

# Übungen zum Praktikum Control of Robot Manipulators

## Player and Gazebo

### Available computers

For the exercises on this slide you can use the computers in our Terminalraum2 (02.05.014). You can get logins and passwords from Andreas Fedrizzi in room 02.09.039. Please change your password after your first login with the linux command 'passwd'. Also make sure that you use a computer with a nVidia graphics card, since ATI does not provide the hardware support that Gazebo needs. The simulation is very slow in this case. To figure out whether your computer has a nVidia card or not, use the 'lspci' command. If the line showing the 'VGA compatible controller' has a nVidia card entry, you are lucky.

### Setting up the environment

To be able to develop own programs for Player or for changing existing programs, you need to install some source code files. You will find the file kibo.tar.gz on the webpage. The following procedure will uncompress the file, compile the source files, compile the world files and start Gazebo and Player.

- Uncompress the tar archive

```
cd ~
tar xvfz kibo.tar.gz
```
- Compile the source files (this can take some minutes)

```
cd kibo/pg-sim/src
make
```
- Set the environment variables so that the binaries and libraries are found  
Open the file .bashrc - which is located in your home directory - in your favorite text editor and add the following lines:

```
export PLAYERPATH=${HOME}/kibo/pg-sim/lib/${ARCH_DIR}/player-plugins
export PATH=${HOME}/kibo/pg-sim/bin/${ARCH_DIR}:${PATH}
```
- Make the new environment variables available in this shell session

```
source ~/.bashrc
```
- Compile the world files

```
cd ~/kibo/worlds/empty_world
make
```

Now that your installation is complete, you can start Gazebo and Player and try to move the robot.

- Start Gazebo - you will see the B21 robot standing in an empty world. The switch -v 2.0 makes Gazebo run faster than real time.

```
cd ~/kibo/worlds/empty_world/
wxgazebo default.world -v 2.0
```
- Open a new shell to start Player

```
cd ~/kibo/worlds/empty_world/
player default.config
```

- Open a new shell to execute a player-program - this makes the robot lift its arm into a horizontal position  
`pi-actarray -i 119 -p -0 1.5708`

`pi-actarray` is a player-program which can be used to communicate with a manipulator (`actarray` is an abbreviation for `actuator-array`). You can find the source code for this program in the file `~/kibo/pg-sim/src/player-progs/actarray/actarray.c`. `pi-actarray` takes many parameters, which are listed when you execute `'pi-actarray -?'`. Some important parameters are:

Parameter name	Parameter description
-i	index number of the player-device that receives the command (119 is the right arm, 129 is the left arm)
-m	multi command - if one of the joint parameters is not specified, its value is set to 0.0 (default is that joints keep their angle if unspecified)
-h	home command - when used together with -m, this moves the manipulator to its home position by setting every joint angle to 0.0
-p	specifies a position command - following joint-parameters will specify the absolute goal angle of that joint
-s	specifies a speed command - following joint-parameters will specify the speed at which that joint should rotate
-c	specifies a current command - following joint-parameters will specify the current that is applied to the joint motor
-d	following angles are specified in degrees (default is radians)
-0	joint-parameter for joint 1 (joint numeration begins with 0!)
-1	joint-parameter for joint 2
..	...
-5	joint-parameter for joint 6

So we see that the command above was a position command, where the first joint should turn to a goal angle of 1.5708 radians.

The difference between a normal (no extra parameter) and a multi command (-m) is that the normal command leaves joint parameters of unspecified joints untouched, while the multi command sets joint parameters of unspecified joints to 0.0. When you enter the following command you will see, that the robot will set the joint angles of joint 2, 3, and 5 to the specified values, while setting the angles of the other joints to 0.0. Try the following command and see that the first joint moves, too.

```
pi-actarray -i 119 -m -p -d -1 -60.0 -2 90.0 -4 -90.0
```

## Adding new programs to player-prog

In the exercise you will have to write a new player-program. All player-programs are located in the directory `~/kibo/pg-sim/src/player-progs`. For adding a new player-prog you need to

- add a new directory (e.g. `exercise02`)
- add this new directory to the makefile in `~/kibo/pg-sim/src/player-progs/makefile`
- create a player-program in your new directory (you can start by copying another player-prog like `actarray.c`)
- copy a makefile from a different player-prog directory to your directory
- patch the makefile so that it fits to your player-program. If you do not require additional libraries, you usually have to change just the first two lines. So if your player-program for the exercise02 task 1a) has the name `ex2.ta1a.c`, the first two lines in the makefile would look like:  
`BIN_NAME := pi-ex2.ta1a`  
`SOURCE := ex2.ta1a.c`
- to compile your player-program, just type `make` from the program's directory
- now you can call your program `pi-ex2.ta1a` with the required parameterization

## Homework

1. The goal of this exercise is to execute a simple player program and become familiar with Gazebo and Player. You will have to use the player-program `pi-actarray` and modify it. A very useful resource for this and future work with Player will be the manual page at <http://playerstage.sourceforge.net/doc/Player-cvs/player/index.html>. When you need to investigate which attributes a data structures like `playerc_actarray_t` has, this is the place to search. The above data structure could be found under “Client libraries” -> “libplayerc” -> “Device proxies” -> “actarray”. Data structures that start with `player_xyz` instead of `playerc_xyz` can be found under “libplayercore” -> “interface specifications”. Another possibility is to download the source code of player and search for the headers `playerc.h` and `player.h`.
  - (a) A homing command is a command that sets every joint angle back to  $0^\circ$ . Write a homing command, where a manipulator is moved from an arbitrary position to its home position. The sequence should move the last (sixth) joint first, then the fifth, and so forth until it has homed the first joint. There is a homing command already implemented in `pi-actarray`. The problem with the homing command of `pi-actarray` is that it moves all joints in parallel. This can lead to collisions, if executed on the real robot, so we need our more robust homing command! You can use `actarray.c` as a starting point and patch it so that it executes the desired motions. Note that you have to constantly read from the message queue with `playerc_client_read(client)`; in order to receive current data. Otherwise the player server will fill up your message queue and you will just get old data, that does not correspond with the current joint angles of the manipulator.
  - (b) Write a movement-sequence where the robot moves the first joint from  $0^\circ$  to  $+90^\circ$ . After that move the joints from joint 2 to joint 6 from  $0^\circ$ , to  $+90^\circ$  and back to  $0^\circ$ . After this sequence is executed, move the first joint back to  $0^\circ$ . The user should be able to specify on the command line whether the left or right manipulator is used to execute the movement sequence. The expression `actarray->actuators_data[jj].position` gives you the current angle of joint `jj+1` and can be used to check if the target angle is already achieved.
  - (c) Add another movement sequence that you think is cool. The only constraint is that the movement sequence should move both arms in parallel.
  - (d) In which space do you specify the movement goal, when you execute a command like `pi-actarray -i 119 -m -p -d -0 30.0 -1 -60.0 -2 90.0 -3 -40.0 -4 -90.0 -5 -15.0`  
Hint: it is one of the notations from slide “Descriptive power of different notations” in lecture 1.
  - (e) The origin of the B21 mobile roboter is called B21 base and is located in the middle and at the bottom of the B21. The x-axis is going to the front, y is going to the left and z is going upwards. You want the robot to grasp a cup and the vision system computes that the cup is 0.5m in front of the robot and 0.8m above the ground. Do you think `pi-actarray` would be a good program to easily grasp the cup? Which of the kinematic methods that were presented in lecture 1 would be required to get useful input for `pi-actarray`? Explain your answer.