Chair of Sensor Based Robotic Systems and Intelligent Assistance Systems
School of Computation, Information and Technology
Technical University of Munich

Programming of a KUKA LBR iiwa

Robot Programming and Control for Human Interaction

Tutors: Arne Sachtler, Maximilian Mühlbauer

January 7, 2025

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, despite being "lightweight", 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

In this lecture the robot will be used in a control mode called T1. In this mode the velocity of any part of the robot is reduced to 250 mm/s. Furthermore, it is required to press and hold one of the three enabling switches all the time during robot motion.

The following rules are **mandatory** when operating the robot and disobeying these rules may lead to exclusion from the exercises:

- The robot must be operated by at least two persons. One person holds the Smartpad and monitors the robot movements exclusively. This person is responsible for pressinng the emergency stops/releasing enabling switches in case anything goes wrong. The second person can be in interacting with the robot etc. The person holding the Smartpad/emergency stop must never interact with the robot at the same time!
- In the occurrence of unusual behavior or unexpected movements **immediately** press the emergency stop or release the enabling switch. 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, aborting and resuming do not always work. Especially try to avoid using these functionalities without pressed enabling switch or with locked emergency stop. In this case only a restart of the program helps.
The order of execution is important to enable robot movement. Whenever you execute a program, you FIRST need to press the enabling switch and THEN launch the program. Whenever you released the enabling switch or pressed the emergency button, you need to release the emergency button, press the enabling switch and then click on resume in the Control GUI.

Configuration

To be able to communicate with the robot controller, the network adapter needs to be configured with a static IP address. Use

- `172.31.1.x` (x must not be 147)

Other than that, you can optionally configure the path used to store and load data during exercises. By default, data is stored in the `Exercises/data` folder. You can adjust the path by changing the value of the `DataPath` property in `RobotControl/src/utility/DataHandler`.

Getting in Touch with the Robot

The first exercises will teach you the basics of how to use the framework and command simple motions to the robot. For this, the robot will 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 or programmed constraint that prevents moving the robot 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 stopping the program and moving the joint out of the limit using a special control mode. Contact us in case this happens (it will!). Therefore, you should monitor the joint axis on the Smartpad and avoid moving into joint limits!

Exercise 1

Start the program `Task_01_TeachIn.java`. The robot will move into a crane-like position and then transition into a floating movement, in which you can start moving it around.

a. Move the robot to different locations in the workspace.

b. 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 differences, e.g. positions with the elbow at the top and with the elbow turned to the side, respectively.

Exercise 2

Open the program `Task_02_BasicMotions.java`. Modify the program to execute PTP and LIN and CIRC movements.

a. Load some of the frames and joint positions you recorded in Task 1. Use the DataHandler class to load the joint positions and frames.

b. Create some PTP movements to the joint positions and frames.

c. Add blending to the motions. Note, that you have to use the moveAsync() command to allow the controller to blend over the motions.

d. Create some LIN movements to the frames without or with blending.

e. Create some 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()`.

f. Create a CIRC movement. An image depicting the meaning of the parameters is given in the *kuka_sunrise_introduction.pdf* document.

g. Create a motion batch with some of the previous motions and execute it.

Joint Space and Task Space Motions

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

Exercise 3

Open the program `task_03_Elbowconfiguration.java` and execute a movement between two locations.
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?

Exercise 4

Open the program `task_04_JointAndTaskSpace.java`. Modify the program to execute a trajectory connecting four locations. Create the trajectory one time with PTP motions to frames and the second time with LIN motions to the frames.
Use Cartesian blending of 10 mm in both cases and limit the relative joint velocity to 0.2. 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.

a. Log (at least) the movement time, joint space distance and task space distance for both trajectories.

b. 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? Could it be the other way around?

   Keep in mind, that the recorded distances are between two consecutive samples and you need to calculate the sum of all samples for the overall distance. Dividing the distance by the time between two samples provides velocity.

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: As before, it is possible to push the robot into joint limits, so take care to avoid this (especially axis 6 is easily moved beyond its limit in this position)!

Information: For this task you can ask the supervisor to operate the robot for you and disable velocity restrictions, which allows better testing of the impedance control. However, changing the control mode **must not be done on your own**, but only by the supervisor!

Exercise 5

Start the program `Task_05a_JointImpedanceControl.java`. The robot will move into a crane-like position and then switch to joint impedance control. It will try to hold this position, but can be moved away. Inspect the upcoming forces when you push the robot away from its position in the Info GUI, which starts in addition to the Control GUI.

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 and forces are the actual $\hat{\tau}$, which you have learned of in the theoretical part of the lecture.

a. Push the robot away from the position it tries to hold and inspect the torques occurring in each joint.

b. Change the stiffness and damping values for the joint impedance controller and observe the different behavior. Start with high damping (≥ 0.7) and stiffness (≥ 2500) values and decrease them iteratively.

Exercise 6

Start the program `Task_05b_CartesianImpedanceControl.java`. As before in Task 5, the robot will move into a crane-like position and then switch to Cartesian impedance control instead. Inspect the upcoming forces for the different degrees of freedom in the Info GUI while interacting with the robot.

a. Push the robot away from the position it trys to hold and inspect the torques occurring in each Cartesian degree of freedom for the TCP.

b. Change the stiffness and damping values for the Cartesian impedance controller and observe the different behavior. Start with high damping (≥ 0.7) and stiffness (≥ 2500) values and decrease them iteratively.

c. 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.

d. Configure a low nullspace stiffness and inspect how you can move the elbow while the robot maintains the end effector position.

e. Specify force limits for the controller so that you can move the robot farther away despite high stiffnesses.

Exercise 7

Start the program `Task_06_BreakConditions.java`. The robot will move to a starting position and then start repeating LIN movements in y-direction. This movement uses a Cartesian impedance controller and a break condition to interrupt the movements. Define different break conditions and test them during movements.

a. Experiment with different values for the joint torque condition.

b. Create force conditions for different coordinate axes and inspect the susceptibility to false triggering.

Exercise 8

Start the program `Task_07_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.

a. What happens?

b. What is the reason for this effect?

c. In which situation could this be a problem?

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.
Hint: It is valid to assume that the wireloop only consists of straight pieces and 90° curves.

Exercise 9

Use the template provided in `Task_08_Wireloop.java` to create an application that moves the tool along the wireloop by reacting to collisions with it and measuring contact forces. Exploit the possibilities of the impedance controller and break conditions to reach this goal. The code contains a number of utilities, which may be helpful for the task, like moveUp/moveDown/, Direction. Have a look at the code to see more about them!

a. Record a starting position on the loop using the gravitation compensation mode. It is helpfull to align the orientation of this position in y-direction (B) to 0 and in x-direction (C) to $180°$. An example for this is given in the code for the variable `wireloopStart`.

b. Start by focusing only on detecting the collision with a curved piece and then turning always in the same direction.

c. Extend you program to allow the robot to move along arbitrarily shaped wireloops, i.e., any course that can be created using the straight and curved pieces. Evaluate the TCP forces at the moment of collision with the curved piece to decide the direction of the curve.

Exercise 10

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

a. Record data of collision free points during exploration.

b. Create a completely collision free movement along the course.

Hints

Following are some hints that are helpful to complete this task. Those eager to accomplish the task without help can start off without reading any further!

- Use impedance control mode (.setMode() for motion commands) to prevent the robot from pushing away pieces of the wireloop. There is a control mode already preconfigured in the code.
- 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.
- 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.
- Since the joint axes have limits and the most rotatory movement is performed by axis 6, check its value before each curved piece. If it is close to its limit (+-175 deg), simply move up, rotate it by 180° and move down again.
- 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 contact positions and move the robot to the center to realign the tool around the straight piece.
- Align the robot/calculated target frames to $B = 0^\circ$ and $C = 180^\circ$ at some points.
- Align the robot/calculated target frames in z-direction, if the robot is drifting away.
- 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), thus resulting into a force in y-direction.