Università di Genova

Course:

Robot Dynamics & Control

Subject:

Assignment 3:

Dynamic Robot Control

Developer:

Mohammad Reza Haji Hosseini

Register number:

S4394567

E-mail:

4394567@studenti.unige.it

Supervisor:

Giorgio Canata

Years:

2021-22

README

The purpose of the assignment consists of controlling a UR5 manipulator, via block diagram in simulink.

To start the simulink you have to Run the MATLAB script *UR5_manipolator.m*, but before running the script you have to add to path the *visual* file in path of the MATLAB, in the same folder where there are other scripts. The *visual* folder contains mesh visualization of each joint.

Now you can Run the script. After starting it asks you to enter the gravity value and number of the desired exercise.

Information on manipulator

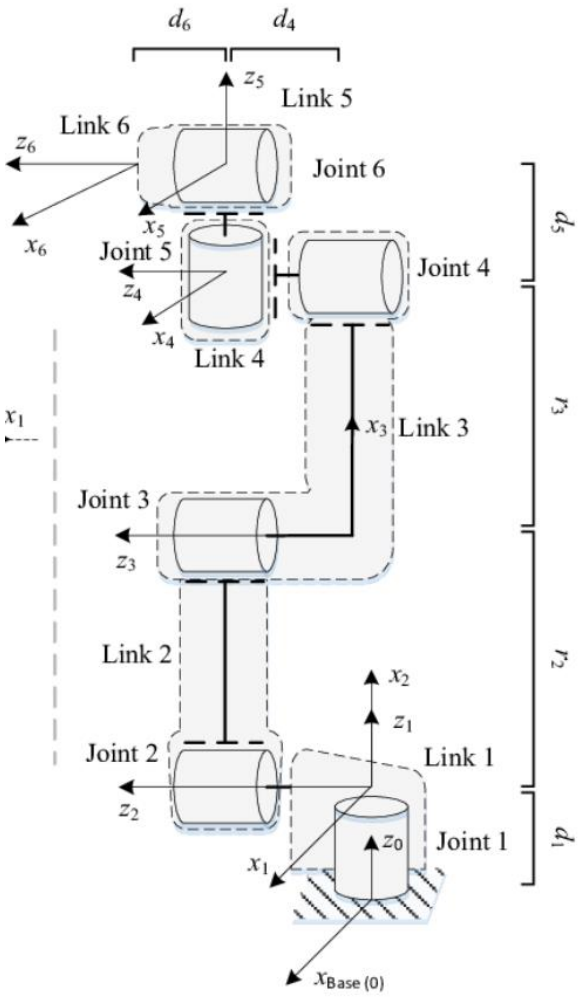


Figure 1. Robot configuration

0.Base:

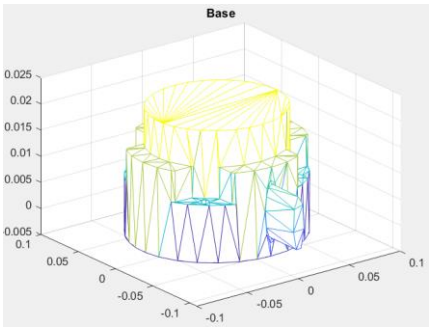
Type : fixed

Name : base_link_base_fixed_joint

Mass : 0 kg

CoM : [0,0,0]

Inertia : zeros(6,1)



1.Shoulder:

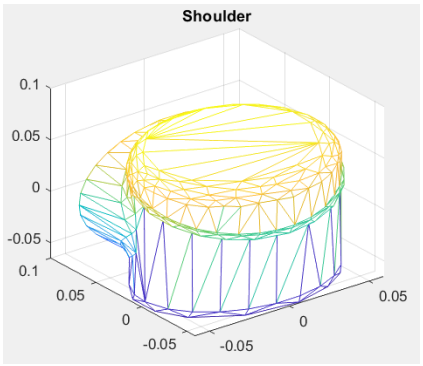
Type : revolute

Name : shoulder_pan_joint

Mass : 3,7 kg

CoM : [0,0,0]

Inertia : [0.01,0.01,0.006,0,0,0]



2.Upper arm:

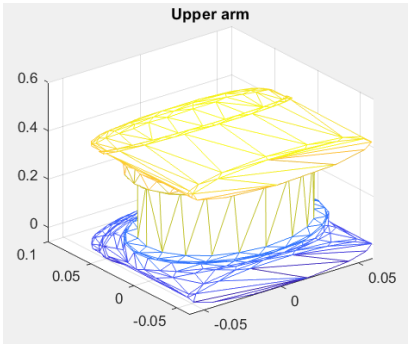
Type : revolute

Name : shoulder_lift_joint

Mass : 8,39 kg

CoM : [0,0,0,28]

Inertia : [0.88,0.88,0.015,0,0,0]



3.Forearm:

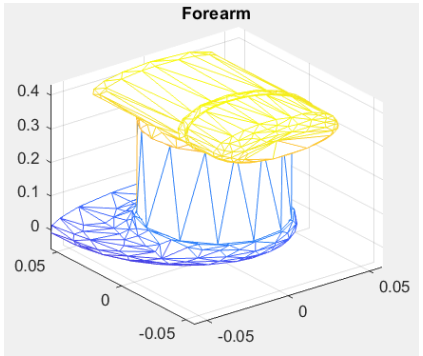
Type : revolute

Name : elbow_joint

Mass : 2,27 kg

CoM : [0,0,0.25]

Inertia : [0.19,0.19,0.0041,0,0,0]



4.Wrist_1:

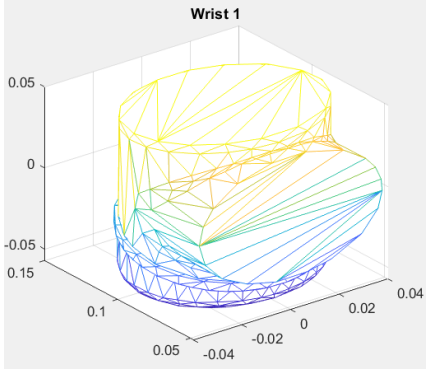
Type : revolute

Name : wrist_1_joint

Mass : 1,2 kg

CoM : [0,0,0]

Inertia : [0.11,0.11,0.21,0,0,0]



5.Wrist_2:

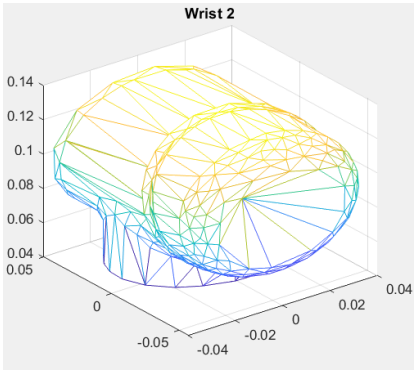
Type : revolute

Name : wrist_2_joint

Mass : 1,2 kg

CoM : [0,0,0]

Inertia : [0.11,0.11,0.21,0,0,0]



6.Wrist_3:

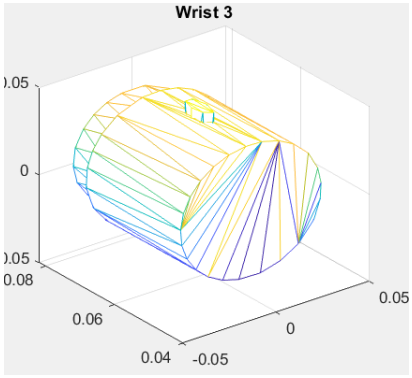
Type : revolute

Name : wrist_3_joint

Mass : 0,18 kg

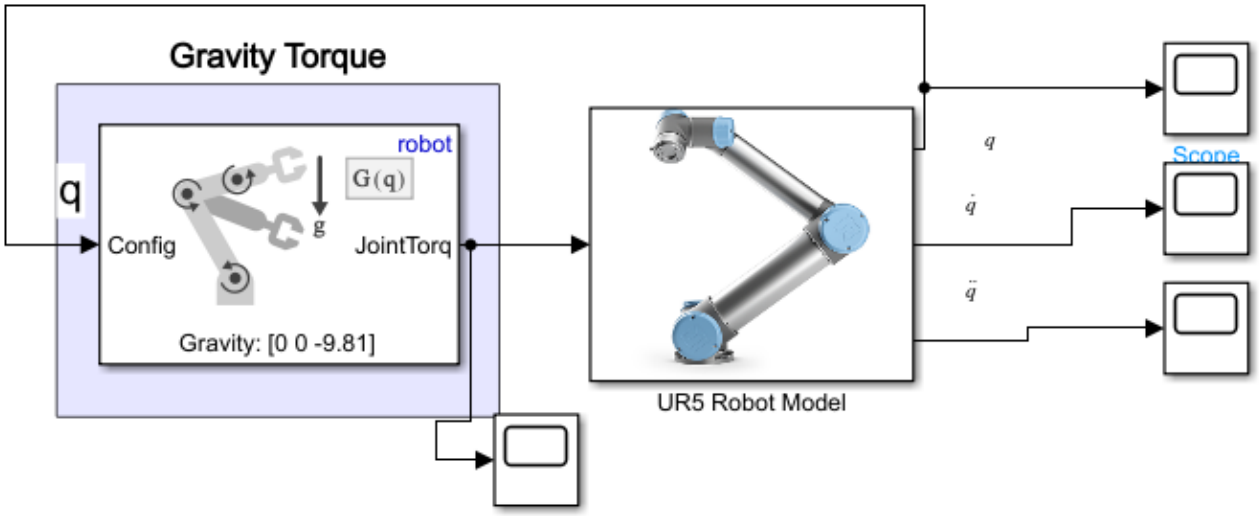
CoM : [0,0,0]

Inertia : [0.017,0.017,0.033,0,0,0]



Exercise 1 – Gravity Compensation

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



I need to define this formula in Simulink to avoid falling manipulator:

$$C(q) = \sum_{i}^{n} {J_{c_{i/0}}}^{T} \begin{bmatrix} 0 \\ -m_i g \end{bmatrix}$$

To make following formula I used.

`gravTorq = gravityTorque(robot)` computes the joint torques required to hold the robot at its home configuration.

The Gravity Torque block returns the joint torques required to hold the robot at a given configuration with the current Gravity setting on the **Rigid body tree** robot model.

Attention the figures portrayed, they were obtained through *Scope block* in simulink.

Scope: display input signals with respect to simulation time.

LEGEND information:

In the figure each number in the legend refers to a joint.

1 → Shoulder

2 → Upper arm

3 → Forearm

4 → Wrist_1

5 → Wrist_2

6 → Wrist_3

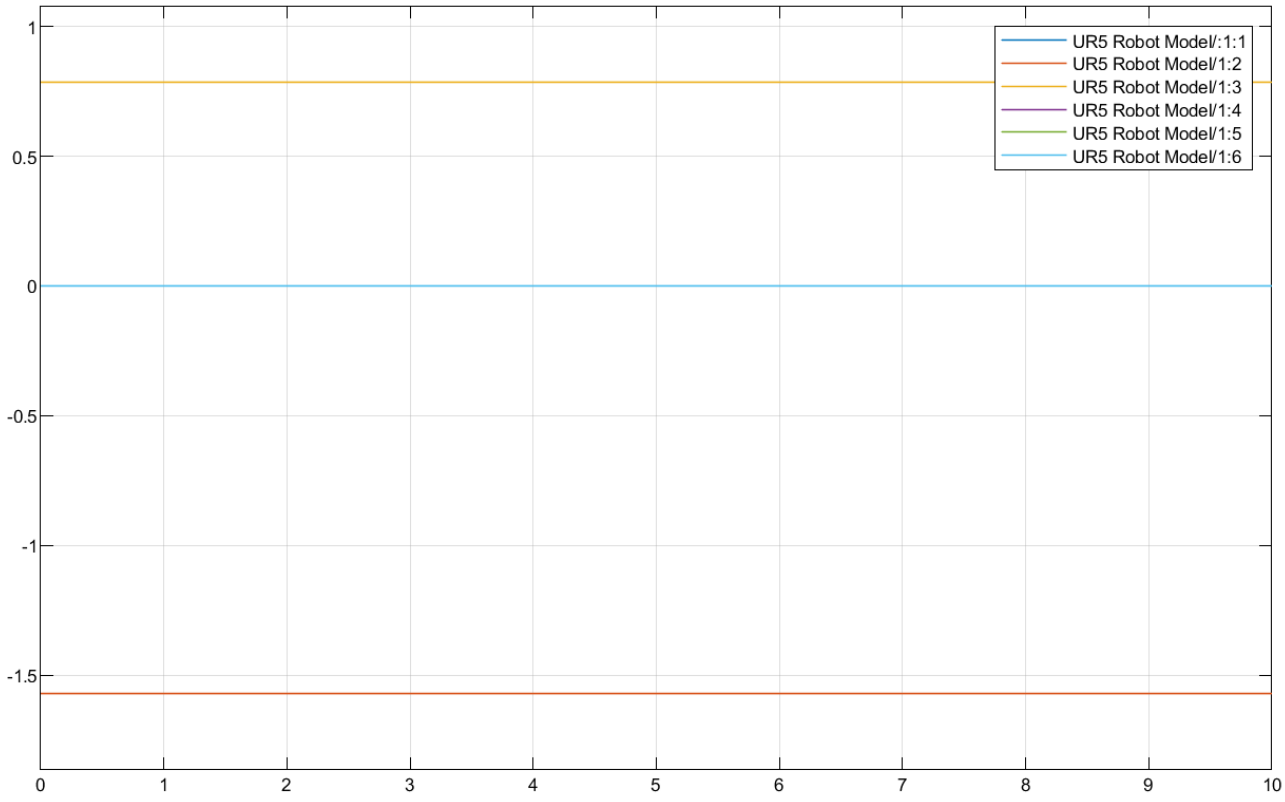


Figure 2. Joint position q [rad]

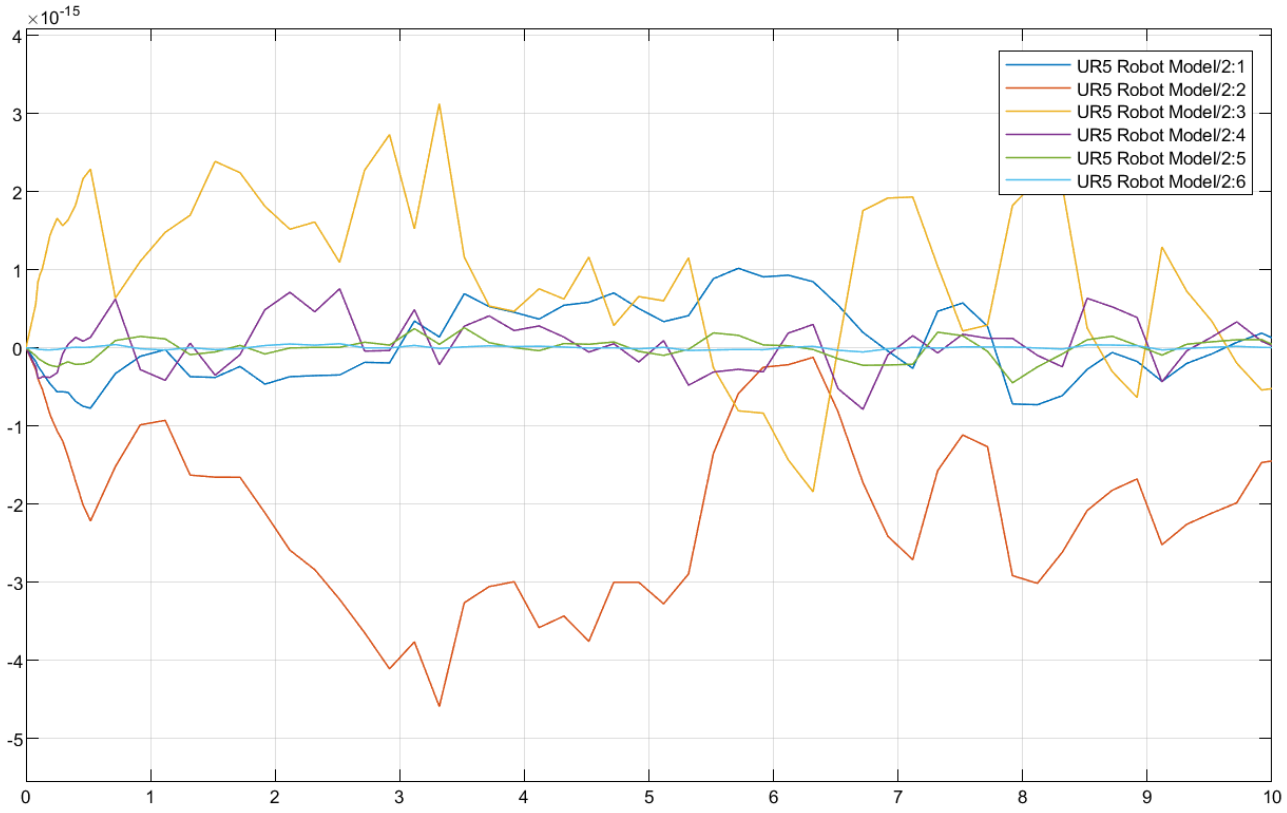


Figure 3. Joint velocity q_dot [rad/s]

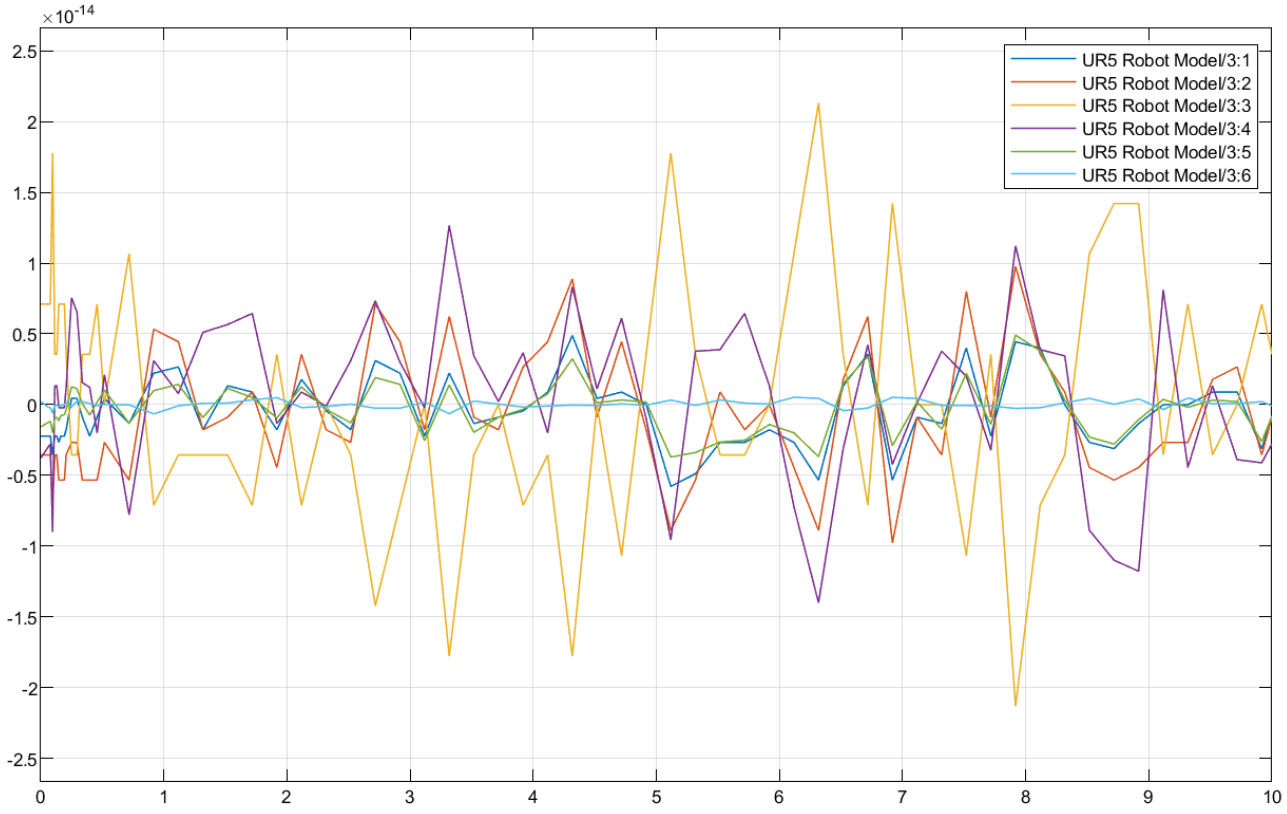


Figure 4. Joint acceleration q_ddot [rad/s^2]

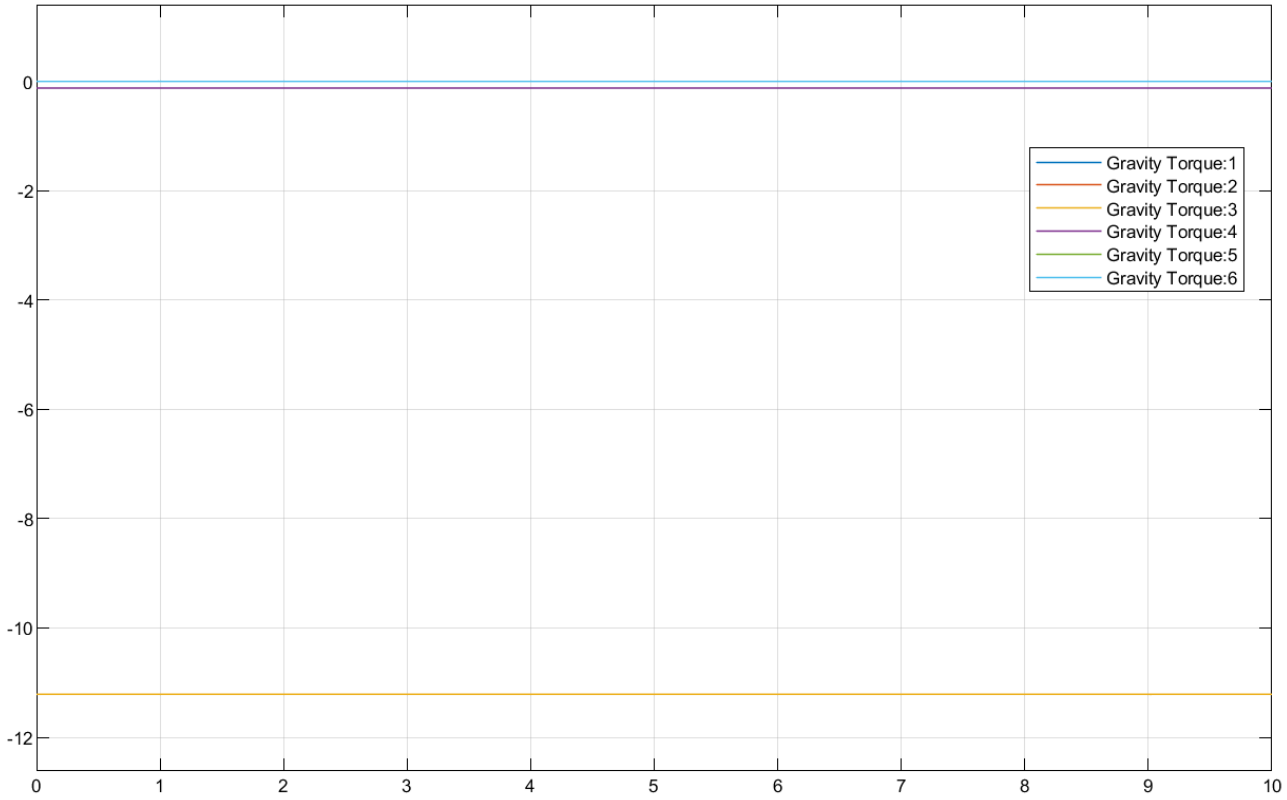
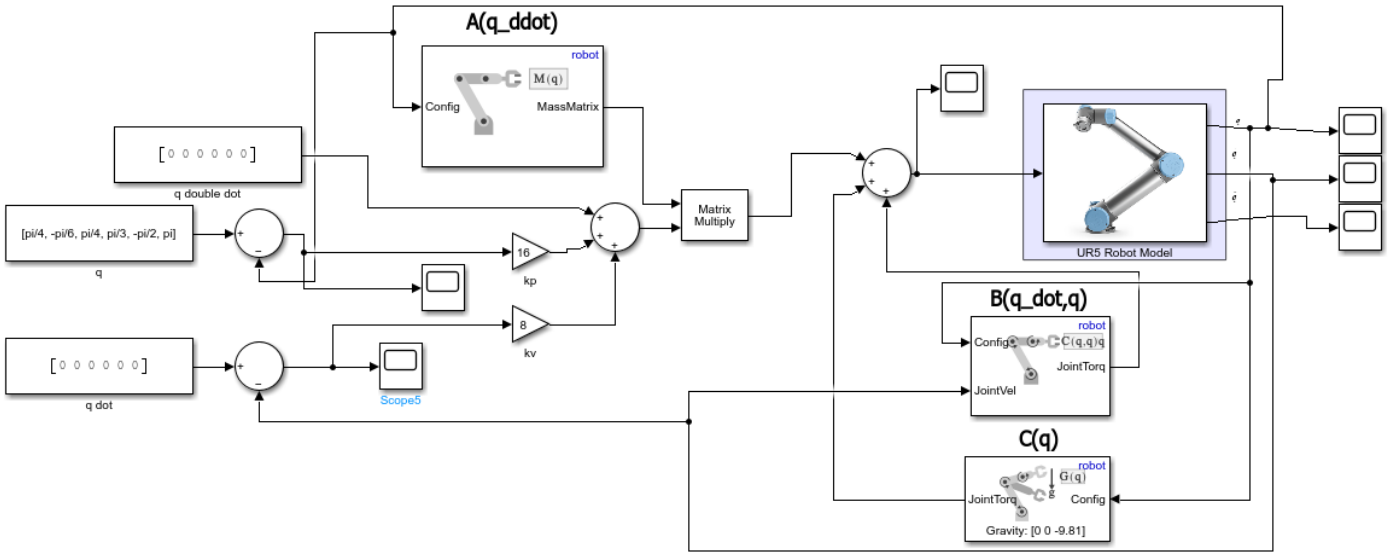


Figure 5. Torque joint

As note from the figures shown in Fig.2 *Joint position* and Fig.5 *Torque joint,* I have no changes in the values during 10 seconds of simulations, because the results of this exercise keeping the initial configuration of the robot and I avoided dropping due to gravity.

Exercise 2 – Linear Joint Control
Given a desired joint configuration q* = q0 + Δq where q0 is the initial joint configuration and Δq = [pi/4, -pi/6, pi/4, pi/3, -pi/2, pi], provide torque commands in order to reach q*.



The formula for controlling manipulator to receive a desired position is:

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

To find following matrices A, B, C are methods explained in second assignment, through Recursive Newton Euler Algorithm (RNEA).

I can use feedback for non-linearity cancelation, and linear control for stabilizing error to zero:

$$\ddot{q} = \ddot{q}^* - k_v(\dot{q} - \dot{q}^*) - k_p(q - q^*)$$

If we move all the components in to left side of equation, I will find following:

$$\delta\ddot{q} + k_v\delta\dot{q} + k_p\delta q = 0$$

Is a second order homogenous differential equation, so I must consider this fact to find *kp* and *kv*:

$$\delta\ddot{q} + 2\xi\omega_n\delta\dot{q} + {\omega_n}^2\delta q = 0$$

where ωn is the natural frequency of oscillation for each link and ξ is damping coefficient, so to set robot behavior in critically damped situation I can choose ξ value equal to one. As define in the robot model at the label 'Internal Mechanics' you will notice that a damping coefficient is provided to simulate joint friction.
Also considering the settling time for a second order system as described below I can find the natural frequency value:

$$t_s = \frac{4}{\xi\omega_n}$$

If I consider t_s equal to 1 sec, natural frequency it will be 4 rad/sec. So, I can calculate the *Kp* and *Kv*:

$$k_v = 2\xi\omega_n = 8$$

$$k_p = {\omega_n}^2 = 16$$

This value is not fixed, it has been calculated only considering a certain situation, however they can be replaced with larger values, in that case movements of the manipulator will be faster.

However, finding the following desired position in Simulink several blocks are already defined that allow me to do it in a simpler way.

To calculate ***A(q_ddot)*** matrix I used the Joint Space Mass Matrix block. Which returns the joint-space mass matrix for the given robot configuration (joint positions) for the **Rigid body tree** robot model.

To calculate ***B(q, q_dot)*** matrix I used the Velocity Product Torque block. Which returns the torques that cancel the velocity-induced forces for the given robot configuration (joint positions) and joint velocities for the **Rigid body tree** robot model.

To calculate ***C(q)*** matrix I used the Gravity Torque. Which block returns the joint torques required to hold the robot at a given configuration with the current Gravity setting on the **Rigid body tree** robot model.

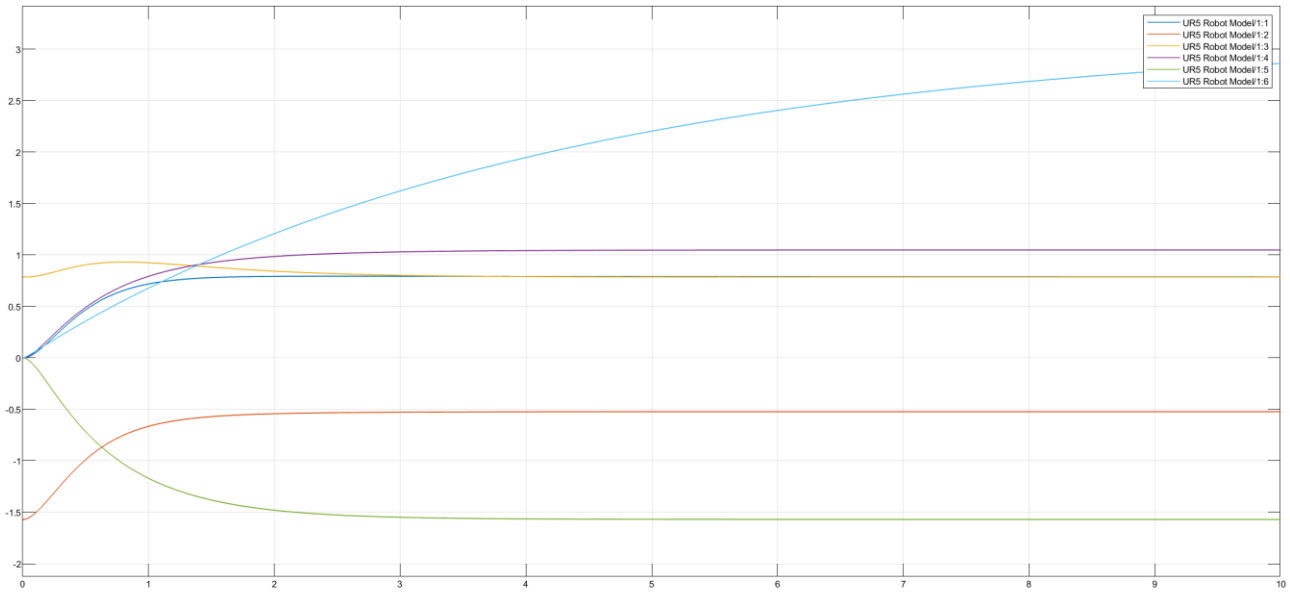


Figure 6. Joint position[rad]

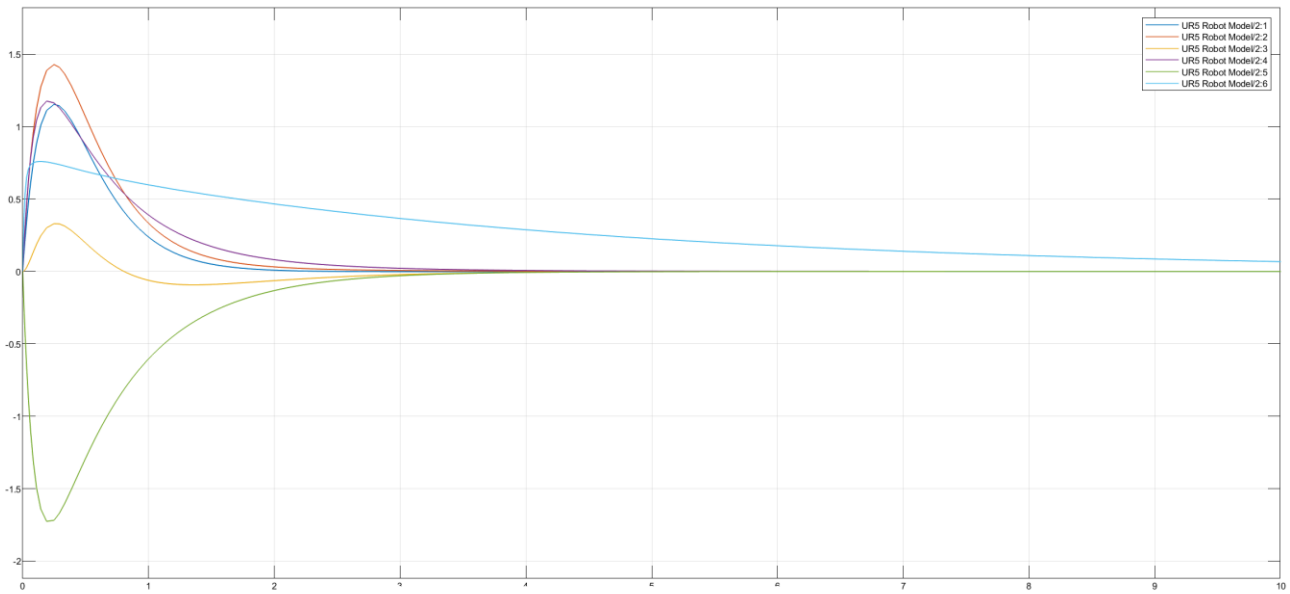


Figure 7. Joint velocity[rad/sec]

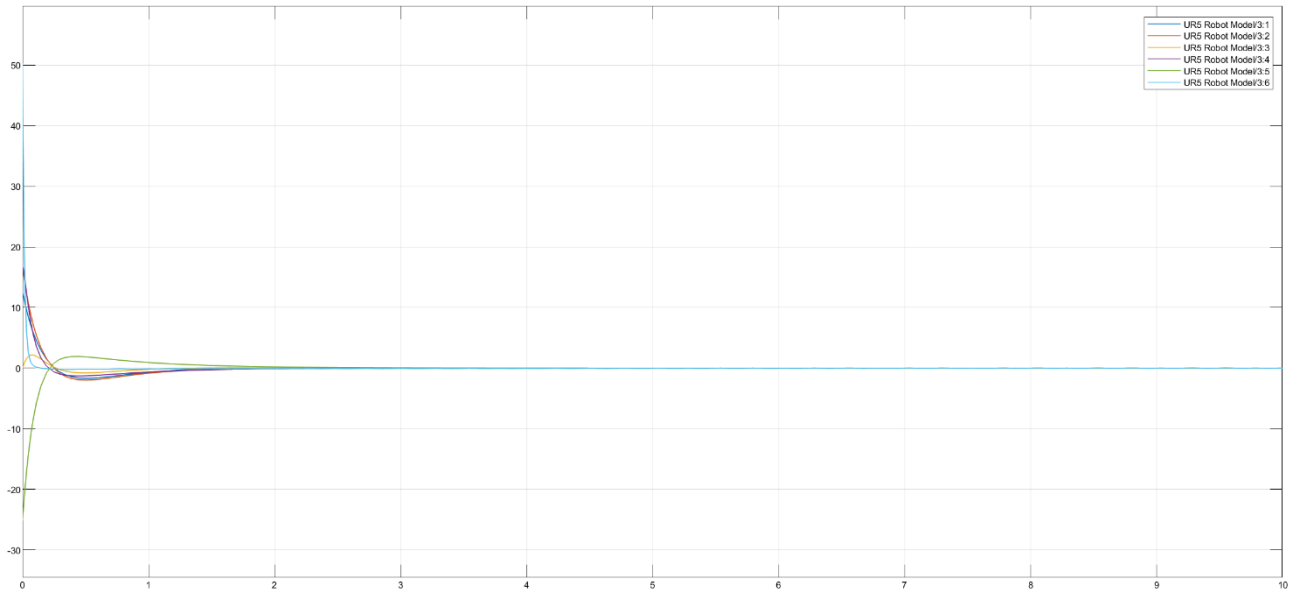


Figure 8. Joint acceleration[rad/sec^2]

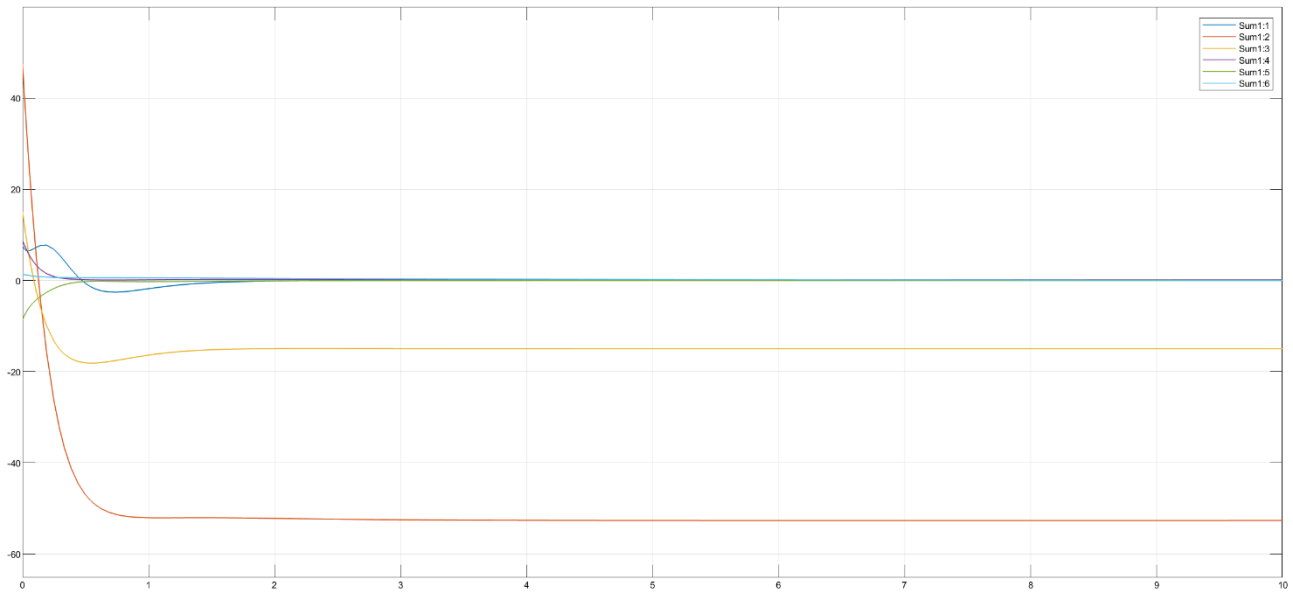


Figure 9. Torque joint

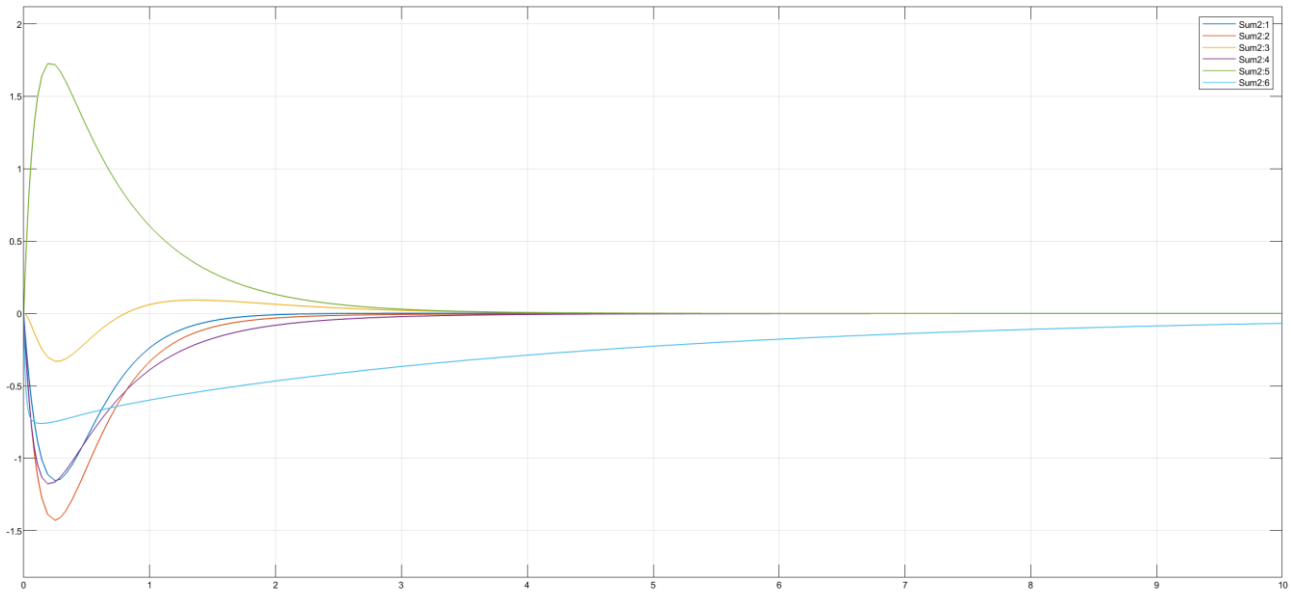


Figure 10. Delta q [rad]

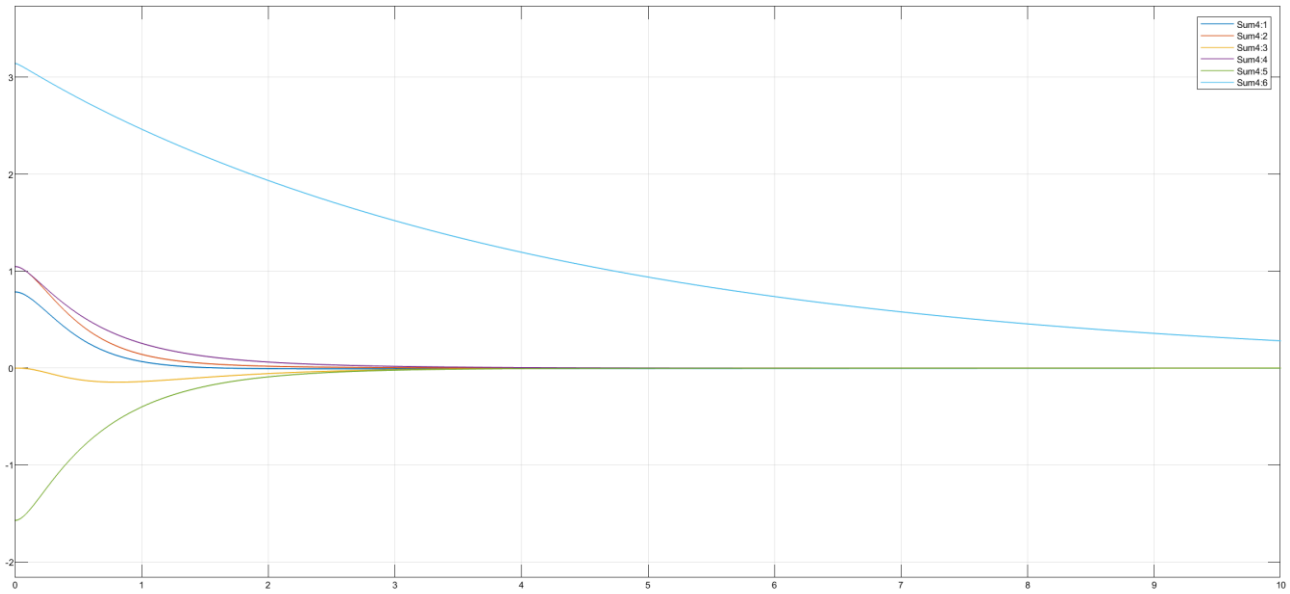


Figure 11. Delta q_dot [rad/sec]

To check the robot control error, you can look at fig. 10 and fig. 11 if they tend to zero it means that the implementation of the scheme is correct.

What happens if you don’t compensate gravity?

In this case it is no longer needed to use block ***C(q)*** Gravity Torque because I consider gravity equal to zero.

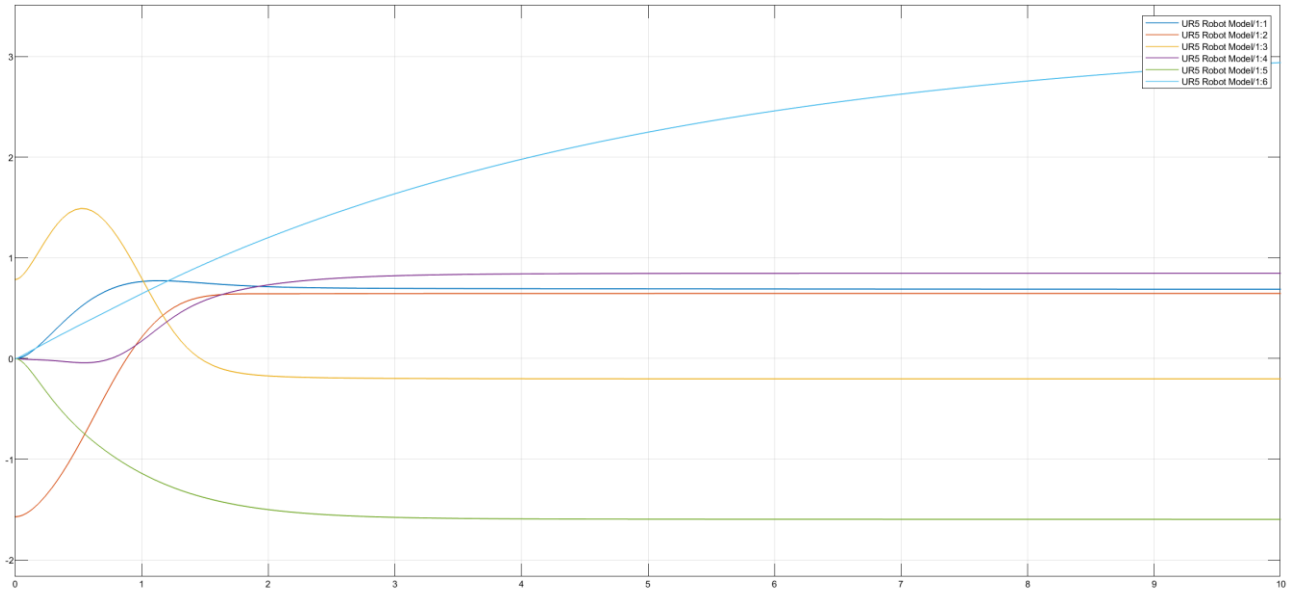


Figure 12. Joint position[rad]

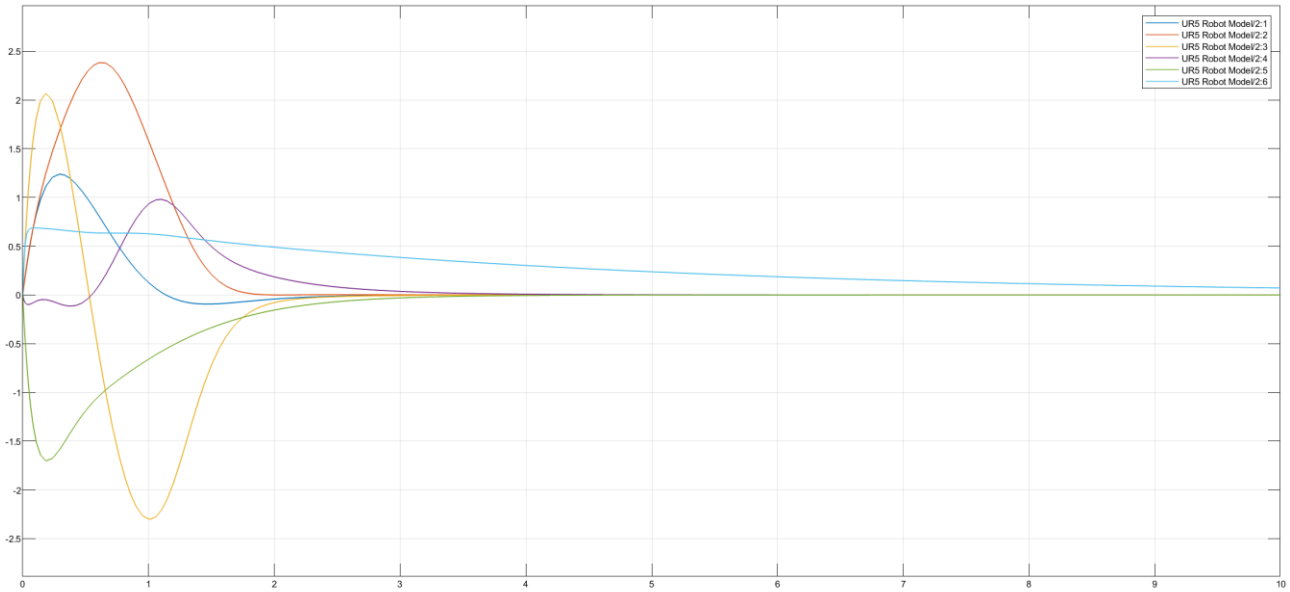


Figure 13. Joint velocity[rad/sec]

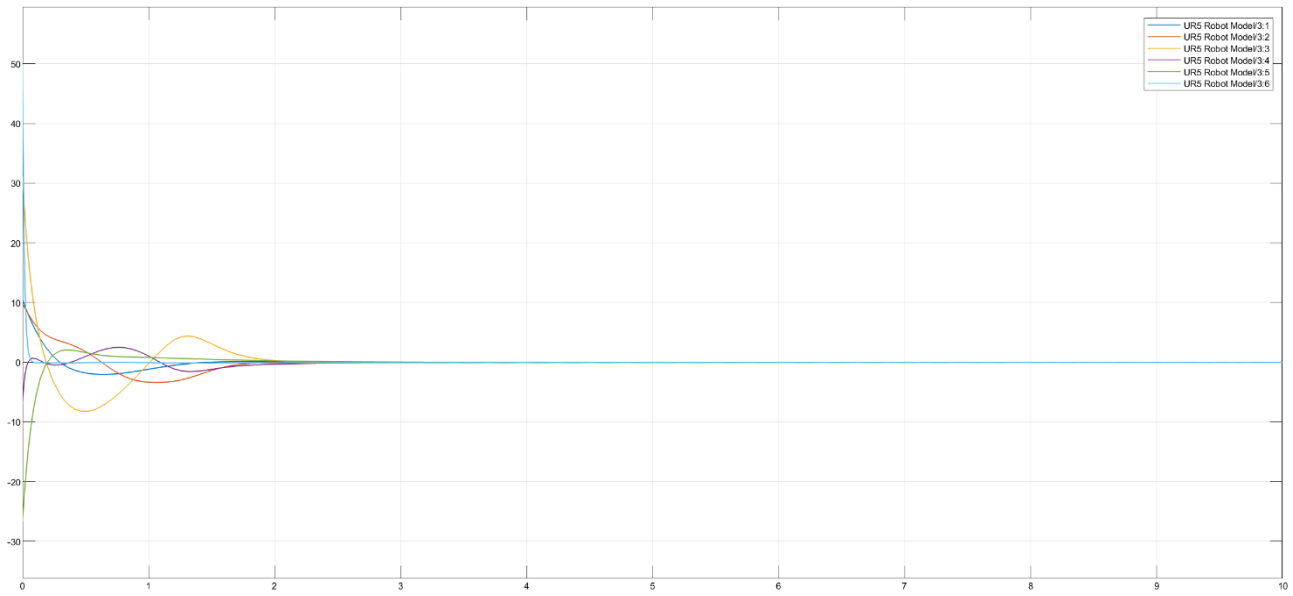


Figure 14. Joint acceleration[rad/sec^2]

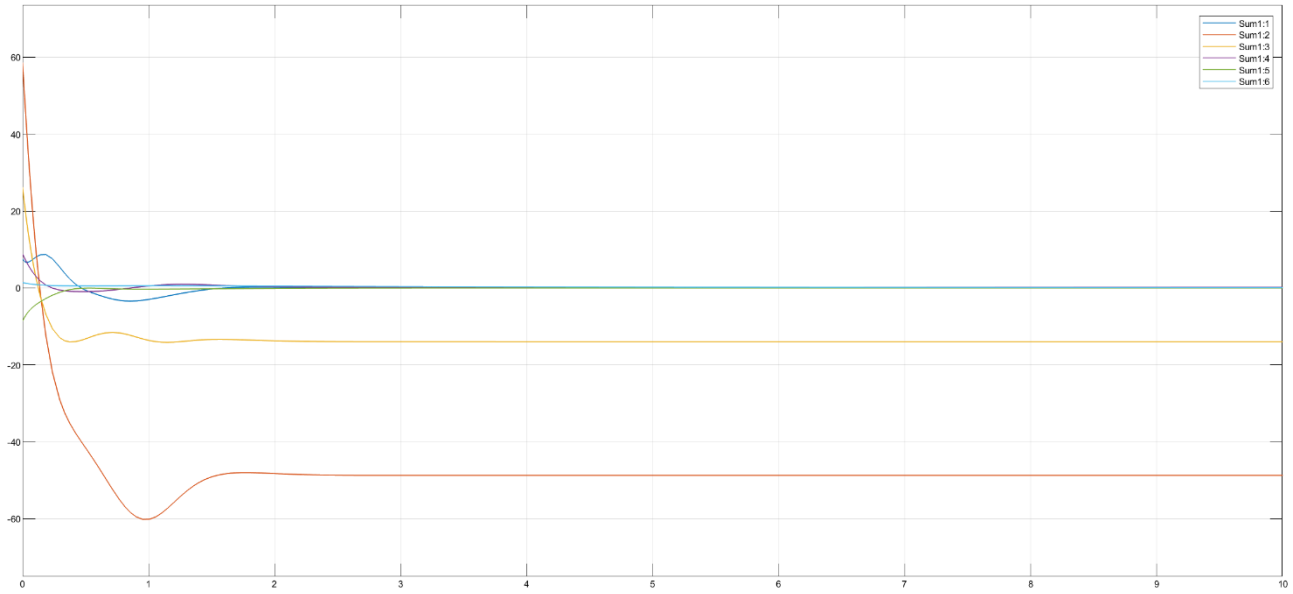


Figure 15. Torque joint

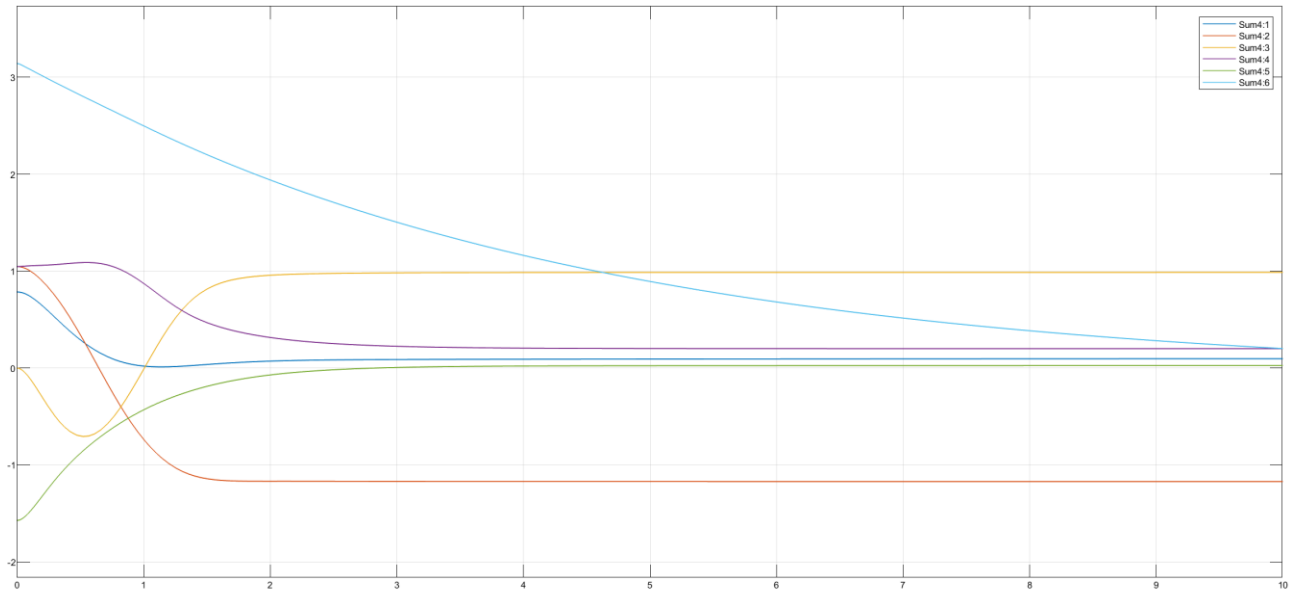


Figure 16. Delta q [rad]

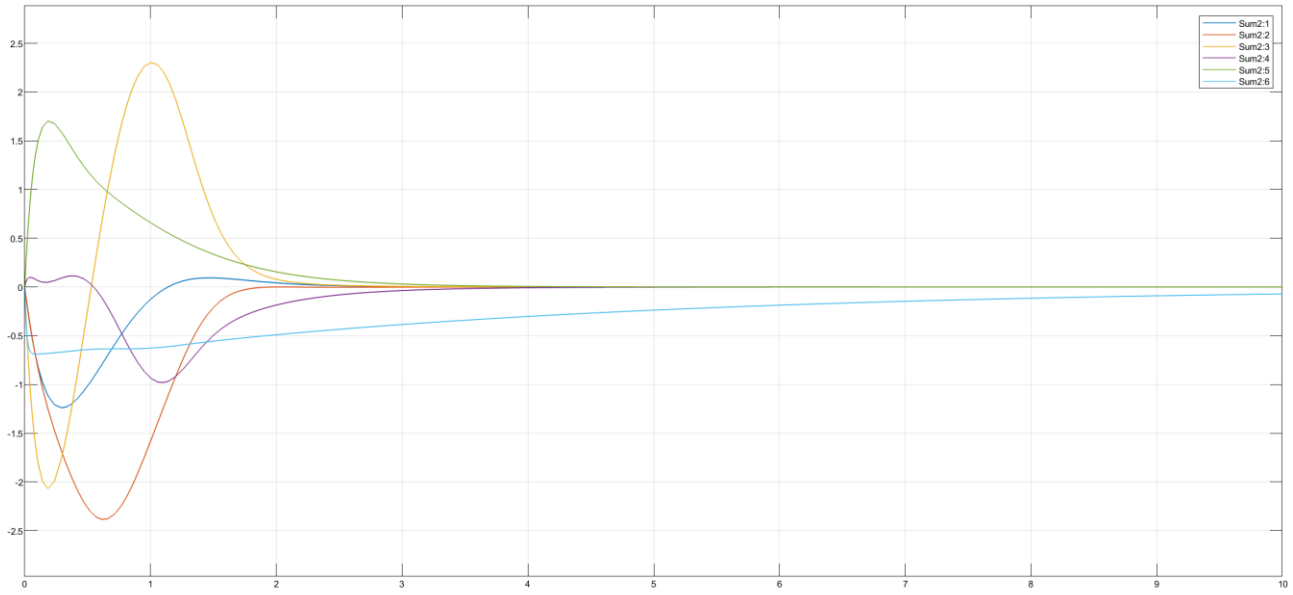
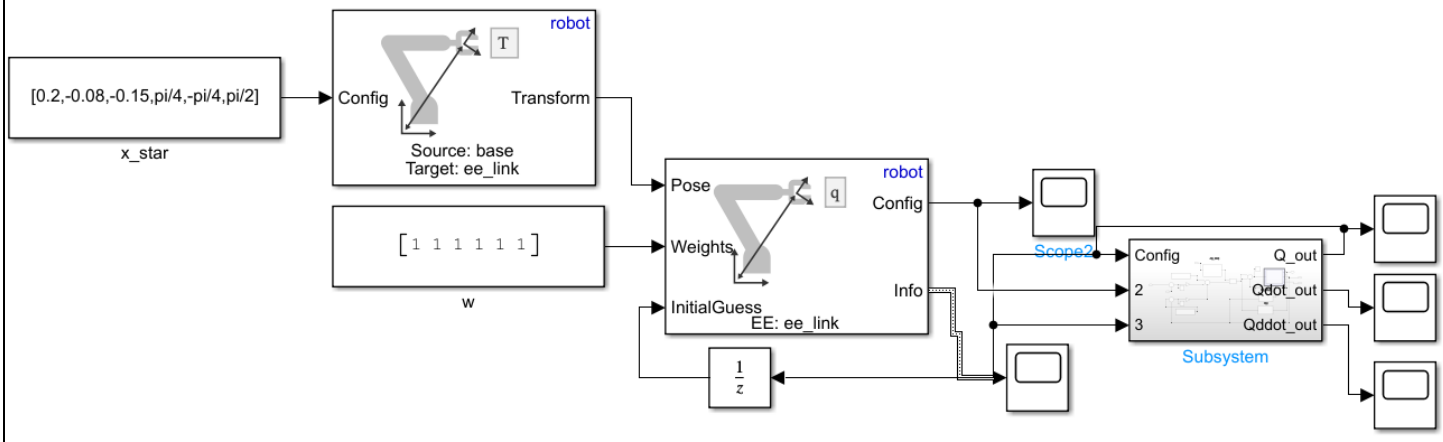


Figure 17. Delta q_dot [rad/sec]

Obviously due to lack of gravity the movement of the robot has changed and has not reached the desired joint position. Even in fig.16 and fig.17 I note that some errors do not tend to zero or it's tended to zero very slowly.

Exercise 3 – Linear Cartesian Control

Given a desired cartesian configuration x* = x0 + Δx where x0 is the initial cartesian pose of the end effector ('ee_link') and Δx = [0.2, -0.08, -0.15, pi/4, -pi/4, pi/2], provide torque commands in order to reach x*.



To find the desired position of the manipulator first I need to calculate the transformation matrix, then the configuration found is brought to subsystem (same scheme of the second exercise).

To calculate the transformation matrix, I used the Get Transform block. Which returns the homogeneous transformation between body frames on the **Rigid body tree** robot model. Specify a rigidBodyTree object for the robot model and select a source and target body in the block, in my case from base to ee_link.

The block uses **Config**, the robot configuration (joint positions) input, to calculate the transformation from the source body to the target body. This transformation is used to convert coordinates from the source to the target body. To convert to base coordinates, use the base body name as the **Target body** parameter.

Output of this block is [4*4] matrix as position of ee_link, so I can't use it directly like my configuration. Because my configuration is [1*6] matrix. Therefore, I use also Inverse kinematics (IK).

To determines joint configurations of a robot model to achieve a desired end-effect position. Robot kinematic constraints are specified in the rigidBodyTree robot model based on the transformation between joints.

After that output is brought to sub system.

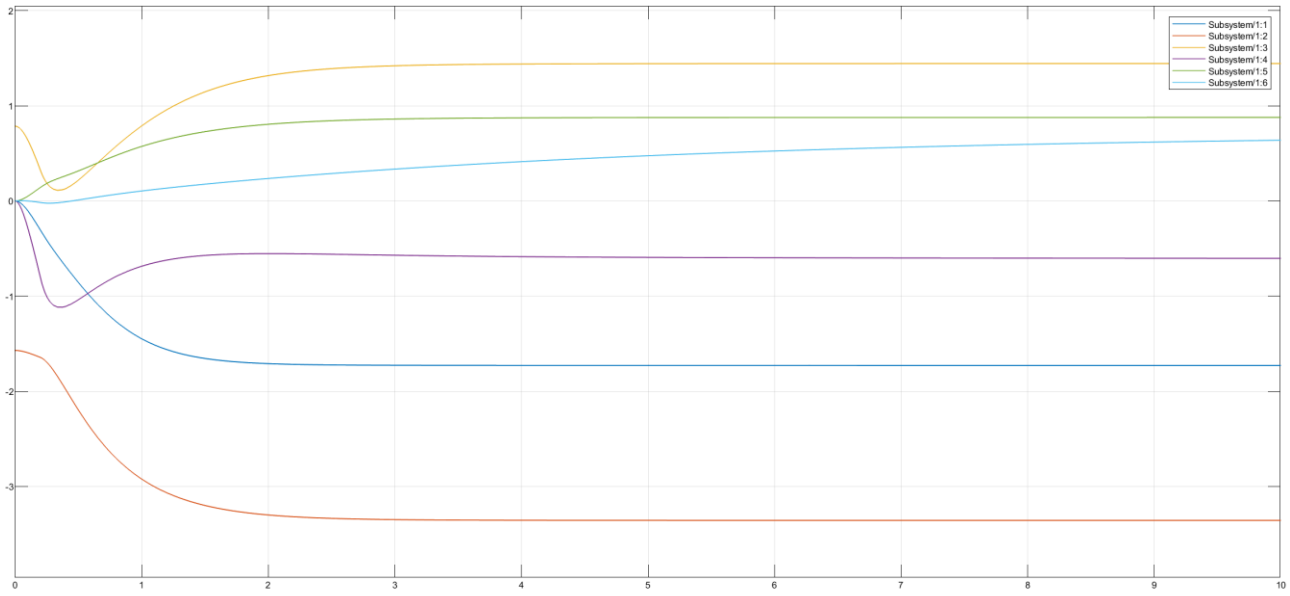


Figure 18. Joint position[rad]

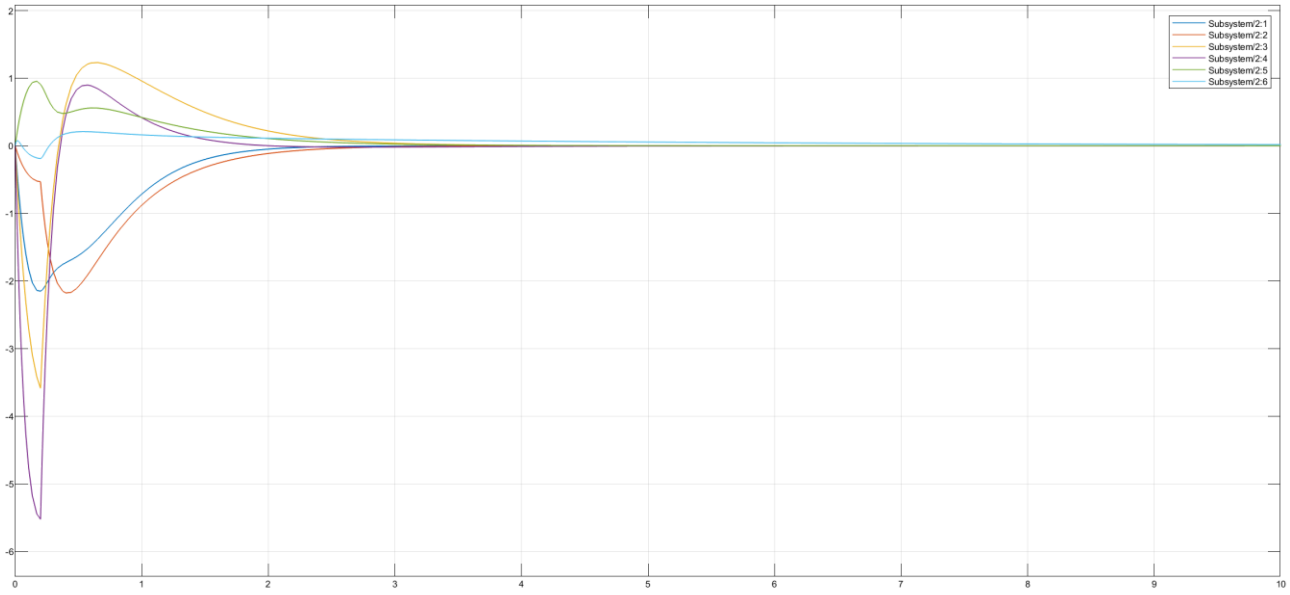


Figure 19. Joint velocity[rad/sec]

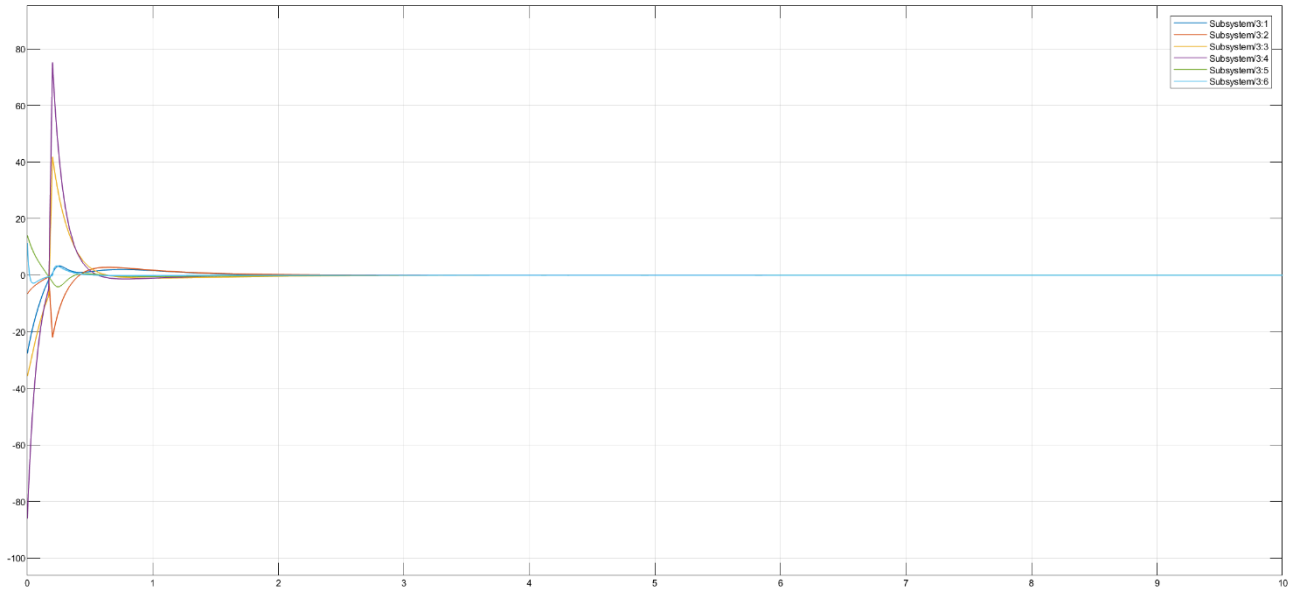


Figure 20. Joint acceleration[rad/sec^2]

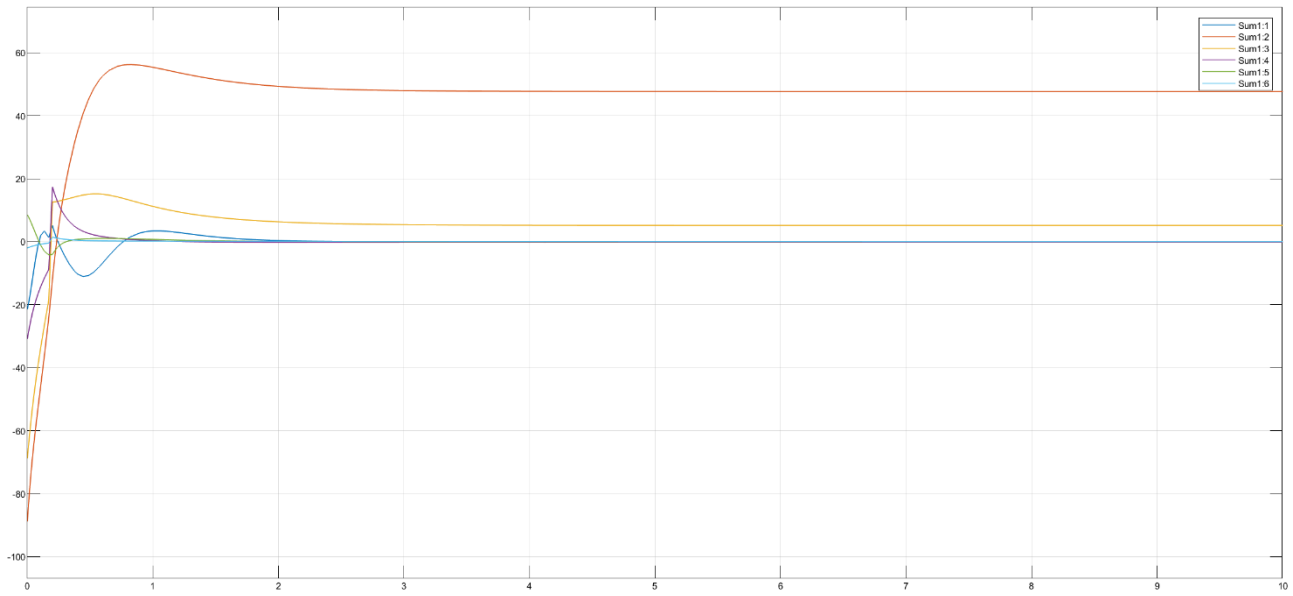


Figure 21. Torque joint

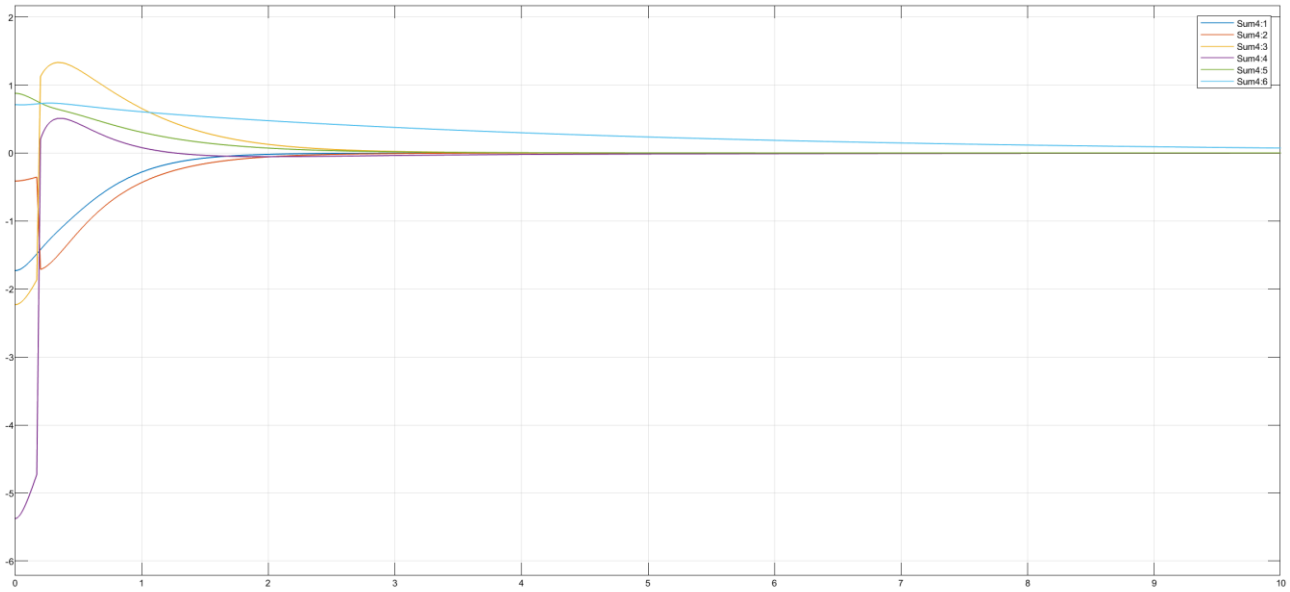


Figure 22. Delta q [rad]

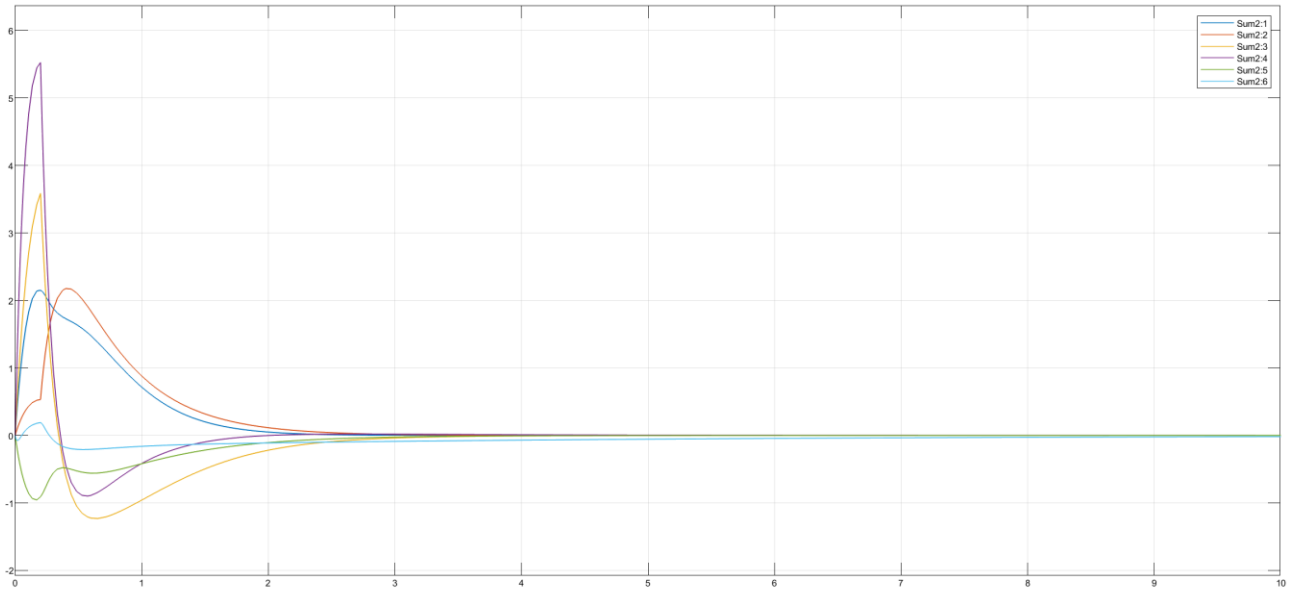


Figure 23. Delta q_dot [rad/sec]

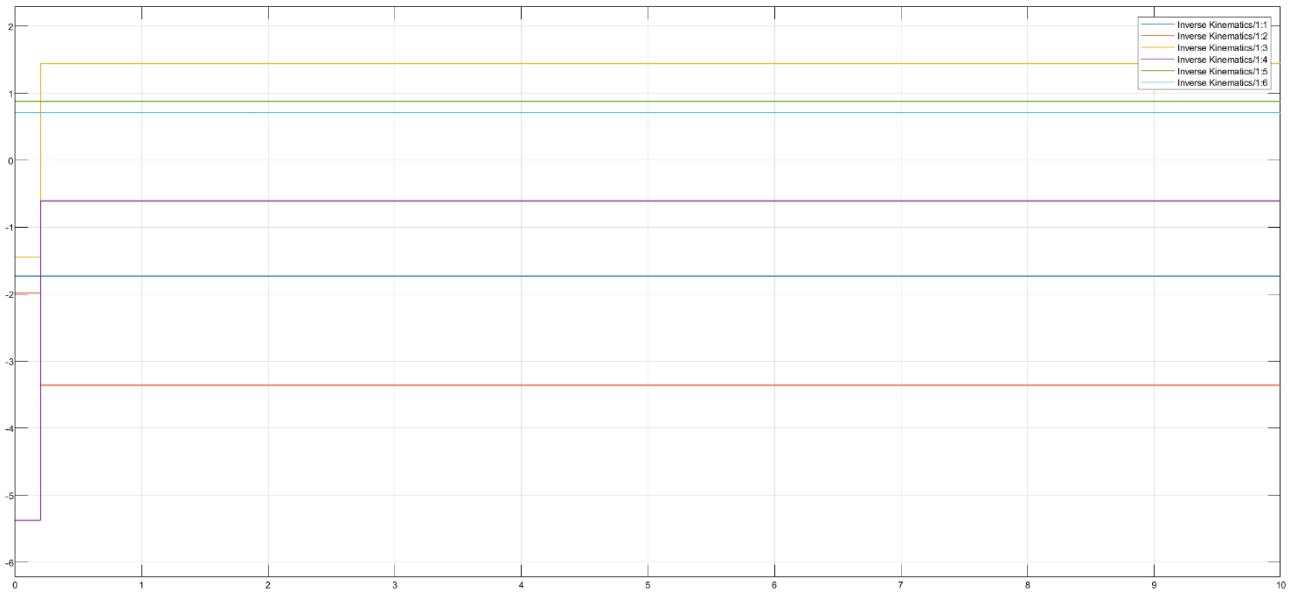


Figure 24. Robot configuration after IK block

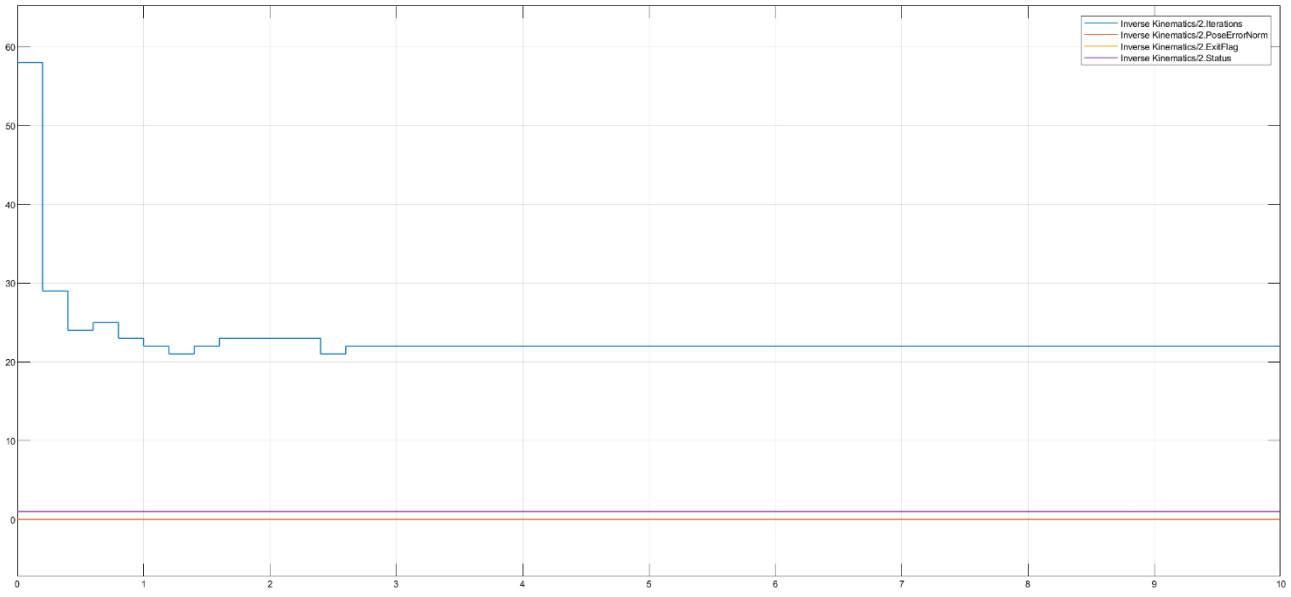
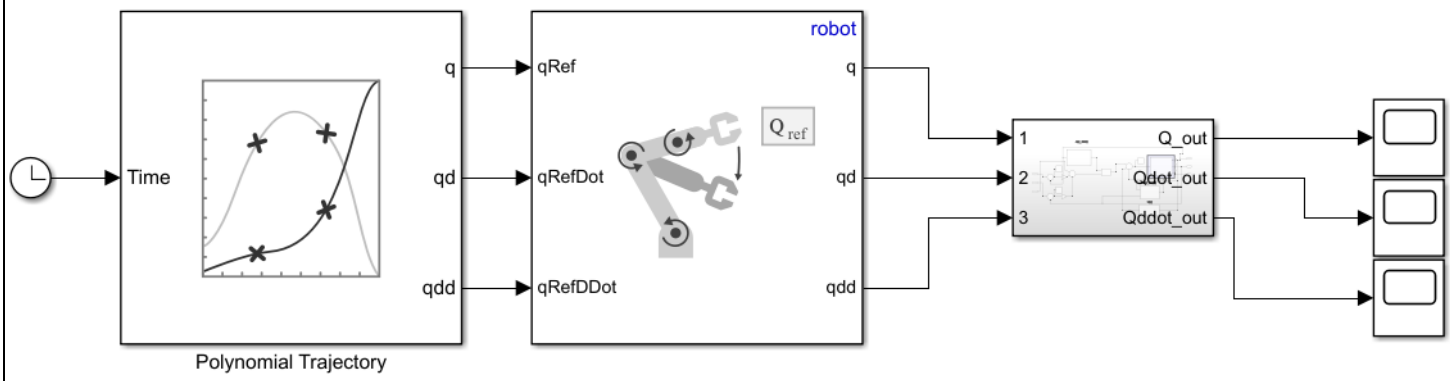


Figure 25. Output info from Ik block

Also, in this case to check robot control error, you can look fig. 21 and fig. 22, error tend to zero. In additional in fig. 24 been reported Pose Error which is equal to zero.

Exercise 4 – Computed Torque Control

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



To arrive at the desired joint position the block was used the Polynomial Trajectory block generates trajectories to travel through waypoints (configuration of second exercise) at the given time points using either cubic, quintic, or B-spline polynomials. The block outputs positions, velocities, and accelerations for achieving this trajectory based on the **Time** input. The initial and final values are held constant outside the time defined in **Time points**.

To generate joint-space trajectories in order to reach qref from q output , I used the Joint Space Motion Model block models the closed-loop joint-space motion of a manipulator robot, specified as a rigidBodyTree object. The motion model behavior is defined by the Motion Type parameter. And rest of system is equal before. In the first case I set to zero the damping coefficient. Damping is the phenomenon by which mechanical energy is dissipated (usually by conversion into internal thermal energy) in dynamic systems.

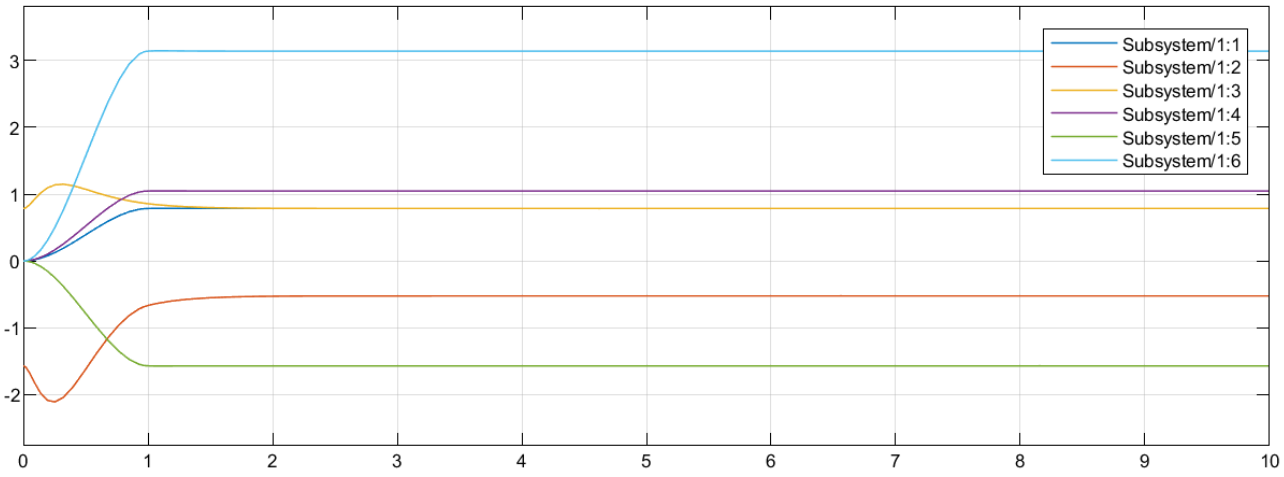


Figure 26. Joint position[rad]

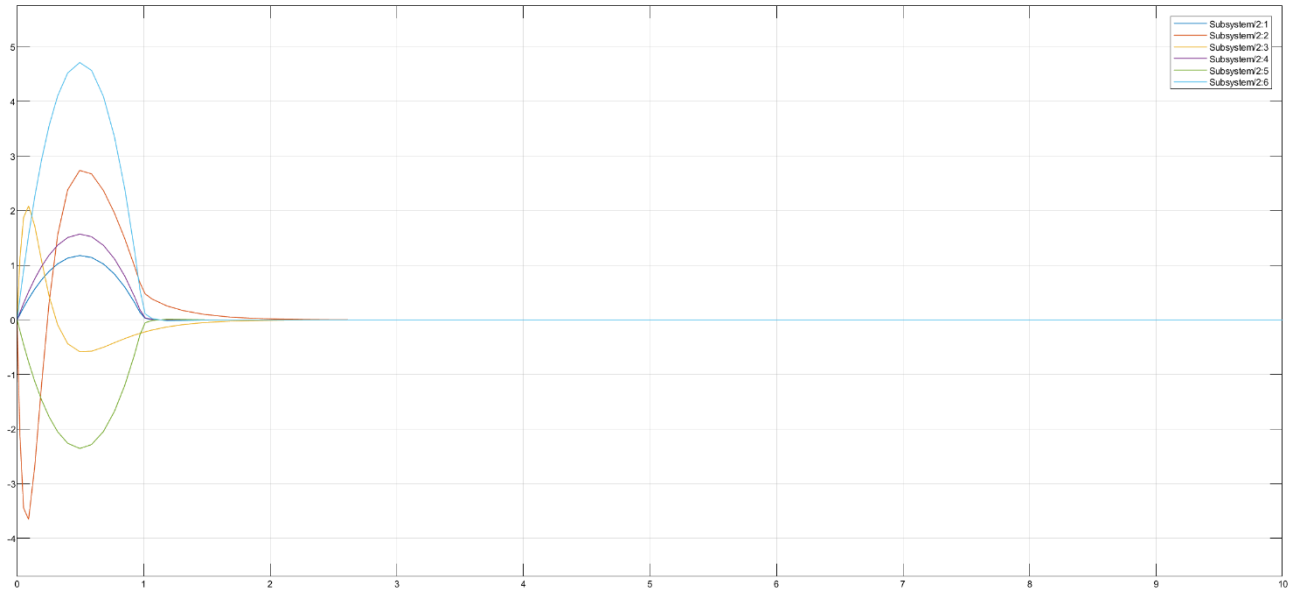


Figure 27. Joint velocity[rad/sec]

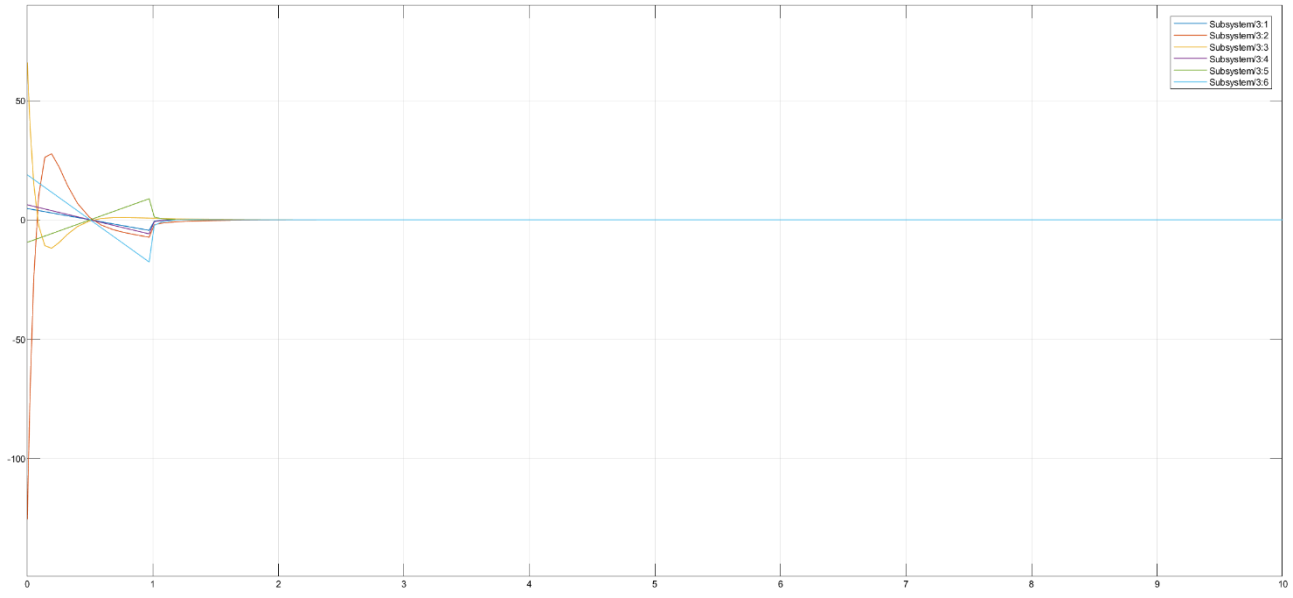


Figure 28. Joint acceleration[rad/sec^2]

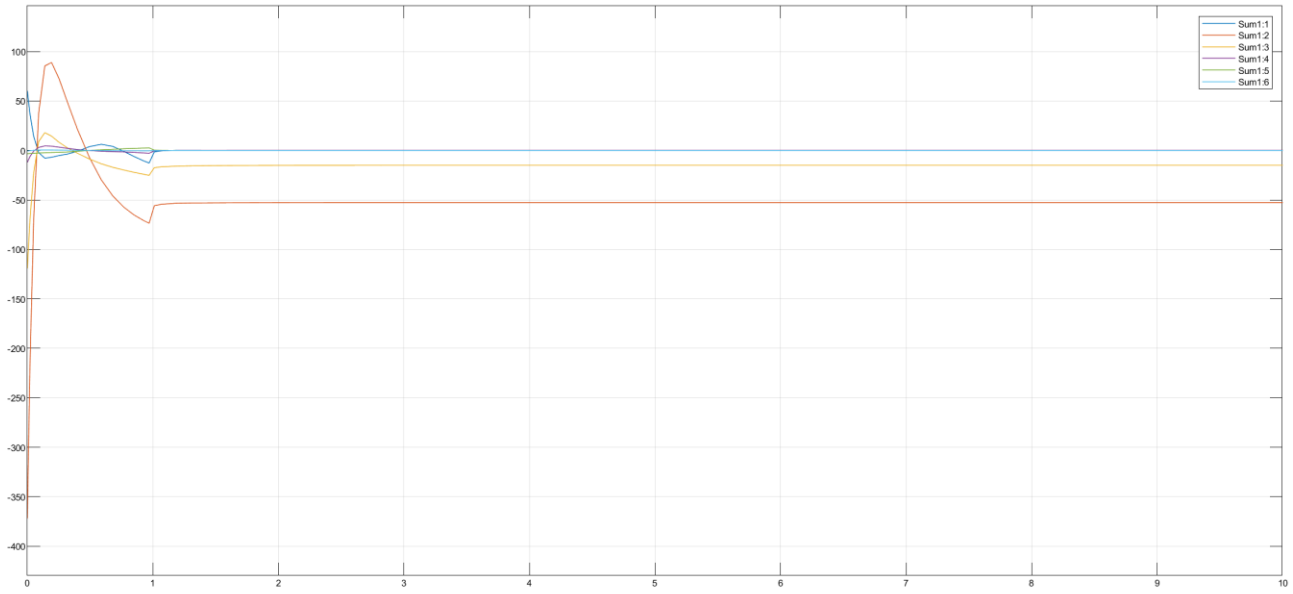


Figure 29. Torque joint

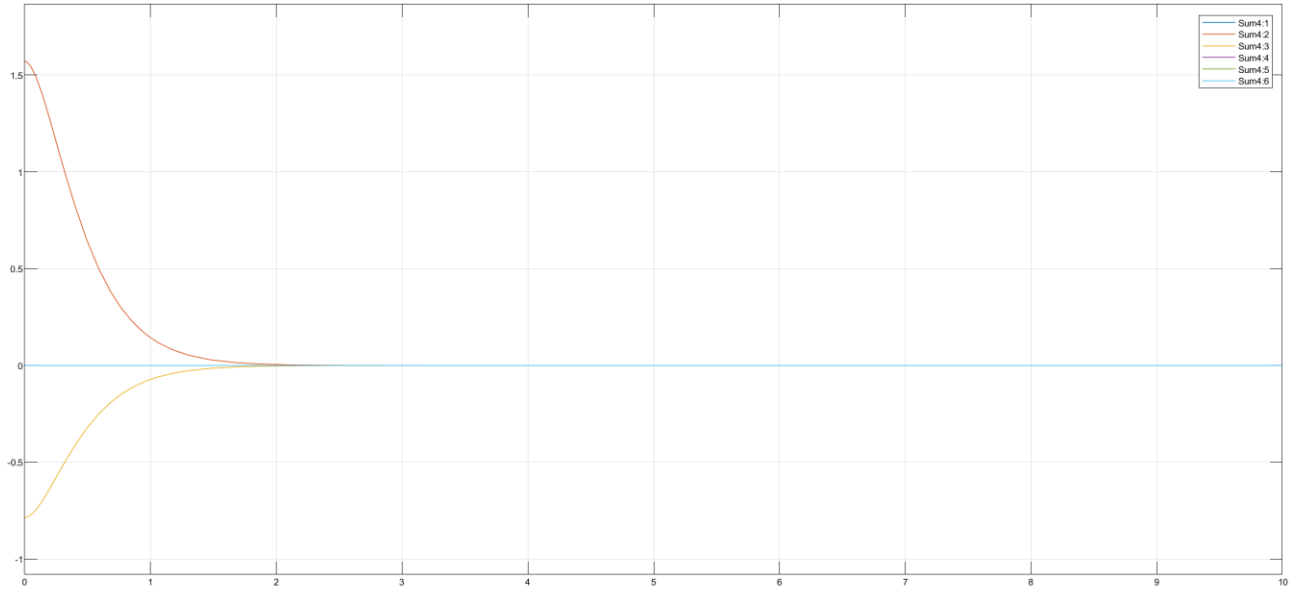


Figure 30. Delta q[rad]

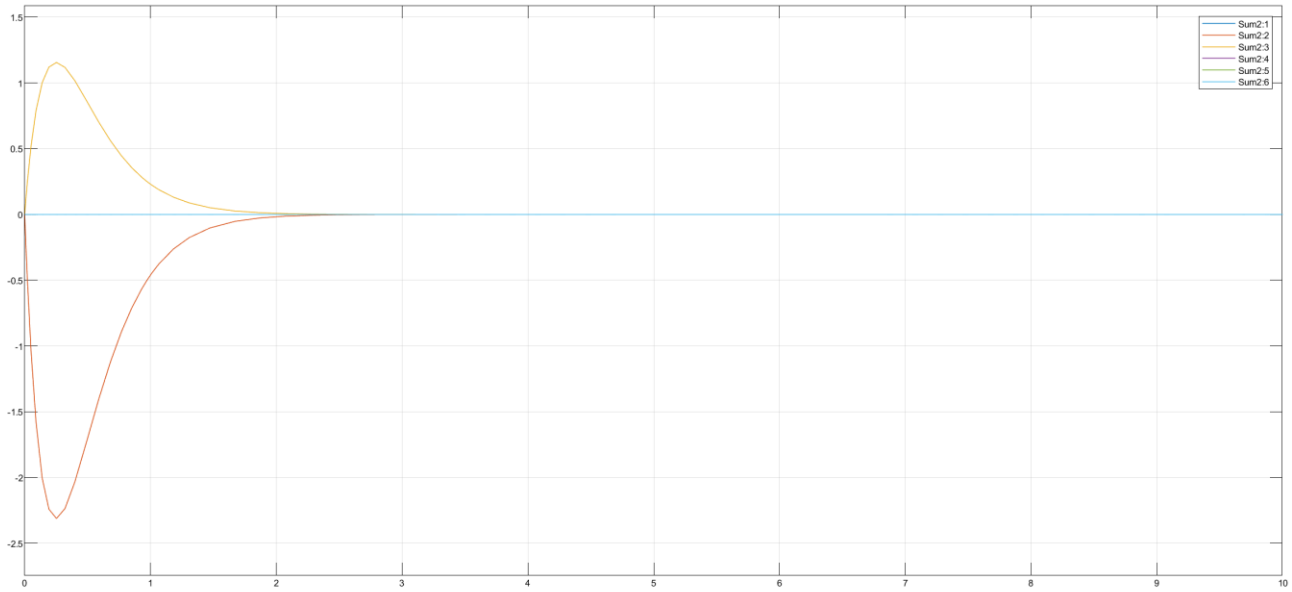


Figure 31. Delta q-dot[rad/sec]

With damping coefficient equal to 2.

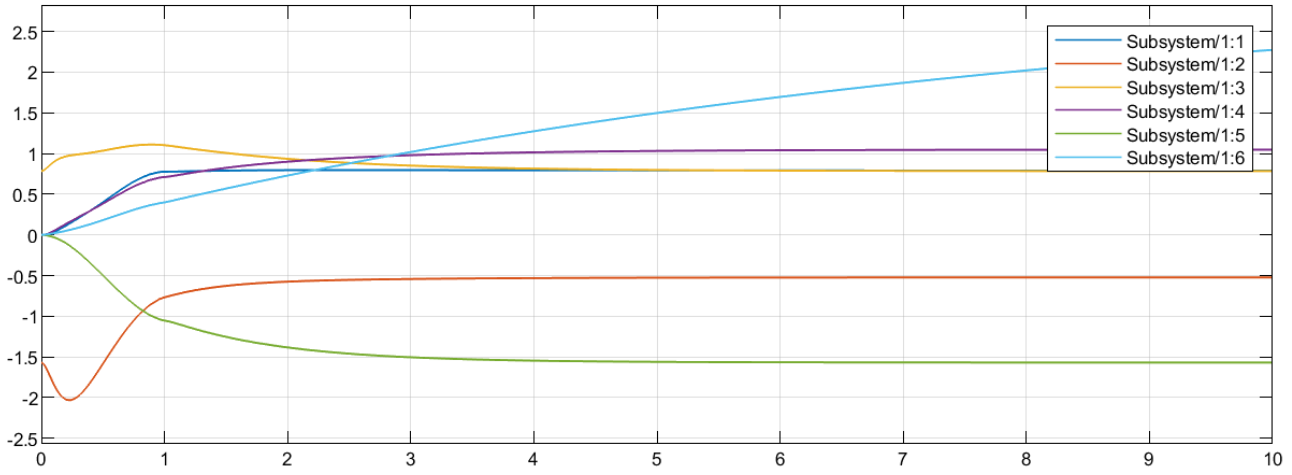


Figure 32. Joint position[rad]

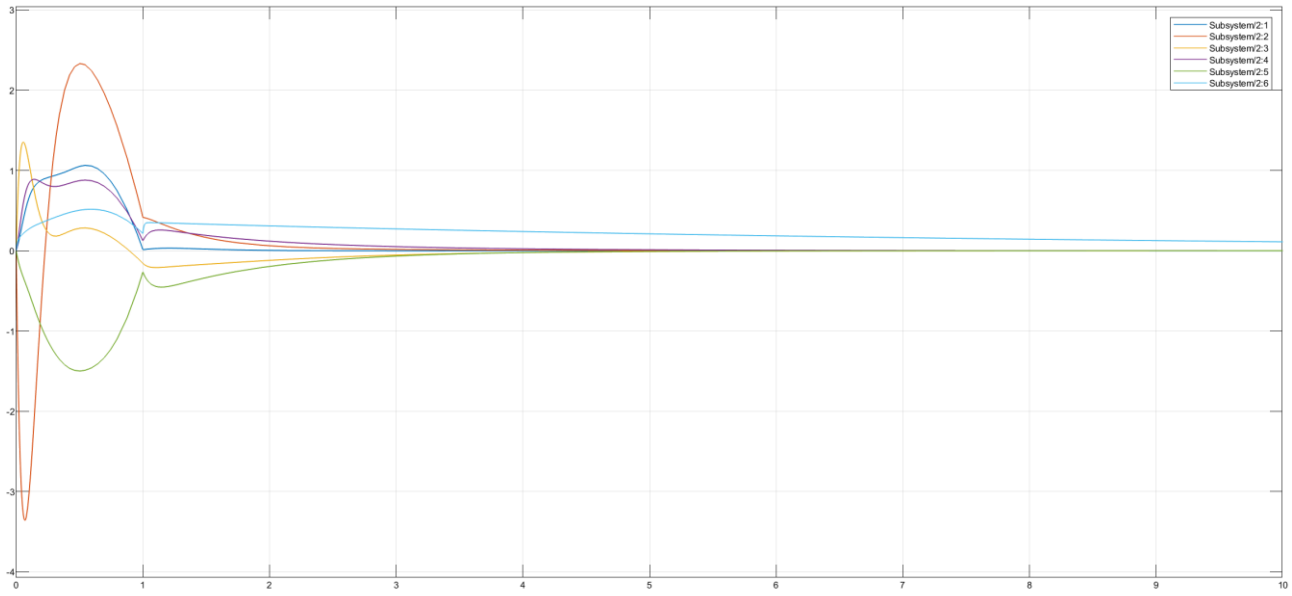


Figure 33. Joint velocity[rad/sec]

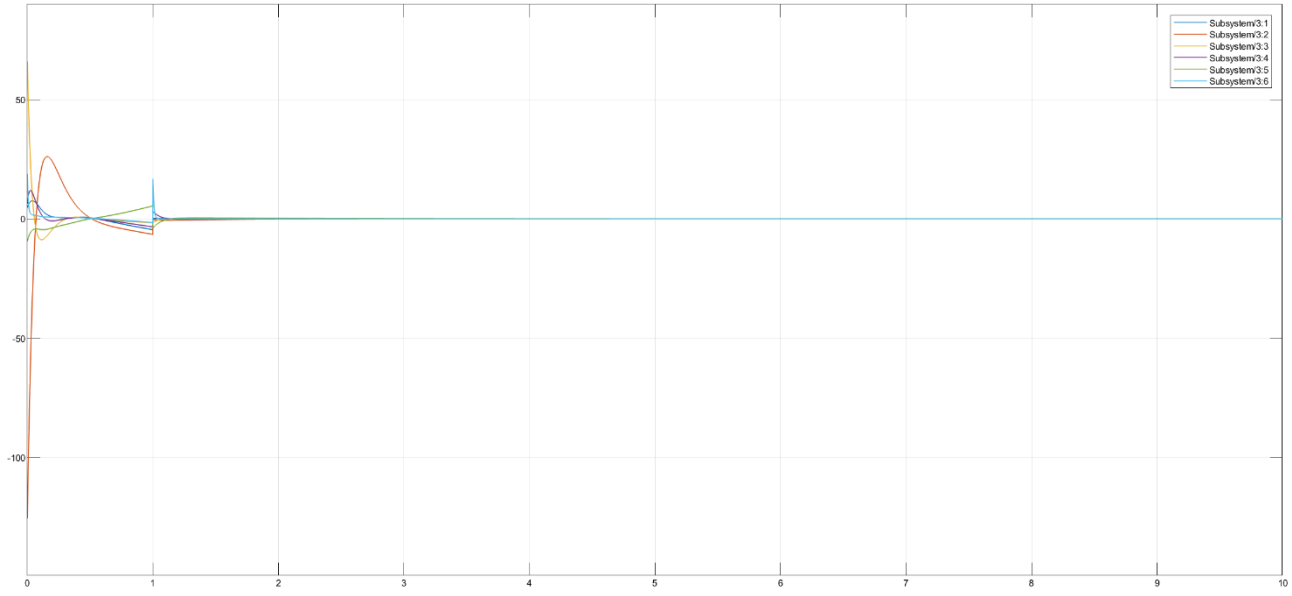


Figure 33. Joint acceleration[rad/sec^2]

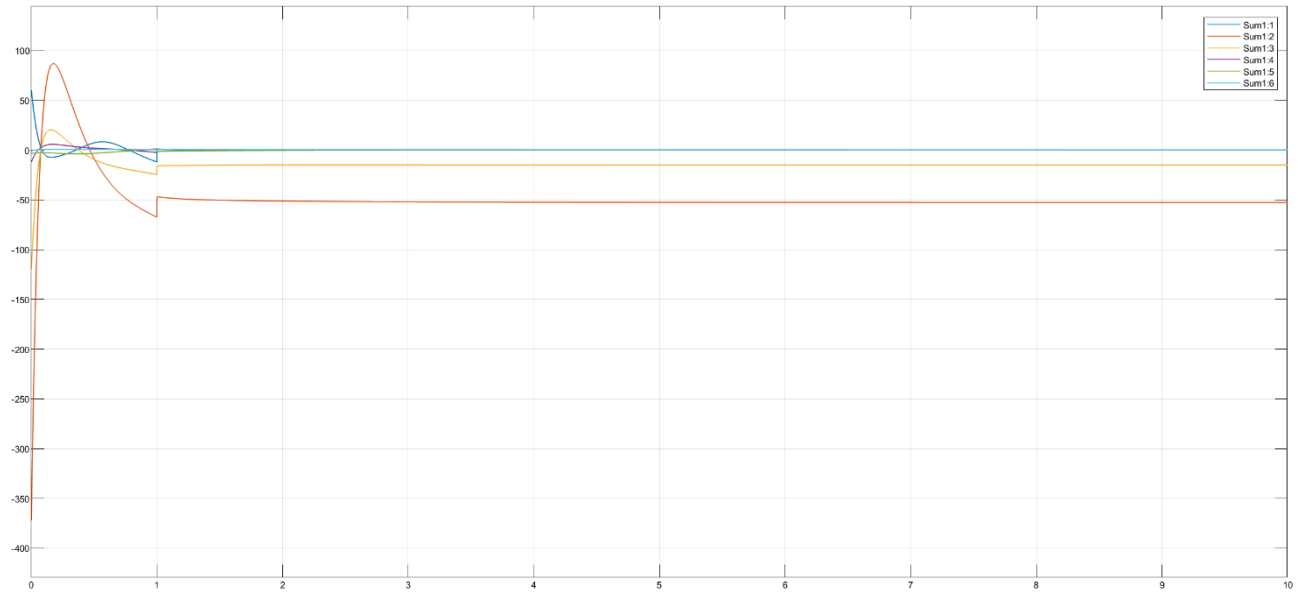


Figure 34. Torque joint

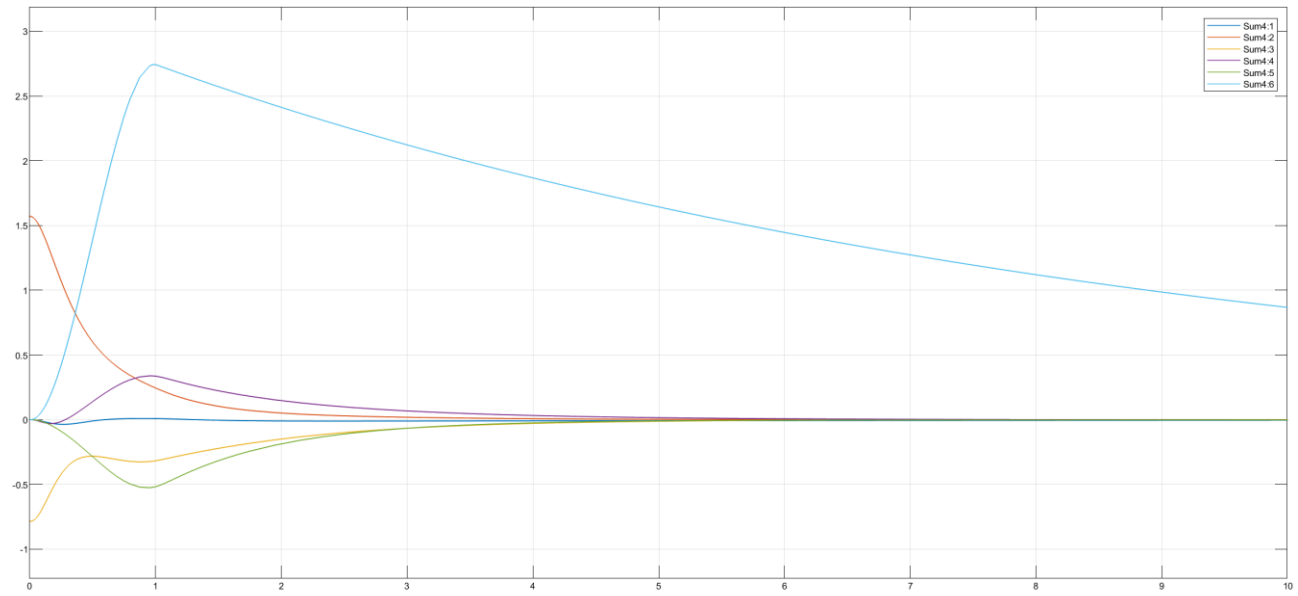


Figure 35. Delta q[rad]

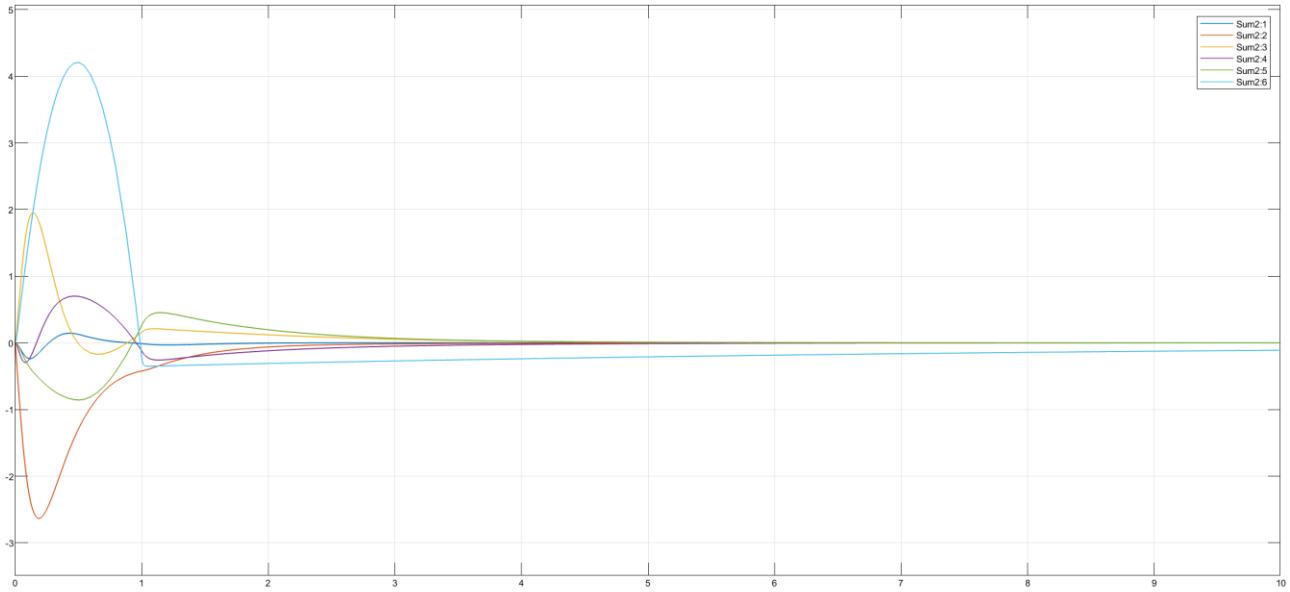


Figure 26. Delta q-dot[rad/sec]

In the first case without damping coefficient the *delta q* and *delta q-dot* tend to zero immediately instead in the case with damping equal to 2 the error slowly tends to zero and as regards the *speed, acceleration, and torque joint* in the case of damping 2 it can be noted a point of discontinuity in the plot also in the movement of the robot which makes the robot less effective