9, June, 2022

Fabio Conti
Mat.4693053

Robot Dynamics and Control

Third assignment: report

Introduction:

The third assignment of the Robot Dynamics and Control course focuses on the **Dynamic control of a manipulator**. The following 4 exercises will show how a simple simulation model of a **UR5 robot** can be controlled providing torque commands to the joints. The focus of the exercises was to create some simple **Simulink programs** capable of solving some fundamental problems of the manipulation control such as: gravity compensation, set point regulation in the cartesian and joint space, and the joint space trajectory tracking problem. The representation of the rigid-body chain's motion is defined by the following generalized equation of motion expressed in the Euler-Lagrange formulation for manipulators with actuated joints:

$$\boldsymbol{\tau} = \boldsymbol{A}(q)\ddot{q} + \boldsymbol{B}(q,\dot{q})\dot{q} + \boldsymbol{C}(q)$$

Where:

- $\boldsymbol{\tau}$: generalized torque or force input vector, ***[nx1]***
- $\boldsymbol{A}(q)$: inertial acceleration-related symmetric matrix, ***[nxn]***
- $\boldsymbol{B}(q,\dot{q})$: nonlinear Coriolis and centrifugal force vector, and ***[nx1]***
- $\boldsymbol{C}(q)$: gravity loading vector. ***[nx1]***

The focus of each problem instance is to provide a **torque command** in the joint space capable of regulating the motion of the manipulator's links to reach a certain target position. Each solution was approached using the **Lyapunov stability theory**. Given a generic control action objective, such as zeroing the velocity of a specific link of the manipulator and considering ***V(x)*** as a positive definite function of the state of the system which exploits the kinetic energy of the system, the **Lyapunov method** is the procedure that aims to find the appropriate **control signal *u(t)*** (input torques vector) that would ensure that the time derivative of ***V(x)*** is always negative. This input vector will cause ***V(x)*** to drop until it reaches zero kinetic energy causing the link to stop moving. All the exercise requirements can be met by applying multiple interpretations to this basic concept of control.
The exercises' system inputs will be generated by creating a Simulink block diagram capable of providing the correct input. Most of the operating blocks used come from some add-on external Simulink packages such as:

- *Robotic Systems Toolbox*
- *Simscape multibody*
- *Robot library data support package*

The output of those blocks make the reconstruction of the input signal fairly easy to compute.

Exercise 1:

Gravity compensation:

The gravity compensation is one of the basics of the control of manipulators. The Gravity Torque block returns runtime the joint torques required to hold the robot at a given configuration with the current Gravity setting on the Rigid body tree robot model. Giving the current configuration of the robot as input to the block, the output would be the torque input vector capable of holding the robot in place. The output of the block is called **C(q)** and it represents the [6(number of links) x 1] gravity loading vector.

Simulink system:

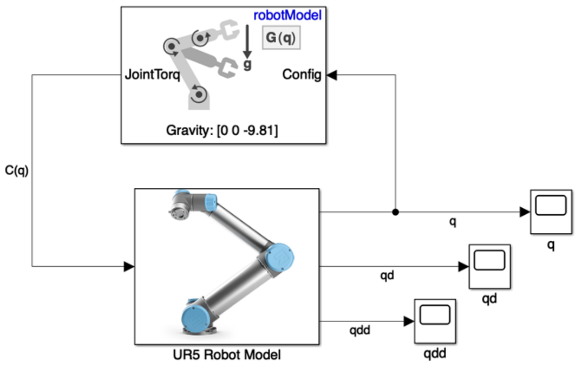
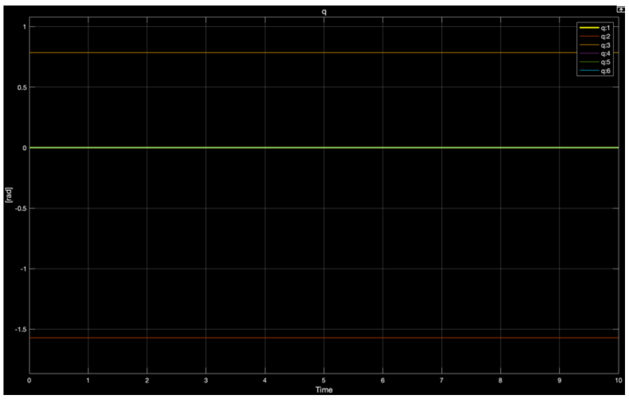


Image 1: block diagram of the ex.1

Graphs:



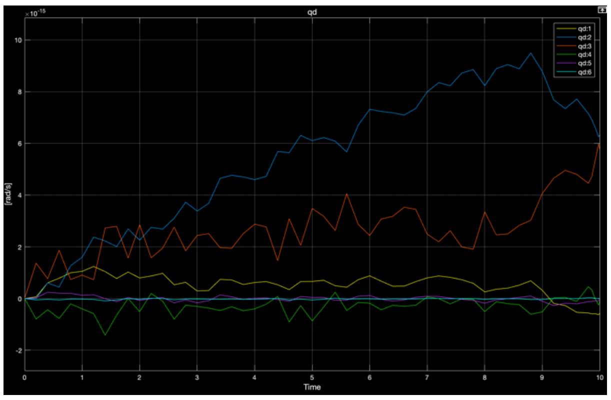
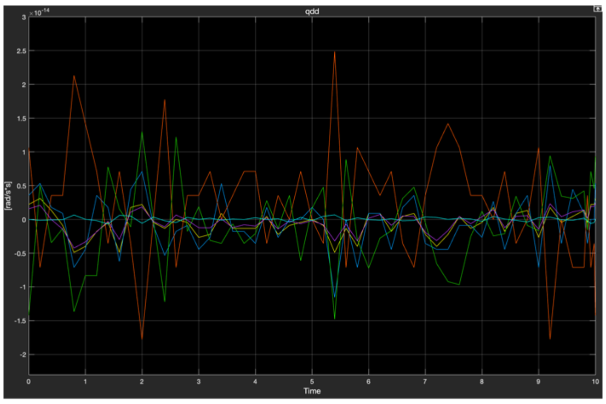


Image 2: configuration of the first exercise

Image 3: joints velocity of the first exercise



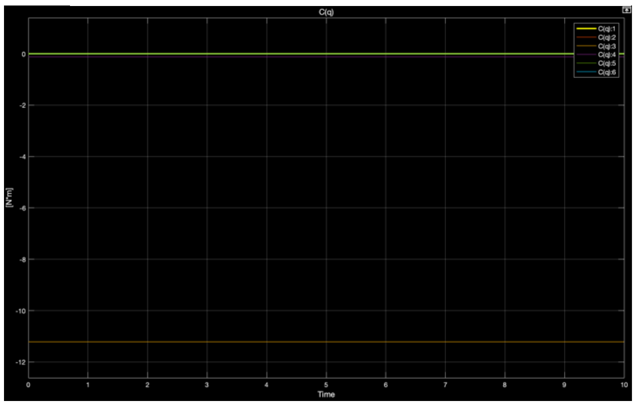


Image 4: joints acceleration input of the first exercise

Image 5: joints torque input of the first exercise

Comments:

From the graph of the joint position and the joints velocity we can clearly see the effect of the Gravity block. The block stabilizes the position of the joints to not change overtime making it constant inside of the simulated time span. The second graph shows some very small oscillations of the joints velocity which can be approximated to zero.

Exercise 2:

Linear joint control:

The second exercise is again another typical problem of manipulation control. Given a desired joint configuration $\mathbf{q}^* = \mathbf{q}_0 + \Delta\mathbf{q}$ where $\mathbf{q}_0$ is the initial joint configuration and $\Delta\mathbf{q} = \left[\frac{\pi}{4}; -\frac{\pi}{6}; \frac{\pi}{4}; \frac{\pi}{3}; -\frac{\pi}{2}; \pi\right]$ the required joint's displacement, the exercise asks to provide torque command to reach the $\mathbf{q}^*$ configuration and compare the results with and without gravity compensation.
The kind of control action required to take care of this task is of the *Proportional-Derivative* kind + Gravity compensation. The reaching of a new $\mathbf{q}$ configuration is called **set point regulation**.

The **control law** created to solve this problem has the following structure:

$$\boldsymbol{u}(t) = -K_v\dot{\boldsymbol{q}}(t) - K_p\boldsymbol{\delta}q(t) + \boldsymbol{C}(\boldsymbol{q})$$

Where:

- $\boldsymbol{u}(t)$: input torque command.
- K_v: Diagonal matrix of velocity (derivative) gain. One element of the diagonal for each of the joint of the manipulator.
- K_p: Diagonal matrix of velocity (proportional) gain. One element of the diagonal for each of the joint of the manipulator.
- $\boldsymbol{\delta}q(t)$: error between the desired position configuration and the actual one.
- $\boldsymbol{C}(\boldsymbol{q})$: gravity loading vector.

Following the effect of the control law, the manipulators' joint will be "attracted" to the final position they have to reach. As a result, the robot will reach the desired configuration set at the beginning of the simulation. The following picture shows the control loop schematic designed to match the input control law.

Simulink system:

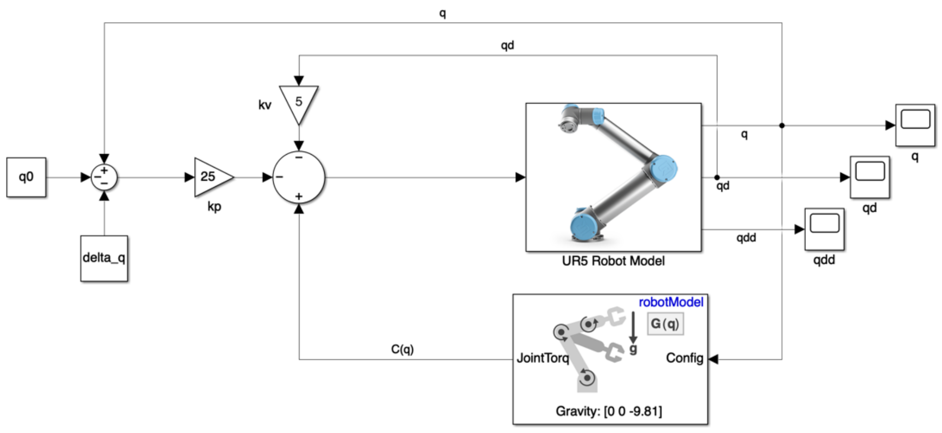


Image 6: block diagram for ex.2

Comments:

The summation block on the left of the schematic shows the calculation of $\boldsymbol{\delta}q(t)$ which expresses the "error" between the final desired position ($\mathbf{q}^* = \mathbf{q}_0 + \Delta\mathbf{q}$) and the actual one $\mathbf{q}$. This value will later be multiplied by the proportional gain K_p and later summed with the $-K_v\dot{\boldsymbol{q}}(t)$ and the gravity compensation term. This simple block diagram designs the desired input.

Graphs:

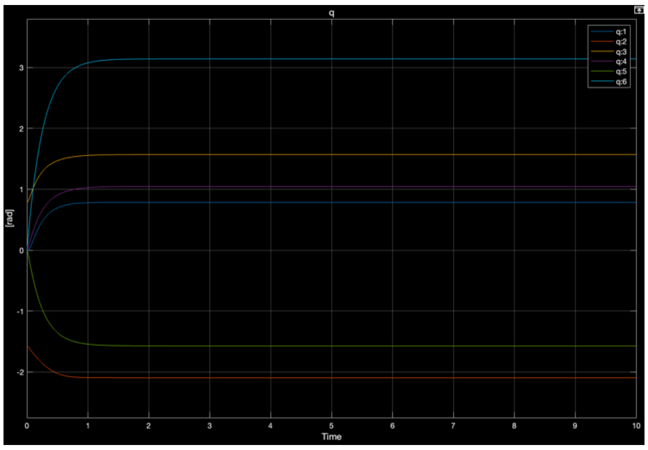


Image 7: configuration of the second exercise

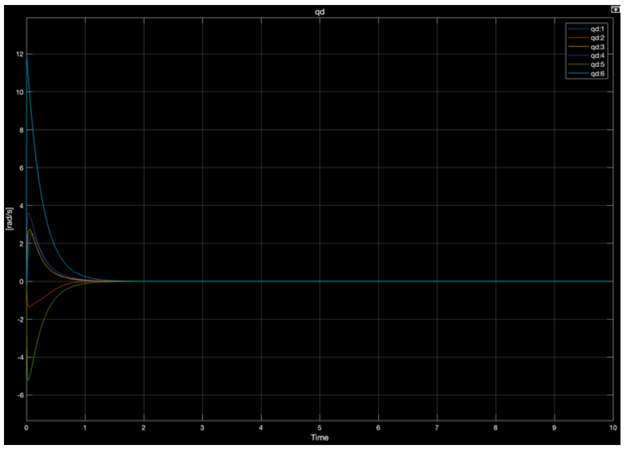


Image 8: joints velocity of the second exercise

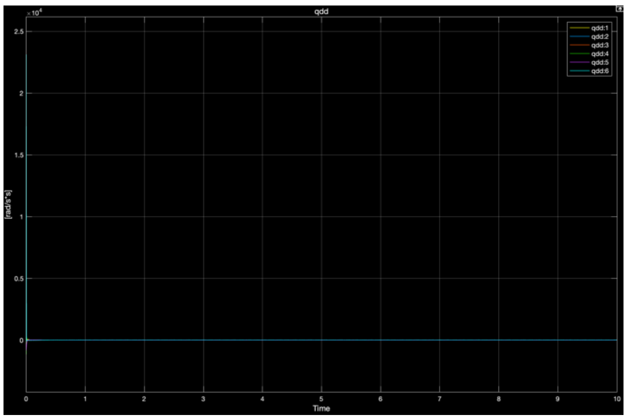


Image 9: joints acceleration output of the second exercise

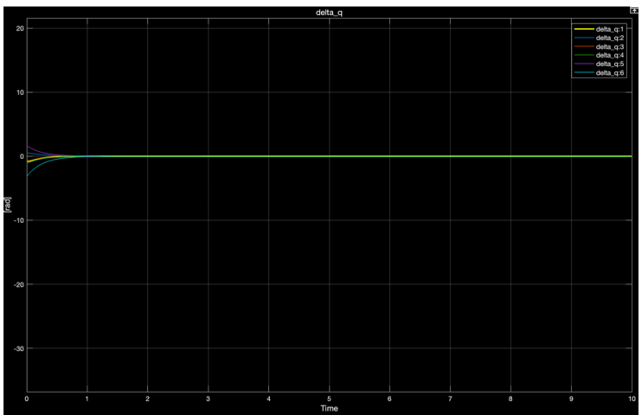


Image 10: joints position error of the second exercise

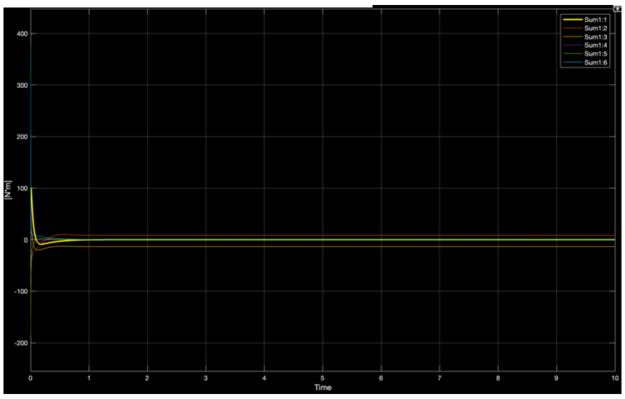


Image 11: joints torque of the second exercise

Comments:

As we can see from the pictures above, the joints will change their orientation form the initial position to the final configuration. For this kind of regulation, no precautions are considered regarding the velocity of the joints' rotation.

What if **the gravity compensation** block is **taken out of the schematic**?

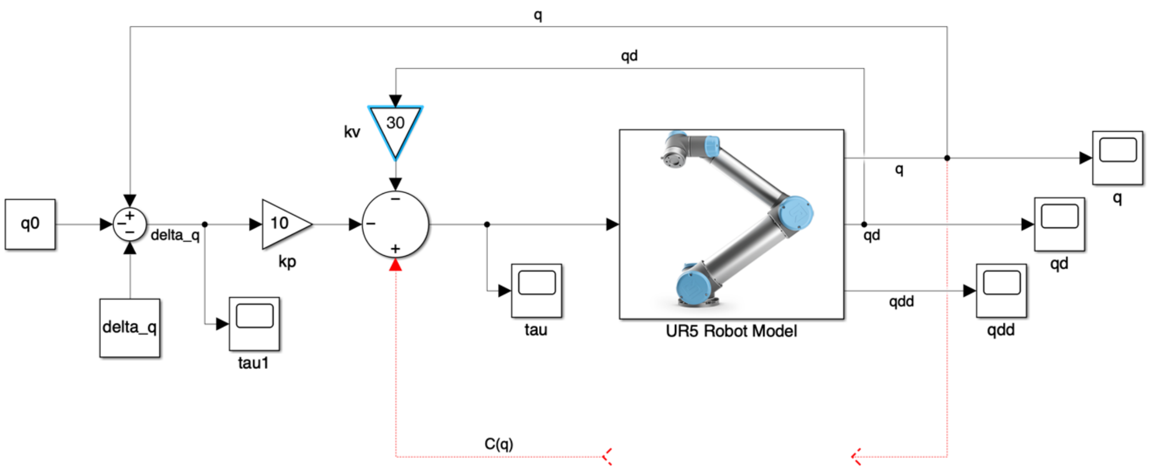
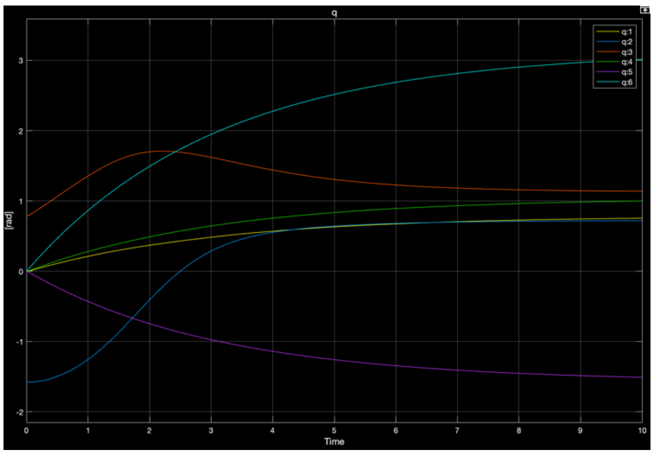


Image 12: second exercise's schematic with no gravity compensator involved

The following graphs will show how the robot dealt with this new block diagram:



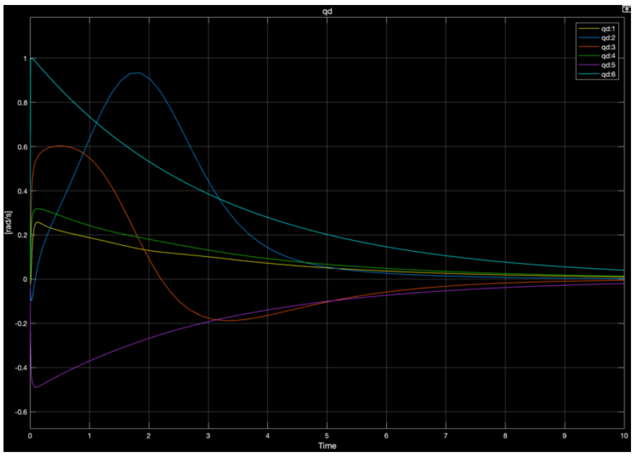
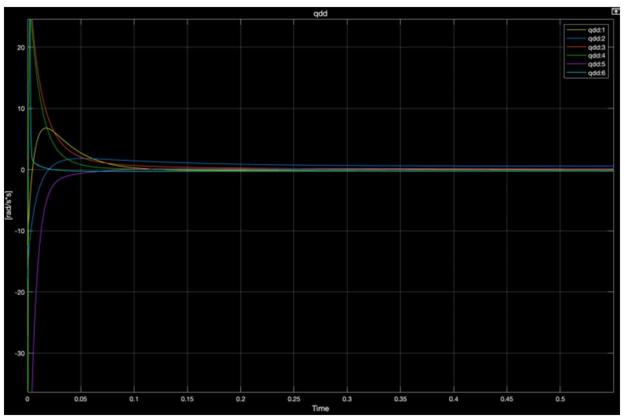


Image 13: configuration of the second exercise (no gravity compensator)

Image 14: joints velocity of the second exercise (no gravity compensator)



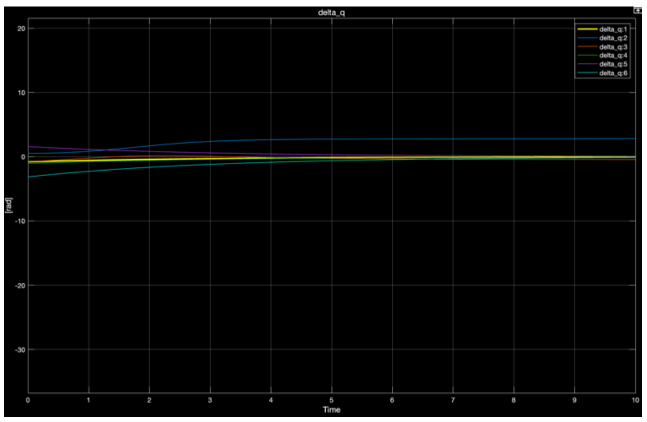


Image 15: zoomed joints acceleration output of the second exercise (no gravity compensator)

Image 16: joints position error of the second exercise (no gravity compensator)

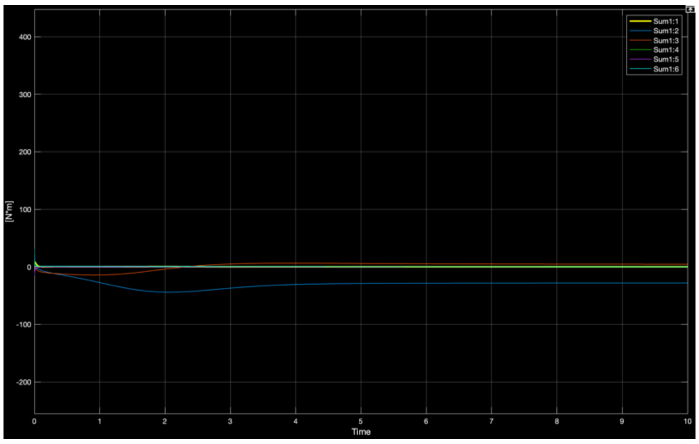


Image 17: joints torque of the second exercise (no gravity compensator)

As we can see form the picture of the configuration **q**, the robot was not able to match the required final position as stated by the input. In this case the regulation of the system was just managed by the ***PD*** controller. If some more realistic values of K_v and K_p are considered in the scheme the resulting motion will be much like the compensated one. From the theory, we can ensure that this change of behavior follows this position error bound law:

$$||\boldsymbol{\delta} q|| \leq \left|\left|{K_p}^{-1}\right|\right| C_o$$

Where:

- C_o: is an upper bound of the **C**(**q**) function.
- ${K_p}^{-1}$: since K_p and K_v are both positive definite and diagonal they can be inverted.

Therefore, a higher and more realistic value of K_p will ensure a much closer behavior of the robot compared to the compensated gravity of the original scheme. The following graphs will underline the similarities between the new values of K_p (*eye(6)*120*) and K_v (*eye(6)*30*) compared to the previous ones.

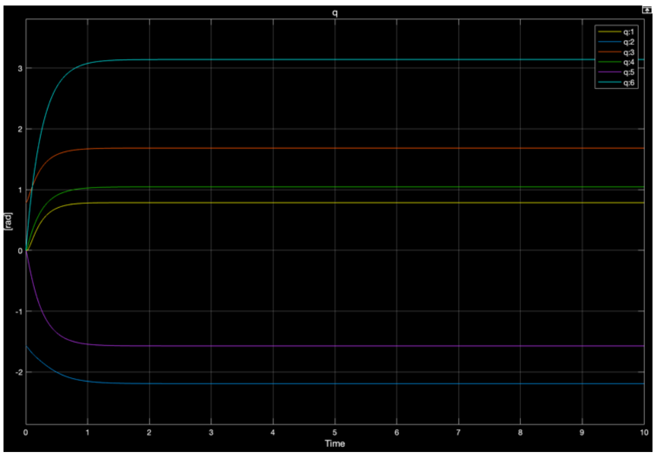


Image 18: configuration of the second exercise (no gravity compensator; new K_p, K_v)

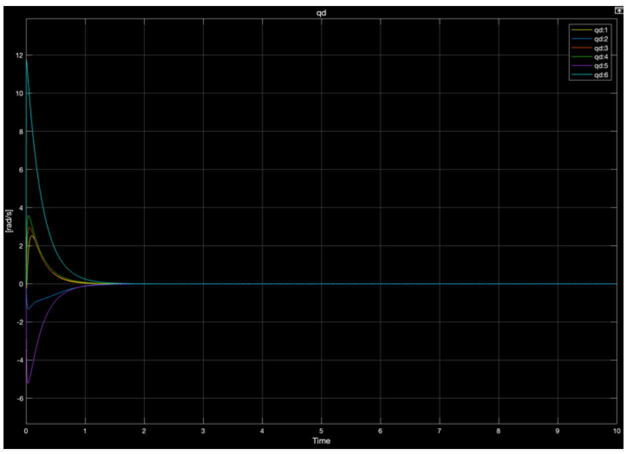


Image 19: joints velocity of the second exercise (no gravity compensator; new K_p, K_v)

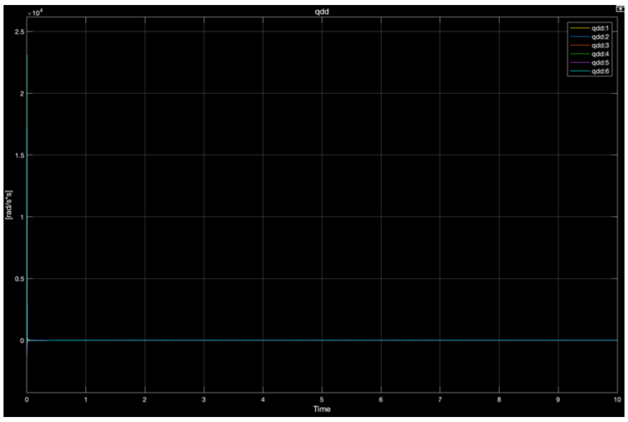


Image 20: joints acceleration output of the second exercise (no gravity compensator; new K_p, K_v)

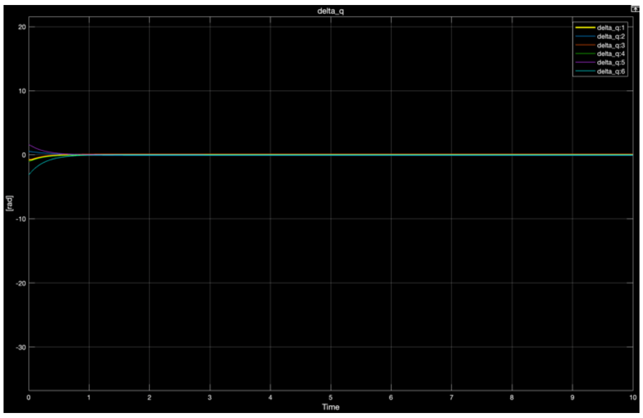


Image 21: joints position error of the second exercise (no gravity compensator; new K_p, K_v)

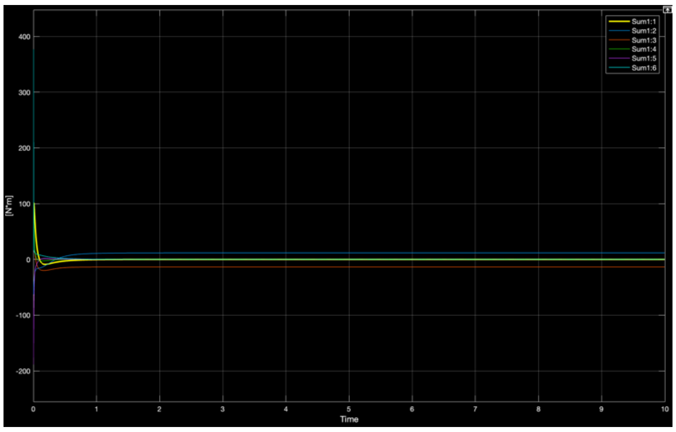


Image 22: joints torque of the second exercise (no gravity compensator; new K_p, K_v)

Exercise 3:

Linear cartesian control:

The third exercise covered the same topic as the second one but instead of having as target position a joint configuration expressed in the joint space, the data was about the position in space of the end effector reference frame. The final configuration required is state as an $\mathbf{x}^* = \mathbf{x}_0 + \Delta\mathbf{x}$ position where the $\mathbf{x}_0$ vector corresponds to the initial positioning of the end effector in space and the $\Delta\mathbf{x}$ vector is the displacement the robot must match. Since the considered robot model is a 6 degrees of freed manipulator, the configuration of the end effector will be univocally represented in the cartesian space by the robot configuration. Therefore, we can compute the unique Cartesian position of the end effector starting from the joint space configuration and add the delta displacement form this configuration. To obtain such vector I exploited some blocks from the initial add-on packages previously mentioned. The first step is to retrieve the transformation matrix between the base frame and the end effector. Given this matrix it is possible to retrieve the global position of the end effector thanks to the Coordinate Transformation conversion block. This computational instance will output the Euler and transformation coordinates to which the $\Delta\mathbf{x}$ values can be added to retrieve the final cartesian end effector position vector. These values can be retransformed back into a transformation matrix that will describe the position of the end effector with respect to the base. All these operations are hold in the global transformation matrix subsystem which inputs the inverse kinematics block. This system will provide as output the configuration of the manipulator in the joint space. This input will represent the $\boldsymbol{\delta}q$ value just like in the previous exercise. The additional inputs to the blocks represents some additional parameters needed to calculate the joints' configuration.

Simulink system:

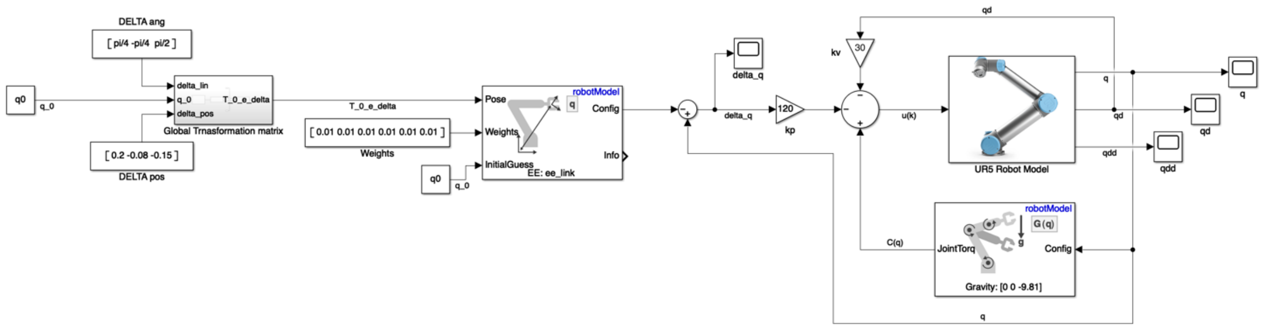


Image 23: exercise 3 full schematic

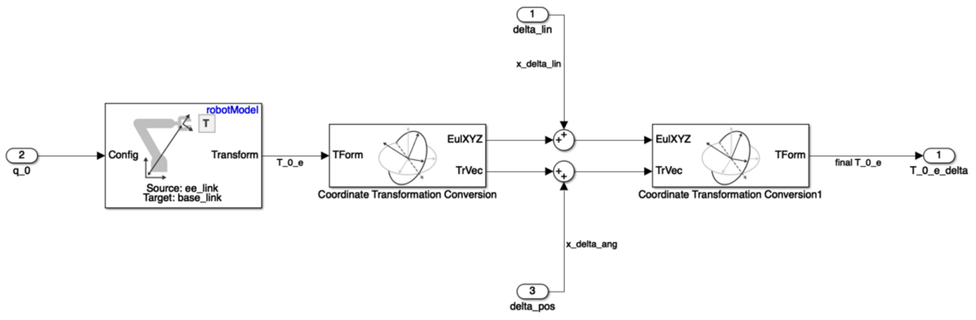
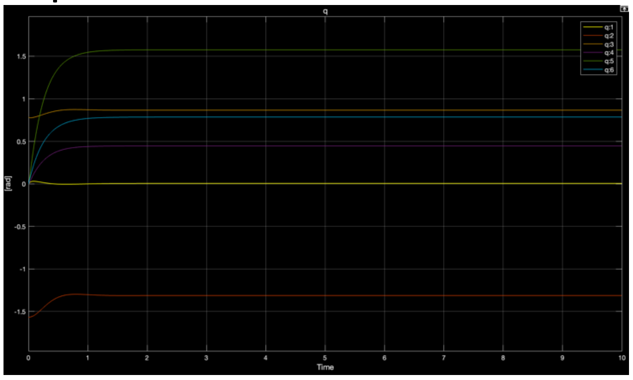


Image 24: exercise 3 subsystem schematic

Comments:

The subsystem will simply calculate the global transformation matrix between the base and the end effector including the initial configuration of the robot and the required delta displacement of this position.

Graphs:



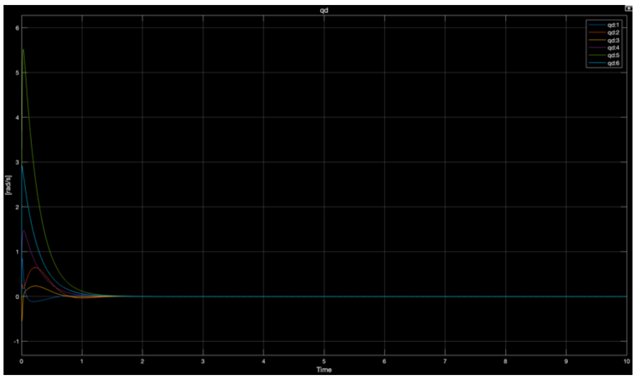


Image 23: configuration of the third exercise

Image 24: joints velocity of the third exercise

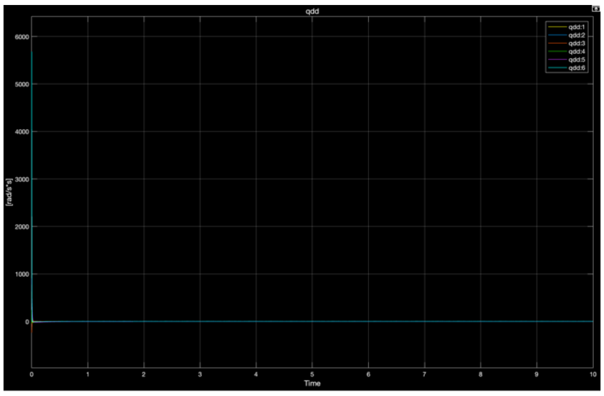


Image 25: joints acceleration output of the third exercise

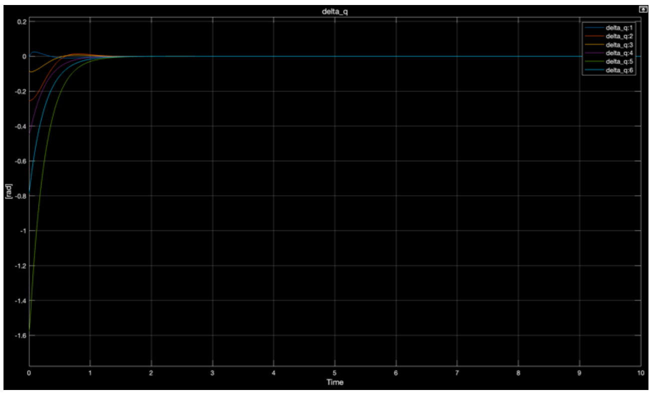


Image 26: joints position error of the third exercise

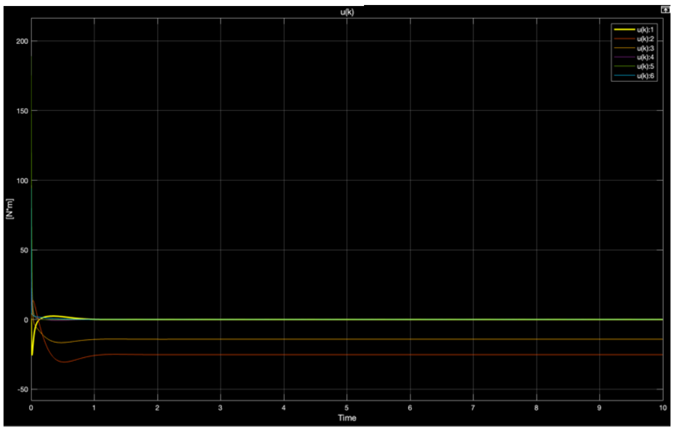


Image 27: joints torque of the third exercise

Comments:

Similarly as before, all the joint positions converged to their final zeroing of the $\boldsymbol{\delta} q$ variable ending up their movements at zero velocity.

Exercise 4:

Computed torque control:

The last exercise required to simulate an actual industrial type of regulation called **joint trajectory tracking**. The objective of the regulation was the same one as the second exercise, but the type of regulation had to follow standard computed-torque controller's control law:

$$\boldsymbol{u}(t) = \boldsymbol{A}(\boldsymbol{q})\left[\,\ddot{\boldsymbol{q}}^* + K_v \boldsymbol{\delta}\dot{q}(t) + K_p \boldsymbol{\delta} q(t)\right] + \boldsymbol{B}(\boldsymbol{q}, \dot{\boldsymbol{q}})\boldsymbol{q} + \boldsymbol{C}(\boldsymbol{q})$$

Where:

- $\boldsymbol{u}(t)$: input torque command.
- K_v: Diagonal matrix of velocity (derivative) gain. One element of the diagonal for each of the joint of the manipulator.
- K_p: Diagonal matrix of velocity (proportional) gain. One element of the diagonal for each of the joint of the manipulator.
- $\boldsymbol{\delta} q(t)$: error between the desired position configuration and the actual one.
- $\boldsymbol{\delta}\dot{q}(t)$: error between the desired joints' velocity and the actual one.
- $\boldsymbol{A}(q)$: inertial acceleration-related symmetric matrix,
- $\boldsymbol{B}(q, \dot{q})$: nonlinear Coriolis and centrifugal force vector, and
- $\boldsymbol{C}(\boldsymbol{q})$: gravity loading vector.

This control law is now capable of managing the trajectory of the robot's motion through a set of points in space considering not only a desired position in space but also the velocity and acceleration to maintain during the movements towards the target points. The objective of the robot is not only set at the begging of the execution, but it is updated runtime throughout the whole simulation. To simulate such feature some way points were set as input of the Trapezoidal velocity profile trajectory block. this block sets some trapezoidal velocity profile to the rotational velocity of the joints to simulate a required velocity. This block will output the required $\boldsymbol{q}^*$, $\dot{\boldsymbol{q}}^*$, and $\ddot{\boldsymbol{q}}^*$ necessary to compute the ***u(t)*** control law. The generated velocity envelope will ramp up to a constant value and then will ramp down to zero. As a result, the acceleration is constant piecewise, and the configuration $\boldsymbol{q}^*$becomes quadratic during the acceleration and deceleration phases, and linear in between.

The benefits of this type of velocity profile are related to the possibility to impose a maximum limit to the manipulator's velocity joints. Moreover, the position will never overshoot its target. These are the primary reasons why this technique is often exploited in industrial safety applications.
The second point of the exercise was to compare the graphs of the normal settings of the robot to a robot model without dumping coefficients. This robot setting would idealize the robot's movements making it way easier for the robot to follow the desired velocity and configuration commands.

Simulink system:

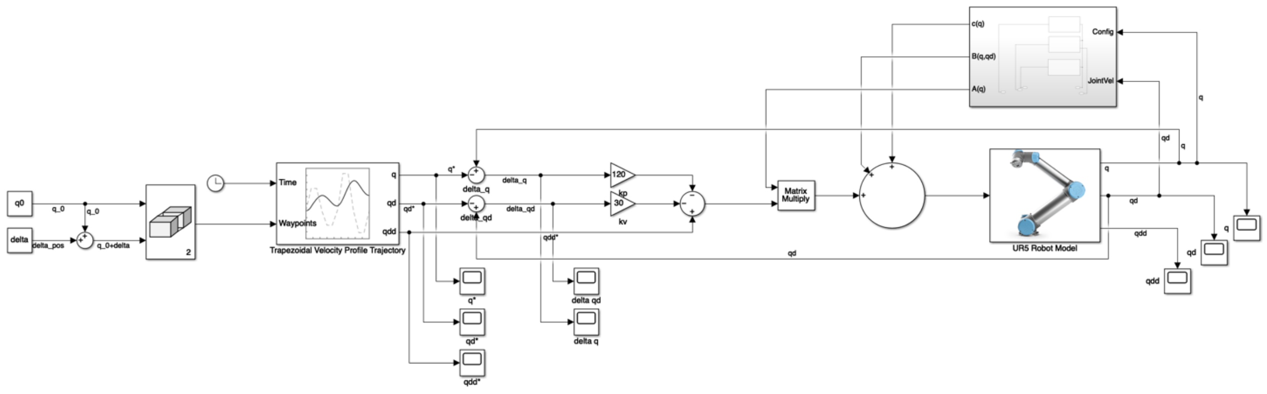


Image 28: exercise 4 full schematic

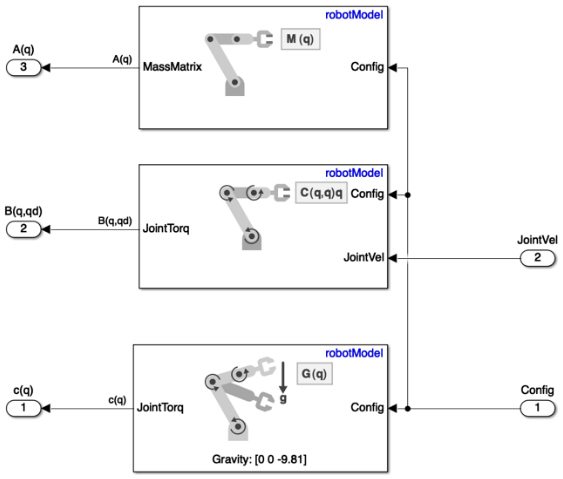


Image 29: exercise 3 subsystem schematic

Graphs:

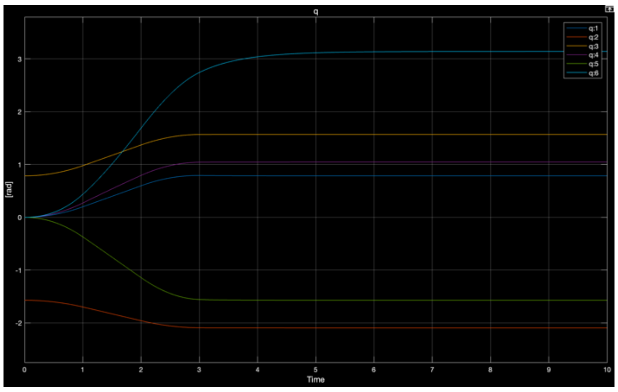


Image 30: configuration of the fourth exercise

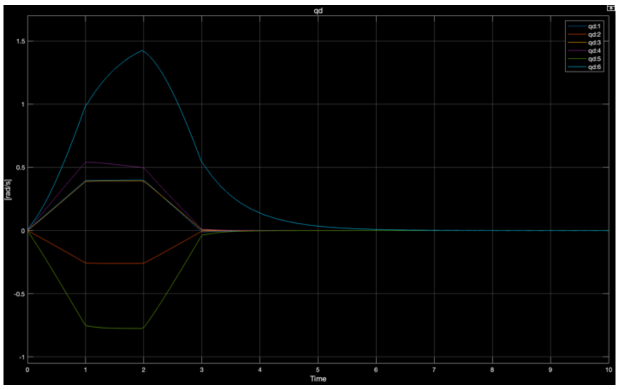


Image 31: joints velocity of the fourth exercise

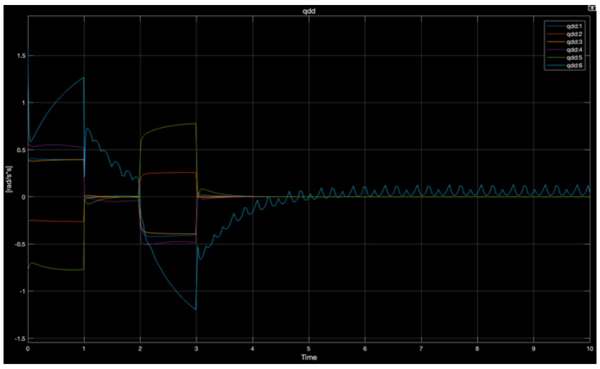


Image 32: joints acceleration output of the fourth exercise

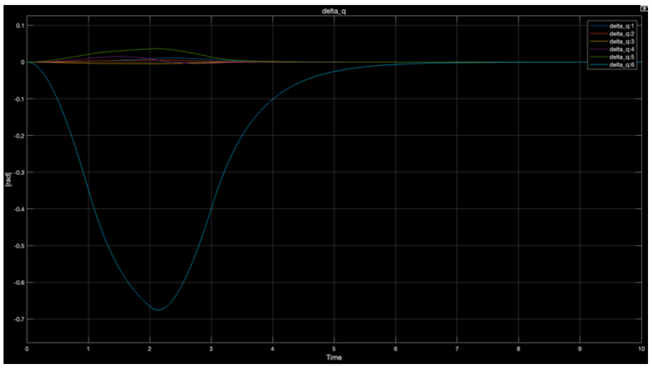


Image 33: joints position error of the fourth exercise

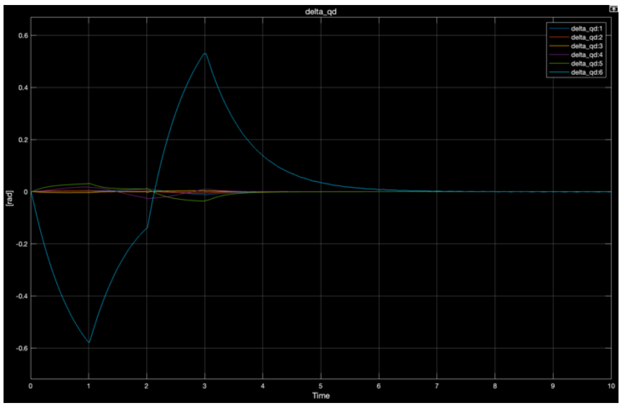


Image 34: joints velocity error of the fourth exercise

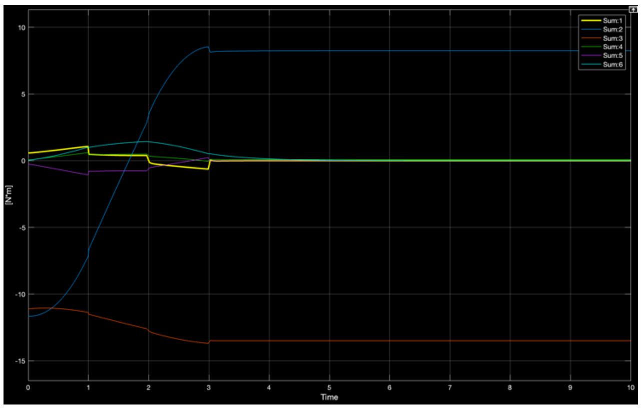


Image 35: joints torque of the fourth exercise

Desired configuration, velocities, and accelerations:

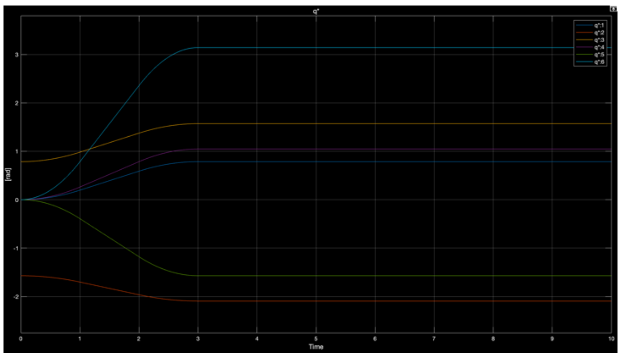


Image 36: desired joint configuration of the fourth exercise

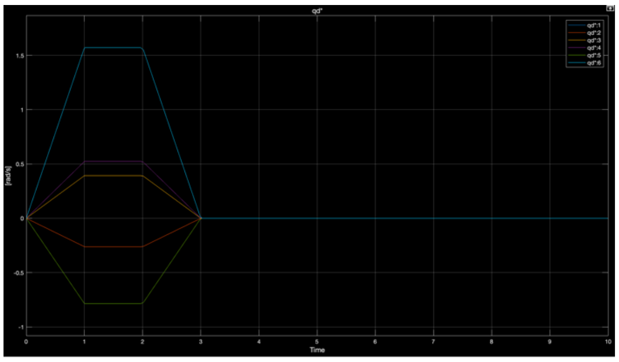


Image 37: Desired joints velocity of the fourth exercise

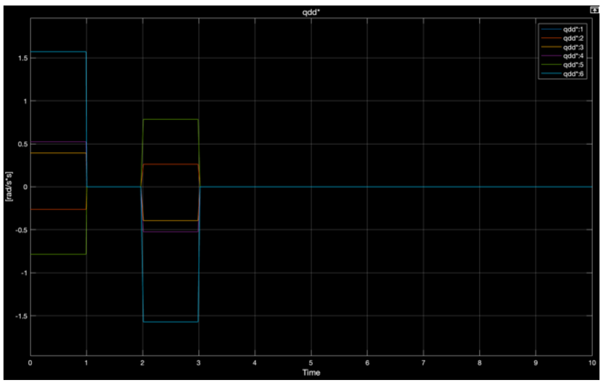


Image 38: Desired joints acceleration of the fourth exercise

No damping coefficient configuration, velocity, and acceleration profiles

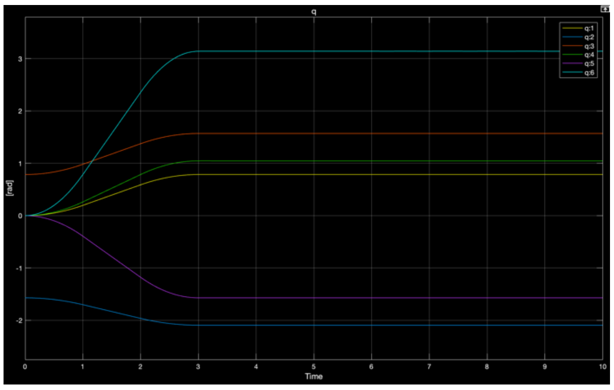


Image 39: configuration of the fourth exercise

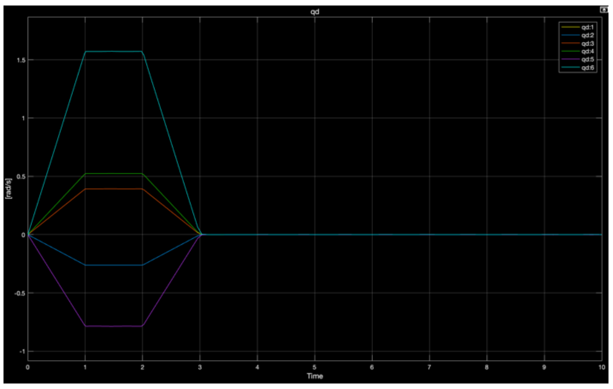


Image 40: joints velocity of the fourth exercise

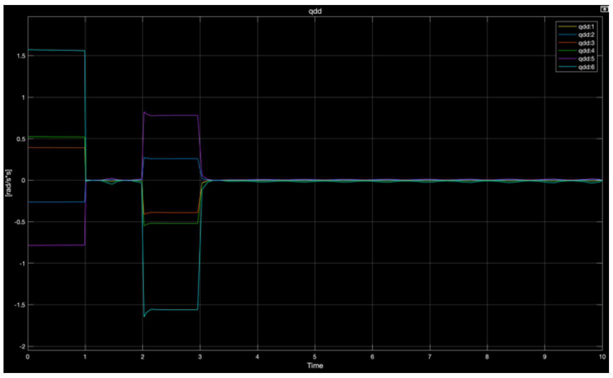


Image 41: joints acceleration output of the fourth exercise

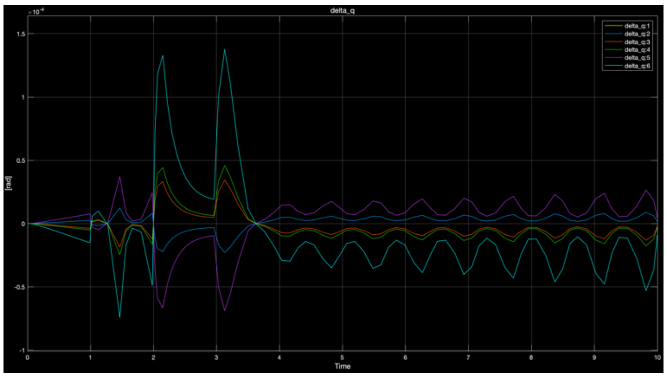


Image 42: joints position error of the fourth exercise

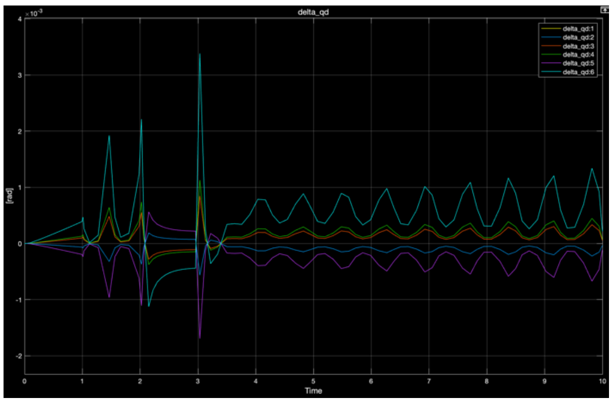


Image 43: joints velocity error of the fourth exercise

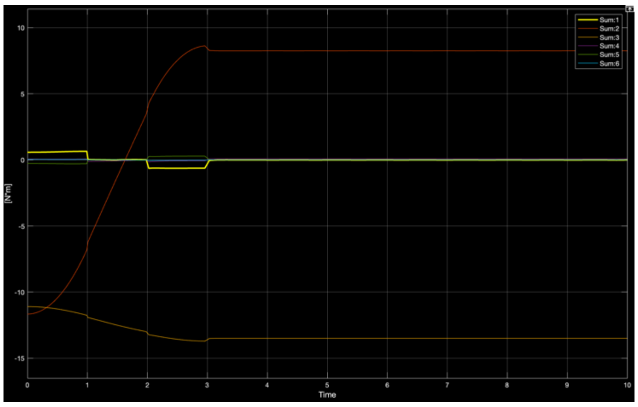


Image 44: joints torque of the fourth exercise

Comments:

The differences between the normal robot and the one without a dumping coefficient are visible in the position and velocity error graphs. The no dumping coefficient one holds some very small error values between the desired and actual values for both velocity and position. The lack of friction in the "ideal" robot joints allows the control signal to be followed much faster than in the original model. As far as the real model is concerned, the robot takes some time to reach the actual configuration. This type of behavior is also visible in the other graphs, where the no dumping robot's velocity and acceleration profiles are more similar to the desired ones than the original robot’s signals are. The original robot's transient behavior is much more visible than the ideal one, with significant differences in the acceleration and velocity graphs which clearly show the transient behavior of the actuation trying to follow the input signals.