# Programmierung und Regelung für Mensch-Roboterinteraktion

Programming of a 7DOF compliant manipulator KUKA LBR IIWA with Sunrise.OS

D. Seidel, Prof. A. Albu-Schäffer

June 28, 2014

## 1 Introduction

In this practical work you will use a KUKA LBR IIWA industrial lightweight robot to test the concepts known from theory and simulation on real hardware. The most important thing to bear in mind is that this robot is a powerful machine, that can cause severe damage and injuries on misuse. One has to operate it **always** with the appropriate care and full concentration.

### Controlling the Robot and Safety

The robot can be operated in three different modes: *T1*, *T2* and *automatic.* T1 and T2 require the user to press one of the three enabling switches on the Smartpad for any motion to be executed. T1 reduces the velocity of any point of the robot to 250 mm/s, while T2 allows to operate the robot at full speed. Automatic mode operates at full speed without the need to press an enabling switch. We will operate the robot **always** in T1 mode.

The following rules are **mandatory** when operating the robot:

- The robot is to be operated by at least two persons. The person holding the Smartpad must focus on the robot alone. A second person should hold the external emergency stop and also closely monitor the behavior of the robot for abnormalities.
- In the occurrence of abnormalities or unexpected movements do not hesitate to release the enabling switch or press the emergency stop. A movement can still be continued on false alarm.

Every program you start will launch a Control GUI that allows to pause/resume movements, abort them and execute some other functionality. However, after aborting sometimes the program looses the control. In this case only a program restart helps.

### Configuration

To be able to communicate with the robot controller, the network adapter needs to be configured with a static IP address. Use `192.168.42.x`, where x is any number but 2. Other than that, the software framework needs only one path to be configured. In the `RobotControl` project the class `data.DataHandler` has got the field `DataPath`, which defaults to `$HOME/RobotExercises`. The folder specified in this field is used to store and load data during the exercises.

## 2 Exercise 1 - Getting in touch with the robot

This exercise will teach you the basics of how to use the framework and command simple motions to the robot. For this, the robot will first be started in a pseudo gravitation compensation mode that allows to move the robot around by physical guidance. Using this mode you will move the robot to different locations of the workspace and record some frames. Afterwards you will use these frames to command motions to the robot.

**CAREFUL: there is no physical constraint that prevents moving past joint limits. If a joint limit is exceeded the brakes are activated immediately and the system halts with a safety stop. It can only be reactivated by manually moving the joint out of the limit and remastering it. Therefore, you should monitor the joint axis on the smartpad and prevent moving into joint limits!**

**Task 1:** *Start the program* `Exercise1a_TeachIn.java`*. As soon as the enabling switch is pressed the robot will be in a floating movement and you can start moving it around.*

1. Move the robot to different locations in the workspace.
2. Record at least four different locations using the Control GUI. For each location, record the frame and the joint position. Try to record joint positions with large difference, i.e. positions with the elbow at the top and another one with the elbow turned to one side.

**Task 2:** *Open the program* `Exercise1b_BasicMotions.java`*. Modify the program to execute PTP and LIN movements.*

1. Load some of the frames and joint positions you recorded in Task 1. Use the DataHandler class to load the joint positions and frames.
2. Create PTP movements to the joint positions or frames.
3. Create PTP movements and add blending to the motions. Note, that you have to use the moveAsync() commant to allow the controller to blend over the motions.
4. Create LIN movements to the frames without/with blending.
5. Create LINREL movements. LINREL movements can be parameterized with a reference frame. This way the relative movement can be specified in the world coordinate system instead of the current gripper orientation. For the world frame you can use `robot.getRootFrame()`, which in this scenario is identical to the world frame.
6. Create a motion batch with some of the previous motions and execute it.
7. Take a combination of two locations recorded with different elbow configurations. Create a PTP movement to the joint position of the first location and then to the joint position of the second location. Then create a PTP movement back to the first location, but instead use the recorded frame as target. What can you observe about how the robot is approaching the first location?

## 3 Exercise 2 - PTP and LIN motions

The objective of this exercise is to better understand the difference between joint space movements and task space (= Cartesian space) movements. For this you have to analyse a motion regarding its properties when executed as a joint space movement and as a task space movement.

**Task 3:** *Open the program* `Exercise2_JointAndTaskSpace.java`*. Modify the program to execute a trajectory connecting four locations. Create the trajectory one time with PTP motions to the joint positions and the second time with LIN motions to the frames. Use Cartesian blending of 10 mm in both cases and limit the joint velocity to 0.2.*

1. Use the DataLogger class to record information about the trajectories. This class can be configured to record different information about a movement, like task and joint space positions and distances. The calculated distances represent the movement between two consecutive data samples. Inspect the class for further details.
2. Log (at least) the movement time, joint space distance and task space distance for both trajectories.
3. Use a tool of your choice (Matlab, Python, ...) to analyse the recorded data. Compare the two trajectories with each other. Which one is faster? Which one is longer in task space, which one in joint space? Why?

## 4 Exercise 3 - Impedance Control Mode

In this exercise you will start using the robot with an impedance controller. The target is to test the two available controllers, the joint impedance controller and the Cartesian impedance controller, and to get a feeling for the parameterization before using them in the next exercise to accomplish a more complicated task.

**CAREFUL: here it is possible to push the robot into joint limits, so take care to avoid this (especally axis 6 is easily moved into its limit in this position)!**

**Task 4:** *Start the program* `Exercise3a_JointImpedanceControl.java`*. The robot will move into a crane position and then switch to joint impedance control. It will try to hold this position, but can be moved away. Inspect the upcoming forces in the Info GUI when you push away the robot from its position.*

The left side of the GUI displays the joint torques. The green bars show the torques that are measured in each joint and the orange bars show the calculated external torques that are applied to the robot. The right side shows the torques and Cartesian forces at the TCP of the robot, which are backward calculated from the joint torques. These torques/forces are the actual $\hat{\tau}$, which you have learned of in the theoretical part of the lecture.

1. Push the robot away from the position it trys to hold and inspect the torques occurring in each joint.
2. Change the stiffness and damping values for the joint impedance controller and observe the different behavior. Start with high damping ($\geq 0.7$) and stiffness ($\geq 2500$) values and decrease them iteratively.

**Task 5:** *Start the program* `Exercise3b_CartesianImpedanceControl.java`*. The robot will move into a crane position and then switch to Cartesian impedance control. It will try to hold this position, but can be moved away. Inspect the upcoming forces in the Info GUI when you push away the robot from its position.*

1. Push the robot away from the position it trys to hold and inspect the torques occurring in each Cartesian degree of freedom.
2. Change the stiffness and damping values for the joint impedance controller and observe the different behavior. Start with high damping ($\geq 0.7$) and stiffness ($\geq 2500$) values and decrease them iteratively.

3. You can set stiffness and damping values for each degree of freedom independently. Set a low stiffness for one translational DOF and high stiffnesses for the other two and observe the behavior.
4. Configure a low nullspace stiffness and inspect how you can move the elbow while the robot maintains the end effector position.
5. Specify force limits for the controller so that you can move the robot farther away despite high stiffnesses.

**Task 6:** *Start the program* `Exercise3c_BreakConditions.java`*. The robot will move to a starting position at one side of the robot and then start LIN movements in y direction to the other side and back in an endless loop. This movement uses a Cartesian impedance controller and a break condition to interrupt the movements. Define different break conditions and test them during the movement.*

1. Experiment with different values for the joint torque condition.
2. Create force conditions for different coordinate axis and inspect the susceptibility to errors.

**Task 7:** *Start the program* `Exercise3d_SingularPositions.java`*. This program executes a series of (safe) movements into singular positions. Inspect the Cartesian forces when the robot draws closer to the singularities.*

1. What happens?
2. What is the reason for this effect?
3. How would this affect the Cartesian impedance controller?

## 5 Exercise 4 - Wireloop Exploration

After learning how to create motions for the robot and parametrize the impedance controllers, this exercise focuses on using this knowledge to implement a small application for the robot. The target is to explore a wireloop mounted on the table using only the sensory feedback of the robot

**Task 8:** *Use the template provided in* `Exercise4_Wireloop.java` *to create an application that moves along the wireloop by reacting to collisions with it. Exploit the possibilities of the impedance controller and break conditions to reach this goal.*

1. Record a starting position on the loop using the gravitation compensation mode. It is helpfull to align this position in $B$ to 0 and $C$ to 180°.
2. Start by focusing only on detecting the collision with a curved piece and then turning always in the same direction.
3. Extend you program to allow the robot to move along arbitrarily shaped wireloops. Evaluate the TCP forces at the moment of collision with the curved piece to decide the direction of the curve.

Following now are some hints that are helpful to complete this task. For those, who are eager to accomplish the task without help, you can start off without reading any further!

- Split the program in two methods: the first for approaching a curved piece along a straight one and the second to perform the movement along the curve.
- Since the axis have limits and the most curve movement is performed by axis 6, check its value before each curved piece. If it is close to its limit, simply move up, rotate it by 180° and move down again.
- Use a CIRC movement to move collision free along the curved piece.
- Although on the first view more complicated, move in the tool frame instead of the world frame. This prevents the need to adjust the movements to the gripper orientation.
- Use "calibration movements" to reduce accumulating errors. After finishing the movement along the curved piece, move (approximately) to the middle of the straight piece and then move in positive and negative Y direction until contact. Save both both contact positions and move the robot to the middle of them to center the tool around the straight piece.
- Align the robot / calculated target frames to $B = 0°$ and $C = 180°$ at some points.
- To encounter inaccuracies in the force measurements, do not use static force thresholds, but take the measured forces before a movement into account. Read the force values and add offsets to them to create better force conditions.
- For determining the direction of a curve, have a look at the movement of the tool when it collides with the obstacle. It is dragged into either positive or negative Y direction (of the gripper frame).

**Task 9:** *Extend the program implemented in Task 8 to create a collision free movement along a course.*

1. Record data of collision free points during exploration.
2. Create a completely collision free movement along the course.

## 6 Exercise 5 - Human-Robot Collaboration Task

Since the topic of the lecture is control and programming for human-robot-interaction, this exercise provides the opportunity to account for this aspect. There is no predefined task to accomplish, but instead the task is to design an human-robot collaboration task by yourself and implement it. For example, think of a worker, whose assistant robot carries heavy objects physically guided by the worker or helps to accomplish precise straight movements with some tool (welding!). Try to use impedance control as good as possible!

As a side note, of course there is no "real" tool attached to the robot that you can use, like a gripper. Instead, just pretend to have a certain tool attached and proceed other than that just as your task requires. In example, moving to a storage to pick up an object reduces to only moving to the storage and back to the place of operation.