Report of the Third Assignment

Report of the third assignment of Robot Dynamics and Control

Marco Macchia, mat. 4687564

0. Introduction

The third assignment focuses on the Dynamic Control of a Manipulator.

It was given a basic simulation model of a UR5 robot that I can control by giving torque signals to the joints. I am able to read joint position, velocity, and acceleration.

In order to complete all of the necessary exercises, I created a multi-blocks system on **Simulink**, developing a unique network system for each one.

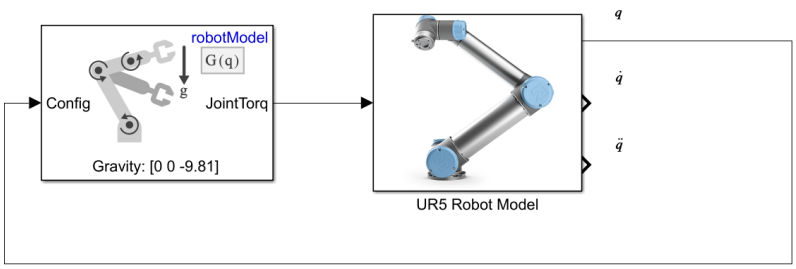
1. Exercise 1 - Gravity Compensation

Provide torque commands to the robot such that is doesn’t fall due to gravity but holds the initial configuration.

The key challenge is controlling the robot's torque in order to not fall due to gravity but keeo its starting position.

The Robotics System Toolbox provide a block called **Gravity Torque** that, given the current configuration as input, provides the required torque as output.

Here’s the block diagram used to solve this exercise:



Block diagram for Exercise 1

Since the system is just counteracting the effect of gravity, which is obiously constant, the joint configuration continues to be the same throughout time as expected. The same logic applies to torques because their sole purpose is to counterbalance gravity.

τ :

Figure 1.1

q :

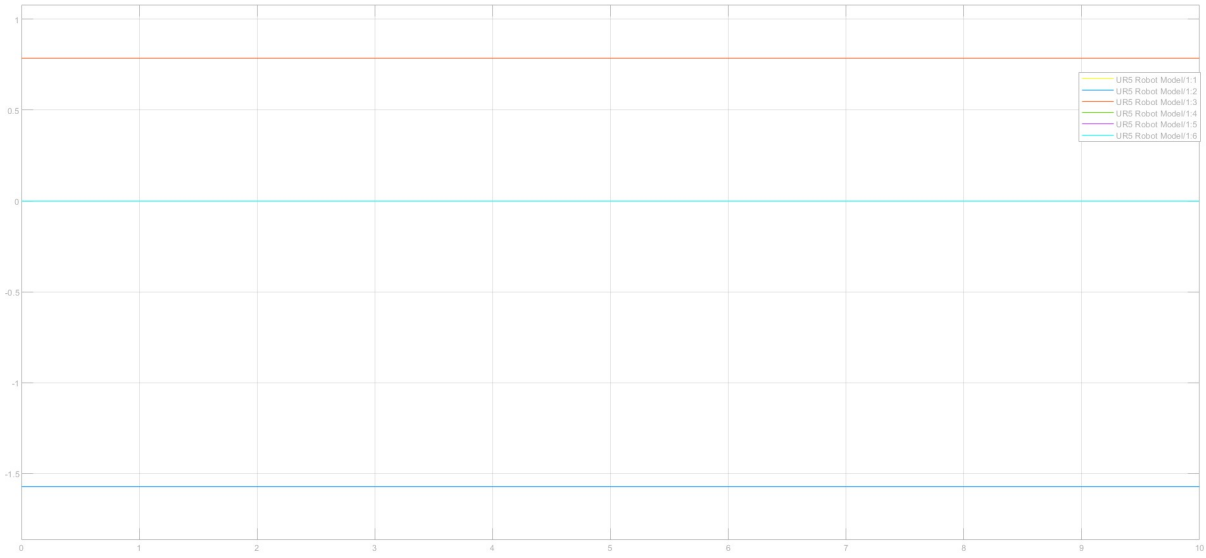


Figure 1.2

2. Exercise 2 – Linear Joint Control

Given a desired joint configuration $q^* = q_0 + \Delta q$ *where* q_0 *is the initial joint configuration and*

$$\Delta q = [\frac{\pi}{4}, -\frac{\pi}{6}, \frac{\pi}{4}, \frac{\pi}{3}, -\frac{\pi}{2}, \pi],$$

provide torque commands in order to reach q^**. What happens if you don't compensate gravity? Make a comparative analysis.*

The formula used to solve this exercise is the following:

$$\tau = -k_v \dot{q} - k_e \delta q + C(q),$$

where $\delta q = q - q^*$.

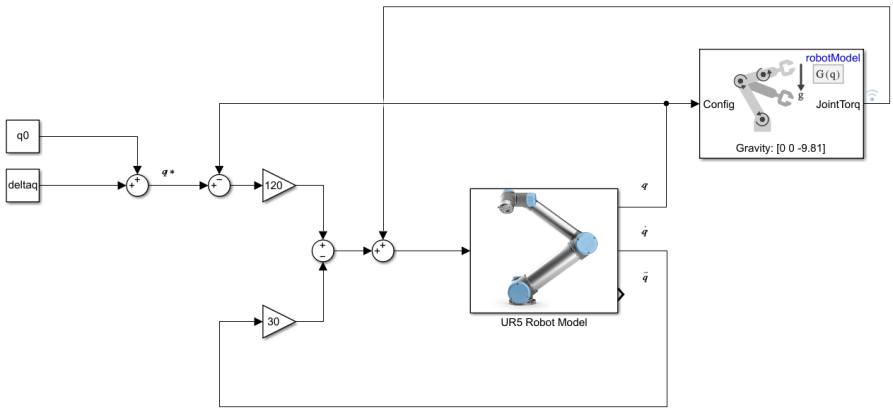
In my simulation I choose to use $k_v = 30$ and $k_e = 120$, which results in a fast movement towards the desired pose, without fluctuations.

The only thing to do here si to implement the formula in the simulink workspace. First I have to build q^*, and then I can use several adder nodes to compute the required torque τ.

If I don't compensate the gravity effect, the only change to made is to remove the gravity compensation block, and the adder node near the robot model input.

2.1. Gravity compensation

Here's the block diagram used to solve this exercise:



Block diagram for Exercise 2

Contrary to the previous exercise, the joints' configuration changes to achieve the desired configuration, as shown in the pictures below.

All of the parameters shown in the images gradually stabilize once the robot manipulator reaches the desired stance, as compared to how they initially changed in order to move the robot.

τ :

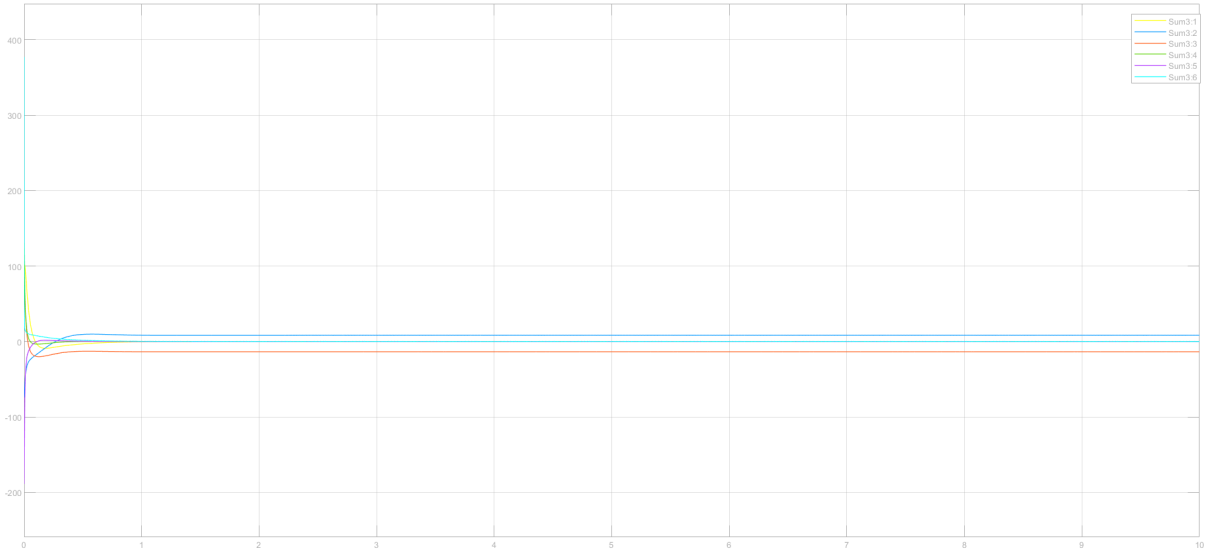


Figure 2.1

τ (zoom) :

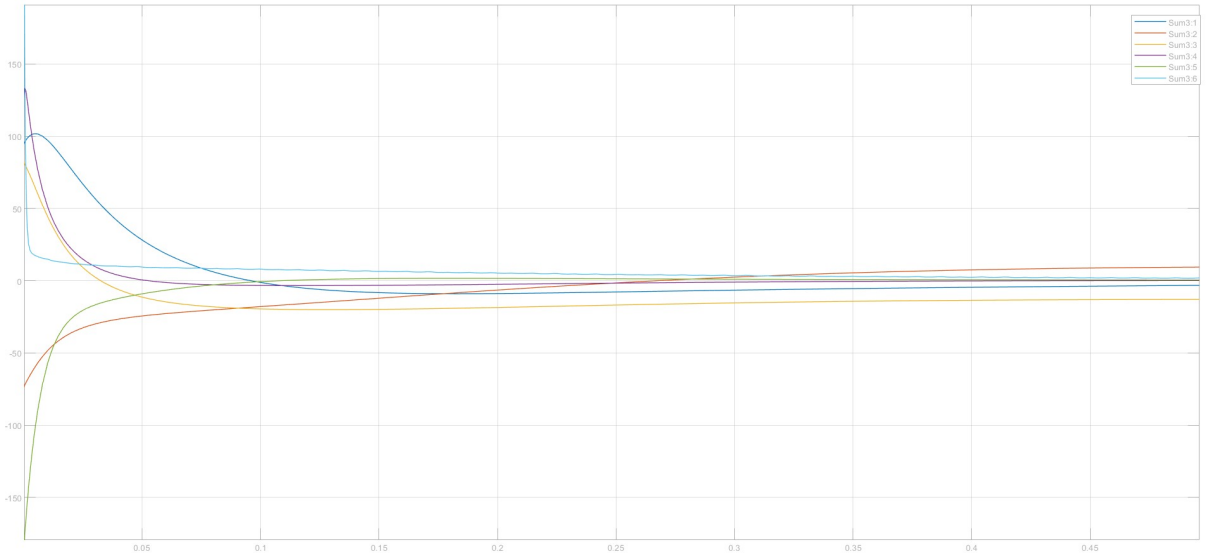


Figure 2.2

q :

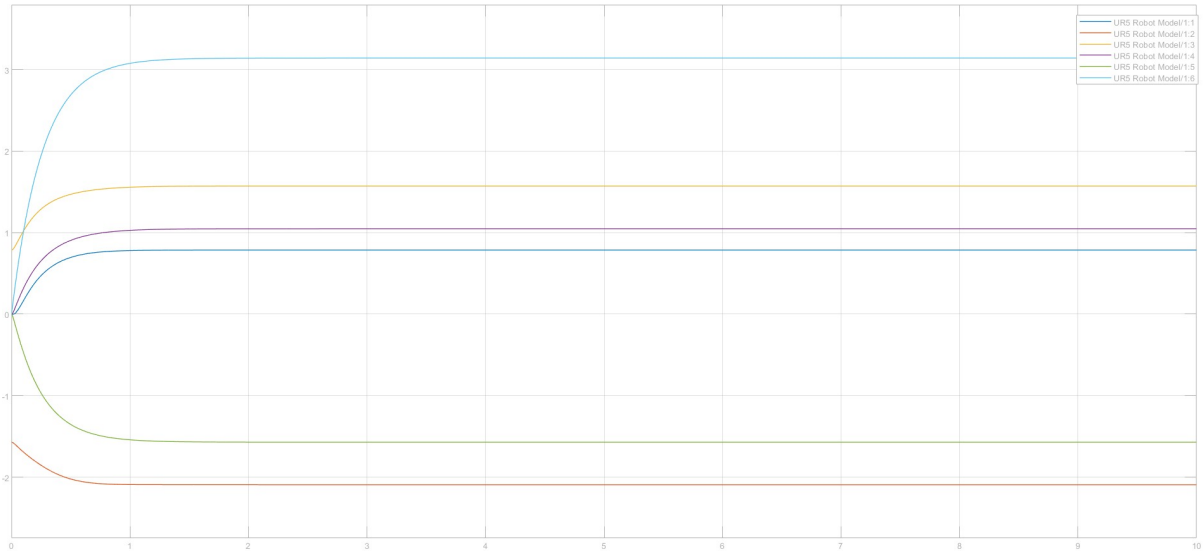


Figure 2.3

$\dot{q}$:

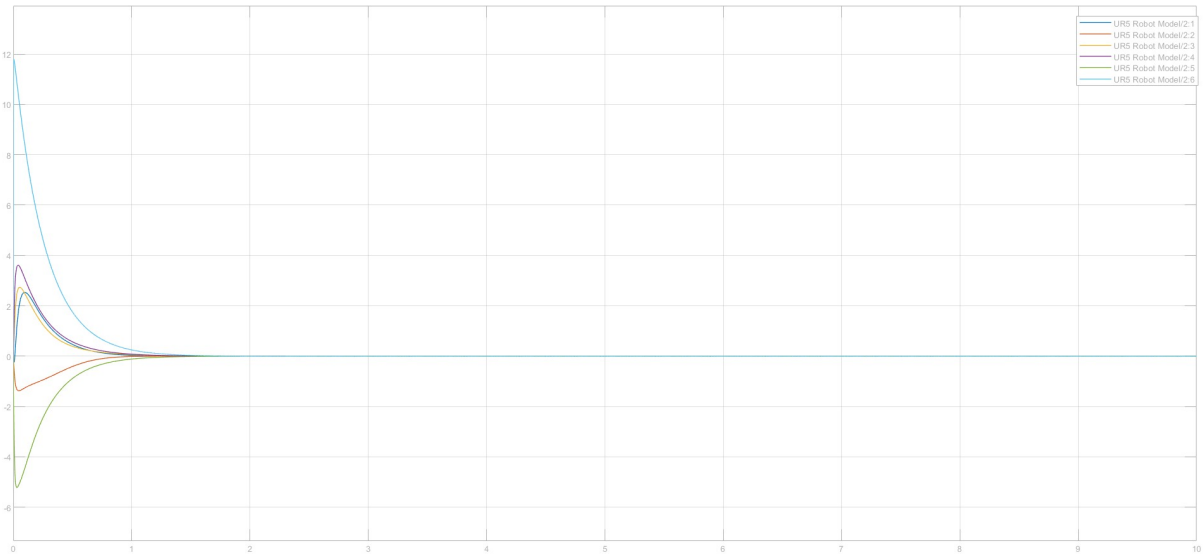


Figure 2.4

$\ddot{q}$ (zoom) :

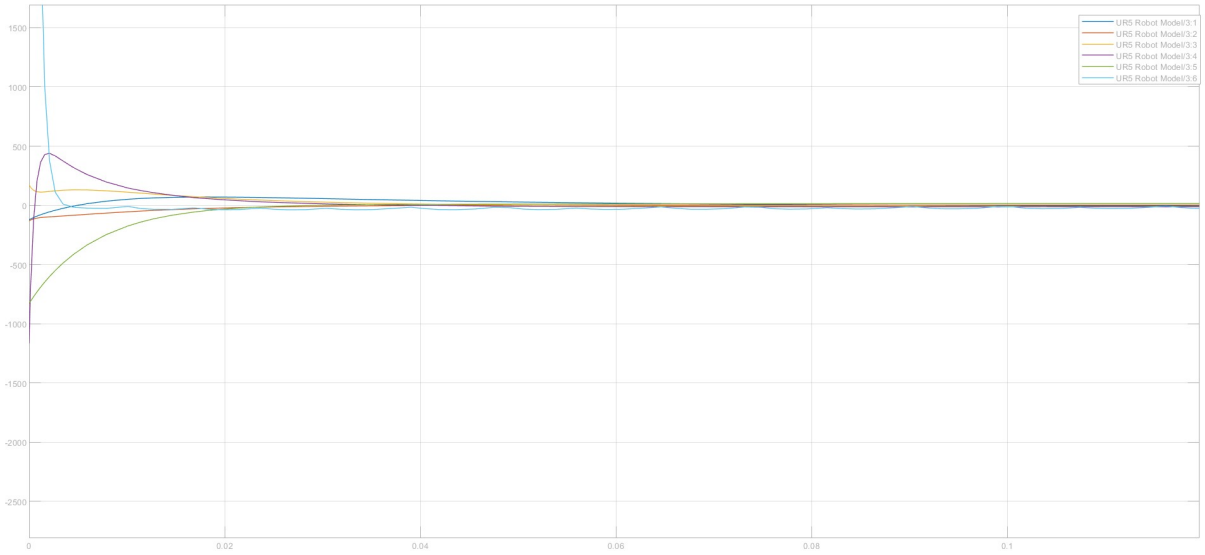


Figure 2.5

Control errors:

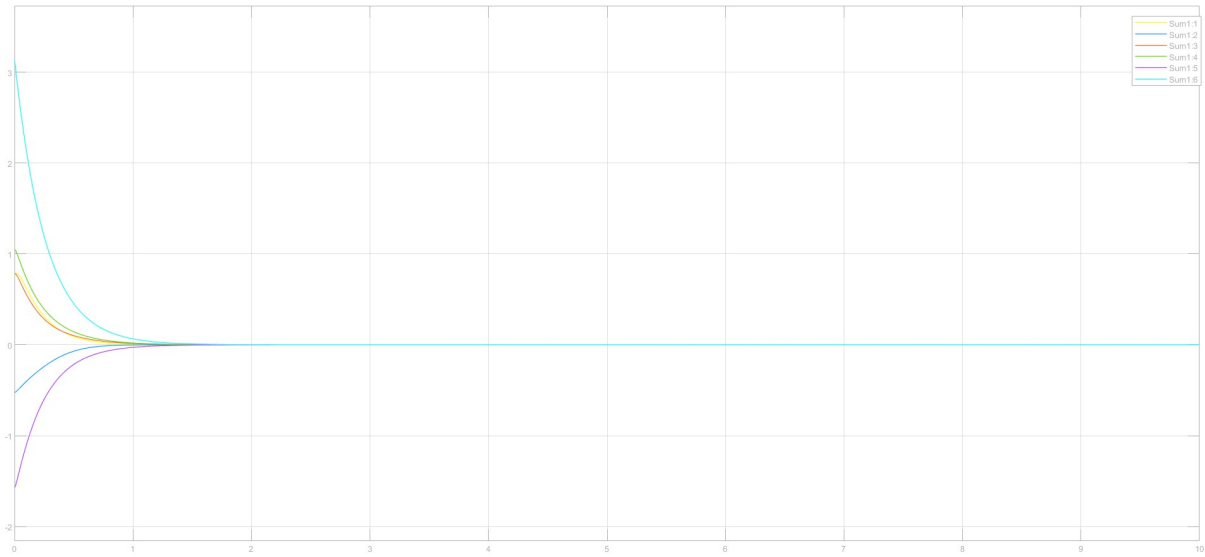
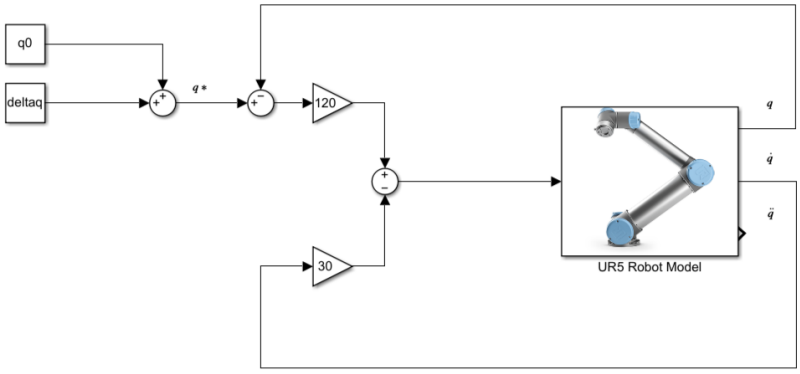


Figure 2.6

2.2. Without gravity compensation

Here's the block diagram used to solve this exercise:



Block diagram for Exercise 2

Here I can not identify many differences, because with the values of k_v, k_e that I used the robot is able to stand still even without gravity compensation. I also tried with lower values, in particular $k_v = k_e = 1$ and the robot immediately falls down due to gravity.

Also from the plots shown below it can be highlighted that there aren't any important differences from the exercise 2.1

τ :

Figure 2.7

τ (zoom) :

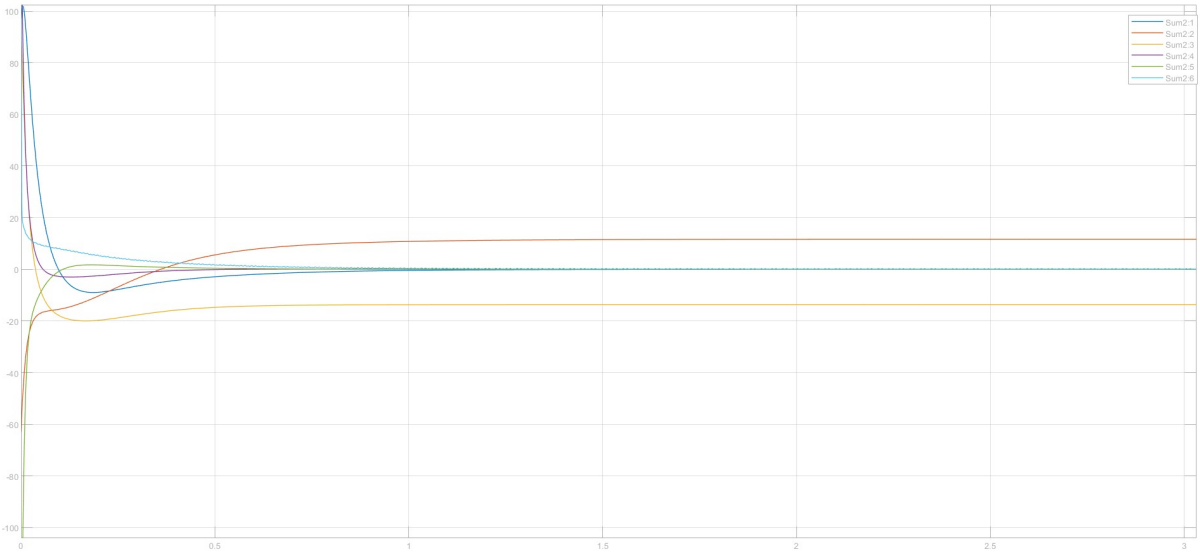


Figure 2.8

q :

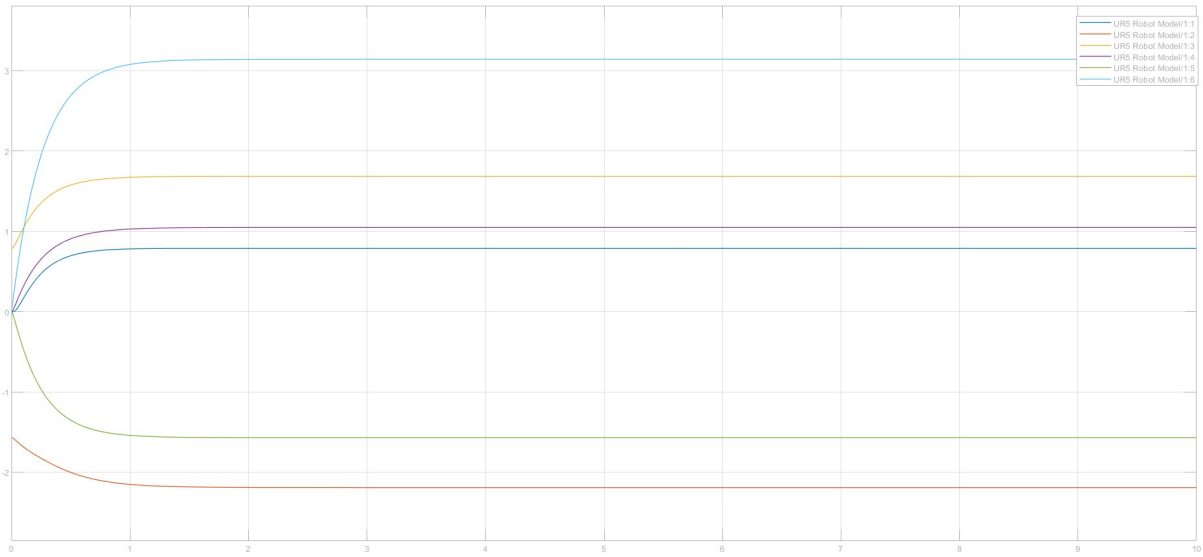


Figure 2.9

$\dot{q}$:

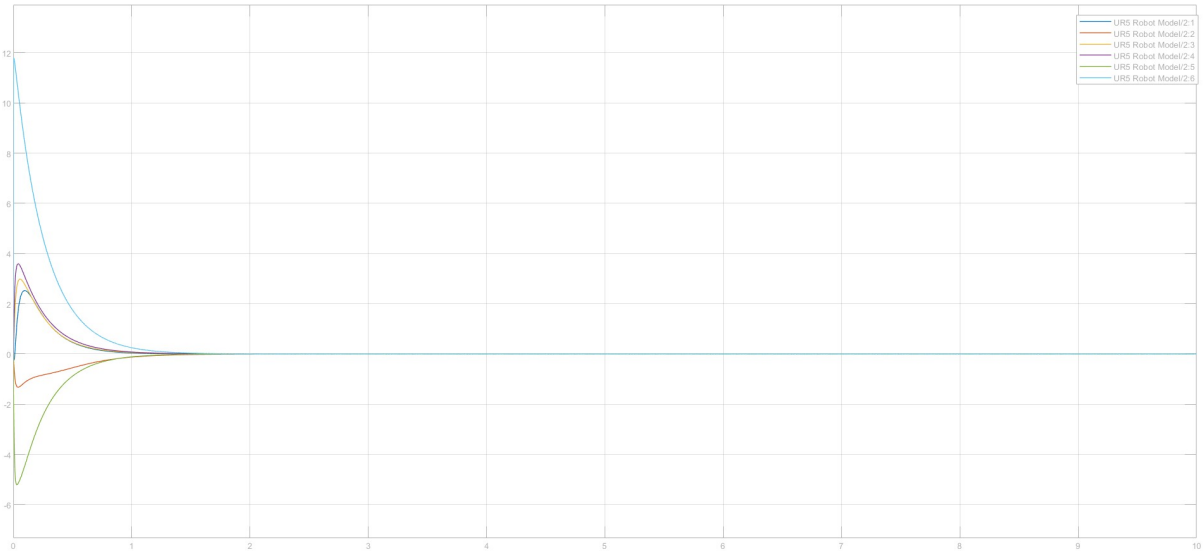


Figure 2.10

$\ddot{q}$ **(zoom) :**

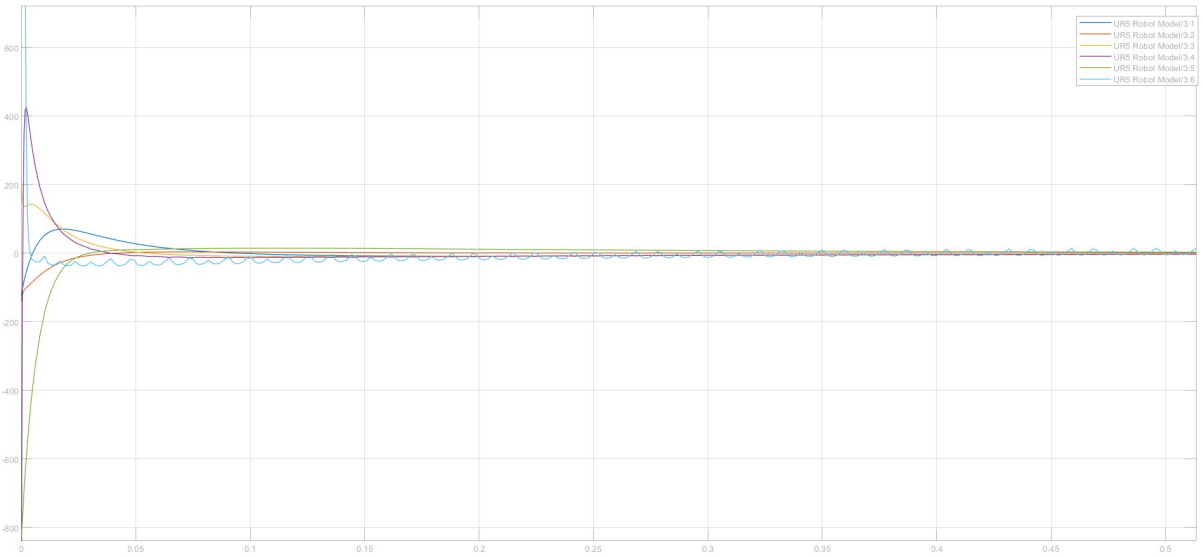


Figure 2.11

Control errors:

Figure 2.12

3. Exercise 3

Given a desired cartesian configuration $x^* = x_0 + x$, where x_0 is the starting cartesian pose of the end effector ('ee link') and

$$x = [0.2, -0.08, -0.15, \frac{\pi}{4}, -\frac{\pi}{4}, \frac{\pi}{2}],$$

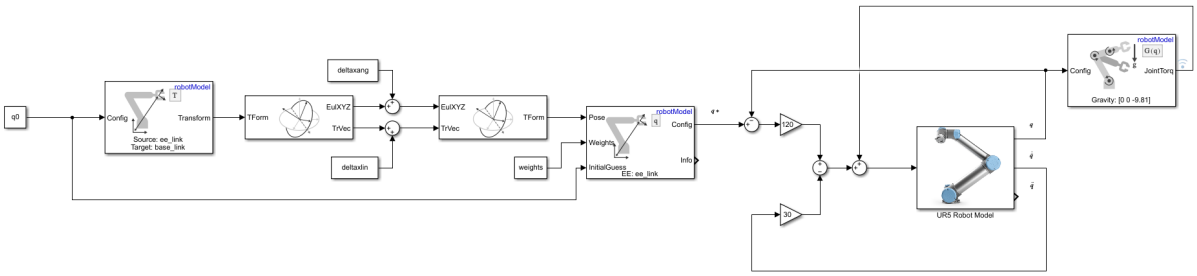
provide torque commands in order to reach x^*.

As you can see the right part of the diagram is the same as for the previous exercise. What changes here is how I can compute q^*. The exercise gives me x_0 and the x vector, that added together represents the desired pose of the end effector expressed in the cartesian space.

First of all I have to compute the initial pose of the end effector in the cartesian space, that gives me the initial Euler angles and the initial translation vector. Then I can add x_0 and x to get the desired Euler angles and the desired translation vector. Now I can compute the desired pose in the joint space.

After that, using the **Inverse Kinematics** block I can finally obtain q^*, giving as initial guess the initial configuration q_0 and an array of weights defined as $weights = [0.1,\ 0.1,\ 0.1,\ 0.1,\ 0.1,\ 0.1]$.

Here’s the final block diagram:



Block diagram for Exercise 3

As in exercise 2, the robot must achieve the target position in this exercise while keeping in mind the end effector's cartesian pose, which is given. In the end the exercise is the same of the previous one.

As shown in figure 3.1, the manipulator moves initially from its starting position while adjusting the joint torques, which then stabilize at specific values because the robot must be in the final position.

The same reasoning can be applied to the other parameters, q, $\dot{q}$, $\ddot{q}$ and control errors, since at first the robot leaves its still position, making movements till it stabilize again in the new target position.

τ :

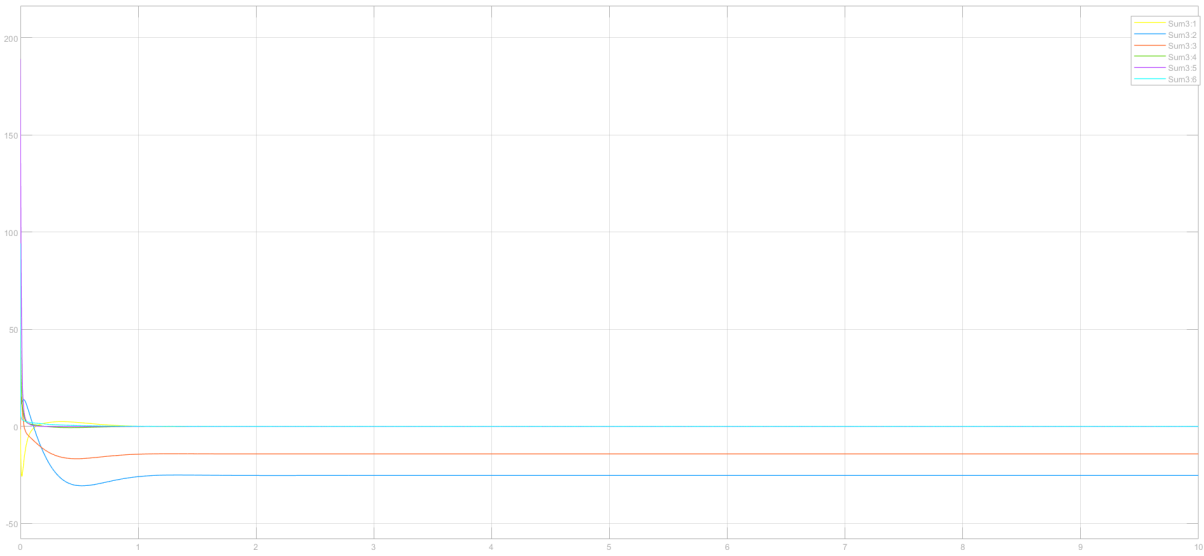


Figure 3.1

τ (zoom) :

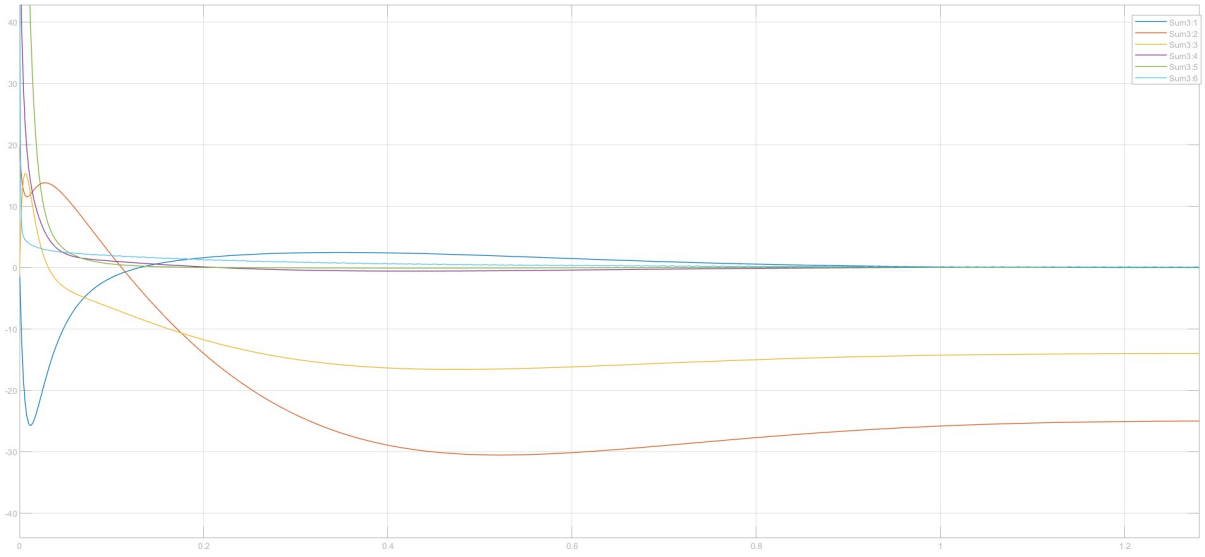


Figure 3.2

***q*:**

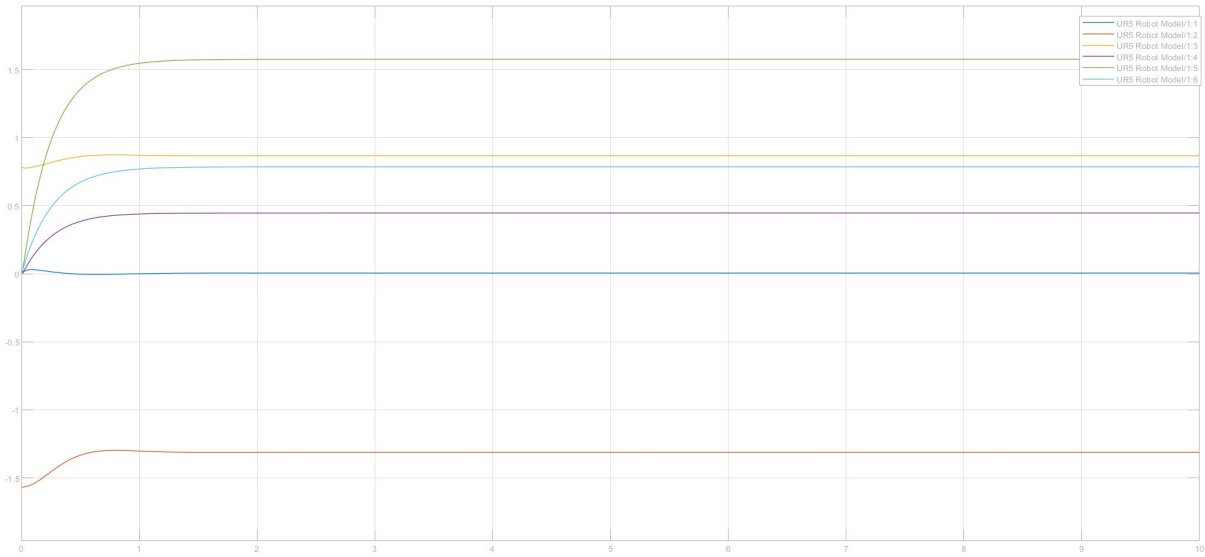


Figure 3.3

$\dot{q}$:

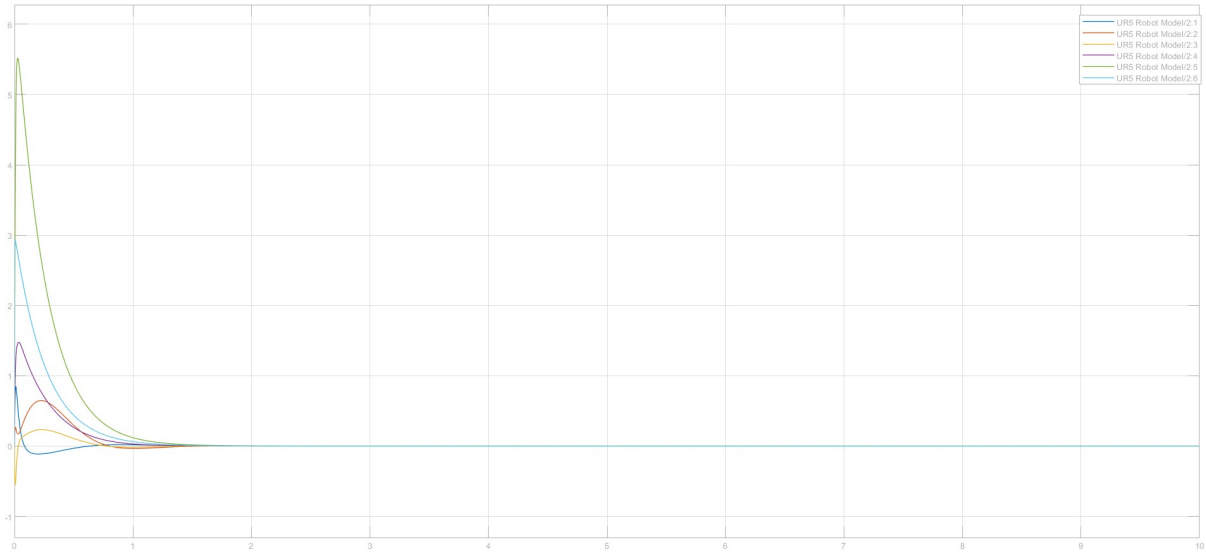


Figure 3.4

$\ddot{q}$ **(zoom) :**

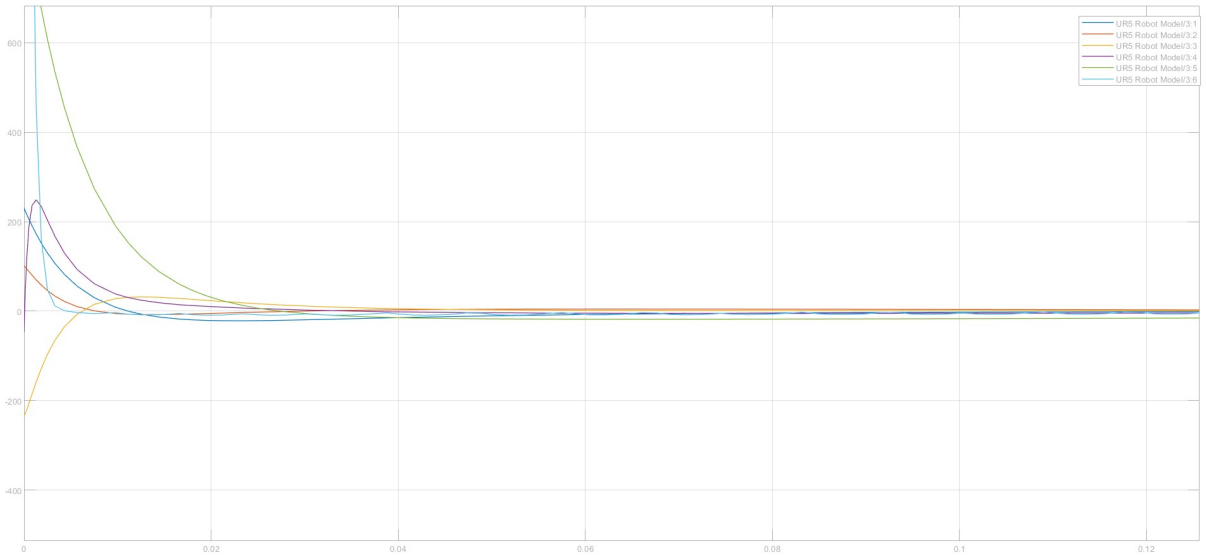


Figure 3.5

Control errors:

Figure 3.6

4. Exercise 4

In industrial robot control systems even a simple regulation from a setpoint to another (like in Ex.2 and Ex.3) is done via joint trajectory tracking. Consider again q from Ex.2 and generate joint-space trajectories in order to reach it from q0.*

In order to track the required joint trajectories, use feedback linearization to construct a calculated torque control.

The formula used to solve this exercise is the following:

$$\tau = A(q)[\ddot{q}^* - k_v(\dot{q} - \dot{q}^*) - k_p(q - q^*)] + B(q, \dot{q}) + C(q),$$

where $A(q)$ generalized inertial matrix, $B(q, \dot{q})$ is the velocity product torque $C(q)$ is the gravity compensation torque.

In order to generate a joint trajectory, in this particular simulation we can use the **Trapezoidal Velocity Profile Trajectory** block, which requires a vector of waypoints as input (both internal or external) and the time required to perform the movement. This particular block provides as outputs the desired $q, \dot{q}, \ddot{q}$.

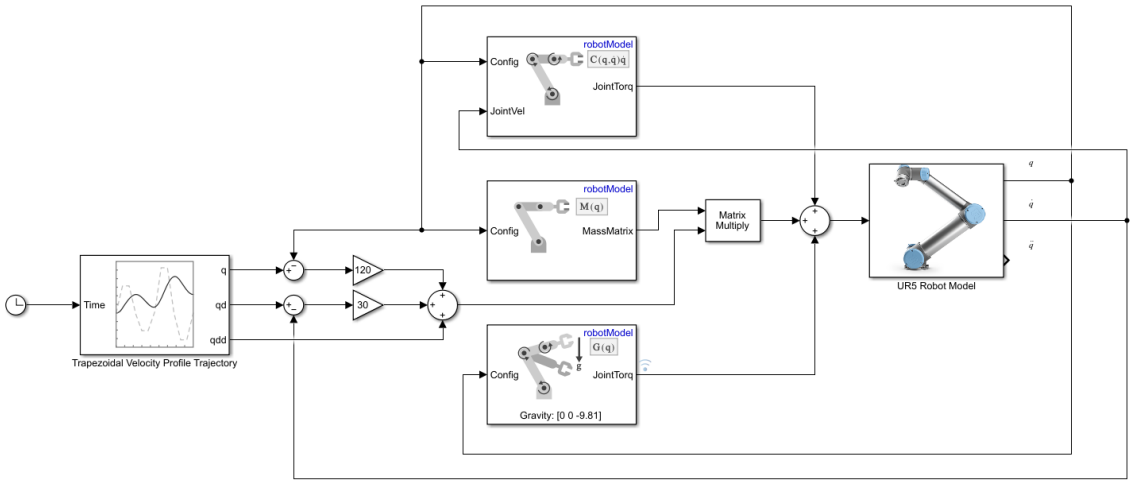
The next step requires me to compute the first part of the formula above, and to do that I can first find $k_v(\dot{q} - \dot{q}^*)$ and $k_p(q - q^*)$, and add this two vectors with $\ddot{q}^*$ to get the vector that should be pre-multiplied by $A(q)$.

Then using the **Velocity Product Torque** block I can compute $B(q, \dot{q})$ (giving q and $\dot{q}$ as inputs), and as before using the **Gravity Torque** block I can obtain $C(q)$.

Finally, by a simple addition I can finally found the required τ.

In this particular solution I choose to use $k_p = 120$ and $k_v = 30$.

Here's the block diagram used for this exercise:



Block diagram for Exercise 4

4.1. With joint friction

In order to track the intended joint trajectories, feedback linearization is used to construct a calculated torque control.

For this specific scenario, I continue to use the active damping coefficient configuration, which provides the ability to simulate joint friction (In the next scenario, 4.2, I'll set it to 0 in order to remove the simulate friction).

The plots of $\dot{q}$ and $\ddot{q}$ doesn't follows regular shapes, although a more regular geometry may be seen here. The values of $\dot{q}$ have a trapezoidal form (isosceles-like trapezoid), and the components of $\ddot{q}$ are made up of two rectangle-like shapes of differnet heights.

Here are the figures produced by the scenario with joint friction:

τ :

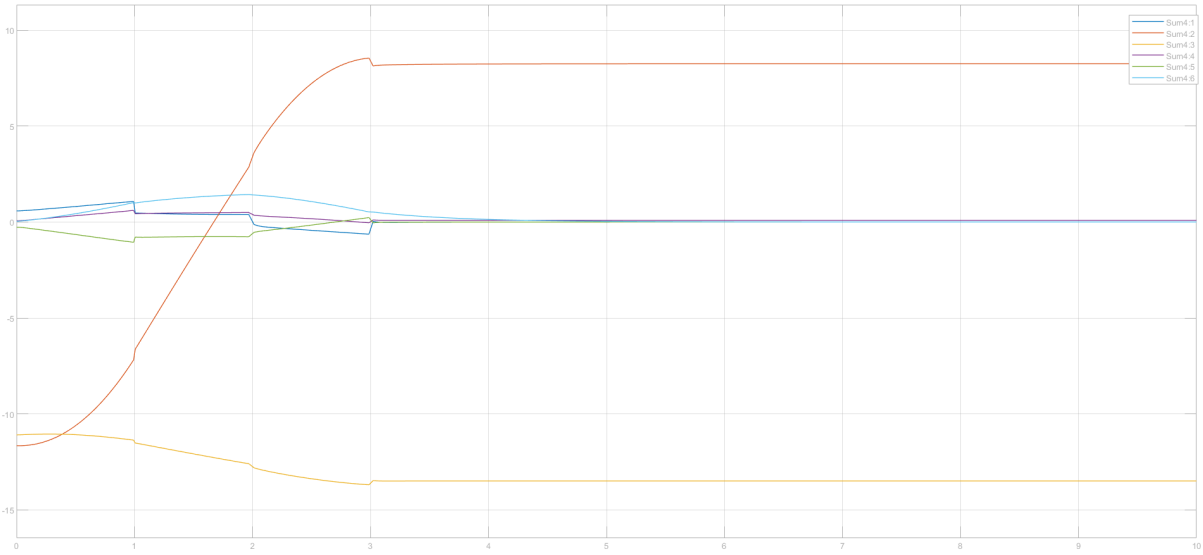


Figure 4.1

q :

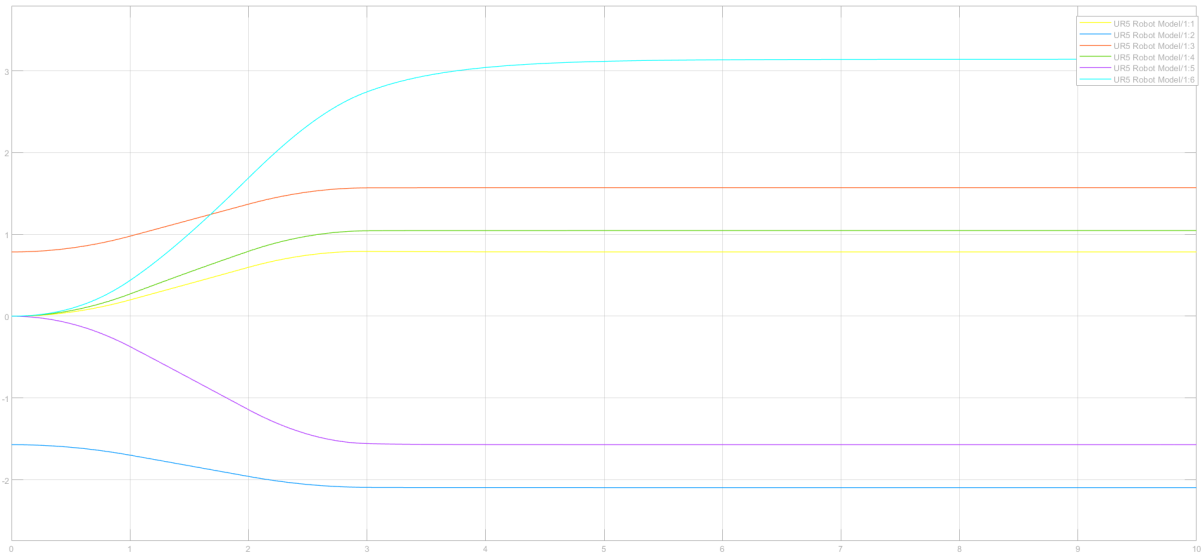


Figure 4.2

$\dot{q}$:

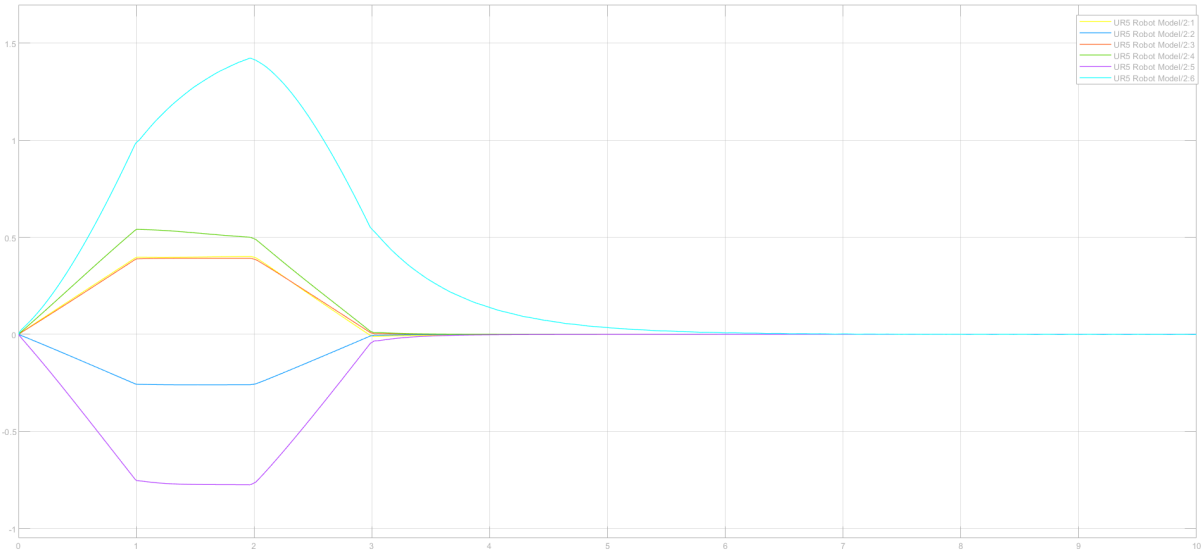


Figure 4.3

$\ddot{q}$:

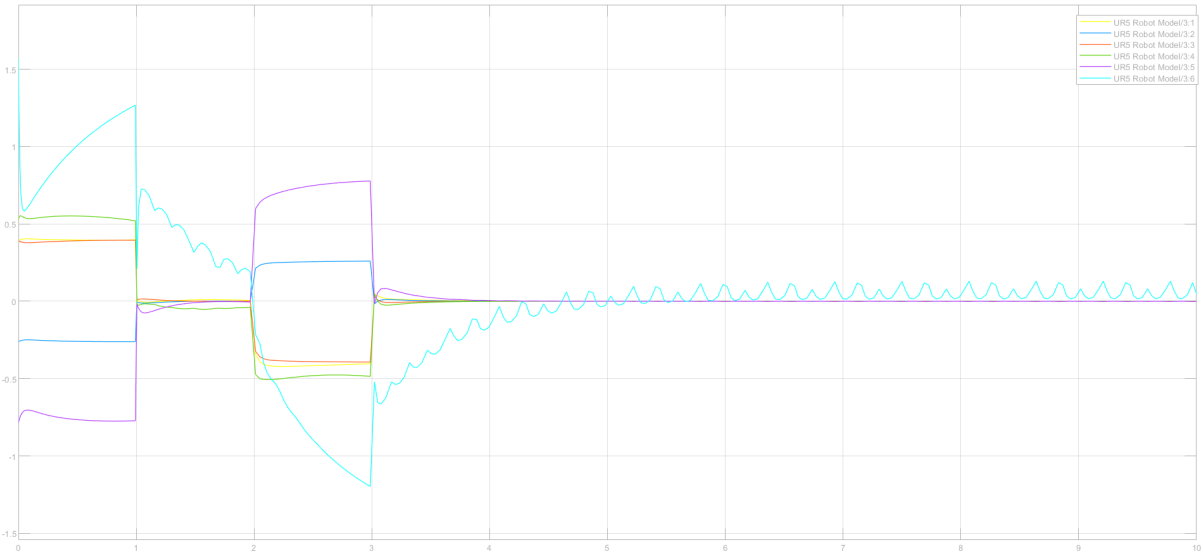


Figure 4.4

Control errors:

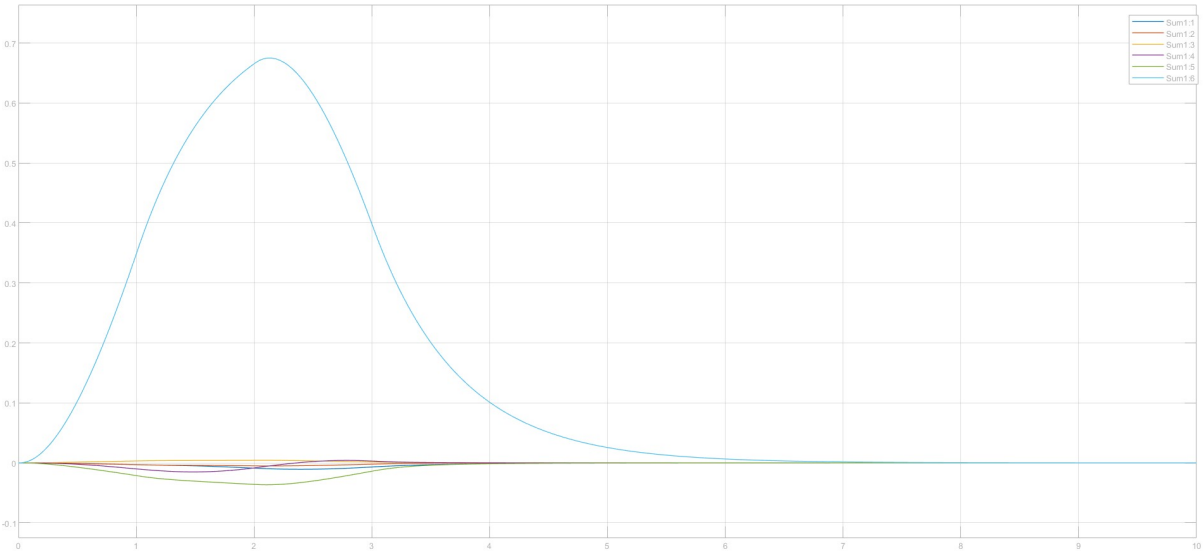


Figure 4.5

4.2. Without joint friction

Here I examined the joint objects in the robot model and altered the damping coefficient under the label *Internal Mechanics*, which is used to simulate joint friction. I removed it from each joint as required and compared the performance of the calculated torque controller.

It can be immediately highlighted the difference between the following plots compared to the previous ones: while in exercise 4.1 the plots of $\dot{q}$ and $\ddot{q}$ doesn't have a regular shape, here a more regular geometry is clearly visible. The values of $\dot{q}$ have a clear trapezoidal shape (isosceles trapezoid, to be more precise), while the components of $\ddot{q}$ are made of two rectangles with different heights.

q and τ are very similar to the previous ones, instead.

Here there are the plots generated using the configuration without joint friction:

τ :

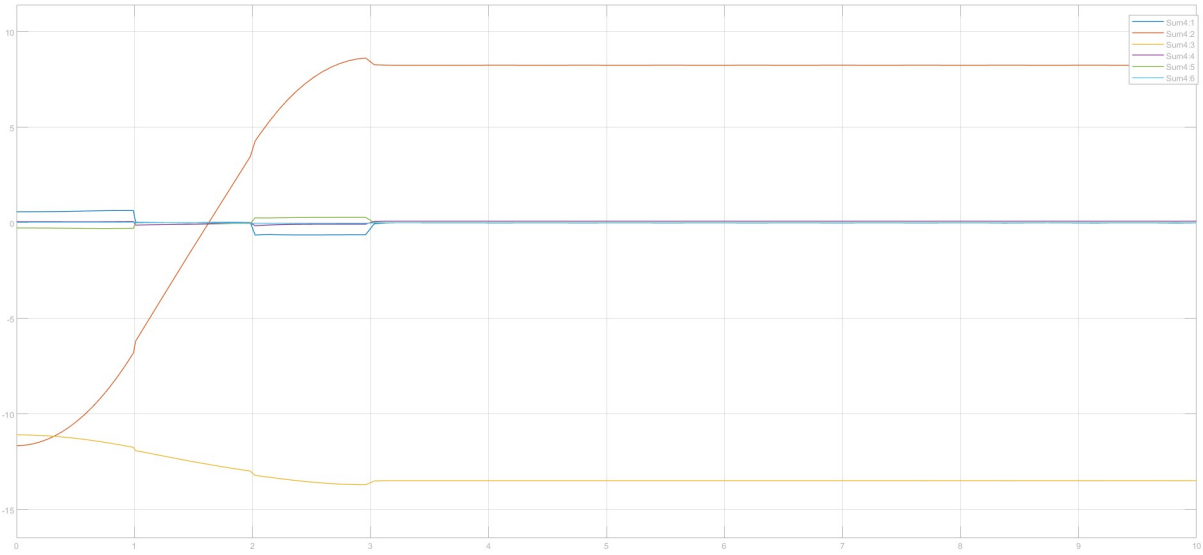


Figure 4.6

q :

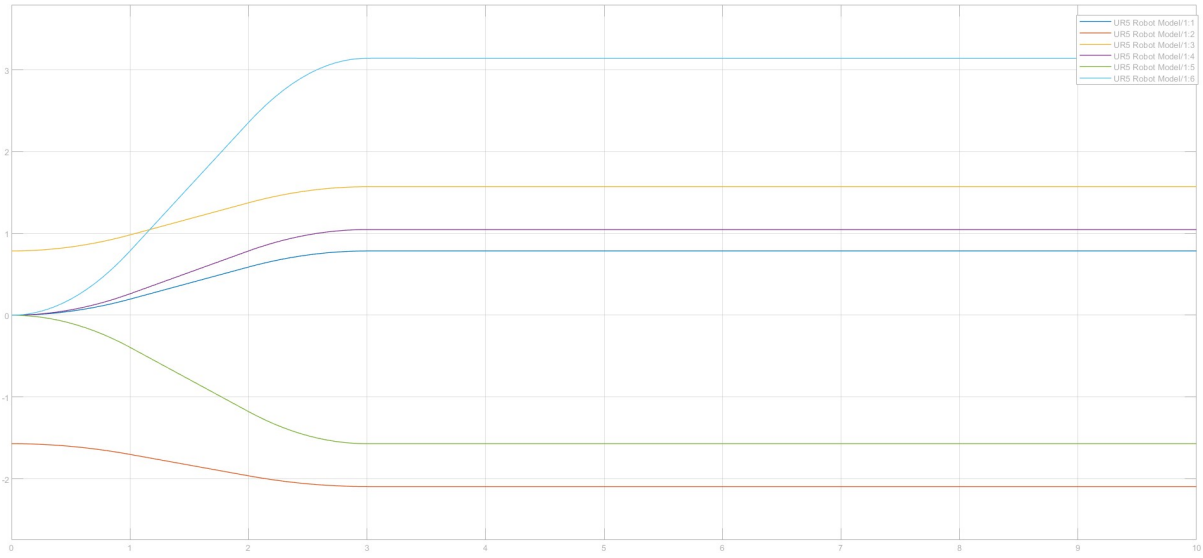


Figure 4.7

$\dot{q}$:

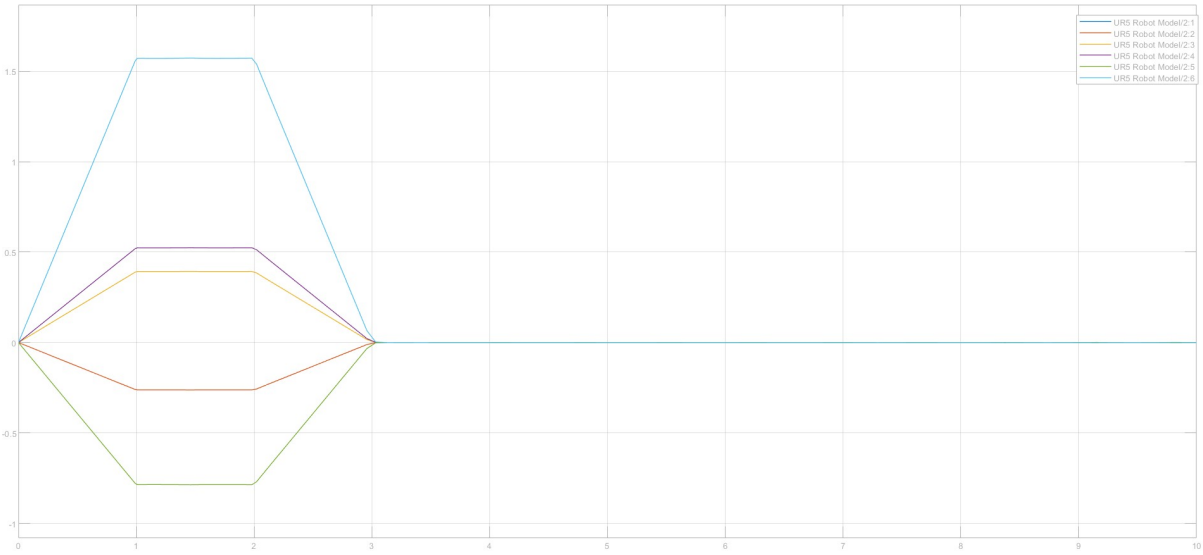


Figure 4.8

$\ddot{q}$:

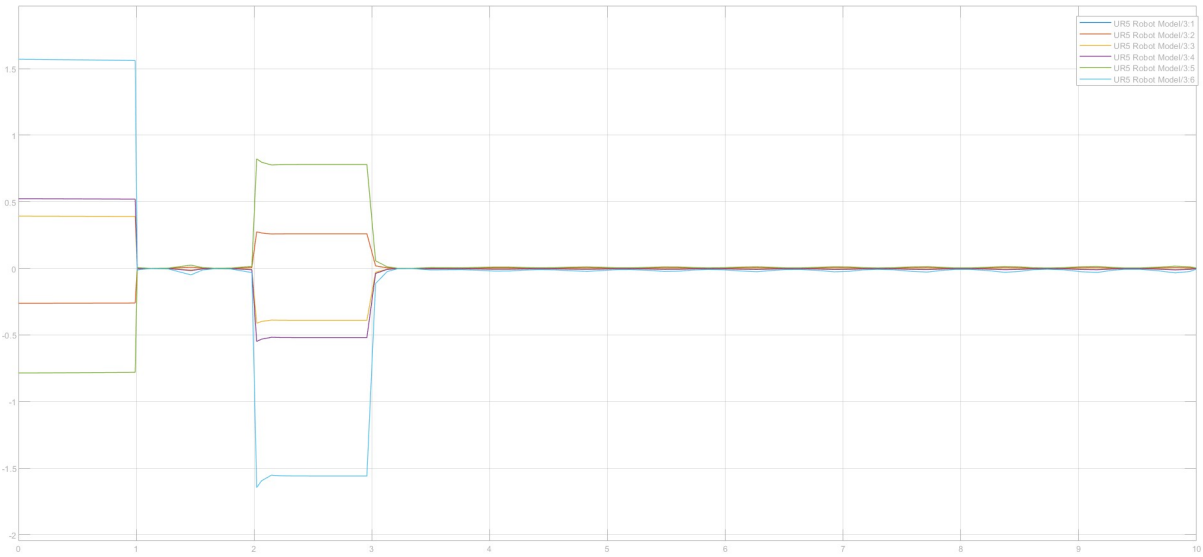


Figure 4.9

Control errors:

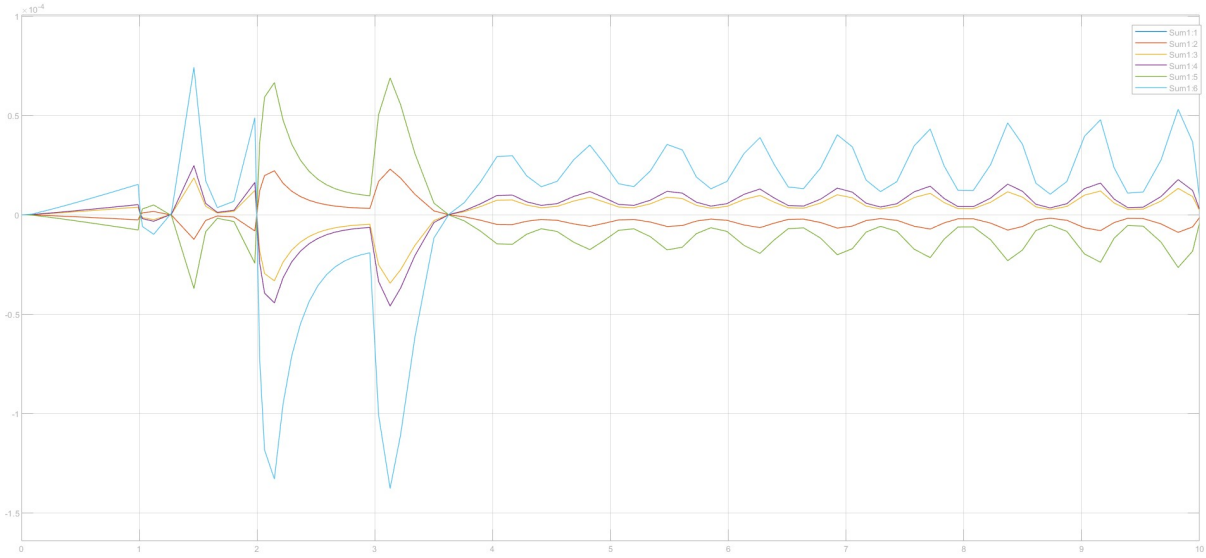


Figure 4.10

5. Conclusion

In this assignment we were asked to use for the first time the Simulink software, which is part of the Matlab Suite. It wasn't hard to learn how to use this software, also because it is very intuitive in my opinion.

The tasks were quite simple to complete using the blocks made available by the Robotics System Toolbox. The only challenging aspect were find the correct block that could provide me the correct data, and also it wasn't easy to always understand what were the required inputs of the blocks.

I am happy with the work done, because I had the opportunity of practically use what I have learned during the lessons. It was also very interesting to visualize the manipulator behaviour, and most of all it gave me the possibility of immediately understand if the implementation that I sketched in Simulink was the one that correctly solves the problem.