General outline of lab assignments

In these lab assignments you will derive the kinematics and dynamics of the robot manipulator ABB IRB 120 (see description below) to design several controllers. Throughout the three assignments, you will first develop a model library of Matlab functions describing the kinematics (Lab 1) and then dynamics (Lab 2) to finally develop a set of controllers for the manipulator (Lab 3). Each assignment consists of compulsory exercises where you need to do some implementation in Matlab as well as voluntary exercises worth in total **5 points**. The voluntary exercises are clearly marked as **OPTIONAL**.

For each assignment you can download the code from the Canvas Assignments page in a ZIP folder and fill in the necessary functions on your local computer. Once you have completed an exercise by filling in the *TODOs* in the files, you can copy paste your code to the correct location on the Canvas MATLAB Grader page to get immediate feedback. Working on your local computer is recommended, since in the ZIP folder you are also provided with a Simulink file to visualize the robot simulations. Note that for the visualization to work, you need Simulink version R2020b or higher and have the *Robotics System Toolbox* add-on installed. You can run the visualization using the Matlab script `run_XXXXX_YYYYY.m` files found in each folder.

Submission

For each assignment there will be a coding component to be finished on Canvas through the MATLAB Grader tool. The results for this component are immediately visible when you submit your solution. All MATLAB Grader exercises have an unlimited number of submissions. Additionally, a short report with the answers to the questions, the manual calculations (e.g., D-H table, robot schematics, etc.), and the discussions should be submitted in a PDF in the assignment page. You should submit your Matlab functions and scripts, and Simulink modules as a zip file to the same assignment page.

The ABB IRB 120 robot manipulator

The system used in this assignment will be the ABB robot IRB120 (see Fig.1). The ABB IRB120 is a robot with six revolute joints. However, only four of the joints will be used in this assignment. The four actuated joints **Axis 1**, **Axis 2**, **Axis 3**, and **Axis 5** are shown in Fig.1 a), and the physical parameters for the system are shown in Fig. 1 b). The data sheet with the robot's specifications provided by the manufacturer can be found as part of the zip in Canvas (**abbIRB120_specs.pdf**).

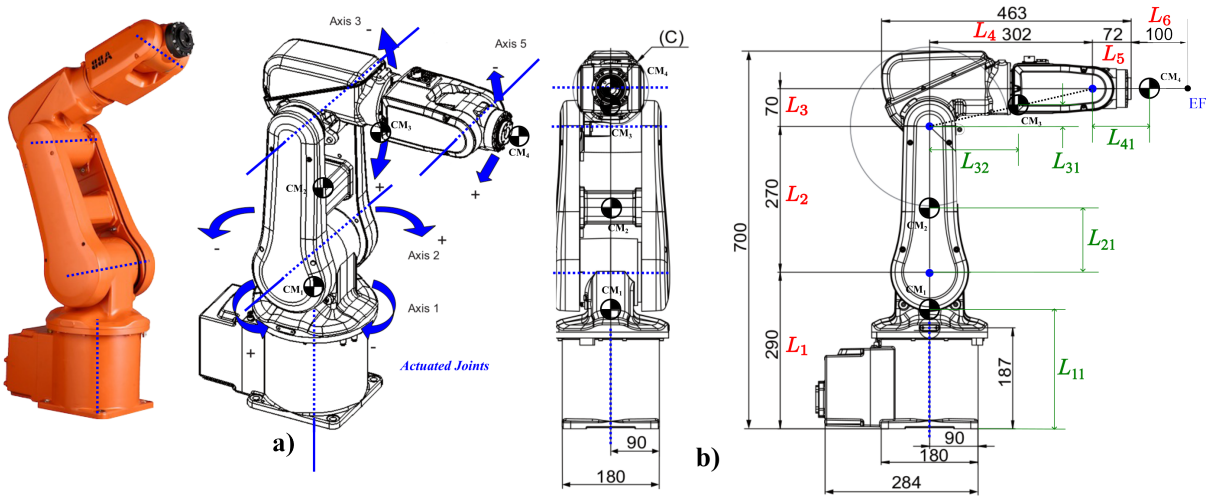

Figure 1: Robot ABB IRB 120: The robot only uses the joints 1, 2, 3, and 5.

Assignment 2 - Dynamics

This assignment consists of two parts. The first part involves dynamic modelling of a robot, by deriving the equations of motion using the Euler-Lagrange formulation. The second part includes a validation and application task, where you will validate and use the obtained models.

Modelling: Dynamic Model

The main task is to obtain the Dynamic model of the robot in the form an equation of motion:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}\left(\mathbf{q}, \dot{\mathbf{q}}\right)\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{B}\dot{\mathbf{q}} = \boldsymbol{\xi} \tag{1}$$

To obtain the dynamic model, you will use the Homogeneous Transformations ($\mathbf{H}^0_{cm_i}$) and geometric Jacobians ($\mathbf{J}^0_{cm_i}$) of each center of mass (cm_i). To complete this tasks is recommended that you use the symbolic computation scripts developed in the Lab 1 with the DH table of the centers of mass. The closed form (symbolic form) HTs and Jacobians obtained in this step will be used to compute the $\mathbf{M}, \mathbf{C}$ matrices and the $\mathbf{G}$ vector. These symbolic form functions will be used to complete the functions found in the folder `RobotABB_IRB120_4DOFS/Models`. In the second part, you will use the obtained models and validate them in simulations by finishing the functions in `RobotABB_IRB120_4DOFS/SimulatorABB_IRB_4DOF`.

Exercise 1 .....

First, compute the relative ($\mathbf{H}^{i-1}_{i}, \mathbf{H}^{i-1}_{cm_i}$) and absolute Homogeneous Transformation (HT) matrices ($\mathbf{H}^0_i, \mathbf{H}^0_{cm_i}$) to obtain the pose of each link, and center of mass (cm). It is recommended that you use the symbolic functions developed in the Lab 1. Finalize the functions `Model/abbIRB4_dyn_params.m` and `Model/getAbsoluteHTcm_abbIRB4.m`. The first function generates an array with the kinematic and dynamic parameters of the robot. Some of the parameters are already defined in Fig. 1, e.g., $L_1, L_2, \ldots, L_6$. You need to calculate the additional parameters in the parameter function. The second function receives a joint position vector $\mathbf{q} = [q_1, q_2, q_3, q_4]$, an array with the dynamic and kinematic parameters, and the HT of the robot base ($link_0$) with respect to (wrt) the world coordinated frame (wcf). The output of this function is two Cell Arrays, one with the HTs of each cm wrt the robot base ($link_0$), and the other with the HTs of each cm with respect to the wcf (w). You will also resuse `Model/getAbsoluteHT_abbIRB4.m` from assignment 1.

Exercise 2 .....

Compute the Jacobian of each cm $\mathbf{J}^0_{cm_i}$ relative to the robot base ($link\,0$). It is also recommended that you use the generated generic functions from Lab 1 to complete this task. Then, finalize the function `Model/Jcm_abbIRB4.m`. This function receives a vector of joint positions ($\mathbf{q}$) and the array of kinematic and dynamic parameters L_d. The output is four Jacobian matrices corresponding to each of the cms, $\mathbf{J}^0_{cm_i} \in \mathbb{R}^{m\times n}, i = 1, 2, 3, 4$.

NOTE: Exercises 1 and 2 apply the knowledge and symbolic functions generated in the Lab 1.

Exercise 3 .....

The models obtained in Exercise 1 and 2 will be used to compute the $\mathbf{M}(\mathbf{q}), \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ matrices and $\mathbf{G}(\mathbf{q})$ vector, from equation (1). These models will be generated following the Euler-Lagrange formulation explained in the course (see **SSY156_Lecture09_Dynamics_II.pdf**, pp. 140-141). It is recommended that you create general functions, using, for example, Matlab's Symbolic Toolbox to automate this process. The iterative method requires the inertia tensors of each link $\mathcal{I}_{cm_i}$ relative to its coordinate frame O_{cm_i}. Use a generic inertia tensor, e.g.,

$$\mathcal{I}_{cm_i} = \begin{bmatrix} I_{i_{11}} & I_{i_{12}} & I_{i_{13}} \\ I_{i_{12}} & I_{i_{22}} & I_{i_{23}} \\ I_{i_{13}} & I_{i_{23}} & I_{i_{33}} \end{bmatrix}$$

NOTE: Please notice that $\mathcal{I}_{cm_i}$ is a symmetric matrix, therefore the upper- and lower- off-diagonal elements are mirrored, i.e. $\mathcal{I}_{i_{jk}} = \mathcal{I}_{i_{kj}}$.

In Matlab, you can define the inertia tensor of cm_1 as:

```
syms I111 I112 I113 I122 I123 I133 real
I1=[I111, I112, I113; I112, I122, I223; I113, I123, I133]
```

The obtained models (matrices and vectors) should be used to fill in the function `Model/Dynamic_abbIRB4v2.m`, for the $\mathbf{M}(\mathbf{q}), \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ matrices and $\mathbf{G}(\mathbf{q})$ vector.

Dynamic Model Validation

Now you will validate the dynamic models (equations of motion in matrix form) generated in Exercises 1, 2, and 3. You will use the dynamic simulator `SimulatorABB_IRB_4DOF/DSimulator_abbIRB4v2.slx` to validate the different models.

Exercise 4 .....

The evaluation consists of two parts: 1) the cm poses, and 2) the robot dynamics:

1) Look at the function `SimulatorABB_IRB_4DOF/plotRobot_dynamic.m`. This function calculates the HTs of all the cms (wrt the robot base and w) which will be used to plot the cm's coordinate frames. This function receives the joint positions ($\mathbf{q}$), and the current simulation time (t).

2) Complete function `Models/Dynamic_abbIRB4v2.m`. This function calculates the forward dynamics model generating as output the joint accelerations ($\ddot{\mathbf{q}} \in \mathbb{R}^4$). This function receives the joint positions and velocities ($\mathbf{q}, \dot{\mathbf{q}}$), the magnitude of the gravitational acceleration (g), the viscous friction parameters $[b_1, b_2, b_3, b_4]$, and the current simulation time (t).

Exercise 5 .....

You can launch the simulator by running `SimulatorABB_IRB_4DOF/run_abb_irb1204_dynamicv2.m`. This will open the robot simulation and the joint position, velocity, and acceleration plots. You can run the simulation by clicking on "Run" (see Fig.2). In your submitted report, please provide figures showing the robot cf (ef and cms) following the DH convention, and the DH tables for links and cms. Also, provide a brief discussion explaining your interpretation of the obtained results. Support your discussion with plots and figures that you obtained running the simulation with your code. Validate your models with different viscous parameters, e.g., $\mathbf{B} = [0.1, 0.1, 0.1, 0.1]$, $\mathbf{B} = [0.2, 0.31, 0.51, 0.71]$, and in different initial conditions, e.g., $\mathbf{q}_{initial} = [0, 0, 0, 0]$, $\mathbf{q}_{initial} = [\frac{\pi}{4}, -\frac{\pi}{2}, \pi, -\pi]$, etc.

Exercise 6 .....

OPTIONAL: 0.25 p

Find the best initial joint position $\mathbf{q}_{initial} = [?, ?, ?, ?]$ that approximates the robot to an unstable equilibrium point. Include the joint position it in the report, as well as a figure similar to Fig. 3 with the simulation results. A system is in unstable equilibrium if, when a disturbing force is applied, the center of gravity is shifted and the object moves away from its original position.

NOTE: In simulation, the robot will not keep the unstable equilibrium due to numerical errors. This means, even with the correct $\mathbf{q}_{initial}$, the robot eventually will be attracted to the stable equilibrium point. Therefore, find the *best* $\mathbf{q}_{initial}$ that keeps the robot in equilibrium for at least a couple of seconds.

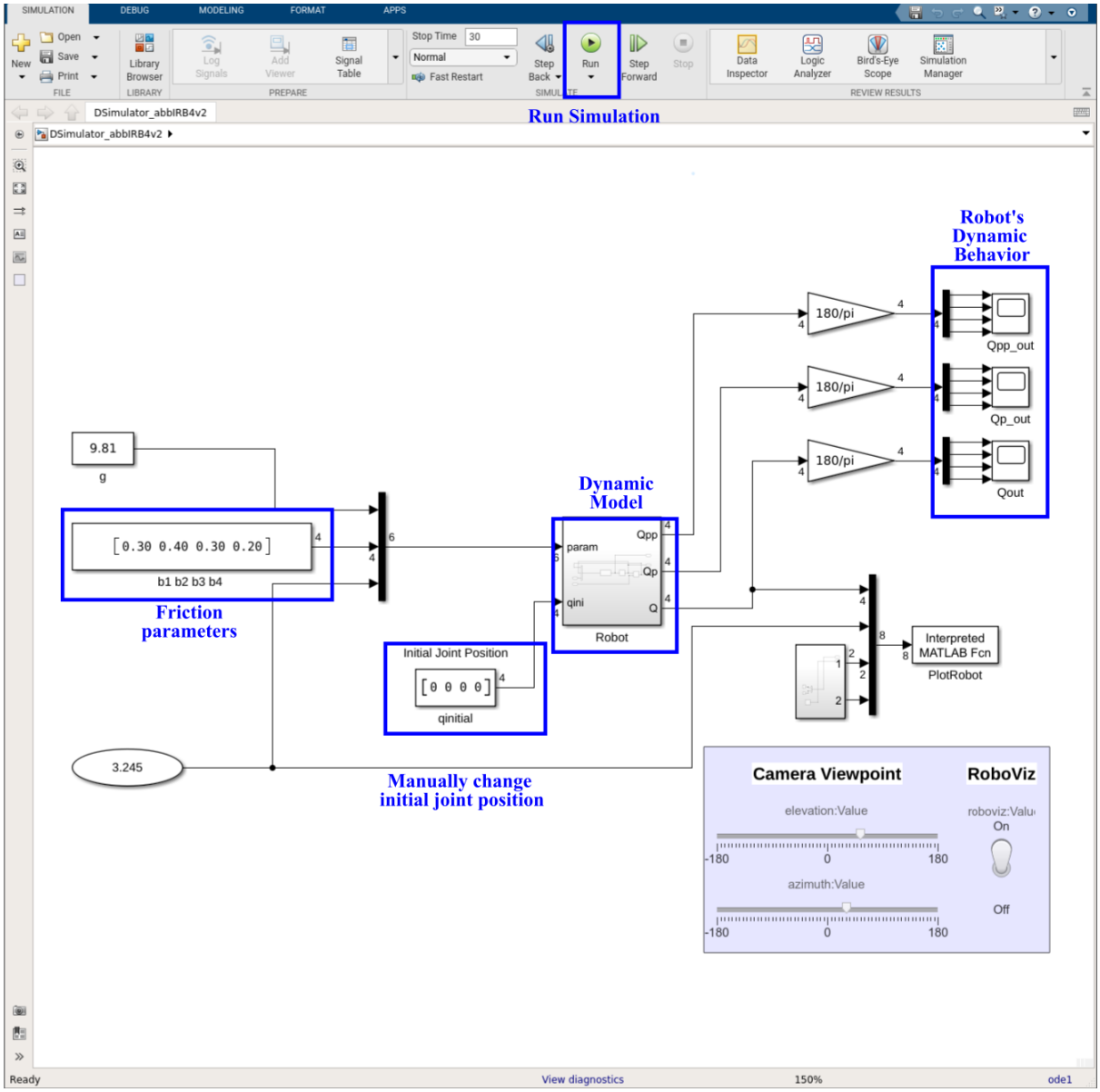

Figure 2: Simulink model for the dynamic model test.

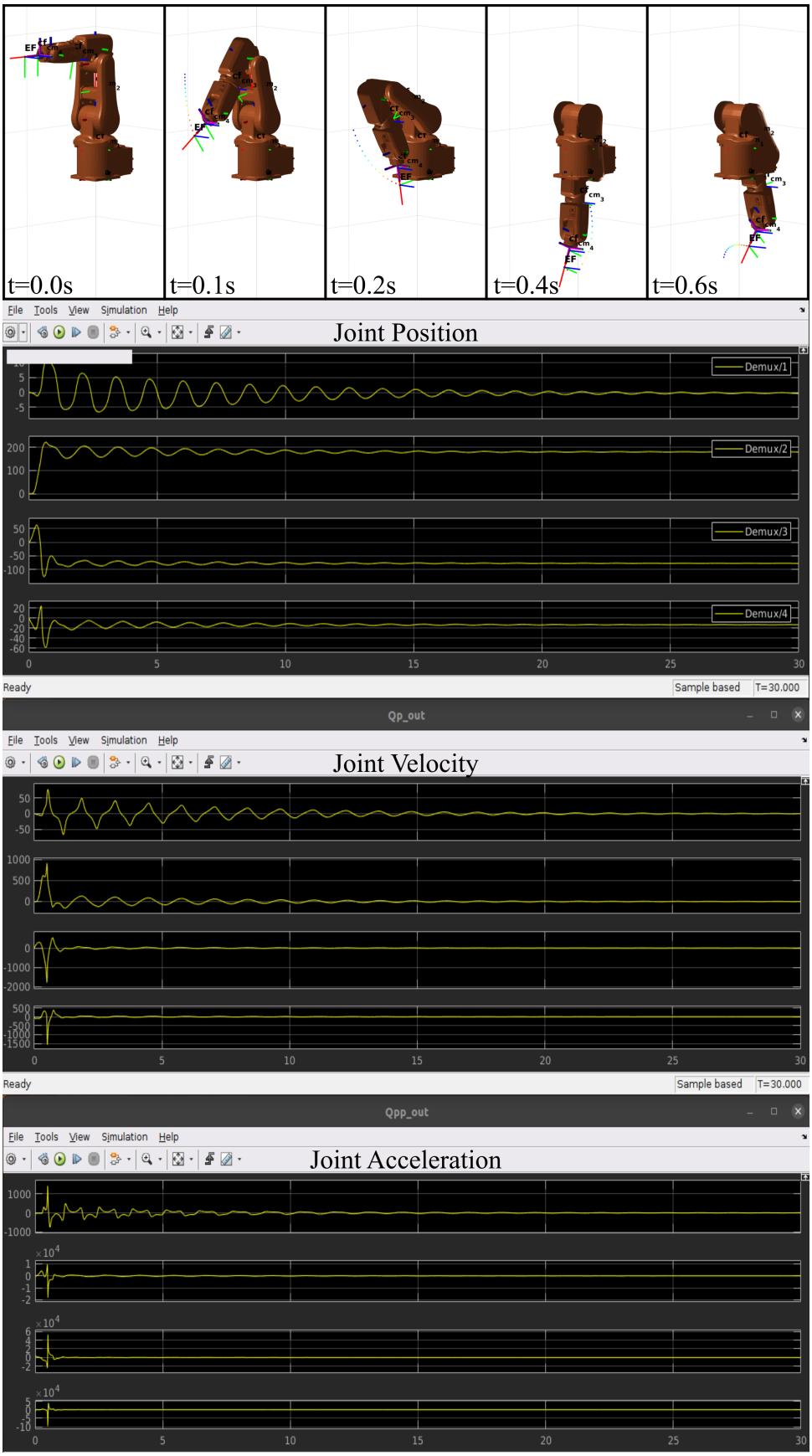

Figure 3: Expected results from Exercise 5. Top: Snapshots of the robot's behavior, Bottom: Joint positions, velocities, and accelerations.

Kinematic Control KUKA Robot (Operational Space)

Differential kinematics can be used to control the joint velocities to follow a desired trajectory. The control concept using joint velocities as system input is known as kinematic control. In our case, the ABB IRB 120 robot has 4 DOFs which means that we cannot control the full pose of the end-effector (6D). In the following exercises, you will control the KUKA IIWA 7 robot, which has 7 DOF. Fig. 4 shows the kinematic chain of the robot and its parameters.

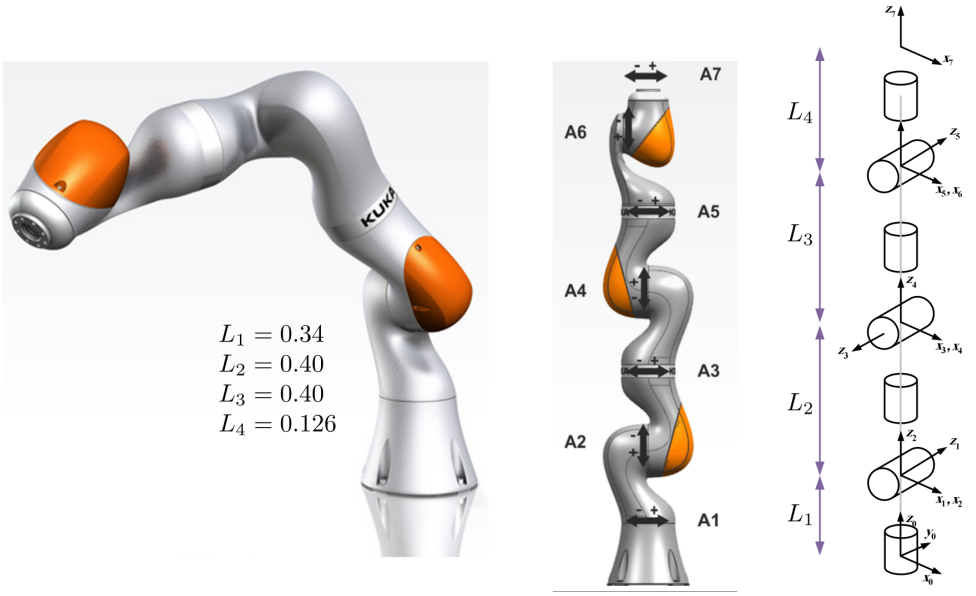

Figure 4: KUKA IIWA 7 robot. The figure shows the kinematic parameters and the kinematic chain for the robot.

For these exercises, you will use the models and the functions developed in the optional part of Lab 1 namely, `J_EF_kukaIIWA7.m`, `getAbsoluteHT_kukaIIWA7.m`, and `kukaIIWA7_params.m`. You will also need to complete new functions located in the folder `Robot_Kuka_IIWA_7DOFs/`. Specifically, inside the `Model`, `Control`, and `KinematicControl` folders.

Exercise 7 .....

OPTIONAL: 0.5 p

Complete Matlab functions `Model/R2_euler_zyz.m` and `Model/euler_ZYZ_2R.m`, without using Matlab inbuilt functions.

The first function receives as input a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ and transforms it into an Euler Angles (ZYZ) representation, i.e., the output is a vector $\boldsymbol{\phi} = [\varphi, \theta, \psi]$ with the three Euler angles corresponding to the consecutive rotations: $\mathbf{R}(\boldsymbol{\phi}) = \mathbf{R}_z(\varphi)\mathbf{R}_y(\theta)\mathbf{R}_z(\psi)$.

The second function is the inverse operator of the first function. This function receives as input the Euler angles (ZYZ) $\boldsymbol{\phi} = [\varphi, \theta, \psi]$ and generates the corresponding rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$.

In the same folder, you will find a Matlab function `Model/FK_kukaIIWA7.m`. This function represents the Forward Kinematic model for the robot, i.e., transforms the HT ($\mathbf{H}^0_{ef}$) to a vector $\mathbf{x}^0_{ef} = [\mathbf{t}^0_{ef}, \boldsymbol{\phi}^0_{ef}]^\top \in \mathbb{R}^6$. You don't have to modify this function.

Exercise 8 .....

OPTIONAL: 0.25 p

Complete function `KinematicControl/Tf.m`. The input argument of this function is the Euler Angles (ZYZ) vector $\boldsymbol{\phi} = [\varphi, \theta, \psi]$. This function computes the matrix $\mathbf{T}$ that transforms time derivatives of Euler Angles (ZYZ) into angular velocities wrt to robot base (Link 0), i.e. $\boldsymbol{\omega} = \mathbf{T}\dot{\boldsymbol{\phi}}$, with $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]$ and $\dot{\boldsymbol{\phi}} = [\dot{\varphi}, \dot{\theta}, \dot{\psi}]^\top$ (see **SSY156_Lecture05_DifferentialKinematics_I.pdf**, pp. 13-14).

Exercise 9 .....

OPTIONAL: 1 p

Complete the following Matlab functions, without using Matlab inbuilt functions:

`KinematicControl/Pol5th3DOF.m`: This function calculates a smooth trajectory for the end-effector using a polynomial function (see **SSY156_Lecture07_Differential- Kinematics_III.pdf**, pp. 133-140). The input arguments are initial and final position or orientation vector $P_{ini}, P_{end} \in \mathbb{R}^3$, the initial and final time for the trajectory t_{ini}, t_{end}, and the current time t. The output is the vector $P(t) \in \mathbb{R}^3$ which defines the desired position or orientation of the end-effector in the time t. It is recommended to use a minimum Jerk trajectory, i.e., the initial and final acceleration are zero.

`KinematicControl/desired_op_trajectory.m`: This function samples generates the desired pose of the end-effector wrt the world coordinate frame (wcf), $\mathbf{H}_d^W(t) \in \mathbb{R}^{4\times4}$. The function has two modalities (defined by the variable `splineFlag`):

1: The function generates a spline trajectory from the initial ef pose $\mathbf{x}_{ef}^W(t_0) \in \mathbb{R}^6$ to the desired pose $\mathbf{x}_d^W(t_f) \in \mathbb{R}^6$ at the time $t = t_f$. In this case, the output is the desired end-effector pose sampled at the time t, i.e., $\mathbf{x}_d^W(t) \in \mathbb{R}^6$ and $\mathbf{H}_d^W(t) \in \mathbb{R}^{4\times4}$.

0: The function sends a constant desired end-effector position. In this case, the output of the function is: $\mathbf{x}_d^W(t) = \bar{\mathbf{x}}_d^W$ and $\mathbf{H}_d^W(t) = \bar{\mathbf{H}}_d^W \forall t$.

Exercise 10 .....

OPTIONAL: 1.5 p

Complete function `Robot_Kuka_IIWA_7DOFs/Control/kinematicCtrlOp.m`. In this function, you will code a kinematic controller that commands the joint velocity of the robot ($\dot{\mathbf{q}} \in \mathbb{R}^7$). The controller is defined in operational space. This means, we will define the desired pose of the end-effector (ef) wrt link 0 ($\mathbf{x}_d^0 \in \mathbb{R}^{3 \text{ or } 6}$), and the controller generates the commanded joint velocities ($\dot{\mathbf{q}} \in \mathbb{R}^7$) to track the desired pose with the end-effector, i.e., $\mathbf{x}_{ef}^0(t) \approx \mathbf{x}_d^0 \ \forall t$. The kinematic controller has two modalities defined by the variable `ctrl_type`:

1: The controller regulates the pose of the end-effector, i.e., $\Delta\mathbf{x}_{ef}^0 = \mathbf{x}_d^0 - \mathbf{x}_{ef}^0 \in \mathbb{R}^6$. Where $\mathbf{x}_d^0 = [\mathbf{t}_d^0, \boldsymbol{\phi}_d^0]^\top$, with $\mathbf{t}_d^0$ as any reachable position by the ef, e.g., $[0.55, 0.2, 0.3]^\top$, and $\boldsymbol{\phi}_d = [0.1, \frac{\pi}{2}, 0.2]^\top$. In this case, you need to use the full analytic Jacobian ($\mathbf{Ja}_{ef}^0 \in \mathbb{R}^{6\times7}$) (see **SSY156_Lecture06_Differential Kinematics_II.pdf**, pp. 90-92).

0: The controller only regulates the position of the end-effector, $\Delta\mathbf{x}_{ef}^0 = \mathbf{t}_d^0 - \mathbf{t}_{ef}^0 \in \mathbb{R}^3$, where $\mathbf{t}_d^0$ is any reachable position by the ef, e.g., $[0.55, 0.2, 0.3]^\top$. In this case, you will only use the linear velocity part of the analytic Jacobian ($\mathbf{Ja}_{v_{ef}}^0 \in \mathbb{R}^{3\times7}$).

Finally, you will implement a simple emergency stop strategy in this function. The strategy should stop the robot, i.e. $\dot{\mathbf{q}} = 0 \in \mathbb{R}^7$, when the manipulator is close to a singularity.

Exercise 11 .....

OPTIONAL: 1.5 p

In this exercise, you will simulate the operational kinematic controller under different conditions. To run the simulator, run the `KinematicControl/run_kuka_iiwa7_KinCtrl.m` script, which will open `KSimulator_kuka_iiwa7_kinCtrl.slx` (see Fig. 5). This Simulink model offers two switches to control the output of the trajectory generator (Spline/constant) and the dimension of the operational task (6DOF/pose or 3DOF/position).

To complete this exercise, you need to do the following tasks and report the results:

Task 1: The initial joint position is defined by a Constant block, see Fig. 5. The default initial position sets the robot in a singular configuration, i.e., the Jacobian is not full-rank. You need to define a initial position $\mathbf{q} = [q_1, q_2, q_3, q_4, q_5, q_6, q_7]^\top$ in such a way that the robot is not in the singular configuration.

Task 2: Test the kinematic controller to regulate the operational position only (3DOF) [Switch Pose: off]. You will evaluate the controller with constant desired ef position [Switch Spline: off], and time-varying desired ef position [Switch Spline: on].

Task 3: Test the kinematic controller to regulate the operational full-pose (6DOF) [Switch Pose: on]. You will evaluate the controller with constant desired ef pose [Switch Spline: off], and time-varying desired pose [Switch Spline: on].

Task 4: Test the emergency stop. In this case, you will set the ef desired position outside the manipulator's reachability space. The emergency stop should be triggered before the robot shows an unstable behavior. Use the operational controller for position only [Switch Pose: off].

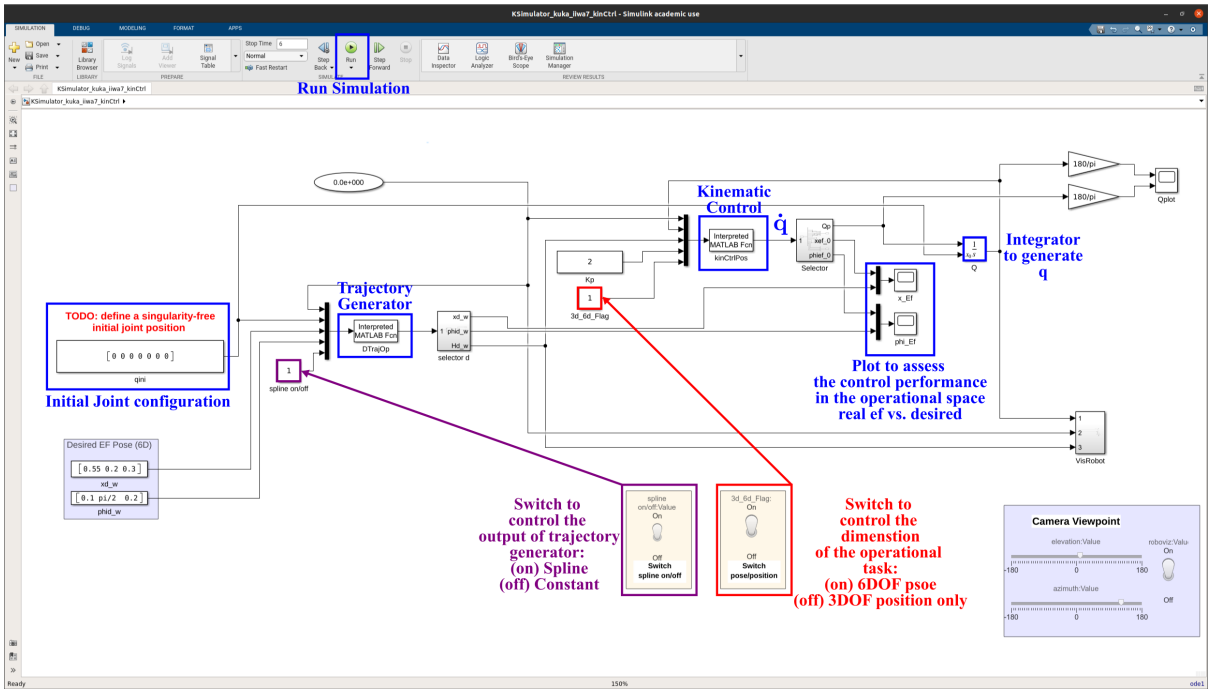

Figure 5: Simulink model for the Kinematic Controller in the Operational Space.

For **Task 1**, provide the selected initial joint position $\mathbf{q} = [q_1, q_2, q_3, q_4, q_5, q_6, q_7]^\top$ in the report. For **Tasks 2**, **3**, and **4**, please provide in the report, plots/figures showing the behavior of the robot, and a brief discussion with your reflections. Note that Fig. 6 shows examples of the expected results from Exercise 11.

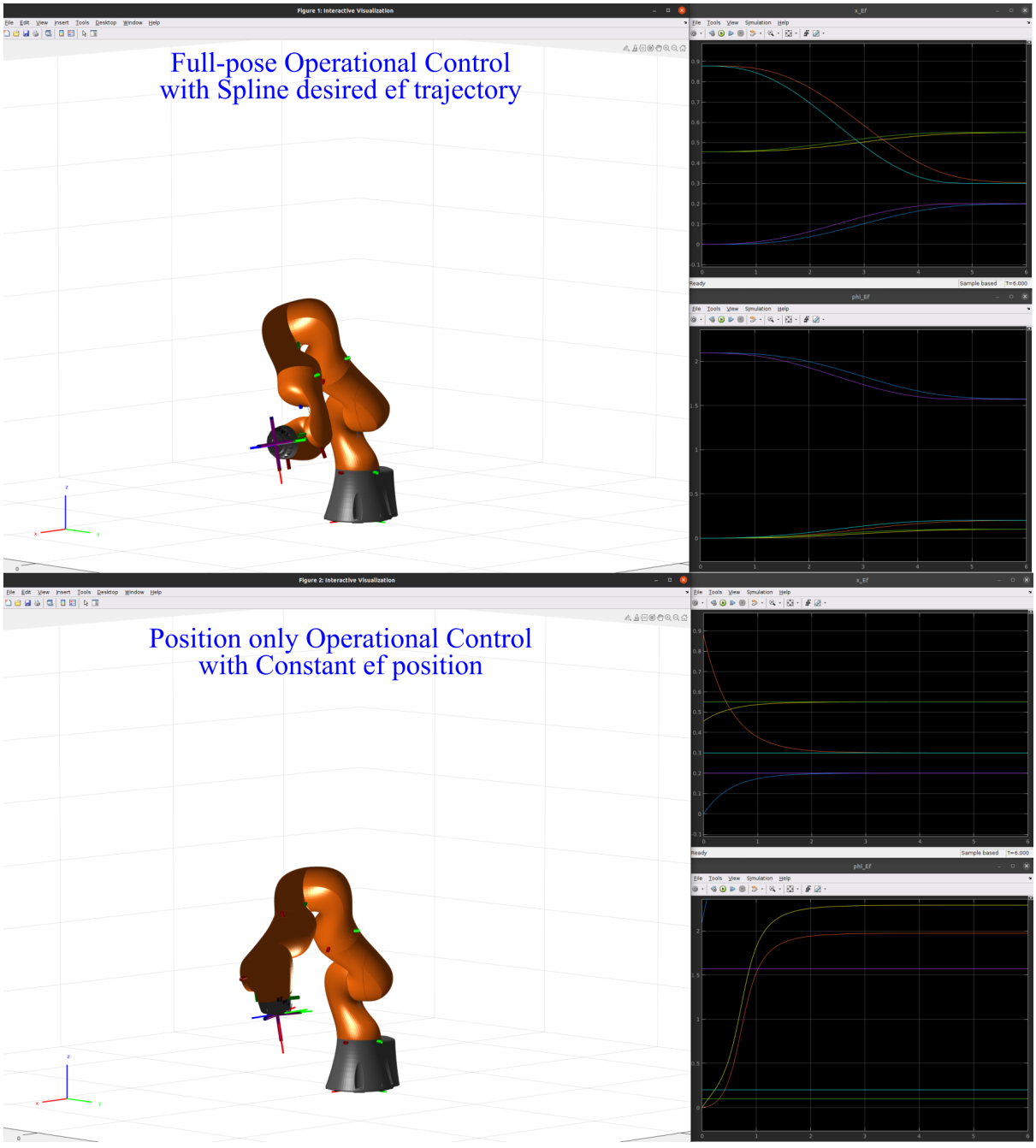

Figure 6: The figure shows examples of the expected results from Exercise 11. Top: Full-pose operational control with Spline function to generate the desired end-effector (ef) pose. Bottom: Position-only operational control with constant desired ef position.