  Parameter Number : 8

---

// get internal camera parameters:
read_cam_par('campar.dat',CamParam).

HALCON 6.0
ReadCamPar returns TRUE if all parameter values are correct and the file has been read successfully. If necessary an exception handling is raised.

Parallelization Information

ReadCamPar is reentrant and processed without parallelization.

Possible Successor Functions

FindMarksAndPose, SimCaltab, CreateCaltab, DispCaltab, CameraCalibration

See Also

FindCaltab, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab, WriteCamPar, WritePose, ReadPose, Project3DPoint, GetLineOfSight

Module

Camera calibration

```c
[out] <none>
Pose HPoseX.ReadPose ([in] String PoseFile )
```

```c
void HOperatorSetX.ReadPose ([in] String PoseFile, [out] VARIANT Pose )
```

Read 3D pose data from text file.

ReadPose is used to read the 3D pose Pose from a text file with the name PoseFile.

The output tuple Pose describes 3D pose parameters which describe a 3D transform, for example the external camera parameters. These are given by the 3D translational vector (in meters), the three rotation angles and the code of the representation type of the 3D transform: E.g., the value '0' is the code of the representation type 1 and says, that the transform of a point is described, hereby the 3D rotation is performed before the 3D translation, that the three rotations are specified as angles (in degrees), and that the rotation order is \(\gamma/\beta/\alpha\), i.e., the first rotation is around the Z-axis, the second one around the new Y-axis, and the third one around the new X-axis, which is generated after the second rotation. PoseToHomMat3d operates with all types of 3D transforms. The sense of the other representation types is explained at CreatePose.

A suitable file can be generated by the operator WritePose and looks like the following:

```text
# 3D POSE PARAMETERS: rotation and translation

# Used representation type:
  f 0

# Rotation angles [deg] or Rodriguez-vector:
  r -17.8134 1.83816 0.288092

# Translational vector (x y z [m]):
  t 0.280164 0.150644 1.7554
```

Parameter

- **PoseFile** (input control) .................................................... filename \(\sim\) String
  
  File name of the external camera parameters.

  **Default Value**: 'campose.dat'

  **Value List**: PoseFile \(\in\) {'campose.dat', 'campose.initial', 'campose.final'}

- **Pose** (output control) .................................................... pose \(\sim\) {none};X / VARIANT ( integer, real )
  
  External camera parameters.

  **Parameter Number**: 7

Example

```c
// get pose (external camera parameters):
read_pose('campose.dat',CamPose).
```
ReadPose returns TRUE if all parameter values are correct and the file has been read successfully. If necessary, an exception handling is raised.

**Parallelization Information**

ReadPose is reentrant and processed without parallelization.

**Possible Predecessor Functions**

CreatePose, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab

**Possible Successor Functions**

PoseToHomMat3d, CameraCalibration, DispCaltab, SimCaltab, WritePose, PoseToHomMat3d, HomMat3dToPose

**See Also**

CreatePose, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab, WritePose, PoseToHomMat3d, HomMat3dToPose

**Module**

Camera calibration

---

### Translate the origin of the external camera parameters.

SetOriginPose translates the origin of the external camera parameters (pose) by a vector, determined by DX, DY and DZ. This can be used, for example, to correct the obtained parameters from camera calibration (PoseIn) with respect to the thickness of the calibration plate. To achieve this, the translation vector must have the form (0,0,-D), where D is the thickness of the calibration plate. The resulting new parameters PoseNewOrigin correspond to a calibration with a calibration plate of infinitely small thickness. Therefore, it is possible to measure points in the world coordinate system (e.g., by calling ImagePointsToWorldPlane or ContourToWorldPlaneXld) directly in the requested measurement plane instead of the plane of the calibration plate. In this context it is now also possible to avoid negative coordinates in the world coordinate system by choosing suitable values for DX and DY.

---

### Parameter

- **PoseIn** (input control) original 3D transformation.
  
  **Default Value**: 0

- **DX** (input control) translation of the origin in x-direction.
  
  **Default Value**: 0

- **DY** (input control) translation of the origin in y-direction.
  
  **Default Value**: 0

- **DZ** (input control) translation of the origin in z-direction.
  
  **Default Value**: 0

- **PoseNewOrigin** (output control) new 3D description after applying the translation.
  
  **Default Value**: 0

---

### Result

SetOriginPose returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

---

**Parallelization Information**

SetOriginPose is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration
Possible Successor Functions

WritePose, PoseToHomMat3d, ImagePointsToWorldPlane, ContourToWorldPlaneXld

Camera calibration

```
```

Simulate a video image with calibration table.

SimCaltab is used to generate a simulated calibration image. The calibration table description is read from the file CalTabDescrFile and will be projected into the image plane using the given camera parameters (internal camera parameters CamParam and external camera parameters CamPose), see also Project3dPoint.

In the simulated image only the calibration table is shown. The image background is set to the gray value GrayBackground, the calibration table background is set to GrayCaltab, and the calibration marks are set to the gray value GrayMarks. The parameter ScaleFac influences the number of supporting points to approximate the elliptic contours of the calibration marks, see also DispCaltab. Increasing the number of supporting points causes a more accurate determination of the mark boundary, but increases the computation time, too. For each pixel of the simulated video image, which touches a subpixel-boundary of this kind, the gray value is set linearly between GrayMarks and GrayCaltab dependent on the proportion Inside/Outside.

By applying the operator SimCaltab you can generate synthetic calibration images (with known camera parameters!) to test the quality of the calibration algorithm (see CameraCalibration).

**Parameter**

- **SimImage** (output iconic) ...................... image ~ HImageX / HUntypedObjectX (byte ) Simulated calibration image.
- **CalTabDescrFile** (input control) .................. string ~ String File name of the calibration table description.
  
  **Default Value** : 'caltab.descr'
- **CamParam** (input control) ...................... number ~ VARIANT (integer, real ) Internal camera parameters.
  
  **Parameter Number** : 8
- **CamPose** (input control) ...................... pose ~ ;none;X / VARIANT (integer, real ) External camera parameters.
  
  **Parameter Number** : 7
- **GrayBackground** (input control) .................. integer ~ long Gray value of image background.
  
  **Default Value** : 128
  
  **Value Suggestions** : GrayBackground ∈ {0, 32, 64, 96, 128, 160}
  
  **Restriction** : \((0 \leq \text{GrayBackground} \leq 255)\)
- **GrayCaltab** (input control) ...................... integer ~ long Gray value of calibration table.
  
  **Default Value** : 224
  
  **Value Suggestions** : GrayCaltab ∈ {144, 160, 176, 192, 208, 224, 240}
  
  **Restriction** : \((0 \leq \text{GrayCaltab} \leq 255)\)
- **GrayMarks** (input control) ...................... integer ~ long Gray value of calibration marks.
  
  **Default Value** : 80
  
  **Value Suggestions** : GrayMarks ∈ {16, 32, 48, 64, 80, 96, 112}
  
  **Restriction** : \((0 \leq \text{GrayMarks} \leq 255)\)
12.4. CALIBRATION 717

**ScaleFac** (input control) ........................................................real ~ double
Scaling factor to reduce oversampling.

**Default Value:** 1.0

**Value Suggestions:** ScaleFac ∈ \{1.0, 0.5, 0.25, 0.125\}

**Recommended Value Step:** 0.05

**Restriction:** (1.0 ≥ ScaleFac)

---

Example

```c
// read calibration image
read_image(Image1,'calib-01.tiff')
// find calibration pattern
find_caltab(Image1,Caltab1,'caltab.descr',3,112,5)
// find calibration marks and initial pose
find_marks_and_pose(Image1,Caltab1,'caltab.descr',
                     [0.008,0.0,0.000011,0.000011,384,288,768,576],128,10,18,0.9,15.0,100.0,
                     RCoord1,CCoord1,StartPose1)
// read 3D positions of calibration marks
caltab_points('caltab.descr',NX,NY,NZ)
// camera calibration
camera_calibration(NX,NY,NZ,RCoord1,CCoord1,
                    [0.008,0.0,0.000011,0.000011,384,288,768,576],
                    StartPose1,11,CamParam,FinalPose,Errors)
// simulate calibration image
sim_caltab(Image1Sim,'caltab.descr',CamParam,FinalPose,128,224,80,1).
```

Result

SimCaltab returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

---

Parallelization Information

SimCaltab is processed under mutual exclusion against itself and without parallelization.

Possible Predecessor Functions

CameraCalibration,FindMarksAndPose,ReadPose,ReadCamPar,HomMat3dToPose

Possible Successor Functions

FindCaltab

---

See Also

FindCaltab,FindMarksAndPose,CameraCalibration,DispCaltab,CreatePose,
HomMat3dToPose,Project3DPoint,CreateCaltab

Module

Camera calibration

---

```c
void HMiscX.WriteCamPar ([in] VARIANT CamParam, [in] String CamParFile )
void HOperatorSetX.WriteCamPar ([in] VARIANT CamParam,
                             [in] String CamParFile )
```
The operator WriteCamPar writes the parameter group Camera:Parameter into the text file CamParFile. This parameter group consists of the 8 parameters Focus, Sx, Sy, Cx, Cy, Kappa (κ), ImageWidth and ImageHeight.

The underlying camera model is a **pinhole camera with radial distortions** if the focal length passed in CamParam is greater than 0. It describes the transform of a 3D point \( P_C \) into a (sub-)pixel \([r,c]\) of the video image by the following equations:

\[
P_C = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^T
\]

\[
u = \text{Focus} \cdot \frac{x}{z} \quad \text{and} \quad v = \text{Focus} \cdot \frac{y}{z}
\]

\[
\tilde{u} = \frac{2\nu}{1 + \sqrt{1 - 4\nu^2(u^2 + v^2)}} \quad \text{and} \quad \tilde{v} = \frac{2v}{1 + \sqrt{1 - 4v^2(u^2 + v^2)}}
\]

\[
c = \frac{\tilde{u}}{S_x} + C_x \quad \text{and} \quad r = \frac{\tilde{v}}{S_y} + C_y
\]

If the focal length is passed as 0 in CamParam, the camera model of a **telecentric camera with radial distortions** is used, i.e., it is assumed that the optics of the lens of the camera performs a parallel projection. In this case, the corresponding equations are:

\[
P_C = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^T = \mathcal{R} \cdot P_W + \mathcal{T}
\]

\[
u = x \quad \text{and} \quad v = y
\]

\[
\tilde{u} = \frac{2\nu}{1 + \sqrt{1 - 4\nu^2(u^2 + v^2)}} \quad \text{bzw.} \quad \tilde{v} = \frac{2v}{1 + \sqrt{1 - 4v^2(u^2 + v^2)}}
\]

\[
c = \frac{\tilde{u}}{S_x} + C_x \quad \text{bzw.} \quad r = \frac{\tilde{v}}{S_y} + C_y
\]

These equations consist of a coordinate transform from the world coordinate system into the camera coordinate system, a perspective or parallel projection into the image plane, a radial distortion of the projected point, and finally a sampling and an image center displacement.

The parameter \( \mathcal{R} \) and \( \mathcal{T} \) describe the 3D pose of the camera coordinate system in the world coordinate system. They are called **external camera parameters**, see also HomMat3dToPose and CreatePose.

The **internal camera parameters** consist of the focal length (Focus), the radial distortion coefficient (κ), the scale factor \( S_x \) and \( S_y \) (horizontal and vertical distance between cells on the CCD chip), the image center \([C_x, C_y]^T\), and the used image width and height. The internal camera parameters are - in contrast to the external parameters - independent of the actual pose of the CCD camera. They describe the projection process of the used combination of camera, lens, and frame grabber. The determination of these parameters is done by the camera calibration, see CameraCalibration.

---

**Parameter**

- **CamParam** (input control) ........................................... number \( \sim \) VARIANT ( integer, real )
  Internal camera parameters.
  **Parameter Number**: 8

- **CamParFile** (input control) ........................................... string \( \sim \) String
  File name of internal camera parameters.
  **Default Value**: ‘campar.dat’
  **Value List**: CamParFile \( \in \{ \text{‘campar.dat’}, \text{‘campar.initial’}, \text{‘campar.final’} \} \)

---

// read calibration images
read_image(Image1,’calib-01.tiff’)
read_image(Image2,’calib-02.tiff’)
read_image(Image3,’calib-03.tiff’)
// find calibration pattern
find_caltab(Image1,Caltab1,’caltab.descr’,3,112,5)
find_caltab(Image2,Caltab2,’caltab.descr’,3,112,5)
find_caltab(Image3,Caltab,’caltab.descr’,3,112,5)
// find calibration marks and start poses
find_marks_and_pose(Image1,Caltab1,’caltab.descr’,[0.008,0.0,0.0]...
0.000011, 0.000011, 384, 288, 768, 576], 128, 10, 18, 0.9, 15.0, 100.0, RCoord1, CCoord1, StartPose1;
f
0.000011, 0.000011, 384, 288, 768, 576], 128, 10, 18, 0.9, 15.0, 100.0, RCoord2, CCoord2, StartPose2;
f
0.000011, 0.000011, 384, 288, 768, 576], 128, 10, 18, 0.9, 15.0, 100.0, RCoord3, CCoord3, StartPose3;
// read 3D positions of calibration marks
caltab_points('caltab.descr', NX, NY, NZ)
// camera calibration
camera_calibration(NX, NY, NZ, [RCoord1, RCoord2, RCoord3],
[CCoord1, CCoord2, CCoord3], [0.008, 0.0, 0.000011, 0.000011, 384, 288, 768, 576],
[StartPose1, StartPose2, StartPose3], 11, CamParam, NFinalPose, Errors)
// write internal camera parameters to file
write_cam_par(CamParam, ‘campar.dat’).

WriteCamPar returns TRUE if all parameter values are correct and the file has been written successfully. If necessary an exception handling is raised.

Parallelization Information
WriteCamPar is local and processed completely exclusively without parallelization.

Possible Predecessor Functions
CameraCalibration

See Also
FindCaltab, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab, ReadCamPar, WritePose, ReadPose, Project3DPoint, GetLineOfSight

Module
Camera calibration

void HPoseX.WritePose ([in] <none>X Pose, [in] String PoseFile )
void HOperatorSetX.WritePose ([in] VARIANT Pose, [in] String PoseFile )

Write 3D pose data to text file.
WritePose is used to write 3D pose data Pose into a text file with the name PoseFile.
The input tuple Pose describes 3D pose parameters which describe a 3D transform, for example the external camera parameters. These are given by the 3D translational vector (in meters), the three rotation angles and the code of the representation type of the 3D transform; E.g., the value ‘0’ is the code of the representation type 1 and says, that the transform of a point is described, hereby the 3D rotation is performed before the 3D translation, that the three rotations are specified as angles (in degrees), and that the rotation order is \( \gamma - \beta - \alpha \), i.e., the first rotation is around the Z-axis, the second one around the new Y-axis, and the third one around the new X-axis, which is generated after the second rotation. PoseToHomMat3d operates with all types of 3D transform. The sense of the other representation types are explained at CreatePose.

A file generated by WritePose looks like the following:

# 3D POSE PARAMETERS: rotation and translation

# Used representation type:
  f 0

# Rotation angles [deg] or Rodriguez-vector:
  r -17.8134 1.83816 0.288092

# Translational vector \( x \ y \ z \ [m] \):
  t 0.280164 0.150644 1.7554
Parameter

<table>
<thead>
<tr>
<th>Pose (input control)</th>
<th>pose ~ none; X / VARIANT ( integer, real )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PoseFile (input control)</td>
<td>filename ~ String</td>
</tr>
</tbody>
</table>

External camera parameters.

Parameter Number : 7

File name of the external camera parameters.

Default Value : 'campose.dat'

Value List: PoseFile ∈ { 'campose.dat', 'campose.initial', 'campose.final' }

Example

```c++
// read calibration images
read_image(Image1,'calib-01.tiff')
read_image(Image2,'calib-02.tiff')
read_image(Image3,'calib-03.tiff')

// find calibration pattern
find_caltab(Image1,Caltab1,'caltab.descr',3,112,5)
find_caltab(Image2,Caltab2,'caltab.descr',3,112,5)
find_caltab(Image3,Caltab3,'caltab.descr',3,112,5)

// find calibration marks and start poses */
find_marks_and_pose(Image1,Caltab1,'caltab.descr',[0.008,0.0,0.000011,0.000011,384,288,768,576],128,10,18,0.9,15.0,100.0,RCoord1,CCoord1,StartPose1)
find_marks_and_pose(Image2,Caltab2,'caltab.descr',[0.008,0.0,0.000011,0.000011,384,288,768,576],128,10,18,0.9,15.0,100.0,RCoord2,CCoord2,StartPose2)
find_marks_and_pose(Image3,Caltab3,'caltab.descr',[0.008,0.0,0.000011,0.000011,384,288,768,576],128,10,18,0.9,15.0,100.0,RCoord3,CCoord3,StartPose3)

// read 3D positions of calibration marks
caltab_points('caltab.descr',NX,NY,NZ)

// camera calibration
camera_calibration(NX,NY,NZ,[RCoord1,RCoord2,RCoord3],
[CCoord1,CCoord2,CCoord3],[0.008,0.0,0.0,0.0,0.000011,0.000011,384,288,768,576],
[StartPose1,StartPose2,StartPose3],11,CamParam,NFinalPose,Errors)

// write external camera parameters of first calibration image
write_pose(NFinalPose[0],NFinalPose[1],NFinalPose[2],NFinalPose[3],NFinalPose[4],NFinalPose[5],NFinalPose[6],'campose.dat').
```

Result

WritePose returns TRUE if all parameter values are correct and the file has been written successfully. If necessary an exception handling is raised.

Parallelization Information

WritePose is local and processed completely exclusively without parallelization.

Possible Predecessor Functions

CameraCalibration, HomMat3dToPose

See Also

CreatePose, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab, ReadPose, PoseToHomMat3d, HomMat3dToPose

Module

Camera calibration
12.5 Fourier-Descriptor

**Normalizing of the Fourier coefficients with respect to the displacement of the starting point.**

The operator `AbsInvarFourierCoeff` normalizes the Fourier coefficients with regard to the displacements of the starting point. These occur when an object is rotated. The contour tracer `GetRegionContour` starts with recording the contour in the upper lefthand corner of the region and follows the contour clockwise. If the object is rotated, the starting value for the contour point chain is different which leads to a phase shift in the frequency space. The following two kinds of normalizing are available:

- **abs_amount**: The phase information will be eliminated; the normalizing does not retain the structure, i.e. if the AZ-invariants are backtransformed, no similarity with the pattern can be recognized anymore.
- **az_invar1**: AZ-invariants of the 1st order execute the normalizing with respect to displacing the starting point so that the structure is retained; they are however more prone to local and global disturbances, in particular to projective distortions.

---

**Parameter**

- **RealInvar** (input control) ............................................ real \(\sim\) VARIANT( real )
  - Real parts of the normalized Fourier coefficients.
- **ImaginaryInvar** (input control) ................................. real \(\sim\) VARIANT( real )
  - Imaginary parts of the normalized Fourier coefficients.
- **CoeffP** (input control) ............................................. integer \(\sim\) long
  - Normalizing coefficients p.
  - Default Value : 1
  - Value Suggestions : CoefP \(\in\) \{1, 2\}
  - Restriction : \((\text{CoefP} \geq 1)\)
- **CoeffQ** (input control) ............................................. integer \(\sim\) long
  - Normalizing coefficients q.
  - Default Value : 1
  - Value Suggestions : CoefQ \(\in\) \{1, 2\}
  - Restriction : \((\text{CoefQ} \geq 1) \land (\text{CoefQ} \neq \text{CoefP})\)
- **AZInvar** (input control) ......................................... string \(\sim\) String
  - Order of the AZ-invariants.
  - Default Value : ‘abs_amount’
  - Value List : AZInvar \(\in\) \{'abs_amount’, ‘az_invar1’\}
- **RealAbsInvar** (output control) .................................. real \(\sim\) VARIANT( real )
  - Real parts of the normalized Fourier coefficients.
- **ImaginaryAbsInvar** (output control) ......................... real \(\sim\) VARIANT( real )
  - Imaginary parts of the normalized Fourier coefficients.

---

**Parallelization Information**

`AbsInvarFourierCoeff` is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

- `InvarFourierCoeff`

**Possible Successor Functions**

- `Fourier1DimInv`, `MatchFourierCoeff`

---

**Module**

Fourier descriptors
Calculate the Fourier coefficients of a parameterized contour.

The operator `Fourier1Dim` calculates the Fourier coefficients of a parameterized contour by using a valid parameter scale. This parameter scale may, for instance, be created with the help of the procedure `PrepContourFourier`. This function serves to calculate the Fourier coefficients of closed contours which are treated like complex-valued curves. Therefore, in order to determine the Fourier coefficients, the Fourier transform for periodical functions is used. Hereby the parameter `MaxCoef` determines the absolute value +1 of the maximal number of Fourier coefficients, i.e. if `n` coefficients are indicated, the procedure will calculate coefficients ranging from `−n` to `n`. The contour will be approximated without loss, if `n = number of the contour points`, whereby `n = 100` approximates the contour so well that an error can hardly be distinguished; `n ∈ [40, 50]` however is sufficient for most applications. If the parameter `MaxCoef` is set to 0, all coefficients will be determined.

**Parameter**

- **Rows** (input control) .......................................................... `contour.y` ⊆ `VARIANT (integer)`
  Row coordinates of the contour.
- **Columns** (input control) .................................................. `contour.x` ⊆ `VARIANT (integer)`
  Column coordinates of the contour.
- **ParContour** (input control) ............................................... `real` ⊆ `VARIANT (real)`
  Parameter scale.
- **MaxCoef** (input control) .................................................... `integer` ⊆ `long`
  Desired number of Fourier coefficients or all of them (0).
  Default Value : 50
  Value Suggestions : `MaxCoef ∈ {0, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 400}`
  Restriction : `MaxCoef ≥ 0`  
- **RealCoef** (output control) ............................................... `real` ⊆ `VARIANT (real)`
  Real parts of the Fourier coefficients.
- **ImaginaryCoef** (output control) ....................................... `real` ⊆ `VARIANT (real)`
  Imaginary parts of the Fourier coefficients.

**Parallelization Information**

`Fourier1Dim` is reentrant and processed without parallelization.

**Possible Predecessor Functions**

`PrepContourFourier`

**Possible Successor Functions**

`InvarFourierCoeff, DispPolygon`

**Module**

Fourier descriptors

One dimensional Fourier synthesis (inverse Fourier transform).

Backtransformation of Fourier coefficients respectively of Fourier descriptors. The number of values to be backtransformed should not exceed the length of the transformed contour.
12.5. FOURIER-DESCRIPTOR

Parameter

▷ RealCoef (input control) ..................................................real ~ VARIANT (real)
Real parts.

▷ ImaginaryCoef (input control) ...........................................real ~ VARIANT (real)
Imaginary parts.

▷ MaxCoef (input control) ..................................................integer ~ long
Input of the steps for the backtransformation.
Default Value : 100
Value Suggestions : MaxCoef Є {5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 400}
Restriction : (MaxCoef Є 1)

▷ Rows (output control) ..................................................contour.y ~ VARIANT (real)
Row coordinates.

▷ Columns (output control) ..............................................contour.x ~ VARIANT (real)
Column coordinates.

Parallelization Information
Fourier1DimInv is reentrant and processed without parallelization.

Possible Predecessor Functions
InvarFourierCoeff, Fourier1Dim

Possible Successor Functions
DispPolygon

Module

Fourier descriptors

 Normalize the Fourier coefficients.
Elimination of affine information from the Fourier coefficients, determination of affine invariants. The Fourier
coefficients will be normalized suitably so that all affine correlated contours will be projected to one and the same
contour. The following levels of affine mappings are available:

1. Translations (InvarType = 'transInvar')
2. + Rotations (InvarType = 'congrInvar')
3. + Scalings (InvarType = 'similInvar')
4. + Slanting (InvarType = 'affineInvar')

The control parameter InvarType indicates up to which level the affine representation shall be normalized.
Please note that indicating a certain level implies that the normalizing is executed with regard to all levels below.
For most applications a subsequent normalizing of the starting point is recommended!

Parameter

▷ RealCoef (input control) ..................................................real ~ VARIANT (real)
Real parts of the Fourier coefficients.

▷ ImaginaryCoef (input control) ...........................................real ~ VARIANT (real)
Imaginary parts of the Fourier coefficients.

▷ NormPar (input control) ..................................................integer ~ long
Input of the normalizing coefficients.
Default Value : 1
Value Suggestions : NormPar Є {1, 2}
Restriction : (NormPar Є 1)
**InvarType** (input control)  
Indicates the level of the affine mappings. 

**Default Value**: 'affine_invar' 
**Value List**: InvarType ∈ {'affine_invar', 'simil_invar', 'congr_invar', 'transl_invar'}

**RealInvar** (output control)  
Real parts of the normalized Fourier coefficients.

**ImaginaryInvar** (output control)  
Imaginary parts of the normalized Fourier coefficients.

---

**Parallelization Information**

InvarFourierCoeff is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

Fourier1Dim

---

**Possible Successor Functions**

InvarFourierCoeff

---

**Module**

Fourier descriptors

---

```plaintext
[out] double Distance HMiscX.MatchFourierCoeff ([in] VARIANT RealCoef1,
[in] VARIANT ImaginaryCoef1, [in] VARIANT RealCoef2,

void HOperatorSetX.MatchFourierCoeff ([in] VARIANT RealCoef1,
[in] VARIANT ImaginaryCoef1, [in] VARIANT RealCoef2,
[in] VARIANT ImaginaryCoef2, [in] long MaxCoef, [in] String Damping,
[out] double Distance )
```

**Similarity of two contours.**

The operator MatchFourierCoeff calculates the Euclidean distance between two contours which are available as Fourier coefficients. In order to avoid that the higher frequencies are in some way too dominant, the following attenuation can be used:

**none**: No attenuation.

**1/index**: Absolute amounts of the Fourier coefficients will be divided by their index.

**1/(index*index)**: Absolute amounts of the Fourier coefficients will be divided by their square index.

The higher the result value, the greater the differences between the pattern and the test contour. If the number of coefficients is not the same, only the first \(n\) coefficients will be compared. The parameter MaxCoef indicates the number of the coefficients to be compared. If MaxCoef is set to zero, all coefficients will be used.

---

**Parameter**

**RealCoef1** (input control)  
Real parts of the pattern Fourier coefficients.

**ImaginaryCoef1** (input control)  
Imaginary parts of the pattern Fourier coefficients.

**RealCoef2** (input control)  
Real parts of the Fourier coefficients to be compared.

**ImaginaryCoef2** (input control)  
Imaginary parts of the Fourier coefficients to be compared.

**MaxCoef** (input control)  
Total number of Fourier coefficients. 

**Default Value**: 50 
**Value Suggestions**: \(\text{MaxCoef} \in \{0, 5, 10, 15, 20, 30, 40, 50, 70, 100, 200, 400\}\) 
**Restriction**: \(\text{MaxCoef} \geq 0\)
Damping (input control) ........................................... string $\rightarrow$ String
Kind of attenuation.
Default Value : '1/index'
Value Suggestions: Damping $\in \{ 'none', '1/index', '1/(index*index)' \}$

Distance (output control) ........................................... real $\rightarrow$ double
Similarity of the contours.

Parallelization Information
MatchFourierCoeff is reentrant and processed without parallelization.

Possible Predecessor Functions
Fourier descriptors

Parameter

Rows (input control) ........................................... contour.y $\rightarrow$ VARIANT ( integer )
Row coordinates of the contour.

Columns (input control) ........................................... contour.x $\rightarrow$ VARIANT ( integer )
Column coordinates of the contour.

RowsMoved (output control) ........................................... contour.y $\rightarrow$ VARIANT ( integer )
Row coordinates of the displaced contour.

ColumnsMoved (output control) ........................................... contour.x $\rightarrow$ VARIANT ( integer )
Column coordinates of the displaced contour.

Parallelization Information
MoveContourOrig is processed completely exclusively without parallelization.

Possible Predecessor Functions
Fourier descriptors

Parameterize the passed contour.
The operator PrepContourFourier parameterizes the transmitted contour in order to prepare it for the one dimensional Fourier transformation. Hereby the contour must be available in closed form. Three parameter functions are available for the control parameter TransMode:

arc: Parameterization by the radian.
signed_area: Parameterization by the signed area.
**unsigned_area**: Parameterization by the absolute area.

Please note that in contrast to the signed or unsigned area the affine mapping of the radian will not be transformed linearly.

```
Parameter

Rows (input control) .................................................. contour.y ~ VARIANT( integer )
Row indices of the contour.

Columns (input control) ............................................. contour.x ~ VARIANT( integer )
Column indices of the contour.

TransMode (input control) ............................................. string ~ String
Kind of parameterization.

Default Value: 'signed_area'
Value Suggestions: TransMode ∈ { 'arc', 'unsigned_area', 'signed_area' }

ParContour (output control) ......................................... real ~ VARIANT( real )
Parameterized contour.
```

---

**Parallelization Information**

PrepContourFourier is reentrant and processed without parallelization.

**Possible Predecessor Functions**

MoveContourOrig

**Possible Successor Functions**

Fourier1Dim

---

**Module**

Fourier descriptors

---

### 12.6 Function

```c
[out] HFunction1dX FunctionAbsolute HFunction1dX.AbsFunct1D (
)

void HOperatorSetX.AbsFunct1D ([in] VARIANT Function,
[out] VARIANT FunctionAbsolute )
```

Absolute value of the y values.

AbsFunct1d calculates the absolute values of all y values of Function.

```
Parameter

Function (input control) ............... function._Id ~ HFunction1dX / VARIANT( real, integer )
Input function.

FunctionAbsolute (output control) ...... function._Id ~ HFunction1dX / VARIANT( real, integer )
Function with the absolute values of the y values.
```

---

**Parallelization Information**

AbsFunct1D is reentrant and processed without parallelization.

**Possible Predecessor Functions**

CreateFunct1DPairs, CreateFunct1DArray

---

**Module**

Tools

```c
void HFunction1dX.CreateFunct1DArray ([in] VARIANT YValues )

void HOperatorSetX.CreateFunct1DArray ([in] VARIANT YValues,
[out] VARIANT Function )
```

Create a function from a sequence of y-values.
CreateFunct1dArray creates a one-dimensional function from a set of y-values YValues. The resulting function can then be processed and analyzed with the operators for 1d functions. YValues is interpreted as follows: the first value of YValues is the function value at zero, the second value is the function value at one, etc. Thus, the values define a function at equidistant x values (with distance 1), starting at 0.

Alternatively, the operator CreateFunct1dPairs can be used to create a function. CreateFunct1dPairs also allows to define a function with non-equidistant x values by specifying them explicitly. Thus to get the same definition as with CreateFunct1dArray, one would pass a tuple of x values to CreateFunct1dPairs that has the same length as YValues and contains values starting at 0 and increasing by 1 in each position. Note, however, that CreateFunct1dPairs leads to a different internal representation of the function which needs more storage (because all (x,y) pairs are stored) and sometimes cannot be processed as efficiently as functions created by CreateFunct1dArray.

Parameter

- **YValues** (input control) ........................................... number(-array) → VARIANT ( real, integer )
  
  X value for function points.

- **Function** (output control) ............................... function_id → HFunction1dX / VARIANT ( real, integer )
  
  Created function.

Parallelization Information

CreateFunct1dArray is reentrant and processed without parallelization.

Possible Successor Functions

WriteFunct1D, GnuplotPlotFunct1D, YRangeFunct1D, GetPairFunct1D, TransformFunct1D

See Also

CreateFunct1dPairs, ReadFunct1D

Alternatives

CreateFunct1dPairs, ReadFunct1D

Module

Tools

```c
void HFunction1dX.CreateFunct1dPairs ([in] VARIANT XValues,
[in] VARIANT YValues )

void HOperatorSetX.CreateFunct1dPairs ([in] VARIANT XValues,
[in] VARIANT YValues, [out] VARIANT Function )
```

Create a function from a set of (x,y) pairs.

CreateFunct1dPairs creates a one-dimensional function from a set of pairs of (x,y) values. The XValues of the functions have to be passed in ascending order. The resulting function can then be processed and analyzed with the operators for 1d functions.

Alternatively, functions can be created with the operator CreateFunct1dArray. In contrast to this operator, x values with arbitrary positions can be specified with CreateFunct1dPairs. Hence, it is the more general operator. It should be noted, however, that because of this generality the processing of a function created with CreateFunct1dPairs cannot be carried out as efficiently as for equidistant functions. In particular, not all operators accept such functions. If necessary, a function can be transformed into an equidistant function with the operator SampleFunct1d.

Parameter

- **XValues** (input control) ........................................... number(-array) → VARIANT ( real, integer )
  
  X value for function points.

- **YValues** (input control) ........................................... number(-array) → VARIANT ( real, integer )
  
  Y-value for function points.

- **Function** (output control) ............................... function_id → HFunction1dX / VARIANT ( real, integer )
  
  Created function.

Parallelization Information

CreateFunct1dPairs is reentrant and processed without parallelization.
Possible Successor Functions

WriteFunct1D, GnuPlotPlotFunct1D, YRangeFunct1D, GetPairFunct1D

See Also

Funct1DToPairs

Alternatives

CreateFunct1DArray, ReadFunct1D

Tools

\[
\text{void HOperatorSetX.DistanceFunct1D (}
\begin{array}{l}
\text{[in] VARIANT Function1}, \\
\quad \text{[in] VARIANT Function2}, \\
\quad \text{[in] VARIANT Mode}, \\
\quad \text{[in] VARIANT Sigma}, \\
\quad \text{[out] VARIANT Distance}
\end{array}
\)
\]

Compute the distance of two functions.
DistanceFunct1D calculates the distance of two functions. The two functions may differ in length.

Parameter

- **Function1** (input control) .................. function_1d \sim HFunction1dX / VARIANT ( real, integer )
  Input function 1.
- **Function2** (input control) .................. function_1d \sim HFunction1dX / VARIANT ( real, integer )
  Input function 2.
- **Mode** (input control) .............................. string(-array) \sim VARIANT ( string )
  Modes of invariants.
  Default Value : ‘length’
  Value List : Mode \in \{ ‘length’, ‘mean’ \}
- **Sigma** (input control) .............................. number(-array) \sim VARIANT ( real )
  Variance of the optional smoothing with a Gaussian filter.
  Default Value : 0.0
  Value Suggestions : Sigma \in \{ 0.0, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0, 15.0, 20.0, 25.0, 30.0, 40.0, 50.0 \}
- **Distance** (output control) ....................... real \sim VARIANT ( real, integer )
  Distance of the functions.

Parallelization Information

DistanceFunct1D is reentrant and processed without parallelization.

Module

Tools

\[
\text{void HOperatorSetX.Funct1DToPairs (}
\begin{array}{l}
\text{[in] VARIANT Function}, \\
\quad \text{[out] VARIANT XValues}, \\
\quad \text{[out] VARIANT YValues}
\end{array}
\)
\]

Access to the x/y values of a function.
Funct1DToPairs splits the input function Function into tuples for the x and y values.

Parameter

- **Function** (input control) ..................... function_1d \sim HFunction1dX / VARIANT ( real, integer )
  Input function.
- **XValues** (output control) ...................... number \sim VARIANT ( integer, real )
  X values of the function.
Access a function value using the index of the control points.

GetPairFunct1d accesses a function value of Function. This is done by specifying the index of one or more control points of the function.

Parameter

- **Function** (input control) .................. function_1d  \(\sim\) *HFunction1dX/VARIANT* (real, integer) 
  Input function.
- **Index** (input control) ...................... integer(-array)  \(\sim\) *VARIANT* (integer) 
  Index of the control points.
- **X** (output control) .......................... number(-array)  \(\sim\) *VARIANT* (real) 
  X value at the given control points.
- **Y** (output control) .......................... number(-array)  \(\sim\) *VARIANT* (real) 
  Y value at the given control points.

Parallelization Information
GetPairFunct1d is reentrant and processed without parallelization.

Possible Predecessor Functions
CreateFunct1DPairs, CreateFunct1DArray

Tools

```cpp
[out] VARIANT X *HFunction1dX.GetPairFunct1D* ([in] VARIANT Index,
[out] VARIANT Y )
```

```cpp
void *HOperatorSetX.GetPairFunct1D* ([in] VARIANT Function,
```

Compute the positive and negative areas of a function.

IntegrateFunct1d integrates the function Function (see CreateFunct1dArray and CreateFunct1dpairs) and returns the integral of the positive and negative parts of the function in Positive and Negative, respectively. Hence, the integral of the function is the difference Positive-Negative. The integration is done on the interval on which the function is defined. For the integration, the function is interpolated linearly.

Parameter

- **Function** (input control) .................. function_1d  \(\sim\) *HFunction1dX/VARIANT* (real) 
  Input function.
- **Positive** (output control) ................ number  \(\sim\) *double* 
  Area under the positive part of the function.
- **Negative** (output control) ............... number 

HALCON 6.0
Negative (output control) .............................................. number ~ VARIANT ( real )

Area under the negative part of the function.

Parallelization Information
IntegrateFunct1D is reentrant and processed without parallelization.

Possible Predecessor Functions
CreateFunct1DPairs, CreateFunct1DArray

See Also
CreateFunct1DArray, CreateFunct1DPairs

Module

Tools

[out] VARIANT Params HFunction1dx.MatchFunct1dTrans
([in] HFunction1dx Function2, [in] String Border, [in] VARIANT ParamsConst,
[in] VARIANT UseParams, [out] double ChiSquare, [out] VARIANT Covar )

void HOperatorSetX.MatchFunct1dTrans ([in] VARIANT Function1,
[in] VARIANT UseParams, [out] VARIANT Params, [out] double ChiSquare,
[out] VARIANT Covar )

Calculate transformation parameters between two functions.

MatchFunct1dTrans calculates the transformation parameters between two functions given as the tuples Function1 and Function2 (see CreateFunct1dArray and CreateFunct1dPairs). The following model is used for the transformation between the two functions:

\[ y_1(x) = a_1 y_2(a_3 x + a_4) + a_2 . \]

The transformation parameters are determined by a least-squares minimization of the following function:

\[ \sum_{i=0}^{n-1} (y_1(x_i) - a_1 y_2(a_3 x_i + a_4) + a_2)^2 . \]

The values of the function \( y_2 \) are obtained by linear interpolation. The parameter Border determines the values of the function Function2 outside of its domain. For Border='zero' these values are set to 0, for Border='constant' they are set to the corresponding value at the border, for Border='mirror' they are mirrored at the border, and for Border='cyclic' they are continued cyclically. The calculated transformation parameters are returned as a 4-tuple in Params. If some of the parameter values are known, the respective parameters can be excluded from the least-squares adjustment by setting the corresponding value in the tuple UseParams to the value 'false'. In this case, the tuple ParamsConst must contain the known value of the respective parameter. If a parameter is used for the adjustment (UseParams = 'true'), the corresponding parameter in ParamsConst is ignored. On output, MatchFunct1dTrans additionally returns the sum of the squared errors ChiSquare of the resulting function, i.e., the function obtained by transforming the input function with the transformation parameters, as well as the covariance matrix Covar of the transformation parameters Params. These parameters can be used to decide whether a successful matching of the functions was possible.

Parameter

- **Function1** (input control) ......................... function_1d ~ HFunction1dx / VARIANT ( real, integer )
  Function 1.

- **Function2** (input control) ......................... function_1d ~ HFunction1dx / VARIANT ( real, integer )
  Function 2.

- **Border** (input control) ............................................ string ~ String
  Border treatment for function 2.

  **Default Value**: 'constant'

  **Value List**: Border ∈ {'zero', 'constant', 'mirror', 'cyclic'}
**12.6. FUNCTION**

- **ParamsConst** (input control) ........................................... number \( \sim \) VARIANT ( real )
  Values of the parameters to remain constant.
  Default Value : [1.0, 0.0, 1.0, 0.0]
  Parameter Number : 4

- **UseParams** (input control) ........................................... string \( \sim \) VARIANT ( string )
  Should a parameter be adapted for it?
  Default Value : ['true', 'true', 'true', 'true']
  Value List : UseParams \( \in \) { 'true', 'false' }
  Parameter Number : 4

- **Params** (output control) ........................................... number \( \sim \) VARIANT ( real )
  Transformation parameters between the functions.
  Parameter Number : 4

- **ChiSquare** (output control) ........................................... number \( \sim \) double
  Quadratic error of the output function.

- **Covar** (output control) ........................................... number \( \sim \) VARIANT ( real )
  Covariance Matrix of the transformation parameters.
  Parameter Number : 16

---

**Parallelization Information**

MatchFunct1DTrans is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

CreateFunct1DArray, CreateFunct1DPairs

---

**See Also**

GrayProjections

---

**Module**

---

**Tools**

```c
[out] HFunction1dX FunctionInverted HFunction1dX.NegateFunct1D ()
void HOperatorSetX.NegateFunct1D ([in] VARIANT Function,
[out] VARIANT FunctionInverted )
```

Negation of the y values.

NegateFunct1d negates all y values of Function.

---

**Parameter**

- **Function** (input control) ................. function_id \( \sim \) HFunction1dX / VARIANT ( real, integer )
  Input function.

- **FunctionInverted** (output control) ....... function_id \( \sim \) HFunction1dX / VARIANT ( real, integer )
  Function with the negated y values.

---

**Parallelization Information**

NegateFunct1D is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

CreateFunct1DPairs, CreateFunct1DArray

---

**Module**

---

```c
[out] long Length HFunction1dX.NumPointsFunct1D ()
void HOperatorSetX.NumPointsFunct1D ([in] VARIANT Function,
[out] long Length )
```

Number of control points of the function.

NumPointsFunct1d calculates the number of control points of Function.
**Parameter**

- **Function** (input control) 
  
  function
  
  Input function.

- **Length** (output control) 
  
  Number of control points.

**Parallelization Information**

NumPointsFunct1D is *reentrant* and processed *without* parallelization.

**Possible Predecessor Functions**

CreateFunct1DPairs, CreateFunct1DArray

**Module**

Tools

```c
void HFunction1dX.ReadFunct1D ([in] String FileName )
void HOperatorSetX.ReadFunct1D ([in] String FileName, [out] VARIANT Function )
```

Read a function from a file.

The operator ReadFunct1d reads the contents of FileName and converts it into the function Function. The file has be generated by WriteFunct1d.

**Parameter**

- **FileName** (input control) 
  
  Name of the file to be read.

- **Function** (output control) 
  
  Function from the file.

**Result**

If the parameters are correct the operator ReadFunct1d returns the value TRUE. If the file could not be opened ReadFunct1d returns FAIL. Otherwise an exception handling is raised.

**Parallelization Information**

ReadFunct1D is *reentrant* and processed *without* parallelization.

**See Also**

WriteFunct1D, GnuplotPlotCtrl, WriteImage, WriteRegion, OpenFile

**Alternatives**

FreadString, ReadTuple

**Module**

Basic operators

```c
[out] HFunction1dX SampledFunction HFunction1dX.SampleFunct1D

```

Sample a function equidistantly in an interval.

SampleFunct1d samples the input function Function in the interval \([XMin, XMax]\) at equidistant points with the distance XDist. The last point lies in the interval if \(XMax-XMin\) is not an integer multiple of \(XDist\). To obtain the samples, the input function is interpolated linearly. The parameter Border determines the values of the function Function outside of its domain. For Border=’zero’ these values are set to 0, for Border=’constant’ they are set to the corresponding value at the border, for Border=’mirror’ they are mirrored at the border, and for Border=’cyclic’ they are continued cyclically.
12.6. FUNCTION

**Parameter**

- **Function** (input control) ........................... `function_id \mapsto HFunction1dX / VARIANT ( real, integer )`
  Input function.
- **XMin** (input control) ............................... `number \mapsto VARIANT ( real, integer )`
  Minimum x value of the output function.
- **XMax** (input control) ............................... `number \mapsto VARIANT ( real, integer )`
  Maximum x value of the output function.
  **Restriction**: \((XMax > XMin)\)
- **XDist** (input control) .............................. `number \mapsto VARIANT ( real, integer )`
  Distance of the samples.
  **Restriction**: \((XDist > 0)\)
- **Border** (input control) ................................. `string \mapsto String`
  Border treatment for the input function.
  **Default Value**: ‘constant’
  **Value List**: Border ∈ \{"zero", ‘constant’, ‘mirror’, ‘cyclic’\}
- **SampledFunction** (output control) .................... `function_id \mapsto HFunction1dX / VARIANT ( real )`
  Sampled function.

**Parallelization Information**

SampleFunct1D is reentrant and processed without parallelization.

**Possible Predecessor Functions**

TransformFunct1D, CreateFunct1DArray, CreateFunct1DPairs

**Module**

Tools

```
[out] HFunction1dX FunctionScaled HFunction1dX.ScaleYFunct1D
([in] double Mult, [in] double Add )
```

```
void HOperatorSetX.ScaleYFunct1D ([in] VARIANT Function,
```

**Multiplication and addition of the y values.**

ScaleYFunct1D multiplies and adds the y values of Function with the parameters Mult and Add.

**Parameter**

- **Function** (input control) ........................... `function_id \mapsto HFunction1dX / VARIANT ( real, integer )`
  Input function.
- **Mult** (input control) .................................. `number \mapsto double`
  Factor for scaling of the y values.
  **Default Value**: 2
  **Value Suggestions**: Mult ∈ \{0.1, 0.3, 0.5, 1, 2, 5, 10\}
- **Add** (input control) .................................... `number \mapsto double`
  Constant which is added to the y values.
  **Default Value**: 0
  **Value Suggestions**: Add ∈ \{-10, -5, 0, 5, 10\}
- **FunctionScaled** (output control) ...................... `function_id \mapsto HFunction1dX / VARIANT ( real, integer )`
  Transformed function.

**Parallelization Information**

ScaleYFunct1D is reentrant and processed without parallelization.

**Possible Predecessor Functions**

CreateFunct1DPairs, CreateFunct1DArray

**Module**

Tools

HALCON 6.0
Smooth an equidistant function with a Gaussian function.

The operator SmoothFunct1dGauss smooths a one-dimensional function with a Gaussian function.

Parameter

- **Function** (input control) ................. function \_id \sim HFunction1dX / VARIANT ( real, integer )
  Function to be smoothed.
- **Sigma** (input control) .................. number \sim double
  Sigma of the Gaussian function for the smoothing.
  Default Value: 2.0
  Value Suggestions: Sigma \in \{0.5, 1.0, 2.0, 3.0, 4.0, 5.0\}
  Typical Range of Values: 0.1 \leq Sigma \leq 0.1(lin)
  Minimal Value Step: 0.01
  Recommended Value Step: 0.2
- **SmoothedFunction** (output control) .......... function \_id \sim HFunction1dX / VARIANT( real )
  Smoothed function.

Parallelization Information

SmoothFunct1DGauss is *reentrant* and processed *without* parallelization.

Possible Predecessor Functions

CreateFunct1DPairs, CreateFunct1DArray

Possible Successor Functions

MatchFunct1DTrans, DistanceFunct1D

Module

Tools

Smooth a 1D function by averaging its values.

The operator SmoothFunct1dMean smooths a one dimensional function by applying an average (mean) filter multiple times.

Parameter

- **Function** (input control) ................. function \_id \sim HFunction1dX / VARIANT ( real, integer )
  1D function.
- **SmoothSize** (input control) .............. integer \sim long
  Size of the averaging mask.
  Default Value: 10
  Value Suggestions: SmoothSize \in \{1, 3, 5, 7, 9, 11, 13, 15, 21, 31, 51\}
  Typical Range of Values: 1 \leq SmoothSize \leq 1(lin)
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: (SmoothSize > 0)
12.6. FUNCTION

- **Iterations** (input control) .................................................. integer \( \sim \) long
  Number of iterations for the smoothing.
  
  Default Value: 3
  Value Suggestions: \( \text{Iterations} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \)
  Typical Range of Values: \( 1 \leq \text{Iterations} \leq 10 \)
  Minimal Value Step: 1
  Recommended Value Step: 1
  Restriction: \((\text{Iterations} \geq 1)\)

- **SmoothedFunction** (output control) .................. function \_Id \( \sim \) HFunction1dX / VARIANT ( real )
  Smoothed function.

---

**Parallelization Information**

SmoothFunct1DMean is **reentrant** and processed **without** parallelization.

---

**Possible Predecessor Functions**

CreateFunct1DArray

---

**Module**

---

Tools

---

```c
[out] HFunction1dX TransformedFunction HFunction1dX.TransformFunct1D
([in] VARIANT Params )

void HOperatorSetX.TransformFunct1D ([in] VARIANT Function, 
[in] VARIANT Params, [out] VARIANT TransformedFunction )
```

**Transform a function using given transformation parameters.**

TransformFunct1d transforms the input function Function using the transformation parameters given in Params. The function Function is passed as a tuple (see CreateFunct1dArray and CreateFunct1dPairs). The following model is used for the transformation between the two functions (see MatchFunct1dTrans):

\[
y_h(x) = a_1y(a_0x + a_4) + a_2.
\]

The output function TransformedFunction is obtained by transforming the x and y values of the input function separately with the above formula, i.e., the output function is not sampled again. To do so, the operator SampleFunct1d can be used.

---

**Parameter**

- **Function** (input control) .................. function \_Id \( \sim \) HFunction1dX / VARIANT ( real, integer )
  Input function.

- **Params** (input control) ............................................................ number \( \sim \) VARIANT ( real )
  Transformation parameters between the functions.

  Parameter Number: 4

- **TransformedFunction** (output control) .... function \_Id \( \sim \) HFunction1dX / VARIANT ( real, integer )
  Transformed function.

---

**Parallelization Information**

TransformFunct1D is **reentrant** and processed **without** parallelization.

---

**Possible Predecessor Functions**

CreateFunct1dPairs, CreateFunct1dArray, MatchFunct1dTrans

---

**Module**

---

Tools

---
void HFunction1dX.WriteFunct1D ([in] String FileName )
void HOperatorSetX.WriteFunct1D ([in] VARIANT Function, [in] String FileName )

Write a function to a file.

The operator WriteFunct1d writes the contents of Function to a file. The data is written in an ASCII format. Therefore, the file can be exchanged between different architectures. The data can be read by the operator ReadFunct1d. There is no specific extension for this kind of file.

Parameter

- Function (input control) ................. function_id ~ HFunction1dX / VARIANT ( real, integer )
  Function to be written.
- FileName (input control) .......................................................... filename ~ String
  Name of the file to be written.

Result

If the parameters are correct the operator WriteFunct1d returns the value TRUE. If the file could not be opened WriteFunct1d returns FAIL. Otherwise an exception handling is raised.

Parallelization Information

WriteFunct1D is reentrant and processed without parallelization.

Possible Predecessor Functions

CreateFunct1DPairs, CreateFunct1DArray

See Also

ReadFunct1D, WriteImage, WriteRegion, OpenFile

Alternatives

WriteTuple, FwriteString

Module

Basic operators

[out] double XMin HFunction1dX.XRangeFunct1D ([out] double XMax )
void HOperatorSetX.XRangeFunct1D ([in] VARIANT Function, [out] double XMin, [out] double XMax )

Smallest and largest x value of the function.

XRangeFunct1d calculates the smallest and the largest x value of Function.

Parameter

- Function (input control) ................. function_id ~ HFunction1dX / VARIANT ( real, integer )
  Input function.
- XMin (output control) ............................... number ~ double
  Smallest x value.
- XMax (output control) ............................... number ~ double
  Largest x value.

Parallelization Information

XRangeFunct1D is reentrant and processed without parallelization.

Possible Predecessor Functions

CreateFunct1DPairs, CreateFunct1DArray

Module

Tools
12.7. GEOMETRY

YRangeFunct1d calculates the smallest and the largest y value of function.

Parameter

- Function (input control) ...................... function_id  \( \sim \) HFunction1dX / VARIANT ( real, integer )
  Input function.
- YMin (output control) .............................. number  \( \sim \) double
  Smallest y value.
- YMax (output control) .............................. number  \( \sim \) double
  Largest y value.

Parallelization Information

YRangeFunct1D is reentrant and processed without parallelization.

Possible Predecessor Functions

CreateFunct1DPairs, CreateFunct1DArray

Module

12.7 Geometry

Calculate the angle between two lines.

The operator AngleLl calculates the angle between two lines. As input the rows and columns of the first line (RowA1,ColumnA1,RowA2,ColumnA2) and of the second line (RowB1,ColumnB1,RowB2,ColumnB2) are expected. The calculation in done as follows: We interpret the lines as vectors with starting points RowA1,ColumnA1 and RowB1,ColumnB1 and end points RowA2,ColumnA2 and RowB2,ColumnB2, respectively. Turning the vector A counterclockwise onto the vector B (the center of rotation is the intersection point of the two lines) yields the angle. The result depends on the order of the points and on the order of the lines. The parameter Angle returns the angle in radians. The angles range from \(-\pi \leq \text{Angle} \leq \pi\).

Parameter

- RowA1 (input control) .............................. point.y(-array)  \( \sim \) VARIANT ( real, integer )
  Row of the first point of the first line.
- ColumnA1 (input control) .............................. point.x(-array)  \( \sim \) VARIANT ( real, integer )
  Column of the first point of the first line.
- RowA2 (input control) .............................. point.y(-array)  \( \sim \) VARIANT ( real, integer )
  Row of the second point of the first line.
- ColumnA2 (input control) .............................. point.x(-array)  \( \sim \) VARIANT ( real, integer )
  Column of the second point of the first line.
- RowB1 (input control) .............................. point.y(-array)  \( \sim \) VARIANT ( real, integer )
  Row of the first point of the second line.
- RowB2 (input control) .............................. point.y(-array)  \( \sim \) VARIANT ( real, integer )
  Row of the second point of the second line.
- ColumnB1 (input control) .............................. point.x(-array)  \( \sim \) VARIANT ( real, integer )
  Column of the first point of the second line.
- ColumnB2 (input control) .............................. point.x(-array)  \( \sim \) VARIANT ( real, integer )
  Column of the second point of the second line.
Chapter 12. Tools

ColumnB1 (input control) ......................... point.x(-array) \rightarrow VARIANT (real, integer)
Column of the first point of the second line.

RowB2 (input control) ............................. point.y(-array) \rightarrow VARIANT (real, integer)
Row of the second point of the second line.

ColumnB2 (input control) ......................... point.x(-array) \rightarrow VARIANT (real, integer)
Column of the second point of the second line.

Angle (output control) ............................. number(-array) \rightarrow VARIANT
Angle between the lines.

Example

RowA1 := 255
ColumnA1 := 10
RowA2 := 255
ColumnA2 := 501
disp_line (WindowHandle, RowA1, ColumnA1, RowA2, ColumnA2)
RowB1 := 255
ColumnB1 := 255
for i := 1 to 360 by 1
  RowB2 := 255 + sin(rad(i)) * 200
  ColumnB2 := 255 + cos(rad(i)) * 200
disp_line (WindowHandle, RowB1, ColumnB1, RowB2, ColumnB2)
  angle_ll (RowA1, ColumnA1, RowA2, ColumnA2,
             RowB1, ColumnB1, RowB2, ColumnB2, Angle)
endfor

Result

AngleLl returns TRUE.

Parallelization Information

AngleLl is reentrant and processed without parallelization.

Alternatives

AngleLx

Module

Basic operators

[out] VARIANT Angle HMisc.AngleLx ([in] VARIANT Row1,

void HOperatorSetX.AngleLx ([in] VARIANT Row1, [in] VARIANT Column1,

Calculate the angle between one line and the vertical axis.

The operator AngleLx calculates the angle between one line and the abscissa. As input the row and column of
the line (Row1,Column1,Row2,Column2) are expected. The calculation is done as follows: We interpret
the line as a vector with starting point Row1,Column1 and end point Row2,Column2. Turning the vector
clockwise onto the abscissa (center of rotation is the intersection point of the abscissa) yields the angle. The result
is dependant on the order of points of line. The parameters Angle returns the angle in radians. The angles range
from \(-\pi \leq \text{Angle} \leq \pi\).

Parameter

- Row1 (input control) .............................. point.y(-array) \rightarrow VARIANT (real, integer)
  Row of the first point of the line.
- Column1 (input control) .......................... point.x(-array) \rightarrow VARIANT (real, integer)
  Column of the first point of the line.
- Row2 (input control) .............................. point.y(-array) \rightarrow VARIANT (real, integer)
  Row of the second point of the line.
12.7. GEOMETRY

Column2 (input control) ........................................ point.x(-array)  \rightarrow  VARIANT ( real, integer )
Column of the second point of the line.

Angle (output control) ........................................... number(-array)  \rightarrow  VARIANT
Angle between the line and the abscissa.

Example

RowX1 := 255
ColumnX1 := 10
RowX2 := 255
ColumnX2 := 501
disp_line (WindowHandle, RowX1, ColumnX1, RowX2, ColumnX2)
Row1 := 255
Column1 := 255
for i := 1 to 360 by 1
  Row2 := 255 + sin(rad(i)) * 200
  Column2 := 255 + cos(rad(i)) * 200
disp_line (WindowHandle, Row1, Column1, Row2, Column2)
  angle_lx (Row1, Column1, Row2, Column2, Angle)
endfor

Result

AngleLx returns TRUE.

Parallelization Information

AngleLx is reentrant and processed without parallelization.

Alternatives

AngleLl
Module

Basic operators

[out] VARIANT DistanceMin HRegionX.DistanceLr ([in] VARIANT Row1,
[in] VARIANT Column1, [in] VARIANT Row2, [in] VARIANT Column2,
[out] VARIANT DistanceMax )

void HOperatorSetX.DistanceLr ([in] IObjectX Region,
[in] VARIANT Row1, [in] VARIANT Column1, [in] VARIANT Row2,
[in] VARIANT Column2, [out] VARIANT DistanceMin, [out] VARIANT DistanceMax )

Calculate the distance between one line and one region.
The operator DistanceLr calculates the orthogonal distance between one line and one region. As input the
cordinates of 2 points that the line represent (Row1,Column1, Row2,Column2) and one region are expected.
The parameters DistanceMin and DistanceMax return the result of the calculation.

Attention

Due to efficiency of DistanceLr holes are ignored. Furthermore, if the lines intersects the region a minimal
distance larger than 0.5 can be returned.

Parameter

Region (input iconic) .................................................region  \rightarrow  HRegionX / IObjectX
Input region.

Row1 (input control) ............................................. point.y(-array)  \rightarrow  VARIANT ( real, integer )
Row of the first point of the line.

Column1 (input control) ...................................... point.x(-array)  \rightarrow  VARIANT ( real, integer )
Column of the first point of the line.

Row2 (input control) ............................................. point.y(-array)  \rightarrow  VARIANT ( real, integer )
Row of the second point of the line.
Column2 (input control) .................................. point.x(-array) \rightarrow \text{VARIANT (real, integer)}
Column of the second point of the line.

DistanceMin (output control) ............................... number(-array) \rightarrow \text{VARIANT}
Minimal distance between the line and the region

DistanceMax (output control) ............................... number(-array) \rightarrow \text{VARIANT}
Maximal distance between the line and the region

Example

dev_close_window ()
read_image (Image, 'fabrik')
dev_open_window (0, 0, 512, 512, 'white', WindowHandle)
threshold (Image, Region, 180, 255)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and',
5000, 100000000)
dev_clear_window ()
dev_set_color ('black')
dev_display (SelectedRegions)
dev_set_color ('red')
Row1 := 100
Row2 := 400
for Col := 50 to 400 by 4
  disp_line (WindowHandle, Row1, Col+100, Row2, Col)
  distance_lr (SelectedRegions, Row1, Col+100, Row2, Col,
    DistanceMin, DistanceMax)
endfor

Result

DistanceLr returns TRUE.

Parallelization Information

DistanceLr is reentrant and processed without parallelization.

See Also

HammingDistance, SelectRegionPoint, TestRegionPoint, SmallestRectangle2

Alternatives

DistancePr, DistanceSr, DiameterRegion

Module

Basic operators

[out] VARIANT Distance HMiscX.DistancePl ([in] VARIANT Row,
[in] VARIANT Column, [in] VARIANT Row1, [in] VARIANT Column1,
[in] VARIANT Row2, [in] VARIANT Column2 )

void HOperatorSetX.DistancePl ([in] VARIANT Row, [in] VARIANT Column,
[in] VARIANT Row1, [in] VARIANT Column1, [in] VARIANT Row2,
[in] VARIANT Column2, [out] VARIANT Distance )

Calculate the distance between one point and one line.

The operator DistancePl calculates the orthogonal distance between a point (Row,Column) and a line, given by two arbitrary points of the line. The result is passed in Distance.

Parameter

\begin{itemize}
  \item Row (input control) .................................. point.y(-array) \rightarrow \text{VARIANT (real, integer)}
  Row of the point.
  \item Column (input control) ............................. point.x(-array) \rightarrow \text{VARIANT (real, integer)}
  Column of the point.
\end{itemize}
Example

```c
read_image (Image, 'mreut')
dev_open_window (0, 0, 512, 512, 'white', WindowHandle)
dev_display (Image)
dev_set_color ('black')
threshold (Image, Region, 180, 255)
dev_clear_window ()
dev_display (Region)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and',
          10000, 100000000)
get_region_contour (SelectedRegions, Rows, Columns)
RowLine1 := 5
ColLine1 := 300
RowLine2 := 300
ColLine2 := 400
NumberTuple := |Rows|
dev_set_color ('red')
disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
dev_set_color ('green')
for i := 1 to NumberTuple by 5
    disp_line (WindowHandle, Rows[i], Columns[i]-2, Rows[i], Columns[i]+2)
    disp_line (WindowHandle, Rows[i]-2, Columns[i], Rows[i]+2, Columns[i])
    distance_pl (Rows[i], Columns[i], RowLine1, ColLine1,
                 RowLine2, ColLine2, Distance)
endfor
```

Result

DistancePl returns TRUE.

Parallelization Information

DistancePl is reentrant and processed without parallelization.

See Also

DistancePp, DistancePr

Alternatives

DistancePs

Module

Basic operators

```c
[out] VARIANT Distance HMiscX.DistancePp ([in] VARIANT Row1,

void HOperatorSetX.DistancePp ([in] VARIANT Row1, [in] VARIANT Column1,
[in] VARIANT Row2, [in] VARIANT Column2, [out] VARIANT Distance )
```

Calculate the distance between two points.
The operator DistancePp calculates the distance between pairs of points according to the following formula:

\[
\text{Distance} = \sqrt{((\text{Row}1 - \text{Row}2)^2 + (\text{Column}1 - \text{Column}2)^2)}
\]

The result is passed in Distance.

---

**Parameter**

- **Row1** (input control) point.y(-array) ~ VARIANT (real, integer)
  Row of the first point.
- **Column1** (input control) point.x(-array) ~ VARIANT (real, integer)
  Column of the first point.
- **Row2** (input control) point.y(-array) ~ VARIANT (real, integer)
  Row of the second point.
- **Column2** (input control) point.x(-array) ~ VARIANT (real, integer)
  Column of the second point.
- **Distance** (output control) number(-array) ~ VARIANT
  Distance between the points

---

**Example**

```
dev_close_window()
read_image(Image, 'mreut')
dev_open_window(0, 0, 512, 512, 'white', WindowHandle)
dev_display(Image)
dev_set_color('black')
threshold(Image, Region, 180, 255)
dev_clear_window()
dev_display(Region)
connection(Region, ConnectedRegions)
select_shape(ConnectedRegions, SelectedRegions, 'area', 'and', 10000, 100000000)
get_region_contour(SelectedRegions, Rows, Columns)
RowPoint := 80
ColPoint := 250
NumberTuple := |Rows|
dev_set_color('red')
set_draw(WindowHandle, 'margin')
disp_circle(WindowHandle, RowPoint, ColPoint, 10)
dev_set_color('green')
for i := 1 to NumberTuple by 10
  disp_line(WindowHandle, Rows[i], Columns[i]-2, Rows[i], Columns[i]+2)
  disp_line(WindowHandle, Rows[i]-2, Columns[i], Rows[i]+2, Columns[i])
distance_pp(RowPoint, ColPoint, Rows[i], Columns[i], Distance)
endfor
```

**Result**

DistancePp returns TRUE.

**Parallelization Information**

DistancePp is **reentrant** and processed **without** parallelization.

**See Also**

DistancePl, DistancePr

**Alternatives**

DistancePs

**Module**

Basic operators
# 12.7. GEOMETRY

---

**Calculate the distance between one point and one region.**

The operator DistancePr calculates the distance between one point and one region. As input the column and Row of the point (Row, Column) and one region are expected. If the point is inside the region the minimal distance is zero. The parameters DistanceMin and DistanceMax return the result of the calculation.

### Parameter

- **Region** (input iconic) region  ~ HRegionX / IObjectX  Input region.
- **Row** (input control) point.y(-array)  ~ VARIANT (real, integer)  Row of the point.
- **Column** (input control) point.x(-array)  ~ VARIANT (real, integer)  Column of the point.
- **DistanceMin** (output control) number(-array)  ~ VARIANT  Minimal distance between the point and the region
- **DistanceMax** (output control) number(-array)  ~ VARIANT  Maximal distance between the point and the region

### Example

```c
dev_close_window ()
read_image (Image, 'mreut')
dev_open_window (0, 0, 512, 512, 'white', WindowHandle)
de_set_color ('black')
threshold (Image, Region, 180, 255)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 10000, 10000000)
Row1 := 255
Column1 := 255
de_clear_window ()
de_display (SelectedRegions)
de_set_color ('red')
for i := 1 to 360 by 1
  Row2 := 255 + sin(rad(i)) * 200
  Column2 := 255 + cos(rad(i)) * 200
disp_line (WindowHandle, Row1, Column1, Row2, Column2)
distance_pr (SelectedRegions, Row2, Column2, DistanceMin, DistanceMax)
endfor
```

### Result

DistancePr returns TRUE.

### Parallelization Information

DistancePr is reentrant and processed without parallelization.

### See Also

HammingDistance, SelectRegionPoint, TestRegionPoint, SmallestRectangle2

### Alternatives

DistanceLr, DistanceSr, DiameterRegion

### Module

Basic operators
Calculate the distances between a point and a line segment.

The operator DistancePs calculates the minimal and maximal distance between a point \((\text{Row}, \text{Column})\) and a line segment which is represented by the start point \((\text{Row1}, \text{Column1})\) and the end point \((\text{Row2}, \text{Column2})\). DistanceMax is the maximal distance between the point and the end points of the line segment. DistanceMin is identical to DistancePl in the case that the point is “between” the two endpoints. Otherwise the minimal distance to one of the endpoints is used.

**Parameter**

- **Row** (input control) \.................................................. point.y(-array) \(\leadsto\) VARIANT (real, integer)
  Row of the first point.

- **Column** (input control) \.................................................. point.x(-array) \(\leadsto\) VARIANT (real, integer)
  Column of the first point.

- **Row1** (input control) \.................................................. point.y(-array) \(\leadsto\) VARIANT (real, integer)
  Row of the first point of the line segment.

- **Column1** (input control) \.................................................. point.x(-array) \(\leadsto\) VARIANT (real, integer)
  Column of the first point of the line segment.

- **Row2** (input control) \.................................................. point.y(-array) \(\leadsto\) VARIANT (real, integer)
  Row of the second point of the line segment.

- **Column2** (input control) \.................................................. point.x(-array) \(\leadsto\) VARIANT (real, integer)
  Column of the second point of the line segment.

- **DistanceMin** (output control) \.................................................. number(-array) \(\leadsto\) VARIANT
  Minimal distance between the point and the line segment

- **DistanceMax** (output control) \.................................................. number(-array) \(\leadsto\) VARIANT
  Maximal distance between the point and the line segment

**Example**

read_image (Image, ’mreut’)
dev_open_window (0, 0, 512, 512, ’white’, WindowHandle)
dev_display (Image)
dev_set_color (’black’)
threshold (Image, Region, 180, 255)
dev_clear_window ()
dev_display (Region)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, ’area’, ’and’,
            10000, 100000000)
get_region_contour (SelectedRegions, Rows, Columns)
RowLine1 := 400
ColLine1 := 50
RowLine2 := 50
ColLine2 := 450
NumberTuple := |Rows|
dev_set_color (’red’)
disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
dev_set_color (’green’)
for i := 1 to NumberTuple by 10
  disp_line (WindowHandle, Rows[i], Columns[i]-2, Rows[i], Columns[i]+2)
  disp_line (WindowHandle, Rows[i]-2, Columns[i], Rows[i]+2, Columns[i])
  distance_ps (Rows[i], Columns[i], RowLine1, ColLine1, RowLine2, ColLine2,
             DistanceMin, DistanceMax)
DistancePs returns TRUE.

DistancePs is reentrant and processed without parallelization.

DistancePp, DistancePr

DistancePl

Module

Basic operators

\[
\text{HRegionX.DistanceRrMin} ([\text{in}] \text{HRegionX Regions2}, [\text{out}] \text{VARIANT Row1}, [\text{out}] \text{VARIANT Column1}, \\
[\text{out}] \text{VARIANT Row2}, [\text{out}] \text{VARIANT Column2})
\]

\[
\text{void HOperatorSetX.DistanceRrMin} ([\text{in}] \text{IObjectX Regions1}, \\
[\text{in}] \text{IObjectX Regions2}, [\text{out}] \text{VARIANT MinDistance}, [\text{out}] \text{VARIANT Row1}, \\
[\text{out}] \text{VARIANT Column1}, [\text{out}] \text{VARIANT Row2}, [\text{out}] \text{VARIANT Column2})
\]

Minimum distance between the contour pixels of two regions each.

The operator DistanceRrMin calculates the minimum distance of pairs of regions. If several regions are passed in Regions1 and Regions2 the distance between the contour pixels of each i-th element is calculated and then forms the i-th entry in the output parameter MinDistance. The calculation is carried out by comparing all contour pixels. The Euclidean distance is used. The parameters (Row1, Column1) and (Row2, Column2) indicate the position on the contour of Regions1 and Regions2, respectively, the distance between which is the minimum distance.

Attention

Each region must consist of exactly one connection component. Both input parameters must contain the same number of regions. The regions must not be empty. If the regions overlap the distance is indicated as 0.0. In this case the positions are not reliable.

Parameter

- **Regions1** (input iconic) region(-array) \(\sim\) HRegionX / IObjectX Regions to be examined.
- **Regions2** (input iconic) region(-array) \(\sim\) HRegionX / IObjectX Regions to be examined.
- **MinDistance** (output control) real(-array) \(\sim\) VARIANT(real) Minimum distance between contours of the regions.
  
  **Restriction:** \(0 \leq \text{MinDistance}\)
- **Row1** (output control) point.y(-array) \(\sim\) VARIANT(integer) Line index on contour in Regions1.
- **Column1** (output control) point.x(-array) \(\sim\) VARIANT(integer) Column index on contour in Regions1.
- **Row2** (output control) point.y(-array) \(\sim\) VARIANT(integer) Line index on contour in Regions2.
- **Column2** (output control) point.x(-array) \(\sim\) VARIANT(integer) Column index on contour in Regions2.

Result

The operator DistanceRrMin returns the value TRUE if the input is not empty. Otherwise an exception handling is raised.

Parallelization Information

DistanceRrMin is reentrant and processed without parallelization.
Possible Predecessor Functions
Threshold, Region-growing, Connection

Alternatives
DistanceRrMinDil, Dilation1, Intersection

Module
Region processing

Minimum distance between two regions with the help of dilatation.
The operator DistanceRrMinDil calculates the minimum distance between pairs of regions. If several regions are passed in Regions1 and Regions2 the distance between the i-th elements in each case is calculated. It then forms the i-th entry in the output parameter MinDistance. The calculation is carried out with the help of dilatation with the Golay element 'h'. The result is:

\[ \text{Number of iterations} = 2^{\text{MinDistance}} - 1 \]

The mask 'h' has the effect that precisely the maximum metrics are calculated.

Attention
Both parameters must contain the same number of regions. The regions must not be empty.

Parameter

- **Regions1** (input iconic) ........................................... region(-array) \( \bowtie \) HRegionX / IObjectX
Regions to be examined.

- **Regions2** (input iconic) ........................................... region(-array) \( \bowtie \) HRegionX / IObjectX
Regions to be examined.

- **MinDistance** (output control) .................................. integer(-array) \( \bowtie \) VARIANT (integer)
Minimum distances of the regions.

**Restriction**: \( -1 \leq \text{MinDistance} \)

Result
The operator DistanceRrMinDil returns the value TRUE if the input is not empty. Otherwise an exception handling is raised.

Parallelization Information
DistanceRrMinDil is reentrant and processed without parallelization.

Possible Predecessor Functions
Threshold, Region-growing, Connection

Alternatives
DistanceRrMin, Dilation1, Intersection

Module
Region processing
Calculate the distances between one line segment and one line.

The operator \texttt{DistanceSl} calculates the minimal and maximal orthogonal distance between one line segment and one line. As input the columns and rows of the line segment (RowA1, ColumnA1, RowA2, ColumnA2) and of the line (RowB1, ColumnB1, RowB2, ColumnB2) are expected. The parameters DistanceMin and DistanceMax return the result of the calculation. If the line segments are intersecting DistanceMin returns zero.

Parameter

- **\texttt{RowA1}** (input control) \quad \texttt{point.y(-array)} \quad \texttt{VARIANT (real, integer)}
  Row of the first point of the line segment.

- **\texttt{ColumnA1}** (input control) \quad \texttt{point.x(-array)} \quad \texttt{VARIANT (real, integer)}
  Column of the first point of the line segment.

- **\texttt{RowA2}** (input control) \quad \texttt{point.y(-array)} \quad \texttt{VARIANT (real, integer)}
  Row of the second point of the line segment.

- **\texttt{ColumnA2}** (input control) \quad \texttt{point.x(-array)} \quad \texttt{VARIANT (real, integer)}
  Column of the second point of the line segment.

- **\texttt{RowB1}** (input control) \quad \texttt{point.y(-array)} \quad \texttt{VARIANT (real, integer)}
  Row of the first point of the line.

- **\texttt{ColumnB1}** (input control) \quad \texttt{point.x(-array)} \quad \texttt{VARIANT (real, integer)}
  Column of the first point of the line.

- **\texttt{RowB2}** (input control) \quad \texttt{point.y(-array)} \quad \texttt{VARIANT (real, integer)}
  Row of the second point of the line.

- **\texttt{ColumnB2}** (input control) \quad \texttt{point.x(-array)} \quad \texttt{VARIANT (real, integer)}
  Column of the second point of the line.

- **\texttt{DistanceMin}** (output control) \quad \texttt{number(-array)} \quad \texttt{VARIANT}
  Minimal distance between the line segment and the line

- **\texttt{DistanceMax}** (output control) \quad \texttt{number(-array)} \quad \texttt{VARIANT}
  Maximal distance between the line segment and the line

Example

```plaintext
dev_set_color ('black')
RowLine1 := 400
Colline1 := 200
RowLine2 := 200
Colline2 := 400
Rows := 300
Columns := 50
disp_line (WindowHandle, RowLine1, CollLine1, RowLine2, CollLine2)
dev_set_color ('green')
n := 0
for Rows := 40 to 200 by 4
  disp_line (WindowHandle, Rows+n, Columns+n, Rows, Columns+n)
distance_sl (Rows+n, Columns+n, Rows, Columns+n, RowLine1, CollLine1, RowLine2, CollLine2, DistanceMin, DistanceMax)
n := n+10
endfor
```

HALCON 6.0
DistanceSl returns TRUE.

**Parallelization Information**

DistanceSl is reentrant and processed without parallelization.

**See Also**

DistancePs, DistancePp

**Alternatives**

DistancePl

**Module**

Basic operators

```c
[out] VARIANT DistanceMin HRegionX.DistanceSr ([in] VARIANT Row1, [in] VARIANT Column1, [in] VARIANT Row2, [in] VARIANT Column2, [out] VARIANT DistanceMax )

```

Calculate the distance between one line segment and one region.

The operator `DistanceSr` calculates the distance between one line segment and one region. Row1, Column1, Row2, Column2 are the initial and end coordinates of the line segment. The parameters DistanceMin and DistanceMax contain the resulting distances.

**Attention**

Due to efficiency of `DistanceSr` holes are ignored. Furthermore, if the lines intersect the region a minimal distance larger than 0.5 can be returned.

**Parameter**

- **Region** (input iconic) 
  Input region.
- **Row1** (input control) 
  Row of the first point of the line segment.
- **Column1** (input control) 
  Column of the first point of the line segment.
- **Row2** (input control) 
  Row of the second point of the line segment.
- **Column2** (input control) 
  Column of the second point of the line segment.
- **DistanceMin** (output control) 
  Minimal distance between the line segment and the region
- **DistanceMax** (output control) 
  Maximal distance between the line segment and the region

**Example**

```c
read_image (Image, 'fabrik')
dev_open_window (0, 0, 512, 512, 'white', WindowHandle)
dev_display (Image)
threshold (Image, Region, 180, 255)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 5000, 100000000)
dev_clear_window ()
dev_set_color ('black')
dev_display (SelectedRegions)
```
dev_set_color ('red')
Row1 := 100
Row2 := 400
n := 0
for Col := 50 to 400 by 5
    disp_line (WindowHandle, Row1+n, Col, Row2-n, Col+100)
    distance_sr (SelectedRegions, Row1+n, Col, Row2-n, Col+100, DistanceMin, DistanceMax)
    n := n+5
endfor

Result
DistanceSr returns TRUE.

Parallelization Information
DistanceSr is reentrant and processed without parallelization.

See Also
HammingDistance, SelectRegionPoint, TestRegionPoint, SmallestRectangle2

Alternatives
DistanceLr, DistancePr, DiameterRegion

Module
Basic operators

Calculate the distances between two line segments.
The operator DistanceSs calculates the minimal and maximal distance between two line segments. As input the rows and columns of the first line segments (RowA1, ColumnA1, RowA2, ColumnA2) and of the second line segment (RowB1, ColumnB1, RowB2, ColumnB2) are used. The parameters DistanceMin and DistanceMax return the result of the calculation. If the line segments are intersecting DistanceMin returns zero.

Parameter

- **RowA1** (input control) point.y(-array) ~ VARIANT (real, integer)
  Row of the first point of the line segment.
- **ColumnA1** (input control) point.x(-array) ~ VARIANT (real, integer)
  Column of the first point of the line segment.
- **RowA2** (input control) point.y(-array) ~ VARIANT (real, integer)
  Row of the second point of the line segment.
- **ColumnA2** (input control) point.x(-array) ~ VARIANT (real, integer)
  Column of the second point of the line segment.
- **RowB1** (input control) point.y(-array) ~ VARIANT (real, integer)
  Row of the first point of the line.
- **ColumnB1** (input control) point.x(-array) ~ VARIANT (real, integer)
  Column of the first point of the line.
- **RowB2** (input control) point.y(-array) ~ VARIANT (real, integer)
  Row of the second point of the line.
- **ColumnB2** (input control) point.x(-array) ~ VARIANT (real, integer)
  Column of the second point of the line.
CHAPTER 12. TOOLS

- **DistanceMin** (output control) ........................................... number(-array) \(\sim\) VARIANT
  
  Minimal distance between the line segments

- **DistanceMax** (output control) ........................................... number(-array) \(\sim\) VARIANT
  
  Maximal distance between the line segments

---

**Example**

```c
dev_set_color (‘black’)
RowLine1 := 400
ColLine1 := 200
RowLine2 := 240
ColLine2 := 400
Rows := 300
Columns := 50
disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
dev_set_color (‘red’)
n := 0
for Rows := 40 to 200 by 4
  disp_line (WindowHandle, Rows, Columns, Rows+n, Columns+n)
  distance_ss (Rows, Columns, Rows+n, Columns+n, RowLine1, ColLine1,
  RowLine2, ColLine2, DistanceMin, DistanceMax)
  n := n+8
endfor
```

**Result**

DistanceSs returns TRUE.

---

**Parallelization Information**

DistanceSs is reentrant and processed without parallelization.

---

**See Also**

DistancePl, DistancePs

---

**Alternatives**

DistancePp

---

**Module**

Basic operators

---

```c
[out] VARIANT RowPoint HMiscX.GetPointsEllipse ([in] VARIANT Angle,
[in] double Radius2, [out] VARIANT ColPoint )
```

```c
void HOperatorSetX.GetPointsEllipse ([in] VARIANT Angle,
[in] double Radius2, [out] VARIANT RowPoint, [out] VARIANT ColPoint )
```

**Points of an ellipse corresponding to specific angles.**

GetPointsEllipse returns the points (RowPoint,ColPoint) on the specified ellipse corresponding to the angles in Angle, which refer to the main axis of the ellipse. The ellipse itself is characterized by the center (Row, Column), the orientation of the main axis Phi, the length of the larger half axis Radius1, and the length of the smaller half axis Radius2.

---

**Parameter**

- **Angle** (input control) ............................................... real(-array) \(\sim\) VARIANT ( real )
  
  Angles corresponding to the resulting points [rad].

  **Default Value:** 0

  **Restriction:** \(((\text{Angle} \geq 0) \land (\text{Angle} \leq 6.283185307))\)

- **Row** (input control) ............................................... ellipse.center.y \(\sim\) double
  
  Row coordinate of the center of the ellipse.
Column (input control) ............................................ ellipse.center.x  \sim \text{double} \\
Column coordinate of the center of the ellipse.

Phi (input control) ............................................. ellipse.angle.rad  \sim \text{double} \\
Orientation of the main axis [rad]. 
\textbf{Restriction:} \((\Phi \geq 0) \land (\Phi \leq 6.283185307)\)

Radius1 (input control) .................................... ellipse.radius1  \sim \text{double} \\
Length of the larger half axis. 
\textbf{Restriction:} \((\text{Radius1} > 0)\)

Radius2 (input control) .................................... ellipse.radius2  \sim \text{double} \\
Length of the smaller half axis. 
\textbf{Restriction:} \((\text{Radius2} \geq 0)\)

RowPoint (output control) .................................. point.row(-array) \sim \text{VARIANT (real)} \\
Row coordinates of the points on the ellipse.

ColPoint (output control) .................................. point.column(-array) \sim \text{VARIANT (real)} \\
Column coordinates of the points on the ellipse.

Example

draw_ellipse(WindowHandle, Row, Column, Phi, Radius1, Radius2) 
get_points_ellipse([0,3.14], Row, Column, Phi, Radius1, Radius2, RowPoint, ColPoint)

Result
GetPointsEllipse returns TRUE if all parameter values are correct. If necessary, an exception is raised.

Parallelization Information
GetPointsEllipse is \textit{reentrant} and processed \textit{without} parallelization.

Possible Predecessor Functions
FitEllipseContourXld, DrawEllipse, GenEllipseContourXld
See Also
GenEllipseContourXld

Module

Basic operators

\[
\begin{align*}
\text{[out]} \ \text{VARIANT Row} & \quad \text{HMiscX.IntersectionLl} \quad \text{([in]} \ \text{VARIANT RowA1}, \\
& \quad \text{[in]} \ \text{VARIANT ColumnA1}, \quad \text{[in]} \ \text{VARIANT RowA2}, \quad \text{[in]} \ \text{VARIANT ColumnA2}, \\
& \quad \text{[in]} \ \text{VARIANT RowB1}, \quad \text{[in]} \ \text{VARIANT ColumnB1}, \quad \text{[in]} \ \text{VARIANT RowB2}, \\
& \quad \text{[in]} \ \text{VARIANT ColumnB2}, \quad \text{[out]} \ \text{VARIANT Column}, \quad \text{[out]} \ \text{VARIANT IsParallel})
\end{align*}
\]

\[
\begin{align*}
\text{void HOperatorSetX.IntersectionLl} \quad \text{([in]} \ \text{VARIANT RowA1}, \\
& \quad \text{[in]} \ \text{VARIANT ColumnA1}, \quad \text{[in]} \ \text{VARIANT RowA2}, \quad \text{[in]} \ \text{VARIANT ColumnA2}, \\
& \quad \text{[in]} \ \text{VARIANT RowB1}, \quad \text{[in]} \ \text{VARIANT ColumnB1}, \quad \text{[in]} \ \text{VARIANT RowB2}, \\
& \quad \text{[in]} \ \text{VARIANT ColumnB2}, \quad \text{[out]} \ \text{VARIANT Row}, \quad \text{[out]} \ \text{VARIANT Column}, \\
& \quad \text{[out]} \ \text{VARIANT IsParallel})
\end{align*}
\]

Calculate the intersection point of two lines.

The operator IntersectionLl calculates the intersection point of two lines. As input the columns and rows of the lines (RowA1,ColumnA1,RowA2,ColumnA2) and (RowB1,ColumnB1,RowB2,ColumnB2) are expected. The parameters Row and Column return the result of the calculation. If the lines are parallel IsParallel is 1 else 0. In addition the values of Row and Column are undefined.

Attention
If the lines are parallel the values of Row and Column are undefined.
Parameter

- **RowA1** (input control) .......................................................... point.y(-array) \(\sim\) VARIANT (real, integer)
  Row of the first point of the first line.
- **ColumnA1** (input control) .................................................. point.x(-array) \(\sim\) VARIANT (real, integer)
  Column of the first point of the first line.
- **RowA2** (input control) .......................................................... point.y(-array) \(\sim\) VARIANT (real, integer)
  Row of the second point of the first line.
- **ColumnA2** (input control) .................................................. point.x(-array) \(\sim\) VARIANT (real, integer)
  Column of the second point of the first line.
- **RowB1** (input control) .......................................................... point.y(-array) \(\sim\) VARIANT (real, integer)
  Row of the first point of the second line.
- **ColumnB1** (input control) .................................................. point.x(-array) \(\sim\) VARIANT (real, integer)
  Column of the first point of the second line.
- **RowB2** (input control) .......................................................... point.y(-array) \(\sim\) VARIANT (real, integer)
  Row of the second point of the second line.
- **ColumnB2** (input control) .................................................. point.x(-array) \(\sim\) VARIANT (real, integer)
  Column of the second point of the second line.
- **Row** (output control) .......................................................... point.y(-array) \(\sim\) VARIANT
  Row of the intersection point
- **Column** (output control) .................................................. point.x(-array) \(\sim\) VARIANT
  Column of the intersection point
- **IsParallel** (output control) .............................................. number(-array) \(\sim\) VARIANT
  Are the two lines parallel?

Example

```haskell
dev_set_color ('black')
RowLine1 := 350
ColLine1 := 250
RowLine2 := 300
ColLine2 := 300
Rows := 300
Columns := 50
disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
n := 0
for Rows := 40 to 200 by 4
  dev_set_color ('red')
disp_line (WindowHandle, Rows, Columns, Rows+n, Columns+n)
  intersection_ll (Rows, Columns, Rows+n, Columns+n, RowLine1, ColLine1,
    RowLine2, ColLine2, Row, Column, IsParallel)
  disp_line (WindowHandle, Row, Column-2, Row, Column+2)
  disp_line (WindowHandle, Row-2, Column, Row+2, Column)
  n := n+8
endfor
```

Result

IntersectionLl returns TRUE.

Parallelization Information

IntersectionLl is reentrant and processed without parallelization.

Module

Basic operators
Calculate the projection of a point onto a line.

The operator ProjectionPl calculates the projection of a point (Row, Column) onto a line which is represented by the start point (Row1, Column1) and the end point (Row2, Column2). RowProj is the row of the projection point and ColProj is the column of the projection point.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row (input control)</td>
<td>point.y(-array) ~ VARIANT (real, integer)</td>
</tr>
<tr>
<td>Column (input control)</td>
<td>point.x(-array) ~ VARIANT (real, integer)</td>
</tr>
<tr>
<td>Row1 (input control)</td>
<td>point.y(-array) ~ VARIANT (real, integer)</td>
</tr>
<tr>
<td>Column1 (input control)</td>
<td>point.x(-array) ~ VARIANT (real, integer)</td>
</tr>
<tr>
<td>Row2 (input control)</td>
<td>point.y(-array) ~ VARIANT (real, integer)</td>
</tr>
<tr>
<td>Column2 (input control)</td>
<td>point.x(-array) ~ VARIANT (real, integer)</td>
</tr>
<tr>
<td>RowProj (output control)</td>
<td>number(-array) ~ VARIANT</td>
</tr>
<tr>
<td>ColProj (output control)</td>
<td>number(-array) ~ VARIANT</td>
</tr>
</tbody>
</table>

Example

```
dev_set_color ('black')
RowLine1 := 400
ColLine1 := 200
RowLine2 := 240
ColLine2 := 400
Rows := 300
Columns := 50
disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
n := 0
for Rows := 40 to 200 by 4
  dev_set_color ('red')
  disp_circle (WindowHandle, Rows+n, Columns, 2)
  projection_pl (Rows+n, Columns, RowLine1, ColLine1, RowLine2, ColLine2, RowProj, ColProj)
  dev_set_color ('blue')
  disp_line (WindowHandle, RowProj-2, ColProj, RowProj+2, ColProj)
  disp_line (WindowHandle, RowProj, ColProj-2, RowProj, ColProj+2)
n := n+8
endfor

Result

ProjectionPl returns TRUE.

Parallelization Information

ProjectionPl is reentrant and processed without parallelization.
# 12.8 Hough

Return the Hough-Transform for circles with a given radius.

The operator `HoughCircleTrans` calculates the Hough transform for circles with a certain radius in the regions passed by `Region`. Hereby the centres of all possible circles in the parameter space (the Hough or accumulator space respectively) will be accumulated for each point in the image space. Circle hypotheses supported by many points in the input region thereby generate a maximum in the area showing the circle’s centre in the output image (`HoughImage`). The circles’ centres in the image space can be deduced from the coordinates of these maximums by subtracting the `Radius`. If more than one radius is transmitted, all Hough images will be shifted according to the maximal radius.

Parameter

- **Region** (input iconic)  
  Binary edge image in which the circles are to be detected.
- **HoughImage** (output iconic)  
  Hough transform for circles with a given radius.
- **Radius** (input control)  
  Radius of the circle to be searched in the image.
  - **Default Value:** 12
  - **Recommended Value Step:** 1

Result

The operator `HoughCircleTrans` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>)`, the behavior in case of empty region is set via `SetSystem('emptyRegionResult',<Result>)`. If necessary an exception handling is raised.

Parallelization Information

`HoughCircleTrans` is reentrant and processed without parallelization.

Module

```c
[out] HImageX HoughImage HRegionX.HoughCircleTrans  
([in] VARIANT Radius )

void HOperatorSetX.HoughCircleTrans ([in] IObjectX Region,  
```

Centres of circles for a specific radius.

`HoughCircleTrans` detects the centres of circles in regions with the help of the Hough transform for circles with a specific radius.
12.8. HOUGH

**Parameter**

- **RegionIn** (input iconic)  
  Binary edge image in which the circles are to be detected.
- **RegionOut** (output iconic)  
  Centres of those circles which are included in the edge image by percent percent. 
  Parameter Number: \( \text{RegionOut} = (\text{Radius} \cdot \text{Percent}) \cdot \text{Mode} \)
- **Radius** (input control)  
  Radius of the circle to be searched in the image. 
  Default Value: 12 
  Typical Range of Values: \( 2 \leq \text{Radius} \leq 2 \) (lin) 
  Minimal Value Step: 1 
  Recommended Value Step: 1 
  Parameter Number: \( (1 \leq \text{Radius}) \leq 500 \)
- **Percent** (input control)  
  Indicates the percentage (approximately) of the (ideal) circle which must be present in the edge image RegionIn. 
  Default Value: 60 
  Typical Range of Values: \( 10 \leq \text{Percent} \leq 10 \) (lin) 
  Minimal Value Step: 1 
  Recommended Value Step: 5 
  Parameter Number: \( (1 \leq \text{Percent}) \leq 100 \)
- **Mode** (input control)  
  Defines the position of the circle in question: 
  0 - the radius is equivalent to the outer border of the set pixels. 
  1 - the radius is equivalent to the centres of the circle lines pixels. 
  2 - both 0 and 1 (a little more fuzzy, but more reliable in contrast to circles set slightly differently, necessitates 50 % more processing capacity compared to 0 or 1 alone). 
  Value List: Mode \( \in \{0, 1, 2\} \) 
  Parameter Number: \( (1 \leq \text{Mode}) \leq 3 \)

**Result**

The operator HoughCircles returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem('noObjectResult',<Result>). the behavior in case of empty region is set via SetSystem('emptyRegionResult',<Result>). If necessary an exception handling is raised.

**Parallelization Information**

HoughCircles is reentrant and processed without parallelization.

**Module**

```plaintext
[out] HImageX HoughImage HRegionX.HoughLineTrans 
([in] long AngleResolution )
```

```plaintext
void HOperatorSetX.HoughLineTrans ([in] IHObjectX Region, 
```

Produce the Hough transform for lines within regions.

The operator HoughLineTrans calculates the Hough transform for lines in those regions transmitted by Region. Thereby the angle and the lengths of the lines normal vectors are registered in the parameter space (the Hough- or accumulator space respectively). This means that the parameterization is executed according to the HNF.

The result is registered in a newly generated Int2-Image (HoughImage), whereby the x-axis is equivalent to the angle between the normal vector and the x-axis (in the original image), and the y-axis is equivalent to the distance of the line from the origin.

The angle ranges from -90 to 180 degrees and will be registered with a resolution of \( 1 / \text{AngleResolution} \), which means that one pixel in x-direction is equivalent to \( 1 / \text{AngleResolution} \) and that the HoughImage has a width of...
The height of the HoughImage corresponds to the diagonal of the surrounding rectangle of the input region.

The maxima in the result image are equivalent to the parameter values of the lines in the original image.

Parameter

- **Region** (input iconic) ... region \( \bowtie \text{HRegionX} / \text{IOObjectX} \)
  Binary edge image in which lines are to be detected.

- **HoughImage** (output iconic) ... image \( \bowtie \text{HImageX} / \text{HUntypedObjectX} (\text{int2}) \)
  Hough transform for lines.

- **AngleResolution** (input control) ... integer \( \bowtie \text{long} \)
  Adjusting the resolution in the angle area.

  - **Default Value**: 4
  - **Value List**: \( \text{AngleResolution} \in \{1, 2, 4, 8\} \)

Result

The operator HoughLineTrans returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

HoughLineTrans is reentrant and processed without parallelization.

Possible Predecessor Functions

Threshold, Skeleton

Possible Successor Functions

Threshold, LocalMax

See Also

HoughCircleTrans, GenRegionHline

Module

Region processing

```
[out] VARIANT Angle \text{HRegionX.HoughLines} ([in] \text{long AngleResolution},
[out] VARIANT Dist )
```

Detect lines in edge images with the help of the Hough transform and returns it in HNF.

The operator HoughLines allows the selection of linelike structures in a region, whereby it is not necessary that the individual points of a line are connected. This process is based on the Hough transform. The lines are returned in HNF, that is by the direction and length of their normal vector.

The parameter AngleResolution defines the degree of exactness concerning the determination of the angles. It amounts to \(1 / \text{AngleResolution} \) degree. The parameter Threshold determines by how many points of the original region a line’s hypothesis has to be supported at least in order to be taken over into the output. The parameters AngleGap and DistGap define a neighborhood of the points in the Hough image in order to determine the local maxima. The lines are returned in HNF.

Parameter

- **RegionIn** (input iconic) ... region \( \bowtie \text{HRegionX} / \text{IOObjectX} \)
  Binary edge image in which the lines are to be detected.

- **AngleResolution** (input control) ... integer \( \bowtie \text{long} \)
  Adjusting the resolution in the angle area.

  - **Default Value**: 4
  - **Value List**: \( \text{AngleResolution} \in \{1, 2, 4, 8\} \)
Threshold (input control) .......................................................... integer \rightarrow long
\textbf{Default Value: 100}

\textbf{AngleGap} (input control) ................................................. integer \rightarrow long
Minimal distance of two maxima in the Hough image (direction: angle).
\textbf{Default Value: 5}

\textbf{DistGap} (input control) .................................................. integer \rightarrow long
Minimal distance of two maxima in the Hough image (direction: distance).
\textbf{Default Value: 5}

\textbf{Angle} (output control) .................................................... hesseqline.angle.rad \rightarrow \text{VARIANT( real )}
Angles (in radians) of the detected lines’ normal vectors.
\textbf{Typical Range of Values: \(-1.5707963 \leq \text{Angle} \leq -1.5707963\)}

\textbf{Dist} (output control) ...................................................... hesseline.distance \rightarrow \text{VARIANT( real )}
Distance of the detected lines from the origin.
\textbf{Typical Range of Values: \(-1.5707963 \leq \text{Dist} \leq -1.5707963\)}

\textbf{Parameter Number: \((\text{Dist} = \text{Angle})\)}

\textbf{Result}

The operator \texttt{HoughLines} returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem('noObjectResult',<Result>)}, the behavior in case of empty region is set via \texttt{SetSystem('emptyRegionResult',<Result>)}. If necessary an exception handling is raised.

---

\textbf{Parallelization Information}

\texttt{HoughLines} is \textit{reentrant} and processed \textit{without} parallelization.

---

\textbf{Possible Predecessor Functions}

SelectMatchingLines

---

\textbf{Possible Successor Functions}

See Also

\texttt{HoughLineTrans}, \texttt{GenRegionHline}, \texttt{HoughCircles}

---

\textbf{Module}

\texttt{Region processing}

---

\texttt{[out] HRegionX RegionLines HRegionX.SelectMatchingLines}
\texttt{([in] VARIANT AngleIn, [in] VARIANT DistIn, [in] long LineWidth,}
\texttt{[in] long Thresh, [out] VARIANT AngleOut, [out] VARIANT DistOut )}

\texttt{void HOperatorSetX.SelectMatchingLines ([in] IObjectX RegionIn,}
\texttt{[out] HUntypedObjectX RegionLines, [in] VARIANT AngleIn, [in] VARIANT DistIn,}
\texttt{[in] long LineWidth, [in] long Thresh, [out] VARIANT AngleOut,}
\texttt{[out] VARIANT DistOut )}

\textbf{Select those lines from a set of lines (in \textit{HNF}) which fit best into a region.}

Lines which fit best into a region can be selected from a set of lines which are available in \textit{HNF} with the help of the operator \texttt{SelectMatchingLines}; the region itself is also transmitted as a parameter (\texttt{RegionIn}). The width of the lines can be indicated by the parameter \texttt{LineWidth}. The selected lines will be returned in \textit{HNF} and as regions (\texttt{RegionLines}).

The lines are selected iteratively in a loop: At first, the line showing the greatest overlap with the input region is selected from the set of input lines. This line will then be taken over into the output set whereby all points belonging to that line will not be considered in the further steps determining overlaps. The loop will be left when the maximum overlap value of the region and the lines falls below a certain threshold value (\texttt{Thresh}). The selected lines will be returned as regions as well as in \textit{HNF}. 

---
The operator SelectMatchingLines returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem('noObjectResult',<Result>), the behavior in case of empty region is set via SetSystem('emptyRegionResult',<Result>). If necessary an exception handling is raised.

### Parallelization Information

SelectMatchingLines is reentrant and processed without parallelization.

### Possible Predecessor Functions

**HoughLines Module**

### 12.9 Kalman-Filter

Estimate the current state of a system with the help of the Kalman filtering.

The operator FilterKalman returns an estimate of the current state (or also a prediction of a future state) of a discrete, stochastically disturbed, linear system. In practice, Kalman filters are used successfully in image processing in the analysis of image sequences (background identification, lane tracking with the help of line tracing or region analysis, etc.). A short introduction concerning the theory of the Kalman filters will be followed by a detailed description of the routine FilterKalman itself.

**KALMAN FILTER:** A discrete, stochastically disturbed, linear system is characterized by the following markers:
... State \( x(t) \): Describes the current state of the system (speeds, temperatures,...).
- Parameter \( u(t) \): Inputs from outside into the system.
- Measurement \( y(t) \): Measurements gained by observing the system. They indicate the state of the system (or at least parts of it).
- An output function describing the dependence of the measurements on the state.
- A transition function indicating how the state changes with regard to time, the current value and the parameters.

The output function and the transition function are linear. Their application can therefore be written as a multiplication with a matrix.

The transition function is described with the help of the transition matrix \( A(t) \) and the parameter matrix \( G(t) \), the initial function is described by the measurement matrix \( C(t) \). Hereby \( C(t) \) characterizes the dependency of the new state on the old, \( G(t) \) indicates the dependency on the parameters. In practice it is rarely possible (or at least too time consuming) to describe a real system and its behaviour in a complete and exact way. Normally only a relatively small number of variables will be used to simulate the behaviour of the system. This leads to an error, the so called system error (also called system disturbance) \( v(t) \).

The output function, too, is usually not exact. Each measurement is faulty. The measurement errors will be called \( w(t) \). Therefore the following system equations arise:

\[
x(t + 1) = A(t)x(t) + G(t)u(t) + v(t)
\]
\[
y(t) = c(t)x(t) + w(t)
\]

The system error \( v(t) \) and the measurement error \( w(t) \) are not known. As far as systems are concerned which are interpreted with the help of the Kalman filter, these two errors are considered as Gaussian distributed random vectors (therefore the expression ”stochastically disturbed systems”). Therefore the system can be calculated, if the corresponding expected values for \( v(t) \) and \( w(t) \) as well as the covariance matrices are known.

The estimation of the state of the system is carried out in the same way as in the Gaussian-Markov-estimation. However, the Kalman filter is a recursive algorithm which is based only on the current measurements \( y(t) \) and the latest state \( x(t) \). The latter implicitly also includes the knowledge about earlier measurements.

A suitable estimate value \( x(t) \), which is interpreted as the expected value of a random variable for \( x(0) \), must be indicated for the initial value \( x(0) \). This variable should have an expected error value of 0 and the covariance matrix \( P(0) \) which also has to be indicated. At a certain time \( t \) the expected values of both disturbances \( v(t) \) and \( w(t) \) should be 0 and their covariances should be \( Q(t) \) and \( R(t) \). \( x(t) \), \( v(t) \) and \( w(t) \) will usually be assumed to be not correlated (any kind of noise-process can be modelled - however the development of the necessary matrices by the user will be considerably more demanding). The following conditions must be met by the searched estimate values \( x(t) \):

- The estimate values \( x(t) \) are linearly dependent on the actual value \( x(t) \) and on the measurement sequence \( y(0), y(1), \cdots, y(t) \).
- \( x(t) \) being hereby considered to meet its expectations, i.e. \( E x(t) = E x(t) \).
- The grade criterion for \( x(t) \) is the criterion of minimal variance, i.e. the variance of the estimation error defined as \( x(t) - x(t) \), being as small as possible.

After the initialization

\[
x(0) = x_0, P(0) = P_0
\]

at each point in time \( t \) the Kalman filter executes the following calculation steps:

\[
(K - III) \quad K(t) = \frac{P(t)C(t)}{C(t)P(t)C(t) + R(t)}
\]
\[
(K - IV) \quad x(t) = x(t) + K(t)[y(t) - C(t)x(t)]
\]
\[
(K - V) \quad P(t) = P(t) - K(t)C(t)P(t)
\]
\[
(K - I) \quad x(t + 1) = A(t)x(t) + G(t)u(t)
\]
\[
(K - II) \quad P(t + 1) = A(t)P(t)A(t)' + Q(t)
\]

Hereby \( P(t) \) is the covariance matrix of the estimation error, \( x(t) \) is the extrapolation value respective the prediction value of the state, \( P(t) \) are the covariances of the prediction error \( x - x \), \( K \) is the amplifier matrix (the so called Kalman gain), and \( X' \) is the transposed of a matrix \( X \).
Please note that the prediction of the future state is also possible with the equation (K-I). Sometimes this is very useful in image processing in order to determine "regions of interest" in the next image.

As mentioned above, it is much more demanding to model any kind of noise processes. If for example the system noise and the measurement noise are correlated with the corresponding covariance matrix \( L \), the equations for the Kalman gain and the error covariance matrix have to be modified:

\[
\begin{align*}
(K - II) & \quad K(t) = \frac{P(t)C^*(t) + L(t)}{C(t)P(t)C^*(t) + L(t)} \\
(K - V) & \quad \hat{P}(t) = (P(t) - K(t)C(t)P(t))P(t) - K(t)L(t)
\end{align*}
\]

This means that the user himself has to establish the linear system equations from (K-I) up to (K-V) with respect to the actual problem. The user must therefore develop a mathematical model upon which the solution to the problem can be based. Statistical characteristics describing the inaccuracies of the system as well as the measurement errors, which are to be expected, thereby have to be estimated if they cannot be calculated exactly. Therefore the following individual steps are necessary:

1. Developing a mathematical model
2. Selecting characteristic state variables
3. Establishing the equations describing the changes of these state variables and their linearization (matrices \( A \) and \( G \))
4. Establishing the equations describing the dependency of the measurement values of the system on the state variables and their linearization (matrix \( C \))
5. Developing or estimating of statistical dependencies between the system disturbances (matrix \( Q \))
6. Developing or estimating of statistical dependencies between the measurement errors (matrix \( R \))
7. Initialization of the initial state

As mentioned above, the initialization of the system (point 7) hereby necessitates to indicate an estimate \( x_0 \) of the state of the system at the time 0 and the corresponding covariance matrix \( P_0 \). If the exact initial state is not known, it is recommendable to set the components of the vector \( x_0 \) to the average values of the corresponding range, and to set high values for \( P_0 \) (about the size of the squares of the range). After a few iterations (when the number of the accumulated measurement values in total has exceeded the number of the system values), the values which have been determined in this way are also useable.

If on the other hand the initial state is known exactly, all entries for \( P_0 \) have to be set to 0, because \( P_0 \) describes the covariances of the error between the estimated value \( x_0 \) and the actual value \( x(0) \).

THE FILTER ROUTINE:

A Kalman filter is dependent on a range of data which can be organized in four groups:

**Model parameter:** transition matrix \( A \), control matrix \( G \) including the parameter \( u \) and the measurement matrix \( C \)

**Model stochastic:** system-error covariance matrix \( Q \), system-error - measurement-error covariance matrix \( L \), and measurement-error covariance matrix \( R \)

**Measurement vector:** \( y \)

**History of the system:** extrapolation vector \( \hat{x} \) and extrapolation-error covariance matrix \( \hat{P} \)

Thereby many systems can work without input "from outside", i.e. without \( G \) and \( u \). Further, system errors and measurement errors are normally not correlated (\( L \) is dropped).

Actually the data necessary for the routine will be set by the following parameters:

**Dimension:** This parameter includes the dimensions of the status vector, the measurement vector and the controller vector. \( \text{Dimension} \) thereby is a vector \([n,m,p]\), whereby \( n \) indicates the number of the state variables, \( m \) the number of the measurement values and \( p \) the number of the controller members. For a system without determining control (i.e. without influence "from outside") therefore \([n,m,0]\) has to be passed.

**Model:** This parameter includes the lined up matrices (vectors) \( A,C,Q,G,u \) and (if necessary) \( L \) having been stored in row-major order. \( \text{Model} \) therefore is a vector of the length \( n \times n + n \times m + n \times n + n \times p + [n \times m] \). The last summand is dropped, in case the system errors and measurement errors are not correlated, i.e. there is no value for \( L \).
12.9. KALMAN-FILTER

Measurement: This parameter includes the matrix $R$ which has been stored in row-major order, and the measurement vector $y$ lined up. Measurement therefore is a vector of the dimension $m \times m + m$.

PredictionIn / PredictionOut: These two parameters include the matrix $\hat{P}$ (the extrapolation-error covariance matrix) which has been stored in row-major order and the extrapolation vector $\hat{x}$ lined up. This means, they are vectors of the length $n \times n + n$. PredictionIn therefore is an input parameter, which must contain $\hat{P}(t)$ and $\hat{x}(t)$ at the current time $t$. With PredictionOut the routine returns the corresponding predictions $\hat{P}(t + 1)$ and $\hat{x}(t + 1)$.

Estimate: With this parameter the routine returns the matrix $\hat{P}$ (the estimation-error covariance matrix) which has been stored in row-major order and the estimated state $\hat{x}$ lined up. Estimate therefore is a vector of the length $n \times n + n$.

Please note that the covariance matrices $(Q, R, P, \hat{P})$ must of course be symmetric.

Parameter

- **Dimension** (input control) .................. integer ~ VARIANT (integer)
  The dimensions of the state vector, the measurement and the controller vector.
  Default Value : [3,1,0]
  Typical Range of Values : $0 \leq \text{Dimension} \leq 0$

- **Model** (input control) ..................... real ~ VARIANT (real)
  The lined up matrices $A, C, Q$, possibly $G$ and $u$, and if necessary $L$ which have been stored in row-major order.
  Default Value : [1.0,1.0,0.5,0.0,1.0,1.0,0.0,0.0,0.10,1.0,0.0,0.0,0.54,3.37,9.48,0.37,9.34,3.42,5.48,0.42,5.43,7]
  Typical Range of Values : $0.0 \leq \text{Model} \leq 0.0$

- **Measurement** (input control) ............. real ~ VARIANT (real)
  The matrix $R$ stored in row-major order and the measurement vector $y$ lined up.
  Default Value : [1.2,1.0]
  Typical Range of Values : $0.0 \leq \text{Measurement} \leq 0.0$

- **PredictionIn** (input control) .............. real ~ VARIANT (real)
  The matrix $P$ (the extrapolation-error covariances) stored in row-major order and the extrapolation vector $\hat{x}$ lined up.
  Default Value : [0.00,0.00,0.00,0.180.5,0.00,0.0,100.0,0.0,0,100.0,0.0]
  Typical Range of Values : $0.0 \leq \text{PredictionIn} \leq 0.0$

- **PredictionOut** (output control) .......... real ~ VARIANT (real)
  The matrix $P\hat{s}$ (the extrapolation-error covariances) stored in row-major order and the extrapolation vector $\hat{x}$ lined up.

- **Estimate** (output control) ................. real ~ VARIANT (real)
  The matrix $\hat{P}$ (the estimation-error covariances) stored in row-major order and the estimated state $\hat{x}$ lined up.

Example

// Typical procedure:
// 1. To initialize the variables, which describe the model, e.g. with read_kalman(‘kalman.init’, Dim, Mod, Meas, Pred)
// Generation of the first measurements (typical of the first image of an
// image series) with an appropriate problem-specific routine (there is a
// fictitious routine extract_features in example):
extract_features(Image1, Meas, Meas1)
// first Kalman-Filtering:
filter_kalman(Dim, Mod, Meas1, Pred, Pred1, Est1)
// To use the estimate value (if need be the prediction too)
// with a problem-specific routine (here use_est):
use_est(Est1)
// To get the next measurements (e.g. from the next image):
extract_next_features(Image2, Meas1, Meas2)
// if need be Update of the model parameter (a constant model)
// second Kalman-Filtering:

HALCON 6.0
filter_kalman(Dim,Mod,Meas2,Pred1,Pred2,Est2)
use_est(Est2)
exttract_next_features(Image3,Meas2,Meas3).
// etc.

___ Result 
If the parameter values are correct, the operator FilterKalman returns the value TRUE. Otherwise an exception handling will be raised.

___ Parallelization Information 
FilterKalman is reentrant and processed without parallelization.

___ Possible Predecessor Functions 
ReadKalman, SensorKalman

___ Possible Successor Functions 
UpdateKalman

___ See Also 
ReadKalman, UpdateKalman, SensorKalman

___ Bibliography 
W. Hartinger: "'Entwurf eines anwendungsunabhängigen Kalman-Filters mit Untersuchungen im Bereich der Bildfolgenanalyse''; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

___ Module 
Tools

```
[out] VARIANT Dimension HMiscX.ReadKalman ([in] String FileName,
```

```
void HOperatorSetX.ReadKalman ([in] String FileName,
[out] VARIANT Dimension, [out] VARIANT Model, [out] VARIANT Measurement,
[out] VARIANT Prediction )
```

Read the description file of a Kalman filter.
The operator ReadKalman reads the description file FileName of a Kalman filter. Kalman filters return an estimate of the current state (or even the prediction of a future state) of a discrete, stochastically disturbed, linear system. They are successfully used in image processing, especially in the analysis of image sequences. A Kalman filtering is based on a mathematical model of the system to be examined which at any point in time has the following characteristics:

**Model parameter:** transition matrix $A$, control matrix $G$ including the controller output $u$ and the measurement matrix $C$

**Model stochastic:** system-error covariance matrix $Q$, system-error - measurement-error covariance matrix $L$ and measurement-error covariance matrix $R$

**Estimate of the initial state of the system:** state $x_0$ and corresponding covariance matrix $P_0$

Many systems do not need entries "’from outside’’, and therefore $G$ and $u$ can be dropped. Further, system errors and measurement errors are normally not correlated ($L$ is dropped). The characteristics mentioned above can be stored in an ASCII-file and then can be read with the help of the operator ReadKalman. This ASCII-file must have the following structure:
Dimension row
+ content row
+ matrix A
+ matrix C
+ matrix Q
[ + matrix G + vector u ]
[ + matrix L ]
+ matrix R
[ + matrix F0 ]
[ + vector x0 ]

The dimension row thereby is always of the following form:
\[ n = \text{integer} \quad m = \text{integer} \quad p = \text{integer} \]
whereby \( n \) indicates the number of the state variables, \( m \) the number of the measurement values and \( p \) the number of the controller members (see also Dimension). The maximal dimension will hereby be limited by a system constant (= 30 for the time being).

The content row has the following form:
\[ A \times C \times Q \times G \times u \times L \times R \times P \times x \]
and describes the following content of the file. Instead of `\( ,\)` , `\( +\)` (= parameter is available) respectively `\( -\)` (= parameter is missing) have to be set. Please note that only the parameters marked by `[ ... ]` in the above list may be left out in the description file. If the initial state estimate \( a_0 \) is missing (i.e. `x-`), the components of the vector will supposed to be 0.0. If the covariance matrix \( P_0 \) of the initial state estimate is missing (i.e. `P-`), the error will be supposed to be tremendous. In this case the matrix elements will be set to 10000.0. This value seems to be very high, however, it is only sufficient if the range of components of the state vector \( x \) is smaller to the tenth power.

\((r \times s)\) matrices will be stored per row in the following form:
\[
\begin{bmatrix}
< \text{Komentar, d.h. string} > \\
< a_{11} > & < a_{12} > & \cdots & < a_{1s} > \\
\vdots & \ddots & \vdots \\
< a_{r1} > & < a_{r2} > & \cdots & < a_{rs} > \\
\end{bmatrix}
\]
(the spaces and line feed characters can be chosen at will),

vectors will be stored correspondingly in the following form:
\[
\begin{bmatrix}
< \text{comment, i.e.string} > \\
< a_1 > & \cdots & < a_k > \\
\end{bmatrix}
\]

The following parameter values are returned by the operator ReadKalman:

**Dimension:** This parameter includes the dimensions of the status vector, the measurement vector and the controller vector. **Dimension** thereby is a vector \([n,m,p]\), whereby \( n \) indicates the number of the state variables, \( m \) the number of the measurement values and \( p \) the number of the controller members. For a system without determining control (i.e. without influence “from outside”) therefore Dimension = \([n,m,0]\).

**Model:** This parameter includes the lined up matrices (vectors) \( A, C, Q, G, u \) and (if necessary) \( L \) having been stored in row-major order. **Model** therefore is a vector of the length \( n \times n + n \times m + n \times n + n \times p + p[+n \times m] \).

The last summand is dropped, in case the system errors and measurement errors are not correlated, i.e. there is no value for \( L \).

**Measurement:** This parameter includes the matrix \( R \) which has been stored in row-major order. **Measurement** therefore is vector of the dimension \( m \times m \).

**Prediction:** This parameter includes the matrix \( F_0 \) (the error covariance matrix of the initial state estimate) and the initial state estimate \( x_0 \) lined up. This means, it is a vector of the length \( n \times n + n \).

**Attention**
**Parameter**

- **FileName (input control)** .......................... filename
  Description file for a Kalman filter.
  Default Value: ’kalman.init’

- **Dimension (output control)** ....................... integer
  The dimensions of the state vector, the measurement vector and the controller vector.

- **Model (output control)** ............................ real
  The lined up matrices $A, C, Q$, possibly $G$ and $u$, and if necessary $L$ stored in row-major order.

- **Measurement (output control)** .................... real
  The matrix $R$ stored in row-major order.

- **Prediction (output control)** ........................ real
  The matrix $P_0$ (error covariance matrix of the initial state estimate) stored in row-major order and the initial state estimate $x_0$ lined up.

**Example**

```plaintext
% An example of the description-file:
% n=3 m=1 p=0
% A+C+Q+G-u-L+R+P+x+
% transition matrix A:
% 1 1 0.5
% 0 1 1
% 0 0 1
% measurement matrix C:
% 1 0 0
% system-error covariance matrix Q:
% 54.3 37.9 48.0 
% 37.9 34.3 42.5 
% 48.0 42.5 43.7 
% measurement-error covariance matrix R:
% 1.2 
% estimation-error covariance matrix (for the initial estimate) P0:
% 0 0 0 
% 0 180.5 0 
% 0 0 100
% initial estimate x0:
% 0 100 0 
% the result of read_kalman with the upper descriptionfile
% as input parameter:
%
% Dimension = [3,1,0]
% Model = [1.0,1.0,0.5,0.0,1.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
% Measurement = [1.2]
% Prediction = [0.0,0.0,0.0,0.0,180.5,0.0,0.0,0.0,100.0,0.0,100.0,0.0,100.0,0.0,100.0,0.0,100.0].
```

**Result**

If the description file is readable and correct, the operator ReadKalman returns the value TRUE. Otherwise an exception handling will be raised.

**Parallelization Information**

ReadKalman is reentrant and processed without parallelization.

**Possible Successor Functions**

FilterKalman

**See Also**

UpdateKalman, FilterKalman, SensorKalman
Interactive input of measurement values for a Kalman filtering.

The operator SensorKalman supports the interactive input of measurement values for a Kalman filtering. Kalman filters return an estimate of the current state (or even the prediction of a future state) of a discrete, stochastically disturbed, linear system. They are successfully used in image processing, especially in the analysis of image sequences.

Each filtering is hereby based on certain measurement values. How these values are extracted from images or sensor data depends strongly on the individual application and therefore must be entirely up to the user. However, the operator SensorKalman allows an interactive input of (fictitious) measurement values and the corresponding measurement-error covariance matrix $R$. Especially the testing of Kalman filters during the development can hereby be facilitated.

The parameters MeasurementIn and MeasurementOut include the matrix $R$ which has been stored in row-major order and the measurement vector $y$ lined up, i.e. they are vectors of the length $\text{Dimension} \times \text{Dimension} + \text{Dimension}$.

Attention

Parameter

- **Dimension** (input control) ........................................integer $\sim$ long
  Number of measurement values.
  Default Value : 1
  Typical Range of Values : $0 \leq \text{Dimension} \leq 0$

- **MeasurementIn** (input control) ..................................real $\sim$ VARIANT( real )
  The matrix $R$ stored in row-major order and the measurement vector $y$ lined up.
  Default Value : [1,2,1.0]
  Typical Range of Values : $0.0 \leq \text{MeasurementIn} \leq 0.0$

- **MeasurementOut** (output control) ...............................real $\sim$ VARIANT( real )
  The matrix $R$ stored in row-major order and the measurement vector $y$ lined up.

Result

If the parameters are correct, the operator SensorKalman returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

SensorKalman is reentrant and processed without parallelization.

Possible Successor Functions

FilterKalman

See Also

FilterKalman, ReadKalman, UpdateKalman

Module

Tools
Read an update file of a Kalman filter.

The operator `UpdateKalman` reads the update file `FileName` of a Kalman filter. Kalman filters return an estimate of the current state (or even the prediction of a future state) of a discrete, stochastically disturbed, linear system.

A Kalman filtering is based on a mathematical model of the system to be examined which at any point in time has the following characteristics:

Model parameter: transition matrix $A$, control matrix $G$ including the controller output $u$ and the measurement matrix $C$

Model stochastic: system-error covariance matrix $Q$, system-error - measurement-error covariance matrix $L$ and measurement-error covariance matrix $R$

Measurement vector: $y$

History of the system: extrapolation vector $\hat{x}$ and extrapolation-error covariance matrix $\hat{P}$

Many systems do not need entries "from outside" and therefore $G$ and $u$ can be dropped. Further, system errors and measurement errors are normally not correlated ($L$ is dropped). Some of the characteristics mentioned above may change dynamically (from one iteration to the next). The operator `UpdateKalman` serves to modify parts of the system according to an update file (ASCII) with the following structure (see also `ReadKalman`):

```
Dimension row + content row + matrix A + matrix C + matrix Q + matrix $G$ + vector $u$ + matrix $L$ + matrix $R$
```

The dimension row thereby has the following form:

$$n = <\text{integer}> \quad m = <\text{integer}> \quad p = <\text{integer}>$$

whereby $n$ indicates the number of the state variables, $m$ the number of the measurement values and $p$ the number of the controller members (see also `DimensionIn`/`DimensionOut`). The maximal dimension will hereby be limited by a system constant ($= 30$ for the time being). As in this case changes should take effect at a valid model, the dimensions $n$ and $m$ are invariant (and will only be indicated for purposes of control).

The content row has the following form:

$$A \times C \times Q \times G \times u \times L \times R$$

and describes the further content of the file. Instead of `$+$` ("parameter is available") respectively `$-$` ("parameter is missing") has to be set. In contrast to description files for `ReadKalman`, the system description needs not be complete in this case. Only those parts of the system which are changed must be indicated. The indication of estimated values is unnecessary, as these values must stem from the latest filtering according to the structure of the filter.

$(r \times s)$ matrices will be stored in row-major order in the following form:

```
< comment, i.e. string >
< a_{11} > < a_{12} > \ldots < a_{1s} >
< a_{r1} > < a_{r2} > \ldots < a_{rs} >
```

vectors will be stored correspondingly in the following form:

\[
\begin{align*}
&\text{\textit{comment}, i.e. \textit{string}} > \\
&< a_1 > \cdots < a_k >
\end{align*}
\]

The following parameter values of the operator \texttt{ReadKalman} will be changed:

\textbf{DimensionIn/DimensionOut}: These parameters include the dimensions of the state vector, measurement vector and controller vector and therefore are vectors \([n,m,p]\), whereby \(n\) indicates the number of the state variables, \(m\) the number of the measurement values and \(p\) the number of the controller members. \(n\) and \(m\) are invariant for a given system, i.e. they must not differ from corresponding input values of the update file. For a system without influence ”from outside” \(p = 0\).

\textbf{ModelIn/ModelOut}: These parameters include the lined up matrices (vectors) \(A, C, Q, G, u\) and if necessary \(L\) which have been stored in row-major order. \textbf{ModelIn/ModelOut} therefore are vectors of the length \(n \times n + n \times m + n \times n \times p + p^2 + n \times m\). The last summand is dropped if system errors and measurement errors are not correlated, i.e. no value has been set for \(L\).

\textbf{MeasurementIn/MeasurementOut}: These parameters include the matrix \(R\) stored in row-major order, and therefore are vectors of the dimension \(m \times m\).

\begin{verbatim}
%The following values are describing the system
%DimensionIn  = [3,1,0]
%ModelIn     = [1.0,1.0,0.5,0.0,1.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,
%               54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
%MeasurementIn = [1,2]
%An example of the Updatefile:
\end{verbatim}
% n=3 m=1 p=0
% A+C-Q-G-u-L-R-
% transitions at time t=15:
% 2 1 1
% 0 2 2
% 0 0 2
%
% the results of update_kalman:
%
% DimensionOut = [3,1,0]
% ModelOut = [2.0,1.0,1.0,0.0,2.0,2.0,0.0,0.0,2.0,1.0,0.0,0.0, 54.3, 37.9, 48.0, 37.9, 34.3, 42.5, 48.0, 42.5, 43.7]
% MeasurementOut = [1.2]

If the update file is readable and correct, the operator UpdateKalman returns the value TRUE. Otherwise an exception handling is raised.

UpdateKalman is reentrant and processed without parallelization.

Possible Successor Functions

FilterKalman

See Also

ReadKalman, FilterKalman, SensorKalman

Tools

12.10 Matching

void HOperatorSetX.ClearShapeModel ([in] long ModelID )

Free the memory of a shape model.

The operator ClearShapeModel frees the memory of a shape model which has been created by CreateShapeModel. After execution of the operator ClearShapeModel the model can no longer be used. The handle ModelID becomes invalid.

Parameter

ModelID (input control) ................... shape_model ~ HShapeModelX / long Handle of the model.

Result

If the handle of the model is valid, the operator ClearShapeModel returns the value TRUE. If necessary an exception handling is raised.

Parallelization Information

ClearShapeModel is processed completely exclusively without parallelization.

Possible Predecessor Functions

CreateShapeModel, ReadShapeModel, WriteShapeModel

Module

Template matching
Prepare a shape model for matching.

The operator CreateShapeModel prepares a template, which is passed in the image Template, as a shape model used for matching.

The model is generated using multiple image pyramid levels and multiple rotations and is stored in memory. The output parameter ModelID is a handle for this model, which is used in subsequent calls to FindShapeModel.

The number of pyramid levels is determined with the parameter NumLevels. It should be chosen as large as possible because by this the time necessary to find the object is significantly reduced. On the other hand, NumLevels must be chosen such that the model is still recognizable and contains a sufficient number of points (at least four) on the highest pyramid level. This can be checked using the output of InspectShapeModel. If not enough model points are generated, CreateShapeModel returns with an error message.

The parameters AngleStart and AngleExtent determine the range of possible rotations, in which the model can occur in the image. Note that the model can only be found in this range of angles by FindShapeModel. The parameter AngleStep determines the step length within the selected range of angles. Hence, if subpixel accuracy is not specified in FindShapeModel, this parameter specifies the accuracy that is achievable for the angles in FindShapeModel. AngleStep should be chosen based on the size of the object. Smaller models do not possess many (discrete) rotations in the image, and hence AngleStep should be chosen larger for smaller models. If AngleExtent is not an integer multiple of AngleStep, AngleStep is modified accordingly. The model is pre-generated for the selected angle range and stored in memory. The memory required to store the model is proportional to the number of angle steps and the number of points in the model. Hence, if AngleStep is too small or AngleExtent too big, it may happen that the model no longer fits into the (virtual) memory. In this case, either AngleStep must be enlarged or AngleExtent must be reduced. In any case, it is desirable that the model completely fits into the main memory, because this avoids paging by the operating system, and hence the time to find the object will be much smaller. Since angles can be determined with subpixel resolution by FindShapeModel, AngleStep ≥ 1 can be selected for models of a diameter smaller than 300 pixels.

For particularly large models, it may be useful to reduce the number of model points by setting Optimization to a value different from 'none'. If Optimization = 'none', all model points are stored. In all other cases, the number of points is reduced according to the value of Optimization. If the number of points is reduced, it may be necessary in FindShapeModel to set the parameter Greediness to a smaller value, e.g., 0.7 or 0.8. For small models, the reduction of the number of model points does not result in a speed-up of the search because in this case usually significantly more potential instances of the model must be examined.

The parameter Contrast determines, which contrast the model points must have. The contrast is a measure for local gray value differences between the object and the background and between different parts of the object. Contrast should be chosen such that only the significant features of the template are used for the model. The effect of this parameter can be checked in advance with InspectShapeModel.

With MinContrast, it can be determined which contrast the model must at least have in the recognition performed by FindShapeModel. In other words, this parameter separates the model from the noise in the image. Therefore, a good choice is the range of gray value changes caused by the noise in the image. If, for example, the gray values fluctuate within a range of 10 gray levels, MinContrast should be set to 10. Obviously, MinContrast must be smaller than Contrast. If the model should be recognized in very low contrast images, MinContrast must be set to a correspondingly small value. If the model should be recognized even if it is severely occluded, MinContrast should be slightly larger than the range of gray value fluctuations created by noise in order to ensure that the position and rotation of the model are extracted robustly and accurately by FindShapeModel.


The parameter `Metric` determines the conditions under which the model is recognized in the image. If `Metric = 'use_polarity'`, the object in the image and the model must have the same contrast. If, for example, the model is a bright object on a dark background, the object is found only if it is also brighter than the background. If `Metric = 'ignore_global_polarity'`, the object is found in the image also if the the contrast reverses globally. In the above example, the object hence is also found if it is darker than the background. The runtime of `FindShapeModel` will increase slightly in this case. If `Metric = 'ignore_local_polarity'`, the model is found even if the contrast changes locally. This mode can, for example, be useful if the object consists of a part with medium gray value, within which either darker or brighter sub-objects lie. Since in this case the runtime of `FindShapeModel` increases significantly, it is usually better to create several models that reflect the possible contrast variations of the object with `CreateShapeModel`, and to match them separately with `FindShapeModel`.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Template</code></td>
<td>(input iconic) image ~ <code>HImageX / IHObjectX (byte)</code> Input image whose domain will be used to create the model.</td>
</tr>
<tr>
<td><code>NumLevels</code></td>
<td>(input control) integer ~ <code>long</code> Maximum number of pyramid levels. Default Value: 4 Value Suggestion: <code>NumLevels ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}</code></td>
</tr>
<tr>
<td><code>AngleStart</code></td>
<td>(input control) angle.rad ~ <code>double</code> Smallest rotation of the pattern. Default Value: -0.39 Value Suggestion: <code>AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}</code></td>
</tr>
<tr>
<td><code>AngleExtent</code></td>
<td>(input control) angle.rad ~ <code>double</code> Extent of the rotation angles. Default Value: 0.79 Value Suggestion: <code>AngleExtent ∈ {6.28, 3.14, 1.57, 0.79, 0.39}</code> Restriction: <code>(AngleExtent &gt; 0)</code></td>
</tr>
<tr>
<td><code>AngleStep</code></td>
<td>(input control) angle.rad ~ <code>double</code> Step length of the angles (resolution). Default Value: 0.0982 Value Suggestion: <code>AngleStep ∈ {0.3927, 0.1963, 0.0982, 0.0491, 0.0245}</code> Restriction: <code>(AngleStep &gt; 0)</code></td>
</tr>
<tr>
<td><code>Optimization</code></td>
<td>(input control) string ~ <code>String</code> Kind of optimization. Default Value: 'none' Value Suggestion: <code>Optimization ∈ {'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}</code></td>
</tr>
<tr>
<td><code>Metric</code></td>
<td>(input control) string ~ <code>String</code> Match metric. Default Value: 'use_polarity' Value Suggestion: <code>Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity'}</code></td>
</tr>
<tr>
<td><code>Contrast</code></td>
<td>(input control) number ~ <code>long</code> Contrast of the object in the template image. Default Value: 60 Value Suggestion: <code>Contrast ∈ {20, 40, 60, 80, 100, 120, 140, 160}</code></td>
</tr>
<tr>
<td><code>MinContrast</code></td>
<td>(input control) number ~ <code>long</code> Minimum contrast of the objects in the search images. Default Value: 10 Value Suggestion: <code>MinContrast ∈ {10, 20, 40}</code> Restriction: <code>(MinContrast &lt; Contrast)</code></td>
</tr>
<tr>
<td><code>ModelID</code></td>
<td>(output control) shape_model ~ <code>HShapeModelX / long</code> Handle of the model.</td>
</tr>
</tbody>
</table>

### Result

If the parameters are valid, the operator `CreateShapeModel` returns the value `TRUE`. If necessary an exception handling is raised. If the parameters `NumLevels` and `Contrast` are chosen such that the model contains too few points, the error `8510` is raised.

### Parallelization Information

CreateShapeModel is processed completely exclusively without parallelization.
Find the best matches of a shape model in an image.

The operator FindShapeModel finds the best NumMatches instances of the shape model ModelID in the input image Image. The model must have been created previously by calling CreateShapeModel or ReadShapeModel.

The position and rotation of the found instances of the model is returned in Row, Column and Angle. Additionally, the score of each found instance is returned in Score. The score is a number between 0 and 1, which is an approximate measure of how much of the model is visible in the image. If, for example, half of the model is occluded, the score cannot exceed 0.5.

The model is searched within those points of the domain of the image, in which the model lies completely within the image. This means that the model will not be found if it extends beyond the borders of the image, even if it would achieve a score greater than MinScore (see below). The parameters AngleStart and AngleExtent determine the range of rotations for which the model is searched. If necessary, the range of rotations is clipped to the range given when the model was created with CreateShapeModel.

The parameter MinScore determines what score a potential match must at least have to be regarded as an instance of the model in the image. The larger MinScore is chosen, the faster the search is. If the model can be expected never to be occluded in the images, MinScore may be set as high as 0.8 or even 0.9.

The maximum number of instances to be found can be determined with NumMatches. If more than NumMatches instances with a score greater than MinScore are found in the image, only the best NumMatches instances are returned. If fewer than NumMatches are found, only that number is returned, i.e., the parameter MinScore takes precedence over NumMatches.

If the model exhibits symmetries it may happen that multiple instances with similar positions but different rotations are found in the image. The parameter MaxOverlap determines by what fraction (i.e., a number between 0 and 1) two instances may at most overlap in order to consider them as different instances, and hence to be returned separately. If two instances overlap each other by more than MaxOverlap only the best instance is returned. The calculation of the overlap is based on the smallest enclosing rectangle of arbitrary orientation (see SmallestRectangle2) of the found instances. If MaxOverlap = 0, the found instances may not overlap at all, while for MaxOverlap = 1 all instances are returned.
The parameter SubPixel determines whether the instances should be extracted with subpixel accuracy. If SubPixel is set to 'true', the position as well as the rotation is determined with subpixel accuracy.

The number of pyramid levels used during the search is determined with NumLevels. If necessary, the number of levels is clipped to the range given when the template was created with CreateShapeModel.

The parameter Greediness determines how “greedily” the search should be carried out. If Greediness = 0, a safe search heuristic is used, which always finds the model if it is visible in the image. However, the search will be relatively time consuming in this case. If Greediness = 1, an unsafe search heuristic is used, which may cause the model not to be found in rare cases, even though it is visible in the image. For Greediness = 1, the maximum search speed is achieved. In almost all cases, the template will always be found for Greediness = 0.9.
12.10. MATCHING

Row (output control) ..................................................... point.y ~ VARIANT (real)
Row coordinate of the found instances of the model.

Column (output control) .................................................. point.x ~ VARIANT (real)
Column coordinate of the found instances of the model.

Angle (output control) ................................................. angle.rad ~ VARIANT (real)
Rotation angle of the found instances of the model.

Score (output control) ..................................................... real ~ VARIANT (real)
Score of the found instances of the model.

Result
If the parameter values are correct, the operator FindShapeModel returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information
FindShapeModel is reentrant and processed without parallelization.

Possible Predecessor Functions
CreateShapeModel, ReadShapeModel

Alternatives
BestMatchRotMg

Module
Template matching

Create the representation of a shape model.
InspectShapeModel creates a representation of a shape model. The operator is particularly useful in order to determine the parameters NumLevels and Contrast, which are used in CreateShapeModel, quickly and conveniently. The representation of the model is created on multiple image pyramid levels, where the number of levels is determined by NumLevels. In contrast to CreateShapeModel, the model is only created for the rotation of the object in the input image, i.e., 0°. As output, InspectShapeModel creates an image object ModelImages containing the images of the individual levels of the image pyramid as well as a region in ModelRegions for each pyramid level that represents the model at the respective pyramid level. The individual objects can be accessed with SelectObj. As described for CreateShapeModel, the number of pyramid levels should be chosen as large as possible, while taking into account that the model must be recognizable on the highest pyramid level and must have enough model points. The parameter Contrast should be chosen such that only the significant features of the template object are used for the model. In its typical use, InspectShapeModel is called interactively multiple times with different parameters for NumLevels and Contrast, until a satisfactory model is obtained. After this, CreateShapeModel is called with the parameters thus obtained.

Parameter

Image (input iconic) ................................. image ~ HImageX / IHObjectX (byte)
Input image.

ModelImages (output iconic) ..................... image ~ HImageX / HUntypedObjectX (byte)
Image pyramid of the input image

ModelRegions (output iconic) ...................... region ~ HRegionX / HUntypedObjectX
Model region pyramid

NumLevels (input control) .......................... integer ~ long
Number of pyramid levels.
Default Value: 4
Value List: NumLevels ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Contrast (input control) .......................................................... number \( \sim \) long
Contrast of the object in the image.

Default Value: 60

Value Suggestions: \( \text{Contrast} \in \{ 20, 40, 60, 80, 100, 120, 140, 160 \} \)

If the parameters are valid, the operator InspectShapeModel returns the value TRUE. If necessary an exception handling is raised.

Result

InspectShapeModel is reentrant and processed without parallelization.

Parallelization Information

Possible Predecessor Functions

ReduceDomain

Possible Successor Functions

SelectObj

See Also

CreateShapeModel

Template matching

void \texttt{HShapeModelX.ReadShapeModel} ([in] String FileName )

void \texttt{HOperatorSetX.ReadShapeModel} ([in] String FileName, [out] long ModelID )

Read a shape model from a file.
The operator ReadShapeModel reads a shape model, which has been written with WriteShapeModel, from the file FileName.

Parameter

\begin{itemize}
  \item \textbf{FileName} (input control) .................................. filename \( \sim \) String
  File name.
  \item \textbf{ModelID} (output control) .................................. shape_model \( \sim \) HShapeModelX / long
  Handle of the model.
\end{itemize}

Result

If the file name is valid, the operator ReadShapeModel returns the value TRUE. If necessary an exception handling is raised.

Parallelization Information

ReadShapeModel is processed completely exclusively without parallelization.

Possible Successor Functions

FindShapeModel

See Also

CreateShapeModel, ClearShapeModel

Module

Template matching

void \texttt{HShapeModelX.WriteShapeModel} ([in] String FileName )

void \texttt{HOperatorSetX.WriteShapeModel} ([in] long ModelID, [in] String FileName )

Write a shape model to a file.
The operator WriteShapeModel writes a shape model to the file FileName. The model can be read again with ReadShapeModel.
12.11 Measure

**void** HOperatorSetX.CloseMeasure ([in] long MeasureHandle )

Delete a measure object.

CloseMeasure deletes the measure object given by MeasureHandle. The memory used for the measure object is freed.

**Parameter**

- **MeasureHandle** (input control) ..........................measure_id  \( \sim \) HMeasureX / long
  Measure object handle.

**Result**

If the parameter values are correct the operator CloseMeasure returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

CloseMeasure is reentrant and processed without parallelization.

**Possible Predecessor Functions**

GenMeasureRectangle2, MeasurePos, MeasurePairs

**Module**

Sub-pixel operators

**void** HMeasureX.GenMeasureArc ([in] VARIANT CenterRow,
[in] long Height, [in] String Interpolation )

**void** HOperatorSetX.GenMeasureArc ([in] VARIANT CenterRow,
[in] long Height, [in] String Interpolation, [out] long MeasureHandle )

Prepare the extraction of straight edges perpendicular to an annular arc.

GenMeasureArc prepares the extraction of straight edges which lie perpendicular to an annular arc. Here, annular arc denotes a circular arc with an associated width. The center of the arc is passed in the parameters CenterRow and CenterCol, its radius in Radius, the starting angle in AngleStart, and its angular extent relative to the starting angle in AngleExtent. If AngleExtent > 0, an arc with counterclockwise orientation
is generated, otherwise an arc with clockwise orientation. The radius of the annular arc, i.e., half its width, is
determined by AnnulusRadius.

The edge extraction algorithm is described in the documentation of the operator MeasurePos. As discussed
there, different types of interpolation can be used for the calculation of the one-dimensional gray value profile. For
Interpolation = 'nearest neighbor', the gray values in the measurement are obtained from the gray values
of the closest pixel, i.e., by constant interpolation. For Interpolation = 'bilinear', bilinear interpolation is
used, while for Interpolation = 'bicubic', bicubic interpolation is used.

With GenMeasureRectangle2, all computations which can be used for multiple measurements are removed
from the actual measurement. To effect this, an optimized data structure, a so-called measure object, is constructed
and returned in MeasureHandle. In order to perform the measurement at the optimum speed, the size of the im-
ages for which subsequent measurements will be made must already be specified in GenMeasureRectangle2
with the parameters Width and Height. This technique increases the speed of the actual measurement signifi-
cantly.

Parameter

- **CenterRow** (input control) .......................... point.y 〜 VARIANT ( integer, real )
  Row coordinate of the center of the arc.
  Default Value : 100.0
  Value Suggestions : CenterRow ∈ [10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0]
  Typical Range of Values : 0.0 ≤ CenterRow ≤ 0.0(lin)
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0

- **CenterCol** (input control) .......................... point.x 〜 VARIANT ( integer, real )
  Column coordinate of the center of the arc.
  Default Value : 100.0
  Value Suggestions : CenterCol ∈ [10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0]
  Typical Range of Values : 0.0 ≤ CenterCol ≤ 0.0(lin)
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0

- **Radius** (input control) .......................... number 〜 VARIANT ( integer, real )
  Radius of the arc.
  Default Value : 50.0
  Value Suggestions : Radius ∈ [10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0]
  Typical Range of Values : 0.0 ≤ Radius ≤ 0.0(lin)
  Minimal Value Step : 1.0
  Recommended Value Step : 10.0

- **AngleStart** (input control) ......................... angle.rad 〜 VARIANT ( integer, real )
  Start angle of the arc in radians.
  Default Value : 0.0
  Value Suggestions : AngleStart ∈ [-3.14159265359, -2.35619449019, -1.5707963268, -0.785398163398, 0.0, 0.785398163398, 1.5707963268, 2.35619449019, 3.14159265359]
  Typical Range of Values : -3.14159265359 ≤ AngleStart ≤ -3.14159265359(lin)
  Minimal Value Step : 0.0314159265359
  Recommended Value Step : 0.314159265359

- **AngleExtent** (input control) ......................... angle.rad 〜 VARIANT ( integer, real )
  Angular extent of the arc in radians.
  Default Value : 6.28318530718
  Value Suggestions : AngleExtent ∈ [-6.28318530718, -5.49778714378, -4.71238898038, -3.926990817, -3.14159265359, -2.35619449019, -1.5707963268, -0.785398163398, 0.0, 0.785398163398, 1.5707963268, 2.35619449019, 3.14159265359, 3.926990817, 4.71238898038, 5.49778714378, 6.28318530718]
  Typical Range of Values : -6.28318530718 ≤ AngleExtent ≤ -6.28318530718(lin)
  Minimal Value Step : 0.0314159265359
  Recommended Value Step : 0.314159265359
  Restriction : (AngleExtent ≠ 0.0)
**AnnulusRadius** (input control) ................................. number  \(\sim\) VARIANT (integer, real)
Radius (half width) of the annulus.

Default Value: 10.0
Value Suggestions: \(\text{AnnulusRadius} \in \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0\}\)
Typical Range of Values: \(0.0 \leq \text{AnnulusRadius} \leq 0.0\) (lin)
Minimal Value Step: 1.0
Recommended Value Step: 10.0
Restriction: \((\text{AnnulusRadius} \leq \text{Radius})\)

**Width** (input control) ........................................... extent.x  \(\sim\) long
Width of the image to be processed subsequently.

Default Value: 512
Value Suggestions: \(\text{Width} \in \{128, 160, 192, 256, 320, 384, 512, 640, 768\}\)
Typical Range of Values: \(0 \leq \text{Width} \leq 0\) (lin)
Minimal Value Step: 1
Recommended Value Step: 16

**Height** (input control) ......................................... extent.y  \(\sim\) long
Height of the image to be processed subsequently.

Default Value: 512
Value Suggestions: \(\text{Height} \in \{120, 128, 144, 240, 256, 288, 480, 512, 576\}\)
Typical Range of Values: \(0 \leq \text{Height} \leq 0\) (lin)
Minimal Value Step: 1
Recommended Value Step: 16

**Interpolation** (input control) ................................. string  \(\sim\) String
Type of interpolation to be used.

Default Value: ’nearest_neighbor’
Value List: \{’nearest_neighbor’, ‘bilinear’, ‘bicubic’\}

**MeasureHandle** (output control) ............................ measure_id  \(\sim\) HMeasureX/long
Measure object handle.

---

If the parameter values are correct, the operator GenMeasureArc returns the value TRUE. Otherwise an exception handling is raised.

---

**Parallelization Information**
GenMeasureArc is reentrant and processed without parallelization.

**Possible Predecessor Functions**
DrawCircle
MeasurePos, MeasurePairs

**Possible Successor Functions**

---

**Alternatives**

---

**Module**

---

Prepare the extraction of straight edges perpendicular to a rectangle.

GenMeasureRectangle2 prepares the extraction of straight edges which lie perpendicular to the major axis of a rectangle. The center of the rectangle is passed in the parameters Row and Column, the direction of the major

---

**HMeasureX.GenMeasureRectangle2** ([in] VARIANT Row,
[in] VARIANT Column, [in] VARIANT Phi, [in] VARIANT Length1,
[in] VARIANT Length2, [in] long Width, [in] long Height,
in String Interpolation)

**HOperatorSetX.GenMeasureRectangle2** ([in] VARIANT Row,
in] VARIANT Column, [in] VARIANT Phi, [in] VARIANT Length1,
in] VARIANT Length2, [in] long Width, [in] long Height,
in String Interpolation, [out] long MeasureHandle)
axis of the rectangle in $\Phi$, and the length of the two axes, i.e., half the diameter of the rectangle, in $\text{Length1}$ and $\text{Length2}$.

The edge extraction algorithm is described in the documentation of the operator $\text{MeasurePos}$. As discussed there, different types of interpolation can be used for the calculation of the one-dimensional gray value profile. For $\text{Interpolation} = \text{'nearest neighbor'}$, the gray values in the measurement are obtained from the gray values of the closest pixel, i.e., by constant interpolation. For $\text{Interpolation} = \text{'bilinear'}$, bilinear interpolation is used, while for $\text{Interpolation} = \text{'bicubic'}$, bicubic interpolation is used.

With $\text{GenMeasureRectangle2}$, all computations which can be used for multiple measurements are removed from the actual measurement. To effect this, an optimized data structure, a so-called measure object, is constructed and returned in $\text{MeasureHandle}$. In order to perform the measurement at the optimum speed, the size of the images for which subsequent measurements will be made must already be specified in $\text{GenMeasureRectangle2}$ with the parameters $\text{Width}$ and $\text{Height}$. This technique increases the speed of the actual measurement significantly.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Row}$ (input control) ...</td>
</tr>
<tr>
<td>Row coordinate of the center of the rectangle.</td>
</tr>
<tr>
<td>Default Value : 50.0</td>
</tr>
<tr>
<td>Value Suggestions : $\text{Row} \in {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}$</td>
</tr>
<tr>
<td>Typical Range of Values : $0.0 \leq \text{Row} \leq 0.0$ (lin)</td>
</tr>
<tr>
<td>Minimal Value Step : 1.0</td>
</tr>
<tr>
<td>Recommended Value Step : 10.0</td>
</tr>
</tbody>
</table>

| $\text{Column}$ (input control) | $\text{rectangle2.center.x} \sim \text{VARIANT (integer, real)}$ |
| Column coordinate of the center of the rectangle. |
| Default Value : 100.0 |
| Value Suggestions : $\text{Column} \in \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0\}$ |
| Typical Range of Values : $0.0 \leq \text{Column} \leq 0.0$ (lin) |
| Minimal Value Step : 1.0 |
| Recommended Value Step : 10.0 |

| $\text{Phi}$ (input control) | $\text{rectangle2.angle.rad} \sim \text{VARIANT (integer, real)}$ |
| Angle of longitudinal axis to horizontal (radians). |
| Default Value : 0.0 |
| Value Suggestions : $\text{Phi} \in \{-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097\}$ |
| Typical Range of Values : $-1.178097 \leq \text{Phi} \leq -1.178097$ (lin) |
| Minimal Value Step : 0.001 |
| Recommended Value Step : 0.1 |
| Restriction : $\left((-\pi) < \Phi \leq \pi) \wedge (\Phi \leq \pi)\right)$ |

| $\text{Length1}$ (input control) | $\text{rectangle2.hwidth} \sim \text{VARIANT (integer, real)}$ |
| Half width. |
| Default Value : 200.0 |
| Value Suggestions : $\text{Length1} \in \{3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0, 300.0, 500.0\}$ |
| Typical Range of Values : $0.0 \leq \text{Length1} \leq 0.0$ (lin) |
| Minimal Value Step : 1.0 |
| Recommended Value Step : 10.0 |

| $\text{Length2}$ (input control) | $\text{rectangle2.hheight} \sim \text{VARIANT (integer, real)}$ |
| Half height. |
| Default Value : 100.0 |
| Value Suggestions : $\text{Length2} \in \{1.0, 2.0, 3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0\}$ |
| Typical Range of Values : $0.0 \leq \text{Length2} \leq 0.0$ (lin) |
| Minimal Value Step : 1.0 |
| Recommended Value Step : 10.0 |
| Restriction : $\left(\text{Length2} \leq \text{Length1}\right)$ |
Width (input control) .............................................................. extent.x ~ long
Width of the image to be processed subsequently.
Default Value: 512
Value Suggestions: Width ∈ \{128, 160, 192, 256, 320, 384, 512, 640, 768\}
Typical Range of Values: 0 ≤ Width ≤ 0(lin)
Minimal Value Step: 1
Recommended Value Step: 16

Height (input control) ......................................................... extent.y ~ long
Height of the image to be processed subsequently.
Default Value: 512
Value Suggestions: Height ∈ \{120, 128, 144, 240, 256, 288, 480, 512, 576\}
Typical Range of Values: 0 ≤ Height ≤ 0(lin)
Minimal Value Step: 1
Recommended Value Step: 16

Interpolation (input control) .................................................. string ~ String
Type of interpolation to be used.
Default Value: ‘nearest neighbor’
Value List: Interpolation ∈ \{'nearest neighbor', 'bilinear', 'bicubic'\}

MeasureHandle (output control) .......................................... measure_id ~ HMeasureX / long
Measure object handle.

If the parameter values are correct the operator GenMeasureRectangle2 returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information
GenMeasureRectangle2 is reentrant and processed without parallelization.

Possible Predecessor Functions
DrawRectangle2

Possible Successor Functions
MeasurePos, MeasurePairs, MeasureThresh

Alternatives
EdgesSubPix

Module

Sub-pixel operators

```
[out] VARIANT RowEdgeFirst HImageX.MeasurePairs
[in] long MeasureHandle, [in] double Sigma, [in] double Threshold,
[in] String Transition, [in] String Select, [out] VARIANT ColumnEdgeFirst,
[out] VARIANT AmplitudeFirst, [out] VARIANT RowEdgeSecond,
[out] VARIANT ColumnEdgeSecond, [out] VARIANT AmplitudeSecond,
[out] VARIANT IntraDistance, [out] VARIANT InterDistance

[out] VARIANT RowEdgeFirst HMeasureX.MeasurePairs ([in] HImageX Image,
[in] String Select, [out] VARIANT ColumnEdgeFirst,
[out] VARIANT AmplitudeFirst, [out] VARIANT RowEdgeSecond,
[out] VARIANT ColumnEdgeSecond, [out] VARIANT AmplitudeSecond,
[out] VARIANT IntraDistance, [out] VARIANT InterDistance

void HOperatorSetX.MeasurePairs ([in] IHObjectX Image,
[in] long MeasureHandle, [in] double Sigma, [in] double Threshold,
[in] String Transition, [in] String Select, [out] VARIANT RowEdgeFirst,
[out] VARIANT ColumnEdgeFirst, [out] VARIANT AmplitudeFirst,
[out] VARIANT RowEdgeSecond, [out] VARIANT ColumnEdgeSecond,
[out] VARIANT AmplitudeSecond, [out] VARIANT IntraDistance,
[out] VARIANT InterDistance
```

Extract straight edge pairs perpendicular to a rectangle.
MeasurePairs serves to extract \textit{straight} edge pairs which lie \textit{perpendicular} to the major axis of a rectangle. The extraction algorithm is identical to MeasurePos. In addition the edges are grouped to pairs: If $\text{Transition} = \text{'positive'}$, the edge points with a dark-to-light transition in the direction of the major axis of the rectangle are returned in $\text{RowEdgeFirst}$ and $\text{ColumnEdgeFirst}$. In this case, the corresponding edges with a light-to-dark transition are returned in $\text{RowEdgeSecond}$ and $\text{ColumnEdgeSecond}$. If $\text{Transition} = \text{'negative'}$, the behavior is exactly opposite. If $\text{Transition} = \text{'all'}$, the first detected edge defines the transition for $\text{RowEdgeFirst}$ and $\text{ColumnEdgeFirst}$. If more than one edge with the same transition is found, the first one is used as a pair element. Finally, it is possible to select which edge pairs are returned. If $\text{Select} \text{ for } \text{RowEdgeFirst}$ is set to $\text{'last'}$, only the last one is returned. The extracted edges are returned as single points which lie on the major axis of the rectangle. The corresponding edge amplitudes are returned in $\text{AmplitudeFirst}$ and $\text{AmplitudeSecond}$. In addition, the distance between each edge pair is returned in $\text{IntraDistance}$ and the distance between consecutive edge pairs is returned in $\text{InterDistance}$. Here, $\text{IntraDistance}[i]$ corresponds to the distance between $\text{EdgeFirst}[i]$ and $\text{EdgeSecond}[i]$, i.e., the tuple $\text{IntraDistance}$ contains one element less than the tuples of the edge pairs. $\text{InterDistance}[i]$ corresponds to the distance between $\text{EdgeFirst}[i]$ and $\text{EdgeSec-}
\text{ond}[i+1]$, i.e., the tuple $\text{InterDistance}$ contains one element less than the tuples of the edge pairs.

\begin{tabular}{p{15cm}}
\textbf{Attention} & \\
\hline
MeasurePairs only returns meaningful results if the assumptions that the edges are straight and perpendicular to the major axis of the rectangle are fulfilled. Thus, it should not be used to extract edges from curved objects, for example. Furthermore, the user should ensure that the rectangle is as close to perpendicular as possible to the edges in the image. & \\
\hline
\end{tabular}

\begin{tabular}{p{15cm}}
\textbf{Parameter} & \\
\hline
$\textbf{Image}$ (input iconic) & $\text{image} \sim \text{HImageX} / \text{HObjectX}$ (byte) \text{Input image.} & \\
$\textbf{MeasureHandle}$ (input control) & $\text{measure_id} \sim \text{long} / \text{HMeasureX}$ \text{Measure object handle.} & \\
$\textbf{Sigma}$ (input control) & $\text{number} \sim \text{double}$ \text{Sigma of gaussian smoothing.} & \\
\text{Default Value} : 1.0 & \\
\text{Value Suggestions} : $\text{Sigma} \in \{0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0\}$ & \\
Typical Range of Values : $0.4 \leq \text{Sigma} \leq 0.4$ (lin) & \\
Minimal Value Step : 0.01 & \\
Recommended Value Step : 0.1 & \\
Restriction : $(\text{Sigma} \geq 0.4)$ & \\
$\textbf{Threshold}$ (input control) & $\text{number} \sim \text{double}$ \text{Minimum edge amplitude.} & \\
\text{Default Value} : 30.0 & \\
\text{Value Suggestions} : $\text{Threshold} \in \{5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0\}$ & \\
Typical Range of Values : $1 \leq \text{Threshold} \leq 1$ (lin) & \\
Minimal Value Step : 0.5 & \\
Recommended Value Step : 2 & \\
$\textbf{Transition}$ (input control) & $\text{string} \sim \text{String}$ \text{Light/dark or dark/light edge.} & \\
\text{Default Value} : \text{'all'} & \\
\text{Value List} : $\text{Transition} \in \{\text{‘all’, ‘positive’, ‘negative’}\}$ & \\
$\textbf{Select}$ (input control) & $\text{string} \sim \text{String}$ \text{Selection of end points.} & \\
\text{Default Value} : \text{'all'} & \\
\text{Value List} : $\text{Select} \in \{\text{‘all’, ‘first’, ‘last’}\}$ & \\
$\textbf{RowEdgeFirst}$ (output control) & $\text{point.y} \sim \text{VARIANT}$ (real) \text{Row coordinate of the center of the edge.} & \\
$\textbf{ColumnEdgeFirst}$ (output control) & $\text{point.x} \sim \text{VARIANT}$ (real) \text{Column coordinate of the center of the edge.} & \\
$\textbf{AmplitudeFirst}$ (output control) & $\text{real} \sim \text{VARIANT}$ (real) \text{Edge amplitude of found edges (with sign).} & \\
$\textbf{RowEdgeSecond}$ (output control) & $\text{point.y} \sim \text{VARIANT}$ (real) \text{Row coordinate of the center of the edge.} & \\
$\textbf{ColumnEdgeSecond}$ (output control) & $\text{point.x} \sim \text{VARIANT}$ (real) \text{Column coordinate of the center of the edge.} & \\
\end{tabular}
Column coordinate of the center of the edge.

Edge amplitude of found edges (with sign).

Distance between edges of an edge pair.

Distance between consecutive edge pairs.

If the parameter values are correct the operator MeasurePairs returns the value TRUE. Otherwise an exception handling is raised.

MeasurePairs is reentrant and processed without parallelization.

GenMeasureRectangle2

CloseMeasure

EdgesSubPix, MeasurePos

HImageX.MeasurePos

HMeasureX.MeasurePos

HOperatorSetX.MeasurePos

Extract straight edges perpendicular to a rectangle.

MeasurePos extracts straight edges which lie perpendicular to the major axis of a rectangle.

The algorithm works by averaging the gray values in “slices” perpendicular to the major axis of the rectangle in order to obtain a one-dimensional edge profile. The sampling is done at subpixel positions in the image Image at integer row and column distances (in the coordinate frame of the rectangle) from the center of the rectangle. Since this involves some calculations which can be used repeatedly in several mesurements, the operator GenMeasureRectangle2 is used to perform these calculations only once, and thus to increase the speed of MeasurePos significantly. Since there is a trade-off between accuracy and speed in the subpixel calculations of the gray values, and thus in the accuracy of the extracted edge positions, different interpolation schemes can be selected in GenMeasureRectangle2. (The interpolation only influences rectangles not aligned with the image axes.) The measure object generated with GenMeasureRectangle2 is passed in MeasureHandle.

After the one-dimensional edge profile has been calculated, subpixel edge locations are computed by convolving the profile with the derivatives of a Gaussian smoothing kernel of standard deviation Sigma. Salient edges can be selected with the parameter Threshold, which constitutes a threshold on the amplitude, i.e., the absolute value of the first derivative, of the edge. Additionally, it is possible to select only positive edges, i.e., edges which constitute a dark-to-light transition in the direction of the major axis of the rectangle (Transition = 'positive'), only negative edges, i.e., light-to-dark transitions (Transition = 'negative'), or both types of edges (Transition = 'all'). Finally, it is possible to select which edge points are returned. If Select is set to 'all', all edge points
are returned. If it is set to 'first', only the first of the extracted edge points is returned, while it is set to 'last', only the last one is returned.

The extracted edges are returned as single points which lie on the major axis of the rectangle in (RowEdge,ColumnEdge). The corresponding edge amplitudes are returned in Amplitude. In addition, the distance between consecutive edge points is returned in Distance. Here, Distance[i] corresponds to the distance between Edge[i] and Edge[i+1], i.e., the tuple Distance contains one element less than the tuples RowEdge and ColumnEdge.

Attention

MeasurePos only returns meaningful results if the assumptions that the edges are straight and perpendicular to the major axis of the rectangle are fulfilled. Thus, it should not be used to extract edges from curved objects, for example. Furthermore, the user should ensure that the rectangle is as close to perpendicular as possible to the edges in the image.

Parameter

- **Image** (input iconic) .............................................. image ~ HImageX / IHObjectX (byte)
  Input image.

- **MeasureHandle** (input control) ............................... measure_id ~ long / HMeasureX
  Measure object handle.

- **Sigma** (input control) .............................................. number ~ double
  Sigma of gaussian smoothing.
  Default Value : 1.0
  Value Suggestions : Sigma ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
  Typical Range of Values : 0.4 ≤ Sigma ≤ 0.4(lin)
  Minimal Value Step : 0.01
  Recommended Value Step : 0.1
  Restriction : (Sigma ≥ 0.4)

- **Threshold** (input control) ............................................. number ~ double
  Minimum edge amplitude.
  Default Value : 30.0
  Value Suggestions : Threshold ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
  Typical Range of Values : 1 ≤ Threshold ≤ 1(lin)
  Minimal Value Step : 2
  Recommended Value Step : 0.5

- **Transition** (input control) ......................................... string ~ String
  Light/dark or dark/light edge.
  Default Value : ’all’
  Value List : Transition ∈ {'all', ’positive’, ’negative’}

- **Select** (input control) ............................................... string ~ String
  Selection of end points.
  Default Value : ’all’
  Value List : Select ∈ {'all', ’first’, ’last’}

- **RowEdge** (output control) .......................................... point.y ~ VARIANT (real)
  Row coordinate of the center of the edge.

- **ColumnEdge** (output control) ...................................... point.x ~ VARIANT (real)
  Column coordinate of the center of the edge.

- **Amplitude** (output control) ........................................ real ~ VARIANT (real)
  Edge amplitude of found edges (with sign).

- **Distance** (output control) ......................................... real ~ VARIANT (real)
  Distance between consecutive edges.

Result

If the parameter values are correct the operator MeasurePos returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

MeasurePos is reentrant and processed without parallelization.

Possible Predecessor Functions

GenMeasureRectangle2
Extract a gray value profile perpendicular to a rectangle.

MeasureProjection extracts a one-dimensional gray value profile perpendicular to a rectangle. This is done by averaging the gray values in “slices” perpendicular to the major axis of the rectangle. The sampling is done at subpixel positions in the image Image at integer row and column distances (in the coordinate frame of the rectangle) from the center of the rectangle. Since this involves some calculations which can be used repeatedly in several projections, the operator GenMeasureRectangle2 is used to perform these calculations only once, and thus to increase the speed of MeasureProjection significantly. Since there is a trade-off between accuracy and speed in the subpixel calculations of the gray values, different interpolation schemes can be selected in GenMeasureRectangle2 (the interpolation only influences rectangles not aligned with the image axes). The measure object generated with GenMeasureRectangle2 is passed in MeasureHandle.

Parameter

- **Image** (input iconic) .............................. image \( \sim \) HImageX / IHObjectX (byte )
  - Input image.
- **MeasureHandle** (input control) .......................... measure\_id \( \sim \) long / HMeasureX
  - Measure object handle.
- **GrayValues** (output control) .......................... number \( \sim \) VARIANT (real )
  - Gray value profile.

Result

If the parameter values are correct the operator MeasureProjection returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

MeasureProjection is reentrant and processed without parallelization.

Possible Predecessor Functions

GenMeasureRectangle2

Possible Successor Functions

CloseMeasure

Alternatives

GrayProjections

Module

Sub-pixel operators

HALCON 6.0
Extracting points with a particular grey value along a rectangle.

MeasureThresh extracts points for which the gray value within an one-dimensional gray value profile is equal to the specified threshold Threshold. The gray value profile is projected onto the major axis of the measure rectangle which is passed with the parameter MeasureHandle, so the threshold points calculated within the gray value profile correspond to certain image coordinates on the rectangle’s major axis. These coordinates are returned as the operator results in RowThresh and ColumnThresh.

If the gray value profile intersects the threshold line for several times, the parameter Select determines which values to return. Possible settings are 'first', 'last', 'first_last' (first and last) or 'all'. For the last two cases Distance returns the distances between the calculated points.

The gray value profile is created by averaging the gray values along all line segments, which are defined by the measure rectangle as follows:

1. The segments are perpendicular to the major axis of the rectangle,
2. they have an integer distance to the center of the rectangle,
3. the rectangle bounds the segments.

For every line segment, the average of the gray values of all points with an integer distance to the major axis is calculated. Due to translation and rotation of the measure rectangle with respect to the image coordinates the input image Image is in general sampled at subpixel positions.

Since this involves some calculations which can be used repeatedly in several projections, the operator GenMeasureRectangle2 is used to perform these calculations only once in advance. Here, the measure object MeasureHandle is generated and different interpolation schemes can be selected.

Attention

MeasureThresh only returns meaningful results if the assumptions that the edges are straight and perpendicular to the major axis of the rectangle are fulfilled. Thus, it should not be used to extract edges from curved objects, for example. Furthermore, the user should ensure that the rectangle is as close to perpendicular as possible to the edges in the image.

Parameter

- **Image** (input iconic) ............................................ image \(\sim\) HImageX / IObjectX (byte)
  Input image.
- **MeasureHandle** (input control) .................................... measure_id \(\sim\) long / HMeasureX
  Measure object handle.
- **Sigma** (input control) .................................................. number \(\sim\) double
  Sigma of gaussian smoothing.
  **Default Value**: 1.0
  **Value Suggestions**: Sigma \(\in\) \{0.0, 0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0\}
  **Typical Range of Values**: 0.4 \(\leq\) Sigma \(\leq\) 0.4 (lin)
  **Minimal Value Step**: 0.01
  **Recommended Value Step**: 0.1
  **Restriction**: (Sigma \(\geq\) 0.0)
12.12. OCR

- **Threshold** (input control) .................................................. number  \(\sim\) double
  Threshold.
  - Default Value: 128.0
  - Typical Range of Values: 0 \(\leq\) Threshold \(\leq\) 0 (lin)
  - Minimal Value Step: 1
  - Recommended Value Step: 0.5

- **Select** (input control) .................................................. string  \(\sim\) String
  Selection of points.
  - Default Value: 'all'
  - Value List: Select \(\in\) \{'all', 'first', 'last', 'first and last'

- **RowThreshold** (output control) .................................................. point.y  \(\sim\) VARIANT (real)
  Row coordinates of points with threshold value.

- **ColumnThreshold** (output control) ............................................... point.x  \(\sim\) VARIANT (real)
  Column coordinates of points with threshold value.

- **Distance** (output control) ............................................... real  \(\sim\) VARIANT (real)
  Distance between consecutive points.

If the parameter values are correct the operator MeasureThreshold returns the value TRUE. Otherwise, an exception handling is raised.

---

### Parallelization Information

MeasureThreshold is *reentrant* and processed *without* parallelization.

### Possible Predecessor Functions

- GenMeasureRectangle2

### Possible Successor Functions

- CloseMeasure

### Alternatives

- MeasurePos, EdgesSubPix, MeasurePairs

---

### Module

Sub-pixel operators

---

12.12 OCR

```c

```

Adding characters to a training file.

The operator AppendOcrTrainf serves to prepare the training with the operator TrainfOcrClassBox. Hereby regions, representing characters, including their grayvalues (region and pixel) and the corresponding class name will be written into a file. An arbitrary number of regions within one image is supported. For each character (region) in Character the corresponding class name must be specified in Class. The grayvalues are passed via the parameter Image. The file name can be chosen at will. In contrast to WriteOcrTrainf the characters are appended to the file. If the file does not exist a new file is generated.

---

- **Character** (input iconic) ........................................................ region(-array)  \(\sim\) HRegionX / IHOBJECTX
  Characters to be trained.

- **Image** (input iconic) ........................................................ image  \(\sim\) HImageX / IHOBJECTX
  Grayvalues of the characters.

- **Class** (input control) ........................................................ string(-array)  \(\sim\) VARIANT (string)
  Class (name) of the characters.
**FileName** (input control)  
Name of the training file.  
**Default Value**: 'train.ocr'  

If the parameters are correct, the operator `AppendOcrTrainf` returns the value TRUE. Otherwise an exception will be raised.

**Parallelization Information**  
`AppendOcrTrainf` is processed under **mutual exclusion** against itself and without parallelization.

**Possible Predecessor Functions**  
Threshold, Connection, CreateOcrClassBox, ReadOcr

**Possible Successor Functions**  
TrainfOcrClassBox, InfoOcrClassBox, WriteOcr, DoOcrMulti, DoOcrSingle

**Alternatives**  
WriteOcrTrainf, WriteOcrTrainfImage

**Module**  
Optical character recognition

---

```void HMiscX.CloseAllOcrs ( )
void HOperatorSetX.CloseAllOcrs ( )
```

Destroy all OCR classificators.

CloseAllOcrs deletes all OCR classificators and frees the used memory space. All the trained data will be lost.

**Attention**  
Since all classificators are closed by CloseAllOcrs all handles become invalid.

**Result**  
If it is possible to close the OCR classificators the operator `CloseAllOcrs` returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**  
`CloseAllOcrs` is processed **completely exclusively** without parallelization.

**Alternatives**  
CloseOcr

**Module**  
Optical character recognition

---

```void HOperatorSetX.CloseOcr ([in] long OcrHandle )
```

Deallocation of the memory of an OCR classifier.

The operator `CloseOcr` deallocates the memory of the classifier having the number `OcrHandle`. Hereby all corresponding data will be deleted. However, if necessary, they can be saved in advance using the operator `WriteOcr`. The number `OcrHandle` will be invalid after the call; but later the system can use it again for new classifiers.

**Attention**  
All data of the classifier will be deleted in main memory (not on the hard disk).

**Parameter**  

- **OcrHandle** (input control)  
  
  **Example**
If the parameter OcrHandle is valid, the operator CloseOcr returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information
CloseOcr is reentrant and processed without parallelization.

Possible Predecessor Functions
WriteOcrTrainf

Possible Successor Functions
ReadOcr

Module
Optical character recognition


Concatenation of trainings files.
The operator ConcatOcrTrainf stores all characters which are contained in the files SingleFiles into a new file with the name ComposedFile.

Parameter

▷ SingleFiles (input control) ................................. filename ~ VARIANT( string )
Name of the single training files.
Default Value : ”

▷ ComposedFile (input control) ................................. filename ~ String
Name of the composed training file.
Default Value : ’all characters’

Result
If the parameters are correct, the operator ConcatOcrTrainf returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information
ConcatOcrTrainf is processed under mutual exclusion against itself and without parallelization.

Possible Predecessor Functions
WriteOcrTrainf, AppendOcrTrainf

Possible Successor Functions
TrainfOcrClassBox, InfoOcrClassBox, WriteOcr, DoOcrMulti, DoOcrSingle

Module
Optical character recognition


Creating a new OCR-classifier.
The operator CreateOcrClassBox creates a new OCR classifier. For a description of this classifier see operator LearnClassBox. This classifier must then be trained with the help of the operators TrainfOcrClassBox or TrainOcrClassBox.
The parameters `WidthPattern` and `HeightPattern` indicate the size of the input-layer of the network. This size is used for the features `projection_horizontal`, `projection_vertical` and `pixel`. The bigger it is, the more characters can be distinguished. Hereby the amount of time necessary for the training (as well as the number of training random samples) and the time necessary for the recognition, however, will increase as well. The larger the input level, the more specific the training is for a certain font. The parameter `Interpolation` indicates the interpolation mode concerning the adaptation of characters in the image to the network. For more detailed information on this parameter see also `AffineTransImage`.

The parameter `Character` determines all the characters which have to be recognized. Normally the transmitted strings consist of one character (e.g. alphabet). But also strings of any length can be learned. The number of distinguishable characters (number of strings in `Character`) is limited to 2048.

The parameter `Features` helps to chose additional features besides grayvalues in order to recognize characters. By using `default` the features `ratio`, `projection_horizontal`, `projection_vertical`, `moments_region_2nd_invar`, and `moments_region_2nd_rel_invar` will be set.

The following features are available:

- `ratio` Ratio of the character.
- `width` Width of the character (not invariant to scaling).
- `height` Height of the character (not invariant to scaling).
- `zoom_factor` Difference in size between the current character and the values of `WidthPattern` and `HeightPattern` (not invariant to scaling).
- `foreground` Relative number of pixels in the foreground.
- `foreground_grid_9` Relative number of foreground pixels in a $3 \times 3$ grid within the surrounding rectangle of the character.
- `foreground_grid_16` Relative number of foreground pixels in a $4 \times 4$ grid within the surrounding rectangle of the character.
- `anisometry` Form feature anisometry.
- `compactness` Form feature compactness.
- `convexity` Form feature convexity.
- `moments_region_2nd_invar` Normed 2nd geometric moments of the region. See also `MomentsRegion2ndInvar`.
- `moments_region_2nd_rel_invar` Normed 2nd relativ geometric moments of the region. See also `MomentsRegion2ndRelInvar`.
- `moments_region_3rd_invar` Normed 3rd geometric moments of the region. See also `MomentsRegion3rdInvar`.
- `moments_central` Normed central geometric moments of the region. See also `MomentsRegionCentral`.
- `phi` Orientation (angle) of the character.
- `num_connect` Number of connecting components.
- `num_holes` Number of holes.
- `projection_horizontal` Horizontal projection of the grayvalues.
- `projection_horizontal_invar` Horizontal projection of the grayvalues with are automatically scaled to maximum range.
- `projection_vertical` Vertical projection of the grayvalues.
- `projection_vertical_invar` Vertical projection of the grayvalues with are automatically scaled to maximum range.
- `cooc` Values of the binary cooccurrence matrix.
- `moments_gray_plane` Normed grayvalue moments and the angles of the grayvalue level.
- `num_runs` Number of chords in the region normed to the area.
- `chord_histo` Frequency of the chords per row.
- `pixel` Grayvalue of the character.
- `pixel_invar` Grayvalues of the character with automatic maximal scaling of the gray values.
Attention

Parameter

- **WidthPattern** (input control) ................................................... integer \( \sim \) long
  Width of the input layer of the network.
  - Default Value: 8
  - Value Suggestions: \( \text{WidthPattern} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20\}\)
  - Typical Range of Values: \( 1 \leq \text{WidthPattern} \leq 1 \)

- **HeightPattern** (input control) ................................................... integer \( \sim \) long
  Height of the input layer of the network.
  - Default Value: 10
  - Value Suggestions: \( \text{HeightPattern} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20\}\)
  - Typical Range of Values: \( 1 \leq \text{HeightPattern} \leq 1 \)

- **Interpolation** (input control) ................................................... integer \( \sim \) long
  Interpolation mode concerning scaling of characters.
  - Default Value: 0
  - Value List: \( \{0, 1, 2\}\)
  - Value Suggestions: \( \text{Interpolation} \in \{0, 1, 2\}\)

- **Features** (input control) ................................................... string(-array) \( \sim \) VARIANT ( string )
  Additional features.
  - Default Value: ‘default’
    \text{‘moments_region_2nd_invar’, ‘moments_region_2nd_rel_invar’, ‘moments_region_3rd_invar’,}
    \text{‘cooc’, ‘moments_gray_plane’}\}\)

- **Character** (input control) ................................................... string \( \sim \) VARIANT ( string )
  All characters of a set.
  - Default Value: \[\{‘a’, ‘b’, ‘c’\}\]

- **OcrHandle** (output control) ................................................... ocr \( \sim \) HOcrX / long
  ID of the created OCR classifier.

Example

Result

If the parameters are correct, the operator CreateOcrClassBox returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information

CreateOcrClassBox is processed completely exclusively without parallelization.

Possible Predecessor Functions

ResetObjDb

Possible Successor Functions

TrainOcrClassBox, TrainfOcrClassBox, InfoOcrClassBox, WriteOcr, OcrChangeChar

See Also

AffineTransImage, OcrChangeChar, MomentsRegion2NdInvar, MomentsRegion2NdRelInvar, MomentsRegion3RdInvar, MomentsRegionCentral

Module

Optical character recognition

HALCON 6.0
Classification of characters.

The operator DoOcrMulti assigns a class to every Character (character). For grayvalue features the grayvalues from the surrounding rectangles of the regions are used. The grayvalues will be taken from the parameter Image. For each character the corresponding class will be returned in Class and a confidence value will be returned in Confidence. The confidence value indicates the similarity between the input pattern and the assigned character.

Parameter

- **Character** (input iconic) region(-array) \( \sim \) HRegionX / IHOBJECTX Characters to be recognized.
- **Image** (input iconic) image \( \sim \) HImageX / IHOBJECTX Grayvalues for the characters.
- **OcrHandle** (input control) ocr \( \sim \) HOcrX / long ID of the OCR classifier.
- **Class** (output control) string(-array) \( \sim \) VARIANT ( string ) Class (name) of the characters.
- **Confidence** (output control) real(-array) \( \sim \) VARIANT ( real ) Confidence values of the characters.

Attention

DoOcrMulti is reentrant and automatically parallelized (on tuple level).

Parallelization Information

Possible Predecessor Functions

TrainOcrClassBox, TrainfOcrClassBox, ReadOcr, Connection, SortRegion

See Also

WriteOcr

Alternatives

DoOcrSingle

Module

Optical character recognition
The operator DoOcrSingle assigns classes to the Character (characters). For grayscale features, grayvalues of the surrounding rectangles of the regions will be used. The grayvalues will be taken from the parameter Image. For each character, the two classes with the highest confidences will be returned in Classes. The corresponding confidences will be returned in Confidences. The confidence value indicates the similarity between the input pattern and the assigned character.

Parameter

- **Character** (input iconic) 
  Character to be recognized.
- **Image** (input iconic) 
  Grayvalues of the characters.
- **OcrHandle** (input control) 
  ID of the OCR classifier.
- **Classes** (output control) 
  Classes (names) of the characters.
- **Confidences** (output control) 
  Confidence values of the characters.

Example

If the input parameters are correct, the operator DoOcrSingle returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information

DoOcrSingle is reentrant and processed without parallelization.

Possible Predecessor Functions

TrainOcrClassBox, TrainfOcrClassBox, ReadOcr, Connection, SortRegion

See Also

WriteOcr

Alternatives

DoOcrMulti

Module

Optical character recognition

```
[out] long WidthPattern HOCRX/INFOOCRCLASSBOX ([out] long HeightPattern,
[out] long Interpolation, [out] long WidthMaxChar, [out] long HeightMaxChar,
[out] VARIANT Features, [out] VARIANT Characters )

void HOPERATORSETX/INFOOCRCLASSBOX ([in] long OcrHandle,
[out] long WidthPattern, [out] long HeightPattern, [out] long Interpolation,
[out] long WidthMaxChar, [out] long HeightMaxChar, [out] VARIANT Features,
[out] VARIANT Characters )
```

Information about an OCR classifier.

The operator InfoOcrClassBox returns some information about an OCR classifier. The parameters are equivalent to those of CreateOcrClassBox. The parameters WidthMaxChar and HeightMaxChar indicate the extension of the largest trained character. These values can be used to control the segmentation.
**Parameter**

- **OcrHandle** (input control)  
  ID of the OCR classifier.

- **WidthPattern** (output control)  
  Width of the scaled characters.

- **HeightPattern** (output control)  
  Height of the scaled characters.

- **Interpolation** (output control)  
  Interpolation mode for scaling the characters.

- **WidthMaxChar** (output control)  
  Width of the largest trained character.

- **HeightMaxChar** (output control)  
  Height of the largest trained character.

- **Features** (output control)  
  Used features.

- **Characters** (output control)  
  All characters of the set.

**Example**

The operator `InfoOcrClassBox` always returns TRUE.

**Parallelization Information**

`InfoOcrClassBox` is reentrant and processed without parallelization.

**Possible Predecessor Functions**

- `ReadOcr`, `CreateOcrClassBox`

**Possible Successor Functions**

- `WriteOcr`

**Module**

Optical character recognition

```c
void HOcrX.OcrChangeChar ([in] VARIANT Character )

void HOperatorSetX.OcrChangeChar ([in] long OcrHandle, [in] VARIANT Character )
```

**Defining a new conversion table for the characters.**

The operator `OcrChangeChar` establishes a new look-up table for the characters. Hereby the number of strings of `Character` must be the same as of the classifier `OcrHandle`. In order to enlarge the font, the operator `OcrChangeChar` may be used as follows: More characters than actually needed will be indicated when creating a network using `CreateOcrClassBox`. The last $n$ characters will not be used so far. If more characters are needed at a later stage, these unused characters will be allocated and then trained with the help of the operator `OcrChangeChar`.

**Attention**

- **OcrHandle** (input control)  
  ID of the OCR-network to be changed.

- **Character** (input control)  
  New assign of characters.
  
  Default Value: `['a', 'b', 'c']`

Example

If the number of characters in Character is identical with the number of the characters of the network, the
operator OcrChangeChar returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information
OcrChangeChar is processed completely exclusively without parallelization.

Possible Predecessor Functions
ReadOcr

Possible Successor Functions
DoOcrMulti, DoOcrSingle, WriteOcr

Module

Optical character recognition

Example

If the number of characters in Character is identical with the number of the characters of the network, the
operator OcrChangeChar returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information
OcrChangeChar is processed completely exclusively without parallelization.

Possible Predecessor Functions
ReadOcr

Possible Successor Functions
DoOcrMulti, DoOcrSingle, WriteOcr

Module

Optical character recognition

---

Result

If the number of characters in Character is identical with the number of the characters of the network, the
operator OcrChangeChar returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information
OcrChangeChar is processed completely exclusively without parallelization.

Possible Predecessor Functions
ReadOcr

Possible Successor Functions
DoOcrMulti, DoOcrSingle, WriteOcr

Module

Optical character recognition

---

Access the features which correspond to a character.
The operator OcrGetFeatures calculates the features for the given Character. The type and number of
features is determined by the classifier OcrHandle. FeatureVector contains the same values which are used
inside operators like TraindOcrClassBox or TrainfOcrClassBox.

Parameter

- **Character** (input iconic) .................................. image \( \sim \) HImageX / IHObjectX
  Characters to be trained.
- **OcrHandle** (input control) .................................. ocr \( \sim \) HOcrX / long
  ID of the desired OCR-classifier.
- **FeatureVector** (output control) ............................... real \( \sim \) VARIANT ( real )
  Feature vector.

Result

If the parameters are correct, the operator OcrGetFeatures returns the value TRUE. Otherwise an exception
will be raised.

Parallelization Information
OcrGetFeatures is reentrant and processed without parallelization.

Possible Predecessor Functions
CreateOcrClassBox, ReadOcr, ReduceDomain, Threshold, Connection

Possible Successor Functions
LearnClassBox

See Also

TrainfOcrClassBox, TraindOcrClassBox

Module

Optical character recognition
void HOCRX.ReadOcr ([in] String FileName )
void HOperatorSetX.ReadOcr ([in] String FileName, [out] long OcrHandle )

Reading of an OCR classifier from a file.

The operator ReadOcr reads an OCR classifier from a file FileName. This file will hereby be searched in the directory ($HALCONROOT/ocr/) as well as in the currently used directory. If too many classifiers have been loaded, an error message will be displayed.

Attention

Parameter

(fileName String)
Name of the OCR classifier file.

Default Value : 'testnet'

(ocr HOCRX / long)
ID of the read OCR classifier.

Example

Result

If the indicated file is available and the format is correct, the operator ReadOcr returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information

ReadOcr is processed completely exclusively without parallelization.

Possible Predecessor Functions

ResetObjDb

Possible Successor Functions

DoOcrMulti, DoOcrSingle, TrainOcrClassBox, TrainfOcrClassBox

See Also

WriteOcr, DoOcrMulti, TrainOcrClassBox, TrainfOcrClassBox

Module

Optical character recognition

Read training characters from files and convert to images.

ReadOcrTrainf reads all characters from the specified file names and converts them into images. The domain is defined according to the foreground of the characters (as specified in WriteOcrTrainf). The names of the characters are returned in CharacterNames. If more than one file name is given the files are processed in the order the file names.

Parameter

(image HImageX / HUntypedObjectX ( byte))
Images read from file.

(filename named(-array) VARIANT ( string))
Names of the training files.

(Characters HImageX / HUntypedObjectX ( byte))
Images read from file.

(string VARIANT ( string))
Names of the read characters.
If the parameter values are correct the operator \texttt{ReadOcrTrainf} returns the value TRUE. Otherwise an exception handling is raised.

\textbf{Parallelization Information}\n
\texttt{ReadOcrTrainf} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{Possible Predecessor Functions}\n
\textbf{Possible Successor Functions}\n
\textbf{Possible Predecessor Functions}\n
\textbf{Possible Successor Functions}\n
\textbf{See Also}\n
\textbf{Alternatives}\n
\textbf{Module}\n
\textbf{Optical character recognition}\n
\begin{verbatim}
[out] VARIANT CharacterNames \texttt{HMiscX.ReadOcrTrainfNames}
  ([in] VARIANT TrainFileNames, [out] VARIANT CharacterCount )

void \texttt{HOperatorSetX.ReadOcrTrainfNames} ([in] VARIANT TrainFileNames,
  [out] VARIANT CharacterNames, [out] VARIANT CharacterCount )
\end{verbatim}

Query which characters are stored in a training file.

\texttt{ReadOcrTrainfNames} extracts the names and frequency of all characters in the specified training files.

\textbf{Parameter}\n
\begin{itemize}
  \item \textbf{TrainFileNames} (input control) \texttt{filename.named(-array)} $\leadsto$ \texttt{VARIANT( string )}
    \hspace{1em} Names of the training files.
  \item \textbf{CharacterNames} (output control) \texttt{string(-array)} $\leadsto$ \texttt{VARIANT( string )}
    \hspace{1em} Names of the read characters.
  \item \textbf{CharacterCount} (output control) \texttt{integer(-array)} $\leadsto$ \texttt{VARIANT( integer )}
    \hspace{1em} Number of the characters.
\end{itemize}

\textbf{Result}\n
If the parameter values are correct the operator \texttt{ReadOcrTrainfNames} returns the value TRUE. Otherwise an exception handling is raised.

\textbf{Parallelization Information}\n
\texttt{ReadOcrTrainfNames} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{Possible Predecessor Functions}\n
\textbf{Possible Predecessor Functions}\n
\textbf{See Also}\n
\textbf{Alternatives}\n
\textbf{Module}\n
\textbf{Optical character recognition}\n
\begin{verbatim}
void \texttt{HImageX.ReadOcrTrainfSelect} ([in] VARIANT TrainFileNames,
  [in] VARIANT SearchNames, [out] VARIANT FoundNames )

void \texttt{HOperatorSetX.ReadOcrTrainfSelect} ([out] HUntypedObjectX Characters,
\end{verbatim}

Read training specific characters from files and convert to images.
ReadOcrTrainSelect reads the characters given in SearchNames from the specified files and converts them into images. It works similar to ReadOcrTrain but here the characters which are extracted can be specified.

---

**Parameter**

- **Characters** (output iconic) ................... image  \( \sim \) HImageX / HUtypedObjectX (byte)
  Images read from file.

- **TrainFileNames** (input control) .................. filename.named(-array)  \( \sim \) VARIANT (string)
  Names of the training files.
  **Default Value:** "

- **SearchNames** (input control) ................... string(-array)  \( \sim \) VARIANT (string)
  Names of the characters to be extracted.
  **Default Value:** '0'

- **FoundNames** (output control) ................... string(-array)
  Names of the read characters.

---

**Result**

If the parameter values are correct the operator ReadOcrTrainSelect returns the value TRUE. Otherwise an exception handling is raised.

---

**Parallelization Information**

ReadOcrTrainSelect is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

- DispImage, SelectObj, ZoomImageSize

---

**Possible Successor Functions**

- TrainOcrClassBox

---

**See Also**

- ReadOcrTrain

---

**Alternatives**

- HOperatorSetX.TestOcrClassBox

---

**Module**

- Optical character recognition

---

### Training of an OCR classifier by the input of regions.

The operator TestdOcrClassBox tests the confidence with which a character belongs to a given class. Any number of regions of an image can be passed. For each character (region) in Character the corresponding name (class) **Class** must be specified. The grayvalues are passed in **Image**. When the operator has finished the parameter **Confidence** provides information about how sure a character belongs to the (arbitrary chosen) class.

---

**Parameter**

- **Character** (input iconic) .................... region(-array)  \( \sim \) HRegionX / IHObjectX
  Characters to be trained.

- **Image** (input iconic) .......................... image  \( \sim \) HImageX / IHObjectX
  Grayvalues for the characters.

- **OcrHandle** (input control) .................... ocr  \( \sim \) HOcrX / long
  ID of the desired OCR-classifier.
**Class** (input control) ................................. string(-array)  ~  VARIANT ( string )
Class (name) of the characters.
*Default Value*: ‘a’

**Confidence** (output control) ............................ real(-array)  ~  VARIANT ( real )
Confidence for the character to belong to the class.

If the parameters are correct, the operator `TestdOcrClassBox` returns the value TRUE. Otherwise an exception will be raised.

---

**Parallelization Information**

`TestdOcrClassBox` is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

`ReadOcr`, `TrainfOcrClassBox`, `TraindOcrClassBox`

---

**Module**

Optical character recognition

---

**Training of an OCR classifier by the input of regions.**

The operator `TraindOcrClassBox` trains the classifier directly via the input of regions in an image. Any number of regions of an image can be passed. For each character (region) in `Character` the corresponding `name` (class) `Class` must be specified. The grayvalues are passed in `Image`. When the procedure has finished the parameter `AvgConfidence` provides information about the success of the training: It contains the average confidence of the trained characters measured by a re-classification. The confidence of mismatched characters is set to 0 (thus, the average confidence will be decreased significantly).

---

**Attention**

---

**Parameter**

- **Character** (input iconic) ................................. region(-array)  ~  HRegionX / IHObjectX
Characters to be trained.

- **Image** (input iconic) ................................. image  ~  HImageX / IHObjectX
Grayvalues for the characters.

- **OcrHandle** (input control) ................................. ocr  ~  HOcrX / long
ID of the desired OCR-classifier.

- **Class** (input control) ................................. string(-array)  ~  VARIANT ( string )
Class (name) of the characters.
*Default Value*: ‘a’

- **AvgConfidence** (output control) ............................ real  ~  double
Average confidence during a re-classification of the trained characters.

---

**Example**

If the parameters are correct, the operator `TraindOcrClassBox` returns the value TRUE. Otherwise an exception will be raised.

---

**Parallelization Information**

`TraindOcrClassBox` is processed *completely exclusively* without parallelization.

---

HALCON 6.0
Training of an OCR classifier with the help of a training file.
The operator TrainfOcrClassBox trains the classifier OcrHandle via the indicated training files. Any number of files can be indicated. The parameter AvgConfidence provides information about the success of the training: It contains the average confidence of the trained characters measured by a re-classification. The confidence of mismatched characters is set to 0 (thus, the average confidence will be decreased significantly).

**Attention**
The names of the characters in the file must fit the network.

**Parameter**
- **OcrHandle** (input control) ............................ ocr ~ HOcrX/long
  ID of the desired OCR-network.
- **FileName** (input control) ............................ filename(-array) ~ VARIANT (string)
  Name(s) of the training file(s).
  **Default Value**: 'trainocr'
- **AvgConfidence** (output control) .................... real ~ double
  Average confidence during a re-classification of the trained characters.

**Example**
If the file name is correct and the data fit the network, the operator TrainfOcrClassBox returns the value TRUE. Otherwise an exception will be raised.

**Parallelization Information**
TrainfOcrClassBox is processed completely exclusively without parallelization.
The operator `WriteOcr` writes the OCR classifier `OcrHandle` into the file `FileName`. Since the data of the classifier will be lost when the program is finished, they have to be stored after the training if the user wants to use them again at a later execution of the program. The data can then be read with the help of the operator `ReadOcr`. The extension will be added automatically to the parameter `FileName`.

Attention

The output file `FileName` must be given without extension.

Parameter

- `OcrHandle` (input control) ~ `HOCRX / long`
  ID of the OCR classifier.
- `FileName` (input control) ~ `String`
  Name of the file for the OCR classifier (without extension).
  Default Value: 'my_ocr'

Example

If the parameter `OcrHandle` is valid and the indicated file can be written, the operator `WriteOcr` returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information

`WriteOcr` is reentrant and processed without parallelization.

Possible Predecessor Functions

`TrainOcrClassBox, TrainfOcrClassBox`

Possible Successor Functions

`DoOcrMulti, DoOcrSingle`

See Also

`ReadOcr, DoOcrMulti, TrainOcrClassBox, TrainfOcrClassBox`

Module

Optical character recognition

```c
void HRegionX.WriteOcrTrainf ([in] HImageX Image, [in] VARIANT Class,
                             [in] String FileName )
void HOperatorSetX.WriteOcrTrainf ([in] IQObjectX Character,
```

Storing of trained characters into a file.

The operator `WriteOcrTrainf` serves to prepare the training with the operator `TrainfOcrClassBox`. Hereby regions, representing characters, including their grayvalues (region and pixel) and the corresponding class-name will be written into a file. An arbitrary number of regions within one image is supported. For each character (region) in `Character` the corresponding class name must be specified in `Class`. The grayvalues are passed via the parameter `Image`. The file name can be chosen at will.

Attention

Parameter

- `Character` (input iconic) ~ `HRegionX / IQObjectX`
  Characters to be trained.
- `Image` (input iconic) ~ `HImageX / IQObjectX`
  Grayvalues of the characters.
- `Class` (input control) ~ `VARIANT ( string )`
  Class (name) of the characters.
- `FileName` (input control) ~ `String`
  Name of the training file.
  Default Value: 'train_ocr'

HALCON 6.0
Example

Result

If the parameters are correct, the operator `WriteOcrTrainf` returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information

`WriteOcrTrainf` is reentrant and processed without parallelization.

Possible Predecessor Functions

Threshold, Connection, CreateOcrClassBox, ReadOcr

Possible Successor Functions

TrainfOcrClassBox, InfoOcrClassBox, WriteOcr, DoOcrMulti, DoOcrSingle

Module

Optical character recognition

```
void HImageX.WriteOcrTrainfImage ([in] VARIANT Class,
                                 [in] String FileName )

void HOperatorSetX.WriteOcrTrainfImage ([in] IHObjectX Character,
                                        [in] VARIANT Class,
                                        [in] String FileName )
```

Write characters into a training file.

The operator `WriteOcrTrainfImage` is used to prepare the training with the operator `TrainfOcrClassBox`. Hereby regions, representing characters, including their grayvalues (region and pixel) and the corresponding classname will be written into a file. An arbitrary number of regions within one image is supported. For each character (region) in `Character` the corresponding class name must be specified in `Class`. The file name can be chosen at will. In contrast to `WriteOcrTrainf` one image per character is passed. The domain of this image defines the pixels which belong to the character.

Parameter

- **Character** (input iconic) .......................... image(-array)  \( \sim \) HImageX / IHObjectX
  - Characters to be trained.
- **Class** (input control) .............................. string(-array)  \( \sim \) VARIANT ( string )
  - Class (name) of the characters.
- **FileName** (input control) ......................... filename  \( \sim \) String
  - Name of the training file.
  - Default Value : 'trainocr'

Result

If the parameters are correct, the operator `WriteOcrTrainfImage` returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information

`WriteOcrTrainfImage` is reentrant and processed without parallelization.

Possible Predecessor Functions

Threshold, Connection, CreateOcrClassBox, ReadOcr

Possible Successor Functions

TrainfOcrClassBox, InfoOcrClassBox, WriteOcr, DoOcrMulti, DoOcrSingle

Alternatives

`WriteOcrTrainf`, `AppendOcrTrainf`

Module

Optical character recognition
12.13 OCV

```c
void HMiscX.CloseAllOcvs ( )
void HOperatorSetX.CloseAllOcvs ( )
```

Clear all OCV tools.

CloseAllOcvs closes all OCV tools which have been opened using CreateOcvProj or ReadOcv. All handles are invalid after this call.

```text
Attention
This operator is only for internal use (like e.g. a reset program in HDevelop).
Result
CloseAllOcvs returns always TRUE.
```

```text
Parallelization Information
CloseAllOcvs is processed completely exclusively without parallelization.
```

```text
Possible Predecessor Functions
ReadOcv, CreateOcvProj
```

```text
Alternatives
CloseOcv
```

```text
Module
Optical character verification
```

```c
void HOperatorSetX.CloseOcv ([in] long OCVHandle )
```

Clear an OCV tool.

CloseOcv closes an open OCV tool and frees the memory. The OCV tool has been created using CreateOcvProj or ReadOcv. The handle is after this call no longer valid.

```text
Parameter
\> OCVHandle (input control) .................................................. ocv \sim HOcvX / long
Handle of the OCV tool which has to be freed.
Result
CloseOcv returns TRUE, if the handle is valid. Otherwise, an exception handling is raised.
```

```text
Parallelization Information
CloseOcv is processed completely exclusively without parallelization.
```

```text
Possible Predecessor Functions
ReadOcv, CreateOcvProj
```

```text
Alternatives
CloseOcr
```

```text
Module
Optical character verification
```

```c
void HOcvX.CreateOcvProj ([in] VARIANT PatternNames )
void HOperatorSetX.CreateOcvProj ([in] VARIANT PatternNames, [out] long OCVHandle )
```

Create a new OCV tool based on gray value projections.

CreateOcvProj creates a new OCV tool. This tool will be used to train good-patterns for the optical character verification. The training is done using the operator TraindOcvProj. Thus TraindOcvProj is normally called after CreateOcvProj.
The pattern comparison is based on the gray projections: For every training pattern the horizontal and vertical gray projections are calculated by summing up the gray values along the rows and columns inside the region of the pattern. This operation is applied to the training patterns and the test patterns. For the training patterns the result is stored inside the OCV tool to save runtime while comparing patterns. The OCV is done by comparing the corresponding projections. The Quality is the similarity of the projections.

Input for CreateOcvProj are the names of the patterns (PatternNames) which have to be trained. The number and the names can be chosen arbitrary. In most cases only one pattern will be trained, thus only one name has to be specified. The names will be used when doing the OCV (DoOcvSimple). It is possible to specify more names than actually used. These might be trained later.

To close the OCV tool, i.e. to free the memory, the operator CloseOcv is called.

Parameter

- **PatternNames** (input control) .. string(-array) ~ VARIANT ( string )
  List of names for patterns to be trained.
  Default Value : 'a'
- **OCVHandle** (output control) .. ocv ~ HOcvX / long
  Handle of the created OCV tool.

Result

CreateOcvProj returns TRUE, if the parameters are correct. Otherwise, an exception handling is raised.

Parallelization Information

CreateOcvProj is processed completely exclusively without parallelization.

Possible Successor Functions

TraindOcvProj, WriteOcv, CloseOcv

See Also

CreateOcrClassBox

Alternatives

ReadOcv

Module

Optical character verification

Verification of a pattern using an OCV tool.

DoOcvSimple evaluates the pattern in (Pattern). Before the evaluation the good-pattern has been trained by using the operator TraindOcvProj. Both patterns should have roughly the same (relative) extent and shape. To specify which of the trained patterns is used as reference its name is specified in PatternName. The next four parameters influence the automatic adaption: AdaptPos and AdaptSize refer to the geometry of the pattern. AdaptPos specifies whether a shift of the position will be adapted automatically. AdaptSize is used to adapt to changes in the size of the pattern. AdaptAngle is not yet implemented. The parameter AdaptGray controls the adaption to changes of the grayvalues. This comprises additive and multiplicative changes of the intensity.

The parameter Threshold specifies the minimum difference of the gray values to be treated as an error. In this case the percentage of wrong pixels is returned. If the value is below 0 the sum of all errors normalized with respect to the size is returned.
The result of the operator is the Quality of the pattern with a value between 0 and 1. The value 1 corresponds to a pattern with no faults. The value 0 corresponds to a very big fault.

**Parameter**

- **Pattern** (input iconic) ............................................ image(-array) \(\sim\) HImageX / IHObjectX
  Characters to be verified.
- **OCVHandle** (input control) ...................................... ocv \(\sim\) HOcvX / long
  Handle of the OCV tool.
- **PatternName** (input control) .................................... string(-array) \(\sim\) VARIANT ( string )
  Name of the character.
  Default Value : 'a'
- **AdaptPos** (input control) ....................................... string \(\sim\) String
  Adaption to vertical and horizontal translation.
  Default Value : 'true'
  Value List : AdaptPos \(\in\) {'true', 'false'}
- **AdaptSize** (input control) ....................................... string \(\sim\) String
  Adaption to vertical and horizontal scaling of the size.
  Default Value : 'true'
  Value List : AdaptSize \(\in\) {'true', 'false'}
- **AdaptAngle** (input control) ..................................... string \(\sim\) String
  Adaption to changes of the orientation (not yet implemented).
  Default Value : 'false'
  Value List : AdaptAngle \(\in\) {'false'}
- **AdaptGray** (input control) ...................................... string \(\sim\) String
  Adaption to additive and scaling gray value changes.
  Default Value : 'true'
  Value List : AdaptGray \(\in\) {'true', 'false'}
- **Threshold** (input control) ...................................... number \(\sim\) double
  Minimum difference between objects.
  Default Value : 10
  Value Suggestions : Threshold \(\in\) {-1, 0, 1, 5, 10, 15, 20, 30, 40, 50, 60, 80, 100, 150}
- **Quality** (output control) ........................................ real(-array) \(\sim\) VARIANT ( real )
  Evaluation of the character.
  Typical Range of Values : 0.0 \(\leq\) Quality \(\leq\) 0.0

**Result**

DoOcvSimple returns TRUE, if the handle and the characters are correct. Otherwise, an exception handling is raised.

**Parallelization Information**

DoOcvSimple is reentrant and automatically parallelized (on tuple level).

**Possible Predecessor Functions**

TrainOcrClassBox, TrainfOcrClassBox, ReadOcv, Threshold, Connection, SelectShape

**Possible Successor Functions**

CloseOcv

---

See Also

---

CreateOcvProj

---

Module

---

Optical character verification

---

```
void HOcvX.ReadOcv ([in] String FileName )
void HOperatorSetX.ReadOcv ([in] String FileName, [out] long OCVHandle )
```

Reading an OCV tool from file.

ReadOcv reads an OCV tool from file. The tool will contain the same information that it contained when saving it with WriteOcv. After reading the tool the training can be completed for those patterns which have not been trained so far. Otherwise a pattern comparison can be applied directly by calling DoOcvSimple.
As extension ".ocv" is used. If this extension is not given with the file name it will be added automatically.

Parameter

- **FileName** (input control) .......................................................... filename \( \rightsquigarrow \) String
  Name of the file which has to be read.
  Default Value: 'test.ocv'

- **OCVHandle** (output control) .................................................... ocv \( \rightsquigarrow \) HOcvX / long
  Handle of read OCV tool.

ReadOcv returns TRUE, if the file is correct. If the file could not be opened ReadOcv returns FAIL. Otherwise, an exception handling is raised.

Parallelization Information
ReadOcv is processed completely exclusively without parallelization.

Possible Predecessor Functions
WriteOcv

Possible Successor Functions
DoOcvSimple, CloseOcv

See Also
ReadOcr

Module
Optical character verification

```c
void HImageX.TrainOcvProj ([in] HOcvX OCVHandle, [in] VARIANT Name, [in] String Mode )


```

Training of an OCV tool.

TrainOcvProj trains patterns for an OCV tool that has been created using the operators CreateOcvProj or ReadOcv. For this training one or multiple patterns are offered the system. Such a pattern consists of an image with a reduced domain (ROI) for the area of the pattern. Note that the pattern should not only contain foreground pixels (e.g. dark pixels of a character) but also background pixels. This can be implemented e.g. by the smallest surrounding rectangle of the pattern. Without this context an evaluation of the pattern is not possible.

If more than one pattern has to be trained this can be achieved by multiple calls (one for each pattern) or by calling TrainOcvProj once with all patterns and a tuple of the corresponding names. The result will be in both cases the same. However using multiple calls will normally result in a longer execution time than using one call with all patterns.

Parameter

- **Pattern** (input iconic) ......................................................... image(-array) \( \rightsquigarrow \) HImageX / IObjectX
  Pattern to be trained.

- **OCVHandle** (input control) .................................................. ocv \( \rightsquigarrow \) HOcvX / long
  Handle of the OCV tool to be trained.

- **Name** (input control) .......................................................... string(-array) \( \rightsquigarrow \) VARIANT( string )
  Name(s) of the object(s) to analyse.
  Default Value: 'a'

- **Mode** (input control) .......................................................... string \( \rightsquigarrow \) String
  Mode for training (only one mode implemented).
  Default Value: 'single'
  Value List: Mode \( \in \) \{ 'single' \}
TrainOcvProj returns TRUE, if the handle and the training pattern(s) are correct. Otherwise, an exception handling is raised.

Parallelization Information
TrainOcvProj is processed completely exclusively without parallelization.

Possible Predecessor Functions
WriteOcrTrainf, CreateOcvProj, ReadOcv, Threshold, Connection, SelectShape

Possible Successor Functions
CloseOcv

See Also
TrainOcrClassBox

Module
Optical character verification

void HOcvX.WriteOcv ([in] String FileName )
void HOperatorSetX.WriteOcv ([in] long OCVHandle, [in] String FileName )

Saving an OCV tool to file.
WriteOcv writes an OCV tool to file. This can be used to save the result of a training (TrainOcvProj). The whole information contained in the OCV tool is stored in the file. The file can be reloaded afterwards using the operator ReadOcv.

As file extension `.ocv' is used. If this extension is not given with the file name, it will be added automatically.

Parameter
- **OCVHandle** (input control) ......................................................... ocv ~ HOcvX/long Handle of the OCV tool to be written.
- **FileName** (input control) ........................................................ filename ~ String Name of the file where the tool has to be saved.
  Default Value: 'test.ocv'

Result
WriteOcv returns TRUE, if the data is correct and the file can be written. Otherwise, an exception handling is raised.

Parallelization Information
WriteOcv is reentrant and processed without parallelization.

Possible Predecessor Functions
TrainOcvProj

Possible Successor Functions
CloseOcv

See Also
WriteOcr

Module
Optical character verification
12.14  Shape-from

```plaintext
[out] HImageX Depth  HImageX.DepthFromFocus  ([out] HImageX Confidence,

void HOperatorSetX.DepthFromFocus  ([in] IHObjectX MultiFocusImage,
[out] HUntypedObjectX Depth, [out] HUntypedObjectX Confidence,
```

Extract depth using multiple focus levels.

The operator DepthFromFocus extracts the depth using a focus sequence. The images of the focus sequence have to passed as a multi channel image (MultiFocusImage). The depth for each pixel will be returned in Depth as the channel number. The parameter Confidence returns a confidence value for each depth estimation: The larger this value, the higher the confidence of the depth estimation is.

DepthFromFocus selects the pixels with the best focus of all focus levels. The method used to extract these pixels is specified by the parameters Filter and Selection:

'highpass'  The value of the focus is estimated by a highpass filter.

'bandpass'  The value of the focus is estimated by a bandpass filter.

'next maximum'  To decide which focus level has been selected, the pixel in the neighborhood with the best confidence is used to determine this value.

'local'  The decision for a focus level is based only on the locally calculated focus values.

Parameter

- **MultiFocusImage** (input iconic)  multichannel-image-array  Multichannel gray image consisting of multiple focus levels.
- **Depth** (output iconic)  image(-array)  Depth image.
- **Confidence** (output iconic)  image(-array)  Confidence of depth estimation.
- **Filter** (input control)  string(-array)  Filter used to find sharp pixels.
  Default Value: 'highpass'
  Value List: Filter ∈ {'highpass', 'bandpass'}
- **Selection** (input control)  string(-array)  Method used to find sharp pixels.
  Default Value: 'next maximum'
  Value List: Selection ∈ {'next maximum', 'local'}

Parallelization Information

DepthFromFocus is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

Compose2, Compose3, Compose4, AddChannels, ReadImage, ReadSequence

Possible Successor Functions

SelectGrayvaluesFromChannels, MeanImage, GaussImage, Threshold

See Also

CountChannels

Module

Tools

```plaintext
[out] VARIANT Albedo  HImageX.EstimateAlAm  ([out] VARIANT Ambient )

void HOperatorSetX.EstimateAlAm  ([in] IHObjectX Image,
[out] VARIANT Albedo, [out] VARIANT Ambient )
```

Estimate the albedo of a surface and the amount of ambient light.

EstimateAlAm estimates the Albedo of a surface, i.e., the percentage of light reflected by the surface, and the amount of ambient light Ambient by using the maximum and minimum gray values of the image.

**Attention**

It is assumed that the image contains at least one point for which the reflection function assumes its minimum, e.g., points in shadows. Furthermore, it is assumed that the image contains at least one point for which the reflection function assumes its maximum. If this is not the case, wrong values will be estimated.

**Parameter**

- **Image** (input iconic) ................................. image(-array) \(\rightsquigarrow\) HImageX / IHObjectX  
  Image for which albedo and ambient are to be estimated.
- **Albedo** (output control) ............................... real(-array) \(\rightsquigarrow\) VARIANT (real)  
  Amount of light reflected by the surface.
- **Ambient** (output control) ............................ real(-array) \(\rightsquigarrow\) VARIANT (real)  
  Amount of ambient light.

**Result**

EstimateAlAm always returns the value TRUE.

**Parallelization Information**

EstimateAlAm is reentrant and automatically parallelized (on tuple level).

**Possible Successor Functions**

SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

**Tools**

```c
[out] VARIANT Slant HImageX.EstimateSlAlLr ([out] VARIANT Albedo )
```

Estimate the slant of a light source and the albedo of a surface.

EstimateSlAlLr estimates the Slant of a light source, i.e., the angle between the light source and the positive z-axis, and the albedo of the surface in the input image Image, i.e., the percentage of light reflected by the surface, using the algorithm of Lee and Rosenfeld.

**Attention**

The Albedo is assumed constant for the entire surface depicted in the image.

**Parameter**

- **Image** (input iconic) ................................. image(-array) \(\rightsquigarrow\) HImageX / IHObjectX  
  Image for which slant and albedo are to be estimated.
- **Slant** (output control) ............................... angle.deg(-array) \(\rightsquigarrow\) VARIANT (real)  
  Angle between the light sources and the positive z-axis (in degrees).
- **Albedo** (output control) ............................ real(-array) \(\rightsquigarrow\) VARIANT (real)  
  Amount of light reflected by the surface.

**Result**

EstimateSlAlLr always returns the value TRUE.

**Parallelization Information**

EstimateSlAlLr is reentrant and automatically parallelized (on tuple level).

**Possible Successor Functions**

SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

**Tools**

HALCON 6.0
Estimate the slant of a light source and the albedo of a surface.

EstimateSlAlZc estimates the slant of a light source, i.e. the angle between the light source and the positive z-axis, and the albedo of the surface in the input image Image, i.e. the percentage of light reflected by the surface, using the algorithm of Zheng and Chellappa.

Attention

The Albedo is assumed constant for the entire surface depicted in the image.

Parameter

- **Image** (input iconic) .................. image(-array) \(\sim\) HImageX / IObjectX Image for which slant and albedo are to be estimated.
- **Slant** (output control) .................. angle.deg(-array) \(\sim\) VARIANT (real) Angle of the light sources and the positive z-axis (in degrees).
- **Albedo** (output control) ................. real(-array) \(\sim\) VARIANT (real) Amount of light reflected by the surface.

Result

EstimateSlAlZc always returns the value TRUE.

Parallelization Information

EstimateSlAlZc is reentrant and automatically parallelized (on tuple level).

Possible Successor Functions

SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

Tools

Estimate the tilt of a light source.

EstimateTiltLr estimates the tilt of a light source, i.e. the angle between the light source and the x-axis after projection into the xy-plane, from the image Image using the algorithm of Lee and Rosenfeld.

Parameter

- **Image** (input iconic) .................. image(-array) \(\sim\) HImageX / IObjectX Image for which the tilt is to be estimated.
- **Tilt** (output control) .................. angle.deg(-array) \(\sim\) VARIANT (real) Angle between the light source and the x-axis after projection into the xy-plane (in degrees).

Result

EstimateTiltLr always returns the value TRUE.

Parallelization Information

EstimateTiltLr is reentrant and automatically parallelized (on tuple level).

Possible Successor Functions

SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

Tools
Estimate the tilt of a light source.

EstimateTiltZc estimates the tilt of a light source, i.e., the angle between the light source and the x-axis after projection into the xy-plane, from the image Image using the algorithm of Zheng and Chellappa.

Parameter

- **Image** (input iconic) \( \text{image-array} \) \( \sim \) \( \text{HImageX / IObjectX} \)
  Image for which the tilt is to be estimated.

- **Tilt** (output control) \( \text{angle.deg-array} \) \( \sim \) \( \text{VARIANT (real)} \)
  Angle between the light source and the x-axis after projection into the xy-plane (in degrees).

Result

EstimateTiltZc always returns the value TRUE.

Parallelization Information

EstimateTiltZc is reentrant and automatically parallelized (on tuple level).

Possible Successor Functions

SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

Tools

Reconstruct a surface from at least three gray value images.

PhotStereo reconstructs a surface (i.e., the relative height of each image point) using the algorithm of Woodham from at least three gray value images given by the multi-channel image Images. The light sources corresponding to the individual images are given by the parameters Slants and Tilts and are assumed to lie infinitely far away.

Attention

PhotStereo assumes that the heights are to be extracted on a lattice with step width 1. If this is not the case, the calculated heights must be multiplied by the step width after the call to PhotStereo. A Cartesian coordinate system with the origin in the lower left corner of the image is used internally. Since the operator is based on the Fast Fourier Transform, only square images with an edge length being a power of 2 are accepted. All given images must be byte-images. At least three images must be given in a multi-channel image. Slants and Tilts must contain exactly as many light sources as the number of channels in Images. At least three of the light source directions must be linearly independent.

Parameter

- **Images** (input iconic) \( \text{multichannel-image} \) \( \sim \) \( \text{HImageX / IObjectX} \)
  Shaded input image with at least three channels.

- **Height** (output iconic) \( \text{image} \) \( \sim \) \( \text{HImageX / HUntypedObjectX} \)
  Reconstructed height field.

- **Slants** (input control) \( \text{angle.deg} \) \( \sim \) \( \text{VARIANT (real, integer)} \)
  Angle between the light sources and the positive z-axis (in degrees).

Default Value: 45.0

Value Suggestions: \( \text{Slants} \in \{ 1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0 \} \)

Typical Range of Values: \( 0.0 \leq \text{Slants} \leq 0.0(\text{lin}) \)

Minimal Value Step: 0.01

Recommended Value Step: 10.0

HALCON 6.0
**Tilts** (input control) ................................. angle.deg $\rightarrow$ VARIANT ( real, integer )
Angle between the light source and the x-axis after projection into the xy-plane (in degrees).

**Default Value**: 45.0

**Value Suggestions**: Tilts $\in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}$

**Typical Range of Values**: 0.0 $\leq$ Tilts $\leq$ 0.0 (lin)

**Minimal Value Step**: 0.01

**Recommended Value Step**: 10.0

---

**Result**

If all parameters are correct PhotStereo returns the value TRUE. Otherwise, an exception is raised.

---

**Parallelization Information**

PhotStereo is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

EstimateSlAllr, EstimateSlAllzc, EstimateTiltLr, EstimateTiltZc

---

**Possible Successor Functions**

ShadeHeightField

---

**Tools**

```haskell
[out] HImageX Selected HImageX.SelectGrayvaluesFromChannels
([in] HImageX SelectGrayvaluesFromChannels )

void HOperatorSetX.SelectGrayvaluesFromChannels
([in] IHObjectX MultichannelImage, [in] IHObjectX IndexImage,
[out] HUntypedObjectX Selected )
```

Selection of gray values of a multi channel image using an index image.

The operator SelectGrayvaluesFromChannels selects gray values from a MultichannelImage. The channel number for each pixel is determined by using IndexImage: The gray value in IndexImage is used as the channel number in MultichannelImage.

---

**Parameter**

- **MultichannelImage** (input iconic) .................................. multichannel-image-array $\rightarrow$ HImageX / IHObjectX ( byte )
  Multichannel gray image.

- **IndexImage** (input iconic) ................................. image(-array) $\rightarrow$ HImageX / IHObjectX ( byte )
  Image, where gray values are interpreted as indexes.

- **Selected** (output iconic) ................................. image(-array) $\rightarrow$ HImageX / HUntypedObjectX ( byte )
  Depth image.

---

**Parallelization Information**

SelectGrayvaluesFromChannels is reentrant and automatically parallelized (on tuple level, domain level).

---

**Possible Predecessor Functions**

DepthFromFocus, MeanImage

---

**Possible Successor Functions**

DispImage

---

**See Also**

CountChannels

---

**Tools**
Reconstruct a surface from a gray value image.

SfsModLr reconstructs a surface (i.e. the relative height of each image point) using the modified algorithm of Lee and Rosenfeld. The surface is reconstructed from the input image Image, and the light source given by the parameters Slant, Tilt, Albedo and Ambient, and is assumed to lie infinitely far away in the direction given by Slant and Tilt. The parameter Albedo determines the albedo of the surface, i.e. the percentage of light reflected in all directions. Ambient determines the amount of ambient light falling onto the surface. It can be set to values greater than zero if, for example, the white balance of the camera was badly adjusted at the moment the image was taken.

Attention

SfsModLr assumes that the heights are to be extracted on a lattice with step width 1. If this is not the case, the calculated heights must be multiplied with the step width after the call to SfsModLr. A Cartesian coordinate system with the origin in the lower left corner of the image is used internally. Since the operator is based on the Fast Fourier Transform, only square images with the edge length being a power of 2 are accepted. SfsModLr can only handle byte-images.

Parameter

- **Image** (input iconic) ............................. image(-array) HImageX / IHObjectX
  Shaded input image.

- **Height** (output iconic) ............................. image(-array) HImageX / HUntypedObjectX
  Reconstructed height field.

- **Slant** (input control) ......................... angle.deg VARIANT ( real, integer )
  Angle between the light source and the positive z-axis (in degrees).
  Default Value : 45.0
  Value Suggestions : Slant ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typical Range of Values : 0.0 ≤ Slant ≤ 0.0(lin)
  Minimal Value Step : 0.01
  Recommended Value Step : 10.0

- **Tilt** (input control) .............................. angle.deg VARIANT ( real, integer )
  Angle between the light source and the x-axis after projection into the xy-plane (in degrees).
  Default Value : 45.0
  Value Suggestions : Tilt ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typical Range of Values : 0.0 ≤ Tilt ≤ 0.0(lin)
  Minimal Value Step : 0.01
  Recommended Value Step : 10.0

- **Albedo** (input control) .......................... number VARIANT ( real, integer )
  Amount of light reflected by the surface.
  Default Value : 1.0
  Value Suggestions : Albedo ∈ {0.1, 0.5, 1.0, 5.0}
  Typical Range of Values : 0.0 ≤ Albedo ≤ 0.0(lin)
  Minimal Value Step : 0.01
  Recommended Value Step : 0.1
  Restriction : (Albedo ≥ 0.0)

- **Ambient** (input control) ......................... number VARIANT ( real, integer )
  Amount of ambient light.
  Default Value : 0.0
  Value Suggestions : Ambient ∈ {0.1, 0.5, 1.0}
  Typical Range of Values : 0.0 ≤ Ambient ≤ 0.0(lin)
  Minimal Value Step : 0.01
  Recommended Value Step : 0.1
  Restriction : (Ambient ≥ 0.0)

Result

If all parameters are correct SfsModLr returns the value TRUE. Otherwise, an exception is raised.
Parallelization Information

SfsModLr is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

EstimateAlAm, EstimateSlAlLr, EstimateSlAlZc, EstimateTiltLr, EstimateTiltZc

Possible Successor Functions

ShadeHeightField

Module

Tools

Reconstruct a surface from a gray value image.

SfsOrigLr reconstructs a surface (i.e. the relative height of each image point) using the original algorithm of Lee and Rosenfeld. The surface is reconstructed from the input image Image. The light source is to be given by the parameters Slant, Tilt, Albedo and Ambient, and is assumed to lie infinitely far away in the direction given by Slant and Tilt. The parameter Albedo determines the albedo of the surface, i.e. the percentage of light reflected in all directions. Ambient determines the amount of ambient light falling onto the surface. It can be set to values greater than zero if, for example, the white balance of the camera was badly adjusted at the moment the image was taken.

Attention

SfsOrigLr assumes that the heights are to be extracted on a lattice with step width 1. If this is not the case, the calculated heights must be multiplied with the step width after the call to SfsOrigLr. A Cartesian coordinate system with the origin in the lower left corner of the image is used internally. Since the operator is based on the Fast Fourier Transform, only square images with the edge length being a power of 2 are accepted. SfsOrigLr can only handle byte-images.

Parameter

- **Image** (input iconic) .................................................. image(-array) \( \rightarrow \) HImageX / IHObjectX
  Shaded input image.

- **Height** (output iconic) ........................................... image(-array) \( \rightarrow \) HI mageX / H UntypedObjectX
  Reconstructed height field.

- **Slant** (input control) ............................................. angle.deg \( \rightarrow \) VARIANT (real, integer)
  Angle between the light source and the positive z-axis (in degrees).
  Default Value : 45.0
  Value Suggestions : Slant \( \in \{ 1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0 \} \)
  Typical Range of Values : \( 0.0 \leq \text{Slant} \leq 0.0 \text{(lin)} \)
  Minimal Value Step : 0.01
  Recommended Value Step : 10.0

- **Tilt** (input control) ............................................... angle.deg \( \rightarrow \) VARIANT (real, integer)
  Angle between the light source and the x-axis after projection into the xy-plane (in degrees).
  Default Value : 45.0
  Value Suggestions : Tilt \( \in \{ 1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0 \} \)
  Typical Range of Values : \( 0.0 \leq \text{Tilt} \leq 0.0 \text{(lin)} \)
  Minimal Value Step : 0.01
  Recommended Value Step : 10.0
Albedo (input control) ........................................... number → VARIANT (real, integer)
Amount of light reflected by the surface.
Default Value: 1.0
Value Suggestions: Albedo ∈ {0.1, 0.5, 1.0, 5.0}
Typical Range of Values: 0.0 ≤ Albedo ≤ 0.0(lin)
Minimal Value Step: 0.01
Recommended Value Step: 0.1
Restriction: (Albedo ≥ 0.0)

Ambient (input control) ............................ number → VARIANT (real, integer)
Amount of ambient light.
Default Value: 0.0
Value Suggestions: Ambient ∈ {0.1, 0.5, 1.0}
Typical Range of Values: 0.0 ≤ Ambient ≤ 0.0(lin)
Minimal Value Step: 0.01
Recommended Value Step: 0.1
Restriction: (Ambient ≥ 0.0)

Result
If all parameters are correct SfsOrigLr returns the value TRUE. Otherwise, an exception is raised.

Parallelization Information
SfsOrigLr is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions
EstimateAlAm, EstimateSlAlLr, EstimateSlAlZc, EstimateTiltLr, EstimateTiltZc

Possible Successor Functions
ShadeHeightField

Module

Tools

[out] HImageX Height HImageX.SfsPentland ([in] VARIANT Slant,

void HOperatorSetX.SfsPentland ([in] IObjectX Image,
[out] HUntypedObjectX Height, [in] VARIANT Slant, [in] VARIANT Tilt,

Reconstruct a surface from a gray value image.
SfsPentland reconstructs a surface (i.e. the relative height of each image point) using the algorithm of Pentland. The surface is reconstructed from the input image Image. The light source must be given by the parameters Slant, Tilt, Albedo and Ambient, and is assumed to lie infinitely far away in the direction given by Slant and Tilt. The parameter Albedo determines the albedo of the surface, i.e. the percentage of light reflected in all directions. Ambient determines the amount of ambient light falling onto the surface. It can be set to values greater than zero if, for example, the white balance of the camera was badly adjusted at the moment the image was taken.

Attention
SfsPentland assumes that the heights are to be extracted on a lattice with step width 1. If this is not the case, the calculated heights must be multiplied with the step width after the call to SfsPentland. A Cartesian coordinate system with the origin in the lower left corner of the image is used internally. Since the operator is based on the Fast Fourier Transform, only square images with the edge length being a power of 2 are accepted. SfsPentland can only handle byte-images.

Parameter

Image (input iconic) ........................... image(-array) → HImageX / IObjectX
Shaded input image.

Height (output iconic) ................................. image(-array) → HImageX / HUntypedObjectX
Reconstructed height field.
\( \textbf{Slant} \) (input control) \( \text{angle.deg} \) \( \rightarrow \text{VARIANT (real, integer)} \)

Angle between the light source and the positive z-axis (in degrees).

- **Default Value**: 45.0
- **Value Suggestions**: \( \text{Slant} \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\} \)
- **Typical Range of Values**: \( 0.0 \leq \text{Slant} \leq 0.0\) (lin)
- **Minimal Value Step**: 1.0
- **Recommended Value Step**: 10.0

\( \textbf{Tilt} \) (input control) \( \text{angle.deg} \) \( \rightarrow \text{VARIANT (real, integer)} \)

Angle between the light source and the x-axis after projection into the xy-plane (in degrees).

- **Default Value**: 45.0
- **Value Suggestions**: \( \text{Tilt} \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\} \)
- **Typical Range of Values**: \( 0.0 \leq \text{Tilt} \leq 0.0\) (lin)
- **Minimal Value Step**: 1.0
- **Recommended Value Step**: 10.0

\( \textbf{Albedo} \) (input control) \( \text{number} \) \( \rightarrow \text{VARIANT (real, integer)} \)

Amount of light reflected by the surface.

- **Default Value**: 1.0
- **Value Suggestions**: \( \text{Albedo} \in \{0.1, 0.5, 1.0, 5.0\} \)
- **Typical Range of Values**: \( 0.0 \leq \text{Albedo} \leq 0.0\) (lin)
- **Minimal Value Step**: 0.01
- **Recommended Value Step**: 0.1
- **Restriction**: \( \text{Albedo} \geq 0.0 \)

\( \textbf{Ambient} \) (input control) \( \text{number} \) \( \rightarrow \text{VARIANT (real, integer)} \)

Amount of ambient light.

- **Default Value**: 0.0
- **Value Suggestions**: \( \text{Ambient} \in \{0.1, 0.5, 1.0\} \)
- **Typical Range of Values**: \( 0.0 \leq \text{Ambient} \leq 0.0\) (lin)
- **Minimal Value Step**: 0.01
- **Recommended Value Step**: 0.1
- **Restriction**: \( \text{Ambient} \geq 0.0 \)

---

**Result**

If all parameters are correct \textit{SfsPentland} returns the value TRUE. Otherwise, an exception is raised.

---

**Parallelization Information**

\textit{SfsPentland} is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessor Functions**

EstimateAlAm, EstimateSlAlLr, EstimateSlAlZc, EstimateTiltLr, EstimateTiltZc

---

**Possible Successor Functions**

ShadeHeightField

---

**Tools**


---

Shade a height field.

\textit{ShadeHeightField} computes a shaded image from the height field \textit{ImageHeight} as if the image were illuminated by an infinitely far away light source. It is assumed that the surface described by the height field has Lambertian reflection properties determined by \textit{Albedo} and \textit{Ambient}. The parameter \textit{Shadows} determines whether shadows are to be calculated.

---

**Attention**

\textit{ShadeHeightField} assumes that the heights are given on a lattice with step width 1. If this is not the case, the
Heights must be divided by the step width before the call to ShadeHeightField. Otherwise, the derivatives used internally to compute the orientation of the surface will be estimated too steep or too flat. Example: The height field is given on 100*100 points on the square \([0,1]*[0,1]\). Then the heights must be divided by \(1/100\) first. A Cartesian coordinate system with the origin in the lower left corner of the image is used internally.

### Parameter

- **ImageHeight** (input iconic) .......................... image(-array) \(\sim HImageX / HObjectX\)
  - Height field to be shaded.

- **ImageShade** (output iconic) .......................... image(-array) \(\sim HImageX / HUntypedObjectX\)
  - Shaded image.

- **Slant** (input control) .............................. angle.deg \(\sim\) VARIANT (real, integer)
  - Angle between the light source and the positive z-axis (in degrees).
  - Default Value: \(0.0\)
  - Value Suggestions: \(Slant \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}\)
  - Typical Range of Values: \(0.0 \leq Slant \leq 0.0\) (lin)
  - Minimal Value Step: \(0.01\)
  - Recommended Value Step: \(10.0\)

- **Tilt** (input control) .............................. angle.deg \(\sim\) VARIANT (real, integer)
  - Angle between the light source and the x-axis after projection into the xy-plane (in degrees).
  - Default Value: \(0.0\)
  - Value Suggestions: \(Tilt \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}\)
  - Typical Range of Values: \(0.0 \leq Tilt \leq 0.0\) (lin)
  - Minimal Value Step: \(0.01\)
  - Recommended Value Step: \(10.0\)

- **Albedo** (input control) .............................. number \(\sim\) VARIANT (real, integer)
  - Amount of light reflected by the surface.
  - Default Value: \(1.0\)
  - Value Suggestions: \(Albedo \in \{0.1, 0.5, 1.0, 5.0\}\)
  - Typical Range of Values: \(0.0 \leq Albedo \leq 0.0\) (lin)
  - Minimal Value Step: \(0.01\)
  - Recommended Value Step: \(0.1\)
  - Restriction: \((Albedo \geq 0.0)\)

- **Ambient** (input control) .............................. number \(\sim\) VARIANT (real, integer)
  - Amount of ambient light.
  - Default Value: \(0.0\)
  - Value Suggestions: \(Ambient \in \{0.1, 0.5, 1.0\}\)
  - Typical Range of Values: \(0.0 \leq Ambient \leq 0.0\) (lin)
  - Minimal Value Step: \(0.01\)
  - Recommended Value Step: \(0.1\)
  - Restriction: \((Ambient \geq 0.0)\)

- **Shadows** (input control) ............................... string \(\sim\) String
  - Should shadows be calculated?
  - Default Value: 'false'
  - Value Suggestions: \(Shadows \in \{'true', 'false'\}\)

### Result

If all parameters are correct ShadeHeightField returns the value TRUE. Otherwise, an exception is raised.

### Parallelization Information

ShadeHeightField is reentrant and automatically parallelized (on tuple level).

### Possible Predecessor Functions

SfsModLr, SfsOrigLr, SfsPentland, PhotStereo

### Module

Tools

HALCON 6.0
Chapter 13

Tuple

13.1 Arithmetic

```
[out] VARIANT Abs HTupleX.TupleAbs ([in] VARIANT T )
void HOperatorSetX.TupleAbs ([in] VARIANT T, [out] VARIANT Abs )
```

Compute the absolute value of a tuple.
TupleAbs computes the absolute value of the input tuple T. The absolute value of an integer number is again an integer number. The absolute value of a floating point number is a floating point number. The absolute value of a string is not allowed.

---

**Parameter**

- ▶ T (input control) .................................................. number(-array) \(\sim\) VARIANT ( real, integer )
  Input tuple.
- ▶ Abs (output control) .................................................. number(-array) \(\sim\) VARIANT ( real, integer )
  Absolute value of the input tuple.

---

Parallelization Information

TupleAbs is reentrant and processed without parallelization.

---

Alternatives

- TupleFabs

---

Module

Operators not requiring licensing

```
[out] VARIANT ACos HTupleX.TupleAcos ([in] VARIANT T )
void HOperatorSetX.TupleAcos ([in] VARIANT T, [out] VARIANT ACos )
```

Compute the arccosine of a tuple.
TupleAcos computes the arccosine of the input tuple T. The arccosine is always returned as a floating point number. The arccosine of a string is not allowed.

---

**Parameter**

- ▶ T (input control) .................................................. number(-array) \(\sim\) VARIANT ( real, integer )
  Input tuple.
- **Restriction**: \( (-1 \leq T ) \leq 1 \)
- ▶ ACos (output control) .................................................. number(-array) \(\sim\) VARIANT ( real )
  Arccosine of the input tuple.

---

Parallelization Information

TupleAcos is reentrant and processed without parallelization.
Add two tuples.

**TupleAdd** computes the sum of the input tuples `S1` and `S2`. If both tuples have the same length the corresponding elements of both tuples are added. Otherwise, either `S1` or `S2` must have length 1. In this case, the addition is performed for each element of the longer tuple with the single element of the other tuple. If two integer numbers are added, the result is again an integer number. If a floating point number is added to another number, the result is a floating point number. If two strings are added, the addition corresponds to a string concatenation. If a number and a string are added, the number is converted to a string first. Thus, the addition also corresponds to a string concatenation in this case.

**Parameter**

- `S1` (input control) .................................................. number(-array) `~` VARIANT ( real, integer, string ) 
  Input tuple 1.
- `S2` (input control) .................................................. number(-array) `~` VARIANT ( real, integer, string ) 
  Input tuple 2.
- `Sum` (output control) ............................................... number(-array) `~` VARIANT ( real, integer, string ) 
  Sum of the input tuples.

**Parallelization Information**

**TupleAdd** is reentrant and processed without parallelization.

**TupleSub**

Operators not requiring licensing

**Parameter**

- `T` (input control) .................................................. number(-array) `~` VARIANT ( real, integer ) 
  Input tuple.
  **Restriction** : `([-1 ≤ T] ≤ 1]`
- `ASin` (output control) ............................................... number(-array) `~` VARIANT ( real ) 
  Arcsine of the input tuple.
Compute the arctangent of a tuple.

TupleAtan computes the arctangent of the input tuple T. The arctangent is always returned as a floating point number. The arctangent of a string is not allowed.

Parameter

- **T** (input control) : number(-array) \(\rightarrow\) VARIANT (real, integer)
  - Input tuple.
- **ATan** (output control) : number(-array) \(\rightarrow\) VARIANT (real)
  - Arctangent of the input tuple.

Parallelization Information

TupleAtan is reentrant and processed without parallelization.

See Also

TupleTan

Alternatives

TupleAtan2, TupleAsin, TupleAcos

Operators not requiring licensing

Compute the arctangent of a tuple for all four quadrants.

TupleAtan2 computes the arctangent of the input tuples Y/X while treating all four quadrants correctly. The arctangent is always returned as a floating point number. The arctangent of a string is not allowed.

Parameter

- **Y** (input control) : number(-array) \(\rightarrow\) VARIANT (real, integer)
  - Input tuple of the y-values.
- **X** (input control) : number(-array) \(\rightarrow\) VARIANT (real, integer)
  - Input tuple of the x-values.
- **ATan** (output control) : number(-array) \(\rightarrow\) VARIANT (real)
  - Arctangent of the input tuple.

Parallelization Information

TupleAtan2 is reentrant and processed without parallelization.

See Also

TupleTan

Alternatives

TupleAtan, TupleAsin, TupleAcos

Operators not requiring licensing
### TupleCos

**Function**

```c
void HOperatorSetX.TupleCos ( [in] VARIANT T, [out] VARIANT Cos )
```

**Description**

Compute the cosine of a tuple.

TupleCos computes the cosine of the input tuple \( T \). The cosine is always returned as a floating point number. The cosine of a string is not allowed.

**Parameter**

- \( T \) (input control) : number(-array) \( \sim \) VARIANT ( real, integer )
  - Input tuple.
- \( Cos \) (output control) : number(-array) \( \sim \) VARIANT ( real )
  - Cosine of the input tuple.

**Parallelization Information**

TupleCos is **reentrant** and processed **without** parallelization.

**See Also**

TupleAcos

**Alternatives**

TupleSin, TupleTan

### TupleCosh

**Function**

```c
void HOperatorSetX.TupleCosh ( [in] VARIANT T, [out] VARIANT Cosh )
```

**Description**

Compute the hyperbolic cosine of a tuple.

TupleCosh computes the hyperbolic cosine of the input tuple \( T \). The hyperbolic cosine is always returned as a floating point number. The hyperbolic cosine of a string is not allowed.

**Parameter**

- \( T \) (input control) : number(-array) \( \sim \) VARIANT ( real, integer )
  - Input tuple.
- \( Cosh \) (output control) : number(-array) \( \sim \) VARIANT ( real )
  - Hyperbolic cosine of the input tuple.

**Parallelization Information**

TupleCosh is **reentrant** and processed **without** parallelization.

**Alternatives**

TupleSinh, TupleTanh

### TupleDiv

**Function**

```c
```

**Description**

Divide two tuples.

TupleDiv computes the quotient of the input tuples \( Q1 \) and \( Q2 \). If both tuples have the same length the corresponding elements of both tuples are divided. Otherwise, either \( Q1 \) or \( Q2 \) must have length 1. In this case, the division is performed for each element of the longer tuple with the single element of the other tuple. If two integer numbers are divided, the result is again an integer number. If one of the operands is a floating point number, the result is a floating point number. The division of strings is not allowed.
Parameter

- Q1 (input control) .......................... number(-array) \( \mapsto \) VARIANT (real, integer)
  Input tuple 1.
- Q2 (input control) .......................... number(-array) \( \mapsto \) VARIANT (real, integer)
  Input tuple 2.
- Quot (output control) .......................... number(-array) \( \mapsto \) VARIANT (real, integer)
  Quotient of the input tuples.

Parallelization Information

TupleDiv is reentrant and processed without parallelization.

Alternatives

TupleMult

Module

Operators not requiring licensing

\[
\begin{align*}
\text{[out]} & \text{ VARIANT } \text{Exp} \quad \text{HTupleX.TupleExp} \quad ([\text{in}] \text{ VARIANT } T) \\
\text{void} & \quad \text{HOperatorSetX.TupleExp} \quad ([\text{in}] \text{ VARIANT } T, [\text{out}] \text{ VARIANT } \text{Exp})
\end{align*}
\]

Compute the exponential of a tuple.

TupleExp computes the exponential of the input tuple \( T \). The exponential is always returned as a floating point number. The exponential of a string is not allowed.

Parameter

- \( T \) (input control) .......................... number(-array) \( \mapsto \) VARIANT (real, integer)
  Input tuple.
- \( \text{Exp} \) (output control) .......................... number(-array) \( \mapsto \) VARIANT (real)
  Exponential of the input tuple.

Parallelization Information

TupleExp is reentrant and processed without parallelization.

See Also

TupleLog, TupleLog10

Alternatives

TuplePow

Module

Operators not requiring licensing

\[
\begin{align*}
\text{[out]} & \text{ VARIANT } \text{Abs} \quad \text{HTupleX.TupleFabs} \quad ([\text{in}] \text{ VARIANT } T) \\
\text{void} & \quad \text{HOperatorSetX.TupleFabs} \quad ([\text{in}] \text{ VARIANT } T, [\text{out}] \text{ VARIANT } \text{Abs})
\end{align*}
\]

Compute the absolute value of a tuple (as floating point numbers).

TupleFabs computes the absolute value of the input tuple \( T \). In contrast to TupleAbs, the absolute value is always returned as a floating point number by TupleFabs. The absolute value of a string is not allowed.

Parameter

- \( T \) (input control) .......................... number(-array) \( \mapsto \) VARIANT (real, integer)
  Input tuple.
- \( \text{Abs} \) (output control) .......................... number(-array) \( \mapsto \) VARIANT (real)
  Absolute value of the input tuple.

Parallelization Information

TupleFabs is reentrant and processed without parallelization.
Calculate the remainder of the floating point division of two tuples.

TupleFmod computes the remainder of the floating point division of the input tuples \( T_1/T_2 \). If both tuples have the same length the division is performed for the corresponding elements of both tuples. Otherwise, either \( T_1 \) or \( T_2 \) must have length 1. In this case, the division is performed for each element of the longer tuple with the single element of the other tuple. The result is always a floating point number. The division of strings is not allowed.

**Parameter**

- **\( T_1 \) (input control)** 
  Number(-array) \( \sim \) VARIANT (real, integer) 
  Input tuple 1.

- **\( T_2 \) (input control)** 
  Number(-array) \( \sim \) VARIANT (real, integer) 
  Input tuple 2.

  **Restriction:** \( (T_2 \neq 0.0) \)

- **\( \text{Fmod} \) (output control)** 
  Number(-array) \( \sim \) VARIANT (real) 
  Remainder of the division of the input tuples.

**Parallelization Information**

TupleFmod is reentrant and processed without parallelization.

See Also

TupleFloor, TupleCeil

Operators not requiring licensing

---

Calculate the ldexp function of two tuples.

TupleLdexp computes the ldexp function of the input tuples, i.e., \( T_1 \times 2^{T_2} \). If both tuples have the same length the operation is performed for the corresponding elements of both tuples. Otherwise, either \( T_1 \) or \( T_2 \) must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The result is always a floating point number. The ldexp function of strings is not allowed.

**Parameter**

- **\( T_1 \) (input control)** 
  Number(-array) \( \sim \) VARIANT (real, integer) 
  Input tuple 1.

- **\( T_2 \) (input control)** 
  Number(-array) \( \sim \) VARIANT (real, integer) 
  Input tuple 2.

- **\( \text{Ldexp} \) (output control)** 
  Number(-array) \( \sim \) VARIANT (real) 
  Ldexp function of the input tuples.

**Parallelization Information**

TupleLdexp is reentrant and processed without parallelization.

See Also

TupleExp
Operators not requiring licensing

```
void HOperatorSetX.TupleLog ([in] VARIANT T, [out] VARIANT Log)
```

Compute the natural logarithm of a tuple.

`TupleLog` computes the natural logarithm of the input tuple `T`. The natural logarithm is always returned as a floating point number. The natural logarithm of a string is not allowed.

- **`T` (input control)**: number(-array) \(\Rightarrow\) VARIANT (real, integer)
  - **Restriction**: \(T > 0\)
- **`Log` (output control)**: number(-array) \(\Rightarrow\) VARIANT (real)
  - Natural logarithm of the input tuple.

**Parallelization Information**

`TupleLog` is reentrant and processed without parallelization.

**See Also**

`TupleExp, TuplePow`

**Alternatives**

`TupleLog10`

```
void HOperatorSetX.TupleLog10 ([in] VARIANT T, [out] VARIANT Log)
```

Compute the base 10 logarithm of a tuple.

`TupleLog10` computes the base 10 logarithm of the input tuple `T`. The logarithm is always returned as a floating point number. The logarithm of a string is not allowed.

- **`T` (input control)**: number(-array) \(\Rightarrow\) VARIANT (real, integer)
  - **Restriction**: \(T > 0\)
- **`Log` (output control)**: number(-array) \(\Rightarrow\) VARIANT (real)
  - Base 10 logarithm of the input tuple.

**Parallelization Information**

`TupleLog10` is reentrant and processed without parallelization.

**See Also**

`TupleExp, TuplePow`

**Alternatives**

`TupleLog`

Operators not requiring licensing
Multiply two tuples.

TupleMult computes the product of the input tuples \( P_1 \) and \( P_2 \). If both tuples have the same length the corresponding elements of both tuples are subtracted. Otherwise, either \( P_1 \) or \( P_2 \) must have length 1. In this case, the multiplication is performed for each element of the longer tuple with the single element of the other tuple. If two integer numbers are multiplied, the result is again an integer number. If one of the operands is a floating point number, the result is a floating point number. The multiplication of strings is not allowed.

Parameter

- \( P_1 \) (input control) .............................. number(-array) \( \rightsquigarrow \) VARIANT (real, integer) Input tuple 1.
- \( P_2 \) (input control) .............................. number(-array) \( \rightsquigarrow \) VARIANT (real, integer) Input tuple 2.
- \( \text{Prod} \) (output control) ....................... number(-array) \( \rightsquigarrow \) VARIANT (real, integer) Product of the input tuples.

Parallelization Information

TupleMult is reentrant and processed without parallelization.

Alternatives

Module

Operators not requiring licensing

Negate a tuple.

TupleNeg computes the negation of the input tuple \( T \), i.e., \( \text{Neg} = -T \). The negation of an integer number is again an integer number. The negation of a floating point number is a floating point number. The negation of a string is not allowed. The negation of an empty input tuple results in an empty output tuple.

Parameter

- \( T \) (input control) .............................. number(-array) \( \rightsquigarrow \) VARIANT (real, integer) Input tuple.
- \( \text{Neg} \) (output control) ....................... number(-array) \( \rightsquigarrow \) VARIANT (real, integer) Negation of the input tuple.

Parallelization Information

TupleNeg is reentrant and processed without parallelization.

Module

Operators not requiring licensing

Calculate the power function two tuples.

TuplePow computes the power function of the input tuples \( T_1^{T_2} \). If both tuples have the same length the power function is applied to the corresponding elements of both tuples. Otherwise, either \( T_1 \) or \( T_2 \) must have length 1. In this case, the power function is performed for each element of the longer tuple with the single element of the other tuple. The result is always a floating point number. The power function of strings is not allowed.
13.1. ARITHMETIC

Parameter

▶ T1 (input control) .......................................................... number(-array) \(\leadsto\) VARIANT (real, integer)
Input tuple 1.
▶ T2 (input control) .......................................................... number(-array) \(\leadsto\) VARIANT (real, integer)
Input tuple 2.
▶ Pow (output control) ...................................................... number(-array) \(\leadsto\) VARIANT (real, integer)
Power function of the input tuples.

Parallelization Information

TuplePow is reentrant and processed without parallelization.

See Also

TupleLog, TupleLog10

Alternatives

TupleExp

Module

Operators not requiring licensing

HOperatorSetX.TupleRad ([in] VARIANT Deg )

Convert a tuple from degrees to radians.

TupleRad converts the input tuple Deg from degrees to radians. The result is always returned as a floating point number. The conversion of a string is not allowed.

Parameter

▶ Deg (input control) .......................................................... number(-array) \(\leadsto\) VARIANT (real, integer)
Input tuple.
▶ Rad (output control) ...................................................... number(-array) \(\leadsto\) VARIANT (real)
Input tuple in radians.

Parallelization Information

TupleRad is reentrant and processed without parallelization.

See Also

TupleDeg

Module

Operators not requiring licensing

HOperatorSetX.TupleSin ([in] VARIANT T )

Compute the sine of a tuple.

TupleSin computes the sine of the input tuple T. The sine is always returned as a floating point number. The sine of a string is not allowed.

Parameter

▶ T (input control) .......................................................... number(-array) \(\leadsto\) VARIANT (real, integer)
Input tuple.
▶ Sin (output control) ...................................................... number(-array) \(\leadsto\) VARIANT (real)
Sine of the input tuple.

Parallelization Information

TupleSin is reentrant and processed without parallelization.

See Also

TupleAsin
Compute the hyperbolic sine of a tuple.

TupleSinh computes the hyperbolic sine of the input tuple T. The hyperbolic sine is always returned as a floating point number. The hyperbolic sine of a string is not allowed.

**Parameter**

- **T** (input control) .......................... number(-array) ~> VARIANT ( real, integer )
  Input tuple.
- **Sinh** (output control) .......................... number(-array) ~> VARIANT ( real )
  Hyperbolic sine of the input tuple.

**Parallelization Information**

TupleSinh is reentrant and processed without parallelization.

**Alternatives**

TupleCosh, TupleTanh

**Module**

Operators not requiring licensing

---

Compute the square root of a tuple.

TupleSqrt computes the square root of the input tuple T. The square root is always returned as a floating point number. The square root of a string is not allowed.

**Parameter**

- **T** (input control) .......................... number(-array) ~> VARIANT ( real, integer )
  Input tuple.
- **Sqrt** (output control) .......................... number(-array) ~> VARIANT ( real )
  Square root of the input tuple.

**Parallelization Information**

TupleSqrt is reentrant and processed without parallelization.

**Module**

Operators not requiring licensing

---

Subtract two tuples.

TupleSub computes the difference of the input tuples D1 and D2. If both tuples have the same length the corresponding elements of both tuples are subtracted. Otherwise, either D1 or D2 must have length 1. In this case,
the subtraction is performed for each element of the longer tuple with the single element of the other tuple. If two
integer numbers are subtracted, the result is again an integer number. If one of the operands is a floating point
number, the result is a floating point number. The subtraction of strings is not allowed.

Parameter

- **D1** (input control) ........................................... number(-array) \( \rightarrow \) VARIANT (real, integer)
  Input tuple 1.
- **D2** (input control) ........................................... number(-array) \( \rightarrow \) VARIANT (real, integer)
  Input tuple 2.
- **Diff** (output control) ........................................ number(-array) \( \rightarrow \) VARIANT (real, integer)
  Difference of the input tuples.

Parallelization Information

TupleSub is reentrant and processed without parallelization.

Alternatives

TupleAdd

Operators not requiring licensing

---

```plaintext
[out] VARIANT Tan HTupleX.TupleTan ([in] VARIANT T )
void HOperatorSetX.TupleTan ([in] VARIANT T , [out] VARIANT Tan )
```

*Compute the tangent of a tuple.*

TupleTan computes the tangent of the input tuple T. The tangent is always returned as a floating point number.
The tangent of a string is not allowed.

Parameter

- **T** (input control) ........................................... number(-array) \( \rightarrow \) VARIANT (real, integer)
  Input tuple.
- **Tan** (output control) ........................................ number(-array) \( \rightarrow \) VARIANT (real)
  Tangent of the input tuple.

Parallelization Information

TupleTan is reentrant and processed without parallelization.

See Also

TupleAtan, TupleAtan2

Alternatives

TupleSin, TupleCos

Module

---

```plaintext
[out] VARIANT Tanh HTupleX.TupleTanh ([in] VARIANT T )
void HOperatorSetX.TupleTanh ([in] VARIANT T , [out] VARIANT Tanh )
```

*Compute the hyperbolic tangent of a tuple.*

TupleTanh computes the hyperbolic tangent of the input tuple T. The hyperbolic tangent is always returned as a
floating point number. The hyperbolic tangent of a string is not allowed.

Parameter

- **T** (input control) ........................................... number(-array) \( \rightarrow \) VARIANT (real, integer)
  Input tuple.
- **Tanh** (output control) ...................................... number(-array) \( \rightarrow \) VARIANT (real)
  Hyperbolic tangent of the input tuple.
Parallelization Information

TupleTanh is reentrant, local, and processed without parallelization.

Alternatives

TupleSinh, TupleCosh

Operators not requiring licensing

13.2 Bit-Operations

```
[不出] VARIANT BAnd HTupleX.TupleBand ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleBand ([in] VARIANT T1, [in] VARIANT T2,
[out] VARIANT BAnd )
```

Compute the bitwise and of two tuples.

TupleBand computes the bitwise and of the input tuples T1 and T2. If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either T1 or T2 must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

Parameter

- T1 (input control) ........................................ integer(-array) ~ VARIANT( integer )
  Input tuple 1.
- T2 (input control) ........................................ integer(-array) ~ VARIANT( integer )
  Input tuple 2.
- BAnd (output control) ................................. integer(-array) ~ VARIANT( integer )
  Binary and of the input tuples.

Parallelization Information

TupleBand is reentrant and processed without parallelization.

See Also

TupleAnd, TupleOr, TupleXor, TupleNot

```
[不出] VARIANT BNot HTupleX.TupleBnot ([in] VARIANT T )
void HOperatorSetX.TupleBnot ([in] VARIANT T, [out] VARIANT BNot )
```

Compute the bitwise not of two tuples.

TupleBnot computes the bitwise not of the input tuple T. The input tuple must contain only integer numbers.

Parameter

- T (input control) ........................................ integer(-array) ~ VARIANT( integer )
  Input tuple.
- BNot (output control) ................................. integer(-array) ~ VARIANT( integer )
  Binary not of the input tuple.

Parallelization Information

TupleBnot is reentrant and processed without parallelization.

See Also

TupleAnd, TupleOr, TupleXor, TupleNot

13.2. BIT-OPERATIONS

Alternatives
TupleBand, TupleBor, TupleBxor

Operators not requiring licensing

[out] VARIANT Bor HTupleX.TupleBor ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleBor ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Bor )

Compute the bitwise or of two tuples.

TupleBor computes the bitwise or of the input tuples T1 and T2. If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either T1 or T2 must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

Parameter

▷ T1 (input control) ................................................. integer(-array) ⇆ VARIANT (integer)  
   Input tuple 1.
▷ T2 (input control) ................................................. integer(-array) ⇆ VARIANT (integer)  
   Input tuple 2.
▷ Bor (output control) .............................................. integer(-array) ⇆ VARIANT (integer)  
   Binary or of the input tuples.

Parallelization Information
TupleBor is reentrant and processed without parallelization.

See Also
TupleAnd, TupleOr, TupleXor, TupleNot

Alternatives
TupleBand, TupleBxor, TupleBnot

Operators not requiring licensing

[out] VARIANT Bxor HTupleX.TupleBxor ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleBxor ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Bxor )

Compute the bitwise exclusive or of two tuples.

TupleBxor computes the bitwise exclusive or of the input tuples T1 and T2. If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either T1 or T2 must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

Parameter

▷ T1 (input control) ................................................. integer(-array) ⇆ VARIANT (integer)  
   Input tuple 1.
▷ T2 (input control) ................................................. integer(-array) ⇆ VARIANT (integer)  
   Input tuple 2.
▷ Bxor (output control) ............................................ integer(-array) ⇆ VARIANT (integer)  
   Binary exclusive or of the input tuples.

Parallelization Information
TupleBxor is reentrant and processed without parallelization.

See Also
TupleAnd, TupleOr, TupleXor, TupleNot

HALCON 6.0
Alternatives

TupleBand, TupleBor, TupleBnot

Module

Operators not requiring licensing

```c
```

Shift a tuple bitwise to the left.

TupleLsh shifts the tuple `T` bitwise to the left by `Shift` places. If no overflow occurs, this operation is equivalent to a multiplication by $2^{\text{Shift}}$. If `T` is negative, the result depends on the hardware. If `Shift` is negative or larger than 32, the result is undefined. If both tuples have the same length the corresponding elements of both tuples are shifted. Otherwise, either `T` or `Shift` must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

**Parameter**

- **T** (input control) ........................................integer(-array) → VARIANT( integer )
  Input tuple.
- **Shift** (input control) ............................integer(-array) → VARIANT( integer )
  Number of places to shift the input tuple.
- **Lsh** (output control) ............................integer(-array) → VARIANT( integer )
  Shifted input tuple.

Parallelization Information

TupleLsh is reentrant and processed without parallelization.

See Also

TupleRsh

Alternatives

TupleMult

Module

Operators not requiring licensing

```c
```

Shift a tuple bitwise to the right.

TupleRsh shifts the tuple `T` bitwise to the right by `Shift` places. This operation is equivalent to a division by $2^{\text{Shift}}$. If `T` is negative, the result depends on the hardware. If `Shift` is negative or larger than 32, the result is undefined. If both tuples have the same length the corresponding elements of both tuples are shifted. Otherwise, either `T` or `Shift` must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

**Parameter**

- **T** (input control) ........................................integer(-array) → VARIANT( integer )
  Input tuple.
- **Shift** (input control) ............................integer(-array) → VARIANT( integer )
  Number of places to shift the input tuple.
- **Rsh** (output control) ............................integer(-array) → VARIANT( integer )
  Shifted input tuple.
13.3 Comparison

```plaintext
[out] long Equal HTupleX.TupleEqual ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleEqual ([in] VARIANT T1, [in] VARIANT T2,
[out] long Equal )
```

Test, whether two tuples are equal.

TupleEqual tests whether the two input tuples T1 and T2 are equal by comparing the tuples elementwise. Two tuples are equal, if they have got the same number of elements and if their elements are equal. Two tuple elements are equal, if they are both (integer or floating point) numbers or both are strings and contain the same value.

Parameter

- T1 (input control) ...............................................................integer(-array) ~ VARIANT (integer, real, string )
  Input tuple 1.
- T2 (input control) ...............................................................integer(-array) ~ VARIANT (integer, real, string )
  Input tuple 2.
- Equal (output control) ............................................................integer ~ long
  Result of the comparison of the input tuples.

Parallelization Information

TupleEqual is reentrant and processed without parallelization.

See Also

TupleLsh

Alternatives

TupleDiv

Operators not requiring licensing

```plaintext
[out] long Greater HTupleX.TupleGreater ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleGreater ([in] VARIANT T1, [in] VARIANT T2,
[out] long Greater )
```

Test, whether a tuple is greater than another tuple.

TupleGreater tests whether the input tuple T1 is greater than T2. A tuple T1 is said to be greater than a tuple T2, if T1 has been found to be greater when comparing it elementwise to T2 or (for the case that the elementwise comparison did not show that T1 is greater than T2) if T1 has got more elements than T2. With the elementwise comparison the single elements of T1 and T2 are compared with each other one after another (i.e., the first element of T1 is compared to the first element of T2 and the second element of T1 is compared to the second element of T2 etc.). If an element of T1 is greater than its counterpart of T2, T1 is said to be greater than T2. As a precondition for comparing the tuples elementwise two corresponding elements must either both be (integer or floating point) numbers or both be strings. Otherwise TupleGreater returns an error.
Parameter

▷ T1 (input control) ................................. number(-array) \sim\text{VARIANT (integer, real, string)}
Input tuple 1.
▷ T2 (input control) ................................. number(-array) \sim\text{VARIANT (integer, real, string)}
Input tuple 2.
▷ Greater (output control) ........................ integer \sim\text{long}
Result of the comparison of the input tuples.

Parallelization Information

TupleGreater is reentrant and processed without parallelization.

Alternatives

TupleGreaterEqual, TupleLess, TupleLessEqual, TupleEqual, TupleNotEqual

Module

Operators not requiring licensing

[out] long Greatereq HTupleX.TupleGreaterEqual ([in] VARIANT T1, [in] VARIANT T2 )

void HOperatorSetX.TupleGreaterEqual ([in] VARIANT T1, [in] VARIANT T2, [out] long Greatereq )

Test, whether a tuple is greater or equal to another tuple.

TupleGreaterEqual tests whether the input tuple T1 is greater or equal to T2. A tuple T1 is said to be greater or equal to a tuple T2, if T1 is not less than T2. A tuple T1 is said to be less than a tuple T2, if T1 has been found to be less when comparing it elementwise to T2 or (for the case that the elementwise comparison did not show that T1 is less than T2) if T1 has got less elements than T2. With the elementwise comparison the single elements of T1 and T2 are compared with each other one after another (i.e., the first element of T1 is compared to the first element of T2 and the second element of T1 is compared to the second element of T2 etc.). If an element of T1 is less than its counterpart of T2, T1 is said to be less than T2. As a precondition for comparing the tuples elementwise two corresponding elements must either both be (integer or floating point) numbers or both be strings. Otherwise TupleGreaterEqual returns an error.

Parameter

▷ T1 (input control) ................................. number(-array) \sim\text{VARIANT (integer, real, string)}
Input tuple 1.
▷ T2 (input control) ................................. number(-array) \sim\text{VARIANT (integer, real, string)}
Input tuple 2.
▷ Greatereq (output control) ........................ integer \sim\text{long}
Result of the comparison of the input tuples.

Parallelization Information

TupleGreaterEqual is reentrant and processed without parallelization.

Alternatives

TupleGreater, TupleLess, TupleLessEqual, TupleEqual, TupleNotEqual

Module

Operators not requiring licensing

[out] long Less HTupleX.TupleLess ([in] VARIANT T1, [in] VARIANT T2 )

void HOperatorSetX.TupleLess ([in] VARIANT T1, [in] VARIANT T2, [out] long Less )

Test, whether a tuple is less than another tuple.

TupleLess tests whether the input tuple T1 is less than T2. A tuple T1 is said to be less than a tuple T2, if T1 has been found to be less when comparing it elementwise to T2 or (for the case that the elementwise comparison
did not show that $T1$ is less than $T2$ if $T1$ has got less elements than $T2$. With the elementwise comparison the single elements of $T1$ and $T2$ are compared with each other one after another (i.e., the first element of $T1$ is compared to the first element of $T2$ and the second element of $T1$ is compared to the second element of $T2$ etc.). If an element of $T1$ is less than its counterpart of $T2$, $T1$ is said to be less than $T2$. As a precondition for comparing the tuples elementwise two corresponding elements must either both be (integer or floating point) numbers or both be strings. Otherwise TupleLess returns an error.

Parameter

- $T1$ (input control) .............................................. number(-array) $\sim$ VARIANT ( integer, real, string )
  Input tuple 1.
- $T2$ (input control) .............................................. number(-array) $\sim$ VARIANT ( integer, real, string )
  Input tuple 2.
- $\text{Less}$ (output control) ...................................... integer $\sim$ long
  Result of the comparison of the input tuples.

Parallelization Information

TupleLess is reentrant and processed without parallelization.

Alternatives

TupleLessEqual, TupleGreater, TupleGreaterEqual, TupleEqual, TupleNotEqual

Module

Operators not requiring licensing

```{halcon}
[out] long Lesseq HTupleX.TupleLessEqual ([in] VARIANT T1, [in] VARIANT T2 )

void HOperatorSetX.TupleLessEqual ([in] VARIANT T1, [in] VARIANT T2, [out] long Lesseq )
```

Test, whether a tuple is less or equal to another tuple.

TupleLessEqual tests whether the input tuple $T1$ is less or equal to $T2$. A tuple $T1$ is said to be less or equal to a tuple $T2$, if $T1$ is not greater than $T2$. A tuple $T1$ is said to be greater than a tuple $T2$, if $T1$ has been found to be greater when comparing it elementwise to $T2$ or (for the case that the elementwise comparison did not show that $T1$ is greater than $T2$) if $T1$ has got more elements than $T2$. With the elementwise comparison the single elements of $T1$ and $T2$ are compared with each other one after another (i.e., the first element of $T1$ is compared to the first element of $T2$ and the second element of $T1$ is compared to the second element of $T2$ etc.). If an element of $T1$ is greater than its counterpart of $T2$, $T1$ is said to be greater than $T2$. As a precondition for comparing the tuples elementwise two corresponding elements must either both be (integer or floating point) numbers or both be strings. Otherwise TupleLessEqual returns an error.

Parameter

- $T1$ (input control) .............................................. number(-array) $\sim$ VARIANT ( integer, real, string )
  Input tuple 1.
- $T2$ (input control) .............................................. number(-array) $\sim$ VARIANT ( integer, real, string )
  Input tuple 2.
- $\text{Lesseq}$ (output control) ...................................... integer $\sim$ long
  Result of the comparison of the input tuples.

Parallelization Information

TupleLessEqual is reentrant and processed without parallelization.

Alternatives

TupleLess, TupleGreater, TupleGreaterEqual, TupleEqual, TupleNotEqual

Module

Operators not requiring licensing
**CHAPTER 13. TUPLE**

```plaintext
[out] long Nequal HTupleX.TupleNotEqual ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleNotEqual ([in] VARIANT T1, [in] VARIANT T2, [out] long Nequal )
```

Test whether two tuples are not equal.

TupleNotEqual tests whether the two input tuples T1 and T2 are not equal. Two tuples are not equal, if they consist of a different number of elements or if they differ when compared elementwise. Two tuple elements differ, if they are of types that may not be compared (e.g. one string and one integer) or if they differ in their values.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1 (input control) number(-array) ~ VARIANT (integer, real, string) Input tuple 1.</td>
</tr>
<tr>
<td>T2 (input control) number(-array) ~ VARIANT (integer, real, string) Input tuple 2.</td>
</tr>
<tr>
<td>Nequal (output control) integer ~ long Result of the comparison of the input tuples.</td>
</tr>
</tbody>
</table>

TupleNotEqual is reentrant and processed without parallelization.

Alternatives

TupleEqual, TupleLess, TupleGreater, TupleLessEqual, TupleGreaterEqual

Module Operators not requiring licensing

---

### 13.4 Conversion

Convert a tuple of integers into strings with the corresponding ASCII codes.

TupleChr converts the input tuple T, consisting of integer numbers, into a tuple of strings of length 1, the characters of which have the ASCII code of the corresponding input number.

```plaintext
[out] VARIANT Chr HTupleX.TupleChr ([in] VARIANT T )
void HOperatorSetX.TupleChr ([in] VARIANT T, [out] VARIANT Chr )
```

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (input control) integer(-array) ~ VARIANT (integer) Input tuple.</td>
</tr>
<tr>
<td>Chr (output control) string(-array) ~ VARIANT (string) Strings corresponding to the ASCII code of the input tuple.</td>
</tr>
</tbody>
</table>

TupleChr is reentrant and processed without parallelization.

See Also

TupleOrd, TupleOrds

Alternatives

TupleChrt

Module Operators not requiring licensing

---

Convert a tuple of integers into strings with the corresponding ASCII codes.
TupleChrt converts the input tuple \( T \), consisting of integer numbers, into a tuple of strings and integer numbers (where only the number 0 can occur in the output), the characters of which have the ASCII code of the corresponding input number. The operator tries to pack as many of the input numbers into one string as possible. Only if the value 0 occurs in \( T \) the current string is terminated at this point, the integer number 0 is inserted into the output, and a new string with the remaining input values is started. This operator can be used to convert data read with ReadSerial into strings. With this mechanism it is possible to read bytes with the value 0.

**Parameter**

- \( T \) (input control) ........................................ integer(-array) \( \sim \) VARIANT( integer )
  Input tuple.
- \( \text{Chrt} \) (output control) ................................. string(-array) \( \sim \) VARIANT( string, integer )
  Strings corresponding to the ASCII code of the input tuple.

**Example**

```plaintext```
read_serial (SerialHandle, 100, Data)  
tuple_chrt (Data, Strings)
```

**Parallelization Information**

TupleChrt is reentrant and processed without parallelization.

**See Also**

TupleOrd, TupleOrds, ReadSerial

**Alternatives**

TupleChr

**Module**

Operators not requiring licensing

```plaintext```
[out] VARIANT Deg HTupleX.TupleDeg ( [in] VARIANT Rad )  
void HOperatorSetX.TupleDeg ( [in] VARIANT Rad, [out] VARIANT Deg )
```

Convert a tuple from radians to degrees.

TupleDeg converts the input tuple \( \text{Rad} \) from radians to degrees. The result is always returned as a floating point number. The conversion of a string is not allowed.

**Parameter**

- \( \text{Rad} \) (input control) ........................................ number(-array) \( \sim \) VARIANT( real, integer )
  Input tuple.
- \( \text{Deg} \) (output control) ........................................ number(-array) \( \sim \) VARIANT( real )
  Input tuple in degrees.

**Parallelization Information**

TupleDeg is reentrant and processed without parallelization.

**See Also**

TupleRad

**Module**

Operators not requiring licensing

```plaintext```
[out] VARIANT Number HTupleX.TupleNumber ( [in] VARIANT T )  
void HOperatorSetX.TupleNumber ( [in] VARIANT T, [out] VARIANT Number )
```

Convert a tuple (of strings) into a tuple of numbers.
**TupleNumber** converts the input tuple T into a tuple of numbers. If the input tuple contains numbers, they are simply copied into the output tuple. Strings are converted into the appropriate type of number (integer or floating point numbers) or are copied as strings if they do not represent a number.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (input control) number(-array) VARIANT (real, integer, string) Input tuple.</td>
</tr>
<tr>
<td>Number (output control) number(-array) VARIANT (real, integer, string) Input tuple as numbers.</td>
</tr>
</tbody>
</table>

**Parallelization Information**

TupleNumber is reentrant and processed without parallelization.

**See Also**

TupleIsNumber, TupleString

Operators not requiring licensing

```plaintext
[out] VARIANT Ord HTupleX.TupleOrd ([in] VARIANT T )
void HOperatorSetX.TupleOrd ([in] VARIANT T, [out] VARIANT Ord )
```

Convert a tuple of strings of length 1 into a tuple of their ASCII codes.

**TupleOrd** converts the input tuple T, which may only contain strings of length 1, into a tuple of integer numbers that represent the ASCII code of the characters of the strings.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (input control) string(-array) VARIANT (string) Input tuple.</td>
</tr>
<tr>
<td>Ord (output control) integer(-array) VARIANT (integer) ASCII code of the input tuple.</td>
</tr>
</tbody>
</table>

**Parallelization Information**

TupleOrd is reentrant and processed without parallelization.

**See Also**

TupleChr, TupleChrt

Operators not requiring licensing

```plaintext
[out] VARIANT Ords HTupleX.TupleOrds ([in] VARIANT T )
void HOperatorSetX.TupleOrds ([in] VARIANT T, [out] VARIANT Ords )
```

Convert a tuple of strings into a tuple of their ASCII codes.

**TupleOrds** converts the input tuple T, which may only contain strings and integer numbers, into a tuple of integer numbers that represent the ASCII code of the characters of the strings. The characters of the individual strings are output according to their order within the string and within the tuple. Integer numbers are simply copied to an appropriate position in the output string. This operator can be used to prepare outputs with WriteSerial. In particular, a byte with value 0 can be written by inserting the integer number 0 into T.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (input control) string(-array) VARIANT (string, integer) Input tuple.</td>
</tr>
<tr>
<td>Ords (output control) integer(-array) VARIANT (integer) ASCII code of the input tuple.</td>
</tr>
</tbody>
</table>
Example

tuple_ords (["String 1", 0, "String 2", 0], Data)
write_serial (SerialHandle, Data)

Parallelization Information

TupleOrds is reentrant and processed without parallelization.

See Also

TupleChr, TupleChrt, WriteSerial

Alternatives

TupleOrd

Module

Operators not requiring licensing

Convert a tuple into a tuple of floating point numbers.

TupleReal converts the input tuple T into a tuple of floating point numbers. The conversion of a string is not allowed.

Parameter

▷ T (input control) ........................................ number(-array) ~> VARIANT ( real, integer )
  Input tuple.
▷ Real (output control) ................................... number(-array) ~> VARIANT ( real )
  Input tuple as floating point numbers.

Parallelization Information

TupleReal is reentrant and processed without parallelization.

Module

Operators not requiring licensing

Convert a tuple into a tuple of integer numbers.

TupleRound converts the input tuple T into a tuple of integer numbers by rounding T to the nearest integer. The conversion of a string is not allowed.

Parameter

▷ T (input control) ........................................ number(-array) ~> VARIANT ( real, integer )
  Input tuple.
▷ Round (output control) ................................ number(-array) ~> VARIANT ( integer )
  Input tuple as integer numbers.

Parallelization Information

TupleRound is reentrant and processed without parallelization.

Module

Operators not requiring licensing
TupleString converts numbers to strings or modifies strings. The operator has two parameters: \( T \) is the number or string that has to be converted. Format specifies the conversion. This format string consists of the following four parts:

\[
\text{flags} \text{field width} \text{precision} \text{conversion characters}
\]

So a conversion might look like

\[
tuple\_string(1332.4554, \'.6e', \text{String})
\]

**flags** Zero or more flags, in any order, which modify the meaning of the conversion specification. Flags may consist of the following characters:

- The result of the conversion is left justified within the field.
- The result of a signed conversion always begins with a sign, + or -.

**<space>** If the first character of a signed conversion is not a sign, a space character is prefixed to the result. This means that if the space flag and + flag both appear, the space flag is ignored.

**#** The value is to be converted to an "alternate form". For d and s conversions, this flag has no effect. For o conversion (see below), it increases the precision to force the first digit of the result to be a zero. For x or X conversion (see below), a non-zero result has 0x or 0X prefixed to it. For e, E, f, g, and G conversions, the result always contains a radix character, even if no digits follow the radix character. For g and G conversions, trailing zeros are not removed from the result, contrary to usual behavior.

**field width** An optional string of decimal digits to specify a minimum field width. For an output field, if the converted value has fewer characters than the field width, it is padded on the left (or right, if the left-adjustment flag, - has been given) to the field width.

**precision** The precision specifies the minimum number of digits to appear for the d, o, x, or X conversions (the field is padded with leading zeros), the number of digits to appear after the radix character for the e and f conversions, the maximum number of significant digits for the g conversion, or the maximum number of characters to be printed from a string in s conversion. The precision takes the form of a period . followed by a decimal digit string. A null digit string is treated as a zero.

**conversion characters** A conversion character indicates the type of conversion to be applied:

- \( d, o, x, X \) The integer argument is printed in signed decimal (d), unsigned octal (o), or unsigned hexadecimal notation (x and X). The x conversion uses the numbers and letters 0123456789abcdef, and the X conversion uses the numbers and letters 0123456789ABCDEF. The precision component of the argument specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it is expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of 0 is no characters.
- \( f \) The floating-point number argument is printed in decimal notation in the style \[-\]d.ddddd, where the number of digits after the radix character, \( x \), is equal to the precision specification. If the precision is omitted from the argument, six digits are output; if the precision is explicitly 0, no radix appears.
- \( e, E \) The floating-point-number argument is printed in the style \[-\]d.ddde\( \pm \)ddd, where there is one digit before the radix character, and the number of digits after it is equal to the precision. When the precision is missing, six digits are produced; if the precision is 0, no radix character appears. The E conversion character produces a number with E introducing the exponent instead of e. The exponent always contains at least two digits. However, if the value to be printed requires an exponent greater than two digits, additional exponent digits are printed as necessary.
- \( g, G \) The floating-point-number argument is printed in style \( f \) or \( e \) (or in style \( E \) in the case of a G conversion character), with the precision specifying the number of significant digits. The style used depends on the value converted; style \( e \) is used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character appears only if it is followed by a digit.
The argument is taken to be a string, and characters from the string are printed until the end of the string or the number of characters indicated by the precision specification of the argument is reached. If the precision is omitted from the argument, it is interpreted as infinite and all characters up to the end of the string are printed.

Similar to the \s conversion specifier, except that the string can contain backslash-escape sequences which are then converted to the characters they represent.

In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

\begin{verbatim}
\textbf{Parameter}

\begin{itemize}
  \item[T] (input control) \hspace{1cm} number(-array) \rightarrow VARIANT ( real, integer, string )
  Input tuple.
  \item[Format] (input control) \hspace{1cm} string \rightarrow String
  Format string.
  \item[String] (output control) \hspace{1cm} string(-array) \rightarrow VARIANT ( string )
  Input tuple converted to strings.
\end{itemize}
\end{verbatim}

\begin{verbatim}
\textbf{Parallelization Information}
TupleString is \textit{reentrant} and processed \textit{without} parallelization.
\end{verbatim}

\section*{13.5 Creation}

\begin{verbatim}
\begin{verbatim}
\textbf{TupleConcat}
concatenates the input tuples \texttt{T1} and \texttt{T2} to a new tuple \texttt{Concat}. The first elements of \texttt{Concat} conform to the elements of \texttt{T1} and the remaining elements of \texttt{Concat} conform to those of \texttt{T2}

\begin{verbatim}
\textbf{Parameter}

\begin{itemize}
  \item[T1] (input control) \hspace{1cm} number(-array) \rightarrow VARIANT ( real, integer, string )
  Input tuple 1.
  \item[T2] (input control) \hspace{1cm} number(-array) \rightarrow VARIANT ( real, integer, string )
  Input tuple 2.
  \item[Concat] (output control) \hspace{1cm} number(-array) \rightarrow VARIANT ( real, integer, string )
  Concatentaion of input tuples.
\end{itemize}
\end{verbatim}

\end{verbatim}

\textbf{Parallelization Information}
TupleConcat is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{Alternatives}
TupleGenConst, TupleStrBitSelect, TupleSelect, TupleStrFirstN, TupleStrLastN

\textbf{Module}

Operators not requiring licensing
Generate a tuple of a specific length and initialize its elements.

TupleGenConst generates a new tuple in Newtuple. The input parameter Length determines the number of elements for the new tuple. Thus Length may only consist of a single number. If Length contains a floating point number, this may only represent an integer value (without fraction). The data type and value of each element of the new generated tuple is determined by the input parameter Const that may only consist of a single element. All elements in Newtuple have got the same data type and value as the single element in Const.

Parameter

- **Length** (input control) ............... number ~ VARIANT ( real, integer )
  Length of tuple to generate.
- **Const** (input control) ............... number ~ VARIANT ( real, integer, string )
  < no description available>
- **Newtuple** (output control) ........... number(-array) ~ VARIANT ( real, integer, string )
  New Tuple.

Parallelization Information

TupleGenConst is reentrant and processed without parallelization.

Alternatives

TupleStrBitSelect, TupleSelect, TupleStrFirstN, TupleStrLastN, TupleConcat

Module

Operators not requiring licensing

### 13.6 Element-Order

Invert a tuple.

TupleInverse inverts the input tuple Tuple. Thus Inverted contains the same elements as Tuple but with the reverse order.

Parameter

- **Tuple** (input control) ............... number(-array) ~ VARIANT ( real, integer, string )
  Input tuple.
- **Inverted** (output control) ............ number(-array) ~ VARIANT ( real, integer, string )
  Inverted input tuple.

Parallelization Information

TupleInverse is reentrant and processed without parallelization.

Alternatives

TupleSort, TupleSortIndex

Module

Operators not requiring licensing

Sorts the elements of a tuple in ascending order.
TupleSort sorts all elements of Tuple in ascending order and returns the result with Sorted. As a precondition the single elements of Tuple must be comparable. Thus Tuple must either exclusively consist of strings or it must only contain (integer or floating point) numbers. In the latter case integers and floating point numbers may be mixed.

```
Parameter

Tuple (input control) number(-array) ~ VARIANT ( real, integer, string )
Input tuple.

Sorted (output control) number(-array) ~ VARIANT ( real, integer, string )
Sorted tuple.
```

Parallelization Information

TupleSort is reentrant and processed without parallelization.

Alternatives

TupleSortIndex, TupleInverse

Module

Operators not requiring licensing

```
void HOperatorSetX.TupleSortIndex ([in] VARIANT Tuple, [out] VARIANT Indices )
```

Sorts the elements of a tuple and returns the indices of the sorted tuple.

TupleSortIndex sorts all elements of Tuple in ascending order and returns the indices of the elements of the sorted tuple (in relation to the input tuple Tuple) with Indices. As a precondition the single elements of Tuple must be comparable. Thus Tuple must either exclusively consist of strings or it must only contain (integer or floating point) numbers. In the latter case integers and floating point numbers may be mixed.

```
Parameter

Tuple (input control) number(-array) ~ VARIANT ( real, integer, string )
Input tuple.

Indices (output control) number(-array) ~ VARIANT ( integer )
Sorted tuple.
```

Parallelization Information

TupleSortIndex is reentrant and processed without parallelization.

Alternatives

TupleSort, TupleInverse

Module

Operators not requiring licensing

13.7 Features

```
void HOperatorSetX.TupleCeil ([in] VARIANT T, [out] VARIANT Ceil )
```

Compute the ceiling function of a tuple.

TupleCeil computes the ceiling function of the input tuple T, i.e., the smallest integer greater than or equal to T. The ceiling function is always returned as a floating point number. The ceiling function of a string is not allowed.
Parameter

- **T** (input control) ... number(-array) ~ VARIANT (real, integer)
  Input tuple.

- **Ceil** (output control) ... number(-array) ~ VARIANT (real)
  Ceiling function of the input tuple.

Parallelization Information

TupleCeil is reentrant and processed without parallelization.

Alternatives

Operators not requiring licensing

---

**Return the standard deviation of the elements of a tuple.**

 TupleDeviation calculates the standard derivation of all elements of the input tuple `Tuple`. It returns the derivation as a floating point number in the output parameter `Deviation`. The input tuple may only consist of numbers (integer or floating point numbers).

Parameter

- **Tuple** (input control) ... number(-array) ~ VARIANT (real, integer, string)
  Input tuple.

- **Deviation** (output control) ... number(-array) ~ VARIANT (real)
  Standard deviation of tuple elements.

Parallelization Information

TupleDeviation is reentrant and processed without parallelization.

Alternatives

TupleMean, TupleSum, TupleMin, TupleMax, TupleLength

Module

Operators not requiring licensing

---

**Compute the floor function of a tuple.**

TupleFloor computes the floor function of the input tuple `T`, i.e., the largest integer less than or equal to `T`. The floor function is always returned as a floating point number. The floor function of a string is not allowed.

Parameter

- **T** (input control) ... number(-array) ~ VARIANT (real, integer)
  Input tuple.

- **Floor** (output control) ... number(-array) ~ VARIANT (real)
  Floor function of the input tuple.

Parallelization Information

TupleFloor is reentrant and processed without parallelization.

Alternatives

TupleCeil

Module

Operators not requiring licensing
Check a tuple (of strings) whether it represents numbers.

TupleIsNumber checks each element of the input tuple $T$ whether it represents a number. If the element is a number (real or integer), 1 is returned for that element. If the element is a string, it is checked whether the string represents a number. If so, 1 is returned, otherwise 0.

**Parallelization Information**

TupleIsNumber is reentrant and processed without parallelization.

**See Also**

TupleNumber

**Module**

Operators not requiring licensing

---

Returns the maximal element of a tuple.

TupleMax returns the maximal element of all elements of the input tuple $T$. All elements of $T$ either have to be strings or numbers (integer or floating point numbers). It is not allowed to mix strings with numerical values. The result parameter $Max$ will contain a floating point number, if at least one element of $T$ is a floating point number. If all elements of $T$ are integer numbers the resulting sum will also be an integer number.

**Parallelization Information**

TupleMax is reentrant and processed without parallelization.
Parameter

Tuple (input control) .................................................. number(-array) ~ VARIANT ( real, integer, string )
Input tuple.

Max (output control) .................................................. number(-array) ~ VARIANT ( real )
Maximal element of the input tuple elements.

Parallelization Information

TupleMax is reentrant and processed without parallelization.

Alternatives

TupleMin, TupleMean, TupleDeviation, TupleSum, TupleLength

Module

Alternatives

TupleMin, TupleMean, TupleDeviation, TupleSum, TupleLength

Module

Operators not requiring licensing

[out] VARIANT Mean HTupleX.TupleMean ([in] VARIANT Tuple )

void HOperatorSetX.TupleMean ([in] VARIANT Tuple, [out] VARIANT Mean )

Return the mean value of a tuple of numbers.

TupleMean returns the mean value of all elements of the input tuple Tuple as a floating point number in the
output parameter Mean. The input tuple may only consist of numbers (integer or floating point numbers).

Parameter

Tuple (input control) .................................................. number(-array) ~ VARIANT ( real, integer, string )
Input tuple.

Mean (output control) .................................................. number(-array) ~ VARIANT ( real )
Mean value of tuple elements.

Parallelization Information

TupleMean is reentrant and processed without parallelization.

Alternatives

TupleMin, TupleMean, TupleDeviation, TupleSum, TupleLength

Module

Operators not requiring licensing

[out] VARIANT Min HTupleX.TupleMin ([in] VARIANT Tuple )

void HOperatorSetX.TupleMin ([in] VARIANT Tuple, [out] VARIANT Min )

Returns the minimal element of a tuple.

TupleMin returns the minimal element of all elements of the input tuple Tuple. All elements of Tuple either
have to be strings or numbers (integer or floating point numbers). It is not allowed to mix strings with numerical
values. The result parameter Min will contain a floating point number, if at least one element of Tuple is a
floating point number. If all elements of Tuple are integer numbers the resulting sum will also be an integer
number.

Parameter

Tuple (input control) .................................................. number(-array) ~ VARIANT ( real, integer, string )
Input tuple.

Min (output control) .................................................. number(-array) ~ VARIANT ( real )
Minimal element of the input tuple elements.

Parallelization Information

TupleMin is reentrant and processed without parallelization.

Alternatives

TupleMax, TupleMean, TupleDeviation, TupleSum, TupleLength

Module

Operators not requiring licensing
13.8 Logical-Operations

 TupleSum returns the sum of all elements of the input tuple Tuple. All elements of Tuple either have to be strings or numbers (integer or floating point numbers). It is not allowed to mix strings with numerical values. The result parameter Sum will contain a floating point number, if at least one element of Tuple is a floating point number. If all elements of Tuple are integer numbers the resulting sum will also be an integer number. If Tuple contains strings, the concatenation will be used for building the sum.

Parameter

Tuple (input control) number(-array) VARIANT (real, integer, string)
Input tuple.

Sum (output control) number(-array) VARIANT (real)
Sum of tuple elements.

Parallelization Information

TupleSum is reentrant and processed without parallelization.

Alternatives

TupleMean, TupleDeviation, TupleMin, TupleMax, TupleLength

Module

Operators not requiring licensing

13.8 Logical-Operations

TupleAnd computes the logical and of the input tuples T1 and T2. If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either T1 or T2 must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

Parameter

T1 (input control) integer(-array) VARIANT (integer)
Input tuple 1.

T2 (input control) integer(-array) VARIANT (integer)
Input tuple 2.

And (output control) integer(-array) VARIANT (integer)
Logical and of the input tuples.

Parallelization Information

TupleAnd is reentrant and processed without parallelization.

See Also

TupleBand, TupleBor, TupleBxor, TupleBnot

Alternatives

TupleOr, TupleXor, TupleNot

Module

Operators not requiring licensing
Compute the logical not of two tuples.

TupleNot computes the logical not of the input tuple \( T \). The input tuple must contain only integer numbers.

\[
\begin{align*}
&\text{Parameter} \\
&\triangleright T \text{ (input control)} \quad \text{integer(-array)} \sim \text{VARIANT (integer)} \\
&\quad \text{Input tuple.} \\
&\triangleright \text{Not (output control)} \quad \text{integer(-array)} \sim \text{VARIANT (integer)} \\
&\quad \text{Binary not of the input tuple.}
\end{align*}
\]

TupleNot is reentrant, local, and processed without parallelization.

See Also

TupleBand, TupleBor, TupleBxor, TupleBnot

Alternatives

TupleAnd, TupleOr, TupleXor

Module

Operators not requiring licensing

Compute the logical or of two tuples.

TupleOr computes the logical or of the input tuples \( T_1 \) and \( T_2 \). If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either \( T_1 \) or \( T_2 \) must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

\[
\begin{align*}
&\text{Parameter} \\
&\triangleright T_1 \text{ (input control)} \quad \text{integer(-array)} \sim \text{VARIANT (integer)} \\
&\quad \text{Input tuple 1.} \\
&\triangleright T_2 \text{ (input control)} \quad \text{integer(-array)} \sim \text{VARIANT (integer)} \\
&\quad \text{Input tuple 2.} \\
&\triangleright \text{Or (output control)} \quad \text{integer(-array)} \sim \text{VARIANT (integer)} \\
&\quad \text{Logical or of the input tuples.}
\end{align*}
\]

TupleOr is reentrant and processed without parallelization.

See Also

TupleBand, TupleBor, TupleBxor, TupleBnot

Alternatives

TupleAnd, TupleXor, TupleNot

Module

Operators not requiring licensing

Compute the logical exclusive or of two tuples.

\[
\begin{align*}
&\text{Parameter} \\
&\triangleright T_1 \text{ (input control)} \quad \text{integer(-array)} \sim \text{VARIANT (integer)} \\
&\quad \text{Input tuple 1.} \\
&\triangleright T_2 \text{ (input control)} \quad \text{integer(-array)} \sim \text{VARIANT (integer)} \\
&\quad \text{Input tuple 2.} \\
&\triangleright \text{Xor (output control)} \quad \text{integer(-array)} \sim \text{VARIANT (integer)} \\
&\quad \text{Logical or of the input tuples.}
\end{align*}
\]
TupleXor computes the logical exclusive or of the input tuples \( T_1 \) and \( T_2 \). If both tuples have the same length, the operation is performed on the corresponding elements of both tuples. Otherwise, either \( T_1 \) or \( T_2 \) must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

Parameter

- **T1** (input control) .......................................................... integer(-array) \( \sim \) VARIANT ( integer )
  Input tuple 1.
- **T2** (input control) .......................................................... integer(-array) \( \sim \) VARIANT ( integer )
  Input tuple 2.
- **Xor** (output control) ...................................................... integer(-array) \( \sim \) VARIANT ( integer )
  Binary exclusive or of the input tuples.

Parallelization Information

TupleXor is *reentrant* and processed *without* parallelization.

See Also

TupleBand, TupleBor, TupleBxor, TupleBnot

 TupleAnd, TupleOr, TupleNot

Alternatives

Module Operators not requiring licensing

### 13.9 Selection

Select the first elements of a tuple.

TupleFirstN selects the first elements of \( \text{Tuple} \) and returns them with \( \text{Selected} \). Thus \( \text{Selected} \) contains all elements of \( \text{Tuple} \) from the first element up to the “n-th” element of \( \text{Tuple} \) (including the “n-th” element). The index “n” is determined by the input parameter \( \text{Index} \). Thus \( \text{Index} \) must contain a single integer value (if \( \text{Index} \) consists of a floating point number, this must represent an integer value without fraction). Indices of tuple elements start at 0, that means, the first tuple element has got the index 0.

Parameter

- **Tuple** (input control) ..................................................number(-array) \( \sim \) VARIANT ( real, integer, string )
  Input tuple.
- **Index** (input control) ....................................................number \( \sim \) VARIANT ( real, integer )
  Index of the first element to select.
- **Selected** (output control) .................................................number(-array) \( \sim \) VARIANT ( real, integer, string )
  Selected tuple elements.

Parallelization Information

TupleFirstN is *reentrant* and processed *without* parallelization.

Alternatives

TupleLastN, TupleSelect, TupleLastN, TupleStrBitSelect, TupleConcat

Module Operators not requiring licensing
Select all elements from index “n” to the end of a tuple.

Starting with the “n-th” element of the tuple Tuple, TupleLastN selects every element of Tuple and returns it with Selected. Thus Selected contains all elements of Tuple from index “n” up to the last element of Tuple (including the element at position “n”). The index “n” is determined by the input parameter Index. Thus Index must contain a single integer value (if Index consists of a floating point number, this must represent an integer value without fraction). Indices of tuple elements start at 0, that means, the first tuple element has got the index 0.

Parameter

- **Tuple** (input control) number(-array) VARIANT (real, integer, string)
  Input tuple.
- **Index** (input control) number VARIANT (real, integer)
  Index of the first element to select.
- **Selected** (output control) number(-array) VARIANT (real, integer, string)
  Selected tuple elements.

Parallelization Information

TupleLastN is reentrant and processed without parallelization.

Alternatives

TupleFirstN, TupleSelect, TupleStrBitSelect, TupleConcat

Module

Operators not requiring licensing

Select single elements of a tuple.

TupleSelect selects one or more single elements of the tuple Tuple and returns them with Selected. At this, Index determines the indices of the elements to select. Thus Index may only contain integer values (any floating point number within Index must represent an integer value without fraction). Indices of tuple elements start at 0, that means, the first tuple element has got the index 0.

Parameter

- **Tuple** (input control) number(-array) VARIANT (real, integer, string)
  Input tuple.
- **Index** (input control) number(-array) VARIANT (real, integer)
  Indices of the elements to select.
- **Selected** (output control) number(-array) VARIANT (real, integer, string)
  Selected tuple element.

Parallelization Information

TupleSelect is reentrant and processed without parallelization.

Alternatives

TupleFirstN, TupleLastN, TupleStrBitSelect, TupleConcat

Module

Operators not requiring licensing
13.9. SELECTION

|----------------------------------------------------------------------------------------------------------------------------------|

Select several elements of a tuple.

TupleSelectRange selects several consecutive elements of the input tuple Tuple and returns them with Selected. At this, Leftindex determines the index of the first element and Rightindex determines the index of the last element to select. Thus both parameters Leftindex and Rightindex must contain a single integer value (any floating point number must represent an integer value without fraction). Indices of tuple elements start at 0, that means, the first tuple element has got the index 0. The result tuple Selected contains every element from the tuple Tuple that has got an index between Leftindex and Rightindex (including the elements at position Leftindex and Rightindex). The index Rightindex must be greater or equal to Leftindex. If the parameters contain equal values, only one element is selected.

---

Parameter

- **Tuple** (input control) ........................................ number(-array) ~ VARIANT ( real, integer, string )
  Input tuple.
- **Leftindex** (input control) .............................. number(-array) ~ VARIANT ( real, integer )
  Index of first element to select.
- **Rightindex** (input control) .............................. number(-array) ~ VARIANT ( real, integer )
  Index of last element to select.
- **Selected** (output control) .............................. number(-array) ~ VARIANT ( real, integer, string )
  Selected tuple elements.

---

Parallelization Information

TupleSelectRange is **reentrant** and processed **without** parallelization.

---

Alternatives

TupleSelect, TupleFirstN, TupleLastN, TupleStrBitSelect, TupleConcat

---

Module

Operators not requiring licensing

<table>
<thead>
<tr>
<th>[out] VARIANT Selected HTupleX.TupleStrBitSelect ([in] VARIANT Tuple, [in] VARIANT Index )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void HOperatorSetX.TupleStrBitSelect ([in] VARIANT Tuple, [in] VARIANT Index, [out] VARIANT Selected )</td>
</tr>
</tbody>
</table>

Select single character or bit from a tuple.

TupleStrBitSelect selects a single character or bit from a tuple Tuple of integer numbers and/or strings. The input parameter Index determines the character or bit position to select. Index must contain a single number. If Index contains a floating point number, this may only represent an integer value (without fraction). The result tuple Selected contains a new element for each element of Tuple. Let Index contain the number “n” then each element of Selected consists of the “n-th” character (for strings) or “n-th” bit (for integers) of the corresponding element of Tuple.

---

Parameter

- **Tuple** (input control) ........................................ number(-array) ~ VARIANT ( string, integer )
  Input tuple.
- **Index** (input control) .................................... number ~ VARIANT ( real, integer )
  Position of character or bit to select.
- **Selected** (output control) ................................ number(-array) ~ VARIANT ( string, integer )
  Tuple containing the selected characters and bits.

---

Parallelization Information

TupleStrBitSelect is **reentrant** and processed **without** parallelization.

HALCON 6.0
Alternatives
TupleStrBitSelect, TupleSelect, TupleFirstN, TupleLastN, TupleConcat, TupleStrchr, TupleStrrchr, TupleStrFirstN, TupleStrLastN, TupleAnd, TupleOr, TupleXor, TupleNot

Module

Operators not requiring licensing

13.10 String-Operators

Parameters

- **Names** (input control) ........................................string(-array) \(\sim\) VARIANT( string )
  Tuple containing name(s) of the environment variable(s).
- **Values** (output control) ........................................string(-array) \(\sim\) VARIANT( string )
  Content of the environment variable(s).

Parallelization Information

TupleEnvironment is reentrant and processed without parallelization.

Alternatives

TupleStrstr, TupleStrrstr, TupleStrchr, TupleStrrchr, TupleStrlen, TupleStrFirstN, TupleStrLastN, TupleSplit

Module

Operators not requiring licensing

Parameters

- **Splitted** (output control) .....................................string(-array)
  Content of the environment variable(s).

Parallelization Information

TupleSplit is reentrant and processed without parallelization.

Alternatives

TupleStrstr, TupleStrrstr, TupleStrchr, TupleStrrchr, TupleStrlen, TupleStrFirstN, TupleStrLastN, TupleSplit

Module

Operators not requiring licensing

String-Operators

**TupleEnvironment**

Read one or more environment variables.

TupleEnvironment reads the content of all environment variables that are referenced by their names in the input tuple Names and returns the content with the output tuple Values. The input tuple may only contain strings. An empty string is returned for every name within Names that does not denote a valid environment variable.

**Parameter**

- **Names** (input control) ........................................string(-array) \(\sim\) VARIANT( string )
  Tuple containing name(s) of the environment variable(s).
- **Values** (output control) ........................................string(-array) \(\sim\) VARIANT( string )
  Content of the environment variable(s).

**TupleSplit**

Split strings into substrings between predefined separator symbol(s).

TupleSplit searches within the strings of the input tuple T1 for one or more separators defined in the input tuple T2. TupleSplit then splits the examined strings into the substrings between the separators. Both input tuples may only consist of strings. Otherwise TupleSplit returns an error. If the elements of T2 contain more than one character, each character defines a separator. If T1 consists only of one string, this is split up several times according to the elements of T2. For example: If T1 consists of the string “data1;data2:7;data3” and T2 contains the strings “;” and “:;”, the output tuple Splitted will comprise the strings “data1”, “data2:7”, “data3” as the result of splitting the string of T1 according to the first element of T2 and “data1”, “data2”, “7” and “data3” as the result of splitting according to the second element of T2. If both input tuples show the same number of elements, the search is done elementwise. I.e., TupleSplit will split the first string of T1 according to the separators in the first element of T2, the second string of T1 according to the separators in the second element of T2 and so on. If T2 only contains one string, the separators defined in this string will be used to split up all the strings of T1. If both input tuples contain more than one element and the number of elements differs for the input tuples, TupleSplit returns an error.
Parameter

- **T1** (input control) ........................................ string(-array) ~ VARIANT( string )
  Input tuple 1.
- **T2** (input control) ........................................ string(-array) ~ VARIANT( string )
  Input tuple 2.
- **Substring** (output control) ................................. string(-array) ~ VARIANT( string )
  The first characters up to position “n” of each string.

Parallelization Information

TupleStrFirstN is **reentrant** and processed **without** parallelization.

Alternatives

TupleStrLastN, TupleStrstr, TupleStrrstr, TupleStrchr, TupleStrrchr, TupleStrlen, TupleStrFirstN, TupleSplit, TupleEnvironment

Operators not requiring licensing

```plaintext
[out] VARIANT Substring HTupleX.TupleStrFirstN ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleStrFirstN ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Substring )
```

Cut the first characters up to position “n” out of string tuple.

TupleStrFirstN cuts the first characters up to position “n” out of each string of the input tuple T1 and returns them as new strings in the output tuple Substring (remark: the position within strings starts with 0 for the first character of a string). The number “n” is determined by the second input tuple T2. If T2 only contains one element, this element contains the number “n”. If T1 and T2 have got the same number of elements, the first element of T2 determines the number “n” of characters to cut out of the first string of T1, the second element of T2 does so for the second string of T1 and so on. If T2 contains more than one element and T1 contains only one string, TupleStrFirstN cuts more than one substrings out of this string. The elements of T2 then determine the lengths of these substrings. If both input tuples contain more than one element but differ in the number of elements, TupleStrFirstN returns an error.

Parameter

- **T1** (input control) ........................................ string(-array) ~ VARIANT( string )
  Input tuple 1.
- **T2** (input control) ........................................ number(-array) ~ VARIANT( real, integer )
  Input tuple 2.
- **Substring** (output control) ................................. string(-array) ~ VARIANT( string )
  The first characters up to position “n” of each string.

Parallelization Information

TupleStrFirstN is **reentrant** and processed **without** parallelization.

Alternatives

TupleStrLastN, TupleStrstr, TupleStrrstr, TupleStrchr, TupleStrrchr, TupleStrlen, TupleStrFirstN, TupleSplit, TupleEnvironment

Operators not requiring licensing

```plaintext
[out] VARIANT Substring HTupleX.TupleStrLastN ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleStrLastN ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Substring )
```

Cut all characters starting at position “n” out of string tuple.
TupleStrLastN cuts all characters from position “n” to the end of the string out of each string of the input tuple T1 and returns them as new strings in the output tuple Substring. The position “n” is determined by the second input tuple T2. If T2 only contains one element, this element contains “n”. If T1 and T2 have got the same number of elements, the first element of T2 determines the start position for the first string of T1, the second element of T2 does so for the second string of T1 and so on. If T2 contains more than one element and T1 contains only one string, TupleStrLastN cuts more than one substrings out of this string. The elements of T2 then determine the start positions for these substrings. If both input tuples contain more than one element but differ in the number of elements, TupleStrLastN returns an error.

Parameter

- **T1** (input control) string(-array) ~ VARIANT ( string )
  Input tuple 1.
- **T2** (input control) number(-array) ~ VARIANT ( real, integer )
  Input tuple 2.
- **Substring** (output control) string(-array) ~ VARIANT ( string )
  The last characters starting at position “n”.

Parallelization Information

TupleStrLastN is reentrant and processed without parallelization.

Alternatives

TupleStrLastN, TupleStrstr, TupleStrrstr, TupleStrlen, TupleStrchr, TupleStrrchr, TupleSplit, TupleEnvironment

Module

Operators not requiring licensing

---

```
[out] VARIANT Position HTupleX.TupleStrchr ([in] VARIANT T1,
[in] VARIANT T2 )

void HOperatorSetX.TupleStrchr ([in] VARIANT T1, [in] VARIANT T2,
[out] VARIANT Position )
```

Forward search for a character within a string tuple.

TupleStrchr searches within the strings of the input tuple T1 for the characters of the input tuple T2. Both input tuples may only consist of strings. Otherwise TupleStrchr returns an error. If the elements of T2 contain more than one character, only the first character of each element is considered for searching. If T1 contains only one string, all the characters defined in T2 are searched in this string. Thus the output tuple consists of as many elements as T2. Whenever a searched character has been found, the position of its first occurrence gets stored in the output tuple Position (remark: the position starts at 0 for the first character of a string). If a character can not be found, -1 will be returned instead of its position. If both input tuples show the same number of elements, the search is done elementwise. I.e., the first character of the first element of T2 is searched within the first string of T1, the first character of the second element of T2 is searched within the first string of T1 and so on. The results of the elementwise searches are returned with Position that contains as many elements as T1 and T2. If T2 only contains one string, its first character is searched within all strings of T1. Thus in this case Position consists of as many elements as T1. If both input tuples contain more than one element and the number of elements differs for the input tuples, TupleStrchr returns an error.

Parameter

- **T1** (input control) string(-array) ~ VARIANT ( string )
  Input tuple 1.
- **T2** (input control) string(-array) ~ VARIANT ( string )
  Input tuple 2.
- **Position** (output control) integer(-array) ~ VARIANT ( integer )
  Position of searched character(s) within the string(s).

Parallelization Information

TupleStrchr is reentrant and processed without parallelization.

Alternatives

TupleStrrchr, TupleStrstr, TupleStrrstr, TupleStrlen, TupleStrFirstN, TupleStrLastN, TupleSplit, TupleEnvironment
13.10. STRING-OPERATORS

Module

Operators not requiring licensing

```c
[out] VARIANT Length HTupleX.TupleStrlen ([in] VARIANT T1 )
void HOperatorSetX.TupleStrlen ([in] VARIANT T1, [out] VARIANT Length )
```

Length of every string within a tuple of strings.

TupleStrlen checks the length of every string within the input tuple T1 and returns the length of each string with the output tuple Length. All elements of T1 may only consist of strings. Otherwise TupleStrlen returns an error.

Parameter

- **T1** (input control) ........................................... number(-array) ↼ VARIANT ( integer, real, string )
  - Input tuple.
- **Length** (output control) ................................. integer(-array) ↼ VARIANT ( integer )
  - Length of the single strings of the input tuple.

Parallelization Information

TupleStrlen is reentrant and processed without parallelization.

Alternatives

TupleStrstr, TupleStrrstr, TupleStrchr, TupleStrrchr, TupleStrFirstN, TupleStrLastN, TupleSplit, TupleEnvironment

Module

Operators not requiring licensing

```c
[out] VARIANT Position HTupleX.TupleStrrchr ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleStrrchr ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Position )
```

Backward search for a character within a string tuple.

TupleStrrchr searches within the strings of the input tuple T1 for the characters of the input tuple T2. Both input tuples may only consist of strings. Otherwise TupleStrrchr returns an error. In any case backward search is used, i.e., every string is examined from its last to its first character. If the elements of T2 contain more than one character, only the first character of each element is considered for searching. If T1 contains only one string, all the characters defined in T2 are searched in this string. Thus the output tuple consists of as many elements as T2. Whenever a searched character has been found, the position of its first occurrence gets stored in the output tuple Position. If a character can not be found, -1 will be returned instead of its position (remark: the position starts at 0 for the first character of a string). If both input tuples show the same number of elements, the search is done elementwise. I.e., the first character of the first element of T2 is searched within the first string of T1, the first character of the second element of T2 is searched within the second string of T1 and so on. The results of the elementwise searches are returned with Position that contains as many elements as T1 and T2. If T2 only contains one string, its first character is searched within all strings of T1. Thus in this case Position consists of as many elements as T1. If both input tuples contain more than one element and the number of elements differs for the input tuples, TupleStrrchr returns an error.

Parameter

- **T1** (input control) ........................................... string(-array) ↼ VARIANT ( string )
  - Input tuple 1.
- **T2** (input control) ........................................... string(-array) ↼ VARIANT ( string )
  - Input tuple 2.
- **Position** (output control) ................................. integer(-array) ↼ VARIANT ( integer )
  - Position of searched character within the string.
Parallelization Information

TupleStrrc is reentrant and processed without parallelization.

Alternatives

TupleStrchr, TupleStrstr, TupleStrrstr, TupleStrlen, TupleStrFirstN, TupleStrLastN, TupleSplit, TupleEnvironment

Operators not requiring licensing

<table>
<thead>
<tr>
<th>[out] VARIANT Position</th>
<th>HTupleX.TupleStrrstr ([in] VARIANT T1, [in] VARIANT T2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void HOperatorSetX.TupleStrrstr ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Position )</td>
<td></td>
</tr>
</tbody>
</table>

Backward search for string(s) within a string tuple.

TupleStrrstr searches within the strings of the input tuple T1 for the strings of the input tuple T2. Both input tuples may only consist of strings. Otherwise TupleStrrstr returns an error. In any case backward search is used, i.e., every string is examined from its last to its first character. If T1 contains only one string, all strings of T2 are searched in it. Thus the output tuple consists of as many elements as T2. Whenever a searched string has been found, the position of its first occurrence gets stored in the output tuple Position (positions in strings are counted starting with 0). If a string can not be found, -1 will be returned instead of its position. If both input tuples show the same number of elements, the strings are searched elementwise. I.e., the first string of T2 is searched within the first string of T1, the second string of T2 is searched within the second string of T1 and so on. The results of the elementwise searches are returned with Position that contains as many elements as T1 and T2. If T2 only contains one string, this is searched within all strings of T1. Thus in this case Position consists of as many elements as T1. If both input tuples contain more than one element and the number of elements differs for the input tuples, TupleStrrstr returns an error.

Parameter

▶ T1 (input control) ........................................ string(-array) ~ VARIANT (string) Input tuple 1.
▶ T2 (input control) ........................................ string(-array) ~ VARIANT (string) Input tuple 2.
▶ Position (output control) ........................... integer(-array) ~ VARIANT (integer) Position of searched string(s) within the examined string(s).

Parallelization Information

TupleStrrstr is reentrant and processed without parallelization.

Alternatives

TupleStrchr, TupleStrlen, TupleStrrc, TupleStrrchr, TupleStrFirstN, TupleStrLastN, TupleSplit, TupleEnvironment

Module

Operators not requiring licensing

<table>
<thead>
<tr>
<th>[out] VARIANT Position</th>
<th>HTupleX.TupleStrstr ([in] VARIANT T1, [in] VARIANT T2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void HOperatorSetX.TupleStrstr ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Position )</td>
<td></td>
</tr>
</tbody>
</table>

Forward search for string(s) within a string tuple.

TupleStrstr searches within the strings of the input tuple T1 for the strings of the input tuple T2. Both input tuples may only consist of strings. Otherwise TupleStrstr returns an error. If T1 contains only one string, all strings of T2 are searched in it. Thus the output tuple consists of as many elements as T2. Whenever a searched string has been found, the position of its first occurrence gets stored in the output tuple Position (positions in...
strings are counted starting with 0). If a string can not be found, -1 will be returned instead of its position. If both input tuples show the same number of elements, the strings are searched elementwise. I.e., the first string of T2 is searched within the first string of T1, the second string of T2 is searched within the second string of T1 and so on. The results of the elementwise searches are returned with Position that contains as many elements as T1 and T2. If T2 only contains one string, this is searched within all strings of T1. Thus in this case Position consists of as many elements as T1. If both input tuples contain more than one element and the number of elements differs for the input tuples, TupleStrstr returns an error.

Parameter

- T1 (input control) .................................................. string(-array) → VARIANT ( string )
  Input tuple 1.
- T2 (input control) .................................................. string(-array) → VARIANT ( string )
  Input tuple 2.
- Position (output control) .............................. integer(-array) → VARIANT ( integer )
  Position of searched string(s) within the examined string(s).

Parallelization Information

TupleStrstr is reentrant and processed without parallelization.

Alternatives

TupleStrstr, TupleStrlen, TupleStrchr, TupleStrrchr, TupleStrFirstN, TupleStrLastN, TupleSplit, TupleEnvironment

Module

Operators not requiring licensing
Chapter 14

XLD

14.1 Access

```
[out] VARIANT Row HXLDContX.GetContourXld ([out] VARIANT Col )
void HOperatorSetX.GetContourXld ([in] IObjectX Contour, [out] VARIANT Row, [out] VARIANT Col )
```

Return the coordinates of an XLD contour.

GetContourXld returns the following values of the XLD contour Contour:

- **Row**: Row coordinate of the contour’s points
- **Col**: Column coordinate of the contour’s points

---

Parallelization Information

GetContourXld is reentrant and processed without parallelization.

---

Possible Predecessor Functions

- GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

---

See Also

- GetContourAttribXld, QueryContourAttribsXld, GetContourGlobalAttribXld, QueryContourGlobalAttribsXld

---

Module

Sub-pixel operators

```
```

Return an XLD polygon’s data (as lines).

GetLinesXld returns the XLD polygon Polygon as a set of lines. The following values are returned:
GetLinesXld is reentrant and automatically parallelized (on tuple level).

class HXLDX

possible Predecessor Functions

GetPolygonXld

module

Sub-pixel operators

Parameter

Parallels

Input XLD parallels.

Row1

Row coordinates of the points on polygon P1.

Col1

Column coordinates of the points on polygon P1.

Length1

Lengths of the line segments on polygon P1.

Phi1

Angles of the line segments on polygon P1.

Row2

Row coordinates of the points on polygon P2.

Col2

Column coordinates of the points on polygon P2.

Length2

Lengths of the line segments on polygon P2.

Phi2

Angles of the line segments on polygon P2.

Return an XLD parallel’s data (as lines).

GetParallelsXld returns the following values of the XLD parallels Parallels:

Row1: Row coordinates of the points on polygon P1
Col1: Column coordinates of the points on polygon P1
Length1: Lengths of the line segments on polygon P1
Phi1: Angles of the line segments on polygon P1
Row2: Row coordinates of the points on polygon P2
Col2: Column coordinates of the points on polygon P2
Length2: Lengths of the line segments on polygon P2
Phi2: Angles of the line segments on polygon P2

Input XLD parallels.

Row1: Row coordinates of the points on polygon P1.

Return an XLD parallel’s data (as lines).

GetParallelsXld returns the following values of the XLD parallels Parallels:
GetPolygonXld returns the following values of the XLD polygon Polygon:

- **Row**: Row coordinates of the polygons’ points
- **Col**: Column coordinates of the polygons’ points
- **Length**: Lengths of the line segments between points \( i \) and \( i+1 \), respectively
- **Phi**: Angles of the line segments between points \( i \) and \( i+1 \), respectively

**Parameter**

- **Polygon** (input iconic) ~~~~ xld_poly ~~~~ HXLDPolyX / IHObjectX
  - Input XLD polygon.
- **Row** (output control) ~~~~ point.y ~~~~ VARIANT ( real )
  - Row coordinates of the polygons’ points.
- **Col** (output control) ~~~~ point.x ~~~~ VARIANT ( real )
  - Column coordinates of the polygons’ points.
- **Length** (output control) ~~~~ real ~~~~ VARIANT ( real )
  - Lengths of the line segments.
- **Phi** (output control) ~~~~ angle.rad ~~~~ VARIANT ( real )
  - Angles of the line segments.

**Parallelization Information**

GetPolygonXld is reentrant and processed without parallelization.

**Possible Predecessor Functions**

GetPolygonsXld, GetLinesXld

**See Also**

Sub-pixel operators

```cpp
[out] VARIANT Row HXLDPolyX.GetPolygonXld ([out] VARIANT Col,
[out] VARIANT Length, [out] VARIANT Phi )
```

```cpp
void HOperatorSetX.GetPolygonXld ([in] IHObjectX Polygon,
[out] VARIANT Row, [out] VARIANT Col, [out] VARIANT Length,
[out] VARIANT Phi )
```
Sub-pixel operators

### 14.2 Creation

#### void HXLDContX.GenContourPolygonXld ([in] VARIANT Row, [in] VARIANT Col )


Generate an XLD contour from a polygon (given as tuples).

GenContourPolygonXld generates an XLD contour Contour from a polygon given in the tuples Row and Col. This operator is useful if contours have been obtained from routines outside the core library, but higher level operators, e.g., polygon approximation and extraction of parallels, are to be performed on the contours.

**Parameter**

- **Contour** (output iconic) ......................... xld_cont ~ HXLDContX / HUntypedObjectX
  - Resulting contour.
- **Row** (input control) ........................... number ~ VARIANT ( real, integer )
  - Row coordinates of the polygon.
  - Default Value : [0,1,2,2,2]
  - Value Suggestions : Row \in \{0, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500\}
- **Col** (input control) ........................... number ~ VARIANT ( real, integer )
  - Column coordinates of the polygon.
  - Default Value : [0,0,0,1,2]
  - Value Suggestions : Col \in \{0, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500\}

**Parallelization Information**

GenContourPolygonXld is reentrant and processed without parallelization.

**Possible Predecessor Functions**

GetRegionContour

**Possible Successor Functions**

SmoothContoursXld, GenPolygonsXld

**See Also**

GenContoursSkeletonXld

---

Sub-pixel operators

#### [out] HXLDContX Contours HRegionX.GenContourRegionXld ([in] String Mode )


Generate XLD contours from regions.

GenContourRegionXld generates XLD contours Contours from the regions given in Regions. This operator is useful if regions have been obtained from segmentation operations, but higher level operators, e.g., polygon approximation and extraction of parallels, are to be performed on their boundaries. For each connected component of the input regions a closed contour of the boundary is generated. The parameter Mode can take on the following values:

- 'center': The centers of the border pixels are used as contour points.
- 'border': The outer border of the border pixels is used as contour points.
The difference between the two modes can be seen by considering the following region:

where □ symbolizes a single pixel.

Then, computing the contour with 'border' and 'center' results in the following two contours, respectively:

This means, for example, that contours generated with 'border' will in general have a much larger Euclidean length (see LengthXld) than contours generated with 'center'. This results from the fact that for diagonal border elements 'border' uses two contour segments of length 1 each, whereas 'center' uses a single segment of length $\sqrt{2}$. Other features, e.g., the area (see AreaCenterXld), will obviously also have different values.

---

**Parameter**

- **Regions** (input iconic) region(-array) → HRegionX / IObjectX
  Input regions.
- **Contours** (output iconic) xld_cont(-array) → HXLDCntX / HUntypedObjectX
  Resulting contours.
- **Mode** (input control) string → String
  Mode of contour generation.
  **Default Value**: 'border'
  **Value List**: Mode ∈ {'border', 'center'}

---

**Parallelization Information**

GenContourRegionXld is *reentrant* and processed *without* parallelization.

---

**Possible Successor Functions**

SmoothContoursXld, GenPolygonsXld

**See Also**

GenContoursSkeletonXld

**Alternatives**

GenContourPolygonXld, GetRegionContour

---

Sub-pixel operators

```
[out] HXLDCntX Contours HRegionX.GenContoursSkeletonXld
([in] long Length, [in] String Mode )

void HOperatorSetX.GenContoursSkeletonXld ([in] IObjectX Skeleton,
```

**Convert a skeleton into XLD contours.**

GenContoursSkeletonXld converts the input skeleton (e.g., edges) Skeleton, which is assumed to contain mostly one pixel wide regions (see Skeleton), into XLD contours. The regions are first transformed to contain only line segments in 8-neighborhood. In the process 12 special configurations are taken into account: Points for which there is a junction of three or more lines in 8-neighborhood are preserved (in all four rotations):
In a second step, all junction points are labelled, taking six different characteristic configurations of all four possible rotations into account:

\[
\begin{array}{ccccccc}
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
\end{array}
\]

where 0 = background, 1 = foreground, and 2 = junction point.

After this, all contours having at least \( \text{Length} \) points are returned. The contours generated by \text{GenContoursSkeletonXld} always end in junction or end points. For closed contours the first point lies in the 8-neighborhood of the last point of the contour. Therefore, in order to determine the adjacency of contours it is sufficient to just take the end points into account.

Since contours are split at junction points, possibly long contours may be split into several short segments because of short adjacent lines, even if they are longer than \( \text{Length} \) points, if \text{Mode} = 'filter' was selected. This can be avoided by setting \text{Mode} = 'generalize1'. In this case, the contours are generated as if the segments shorter than \( \text{Length} \) were not contained in the input region. In order to preserve line segments, which are split into very short segments by the crossing of short lines, \text{Mode} = 'generalize2' can be selected. In this case, line segments are preserved if they end in two junction points, even if they are shorter than \( \text{Length} \).

---

**Parameter**

- **Skeleton** (input iconic) .............................. region \( \rightarrow \) \text{HRegionX} / \text{HObjectX}
  - Skeleton of which the contours are to be determined.
- **Contours** (output iconic) .............................. \text{xld_cont} \( \rightarrow \) \text{HXLDContX} / \text{HUntypedObjectX}
  - Resulting contours.
- **Length** (input control) .............................. integer \( \rightarrow \) \text{long}
  - Minimum number of points a contour has to have.
    - **Default Value**: 1
    - **Value Suggestions**: \( \text{Length} \in \{1, 2, 3, 5, 10, 20\} \)
- **Mode** (input control) .............................. string \( \rightarrow \) \text{String}
  - Contour filter mode.
    - **Default Value**: 'filter'
    - **Value List**: \( \text{Mode} \in \{\text{‘filter’, ‘generalize1’, ‘generalize2’}\} \)

---

**Parallelization Information**

\text{GenContoursSkeletonXld} is reentrant and processed without parallelization.

**Possible Predecessor Functions**

\text{SmoothContoursXld}, \text{GetContourXld}, \text{GenPolygonsXld}

**Possible Successor Functions**

\text{EdgesImage}, \text{Threshold}, \text{GetRegionContour}

**See Also**

\text{Sub-pixel operators}

---

Creation of an XLD contour corresponding to an elliptic arc.

GenEllipseContourXld creates one or more elliptic arcs or closed ellipses. Ellipses are specified by their center (Row, Column), the orientation of the main axis Phi, the length of the larger half axis Radius1, and the length of the smaller half axis Radius2. In addition to that, elliptic arcs are characterized by the angle of the start point StartPhi, the angle of the end point EndPhi, and the point order PointOrder along the boundary. The angles in the interval \([0, 2\pi]\) are measured in the coordinate system of the ellipse relative to the main axis. Thus, the two main poles correspond to the angles 0 and \(\pi\), the two other poles to the angles \(\pi/2\) and \(3\pi/2\). To create a closed ellipse the values 0 and \(2\pi\) (with positive point order) have to be passed to the operator. The resolution of the resulting contours ContEllipse is controlled via Resolution containing the maximum Euclidean distance between neighboring contour points.

**Parameter**

- **ContEllipse** (output iconic) .......................... xld_cont(-array) ↳ HXLDContX / HUntypedObjectX Resulting contour.
- **Row** (input control) .......................... ellipse.center.y(-array) ↳ VARIANT (real) Row coordinate of the center of the ellipse.
- **Column** (input control) .......................... ellipse.center.x(-array) ↳ VARIANT (real) Column coordinate of the center of the ellipse.
- **Phi** (input control) .......................... ellipse.angle.rad(-array) ↳ VARIANT (real) Orientation of the main axis [rad].
  - Restriction : \(((\text{Phi} \geq 0) \land (\text{Phi} \leq 6.283185307))\)
- **Radius1** (input control) .......................... ellipse.radius1(-array) ↳ VARIANT (real) Length of the larger half axis.
  - Restriction : \((\text{Radius1} > 0)\)
- **Radius2** (input control) .......................... ellipse.radius2(-array) ↳ VARIANT (real) Length of the smaller half axis.
  - Restriction : \((\text{Radius2} \geq 0)\)
- **StartPhi** (input control) .......................... real(-array) ↳ VARIANT (real) Angle of the start point [rad].
  - Restriction : \(((\text{StartPhi} \geq 0) \land (\text{StartPhi} \leq 6.283185307))\)
- **EndPhi** (input control) .......................... real(-array) ↳ VARIANT (real) Angle of the end point [rad].
  - Restriction : \(((\text{EndPhi} \geq 0) \land (\text{EndPhi} \leq 6.283185307))\)
- **PointOrder** (input control) .......................... string(-array) ↳ VARIANT (string) point order along the boundary.
  - Value List : PointOrder ∈ \{‘positive’, ‘negative’\}
- **Resolution** (input control) .......................... real ↳ double Resolution: Maximum distance between neighboring contour points.
  - Default Value : 1.5
  - Restriction : \((\text{Resolution} > 0)\)

**Example**

draw_ellipse(WindowHandle, Row, Column, Phi, Radius1, Radius2)
gen_ellipse_contour_xld(Ellipse, Row, Column, Phi, Radius1, Radius2, 0, 6.28319, ‘positive’, 1.5)
length_xld(Ellipse, Length).

HALCON 6.0
Result

GenEllipseContourXld returns TRUE if all parameter values are correct. If necessary, an exception is raised.

Parallelization Information

GenEllipseContourXld is reentrant and processed without parallelization.

DrawEllipse

Possible Predecessor Functions

DispXld, GetPointsEllipse

Possible Successor Functions

Module

Sub-pixel operators

Extract parallel XLD polygons.

GenParallelsXld examines the XLD polygons passed in Polygons for parallelism. The resulting parallel polygons are returned in Parallels. If the parameter Merge is set to 'true', adjacent parallel polygons are returned in a single parallel relation. Otherwise, one parallel relation is returned for each pair of parallel line segments. Whether two polygon segments are parallel depends on their distance (smaller than Dist), a maximum allowed angle difference (Alpha, in radians), and a minimum length of the two polygon segments. Furthermore, the two segments have to overlap. As a side effect, a quality factor is calculated for each pair of parallels. It is based on the normalized angular difference and the normalized length of the overlapping area:

\[
\text{quality} = \frac{\pi - \delta \alpha}{\pi/2} \frac{2 \text{overlap}}{\text{len}_1 + \text{len}_2} \quad \text{with} \quad 0 \leq q \leq 1
\]

Here, \(\delta \alpha\) is the angle difference of the polygon segments, overlap is the length of the overlap area, \(\text{len}_1\) and \(\text{len}_2\) the length of the polygon segments, and quality the resulting quality factor.

The quality factor is a measure of parallelism (the larger its value, the “more parallel” the polygons). Finally, the quality factors of all parallel polygon segments contained in a single polygon are added, weighted with their length of the overlapping area.

Parameter

- **Polygons** (input iconic) .......................... xld_poly \(\leadsto\) HXLDPolyX / IObjectX
  - Input polygons.

- **Parallels** (output iconic) .......................... xld_para \(\leadsto\) HXLDParaX / HUntypedObjectX
  - Parallel polygons.

- **Len** (input control) ................................. number \(\leadsto\) VARIANT (real, integer)
  - Minimum length of the individual polygon segments.
  - **Default Value:** 10.0
  - **Value Suggestions:** Len \(\in\) \{5.0, 10.0, 15.0, 20.0\}
  - **Restriction:** (Len > 0.0)

- **Dist** (input control) ................................. number \(\leadsto\) VARIANT (real, integer)
  - Maximum distance between the polygon segments.
  - **Default Value:** 30.0
  - **Value Suggestions:** Dist \(\in\) \{20.0, 25.0, 30.0, 40.0, 50.0, 75.0\}
  - **Restriction:** (Dist > 0.0)

- **Alpha** (input control) ............................. number \(\leadsto\) VARIANT (real, integer)
  - Maximum angle difference of the polygon segments.
  - **Default Value:** 0.15
  - **Value Suggestions:** Alpha \(\in\) \{0.05, 0.10, 0.15, 0.20, 0.30\}
  - **Restriction:** \((0 \leq \text{Alpha}) \land (\text{Alpha} \leq (\pi/2))\)
14.2. CREATION

Should adjacent parallel relations be merged?

**Default Value:** 'true'

**Value List:** Merge ∈ \{'true’, ‘false’\}

---

**Parallelization Information**

GenParallelsXld is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

---

**Possible Successor Functions**

GenParallelsXld, GetParallelsXld

---

Sub-pixel operators

---

Approximate XLD contours by polygons.

GenPolygonsXld approximates XLD contours (Contours) by polygons. The type of the approximation can be set by Type. The threshold for the approximation is set via Alpha. The procedure is able to process open as well as closed contours. The resulting approximating XLD polygons are returned in Polygons.

Contours can be approximated by the algorithms of Ramer, Ray, and Sato. The algorithm of Ramer approximates contours such that the Euclidian distance of the approximating polygon to the contour is at most Alpha pixel units. The algorithm of Ray does not need a threshold, and hence Alpha is ignored. Here, the contour is approximated by maximizing the line length, while minimizing the sum of the distances of the line segment from the contour. The algorithm of Sato produces a polygon point at the point of the contour in which the distance to the end points of the contour is maximal. The total approximation error for each iteration is then given by \((L - L')/L,\) where \(L\) is the Euclidian contour length, and \(L'\) is the length of the approximating polygon.

---

**Parameter**

- **Contours** (input iconic) \(\rightarrow\) xld_cont \(\sim\) HXLDContX / IHObjectX
  
  Contours to be approximated.

- **Polygons** (output iconic) \(\rightarrow\) xld_poly \(\sim\) HXLDPolyX / HUntypedObjectX
  
  Approximating polygons.

- **Type** (input control) \(\rightarrow\) String
  
  Type of approximation.

  **Default Value:** 'ramer'

  **Value List:** Type ∈ \{'ramer’, ‘ray’, ‘sato’\}

- **Alpha** (input control) \(\rightarrow\) VARIANT \(\sim\) (real, integer)
  
  Threshold for the approximation.

  **Default Value:** 2.0

  **Value Suggestions:** Alpha ∈ \{1.0, 1.5, 2.0, 3.0, 4.0\}

  **Restriction:** (\(\alpha > 0.0\))

---

GenPolygonsXld is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

---

**Possible Successor Functions**

GenParallelsXld, GetRegionPolygon

---

Sub-pixel operators

HALCON 6.0
Extract parallel XLD polygons enclosing a homogeneous area.

ModParallelsXld selects XLD parallels enclosing homogeneous areas from the input parallels Parallels. The parameter Image contains the corresponding gray value image.

Only parallels having a quality factor larger than Quality are examined. The algorithm performs parallel cross sections one pixel apart, and parallel to the line segments in the area of overlap between two parallel line segments. In the first iteration, the mean gray value for each of the lines in the cross section is calculated. In the second iteration, the standard deviations of the gray values along each line are computed.

If the mean gray value of an area between parallels lies in the interval [MinGray, MaxGray], and if the mean of all standard deviations is smaller than the upper threshold MaxStandard, the corresponding parallels are returned as modified parallels in ModParallels.

In a second step, all polygon segments adjacent to parallels bordering homogeneous areas are checked for homogeneity. To do so, a rectangle having the width of the last area enclosed by a modified parallel is constructed, and processed without parallelization. This process is continued as long as there are adjacent polygon segments. The polygons thus found are returned as extended parallels in ExtParallels.

**Parameter**

- **Parallels** (input iconic) ............................. xld para ~ HXLDParaX / IHObjectX
  - Input XLD parallels.

- **Image** (input iconic) ............................. image ~ HImageX / IHObjectX
  - Corresponding gray value image.

- **ModParallels** (output iconic) .................... xld_mod para ~ HXLDModParaX / HUntypedObjectX
  - Modified XLD parallels.

- **ExtParallels** (output iconic) ................. xldExtPara ~ HXLDExtParaX / HUntypedObjectX
  - Modified XLD parallels.

- **Quality** (input control) ............................. number ~ VARIANT (real, integer)
  - Minimum quality factor (measure of parallelism).
    - **Default Value**: 0.4
    - **Value Suggestions**: Quality \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}
    - **Restriction**: \((0.0 \leq \text{Quality}) \land (\text{Quality} \leq 1.0)\)

- **MinGray** (input control) ............................. integer ~ long
  - Minimum mean gray value.
    - **Default Value**: 160
    - **Value Suggestions**: MinGray \in \{80, 100, 120, 140, 160, 180\}
    - **Restriction**: \((0 \leq \text{MinGray}) \land (\text{MinGray} \leq 255)\)

- **MaxGray** (input control) ............................. integer ~ long
  - Maximum mean gray value.
    - **Default Value**: 220
    - **Value Suggestions**: MaxGray \in \{140, 160, 180, 200, 220, 240\}
    - **Restriction**: \((0 \leq \text{MaxGray}) \land (\text{MaxGray} \leq 255)\) \land (MaxGray \geq MinGray)

- **MaxStandard** (input control) ...................... number ~ VARIANT (real, integer)
  - Maximum allowed standard deviation.
    - **Default Value**: 10.0
    - **Value Suggestions**: MaxStandard \in \{5.0, 10.0, 15.0, 20.0\}
    - **Restriction**: \((\text{MaxStandard} \geq 0.0)\)

**Parallelization Information**

ModParallelsXld is reentrant and processed without parallelization.

**Possible Predecessor Functions**

GenParallelsXld
14.3 Features

Possible Successor Functions

MaxParallelsXld

See Also

InfoParallelsXld

Module

Sub-pixel operators

14.3 Features

Possible Predecessor Functions

GenContoursSkeletonXld, SmoothContoursXld, GenPolygonsXld

See Also

MomentsXld, AreaCenter, MomentsRegion2Nd

Module

Sub-pixel operators

Area and center of gravity (centroid) of contours and polygons.

AreaCenterXld calculates the area and center of gravity (centroid) of the region enclosed by the contours or polygons XLD as well as the order of the points along the boundary. The area and centroid are computed by applying Green’s theorem using only the points on the contour or polygon, i.e., no region is generated explicitly for the purpose of calculating the features. It is assumed that the contours or polygons are closed. If this is not the case AreaCenterXld will add one point to artificially produce a closed shape. If more than one contour or polygon is passed the results are stored as tuples in which the index of a value corresponds to the index of the respective contour or polygon in XLD.

Parameter

AreaCenterXld returns TRUE if the input is not empty. If the input is empty the behaviour can be set via SetSystem(::’noObjectResult’,<Result>);. If necessary, an exception is raised.

Parallelization Information

AreaCenterXld is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

GenContoursSkeletonXld, SmoothContoursXld, GenPolygonsXld

See Also

MomentsXld, AreaCenter, MomentsRegion2Nd

Module

Sub-pixel operators

Return the number of points in an XLD contour.
ContourPointNumXld returns the length of the input contour Contour, i.e. its number of points, in Length.

Parameter Contour (input iconic) returns the input XLD contour.

Parameter Length (output control) returns the length of the input contour.

Parallelization Information

ContourPointNumXld is reentrant and processed without parallelization.

Possible Predecessor Functions

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

Possible Successor Functions

GetContourXld, GetContourAttribXld

QueryContourAttribsXld

GetRegressParamsXld

See Also

Alternatives

Module

Sub-pixel operators

Distance of contours to an ellipse.

DistEllipseContourXld determines the distance between the contours in Contours and an ellipse specified by the center (Row, Column), the orientation of the main axis Phi, the length of the larger half axis Radius1, and the length of the smaller half axis Radius2. Different measures for the distance of a contour point \( X = (x_i, y_i) \) to the ellipse are available (Mode):

'**algebraic**' The distance is measured by the algebraic distance \( a x_i^2 + b x_i y_i + c y_i^2 + d x_i + e y_i + f \), where the parameters \( a - f \) describing the ellipse are normalized in order to obtain Radius2 as distance of the center of the ellipse. This measure shows a high curvature bias: Near points of high curvature on the ellipse (like the poles on the main axis) the distance is smaller than near points with low curvature.

'**geometric**' The distance is measured by the deviation \( X F_1 + X F_2 - 2a \), where \( F_1, F_2 \) are the focal points and \( a \) corresponds to Radius1. This measure shows a low curvature bias: Near points of high curvature on the ellipse (like the poles on the main axis) the distance is larger than near points with low curvature.

'**bisection**' The distance is measured by the distance between \( X \) and the intersection of the angular bisector of the two lines through \( X \) and the focal points with the ellipse. This is a very good approximation of the orthogonal distance from \( X \) to the ellipse.

The operator returns the minimum absolute distance MinDist, the maximum absolute distance MaxDist, the average absolute distance AvgDist, and the standard deviation of the absolute distances SigmaDist of all contour points.

To reduce the computational load, the computation of the distances can be restricted to a subset of the contour points: If a value other than -1 is assigned to MaxNumPoints, only up to MaxNumPoints points - uniformly distributed over the contour - are used. Due to artefacts in the pre-processing the start and end points of a contour might be faulty. Therefore, it is possible to exclude ClippingEndPoints at the beginning and at the end of a contour from the computation.
14.3. FEATURES

Parameter

- **Contours** (input iconic) : xld_cont(-array)  \sim  HXLDContX / IHOBJECTX
  Input contours.

- **Mode** (input control) : string  \sim  String
  Method for the determination of the distances.
  **Default Value**: 'bisec'
  **Value List**: Mode \in \{ 'geometric', 'algebraic', 'bisec' \}

- **MaxNumPoints** (input control) : integer  \sim  long
  Maximum number of contour points used for the computation (-1 for all points).
  **Default Value**: -1
  **Restriction**: \( MaxNumPoints \geq 3 \)

- **ClippingEndPoints** (input control) : integer  \sim  long
  Number of points at the beginning and the end of the contours to be ignored for the computation of distances.
  **Default Value**: 0
  **Restriction**: \( ClippingEndPoints \geq 0 \)

- **Row** (input control) : ellipse.center.y  \sim  double
  Row coordinate of the center of the ellipse.

- **Column** (input control) : ellipse.center.x  \sim  double
  Column coordinate of the center of the ellipse.

- **Phi** (input control) : ellipse.angle.rad  \sim  double
  Orientation of the main axis [rad].
  **Restriction**: \( (\Phi \geq 0) \land (\Phi \leq 6.283185307) \)

- **Radius1** (input control) : ellipse.radius1  \sim  double
  Length of the larger half axis.
  **Restriction**: \( Radius1 > 0 \)

- **Radius2** (input control) : ellipse.radius2  \sim  double
  Length of the smaller half axis.
  **Restriction**: \( Radius2 \geq 0 \)

- **MinDist** (output control) : real(-array)  \sim  VARIANT( real )
  <no description available>

- **MaxDist** (output control) : real(-array)  \sim  VARIANT( real )
  <no description available>

- **AvgDist** (output control) : real(-array)  \sim  VARIANT( real )
  <no description available>

- **SigmaDist** (output control) : real(-array)  \sim  VARIANT( real )
  <no description available>

Example

```plaintext
read_image (Image, 'caltab')
find_caltab (Image, Caltab, 'caltabBig.descr', 3, 112, 5)
reduce_domain (Image, Caltab, ImageReduced)
edges_sub_pix (ImageReduced, Edges, 'lanser2', 0.5, 20, 40)
select_contours_xld (Edges, EdgesClosed, 'closed', 0, 2.0, 0, 0)
select_contours_xld (EdgesClosed, EdgesMarks, 'length', 20, 80, 0, 0)
fit_ellipse_contour_xld (EdgesMarks, 'fitzgibbon', -1, 2, 0, 200, 3, 2.0,
                         Row, Column, Phi, Radius1, Radius2, StartPhi,
                         EndPhi, PointOrder).
```

Result

DistEllipseContourXld returns TRUE if all parameter values are correct. If necessary, an exception is raised.

Parallelization Information

DistEllipseContourXld is *local* and processed *completely exclusively* without parallelization.

Possible Predecessor Functions

- **FitEllipseContourXld**
- **SelectContoursXld**
- **FindCaltab**
- **ReadImage**
- **ReduceDomain**
- **EdgesSubPix**
- **SelectContoursXld**
- **FitEllipseContourXld**

Sub-pixel operators

HALCON 6.0
Approximation of XLD contours by circles.

FitCircleContourXld approximates the XLD contours Contours by circles. It does not perform a segmentation of the input contours. Thus, one has to make sure that each contour corresponds to one and only one circle. The operator returns for each contour the center (Row, Column), the Radius, the StartPhi and EndPhi.

The algorithm used for the fitting of circles can be selected via Algorithm:

- **algebraic** This approach minimizes the algebraic distance between the contour points and the resulting circle.
- **ahuber** Similar to `algebraic`. Here the contour points are weighted to decrease the impact of outliers based on the approach of Huber.
- **atukey** Similar to `algebraic`. Here the contour points are weighted to decrease the impact of outliers based on the approach of Tukey.

For `ahuber` and `atukey` a robust error statistics is used to estimate the standard deviation of the distances from the contour points without outliers from the approximating circle. The parameter ClippingFactor (a scaling factor for the standard deviation) controls the amount of damping outliers: The smaller the value chosen for ClippingFactor the more outliers are detected. The detection of outliers and the least squares fitting is repeated. The parameter Iterations specifies the number of iterations.

To reduce the computational load, the fitting of circles can be restricted to a subset of the contour points: If a value other than -1 is assigned to MaxNumPoints, only up to MaxNumPoints points - uniformly distributed over the contour - are used.

For circular arcs, the points on the circle closest to the start points and end points of the original contours are chosen as start and end points. The corresponding angles referring to the X-axis are returned in StartPhi and EndPhi, see also GenEllipseContourXld. Contours, for which the distance between their start points and their end points is less or equal MaxClosureDist are considered to be closed. Thus, they are approximated by circles instead of circular arcs. Due to artefacts in the pre-processing the start and end points of a contour might be faulty. Therefore, it is possible to exclude ClippingEndPoints at the beginning and at the end of a contour from the fitting of circles. However, they are still used for the determination of StartPhi and EndPhi.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Contours</strong></td>
<td>Input contours.</td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>Algorithm for the fitting of circles.</td>
</tr>
<tr>
<td><strong>MaxNumPoints</strong></td>
<td>Maximum number of contour points used for the computation (-1 for all points).</td>
</tr>
<tr>
<td><strong>MaxClosureDist</strong></td>
<td>Maximum distance between the end points of a contour to be considered as 'closed'.</td>
</tr>
</tbody>
</table>

```c
[out] VARIANT Row HXLDContX.FitCircleContourXld
[in] long ClippingEndPoints, [in] long Iterations,
[in] double ClippingFactor, [out] VARIANT Column, [out] VARIANT Radius,
[out] VARIANT StartPhi, [out] VARIANT EndPhi, [out] VARIANT PointOrder )
```

```c
void HOperatorSetX.FitCircleContourXld {[in] IHObjectX Contours,
[in] long ClippingEndPoints, [in] long Iterations,
[in] double ClippingFactor, [out] VARIANT Row, [out] VARIANT Column,
[out] VARIANT Radius, [out] VARIANT StartPhi, [out] VARIANT EndPhi,
[out] VARIANT PointOrder )
```
Approximation of XLD contours by ellipses or elliptic arcs.

Sub-pixel operators

FitEllipseContourXld
GenEllipseContourXld
GenContoursSkeletonXld

is

FitCircleContourXld
If necessary, an exception is raised.

contours. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>).

FitCircleContourXld returns TRUE if all parameter values are correct, and circles could be fitted to the input contours. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception is raised.

__Parallelization Information__

FitCircleContourXld is `local` and processed completely exclusively without parallelization.

**Possible Predecessor Functions**

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix, SmoothContoursXld

**Possible Successor Functions**

GenEllipseContourXld, DispCircle, GetPointsEllipse

See Also

FitEllipseContourXld, FitLineContourXld

Sub-pixel operators

<table>
<thead>
<tr>
<th>[out] VARIANT Row</th>
<th>HXLDContX.FitEllipseContourXld</th>
</tr>
</thead>
</table>


Approximation of XLD contours by ellipses or elliptic arcs.
FitEllipseContourXld approximates the XLD contours Contours by elliptic arcs or closed ellipses. It does not perform a segmentation of the input contours. Thus, one has to make sure that each contour corresponds to one and only one elliptic structure. The operator returns for each contour the center (Row, Column), the orientation of the main axis \( \text{Phi} \), the length of the larger half axis \( \text{Radius1} \), and the length of the smaller half axis \( \text{Radius2} \) of the underlying ellipse. In addition to that, the angle corresponding to the start point and the end point \( \text{StartPhi}, \text{EndPhi} \), and the point order along the boundary \( \text{PointOrder} \) is returned for elliptic arcs. These parameters are set to 0, \( 2\pi \), and ‘positive’ for closed ellipses. The algorithm used for the fitting of ellipses can be selected via Algorithm:

- **'fitzgibbon'**: This approach approximates the algebraic distance \( ax_1^2 + bx_1y_1 + cy_1^2 + dx_1 + ey_1 + f \) between the contour points \((x_i, y_i)\) and the resulting ellipse. The constraint \( 4ac - b^2 = 1 \) guarantees that the resulting polynomial characterizes an ellipse (instead of a hyperbola or a parabola).
- **'fhuber'**: Similar to 'fitzgibbon'. Here the contour points are weighted to decrease the impact of outliers based on the approach of Huber.
- **'ftukey'**: Similar to 'fitzgibbon'. Here the contour points are weighted to decrease the impact of outliers based on the approach of Tukey.
- **'voss'**: Each input contour is transformed in an affine standard position. Based on the moments of the transformed contour (that is of the enclosed image region) the standard circular segment is chosen whose standard position matches best with the standard position of the contour. The ellipse corresponding to the standard position of the selected circular segment is re-transformed based on the affine transformation which produced the standard position of the contour resulting in the ellipse matching the original contour. \( \text{VossTabSize} \) standard circular segments are used for this computation. To speed up the process the corresponding moments and other data is stored in a table which is created during the first call (with a specific value for \( \text{VossTabSize} \)) to \text{FitEllipseContourXld}.
- **'focpoints'**: Each point \( P \) on an ellipse satisfies the constraint that the sum of the distances to the focal points \( F_1,F_2 \) equals twice the length of the larger half axis \( a \). In this approach, the deviation \( PF_1 + PF_2 - 2a \) is minimized for all contour points by a least squares optimization.
- **'fphuber'**: Similar to 'focpoints'. Here a weighted least squares optimization is done to decrease the impact of outliers based on the approach of Huber.
- **'fptukey'**: Similar to 'focpoints'. Here a weighted least squares optimization is done to decrease the impact of outliers based on the approach of Tukey.

For '*'Huber' and '*'Tukey' a robust error statistics is used to estimate the standard deviation of the distances from the contour points without outliers from the approximating ellipse. The parameter \( \text{ClippingFactor} \) (a scaling factor for the standard deviation) controls the amount of damping outliers: The smaller the value chosen for \( \text{ClippingFactor} \) the more outliers are detected. The detection of outliers and the least squares fitting in the mode 'focpoints' is repeated. The parameter \( \text{Iterations} \) specifies the number of iterations.

To reduce the computational load, the fitting of ellipses can be restricted to a subset of the contour points: If a value other than -1 is assigned to \( \text{MaxNumPoints} \), only up to \( \text{MaxNumPoints} \) points - uniformly distributed over the contour - are used.

For elliptic arcs, the points on the ellipse closest to the start points and end points of the original contours are chosen as start and end points. The corresponding angles referring to the main axis of the ellipse are returned in \( \text{StartPhi} \) and \( \text{EndPhi} \), see also \text{GenEllipseContourXld}. Contours, for which the distance between their start points and end points is less or equal \( \text{MaxClosureDist} \) are considered to be closed. Thus, they are approximated by ellipses instead of elliptic arcs. Due to artefacts in the pre-processing the start and end points of a contour might be faulty. Therefore, it is possible to exclude \( \text{ClippingEndpoints} \) at the beginning and at the end of a contour from the fitting of ellipses. However, they are still used for the determination of \( \text{StartPhi} \) and \( \text{EndPhi} \).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contours (input iconic)</td>
<td>\text{xld_cont}(-array) \xrightarrow{HXLDContX/IHObjectX} Input contours.</td>
</tr>
<tr>
<td>Algorithm (input control)</td>
<td>\text{string} \xrightarrow{String} Algorithm for the fitting of ellipses.</td>
</tr>
<tr>
<td>Default Value: 'fitzgibbon'</td>
<td></td>
</tr>
<tr>
<td>Value List: Algorithm \in { 'fitzgibbon', 'fhuber', 'ftukey', 'voss', 'focpoints', 'fphuber', 'fptukey' }</td>
<td></td>
</tr>
</tbody>
</table>
14.3. FEATURES

MaxNumPoints (input control) ........................................................ integer
Maximum number of contour points used for the computation (-1 for all points).
Default Value : -1
Restriction : \( \text{MaxNumPoints} \geq 3 \)

MaxClosureDist (input control) .................................................. real
Maximum distance between the end points of a contour to be considered as 'closed'.
Default Value : 0
Restriction : \( \text{MaxClosureDist} \geq 0.0 \)

ClippingEndPoints (input control) .............................................. integer
Number of points at the beginning and at the end of the contours to be ignored for the fitting.
Default Value : 0
Restriction : \( \text{ClippingEndPoints} \geq 0 \)

VossTabSize (input control) ...................................................... integer
Number of circular segments used for the Voss approach.
Default Value : 200
Restriction : \( ((\text{VossTabSize} \geq 25) \land (\text{VossTabSize} \leq 5000)) \)

Iterations (input control) ........................................................ integer
Maximum number of iterations (unused for 'fitzgibbon' and 'voss').
Default Value : 3
Restriction : \( \text{Iterations} \geq 0 \)

ClippingFactor (input control) .................................................. real
Clipping factor for the elimination of outliers (typical: 1.0 for '*Huber' and 2.0 for '*Tukey').
Default Value : 2.0
Value List : \( \text{ClippingFactor} \in \{1.0, 1.5, 2.0, 2.5, 3.0\} \)
Restriction : \( \text{ClippingFactor} > 0 \)

Row (output control) ............................................................... ellipse.center.y(-array) \(
\text{VARIANT} (\text{real}) \)
Row coordinate of the center of the ellipse.

Column (output control) ........................................................... ellipse.center.x(-array) \(
\text{VARIANT} (\text{real}) \)
Column coordinate of the center of the ellipse.

Phi (output control) ............................................................... ellipse.angle.rad(-array) \(
\text{VARIANT} (\text{real}) \)
Orientation of the main axis [rad].

Radius1 (output control) ........................................................ ellipse.radius1(-array) \(
\text{VARIANT} (\text{real}) \)
Length of the larger half axis.

Radius2 (output control) ........................................................ ellipse.radius2(-array) \(
\text{VARIANT} (\text{real}) \)
Length of the smaller half axis.

StartPhi (output control) ....................................................... real(-array) \(
\text{VARIANT} (\text{real}) \)
Angle of the start point [rad].

EndPhi (output control) ......................................................... real(-array) \(
\text{VARIANT} (\text{real}) \)
Angle of the end point [rad].

PointOrder (output control) ..................................................... string(-array) \(
\text{VARIANT} (\text{string}) \)
Point order along the boundary.
Value List : \( \text{PointOrder} \in \{\text{positive}, \text{negative}\} \)

Example

\[
\text{read_image (Image, 'caltab')}
\]
\[
\text{find_caltab (Image, Caltab, 'caltabBig.descr', 3, 112, 5)}
\]
\[
\text{reduce_domain (Image, Caltab, ImageReduced)}
\]
\[
\text{edges_sub_pix (ImageReduced, Edges, 'lanser2', 0.5, 20, 40)}
\]
\[
\text{select_contours_xld (Edges, EdgesClosed, 'closed', 0, 2.0, 0, 0)}
\]
\[
\text{select_contours_xld (EdgesClosed, EdgesMarks, 'length', 20, 80, 0, 0)}
\]
\[
\text{fit_ellipse_contour_xld (EdgesMarks, 'fitzgibbon', -1, 2, 0, 200, 3, 2.0,}
\]
\[
\text{Row, Column, Phi, Radius1, Radius2, StartPhi,}
\]
\[
\text{EndPhi, PointOrder)}
\]
\[
\text{gen_ellipse_contour_xld (EllMarks, Row, Column, Phi, Radius1, Radius2,}
\]
\[
\text{StartPhi, EndPhi, PointOrder, 1.5)}
\]
\[
\text{length_xld(EllMarks,Length)}.
\]
**Chapter 14. XLD**

**FitEllipseContourXld** returns TRUE if all parameter values are correct, and ellipses could be fitted to the input contours. If the input is empty the behaviour can be set via `SetSystem('noObjectResult',<Result>)`. If necessary, an exception is raised.

**Parallelization Information**

`FitEllipseContourXld` is **local** and processed **completely exclusively** without parallelization.

**Possible Predecessor Functions**

`GenContoursSkeletonXld`, `LinesGauss`, `LinesFacet`, `EdgesSubPix`, `SmoothContoursXld`

**Possible Successor Functions**

`GenEllipseContourXld`, `DispEllipse`, `GetPointsEllipse`

**See Also**

`FitLineContourXld`

---

**Module**

Sub-pixel operators

---

```
[out] VARIANT RowBegin  HXLDContX.FitLineContourXld
```

```
```

**Approximation of XLD contours by line segments.**

`FitLineContourXld` approximates the XLD contours `Contours` by line segments. It does not perform a segmentation of the input contours. Thus, one has to make sure that each contour corresponds to one and only one line segment. The operator returns for each contour the start point (`RowBegin`, `ColBegin`), the end point (`RowEnd`, `ColEnd`), and the regression line to the contour given by the normal vector (`Nr`, `Nc`) of the line and its distance `Dist` from the origin, i.e., the line equation is given by $\frac{\text{RowBegin}}{\text{RowEnd}} + \frac{\text{ColBegin}}{\text{ColEnd}} - d = 0$.

The algorithm used for the fitting of lines can be selected via `Algorithm`:

- `'regression'` Standard 'least squares' line fitting.
- `'huber'` Weighted 'least squares' line fitting, where the impact of outliers is decreased based on the approach of Huber.
- `'tukey'` Weighted 'least squares' line fitting, where the impact of outliers is decreased based on the approach of Tukey.
- `'drop'` Weighted 'least squares' line fitting, where outliers are eliminated.
- `'gauss'` Weighted 'least squares' line fitting, where the impact of outliers is decreased based on the mean value and the standard deviation of the distances of all contour points from the approximating line.

For `'huber'`, `'tukey'`, and `'drop'` a robust error statistics is used to estimate the standard deviation of the distances from the contour points **without** outliers from the approximating line. The parameter `ClippingFactor` (a scaling factor for the standard deviation) controls the amount of damping outliers: The smaller the value chosen for `ClippingFactor` the more outliers are detected. The detection of outliers is repeated. The parameter `Iterations` specifies the number of iterations. In the modus `regression` this value is ignored.

To reduce the computational load, the fitting of lines can be restricted to a subset of the contour points: If a value other than -1 is assigned to `MaxNumPoints`, only up to `MaxNumPoints` points - uniformly distributed over the contour - are used.

The start point and the end point of a line segment is determined by projecting the first and the last point of the corresponding contour to the approximating line. Due to artefacts in the pre-processing the start and end points of a contour might be faulty. Therefore, it is possible to exclude `ClippingEndPoints` at the beginning and at the
end of a contour from the line fitting. However, they are still used for the determination of the start point and the end point of the line segment.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Contours</strong> (input iconic)</td>
<td>xld_cont(-array) ~ HXLDContX / IHOObjectX</td>
</tr>
<tr>
<td><strong>Algorithm</strong> (input control)</td>
<td>string ~ String</td>
</tr>
<tr>
<td><strong>MaxNumPoints</strong> (input control)</td>
<td>integer ~ long</td>
</tr>
<tr>
<td><strong>ClippingEndPoints</strong> (input control)</td>
<td>integer ~ long</td>
</tr>
<tr>
<td><strong>Iterations</strong> (input control)</td>
<td>integer ~ long</td>
</tr>
<tr>
<td><strong>ClippingFactor</strong> (input control)</td>
<td>real ~ double</td>
</tr>
<tr>
<td><strong>RowBegin</strong> (output control)</td>
<td>line.begin.y(-array) ~ VARIANT(real)</td>
</tr>
<tr>
<td><strong>ColBegin</strong> (output control)</td>
<td>line.begin.x(-array) ~ VARIANT(real)</td>
</tr>
<tr>
<td><strong>RowEnd</strong> (output control)</td>
<td>line.end.y(-array) ~ VARIANT(real)</td>
</tr>
<tr>
<td><strong>ColEnd</strong> (output control)</td>
<td>line.end.x(-array) ~ VARIANT(real)</td>
</tr>
<tr>
<td><strong>Nr</strong> (output control)</td>
<td>number(-array) ~ VARIANT(real)</td>
</tr>
<tr>
<td><strong>Nc</strong> (output control)</td>
<td>number(-array) ~ VARIANT(real)</td>
</tr>
<tr>
<td><strong>Dist</strong> (output control)</td>
<td>number(-array) ~ VARIANT(real)</td>
</tr>
</tbody>
</table>

**Example**

draw_ellipse(WindowHandle, Row, Column, Phi, Radius1, Radius2)
read_image (Image, 'mreut')
edges_sub_pix (Image, Edges, 'lanser2', 0.5, 20, 40)
gen_polygons_xld (Edges, Polygons, 'ramer', 2)
split_contours_xld (Polygons, Contours, 'polygon', 1, 5)
fit_line_contour_xld (Contours, 'regression', -1, 0, 5, 2, RowBegin, ColBegin, RowEnd, ColEnd, Nr, Nc).

**Result**

FitLineContourXld returns TRUE if all parameter values are correct, and line segments could be fitted to the input contours. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception is raised.
FitLineContourXld is reentrant and processed without parallelization.

Possible Predecessor Functions
GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix, SmoothContoursXld

Possible Successor Functions
DispLine, SelectLines, LineOrientation

See Also
RegressContoursXld, GetRegressParamsXld

Sub-pixel operators

GetContourAngleXld calculates for each point of the XLD contour Contour the direction of its tangent. Two modes for the output values can be chosen: By passing 'abs' for AngleMode, the resulting angles are returned relative to the horizontal axis as angles between 0 and 2\( \pi \) (counter-clockwise). By passing 'rel', the absolute value of the difference of angles to the previous contour point is returned. In this case the range of values is between 0 and \( \pi \).

There are three different ways of calculating the tangent direction (CalcMode) of the contour point using the contour points in the interval from \( i - \text{Lookaround} \) to \( i + \text{Lookaround} \). By using 'range', the angle of the line segment between the first and last point of the interval is used. For 'mean', the average of all angles between consecutive points of the contour is used. Finally, for 'regress', the direction of the regression line (the least squares fit of a line to the contour points in the interval) is used. Lookaround is a measure of how strongly the contour is smoothed. The angles are returned in Angles in radians.

Parameter

- **Contour** (input iconic) ................................. xld_cont \( \rightarrow \) HXLDContX / IObjectX
  - Input contour.
- **AngleMode** (input control) .......................... string \( \rightarrow \) String
  - Return type of the angles.
  - Default Value: 'abs'
  - Value List: AngleMode \( \in \) \{'abs', 'rel'\}
- **CalcMode** (input control) .............................. string \( \rightarrow \) String
  - Method for computing the angles.
  - Default Value: 'range'
  - Value List: CalcMode \( \in \) \{'range', 'mean', 'regress'\}
- **Lookaround** (input control) ......................... integer \( \rightarrow \) long
  - Number of points to take into account.
  - Default Value: 3
  - Restriction: (Lookaround > 0)
- **Angles** (output control) ............................... real \( \rightarrow \) VARIANT(real)
  - Direction of the tangent to the contour points.

GetContourAngleXld is reentrant and processed without parallelization.

Possible Predecessor Functions
GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

See Also
GetContourXld, GetContourAttribXld
Return point attribute values of an XLD contour.

GetContourAttribXld returns the values of the attribute Name of the XLD contour Contour in Attrib. Attributes are additional values defined for each contour point, e.g., the direction perpendicular to the contour (‘angle’). Operators which define this kind of attributes, e.g., LinesGauss and EdgesSubPix, contain a description of the name and semantics of the defined values. QueryContourAttributesXld can be used to query which attributes are defined for a particular contour.

Parameter

- **Contour** (input iconic) ............................... xld_cont ~ HXLDContX / IHObjectX
  Input XLD contour.
- **Name** (input control) ...................................... string ~ String
  Name of the attribute.
- **Attrib** (output control) ..................................... real ~ VARIANT( real )
  Attribute values.

Parallelization Information

GetContourAttribXld is reentrant and processed without parallelization.

Possible Predecessor Functions

LinesGauss, LinesFacet, EdgesSubPix

See Also

QueryContourAttributesXld, GetContourGlobalAttribXld, QueryContourGlobalAttributesXld

Sub-pixel operators

Return global attributes values of an XLD contour.

GetContourGlobalAttribXld returns the values of the global attribute Name of the XLD contour Contour in Attrib. Global attributes are additional values defined for each contour, e.g., the normal vector of the regression line of a contour (‘regr_norm_row’ and ‘regr_norm_col’). Operators which define this kind of attributes contain a description of the name and semantics of the defined values. QueryContourGlobalAttributesXld can be used to query which attributes are defined for a particular contour.

Parameter

- **Contour** (input iconic) ............................... xld_cont ~ HXLDContX / IHObjectX
  Input XLD contour.
- **Name** (input control) ...................................... string(-array) ~ VARIANT( string )
  Name of the attribute.
- **Attrib** (output control) ..................................... real ~ VARIANT( real )
  Attribute values.
GetContourAttribXld is reentrant and processed without parallelization.

### Possible Predecessor Functions

- LinesGauss
- LinesFacet
- EdgesSubPix

### See Also

- QueryContourAttribsXld
- GetContourAttribXld
- QueryContourAttribsXld

**Module**

Sub-pixel operators

```c

```

Return XLD contour parameters. GetRegressParamsXld returns the following parameters for all XLD contours given in Contours:

- the number of contour points Length,
- the coordinates Nx and Ny of the normal vector of the regression line (i.e., least-squares approximating line),
- the distance Dist of the regression line to the origin
- the sub-pixel precise coordinates Fpx and Fpy of the perpendicular projection of the start point of the contour onto the regression line,
- the sub-pixel precise coordinates Lpx and Lpy of the perpendicular projection of the end point of the contour onto the regression line,
- the mean of the Euclidian distance of the contour points from the regression line,
- the standard deviation of these distances.

### Attention

Before the contour parameters can be returned by GetRegressParamsXld, the parameters of the regression line to the contour must be calculated by calling RegressContoursXld.

### Parameter

- Contours (input iconic) xld_cont ~ HXLDContX / IHObjectX
- Length (output control) integer ~ VARIANT ( integer )
- Nx (output control) point.x ~ VARIANT ( real )
- Ny (output control) point.y ~ VARIANT ( real )
- Dist (output control) number ~ VARIANT ( real )
- Fpx (output control) point.x ~ VARIANT ( real )
- Fpy (output control) point.y ~ VARIANT ( real )
14.3. FEATURES

- **Lpx** (output control) ................................. point.x  \(\sim\) VARIANT (real)
  X-coordinate of the projection of the end point of the contour onto the regression line.

- **Lpy** (output control) ................................. point.y  \(\sim\) VARIANT (real)
  Y-coordinate of the projection of the end point of the contour onto the regression line.

- **Mean** (output control) .............................. real  \(\sim\) VARIANT (real)
  Mean distance of the contour points from the regression line.

- **Deviation** (output control) ........................ real  \(\sim\) VARIANT (real)
  Standard deviation of the distances from the regression line.

**Parallelization Information**

GetRegressParamsXld is reentrant and processed without parallelization.

**Possible Predecessor Functions**

RegressContoursXld

**Possible Successor Functions**

DispLine, SelectLines, LineOrientation

See Also

FitLineContourXld, GetContourGlobalAttribXld, QueryContourGlobalAttribsXld, GetContourXld, GetContourAttribXld, GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

Sub-pixel operators

```c
[out] double QualityMin HXLDParaX.InfoParallelsXld ([in] HImageX Image,
[out] double QualityMax, [out] long GrayMin, [out] long GrayMax,
[out] double StandardMin, [out] double StandardMax )

void HOperatorSetX.InfoParallelsXld ([in] IObjectX Parallels,
[in] IObjectX Image, [out] double QualityMin, [out] double QualityMax,
[out] long GrayMin, [out] long GrayMax, [out] double StandardMin,
[out] double StandardMax )
```

Return information about the gray values of the area enclosed by XLD parallels.

InfoParallelsXld calculates various gray value features of the area enclosed by the XLD parallels. The input image Image is used to get the gray values needed for this. The algorithm used in this operator is very similar to the one used in ModParallelsXld. The operator returns ranges for the quality factor (QualityMin and QualityMax), the mean gray value (GrayMin and GrayMax), and the standard deviation with respect to the mean gray value (StandardMin and StandardMax).

This operator serves to determine appropriate thresholds for ModParallelsXld.

**Parameter**

- **Parallels** (input iconic) ............................ xld_para  \(\sim\) HXLDParaX / IObjectX
  Input XLD Parallels.

- **Image** (input iconic) .............................. image  \(\sim\) HImageX / IObjectX
  Corresponding gray value image.

- **QualityMin** (output control) ........................ real  \(\sim\) double
  Minimum quality factor.

- **QualityMax** (output control) ........................ real  \(\sim\) double
  Maximum quality factor.

- **GrayMin** (output control) ............................ integer  \(\sim\) long
  Minimum mean gray value.

- **GrayMax** (output control) ............................ integer  \(\sim\) long
  Maximum mean gray value.

- **StandardMin** (output control) ........................ real  \(\sim\) double
  Minimum standard deviation.

- **StandardMax** (output control) ........................ real  \(\sim\) double
  Maximum standard deviation.
Parallelization Information

InfoParallelsXld is reentrant and processed without parallelization.

Possible Predecessor Functions

GenParallelsXld

Possible Successor Functions

ModParallelsXld

See Also

Intensity, MinMaxGray

Module

Sub-pixel operators

[out] VARIANT Length HXLDX.LengthXld ( )

void HOperatorSetX.LengthXld ([in] IObjectX XLD,
[out] VARIANT Length )

Length of contours or polygons.

LengthXld calculates the length of the contours or polygons XLD. The length is calculated as the sum of the
euclidian distances of successive points on the contour or polygon. If more than one contour or polygon is passed
the results are stored as tuples in which the index of a value corresponds to the index of the respective contour or
polygon in XLD.

Parameter

▷ XLD (input iconic) ................................. xld(-array) ~> HXLDX / IObjectX
Contours or polygons to be examined.
▷ Length (output control) .......................... real(-array) ~> VARIANT ( real )
Length of the contour or polygon.

Restriction : (Length ≥ 0)

Result

LengthXld returns TRUE if the input is not empty. If the input is empty the behaviour can be set via
SetSystem(:'noObjectResult',<Result>:). If necessary, an exception is raised.

Parallelization Information

LengthXld is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

GenContoursSkeletonXld, SmoothContoursXld, GenPolygonsXld

See Also

AreaCenterXld, MomentsAnyXld, MomentsXld, ContLength

Module

Sub-pixel operators

[out] HXLDContX LocalMaxContours HXLDContX.LocalMaxContoursXld

([in] HImageX Image, [in] VARIANT MinPercent, [in] long MinDiff,
[in] long Distance )

void HOperatorSetX.LocalMaxContoursXld ([in] IObjectX Contours,
[in] IObjectX Image, [out] HUntypedObjectX LocalMaxContours,

Select XLD contours with a local maximum of gray values.

LocalMaxContoursXld selects XLD contours from the contours passed in Contours, which have a local
maximum in gray values across the direction of the contour. In order to be selected, at least MinPercent of
the contour points must have a local maximum perpendicular to the direction of the contour. The contours’ direction
is determined by fitting a regression line through five neighboring points of the contour. In order to decide whether
there is a local maximum for a contour point, a gray value profile that is \( \text{Distance} \) points wide and perpendicular to the contour is computed on both sides of the contour. If the gray value at the countour point is at least \( \text{MinDiff} \) larger than the gray value at at least one point on each side of the profile, the contour point is labeled as a local maximum. The selected contours are returned in \( \text{LocalMaxContours} \).

Parameter

- **Contours** (input iconic) \( \sim \text{HXLDContX} / \text{IHObjectX} \)
  - XLD contours to be examined.
- **Image** (input iconic) \( \sim \text{HImageX} / \text{IHObjectX} \)
  - Corresponding gray value image.
- **LocalMaxContours** (output iconic) \( \sim \text{HXLDContX} / \text{HUntypedObjectX} \)
  - Selected contours.
- **MinPercent** (input control) \( \sim \text{VARIANT} (\text{integer, real}) \)
  - Minimum percentage of maximum points.
  - Default Value: 70
  - Value Suggestions: \( \text{MinPercent} \in \{60, 70, 75, 80, 85, 90, 95\} \)
  - Restriction: \( (0.0 \leq \text{MinPercent}) \lor (\text{MinPercent} \leq 100.0) \)
- **MinDiff** (input control) \( \sim \text{long} \)
  - Minimum amount by which the gray value at the maximum must be larger than in the profile.
  - Default Value: 15
  - Value Suggestions: \( \text{MinDiff} \in \{5, 8, 10, 12, 15, 20\} \)
  - Restriction: \( (0 \leq \text{MinDiff}) \land (\text{MinDiff} \leq 255) \)
- **Distance** (input control) \( \sim \text{long} \)
  - Maximum width of profile used to check for maxima.
  - Default Value: 4
  - Value Suggestions: \( \text{Distance} \in \{2, 3, 4, 5, 6\} \)
  - Restriction: \( (\text{Distance} \geq 1) \)

Parallelization Information

\( \text{LocalMaxContoursXld} \) is \textit{reentrant} and processed \textit{without} parallelization.

Possible Predecessor Functions

- GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

Possible Successor Functions

- GenPolygonsXld
- SmoothContoursXld

See Also

Sub-pixel operators

---

```c
[out] void HOperatorSetX.MaxParallelsXld ([in] \text{IHObjectX} \text{ExtParallels},
[out] \text{HUntypedObjectX} \text{MaxPolygons})
```

Join modified XLD parallels lying on the same polygon.

\( \text{MaxParallelsXld} \) joins all modified XLD parallels in \( \text{ExtParallels} \) into a polygon if they lie on the same original polygon segment. This means that polygons exhibiting parallelism and enclosing homogeneous areas in several places are joined into one long polygon (from the first parallel line to the last parallel line). The resulting polygons are returned in \( \text{MaxPolygons} \).

Parameter

- **ExtParallels** (input iconic) \( \sim \text{HXLDExtParaX} / \text{IHObjectX} \)
  - Extended XLD parallels.
- **MaxPolygons** (output iconic) \( \sim \text{HXLDPolyX} / \text{HUntypedObjectX} \)
  - Maximally extended parallels.
Parallelization Information
MaxParallelsXld is reentrant and processed without parallelization.

Possible Predecessor Functions
ModParallelsXld

Possible Successor Functions
GetPolygonXld, GetLinesXld

Module

Sub-pixel operators

Arbitrary geometric moments of contours or polygons.

MomentsAnyXld calculates arbitrary moments M of the region enclosed by the contours or polygons XLD. The moments are computed by applying Green’s theorem using only the points on the contour or polygon, i.e., no region is generated explicitly for the purpose of calculating the features. It is assumed that the contours or polygons are closed. If this is not the case MomentsAnyXld will add one point to artificially produce a closed shape. The computed moments are normalized depending on the desired mode Mode:

- ‘unnormalized’: No normalization. Let R be the enclosed image region. Then the computed moment \( M_{p,q} \) is equivalent to

\[
M_{p,q} = \int \int R p^p q^q dx dy
\]

- ‘normalized’: Normalization by the area \( A = \text{Area} \) of the enclosed image region:

\[
M_{p,q} = \frac{1}{A} \int \int R p^p q^q dx dy
\]

- ‘central’ Normalization by the area \( A = \text{Area} \) of the enclosed image region and a shift of the region by its centroid \( [\bar{r}, \bar{c}] = [\text{CenterRow}, \text{CenterCol}] \):

\[
M_{p,q} = \frac{1}{A} \int \int (r - \bar{r})^p (c - \bar{c})^q dx dy
\]

For the normalization of the moments three specific moments are used: The area \( \text{Area} \) of the enclosed image region and the coordinates \( \text{CenterRow}, \text{CenterCol} \) of its centroid, see AreaCenterXld. In addition to that MomentsAnyXld expects information about the point order of the input contours/polygons in \( \text{PointOrder} \), see AreaCenterXld again. If more than one contour or polygon is passed, \( M \) contains all desired moments of the first contour/polygon followed by all the moments of the second contour/polygon and so forth.

Parameter

\[ XLD \] (input iconic) .......................................................... xld(-array) \( \xrightarrow{\text{HXLDX / IObjectX}} \)
Contours or polygons to be examined.

\[ \text{Mode} \] (input control) .................................................. string \( \xrightarrow{\text{String}} \)

computation mode.

Default Value: ‘unnormalized’

Value Suggestions: \( \text{Mode} \in \{ 'unnormalized', 'normalized', 'central' \} \)

\[ \text{PointOrder} \] (input control) ........................................... string(-array) \( \xrightarrow{\text{VARIANT ( string )}} \)
point order along the boundary.

Default Value: ‘positive’

Value Suggestions: \( \text{PointOrder} \in \{ 'positive', 'negative' \} \)
14.3. FEATURES

- **Area** (input control) ........................................ real(-array) \(\rightarrow\) VARIANT (real)
  Area enclosed by the contour or polygon.
- **CenterRow** (input control) ............................... point.y(-array) \(\rightarrow\) VARIANT (real)
  Row coordinate of the centroid.
- **CenterCol** (input control) ............................... point.x(-array) \(\rightarrow\) VARIANT (real)
  Column coordinate of the centroid.
- **P** (input control) ........................................ point.x(-array) \(\rightarrow\) VARIANT (integer)
  first index of the desired moments \(M[P,Q]\).
  Default Value: 1
- **Q** (input control) ........................................ point.x(-array) \(\rightarrow\) VARIANT (integer)
  second index of the desired moments \(M[P,Q]\).
  Default Value: 1
- **M** (output control) ........................................ real(-array) \(\rightarrow\) VARIANT (real)
  the computed moments.

MomentsAnyXld returns TRUE if the input is not empty. If the input is empty the behaviour can be set via SetSystem(:,’noObjectResult’,<Result>:). If necessary, an exception is raised.

---

**Parallelization Information**
MomentsAnyXld is local and processed completely exclusively without parallelization.

**Possible Predecessor Functions**
AreaCenterXld, GenContoursSkeletonXld, SmoothContoursXld, GenPolygonsXld

**See Also**
MomentsXld, AreaCenterXld, MomentsRegion2Nd, AreaCenter

**Alternatives**
MomentsXld

**Module**
Sub-pixel operators

```halcon
[out] VARIANT M11 HXLDX.MomentsXld ([out] VARIANT M20, [out] VARIANT M02 )
void HOperatorSetX.MomentsXld ([in] IHObjectX XLD, [out] VARIANT M11, [out] VARIANT M20, [out] VARIANT M02 )
```

**Parameter**

- **XLD** (input iconic) ........................................ xld(-array) \(\sim\) HXLDX / IHObjectX
  Contours or polygons to be examined.
- **M11** (output control) ...................................... real(-array) \(\sim\) VARIANT (real)
  Mixed second order moment.
- **M20** (output control) ...................................... real(-array) \(\sim\) VARIANT (real)
  Second order moment along the row axis.
- **M02** (output control) ...................................... real(-array) \(\sim\) VARIANT (real)
  Second order moment along the column axis.

MomentsXld returns TRUE if the input is not empty. If the input is empty the behaviour can be set via SetSystem(:,’noObjectResult’,<Result>:). If necessary, an exception is raised.

---

**Result**

HALCON 6.0
MomentsXld is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

GenContoursSkeletonXld, SmoothContoursXld, GenPolygonsXld
GenContoursXld, SmoothContoursXld, GenPolygonsXld

See Also

MomentsAnyXld, AreaCenterXld, MomentsRegion2Nd, AreaCenter

Alternatives

MomentsAnyXld

Module

Sub-pixel operators

---

[ out ] VARIANT Attributes HXLDContX.QueryContourAttribsXld ()

void HOperatorSetX.QueryContourAttribsXld ([in] IHObjectX Contour, 
[ out ] VARIANT Attributes )

Return the names of the defined attributes of an XLD contour.

QueryContourAttribsXld returns the names of the defined attributes of an XLD contour Contour in Attributes. Attributes are additional values defined for each contour point, e.g. the direction perpendicular to the contour (‘angle’). Operators which define this kind of attributes contain a description of the name and semantics of the defined values. GetContourAttribXld can be used to access the values of a particular attribute.

Parameter

▷ Contour (input iconic) ................................. xld_cont \rightarrow HXLDContX / IHObjectX
Input contour.

▷ Attributes (output control) ............................ string \rightarrow VARIANT( string )
List of the defined contour attributes.

---

Parallelization Information

QueryContourAttribsXld is reentrant and processed without parallelization.

Possible Predecessor Functions

LinesGauss, LinesFacet, EdgesSubPix

---

GetContourAttribXld, GetContourGlobalAttribXld, QueryContourGlobalAttribsXld

Module

Sub-pixel operators

---

[ out ] VARIANT Attributes HXLDContX.QueryContourGlobalAttribsXld ()

void HOperatorSetX.QueryContourGlobalAttribsXld ([in] IHObjectX Contour, 
[ out ] VARIANT Attributes )

Return the names of the defined global attributes of an XLD contour.

QueryContourGlobalAttribsXld returns the names of the globally defined attributes of an XLD contour Contour in Attributes. Global attributes are additional values defined for each contour, e.g. the normal vector of the regression line of a contour (‘regr_norm_row’ and ‘regr_norm_col’). Operators which define this kind of attributes contain a description of the name and semantics of the defined values. GetContourGlobalAttribXld can be used to access the value of a particular attribute.

Parameter

▷ Contour (input iconic) ................................. xld_cont \rightarrow HXLDContX / IHObjectX
Input contour.

▷ Attributes (output control) ............................ string \rightarrow VARIANT( string )
List of the defined global contour attributes.
14.3. FEATURES

Parallelization Information

QueryContourGlobalAttribsXld is reentrant and processed without parallelization.

Possible Predecessor Functions

LinesGauss, LinesFacet, EdgesSubPix

See Also

GetContourGlobalAttribXld, GetContourAttribXld, QueryContourAttribsXld

Sub-pixel operators

Select XLD contours according to several features.

SelectContoursXld selects XLD contours from the input contours Contours according to the following features (parameter Feature):

- **'length'** all contours whose maximum extent (as measured by their eight extremal points in row and column direction, according to Haralick und Shapiro: Computer and Robot Vision, Addison-Wesley 1992, chapter 3.2) is smaller than Min1 or larger than Max1 are not returned (Min2 and Max2 have no influence here).
- **'direction'** only contours for which the direction of the regression line is between Min1 and Max1 (in radians, counter-clockwise) are returned. Min1 and Max1 are mapped to the range of [0, 2π]. (Min2 and Max2 have no influence here).
- **'curvature'** only contours for which the mean distance from the regression line lies between Min1 and Max1, and for which the standard deviation of the distances is between Min2 and Max2 are returned.
- **'closed'** only contours for which the distance between their start point and their end point is less or equal Max2 pixels are returned.

If Min1 = Max1 = 0 or Min2 = Max2 = 0 is used for the selection according to curvature, the respective feature has no influence on the selection.

Attention

Before contour can be filtered by SelectContoursXld according to 'direction' or 'curvature', the parameters of the regression lines to the contours must be calculated by calling RegressContoursXld.

Parameter

- **Contours** (input iconic) ................................. xld_cont ~ HXLDContX / IHObjectX
  Input XLD contours.
- **SelectedContours** (output iconic) ........................ xld_cont ~ HXLDContX / HUntypedObjectX
  Output XLD contours.
- **Feature** (input control) ................................. string ~ String
  Feature to select contours with.
  Default Value: 'length'
  Value List: Feature ∈ {'length', 'direction', 'curvature', 'closed'}
- **Min1** (input control) ................................. real ~ double
  Lower threshold.
  Default Value: 0.5
- **Max1** (input control) ................................. real ~ double
  Upper threshold.
  Default Value: 200.0
- **Min2** (input control) ................................. real ~ double
  Lower threshold.
  Default Value: -0.5
Add noise to XLD contours.

AddNoiseWhiteContourXld adds noise to the input XLD contours Contours and returns the noisy contours in NoisyContours. For each point along the original contours the local regression line (i.e. a least-squares approximating line) based on NumRegrPoints neighboring points is determined. Then the point is shifted perpendicular to this line. The length of the shifts corresponds to white noise, equally distributed in the interval \([-Amp,Amp]\) generated by using the C function “drand48” with an initial time dependent seed.

**Parameter**

- **Contours** (input iconic) .............................................. xld_cont(-array) \(\approx\) HXLDContX / IHObjectX
  Original contours.

- **NoisyContours** (output iconic) .............................. xld_cont(-array) \(\approx\) HXLDContX / HUntypedObjectX
  Noisy contours.

- **NumRegrPoints** (input control) ................................. integer \(\approx\) long
  Number of points used to calculate the regression line.
  
  **Default Value:** 5
  
  **Value Suggestions:** NumRegrPoints \(\in\) \{3, 5, 7, 9\}
  
  **Restriction:** (NumRegrPoints \& odd)

- **Amp** (input control) .............................................. real \(\approx\) double
  Maximum amplitude of the added noise (equally distributed in \([-Amp,Amp]\)).
  
  **Default Value:** 1.0
  
  **Value Suggestions:** Amp \(\in\) \{0.25, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 10.0\}
  
  **Restriction:** (Amp > 0)

**Example**

draw_ellipse(WindowHandle, Row, Column, Phi, Radius1, Radius2)
gen_ellipse_contour_xld(Ellipse, Row, Column, Phi, Radius1, Radius2, 0, 6.28319, ‘positive’)
add_noise_white_contour_xld(Ellipse, NoisyEllipse, 5, 0.5)
fit_ellipse_contour_xld (NoisyEllipse, ’fitzgibbon’, -1, 2, 0, 200, 3, 2.0,
14.4. TRANSFORMATION


---

Result

AddNoiseWhiteContourXld returns TRUE if all parameter values are correct. If the input is empty the behaviour can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception is raised.

---

Parallelization Information

AddNoiseWhiteContourXld is reentrant, local, and processed without parallelization.

---

Possible Predecessor Functions

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix, GenEllipseContourXld

---

Possible Successor Functions

SmoothContoursXld

See Also

SmoothContoursXld, AddNoiseWhite

---

Module

Sub-pixel operators

---

Apply an arbitrary affine transformation to an XLD contour.

AffineTransContourXld applies an arbitrary affine transformation, i.e., scaling, rotation, translation, and skewing, to an XLD contour. The affine map is described by the transformation matrix given in HomMat2D, which is built up by using HomMat2dIdentity, HomMat2dScale, HomMat2dRotate, and HomMat2dTranslate. The components of the homogeneous transformation matrix are interpreted as follows: the row coordinate of the image corresponds to the x coordinate of the matrix, while the column coordinate of the image corresponds to the y coordinate of the matrix. This is necessary to obtain a right-handed coordinate system, which is assumed for the homogeneous transformation matrices, also for the image. In particular, by doing so rotations are performed in the correct direction. Note that the (x,y) order of the matrices quite naturally corresponds to the usual (row,column) order for coordinates in the image.

---

Attention

The components of the homogeneous transformation matrix are interpreted as follows: the row coordinate of the image corresponds to the x coordinate of the matrix, while the column coordinate of the image corresponds to the y coordinate of the matrix. This is necessary to obtain a right-handed coordinate system, which is assumed for the homogeneous transformation matrices, also for the image. In particular, by doing so rotations are performed in the correct direction. Note that the (x,y) order of the matrices quite naturally corresponds to the usual (row,column) order for coordinates in the image.

---

Parameter

- **Contours** (input iconic) .......................... xld_cont(-array) ˜ HXLDContX / IHObjectX
  Input XLD contours.
- **ContoursAffinTrans** (output iconic) ............ xld_cont(-array) ˜ HXLDContX / HUntypedObjectX
  Transformed XLD contours.
- **HomMat2D** (input control) .......................... affine2d ˜ VARIANT / HHomMat2dX (real)
  Input transformation matrix.

Parameter Number : 6

---

Result

AffineTransContourXld returns TRUE if all parameter values are correct. If necessary, an exception is raised.

---

HALCON 6.0
AffineTransContourXld is reentrant and automatically parallelized (on tuple level).

Possible Predecessor Functions

HomMat2dIdentity, HomMat2dTranslate, HomMat2dRotate, HomMat2dScale

See Also

AffineTransImage, AffineTransRegion

Module

Sub-pixel operators

---

Possible Predecessor Functions

HomMat2dIdentity, HomMat2dTranslate, HomMat2dRotate, HomMat2dScale

See Also

AffineTransImage, AffineTransRegion, AffineTransContourXld

---

Parallelization Information

AffineTransContourXld is reentrant and automatically parallelized (on tuple level).

---

Parallelization Information

AffineTransContourXld is reentrant and automatically parallelized (on tuple level).

---

Sub-pixel operators

---

**Apply an arbitrary affine transformation to an XLD polygon.**

AffineTransPolygonXld applies an arbitrary affine transformation, i.e., scaling, rotation, translation, and skewing, to an XLD polygon. The affine map is described by the transformation matrix given in HomMat2D, which is built up by using HomMat2dIdentity, HomMat2dScale, HomMat2dRotate, and HomMat2dTranslate.

---

**Attention**

The components of the homogeneous transformation matrix are interpreted as follows: the row coordinate of the image corresponds to the x coordinate of the matrix, while the column coordinate of the image corresponds to the y coordinate of the matrix. This is necessary to obtain a right-handed coordinate system, which is assumed for the homogeneous transformation matrices, also for the image. In particular, by doing so rotations are performed in the correct direction. Note that the (x,y) order of the matrices quite naturally corresponds to the usual (row,column) order for coordinates in the image.

The XLD contours that are possibly referenced by Polygons are neither transformed nor stored with the output polygons, since this is generally impossible without creating inconsistencies for the attributes of the XLD contours. Hence, operators that access the contours associated with a polygon, e.g., SplitContoursXld will not work correctly.

---

**Parameter**

- **Polygons** (input iconic) ................. xld_poly(-array)  \(\rightarrow\) HXLDPolyX / IObjectX
  Input XLD polygons.
- **PolygonsAffinTrans** (output iconic) ........... xld_poly(-array)  \(\rightarrow\) HXLDPolyX / HUntypedObjectX
  Transformed XLD polygons.
- **HomMat2D** (input control) .................. affine2d  \(\rightarrow\) VARIANT / HHomMat2dX (real)
  Input transformation matrix.

**Parameter Number : 6**

---

**Result**

AffineTransPolygonXld returns TRUE if all parameter values are correct. If necessary, an exception is raised.

---

Parallelization Information

AffineTransPolygonXld is reentrant and automatically parallelized (on tuple level).

---

Sub-pixel operators
Clip an XLD contour.

ClipContoursXld clips all XLD contours given in Contours, i.e., only contour points contained in the rectangle given by Row1, Column1, Row2, and Column2 are returned on output. If necessary, contours are split, and several new contours are produced. The resulting contours are returned in ClippedContours.

Parameter

- **Contours** (input iconic) \(\xrightarrow{\text{xld_cont}} \text{HXLDContX / IHObjectX} \) Contours to be clipped.
- **ClippedContours** (output iconic) \(\xrightarrow{\text{xld_cont}-\text{array}} \text{HXLDContX / HUntypedObjectX} \) Clipped contours.
- **Row1** (input control) \(\xrightarrow{\text{rectangle.origin.y}} \text{long} \) Row coordinate of the upper left corner of the clip rectangle.
  - Default Value: 0
  - Value Suggestions: \(\text{Row1} \in \{0, 500, 1000, 1500, 2000\} \)
- **Column1** (input control) \(\xrightarrow{\text{rectangle.origin.x}} \text{long} \) Column coordinate of the upper left corner of the clip rectangle.
  - Default Value: 0
  - Value Suggestions: \(\text{Column1} \in \{0, 500, 1000, 1500, 2000\} \)
- **Row2** (input control) \(\xrightarrow{\text{rectangle.corner.y}} \text{long} \) Row coordinate of the lower right corner of the clip rectangle.
  - Default Value: 512
  - Value Suggestions: \(\text{Row2} \in \{512, 1024, 1536, 2048\} \)
- **Column2** (input control) \(\xrightarrow{\text{rectangle.corner.x}} \text{long} \) Column coordinate of the lower right corner of the clip rectangle.
  - Default Value: 512
  - Value Suggestions: \(\text{Column2} \in \{512, 1024, 1536, 2048\} \)

**Parallelization Information**

ClipContoursXld is **reentrant** and processed **without** parallelization.

**Possible Predecessor Functions**

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

**Possible Successor Functions**

GenPolygonsXld

**See Also**

ClipRegion, CropPart

**Module**

Sub-pixel operators
**Combine road hypotheses from two resolution levels.**

CombineRoadsXld combines road hypotheses obtained from two different resolution levels. The algorithm selects only those hypotheses which mutually support each other in both resolution levels. The parameters `EdgePolygons`, `ModParallels` and `ExtParallels` correspond to the road hypotheses from the highest resolution level. The parameter `CenterLines` is the result of road extraction in a lower resolution level. Those of the polygons `EdgePolygons` are returned for which evidence of being roadsides is found in both resolution levels. The parameters `MaxAngleParallel` and `MaxAngleColinear` determine the angle, that may be formed by two parallel collinear line segments, respectively. The parameters `MaxDistanceParallel` and `MaxDistanceColinear` determine the maximum allowed distance of two parallel or collinear line segments, respectively. The combination is done using a number of rules.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value Suggestions</th>
<th>Default Value</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>EdgePolygons</code> (input iconic)</td>
<td><code>xld_poly ~ HXLDPolyX / IHObjectX</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ModParallels</code> (input iconic)</td>
<td><code>xld_mod_para ~ HXLDModParaX / IHObjectX</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>ExtParallels</code> (input iconic)</td>
<td><code>xld_ext_para ~ HXLDExtParaX / IHObjectX</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>CenterLines</code> (input iconic)</td>
<td><code>xld_poly ~ HXLDPolyX / IHObjectX</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>RoadSides</code> (output iconic)</td>
<td><code>xld_poly ~ HXLDPolyX / HUntypedObjectX</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>MaxAngleParallel</code> (input control)</td>
<td><code>angle.rad ~ VARIANT ( real, integer )</code></td>
<td><code>0.523598775598</code></td>
<td><code>0.0 ≤ MaxAngleParallel ≤ (pi/2)</code></td>
</tr>
<tr>
<td><code>MaxAngleColinear</code> (input control)</td>
<td><code>angle.rad ~ VARIANT ( real, integer )</code></td>
<td><code>0.261799387799</code></td>
<td><code>0.0 ≤ MaxAngleColinear ≤ (pi/2)</code></td>
</tr>
<tr>
<td><code>MaxDistanceParallel</code> (input control)</td>
<td><code>real ~ VARIANT ( real, integer )</code></td>
<td><code>40</code></td>
<td><code>MaxDistanceParallel &gt; 0</code></td>
</tr>
</tbody>
</table>

**HALCON/COM Reference Manual / 2000-11-16**
14.4. TRANSFORMATION

MaxDistanceColinear (input control) ......................... real  ~> VARIANT (real, integer)
Maximum distance between two collinear line segments.

default value : 40
value suggestions : MaxDistanceColinear ∈ { 20, 30, 40, 50, 60 }

restriction : (MaxDistanceColinear > 0)

parallelization information

CombineRoadsXld is processed under mutual exclusion against itself and without parallelization.

possible predecessor functions

ModParallelsXld, GenPolygonsXld, AffineTransContourXld

possible successor functions

GetPolygonXld, GetLinesXld

see also

LinesGauss, LinesFacet, GetChannelInfo, EdgesSubPix

bibliography


module

Sub-pixel operators

```c
[out] HXLDContX ParallelContours HXLDContX.GenParallelContourXld
([in] String Mode, [in] VARIANT Distance )

void HOperatorSetX.GenParallelContourXld ([in] IHObjectX Contours,
[out] HUntypedObjectX ParallelContours, [in] String Mode,
[in] VARIANT Distance )
```

Compute the parallel contour of an XLD contour.

GenParallelContourXld computes for each of the input contours Contours a parallel contour with distance Distance. The resulting contours are returned in ParallelContours. To calculate the parallel contour, the normal vector of the input contour is needed in every contour point. The parameter Mode determines how these normal vectors are computed. If Mode = 'gradient', it is assumed that the input contours are edges, and the normal information is obtained from the gradient direction of the edge (see EdgesSubPix). For this, the attribute 'edge_direction' must exist for the input contour. If Mode = 'contour_normal', a possibly existing normal information is used to determine the normals. For this, the contour attribute 'angle' must exist (see LinesGauss, LinesFacet, or EdgesSubPix). Finally, if Mode = 'regression_normal', the normal vectors are determined from a local line fit to each contour point. Here, the normal vectors are oriented such that they point to the right side of the contour when the contour is traversed from start to end. In contrast to the first two modes, this mode can be used for all XLD contours, no matter how they were generated.

parameter

- **Contours (input iconic)** .............................. xld_cont  ~> HXLDContX / IHObjectX
  Contours to be transformed.
- **ParallelContours (output iconic)** .............................. xld_cont  ~> HXLDContX / HUntypedObjectX
  Parallel contours.
- **Mode (input control)** ................................... string  ~> String
  Mode, with which the direction information is computed.

  default value : 'regression_normal'
  value suggestions : Mode ∈ { 'gradient', 'contour_normal', 'regression_normal' }

  - **Distance (input control)** ............................... number  ~> VARIANT (real, integer)
    Distance of the parallel contour.

  default value : 1
  value list : Distance ∈ { 0.2, 0.4, 0.6, 0.8, 1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 40, 50 }

HALCON 6.0
Parallelization Information

GenParallelContourXld is reentrant and processed without parallelization.

Possible Predecessor Functions

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

Possible Successor Functions

GenPolygonsXld

See Also

GetContourXld

Module

Sub-pixel operators

---

Merge XLD contours from successive line scan images.

The operator MergeContLineScanXld connects adjacent XLD contours, which were extracted from adjacent images with the height ImageHeight. This operator was especially designed to connect contours that were extracted from images grabbed by a line scan camera. CurrConts contains the contours from the current image and PrevConts the contours from the previous one.

With the help of the parameter MergeBorder two cases can be distinguished: If the top (first) line of the current image touches the bottom (last) line of the previous image, MergeBorder must be set to 'top', otherwise set MergeBorder to 'bottom'. MergeBorder defines a margin to the border. Only those end points of the contours which are inside this margin are considered for the following merging process.

If the operator MergeContLineScanXld is used recursively, the parameter MaxImagesCont determines the maximum number of images which are covered by a merged contour. All points of the merged contour from an older image are removed.

The operator MergeContLineScanXld returns two contour arrays. PrevMergedConts contains all those contours from the previous input contours PrevConts, which could not be merged with a current contour. CurrMergedConts collects all current contours together with the merged parts from the previous images. Merged contours will exceed the original image, because the previous contours are moved upward (MergeBorder='top') or downward (MergeBorder='bottom') according to the image height.

Parameter

- **CurrConts** (input iconic) 
  Current input contours.

- **PrevConts** (input iconic) 
  Merged contours from the previous iteration.

- **CurrMergedConts** (output iconic) 
  Current contours, merged with old ones where applicable.

- **PrevMergedConts** (output iconic) 
  Contours from the previous iteration which could not be merged with the current ones.

- **ImageHeight** (input control) 
  Height of the line scan images.

  **Default Value**: 512

  **Value List**: ImageHeight ∈ {240, 480, 512}
14.4. **TRANSFORMATION**

**Margin** (input control)  
Maximum distance of contours from the image border.
- **Default Value**: 0
- **Value List**: Margin ∈ {0, 1, 2, 3, 4, 5}

**MergeBorder** (input control)  
Image line of the current image, which touches the previous image.
- **Default Value**: 'top'
- **Value List**: MergeBorder ∈ {'top', 'bottom'}

**MaxImagesCont** (input control)  
Maximum number of images covered by one contour.
- **Default Value**: 3
- **Value Suggestions**: MaxImagesCont ∈ {1, 2, 3, 4, 5}

The operator **MergeContLineScanXld** returns the value TRUE if the given parameters are correct. Otherwise, an exception will be raised.

**Parallelization Information**
**MergeContLineScanXld** is reentrant and processed without parallelization.

**Module**

Sub-pixel operators

<table>
<thead>
<tr>
<th>[out] HXLDCintX RegressContours</th>
<th>HXLDCintX.RegressContoursXld</th>
</tr>
</thead>
<tbody>
<tr>
<td>([in] String Mode, [in] long Iterations )</td>
<td></td>
</tr>
</tbody>
</table>

**HXLDContX.RegressContoursXld**

Calculate the parameters of a regression line to an XLD contour. **RegressContoursXld** calculates the following parameters for the input XLD contours **Contours**, and stores them with the resulting contours as global attributes:

- the coordinates of the normal vector of the regression line, i.e., the least-squares approximating line, of all contour points; the normal vector always points from the origin to the line (attributes: 'regr_norm_row', 'regr_norm_col'),
- the mean of the Euclidian distance of the contour points from the regression line (attribute: 'regr_mean_dist'),
- the standard deviation of these distances to the regression line (attribute: 'regr_dev_dist').

For Mode = 'no', the parameters of the regression line are calculated for all points of the contour. In addition, three different kinds of outlier treatment can be applied. Outliers are contour points which do not lie on the general contour direction in an "obvious" manner, and thus "distort" the resulting regression line.

**Mode** =

- 'drop': All contour points further away from the contour than the mean distance to the regression line are ignored for the calculation of the undistorted regression line.
- 'gauss': The distances of the contour points are weighted according to their probability of occurrence in a Gaussian distribution around the normal regression line.
- 'median': Here, also a normal distribution is assumed for the distances to the normal regression line, however with the outlier-independent standard deviation \(\text{median}(\text{all dist})\). Again, the distances are weighted, and points further away than a certain distance are ignored for the undistorted regression line.

The calculation of the undistorted regression line can be iterated several times (Iterations).
### CHAPTER 14. XLD

**Parameter**

- **Contours** (input iconic) \( \xld_{\text{cont}} \rightarrow \text{HXMLContX} / \IHObjectX \)
  - Input XLD contours.

- **RegressContours** (output iconic) \( \xld_{\text{cont}} \rightarrow \text{HXMLContX} / \IHuntypedObjectX \)
  - Resulting XLD contours.

- **Mode** (input control) \( \rightarrow \text{String} \)
  - Type of outlier treatment.
  - **Default Value**: 'no'
  - **Value List**: 'no', 'drop', 'gauss', 'median'

- **Iterations** (input control)
  - **Default Value**: 1
  - **Value Suggestions**: \{1, 2, 3, 5, 10, 20\}

---

**Parallelization Information**

- **RegressContoursXld** is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

- GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

---

**Possible Successor Functions**

- GetRegressParamsXld

**See Also**

- SmoothContoursXld, GetContourGlobalAttribXld, QueryContourGlobalAttribsXld

---

**Bibliography**


---

**Module**

Sub-pixel operators

---

```c
[out] HXMLContX ContoursSplit HXMLContX.SegmentContoursXld
([in] String Mode, [in] long SmoothCont, [in] double MaxLineDist1,
 [in] double MaxLineDist2 )

void HOperatorSetX.SegmentContoursXld ([in] IObjectX Contours,
 [out] HUntypedObjectX ContoursSplit, [in] String Mode, [in] long SmoothCont,
 [in] double MaxLineDist1, [in] double MaxLineDist2 )
```

**Segment XLD contours into line segments and circular or elliptic arcs.**

SegmentContoursXld segments the input contours Contours into lines if Mode='lines', into lines and circular arcs if Mode='lines Circles', or into lines and elliptic arcs if Mode='lines Ellipses'. The segmented contours are returned in ContoursSplit. The information on whether an output contour represents a line or a circular of elliptic arc is done via the global contour attribute 'cont approx' (see GetContourGlobalAttribXld). If 'cont approx'=-1, the contour is best approximated by a line segment, if 'cont approx'=0, by an elliptic arc, and if 'cont approx'=1, by a circular arc.

SegmentContoursXld first approximates the input contours by polygons. With this, the contours are over-segmented in curved areas. After this, neighboring line segments are substituted by circular or elliptic arcs, respectively, if the contour can be approximated better by an arc. If SmoothCont is set to a value > 0, the input contours are first smoothed (see SmoothContoursXld). This can be necessary to prevent very short segments in the polygonal approximation and to achieve a more robust fit with circular or elliptic arcs, because the smoothing suppresses outliers on the contours.

The initial polygonal approximation is done by using the Ramer algorithm (see GenPolygonsXld) with a maximum distance of MaxLineDist1. After this, circular or elliptic arcs are fit into neighboring line segments. If the maximum distance of the resulting arc to the contour is smaller than the maximum distance of the two line segments, the two line segments are replaced with the arc. This procedure is iterated until no more changes occur.
After this, the parts of the contour that are still approximated by line segments are again segmented with a polygonal approximation with maximum distance $\text{MaxLineDist}_2$, and the newly created line segments are merged to circular or elliptical arcs where possible. Obviously, this changes the output only if $\text{MaxLineDist}_2 < \text{MaxLineDist}_1$. This two-step approach is more efficient than a one-step approach with $\text{MaxLineDist}_2$, since fewer line segments are generated in the first step, and hence the circle or ellipse fit has to be performed less often. Therefore, parts of the input contours that can be approximated by long arcs can be found more efficiently. In the second step, parts of the input contours that can be approximated by short arcs are found and the end parts of long arcs are refined.

**Parameter**

- **Contours** (input iconic) ... Contours to be segmented.
- **ContoursSplit** (output iconic) ... Segmented contours.
- **Mode** (input control) ... Mode for the segmentation of the contours.
  - Default Value: 'lines_circles'
  - Value List: Mode ∈ {'lines', 'lines_circles', 'lines_ellipses'}
- **SmoothCont** (input control) ... Number of points used for smoothing the contours.
  - Default Value: 5
  - Value Suggestions: SmoothCont ∈ {0, 3, 5, 7, 9}
  - Restriction: $(\text{SmoothCont} \geq 3) \lor (\text{SmoothCont} \land \text{odd})$
- **MaxLineDist1** (input control) ... Maximum distance between a contour and the approximating line (first iteration).
  - Default Value: 4.0
  - Value Suggestions: MaxLineDist1 ∈ {1.0, 1.5, 2.0, 2.5, 3.0, 3.5}
  - Restriction: $(\text{MaxLineDist}_1 \geq 0.0)$
- **MaxLineDist2** (input control) ... Maximum distance between a contour and the approximating line (second iteration).
  - Default Value: 2.0
  - Value Suggestions: MaxLineDist2 ∈ {1.0, 1.5, 2.0, 2.5, 3.0, 3.5}
  - Restriction: $(\text{MaxLineDist}_2 \geq 0.0)$

**Example**

```halcon
read_image (Image, 'pumpe')
edges_sub_pix (Image, Edges, 'canny', 1.5, 15, 40)
segment_contours_xld (Edges, ContoursSplit, 'lines_circles', 5, 4, 2)
count_obj (ContoursSplit, Number)
gen_empty_obj (Lines)
gen_empty_obj (Circles)
for I := 1 to Number by 1
  select_obj (ContoursSplit, Contour, I)
  get_contour_global_attrib_xld (Contour, 'cont_approx', Type)
  if (Type = -1)
    concat_obj (Lines, Contour, Lines)
  else
    concat_obj (Circles, Contour, Circles)
  endif
endfor
fit_line_contour_xld (Lines, 'tukey', -1, 0, 5, 2, RowBegin, ColBegin, RowEnd, ColEnd, Nr, Nc)
fit_circle_contour_xld (Circles, 'atukey', -1, 2, 0, 3, 2, Row, Column, Radius, StartPhi, EndPhi, PointOrder)
```

**Parallelization Information**

SegmentContoursXld is *local* and processed *completely exclusively* without parallelization.

HALCON 6.0
**Possible Predecessor Functions**

- GenContoursSkeletonXld, LinesGauss, EdgesSubPix

**Possible Successor Functions**

- FitLineContourXld, FitEllipseContourXld, FitCircleContourXld

**See Also**

- SplitContoursXld, GetContourGlobalAttribXld, SmoothContoursXld, GenPolygonsXld

---

**Module**

Sub-pixel operators

---

```
[out] HXLDContX SmoothedContours HXLDContX.SmoothContoursXld
([in] long NumRegrPoints )
```

---

**Smooth an XLD contour.**

SmoothContoursXld smooths the input XLD contours Contours and returns the smoothed contours in SmoothedContours. The smoothing is done by projecting the contours’ points onto a local regression line (i.e. a least-squares approximating line), which is computed from NumRegrPoints on each side of the current contour point. This operator should be called, for example, before contours are scaled.

---

**Parameter**

- **Contours** (input iconic) ............................... xld_cont \( \sim \) HXLDContX / IHObjectX Contour to be smoothed.
- **SmoothedContours** (output iconic) ..................... xld_cont \( \sim \) HXLDContX / HUntypedObjectX Smoothed contour.
- **NumRegrPoints** (input control) ........................ long Number of points used to calculate the regression line.
  
  **Default Value** : 5
  
  **Value Suggestions** : NumRegrPoints \( \in \{3, 5, 7, 9\}\)

---

**Parallelization Information**

SmoothContoursXld is reentrant and processed without parallelization.

---

**Possible Predecessor Functions**

- GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

---

**Possible Successor Functions**

- AffineTransContourXld, GenPolygonsXld, LocalMaxContoursXld

---

**See Also**

- GetContourXld

---

**Module**

Sub-pixel operators

---

```
```

---

**Split XLD contours at dominant points.**

SplitContoursXld splits the contours which were used to generate the polygons Polygons at prominent points. If the mode 'polygon' is selected, the contours are split at the polygons’ control points. In mode 'dominant', they are split at dominant points, i.e., at points for which the calculated change of direction is larger than
the (empirically determined) threshold \(2\pi\text{Weight}/\text{contour length}\), and for which in the (empirically determined) neighborhood of \(\sqrt{\text{Smooth}} \cdot \log(\text{contour length})\) points no larger change of direction occurs. The contour direction is determined from the direction of the regression line (i.e. the least-squares approximating line) through all points in a neighborhood of \(\text{Smooth}\) points around a contour point. The directions thus determined are smoothed with a Gaussian of width \(\text{Smooth}\). \text{Weight} is a weighting factor for the sensitiveness of the operator. The larger \text{Weight} is selected, the less dominant points are found.

\begin{verbatim}
Parameter

\textbf{Polygons} (input iconic) \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldOTS
The parameters of the regression line are calculated new for merged contours.
The resulting contours cannot be displayed.

**Attention**
Before the contour parameters can be returned by `UnionStraightContoursHistoXld`, the parameters of the regression line to the contour must be calculated by calling `RegressContoursXld`.

**Parameter**

- **Contours** (input iconic)  
  Input XLD contours.

- **UnionContours** (output iconic)  
  Output XLD contours.

- **SelectedContours** (output iconic)  
  Output XLD contours.

- **RefLineStartY** (input control)  
  y coordinate of the starting point of the reference line.  
  **Default Value:** 0

- **RefLineStartX** (input control)  
  x coordinate of the starting point of the reference line.  
  **Default Value:** 0

- **RefLineEndY** (input control)  
  y coordinate of the endpoint of the reference line.  
  **Default Value:** 0

- **RefLineEndX** (input control)  
  x coordinate of the endpoint of the reference line.  
  **Default Value:** 0

- **nWidth** (input control)  
  Maximum distance.  
  **Default Value:** 1

- **MaxWidth** (input control)  
  Maximum Width between two minimas.  
  **Default Value:** 1

- **FilterSize** (input control)  
  Size of Smoothfilter  
  **Default Value:** 1  
  **Typical Range of Values:** 1 ≤ FilterSize ≤ 1

- **HistoValues** (output control)  
  Output Values of Histogram.

**Parallelization Information**
`UnionStraightContoursHistoXld` is reentrant and processed without parallelization.

**Possible Predecessor Functions**

- `RegressContoursXld`

**See Also**

- `FitLineContourXld`, `GetContourXld`, `GetContourAttribXld`, `GenContoursSkeletonXld`, `LinesGauss`, `LinesFacet`, `EdgesSubPix`, `GetRegressParamsXld`, `GetContourGlobalAttribXld`, `QueryContourGlobalAttribsXld`

**Module**

Sub-pixel operators
### Merge neighboring straight contours having a similar direction.

UnionStraightContoursXld merges neighboring XLD contours Contours if certain criteria are fulfilled. The merging is not done recursively, and only between two contours. The parameter Iterations controls how often this union step is executed.

Two contours are merged if the shortest distance between their end points (end points are the projections of the contours' first and last points onto its regression line) is smaller than MaxDist, and if the difference in direction (i.e. the regression lines' direction) is smaller than MaxDiff (radians).

If only one of the criteria is fulfilled, the decision on merging can be influenced by the weighting factor Percent, which allows the exceeding of one limit to be balanced by the other value remaining below the limit by the same amount. The end point distance is weighted by Percent, while the directional difference is weighted by \( \text{Percent} \).

If, for example, two contours have an end point distance of 5.0 and a directional difference of 0.5 (with the limits chosen being \( \text{MaxDist} = 4.0 \) and \( \text{MaxDiff} = 0.625 \)), both values differ from the limits by 25%. By choosing \( \text{Percent} = 60\% \), the larger end point distance is weighted more than the smaller directional difference, and thus the contours are not merged. Contrary, if \( \text{Percent} = 40\% \) is chosen, the contours are merged.

For \( \text{Percent} = 100\% \), only the end point distance is taken into account, while for \( \text{Percent} = 0\% \) only the difference of direction is used. If \( \text{Percent} = 50\% \) both criteria are equally valid.

In case there are parallel contours, there is a danger of merging neighboring contours. If this is to be avoided, \( \text{Mode} = \text{’noparallel’} \) has to be chosen, while otherwise \( \text{Mode} = \text{’paralleltoo’} \) suffices. For \( \text{Mode} = \text{’every’} \), contours are merged unconditionally. All other parameters have no influence in this case.

The parameters of the regression line are calculated anew for merged contours.

---

**Attention**

Before the contour parameters can be returned by GetRegressParamsXld, the parameters of the regression line to the contour must be calculated by calling RegressContoursXld.

---

**Parameter**

- **Contours** (input iconic) ............................................. xld_cont \( \leftrightarrow \) HXLDContX / IHObjectX
  
  Input XLD contours.

- **UnionContours** (output iconic) ................................. xld_cont \( \leftrightarrow \) HXLDContX / HUntypedObjectX
  
  Output XLD contours.

- **MaxDist** (input control) ............................................. real \( \leftrightarrow \) double
  
  Maximum distance of the contours' endpoints.

  **Default Value** : 5.0

- **MaxDiff** (input control) ............................................. angle.rad \( \leftrightarrow \) double
  
  Maximum difference in direction.

  **Default Value** : 0.5

- **Percent** (input control) ............................................. real \( \leftrightarrow \) double
  
  Weighting factor for the two selection criteria.

  **Default Value** : 50.0

- **Mode** (input control) ............................................. string \( \leftrightarrow \) String
  
  Should parallel contours be taken into account?

  **Default Value** : ‘noparallel’

  **Value List** : Mode \( \in \{ \text{’noparallel’}, \text{’paralleltoo’}, \text{’every’} \} \)
**Iterations** (input control) .......................... string  \~  VARIANT ( integer, string )
Number of iterations or 'maximum'.

**Default Value**: 'maximum'

**Value Suggestions**: Iterations ∈ \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'maximum'\}

**Typical Range of Values**: 1 ≤ Iterations ≤ 1(lin)

**Minimal Value Step**: 1

**Recommended Value Step**: 1

---

**Parallelization Information**

UnionStraightContoursXld is **reentrant** and processed **without** parallelization.

---

**Possible Predecessor Functions**

RegressContoursXld

---

**See Also**

FitLineContourXld, GetContourXld, GetContourAttribXld, GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix, GetRegressParamsXld, GetContourGlobalAttribXld, QueryContourGlobalAttribsXld

---

**Module**

Sub-pixel operators
Chapter 15

Class Hierarchy

15.1 HBarcodeX

Grouping class for barcode related functionality.

<table>
<thead>
<tr>
<th>Number of Methods</th>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td></td>
</tr>
<tr>
<td>&lt;none&gt;</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td></td>
</tr>
</tbody>
</table>

Methods

- **Decode1DBarCode**
  Decode a sequence of elements of a barcode.

- **Decode2DBarCode**
  Decode 2D bar code data.

- **Discrete1DBarCode**
  Generate a discrete barcode from the elements widths.

- **Find1DBarCode**
  Look for one barcode in an image.

- **Find1DBarCodeRegion**
  Look for multiple barcode regions in an image.

- **Find2DBarCode**
  Find regions that might contain a 2D bar code.

- **Gen1DBarCodeDescr**
  Generate a description of a 1D bar code.

- **Gen1DBarCodeDescrGen**
  Generate a generic description of a 1D bar code.

- **Gen2DBarCodeDescr**
  Generate a generic description of a 2D bar code class.

- **Get1DBarCode**
  Extract the widths of the elements inside a barcode region.

- **Get2DBarCode**
  Extract the values of the data elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”).

- **Get2DBarCodePos**
  Extract the data values of the elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”) and their positions in the image.
15.2 HBgEstiX

Represents an instance of a background estimator.

- **Class Description**
  - Number of Methods: 6
  - Number of Properties: 0
  - Abstract Class: no
  - Base Class(es): 
  - Automatic Destructor: yes, through CloseBgEsti (not member of this class!)
  - Object Identity: (through internal handle)

- **Constructor(s)**
  - CreateBgEsti: Generate and initialize a data set for the background estimation.

- **Methods**
  - GetBgEstiParams: Return the parameters of the data set.
  - GiveBgEsti: Return the estimated background image.
  - RunBgEsti: Estimate the background and return the foreground region.
  - SetBgEstiParams: Change the parameters of the data set.
  - UpdateBgEsti: Change the estimated background image.

15.3 HClassBoxX

Represents an instance of a classifier.

- **Class Description**
  - Number of Methods: 13
  - Number of Properties: 0
  - Abstract Class: no
  - Base Class(es): 
  - Automatic Destructor: yes, through CloseClassBox (not member of this class!)
  - Object Identity: (through internal handle)

- **Constructor(s)**
  - CreateClassBox: Create a new classifier.

- **Methods**
  - ClassNdimBox: Classify pixels using hyper-cuboids.
  - DescriptClassBox: Description of the classifier.
  - EnquireClassBox: Classify a tuple of attributes.
15.4  HFeatureSetX

Represents an instance of a training used for the classifier.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadSampset</td>
</tr>
<tr>
<td>Read a training data set from a file.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>LearnSampsetBox</td>
</tr>
<tr>
<td>Train the classifier with one data set.</td>
</tr>
<tr>
<td>TestSampsetBox</td>
</tr>
<tr>
<td>Classify a set of arrays.</td>
</tr>
</tbody>
</table>

15.5  HFileX

Represents an instance of a file.
15.6 **HFramegrabberX**

Represents an instance of a frame grabber.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
<tr>
<td>(through internal handle)</td>
</tr>
<tr>
<td>Constructor(s)</td>
</tr>
</tbody>
</table>

### Methods

- **OpenFramegrabber**
  
  Open and configure a frame grabber.

- **GetFramegrabberLut**
  
  Query frame grabber lut.

- **GetFramegrabberParam**
  
  Get specific parameters for a frame grabber.

- **GrabImage**
  
  Grab an image from the specified frame grabber.

- **GrabImageAsync**
  
  Grab of an image from the specified frame grabber and start of the asynchronous grab of the next image.

- **GrabImageStart**
  
  Start an asynchronous grab of an image from the specified frame grabber.

- **GrabRegion**
  
  Grab and segment an image from the specified frame grabber.
GrabRegionAsync ................................................................. P.351
Grab and segment an image from the specified frame grabber and start the asynchronous grab of the next image.

SetFramegrabberLut ......................................................... P.356
Set frame grabber lut.

SetFramegrabberParam .................................................... P.357
Set specific parameters for a frame grabber.

Properties

- **BitsPerChannel** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **ColorSpace** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **ExternalTrigger** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **Field** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **Gain** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **HorizontalResolution** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **ImageHeight** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **ImageWidth** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **LineIn** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **Name** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **Port** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **StartColumn** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **StartRow** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **VerticalResolution** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.
CHAPTER 15. CLASS HIERARCHY

15.7 HFunction1dX

Represents an instance of a 1d function.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

Constructor(s)

- CreateFunct1DArray
  Create a function from a sequence of y-values.
- CreateFunct1DPairs
  Create a function from a set of (x,y) pairs.
- ReadFunct1D
  Read a function from a file.

Methods

- AbsFunct1D
  Absolute value of the y values.
- DistanceFunct1D
  Compute the distance of two functions.
- Funct1DToPairs
  Access to the x/y values of a function.
- GetPairFunct1D
  Access a function value using the index of the control points.
- IntegrateFunct1D
  Compute the positive and negative areas of a function.
- MatchFunct1DTrans
  Calculate transformation parameters between two functions.
- NegateFunct1D
  Negation of the y values.
- NumPointsFunct1D
  Number of control points of the function.
- SampleFunct1D
  Sample a function equidistantly in an interval.
- ScaleYFunct1D
  Multiplication and addition of the y values.
- SmoothFunct1DGauss
  Smooth an equidistant function with a Gaussian function.
- SmoothFunct1DMean
  Smooth a 1D function by averaging its values.
- TransformFunct1D
  Transform a function using given transformation parameters.
- WriteFunct1D
  Write a function to a file.
- XRangeFunct1D
  Smallest and largest x value of the function.
- YRangeFunct1D
  Smallest and largest y value of the function.
15.8 HGNUPLOTX

Represents an instance of a connection to a gnuplot process.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

Constructor(s)

- GnuplotOpenFile .......................... P.193
  Open a gnuplot file for visualization of images and control values.

- GnuplotOpenPipe ......................... P.194
  Open a pipe to a gnuplot process for visualization of images and control values.

Methods

- GnuplotPlotCtrl .......................... P.194
  Plot control values using gnuplot.

- GnuplotPlotFunct1D ...................... P.195
  Plot a function using gnuplot.

- GnuplotPlotImage ........................ P.196
  Visualize images using gnuplot.

15.9 HHistogramX

Group class for histogram functionality.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

Methods

- GenRegionHisto .......................... P.470
  Convert a histogram into a region.

- HistoToThresh ........................... P.558
  Determine gray value thresholds from a histogram.

15.10 HHomMat2dX

Represents a homogeneous 2D transformation matrix.
### Class Description

- **Number of Methods**: 19
- **Number of Properties**: 0
- **Abstract Class**: No
- **Base Class(es)**: <none>
- **Automatic Destructor**: No
- **Object Identity (through internal handle)**: Yes

### Constructor(s)

- **DvToHomMat2d**: P.627
  Approximate an affine map from a displacement vector field.

- **HomMat2dIdentity**: P.628
  Generate the homogeneous transformation matrix of the identical 2D transformation.

- **ReadWorldFile**: P.16
  Read the geo coding from an ARC/INFO world file.

- **VectorAngleToRigid**: P.642
  Compute a rigid affine transformation from points and angles.

- **VectorToHomMat2d**: P.643
  Approximate an affine transformation from point correspondences.

- **VectorToRigid**: P.644
  Approximate a rigid affine transformation from point correspondences.

- **VectorToSimilarity**: P.645
  Approximate an similarity transformation from point correspondences.

### Methods

- **AffineTransContourXld**: P.887
  Apply an arbitrary affine transformation to an XLD contour.

- **AffineTransImage**: P.27
  Apply an arbitrary affine transformation to an image.

- **AffineTransPoint2D**: P.625
  Apply a homogeneous 2D transformation matrix to points.

- **AffineTransPolygonXld**: P.888
  Apply an arbitrary affine transformation to an XLD polygon.

- **AffineTransRegion**: P.454
  Transform a region using an affine transformation.

- **HomMat2dCompose**: P.628
  Compute the affine transformation parameters from a homogeneous 2D transformation matrix.

- **HomMat2dInvert**: P.629
  Invert a homogeneous 2D transformation matrix.

- **HomMat2dRotate**: P.629
  Add a rotation to a homogeneous 2D transformation matrix.

- **HomMat2dScale**: P.630
  Add a scaling to a homogeneous 2D transformation matrix.

- **HomMat2dSlant**: P.631
  Add a slant to a homogeneous 2D transformation matrix.

- **HomMat2dToAffinePar**: P.632
  Compute the affine transformation parameters from a homogeneous 2D transformation matrix.

- **HomMat2dTranslate**: P.633
  Add a translation to a homogeneous 2D transformation matrix.
15.11 HHomMat3dX

Represents a homogeneous 3D transformation matrix.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

*Constructor(s)*

▷ HomMat3dIdentity .................................................. P.635
   Generate the homogeneous transformation matrix of the identical 3D transformation.

*Methods*

▷ AffineTransPoint3D .................................................. P.626
   Apply a homogeneous 3D transformation matrix to points.

▷ HomMat3dCompose .................................................. P.634
   Compose two homogeneous 3D transformation matrices.

▷ HomMat3dInvert .................................................. P.636
   Invert a homogeneous 3D transformation matrix.

▷ HomMat3dRotate .................................................. P.637
   Add a rotation to a homogeneous 3D transformation matrix.

▷ HomMat3dScale .................................................. P.639
   Add a scaling to a homogeneous 3D transformation matrix.

▷ HomMat3dToPose .................................................. P.707
   Convert homogeneous transformation matrix into 3D pose parameter.

▷ HomMat3dTranslate .................................................. P.640
   Add a translation to a homogeneous 3D transformation matrix.

15.12 HImageX

Represents an instance of an image object.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

*Constructor(s)*

▷ GenBandfilter .................................................. P.93
   Generate an ideal band filter.

▷ GenBandpass .................................................. P.94
   Generate an ideal bandpass filter.

▷ GenDiscSe .................................................. P.381
   Generate ellipsoidal structuring elements for gray morphology.
CHAPTER 15. CLASS HIERARCHY

- GenFilterMask ................................................................. P.94
  Store a filter mask in the spatial domain as a real-image.

- GenGabor ................................................................. P.95
  Generate a Gabor filter.

- GenHighpass .................................................................. P.97
  Generate an ideal highpass filter.

- GenImage1 .................................................................. P.300
  Create an image from a pointer to the pixels.

- GenImage1Extern .......................................................... P.301
  Create an image from a pointer on the pixels with storage management.

- GenImage1Rect ............................................................... P.302
  Create an image with a rectangular domain from a pointer on the pixels (with storage management).

- GenImage3 .................................................................. P.304
  Create an image from three pointers to the pixels (red/green/blue).

- GenImageConst ................................................................. P.305
  Create an image with constant gray value.

- GenImageGrayRamp .......................................................... P.306
  Create a gray value ramp.

- GenImageSurfaceSecondOrder .................................................. P.308
  Create a curved gray surface with second order polynomial.

- GenLowpass .................................................................. P.97
  Generate an ideal lowpass filter.

- GenPsfDefocus ................................................................. P.165
  Generate an impulse response of an uniform out-of-focus blurring.

- GenPsfMotion ................................................................. P.166
  Generate an impulse response of a (linearly) motion blurring.

- GenSinBandpass ................................................................. P.98
  Generate a bandpass filter with sinusoidal shape.

- GenStdBandpass ................................................................. P.99
  Generate a bandpass filter with Gaussian or sinusoidal shape.

- GiveBgEsti .................................................................. P.651
  Return the estimated background image.

- GrabImage ................................................................. P.348
  Grab an image from the specified frame grabber.

- GrabImageAsync ............................................................. P.349
  Grab of an image from the specified frame grabber and start of the asynchronous grab of the next image.

- ReadGraySe ................................................................. P.388
  Load a structuring element for gray morphology.

- ReadImage ................................................................. P.11
  Read an image with different file formats.

- ReadOcrTrainf ................................................................. P.794
  Read training characters from files and convert to images.

- ReadOcrTrainfSelect .......................................................... P.795
  Read training specific characters from files and convert to images.

- ReadSequence ................................................................. P.12
  Read images.

- ReceiveImage ................................................................. P.619
  Receive an image over a socket connection.

- SimCalTab ................................................................. P.716
  Simulate a video image with calibration table.
Methods

- **AbsImage**
  Calculate the absolute value (modulus) of an image.
  
- **AccessChannel**
  Access a channel of a multichannel image.

- **AdaptTemplate**
  Adapting a template to the size of an image.

- **AddImage**
  Add two images.

- **AddNoiseDistribution**
  Add noise to an image.

- **AddNoiseWhite**
  Add noise to an image.

- **AffineTransImage**
  Apply an arbitrary affine transformation to an image.

- **AnisotropeDiff**
  Smooth an image by edge-preserving anisotropic diffusion.

- **AppendChannel**
  Append additional matrices (channels) to the image.

- **AutoThreshold**
  Segment an image using thresholds determined from its histogram.

- **BandpassImage**
  Edge extraction using bandpass filters.

- **BestMatch**
  Searching the best matching of a template and an image.

- **BestMatchMg**
  Suche nach dem besten Grauwert-Match in einer Pyramide.

- **BestMatchPreMg**
  Suche nach dem besten Grauwert-Match in einer vorherberechneten Pyramide.

- **BestMatchRot**
  Searching the best matching of a template and an image with rotation.

- **BestMatchRotMg**
  Searching the best matching of a template and a pyramid with rotation.

- **BinThreshold**
  Segment a black-and-white image using an automatically determined threshold.

- **BitAnd**
  Bit-by-bit AND of all pixels of the input images.

- **BitLshift**
  Logical “AND” of each pixel using a bit mask.

- **BitNot**
  Complement all bits of the pixels.

- **BitOr**
  Bit-by-bit OR of all pixels of the input images.

- **BitRshift**
  Right shift of all pixels of the image.

- **BitSlice**
  Extract a bit from the pixels.

- **BitXor**
  Bit-by-bit XOR of all pixels of the input images.
CHAPTER 15. CLASS HIERARCHY

- **ChangeDomain** .......................................................... P.314
  Change definition domain of an image.

- **ChangeFormat** .......................................................... P.338
  Change image size.

- **ChangeRadialDistortionImage** ...................................... P.684
  Change the radial distortion of an image.

- **ChannelsToImage** ....................................................... P.290
  Convert one-channel images into a multichannel image

- **CharThreshold** .......................................................... P.542
  Perform a threshold segmentation for extracting characters.

- **CheckDifference** ....................................................... P.543
  Compare two images pixel by pixel.

- **Class2DimSup** .......................................................... P.545
  Segment an image using two-dimensional pixel classification.

- **Class2DimUnsup** ....................................................... P.546
  Segment two images by clustering.

- **ClassNdimBox** .......................................................... P.547
  Classify pixels using hyper-cuboids.

- **ClassNdimNorm** ........................................................ P.548
  Classify pixels using hyper-spheres or hyper-cubes.

- **ComplexToReal** ....................................................... P.364
  Convert a complex image into two real images.

- **Compose2** .............................................................. P.290
  Convert two images into a two-channel image.

- **Compose3** .............................................................. P.291
  Convert 3 images into a three-channel image.

- **Compose4** .............................................................. P.292
  Convert 4 images into a four-channel image.

- **Compose5** .............................................................. P.292
  Convert 5 images into a five-channel image.

- **Compose6** .............................................................. P.293
  Convert 6 images into a six-channel image.

- **Compose7** .............................................................. P.294
  Convert 7 images into a seven-channel image.

- **ConvertImageType** .................................................... P.364
  Convert the type of an image.

- **ConvolFft** ............................................................. P.88
  Convolve an image with a byte-mask in the frequency domain.

- **ConvolGabor** .......................................................... P.89
  Convolve an image with a Gabor filter in the frequency domain.

- **ConvolImage** .......................................................... P.131
  Convolve an image with an arbitrary filter mask.

- **CoocFeatureMatrix** .................................................. P.319
  Calculate gray value features from a co-occurrence matrix.

- **CopyImage** ............................................................ P.299
  Copy an image and allocate new memory for it.

- **CornerResponse** ...................................................... P.115
  Searching corners in images.

- **CountChannels** ....................................................... P.294
  Count channels of image.

- **CreateBgEsti** ........................................................ P.647
  Generate and initialize a data set for the background estimation.
CreateShapeModel
Prepare a shape model for matching.

CreateTemplate
Preparing a pattern for template matching.

CreateTemplateRot
Preparing a pattern for template matching with rotation.

CropDomain
Cut out of defined gray values.

CropDomainRel
Cut out an image area relative to the domain.

CropPart
Cut out a rectangular image area.

CropRectangle1
Cut out a rectangular image area.

Decompose2
Convert a two-channel image into two images.

Decompose3
Convert a three-channel image into three images.

Decompose4
Convert a four-channel image into four images.

Decompose5
Convert a five-channel image into five images.

Decompose6
Convert a six-channel image into six images.

Decompose7
Convert a seven-channel image into seven images.

DepthFromFocus
Extract depth using multiple focus levels.

DerivateGauss
Convolve an image with derivatives of the Gaussian.

DetectEdgeSegments
Detect straight edge segments.

DeviationImage
Calculate the standard deviation of gray values within rectangular windows.

DiffOfGauss
Approximate the LoG operator (Laplace of Gaussian).

DispChannel
Displays images with several channels.

DispColor
Displays a color (RGB) image

DispImage
Displays gray value images.

DivImage
Divide two images.

DoOcvSimple
Verification of a pattern using an OCV tool.

DualRank
Opening, Median and Closing with circle or rectangle mask.

DualThreshold
Threshold operator for signed images.

DvfToHomMat2d
Approximate an affine map from a displacement vector field.
DvToInt1 ................................................................. P.365
   Convert a displacement vector field into two int1-images.

DynThreshold ....................................................... P.552
   Segment an image using a local threshold.

EdgesImage ........................................................... P.62
   Extract edges using Deriche, Lanser, Shen, or Canny filters.

EdgesSubPix .......................................................... P.65
   Extract sub-pixel precise edges using Deriche, Lanser, Shen, or Canny filters.

EliminateMinMax .................................................... P.143
   Smooth an image in the spatial domain to suppress noise.

EliminateSp .......................................................... P.145
   Replace values outside of thresholds with average value.

Emphasize ........................................................... P.84
   Enhance contrast of the image.

EnergyGabor .......................................................... P.89
   Calculate the energy of a two-channel image.

EntropyImage ......................................................... P.162
   Calculate the entropy of gray values within a rectangular window.

EquHistImage .......................................................... P.85
   Histogram linearisation of images

EstimateAlAm .......................................................... P.806
   Estimate the albedo of a surface and the amount of ambient light.

EstimateSlAlLr .......................................................... P.807
   Estimate the slant of a light source and the albedo of a surface.

EstimateSlAlZc .......................................................... P.808
   Estimate the slant of a light source and the albedo of a surface.

EstimateTiltLr .......................................................... P.808
   Estimate the tilt of a light source.

EstimateTiltZc .......................................................... P.809
   Estimate the tilt of a light source.

ExhaustiveMatch ...................................................... P.120
   Matching of a template and an image.

ExhaustiveMatchMg ................................................... P.121
   Matching a template and an image in a resolution pyramid.

ExpandDomainGray ................................................... P.132
   Expand the domain of an image and set the gray values in the expanded domain.

ExpandLine ........................................................... P.556
   Expand a region starting at a given line.

FastMatch ............................................................ P.123
   Searching all good matches of a template and an image.

FastMatchMg .......................................................... P.124
   Searching all good grayvalue matches in a pyramid.

FastThreshold .......................................................... P.557
   Fast selection of grayvalues within a given gray intervall.

FftGeneric ............................................................ P.90
   Compute the fast Fourier transform of an image.

FftImage ............................................................. P.91
   Compute the fast Fourier transform of an image.

FftImageInv ........................................................... P.92
   Compute the inverse fast Fourier transform of an image.

FillDvf ............................................................. P.125
   Expansion of a displacement vector field onto undefined areas.
- **FillInterlace**: Interpolate 2 video half images. P.146
- **Find1DBarCode**: Look for one barcode in an image. P.659
- **Find1DBarCodeRegion**: Look for multiple barcode regions in an image. P.662
- **Find2DBarCode**: Find regions that might contain a 2D bar code. P.663
- **FindCalTab**: Segment the calibration table region in the image. P.697
- **FindMarksAndPose**: Extract the 2D calibration marks from the video image and calculate initial values for the external camera parameters. P.698
- **FindShapeModel**: Find the best matches of a shape model in an image. P.771
- **FreiAmp**: Detect edges (amplitude) using the Frei-Chen operator. P.67
- **FreiDir**: Detect edges (amplitude and direction) using the Frei-Chen operator. P.68
- **FullDomain**: Expand the domain of an image to maximum. P.315
- **GaussImage**: Smooth using discrete gauss functions. P.147
- **GenGaussPyramid**: Calculating a Gauss pyramid. P.125
- **GenImageProto**: Set the gray values of an image to a specified value. P.307
- **Get1DBarCode**: Extract the widths of the elements inside a barcode region. P.670
- **Get2DBarCode**: Extract the values of the data elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”). P.671
- **Get2DBarCodePos**: Extract the data values of the elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”) and their positions in the image. P.675
- **GetDomain**: Get the domain of an image. P.315
- **GetGrayval**: Access the gray values of an image object. P.357
- **GetImagePointer1**: Access the pointer of a channel. P.358
- **GetImagePointer1Rect**: Access to the image data pointer and the image data inside the smallest rectangle of the domain of the input image. P.359
- **GetImagePointer3**: Access the pointers of a colored image. P.360
- **GetImageTime**: Request time at which the image was created. P.310
- **GnuplotPlotImage**: Visualize images using gnuplot. P.196
- **GrayBothat**: Perform a gray value bottom hat transformation on an image. P.381
GrayClosing
Performs a gray value closing on an image.

GrayDilation
Performs a gray value dilation on an image.

GrayDilationRect
Determines the maximum gray value within a rectangle.

GrayErosion
Performs a gray value erosion on an image.

GrayErosionRect
Determines the minimum gray value within a rectangle.

GrayInside
Calculates the lowest possible gray value on an arbitrary path to the image border for each point in the image.

GrayOpening
Performs a gray value opening on an image.

GrayProjections
Calculates horizontal and vertical gray-value projections.

GrayRangeRect
Determines the gray value range within a rectangle.

GraySkeleton
Thinning of gray value images.

GrayTophat
Performs a gray value top hat transformation on an image.

HighpassImage
Extracts high frequency components from an image.

HysteresisThreshold
Performs a hysteresis threshold operation on an image.

Illuminate
Illuminate image.

ImageToChannels
Converts a multichannel image into one-channel images.

InspectShapeModel
Creates the representation of a shape model.

Int1ToDvf
Converts two int1-images into a displacement vector field.

InvertImage
Inverts an image.

KirschAmp
Detects edges (amplitude) using the Kirsch operator.

KirschDir
Detects edges (amplitude and direction) using the Kirsch operator.

LabelToRegion
Extracts regions with equal gray values from an image.

Laplace
Detects all local maxima in an image.
LutTrans ................................................................. P.134
Transform an image with a gray-value look-up-table.

MaxImage ............................................................... P.37
Calculate the maximum of two images pixel by pixel.

MeanImage ............................................................. P.149
Smooth by averaging.

MeanN ................................................................. P.150
Average gray values over several channels.

MeanSp ............................................................... P.151
Suppress salt and pepper noise.

MeasurePairs ....................................................... P.779
Extract straight edge pairs perpendicular to a rectangle.

MeasurePos ......................................................... P.781
Extract straight edges perpendicular to a rectangle.

MeasureProjection ............................................... P.783
Extract a gray value profile perpendicular to a rectangle.

MeasureThresh .................................................... P.784
Extracting points with a particular grey value along a rectangle.

MedianImage ....................................................... P.152
Median filtering with different rank masks.

MedianSeparate ................................................... P.153
Separated median filtering with rectangle masks.

MedianWeighted .................................................. P.154
Weighted median filtering with different rank masks.

MidrangeImage ................................................... P.155
Calculate the average of maximum and minimum inside any mask.

MinImage ............................................................ P.38
Calculate the minimum of two images pixel by pixel.

MirrorImage ........................................................ P.29
Mirror an image.

Monotony ............................................................ P.126
Calculating the monotony operation.

MultImage ........................................................... P.39
Multiply two images.

NonmaxSuppressionAmp ........................................ P.563
Suppress non-maximum points on an edge.

NonmaxSuppressionDir ......................................... P.564
Suppress non-maximum points on an edge.

OcrGetFeatures .................................................. P.793
Access the features which correspond to a character.

OpticalFlowMatch ............................................... P.127
Calculating the Displacement vector field by correlation methods.

PaintGray .......................................................... P.361
Copy the gray values of an image into another image.

PhaseDeg .......................................................... P.100
Return the phase of a complex image in degrees.

PhaseRad .......................................................... P.100
Return the phase of a complex image in radians.

PhotStereo ......................................................... P.809
Reconstruct a surface from at least three gray value images.

Plateaus ............................................................ P.565
Detect all gray value plateaus.
PlateausCenter
Detect the centers of all gray value plateaus.

PolarTransImage
Transform an image to polar coordinates

Pouring
Segment an image by “pouring water” over it.

PowerByte
Return the power spectrum of a complex image.

PowerLn
Return the power spectrum of a complex image.

PowerReal
Return the power spectrum of a complex image.

PrewittAmp
Detect edges (amplitude) using the Prewitt operator.

PrewittDir
Detect edges (amplitude and direction) using the Prewitt operator.

PrincipalComp
Compute the principal components of multi-channel images.

RankImage
Smooth an image with an arbitrary rank mask.

RealToComplex
Convert two real images into a complex image.

Rectangle1Domain
Reduce the domain of an image to a rectangle.

ReduceDomain
Reduce the domain of an image.

RegionGrowing
Segment an image using region-growing.

RegionGrowingMean
Region-growing using mean gray values.

RegionGrowingN
Region-growing for multi-channel images.

Rgb1ToGray
Transform an RGB image into a gray scale image.

Rgb3ToGray
Transform an RGB image to a gray scale image.

Roberts
Detect edges using the Roberts filter.

RobinsonAmp
Detect edges (amplitude) using the Robinson operator.

RobinsonDir
Detect edges (amplitude and direction) using the Robinson operator.

RotateImage
Rotate an image about its center.

RunBgEsti
Estimate the background and return the foreground region.

ScaleImage
Scale the gray values of an image.

ScaleImageMax
Maximum gray value spreading in the value range 0 bis 255.

SelectGrayvaluesFromChannels
Selection of gray values of a multi-channel image using an index image.
SendImage ................................................................. P.621
Send an image over a socket connection.

SetGrayval ............................................................... P.363
Set single gray values in an image.

SfsModLr ................................................................. P.811
Reconstruct a surface from a gray value image.

SfsOrigLr ................................................................. P.812
Reconstruct a surface from a gray value image.

SfsPentland ............................................................. P.813
Reconstruct a surface from a gray value image.

ShadeHeightField ..................................................... P.814
Shade a height field.

SigmaImage ............................................................. P.158
Non-linear smoothing with the sigma filter.

SimulateDefocus ...................................................... P.168
Simulate an uniform out-of-focus blurring of an image.

SimulateMotion ....................................................... P.169
Simulation of (linearly) motion blur.

SmoothImage ........................................................ P.159
Smooth an image using recursive filters.

SobelAmp ............................................................... P.81
Detect edges (amplitude) using the Sobel operator.

SobelDir ............................................................... P.82
Detect edges (amplitude and direction) using the Sobel operator.

SubImage ............................................................. P.41
Subtract two images.

Symmetry ............................................................. P.136
Symmetry of gray values along a row.

TextureLaws ........................................................ P.163
Filter an image using a Laws texture filter.

Threshold ........................................................... P.575
Select gray values lying within an interval.

ThresholdSubPix ..................................................... P.576
Extract level crossings from an image with subpixel accuracy.

TileChannels ........................................................ P.341
Tile multiple images into a large image.

TileImages .......................................................... P.342
Tile multiple image objects into a large image.

TileImagesOffset ..................................................... P.344
Tile multiple image objects into a large image with explicit positioning information.

TopographicSketch ............................................... P.137
Compute the topographic primal sketch of an image.

TrainOcvProj ........................................................ P.804
Training of an OCV tool.

TransFromRgb ........................................................ P.49
Transform an image from the RGB color space to an arbitrary color space.

TransToRgb .......................................................... P.53
Transform an image from an arbitrary color space to the RGB color space.

TrimmedMean ........................................................ P.160
Smooth an image with an arbitrary rank mask.

UpdateBgEsti ........................................................ P.655
Change the estimated background image.
Watersheds ................................................................. P.576
Extract watersheds and basins from an image.

WienerFilter ............................................................. P.170
Image restoration by Wiener filtering.

WienerFilterNi .......................................................... P.172
Image restoration by Wiener filtering.

WriteImage ............................................................ P.14
Write images in graphic formats.

WriteOcrTrainfImage ................................................ P.800
Write characters into a training file.

ZeroCrossing ........................................................... P.577
Detect zero crossings in an image.

ZeroCrossingSubPix .................................................. P.578
Extract zero crossings from an image with subpixel accuracy.

ZoomImageFactor ................................................... P.32
Zoom an image by a given factor.

ZoomImageSize ....................................................... P.33
Zoom an image to a given size.

15.13 HInfoX

Group class for system information related functionality.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Methods

DispInfo ............................................................... P.589
Display the manual information of a procedure on the screen.

GetChapterInfo ....................................................... P.590
Get information concerning the chapters on procedures.

GetComprise .......................................................... P.224
Get the output treatment of an image matrix.

GetKeywords ........................................................ P.591
Get keywords which are assigned to procedures.

GetOperatorInfo ..................................................... P.591
Get information concerning a HALCON-procedure.

GetOperatorName .................................................... P.593
Get procedures with the given string as a substring of their name.

GetParamInfo ....................................................... P.593
Get information concerning the procedure parameters.

GetParamNames ....................................................... P.595
Get the names of the parameters of a HALCON-procedure.

GetParamNum ........................................................ P.596
Get number of the different parameter classes of a HALCON-procedure.
15.14 HMeasureX

Represents an instance of a tool to measure distances.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

- **GenMeasureArc** P.775
  Prepare the extraction of straight edges perpendicular to an annular arc.

- **GenMeasureRectangle2** P.777
  Prepare the extraction of straight edges perpendicular to a rectangle.

- **MeasurePairs** P.779
  Extract straight edge pairs perpendicular to a rectangle.

- **MeasurePos** P.781
  Extract straight edges perpendicular to a rectangle.

- **MeasureProjection** P.783
  Extract a gray value profile perpendicular to a rectangle.

- **MeasureThresh** P.784
  Extracting points with a particular grey value along a rectangle.

15.15 HMiscX

Group class for methods belonging to no other HALCON class.
Class Description

Number of Methods ......................................................... 52
Number of Properties .................................................... 0
Abstract Class .............................................................. no
Base Class(es) .............................................................. <none>
Automatic Destructor .................................................... no
Object Identity (through internal handle) ......................... no

Methods

- AbsInvarFourierCoeff .................................................. P.721
  Normalizing of the Fourier coefficients with respect to the displacement of the starting point.
- AngleLl ................................................................. P.737
  Calculate the angle between two lines.
- AngleLx ................................................................. P.738
  Calculate the angle between one line and the vertical axis.
- ApproxChain .......................................................... P.367
  Approximate a contour by arcs and lines.
- ApproxChainSimple .................................................. P.370
  Approximate a contour by arcs and lines.
- CaltabPoints .......................................................... P.676
  Read the mark center points from the calibration table description file.
- ChangeRadialDistortionCamPar ................................. P.682
  Determine new camera parameters in accordance to the specified radial distortion.
- CloseAllBgEsti ....................................................... P.645
  Delete all background estimation data sets.
- CloseAllClassBox .................................................... P.1
  Destroy all classifiers.
- CloseAllFiles ........................................................ P.18
  Close all open files.
- CloseAllFramegrabbers ............................................. P.345
  Close all frame grabbers.
- CloseAllOcrs ......................................................... P.786
  Destroy all OCR classifiers.
- CloseAllOcvss ....................................................... P.801
  Clear all OCV tools.
- CloseAllSerials ...................................................... P.611
  Close all serial devices.
- ConcatOcrTrainf ..................................................... P.787
  Concatenation of trainings files.
- CreateCaltab ........................................................ P.687
  Generate calibration table description file and corresponding PostScript file.
- DistancePl .......................................................... P.740
  Calculate the distance between one point and one line.
- DistancePp .......................................................... P.741
  Calculate the distance between two points.
- DistancePs .......................................................... P.744
  Calculate the distances between a point and a line segment.
- DistanceSl .......................................................... P.747
  Calculate the distances between one line segment and one line.
DistanceSs ................................................. P.749
Calculate the distances between two line segments.

FileExists .................................................. P.15
Check whether file exists.

FilterKalman ............................................. P.758
Estimate the current state of a system with the help of the Kalman filtering.

Fourier1Dim .............................................. P.722
Calculate the Fourier coefficients of a parameterized contour.

Fourier1DimInv ......................................... P.722
One dimensional Fourier synthesis (inverse Fourier transform).

GaussDistribution ................................. P.139
Generate a Gaussian noise distribution.

GetLineOfSight .......................................... P.700
Compute the line of sight corresponding to a point in the image.

GetPointsEllipse ........................................ P.750
Points of an ellipse corresponding to specific angles.

ImagePointsToWorldPlane ......................... P.709
Transform image points to a world plane represented by external camera parameters.

InfoEdges ............................................... P.70
Estimate the width of a filter in EdgesImage.

InfoSmooth ............................................... P.148
Information on smoothing filter SmoothImage.

IntersectionLl ........................................... P.751
Calculate the intersection point of two lines.

InvarFourierCoeff ..................................... P.723
Normalize the Fourier coefficients.

LineOrientation ........................................ P.371
Calculate the orientation of lines.

LinePosition ............................................. P.372
Calculate the center of gravity, length, and orientation of a line.

MatchFourierCoeff .................................... P.724
Similarity of two contours.

MoveContourOrig ...................................... P.725
Transformation of the origin into the centre of gravity.

PartitionLines .......................................... P.373
Partition lines according to various criteria.

PrepContourFourier .................................. P.725
Parameterize the passed contour.

Project3DPoint ......................................... P.711
Project 3D points into (sub-)pixels.

ProjectionPl ............................................. P.753
Calculate the projection of a point onto a line.

ReadCamPar ............................................ P.712
Read the internal camera parameters from text file.

ReadKalman ............................................ P.762
Read the description file of a Kalman filter.

ReadOcrTrainfNames ............................. P.795
Query which characters are stored in a training file.

ReadTuple ............................................... P.22
Read a tuple from a file.

SelectLines ............................................. P.375
Select lines according to various criteria.
### 15.16 HObjectX

Instance of an iconic object. Base class for HImage, HRegion,...

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenEmptyObj</td>
</tr>
<tr>
<td>IntegerToObj</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConcatObj</td>
</tr>
<tr>
<td>CopyObj</td>
</tr>
<tr>
<td>CountObj</td>
</tr>
<tr>
<td>DispObj</td>
</tr>
<tr>
<td>GetChannelInfo</td>
</tr>
<tr>
<td>GetObjClass</td>
</tr>
<tr>
<td>ObjToInteger</td>
</tr>
<tr>
<td>SelectObj</td>
</tr>
<tr>
<td>TestEqualObj</td>
</tr>
<tr>
<td>TestObjDef</td>
</tr>
</tbody>
</table>
15.17  HOCRX

Instance of an OCR classifier.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Constructor(s)

- CreateOcrClassBox ........................................... P.787
  Creating a new OCR-classifier.

- ReadOcr ..................................................... P.794
  Reading of an OCR classifier from a file.

Methods

- DoOcrMulti ................................................. P.790
  Classification of characters.

- DoOcrSingle ................................................. P.790
  Classification of a character.

- InfoOcrClassBox ........................................... P.791
  Information about an OCR classifier.

- OcrChangeChar ............................................. P.792
  Defining a new conversion table for the characters.

- OcrGetFeatures ............................................ P.793
  Access the features which correspond to a character.

- TestdOcrClassBox ......................................... P.796
  Training of an OCR classifier by the input of regions.

- TraindOcrClassBox ......................................... P.797
  Training of an OCR classifier by the input of regions.

- TrainfOcrClassBox ......................................... P.798
  Training of an OCR classifier with the help of a training file.

- WriteOcr .................................................... P.798
  Writing an OCR classifier into a file.

15.18  HOcvX

Represents an instance of a tool for optical character verification.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>
15.19 HOperatorSetX

Group class for the handling of operators.

Constructor(s)

- **CreateOcvProj**
  Create a new OCV tool based on gray value projections.
  P.801

- **ReadOcv**
  Reading an OCV tool from file.
  P.803

Methods

- **DoOcvSimple**
  Verification of a pattern using an OCV tool.
  P.802

- **TrainOcvProj**
  Training of an OCV tool.
  P.804

- **WriteOcv**
  Saving an OCV tool to file.
  P.805

Class Description

- Number of Methods: 924
- Number of Properties: 0
- Abstract Class: no
- Base Class(es): none
- Automatic Destructor: no
- Object Identity: (through internal handle) no

Methods

- **AbsFunct1D**
  Absolute value of the y values.
  P.726

- **AbsImage**
  Calculate the absolute value (modulus) of an image.
  P.34

- **AbsInvarFourierCoeff**
  Normalizing of the Fourier coefficients with respect to the displacement of the starting point.
  P.721

- **AccessChannel**
  Access a channel of a multichannel image.
  P.289

- **AdaptTemplate**
  Adapting a template to the size of an image.
  P.108

- **AddChannels**
  Add gray values to regions.
  P.314

- **AddImage**
  Add two images.
  P.34

- **AddNoiseDistribution**
  Add noise to an image.
  P.138

- **AddNoiseWhite**
  Add noise to an image.
  P.139

- **AddNoiseWhiteContourXld**
  Add noise to XLD contours.
  P.886

- **AffineTransContourXld**
  Apply an arbitrary affine transformation to an XLD contour.
  P.887

- **AffineTransImage**
  Apply an arbitrary affine transformation to an image.
  P.27
- **AffineTransPoint2D**
  Apply a homogeneous 2D transformation matrix to points.

- **AffineTransPoint3D**
  Apply a homogeneous 3D transformation matrix to points.

- **AffineTransPolygonXld**
  Apply an arbitrary affine transformation to an XLD polygon.

- **AffineTransRegion**
  Transform a region using an affine transformation.

- **AngleLl**
  Calculate the angle between two lines.

- **AngleLx**
  Calculate the angle between one line and the vertical axis.

- **AnisotropeDiff**
  Smooth an image by edge-preserving anisotropic diffusion.

- **AppendChannel**
  Append additional matrices (channels) to the image.

- **AppendOcrTrainf**
  Adding characters to a training file.

- **ApproxChain**
  Approximate a contour by arcs and lines.

- **ApproxChainSimple**
  Approximate a contour by arcs and lines.

- **AreaCenter**
  Area and center of regions.

- **AreaCenterGray**
  Compute the area and center of gravity of a region in a gray value image.

- **AreaCenterXld**
  Area and center of gravity (centroid) of contours and polygons.

- **AutoThreshold**
  Segment an image using thresholds determined from its histogram.

- **BackgroundSeg**
  Determine the connected components of the background of given regions.

- **BandpassImage**
  Edge extraction using bandpass filters.

- **BestMatch**
  Searching the best matching of a template and an image.

- **BestMatchMg**
  Suche nach dem besten Grauwert-Match in einer Pyramide.

- **BestMatchPreMg**
  Suche nach dem besten Grauwert-Match in einer vorherberechneten Pyramide.

- **BestMatchRot**
  Searching the best matching of a template and an image with rotation.

- **BestMatchRotMg**
  Searching the best matching of a template and a pyramid with rotation.

- **BinThreshold**
  Segment a black-and-white image using an automatically determined threshold.

- **BitAnd**
  Bit-by-bit AND of all pixels of the input images.

- **BitShift**
  Left shift of all pixels of the image.

- **BitMask**
  Logical “AND” of each pixel using a bit mask.
CHAPTER 15. CLASS HIERARCHY

- **BitNot**
  Complement all bits of the pixels. P.44

- **BitOr**
  Bit-by-bit OR of all pixels of the input images. P.45

- **BitRshift**
  Right shift of all pixels of the image. P.46

- **BitSlice**
  Extract a bit from the pixels. P.46

- **BitXor**
  Bit-by-bit XOR of all pixels of the input images. P.47

- **BottomHat**
  Compute the bottom hat of regions. P.389

- **Boundary**
  Reduce a region to its boundary. P.390

- **CaltabPoints**
  Read the mark center points from the calibration table description file. P.676

- **CameraCalibration**
  Determine all camera parameters by a simultaneous minimization process. P.677

- **ChangeDomain**
  Change definition domain of an image. P.314

- **ChangeFormat**
  Change image size. P.338

- **ChangeRadialDistortionCamPar**
  Determine new camera parameters in accordance to the specified radial distortion. P.682

- **ChangeRadialDistortionContoursXld**
  Change the radial distortion of contours. P.683

- **ChangeRadialDistortionImage**
  Change the radial distortion of an image. P.684

- **ChannelsToImage**
  Convert one-channel images into a multichannel image. P.290

- **CharThreshold**
  Perform a threshold segmentation for extracting characters. P.542

- **CheckDifference**
  Compare two images pixel by pixel. P.543

- **CheckParHwPotential**
  Check hardware regarding its potential for parallel processing. P.600

- **Circularity**
  Shape factor for the circularity (similarity to a circle) of a region. P.478

- **Class2DimSup**
  Segment an image using two-dimensional pixel classification. P.545

- **Class2DimUnsup**
  Segment two images by clustering. P.546

- **ClassNdimBox**
  Classify pixels using hyper-cuboids. P.547

- **ClassNdimNorm**
  Classify pixels using hyper-spheres or hyper-cubes. P.548

- **ClearObj**
  Delete an iconic object from the HALCON database. P.443

- **ClearRectangle**
  Delete a rectangle on the output window. P.265

- **ClearSampset**
  Free memory of a data set. P.1

- **ClearSerial** .............................................................. P.610
  Clear the buffer of a serial connection.

- **ClearShapeModel** .................................................. P.768
  Free the memory of a shape model.

- **ClearTemplate** ..................................................... P.115
  Deallocation of the memory of a template.

- **ClearWindow** ....................................................... P.266
  Delete an output window.

- **ClipContoursXld** .................................................. P.889
  Clip an XLD contour.

- **ClipRegion** ........................................................ P.519
  Clip a region to a rectangle.

- **ClipRegionRel** ..................................................... P.520
  Clip a region relative to its size.

- **CloseAllBgEsti** ................................................... P.645
  Delete all background estimation data sets.

- **CloseAllClassBox** ................................................ P.1
  Destroy all classifiers.

- **CloseAllFiles** ..................................................... P.18
  Close all open files.

- **CloseAllFramegrabbers** ........................................ P.345
  Close all frame grabbers.

- **CloseAllOcrs** .................................................... P.786
  Destroy all OCR classificators.

- **CloseAllO cvs** .................................................... P.801
  Clear all OCV tools.

- **CloseAllSerials** .................................................. P.611
  Close all serial devices.

- **CloseBgEsti** ...................................................... P.646
  Delete the background estimation data set.

- **CloseClassBox** .................................................... P.2
  Destroy the classificator.

- **CloseEdges** ........................................................ P.56
  Close edge gaps using the edge amplitude image.

- **CloseEdgesLength** ................................................ P.57
  Close edge gaps using the edge amplitude image.

- **CloseFile** ........................................................ P.18
  Closing a text file.

- **CloseFramegrabber** .............................................. P.346
  Close specified frame grabber.

- **CloseMeasure** ..................................................... P.775
  Delete a measure object.

- **CloseOcr** .......................................................... P.786
  Deallocation of the memory of an OCR classifier.

- **CloseOcv** .......................................................... P.801
  Clear an OCV tool.

- **CloseSerial** ........................................................ P.611
  Close a serial device.

- **CloseSocket** ........................................................ P.616
  Close a socket.

- **CloseWindow** ....................................................... P.267
  Close an output window.
Closing ................................................................. P.390
Close a region.

ClosingCircle ....................................................... P.391
Close a region with a circular structuring element.

ClosingGolay ....................................................... P.392
Close a region with an element from the Golay alphabet.

ClosingRectangle .................................................. P.393
Close a region with a rectangular structuring element.

CombineRoadsXld .................................................. P.890
Combine road hypotheses from two resolution levels.

Compactness ........................................................ P.479
Shape factor for the compactness of a region.

Complement .......................................................... P.514
Return the complement of a region.

ComplexToReal ..................................................... P.364
Convert a complex image into two real images.

Compose2 ............................................................. P.290
Convert two images into a two-channel image.

Compose3 ............................................................. P.291
Convert 3 images into a three-channel image.

Compose4 ............................................................. P.292
Convert 4 images into a four-channel image.

Compose5 ............................................................. P.292
Convert 5 images into a five-channel image.

Compose6 ............................................................. P.293
Convert 6 images into a six-channel image.

Compose7 ............................................................. P.294
Convert 7 images into a seven-channel image.

ConcatObj ............................................................ P.443
Concatenate two iconic object tuples.

ConcatOcrTrainf .................................................. P.787
Concatenation of training files.

ConnectAndHoles ................................................ P.479
Number of connection components and holes

Connection .......................................................... P.521
Compute connected components of a region.

Contlength .......................................................... P.480
Contour length of a region.

ContPointNumXld .................................................. P.867
Return the number of points in an XLD contour.

ContourToWorldPlaneXld ...................................... P.684
Transform an XLD contour to a world plane represented by external camera parameters.

ConvertImageType ................................................ P.364
Convert the type of an image.

ConvertPoseType .................................................. P.685
Change the representation type of 3D pose parameters.

Convexity ............................................................. P.481
Shape factor for the convexity of a region.

ConvolFft ............................................................ P.88
Convolve an image with a byte-mask in the frequency domain.

ConvolGabor ......................................................... P.89
Convolve an image with a Gabor filter in the frequency domain.
Convolve an image with an arbitrary filter mask.

Calculate a co-occurrence matrix and derive gray value features thereof.

Calculate gray value features from a co-occurrence matrix.

Copy an image and allocate new memory for it.

Copy an iconic object in the HALCON database.

Copy all pixels within rectangles between output windows.

Searching corners in images.

Count channels of image.

Number of objects in a tuple.

Number of entries in the HALCON database.

Elapsed processing time since the last call of CountSeconds.

Generate and initialize a data set for the background estimation.

Generate calibration table description file and corresponding PostScript file.

Create a new classifier.

Create a function from a sequence of y-values.

Create a function from a set of (x,y) pairs.

Creating a new OCR-classifier.

Create a new OCV tool based on gray value projections.

Create 3D pose parameters.

Prepare a shape model for matching.

Preparing a pattern for template matching.

Preparing a pattern for template matching with rotation.

Cut out of defined gray values.

Cut out an image area relative to the domain.

Cut out a rectangular image area.

Cut out a rectangular image area.
- **Decode1DBarCode**
  Decoding of a sequence of elements of a barcode.
  - P.656

- **Decode2DBarCode**
  Decode 2D bar code data.
  - P.657

- **Decompose2**
  Convert a two-channel image into two images.
  - P.295

- **Decompose3**
  Convert a three-channel image into three images.
  - P.295

- **Decompose4**
  Convert a four-channel image into four images.
  - P.296

- **Decompose5**
  Convert a five-channel image into five images.
  - P.297

- **Decompose6**
  Convert a six-channel image into six images.
  - P.297

- **Decompose7**
  Convert a seven-channel image into seven images.
  - P.298

- **DepthFromFocus**
  Extract depth using multiple focus levels.
  - P.806

- **DerivateGauss**
  Convolves an image with derivatives of the Gaussian.
  - P.58

- **DescriptClassBox**
  Description of the classificator.
  - P.2

- **DetectEdgeSegments**
  Detect straight edge segments.
  - P.549

- **DeviationImage**
  Calculate the standard deviation of gray values within rectangular windows.
  - P.162

- **DiameterRegion**
  Maximal distance between two boundary points of a region.
  - P.482

- **DiffOfGauss**
  Approximate the LoG operator (Laplace of Gaussian).
  - P.61

- **Difference**
  Calculate the difference of two regions.
  - P.515

- **Dilation1**
  Dilate a region.
  - P.394

- **Dilation2**
  Dilate a region (using a reference point).
  - P.396

- **DilationCircle**
  Dilate a region with a circular structuring element.
  - P.397

- **DilationGolay**
  Dilate a region with an element from the Golay alphabet.
  - P.398

- **DilationRectangle1**
  Dilate a region with a rectangular structuring element.
  - P.399

- **DilationSeq**
  Dilate a region sequentially.
  - P.400

- **DiscretelDBarCode**
  Generate a discrete barcode from the elements widths.
  - P.658

- **DispArc**
  Displays circular arcs in a window.
  - P.209

- **DispArrow**
  Displays arrows in a window.
  - P.210

- **DispCaltab**
  Project and visualize the 3D model of the calibration table in the image.
  - P.695
\begin{itemize}
  \item \textbf{DispChannel} \hspace{1cm} \textbf{P.211}
  Displays images with several channels.
  
  \item \textbf{DispCircle} \hspace{1cm} \textbf{P.212}
  Displays circles in a window.
  
  \item \textbf{DispColor} \hspace{1cm} \textbf{P.214}
  Displays a color (RGB) image.
  
  \item \textbf{DispDistribution} \hspace{1cm} \textbf{P.214}
  Displays a noise distribution.
  
  \item \textbf{DispEllipse} \hspace{1cm} \textbf{P.215}
  Displays ellipses.
  
  \item \textbf{DispImage} \hspace{1cm} \textbf{P.217}
  Displays gray value images.
  
  \item \textbf{DispInfo} \hspace{1cm} \textbf{P.589}
  Displays the manual information of a procedure on the screen.
  
  \item \textbf{DispLine} \hspace{1cm} \textbf{P.218}
  Draws lines in a window.
  
  \item \textbf{DispLut} \hspace{1cm} \textbf{P.197}
  Graphical view of the look-up-table (lut).
  
  \item \textbf{DispObj} \hspace{1cm} \textbf{P.219}
  Displays image objects (image, region, XLD).
  
  \item \textbf{DispPolygon} \hspace{1cm} \textbf{P.220}
  Displays a polyline.
  
  \item \textbf{DispRectangle1} \hspace{1cm} \textbf{P.220}
  Display of rectangles aligned to the coordinate axes.
  
  \item \textbf{DispRectangle2} \hspace{1cm} \textbf{P.222}
  Displays arbitrarily oriented rectangles.
  
  \item \textbf{DispRegion} \hspace{1cm} \textbf{P.223}
  Displays regions in a window.
  
  \item \textbf{DispXld} \hspace{1cm} \textbf{P.224}
  Display an XLD object.
  
  \item \textbf{DistEllipseContourXld} \hspace{1cm} \textbf{P.868}
  Distance of contours to an ellipse.
  
  \item \textbf{DistanceFunct1D} \hspace{1cm} \textbf{P.728}
  Compute the distance of two functions.
  
  \item \textbf{DistanceLr} \hspace{1cm} \textbf{P.739}
  Calculate the distance between one line and one region.
  
  \item \textbf{DistancePl} \hspace{1cm} \textbf{P.740}
  Calculate the distance between one point and one line.
  
  \item \textbf{DistancePp} \hspace{1cm} \textbf{P.741}
  Calculate the distance between two points.
  
  \item \textbf{DistancePr} \hspace{1cm} \textbf{P.743}
  Calculate the distance between one point and one region.
  
  \item \textbf{DistancePs} \hspace{1cm} \textbf{P.744}
  Calculate the distances between a point and a line segment.
  
  \item \textbf{DistanceRrMin} \hspace{1cm} \textbf{P.745}
  Minimum distance between the contour pixels of two regions each.
  
  \item \textbf{DistanceRrMinDil} \hspace{1cm} \textbf{P.746}
  Minimum distance between two regions with the help of dilatation.
  
  \item \textbf{DistanceS1} \hspace{1cm} \textbf{P.747}
  Calculate the distances between one line segment and one line.
  
  \item \textbf{DistanceSr} \hspace{1cm} \textbf{P.748}
  Calculate the distance between one line segment and one region.
\end{itemize}
DistanceSs ................................................................. P.749
Calculate the distances between two line segments.

DistanceTransform .................................................. P.523
Compute the distance transformation of a region.

DivImage ........................................................................ P.35
Divide two images.

DoOcrMulti ................................................................. P.790
Classification of characters.

DoOcrSingle ............................................................... P.790
Classification of a character.

DoOcvSimple ............................................................. P.802
Verification of a pattern using an OCV tool.

DragRegion1 ............................................................... P.175
Interactive moving of a region.

DragRegion2 ............................................................... P.176
Interactive movement of a region with fixpoint specification.

DragRegion3 ............................................................... P.177
Interactive movement of a region with restriction of positions.

DrawCircle ................................................................. P.178
Interactive drawing of a circle.

DrawCircleMod .......................................................... P.179
Interactive drawing of a circle.

DrawEllipse ................................................................. P.180
Interactive drawing of an ellipse.

DrawEllipseMod ........................................................ P.181
Interactive drawing of an ellipse.

DrawLine ................................................................. P.182
Draw a line.

DrawLineMod ............................................................. P.183
Draw a line.

DrawLut ................................................................. P.198
Manipulate look-up-table (lut) interactively.

DualRank ................................................................. P.379
Opening, Median and Closing with circle or rectangle mask.

DualThreshold .......................................................... P.550
Threshold operator for signed images.
- **DumpWindow**
  Write the window content to a file.

- **DvfToHomMat2d**
  Approximate an affine map from a displacement vector field.

- **DvfToInt1**
  Convert a displacement vector field into two int1-images.

- **DynThreshold**
  Segment an image using a local threshold.

- **Eccentricity**
  Shape features derived from the ellipse parameters.

- **EdgesImage**
  Extract edges using Deriche, Lanser, Shen, or Canny filters.

- **EdgesSubPix**
  Extract sub-pixel precise edges using Deriche, Lanser, Shen, or Canny filters.

- **EliminateMinMax**
  Smooth an image in the spatial domain to suppress noise.

- **EliminateRuns**
  Eliminate runs of a given length.

- **EliminateSp**
  Replace values outside of thresholds with average value.

- **EllipticAxis**
  Parameters of the equivalent ellipse.

- **EllipticAxisGray**
  Compute the orientation and major axes of a region in a gray value image.

- **Emphasize**
  Enhance contrast of the image.

- **EnergyGabor**
  Calculate the energy of a two-channel image.

- **EnquireClassBox**
  Classify a tuple of attributes.

- **EnquireRejectClassBox**
  Classify a tuple of attributes with rejection class.

- **EntropyGray**
  Determine the entropy and anisotropy of images.

- **EntropyImage**
  Calculate the entropy of gray values within a rectangular window.

- **EquiHistImage**
  Histogram linearisation of images

- **Erosion1**
  Erode a region.

- **Erosion2**
  Erode a region (using a reference point).

- **ErosionCircle**
  Erode a region with a circular structuring element.

- **ErosionGolay**
  Erode a region with an element from the Golay alphabet.

- **ErosionRectangle**
  Erode a region with a rectangular structuring element.

- **ErosionSeq**
  Erode a region sequentially.

- **EstimateAlAm**
  Estimate the albedo of a surface and the amount of ambient light.
Estimate the slant of a light source and the albedo of a surface.

Estimate the slant of a light source and the albedo of a surface.

Estimate the tilt of a light source.

Estimate the tilt of a light source.

Estimate the slant of a light source and the albedo of a surface.

Estimate the tilt of a light source.

Calculate the Euler number.

Matching of a template and an image.

Matching a template and an image in a resolution pyramid.

Expand the domain of an image and set the gray values in the expanded domain.

Fill gaps between regions (depending on gray value or color) or split overlapping regions.

Fill gaps between regions (depending on gray value or color) or split overlapping regions.

Expand a region starting at a given line.

Fill gaps between regions or split overlapping regions.

Searching all good matches of a template and an image.

Searching all good grayvalue matches in a pyramid.

Fast selection of grayvalues within a given gray intervall.

Compute the fast Fourier transform of an image.

Compute the fast Fourier transform of an image.

Compute the inverse fast Fourier transform of an image.

Check whether file exists.

Expansion of a displacement vector field onto undefined areas.

Interpolate 2 video half images.

Fill up holes in regions.

Fill up holes in regions having given shape features.

Estimate the current state of a system with the help of the Kalman filtering.

Look for one barcode in an image.

Look for multiple barcode regions in an image.
Find2DBarCode ................................................. P.663
Find regions that might contain a 2D bar code.

FindCaltab .................................................. P.697
Segment the calibration table region in the image.

FindMarksAndPose ........................................ P.698
Extract the 2D calibration marks from the video image and calculate initial values for the external camera parameters.

FindNeighbors .............................................. P.485
Search direct neighbors.

FindShapeModel .......................................... P.771
Find the best matches of a shape model in an image.

FitCircleContourXld ................................. P.870
Approximation of XLD contours by circles.

FitEllipseContourXld ................................. P.871
Approximation of XLD contours by ellipses or elliptic arcs.

FitLineContourXld ....................................... P.874
Approximation of XLD contours by line segments.

FitSurfaceFirstOrder ................................. P.322
Calculate gray value moments and approximation by a first order surface (plane).

FitSurfaceSecondOrder .............................. P.323
Calculate gray value moments and approximation by a second order surface.

Fitting ..................................................... P.408
Perform a closing after an opening with multiple structuring elements.

FnewLine .................................................. P.19
Create a line feed.

Fourier1Dim .............................................. P.722
Calculate the Fourier coefficients of a parameterized contour.

Fourier1DimInv ......................................... P.722
One dimensional Fourier synthesis (inverse Fourier transform).

FreadChar ............................................... P.19
Read a character from a text file.

FreadString ............................................. P.20
Read strings from a text file.

FreiAmp .................................................. P.67
Detect edges (amplitude) using the Frei-Chen operator.

FreiDir .................................................. P.68
Detect edges (amplitude and direction) using the Frei-Chen operator.

FullDomain .............................................. P.315
Expand the domain of an image to maximum.

Funct1DToPairs ....................................... P.728
Access to the x/y values of a function.

FuzzyEntropy .......................................... P.325
Determine the fuzzy entropy of regions.

FuzzyPerimeter ......................................... P.326
Calculate the fuzzy perimeter of a region.

fwriteString ............................................ P.21
Write values in a text file.

GaussDistribution ................................. P.139
Generate a Gaussian noise distribution.

GaussImage ............................................ P.147
Smooth using discrete gaussian functions.
CHAPTER 15. CLASS HIERARCHY

- Gen1DBarCodeDescr .......................................................... P.666
  Generate a description of a 1D bar code.
- Gen1DBarCodeDescrGen ..................................................... P.667
  Generate a generic description of a 1D bar code.
- Gen2DBarCodeDescr .......................................................... P.668
  Generate a generic description of a 2D bar code class.
- GenBandfilter ................................................................. P.93
  Generate an ideal band filter.
- GenBandpass ................................................................. P.94
  Generate an ideal bandpass filter.
- GenCheckerRegion .......................................................... P.458
  Create a checkered region.
- GenCircle ........................................................................ P.460
  Create a circle.
- GenContourPolygonXld ..................................................... P.860
  Generate an XLDF contour from a polygon (given as tuples).
- GenContourRegionXld ...................................................... P.860
  Generate XLDF contours from regions.
- GenContoursSkeletonXld .................................................. P.861
  Convert a skeleton into XLDF contours.
- GenCoocMatrix .............................................................. P.327
  Calculate the co-occurrence matrix of a region in an image.
- GenDiscSe ........................................................................ P.381
  Generate ellipsoidal structuring elements for gray morphology.
- GenEllipse ........................................................................ P.461
  Create an ellipse.
- GenEllipseContourXld ..................................................... P.863
  Creation of an XLDF contour corresponding to an elliptic arc.
- GenEmptyObj .................................................................... P.445
  Create an empty object tuple.
- GenEmptyRegion ............................................................. P.462
  Create an empty region.
- GenFilterMask ............................................................... P.94
  Store a filter mask in the spatial domain as a real-image.
- GenGabor .......................................................................... P.95
  Generate a Gabor filter.
- GenGaussPyramid ............................................................ P.125
  Calculating a Gauss pyramid.
- GenGridRegion ............................................................... P.463
  Create a region from lines or pixels.
- GenHighpass ................................................................. P.97
  Generate an ideal highpass filter.
- GenImage1 ........................................................................ P.300
  Create an image from a pointer to the pixels.
- GenImage1Extern ............................................................. P.301
  Create an image from a pointer on the pixels with storage management.
- GenImage1Rect ............................................................... P.302
  Create an image with a rectangular domain from a pointer on the pixels (with storage management).
- GenImage3 .......................................................................... P.304
  Create an image from three pointers to the pixels (red/green/blue).
- GenImageConst ............................................................... P.305
  Create an image with constant gray value.
GenImageGrayRamp ................................................................. P.306
Create a gray value ramp.

GenImageProto ................................................................. P.307
Set the gray values of an image to a specified value.

GenImageSurfaceSecondOrder ........................................... P.308
Create a curved gray surface with second order polynomial.

GenLowpass ........................................................................ P.97
Generate an ideal lowpass filter.

GenMeasureArc ..................................................................... P.775
Prepare the extraction of straight edges perpendicular to an annular arc.

GenMeasureRectangle2 ......................................................... P.777
Prepare the extraction of straight edges perpendicular to a rectangle.

GenParallelContourXld ......................................................... P.891
Compute the parallel contour of an XLD contour.

GenParallelsXld .................................................................... P.864
Extract parallel XLD polygons.

GenPolygonsXld ..................................................................... P.865
Approximate XLD contours by polygons.

GenPsfDefocus .................................................................... P.165
Generate an impulse response of an uniform out-of-focus blurring.

GenPsfMotion ...................................................................... P.166
Generate an impulse response of a (linearly) motion blurring.

GenRandomRegion ............................................................... P.464
Create a random region.

GenRandomRegions ................................................................ P.465
Create random regions like circles, rectangles and ellipses.

GenRectangle1 ...................................................................... P.467
Create a rectangle parallel to the coordinate axes.

GenRectangle2 ...................................................................... P.468
Create a rectangle of any orientation.

GenRegionHisto ..................................................................... P.470
Convert a histogram into a region.

GenRegionHline .................................................................... P.470
Store input lines described in Hesse normal shape as regions.

GenRegionLine ...................................................................... P.471
Store input lines as regions.

GenRegionPoints .................................................................... P.472
Store individual pixels as image region.

GenRegionPolygon ............................................................... P.473
Store a polygon as an image object.

GenRegionPolygonFilled ....................................................... P.474
Store a polygon as a “filled” region.

GenRegionRuns ...................................................................... P.475
Create an image region from a runlength coding.

GenSinBandpass .................................................................... P.98
Generate a bandpass filter with sinusoidal shape.

GenStdBandpass ..................................................................... P.99
Generate a bandpass filter with Gaussian or sinusoidal shape.

GenStructElements .................................................................. P.408
Generate standard structuring elements.

Get1DBarCode ...................................................................... P.670
Extract the widths of the elements inside a barcode region.
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get2DBarCode</td>
<td>671</td>
</tr>
<tr>
<td>Extract the values of the data elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”).</td>
<td></td>
</tr>
<tr>
<td>Get2DBarCodePos</td>
<td>675</td>
</tr>
<tr>
<td>Extract the data values of the elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”) and their positions in the image.</td>
<td></td>
</tr>
<tr>
<td>GetBgEstiParams</td>
<td>649</td>
</tr>
<tr>
<td>Return the parameters of the data set.</td>
<td></td>
</tr>
<tr>
<td>GetChannelInfo</td>
<td>439</td>
</tr>
<tr>
<td>Information about the components of an image object.</td>
<td></td>
</tr>
<tr>
<td>GetChapterInfo</td>
<td>590</td>
</tr>
<tr>
<td>Get information concerning the chapters on procedures.</td>
<td></td>
</tr>
<tr>
<td>GetCheck</td>
<td>584</td>
</tr>
<tr>
<td>State of the HALCON control modes.</td>
<td></td>
</tr>
<tr>
<td>GetClassBoxParam</td>
<td>4</td>
</tr>
<tr>
<td>Get information about the current parameter.</td>
<td></td>
</tr>
<tr>
<td>GetComprise</td>
<td>224</td>
</tr>
<tr>
<td>Get the output treatment of an image matrix.</td>
<td></td>
</tr>
<tr>
<td>GetContourAngleXld</td>
<td>876</td>
</tr>
<tr>
<td>Calculate the direction of an XLD contour for each contour point.</td>
<td></td>
</tr>
<tr>
<td>GetContourAttribXld</td>
<td>877</td>
</tr>
<tr>
<td>Return point attribute values of an XLD contour.</td>
<td></td>
</tr>
<tr>
<td>GetContourGlobalAttribXld</td>
<td>877</td>
</tr>
<tr>
<td>Return global attributes values of an XLD contour.</td>
<td></td>
</tr>
<tr>
<td>GetContourXld</td>
<td>857</td>
</tr>
<tr>
<td>Return the coordinates of an XLD contour.</td>
<td></td>
</tr>
<tr>
<td>GetDomain</td>
<td>315</td>
</tr>
<tr>
<td>Get the domain of an image.</td>
<td></td>
</tr>
<tr>
<td>GetDraw</td>
<td>225</td>
</tr>
<tr>
<td>Get the current region fill mode.</td>
<td></td>
</tr>
<tr>
<td>GetErrorText</td>
<td>584</td>
</tr>
<tr>
<td>Inquiry after the error text of a HALCON error number.</td>
<td></td>
</tr>
<tr>
<td>GetFix</td>
<td>225</td>
</tr>
<tr>
<td>Get mode of fixing of current look-up-table (lut).</td>
<td></td>
</tr>
<tr>
<td>GetFixedLut</td>
<td>199</td>
</tr>
<tr>
<td>Get fixing of ”‘look-up-table’” (lut) for ”‘real color images’”</td>
<td></td>
</tr>
<tr>
<td>GetFont</td>
<td>256</td>
</tr>
<tr>
<td>Get the current font.</td>
<td></td>
</tr>
<tr>
<td>GetFramegrabberLut</td>
<td>346</td>
</tr>
<tr>
<td>Query frame grabber lut.</td>
<td></td>
</tr>
<tr>
<td>GetFramegrabberParam</td>
<td>347</td>
</tr>
<tr>
<td>Get specific parameters for a frame grabber.</td>
<td></td>
</tr>
<tr>
<td>GetGrayval</td>
<td>357</td>
</tr>
<tr>
<td>Access the gray values of an image object.</td>
<td></td>
</tr>
<tr>
<td>GetHsi</td>
<td>226</td>
</tr>
<tr>
<td>Get the HSI coding of the current color.</td>
<td></td>
</tr>
<tr>
<td>GetIcon</td>
<td>227</td>
</tr>
<tr>
<td>Query the icon for region output</td>
<td></td>
</tr>
<tr>
<td>GetImagePointer1</td>
<td>358</td>
</tr>
<tr>
<td>Access the pointer of a channel.</td>
<td></td>
</tr>
</tbody>
</table>
- **GetImagePointer1Rect**
  Access to the image data pointer and the image data inside the smallest rectangle of the domain of the input image.

- **GetImagePointer3**
  Access the pointers of a colored image.

- **GetImageTime**
  Request time at which the image was created.

- **GetInsert**
  Get the current display mode.

- **GetKeywords**
  Get keywords which are assigned to procedures.

- **GetLineApprox**
  Get the current approximation error for contour display.

- **GetLineOfSight**
  Compute the line of sight corresponding to a point in the image.

- **GetLineStyle**
  Get the current graphic mode for contours.

- **GetLineWidth**
  Get the current line width for contour display.

- **GetLinesXld**
  Return an XLD polygon's data (as lines).

- **GetLut**
  Get current look-up-table (lut).

- **GetLutStyle**
  Get modification parameters of look-up-table (lut).

- **GetMbutton**
  Wait until a mouse button is pressed.

- **GetModules**
  Query of used modules and the module key.

- **GetMposition**
  Query the mouse position.

- **GetMshape**
  Query the current mouse pointer shape.

- **GetNextSocketDataType**
  Determine the HALCON data type of the next socket data.

- **GetObjClass**
  Name of the class of an image object.

- **GetOperatorInfo**
  Get information concerning a HALCON-procedure.

- **GetOperatorName**
  Get procedures with the given string as a substring of their name.

- **GetPaint**
  Get the current display mode for grayvalues.

- **GetPairFunct1D**
  Access a function value using the index of the control points.

- **GetParallelsXld**
  Return an XLD parallel's data (as lines).

- **GetParamInfo**
  Get information concerning the procedure parameters.

- **GetParamNames**
  Get the names of the parameters of a HALCON-procedure.
CHAPTER 15. CLASS HIERARCHY

- GetParamNum .................................................. P.596
  Get number of the different parameter classes of a HALCON-procedure.

- GetParamTypes ............................................. P.596
  Get default data type for the control parameters of a HALCON-procedure.

- GetPart ......................................................... P.230
  Get the image part.

- GetPartStyle .................................................. P.230
  Get the current interpolation mode for grayvalue display.

- GetPixel ......................................................... P.231
  Get the current color lookup table index.

- GetPointsEllipse ........................................... P.750
  Points of an ellipse corresponding to specific angles.

- GetPolygonXld ............................................... P.859
  Return an XLD polygon’s data.

- GetPoseType .................................................. P.701
  Get the representation type of 3D pose parameters.

- GetRegionChain ............................................. P.449
  Contour of an object as chain code.

- GetRegionContour ........................................... P.450
  Access the contour of an object.

- GetRegionConvex ............................................. P.450
  Access convex hull as contour.

- GetRegionIndex ............................................. P.486
  Index of all regions containing a given pixel.

- GetRegionPoints ............................................. P.451
  Access the pixels of a region.

- GetRegionPolygon .......................................... P.452
  Polygon approximation of a region.

- GetRegionRuns ............................................... P.453
  Access the runlength coding of a region.

- GetRegionThickness ........................................ P.487
  Access the thickness of a region along the main axis.

- GetRegressParamsXld ....................................... P.878
  Return XLD contour parameters.

- GetRgb ........................................................ P.232
  Get the current color in RGB-coding.

- GetSerialParam ............................................. P.612
  Get the parameters of a serial device.

- GetShape ....................................................... P.232
  Get the current region output shape.

- GetSpy ........................................................ P.585
  Current configuration of the HALCON debugging-tool.

- GetStringExtents .......................................... P.256
  Get the spatial size of a string.

- GetSystem ...................................................... P.603
  Information concerning the currently used HALCON system parameter.

- GetTposition ................................................ P.257
  Get cursor position.

- GetTshape ..................................................... P.258
  Get the shape of the text cursor.

- GetWindowExtents .......................................... P.270
  Information about a window’s size and position.
15.19. HOPERA TORSETX

- GetWindowPointer3
  Access to a window’s pixel data.
- GetWindowType
  Get the window type.
- GiveBgEsti
  Return the estimated background image.
- GnuplotClose
  Close all open gnuplot files or terminate an active gnuplot sub-process.
- GnuplotOpenFile
  Open a gnuplot file for visualization of images and control values.
- GnuplotOpenPipe
  Open a pipe to a gnuplot process for visualization of images and control values.
- GnuplotPlotCtrl
  Plot control values using gnuplot.
- GnuplotPlotFunct1D
  Plot a function using gnuplot.
- GnuplotPlotImage
  Visualize images using gnuplot.
- GolayElements
  Generate the structuring elements of the Golay alphabet.
- GrabImage
  Grab an image from the specified frame grabber.
- GrabImageAsync
  Grab of an image from the specified frame grabber and start of the asynchronous grab of the next image.
- GrabImageStart
  Start an asynchronous grab of an image from the specified frame grabber.
- GrabRegion
  Grab and segment an image from the specified frame grabber.
- GrabRegionAsync
  Grab and segment an image from the specified frame grabber and start the asynchronous grab of the next image.
- GrayBothat
  Perform a gray value bottom hat transformation on an image.
- GrayClosing
  Perform a gray value closing on an image.
- GrayDilation
  Perform a gray value dilation on an image.
- GrayDilationRect
  Determine the maximum gray value within a rectangle.
- GrayErosion
  Perform a gray value erosion on an image.
- GrayErosionRect
  Determine the minimum gray value within a rectangle.
- GrayHisto
  Calculate the gray value distribution.
- GrayInside
  Calculate the lowest possible gray value on an arbitrary path to the image border for each point in the image.
- GrayOpening
  Perform a gray value opening on an image.
- GrayProjections
  Calculate horizontal and vertical gray-value projections.
CHAPTER 15. CLASS HIERARCHY

- **GrayRangeRect**
  - Determine the gray value range within a rectangle.
  - P.386

- **GraySkeleton**
  - Thinning of gray value images.
  - P.134

- **GrayTophat**
  - Perform a gray value top hat transformation on an image.
  - P.387

- **HammingChangeRegion**
  - Generate a region having a given Hamming distance.
  - P.528

- **HammingDistance**
  - Hamming distance between two regions.
  - P.488

- **HammingDistanceNorm**
  - Hamming distance between two regions using normalization.
  - P.489

- **HandEyeCalibration**
  - Perform a hand-eye-calibration.
  - P.702

- **HighpassImage**
  - Extract high frequency components from an image.
  - P.69

- **Histo2Dim**
  - Calculate the histogram of two-channel gray value images.
  - P.330

- **HistoToThresh**
  - Determine gray value thresholds from a histogram.
  - P.558

- **HitOrMiss**
  - Hit-or-miss operation for regions.
  - P.413

- **HitOrMissGolay**
  - Hit-or-miss operation for regions using the Golay alphabet.
  - P.414

- **HitOrMissSeq**
  - Hit-or-miss operation for regions using the Golay alphabet (sequential).
  - P.415

- **HomMat2dCompose**
  - Compose two homogeneous 2D transformation matrices.
  - P.628

- **HomMat2dIdentity**
  - Generate the homogeneous transformation matrix of the identical 2D transformation.
  - P.628

- **HomMat2dInvert**
  - Invert a homogeneous 2D transformation matrix.
  - P.629

- **HomMat2dRotate**
  - Add a rotation to a homogeneous 2D transformation matrix.
  - P.629

- **HomMat2dScale**
  - Add a scaling to a homogeneous 2D transformation matrix.
  - P.630

- **HomMat2dSlant**
  - Add a slant to a homogeneous 2D transformation matrix.
  - P.631

- **HomMat2dToAffinePar**
  - Compute the affine transformation parameters from a homogeneous 2D transformation matrix.
  - P.632

- **HomMat2dTranslate**
  - Add a translation to a homogeneous 2D transformation matrix.
  - P.633

- **HomMat3dCompose**
  - Compose two homogeneous 3D transformation matrices.
  - P.634

- **HomMat3dIdentity**
  - Generate the homogeneous transformation matrix of the identical 3D transformation.
  - P.635

- **HomMat3dInvert**
  - Invert a homogeneous 3D transformation matrix.
  - P.636

- **HomMat3dRotate**
  - Add a rotation to a homogeneous 3D transformation matrix.
  - P.637

- **HomMat3dScale**
  - Add a scaling to a homogeneous 3D transformation matrix.
  - P.639
- **HomMat3dToPose** .......................................................... P.707
  Convert homogeneous transformation matrix into 3D pose parameter.

- **HomMat3dTranslate** ..................................................... P.640
  Add a translation to a homogeneous 3D transformation matrix.

- **HoughCircleTrans** ..................................................... P.754
  Return the Hough-Transform for circles with a given radius.

- **HoughCircles** ............................................................ P.754
  Centres of circles for a specific radius.

- **HoughLineTrans** ....................................................... P.755
  Produce the Hough transform for lines within regions.

- **HoughLines** ............................................................. P.756
  Detect lines in edge images with the help of the Hough transform and returns it in HNF.

- **HysteresisThreshold** ............................................... P.559
  Perform a hysteresis threshold operation on an image.

- **Illuminate** .............................................................. P.86
  Illuminate image.

- **ImagePointsToWorldPlane** ....................................... P.709
  Transform image points to a world plane represented by external camera parameters.

- **ImageToChannels** .................................................... P.299
  Convert a multichannel image into One-channel images.

- **InfoEdges** .............................................................. P.70
  Estimate the width of a filter in EdgesImage.

- **InfoFramegrabber** ................................................... P.352
  Information about the specified frame grabber.

- **InfoOcrClassBox** .................................................... P.791
  Information about an OCR classifier.

- **InfoParallelsXld** ................................................... P.879
  Return information about the gray values of the area enclosed by XLD parallels.

- **InfoSmooth** ........................................................... P.148
  Information on smoothing filter SmoothImage.

- **InnerCircle** .......................................................... P.490
  Largest inner circle of a region.

- **InspectShapeModel** .................................................. P.773
  Create the representation of a shape model.

- **Int1ToDvf** ............................................................. P.365
  Convert two int1-images into a displacement vector field.

- **IntegerToObj** ........................................................ P.445
  Convert an “integer number” into an iconic object.

- **IntegrateFunct1D** .................................................... P.729
  Compute the positive and negative areas of a function.

- **Intensity** .............................................................. P.330
  Calculate the mean and deviation of gray values.

- **Interjacent** ........................................................... P.529
  Partition the image plane using given regions.

- **Intersection** ........................................................ P.516
  Calculate the intersection of two regions.

- **IntersectionLl** ........................................................ P.751
  Calculate the intersection point of two lines.

- **InvarFourierCoeff** .................................................. P.723
  Normalize the Fourier coefficients.

- **InvertImage** ........................................................ P.36
  Invert an image.
- **JunctionsSkeleton**
  Find junctions and end points in a skeleton. P.530
- **KirschAmp**
  Detect edges (amplitude) using the Kirsch operator. P.71
- **KirschDir**
  Detect edges (amplitude and direction) using the Kirsch operator. P.72
- **LabelToRegion**
  Extract regions with equal gray values from an image. P.476
- **Laplace**
  <no description available> P.73
- **LaplaceOfGauss**
  LoG-Operator (Laplace of Gaussian). P.75
- **LearnClassBox**
  Train the classifier. P.5
- **LearnNdimBox**
  Train the current classifier using a multi-channel image. P.560
- **LearnNdimNorm**
  Construct clusters for ClassNdimNorm. P.561
- **LearnSampsetBox**
  Train the classifier with one data set. P.6
- **LengthXld**
  Length of contours or polygons. P.880
- **LineOrientation**
  Calculate the orientation of lines. P.371
- **LinePosition**
  Calculate the center of gravity, length, and orientation of a line. P.372
- **LinesFacet**
  Detection of lines using the facet model. P.104
- **LinesGauss**
  Detect lines and their width. P.105
- **LoadParKnowledge**
  Load knowledge about automatic parallelization from file. P.602
- **LocalMax**
  Detect all local maxima in an image. P.562
- **LocalMaxContoursXld**
  Select XLD contours with a local maximum of gray values. P.880
- **LutTrans**
  Transform an image with a gray-value look-up-table P.134
- **MatchFourierCoeff**
  Similarity of two contours. P.724
- **MatchFunct1DTrans**
  Calculate transformation parameters between two functions. P.730
- **MaxImage**
  Calculate the maximum of two images pixel by pixel. P.37
- **MaxParallelsXld**
  Join modified XLD parallels lying on the same polygon. P.881
- **MeanImage**
  Smooth by averaging. P.149
- **MeanN**
  Average gray values over several channels. P.150
- **MeanSp**
  Suppress salt and pepper noise. P.151
MeasurePairs
Extract straight edge pairs perpendicular to a rectangle.

MeasureProj
Extract a gray value profile perpendicular to a rectangle.

MeasureThresh
Extracting points with a particular grey value along a rectangle.

MedianImage
Median filtering with different rank masks.

MedianSeparate
Separated median filtering with rectangle masks.

MedianWeighted
Weighted median filtering with different rank masks.

MergeContLineScanXld
Merge XLD contours from successive line scan images.

MergeRegionsLineScan
Merge regions from line scan images.

MidrangeImage
Calculate the average of maximum and minimum inside any mask.

MinImage
Calculate the minimum of two images pixel by pixel.

MinMaxGray
Determine the minimum and maximum gray values within regions.

MinkowskiAdd1
Perform a Minkowski addition on a region.

MinkowskiAdd2
Dilate a region (using a reference point).

MinkowskiSub1
Erode a region.

MinkowskiSub2
Erode a region (using a reference point).

MirrorImage
Mirror an image.

MirrorRegion
Reflect a region about the x- or y-axis.

ModParallelsXld
Extract parallel XLD polygons enclosing a homogeneous area.

MomentsAnyXld
Arbitrary geometric moments of contours or polygons.

MomentsGrayPlane
Calculate gray value moments and approximation by a plane.

MomentsRegion2Nd
Geometric moments of regions.

MomentsRegion2NdInvar
Geometric moments of regions.

MomentsRegion2NdRelInvar
Geometric moments of regions.

MomentsRegion3Rd
Geometric moments of regions.

MomentsRegion3RdInvar
Geometric moments of regions.
CHAPTER 15. CLASS HIERARCHY

- MomentsRegionCentral ................................................................. P.496
  Geometric moments of regions.
- MomentsRegionCentralInvar ......................................................... P.497
  Geometric moments of regions.
- MomentsXld ................................................................................. P.883
  Geometric moments M20, M02, and M11 of contours or polygons.
- Monotony ..................................................................................... P.126
  Calculating the monotony operation.
- MorphHat ....................................................................................... P.421
  Compute the union of BottomHat and TopHat.
- MorphSkeleton ............................................................................... P.421
  Compute the morphological skeleton of a region.
- MorphSkiz .................................................................................. P.422
  Thinning of a region.
- MoveContourOrig ........................................................................... P.725
  Transformation of the origin into the centre of gravity.
- MoveRectangle ............................................................................... P.273
  Copy inside an output window.
- MoveRegion .................................................................................. P.456
  Translate a region.
- MultImage ..................................................................................... P.39
  Multiply two images.
- NegateFunct1D ............................................................................. P.731
  Negation of the y values.
- NewExternWindow ........................................................................ P.274
  Create a virtual graphics window under Windows NT.
- NewLine ......................................................................................... P.259
  Set the position of the text cursor to the beginning of the next line.
- NoiseDistributionMean ................................................................ P.140
  Determine the noise distribution of an image.
- NonmaxSuppressionAmp ............................................................... P.563
  Suppress non-maximum points on an edge.
- NonmaxSuppressionDir ............................................................... P.564
  Suppress non-maximum points on an edge.
- NumPointsFunct1D ....................................................................... P.731
  Number of control points of the function.
- ObjToInteger ................................................................................. P.446
  Convert an iconic object into an "integer number."
- OcrChangeChar ............................................................................ P.792
  Defining a new conversion table for the characters.
- OcrGetFeatures ............................................................................ P.793
  Access the features which correspond to a character.
- OpenFile ....................................................................................... P.21
  Open text file.
- OpenFramegrabber ........................................................................ P.354
  Open and configure a frame grabber.
- OpenSerial .................................................................................... P.613
  Open a serial device.
- OpenSocketAccept ........................................................................ P.617
  Open a socket that accepts connection requests.
- OpenSocketConnect ........................................................................ P.618
  Open a socket to an existing socket.
OpenTextwindow
Open a textual window.

OpenWindow
Open a graphics window.

Opening
Open a region.

OpeningCircle
Open a region with a circular structuring element.

OpeningGolay
Open a region with an element from the Golay alphabet.

OpeningRectangle
Open a region with a rectangular structuring element.

OpeningSeg
Separate overlapping regions.

OpticalFlowMatch
Calculating the Displacement vector field by correlation methods.

OrientationRegion
Orientation of a region.

PaintGray
Copy the gray values of an image into another image.

PaintRegion
Paint a region in an image with a constant gray value.

PartitionDynamic
Partition a region horizontally into rectangles.

PartitionLines
Partition lines according to various criteria.

PartitionRectangle
Partition a region into rectangles of equal size.

PhaseDeg
Return the phase of a complex image in degrees.

PhaseRad
Return the phase of a complex image in radians.

PhotStereo
Reconstruct a surface from at least three gray value images.

PlaneDeviation
Calculate the deviation of the gray values from the approximating image plane.

Plateaus
Detect all gray value plateaus.

PlateausCenter
Detect the centers of all gray value plateaus.

PolarTransImage
Transform an image to polar coordinates.

PoseToHomMat3d
Convert 3d pose parameters into homogeneous transformation matrix.

Pouring
Segment an image by “pouring water” over it.

PowerByte
Return the power spectrum of a complex image.

PowerLn
Return the power spectrum of a complex image.

PowerReal
Return the power spectrum of a complex image.
- **PrepContourFourier**
  Parameterize the passed contour.

- **PrewittAmp**
  Detect edges (amplitude) using the Prewitt operator.

- **PrewittDir**
  Detect edges (amplitude and direction) using the Prewitt operator.

- **PrincipalComp**
  Compute the principal components of multi-channel images.

- **Project3DPoint**
  Project 3D points into (sub-)pixels.

- **ProjectionPl**
  Calculate the projection of a point onto a line.

- **Pruning**
  Prune the branches of a region.

- **QueryAllColors**
  Query all color names.

- **QueryColor**
  Query all color names displayable in the window.

- **QueryColored**
  Query the number of colors for color output.

- **QueryContourAttribsXld**
  Return the names of the defined attributes of an XLD contour.

- **QueryContourGlobalAttribsXld**
  Return the names of the defined global attributes of an XLD contour.

- **QueryFont**
  Query the available fonts.

- **QueryGray**
  Query the displayable grayvalues.

- **QueryInsert**
  Query the possible graphic modes.

- **QueryLineWidth**
  Query the possible line widths.

- **QueryLut**
  Query all available look-up-tables (lut).

- **QueryMshape**
  Query all available mouse pointer shapes.

- **QueryOperatorInfo**
  Query slots concerning information with relation to the operator GetOperatorInfo.

- **QueryPaint**
  Query the grayvalue display modes.

- **QueryParamInfo**
  Query slots of the online-information concerning the operator GetParamInfo.

- **QueryShape**
  Query the region display modes.

- **QuerySpy**
  Inquiring for possible settings of the HALCON debugging tool.

- **QueryTshape**
  Query all shapes available for text cursors.

- **QueryWindowType**
  Query all available window types.

- **RankImage**
  Smooth an image with an arbitrary rank mask.
<table>
<thead>
<tr>
<th>Operator Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RankRegion</td>
<td>533</td>
</tr>
<tr>
<td>ReadCamPar</td>
<td>712</td>
</tr>
<tr>
<td>ReadChar</td>
<td>260</td>
</tr>
<tr>
<td>ReadClassBox</td>
<td>7</td>
</tr>
<tr>
<td>ReadContourXldArcInfo</td>
<td>23</td>
</tr>
<tr>
<td>ReadFunct1D</td>
<td>732</td>
</tr>
<tr>
<td>ReadGraySe</td>
<td>388</td>
</tr>
<tr>
<td>ReadImage</td>
<td>11</td>
</tr>
<tr>
<td>ReadKalman</td>
<td>762</td>
</tr>
<tr>
<td>ReadOcr</td>
<td>794</td>
</tr>
<tr>
<td>ReadOcrTrainf</td>
<td>794</td>
</tr>
<tr>
<td>ReadOcrTrainfNames</td>
<td>795</td>
</tr>
<tr>
<td>ReadOcrTrainfSelect</td>
<td>795</td>
</tr>
<tr>
<td>ReadOcv</td>
<td>803</td>
</tr>
<tr>
<td>ReadPolygonXldArcInfo</td>
<td>24</td>
</tr>
<tr>
<td>ReadPose</td>
<td>714</td>
</tr>
<tr>
<td>ReadRegion</td>
<td>16</td>
</tr>
<tr>
<td>ReadSampset</td>
<td>7</td>
</tr>
<tr>
<td>ReadSequence</td>
<td>12</td>
</tr>
<tr>
<td>ReadSerial</td>
<td>613</td>
</tr>
<tr>
<td>ReadShapeModel</td>
<td>774</td>
</tr>
<tr>
<td>ReadString</td>
<td>261</td>
</tr>
<tr>
<td>ReadTemplate</td>
<td>129</td>
</tr>
<tr>
<td>ReadTuple</td>
<td>22</td>
</tr>
<tr>
<td>ReadWorldFile</td>
<td>16</td>
</tr>
<tr>
<td>RealToComplex</td>
<td>366</td>
</tr>
</tbody>
</table>

*Note: Each operator has a brief description of its function.*
CHAPTER 15. CLASS HIERARCHY

- **ReceiveImage** ................................................. P.619
  Receive an image over a socket connection.

- **ReceiveRegion** .................................................. P.619
  Receive regions over a socket connection.

- **ReceiveTuple** .................................................. P.620
  Receive a tuple over a socket connection.

- **ReceiveXld** ..................................................... P.620
  Receive an XLD object over a socket connection.

- **RectangleDomain** ............................................ P.316
  Reduce the domain of an image to a rectangle.

- **ReduceDomain** ............................................... P.317
  Reduce the domain of an image.

- **RegionToBin** .................................................. P.311
  Convert a region into a binary byte-image.

- **RegionToLabel** ............................................... P.312
  Convert regions to a label image.

- **RegionToMean** ............................................... P.313
  Paint regions with their average gray value.

- **RegionGrowing** ............................................... P.567
  Segment an image using region growing.

- **RegionGrowingMean** ....................................... P.569
  Region growing using mean gray values.

- **RegionGrowingN** ............................................ P.570
  Region growing for multi-channel images.

- **RegressContoursXld** ....................................... P.893
  Calculate the parameters of a regression line to an XLD contour.

- **RemoveNoiseRegion** ....................................... P.535
  Remove noise from a region.

- **ResetObjDb** .................................................. P.583
  Initialization of the HALCON system.

- **Rgb1ToGray** .................................................. P.48
  Transform an RGB image into a gray scale image.

- **Rgb3ToGray** .................................................. P.49
  Transform an RGB image to a gray scale image.

- **Roberts** ..................................................... P.78
  Detect edges using the Roberts filter.

- **RobinsonAmp** ............................................... P.79
  Detect edges (amplitude) using the Robinson operator.

- **RobinsonDir** ............................................... P.80
  Detect edges (amplitude and direction) using the Robinson operator.

- **RotateImage** ............................................... P.31
  Rotate an image about its center.

- **Roundness** .................................................. P.498
  Shape factors from contour.

- **RunBgEsti** .................................................. P.652
  Estimate the background and return the foreground region.

- **RunlengthDistribution** ................................... P.500
  Distribution of runs needed for runlength encoding of a region.

- **RunlengthFeatures** ....................................... P.500
  Characteristic values for runlength coding of regions.

- **SampleFunct1D** ......................................... P.732
  Sample a function equidistantly in an interval.
- **ScaleImage**
  Scale the gray values of an image.

- **ScaleImageMax**
  Maximum gray value spreading in the value range $0$ bis $255$.

- **ScaleYFunct1D**
  Multiplication and addition of the $y$ values.

- **SearchOperator**
  Search names of all procedures assigned to one keyword.

- **SegmentContoursXld**
  Segment XLD contours into line segments and circular or elliptic arcs.

- **SelectContoursXld**
  Select XLD contours according to several features.

- **SelectGray**
  Select regions based on gray value features.

- **SelectGrayvaluesFromChannels**
  Selection of gray values of a multi channel image using an index image.

- **SelectLines**
  Select lines according to various criteria.

- **SelectLinesLongest**
  Select the longest input lines.

- **SelectMatchingLines**
  Select those lines from a set of lines (in HNF) which fit best into a region.

- **SelectObj**
  Select an object from an object tuple.

- **SelectRegionPoint**
  Choose all regions containing a given pixel.

- **SelectRegionSpatial**
  Pose relation of regions.

- **SelectShape**
  Choose regions with the aid of shape features.

- **SelectShapeProto**
  Choose regions having a certain relation to each other.

- **SelectShapeStd**
  Select regions of a given shape.

- **SendImage**
  Send an image over a socket connection.

- **SendRegion**
  Send regions over a socket connection.

- **SendTuple**
  Send a tuple over a socket connection.

- **SendXld**
  Send an XLD object over a socket connection.

- **SensorKalman**
  Interactive input of measurement values for a Kalman filtering.

- **SetBgEstiParam**
  Change the parameters of the data set.

- **SetCheck**
  Activating and deactivating of HALCON control modes.

- **SetClassBoxParam**
  Set system parameters for classification.

- **SetColor**
  Set output color.
SetColored.................................................................................P.239
Set multiple output colors.

SetComprise.............................................................................. P.239
Define the image matrix output clipping.

SetDraw.....................................................................................P.240
Define the region fill mode.

SetFix.........................................................................................P.241
Set fixing of "look-up-table" (lut)

SetFixedLut...............................................................................P.201
Fix "look-up-table" (lut) for "real color images".

SetFont......................................................................................P.262
Set the font used for text output.

SetFramegrabberLut....................................................................P.356
Set frame grabber lut.

SetFramegrabberParam................................................................P.357
Set specific parameters for a frame grabber.

SetGray......................................................................................P.242
Define grayvalues for region output.

SetGrayval..................................................................................P.363
Set single gray values in an image.

SetHsi..........................................................................................P.243
Define output colors (HSI-coded).

SetIcon......................................................................................P.244
Icon definition for region output.

SetInsert......................................................................................P.244
Define the pixel output function.

SetLineApprox...........................................................................P.245
Define the approximation error for contour display.

SetLineStyle................................................................................P.246
Define a contour output pattern.

SetLineWidth................................................................................P.247
Define the line width for region contour output.

SetLut..........................................................................................P.201
Set "look-up-table" (lut).

SetLutStyle..................................................................................P.204
Changing the look-up-table (lut).

SetMshape...................................................................................P.208
Set the current mouse pointer shape.

SetOffsetTemplate.......................................................................P.129
Gray value offset for template.

SetOriginPose.............................................................................P.715
Translate the origin of the external camera parameters.

SetPaint.....................................................................................P.247
Define the grayvalue output mode.

SetPart......................................................................................P.251
Modify the displayed image part.

SetPartStyle................................................................................P.252
Define an interpolation method for grayvalue output.

SetPixel.......................................................................................P.253
Define a color lookup table index.

SetReferenceTemplate..................................................................P.130
Define reference position for a matching template.
- **SetRgb**
  
  Set the color definition via RGB values.

- **SetSerialParam**

  Set the parameters of a serial device.

- **SetShape**

  Define the region output shape.

- **SetSpy**

  Control of the HALCON Debugging Tools.

- **SetSystem**

  Setting of HALCON system parameters.

- **SetTposition**

  Set the position of the text cursor.

- **SetTshape**

  Set the shape of the text cursor.

- **SetWindowAttr**

  Set window’s characteristics.

- **SetWindowDc**

  Set the device context of a virtual graphics window (Windows NT).

- **SetWindowExtents**

  Modify position and size of a window.

- **SetWindowType**

  Specify a window type.

- **SfsModLr**

  Reconstruct a surface from a gray value image.

- **SfsOrigLr**

  Reconstruct a surface from a gray value image.

- **SfsPentland**

  Reconstruct a surface from a gray value image.

- **ShadeHeightField**

  Shade a height field.

- **ShapeHistoAll**

  Determine a histogram of features along all threshold values.

- **ShapeHistoPoint**

  Determine a histogram of features along all threshold values.

- **ShapeTrans**

  Transform the shape of a region.

- **SigmaImage**

  Non-linear smoothing with the sigma filter.

- **SimCaltab**

  Simulate a video image with calibration table.

- **SimulateDefocus**

  Simulate an uniform out-of-focus blurring of an image.

- **SimulateMotion**

  Simulation of (linearly) motion blur.

- **Skeleton**

  Compute the skeleton of a region.

- **SlideImage**

  Interactive output from two window buffers.

- **SmallestCircle**

  Smallest surrounding circle of a region.

- **SmallestRectangle**

  Surrounding rectangle parallel to the coordinate axes.
SmallestRectangle2
Smallest surrounding rectangle with any orientation.

SmoothContoursXld
Smooth an XLD contour.

SmoothFunct1DGauss
Smooth an equidistant function with a Gaussian function.

SmoothFunct1DMean
Smooth a 1D function by averaging its values.

SmoothImage
Smooth an image using recursive filters.

SobelAmp
Detect edges (amplitude) using the Sobel operator.

SobelDir
Detect edges (amplitude and direction) using the Sobel operator.

SocketAcceptConnect
Accept a connection request on a listening socket.

SortRegion
Sorting of regions with respect to their relative position.

SpDistribution
Generate a salt-and-pepper noise distribution.

SpatialRelation
Pose relation of regions with regard to the coordinate axes.

SplitContoursXld
Split XLD contours at dominant points.

SplitSkeletonLines
Split lines represented by one pixel wide, non-branching lines.

SplitSkeletonRegion
Split lines represented by one pixel wide, non-branching regions.

StoreParKnowledge
Store knowledge about automatic parallelization in file.

SubImage
Subtract two images.

Symmetry
Symmetry of gray values along a row.

SystemCall
Executes a system command.

TestEqualObj
Compare image objects regarding equality.

TestEqualRegion
Test whether the regions of two objects are identical.

TestObjDef
Test whether an object is already deleted.

TestRegionPoint
Test if the region consists of the given point.

TestSampsetBox
Classify a set of arrays.

TestdOcrClassBox
Training of an OCR classifier by the input of regions.

TextureLaws
Filter an image using a Laws texture filter.

Thickening
Add the result of a hit-or-miss operation to a region.
**ThickeningGolay**
Add the result of a hit-or-miss operation to a region (using a Golay structuring element).

**ThickeningSeq**
Add the result of a hit-or-miss operation to a region (sequential).

**Thinning**
Remove the result of a hit-or-miss operation from a region.

**ThinningGolay**
Remove the result of a hit-or-miss operation from a region (using a Golay structuring element).

**ThinningSeq**
Remove the result of a hit-or-miss operation from a region (sequential).

**Threshold**
Select gray values lying within an interval.

**ThresholdSubPix**
Extract level crossings from an image with subpixel accuracy.

**TileChannels**
Tile multiple images into a large image.

**TileImages**
Tile multiple image objects into a large image.

**TileImagesOffset**
Tile multiple image objects into a large image with explicit positioning information.

**TopHat**
Compute the top hat of regions.

**TopographicSketch**
Compute the topographic primal sketch of an image.

**TrainDocrClassBox**
Training of an OCR classifier by the input of regions.

**TrainDocrProj**
Training of an OCV tool.

**TrainfOcrClassBox**
Training of an OCR classifier with the help of a training file.

**TransFromRgb**
Transform an image from the RGB color space to an arbitrary color space.

**TransToRgb**
Transform an image from an arbitrary color space to the RGB color space.

**TransformFunc1D**
Transform a function using given transformation parameters.

**TransposeRegion**
Reflect a region about a point.

**TrimmedMean**
Smooth an image with an arbitrary rank mask.

**TupleAbs**
Compute the absolute value of a tuple.

**TupleAcos**
Compute the arccosine of a tuple.

**TupleAdd**
Add two tuples.

**TupleAnd**
Compute the logical and of two tuples.

**TupleAsin**
Compute the arcsine of a tuple.

**TupleAtan**
Compute the arctangent of a tuple.
- **TupleAtan2**
  Compute the arctangent of a tuple for all four quadrants.

- **TupleBand**
  Compute the bitwise and of two tuples.

- **TupleBnot**
  Compute the bitwise not of two tuples.

- **TupleBor**
  Compute the bitwise or of two tuples.

- **TupleBxor**
  Compute the bitwise exclusive or of two tuples.

- **TupleCeil**
  Compute the ceiling function of a tuple.

- **TupleChr**
  Convert a tuple of integers into strings with the corresponding ASCII codes.

- **TupleChrt**
  Convert a tuple of integers into strings with the corresponding ASCII codes.

- **TupleConcat**
  Concatenate two tuple to a new one.

- **TupleCos**
  Compute the cosine of a tuple.

- **TupleCosh**
  Compute the hyperbolic cosine of a tuple.

- **TupleDeg**
  Convert a tuple from radians to degrees.

- **TupleDeviation**
  Return the standard deviation of the elements of a tuple.

- **TupleDiv**
  Divide two tuples.

- **TupleEnvironment**
  Read one or more environment variables.

- **TupleEqual**
  Test, whether two tuples are equal.

- **TupleExp**
  Compute the exponential of a tuple.

- **TupleFabs**
  Compute the absolute value of a tuple (as floating point numbers).

- **TupleFirstN**
  Select the first elements of a tuple.

- **TupleFloor**
  Compute the floor function of a tuple.

- **TupleFmod**
  Calculate the remainder of the floating point division of two tuples.

- **TupleGenConst**
  Generate a tuple of a specific length and initialize its elements.

- **TupleGreater**
  Test, whether a tuple is greater than another tuple.

- **TupleGreaterEqual**
  Test, whether a tuple is greater or equal to another tuple.

- **TupleInverse**
  Invert a tuple.

- **TupleIsNumber**
  Check a tuple (of strings) whether it represents numbers.
- **TupleLastN**: Select all elements from index “n” to the end of a tuple.

- **TupleLdexp**: Calculate the ldexp function of two tuples.

- **TupleLength**: Returns the number of elements of a tuple.

- **TupleLess**: Test whether a tuple is less than another tuple.

- **TupleLessEqual**: Test whether a tuple is less or equal to another tuple.

- **TupleLog**: Compute the natural logarithm of a tuple.

- **TupleLog10**: Compute the base 10 logarithm of a tuple.

- **TupleLsh**: Shift a tuple bitwise to the left.

- **TupleMax**: Returns the maximal element of a tuple.

- **TupleMean**: Return the mean value of a tuple of numbers.

- **TupleMin**: Returns the minimal element of a tuple.

- **TupleMult**: Multiply two tuples.

- **TupleNeg**: Negate a tuple.

- **TupleNot**: Compute the logical not of two tuples.

- **TupleNotEqual**: Test whether two tuples are not equal.

- **TupleNumber**: Convert a tuple (of strings) into a tuple of numbers.

- **TupleOr**: Compute the logical or of two tuples.

- **TupleOrds**: Convert a tuple of strings into a tuple of their ASCII codes.

- **TuplePow**: Calculate the power function two tuples.

- **TupleRad**: Convert a tuple from degrees to radians.

- **TupleReal**: Convert a tuple into a tuple of floating point numbers.

- **TupleRound**: Convert a tuple into a tuple of integer numbers.

- **TupleRsh**: Shift a tuple bitwise to the right.

- **TupleSelect**: Select single elements of a tuple.

- **TupleSelectRange**: Select several elements of a tuple.
 TupleSin ................................................................. P.825
Compute the sine of a tuple.

 TupleSinh ........................................................................ P.826
Compute the hyperbolic sine of a tuple.

 TupleSort ............................................................... P.840
Sorts the elements of a tuple in ascending order.

 TupleSortIndex ......................................................... P.841
Sorts the elements of a tuple and returns the indices of the sorted tuple.

 TupleSplit .............................................................. P.850
Split strings into substrings between predefined separator symbol(s).

 TupleSqrt ........................................................................ P.826
Compute the square root of a tuple.

 TupleStrBitSelect ..................................................... P.849
Select single character or bit from a tuple.

 TupleStrFirstN ......................................................... P.851
Cut the first characters up to position “n” out of string tuple.

 TupleStrLastN .......................................................... P.851
Cut all characters starting at position “n” out of string tuple.

 TupleStrchr .............................................................. P.852
Forward search for a character within a string tuple.

 TupleString .............................................................. P.838
Convert a tuple into a tuple of strings.

 TupleStrlen .............................................................. P.853
Length of every string within a tuple of strings.

 TupleStrrchr ............................................................. P.853
Backward search for a character within a string tuple.

 TupleStrrstr ............................................................. P.854
Backward search for string(s) within a string tuple.

 TupleStrstr .............................................................. P.854
Forward search for string(s) within a string tuple.

 TupleSub ................................................................. P.826
Subtract two tuples.

 TupleSum ................................................................. P.845
Return the sum of all elements of a tuple.

 TupleTan ................................................................. P.827
Compute the tangent of a tuple.

 TupleTanh ............................................................... P.827
Compute the hyperbolic tangent of a tuple.

 TupleXor ................................................................. P.846
Compute the logical exclusive or of two tuples.

 Union1 ................................................................. P.516
Return the union of all input regions.

 Union2 ................................................................. P.517
Return the union of two regions.

 UnionStraightContoursHistoXld ................................ P.897
Merge neighboring straight contours having a similar distance from a given line.

 UnionStraightContoursXld ........................................ P.899
Merge neighboring straight contours having a similar direction.

 UpdateBgEsti ......................................................... P.655
Change the estimated background image.

 UpdateKalman ......................................................... P.766
Read an update file of a Kalman filter.
15.19. HOPERATORTX

- **VectorAngleToRigid** .................................................. P.642
  Compute a rigid affine transformation from points and angles.

- **VectorToHomMat2d** .................................................. P.643
  Approximate an affine transformation from point correspondences.

- **VectorToRigid** ....................................................... P.644
  Approximate a rigid affine transformation from point correspondences.

- **VectorToSimilarity** .................................................. P.645
  Approximate an similarity transformation from point correspondences.

- **WaitSeconds** .......................................................... P.600
  Delaying the execution of the program.

- **Watersheds** ............................................................ P.576
  Extract watersheds and basins from an image.

- **WienerFilter** .......................................................... P.170
  Image restoration by Wiener filtering.

- **WienerFilterNi** ........................................................ P.172
  Image restoration by Wiener filtering.

- **WriteCamPar** .......................................................... P.717
  Write the internal camera parameters to text file.

- **WriteClassBox** ....................................................... P.10
  Save the classificator in a file.

- **WriteContourXldArcInfo** .......................................... P.25
  Write XLD contours to a file in ARC/INFO generate format.

- **WriteFunct1D** ........................................................ P.736
  Write a function to a file.

- **WriteImage** ........................................................... P.14
  Write images in graphic formats.

- **WriteLut** ............................................................... P.205
  Write look-up-table (lut) as file.

- **WriteOcr** .............................................................. P.798
  Writing an OCR classifier into a file.

- **WriteOcrTrainf** ..................................................... P.799
  Storing of trained characters into a file.

- **WriteOcrTrainfImage** ............................................. P.800
  Write characters into a training file.

- **WriteOcv** ............................................................. P.805
  Saving an OCV tool to file.

- **WritePolygonXldArcInfo** ........................................ P.26
  Write XLD polygons to a file in ARC/INFO generate format.

- **WritePose** ............................................................. P.719
  Write 3D pose data to text file.

- **WriteRegion** .......................................................... P.17
  Write regions on file.

- **WriteSerial** ........................................................... P.615
  Write to a serial connection.

- **WriteShapeModel** .................................................. P.774
  Write a shape model to a file.

- **WriteString** .......................................................... P.264
  Print text in a window.

- **WriteTemplate** ...................................................... P.130
  Writing a template to file.

- **WriteTuple** ........................................................... P.23
  Write a tuple to a file.
15.20 HPoseX

Grouping class for rigid three-dimensional transformations.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

Methods

- **CameraCalibration** ................................................................. P.677
  Determine all camera parameters by a simultaneous minimization process.
- **ContourToWorldPlaneXld** ............................................................. P.684
  Transform an XLD contour to a world plane represented by external camera parameters.
- **ConvertPoseType** ........................................................................ P.685
  Change the representation type of 3D pose parameters.
- **CreatePose** ................................................................................ P.689
  Create 3D pose parameters.
- **GetPoseType** ................................................................................ P.701
  Get the representation type of 3D pose parameters.
- **HandEyeCalibration** ....................................................................... P.702
  Perform a hand-eye-calibration.
- **PoseToHomMat3d** ........................................................................... P.710
  Convert 3d pose parameters into homogeneous transformation matrix.
- **ReadPose** ...................................................................................... P.714
  Read 3D pose data from text file.
- **SetOriginPose** ............................................................................... P.715
  Translate the origin of the external camera parameters.
- **WritePose** ..................................................................................... P.719
  Write 3D pose data to text file.
## 15.21 HRegionX

Represents an instance of a region object.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DrawPolygon</td>
</tr>
<tr>
<td>Interactive drawing of a polygon row.</td>
</tr>
<tr>
<td>DrawRegion</td>
</tr>
<tr>
<td>Interactive drawing of a closed region.</td>
</tr>
<tr>
<td>GenCheckerRegion</td>
</tr>
<tr>
<td>Create a checkered region.</td>
</tr>
<tr>
<td>GenCircle</td>
</tr>
<tr>
<td>Create a circle.</td>
</tr>
<tr>
<td>GenEllipse</td>
</tr>
<tr>
<td>Create an ellipse.</td>
</tr>
<tr>
<td>GenEmptyRegion</td>
</tr>
<tr>
<td>Create an empty region.</td>
</tr>
<tr>
<td>GenGridRegion</td>
</tr>
<tr>
<td>Create a region from lines or pixels.</td>
</tr>
<tr>
<td>GenRandomRegion</td>
</tr>
<tr>
<td>Create a random region.</td>
</tr>
<tr>
<td>GenRandomRegions</td>
</tr>
<tr>
<td>Create random regions like circles, rectangles and ellipses.</td>
</tr>
<tr>
<td>GenRectangle1</td>
</tr>
<tr>
<td>Create a rectangle parallel to the coordinate axes.</td>
</tr>
<tr>
<td>GenRectangle2</td>
</tr>
<tr>
<td>Create a rectangle of any orientation.</td>
</tr>
<tr>
<td>GenRegionHisto</td>
</tr>
<tr>
<td>Convert a histogram into a region.</td>
</tr>
<tr>
<td>GenRegionHline</td>
</tr>
<tr>
<td>Store input lines described in Hesse normal shape as regions.</td>
</tr>
<tr>
<td>GenRegionLine</td>
</tr>
<tr>
<td>Store input lines as regions.</td>
</tr>
<tr>
<td>GenRegionPoints</td>
</tr>
<tr>
<td>Store individual pixels as image region.</td>
</tr>
<tr>
<td>GenRegionPolygon</td>
</tr>
<tr>
<td>Store a polygon as an image object.</td>
</tr>
<tr>
<td>GenRegionPolygonFilled</td>
</tr>
<tr>
<td>Store a polygon as a “filled” region.</td>
</tr>
<tr>
<td>GenRegionRuns</td>
</tr>
<tr>
<td>Create an image region from a runlength coding.</td>
</tr>
<tr>
<td>GenStructElements</td>
</tr>
<tr>
<td>Generate standard structuring elements.</td>
</tr>
</tbody>
</table>
CHAPTER 15. CLASS HIERARCHY

- **GetIcon**
  Query the icon for region output
  P.227

- **GolayElements**
  Generate the structuring elements of the Golay alphabet.
  P.409

- **GrabRegion**
  Grab and segment an image from the specified frame grabber.
  P.351

- **GrabRegionAsync**
  Grab and segment an image from the specified frame grabber and start the asynchronous grab of the next image.
  P.351

- **ReadRegion**
  Read binary images or HALCON regions.
  P.16

- **ReceiveRegion**
  Receive regions over a socket connection.
  P.619

---

### Methods

- **AddChannels**
  Add gray values to regions.
  P.314

- **AffineTransRegion**
  Transform a region using an affine transformation.
  P.454

- **AppendOcrTrainf**
  Adding characters to a training file.
  P.785

- **AreaCenter**
  Area and center of regions.
  P.477

- **AreaCenterGray**
  Compute the area and center of gravity of a region in a gray value image.
  P.317

- **BackgroundSeg**
  Determine the connected components of the background of given regions.
  P.518

- **BottomHat**
  Compute the bottom hat of regions.
  P.389

- **Boundary**
  Reduce a region to its boundary.
  P.390

- **Circularity**
  Shape factor for the circularity (similarity to a circle) of a region.
  P.478

- **ClipRegion**
  Clip a region to a rectangle.
  P.519

- **ClipRegionRel**
  Clip a region relative to its size.
  P.520

- **CloseEdges**
  Close edge gaps using the edge amplitude image.
  P.56

- **CloseEdgesLength**
  Close edge gaps using the edge amplitude image.
  P.57

- **Closing**
  Close a region.
  P.390

- **ClosingCircle**
  Close a region with a circular structuring element.
  P.391

- **ClosingGolay**
  Close a region with an element from the Golay alphabet.
  P.392

- **ClosingRectangle1**
  Close a region with a rectangular structuring element.
  P.393

- **Compactness**
  Shape factor for the compactness of a region.
  P.479

- **Complement**
  Return the complement of a region.
  P.514
ConnectAndHoles................................................................. P.479
Number of connection components and holes

Connection........................................................................ P.521
Compute connected components of a region.

Contlength........................................................................ P.480
Contour length of a region.

Convexity ......................................................................... P.481
Shape factor for the convexity of a region.

CocFeatureImage............................................................. P.318
Calculate a co-occurrence matrix and derive gray value features thereof.

DiameterRegion............................................................... P.482
Maximal distance between two boundary points of a region.

Difference ....................................................................... P.515
Calculate the difference of two regions.

Dilation1........................................................................ P.394
Dilate a region.

Dilation2 ........................................................................ P.396
Dilate a region (using a reference point).

DilationCircle................................................................. P.397
Dilate a region with a circular structuring element.

DilationGolay................................................................. P.398
Dilate a region with an element from the Golay alphabet.

DilationRectangle1....................................................... P.399
Dilate a region with a rectangular structuring element.

DilationSeq...................................................................... P.400
Dilate a region sequentially.

DispRegion.................................................................. P.223
Displays regions in a window.

DistanceLr...................................................................... P.739
Calculate the distance between one line and one region.

DistancePr...................................................................... P.743
Calculate the distance between one point and one region.

DistanceRrMin............................................................... P.745
Minimum distance between the contour pixels of two regions each.

DistanceRrMinDil........................................................ P.746
Minimum distance between two regions with the help of dilatation.

DistanceSr...................................................................... P.748
Calculate the distance between one line segment and one region.

DistanceTransform....................................................... P.523
Compute the distance transformation of a region.

DoOcrMulti.................................................................... P.790
Classification of characters.

DoOcrSingle.................................................................. P.790
Classification of a character.

DragRegion1.................................................................... P.175
Interactive moving of a region.

DragRegion2.................................................................... P.176
Interactive movement of a region with fixpoint specification.

DragRegion3.................................................................... P.177
Interactive movement of a region with restriction of positions.

Eccentricity.................................................................... P.482
Shape features derived from the ellipse parameters.
 Eliminate runs of a given length.

 Eliminate runs of a given length.

 Compute the orientation and major axes of a region in a gray value image.

 Determine the entropy and anisotropy of images.

 Erode a region.

 Erode a region (using a reference point).

 Erode a region with a circular structuring element.

 Erode a region with an element from the Golay alphabet.

 Erode a region with a rectangular structuring element.

 Erode a region sequentially.

 Calculate the Euler number.

 Fill gaps between regions (depending on gray value or color) or split overlapping regions.

 Fill gaps between regions (depending on gray value or color) or split overlapping regions.

 Fill gaps between regions or split overlapping regions.

 Fill up holes in regions.

 Fill up holes in regions having given shape features.

 Search direct neighbors.

 Calculate gray value moments and approximation by a first order surface (plane).

 Calculate gray value moments and approximation by a second order surface.

 Perform a closing after an opening with multiple structuring elements.

 Determine the fuzzy entropy of regions.

 Calculate the fuzzy perimeter of a region.

 Generate XLD contours from regions.

 Convert a skeleton into XLD contours.

 Calculate the co-occurrence matrix of a region in an image.
- **Get2DBarCode**
  Extract the values of the data elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”).

- **Get2DBarCodePos**
  Extract the data values of the elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”) and their positions in the image.

- **GetRegionChain**
  Contour of an object as chain code.

- **GetRegionContour**
  Access the contour of an object.

- **GetRegionConvex**
  Access convex hull as contour.

- **GetRegionIndex**
  Index of all regions containing a given pixel.

- **GetRegionPoints**
  Access the pixels of a region.

- **GetRegionPolygon**
  Polygon approximation of a region.

- **GetRegionRuns**
  Access the runlength coding of a region.

- **GetRegionThickness**
  Access the thickness of a region along the main axis.

- **GrayHisto**
  Calculate the gray value distribution.

- **GrayProjections**
  Calculate horizontal and vertical gray-value projections.

- **HammingChangeRegion**
  Generate a region having a given Hamming distance.

- **HammingDistance**
  Hamming distance between two regions.

- **HammingDistanceNorm**
  Hamming distance between two regions using normalization.

- **Histo2Dim**
  Calculate the histogram of two-channel gray value images.

- **HitOrMiss**
  Hit-or-miss operation for regions.

- **HitOrMissGolay**
  Hit-or-miss operation for regions using the Golay alphabet.

- **HitOrMissSeq**
  Hit-or-miss operation for regions using the Golay alphabet (sequential).

- **HoughCircleTrans**
  Return the Hough-Transform for circles with a given radius.

- **HoughCircles**
  Centres of circles for a specific radius.

- **HoughLineTrans**
  Produce the Hough transform for lines within regions.

- **HoughLines**
  Detect lines in edge images with the help of the Hough transform and returns it in HNF.

- **InnerCircle**
  Largest inner circle of a region.

- **Intensity**
  Calculate the mean and deviation of gray values.
Partition the image plane using given regions.

Calculate the intersection of two regions.

Find junctions and end points in a skeleton.

Train the current classifier using a multi-channel image.

Construct clusters for ClassNdimNorm.

Merge regions from line scan images.

Determine the minimum and maximum gray values within regions.

Perform a Minkowski addition on a region.

Dilate a region (using a reference point).

Erode a region.

Erode a region (using a reference point).

Reflect a region about the x- or y-axis.

Calculate gray value moments and approximation by a plane.

Geometric moments of regions.

Geometric moments of regions.

Geometric moments of regions.

Geometric moments of regions.

Geometric moments of regions.

Geometric moments of regions.

Compute the union of BottomHat and TopHat.

Compute the morphological skeleton of a region.

Thinning of a region.

Translate a region.

Determine the noise distribution of an image.

Open a region.
OpeningCircle
Open a region with a circular structuring element.

OpeningGolay
Open a region with an element from the Golay alphabet.

OpeningRectangle
Open a region with a rectangular structuring element.

OpeningSeg
Separate overlapping regions.

OrientationRegion
Orientation of a region.

PaintRegion
Paint a region in an image with a constant gray value.

PartitionDynamic
Partition a region horizontally into rectangles.

PartitionRectangle
Partition a region into rectangles of equal size.

PlaneDeviation
Calculate the deviation of the gray values from the approximating image plane.

Pruning
Prune the branches of a region.

RankRegion
Rank operator for regions.

RegionToBin
Convert a region into a binary byte-image.

RegionToLabel
Convert regions to a label image.

RegionToMean
Paint regions with their average gray value.

RemoveNoiseRegion
Remove noise from a region.

Roundness
Shape factors from contour.

RunlengthDistribution
Distribution of runs needed for runlength encoding of a region.

RunlengthFeatures
Characteristic values for runlength coding of regions.

SelectGray
Select regions based on gray value features.

SelectMatchingLines
Select those lines from a set of lines (in HNF) which fit best into a region.

SelectRegionPoint
Choose all regions containing a given pixel.

SelectRegionSpatial
Pose relation of regions.

SelectShape
Choose regions with the aid of shape features.

SelectShapeProto
Choose regions having a certain relation to each other.

SelectShapeStd
Select regions of a given shape.

SendRegion
Send regions over a socket connection.
- `SetIcon` (P.244)
  Icon definition for region output.
- `ShapeHistoAll` (P.336)
  Determine a histogram of features along all threshold values.
- `ShapeHistoPoint` (P.337)
  Determine a histogram of features along all threshold values.
- `ShapeTrans` (P.535)
  Transform the shape of a region.
- `Skeleton` (P.536)
  Compute the skeleton of a region.
- `SmallestCircle` (P.509)
  Smallest surrounding circle of a region.
- `SmallestRectangle1` (P.510)
  Surrounding rectangle parallel to the coordinate axes.
- `SmallestRectangle2` (P.511)
  Smallest surrounding rectangle with any orientation.
- `SortRegion` (P.537)
  Sorting of regions with respect to their relative position.
- `SpatialRelation` (P.512)
  Pose relation of regions with regard to the coordinate axes.
- `SplitSkeletonLines` (P.538)
  Split lines represented by one pixel wide, non-branching lines.
- `SplitSkeletonRegion` (P.539)
  Split lines represented by one pixel wide, non-branching regions.
- `TestEqualRegion` (P.441)
  Test whether the regions of two objects are identical.
- `TestRegionPoint` (P.513)
  Test if the region consists of the given point.
- `TestOCRClassBox` (P.796)
  Training of an OCR classifier by the input of regions.
- `Thickening` (P.429)
  Add the result of a hit-or-miss operation to a region.
- `ThickeningGolay` (P.431)
  Add the result of a hit-or-miss operation to a region (using a Golay structuring element).
- `ThickeningSeq` (P.432)
  Add the result of a hit-or-miss operation to a region (sequential).
- `Thinning` (P.433)
  Remove the result of a hit-or-miss operation from a region.
- `ThinningGolay` (P.434)
  Remove the result of a hit-or-miss operation from a region (using a Golay structuring element).
- `ThinningSeq` (P.435)
  Remove the result of a hit-or-miss operation from a region (sequential).
- `TopHat` (P.436)
  Compute the top hat of regions.
- `TrainOCRClassBox` (P.797)
  Training of an OCR classifier by the input of regions.
- `TransposeRegion` (P.456)
  Reflect a region about a point.
- `Union1` (P.516)
  Return the union of all input regions.
- `Union2` (P.517)
  Return the union of two regions.
15.22 HSerialX

Represents an instance of a connection via a serial port.

---

Class Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>WriteOcrTrainf</td>
<td>P.799</td>
</tr>
<tr>
<td>WriteRegion</td>
<td>P.17</td>
</tr>
<tr>
<td>ZoomRegion</td>
<td>P.457</td>
</tr>
</tbody>
</table>

---

Constructor(s)

OpenSerial

Open a serial device.

---

Methods

- ClearSerial
  Clear the buffer of a serial connection.

- GetSerialParam
  Get the parameters of a serial device.

- ReadSerial
  Read from a serial device.

- SetSerialParam
  Set the parameters of a serial device.

- WriteSerial
  Write to a serial connection.

15.23 HShapeModelX

Represents an instance of a shape model for matching.

---

Class Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Constructor(s)

ClearShapeModel

---

Methods

- SetShapeModel
  Set the shape model.

- GetShapeModel
  Get the shape model.

- ClearShapeModel
  Clear the shape model.

- WriteShapeModel
  Write to a shape model.

- ReadShapeModel
  Read from a shape model.

- UndoShapeModel
  Undo the shape model.
15.24 HSocketX

Represents an instance of a communication via a POSIX socket.

Class Description

<table>
<thead>
<tr>
<th>Number of Methods</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>yes, through CloseSocket (not member of this class!)</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
<td>yes</td>
</tr>
</tbody>
</table>

Constructor(s)

- **OpenSocketAccept** ................................................................. P.617
  Open a socket that accepts connection requests.

- **OpenSocketConnect** .............................................................. P.618
  Open a socket to an existing socket.

Methods

- **GetNextSocketDataType** ....................................................... P.616
  Determine the HALCON data type of the next socket data.

- **ReceiveImage** ................................................................. P.619
  Receive an image over a socket connection.

- **ReceiveRegion** ............................................................... P.619
  Receive regions over a socket connection.

- **ReceiveTuple** .............................................................. P.620
  Receive a tuple over a socket connection.

- **ReceiveXld** .................................................................. P.620
  Receive an XLD object over a socket connection.

- **SendImage** ................................................................. P.621
  Send an image over a socket connection.

- **SendRegion** ............................................................... P.621
  Send regions over a socket connection.

- **SendTuple** .............................................................. P.622
  Send a tuple over a socket connection.

- **SendXld** .................................................................. P.622
  Send an XLD object over a socket connection.

- **SocketAcceptConnect** .................................................. P.623
  Accept a connection request on a listening socket.
15.25 HSystemX

Group class for system information and manipulation related functionality.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckParHwPotential</td>
</tr>
<tr>
<td>CountRelation</td>
</tr>
<tr>
<td>CountSeconds</td>
</tr>
<tr>
<td>GetCheck</td>
</tr>
<tr>
<td>GetErrorText</td>
</tr>
<tr>
<td>GetModules</td>
</tr>
<tr>
<td>GetSpy</td>
</tr>
<tr>
<td>GetSystem</td>
</tr>
<tr>
<td>LoadParKnowledge</td>
</tr>
<tr>
<td>QuerySpy</td>
</tr>
<tr>
<td>QueryWindowType</td>
</tr>
<tr>
<td>ResetObjDb</td>
</tr>
<tr>
<td>SetCheck</td>
</tr>
<tr>
<td>SetSpy</td>
</tr>
<tr>
<td>SetSystem</td>
</tr>
<tr>
<td>SetWindowAttr</td>
</tr>
<tr>
<td>SetWindowType</td>
</tr>
<tr>
<td>StoreParKnowledge</td>
</tr>
<tr>
<td>SystemCall</td>
</tr>
<tr>
<td>WaitSeconds</td>
</tr>
</tbody>
</table>

Delaying the execution of the program.
15.26 HTemplateX

Represents an instance of a template for gray value matching.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Constructor(s)

- **CreateTemplate**
  Preparing a pattern for template matching.

- **CreateTemplateRot**
  Preparing a pattern for template matching with rotation.

- **ReadTemplate**
  Reading a template from file.

Methods

- **AdaptTemplate**
  Adapting a template to the size of an image.

- **BestMatch**
  Searching the best matching of a template and an image.

- **BestMatchMg**
  Suche nach dem besten Grauwert-Match in einer Pyramide.

- **BestMatchPreMg**
  Suche nach dem besten Grauwert-Match in einer vorherberechneten Pyramide.

- **BestMatchRot**
  Searching the best matching of a template and an image with rotation.

- **BestMatchRotMg**
  Searching the best matching of a template and a pyramid with rotation.

- **FastMatch**
  Searching all good matches of a template and an image.

- **FastMatchMg**
  Searching all good grayvalue matches in a pyramid.

- **SetOffsetTemplate**
  Gray value offset for template.

- **SetReferenceTemplate**
  Define reference position for a matching template.

- **WriteTemplate**
  Writing a template to file.

15.27 HTupleX

Grouping class for tuple-related functionality.
### Class Description

<table>
<thead>
<tr>
<th>Number of Methods</th>
<th>78</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>no</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
<td>no</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TupleAbs</strong></td>
</tr>
<tr>
<td>Compute the absolute value of a tuple.</td>
</tr>
<tr>
<td><strong>TupleAcos</strong></td>
</tr>
<tr>
<td>Compute the arccosine of a tuple.</td>
</tr>
<tr>
<td><strong>TupleAdd</strong></td>
</tr>
<tr>
<td>Add two tuples.</td>
</tr>
<tr>
<td><strong>TupleAnd</strong></td>
</tr>
<tr>
<td>Compute the logical and of two tuples.</td>
</tr>
<tr>
<td><strong>TupleAsin</strong></td>
</tr>
<tr>
<td>Compute the arcsine of a tuple.</td>
</tr>
<tr>
<td><strong>TupleAtan</strong></td>
</tr>
<tr>
<td>Compute the arctangent of a tuple.</td>
</tr>
<tr>
<td><strong>TupleAtan2</strong></td>
</tr>
<tr>
<td>Compute the arctangent of a tuple for all four quadrants.</td>
</tr>
<tr>
<td><strong>TupleBand</strong></td>
</tr>
<tr>
<td>Compute the bitwise and of two tuples.</td>
</tr>
<tr>
<td><strong>TupleBnot</strong></td>
</tr>
<tr>
<td>Compute the bitwise not of two tuples.</td>
</tr>
<tr>
<td><strong>TupleBor</strong></td>
</tr>
<tr>
<td>Compute the bitwise or of two tuples.</td>
</tr>
<tr>
<td><strong>TupleBxor</strong></td>
</tr>
<tr>
<td>Compute the bitwise exclusive or of two tuples.</td>
</tr>
<tr>
<td><strong>TupleCeil</strong></td>
</tr>
<tr>
<td>Compute the ceiling function of a tuple.</td>
</tr>
<tr>
<td><strong>TupleChr</strong></td>
</tr>
<tr>
<td>Convert a tuple of integers into strings with the corresponding ASCII codes.</td>
</tr>
<tr>
<td><strong>TupleChrt</strong></td>
</tr>
<tr>
<td>Convert a tuple of integers into strings with the corresponding ASCII codes.</td>
</tr>
<tr>
<td><strong>TupleConcat</strong></td>
</tr>
<tr>
<td>Concatenate two tuple to a new one.</td>
</tr>
<tr>
<td><strong>TupleCos</strong></td>
</tr>
<tr>
<td>Compute the cosine of a tuple.</td>
</tr>
<tr>
<td><strong>TupleCosh</strong></td>
</tr>
<tr>
<td>Compute the hyperbolic cosine of a tuple.</td>
</tr>
<tr>
<td><strong>TupleDeg</strong></td>
</tr>
<tr>
<td>Convert a tuple from radians to degrees.</td>
</tr>
<tr>
<td><strong>TupleDeviation</strong></td>
</tr>
<tr>
<td>Return the standard deviation of the elements of a tuple.</td>
</tr>
<tr>
<td><strong>TupleDiv</strong></td>
</tr>
<tr>
<td>Divide two tuples.</td>
</tr>
<tr>
<td><strong>TupleEnvironment</strong></td>
</tr>
<tr>
<td>Read one or more environment variables.</td>
</tr>
</tbody>
</table>
 TupleEqual ................................................................. P.831
 Test, whether two tuples are equal.

 TupleExp ............................................................... P.821
 Compute the exponential of a tuple.

 TupleFabs .............................................................. P.821
 Compute the absolute value of a tuple (as floating point numbers).

 TupleFirstN ............................................................. P.847
 Select the first elements of a tuple.

 TupleFloor ............................................................. P.842
 Compute the floor function of a tuple.

 TupleFmod ............................................................. P.822
 Calculate the remainder of the floating point division of two tuples.

 TupleGenConst ......................................................... P.840
 Generate a tuple of a specific length and initialize its elements.

 TupleGreater ........................................................ P.831
 Test, whether a tuple is greater than another tuple.

 TupleGreaterEqual .................................................. P.832
 Test, whether a tuple is greater or equal to another tuple.

 TupleInverse ........................................................ P.840
 Invert a tuple.

 TupleIsNumber ......................................................... P.843
 Check a tuple (of strings) whether it represents numbers.

 TupleLastN ............................................................ P.848
 Select all elements from index “n” to the end of a tuple.

 TupleLdexp ............................................................ P.822
 Calculate the ldexp function of two tuples.

 TupleLength .......................................................... P.843
 Returns the number of elements of a tuple.

 TupleLess ............................................................... P.832
 Test, whether a tuple is less than another tuple.

 TupleLessEqual ....................................................... P.833
 Test, whether a tuple is less or equal to another tuple.

 TupleLog ............................................................... P.823
 Compute the natural logarithm of a tuple.

 TupleLog10 ............................................................ P.823
 Compute the base 10 logarithm of a tuple.

 TupleLsh ............................................................... P.830
 Shift a tuple bitwise to the left.

 TupleMax ............................................................. P.843
 Returns the maximal element of a tuple.

 TupleMean ............................................................ P.844
 Return the mean value of a tuple of numbers.

 TupleMin ............................................................. P.844
 Returns the minimal element of a tuple.

 TupleMult ............................................................ P.824
 Multiply two tuples.

 TupleNeg ............................................................. P.824
 Negate a tuple.

 TupleNot ............................................................. P.846
 Compute the logical not of two tuples.

 TupleNotEqual ....................................................... P.834
 Test, whether two tuples are not equal.
 TupleNumber
Convert a tuple (of strings) into a tuple of numbers.

TupleOr
Compute the logical or of two tuples.

TupleOrd
Convert a tuple of strings of length 1 into a tuple of their ASCII codes.

TupleOrds
Convert a tuple of strings into a tuple of their ASCII codes.

TuplePow
Calculate the power function two tuples.

TupleRad
Convert a tuple from degrees to radians.

TupleReal
Convert a tuple into a tuple of floating point numbers.

TupleRound
Convert a tuple into a tuple of integer numbers.

TupleRsh
Shift a tuple bitwise to the right.

TupleSelect
Select single elements of a tuple.

TupleSelectRange
Select several elements of a tuple.

TupleSin
Compute the sine of a tuple.

TupleSinh
Compute the hyperbolic sine of a tuple.

TupleSort
Sorts the elements of a tuple in ascending order.

TupleSortIndex
Sorts the elements of a tuple and returns the indices of the sorted tuple.

TupleSplit
Split strings into substrings between predefined separator symbol(s).

TupleSqrt
Compute the square root of a tuple.

TupleStrBitSelect
Select single character or bit from a tuple.

TupleStrFirstN
Cut the first characters up to position “n” out of string tuple.

TupleStrLastN
Cut all characters starting at position “n” out of string tuple.

TupleStrchr
Forward search for a character within a string tuple.

TupleString
Convert a tuple into a tuple of strings.

TupleStrlen
Length of every string within a tuple of strings.

TupleStrrchr
Backward search for a character within a string tuple.

TupleStrrstr
Backward search for string(s) within a string tuple.

TupleStrstr
Forward search for string(s) within a string tuple.
CHAPTER 15. CLASS HIERARCHY

- **TupleSub**
  Subtract two tuples.

- **TupleSum**
  Return the sum of all elements of a tuple.

- **TupleTan**
  Compute the tangent of a tuple.

- **TupleTanh**
  Compute the hyperbolic tangent of a tuple.

- **TupleXor**
  Compute the logical exclusive or of two tuples.

#### 15.28 HUntypedObjectX

weak typed data type for object parameters.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

#### 15.29 HWindowX

Represents an instance of a Halcon window.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

- **NewExternWindow**
  Create a virtual graphics window under Windows NT.

- **OpenTextwindow**
  Open a textual window.

- **OpenWindow**
  Open a graphics window.
Methods

- **ClearRectangle**
  Delete a rectangle on the output window.

- **ClearWindow**
  Delete an output window.

- **CopyRectangle**
  Copy all pixels within rectangles between output windows.

- **DispArc**
  Displays circular arcs in a window.

- **DispArrow**
  Displays arrows in a window.

- **DispCaltab**
  Project and visualize the 3D model of the calibration table in the image.

- **DispChannel**
  Displays images with several channels.

- **DispCircle**
  Displays circles in a window.

- **DispColor**
  Displays a color (RGB) image.

- **DispDistribution**
  Displays a noise distribution.

- **DispEllipse**
  Displays ellipses.

- **DispImage**
  Displays gray value images.

- **DispLine**
  Draws lines in a window.

- **DispLut**
  Graphical view of the look-up-table (lut).

- **DispObj**
  Displays image objects (image, region, XLD).

- **DispPolygon**
  Displays a polyline.

- **DispRectangle1**
  Display of rectangles aligned to the coordinate axes.

- **DispRectangle2**
  Displays arbitrarily oriented rectangles.

- **DispRegion**
  Displays regions in a window.

- **DispXld**
  Display an XLD object.

- **DragRegion1**
  Interactive moving of a region.

- **DragRegion2**
  Interactive movement of a region with fixpoint specification.

- **DragRegion3**
  Interactive movement of a region with restriction of positions.

- **DrawCircle**
  Interactive drawing of a circle.

- **DrawCircleMod**
  Interactive drawing of a circle.
DrawEllipse
Interactive drawing of an ellipse.

DrawEllipseMod
Interactive drawing of an ellipse.

DrawLine
Draw a line.

DrawLineMod
Draw a line.

DrawLut
Manipulate look-up-table (lut) interactively.

DrawPoint
Draw a point.

DrawPointMod
Draw a point.

DrawPolygon
Interactive drawing of a polygon row.

DrawRectangle1
Draw a rectangle parallel to the coordinate axis.

DrawRectangle1Mod
Draw a rectangle parallel to the coordinate axis.

DrawRectangle2
Interactive drawing of any orientated rectangle.

DrawRectangle2Mod
Interactive drawing of any orientated rectangle.

DrawRegion
Interactive drawing of a closed region.

DumpWindow
Write the window content to a file.

GetDraw
Get the current region fill mode.

GetFix
Get mode of fixing of current look-up-table (lut).

GetFixedLut
Get fixing of "look-up-table" (lut) for "real color images".

GetFont
Get the current font.

GetHsi
Get the HSI coding of the current color.

GetIcon
Query the icon for region output

GetInsert
Get the current display mode.

GetLineApprox
Get the current approximation error for contour display.

GetLineStyle
Get the current graphic mode for contours.

GetLineWidth
Get the current line width for contour display.

GetLut
Get current look-up-table (lut).

GetLutStyle
Get modification parameters of look-up-table (lut).
GetMButton.......................................................... P.205
Wait until a mouse button is pressed.

GetMPosition.................................................. P.206
Query the mouse position.

GetMShape.................................................... P.207
Query the current mouse pointer shape.

GetPaint........................................................ P.229
Get the current display mode for grayvalues.

GetPart.......................................................... P.230
Get the image part.

GetPartStyle.................................................. P.230
Get the current interpolation mode for grayvalue display.

GetPixel........................................................ P.231
Get the current color lookup table index.

GetRgb.......................................................... P.232
Get the current color in RGB-coding.

GetShape........................................................ P.232
Get the current region output shape.

GetStringExtents.......................................... P.256
Get the spatial size of a string.

GetTPosition................................................. P.257
Get cursor position.

GetTShape..................................................... P.258
Get the shape of the text cursor.

GetWindowExtents.......................................... P.270
Information about a window’s size and position.

GetWindowPointer3........................................ P.271
Access to a window’s pixel data.

GetWindowType............................................. P.272
Get the window type.

MoveRectangle............................................. P.273
Copy inside an output window.

NewLine...................................................... P.259
Set the position of the text cursor to the beginning of the next line.

QueryAllColors............................................. P.233
Query all color names.

QueryColor.................................................. P.233
Query all color names displayable in the window.

QueryFont.................................................... P.259
Query the available fonts.

QueryGray.................................................... P.235
Query the displayable grayvalues.

QueryInset................................................... P.235
Query the possible graphic modes.

QueryLut....................................................... P.200
Query all available look-up-tables (lut).

QueryMShape............................................... P.208
Query all available mouse pointer shapes.

QueryPaint.................................................. P.237
Query the grayvalue display modes.

QueryTShape................................................. P.260
Query all shapes available for text cursors.
CHAPTER 15. CLASS HIERARCHY

- **ReadChar** P.260
  Read a character from a text window.

- **ReadString** P.261
  Read a string in a text window.

- **SetColor** P.238
  Set output color.

- **SetColored** P.239
  Set multiple output colors.

- **SetComprise** P.239
  Define the image matrix output clipping.

- **SetDraw** P.240
  Define the region fill mode.

- **SetFix** P.241
  Set fixing of "look-up-table" (lut)

- **SetFixedLut** P.201
  Fix "look-up-table" (lut) for "real color images".

- **SetFont** P.262
  Set the font used for text output.

- **SetGray** P.242
  Define grayvalues for region output.

- **SetHsi** P.243
  Define output colors (HSI-coded).

- **SetIcon** P.244
  Icon definition for region output.

- **SetInsert** P.244
  Define the pixel output function.

- **SetLineApprox** P.245
  Define the approximation error for contour display.

- **SetLineStyle** P.246
  Define a contour output pattern.

- **SetLineWidth** P.247
  Define the line width for region contour output.

- **SetLut** P.201
  Set "look-up-table" (lut).

- **SetLutStyle** P.204
  Changing the look-up-table (lut).

- **SetMshape** P.208
  Set the current mouse pointer shape.

- **SetPaint** P.247
  Define the grayvalue output mode.

- **SetPart** P.251
  Modify the displayed image part.

- **SetPartStyle** P.252
  Define an interpolation method for grayvalue output.

- **SetPixel** P.253
  Define a color lookup table index.

- **SetRgb** P.254
  Set the color definition via RGB values.

- **SetShape** P.255
  Define the region output shape.

- **SetCursorPosition** P.263
  Set the position of the text cursor.
Set the shape of the text cursor.

Set the device context of a virtual graphics window (Windows NT).

Modify position and size of a window.

Interactive output from two window buffers.

Write look-up-table (lut) as file.

Print text in a window.

Properties

AllColors (read-only)
mapped to: QueryAllColors
Query all color names.

Blue
mapped to: GetRgb
Get the current color in RGB-coding.

Color
mapped to: SetColor
Set output color.

Colored
mapped to: SetColored
Set multiple output colors.

Colors (read-only)
mapped to: QueryColor
Query all color names displayable in the window.

Comprise
mapped to: SetComprise
Define the image matrix output clipping.

Draw
mapped to: GetDraw
Get the current region fill mode.

Fix
mapped to: GetFix
Get mode of fixing of current look-up-table (lut).

Font
mapped to: GetFont
Get the current font.

Fonts (read-only)
mapped to: QueryFont
Query the available fonts.

GraphicModes (read-only)
mapped to: QueryInsert
Query the possible graphic modes.

Gray
mapped to: SetGray
Define grayvalues for region output.

GrayValues (read-only)
mapped to: QueryGray
Query the displayable grayvalues.
- **Green**
  mapped to: GetRgb
  Get the current color in RGB-coding.

- **Hue**
  mapped to: GetHsi
  Get the HSI coding of the current color.

- **Insert**
  mapped to: GetInsert
  Get the current display mode.

- **Intensity**
  mapped to: GetHsi
  Get the HSI coding of the current color.

- **LineApprox**
  mapped to: GetLineApprox
  Get the current approximation error for contour display.

- **LineWidth**
  mapped to: GetLineWidth
  Get the current line width for contour display.

- **Lut**
  mapped to: GetLut
  Get current look-up-table (lut).

- **Luts (read-only)**
  mapped to: QueryLut
  Query all available look-up-tables (lut).

- **Paint**
  mapped to: GetPaint
  Get the current display mode for grayvalues.

- **PaintModes (read-only)**
  mapped to: QueryPaint
  Query the grayvalue display modes.

- **PartColumn1**
  mapped to: GetPart
  Get the image part.

- **PartColumn2**
  mapped to: GetPart
  Get the image part.

- **PartRow1**
  mapped to: GetPart
  Get the image part.

- **PartRow2**
  mapped to: GetPart
  Get the image part.

- **PartStyle**
  mapped to: GetPartStyle
  Get the current interpolation mode for grayvalue display.

- **Pixel**
  mapped to: GetPixel
  Get the current color lookup table index.

- **Red**
  mapped to: GetRgb
  Get the current color in RGB-coding.

- **Saturation**
  mapped to: GetHsi
  Get the HSI coding of the current color.
15.30  HXLDX

Instance of a XLD object.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

### Constructor(s)

ReceiveXld  
Receive an XLD object over a socket connection.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>AreaCenterXld</td>
</tr>
<tr>
<td>DispXld</td>
</tr>
<tr>
<td>GetParallelsXld</td>
</tr>
<tr>
<td>LengthXld</td>
</tr>
<tr>
<td>MomentsAnyXld</td>
</tr>
<tr>
<td>MomentsXld</td>
</tr>
<tr>
<td>SendXld</td>
</tr>
</tbody>
</table>

15.31  HXLDCntX

Represents an instance of an XLD contour object.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>
Constructor(s)

- **GenContourPolygonXld**
  
  Generate an XLD contour from a polygon (given as tuples).

- **GenEllipseContourXld**
  
  Creation of an XLD contour corresponding to an elliptic arc.

- **ReadContourXldArcInfo**
  
  Read XLD contours to a file in ARC/INFO generate format.

Methods

- **AddNoiseWhiteContourXld**
  
  Add noise to XLD contours.

- **AffineTransContourXld**
  
  Apply an arbitrary affine transformation to an XLD contour.

- **ChangeRadialDistortionContoursXld**
  
  Change the radial distortion of contours.

- **ClipContoursXld**
  
  Clip an XLD contour.

- **ContourPointNumXld**
  
  Return the number of points in an XLD contour.

- **DistEllipseContourXld**
  
  Distance of contours to an ellipse.

- **FitCircleContourXld**
  
  Approximation of XLD contours by circles.

- **FitEllipseContourXld**
  
  Approximation of XLD contours by ellipses or elliptic arcs.

- **FitLineContourXld**
  
  Approximation of XLD contours by line segments.

- **GenParallelContourXld**
  
  Compute the parallel contour of an XLD contour.

- **GenPolygonsXld**
  
  Approximate XLD contours by polygons.

- **GetContourAngleXld**
  
  Calculate the direction of an XLD contour for each contour point.

- **GetContourAttribXld**
  
  Return point attribute values of an XLD contour.

- **GetContourGlobalAttribXld**
  
  Return global attributes values of an XLD contour.

- **GetContourXld**
  
  Return the coordinates of an XLD contour.

- **GetRegressParamsXld**
  
  Return XLD contour parameters.

- **LocalMaxContoursXld**
  
  Select XLD contours with a local maximum of gray values.

- **MergeContLineScanXld**
  
  Merge XLD contours from successive line scan images.

- **QueryContourAttribsXld**
  
  Return the names of the defined attributes of an XLD contour.

- **QueryContourGlobalAttribsXld**
  
  Return the names of the defined global attributes of an XLD contour.

- **RegressContoursXld**
  
  Calculate the parameters of a regression line to an XLD contour.
15.32  HXLDEXTPARAX

Segment ContoursXld .............................................. P.894
  Segment XLD contours into line segments and circular or elliptic arcs.

Select ContoursXld .............................................. P.885
  Select XLD contours according to several features.

Smooth ContoursXld .............................................. P.896
  Smooth an XLD contour.

Union Straight Contours Histo Xld ............................... P.897
  Merge neighboring straight contours having a similar distance from a given line.

Union Straight Contours Xld ...................................... P.899
  Merge neighboring straight contours having a similar direction.

Write Contour Xld Arc Info ...................................... P.25
  Write XLD contours to a file in ARC/INFO generate format.

15.32  HXLDExtParaX

Represents an instance of an XLD extended parallel object.

Number of Methods .............................................. 1
Number of Properties .......................................... 0
Abstract Class .................................................. no
Base Class(es) .................................................... HObject
Automatic Destructor .......................................... yes, through ClearObj (not member of this class!)
Object Identity (through internal handle) ...................... yes

Max Parallels Xld .............................................. P.881
  Join modified XLD parallels lying on the same polygon.

15.33  HXLDModParaX

Represents an instance of an XLD modified parallel object.

Number of Methods .............................................. 1
Number of Properties .......................................... 0
Abstract Class .................................................. no
Base Class(es) .................................................... HObject
Automatic Destructor .......................................... yes, through ClearObj (not member of this class!)
Object Identity (through internal handle) ...................... yes

Combine Roads Xld .............................................. P.890
  Combine road hypotheses from two resolution levels.

HALCON 6.0
15.34 HXLDParaX

Represents an instance of an XLD parallel object.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

Methods

- **InfoParallelsXld**
  Return information about the gray values of the area enclosed by XLD parallels. (P.879)
- **ModParallelsXld**
  Extract parallel XLD polygons enclosing a homogeneous area. (P.866)

15.35 HXLDPolyX

Represents an instance of an XLD polygon object.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

Constructor(s)

- **ReadPolygonXldArcInfo**
  Read XLD polygons from a file in ARC/INFO generate format. (P.24)

Methods

- **AffineTransPolygonXld**
  Apply an arbitrary affine transformation to an XLD polygon. (P.888)
- **CombineRoadsXld**
  Combine road hypotheses from two resolution levels. (P.890)
- **GenParallelsXld**
  Extract parallel XLD polygons. (P.864)
- **GetLinesXld**
  Return an XLD polygon’s data (as lines). (P.857)
- **GetPolygonXld**
  Return an XLD polygon’s data. (P.859)
- **SplitContoursXld**
  Split XLD contours at dominant points. (P.896)
- **WritePolygonXldArcInfo**
  Write XLD polygons to a file in ARC/INFO generate format. (P.26)
Index

2D-Transformation, 16, 27, 454, 625, 628–633, 887, 888
3D-Projection, 700
3D-Transformation, 626, 634–637, 639, 640
3D-projection, 711
AbsFunct1D, 726
AbsImage, 34
AbsInvarFourierCoeff, 721
Absolute-Value, 34, 726, 817, 821
abstract, 591
Access, 728
AccessChannel, 289
Access, 729
AdaptTemplate, 108
AddChannels, 314
AddImage, 34
Addition, 34, 733, 818
AddNoiseDistribution, 138
AddNoiseWhite, 139
AddNoiseWhiteContourXld, 886
Affine-Map, 27, 29, 31–33, 454, 627, 642–645, 887, 888
Affine-Transformation, 456
Affine-Transformation., 455, 457
AffineTransContourXld, 887
AffineTransImage, 27
AffineTransPoint2D, 625
AffineTransPoint3D, 626
AffineTransPolygonXld, 888
AffineTransRegion, 454
Albedo, 806–808
AllColors (property), 983
Ambient, 806
and, 42
AngleLl, 737
AngleLx, 738
Anisometry, 482, 504
AnisotropeDiff, 142
Anisotropy, 321, 334
AppendChannel, 289
AppendOcrTrainf, 785
ApproxChain, 367
ApproxChainSimple, 370
Approximation, 367, 370
Arc, 209, 367
Arcosine, 817
Arcs, 370
Arcsine, 818
Arctangent, 819
Area, 317, 477, 504, 510, 867
AreaCenter, 477
AreaCenterGray, 317
AreaCenterXld, 867
Arithmetic, 34–41
assurance, 591, 593
Attributes, 877, 884
AutoThreshold, 541
Average, 844, 866, 876, 878, 879, 885
Background, 518, 523
background, 645–647, 649, 651–653, 655
background-estimation, 645–647, 649, 651–653, 655
BackgroundSeg, 518
Band-Filter, 93
Bandpass-Filter, 94, 98, 99, 103
BandpassImage, 103
Barcode, 656, 658, 659, 662, 666, 667, 670
Barcode-description, 666, 667
BestMatch, 108
BestMatchMg, 110
BestMatchPreMg, 111
BestMatchRot, 112
BestMatchRotMg, 114
Binary-image, 16
BinThreshold, 542
Bit, 42–47
Bit-Operations, 828–830, 849
BitAnd, 42
BitLshift, 43
BitMask, 44
BitNot, 44
BitOr, 45
BitRshift, 46
BitSlice, 46
BitsPerPixelChannel (property), 905
BitXor, 47
Blue (property), 983
Blurring, 165, 166, 168, 169
BMP, 11, 14, 16, 269
Border-Treatment, 131
Bottom-Hat, 381, 389, 421
BottomHat, 389
Boundary, 390
Boundary, 390
Branches, 428
Buffer, 276, 279, 603, 606
Bulkiness, 482
C-procedure, 596
Calibration, 626, 634–637, 640, 700
calibration, 676, 677, 682–685, 687, 689, 695, 697, 698, 701, 702, 707, 709–712, 714–717, 719
CaltabPoints, 676
CameraCalibration, 677
Canny-Filter, 62, 65
Ceiling-Function, 841
Center, 317, 460, 477, 504, 510
Center-of-Gravity, 373, 375
Centroid, 867
Chain-code, 449
Chaincode, 367
Chaincodes, 370
ChangeDomain, 314
ChangeFormat, 338
ChangeRadialDistortionCamPar, 682
ChangeRadialDistortionContoursXld, 683
ChangeRadialDistortionImage, 684
Channels, 583
ChannelsToImage, 290
Chapter, 590
chapter, 591
Character, 801–805
Character-sequence, 20
Characters, 794, 795
CharThreshold, 542
CheckDifference, 543
Checkerboard, 458
CheckParHwPotential, 600
Choice-of-regions, 504
Chord, 500
Chord-representation, 453
Circle, 212, 391, 397, 403, 424, 460, 465, 490, 498, 509, 894
circle, 178, 179, 255, 870
Circularity, 478
Class, 440
Class2DimSup, 545
Class2DimUnsup, 546
Classification, 137, 545–548, 560, 561
ClassNdimBox, 547
ClassNdimNorm, 548
Clearing, 307, 443
ClearObj, 443
ClearRectangle, 265
ClearSampset, 1
ClearSerial, 610
ClearShapeModel, 768
ClearTemplate, 115
ClearWindow, 266
ClipContoursXld, 889
Clipping, 519, 520, 522, 532, 889
ClipRegion, 519
ClipRegionRel, 520
Close, 801
CloseAllBgEsti, 645
CloseAllClassBox, 1
CloseAllFiles, 18
CloseAllFramegrabbers, 345
CloseAllOcrs, 786
CloseAllOcvs, 801
CloseAllSerials, 611
CloseBgEsti, 646
CloseClassBox, 2
CloseEdges, 56
CloseEdgesLength, 57
CloseFile, 18
CloseFramegrabber, 346
CloseMeasure, 775
CloseOcr, 786
CloseOcv, 801
CloseSerial, 611
CloseSocket, 616
CloseWindow, 267
Closing, 267, 379, 381, 382, 389–392, 408, 421
Closing, 390
ClosingCircle, 391
ClosingGolay, 392
ClosingRectangle1, 393
Clustering, 546–548, 560, 561
Co-occurrence-Matrix, 318, 319, 321, 327
Coding, 500
Coincidence, 465
Color, 198–201, 204
color, 49, 53, 226, 231–234, 238, 239, 243, 253
Color (property), 983
color-coding, 226, 243
Color-Image, 214
color-image, 48, 49
color-lookup-table, 231, 232
color-space, 49, 53
color-space-transformation, 48, 49, 53
Color-Table, 603, 606
Colored (property), 983
Colors (property), 983
ColorSpace (property), 905
CombineRoadsXld, 890
Compactness, 478, 479, 504
Compactness, 479
Comparison, 441, 802
Comparison-Operators, 831–834
Complement, 514
Complement, 514
ComplexToReal, 364
Components, 439
Compose2, 290
Compose3, 291
Compose4, 292
Compose5, 292
Compose6, 293
Compose7, 294
Comprise (property), 983
Concatenation, 443
ConcatObj, 443
ConcatOcrTrainf, 787
Index

Configuration, 283
configuration, 233–237, 259
ConnectAndHoles, 479
Connected-Components, 336, 337, 518, 521
Connection, 521
Connection-component, 479, 485
Contlength, 480
Contour, 449, 450, 452, 480, 745
contour, 228, 229, 240, 246
Contour-Length, 880
Contour-length, 479, 480, 504
contour-output, 247
ContourPointNumXld, 867
Contours, 56, 57, 390, 857, 860, 861, 863–865, 867, 868, 870, 871, 874, 876–878, 880, 884–887, 889, 891, 893, 894, 896, 897, 899
contours, 683
ContourToWorldPlaneXld, 684
Contrast, 89
contrast, 84
Contrast-enhancement, 85, 86
Control, 585, 600, 602, 603, 606
Control-Modes, 586
control-parameter, 595
Conversion, 834–838, 843
ConvertImageType, 364
ConvertPoseType, 685
Convex, 450
Convexity, 481, 504
Convexity, 481
ConvolfFft, 88
Convolgabor, 89
Convollmage, 131
Convolution, 88, 89, 94, 95, 131
CoocFeatureImage, 318
CoocFeatureMatrix, 319
Coordinate-System, 274, 276, 279, 626, 634–637, 640
CopyImage, 299
Copying, 299, 307, 361, 362, 443, 444, 447
CopyObj, 444
CopyRectangle, 267
CornerResponse, 115
Corners, 115
Correlation, 108, 110–112, 114, 120, 121, 123, 124
Cosine, 820
CountChannels, 294
CountObj, 439
CountRelation, 581
CountSeconds, 599
Create, 801
CreateBgEsti, 647
CreateCaltab, 687
CreateClassBox, 2
CreateFunct1DArray, 726
CreateFunct1DPairs, 727
CreateOcrClassBox, 787
CreateOcvProj, 801
CreatePose, 689
CreateShapeModel, 769
CreateTemplate, 117
CreateTemplateRot, 118
Creation, 839, 840
CropDomain, 339
CropDomainRel, 522
CropPart, 339
CropRectangle1, 340
cross-reference, 591, 598
Cursor, 207, 208
cursor-position, 257
Database, 299, 443–447, 581, 583
Debugging, 585
Decode, 656, 658
Decode1DBarCode, 656
Decode2DBarCode, 657
Decompose2, 295
Decompose3, 295
Decompose4, 296
Decompose5, 297
Decompose6, 297
Decompose7, 298
default-type, 593
Degrees, 825, 835
Delete, 442
Deletion, 265, 266
DepthFromFocus, 806
Deriche-Filter, 62, 65, 159
DerivateGauss, 58
DescribeClassBox, 2
description-file, 11
description-file, 762
DetectEdgeSegments, 549
Deviation, 330, 334
DeviationImage, 162
Devicecontext, 284
Diameter, 482
DiameterRegion, 482
Difference, 515
Difference, 515
Differential-Geometry, 58
diffOfGauss, 61
Diffusion-Filter, 142
Dilatation, 746
Dilation1, 394
Dilation2, 396
DilationCircle, 397
DilationGolay, 398
DilationRectangle1, 399
DilationSeq, 400
Direction, 857–859, 876, 878, 885, 886, 891, 896, 897, 899
direction-code, 449
Discrete, 658
Discrete1DBarCode, 658
DispArc, 209
HALCON 6.0
<table>
<thead>
<tr>
<th>Index</th>
<th>992</th>
</tr>
</thead>
<tbody>
<tr>
<td>DispArrow, 210</td>
<td>DrawRectangle1Mod, 188</td>
</tr>
<tr>
<td>DispCaltab, 695</td>
<td>DrawRectangle2, 190</td>
</tr>
<tr>
<td>DispChannel, 211</td>
<td>DrawRectangle2Mod, 191</td>
</tr>
<tr>
<td>DispCircle, 212</td>
<td>DrawRegion, 192</td>
</tr>
<tr>
<td>DispColor, 214</td>
<td>DualRank, 379</td>
</tr>
<tr>
<td>DispDistribution, 214</td>
<td>DualThreshold, 550</td>
</tr>
<tr>
<td>DispEllipse, 215</td>
<td>DumpWindow, 269</td>
</tr>
<tr>
<td>DispImage, 217</td>
<td>DvfToHomMat2d, 627</td>
</tr>
<tr>
<td>DispInfo, 589</td>
<td>DvfToInt1, 365</td>
</tr>
<tr>
<td>Displacement-Vector-Field, 125, 127, 365</td>
<td>Dynamic-Threshold, 543, 552</td>
</tr>
<tr>
<td>DISPLAY, 276, 279</td>
<td>DynThreshold, 552</td>
</tr>
<tr>
<td>Display, 224</td>
<td>Eccentricity, 482</td>
</tr>
<tr>
<td>display, 227, 235, 236</td>
<td>Edge-Amplitude, 62, 65, 67, 68, 71, 72, 76, 79–82</td>
</tr>
<tr>
<td>DispLine, 218</td>
<td>Edge-Closing, 56, 57</td>
</tr>
<tr>
<td>DispLut, 197</td>
<td>Edge-Detection, 56–58, 61, 62, 67–73, 75, 76, 79–82, 549, 563, 564</td>
</tr>
<tr>
<td>DispObj, 219</td>
<td>Edge-Direction, 62, 65, 68, 72, 76, 80, 82, 891</td>
</tr>
<tr>
<td>DispPolygon, 220</td>
<td>Edge-Preservation, 142, 158</td>
</tr>
<tr>
<td>DispRectangle1, 220</td>
<td>Edge-Thaning, 62, 563, 564</td>
</tr>
<tr>
<td>DispRectangle2, 222</td>
<td>Edges, 56, 57, 549, 550, 563, 564, 577, 578, 775, 777, 779, 781, 783, 784</td>
</tr>
<tr>
<td>DispRegion, 223</td>
<td>EdgesImage, 62</td>
</tr>
<tr>
<td>DispXld, 224</td>
<td>EdgesSubPix, 65</td>
</tr>
<tr>
<td>Distance, 482, 523, 728, 745, 746, 868</td>
<td>Elements, 659</td>
</tr>
<tr>
<td>DistanceFunct1D, 728</td>
<td>EliminateMinMax, 143</td>
</tr>
<tr>
<td>DistanceLr, 739</td>
<td>EliminateRuns, 524</td>
</tr>
<tr>
<td>DistancePl, 740</td>
<td>EliminateSp, 145</td>
</tr>
<tr>
<td>DistancePp, 741</td>
<td>Ellipse, 215, 320, 461, 465, 482, 483, 750, 868, 894</td>
</tr>
<tr>
<td>DistancePr, 743</td>
<td>ellipse, 180, 181, 255, 863, 871</td>
</tr>
<tr>
<td>DistancePs, 744</td>
<td>EllipticAxis, 483</td>
</tr>
<tr>
<td>DistanceRrMin, 745</td>
<td>EllipticAxisGray, 320</td>
</tr>
<tr>
<td>DistanceRrMinDil, 746</td>
<td>Emphasize, 84</td>
</tr>
<tr>
<td>DistanceSl, 747</td>
<td>End-Points, 530</td>
</tr>
<tr>
<td>DistanceSr, 748</td>
<td>Energy, 89</td>
</tr>
<tr>
<td>DistanceSs, 749</td>
<td>EnergyGabor, 89</td>
</tr>
<tr>
<td>DistanceTransform, 523</td>
<td>EnquireClassBox, 3</td>
</tr>
<tr>
<td>DistanceXld, 868</td>
<td>EnquireRejectClassBox, 4</td>
</tr>
<tr>
<td>Distribution, 321, 328, 330, 331, 500</td>
<td>Entropy, 162, 321, 325, 334</td>
</tr>
<tr>
<td>DivImage, 35</td>
<td>EntropyGray, 321</td>
</tr>
<tr>
<td>Division, 35, 820, 822</td>
<td>EntropyImage, 162</td>
</tr>
<tr>
<td>Documentation, 603, 606</td>
<td>EquHistImage, 85</td>
</tr>
<tr>
<td>DoOcrMulti, 790</td>
<td>Erosion1, 401</td>
</tr>
<tr>
<td>DoOcrSingle, 790</td>
<td>Erosion2, 402</td>
</tr>
<tr>
<td>DoOcvSimple, 802</td>
<td>ErosionCircle, 403</td>
</tr>
<tr>
<td>DragRegion1, 175</td>
<td>ErosionGolay, 404</td>
</tr>
<tr>
<td>DragRegion2, 176</td>
<td>ErosionRectangle1, 406</td>
</tr>
<tr>
<td>DragRegion3, 177</td>
<td>ErosionRectangulare2, 406</td>
</tr>
<tr>
<td>Draw (property), 983</td>
<td>ErosionSeq, 407</td>
</tr>
<tr>
<td>DrawCircle, 178</td>
<td>Error-Control, 584</td>
</tr>
<tr>
<td>DrawCircleMod, 179</td>
<td>Error-Message, 584</td>
</tr>
<tr>
<td>DrawEllipse, 180</td>
<td>Error-Text, 584</td>
</tr>
<tr>
<td>DrawEllipseMod, 181</td>
<td>Errors, 586</td>
</tr>
<tr>
<td>drawing, 178, 179, 182, 183, 185, 187, 188, 190–192</td>
<td>estimate, 758, 762, 766</td>
</tr>
<tr>
<td>DrawLine, 182</td>
<td>EstimateAlAm, 806</td>
</tr>
<tr>
<td>DrawLineMod, 183</td>
<td>EstimateSlAllLr, 807</td>
</tr>
<tr>
<td>DrawLut, 198</td>
<td>EstimateSlAllZc, 808</td>
</tr>
<tr>
<td>DrawPoint, 185</td>
<td>DrawPointMod, 185</td>
</tr>
<tr>
<td>DrawPolygon, 186</td>
<td>DrawRectangle1, 187</td>
</tr>
<tr>
<td>Function/Property</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>EstimateTiltLr</td>
<td>808</td>
</tr>
<tr>
<td>EstimateTiltZc</td>
<td>809</td>
</tr>
<tr>
<td>Estimation, estimation</td>
<td>806 – 809</td>
</tr>
<tr>
<td>Euler-number</td>
<td>485</td>
</tr>
<tr>
<td>EulerNumber</td>
<td>485</td>
</tr>
<tr>
<td>Evaluation</td>
<td>808</td>
</tr>
<tr>
<td>example</td>
<td>591</td>
</tr>
<tr>
<td>Exception</td>
<td>584</td>
</tr>
<tr>
<td>ExhaustiveMatch</td>
<td>120</td>
</tr>
<tr>
<td>ExhaustiveMatchMg</td>
<td>121</td>
</tr>
<tr>
<td>ExpandDomainGray</td>
<td>132</td>
</tr>
<tr>
<td>ExpandGray</td>
<td>553</td>
</tr>
<tr>
<td>ExpandGrayRef</td>
<td>555</td>
</tr>
<tr>
<td>ExpandLine</td>
<td>556</td>
</tr>
<tr>
<td>ExpandRegion</td>
<td>525</td>
</tr>
<tr>
<td>Expansion</td>
<td>525, 553, 555, 556</td>
</tr>
<tr>
<td>Exponential</td>
<td>821</td>
</tr>
<tr>
<td>Extension</td>
<td>11</td>
</tr>
<tr>
<td>Extern</td>
<td>274, 284</td>
</tr>
<tr>
<td>ExternalTrigger (property)</td>
<td>905</td>
</tr>
<tr>
<td>FastMatch</td>
<td>123</td>
</tr>
<tr>
<td>FastMatchMg</td>
<td>124</td>
</tr>
<tr>
<td>FastThreshold</td>
<td>557</td>
</tr>
<tr>
<td>Feature-Space</td>
<td>545, 547, 548, 560, 561</td>
</tr>
<tr>
<td>Features</td>
<td>842, 844</td>
</tr>
<tr>
<td>FFT</td>
<td>88 – 95, 97 – 102</td>
</tr>
<tr>
<td>FftGeneric</td>
<td>90</td>
</tr>
<tr>
<td>FftImage</td>
<td>91</td>
</tr>
<tr>
<td>FftImageInv</td>
<td>92</td>
</tr>
<tr>
<td>Field (property)</td>
<td>905</td>
</tr>
<tr>
<td>File</td>
<td>603, 606</td>
</tr>
<tr>
<td>file-access</td>
<td>15</td>
</tr>
<tr>
<td>File-Format</td>
<td>16</td>
</tr>
<tr>
<td>File_format</td>
<td>14</td>
</tr>
<tr>
<td>FileExists</td>
<td>15</td>
</tr>
<tr>
<td>Fileheader</td>
<td>12</td>
</tr>
<tr>
<td>FillDvfl</td>
<td>125</td>
</tr>
<tr>
<td>FillInterlace</td>
<td>146</td>
</tr>
<tr>
<td>FillUp</td>
<td>526</td>
</tr>
<tr>
<td>FillUpShape</td>
<td>527</td>
</tr>
<tr>
<td>Filter</td>
<td>88, 89, 94, 95, 115, 132, 287</td>
</tr>
<tr>
<td>Filter-Mask</td>
<td>131</td>
</tr>
<tr>
<td>Filter-Width</td>
<td>70, 148</td>
</tr>
<tr>
<td>FilterKalman</td>
<td>758</td>
</tr>
<tr>
<td>Find1DBarCode</td>
<td>659</td>
</tr>
<tr>
<td>Find1DBarCodeRegion</td>
<td>662</td>
</tr>
<tr>
<td>Find2DBarCode</td>
<td>663</td>
</tr>
<tr>
<td>FindCaltab</td>
<td>697</td>
</tr>
<tr>
<td>FindMarksAndPose</td>
<td>698</td>
</tr>
<tr>
<td>FindNeighbors</td>
<td>485</td>
</tr>
<tr>
<td>FindShapeModel</td>
<td>771</td>
</tr>
<tr>
<td>FitCircleContourXld</td>
<td>870</td>
</tr>
<tr>
<td>FitEllipseContourXld</td>
<td>871</td>
</tr>
<tr>
<td>FitLineContourXld</td>
<td>874</td>
</tr>
<tr>
<td>FitSurfaceFirstOrder</td>
<td>322</td>
</tr>
<tr>
<td>FitSurfaceSecondOrder</td>
<td>323</td>
</tr>
<tr>
<td>Fitting</td>
<td>408</td>
</tr>
<tr>
<td>Fitting (property)</td>
<td>983</td>
</tr>
<tr>
<td>Fix (property)</td>
<td>983</td>
</tr>
<tr>
<td>Floor-Function</td>
<td>842</td>
</tr>
<tr>
<td>Fnewline</td>
<td>19</td>
</tr>
<tr>
<td>Font</td>
<td>603, 606</td>
</tr>
<tr>
<td>font</td>
<td>256, 259, 262</td>
</tr>
<tr>
<td>Font (property)</td>
<td>983</td>
</tr>
<tr>
<td>font-size</td>
<td>256</td>
</tr>
<tr>
<td>Fonts (property)</td>
<td>983</td>
</tr>
<tr>
<td>Foreground</td>
<td>523</td>
</tr>
<tr>
<td>foreground</td>
<td>645 – 647, 649, 651 – 653, 655</td>
</tr>
<tr>
<td>Form</td>
<td>504</td>
</tr>
<tr>
<td>Form-feature</td>
<td>492 – 497</td>
</tr>
<tr>
<td>Fourier-Transformation</td>
<td>90 – 94, 97 – 102</td>
</tr>
<tr>
<td>Fourier1Dim</td>
<td>722</td>
</tr>
<tr>
<td>Fourier1DimInv</td>
<td>722</td>
</tr>
<tr>
<td>Framegrabber</td>
<td>345 – 352, 354, 356, 357</td>
</tr>
<tr>
<td>FreadChar</td>
<td>19</td>
</tr>
<tr>
<td>FreadString</td>
<td>20</td>
</tr>
<tr>
<td>Frei-Filter</td>
<td>67, 68</td>
</tr>
<tr>
<td>FreiAmp</td>
<td>67</td>
</tr>
<tr>
<td>FreiDir</td>
<td>68</td>
</tr>
<tr>
<td>Frequency</td>
<td>328, 330, 331</td>
</tr>
<tr>
<td>Frequency-Domain</td>
<td>88 – 91, 93 – 95, 97 – 99</td>
</tr>
<tr>
<td>FullDomain</td>
<td>315</td>
</tr>
<tr>
<td>Funct1DToPairs</td>
<td>728</td>
</tr>
<tr>
<td>Function</td>
<td>728</td>
</tr>
<tr>
<td>Functions</td>
<td>726, 728 – 733, 735 – 737</td>
</tr>
<tr>
<td>Fuzzy-Sets</td>
<td>325, 326</td>
</tr>
<tr>
<td>FuzzyEntropy</td>
<td>325</td>
</tr>
<tr>
<td>FuzzyPerimeter</td>
<td>326</td>
</tr>
<tr>
<td>FwriteString</td>
<td>21</td>
</tr>
<tr>
<td>Gabor-Filter</td>
<td>89, 95</td>
</tr>
<tr>
<td>Gain (property)</td>
<td>905</td>
</tr>
<tr>
<td>Gaps</td>
<td>525, 553, 555</td>
</tr>
<tr>
<td>Gauss-Filter</td>
<td>147, 159</td>
</tr>
<tr>
<td>Gauss-Pyramid</td>
<td>125</td>
</tr>
<tr>
<td>GaussDistribution</td>
<td>139</td>
</tr>
<tr>
<td>Gaussian</td>
<td>58, 61, 73, 75, 81, 82, 99, 147, 159, 541, 542, 558</td>
</tr>
<tr>
<td>Gaussian-Function</td>
<td>734</td>
</tr>
<tr>
<td>GaussImage</td>
<td>147</td>
</tr>
<tr>
<td>Gen1DBarcodeDescr</td>
<td>666</td>
</tr>
<tr>
<td>Gen1DBarcodeDescrGen</td>
<td>667</td>
</tr>
<tr>
<td>Gen2DBarcodeDescr</td>
<td>668</td>
</tr>
<tr>
<td>GenBandfilter</td>
<td>93</td>
</tr>
<tr>
<td>GenBandpass</td>
<td>94</td>
</tr>
<tr>
<td>GenCheckerRegion</td>
<td>458</td>
</tr>
<tr>
<td>GenCircle</td>
<td>460</td>
</tr>
<tr>
<td>GenContourPolygonXld</td>
<td>860</td>
</tr>
<tr>
<td>GenContourRegionXld</td>
<td>860</td>
</tr>
<tr>
<td>GenContoursSkeletonXld</td>
<td>861</td>
</tr>
<tr>
<td>GenCoocMatrix</td>
<td>327</td>
</tr>
<tr>
<td>GenDiscSe</td>
<td>381</td>
</tr>
<tr>
<td>GenEllipse</td>
<td>461</td>
</tr>
</tbody>
</table>

HALCON 6.0
GetWindowType, 272
GiveBgEsti, 651
Gnuplot, 193–196
GnuplotClose, 193
GnuplotOpenFile, 193
GnuplotOpenPipe, 194
GnuplotPlotCtrl, 194
GnuplotPlotFunct1D, 195
GnuplotPlotImage, 196
Golay-Alphabet, 392, 398, 400, 404, 409, 414, 415, 421, 422, 425, 431, 432, 434, 435
GolayElements, 409
GrabImage, 348
GrabImageAsync, 349
GrabImageStart, 350
GrabRegion, 351
GrabRegionAsync, 351
Graphic, 603, 606
graphic, 224, 227
GraphicModes (property), 983
Graphics, 209, 212, 215, 274, 276, 279, 284
graphics, 235, 236, 695
Graphics-Software, 272
Gray (property), 983
Gray-Channels, 289–299
Gray-Projections, 659, 662, 670
gray-scale-image, 48, 49
Gray-surface, 308
Gray-Value, 217, 219
Gray-Value-Feature, 317–323, 325, 326, 331, 333, 334, 866, 879
Gray-Value-Morphology, 383, 385, 386
Gray-value-scaling, 87
gray-value-spreading, 87
Gray-Value-Threshold, 541–543, 552, 557, 558, 575
Gray-values, 328, 330, 334, 357
GrayBothat, 381
GrayClosing, 382
GrayDilation, 383
GrayDilationRect, 383
GrayErosion, 384
GrayErosionRect, 385
GrayHisto, 328
GrayInside, 133
GrayOpening, 386
GrayProjections, 329
GrayRangeRect, 386
GraySkeleton, 134
GrayTophat, 387
gray-value, 224, 231, 232, 235, 237, 242, 247, 253
gray-value-interpolation, 230, 252
Grayvalues, 115
GrayValues (property), 983
Green (property), 983
Half-axis, 320, 483
Half-Images, 146
Hamming-distance, 488, 489
Hamming-Distance, 528
HammingChangeRegion, 528
HammingDistance, 488
HammingDistanceNorm, 489
HandEyeCalibration, 702
HBBarcodeX, 901
HBBgEstiX, 902
HClassBoxX, 902
Height-Field, 809, 811–814
height-plot, 247
help, 589, 591, 593, 597
Hesse, 470
HFeatureSetX, 903
HFileX, 903
HFramegrabberX, 904
HFunction1dX, 906
HGnuplotX, 907
HHistogramX, 907
HHomMat2dX, 907
HHomMat3dX, 909
High-Pass-Filter, 69
Highpass-Filter, 97
HighpassImage, 69
Hilbert-Transform, 89, 95
HImageX, 909
HInfoX, 920
Histo2Dim, 330
Histogram, 328, 330, 331, 336, 337, 470, 541, 542, 545, 546, 558
histogram, 247
Histogram-linearisation, 85
Histogram-spread, 85
HistoToThresh, 558
Hit-Or-Miss-Transformation, 409, 413–415, 429, 431–435
HitOrMiss, 413
HitOrMissGolay, 414
HitOrMissSeq, 415
HMeasureX, 921
HMiscX, 921
HNF, 755–757
HObjectX, 924
HOcrX, 925
HOCvX, 925
hole, 479, 485
Holes, 526, 527
HomMat2dCompose, 628
HomMat2dIdentity, 628
HomMat2dInvert, 629
HomMat2dRotate, 629
HomMat2dScale, 630
HomMat2dSlant, 631
HomMat2dToAffinePar, 632
HomMat2dTranslate, 633
HomMat3dCompose, 634
HomMat3dIdentity, 635
HomMat3dInvert, 636
HomMat3dRotate, 637

Index

HomMat3dScale, 639
HomMat3dToPose, 707
HomMat3dTranslate, 640
Homogeneous-Coordinates, 16, 27, 454, 625, 626, 628–637, 639, 640, 887, 888
homogeneous-coordinates, 16, 27, 454, 625, 626, 628–637, 639, 640, 887, 888
Hopfield, 488, 489
HorizontalResolution (property), 905
Hough-Transform, 754–757
HoughCircles, 754
HoughCircleTrans, 754
HoughLines, 756
HoughLineTrans, 755
HPoseX, 962
HRegionX, 963
HSerialX, 971
HShapeModelIX, 971
HSocketX, 972
HSystemX, 973
HTemplateX, 974
HTupleX, 974
Hue (property), 984
Hull, 450, 490, 509, 511
hull, 255
HUntypedObjectX, 978
HWindowX, 978
HXLDContX, 985
HXLDExtParaX, 987
HXLDModParaX, 987
HXLDParaX, 988
HXLDPolyX, 988
HXLDX, 985
Hyper-Cube, 548, 561
Hyper-Cuboid, 547, 560
Hyper-Sphere, 548, 561
Hyperbolic-Cosine, 820
Hyperbolic-Sine, 826
Hyperbolic-Tangent, 827
Hysteresis, 559
Hysteresis-Thresholding, 62, 65
HysteresisThreshold, 559
Iconic-Object, 299, 443–447, 581
Illuminate, 86
illumination, 591
Insert (property), 984
Inside, 133
InvertImage, 36
iteration, 175–183, 185–188, 190–192
Intersection, 516
IntersectionLl, 751
InvarFourierCoeff, 723
Inversion, 36
Iteration, 142
JPEG, 11, 14, 269
Junctions, 530
JunctionsSkeleton, 530
Kalman-filter, 645, 646, 649, 651–653, 655, 758, 762, 765, 766
keyword, 591, 598
Kirsch-Filter, 71, 72
KirschAmp, 71
KirschDir, 72
kKalmanfilter, 647
Label-Image, 312, 476
LabelToRegion, 476
Lanser-Filter, 62, 65
Laplace, 73
Laplace-Filter, 58, 61, 73, 75
Laplace-of-Gaussian, 58, 61, 75
Laplace-Operator, 550
LaplaceOfGauss, 75
Laws-Filter, 163
Ldexp, 822
LearnClassBox, 5
LearnNdImBox, 560
LearnNdImNorm, 561
LearnSampsetBox, 6
Least-Squares-Adjustment, 730
Length, 857–859
LengthXld, 880
Level-Crossings, 576
Licence, 582
Light-Source, 806–809, 811–814
Line, 367, 874, 894
line, 182
Line-Extraction, 103–105
Line-feed, 19
Line-Length, 373, 375
Line-length, 376
Line-segments, 874, 894
line-width, 247
LineApprox (property), 984
LineIn (property), 905
LineOrientation, 371
LinePosition, 372
Lines, 370–373, 375, 376, 463, 465, 470, 471, 538, 539, 549, 857
LinesFacet, 104
LinesGauss, 105
LineWidth (property), 984
LoadParKnowledge, 602
Local-Maximum, 563, 564, 880
Local-Threshold, 543, 552
LocalMax, 562
LocalMaxContoursXld, 880
LoG, 58, 61, 75
Logarithm, 823
Logical-Operations, 845, 846
Look-Up-Table, 134, 197–201, 204, 205, 225, 241
Lowpass-Filter, 97
LUT, 134, 197–201, 204, 205, 225, 241, 346, 356, 603
lut, 253
lut (property), 984
Luts (property), 984
LutTrans, 134
Main-axes-of-inertia, 491–497
Main-radius, 504
Manipulation, 840, 841
Manual, 590
manual, 589, 597
Map, 27, 29, 31–33, 454, 627, 642–645, 887, 888
Mask, 44
MatchFourierCoeff, 724
MatchFunct1DTrans, 730
Matrices, 583
Maxima, 562, 565, 566
MaxImage, 37
Maximum, 37, 126, 134, 331, 334, 383, 736, 737, 843
MaxParallelsXld, 881
Mean, 330, 334, 734
Mean-Filter, 143, 145, 149, 151
MeanImage, 149
MeanN, 150
MeanSp, 151
Measurement, 775, 777, 779, 781, 783, 784
measurements, 758, 762, 765, 766
MeasurePairs, 779
MeasurePos, 781
MeasureProjection, 783
MeasureThresh, 784
Medial-Axis, 536
Median, 533
Median-Filter, 143, 152–156, 160, 379
MedianImage, 152
MedianSeparate, 153
MedianWeighted, 154
Memory, 603
MergeContLineScanXld, 892
MergeRegionsLineScan, 531
Metric, 523
Mexican-Hat-Operator, 58, 61, 75
Midrange-Filter, 155
MidrangeImage, 155
Minima, 541, 542, 558
MinImage, 38
Minimum, 38, 126, 331, 334, 385, 736, 737, 745, 746, 844
Minkowski-Addition, 416, 417, 423–427
Minkowski-Subtraction, 390–392, 418, 419
MinkowskiAdd1, 416
MinkowskiAdd2, 417
MinkowskiSub1, 418
MinkowskiSub2, 419
MinMaxGray, 331
MirrorImage, 29
MirrorRegion, 455
ModParallelsXld, 866
Modules, 582, 658
Moments, 320, 322, 323, 333, 482, 483, 491–497, 882, 883
MomentsAnyXld, 882
MomentsGrayPlane, 333
MomentsRegion2Nd, 491
MomentsRegion2NdInvar, 492
MomentsRegion2NdRelInvar, 493
MomentsRegion3Rd, 494
MomentsRegion3RdInvar, 495
MomentsRegionCentral, 496
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MomentsRegionCentralInvar</td>
<td>497</td>
</tr>
<tr>
<td>MomentsXld</td>
<td>883</td>
</tr>
<tr>
<td>Monotony</td>
<td>126</td>
</tr>
<tr>
<td>Monotony</td>
<td>126</td>
</tr>
<tr>
<td>MorphHat</td>
<td>421</td>
</tr>
<tr>
<td>MorphSkeleton</td>
<td>421</td>
</tr>
<tr>
<td>MorphSkiz</td>
<td>422</td>
</tr>
<tr>
<td>Motion-Blur</td>
<td>166, 169, 170, 172</td>
</tr>
<tr>
<td>Mouse</td>
<td>205–208</td>
</tr>
<tr>
<td>MoveContourOrig</td>
<td>725</td>
</tr>
<tr>
<td>movement</td>
<td>175–177</td>
</tr>
<tr>
<td>MoveRectangle</td>
<td>273</td>
</tr>
<tr>
<td>MoveRegion</td>
<td>456</td>
</tr>
<tr>
<td>multichannel</td>
<td>211</td>
</tr>
<tr>
<td>MultiImage</td>
<td>39</td>
</tr>
<tr>
<td>Multiplication</td>
<td>39, 733, 824</td>
</tr>
<tr>
<td>Name (property)</td>
<td>905</td>
</tr>
<tr>
<td>NegateFunct1D</td>
<td>731</td>
</tr>
<tr>
<td>Negation</td>
<td>731, 824</td>
</tr>
<tr>
<td>Neighborhood</td>
<td>125, 485</td>
</tr>
<tr>
<td>neighborhood</td>
<td>502, 512</td>
</tr>
<tr>
<td>NewExternWindow</td>
<td>274</td>
</tr>
<tr>
<td>NewLine</td>
<td>259</td>
</tr>
<tr>
<td>Noise</td>
<td>138–141, 170, 172, 886</td>
</tr>
<tr>
<td>noise</td>
<td>143</td>
</tr>
<tr>
<td>Noise-Distribution</td>
<td>138–141, 886</td>
</tr>
<tr>
<td>Noise-Removal</td>
<td>535</td>
</tr>
<tr>
<td>NoiseDistributionMean</td>
<td>140</td>
</tr>
<tr>
<td>Non-linear-Filter</td>
<td>142</td>
</tr>
<tr>
<td>Non-Maximum-Suppression</td>
<td>62, 563, 564</td>
</tr>
<tr>
<td>NonmaxSuppressionAmp</td>
<td>563</td>
</tr>
<tr>
<td>NonmaxSuppressionDir</td>
<td>564</td>
</tr>
<tr>
<td>not, 44</td>
<td></td>
</tr>
<tr>
<td>NumPointsFunct1D</td>
<td>731</td>
</tr>
<tr>
<td>Object</td>
<td>219, 442, 445</td>
</tr>
<tr>
<td>object, 12</td>
<td></td>
</tr>
<tr>
<td>Object-choice</td>
<td>486, 501</td>
</tr>
<tr>
<td>Object-comparison</td>
<td>441</td>
</tr>
<tr>
<td>Object-Key</td>
<td>445, 446</td>
</tr>
<tr>
<td>Object-Selection</td>
<td>334</td>
</tr>
<tr>
<td>ObjToInteger</td>
<td>446</td>
</tr>
<tr>
<td>OCR, 794, 795</td>
<td></td>
</tr>
<tr>
<td>OCRChangeChar</td>
<td>792</td>
</tr>
<tr>
<td>OcrGetFeatures</td>
<td>793</td>
</tr>
<tr>
<td>online-text</td>
<td>591, 598</td>
</tr>
<tr>
<td>OpenFile</td>
<td>21</td>
</tr>
<tr>
<td>OpenFramegrabber</td>
<td>354</td>
</tr>
<tr>
<td>Opening</td>
<td>379, 386, 387, 408, 421, 423–427, 436</td>
</tr>
<tr>
<td>Opening, 423</td>
<td></td>
</tr>
<tr>
<td>OpeningCircle</td>
<td>424</td>
</tr>
<tr>
<td>OpeningGolay</td>
<td>425</td>
</tr>
<tr>
<td>OpeningRectangle</td>
<td>426</td>
</tr>
<tr>
<td>OpeningSeg</td>
<td>427</td>
</tr>
<tr>
<td>OpenSerial</td>
<td>613</td>
</tr>
<tr>
<td>OpenSocketAccept</td>
<td>617</td>
</tr>
<tr>
<td>OpenSocketConnect</td>
<td>618</td>
</tr>
<tr>
<td>OpenTextwindow</td>
<td>276</td>
</tr>
<tr>
<td>OpenWindow</td>
<td>279</td>
</tr>
<tr>
<td>Optical-Flow</td>
<td>127, 365</td>
</tr>
<tr>
<td>OpticalFlowMatch</td>
<td>127</td>
</tr>
<tr>
<td>Orientation</td>
<td>300, 371–373, 375, 461, 468, 483, 498, 511, 659</td>
</tr>
<tr>
<td>Orientatio</td>
<td>504</td>
</tr>
<tr>
<td>Out-Of-Focus-Blur</td>
<td>165, 168, 170, 172</td>
</tr>
<tr>
<td>Outer-circle</td>
<td>504</td>
</tr>
<tr>
<td>Outliers</td>
<td>893</td>
</tr>
<tr>
<td>Output</td>
<td>209, 211, 212, 214, 215, 217, 219, 220, 222</td>
</tr>
<tr>
<td>output</td>
<td>224, 228, 229, 264, 695</td>
</tr>
<tr>
<td>output-parameter</td>
<td>595</td>
</tr>
<tr>
<td>Overlap</td>
<td>506</td>
</tr>
<tr>
<td>paint</td>
<td>186</td>
</tr>
<tr>
<td>Paint (property)</td>
<td>984</td>
</tr>
<tr>
<td>PaintGray</td>
<td>361</td>
</tr>
<tr>
<td>PaintModes (property)</td>
<td>984</td>
</tr>
<tr>
<td>PaintRegion</td>
<td>362</td>
</tr>
<tr>
<td>Parallelization</td>
<td>600, 602</td>
</tr>
<tr>
<td>Parallels</td>
<td>858, 864, 866, 879, 881, 890</td>
</tr>
<tr>
<td>parameter</td>
<td>229, 591, 593, 595, 596, 598</td>
</tr>
<tr>
<td>parameter-number</td>
<td>596</td>
</tr>
<tr>
<td>PartColumn1 (property)</td>
<td>984</td>
</tr>
<tr>
<td>PartColumn2 (property)</td>
<td>984</td>
</tr>
<tr>
<td>Partition</td>
<td>532</td>
</tr>
<tr>
<td>PartitionDynamic</td>
<td>532</td>
</tr>
<tr>
<td>PartitionLines</td>
<td>373</td>
</tr>
<tr>
<td>PartitionRectangle</td>
<td>532</td>
</tr>
<tr>
<td>PartRow1 (property)</td>
<td>984</td>
</tr>
<tr>
<td>PartRow2 (property)</td>
<td>984</td>
</tr>
<tr>
<td>PartStyle (property)</td>
<td>984</td>
</tr>
<tr>
<td>Pattern</td>
<td>458, 463, 801–805</td>
</tr>
<tr>
<td>Perimeter</td>
<td>326</td>
</tr>
<tr>
<td>Phase, 100</td>
<td></td>
</tr>
<tr>
<td>PhaseDeg</td>
<td>100</td>
</tr>
<tr>
<td>PhaseRad</td>
<td>100</td>
</tr>
<tr>
<td>Photometric-Stereo</td>
<td>809</td>
</tr>
<tr>
<td>PhotStereo</td>
<td>809</td>
</tr>
<tr>
<td>Pixel, 451, 472</td>
<td></td>
</tr>
<tr>
<td>Pixel (property)</td>
<td>984</td>
</tr>
<tr>
<td>Pixel-Relations</td>
<td>603, 606</td>
</tr>
<tr>
<td>PixelTypes</td>
<td>12</td>
</tr>
<tr>
<td>Pixels</td>
<td>463, 465</td>
</tr>
<tr>
<td>PlaneDeviation</td>
<td>334</td>
</tr>
<tr>
<td>Plateaus</td>
<td>565</td>
</tr>
<tr>
<td>Plateaus</td>
<td>565</td>
</tr>
<tr>
<td>PlateausCenter</td>
<td>565</td>
</tr>
<tr>
<td>plot</td>
<td>247</td>
</tr>
<tr>
<td>point, 185</td>
<td></td>
</tr>
<tr>
<td>Polar-Transformation</td>
<td>29</td>
</tr>
<tr>
<td>PolarTransImage</td>
<td>29</td>
</tr>
<tr>
<td>Polygon, 220, 473, 474, 498</td>
<td></td>
</tr>
<tr>
<td>polygon, 186</td>
<td></td>
</tr>
<tr>
<td>Polygon-Approximation</td>
<td>865</td>
</tr>
</tbody>
</table>
Index

Polygon-approximation, 452
polygon-approximation, 245
Polygon-Length, 880
Polygons, 857–860, 865, 881, 888, 890, 896
Polyline, 220
Port (property), 905
Pose-relation, 502, 512
PoseToHomMat3d, 710
PostScript, 269
Pouring, 566
Power-Function, 824
Power-Spectrum, 100–102
Principal-Components, 135
PrincipalComp, 135
procedure, 596, 597
Procedure-Chapters, 590
procedure-name, 593
Product-of-inertia, 491–497
Project3DPoint, 711
ProjectionPl, 753
Pruning, 428
Pyramid, 121
QueryAllColors, 233
QueryFont, 259
QueryGray, 235
QueryLut, 200
QueryMshape, 208
QueryOperatorInfo, 597
QueryPaint, 237
QueryParamInfo, 598
QueryShape, 237
QuerySpy, 235
QueryTshape, 260
QueryWindowType, 283
Radians, 825, 835
Radius, 320, 460, 461, 483
Random-pattern, 464
Range, 331, 386
Range-Filter, 155
Rank-Filter, 152, 154–156, 160, 379
Rank-Gray-Value, 152, 154–156, 160, 379
Rank-Operator, 533
RankImage, 156
RankRegion, 533
ReadCamPar, 712
ReadChar, 260
ReadCalibrationBox, 7
ReadContourXldArcInfo, 23
ReadFunct1D, 732
ReadGrayScale, 388
ReadImage, 11
ReadKalman, 762
ReadOCR, 794
ReadOCRTrainf, 794
ReadOCRTrainfNames, 795
ReadOCRTrainfSelect, 795
ReadOcv, 803
ReadPolygonXldArcInfo, 24
ReadPose, 714
ReadRegion, 16
ReadSampleSet, 7
ReadSequence, 12
Red, (property), 984
ReduceDomain, 317
reference, 591
Reflection, 29, 455, 456
region, 175–177, 192, 225, 232, 237
Region-choice, 506
Region-growing, 567, 569, 570
Region-growing, 567
Region-growingMean, 569
Region-growingN, 570
RegionToBin, 311
RegionToLabel, 312
RegionToMean, 313
RegressContoursXld, 893
Regression, 874, 876, 878, 880, 885, 886, 891, 893, 896, 897, 899
Rekursive-Filters, 62, 65
Relation, 581
Relations, 583
Remainder, 822
RemoveNoiseRegion, 535
ResetObjDb, 583
Resolution-Pyramid, 121
Restoration, 165, 166, 168, 169
return-value, 591
RGB, 48, 49, 53
Rgb1ToGray, 48
Rgb3ToGray, 49
Ridge, 126
Roberts, 78
Robinson-Filter, 79, 80
RobinsonAmp, 80
RotateImage, 31
Rotation, 31, 629, 631, 632, 637
Roundness, 478, 479
Roundness, 498
Run, 500, 524
RunBgEsti, 652
Runlength-Code, 606
Runlength-Coding, 603
runlength-encoding, 524
RunlengthDistribution, 500
RunlengthFeatures, 500
Runtime, 599
SampleFunct1D, 732
Saturation (property), 984
Save, 803, 805
ScaleImage, 40
ScaleImageMax, 87
ScaleYFunct1D, 733
Scaling, 32, 33, 40, 630, 639
scaling, 87
SearchOperator, 598
Secondary-radius, 504
Segmentation, 351, 367, 370, 476, 518, 545–548, 566, 576, 894
segmentation, 697
SegmentContoursXld, 894
SelectContoursXld, 885
SelectGray, 334
SelectGrayvaluesFromChannels, 810
Selection, 444, 446, 447, 847–849
SelectLines, 375
SelectLinesLongest, 376
SelectMatchingLines, 757
SelectObj, 447
SelectRegionPoint, 501
SelectRegionSpatial, 502
SelectShape, 504
SelectShapeProto, 506
SelectShapeStd, 508
SendImage, 621
SendRegion, 621
SendTuple, 622
SendXld, 622
SensorKalman, 765
Separating-Lines, 529
Serial-Interface, 610–615
SetBgEstiParam, 653
SetCheck, 586
SetClassBoxParam, 8
SetColor, 238
SetColored, 239
SetComprise, 239
SetDraw, 240
SetFix, 241
SetFixedLut, 201
SetFont, 262
SetFramegrabberLut, 356
SetFramegrabberParam, 357
SetGray, 242
SetGrayval, 363
SetHsi, 243
SetIcon, 244
SetInsert, 244
SetLineApprox, 245
SetLineStyle, 246
SetLineWidth, 247
SetLut, 201
SetLutStyle, 204
SetMshape, 208
SetOffsetTemplate, 129
SetOriginPose, 715
SetPaint, 247
SetPart, 251
SetPartStyle, 252
SetPixel, 253
SetReferenceTemplate, 130
SetRgb, 254
SetSerialParam, 614
SetShape, 255
SetSpy, 587
SetSystem, 606
SetTposition, 263
SetTshape, 264
SetWindowAttr, 283
SetWindowDc, 284
SetWindowExtents, 285
SetWindowType, 286
SfsModLr, 811
SfsOrigLr, 812
SfsPentland, 813
ShadeHeightField, 814
Shading, 809, 811–814
Shadows, 814
Shape, 506
Shape (property), 984
Shape-Feature, 880, 882, 883
Shape-feature, 478, 479, 481–483, 488, 489, 491, 498, 508
Shape-Features, 527
Shape-from-Shading, 809, 811–813
Shape-Transformation, 535
ShapeHistoAll, 336
ShapeHistoPoint, 337
ShapeTrans, 535
Sharpness, 84
Shen-Filter, 62, 65, 159
Shift, 43, 46
Sigma-Filter, 158
SigmaImage, 158
Similarity, 508, 728
SimulateDefocus, 168
SimulateMotion, 169
Sine, 98, 99, 825
Skeleton, 421, 422, 428, 433–435, 536, 861
Skeleton, 536
Slant, 807, 808
SlideImage, 287
Slope, 126
SmallestCircle, 509
SmallestRectangle1, 510
SmallestRectangle2, 511
SmoothContoursXld, 896
SmoothFunct1DGauss, 734
SmoothFunct1Dmean, 734
SmoothImage, 159
Smoothing, 58, 142, 143, 145–149, 151–156, 158–160, 379, 734, 896
Sobel-Filter, 81, 82
SobelAmp, 81
SobelDir, 82
SocketAcceptConnect, 623
Sockets, 616–623
SortRegion, 537
Spatial-Domain, 90, 92, 94
SpatialRelation, 512
SpDistribution, 141
Split-And-Merge, 538, 539
SplitContoursXld, 896
SplitSkeletonLines, 538
SplitSkeletonRegion, 539
spreading, 87
Square-Root, 826
Standard-Deviation, 162, 842, 866, 878, 879, 885
StartColumn (property), 905
StartRow (property), 905
Store, 803, 805
StoreParKnowledge, 602
Storing-Capacity, 606
String, 20, 21
String-Operators, 849–854
Structure-factor, 482
Structuring-Elements, 381, 388
SubImage, 41
Subtraction, 41, 826
Sum, 845
Surrogate, 442, 445, 446
Surrounding-circle, 504, 509
surrounding-circle, 255
Surrounding-rectangle, 504, 510, 511
Symmetry, 136
Symmetry, 136
System-Call, 599
System-Parameter, 603, 606
SystemCall, 599
systems, 758, 762, 766
Tangent, 827
Test-pixel, 486, 501, 513
TestdOcrClassBox, 796
TestEqualObj, 441
TestEqualRegion, 441
TestObjDef, 442
TestRegionPoint, 513
TestSampsetBox, 9
Text, 276
text, 256–262, 264
text-cursor, 258, 260, 264
Text-file, 18–21
text-position, 263
text-size, 256
Texture, 153, 162, 163, 318, 319, 321, 327
Texture-Transformation, 163
TextureLaws, 163
Thickening, 429, 431, 432
Thickening, 429
ThickeningGolay, 431
ThickeningSeq, 432
Thinning, 134, 421, 422, 433–435, 536
Thinning, 433
ThinningGolay, 434
ThinningSeq, 435
Threshold, 336, 337, 541–543, 550, 552, 557–559, 575, 784
Threshold, 575
Thresholding, 541, 542, 558
ThresholdSubPix, 576
TIFF, 11, 269
Tiff, 14, 16
TileChannels, 341
TileImages, 342
TileImagesOffset, 344
Tiling, 341, 342, 344
Tilt, 808, 809
Time-Measure, 599
Top-Hat, 387, 421, 436
TopHat, 436
Topographic-Primal-Sketch, 137
TopographicSketch, 137
Trace, 585
TraindOcrClassBox, 797
TraindOcvProj, 804
TrainfOcrClassBox, 798
Training, 560, 561, 794, 795, 804
Transformation, 27, 29, 31–33, 134, 454, 523, 642–645, 730, 732, 735, 887, 888
transformation, 709
Transformation-Matrix, 27, 454, 627, 642–645, 887, 888
TransformFunct1D, 735
TransFromRgb, 49
Translation, 456, 633, 640
TransposeRegion, 456
Transposition, 456
TransToRgb, 49
TrimmedMean, 160
turecolor, 247
Tupple, 445, 462, 817–854
Tuple-Length, 843
tuple-length, 439
TupleAbs, 817
TupleAcos, 817
TupleAdd, 845
TupleAsin, 818
TupleAtan, 819
TupleAtan2, 819
TupleBand, 828
TupleBnot, 828
TupleBor, 829
TupleBxor, 829
TupleCeil, 841
TupleChr, 834
TupleChrt, 834
TupleConcat, 839
TupleCos, 820
TupleCosh, 820
TupleDeg, 835
TupleDeviation, 842
TupleDiv, 820
TupleEnvironment, 850
TupleEqual, 831
TupleExp, 821
TupleFabs, 821
TupleFirstN, 847
TupleFloor, 842
TupleFmod, 822
TupleGenConst, 840
TupleGreater, 831
TupleGreaterEqual, 832
TupleInverse, 840
TupleIsNumber, 843
TupleLastN, 848
TupleLdexp, 822
TupleLength, 843
TupleLess, 832
TupleLessEqual, 833
TupleLog, 823
TupleLog10, 823
TupleLsh, 830
TupleMax, 843
TupleMean, 844
TupleMin, 844
TupleMult, 824
TupleNeg, 824
TupleNot, 846
TupleNotEqual, 834
TupleNumber, 835
TupleOr, 846
TupleOrd, 836
TupleOrds, 836
TuplePow, 824
TupleRad, 825
TupleReal, 837
TupleRound, 837
TupleRsh, 830
TupleSelect, 848
TupleSelectRange, 849
TupleSin, 825
TupleSinh, 826
TupleSort, 840
TupleSortIndex, 841
TupleSplit, 850
TupleSqrt, 826
TupleStrBitSelect, 849
TupleStrchr, 852
TupleStrFirstN, 851
TupleString, 838
TupleStrLastN, 851
TupleStrlen, 853
TupleStrrchr, 853
TupleStrrstr, 854
TupleStrstr, 854
TupleSub, 826
TupleSum, 845
TupleTan, 827
TupleTanh, 827
TupleXor, 846
type, 593
Type (property), 985
Type-Conversion, 364–366
types, 596

Union, 516, 517
Union1, 516
Union2, 517
UnionStraightContoursHistoiXld, 897
UnionStraightContoursXld, 899
update-file, 766
UpdateBgEsti, 655
UpdateKaiman, 766
VectorAngleToRigid, 642
VectorToHomMat2d, 643
VectorToRigid, 644
VectorToSimilarity, 645
Verification, 801–805
VerticalResolution (property), 905
WaitSeconds, 600
Watersheds, 133, 576
Watersheds, 576
Weighting, 154
Wiener-Filter, 165, 166, 170, 172
WienerFilter, 170
WienerFilterNi, 172
Window, 267, 274, 276, 279, 283, 284, 603, 606
Window-Dump, 269
Window-Position, 285
Window-Size, 270, 285
Window-Types, 283
World-File, 16, 23–26
WriteCamPar, 717
WriteClassBox, 10
WriteContourXldArcInfo, 25
WriteFunct1D, 736
WriteImage, 14
WriteLut, 205
WriteOcr, 798
WriteOcrTrainf, 799
WriteOcrTrainfImage, 800
WriteOcv, 805
WritePolygonXldArcInfo, 26
WritePose, 719
WriteRegion, 17
WriteSerial, 615
WriteShapeModel, 774
WriteString, 264
WriteTemplate, 130
WriteTuple, 23

X-Window, 603, 606
XLD-contours, 683
XLD-Parallels, 224, 620, 622, 858, 864, 866, 879, 881, 890
XLD-Polygons, 24, 26, 224, 620, 622, 857–859, 865, 881, 888, 890, 896
xor, 47
XRangeFunct1D, 736
YRangeFunct1D, 737

Zero-Crossings, 550, 577, 578
ZeroCrossing, 577
ZeroCrossingSubPix, 578
ZoomImageFactor, 32
ZoomImageSize, 33
Zooming, 457
ZoomRegion, 457