Friedrich-Alexander-Universität
Erlangen-Nürnberg

WS 2025/26

Lab course: Human Robot Interaction

Task 2: ROS Introduction, Kinematics and Interaction

Lehrstuhl für Autonome Systeme und Mechatronik
Friedrich-Alexander-Universität Erlangen-Nürnberg

Prof. Dr.-Ing. habil. Philipp Beckerle
Martin Rohrmüller, M. Sc.

Introduction

In the previous task you have already learned the basics of ROS, robot kinematics and the practical execution on a real robot. In this task, this will be built upon. In the preparation, you will first be introduced to inverse kinematics. In the execution part of this task, those will be applied in order to move the end effector of the real robot in cartesian space. In the course of this, the ROS concepts of Publishers and Subscribers will be extended. An important algorithm in the physical interaction of robots with environments is impedance or admittance control. This will also be part of this task in order to learn about its usefulness in the interaction of robots with the environment.

Task 2: Preparation

In the preparation phase of this task, foundational concepts are established to support practical applications on the actual robot in the execution phase. The task begins with an introduction to inverse kinematics, essential for understanding compliant control. Following this, the implementation of ROS Subscribers and Publishers is expanded into an object-oriented code structure, which will be important for the lab session.

Inverse kinematics.

In the first task, we examined the robot's forward kinematics, allowing us to determine the position and orientation of the end effector based on joint angles. Conversely, inverse kinematics enables the calculation of joint angles to achieve a desired end effector position.

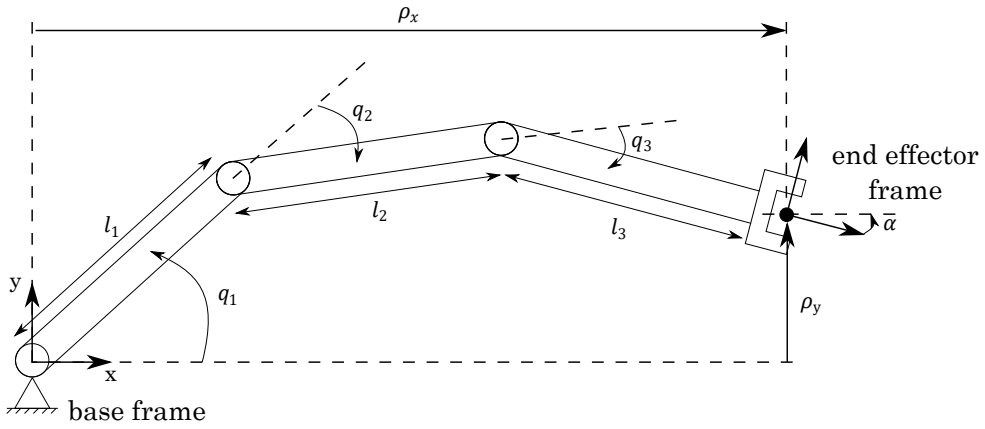

Figure 1: Planar manipulator with three degrees of freedom.

Consider again the two-dimensional robot with three degrees of freedom as shown in Figure 1. The geometric relationships between the end effector positions $\boldsymbol{p} = [p_x, p_y, \alpha]^T$ and the joint angles $\boldsymbol{q} = [q_1, q_2, q_3]^T$ are already known. The inversion here leads to inverse kinematics, but it should be noted that the angular functions in general do not have unique solutions. For assistance on the next exercise, you can use the provided excerpt on inverse kinematics from the book "Robot Mechanisms" [1]. Note that the joints q_2 and q_3 in the book are defined in the other direction compared to Figure 1.

Exercise 1.
Inverse kinematics of the planar robot of Figure 1.

a) Calculate the inverse kinematics of the planar robot and thus the joint angles $\boldsymbol{q}$ as a function of the end effector position $\boldsymbol{p}$.

b) Select three different achievable end effector positions and calculate the joint angles. You can choose any link length (e.g. unit lengths). Check the plausibility by sketching the robots or by applying forward kinematics from the previous task.

[1] J. Lenarcic, T. Bajd, and M. M. Stanisic, in Robot Mechanisms (Springer Netherlands, Dordrecht, 2013)

c) In contrast to forward kinematics, what is the major challenge in computing inverse kinematics?

Add the results and answers to your submission file.

In addition to solving the inverse kinematics analytically, it is also possible to approach the solution numerically. For this purpose, the equations of the forward kinematics are optimized with respect to the joint positions, so that they best approximate the required end effector positions. A basic example of this is in the materials provided (*inverse_kinematics_3DoF_numerical.py*). This option offers the advantage of a universal solution. The disadvantage, however, is the computational effort for the optimization procedures of the solvers and that the equations are not available explicitly.

There is a much higher difficulty to obtain possible solutions for a robot with seven degrees of freedom like the Franka Emika robot. Therefore, existing functions will be used when performing in the laboratory.

Extension of ROS Subscriber and Publisher concepts.

So far, the ROS concepts Publisher and Subscriber have been considered. These were defined individually in a script. However, ROS offers the possibility of running several Publishers or Subscribers in a node and thus receiving and sending different messages. In other words, a ROS Node is not limited to a single Publisher or Subscriber. There would even be the possibility to start multiple Nodes in one script file. Mostly, however, only one Node is used in a script.

Exercise 2.
Take a look at the provided code *inverse_kinematics_3DoF_numerical_node.py*. This Node subscribes to a message for the cartesian position $\boldsymbol{p}$, continuously calculates the associated joint angles $\boldsymbol{q}$ in the callback function with inverse kinematics and publishes them in another message. Think about the questions: At what points were Subscribers and Publishers defined here compared to individual examples from the first task? How can the Publisher access the variable that was read and processed via the Subscriber ?

The implementation tasks for the planar robot were partially completed using classes, incorporating an object-oriented approach. However, the ROS building blocks were initially implemented using a procedural method. This section introduces an object-oriented approach to implement these ROS elements, including ROS Nodes. Figure 2 shows the basic structure, demonstrating how a Node containing Publishers and Subscribers can be implemented more elegantly. In the initialization method, class variables, the Publishers and Subscribers are initialized for this purpose. In the main function, the Node is created as an object and the initialization method is called. In the run function, the Node loop can then run at a defined rate. Here it is important to remember, that the class variables are defined and used with the self keyword and the class methods must receive this as the first passing element. One advantage of object orientation is, that the variables are defined within the scope of the class. Global variables, as for example in the code from Exercise 2, are thus no longer needed, every method can access the class

variables. Also, by using classes it will be easier to create reusable blocks that will allow you to scale your application in an easier way. Further information can be found on the homepage of *Robotics Back-End* [2].

```python
#!/usr/bin/env python
import rospy
from std_msgs.msg import ...
...

class Node:

   def __init__(self):
      # Definition of class parameters and variables
      self.var1 = ...

      # Initialise Publishers and Subscribers
      self.pub1 = rospy.Publisher(...)
      self.sub1 = rospy.Subscriber(..., self.sub1_callback)

   def sub1_callback(self, msg):
      # Writing message data into class variables
      self.var1 = msg.data

   """
   Place for more callback functions
   """


   def run(self):
      #ROS loop and publishing messages
      while not rospy.is_shutdown():
         rate = rospy.Rate(40)
         new_msg = ...
         new_msg.data = ...
         self.pub.publish(new_msg)
         rate.sleep()


if __name__ == '__main__':
   # Instantiating of a node object, running the node
   rospy.init_node("node_name")
   n = Node()
   n.run()
```

Figure 2: Code skeleton for object oriented Python Node.

As mentioned, the code in *inverse_kinematics_3DoF_numerical_node.py* is not object oriented and some variables are stored globally. In order to avoid those global variable

[2] https://roboticsbackend.com/oop-with-ros-in-python/

definitions, in the next exercise you are supposed to bring the code to object oriented structure. The Subscriber in the code should be able to get messages from the Topic */ee_position*.

It is possible to send values to the Topic */ee_position* with

```
$ rostopic pub /ee_position std_msgs/Float64MultiArray "data: [1.0, 0.5,
    0.393]"
```

via the terminal, in this example with end effector positions $p_x = 1.0$, $p_y = 0.5$, $\alpha = \frac{\pi}{8}$. Alternatively, the provided file *send_end_position.py* can be used. In it, a Publisher sends the same fixed cartesian values.

Exercise 3.
Rewrite *inverse_kinematics_3DoF_numerical_node.py* so that it has object oriented structure. To do this, use the code structure that was explained above. Please mind the correct message types and use a separate method for calculating inverse kinematics. Test the functionality by running the script as a Node. Don't forget to modify *CMakeLists.txt* and rebuild the workspace for this purpose. Save it in the *ros_intro* Package in the *robot_ws* Workspace.
Run the node and send values to the Topic */ee_position* either from Terminal, or with the provided file. Add a screenshot of all active terminals while an end effector position is sent to the submission file.

The joint angles could be forwarded to another node, for instance. By adding a subscriber in *robot.py* from the previous task, the joint angles could then be read continuously. This setup allows us to create data pipelines efficiently using ROS.

To leverage an object-oriented approach, this structure will also be applied in the upcoming tasks of the course. First, however, we will discuss compliant control, followed by an example demonstrating its application to the planar robot previously examined.

Impedance control

The realization of compliant behavior is a classical problem in robotics. This is relevant, whenever the robot comes into contact with its environment and especially if the environment is only partly known. A desired disturbance response with respect to external forces has to be ensured.

The concept of impedance control is a classical approach in robotics, particularly for compliant control. A key principle of this methodology is that the controller should adjust the manipulator's mechanical impedance to achieve desired interaction dynamics while ensuring adaptability during operation. In the impedance control literature, the desired setpoint $\boldsymbol{x}_d$ usually is called a virtual desired setpoint (or equilibrium point), since it should only be reached in case of free motion, i.e. when no external forces act on the robot.

From a practical point of view the desired dynamic behavior of the robot, i.e. the disturbance response (regarding the external forces as a disturbance), usually is defined in terms of Cartesian coordinates, describing the pose of the endeffector. In typical

implementations, the controller has the Cartesian position $\boldsymbol{p}$ and velocity $\dot{\boldsymbol{p}}$ of the end effector of a robot as inputs and gives the motor torques $\boldsymbol{\tau}_m$ as outputs. The block diagramm of the impedance control scheme can be seen in Figure 3.

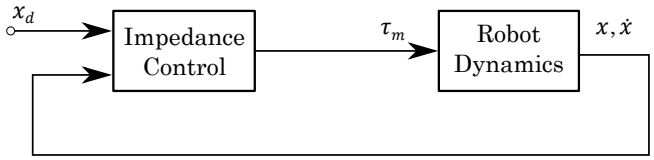

Figure 3: Block diagramm of impedance control structure.

The principle of impedance control will be illustrated with an introductory example. Following this, there will be a brief overview of impedance control in robotics, along with an introduction to admittance control, a related compliant control method.

Impedance control of a 1-DoF mass

The dynamic modell of a mass m without friction can be written as

$$m\ddot{x} = u - f_e - d \tag{1}$$

with the control variable u, an external force f_e and a disturbance d chosen to be $d = 0$. With the impedance control law

$$u = m\Big(\ddot{x}_d - \frac{1}{m_d}(-b_d\dot{e} - k_d e + f_e)\Big) + f_e \tag{2}$$

the closed loop system is given as

$$m_d\ddot{e} + b_d\dot{e} + k_d e = f_e \tag{3}$$

which corresponds to a spring-damper-mass system. Thus, the system can be imprinted with a certain behavior by the control law 2. This behavior is composed of a mass m_d, a damping b_d and the stiffness k_d. Thus, it affects the control error $e = x_d - x$. From a desired set point or trajectory x_d, a quantity u is calculated, which here acts as a force directly on the mass. In the robotics application, this manipulated variable would correspond to a joint torque, that acts through the motor at the respective joint.

Exercise 4.
Impedance Control of a 1-DOF system. Asume that the parameters m_d, b_d, and k_d are positiv and greater than zero

a) Assume a given nominal behavior according to the equation 3 to be achieved. Derive the control law 2 by solving for the acceleration $\ddot{x}$ and inserting it into the system equation 1.
($\ddot{e} = \ddot{x}_d - \ddot{x}$)

b) The mass is to be controlled to a constant target position ($\ddot{x}_d = \dot{x}_d = 0, x_d = x_{d_{const}}$). Calculate the permanent deviation (mass at rest) as a function of the external force f_e.

For the same constant target position, distinguish between the two cases: In the first case, the external force is zero ($f_e = 0$) and the initial velocity of the mass is not zero ($\dot{x} \neq 0$). In the second case, there is an external force applied (for t>0) and the mass is initially at rest ($\ddot{x} = \dot{x} = 0$) and in equilibrium position ($x_d - x = 0$).

c) For each of the two cases, discuss how the closed loop system behaves, when either the damping is zero ($b_d = 0$) or

d) the stiffness is zero ($k_d = 0$).

Add your calculations from a) and b) to the submission file and explain your considerations for c) and d) in one sentence each and each case.

Impedance control of a robot

Compared to a simple mass that moves linearly, a robot is a multibody system with rotational motions. Therefore, instead of the scalar mass, an inertia matrix must be considered. In addition, a Coriolis matrix is added in order to take into account the effects of the Coriolis forces in the robot due to the rotational motions. In an additional part, the effects of gravity are considered. The dynamic model of a robot can be written as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{u} - \boldsymbol{\tau}_e \tag{4}$$

with the mass-matrix $\boldsymbol{M}(\boldsymbol{q})$, the matrix with Coriolis and centrifugal terms $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ and the vector of gravitational components $\boldsymbol{g}(\boldsymbol{q})$. When there is an external moment $\boldsymbol{\tau}_e$, then the robot should behave like a linear spring-mass-damper system

$$\boldsymbol{M}_d\ddot{\boldsymbol{e}} + \boldsymbol{B}_d\dot{\boldsymbol{e}} + \boldsymbol{K}_d\boldsymbol{e} = \boldsymbol{\tau}_e \tag{5}$$

with the mass $\boldsymbol{M}_d$, the damping $\boldsymbol{B}_d$ and the stiffness $\boldsymbol{K}_d$, here considered in joint space. This can be achieved with choosing the control law to

$$\boldsymbol{u} = \boldsymbol{M}(\boldsymbol{q})\Big(\ddot{\boldsymbol{q}}_d - \boldsymbol{M}_d^{-1}(-\boldsymbol{B}_d\dot{\boldsymbol{e}} - \boldsymbol{K}_d\boldsymbol{e} + \boldsymbol{\tau}_e)\Big) + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\tau}_e \tag{6}$$

with the need to measure the external moment to be able to also influence the acting inertia. In the case of no measurement of the external moment and the assumption, that $\boldsymbol{M}(\boldsymbol{q}) = \boldsymbol{M}_d$ the control law simplifies to

$$\boldsymbol{u} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}}_d + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{B}_d(\dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_d) + \boldsymbol{K}_d(\boldsymbol{q} - \boldsymbol{q}_d) \tag{7}$$

which is simliar to a PD-control with feed-forward compensation of the dynamic terms. The advantage of the control law in 7 is, that the external moments do not need to be known or measured respectively.

Transition to admittance control

In the examples just shown, dynamics plays a role. To calculate the equations, the mass, inertia and Coriolis matrices of the robot would have to be known and the influence of

gravity would also have to be taken into account. So far, robot dynamics have not been considered and this is how it should remain for the further tasks on the lightweight robot. Fortunately, as already described, the Franka Emika robot offers the possibility of directly giving it target specifications for the position or velocity. This is the motivation for replacing impedance control, which can be used to calculate moments and forces as a target, with an approach that can be used to determine target positions. This is where admittance control comes in to play. As can be seen in Figure 4, with this approach, a target position for a subordinate position controller can be calculated as a function of the external force on the robot. The position controller thus handles the robot dynamics and, in our case, is already integrated in the robot.

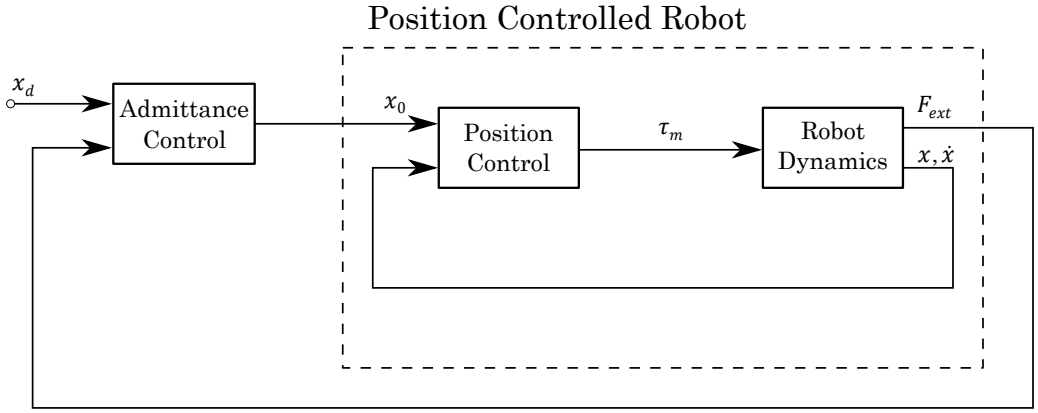

Figure 4: Admittance control structure.

Based on the preset position for the admittance controller, the latter thus calculates the target position for the subordinate controller in such a way that the spring-mass damper behavior also results during movement and interaction, as in the case of the impedance controller. As a reminder and for comparison, the impedance controller directly calculates the joint torques as the manipulated variable for the uncontrolled robot.

The admittance control portion is simplified in that now only the spring-mass damper dynamics need to be considered between the default position and the calculated target position. This means, that the equation

$$\boldsymbol{M}_d(\ddot{\boldsymbol{p}}_0 - \ddot{\boldsymbol{p}}_d) + \boldsymbol{B}_d(\dot{\boldsymbol{p}}_0 - \dot{\boldsymbol{p}}_d) + \boldsymbol{K}_d(\boldsymbol{p}_0 - \boldsymbol{p}_d) = \boldsymbol{F}_e \tag{8}$$

needs to be solved for the desired position $\boldsymbol{p}_0$ for the subordinated position controller. Again, the external force $\boldsymbol{F}_e$ needs to be known by measuring it. Note that this formulation is already in the Cartesian workspace of the robot. The behavior with the admittance controller should thus correspond to a spring-mass-damper system that acts on the end effector of the robot. The subordinate position controller must therefore also receive the cartesian setpoint positions as input here [3].

For the next exercise, there is the template *admittance_control_planar.py*.

[3] C. Ott, Cartesian impedance control of redundant and flexible-joint robots. Springer, 2008

Exercise 5.
Try out the code and answer the questions.

a) Get an understanding of the code and how the admittance equation was implemented. Run the code in the IDE and stop execution after a few seconds with Ctrl+c. This will already automatically create plots with the results in the "Plots" window of Spyder.

b) Try out different values for the system parameters (change inertia, damping and stiffness at least once). Note down, how those changes affect the results.

c) Try out other values than zero for `F_ext` and note, how those affect the results.

d) Discuss and write down, why the *odeint* solver is used here and not an algebraic solution.

Add the answers and plots to the submission file.

With the ROS nodes discussed in the preparatory tasks, it would already be possible to implement a framework as shown in Figure 5. In order to achieve a closed control loop, measured forces would have to be fed back to the robot.

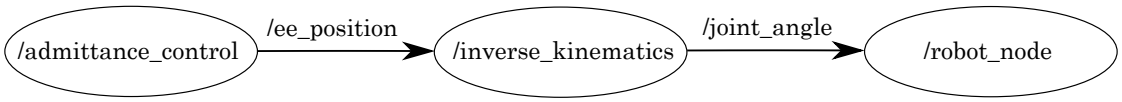

Figure 5: Signal pipeline.

In task 4, you will link several ROS nodes and also include external sensor values. The lab part of this task however, will focus on task space motions and compliant behavior.

Task 2: Execution

In this task, applications for the real robot with seven degrees of freedom are implemented based on the knowledge gained from the preparation. First, the motion planning is extended with the help of inverse kinematics. Then, impedance control is applied to the robot, which can be used in particular to learn about the interaction behavior. Furthermore, collisions and force interactions with the robot are carried out. This task is supposed to be integrated in the same Workspace and Package that were created in the first lab task.

Exercise 6.
Start the computer and the robot and make sure, that your group's workspace is sourced. (Edit with command

```
$ gedit .bashrc
```

and check for commented lines at the end of the file!)

Inverse kinematics for task space motion

In the first task of the lab, a trajectory was planned and executed in the robot's joint space. However, for most applications it is more purposeful to work with the positions in the task space and thus with the cartesian coordinates.

In order to achieve a cartesian trajectory by controlling the joint positions, the cartesian positions must be transferred to the joint space using inverse kinematics. Therefore, the Robotics Toolbox[4] offers a function to compute the inverse kinematics of the Franka Emika robot from a given transformation of the end effector frame

$$\boldsymbol{A}_{ee}^{b} = \begin{bmatrix} \boldsymbol{R}_{\boldsymbol{ee}}^{\boldsymbol{b}} & \boldsymbol{p}_{\boldsymbol{ee}}^{\boldsymbol{b}} \\ \mathbf{0} & 1 \end{bmatrix} \tag{9}$$

with

$$\boldsymbol{q} = ikin(\boldsymbol{A}_{ee}^{b}) \tag{10}$$

when represented as a homogeneous SE3 transformation. In the first part of this task, this will be used to create the cartesian motion in combination with the control on the position level. Thus it is necessary to take into account the intial position as well as the goal position of the end effector.

⚠During the trajectory task, ensure that all participants are outside the workspace of the robot as long as the enabling switch is activated!

Exercise 7.
Implement a robot task space motion. The objective of this exercise is to create a straight motion of the end effector in one dimension.

[4] https://petercorke.github.io/robotics-toolbox-python/intro.html

a) Move the robot to the starting position that is needed for this exercise. Therefore, run the terminal command

```
$ roslaunch franka_example_controllers move_to_start.launch
```

while having the external enabling switch activated.

b) Get the intial joint position of the robot and the initial cartesian position and rotation of the end effector and note them for later. The latter two can be received from *O_T_EE* of the Franka State Topics. As in the first task, you can access those as soon as the position controller has been started with

```
$ roslaunch franka_control franka_control_joint_position.launch
```

Now, use the file *cartesian_trajectory.py*, integrate it in the *hri_lab* Package and work through the TODO comments with the following exercises:

c) Check, if `self.intial_pos` matches the current measured position.

d) Define the goal position that is in a distance of $0.2m$ to the initial position in negative x direction as a 4×4 matrix.

e) Create the `N_step` cartesian points between the positions.

f) Do the inverse kinematics for each of those points represented in $SE3$.

g) In the ROS loop, command the joint positions according to the cartesian points.

h) Add the line to publish the message.

i) Start the position controller with the command

```
$ roslaunch franka_control franka_control_joint_position.launch
```

and run the Node to execute the trajectory.

j) Move the robot back to the initial point by switching the goal and target positions in the script.

k) Define other goal positions and tweek the parameters `rate` and `N_steps` . Before executing, think about the effects you expect from different parameters.

If everything worked well, the robot should have just made a straight-line motion in one direction. You can now ask yourself what happens if the trajectory between the two points is not planned via inverse kinematics but directly in the joint space.

Exercise 8.
Create a copy of the file *cartesian_trajectory.py* and continue with the copy. Carry out the robot motion from the same start to goal point, but this time with planning the trajectory in joint space. Therefore, do the inverse kinematics for the inital position and for the goal position but split into the steps in joint space. What results can you see?

Another option offered by the Franka Emika robot is to directly use an impedance controller that takes in the desired position $\boldsymbol{p}_d$ as input.

Impedance control of the Franka-Emika robot

The general impedance control law, already shown in the preparation part, can be denoted as

$$\boldsymbol{u} = \boldsymbol{M}(\boldsymbol{q})\Big(\ddot{\boldsymbol{q}}_d - \boldsymbol{M}_d^{-1}(-\boldsymbol{B}_d\dot{\boldsymbol{e}} - \boldsymbol{K}_d\boldsymbol{e} + \boldsymbol{\tau}_e)\Big) + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{\tau}_e \tag{11}$$

and the robot's dynamic model as

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{u} - \boldsymbol{\tau}_e \tag{12}$$

both in joint space.

Exercise 9.
Assume the robot 12 being controlled with 11. Calculate the solution of the closed system with the following conditions:

- No inertia-shaping: $\boldsymbol{M}(\boldsymbol{q}) = \boldsymbol{M}_d$
- Only the desired position $\boldsymbol{q}_d$ will be specified, but not the velocity or acceleration ($\ddot{\boldsymbol{q}}_d = \dot{\boldsymbol{q}}_d = \boldsymbol{0}$)

No inertia-shaping means, that only stiffness and damping behavior can be imprinted to the robot, but no specific inertia behavior. This is, what the Franka Emika impedance controller does. The resulting equation is a differential equation for the actual joint positions $\boldsymbol{q}$.

The controller used for the next part works directly in task space. The concept of the formulas described is therefore transferred internally to the task space. This is not discussed further; details can be found in the footnote [5]. This allows cartesian desired positions $\boldsymbol{p}_d$ and the stiffness and damping behavior of the end effector to be specified directly with $\boldsymbol{K}_d$ and $\boldsymbol{B}_d$. Having a fixed position set point $\boldsymbol{p}_d$ should result in a behavior of the end effector, as if it was attached to a spring-damper system fixed at this set point position. The controller creates the motion reactions accordingly. The inverse kinematic is also already taken care of in the internal robot interface.

With the next exercise, you will run the impedance controller and carry out some experiments with different parameters. Use the provided *impedance_control_command.py* for that. This Node will send the desired positions to the low level impedance controller.

Exercise 10.
Follow these steps and the TODO comments:

a) Include the script *impedance_control_command.py* to your Workspace.

[5] C. Ott, Cartesian impedance control of redundant and flexible-joint robots. Springer, 2008

b) Finish the publisher to publish the desired position $\boldsymbol{p}_d$ to the Topic */cartesian_impedance_example_controller/equilibrium_pose* and look at the `run()` function for all the information you need. The message type used here contains the position and the rotation of the end effector. For the later experiment, only the positions will be changed.

c) In the initialization, the *franka_states* are required. Thereforce, instead of the position controller for the rest of this task the impedance controller will be used. To start the controller, at first run the command

```
roslaunch franka_control
   franka_control_cartesian_impedance.launch
```

to start ROS now with the impedance controller. It takes some time to bring up the controller, wait until `--- CONTROLLER STARTED ---` appears in the terminal.

d) Run the script as a Node and check if the desired position is published correctly. It should be constant and reflect the current position of the robot.

After implementing the code that is supposed to publish the desired positions to the impedance controller, you can carry out the next exercise and do some experiments with the robot in order to experience the idea of this control method. The impedance controller used here only needs scalar stiffness values to be set. The damping value depends on it with $b_d = 2\sqrt{k_d}$ which limits overshooting behavior.

⚠ Only one person is allowed to interact with the robot. Another person must supervise it with the non-permanent activation device. Interact carefully and smoothly with the robot. There are safety limits active, that make it stop at aggressive interaction.

Exercise 11.

a) Bring the robot to a position, where the end effector is free to move in all directions, or run

```
$ roslaunch franka_example_controllers move_to_start.launch
```

alternatively.

b) Run the impedance controller and the *impedance_control_command* Node.

c) Run

```
rosrun rqt_reconfigure rqt_reconfigure
```

in a Terminal. This brings up a tool that lets you change the stiffness parameters of the controller. When using it, do only change the values slowly with the sliders!

d) Try different values, grab and move the robot's end effector in space.

e) Figure out what the three different adjustments do.

f) Set the position stiffness to zero. Make yourself aware of the result to expect and try it. Be careful, when setting it back to a value greater than zero, because this will "pull back" the robot to its equilibrium position.

You have now certainly got a feeling for the behavior of the system and can evaluate the behavior with various parameters. Now it is still interesting to move the robot at the same time. Therefore, in the script *impedance_control_command.py* is already a Subscriber, that can read in a desired direction for a motion, defined by the *motion_direction* variable, which is initialized as zero. This needs to be a 3×1 unit vector. With the predefined gain value, the desired position will be continuously changed and thus create a motion of the robot in a specified direction.

Exercise 12.
Again, run the low level impedance controller and the high level impedance command node. After the start up, run

```
$ rostopic pub /motion_direction std_msgs/Float64MultiArray "data:
    [0.0, -1.0, 0.0]"
```

in order to set the robot into motion. You can stop it, with either deactivating the enabling switch, or publishing a zero vector. Again carefully interact with the robot while it is moving. You can try a different motion direction, but the vector should be a unit vector.

In the following task, the external force is read directly from the robot. This information comes with the Franka States via the ROS framework after starting up the robot and controllers. This external force is internally computed from the measured joint torques.

Collision detection and collision handling

In this section, the goal is to apply the impedance controlled motion in order to create a reaction of the robot to an environmental impact. Therefore, an additional Node will be integrated which monitors the joint torques. After detecting a single moment exceeding a certain threshold, the robot is supposed to react with a motion to an opposite direction of the vector of forces acting on the end effector. Figure 6 shows the idea of the next exercise. For industrial robot applications, this could be extended with different approaches of the robot to react to interactions or even to reactions of the robot in its nullspace (keeping the end effector position constant while moving parts of the arm). In this task, the robot should move with giving desired positions to the impedance control of previous exercises with the Topic */motion_direction*. Running the command Node can be used for troubleshooting the next exercise.

Exercise 13.
Implement the missing parts in the script *force_torque_monitor.py*. Since force monitoring does not need to be very fast in this case, this node can run at a much

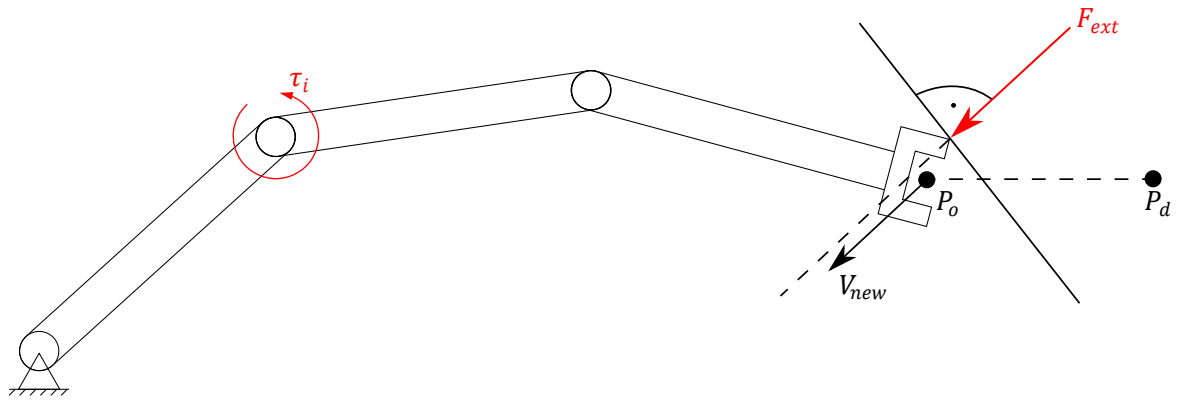

Figure 6: Robot interacting with environment.

lower rate. Follow those steps:

a) Choose a suitable rate and implement it to the adequate position in the code.

b) Finish the callback function to get the joint torques from Franka States into class variables.

c) Implement monitoring of the joint torques in `check_torque_limits()` according to the pseudo algorithm

limit_exceeded $\leftarrow$ *False*
for $i \leftarrow 1$ to N_{joints} **do**
  if $|\tau[k]| > \tau_{lim}[k]$ **then**
    limit_exceeded $\leftarrow$ *True*
  end if
end for

and use the already predefined threshold values for τ_{lim}.

d) If one of the limit values is exceeded, a new vector for the motion velocity direction is to be calculated. Do this in the method `change_velocity_vector()`

This Node should now output the calculated direction of motion. Therefore, a Publisher publishes a `Float64MultiArray` message to the Topic */motion_direction*. If you apply force to the end effector, or alternatively to a robot element, the Node should now change the direction of motion. Combine this with the controller in the next exercise.

Exercise 14.
Again run the impedance control as you did before. Additionally, run the monitoring Node. Carry out the following experiments:

a) Try, if the monitor works. Therefore, apply force on the end effector of the robot. After reaching one of the torque thresholds, the robot should start with a motion in opposite direction to the force. Use the wooden board and place it in the new path of the robot to stop it and change its direction again.

b) Now, instead of interacting on the end effector, apply the forces on links of the robot. How does this change the reaction?

c) During the robot motion, when you apply forces that are below the threshold, can you still feel the compliant behavior? Try it with a different stiffness parameter.

Exercise 15.
Think about the following questions. You should answer them for the follow-up assignment.

a) What problems can arise when, as in Exercise 7, a cartesian trajectory is transferred to a trajectory in joint space using inverse kinematics? Think about the continuity of the solutions. What are the advantages and disadvantages of both methods?

b) In the last task, based on an event, a constant message stream is sent to the other Node by a collision. Which type of transmission would be more suitable for events that occur at certain moments?

c) What can be a problem when deriving external forces from a measurement of joint moments, and what are the advantages and disadvantages of using a force sensor on the end effector of a robot? Are there configurations where certain external forces cannot be measured with joint sensors only?

d) What happens, when we have impedance control for task space, but have interaction on links and not only on the end-effector?