Robot Dynamics and Control Assignment 3

Alessandro Perri, S4476726

A.A. 2021/2022

Introduction

The third assignment of this course is based on the dynamic control of a manipulator. The four exercises that follow will demonstrate how to operate a simple simulation model of a **UR5 robot** (6R manipulator) by giving torque inputs to the joints. Therefore, it is important to recall the generalized equations of motion for a multiple-link robot manipulator, expressed in Lagrange-Euler formulation:

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

where:

- $\tau(t)$, nx1 generalized torque or force input vector,
- $\mathbf{q}(t)$, nx1 vector of manipulator joint variables,
- $\dot{\mathbf{q}}(t)$, nx1 vector of manipulator joint velocities,
- $\ddot{\mathbf{q}}(t)$, nx1 vector of manipulator joint accelerations,
- $\mathbf{A}(\mathbf{q}(t))$, nxn generalized inertia matrix,
- $\mathbf{B}(\mathbf{q}(t), \dot{\mathbf{q}}(t))\dot{\mathbf{q}}(t)$, nx1 nonlinear Coriolis and centrifugal force vector,
- $\mathbf{C}(\mathbf{q}(t))$, nx1 gravity loading vector.

The idea of the assignment is to design simple Simulink models that could solve some basic manipulation control problems including gravity compensation, set point regulation in cartesian and joint space, and joint space trajectory tracking. Each problem instance focuses on providing a toque command in the joint space that can regulate the motion of the manipulator's links to achieve a certain goal position.

The **Lyapunov analysis** is a procedure that aims to find the appropriate control signal u(t) (input torques vector) that would ensure that the time derivative of V(x) (i.e the Lyapunov function) is always negative given a generic control action objective, such as zeroing the joint velocities and considering V(x) as a positive definite function of the state of the system that exploits the kinetic energy of the system. This input vector (i.e. the control signal **u**(t)) will cause V(x) to decrease in kinetic energy until it approaches 0, causing the link to cease moving. Multiple interpretations of this basic idea of control can be used to meet all of the workout criteria.

Exercise 1 - Gravity Compensation

The aim of this exercise is to provide torque commands to the robot such that is doesn't fall due to gravity but holds the initial configuration. In this case, the only contribute needed to be compensated is the one related to the gravitational effect. In Simulink, the **Gravity Torque block** accepts an *n x 1* vector of joint positions (rad or m) and computes the joint torques that compensate gravity for a given configuration, i.e the vector **C(q)**. As a result, the robot will avoid falling due to gravity.

The control law is:

$$\mathbf{u} = \mathbf{C}(\mathbf{q})$$

Therefore, it was sufficient to add this block to the system to obtain the desired behaviour:

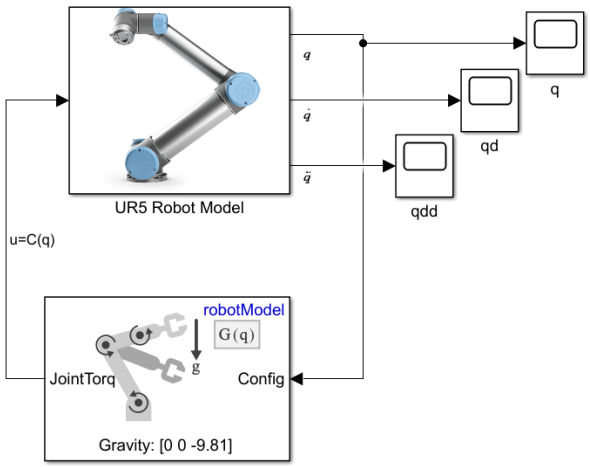

Figure 1a - Gravity Compensation

As you can see in the following figures, the resulting joint positions values $\mathbf{q}$ are (almost) constant (Fig. 1b) as well as the values of the computed torque vector τ (Fig. 1e). Trivially, the values of joint velocities (Fig. 1c) and acceleration will be (almost) null vectors since they are very small oscillations around 0.

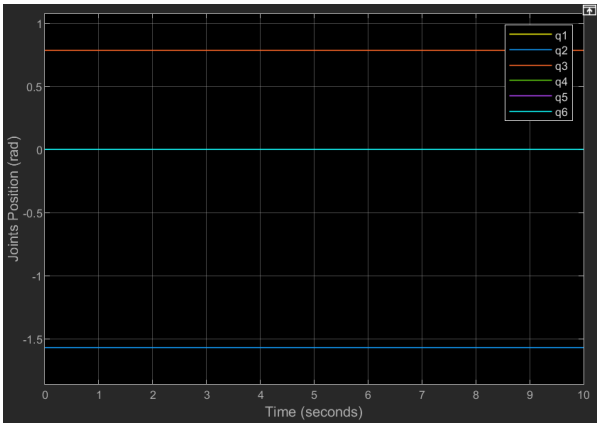

Figure 1b - Joints Position

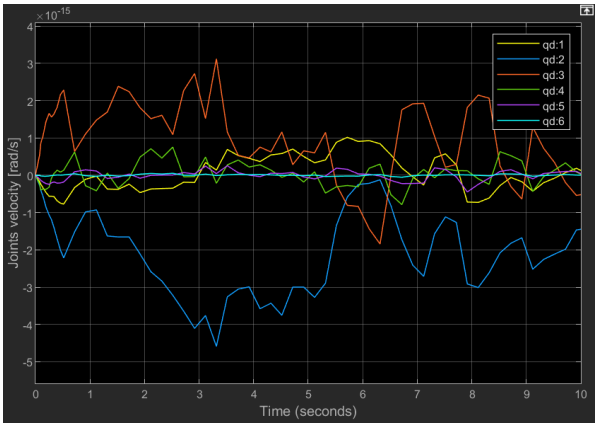

Figure 1c - Joints Velocity

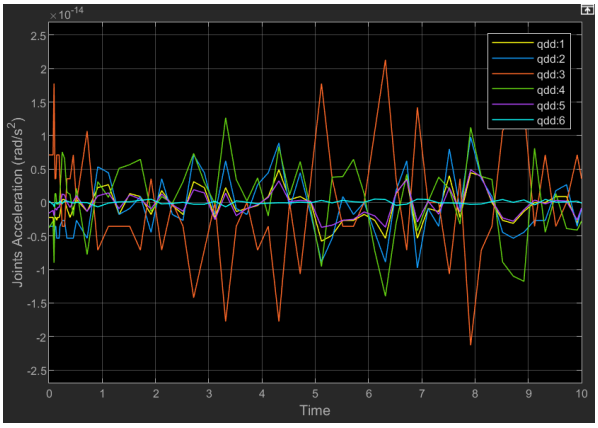

Figure 1d - Joints Acceleration

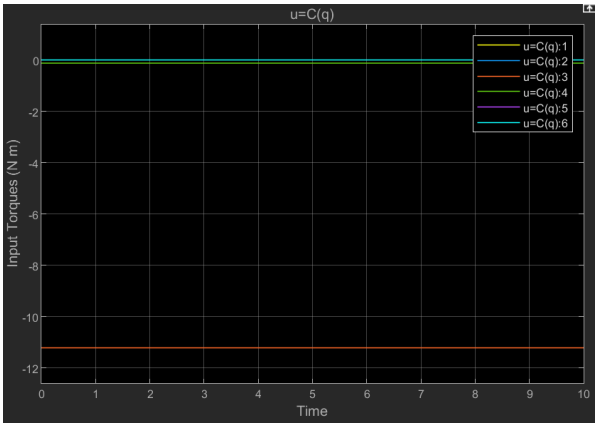

Figure 1e - Input Torques

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], the aim of this exercise was to provide torque commands in order to reach q*. This task can be easily achieved using the *Set Point Regulation,* which is a particular case of the *Joint Space Trajectory Tracking*. In this case, it was sufficient to implement a simple PD controller + Gravity Compensation (if any) which is described by the following control law:

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

where:

- K_v is a diagonal matrix of velocity (derivative) gains consisting of elements, $k_{v_i} > 0$, the velocity gain for the i-th joint $\forall i = 1 \div n$.
- K_p is a diagonal matrix of position (proportional) gains consisting of elements, $k_{p_i} > 0$, the position gain for the i-th joint $\forall i = 1 \div n$.
- $\delta\mathbf{q} = (\mathbf{q} - \mathbf{q}^*)$, the control positioning error
- $\mathbf{C}(\mathbf{q})$ represents the gravity contribute.

Please note that $\mathbf{C}(\mathbf{q})$ is omitted in the case we don't want to compensate gravity.

The chosen derivative and proportional gains are, respectively, $k_{v_i} = 30$ and $k_{p_i} = 120, \forall i = 1 \div n$.

The implemented Simulink model is the following:

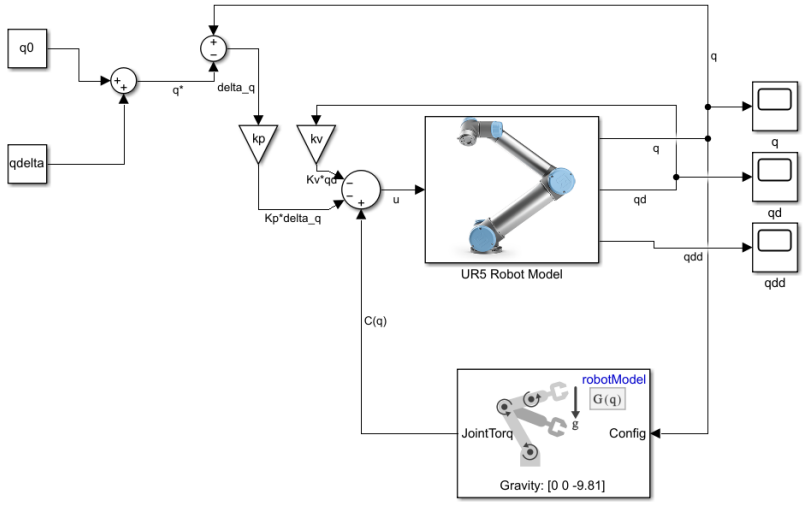

Figure 2a - Linear Joint Control model

Here you can see the plots representing the joints position, velocity, the control error and the torque, both for cases with and without gravity compensation.

Case with gravity compensation

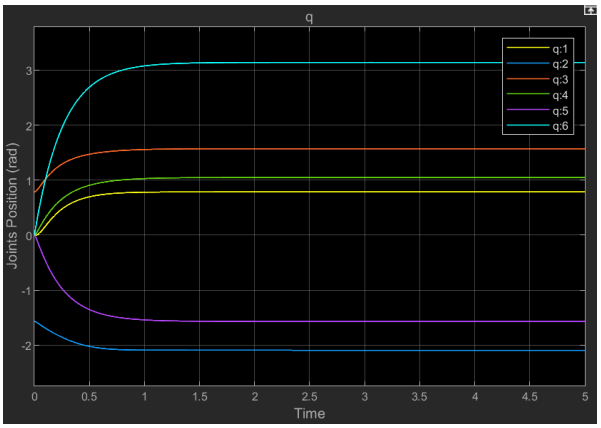

Figure 2b - Joints Position

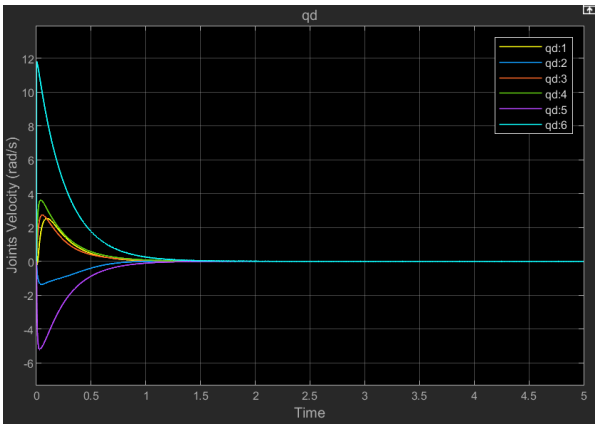

Figure 2c - Joints Velocity

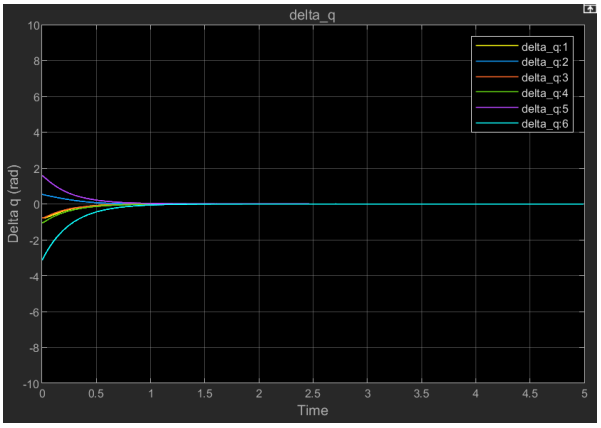

Figure 2d - Control Error

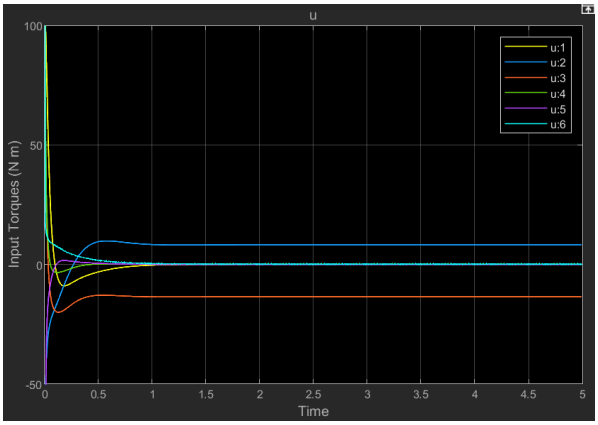

Figure 2e - Input Torques

Case without gravity compensation

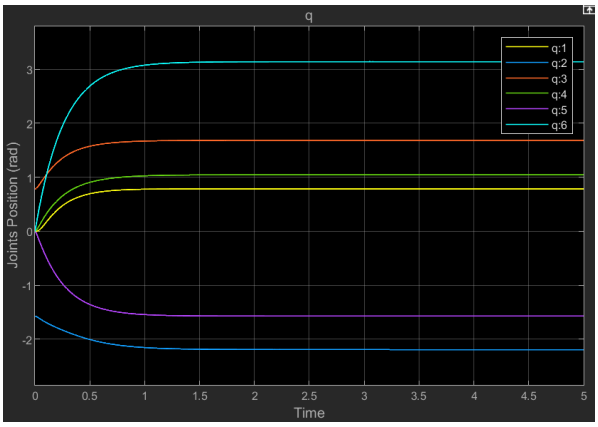

Figure 2f - Joints Position

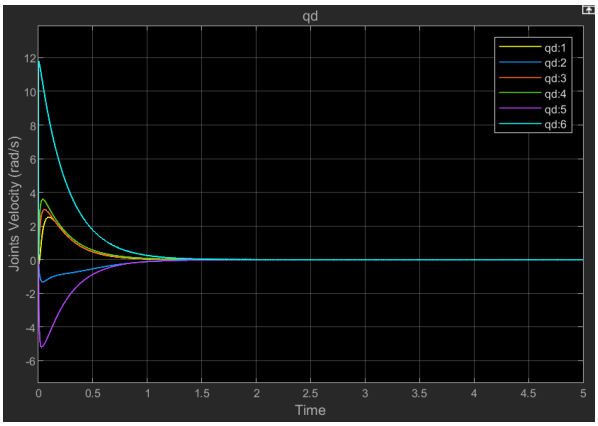

Figure 2g - Joints Velocity

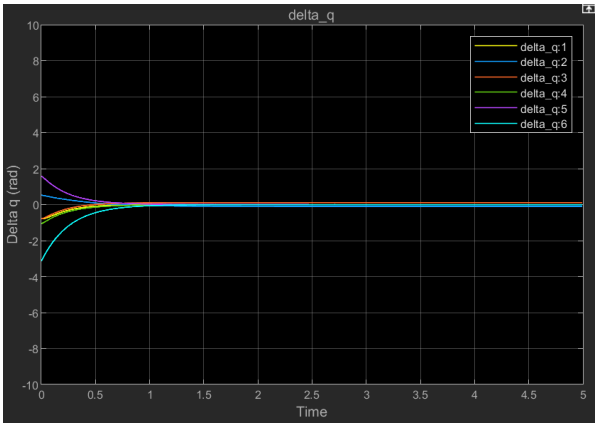

Figure 2h - Control Error

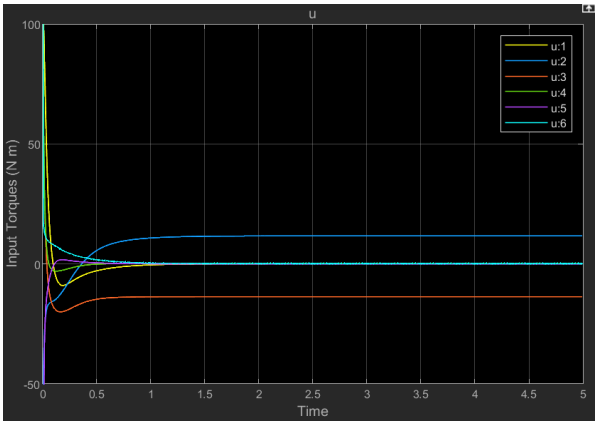

Figure 2i - Input Torques

Comments and differences between the two cases

As a result of the simulation, the robot correctly reaches the desired configuration. Interestingly, also in the case without gravity compensation the robot mantains the desired configuration over the time. The reason is that the set values for K_p (”eye(n)*120”) are high enough to counter the effect of gravity. Indeed, from theory we know that the control positioning error we can expect if we decide to use a purely PD controller without gravity compensation is given by the following formula:

$$||\delta x|| = \left|\left| K_p^{-1} \right|\right| \mathbf{C_0}$$

where $\mathbf{C_0}$ is the gravity block $\mathbf{C(q)}$ in its least position, in order to increase the equation.

It is sufficient to decrease the value of the gains in order to verify a different behaviour of the robot, which now reaches the desired configuration and subsequently falls due to the effect of gravity.

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], the aim of this exercise was to provide torque commands in order to reach x*. Since we have a 6 DoF manipulator, we know that the resulting joint configuration for a given desired cartesian configuration is unically identified. As first thing, I retrieved the transformation matrix between the base and the end effector in the initial configuration using **GetTransform** block. After that, I used **Coordinate Transformation Conversion** block to convert the matrix in Euler Angles coordinates representation. The output of this block is composed by two 3x1 vectors which represents the Distance vector and the Euler Angles vector. Now it is possible to add to these vectors respectively the first three elements of Δx (i.e the linear displacements) and last three ones (i.e the angular displacements). To get the new desired joints configuration it is sufficient to use **Inverse Kinematics** block. From now on, the exercise is similar to the previous one, the control

law is the same, each term has been already introduced in the exercise 2 and the values set for K_v and K_p matrices are also the same ($k_{v_i} = 30$ and $k_{p_i} = 120, \forall i = 1 \div n$):

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

Here's the Simulink model:

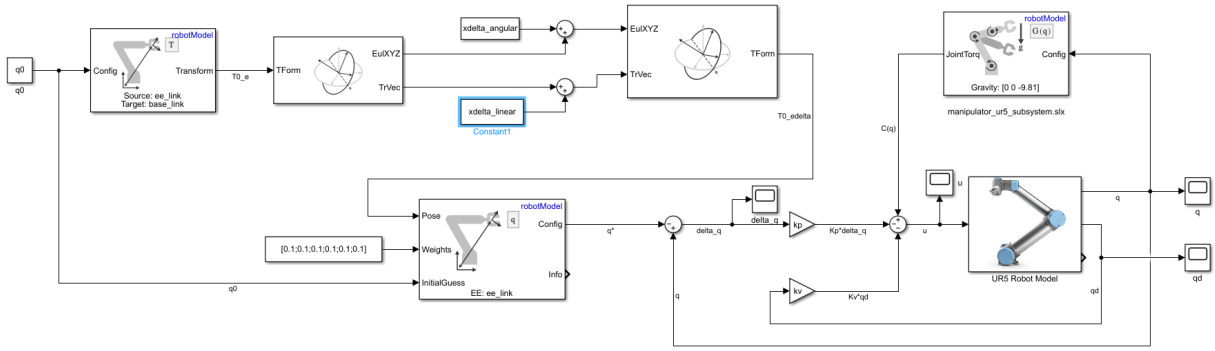

Figure 3a - Linear Cartesian Control model

Here you can see the plots representing the joints position, velocity, the control error and the torque:

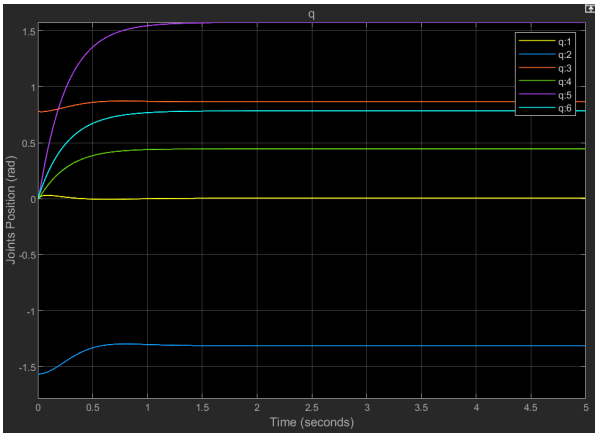

Figure 3b - Joints Position

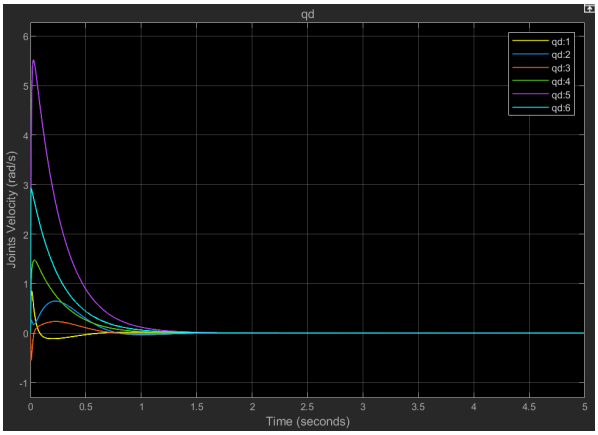

Figure 3c - Joints Velocity

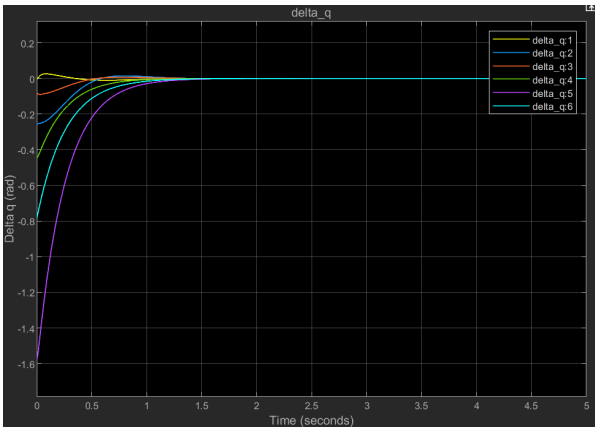

Figure 3d - Control Error

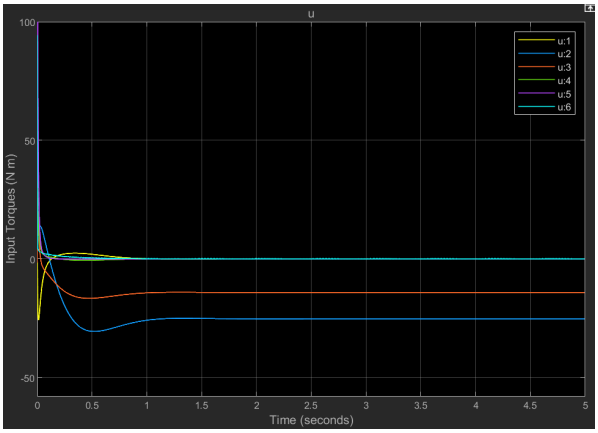

Figure 3e - Input Torques

Comments

The control law is the same as the previous exercise, the only difference is that the torque commands are now given in Cartesian Space.

Exercise 4 - Computed Torque Control

The aim of this exercise was to generate joint-space trajectories in order to reach it from q_0, considering again q^* from Exercise 2 and using feedback linearization to implement a computed torque control in order to track the desired joint trajectories.

In the Joint Space Trajectory Tracking problem the desired joints' position, speed and acceleration are now time-varying: $\mathbf{q}^* = \mathbf{q}^*(t)$, $\dot{\mathbf{q}}^* = \dot{\mathbf{q}}^*(t)$, $\ddot{\mathbf{q}}^* = \ddot{\mathbf{q}}^*(t)$.

The control law is:

$$\mathbf{u} = \mathbf{A}(\mathbf{q})[\ddot{\mathbf{q}}^* - K_v\delta\dot{\mathbf{q}} - K_p\delta\mathbf{q}] + \mathbf{B}(\mathbf{q},\dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{C}(\mathbf{q})$$

where the terms that have not already been introduced are:

- $\ddot{\mathbf{q}}^*$, nx1 vector of desired joints acceleration (rad/s^2),
- $\delta\dot{\mathbf{q}} = (\dot{\mathbf{q}} - \dot{\mathbf{q}}^*)$, the control velocity error(rad/s),

The set values for K_v and K_p matrices are the same as in the previous exercises ($k_{v_i} = 30$ and $k_{p_i} = 120$, $\forall i = 1 \div n$).

I used **Trapezoidal Velocity Profile Trajectory** block to generate trajectories through multiple waypoints using trapezoidal velocity profiles. The desired velocity envelope increases to a constant value and then drecreases to zero. As a result, the acceleration is constant piecewise, and the configuration $\mathbf{q}^*$ becomes quadratic during the acceleration and deceleration phases, and linear in between. The benefits of this type of velocity profile are that you can easily set a maximum limit that you want to enforce and the position will never overshoot its target. These are the primary reasons why this technique is commonly used in industrial safety applications.

The set waypoints inside the block are the initial configuration q_0 and the final one $q_0 + \Delta q$. The matrices $\mathbf{A}(\mathbf{q})$, $\mathbf{B}(\mathbf{q}, \dot{\mathbf{q}})$ and the vector $\mathbf{C}(\mathbf{q})$ are provided by, respectively, **Joint Space Mass Matrix** block, **Velocity Product Torque** block and **Gravity Torque** block. From now on, the implementation of the model is quite straight-forward since it's just a matter of properly connecting all the blocks.

Here's the Simulink model:

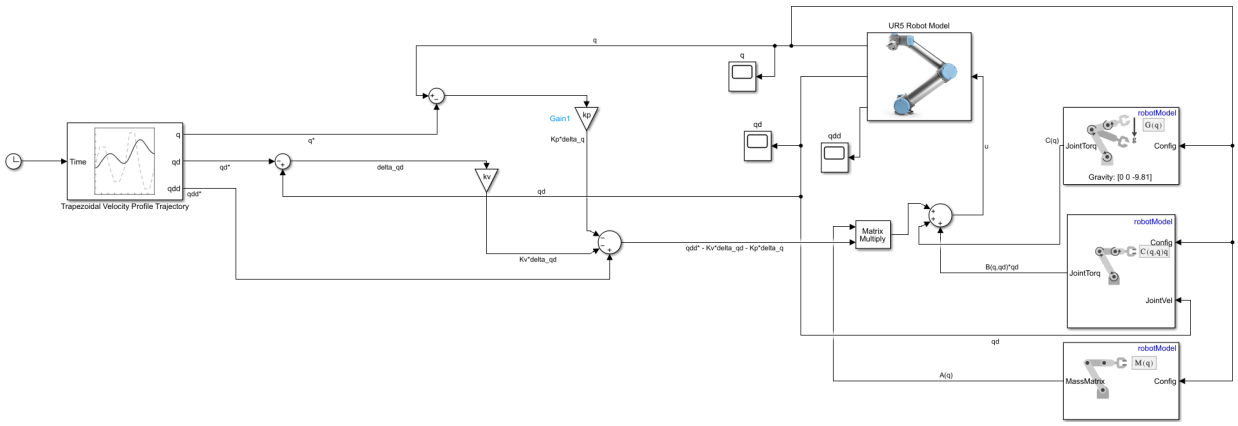

Figure 4a - Computed Torque Control model

Here you can see the plots representing the joints position, velocity, the actual and the desired joints' acceleration, the control (positioning and velocity) error and the torques:

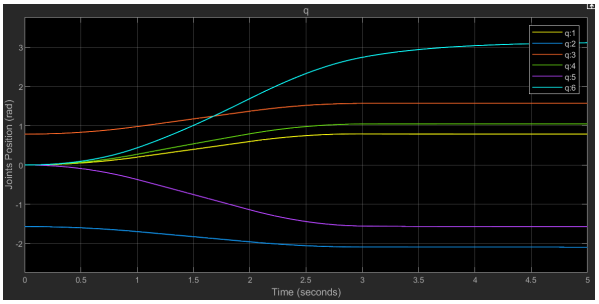

Figure 4b - Joints Position

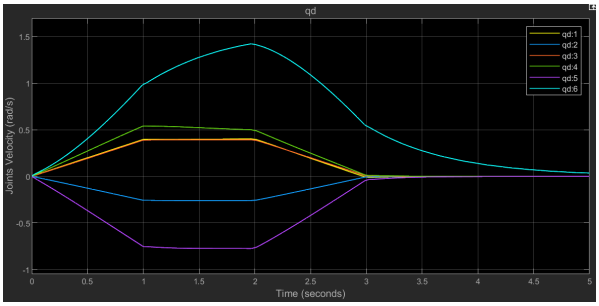

Figure 4c - Joints Velocity

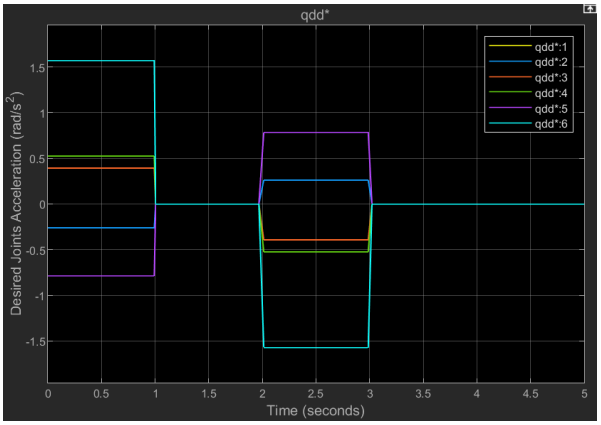

Figure 4f - Desired Joints Acceleration

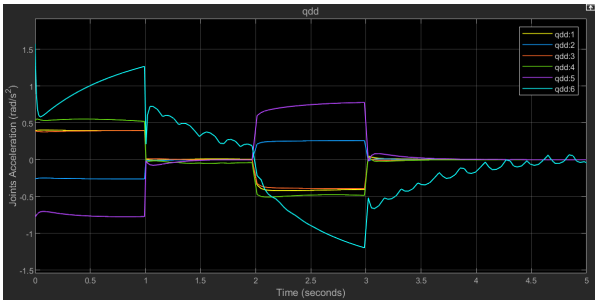

Figure 4g - Joints Acceleration

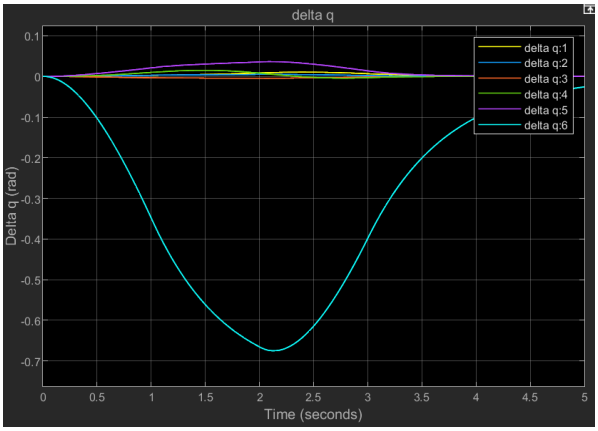

Figure 4d - Control Error (Position)

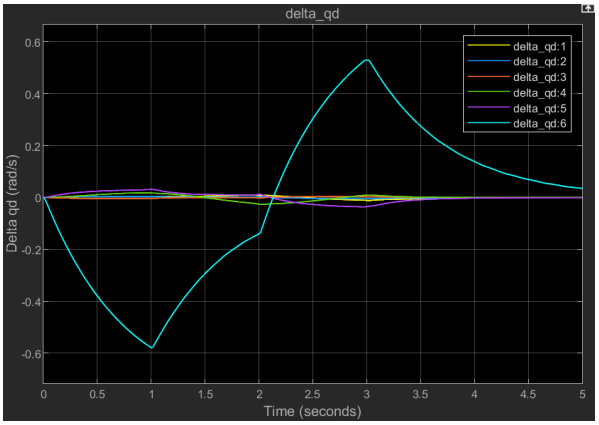

Figure 4e - Control Error (Velocity)

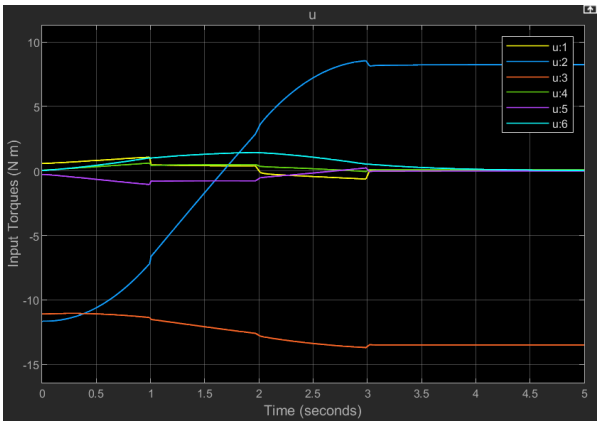

Figure 4g - Input Torques

What if we remove the joint friction?

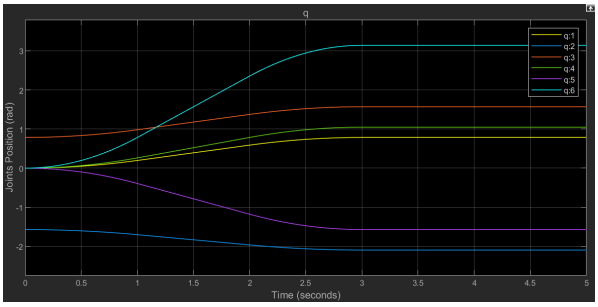

Figure 4i - Joints Position

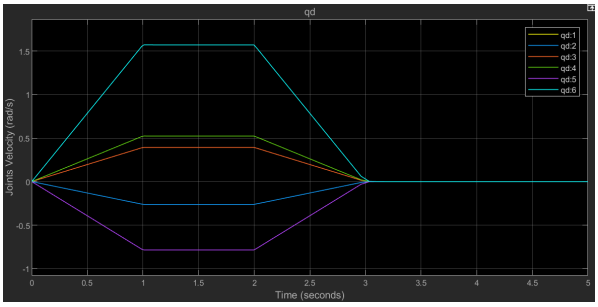

Figure 4l - Joints Velocity

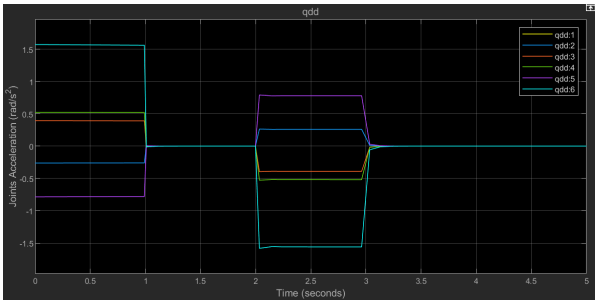

Figure 4m - Joints Acceleration

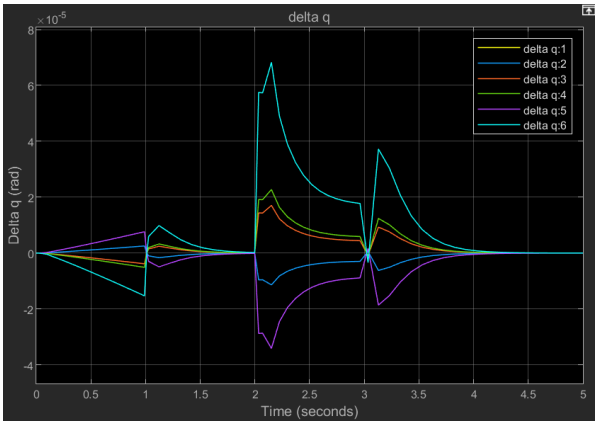

Figure 4n - Control Error (Position)

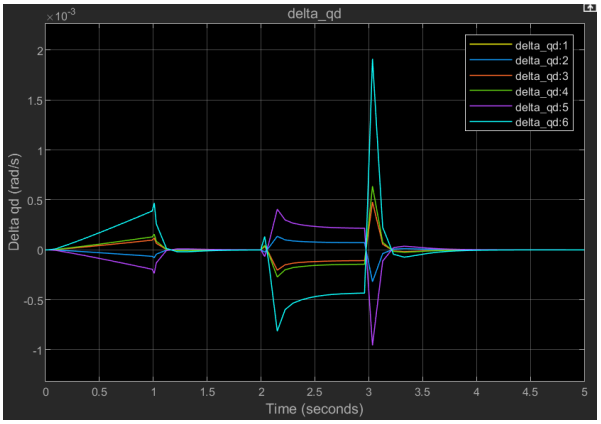

Figure 4o - Control Error (Velocity)

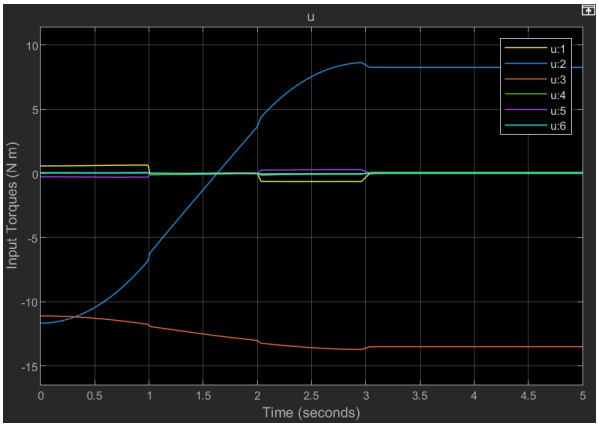

Figure 4p - Input Torques

Comments and comparison of the performances between the two cases

In the Trapezoidal Velocity Profile block the set waypoints are simply the initial position and the desired one, since we are not interested in reaching any particular point in the inbetween.

The plots related to the joints velocity and acceleration are quite different in the two cases. More precisely, the manipulator seems following the profile of the desired velocity and acceleration in a more efficient way. This is because the friction between the joint converts the kinetic energy into thermal energy and therefore a part of the total energy is wasted. In the friction-less case, the control positioning and velocity errors (Fig. 4n, 4o) are generally much smaller than the case with friction because the robot's transient behaviour is very less significative now.