Master's degree in Computer Engineering for Robotics and Smart Industry

Advanced Control Systems – course long project

Academic year: 2021-2022

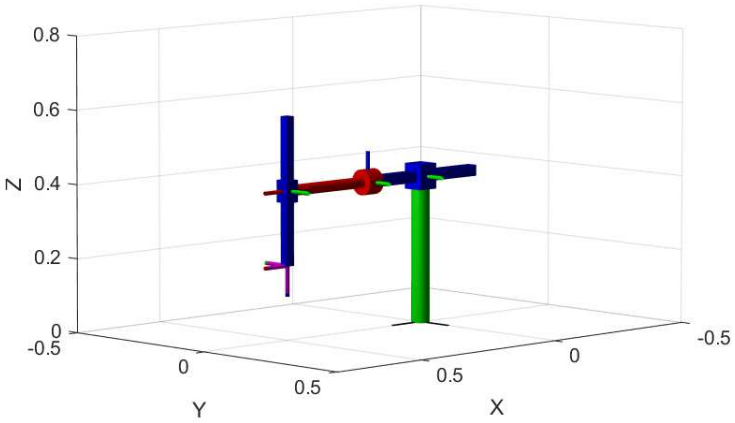
Author: Fabio Castellini (VR464639)

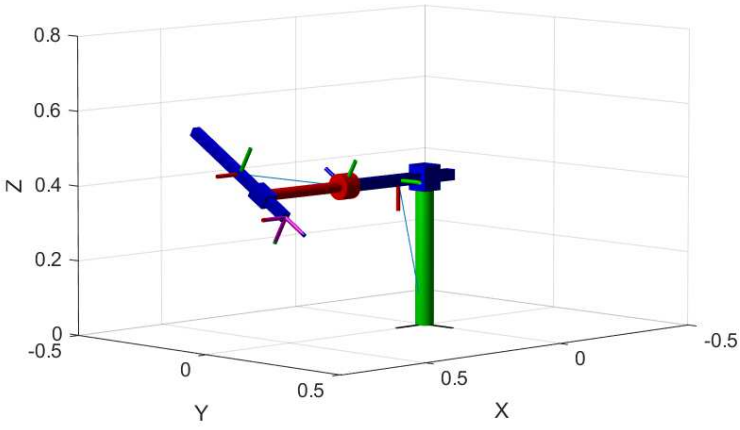
Assignment 1

Given a **3 d**egrees **o**f **f**reedom manipulator, a **P**rismatic – **R**evolute – **P**rismatic configuration in my case, several kinematics properties were computed. Some important remarks can be made:

- The MATLAB visualization of the manipulator uses an URDF file, whose frames do not follow the Denavit Hartenberg convention. Knowing that modifying an URDF file to match the DH convention isn't always feasible, it was kept as given.
- Despite that, while creating the DH table, *base* and *end-effector* frames were chosen coincident with the URDF's ones. This was done, first of all because, in my case, the DH convention allows that and also because it's a convenient strategy that allows to use some of the MATLAB Robotics-Toolbox functions.
- Finally, all the transformation matrices were both computed from Σ_{base} and from Σ_0 with the ability inside the code to choose which one to retrieve. Only transformations and Jacobians from Σ_0 will be used in all the next assignments, though.

- **URDF visualization of the PRP robot in rest position and in a particular pose ($d_1 = 0.1;\ \theta_2 = {}^{\pi}/_{3};\ d_3 = 0.1$):**





$$\text{DH} - \text{table}_{\sum base} = \begin{pmatrix} 0 & \frac{\pi}{2} & \text{L}_0 & \frac{\pi}{2} \\ 0 & 0 & \frac{\text{L}_2}{2} + \text{d}_1(\text{t}) & 0 \\ 0 & -\frac{\pi}{2} & \text{L}_3 & \theta_2(\text{t}) \\ 0 & -\pi & \text{d}_3(\text{t}) - \frac{\text{L}_4}{2} & -\frac{\pi}{2} \end{pmatrix} \qquad DH - table_{\sum 0} = \begin{pmatrix} 0 & 0 & \frac{L_2}{2} + d_1(t) & 0 \\ 0 & -\frac{\pi}{2} & L_3 & \theta_2(t) \\ 0 & -\pi & d_3(t) - \frac{L_4}{2} & -\frac{\pi}{2} \end{pmatrix}$$

- **Computation of the Direct/Inverse Kinematics w.r.t. Σ_0**

$$T_1^0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{L_2}{2} + d_1(t) \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad T_2^0 = \begin{pmatrix} \cos(\theta_2(t)) & 0 & -\sin(\theta_2(t)) & 0 \\ \sin(\theta_2(t)) & 0 & \cos(\theta_2(t)) & 0 \\ 0 & -1 & 0 & \frac{L_2}{2} + L_3 + d_1(t) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_3^0 = T_{ee}^0 = \begin{pmatrix} 0 & -\cos(\theta_2(t)) & \sin(\theta_2(t)) & \sin(\theta_2(t)) \left(\frac{L_4}{2} - d_3(t)\right) \\ 0 & -\sin(\theta_2(t)) & -\cos(\theta_2(t)) & -\cos(\theta_2(t)) \left(\frac{L_4}{2} - d_3(t)\right) \\ 1 & 0 & 0 & \frac{L_2}{2} + L_3 + d_1(t) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$d_1(t) = p_z - L_3 - \frac{L_2}{2} \qquad theta_2(t) = 2\,\mathrm{atan}\left(\frac{p_y \pm \sqrt{{p_x}^2 + {p_y}^2}}{p_x}\right)$$

$$d_3(t) = \frac{L_4}{2} \mp \sqrt{{p_x}^2 + {p_y}^2}$$

- **Computation of the Direct/Inverse Kinematics w.r.t. $\Sigma_{\mathbf{base}}$**

$$T_0^{base} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & L_0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad T_1^{base} = \begin{pmatrix} 0 & 0 & 1 & \frac{L_2}{2} + d_1(t) \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & L_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_2^{base} = \begin{pmatrix} 0 & -1 & 0 & \frac{L_2}{2} + L_3 + d_1(t) \\ \cos(\theta_2(t)) & 0 & -\sin(\theta_2(t)) & 0 \\ \sin(\theta_2(t)) & 0 & \cos(\theta_2(t)) & L_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_3^{base} = T_{ee}^{base} \begin{pmatrix} 1 & 0 & 0 & \frac{L_2}{2} + L_3 + d_1(t) \\ 0 & -\cos(\theta_2(t)) & \sin(\theta_2(t)) & \sin(\theta_2(t)) \left(\frac{L_4}{2} - d_3(t)\right) \\ 0 & -\sin(\theta_2(t)) & -\cos(\theta_2(t)) & L_0 + \cos(\theta_2(t))\, d_3(t) - \frac{L_4\, \cos(\theta_2(t))}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$d_1(t) = p_x - L_3 - \frac{L_2}{2} \qquad theta_2(t) = \mp 2\,\mathrm{atan}\left(\frac{L_0 - p_z + \sqrt{{L_0}^2 - 2\,L_0\,p_z + {p_y}^2 + {p_z}^2}}{p_y}\right)$$

$$d_3(t) = \frac{L_4}{2} \pm \sqrt{{L_0}^2 - 2\,L_0\,p_z + {p_y}^2 + {p_z}^2}$$

- **Computation of the Analytical Jacobian w.r.t. Σ_0**

$$J_a = \begin{pmatrix} 0 & \cos(\theta_2(t))\left(\frac{L_4}{2} - d_3(t)\right) & -\sin(\theta_2(t)) \\ 0 & \sin(\theta_2(t))\left(\frac{L_4}{2} - d_3(t)\right) & \cos(\theta_2(t)) \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- **Computation of the Analytical Jacobian w.r.t. Σ_{base}**

$$J_a = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_2(t))\left(\frac{L_4}{2} - d_3(t)\right) & -\sin(\theta_2(t)) \\ 0 & \sin(\theta_2(t))\left(\frac{L_4}{2} - d_3(t)\right) & \cos(\theta_2(t)) \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

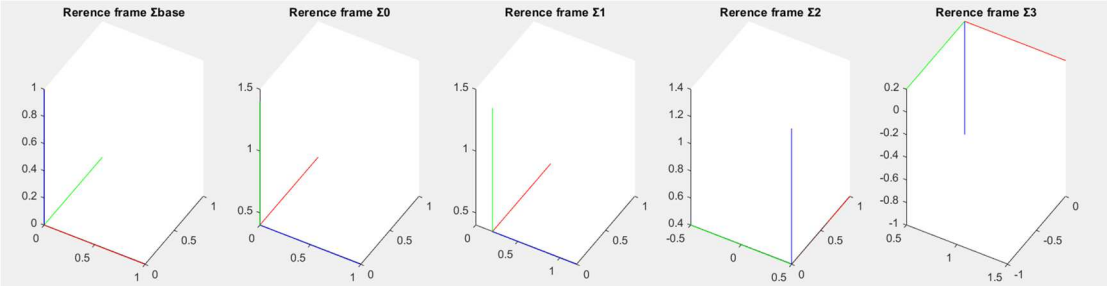
- **Computation of the Geometric Jacobian w.r.t. Σ_0**

$$J_g = \begin{pmatrix} 0 & \cos(\theta_2(t))\left(\frac{L_4}{2} - d_3(t)\right) & -\sin(\theta_2(t)) \\ 0 & \sin(\theta_2(t))\left(\frac{L_4}{2} - d_3(t)\right) & \cos(\theta_2(t)) \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

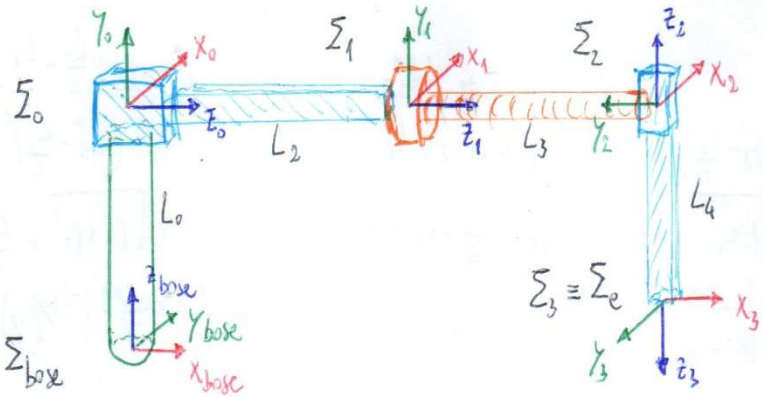
- **Computation of the Geometric Jacobian w.r.t. Σ_{base}**

$$J_g = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_2(t))\left(\frac{L_4}{2} - d_3(t)\right) & -\sin(\theta_2(t)) \\ 0 & \sin(\theta_2(t))\left(\frac{L_4}{2} - d_3(t)\right) & \cos(\theta_2(t)) \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Checking the chosen frames with the "plotTransforms" MATLAB function:



ASSIGNMENT 1: DH table, Direct/Inverse Kinematics, Jacobians



PRP robot

DH table from Σ_{base}:

$(i-1) \to i$	a_{i-1}	α_{i-1}	d_i	θ_i	
base $\to$ 0	0	$+\pi/2$	L_0	$+\pi/2$	→ translation + rotation now that is fixed.
$0 \to 1$	0	0	$d_1^* + L_2/2$	0	(P)
$1 \to 2$	0	$-\pi/2$	L_3	θ_2^*	(R)
$2 \to 3$	0	$-\pi$	$d_3^* - L_4/2$	$-\pi/2$	(P)

→ DH table from Σ_0 (without fixed transformation)

Direct kinematics:

$$A_i^{i-1} = \begin{pmatrix} c_i & -s_i c_\alpha & s_i s_\alpha & a_i c_i \\ s_i & c_i c_\alpha & -c_i s_\alpha & a_i s_i \\ 0 & s_\alpha & c_\alpha & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

From Σ_{base}:

$$A_0^{base} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & L_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \quad A_1^{base} = \begin{pmatrix} 0 & 0 & 1 & d_1^* + \frac{L_2}{2} \\ 1 & 0 & 0 & 0 \\ 0 & +1 & 0 & L_0 \\ 0 & 0 & 0 & 1 \end{pmatrix};$$

$$A_2^{base} = \begin{pmatrix} 0 & -1 & 0 & L_3 + d_1^* + L_2/2 \\ +c_2 & 0 & -s_2 & 0 \\ +s_2 & 0 & c_2 & L_0 \\ 0 & 0 & 0 & 1 \end{pmatrix};$$

$$A_3^{base} = A_e^{base} = \begin{pmatrix} 1 & 0 & 0 & L_3 + d_1^* + L_2/2 \\ 0 & -c_2 & s_2 & -s_2(d_3^* - L_4/2) \\ 0 & -s_2 & -c_2 & L_0 + c_2(d_3^* - L_4/2) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

From Σ_0:

$$A_1^0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_1^* + L_2/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_2^0 = \begin{pmatrix} c_2 & 0 & -s_2 & 0 \\ s_2 & 0 & +c_2 & 0 \\ 0 & -1 & 0 & L_2/2 + L_3 + d_1^* \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_3^0 = A_e^0 = \begin{pmatrix} 0 & -c_2 & +s_2 & -s_2(d_3^* - L_4/2) \\ 0 & -s_2 & -c_2 & +c_2(d_3^* - L_4/2) \\ 1 & 0 & 0 & L_2/2 + L_3 + d_1^* \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Inverse Kinematics

From Σ_{base}

$$p_x = L_3 + d_1^* + L_2/2$$
$$p_y = -s_2\left(d_3^* - L_4/2\right)$$
$$p_z = L_0 + c_2\left(d_3^* - L_4/2\right)$$

- $d_1^* = p_x - L_3 - \frac{L_2}{2}$
- $(p_z - L_1)^2 + p_y^2 = \left(d_3^* - \frac{L_4}{2}\right)^2$
  - $d_3^* = \pm\sqrt{(p_z - L_0)^2 + p_y^2} + \frac{L_4}{2}$
- $\vartheta_2^* = \operatorname{arctg}_2\left(-\frac{p_y}{p_z - L_0}\right)$

From Σ_0

$$p_x = -s_2\left(d_3^* - L_4/2\right)$$
$$p_y = +c_2\left(d_3^* - L_4/2\right)$$
$$p_z = \frac{L_2}{2} + L_3 + d_1^*$$

- $d_1^* = p_z - \frac{L_2}{2} - L_3$
- $p_x^2 + p_y^2 = \left(d_3^* - \frac{L_4}{2}\right)^2$
  - $d_3^* = \pm\sqrt{p_x^2 + p_y^2} + \frac{L_4}{2}$
- $\vartheta_2^* = \operatorname{arctg}_2\left(-p_x/p_y\right)$

Analytical Jacobian (deriving p_{ee})

From Σ_{base}:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -c_2\left(d_3^* - \frac{L_4}{2}\right) & -s_2 \\ 0 & -s_2\left(d_3^* - \frac{L_4}{2}\right) & c_2 \end{pmatrix}$$

From Σ_0

$$\begin{pmatrix} 0 & -c_2\left(d_3^* - \frac{L_4}{2}\right) & -s_2 \\ 0 & -s_2\left(d_3^* - L_4/2\right) & c_2 \\ 1 & 0 & 0 \end{pmatrix}$$

Geometric Jacobian

From Σ_{base}

$$z_{base} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix};\quad z_0 = R_0^{base}\underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{z_{base}} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix};\quad z_1 = R_1^{base}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix};\quad z_2 = R_2^{base}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -s_2 \\ c_2 \end{bmatrix}$$

$$J_G = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & -c_2\left(d_3^* - L_4/2\right) & -s_2 \\ 0 & -s_2\left(d_3^* - L_4/2\right) & c_2 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{array}\right] \begin{matrix} \\ \\ \\ \\ \\ \\ P \end{matrix} \quad \begin{matrix} \\ \\ \\ \\ \\ \\ R \end{matrix} \quad \begin{matrix} \\ \\ \\ \\ \\ \\ P \end{matrix}$$

$$z_1 \times (p_e - p_1) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} L_3 \\ -s_2\left(d_3^* - L_4/2\right) \\ c_2\left(d_3^* - L_4/2\right) \end{pmatrix}$$

$$\begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}$$

From Σ_0

$$z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix};\quad z_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix};\quad z_2 = \begin{bmatrix} -s_2 \\ c_2 \\ 0 \end{bmatrix};$$

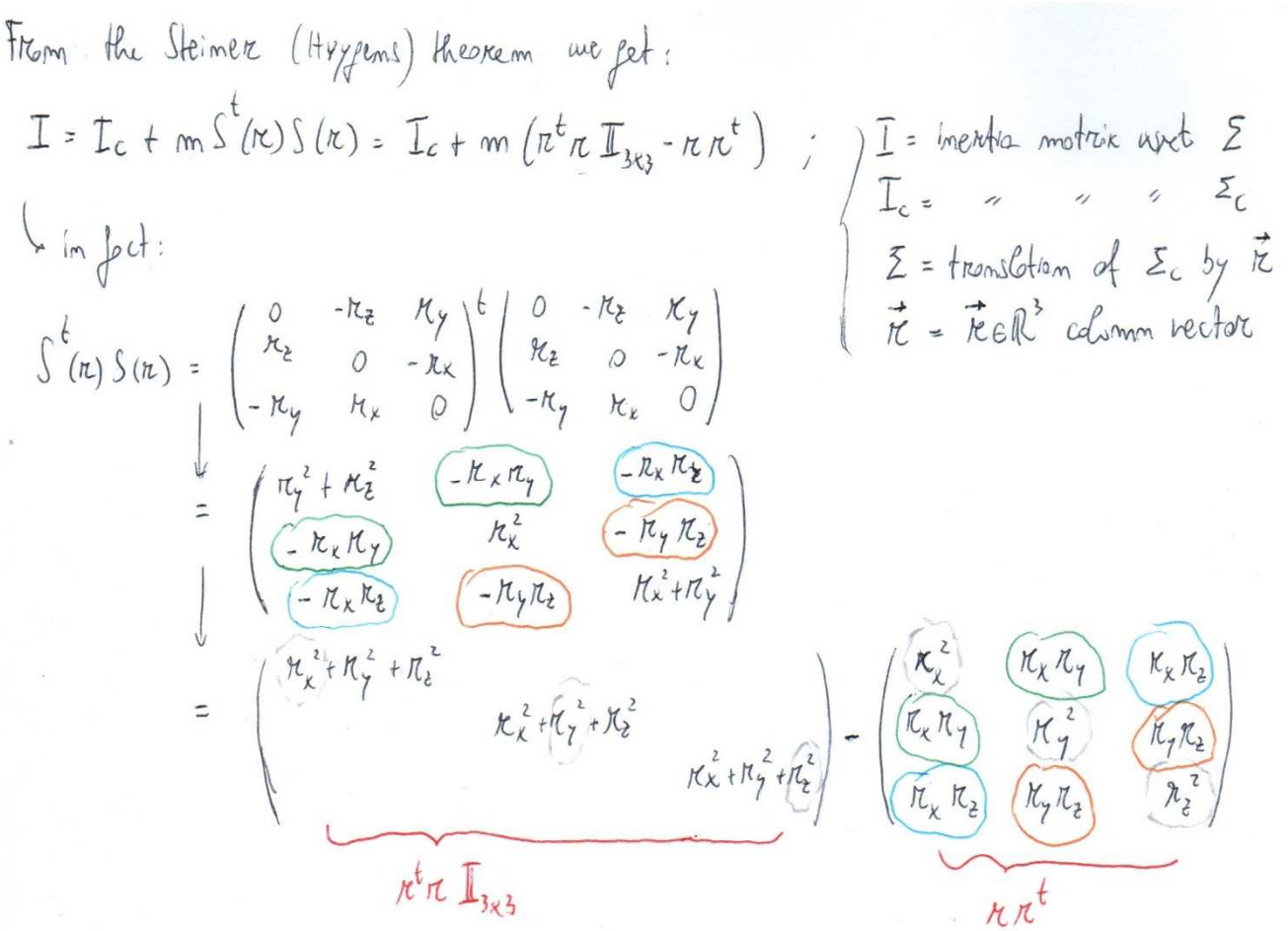
$$J_G = \left[\begin{array}{ccc} 0 & -c_2\left(d_3^* - L_4/2\right) & -s_2 \\ 0 & -s_2\left(d_3^* - L_4/2\right) & c_2 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{array}\right] \begin{matrix} \\ \\ \\ \\ \\ \\ P \end{matrix} \quad \begin{matrix} \\ \\ \\ \\ \\ \\ R \end{matrix} \quad \begin{matrix} \\ \\ \\ \\ \\ \\ P \end{matrix}$$

Assignment 2

- **Proof of the following equation**

Homework. Prove

$$I_C + mS^T(r)S(r) = I_C + m\left(r^T r\, I_{3\times 3} - rr^T\right)$$



- **Brief introduction about the Kinetic Energy computation**

To compute the ***kinetic energy*** the *rigid links* and *rigid transmission* assumptions are made. Moreover, the contributions relative to the motion of each joint motor actuator is neglected, meaning that only T_{L_i} is considered (kinetic energy relative to the motion of link i).

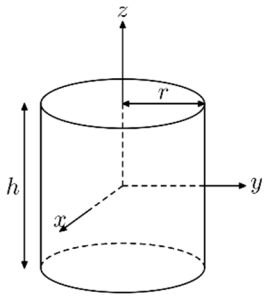
$$\begin{aligned} T_{\ell_i} = \frac{1}{2}\int_{V_{\ell_i}} (\dot{p}_i^*)^T \dot{p}_i^* \rho dV &= \frac{1}{2} m_{\ell_i} \dot{p}_{\ell_i}^T \dot{p}_{\ell_i} + \frac{1}{2}\omega_i^T I_{\ell_i} \omega_i \\ &= \frac{1}{2} m_{\ell_i} \dot{p}_{\ell_i}^T \dot{p}_{\ell_i} + \frac{1}{2}\omega_i^T R_i I_{\ell_i}^i R_i^T \omega_i \end{aligned}$$

T_{L_i} has been rewritten in such a way, to highlight the configuration-independent terms and simplify the inertia tensors computation. The above equation is also strictly related to the ***König's theorem*** (*the kinetic energy of a not necessarily rigid system of particles, is given by the contribution of the center of mass's motion and the movement of particles relative to the center of mass*).

- **Computation of the baricentral Inertia tensors**

  Source: https://it.wikipedia.org/wiki/Lista_dei_momenti_di_inerzia

  Full cylinder:



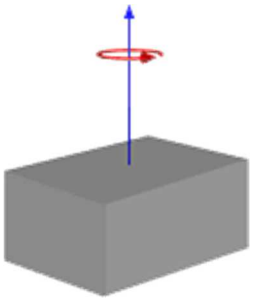
$$I_x = I_y = \frac{1}{12} m \left(3r^2 + h^2\right)$$

$$I_z = \frac{mr^2}{2}$$

  In my case (cylindric link 3):

$$Ic_{Link_3} = \begin{pmatrix} m_{\text{Link3}} \left(\frac{1}{4} a^2 + \frac{1}{12} h^2\right) & 0 & 0 \\ 0 & m_{\text{Link3}} \left(\frac{1}{4} a^2 + \frac{1}{12} h^2\right) & 0 \\ 0 & 0 & \frac{1}{2} a^2 m_{\text{Link3}} \end{pmatrix}$$

  Parallelepiped:



$$I_h = \frac{1}{12} m \left(w^2 + d^2\right)$$

$$I_w = \frac{1}{12} m \left(h^2 + d^2\right)$$

$$I_d = \frac{1}{12} m \left(h^2 + w^2\right)$$

  In my case (prismatic links 2 and 4):

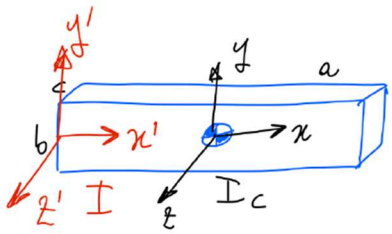
$$Ic_{Link_2} = \begin{pmatrix} \frac{1}{12} m_{\text{Link2}} (a^2 + b^2) & 0 & 0 \\ 0 & \frac{1}{12} m_{\text{Link2}} (a^2 + c^2) & 0 \\ 0 & 0 & \frac{1}{12} m_{\text{Link2}} (b^2 + c^2) \end{pmatrix}$$

$$Ic_{Link_4} = \begin{pmatrix} \frac{1}{12} m_{\text{Link4}} (a^2 + b^2) & 0 & 0 \\ 0 & \frac{1}{12} m_{\text{Link4}} (a^2 + c^2) & 0 \\ 0 & 0 & \frac{1}{12} m_{\text{Link4}} (b^2 + c^2) \end{pmatrix}$$

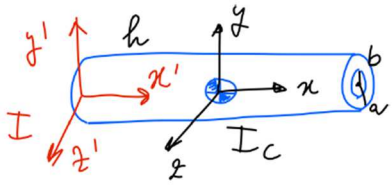
- **Computation of the Inertia tensors w.r.t. the reference frame with origin at the start of the link**

  Using the Huygens-Steiner theorem, it's possible to compute the inertia matrix with respect to a reference frame, obtained by only translating the baricentral one.

  The following formulas were used, keeping in mind that ***in my case, the translation isn't along x but along z*** (as imposed by the DH convention placement of the reference frames)



$$I = I_C + m \left(\begin{bmatrix} -\frac{a}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} -\frac{a}{2} \\ 0 \\ 0 \end{bmatrix} I_{3\times 3} - \begin{bmatrix} -\frac{a}{2} \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} -\frac{a}{2} \\ 0 \\ 0 \end{bmatrix}^T \right)$$



$$I = I_C + m \left(\begin{bmatrix} -\frac{h}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} -\frac{h}{2} \\ 0 \\ 0 \end{bmatrix} I_{3\times 3} - \begin{bmatrix} -\frac{h}{2} \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} -\frac{h}{2} \\ 0 \\ 0 \end{bmatrix}^T \right)$$

$$I_2 = \text{Ic}_{\text{Link2}} + \begin{pmatrix} \frac{1}{4} m_{\text{Link2}}\, a^2 & 0 & 0 \\ 0 & \frac{1}{4} m_{\text{Link2}}\, a^2 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad I_3 = \text{Ic}_{\text{Link3}} + \begin{pmatrix} \frac{1}{4} m_{\text{Link3}}\, h^2 & 0 & 0 \\ 0 & \frac{1}{4} m_{\text{Link3}}\, h^2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$I_4 = \text{Ic}_{\text{Link4}} + \begin{pmatrix} \frac{1}{4} m_{\text{Link4}}\, a^2 & 0 & 0 \\ 0 & \frac{1}{4} m_{\text{Link4}}\, a^2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- **Computation of the Partial Jacobians**

Following the theory, *Partial Jacobians* are needed to compute linear and angular velocities of the center of mass of each link, with respect to Σ_0.

$$\dot{p}_{\ell_i} = J_P^{\ell_i}(q)\dot{q}, \qquad \omega_i = J_O^{\ell_i}(q)\dot{q}$$

Partial Jacobians depend on the manipulator's configuration and are computed as the geometric Jacobian, referring to intermediate reference frames (not only the end effector's reference frame). Linear (J_P) and angular (J_O) components are separated and properly used inside the equation.

The following matrices represent the Partial Jacobians for my robot configuration (PRP):

$$J_P^{Link_2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \qquad J_O^{Link_2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$J_P^{Link_3} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \qquad J_O^{Link_3} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$J_P^{Link_4} = \begin{pmatrix} 0 & -\cos(\theta_2(t))\, d_3(t) & -\sin(\theta_2(t)) \\ 0 & -\sin(\theta_2(t))\, d_3(t) & \cos(\theta_2(t)) \\ 1 & 0 & 0 \end{pmatrix} \qquad J_O^{Link_4} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

- **Computation of the Kinetic energy**

Having all the necessary parameters, all the kinetic energies are computed, to find B(q) and the total kinetic energy of the manipulator.

$$\mathcal{T}_{\ell_i} = \frac{1}{2} m_{\ell_i} \dot{q}^T (J_P^{\ell_i})^T J_P^{\ell_i} \dot{q} + \frac{1}{2} \dot{q}^T (J_O^{\ell_i})^T R_i I_{\ell_i}^i R_i^T J_O^{\ell_i} \dot{q} \qquad \mathcal{T}(q,\dot{q}) = \frac{1}{2} \dot{q}^T B(q) \dot{q}$$

The following equations represent the 3 kinetic energies corresponding to the links:

$$T_{Link_2} = \frac{m_{\text{Link2}}\, \dot{d}_1(t)^2}{2}$$

$$T_{Link_3} = \frac{1}{2}\left(\dot{\theta}_2(t)^2 \left(m_{\text{Link3}} \left(\frac{{a_3}^2}{4} + \frac{{h_3}^2}{12} \right) - \frac{{h_3}^2\, m_{\text{Link3}}}{4} \right) + m_{\text{Link3}}\, \dot{d}_1(t)^2 \right)$$

$$T_{Link_4} = \frac{m_{\text{Link4}}}{24} \left(4{a_4}^2\, \dot{\theta}_2(t)^2 + {b_4}^2\, \dot{\theta}_2(t)^2 + 12\left(\dot{d}_1(t)^2 + \dot{d}_3(t)^2 \right) + 12 \dot{\theta}_2(t)^2\, d_3(t)^2 \right)$$

The following matrix is the total Inertia Matrix (positive defined):

$$B(q) = \begin{pmatrix} m_{\text{Link2}} + m_{\text{Link3}} + m_{\text{Link4}} & 0 & 0 \\ 0 & m_{\text{Link3}} \left(\frac{{a_3}^2}{4} - \frac{{h_3}^2}{6} \right) + m_{\text{Link4}} \left(\frac{{a_4}^2}{3} + \frac{{b_4}^2}{12} + d_3(t)^2 \right) & 0 \\ 0 & 0 & m_{\text{Link4}} \end{pmatrix}$$

- **Computation of the Potential energy**

  Using the proper formula, also the *Potential energy* is computed (neglecting joint motors' contributes and assuming rigid links and transmission).

$$\mathcal{U}_{\ell_i} = -\int_{V_{\ell_i}} g_0^T p_i^* \rho dV = -m_{\ell_i} g_0^T p_{\ell_i}, \qquad \mathcal{U} = -\sum_{i=1}^{n} m_{\ell_i} g_0^T p_{\ell_i}$$

  The following equations represent the 3 potential energies corresponding to the links:

$$U_{Link_2} = 0 \qquad U_{Link_3} = 0 \qquad U_{Link_4} = 9.81\, m_{\text{Link4}} \cos\left(\theta_2(t)\right)\, d_3(t)$$

- **Computation of the Total Kinetic and Potential energies**

$$U_{Total} = 9.81\, m_{\text{Link4}} \cos\left(\theta_2(t)\right)\, d_3(t)$$

$$T_{Total} = \frac{1}{2}\dot{\theta}_2(t)^2 \left(m_{\text{Link3}} \left(\frac{{a_3}^2}{4} + \frac{{h_3}^2}{12}\right) - \frac{{h_3}^2\, m_{\text{Link3}}}{4}\right) + \frac{1}{2} m_{\text{Link2}}\, \dot{d}_1(t)^2 + \frac{1}{2} m_{\text{Link3}}\, \dot{d}_1(t)^2 + \frac{1}{2} m_{\text{Link4}}\, \dot{d}_1(t)^2 + \frac{1}{2} m_{\text{Link4}}\, \dot{d}_3(t)^2 + \frac{1}{2} m_{\text{Link4}}\, \dot{\theta}_2(t)^2\, d_3(t)^2 + \frac{m_{\text{Link4}}\, {a_4}^2\, \dot{\theta}_2(t)^2}{6} + \frac{m_{\text{Link4}}\, {b_4}^2\, \dot{\theta}_2(t)^2}{24}$$

Assignment 3

- **Computation of the Lagrange function**

  Having previously computed (*Assignment 2*) all the necessary components (total potential and kinetic energies), using the Symbolic Toolbox, the Lagrangian function is computed.

$$\mathcal{L}(q,\dot{q}) = \mathcal{T}(q,\dot{q}) - \mathcal{U}(q) = \frac{1}{2}\dot{q}^T B(q)\dot{q} + \sum_{i=1}^{n} m_{\ell_i} g_0^T p_{\ell_i}$$

  My result:

$$\begin{aligned} L = {} & \frac{\dot{\theta}_2(t)^2 \left(m_{Link3} \left(\frac{a_3{}^2}{4} + \frac{h_3{}^2}{12} \right) - \frac{h_3{}^2\, m_{Link3}}{4} \right)}{2} + \frac{m_{Link2}\, \dot{d}_1(t)^2}{2} + \frac{m_{Link3}\, \dot{d}_1(t)^2}{2} + \frac{m_{Link4}\, \dot{d}_1(t)^2}{2} \\ & + \frac{m_{Link4}\, \dot{d}_3(t)^2}{2} + \frac{m_{Link4}\, \dot{\theta}_2(t)^2\, d_3(t)^2}{2} + \frac{a_4{}^2\, m_{Link4}\, \dot{\theta}_2(t)^2}{6} + \frac{b_4{}^2\, m_{Link4}\, \dot{\theta}_2(t)^2}{24} \\ & - \frac{981\, m_{Link4}\, \cos(\theta_2(t))\, d_3(t)}{100} \end{aligned}$$

- **Computation of the Equations of Motion**

  The Lagrange function is derived according to the theory, to retrieve the 3 (one for each degree of freedom) equations of motion.

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right)^T - \left(\frac{\partial \mathcal{L}}{\partial q}\right)^T = \tau \qquad B(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_1}\right) = m_{Link2}\, \ddot{d}_1(t) + m_{Link3}\, \ddot{d}_1(t) + m_{Link4}\, \ddot{d}_1(t)$$

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_2}\right) = {} & \frac{m_{Link4}\, \ddot{\theta}_2(t)\, a_4{}^2}{3} + \frac{m_{Link4}\, \ddot{\theta}_2(t)\, b_4{}^2}{12} + m_{Link4}\, \ddot{\theta}_2(t)\, d_3(t)^2 + 2\, m_{Link4}\, \dot{d}_3(t)\, \dot{\theta}_2(t)\, d_3(t) \\ & + \ddot{\theta}_2(t) \left(m_{Link3} \left(\frac{a_3{}^2}{4} + \frac{h_3{}^2}{12} \right) - \frac{h_3{}^2\, m_{Link3}}{4} \right) \end{aligned}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_3}\right) = m_{Link4}\, \ddot{d}_3(t)$$

$$\frac{\partial L}{\partial q_1} = 0$$

$$\frac{\partial L}{\partial q_2} = 9.81\, m_{Link4}\, \sin(\theta_2(t))\, d_3(t)$$

$$\frac{\partial L}{\partial q_3} = m_{Link4}\, d_3(t)\, \dot{\theta}_2(t)^2 - 9.81\, m_{Link4}\, \cos(\theta_2(t))$$

So, the *final torques* are:

$$\tau_1 = \ddot{d}_1(t)\,(m_{\text{Link2}} + m_{\text{Link3}} + m_{\text{Link4}})$$

$$\tau_2 = \ddot{\theta}_2(t)\left(m_{\text{Link3}}\left(\frac{{a_3}^2}{4} + \frac{{h_3}^2}{12}\right) - \frac{{h_3}^2\, m_{\text{Link3}}}{4}\right) - 9.81\, m_{\text{Link4}}\, \sin\big(\theta_2(t)\big)\, d_3(t) + \frac{{a_4}^2\, m_{\text{Link4}}\, \ddot{\theta}_2(t)}{3} + \frac{{b_4}^2\, m_{\text{Link4}}\, \ddot{\theta}_2(t)}{12} + m_{\text{Link4}}\, \ddot{\theta}_2(t)\, d_3(t)^2 + 2\, m_{\text{Link4}}\, \dot{d}_3(t)\, \dot{\theta}_2(t)\, d_3(t)$$

$$\tau_3 = -m_{\text{Link4}}\, d_3(t)\, \dot{\theta}_2(t)^2 + 9.81\, m_{\text{Link4}}\, \cos\big(\theta_2(t)\big) + m_{\text{Link4}}\, \ddot{d}_3(t)$$

- **Computing the B (*Inertia* term) matrix (*Assignment 2*)**

My result:

$$B = \begin{pmatrix} m_{\text{Link2}} + m_{\text{Link3}} + m_{\text{Link4}} & 0 & 0 \\ 0 & m_{\text{Link3}}\left(\frac{{a_3}^2}{4} - \frac{{h_3}^2}{6}\right) + \frac{m_{\text{Link4}}\left(4\,{a_4}^2 + {b_4}^2 + 12\, d_3(t)^2\right)}{12} & 0 \\ 0 & 0 & m_{\text{Link4}} \end{pmatrix}$$

- **Computing the C (*Coriolis and Centrifugal* term) matrix from B**

$$\sum_{j=1}^{n} c_{ij}(q)\dot{q}_j = \sum_{j=1}^{n}\sum_{k=1}^{n} \underbrace{\frac{1}{2}\left(\frac{\partial b_{ij}}{\partial q_k} + \frac{\partial b_{ik}}{\partial q_j} - \frac{\partial b_{jk}}{\partial q_i}\right)}_{\triangleq c_{ijk}} \dot{q}_k\, \dot{q}_j$$

My result:

$$C(q,\dot{q}) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & m_{\text{Link4}}\, \dot{d}_3(t)\, d_3(t) & m_{\text{Link4}}\, \dot{\theta}_2(t)\, d_3(t) \\ 0 & -m_{\text{Link4}}\, \dot{\theta}_2(t)\, d_3(t) & 0 \end{pmatrix}$$

- **Computing the G (*Gravity* term) matrix extracting $\frac{\partial U}{\partial q}$**

My result:

$$G(q) = \begin{pmatrix} 0 \\ -9.81\, m_{\text{Link4}}\, \sin\big(\theta_2(t)\big)\, d_3(t) \\ 9.81\, m_{\text{Link4}}\, \cos\big(\theta_2(t)\big) \end{pmatrix}$$

Assignment 4

- **Compute the "subsequent" transformations**

  It's worth mentioning that to correctly implement the ***RNE*** algorithm, some data is needed. First of all, *initial conditions* are needed and they're set according to the theory. The *relative transformation matrices* are computed with respect to the $\sum_0$ reference frame (and not the $\sum_{base}$ ref. frame). Also, *inertia matrices* (the ones obtained using the Steiner's theorem) are required.

  So, the intermediate transformations are:

$$T_2^1 = \begin{pmatrix} \cos(\theta_2(t)) & 0 & -\sin(\theta_2(t)) & 0 \\ \sin(\theta_2(t)) & 0 & \cos(\theta_2(t)) & 0 \\ 0 & -1 & 0 & L_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad T_3^2 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & d_3(t) - \frac{L_4}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$T_4^3 = \begin{pmatrix} 0 & 0 & 1 & -\frac{L_2}{2} - L_3 - d_1(t) \\ -\cos(\theta_2(t)) & -\sin(\theta_2(t)) & 0 & 0 \\ \sin(\theta_2(t)) & -\cos(\theta_2(t)) & 0 & d_3(t) - \frac{L_4}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  To compute T_1^2 *we can exploit:* $T_1^2 = (T_2^1)^{-1}$ *and so on…*

- **Compute the Recursive Newton-Euler formulation**

  Forward recursion

  Once the *joint* positions, velocities and accelerations q, $\dot{q}$, $\ddot{q}$ and the velocity and acceleration of the *base link* ω_0, $\ddot{p}_0 - g_0$, $\dot{\omega}_0$ are specified, ω_i, $\dot{\omega}_i$, $\ddot{p}_i$, $\ddot{p}_{C_i}$, $\dot{\omega}_{m_i}$ from the base link to the end-effector can be computed

  Backward recursion

  Once $h_e = \begin{bmatrix} f_{n+1}^T & \mu_{n+1}^T \end{bmatrix}^T$ in known, the Newton and Euler equations are solved to compute f_i and μ_i from the end-effector to the base link.

  Finally the *joint torques* are given by

$$\tau_i = \begin{cases} f_i^T z_{i-1} + k_{ri} I_{m_i} \dot{\omega}_{m_i}^T z_{m_i} + F_{vi} \dot{d}_i + F_{si} \text{sign}(\dot{d}_i), & \text{for a prismatic joint} \\ \mu_i^T z_{i-1} + k_{ri} I_{m_i} \dot{\omega}_{m_i}^T z_{m_i} + F_{vi} \dot{\vartheta}_i + F_{si} \text{sign}(\dot{\vartheta}_i), & \text{for a revolute joint} \end{cases}$$

  The *RNE formulation* is a computationally efficient (recursion) way to derive joint torques of a manipulator ($O(n)$ complexity). The same approach can be applied to any robotic structure; the only drawback is that there are not insights about the manipulator's mechanical behaviour, as opposed to Lagrange method.

- **The Forward algorithm**

1: /* Initial Conditions */
2: $\omega_0,\ \ddot{p}_0 - g_0,\ \dot{\omega}_0$
3: **for** $i = 1$ to n **do**
4: Given current $q_i, \dot{q}_i, \ddot{q}_i$ (i.e. $\vartheta_i, \dot{\vartheta}_i, \ddot{\vartheta}_i$ or $d_i, \dot{d}_i, \ddot{d}_i$)
5: /* if revolute joint add, if prismatic joint add */
6: $R_i^{i-1} = R_i^{i-1}(\vartheta_i)$ or $R_i^{i-1} = R_i^{i-1}(d_i)$
7: $\omega_i^i = (R_i^{i-1})^T \omega_{i-1}^{i-1} + (R_i^{i-1})^T \dot{\vartheta}_i z_0$
8: $\dot{\omega}_i^i = (R_i^{i-1})^T \dot{\omega}_{i-1}^{i-1} + (R_i^{i-1})^T (\ddot{\vartheta}_i z_0 + \dot{\vartheta}_i \omega_{i-1}^{i-1} \times z_0)$
9: $\ddot{p}_i = (R_i^{i-1})^T \ddot{p}_{i-1}^{i-1} + \dot{\omega}_i^i \times r_{i-1,i}^i + \omega_i^i \times (\omega_i^i \times r_{i-1,i}^i) + (R_i^{i-1})^T \ddot{d}_i z_0 + 2\dot{d}_i \omega_i^i \times ((R_i^{i-1})^T z_0)$
10: $\ddot{p}_{C_i}^i = \ddot{p}_i^i + \dot{\omega}_i^i \times r_{i,C_i}^i + \omega_i^i \times (\omega_i^i \times r_{i,C_i}^i)$
11: $\dot{\omega}_{m_i}^{i-1} = \dot{\omega}_{i-1}^{i-1} + k_{ri} \ddot{q}_i z_{m_i}^{i-1} + k_{ri} \dot{q}_i \omega_{i-1}^{i-1} \times z_{m_i}^{i-1}$
12: **end for**

Algorithm 1: Forward equations

$$\ddot{p}_0 - g_0 = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \qquad \dot{\omega}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \omega_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

- **The Backward algorithm**

1: /* Initial Conditions */
2: $h_e = \begin{bmatrix} f_{n+1} \\ \mu_{n+1} \end{bmatrix}$
3: $f_{n+1}^{n+1} = f_{n+1},\ \mu_{n+1}^{n+1} = \mu_{n+1}$
4: **for** $i = n$ to 1 **do**
5: Given current $\omega_i^i,\ \dot{\omega}_i^i,\ \ddot{p}_i^i,\ \ddot{p}_{C_i}^i,\ \dot{\omega}_{m_i}^i$
6: $R_{i+1}^i = R_{i+1}^i(\vartheta_{i+1})$ or $R_{i+1}^i = R_{i+1}^i(d_{i+1})$
7: $f_i^i = R_{i+1}^i f_{i+1}^{i+1} + m_i \ddot{p}_{C_i}^i$
8: $\mu_i^i = -f_i^i \times (r_{i-1,i}^i + r_{i,C_i}^i) + R_{i+1}^i \mu_{i+1}^{i+1} + R_{i+1}^i f_{i+1}^{i+1} \times r_{i,C_i}^i + \bar{I}_i^i \dot{\omega}_i^i + \omega_i^i \times (\bar{I}_i^i \omega_i^i) + k_{r,i+1} \ddot{q}_{i+1} I_{m_{i+1}} z_{m_{i+1}}^i + k_{r,i+1} \dot{q}_{i+1} I_{m_{i+1}} \omega_i^i \times z_{m_{i+1}}^i$
9: $\tau_i = \begin{cases} (f_i^i)^T (R_i^{i-1})^T z_0 + k_{ri} I_{m_i} (\omega_{m_i}^{i-1})^T z_{m_i}^{i-1} + F_{vi} \dot{d}_i + F_{si} \text{sign}(\dot{d}_i) \\ (\mu_i^i)^T (R_i^{i-1})^T z_0 + k_{ri} I_{m_i} (\omega_{m_i}^{i-1})^T z_{m_i}^{i-1} + F_{vi} \dot{\vartheta}_i + F_{si} \text{sign}(\dot{\vartheta}_i) \end{cases}$
10: **end for**

Algorithm 2: Backward equations

- **Computing the Inverse Dynamics with *RNE*, to check with the previously obtained results through *Lagrange***

$$\begin{aligned}\tau_d &= B(q_d)\ddot{q}_d + C(q_d, \dot{q}_d)\dot{q}_d + F_v\dot{q}_d + F_s\text{sign}(\dot{q}_d) + g(q_d) \\ &= NE(q_d, \dot{q}_d, \ddot{q}_d; g_0)\end{aligned}$$

$$B(q) = \begin{bmatrix} B_1(q) & \dots & B_n(q) \end{bmatrix}, \quad B_i(q) = NE(q, 0, e_i; 0), \quad e_i = i\text{-th element equal to 1}$$

$$C(q, \dot{q})\dot{q} = NE(q, \dot{q}, 0; 0) = NE(q, \dot{q}, 0; g_0) - NE(q, 0, 0; g_0)$$

$$g(q) = NE(q, 0, 0; g_0)$$

*My code snippets in order to compute the G, C*Dq vectors and the B matrix (the results are the same obtained with Lagrange formulas):*

```
%Compute the G vector
showPrints = true;
q = [d1(t) theta2(t) d3(t)].';  %syms position col. vector
Dq  = sym([0 0 0]).';           %syms velocity col. vector
DDq = sym([0 0 0]).';           %syms acceleration col. vector
g0 = sym([0 -9.81 0]).';
DDp0_g0 = sym([0 0 0]).' - g0;
G_NE = NE(q, Dq, DDq, DDp0_g0, I_all, T_all, showPrints);
```

```
%Compute the C*Dq vector
showPrints = true;
q =  [d1(t) theta2(t) d3(t)].';     %syms position col. vector
Dq = [Dd1(t) Dtheta2(t) Dd3(t)].';  %syms velocity col. vector
DDq = sym([0 0 0]).';               %syms acceleration col. vector
g0 = sym([0 0 0]).';
DDp0_g0 = sym([0 0 0]).' - g0;
CxDq_NE = NE(q, Dq, DDq, DDp0_g0, I_all, T_all, showPrints);
```

```
%Compute the B matrix
showPrints = false;
B_NE = sym(zeros(3));
q =  [d1(t) theta2(t) d3(t)].';     %syms position col. vector
Dq  = sym([0 0 0]).';               %syms velocity col. vector
g0 = sym([0 0 0]).';
DDp0_g0 = sym([0 0 0]).' - g0;
for k = 1:3
    DDq = sym([0 0 0]).';           %syms acceleration col. vector
    DDq(k) = 1;
    B_NE(:,k) = NE(q, Dq, DDq, DDp0_g0, I_all, T_all, showPrints);
end
```

Assignment 5

- **Computing the Dynamic model in operational space**

$$
\begin{aligned}
B_A(x) &= J_A^{-T} B J_A^{-1} \\
C_A\dot{x} &= J_A^{-T} C\dot{q} - B_A \dot{J}_A \dot{q} \\
g_A(x) &= J_A^{-T} g \\
u &= T_A^T(x) h \\
u_e &= T_A^T(x) h_e
\end{aligned}
$$

My Ja and Ta:

$$
Ja = \begin{pmatrix}
0 & \cos(\theta_2(t))\left(\frac{L_4}{2} - d_3(t)\right) & -\sin(\theta_2(t)) \\
0 & \sin(\theta_2(t))\left(\frac{L_4}{2} - d_3(t)\right) & \cos(\theta_2(t)) \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{pmatrix}
\qquad
Ta = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 1 & 0 & 0
\end{pmatrix}
$$

Even if not so meaningful, Ba (partially evaluated), Ca (partially evaluated), Ga:

$$
B_a(x) = \begin{pmatrix}
\frac{-192000\sin(\theta_2(t))^2 d_3(t)^3 + 10483075\sin(\theta_2(t))^2 d_3(t)^2 - 3962830\sin(\theta_2(t))^2 d_3(t) + 5188603\sin(\theta_2(t))^2 + 480000 d_3(t)^4 - 192000 d_3(t)^3 + 268925 d_3(t)^2 - 30770 d_3(t) + 3077}{3840\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} & \frac{\sin(2\theta_2(t))\left(192000 d_3(t)^3 - 10483075 d_3(t)^2 + 3962830 d_3(t) - 5188603\right)}{7680\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} & 0 \\
\frac{\sin(2\theta_2(t))\left(192000 d_3(t)^3 - 10483075 d_3(t)^2 + 3962830 d_3(t) - 5188603\right)}{7680\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} & \frac{1075200 d_3(t)^5 - 3993600 d_3(t) + 960000 d_3(t)^3\left(2\sin(\theta_2(t))^2 + 5 d_3(t) - 4\right) - 10483075\sin(\theta_2(t))^2 d_3(t)^2 - 5188603\sin(\theta_2(t))^2 + 3962830\sin(\theta_2(t))^2 d_3(t) + 5191680}{3840\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} & 0 \\
0 & 0 & \frac{17}{10} \\
-\frac{\cos(\theta_2(t))\left(960000 d_3(t)^3 - 192000 d_3(t)^2 + 15385 d_3(t) - 3077\right)}{7680\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} & -\frac{\sin(\theta_2(t))\left(960000 d_3(t)^3 - 192000 d_3(t)^2 + 15385 d_3(t) - 3077\right)}{7680\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} & 0 \\
0 & 0 & 0
\end{pmatrix}
\begin{matrix}
0 & -\frac{\cos(\theta_2(t))\left(960000 d_3(t)^3 - 192000 d_3(t)^2 + 15385 d_3(t) - 3077\right)}{7680\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} & 0 & 0 \\
0 & -\frac{\sin(\theta_2(t))\left(960000 d_3(t)^3 - 192000 d_3(t)^2 + 15385 d_3(t) - 3077\right)}{7680\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & \frac{192000 d_3(t)^2 + 3077}{1536\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} & 0 & 0 \\
0 & 0 & 0 & 0
\end{matrix}
$$

$$
C_a\dot{x} = \begin{pmatrix}
c\frac{\dot{\theta}_2(t)\left(3077\cos(\theta_2(t))\dot{d}_3(t) + 519168\sin(\theta_2(t))\dot{\theta}_2(t) + 967630\cos(\theta_2(t))\dot{d}_3(t)d_3(t) - 399360\sin(\theta_2(t))\dot{\theta}_2(t)d_3(t) - 5107075\cos(\theta_2(t))\dot{d}_3(t)d_3(t)^2 + 960000\cos(\theta_2(t))\dot{d}_3(t)d_3(t)^3 + 1075200\sin(\theta_2(t))\dot{\theta}_2(t)d_3(t)^2 - 384000\sin(\theta_2(t))\dot{\theta}_2(t)d_3(t)^3 + 480000\sin(\theta_2(t))\dot{\theta}_2(t)d_3(t)^4\right)}{19200\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} \\
\frac{\dot{\theta}_2(t)\left(3077\sin(\theta_2(t))\dot{d}_3(t) - 519168\cos(\theta_2(t))\dot{\theta}_2(t) + 399360\cos(\theta_2(t))\dot{\theta}_2(t)d_3(t) + 967630\sin(\theta_2(t))\dot{d}_3(t)d_3(t) - 1075200\cos(\theta_2(t))\dot{\theta}_2(t)d_3(t)^2 + 384000\cos(\theta_2(t))\dot{\theta}_2(t)d_3(t)^3 - 480000\cos(\theta_2(t))\dot{\theta}_2(t)d_3(t)^4 - 5107075\sin(\theta_2(t))\dot{d}_3(t)d_3(t)^2 + 960000\sin(\theta_2(t))\dot{d}_3(t)d_3(t)^3\right)}{19200\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} \\
0 \\
\frac{\dot{d}_3(t)\dot{\theta}_2(t)\left(-192000 d_3(t)^2 + 983015 d_3(t) + 3077\right)}{3840\left(25 d_3(t)^2 - 10 d_3(t) + 26\right)^2} \\
0 \\
0
\end{pmatrix}
$$

$$
G_a(x) = \begin{pmatrix}
-\frac{4\cos(\theta_2(t))\left(\frac{981 L_4}{200} - \frac{981 d_3(t)}{100}\right)}{{L_4}^2 - 4 L_4 d_3(t) + 4 d_3(t)^2 + 4} \\
-\frac{4\sin(\theta_2(t))\left(\frac{981 L_4}{200} - \frac{981 d_3(t)}{100}\right)}{{L_4}^2 - 4 L_4 d_3(t) + 4 d_3(t)^2 + 4} \\
0 \\
-\frac{981}{25\left({L_4}^2 - 4 L_4 d_3(t) + 4 d_3(t)^2 + 4\right)} \\
0 \\
0
\end{pmatrix}
$$

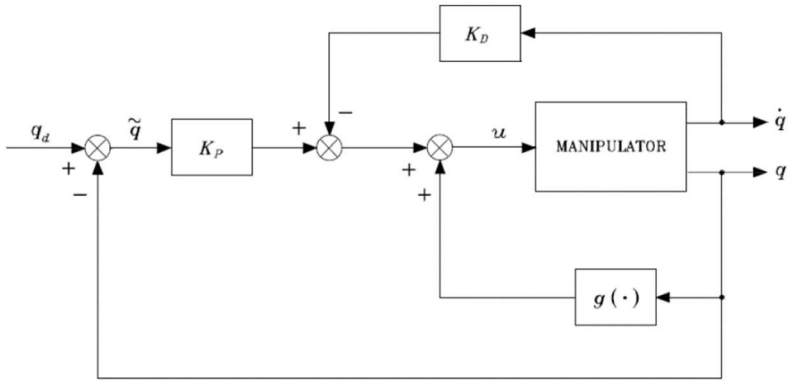
Assignment 6

- **Joint Space PD control law with gravity compensation**

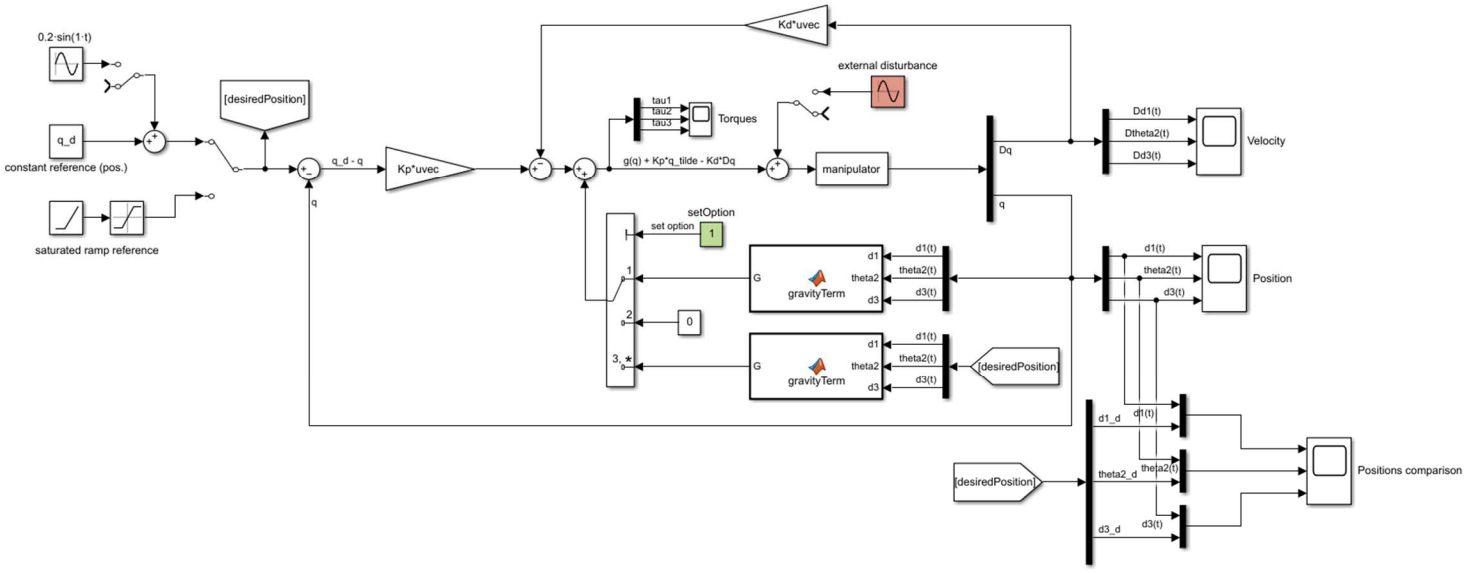
In this case, the goal is to design a controller that ensures global asymptotic stability of the desired pose, solving a joint space regulation problem. According to the theory, a ***PD controller*** is suitable to solve the task, given a good knowledge of the manipulator's dynamics. K_p and K_d are positive definite matrices that represent the *proportional* and *derivative* actions respectively. The following PD control law guarantees global asymptotic stability of the overall system in a single equilibrium point (regulation).

$$u = g(q_g) + K_P(q_d - q) - K_D\dot{q},$$

The non-linear term due to gravity is compensated computing online $g(q_g)$. Moreover, K_p and K_d are chosen diagonal to decentralize the 3 controllers, one for each degree of freedom and are set to reach the desired joint positions approximately at the same time.



My Simulink model:



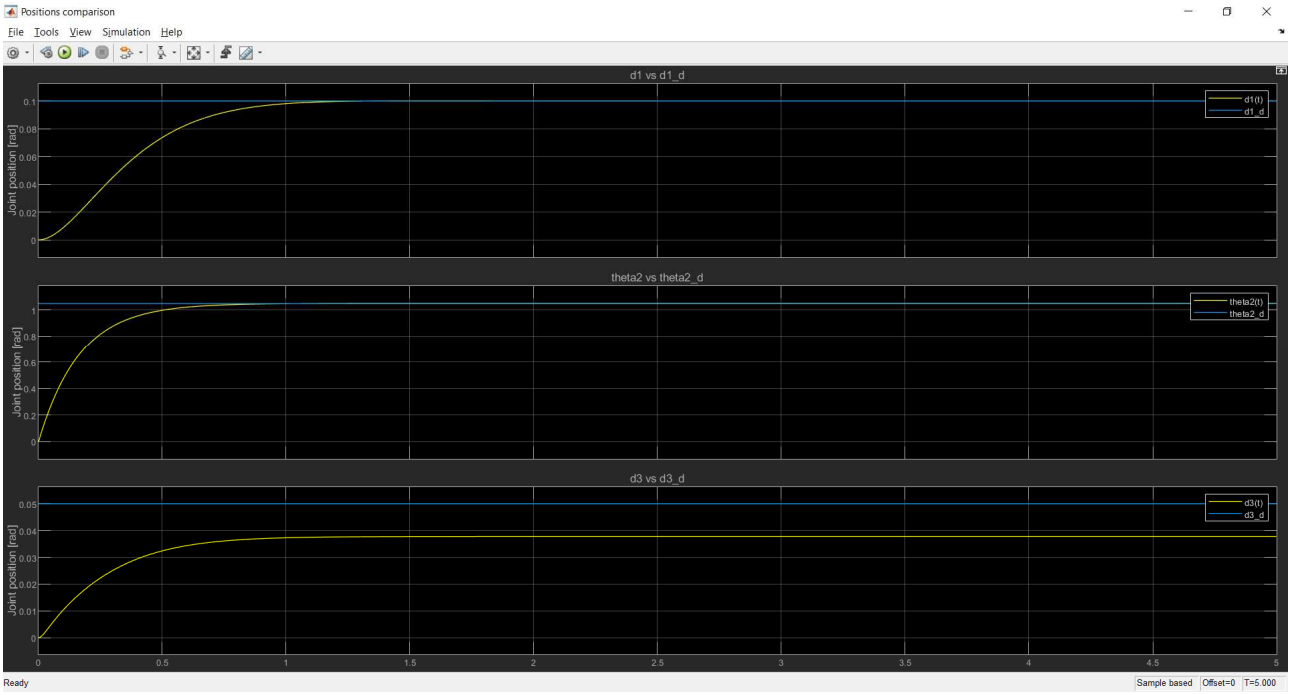
"Normal" response (option 1, q_d with gravity compensation)

In this case the 3 joint positions are reached without steady state error (asymptotically), because gravity is online computed and compensated.

- **What happens if g(q) is not taken into account?**

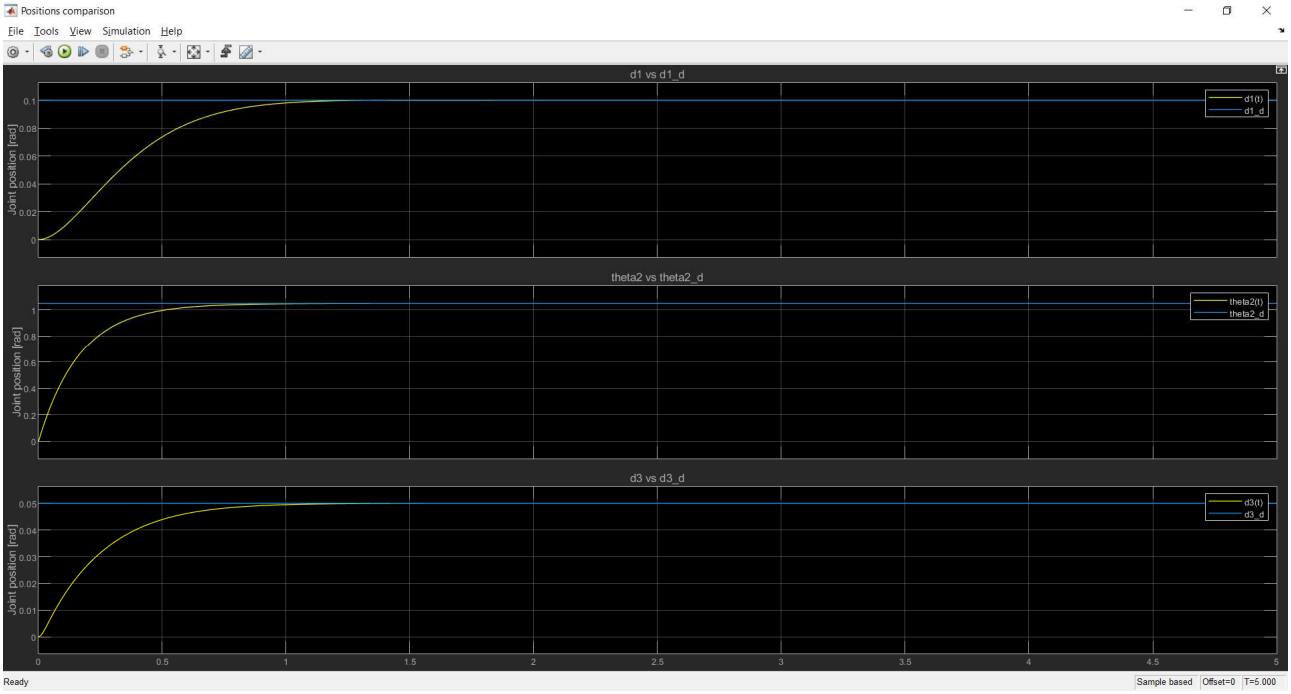
Response with constant 0 into the feedforward gravity loop (option 2, q_d)

Not compensating gravity means that the joints 2 (R) and 3 (P) (the only ones affected by gravity) will reach the desired positions with a steady state constant error.



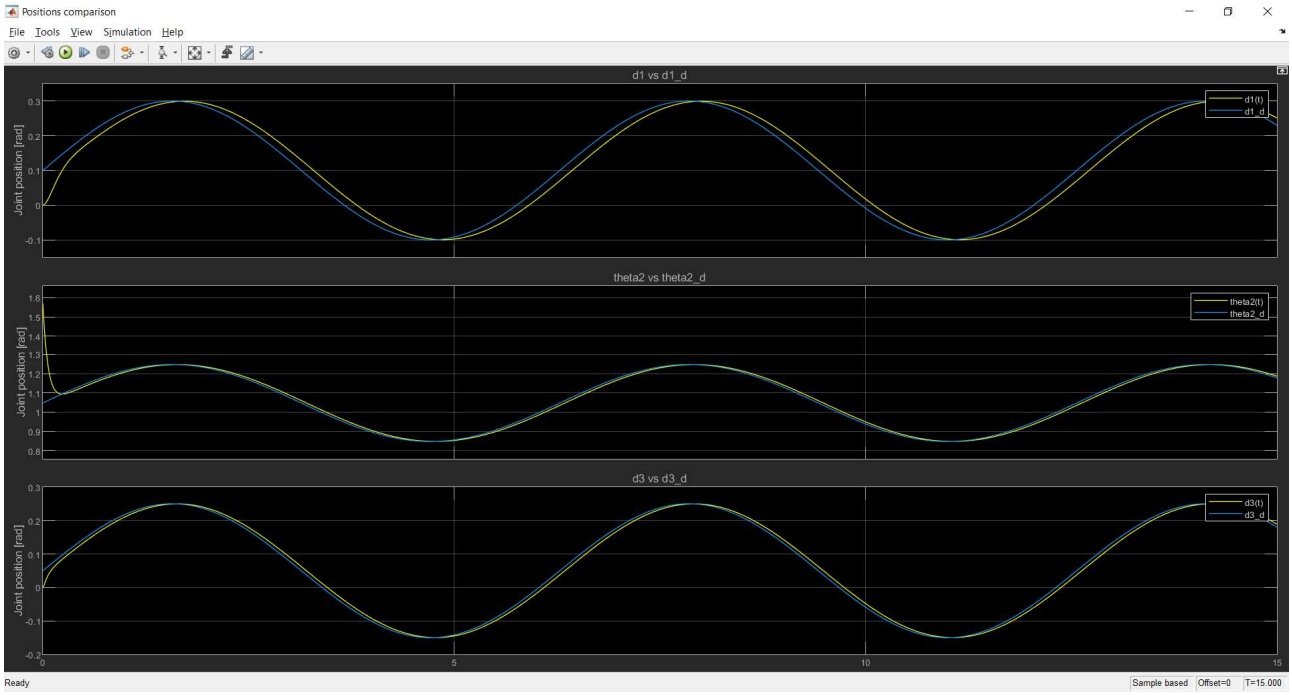
- **What happens if the gravity term is set constant and equal to g(q_d) within the control law?**

Response with gravity term equal to g(q_d) (option 3, q_d), asymptotic stability is achieved.

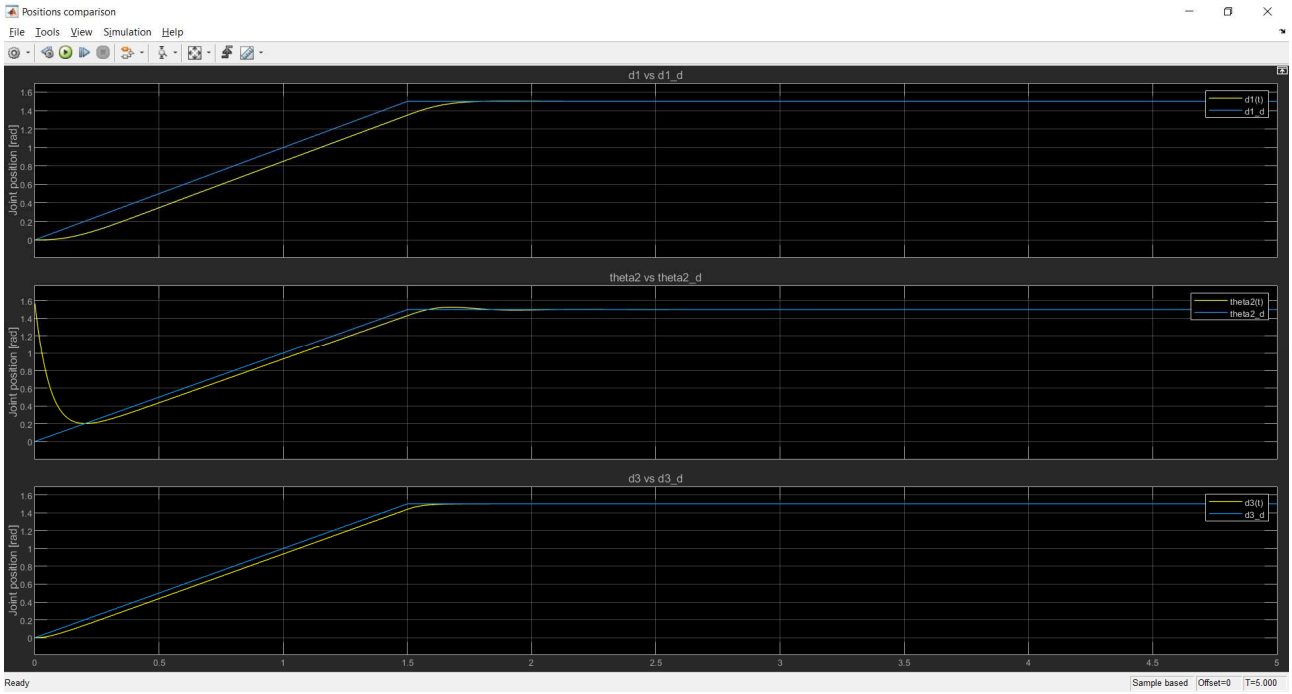


- **What happens if q_d is not constant (e.g. $q_d(t)=q_d+A\sin(\omega t)$)?**

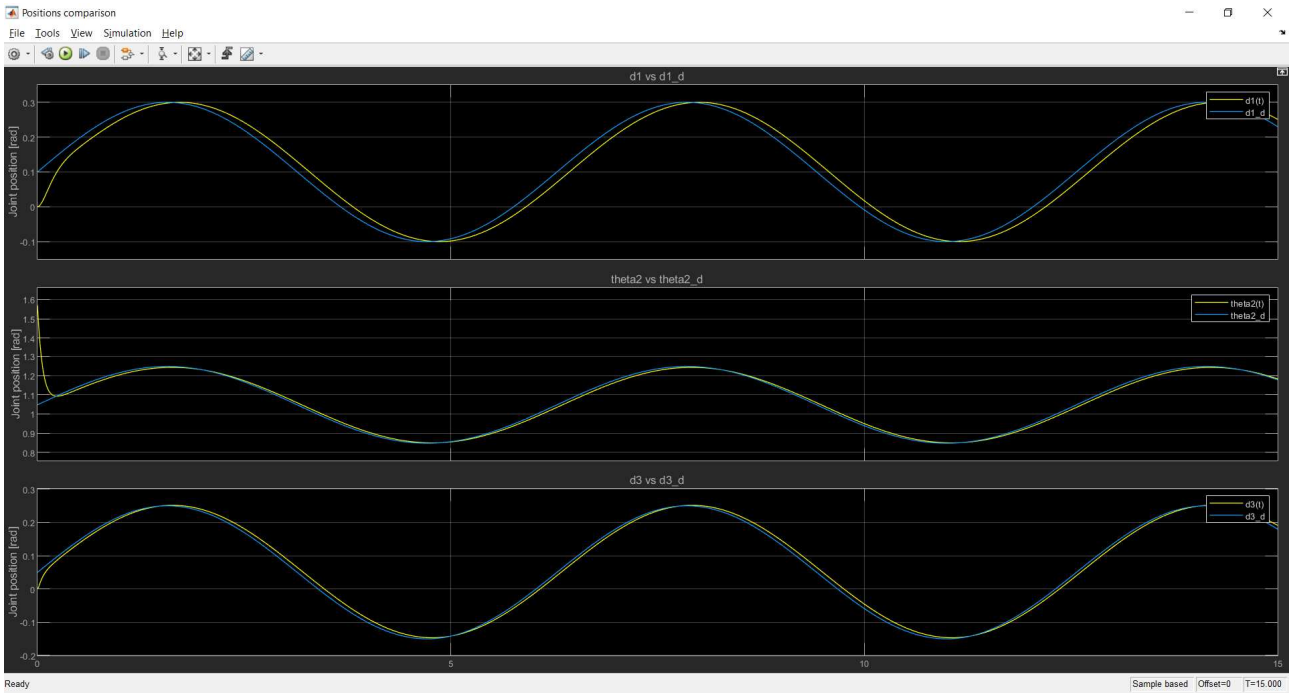
Response with reference signal equal to $q_d+A\sin(\omega t)$ with $A=0.2$; $\omega=1$ rad/s (option 1):



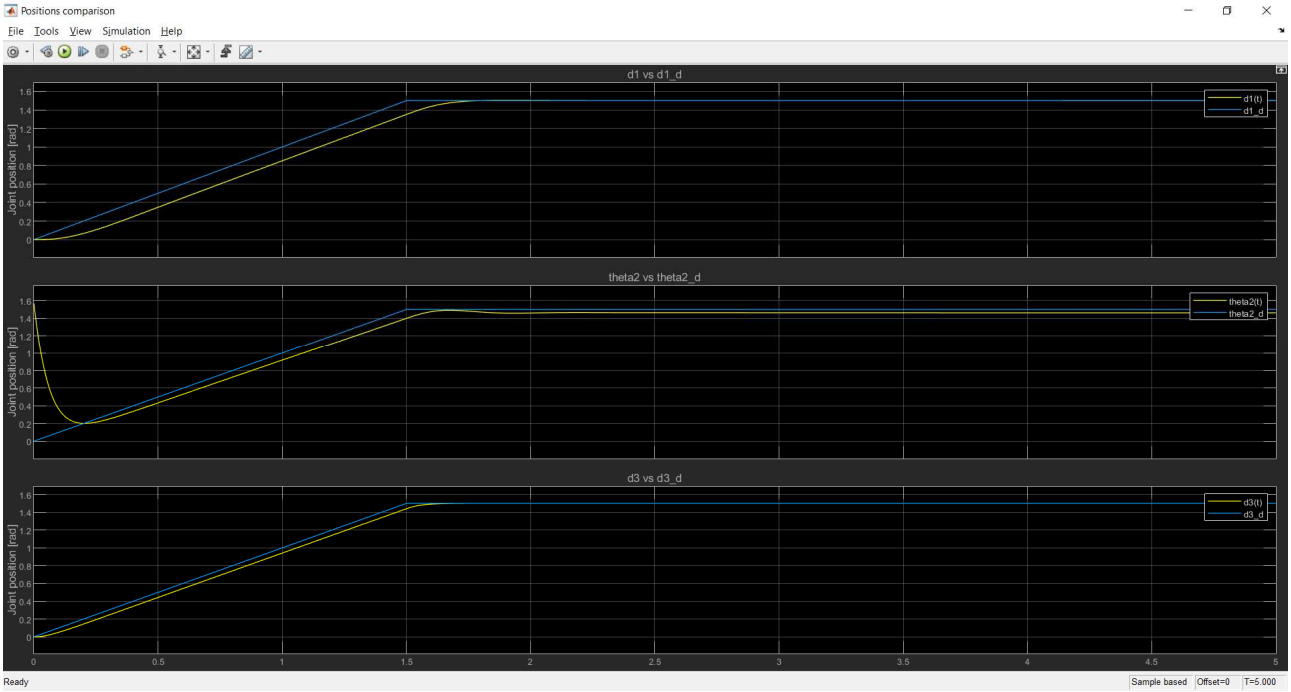
Response with the reference signal equal to a ramp from 0 to 1.5rad and slope of 1 (option 1):



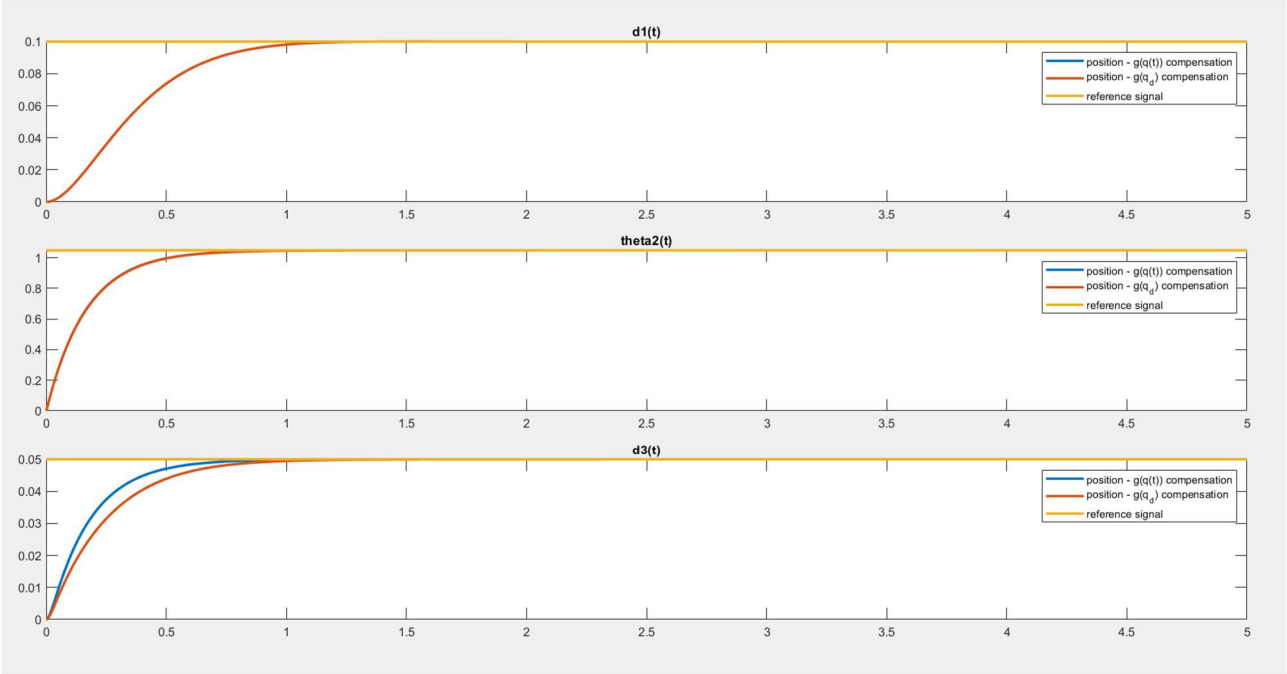
Response with reference signal equal to q_d+Asin(ωt) with A=0.2; ω=1 rad/s; with sinusoidal disturbances and without gravity compensation (option 2):



Response with reference signal equal to a ramp; with sinusoidal disturbances and without gravity compensation (option 2):



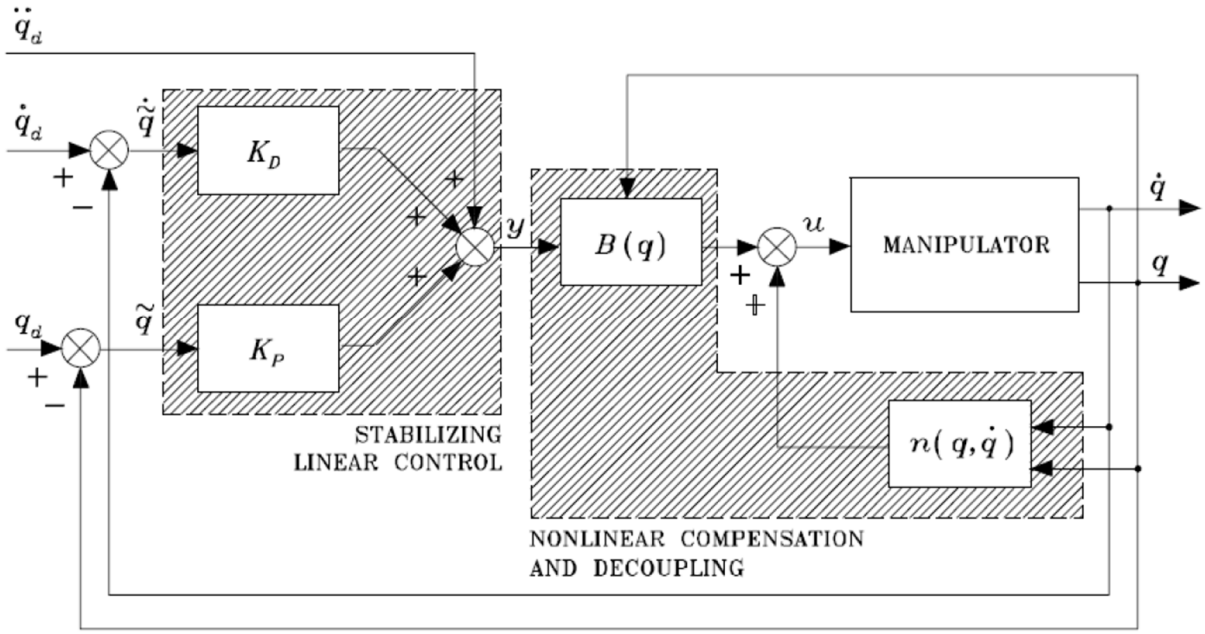
Comparison between online and constant gravity compensation (option 1 vs option 3), $g(q_d)$ has a worst transient behaviour.



Assignment 7

- **Design the Joint Space Inverse Dynamics Control law**

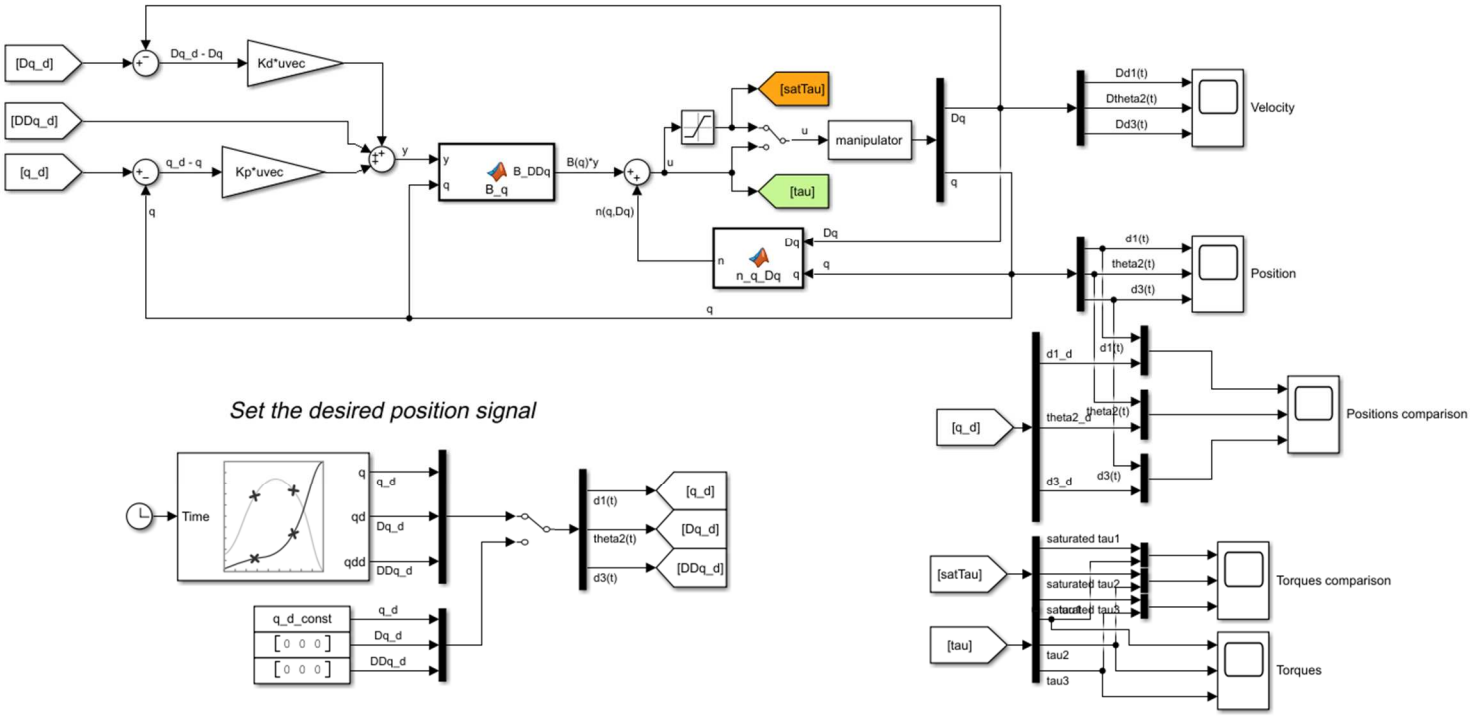
Inverse dynamics control allows to handle a ***tracking problem*** making use of the complete manipulator's dynamic model. Instead, a joint space PD control was used to mainly solve regulation problems converging to an equilibrium point and making only use of the gravity term.



Looking at the above scheme, two main parts compose the inverse dynamics control: the nonlinear state feedback that compensates gravity, Coriolis and centrifugal effects; a linear controller that stabilizes the system. In other terms, the ***inner feedback loop*** computes the inverse dynamics, decoupling each joint variable and linearizing the system; the ***outer feedback loop*** controls the joint tracking error, ensuring good enough performances.

$$\tau = \underbrace{B(q)\ddot{q}_d + C(q,\dot{q})\dot{q}_d + g(q) + F\dot{q}_d}_{\tau_{ff}} + \underbrace{K_D(\dot{q}_d - \dot{q}) + K_P(q_d - q)}_{\text{PD}}$$

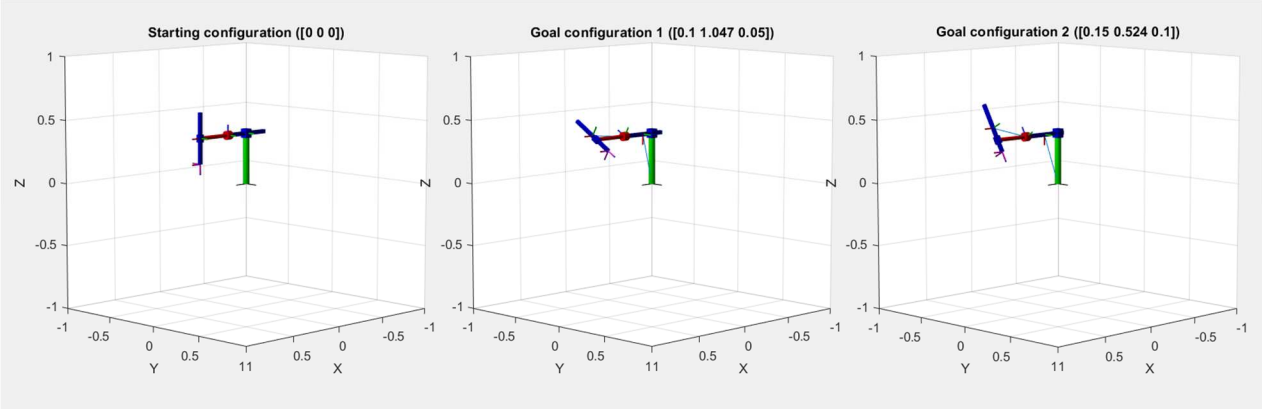
My Simulink model:



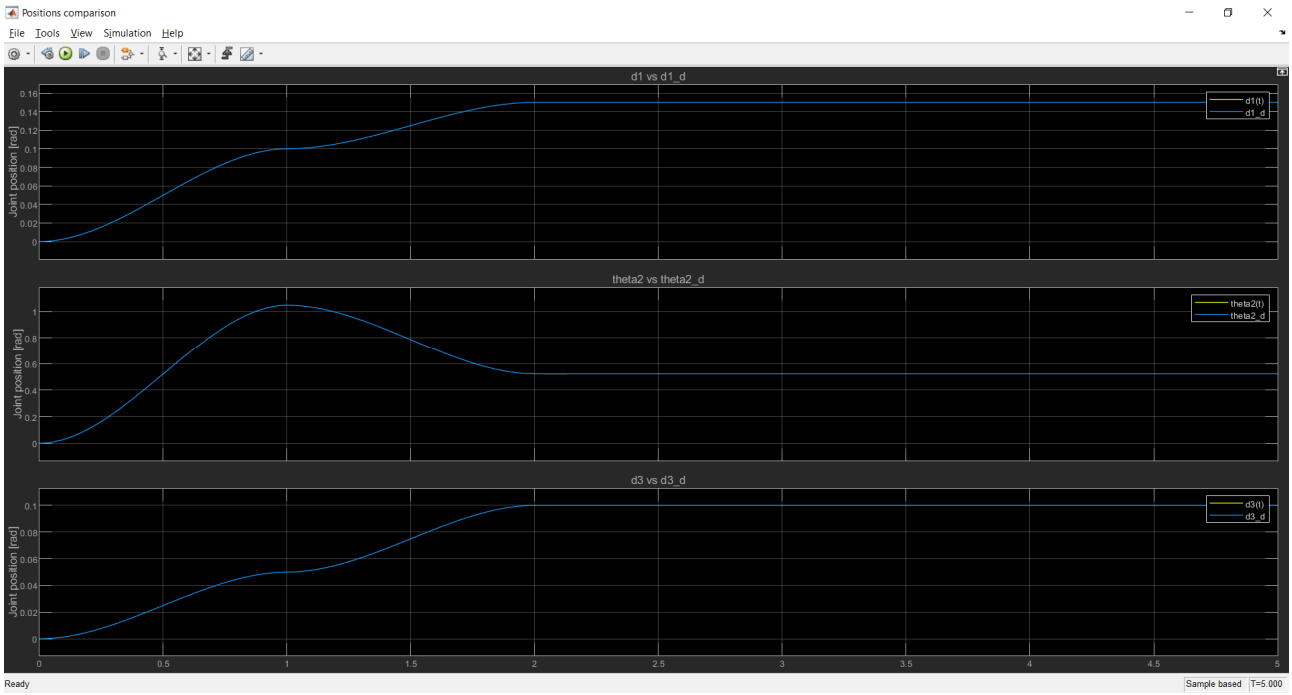
The following result is achieved choosing a desired (joint space) trajectory as such:

```
q_d = [0 0.1   0.15  ;  %joint1
       0 pi/3  pi/6  ;  %joint2
       0 0.05  0.1   ];  %joint3
```

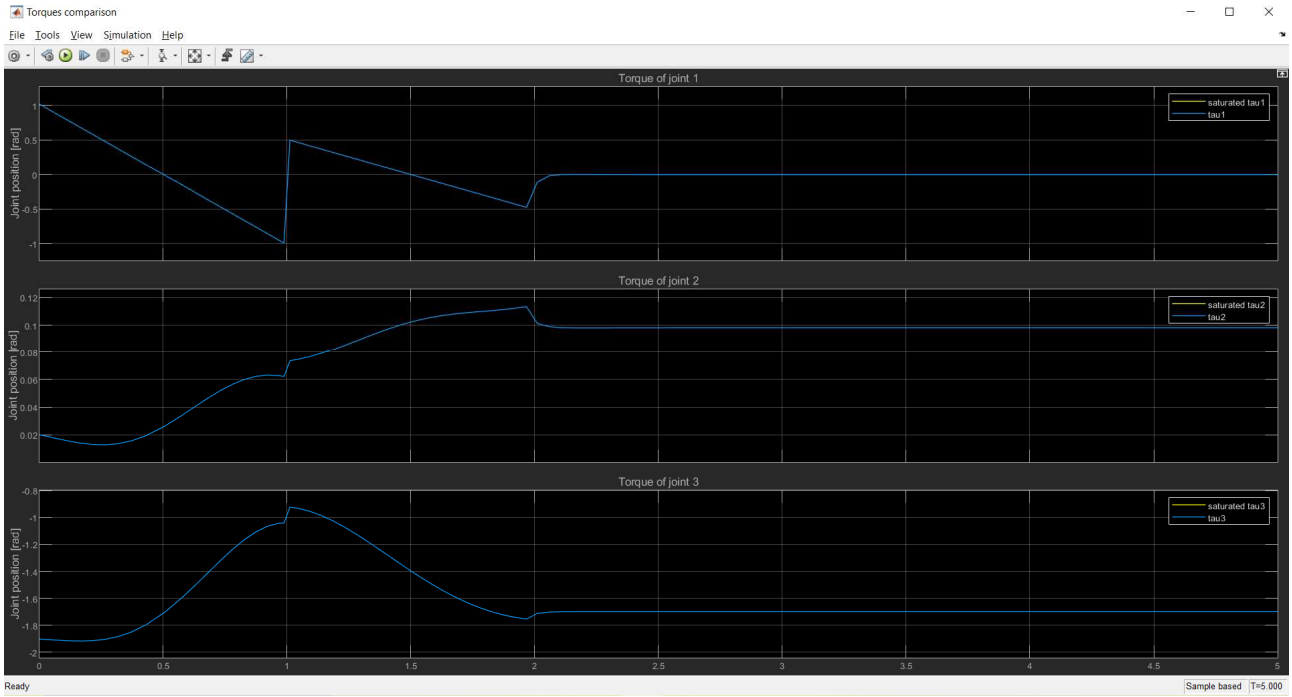
In particular, every joint displacement/angle goes from a starting to an end position, in 3 steps.



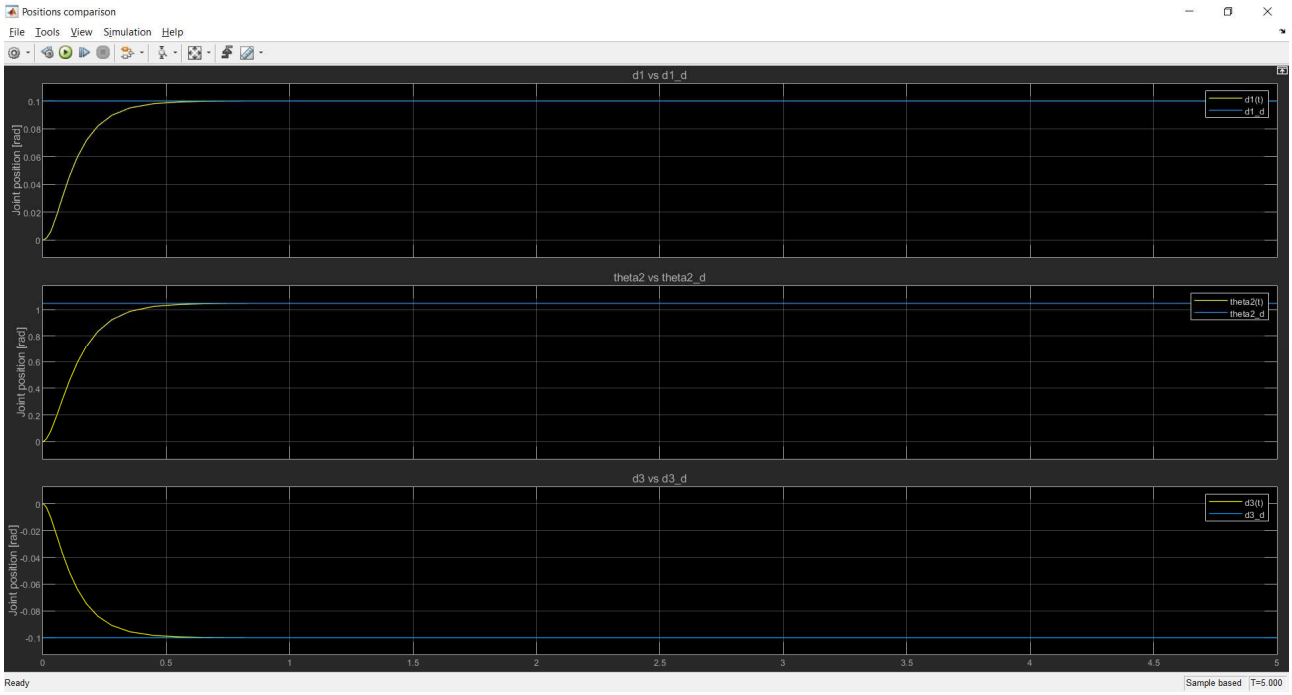
In this case, a perfect estimation of the B and G matrices is assumed, in fact the *n(q,Dq)* term, together with the PD control law, ensure a perfect tracking.



Required torques to achieve the previous result: (no saturation is required)



Check that in the nominal case the dynamic behaviour is equivalent to the one of a set of stabilized double integrators (regulation problem, desired position is kept constant)



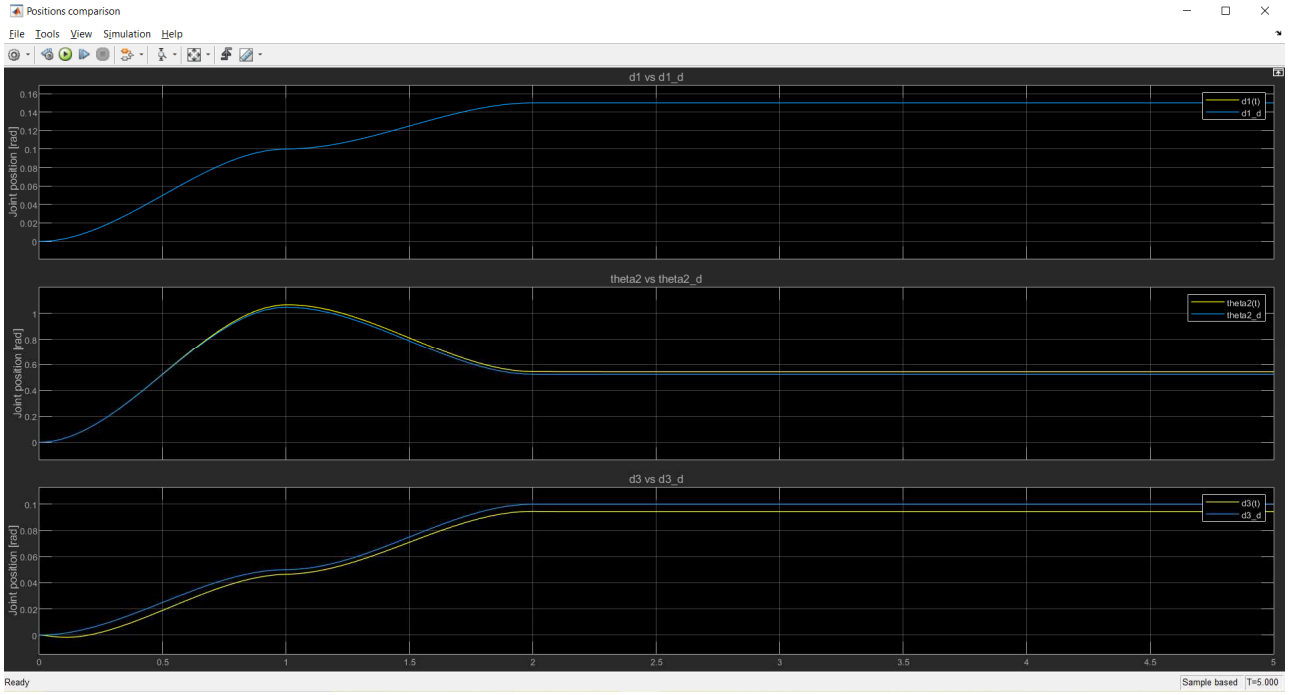
Check the behaviour of the control law when the B, C, g used within the controller are different than the "true ones" B, C, g (e.g. slightly modify the masses, the frictions, ...).

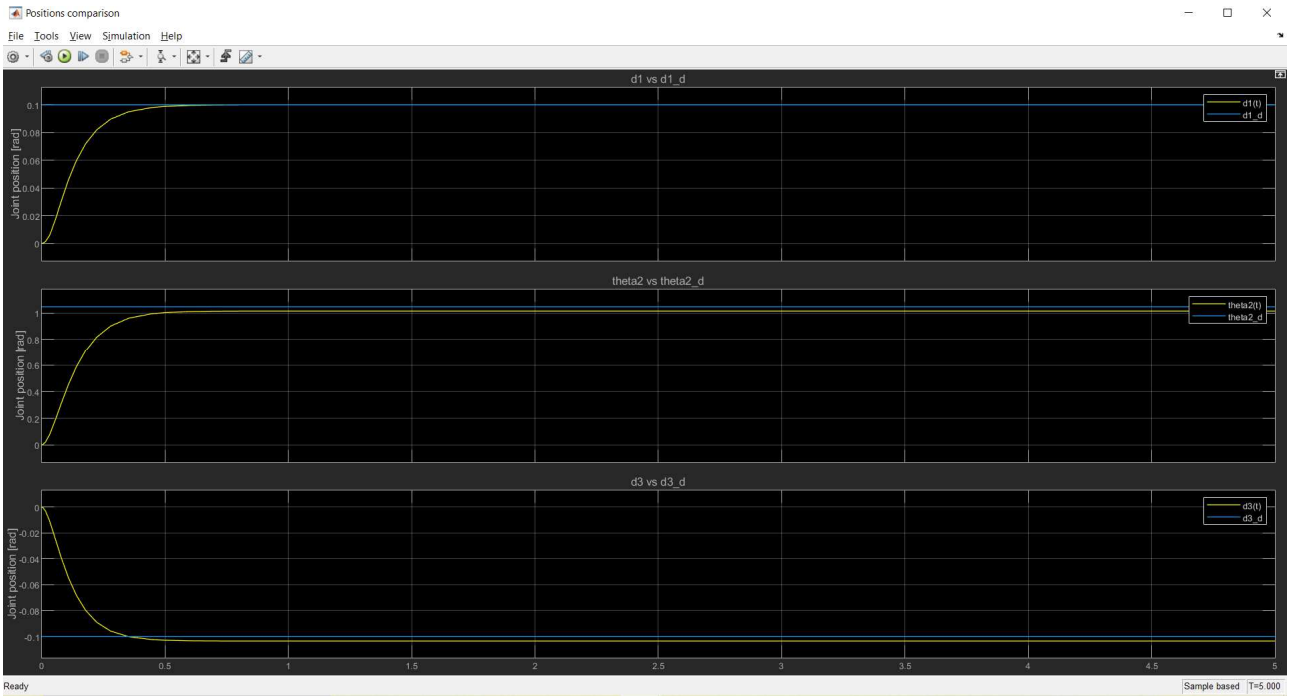
In particular, the MATLAB function "n_q_Dq" code was modified as such, to simulate a non-perfect estimation, and therefore cancellation of the dynamics:

```
%To get the "wrongly estimated" matrices use the following code
theta2 = q(2);   d3 = q(3);
Dtheta2 = Dq(2); Dd3 = Dq(3);

m_Link4 = 0.16;  %0.2kg is the correct value

C = [0,                              0,                              0;
     0,              m_Link4*Dd3*d3,        m_Link4*Dtheta2*d3;
     0,        -m_Link4*Dtheta2*d3,                              0];
G = [            0                                    ;
        -(981*m_Link4*sin(theta2)*d3)/100             ;
             (981*m_Link4*cos(theta2))/100          ];
```

The main effect we expect is a ***steady-state error***, due to the non-perfect compensation of the manipulator's exact dynamics. In particular, errors can be seen in the *second* and *third* joints, that are the ones affected by gravity.

So, the next two scopes show the achieved performances in tracking and regulation:



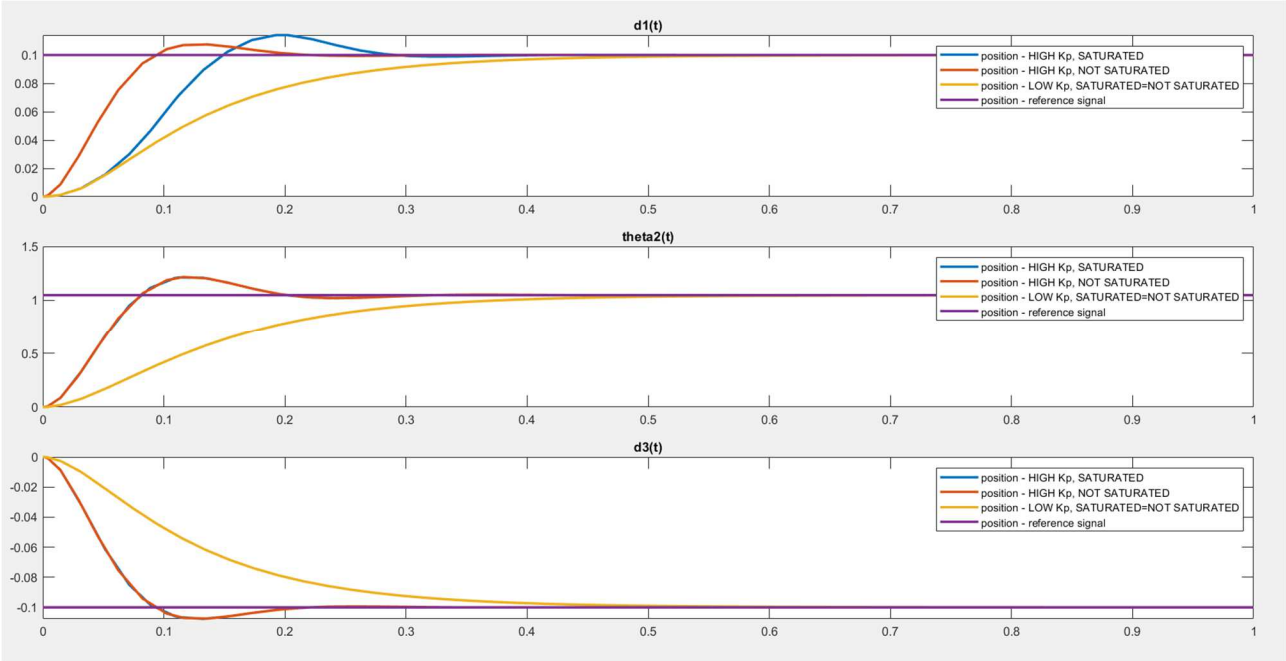


What happens to the torque values when the settling time of the equivalent second order systems is chosen very small?

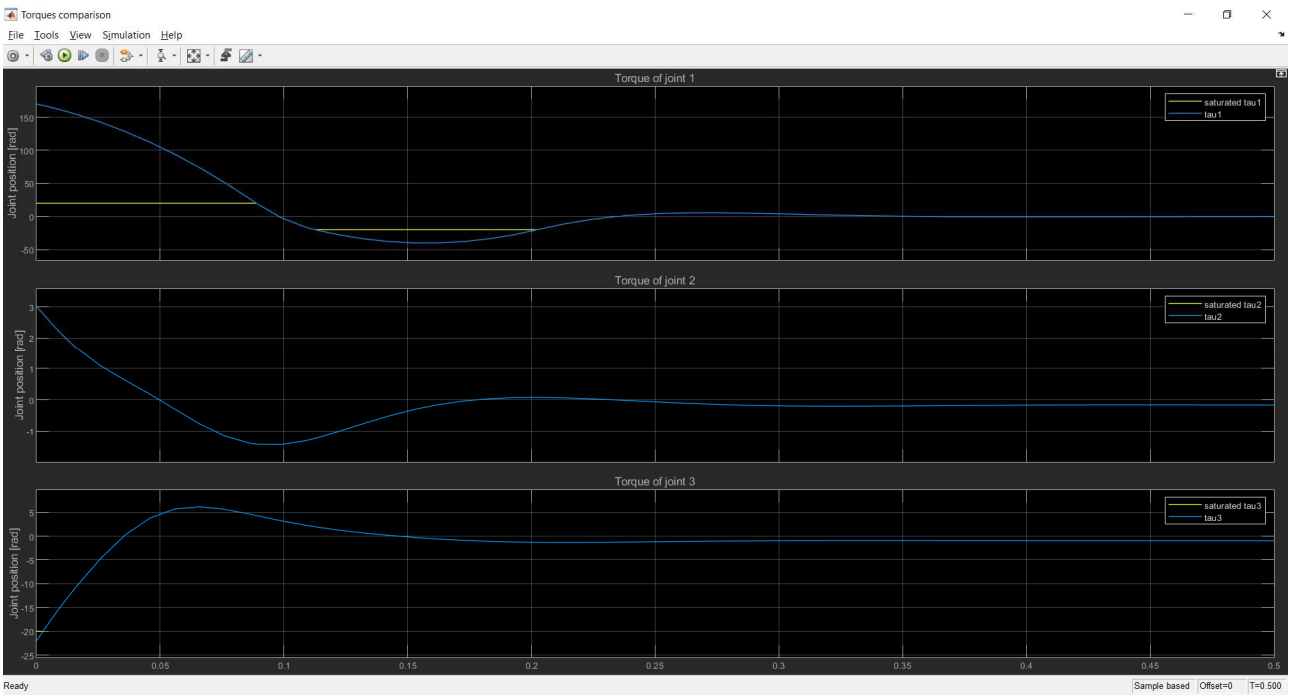
To reduce the settling time, ***K_p values must be increased***, meaning that ***higher torques*** will be required. It can create an effect on the overall stability of the system, potentially damaging the mechanical/electrical components. In real world manipulators, currents/voltages are limited for safety reasons. Adding a "saturator" block inside the Simulink model, this behaviour can be detected.

*Position comparison **with** and **without** using the saturated torques as input of the manipulator (the actual settling in the real-world scenario wouldn't be smaller and overshoot would be higher).*

To notice that the main effect is shown on the first joint, this is because the other two don't actually reach saturation values, but it depends on the real world manufacture specifications. For simulation purposes, saturation was set at ±20N·m.



Torques comparison with high Kp values (saturation set at ±20N·m):



Assignment 8

- **Implement in Simulink the Adaptive Control law for the a 1-DoF link under gravity.**

Adaptive control is an online adaptation of the computational model to the dynamic model of the real manipulator. It's one of the techniques that can be used to deal with uncertainty in the model's parameters, such as dynamic parameters (masses, inertia matrices...), friction coefficients and payload at the end effector. The DH table is assumed to be known and correct.

For simplicity, a *single degree of freedom link* is considered, so writing the dynamic model as such, a regressor matrix can be used to find the best parameters through an iterative process:

$$I\ddot{q} + F\dot{q} + \underbrace{mgd}_{G}\sin q = \tau \qquad \tau = \underbrace{\begin{bmatrix}\ddot{q} & \dot{q} & \sin q\end{bmatrix}}_{Y(q,\dot{q},\ddot{q})} \underbrace{\begin{bmatrix} I \\ F \\ G \end{bmatrix}}_{\Theta}$$

Given a twice differentiable desired trajectory, the goal is to design an asymptotically stable control law despite the model's *parameters uncertainty*. This is done introducing a reference velocity signal, a velocity error σ, positive definite matrices K_d, K_p, Λ and a learning rate $K_\Theta = \Gamma^{-1}$:

Given the desired trajectory $q_d(t)$ for the robotic arm

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F\dot{q} + g(q) = \tau,$$

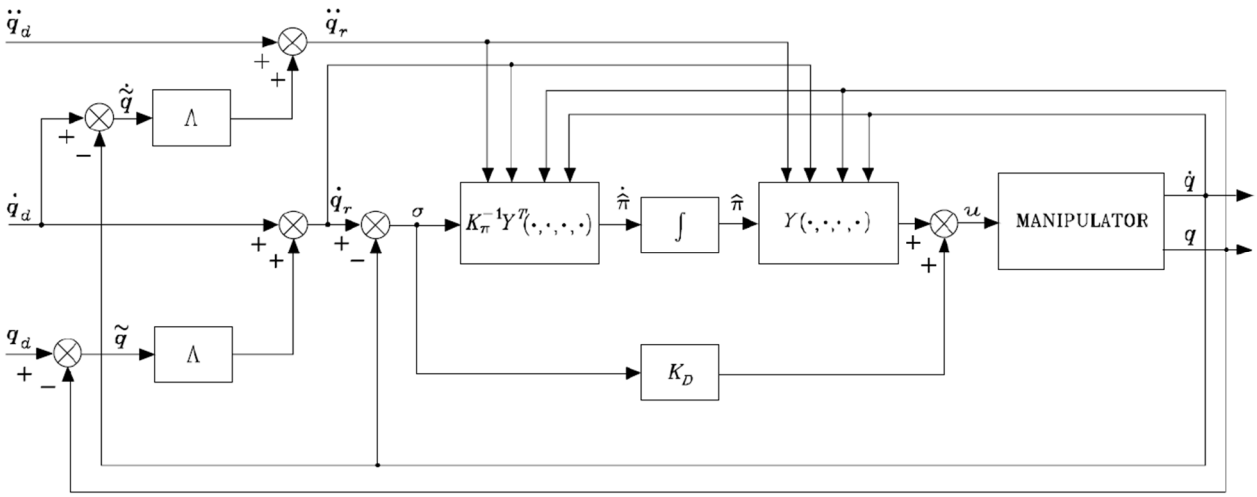
the control law

$$\begin{cases} \tau &= \hat{B}(q)\ddot{q}_r + \hat{C}(q,\dot{q})\dot{q}_r + \hat{F}\dot{q}_r + \hat{g}(q) + K_D\sigma = Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)\hat{\Theta} + K_D\sigma \\ \dot{\hat{\Theta}} &= \Gamma\, Y^T(q,\dot{q},\dot{q}_r,\ddot{q}_r)\sigma, \qquad (\Gamma := K_\Theta^{-1} \succ 0) \end{cases}$$

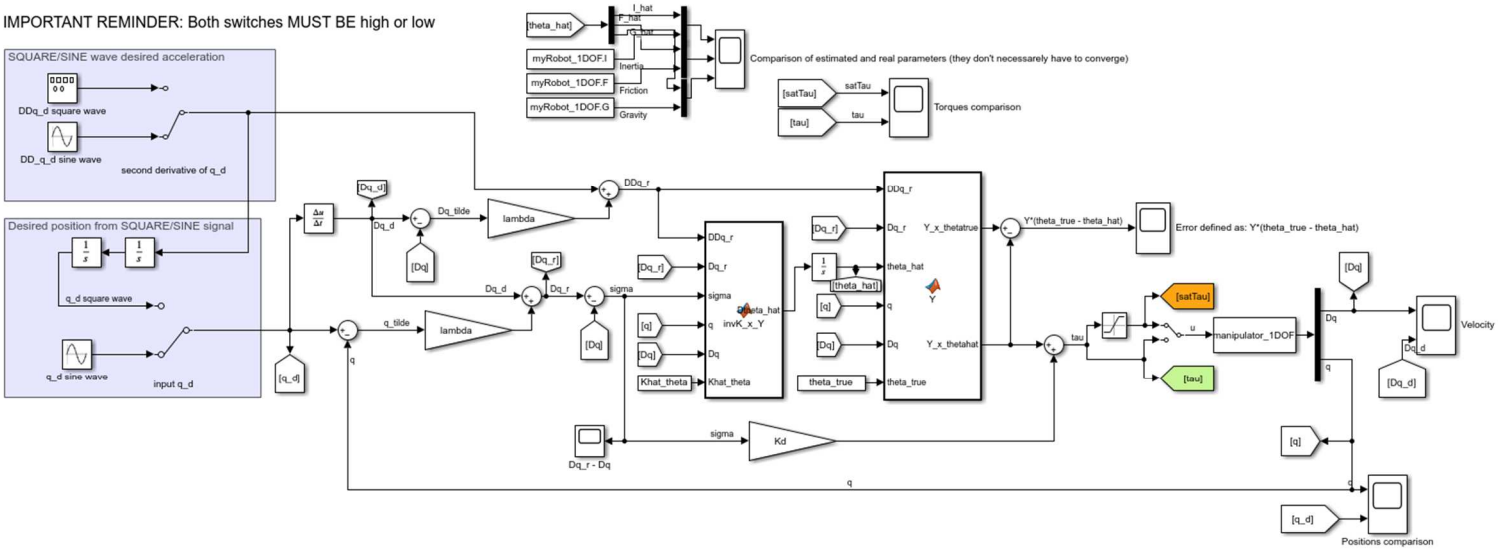
guarantees that the tracking error along the desired trajectory is globally asymptotically stable, i.e. $(\tilde{q}(t), \dot{\tilde{q}}(t)) \to (0,0)$.

$$\begin{aligned} \dot{q}_r &= \dot{q}_d + \Lambda\tilde{q}, & \tilde{q} &= q_d - q \\ \ddot{q}_r &= \ddot{q}_d + \Lambda\dot{\tilde{q}}, & \dot{\tilde{q}} &= \dot{q}_d - \dot{q} \\ \sigma &= \dot{q}_r - \dot{q} \end{aligned}$$

Joint Space Adaptive Control block scheme:



My Simulink model:



The ***initial estimates*** were set as: `[0.2 0.1 0]` (*inertia*, *friction*, *gravity*); while the ***real values*** are set as: `[0.1 0.05 -0.981]`. Theoretically, convergence should be reached from every starting condition.
An interesting point is shown in the following slide of the course and it will be confirmed by the experiments.

Since

$$B(q)\dot{\sigma} + C(q,\dot{q})\sigma + F\sigma + K_D\sigma = -Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)\tilde{\Theta}$$

and $\sigma \to 0$, we have that

$$Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)\tilde{\Theta} \to 0$$

which does not mean that the estimation error $\tilde{\Theta}$ goes to zero, but only that $\hat{\Theta} - \Theta$ belong to the kernel of $Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)$, i.e.

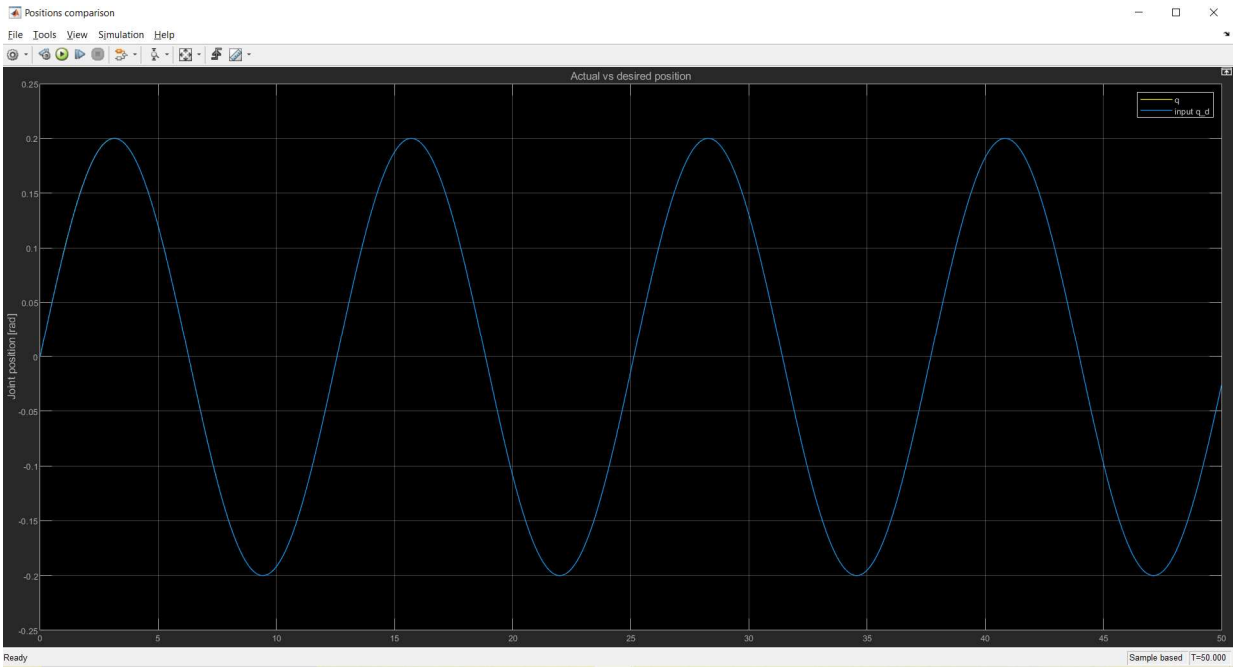
$$Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)\hat{\Theta} \to Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)\Theta$$

even though $\hat{\Theta} \nrightarrow \Theta$.

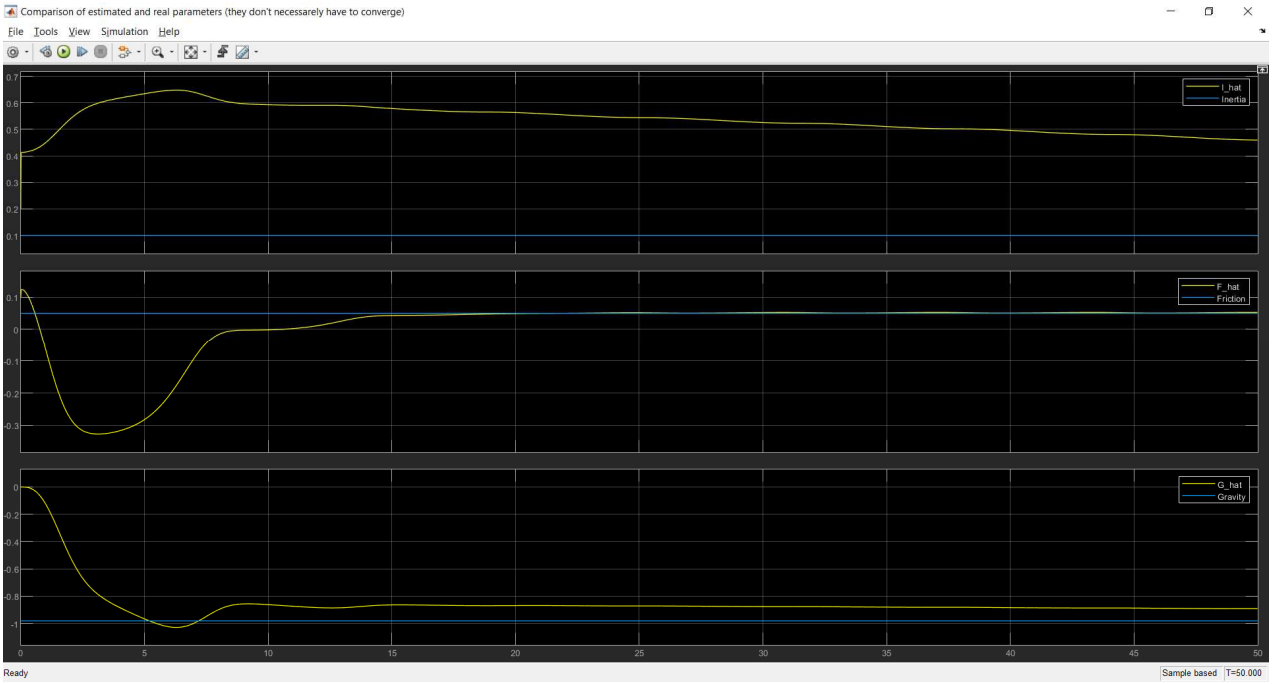
So, solving the ***direct adaptive control*** problem, we ensure limited tracking errors, but we don't determine the actual parameters in a precise way.

Implement in Simulink the Adaptive Control law for the a 1-DoF link under gravity. Choose the dynamic parameters and their initial estimates, and set the desired trajectory as:
(1) $\boldsymbol{q_d(t) = Asin(\omega t)}$
(2) $\boldsymbol{DDq_d(t) =}$ ***periodic square wave*** $\boldsymbol{\pm A}$

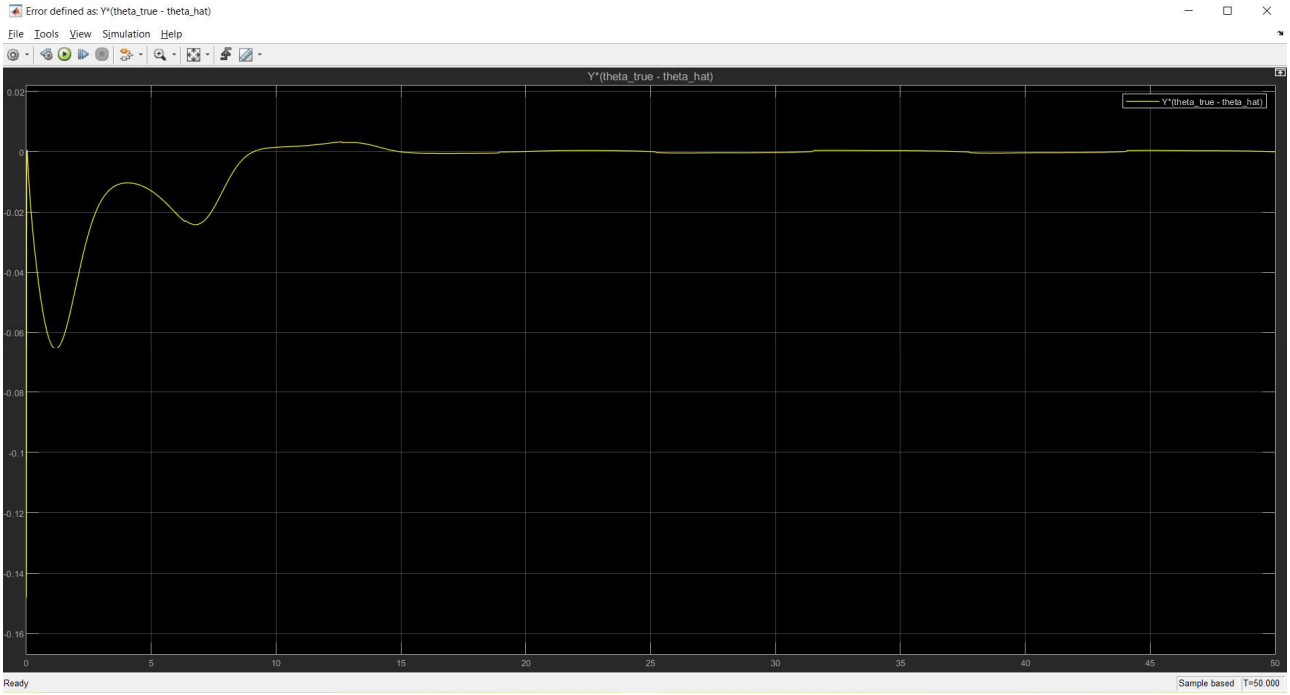
(1) *Position tracking performance:*



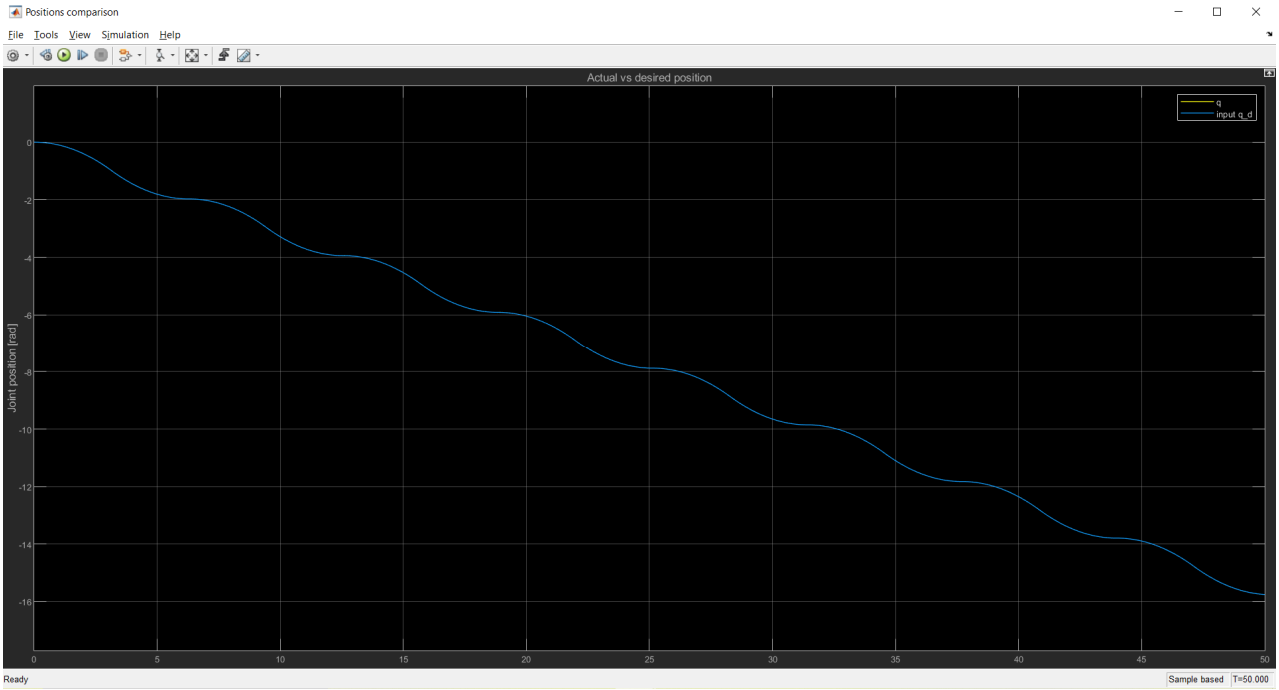
(1) *Comparison between actual and estimated dynamic parameters (I, F, G): they don't converge (50s long simulation)*



(1) Plot of the actual error: $Y \cdot (\hat{\theta} - \theta)$ that tends to zero



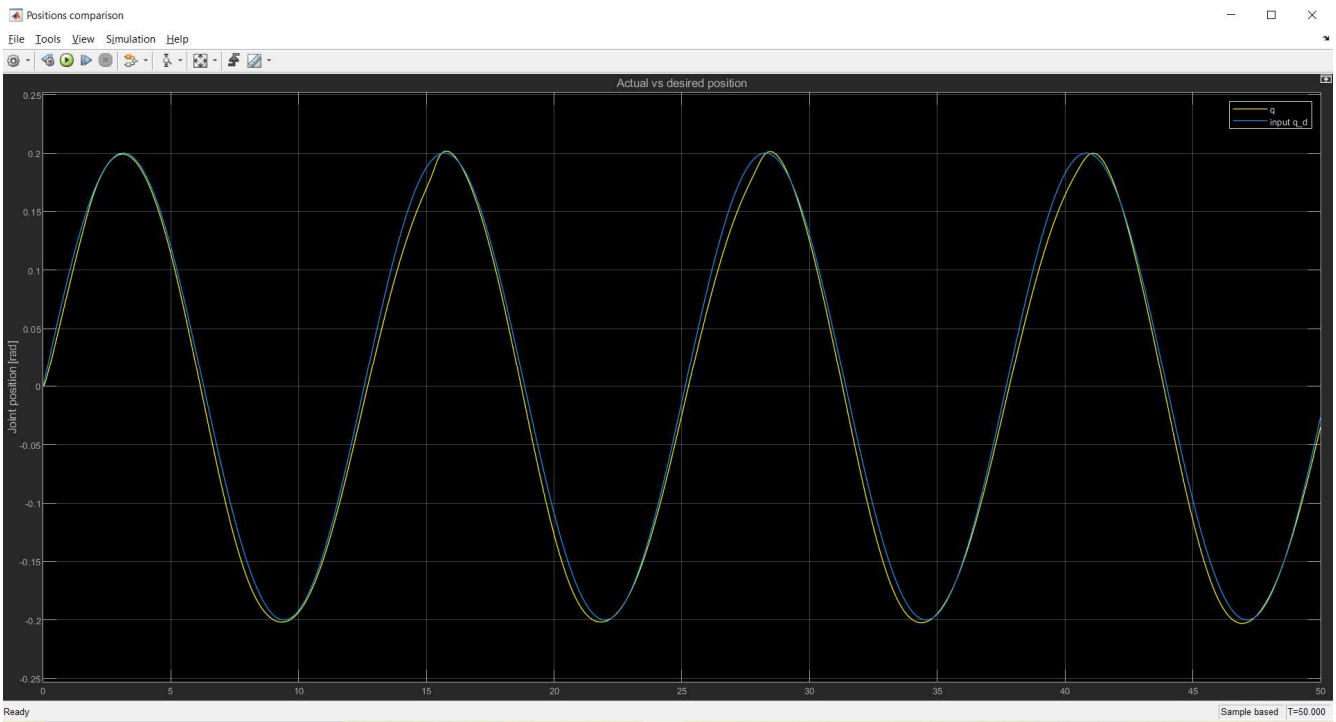
(2) Position tracking performance:



According to the theory, ***adaptive control*** is a well performing technique if the estimated model has the same properties of the real one. In particular, if the real plant's model is something like this, having an additional friction term, results could be worse: $I\ddot{q} + F_1\dot{q} + F_2\dot{q}^2 + G\sin q = \tau$

```
%"Normal" myRobot_1DOF modeling
sys(1:myRobot_1DOF.dof)= -inv(B)*(F*Dq + G*sin(q)  - u);

%"Wrong plant modeling" system (1)
F2 = 10;
sys(1:myRobot_1DOF.dof)= -inv(B)*(F*Dq + F2*Dq^2 + G*sin(q)- u);

%"Wrong plant modeling" system (2)
F2 = 10;
G2 = 3; %just to change further the dynamic model
sys(1:myRobot_1DOF.dof)= -inv(B)*(F*Dq + F2*Dq^2 + G*sin(q) ...
                                  + G2*cos(q)  - u);
```

Testing the first case (the one suggested in the slides), there aren't visible changes in performance, unless F_2 isn't increased by a lot. Testing the second scenario that also affects gravity (maybe due to a wrong modeling of the masses), more evident position tracking mismatches are achieved.



Assignment 9

- **Operational Space PD control law with gravity compensation**

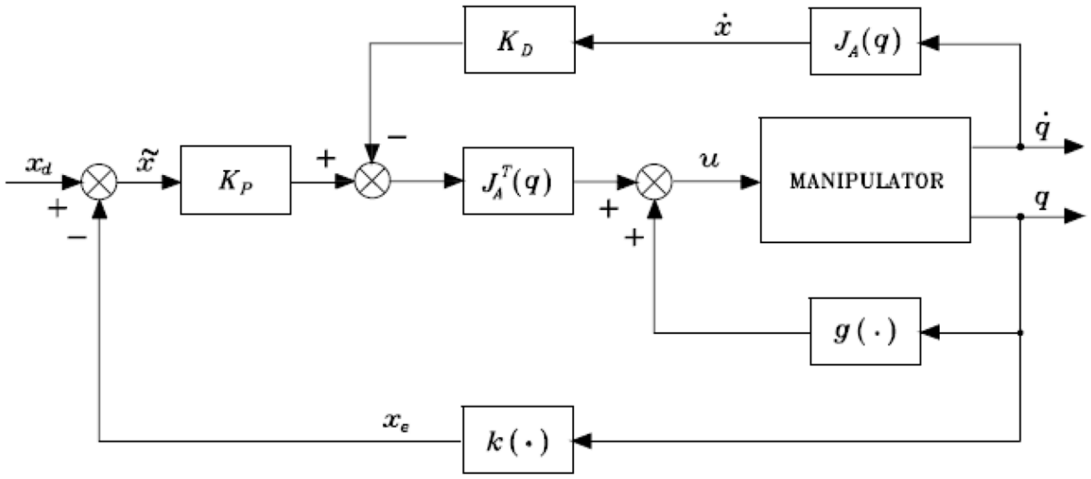
The PD control law with gravity compensation is now applied in *operational space*, being common to compute desired trajectories in cartesian space (x-y-z). It can be proven that performing a *nonlinear compensation* of joint space gravitational forces and an operational space linear *PD control action*, asymptotically stable convergence to the desired pose is ensured. The following control laws are equivalent, the second one will be implemented.

$$\tau = g(q) + J_A^T(q)K_P\tilde{x} - J_A^T(q)K_D J_A(q)\dot{q}$$

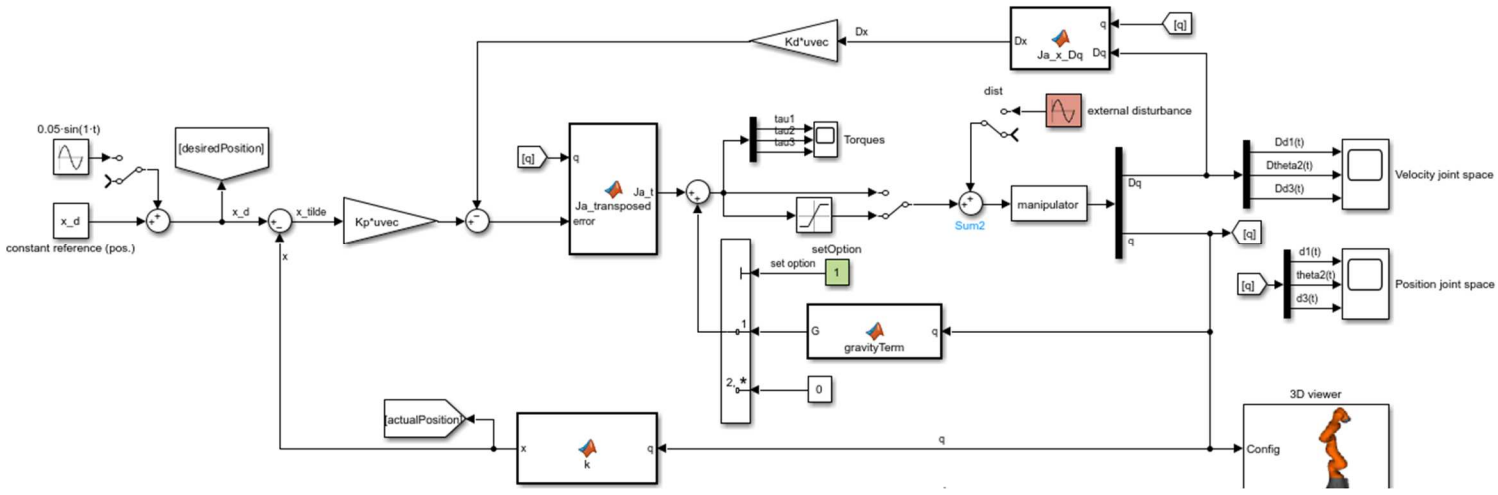
$$\tau = g(q) + J_A^T(q)K_P\tilde{x} - K_D\dot{q}$$

Also, the *analytical Jacobian* is used to relate linear and angular velocities, while *direct kinematics* is used to relate linear and angular positions.

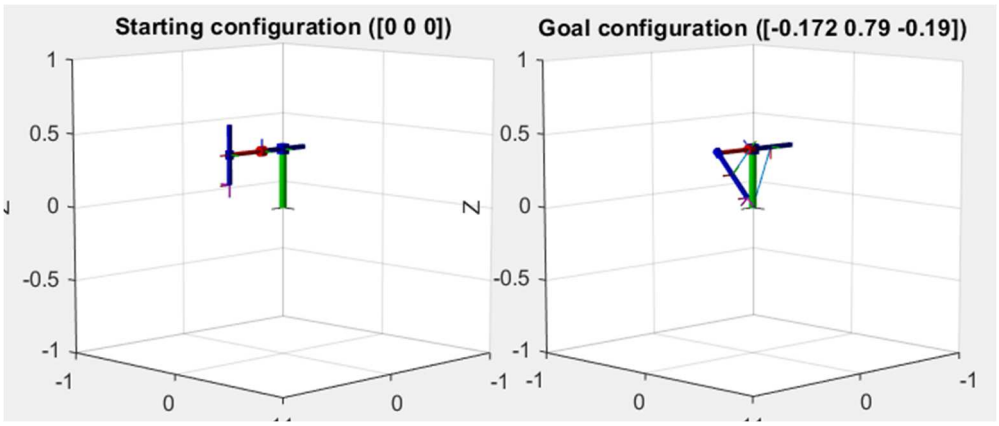
Operational Space PD control with gravity compensation scheme:

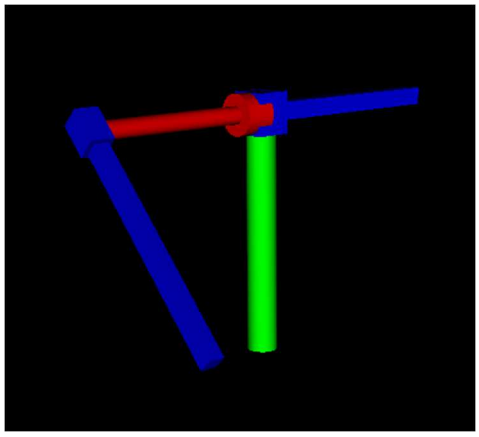


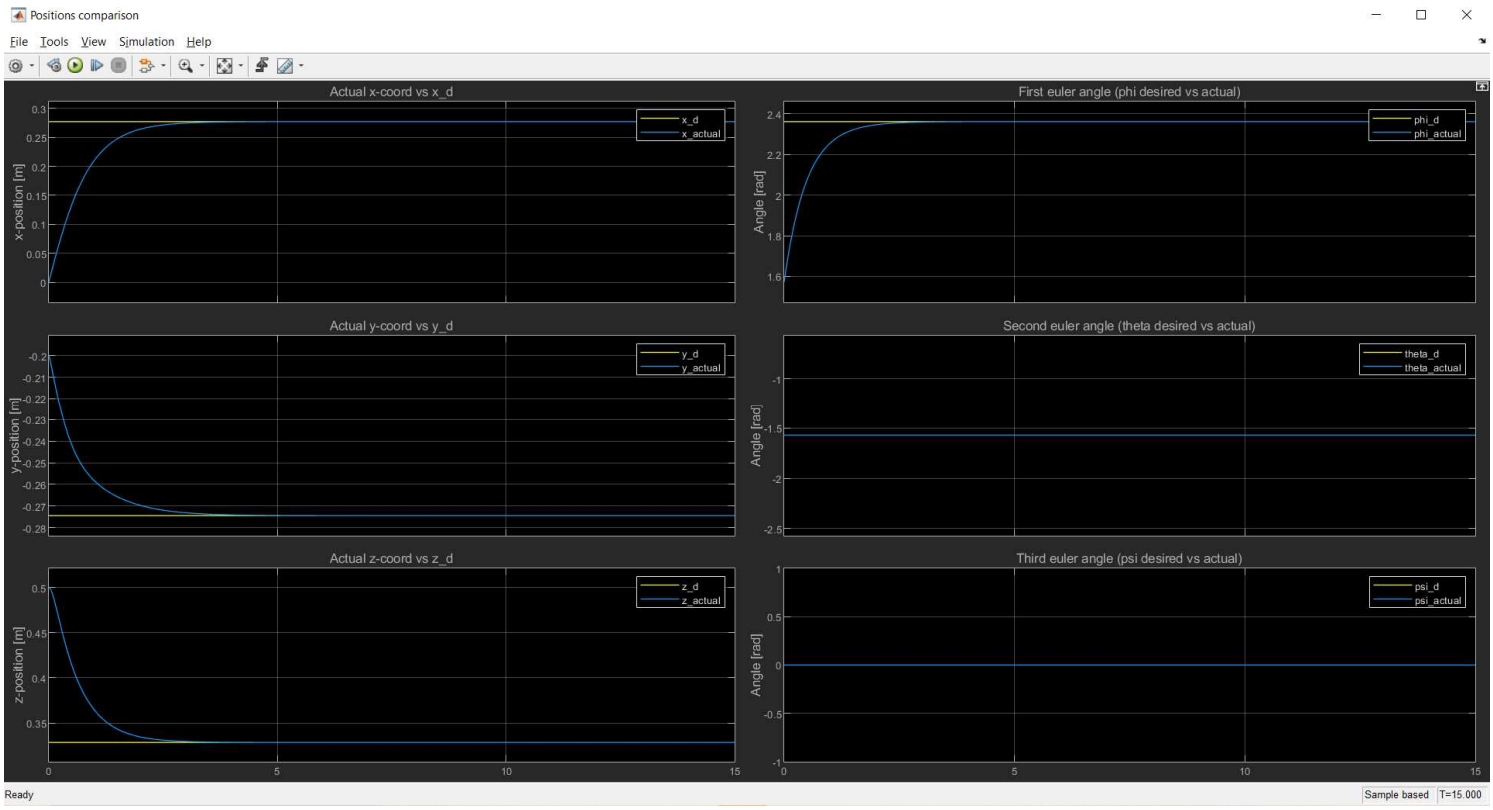
My Simulink model:



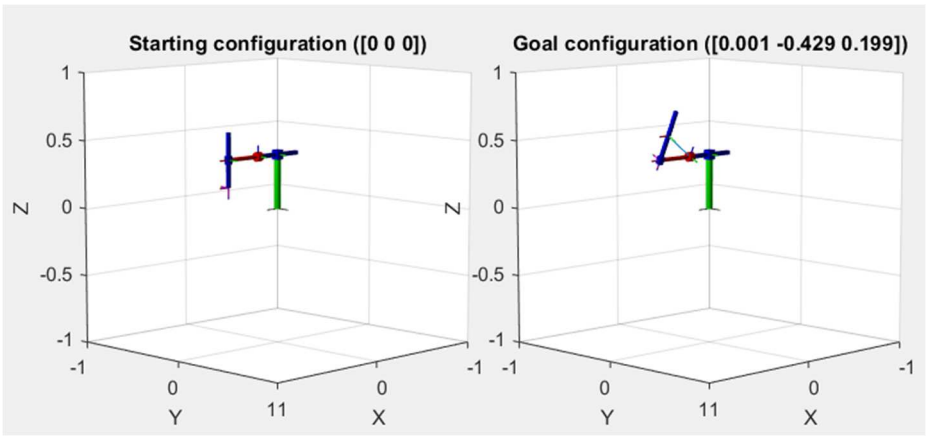
Goal configuration visualization, simulation and results:

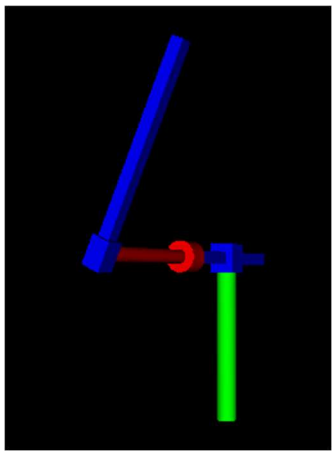


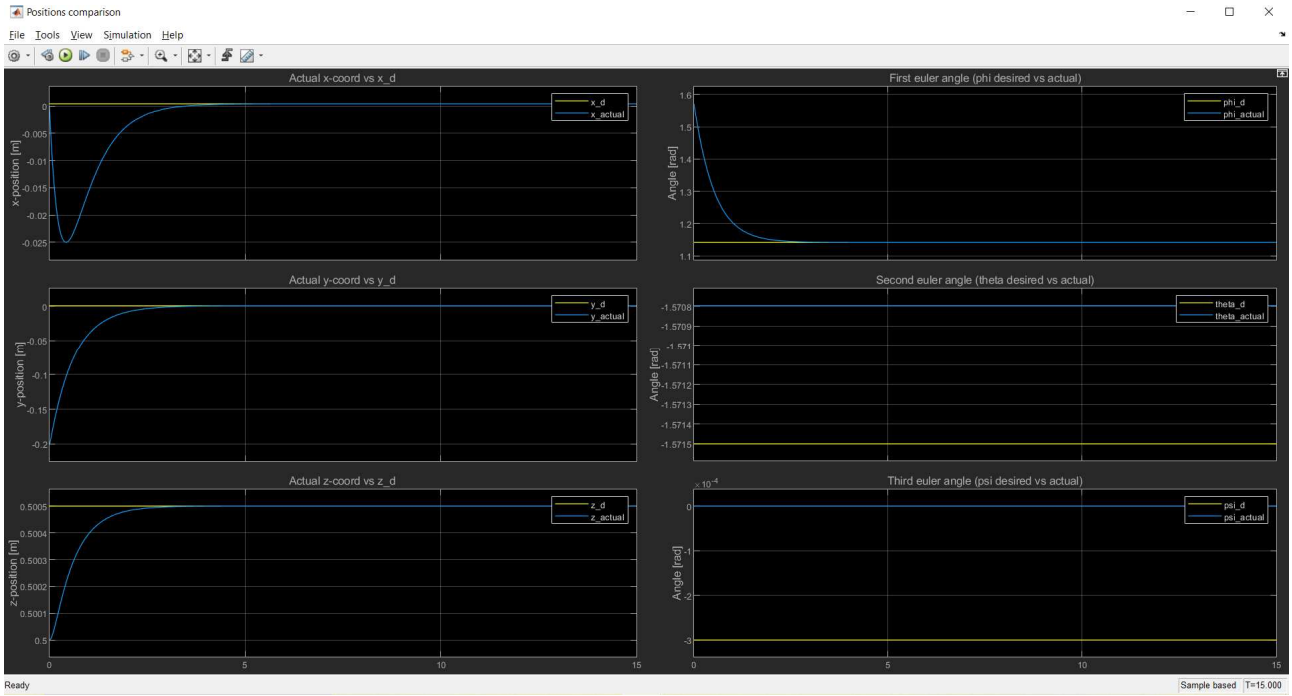




Almost singular goal configuration visualization, simulation and results:






Positions comparison
File Tools View Simulation Help
Actual x-coord vs x_d
x_d
x_actual
x-position [m]
First euler angle (phi desired vs actual)
phi_d
phi_actual
Angle [rad]
Actual y-coord vs y_d
y_d
y_actual
y-position [m]
Second euler angle (theta desired vs actual)
theta_d
theta_actual
Angle [rad]
Actual z-coord vs z_d
z_d
z_actual
z-position [m]
Third euler angle (psi desired vs actual)
psi_d
psi_actual
Angle [rad]
Ready
Sample based
T=15.000

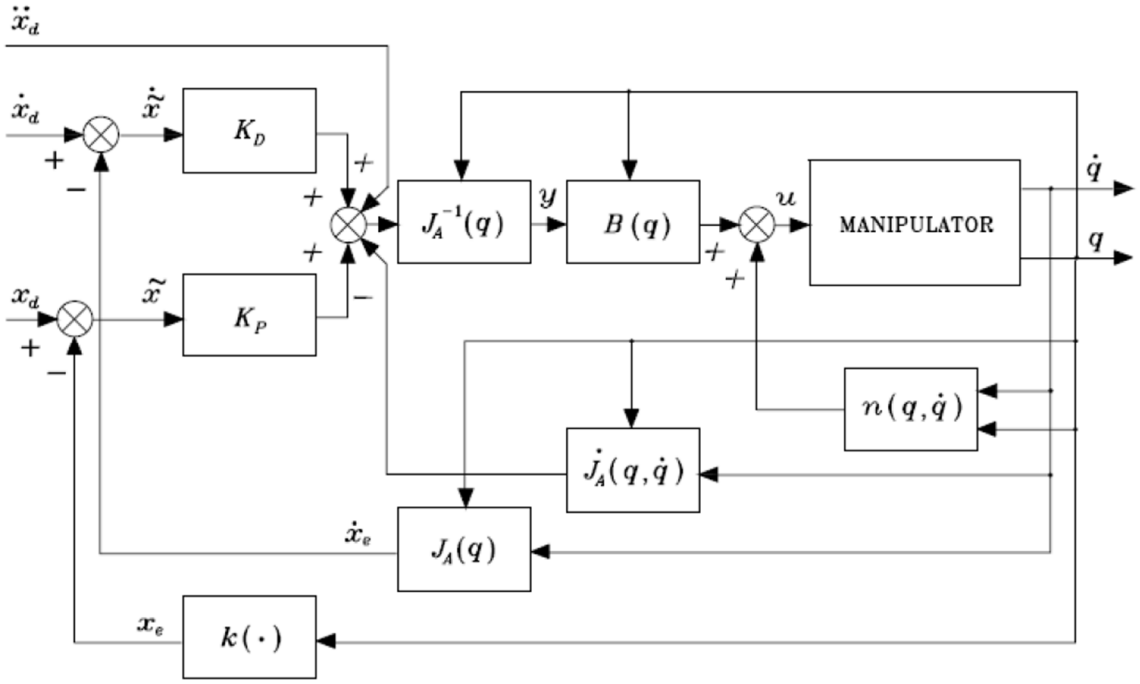
Assignment 10

- **Operational Space Inverse Dynamics Control**

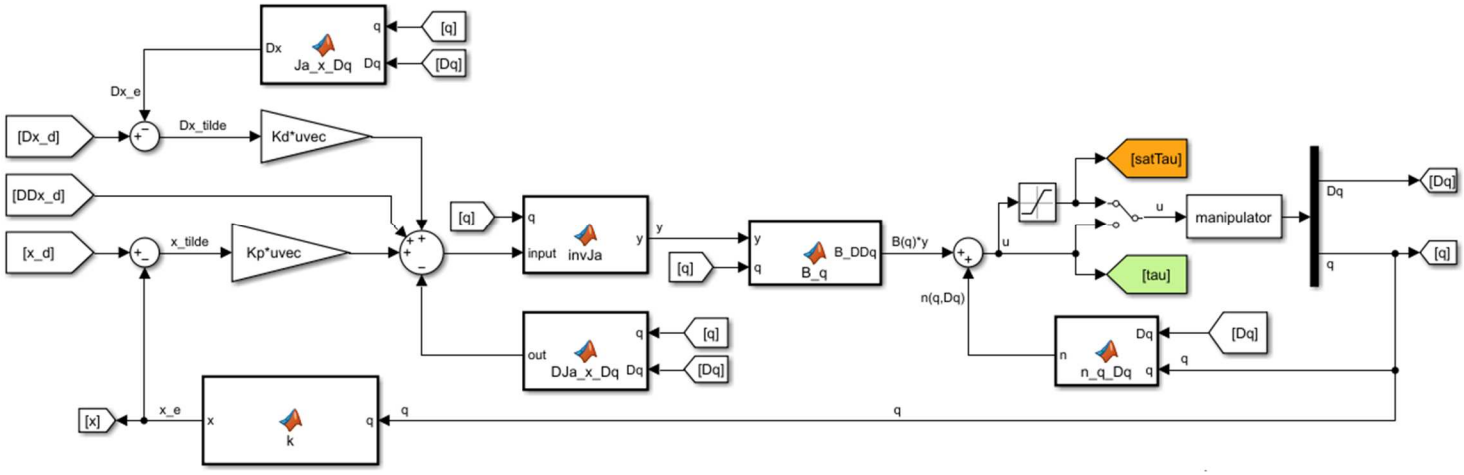
In a similar way to the joint space inverse dynamics control, the goal is to solve a *tracking* problem by cancelling out the nonlinearity terms and decoupling the 3 joint variables. The linearized system is then stabilized through the outer loop control law. In the assumption of non-redundant manipulators, Jacobians and direct kinematics can be exploited to create the model.

$$\tau = B(q)\left[J_A^{-1}(q)(\ddot{x}_d + K_D\dot{\tilde{x}} + K_P\tilde{x} - \dot{J}_A(q,\dot{q})\dot{q})\right] + C(q,\dot{q})\dot{q} + g(q)$$

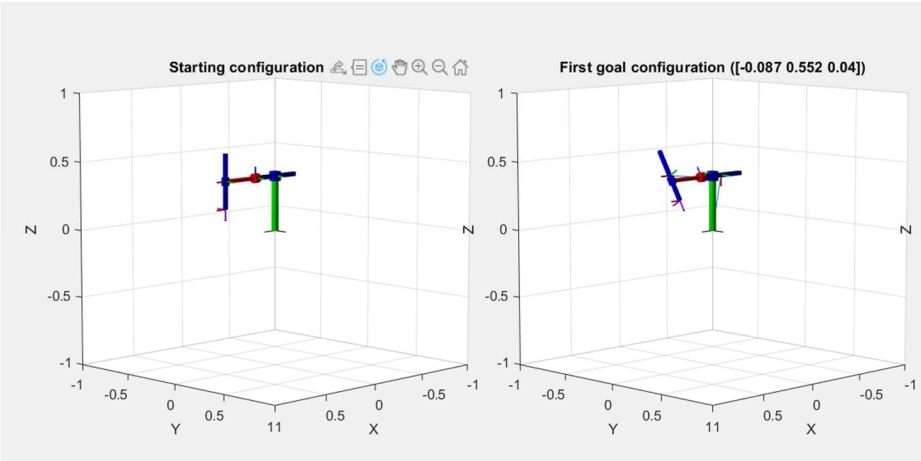
Operational Space Inverse Dynamics control scheme:

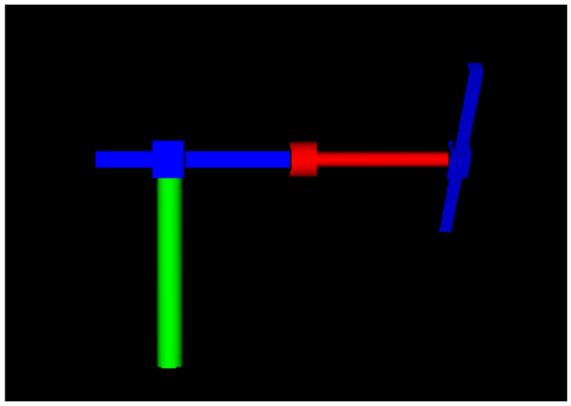


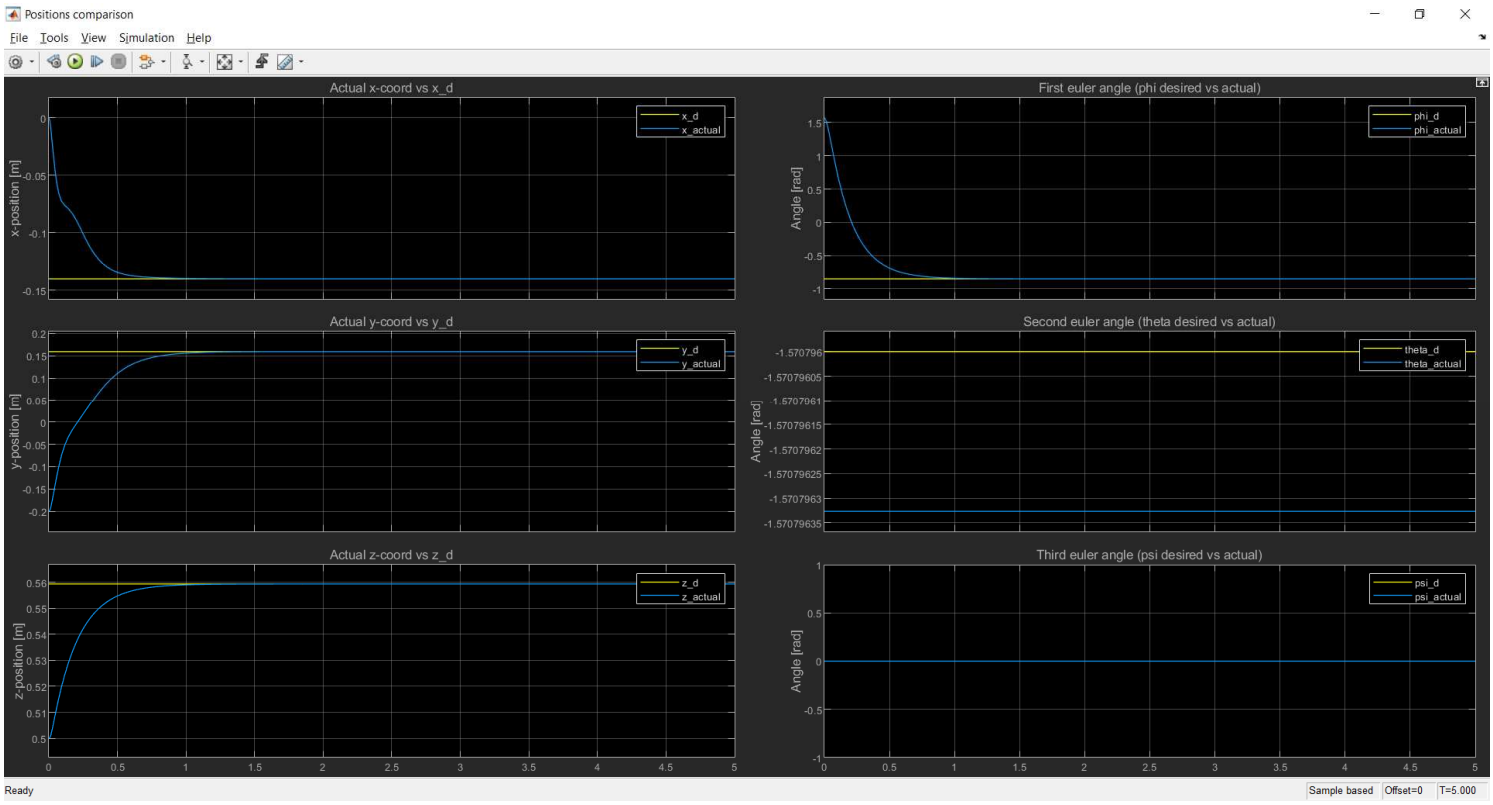
My Simulink model:



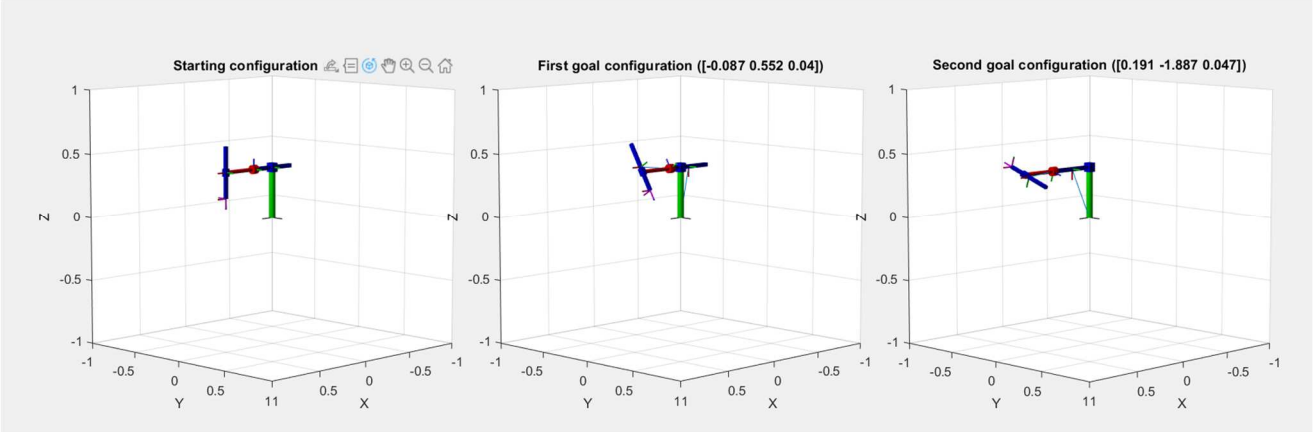
Constant goal configuration (regulation):

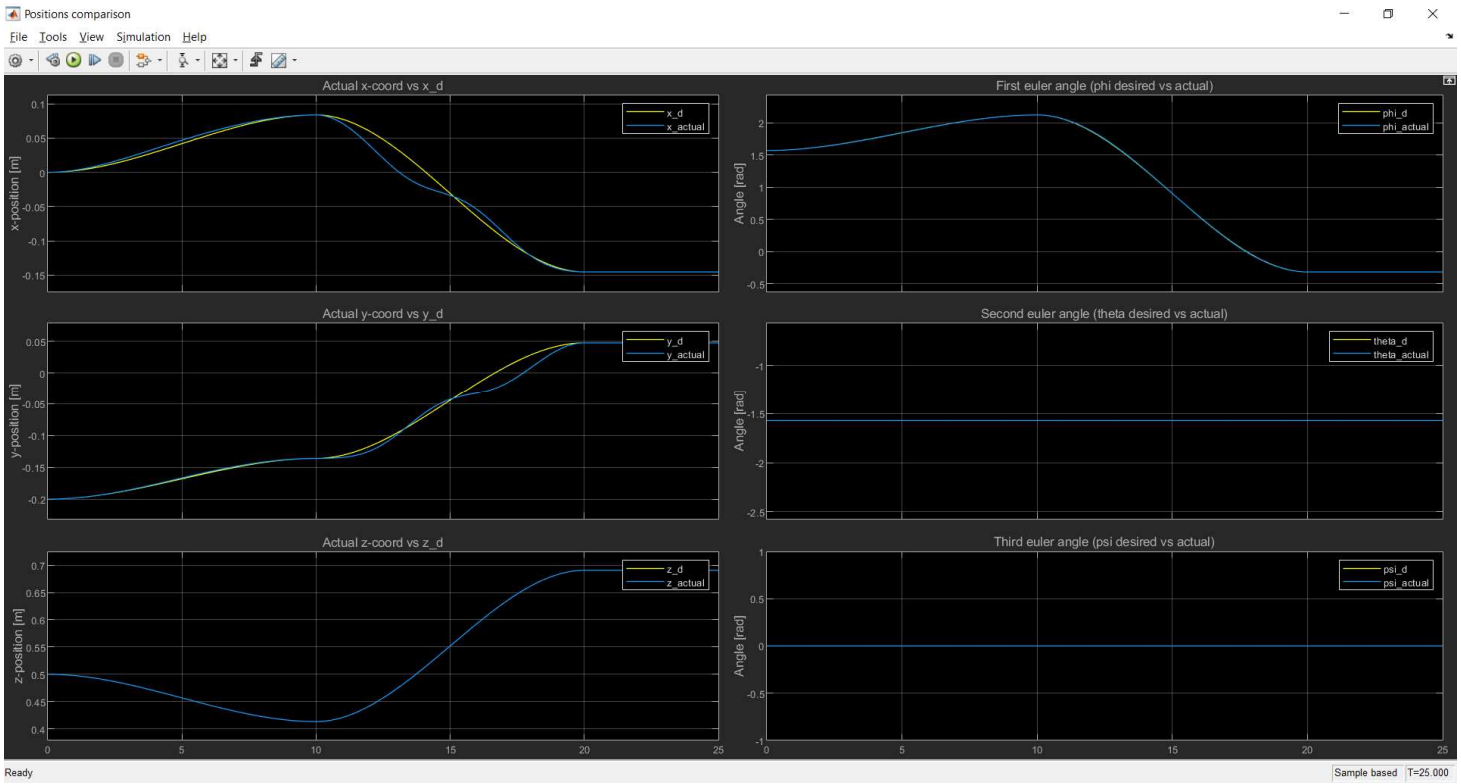




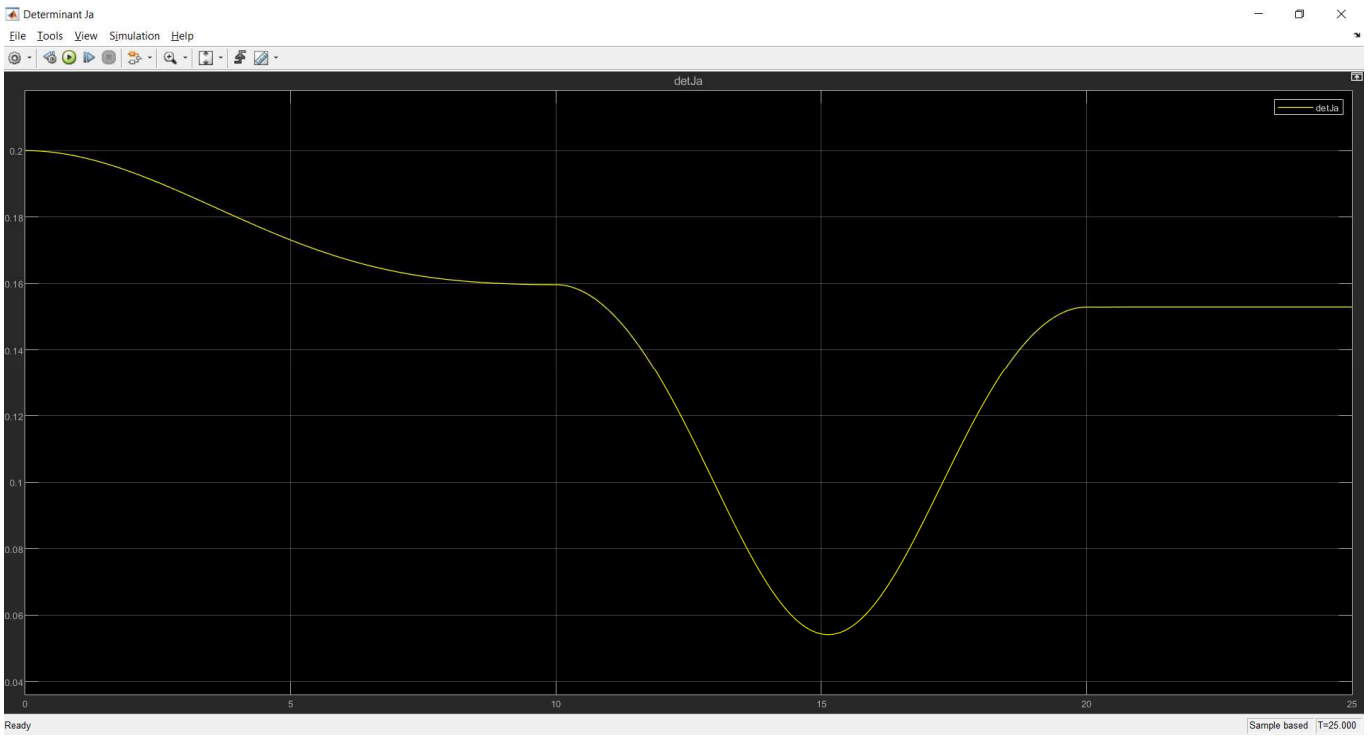


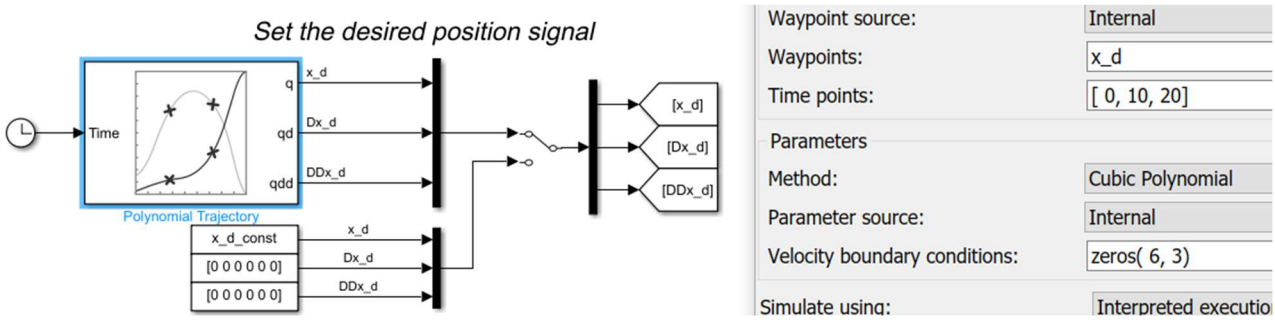
Two interpolated (by Simulink) random goal poses, to solve the tracking problem:





The first and second goal poses are randomly generated inside the MATLAB script. Even trying multiple times, not so good results are achieved along x and y. To explain this behavior, the ***singularity problem*** was studied. In fact, looking at how evolves Jacobian's determinant of its linear part, it drops to almost zero when passing from a pose to the next. This is probably due to the interpolation algorithm that Simulink is using.

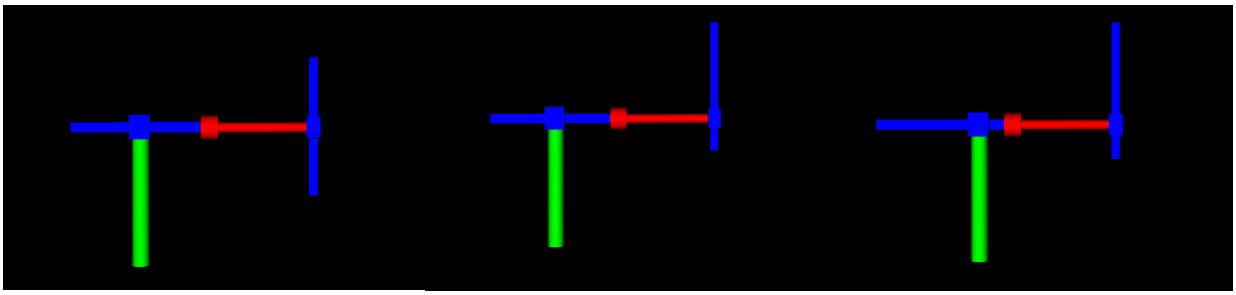


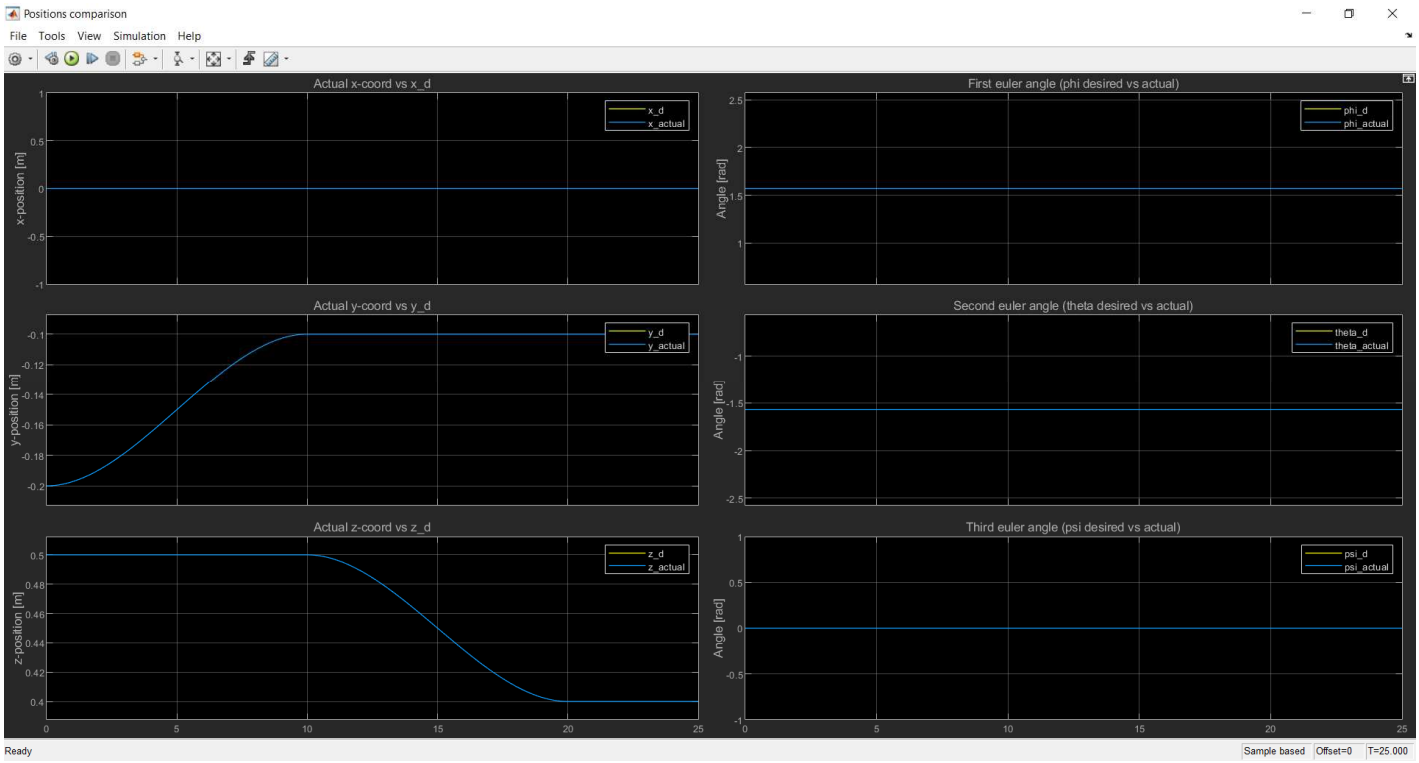


This is confirmed by the fact that the two goal poses are reached asymptotically and they don't have a low determinant in this case (10 and 20 seconds). The worst results are achieved at around 15s, in correspondence with the Jacobian's determinant drop leading to a singularity.

Setting an ***easier trajectory*** (that doesn't consider rotations)

```
x0          = [0 , -0.2, 0.5, pi/2, -pi/2, 0].';
goalPose1_0 = [0 , -0.1, 0.5, pi/2, -pi/2, 0].';
goalPose2_0 = [0 , -0.1, 0.4, pi/2, -pi/2, 0].';
x_d = [x0 goalPose1_0 goalPose2_0]; %desired trajectory
```





Assignment 11

- **Study the Compliance Control**

When the manipulator interacts with the surrounding environment, the previuous control approaches are not suited because they neglect any external wrench. ***Compliance control*** is an indirect force control approach aimed at achieving the desired force through motion control, without an explicit force feedback loop. So, the dynamic model of the manipulator will also include the ***external forces and torques***, represented by $\boldsymbol{h_e}$. If h_e=0 the robot will freely move reaching the desired position, without any constraint.

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F\dot{q} + g(q) = \tau - J^T(q)h_e$$

What we're interested in is ***active compliance***, meaning that we want to be able to decide the manipulator's compliance through a control law. On the other hand, *passive compliance* relates to the structural mechanical properties. To implement the system, a ***desired*** *reference frame* $\boldsymbol{\Sigma_d}$ was introduced. Using the appropriate transformation matrices, the position error, the end-effector's orientation and displacement, the Jacobian matrix, are expressed with respect to the desired frame. The *PD control with gravity compensation* in operational space is given by the following equation.

$$\tau = g(q) + J_{A_d}^T(q,\tilde{x})\left(K_P\tilde{x} - K_D J_{A_d}(q,\tilde{x})\dot{q}\right)$$

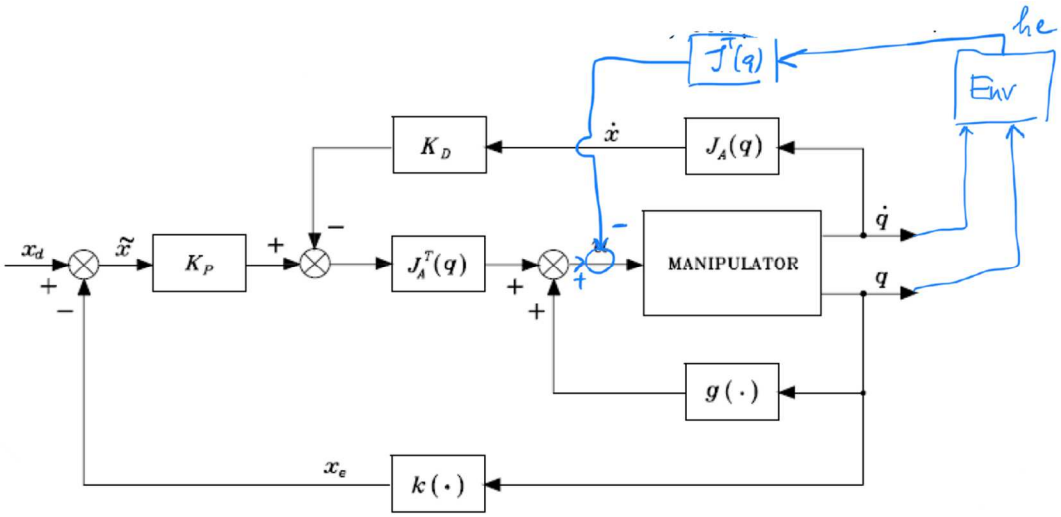
In our case, the environment was modeled as a generalized spring and a new reference frame $\boldsymbol{\Sigma_r}$ was introduced to represent the environment's ***rest*** position.

The interaction force between the end-effector and the environment is given by:

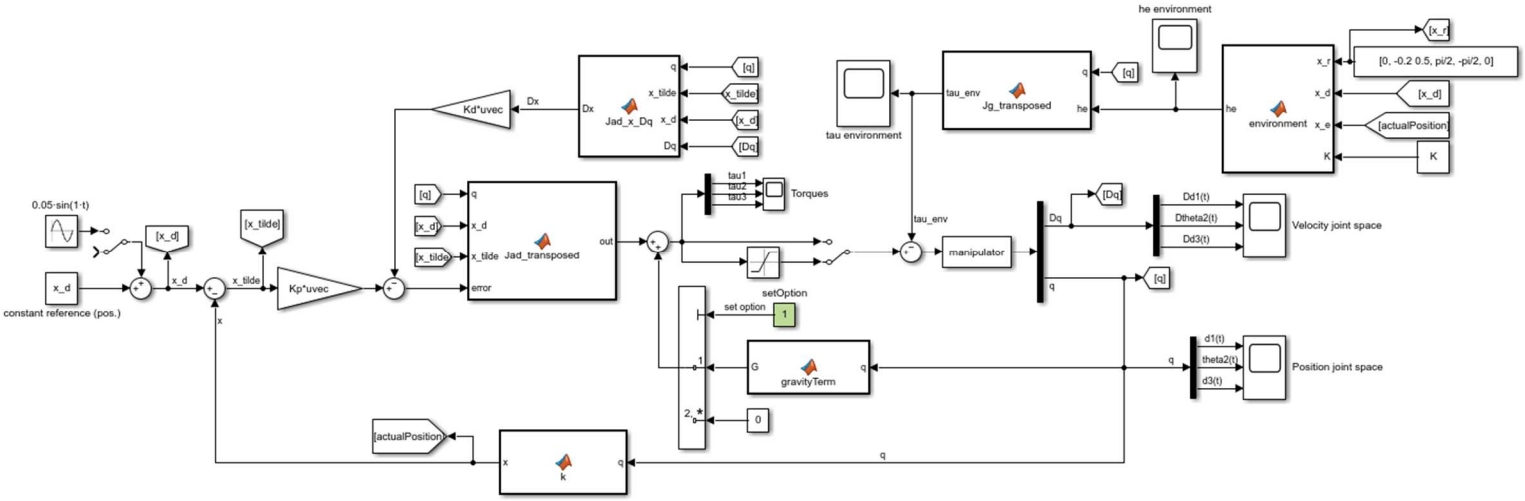
$h_e = K\, dx_{r,e}$; ***K*** is the environment stiffness (positive semi-definite) matrix, and $\boldsymbol{dx_{r,e}}$ is the elementary displacement between $\boldsymbol{\Sigma_r}$ and $\boldsymbol{\Sigma_e}$.

It's interesting to notice that the end-effector's pose error at the equilibrium depends on the *environment rest position*, on the chosen *desired pose*, as well as on the *compliance* (inverse of the stiffness) *matrices*. As it will be shown by the experiments, a softer environment will allow the end-effector to get closer to the desired position than a very stiff one.

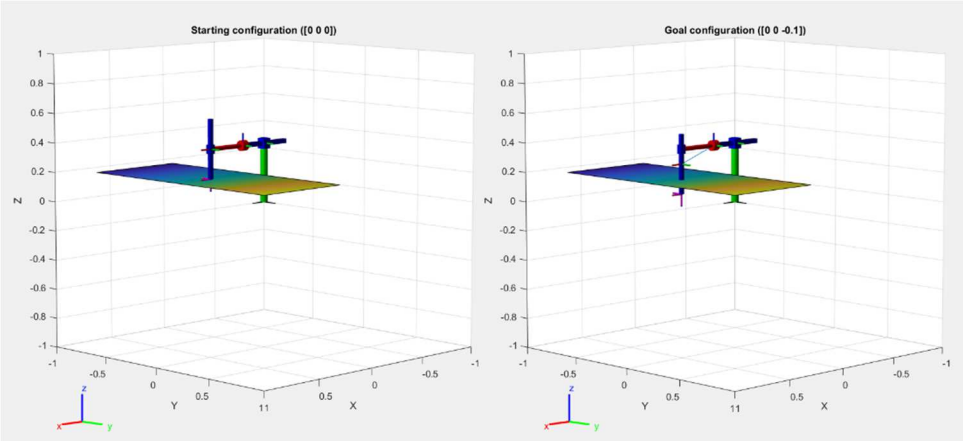
Operational Space PD with gravity compensation control scheme + environment:

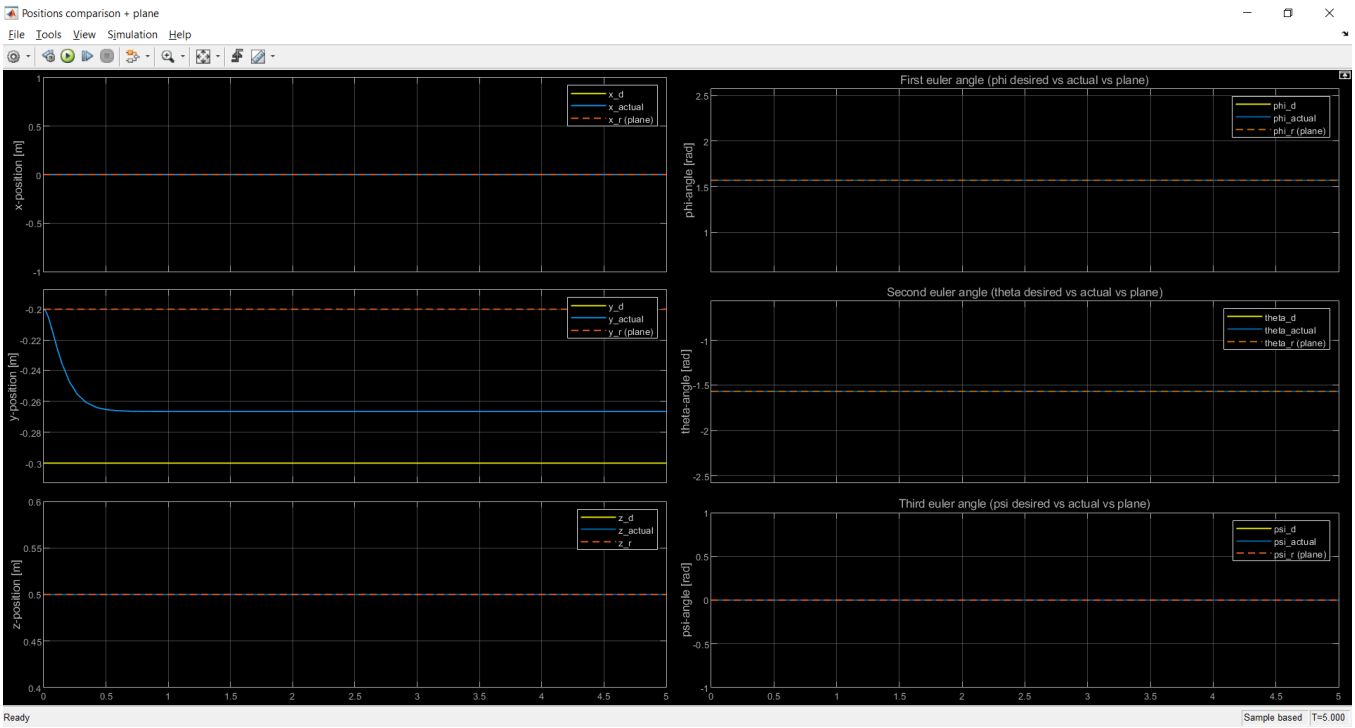


My Simulink model:



Manipulator interacting with a planar "medium stiffness" environment:





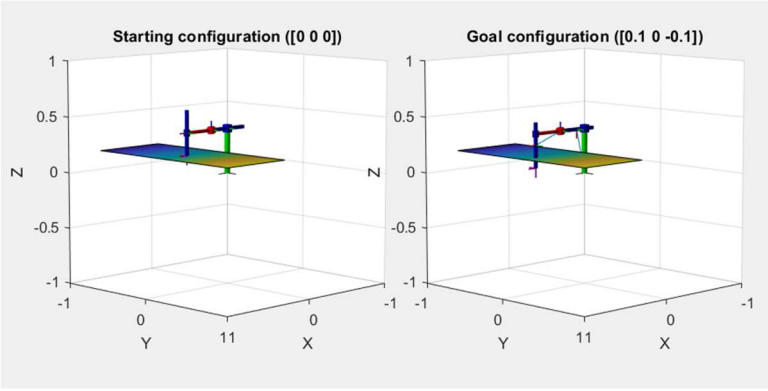
1) Along the directions where the manipulator stiffness is much higher than the environment stiffness, the intensity of the elastic force mainly depends on the stiffness of the environment and on the displacement $dx_{r,e}$ between the equilibrium pose of the end-effector (which practically coincides with the desired pose) and the rest pose of the environment;

2) along the directions where the environment stiffness is much higher than the manipulator stiffness, the intensity of the elastic force mainly depends on the manipulator stiffness and on the displacement $dx_{e,d}$ between the desired pose and the equilibrium pose of the end-effector (which practically coincides with the rest pose of the environment).

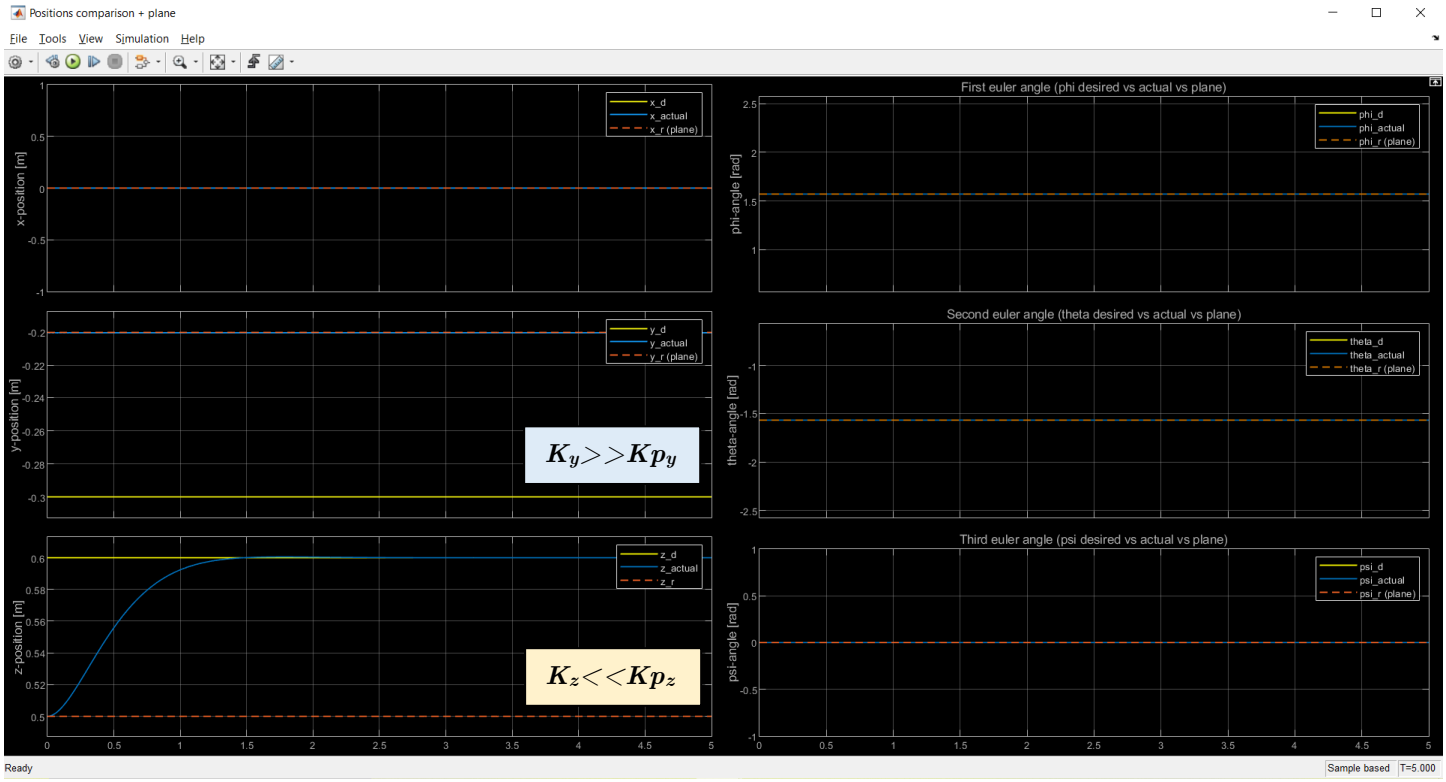
To show these two scenarios in the same simulation, the K environment stiffness matrix and the K_p controller matrix were set as such:

$$K = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10000 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

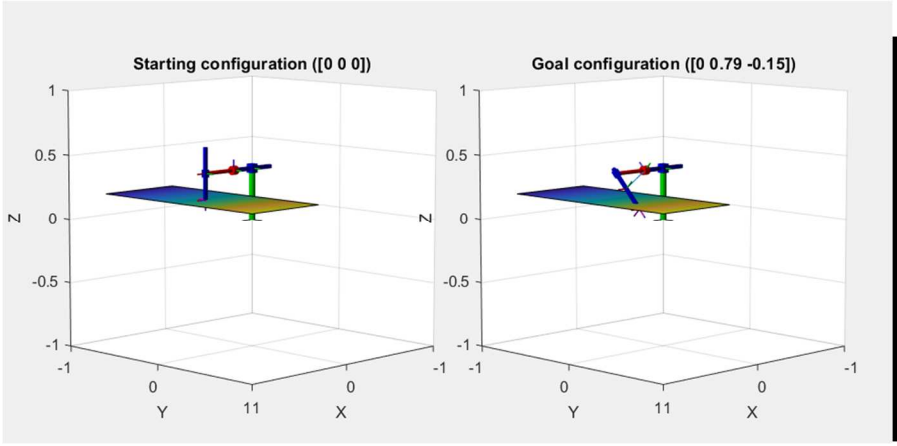
$$K_p = \begin{pmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 50 \end{pmatrix}$$

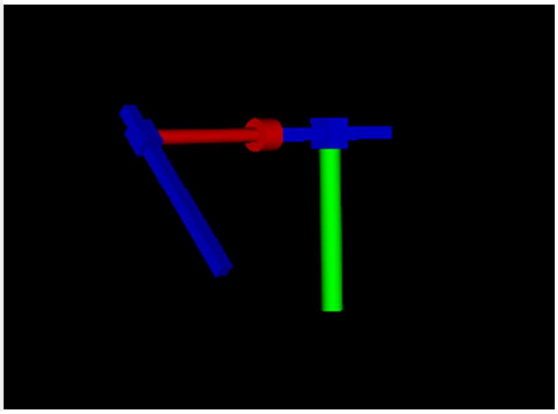


Meaning that, focusing on the linear coordinates, the environment stiffness $\boldsymbol{K_y >> Kp_y}$ manipulator stiffness, while $\boldsymbol{K_z << Kp_z}$. So, the manipulator will easily reach the desired coordinate z (thanks to the low environment stiffness) but basically it won't move along y (because of the high environment stiffness). As the simulation results show, we have the *first behavior along z* and the *second behavior along y.*

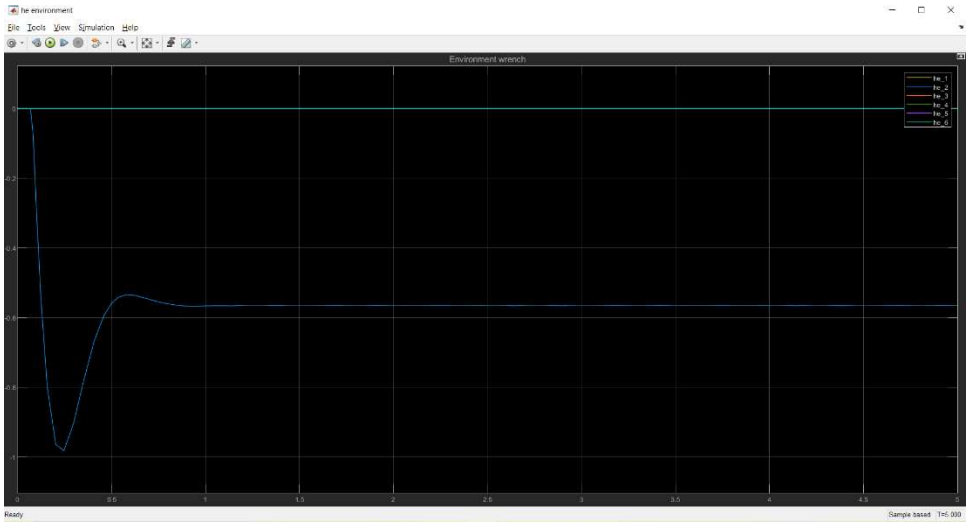


Manipulator interacting with the environment in a "diagonal" pose:

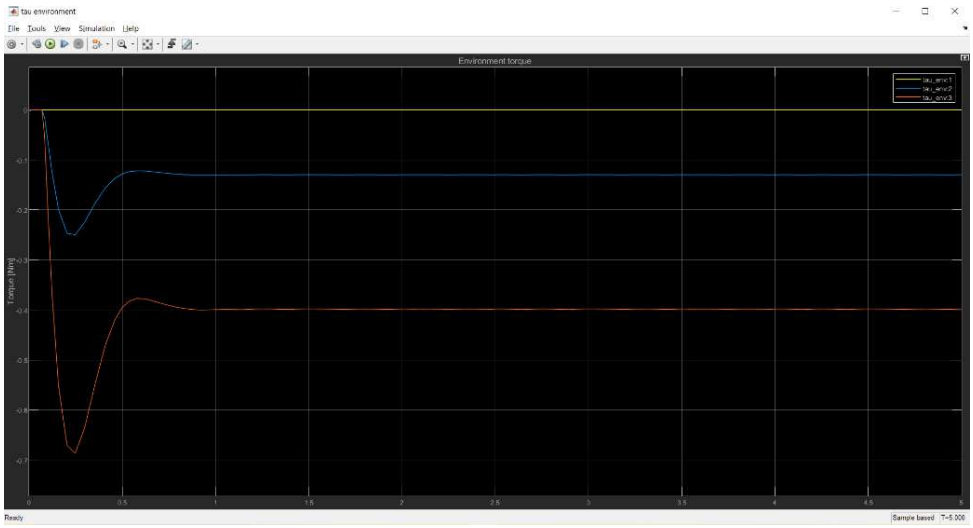




As shown by the next two scopes, the only component inside the *environment wrench* is along y, while the *environment torques*, reacting to the manipulator's desired pose, are about y and z.

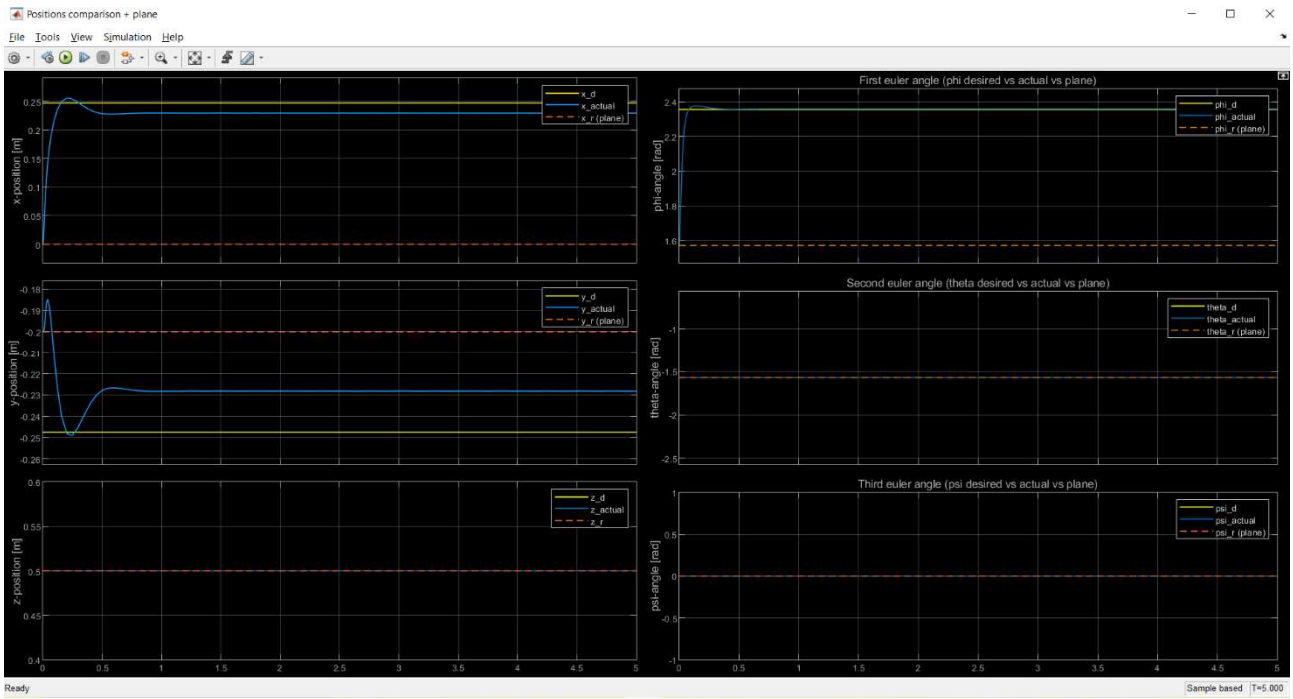


Environment wrench h_e



Environment torques τ_e

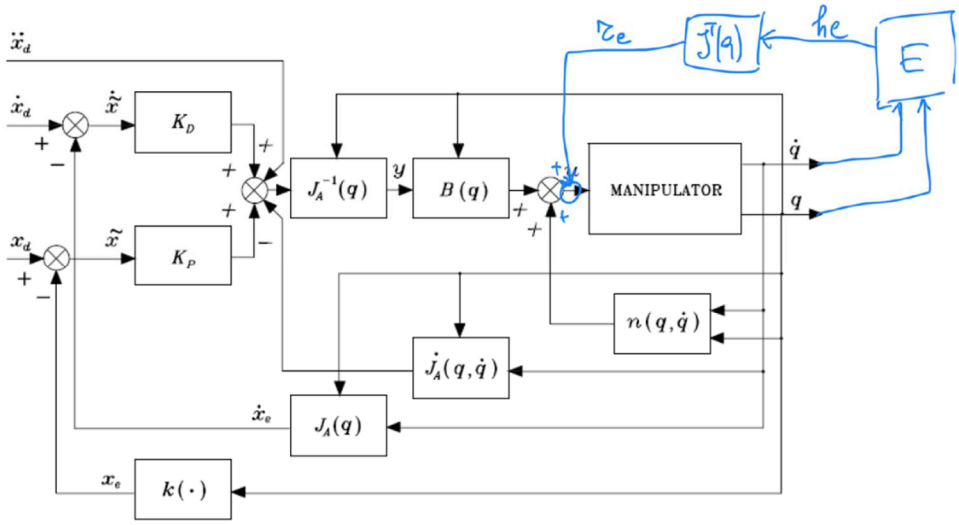
The simulation results:



Assignment 12

- **Implement the Impedance Control in the operational space**

In the previous assignment we started from the PD control with gravity compensation control scheme, while in this case we start from the *operational space inverse dynamics* scheme. Of course the interaction with the external environment has to be considered.



Impendence control, similarly to compliance control, is an *indirect* force control method. It isn't able to both control the motion and the interaction force, but the *port behavior* of the "robot+controller" system. A mechanical impedence allows the operational space mapping between force and displacement. In particular, the impedence is characterized my an *equivalent mass* matrix M_d , an *equivalent damping* matrix K_d and an *equivalent stiffness* matrix K_p. By measuring the contact wrench h_e, a linear and decoupled equivalent system can be obtained.

$$M_d\ddot{\tilde{x}} + K_D\dot{\tilde{x}} + K_P\tilde{x} = h_A$$

Due to the fact that the equivalent linear and decoupled system depends on the actual rotation matrix of the end-effector, it's difficult to relate the equivalent matrices M_d, K_d and K_p with the cartesian motion and rotation. So, the ***control law*** is designed as a *function of the error vector* $\tilde{x} = x_d - x_e$. Using the following relations, the final control law **y** is obtained.

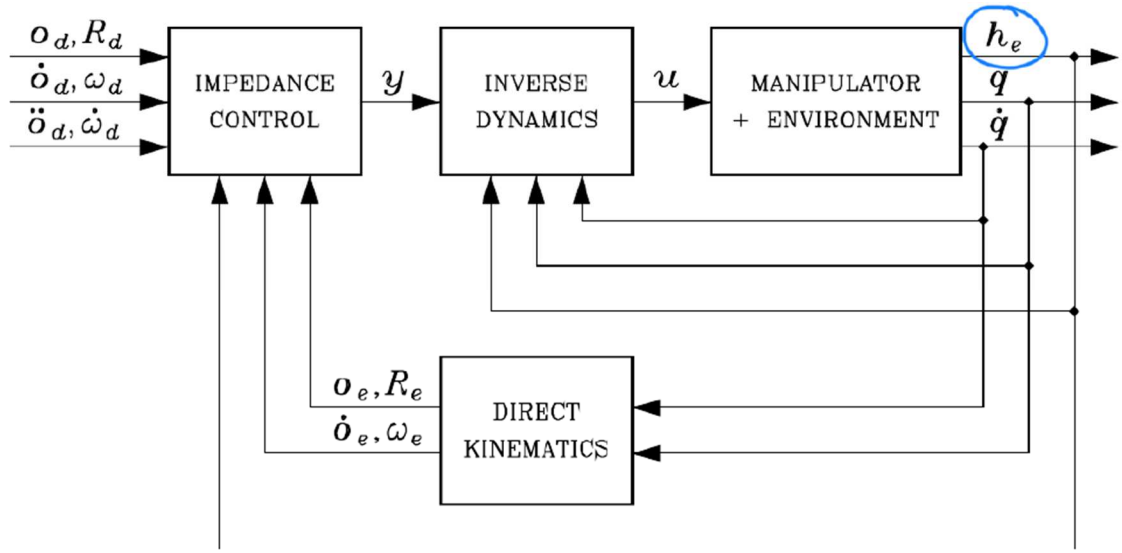
$$J_{A_d}(q,\tilde{x}) = T_A^{-1}(\phi_{d,e}) \begin{bmatrix} R_d^T & O \\ O & R_d^T \end{bmatrix} J(q) \qquad b(\tilde{x}, R_d, \dot{o}_d, \omega_d) = \begin{bmatrix} R_d^T \dot{o}_d + S(\omega_d^d) o_{d,e}^d \\ T^{-1}(\phi_{d,e})\omega_d^d \end{bmatrix}$$

$$\tilde{x} = x_d - x_e = -\begin{bmatrix} o_{d,e}^d \\ \phi_{d,e} \end{bmatrix} \qquad \dot{\tilde{x}} = -J_{A_d}(q,\tilde{x})\dot{q} + b(\tilde{x}, R_d, \dot{o}_d, \omega_d) \qquad \ddot{\tilde{x}} = -J_{A_d}\ddot{q} - \dot{J}_{A_d}\dot{q} + \dot{b};$$

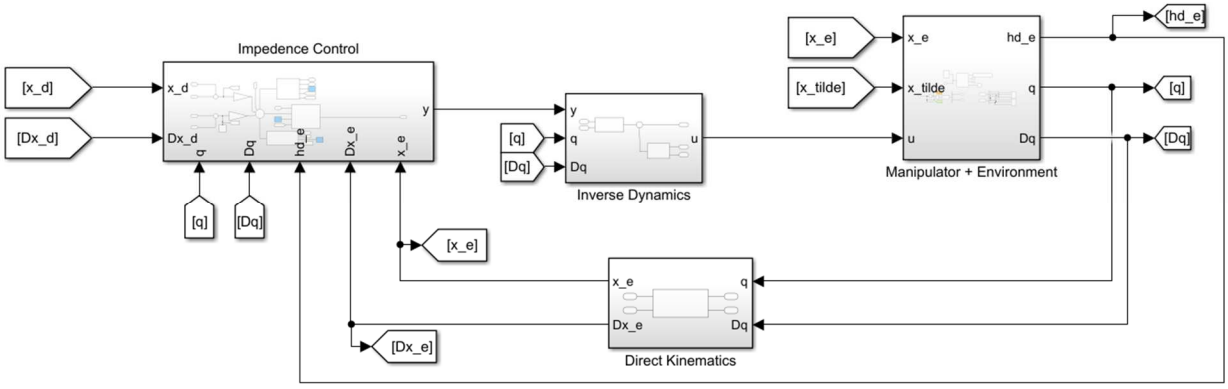
The final control law, with all the vectors refered with respect to Σ_d:

$$y = J_A^{-1}(q)M_d^{-1}\left(K_D\dot{\tilde{x}} + K_P\tilde{x} - M_d\dot{J}_{A_d}(q,\dot{q})\dot{q} - M_d b(\tilde{x}, R_d, \dot{o}_d, \omega_d) - h_e^d\right)$$

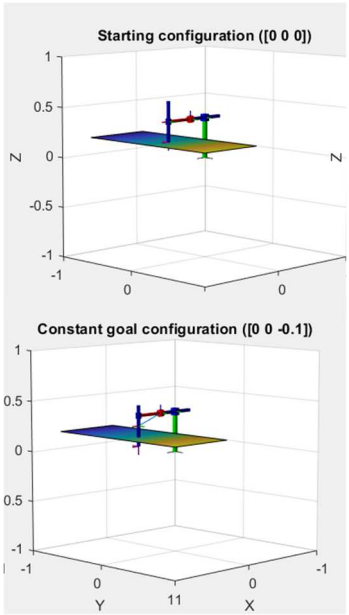
The compact impedence control block scheme:

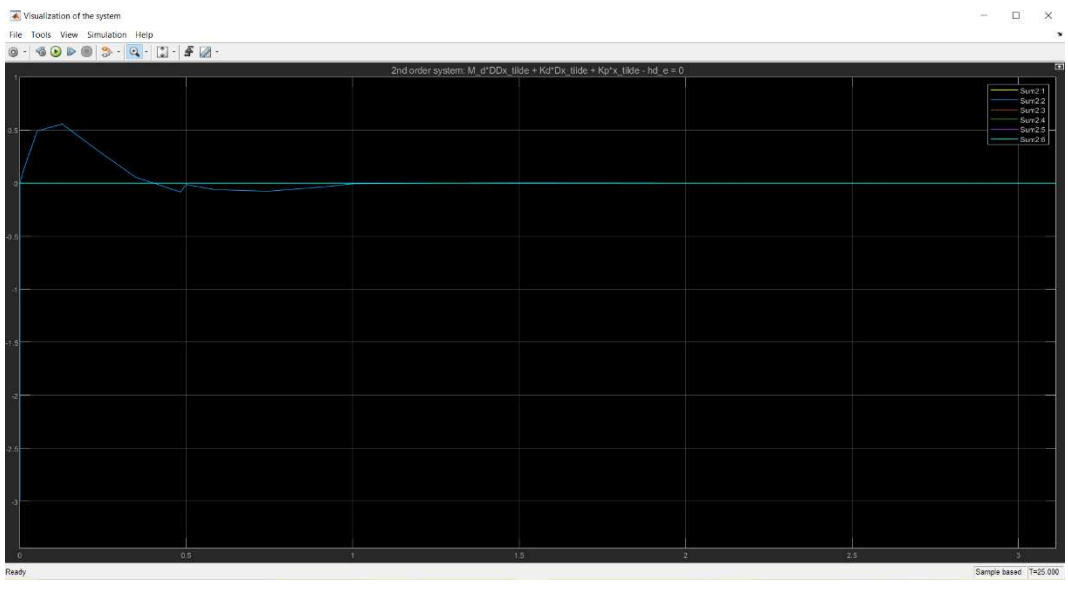


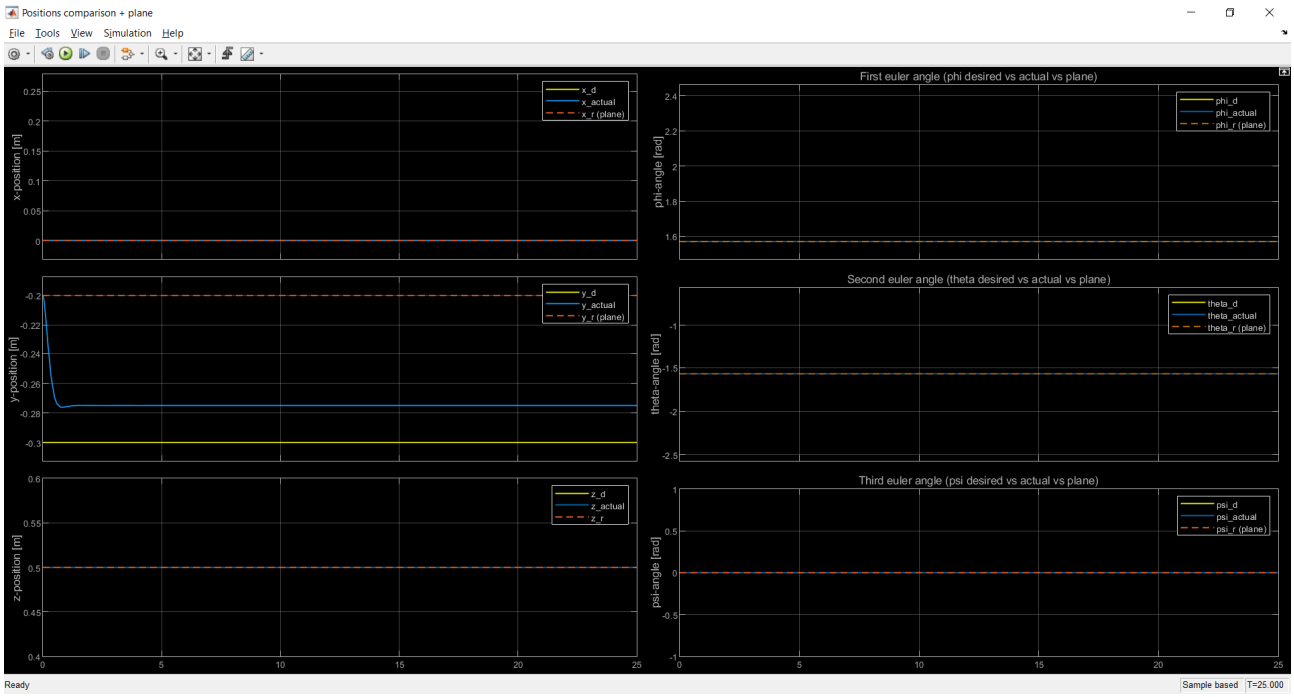
My Simulink model, to resemble the above scheme:



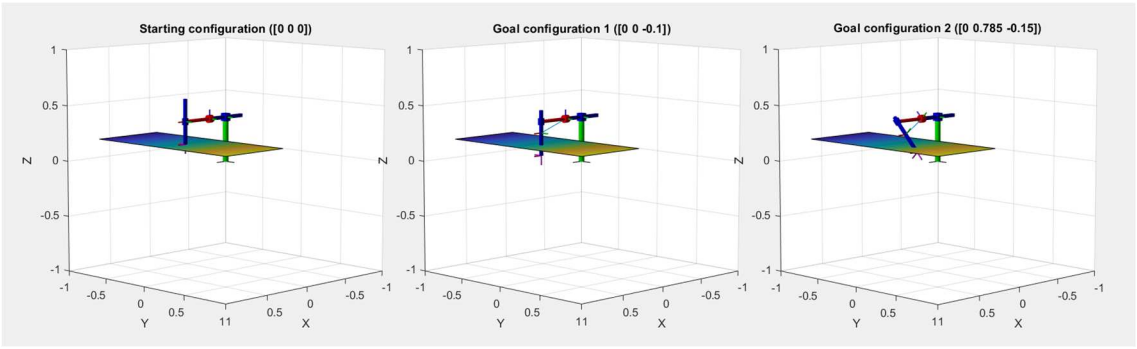
Achieved performances in a regulation problem (interacting with the environment) and visualization of the equivalent second order system:

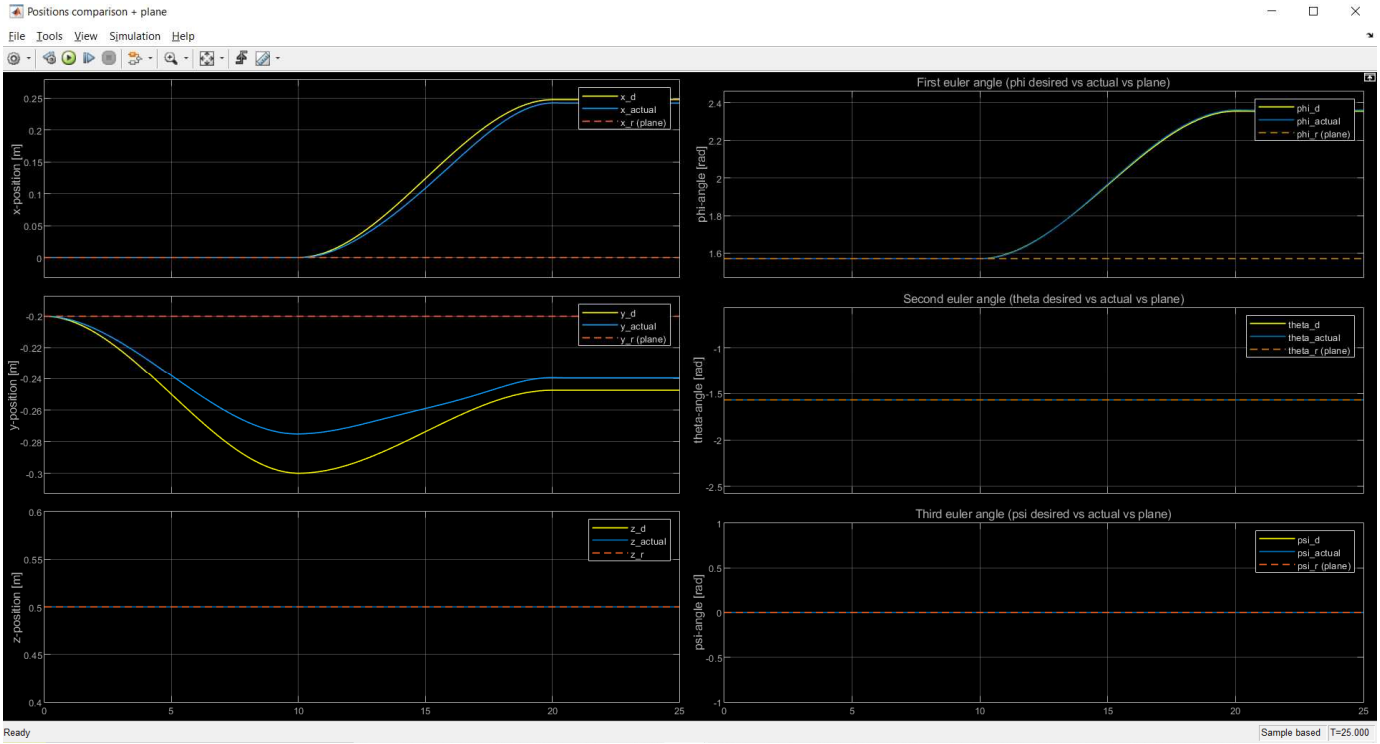






Achieved performances in a tracking problem:





Assignment 13

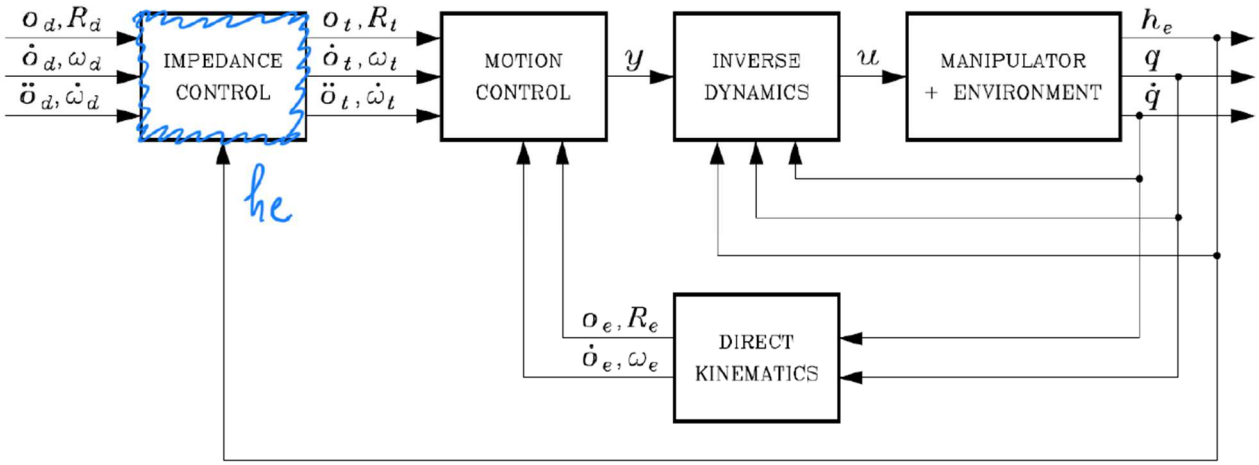
- **Implement the Admittance Control in the operational space**

Admittance control is a force control strategy that allows to separate the motion control problem from the impedance control problem, introdcuting a suitable ***compliant frame*** $\mathbf{\Sigma_t}$ to describe the ideal end-effector behavior under impedance control. In fact, using impedence control a tradeoff between compliant impedance control (low K_p) and high disturbance rejection/model robustness (high K_p) has to be found.

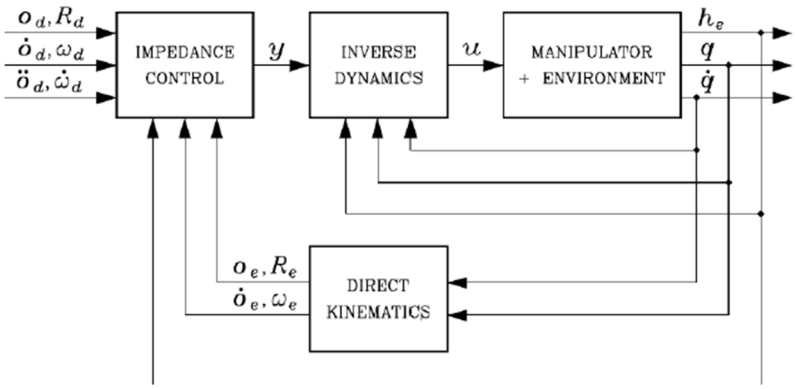
The first block of the scheme, called "*impedence control*" is used as an admittance to map the measured force h_e^d (interaction wrench at the end-effector w.r.t Σ_d) into velocity and position. The equation that describes this behavior is the following one ($\tilde{z}$ represent the displacement between Σ_t and Σ_d):

$$M_t\ddot{\tilde{z}} + K_{Dt}\dot{\tilde{z}} + K_{Pt}\tilde{z} = h_e^d \qquad \tilde{z} = x_d - x_t = -\begin{bmatrix} o_{d,t}^d \\ \phi_{d,t} \end{bmatrix}$$

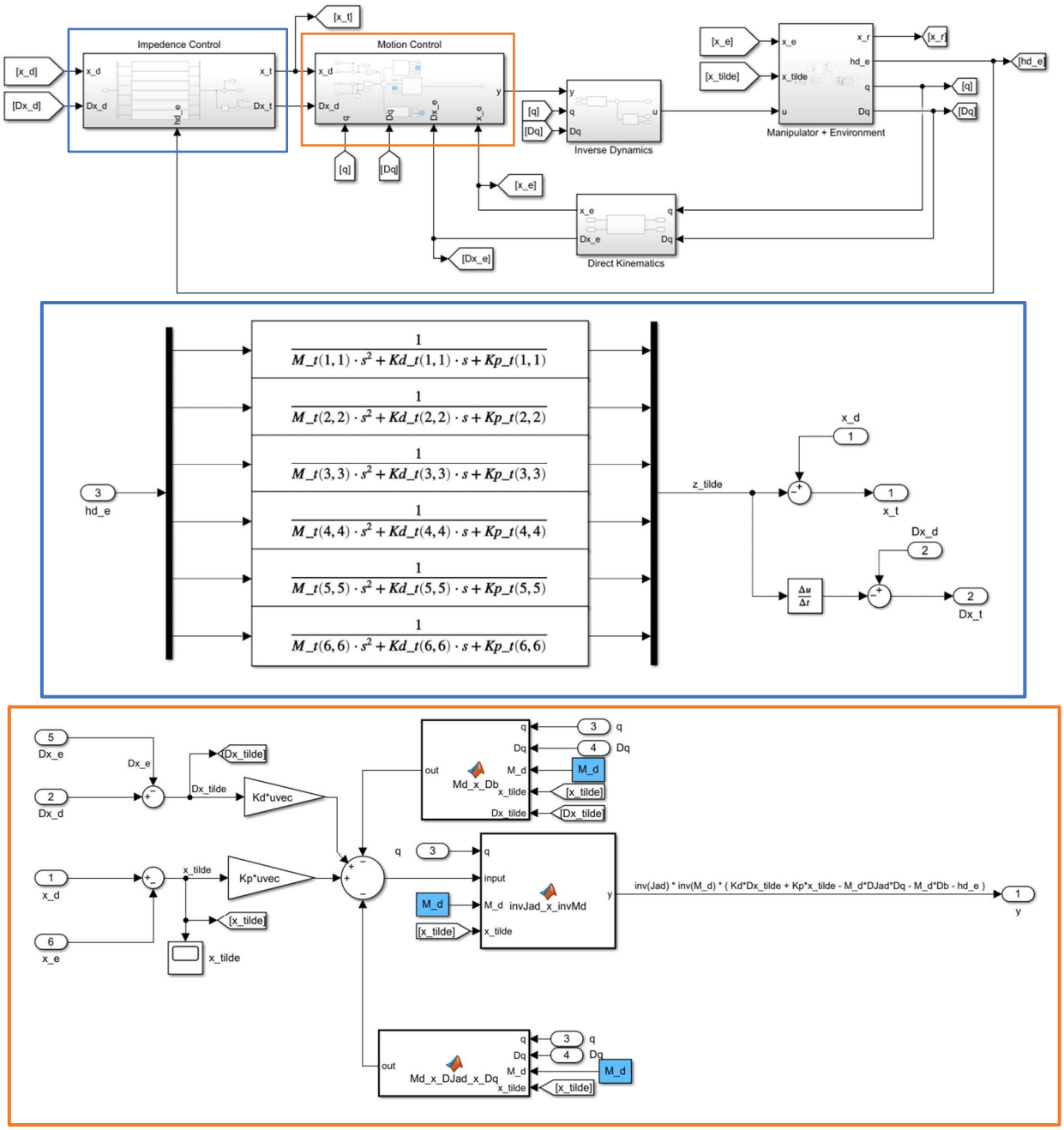
The admittance control scheme:



Its similarity with the impedence control scheme (Assignment 12):



My Simulink model:



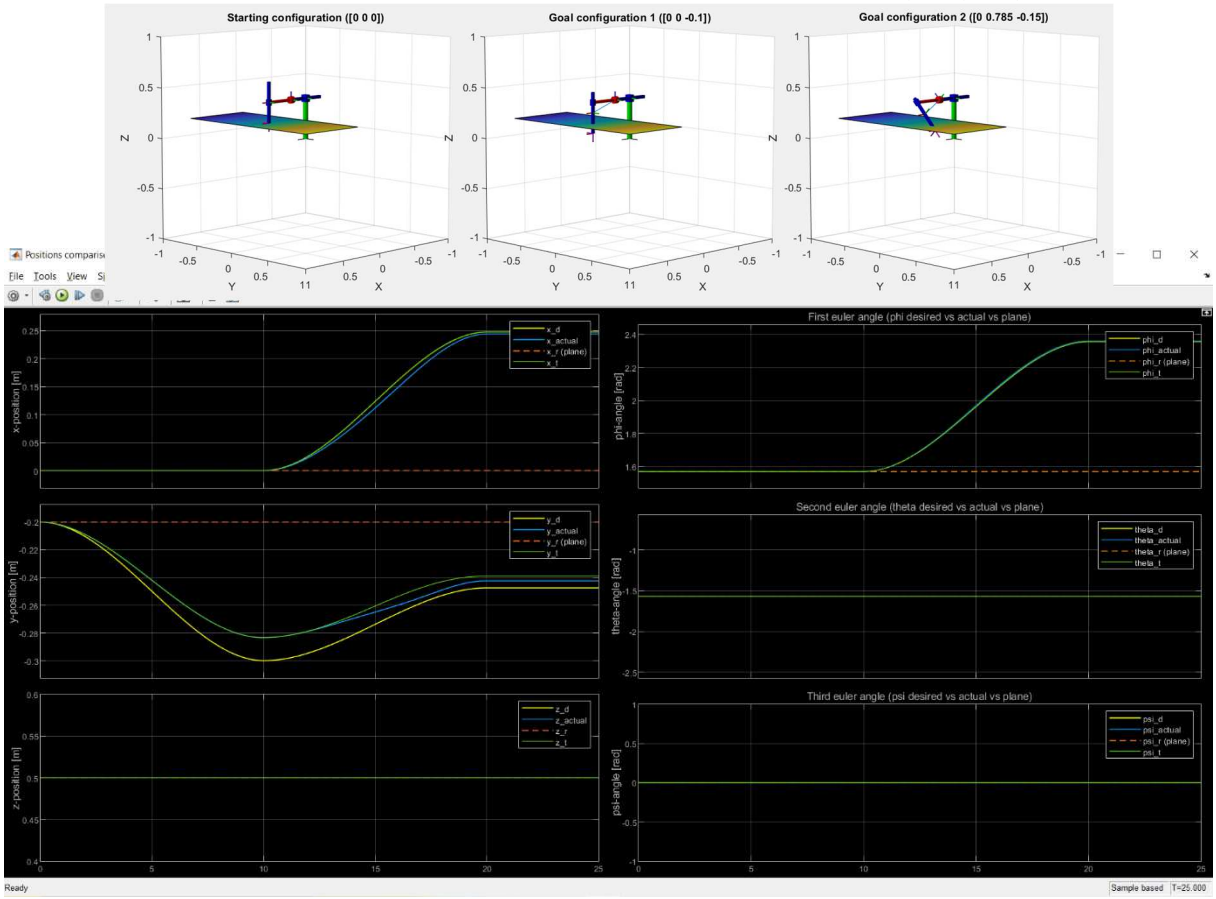
During the parameter tuning, some theoretical guidelines were followed:

- The ***motion control*** law can be tuned to guarantee a high value of the disturbance rejection factor;
- The ***impedance control*** law can be set to guarantee satisfactory behaviours during the interaction with the environment;
- The equivalent bandwidth of the motion control loop should be larger than the equivalent bandwidth of the admittance control loop: ***inner loop faster than outer loop***;

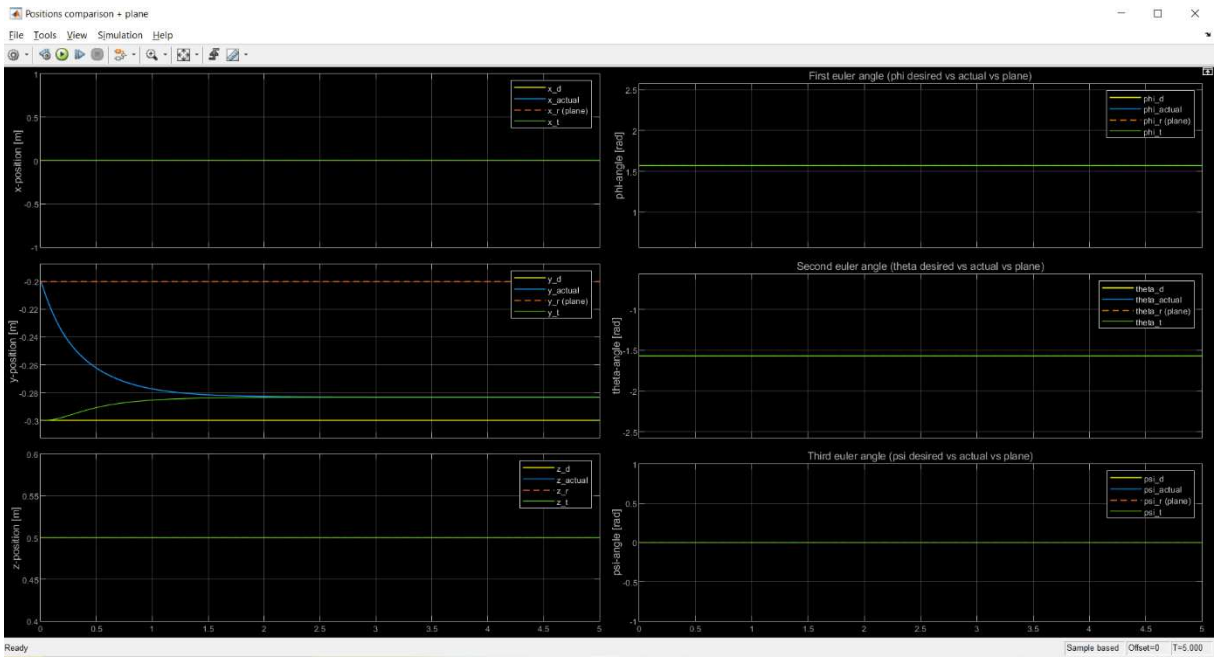
To sum up, the overall control law is mapping motion variables into equivalent end-effector forces/torques, behaving as a ***mechanical impedance***. *Admittance control* is

typically used when there's an embedded position or velocity control loop provided by the robot company that cannot be by-passed (prohibiting direct torque commands for safety reasons).

Position tracking performance (the scope shows respectively Σ_{d}, Σ_{e}, Σ_{r}, Σ_{t}*):*



Position regulation (to goal configuration 1) performance (the scope shows respectively Σ_{d}, Σ_{e}, Σ_{r}, Σ_{t}*):*



Assignment 14

▪ Implement the Force Control with Inner Position Loop

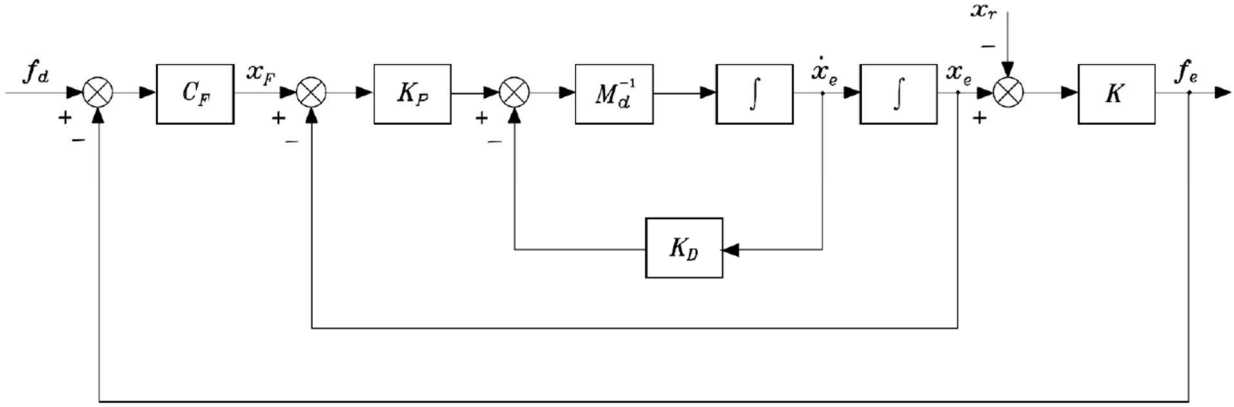
Force control is a *direct* force control meaning that the interaction force h_e can be directly controlled by specifying the desired force in a force feedback loop. To simplify the model, only forces while torques (and Euler angles) are neglected. So, the reference signal is a force f_d, while the force controller acts like a compliance matrix that maps force into position, according to the following equations (as always, the environment is modeled as a spring).

$$x_F = C_F(f_d - f_e) \qquad f_e = K(x_e - x_r)$$

To solve the force control problem, two schemes with nested loops are available:

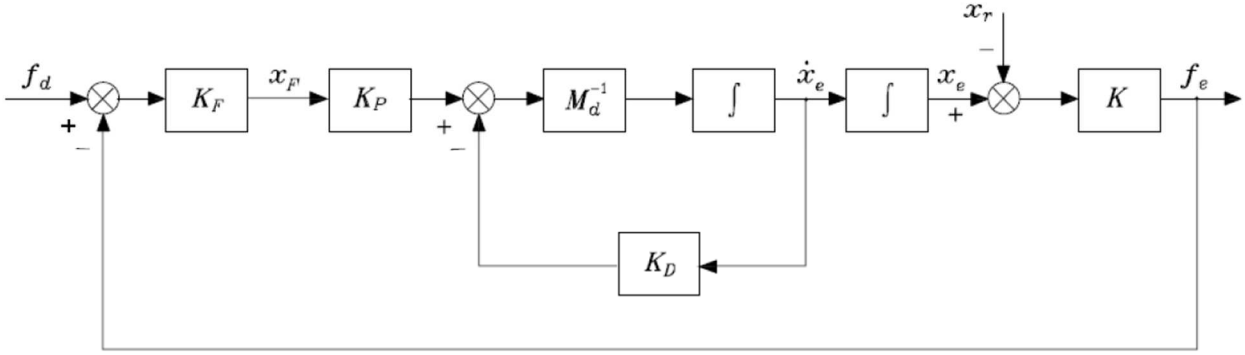
- Inner ***position*** loop and outer force loop

$$y = J^{-1}(q)M_d^{-1}\left(-K_D\dot{x}_e + K_P(x_F - x_e) - M_d\dot{J}(q,\dot{q})\dot{q}\right)$$



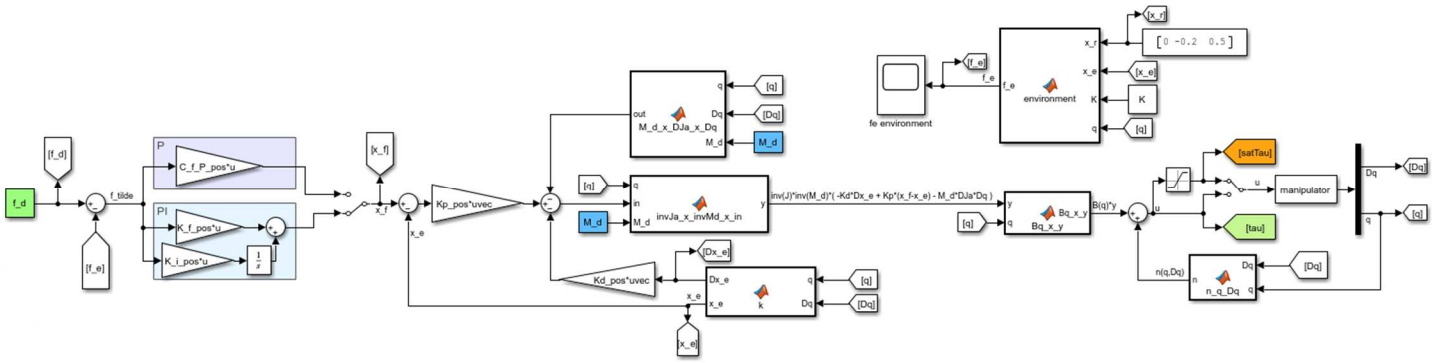
- Inner ***velocity*** loop and outer force loop

$$y = J^{-1}(q)M_d^{-1}\left(-K_D\dot{x}_e + K_P x_F - M_d\dot{J}(q,\dot{q})\dot{q}\right)$$

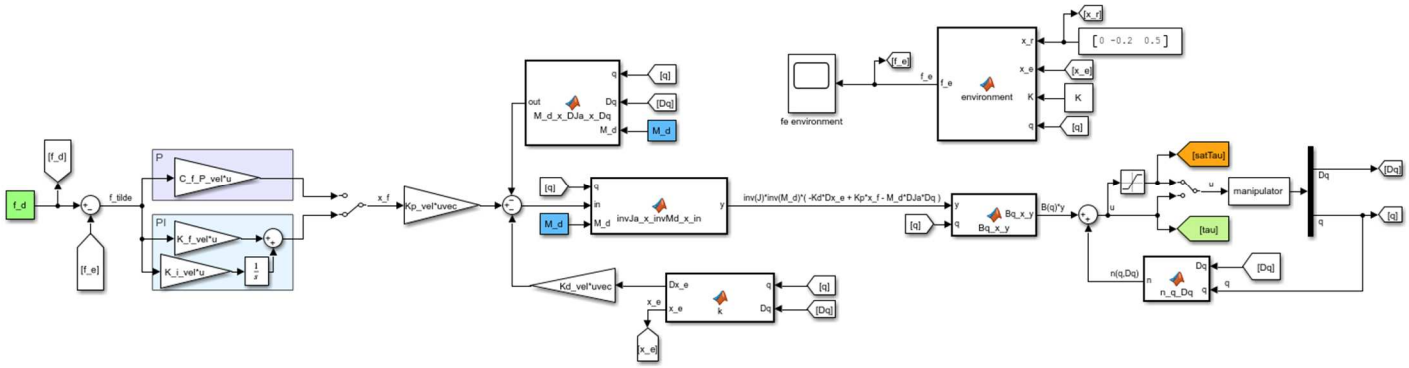


Both Force Control with Inner Position/Velocity loops have a drawback: if f_d has components outside the image of K, they cause a drift of the end-effector position.

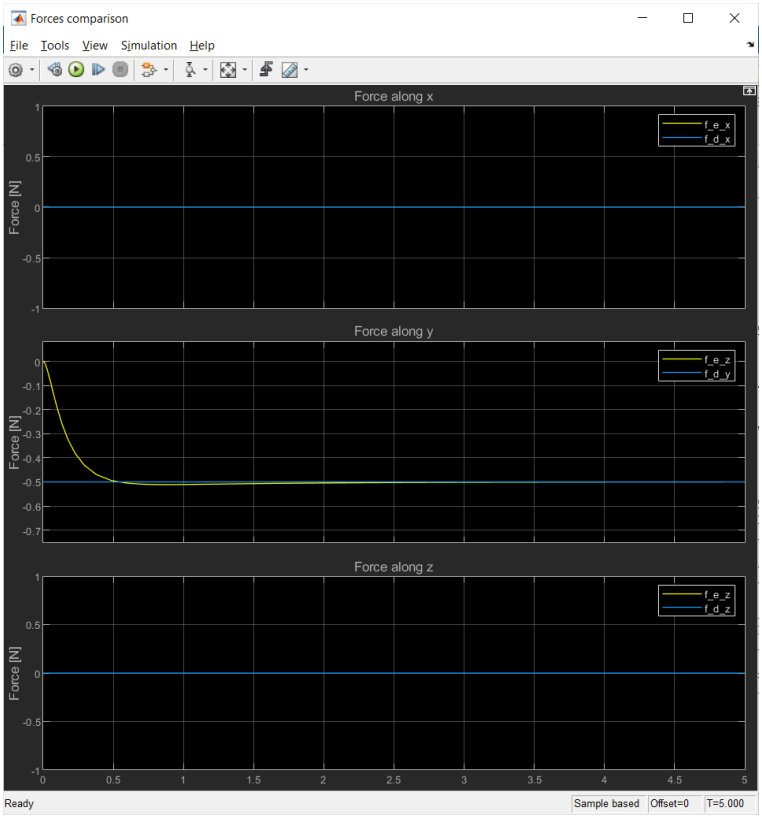
*My force control with inner **position** loop scheme:*

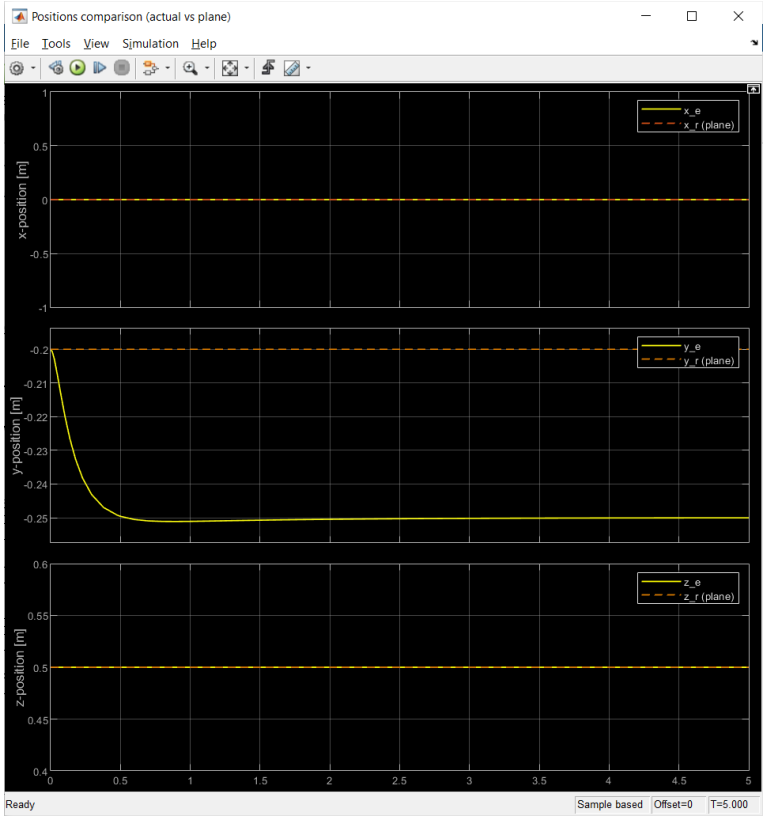


*My force control with inner **velocity** loop scheme:*

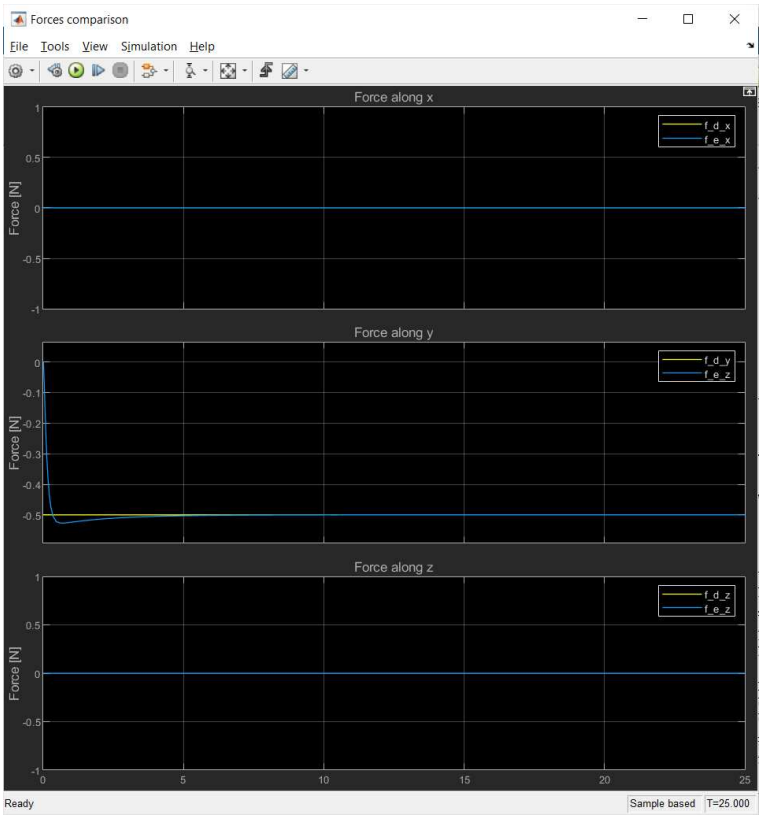


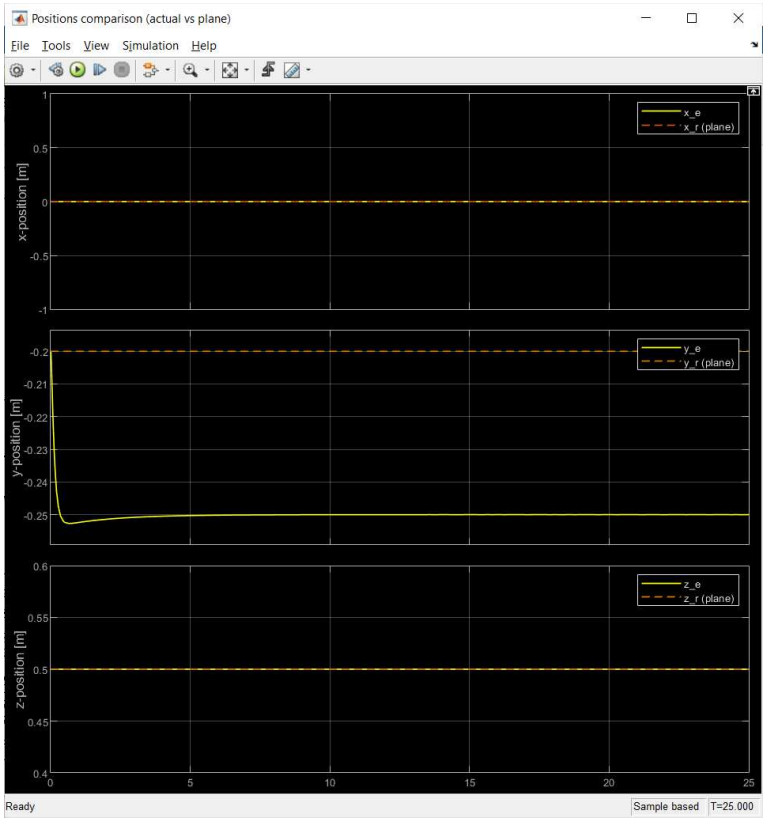
*Regulation performance inner **position** loop (constant y=-0.5N force):*





*Regulation performance inner **velocity** loop (constant y=-0.5N force):*





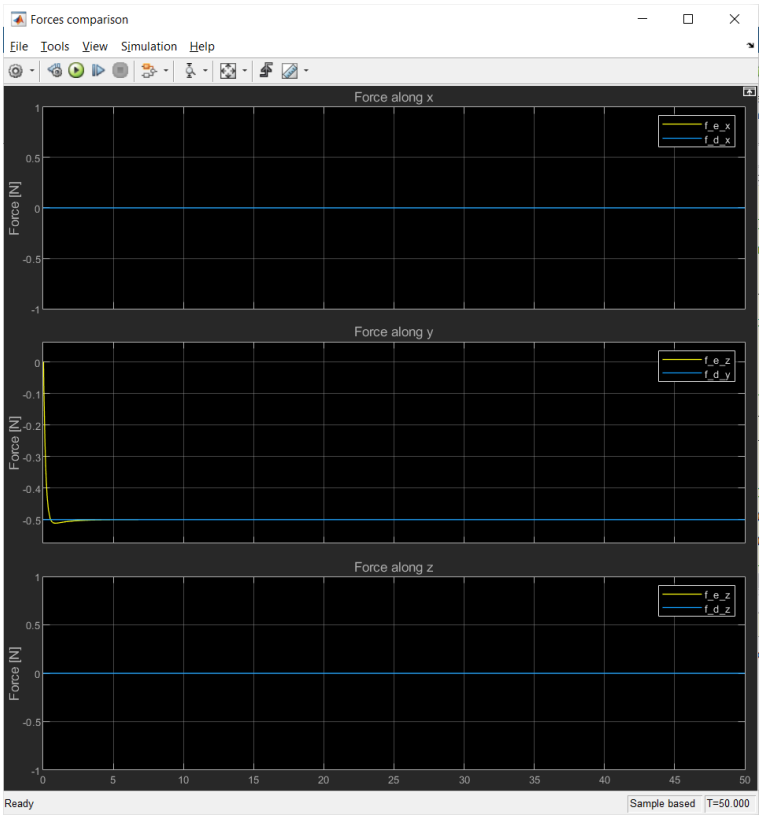
The results are similar and both the control schemes seem to function well. The only noticeable difference is that to avoid unwanted shifts of the manipulator's links (along x or z), the inner position control loop needs to have K_p and K_d matrices without components on x or z. In particular, to be clear, they were set as such:

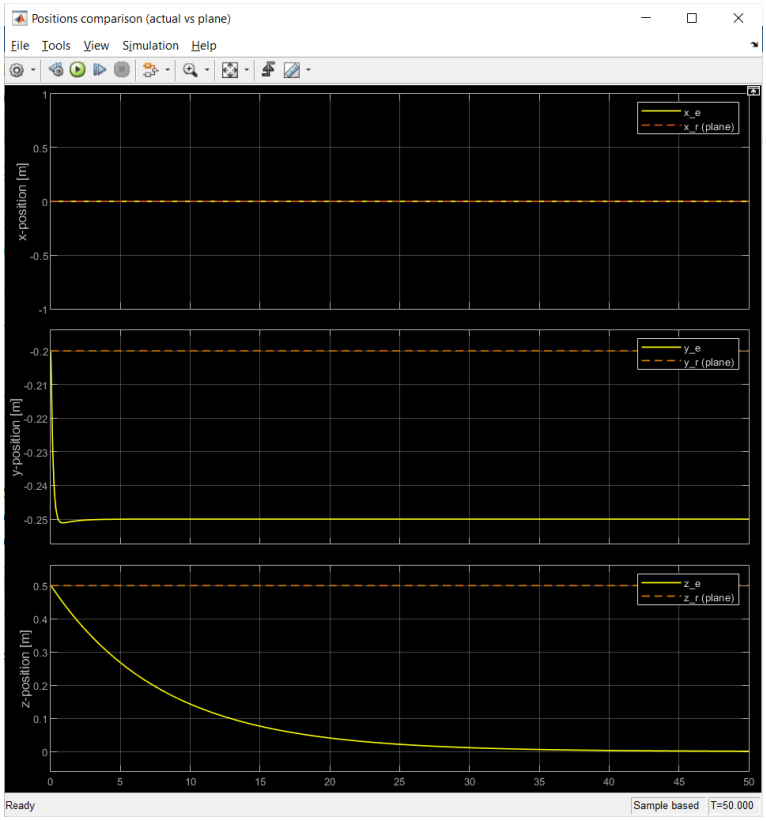
$$K_p = \begin{pmatrix} 0 & 0 & 0 \\ 0 & K_{p_y} & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad K_d = \begin{pmatrix} 0 & 0 & 0 \\ 0 & K_{d_y} & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

This is a "strange" behaviour unexpected from theory because the desired force, in this case, is achievable (inside Image(K)).

I tried to give an ***interpretation*** to this behaviour: this is a force control scheme, not a position control scheme. In terms of force, there aren't issues, while in terms of position, it seems that the manipulator tries to achieve a "wrong" reference signal. In fact, x_F will always have the first and third components equal to zero (because of how it's defined: the environment gives force reactions only along y).

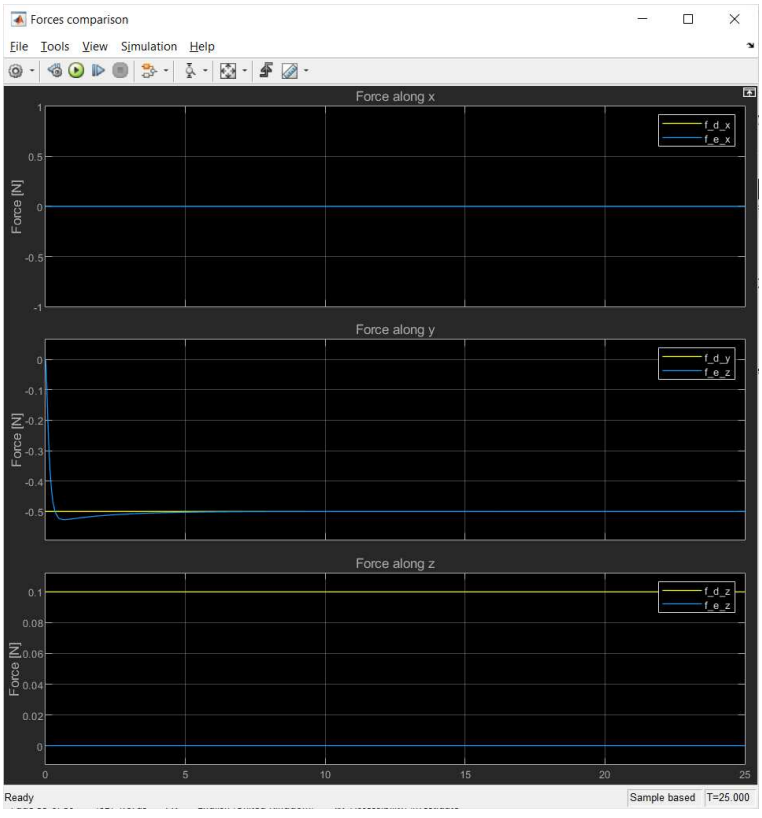
Problem is that the initial position of the robot's end-effector $x_e = [0 \ -0.2 \ 0.5]$, so the inner position controller sees an error along the z-coordinate and, of course, tries to minimize it, going towards zero (because $x_F = K(f_d - f_e) = [0 \ x_{F_y} \ 0]$, assuming that f_d has only the y component). As we can see from the next two plots, force is controlled without any problems, while position has this unexpected behaviour along z ($f_d = [0 \ -0.5 \ 0]$).

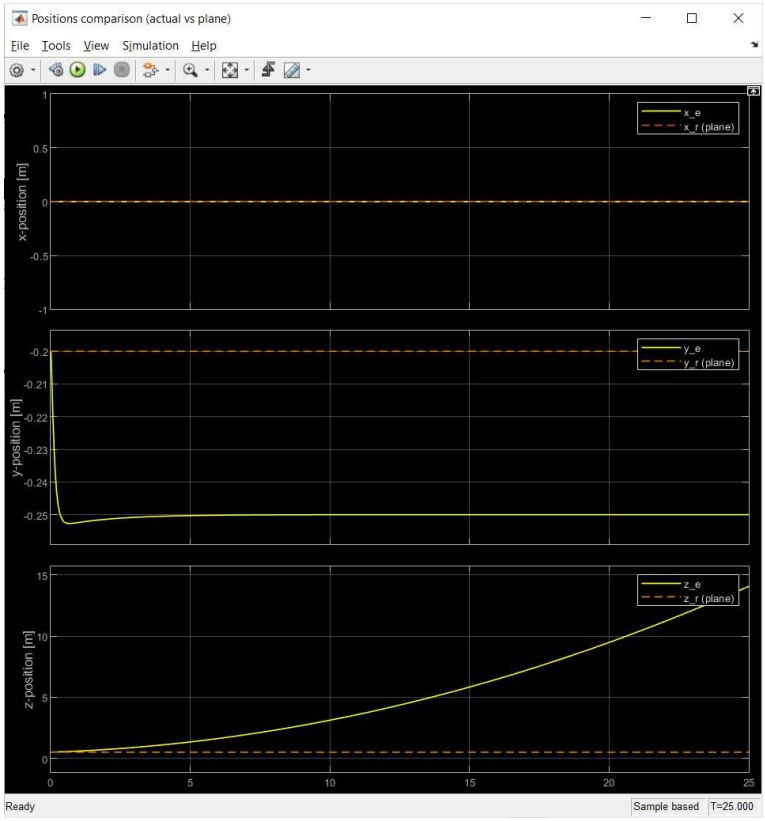




Finally, this isn't a problem for the inner velocity control scheme, because there isn't the position feedback loop that causes this "wrong position reference".

Trying, with the inner velocity control scheme, to impose a force also along z (correct end-effector shift, because there isn't a reaction force along z):





Assignment 15

- **Implement the Parallel Force/Position Control**

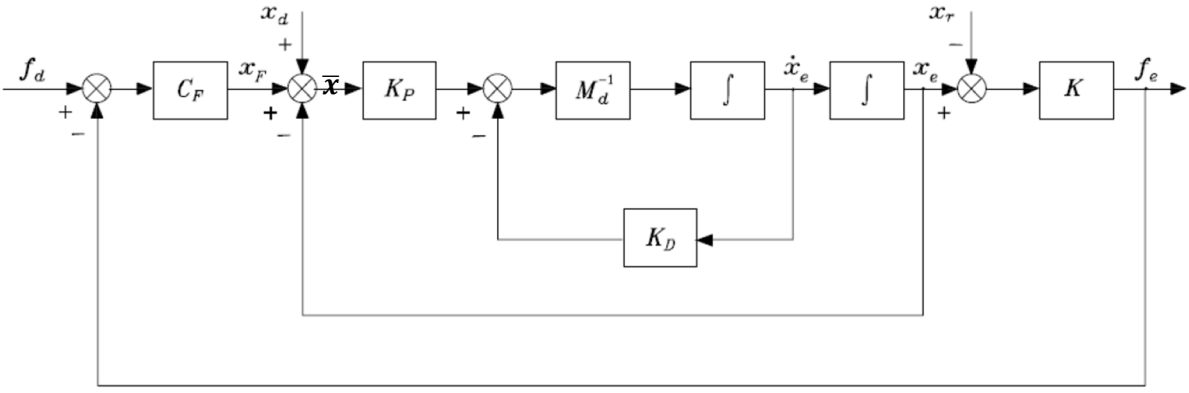
As opposed to the previous scheme, ***parallel force/position control*** is a (direct force) control architecure where both the desired force (f_d) and position (x_d) are provided. It's worth to remind that force and position/velocity can't actually be controlled at the same time, in fact, ***at the equilibrium***:

- Along directions outside Image(K) (***unconstrained motion***): x_d is reached by x_e;
- Along directions belonging to Image(K) (***constrained motion***): x_d acts like a disturbance to be rejected.

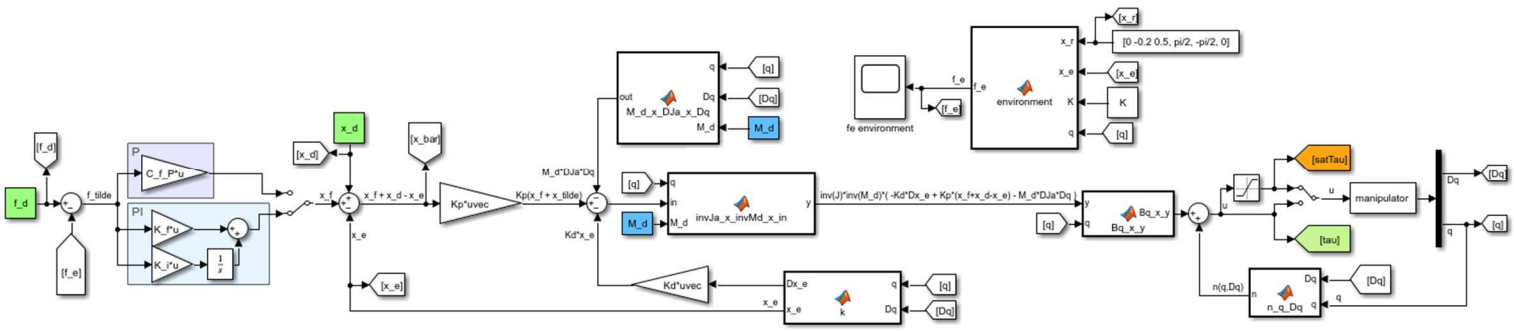
$$x_e = x_d + C_F\big(K(x_r - x_e) + f_d\big)$$

If the force controller C_F has an integral action the desired force f_d will be reached by f_e *at steady state*, while x_e will be closer or further to x_d depending on the environment compliance K.

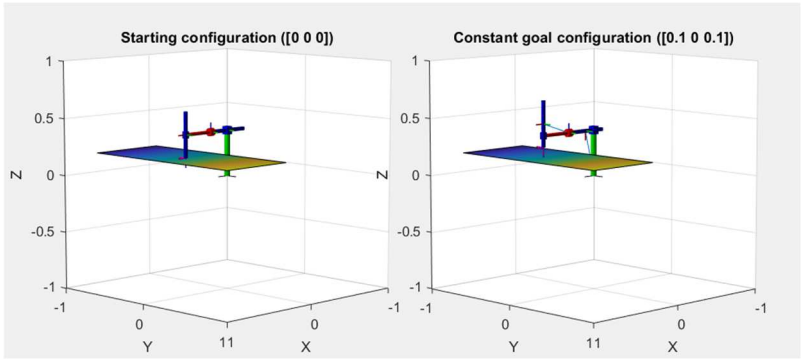
Parallel force/position control scheme:

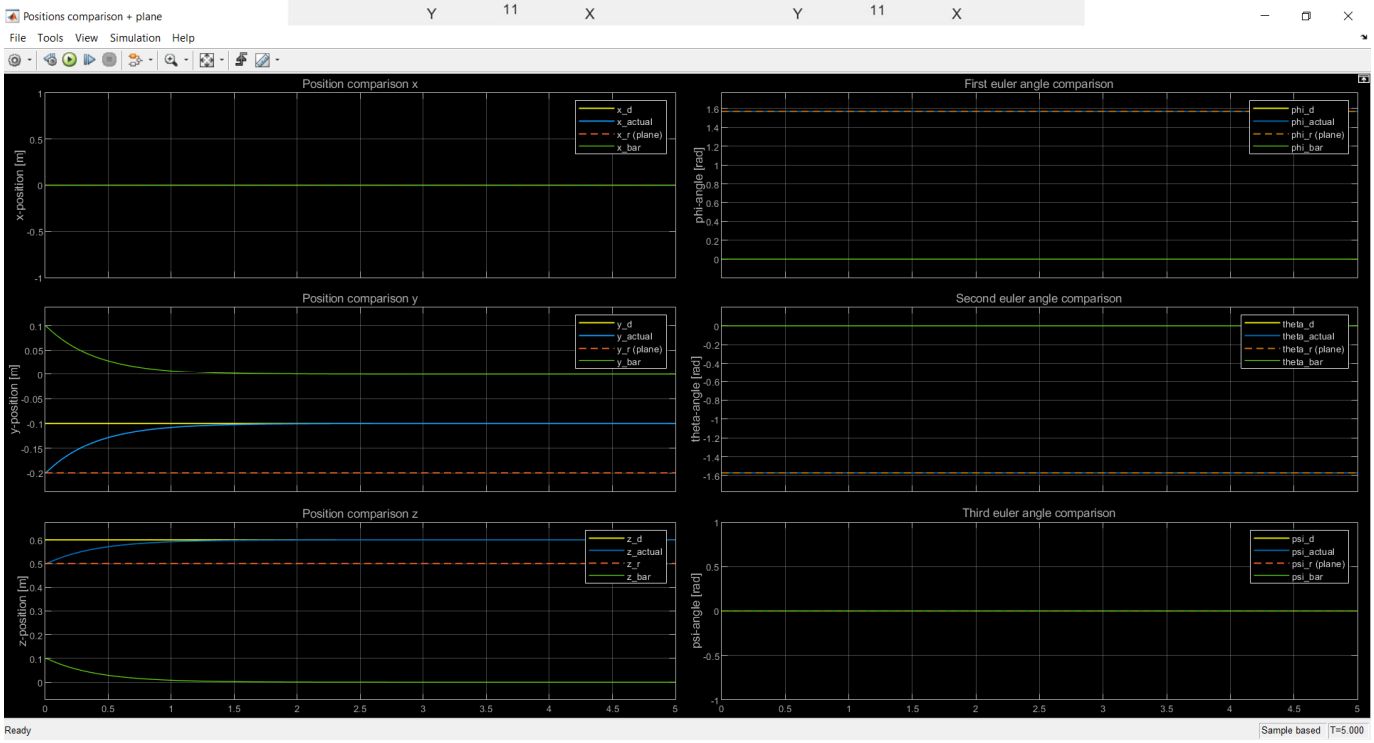


My Simulink model:



Regulation performance in free motion (no forces, free motion position control):



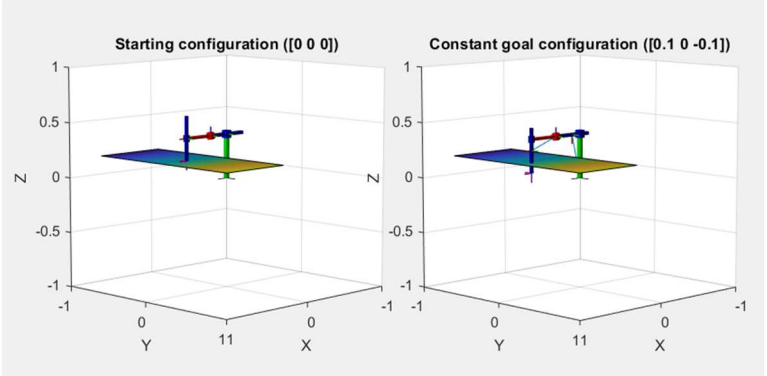


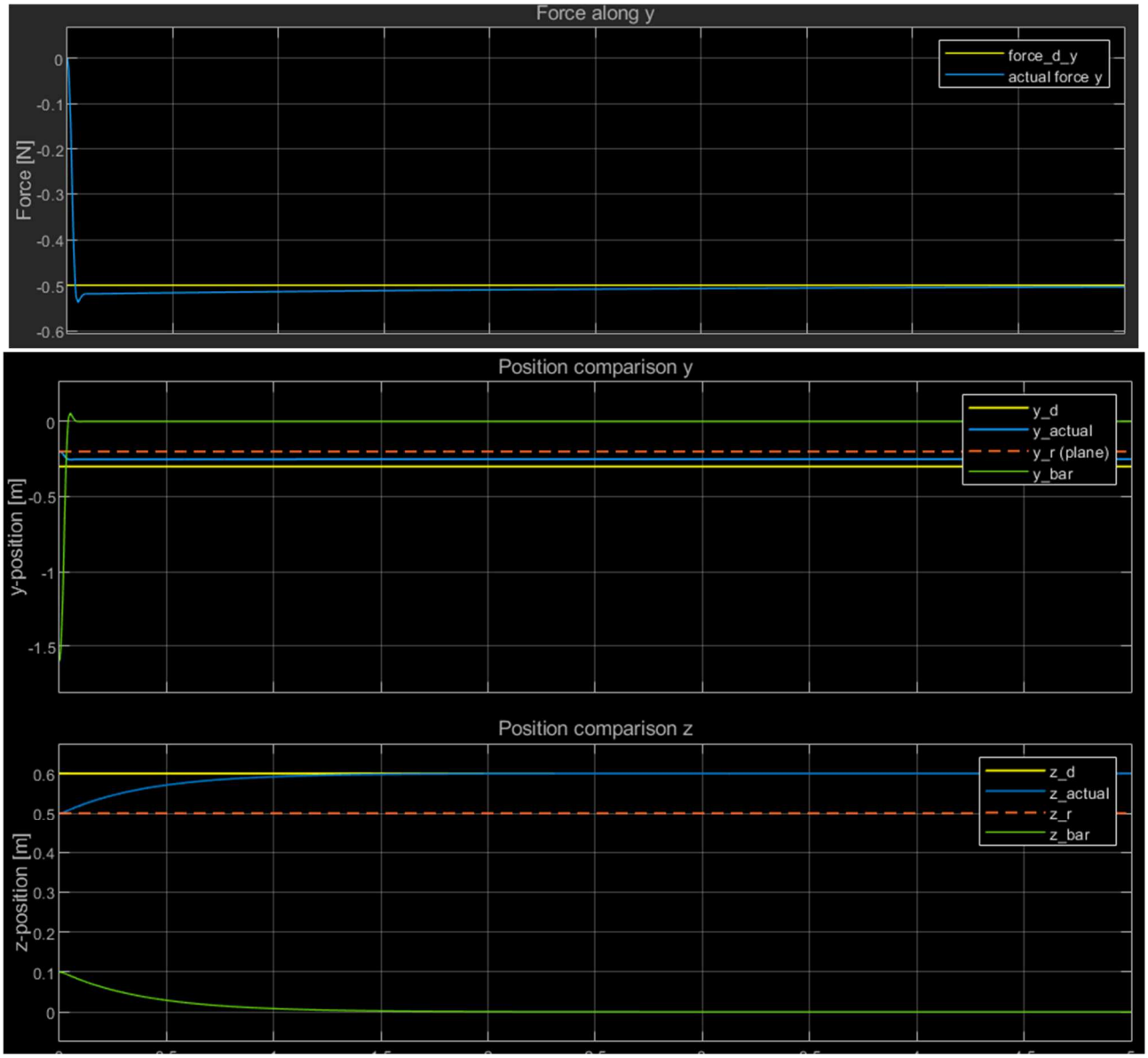
The above scope shows the desired position, the actual position, the plane's reference frame origin coordinates (the only relevant one is actually y) and the $\bar{\boldsymbol{x}}$ error defined as:

$$\bar{x} = x_F + \tilde{x} = x_F + x_d - x_e$$

Even though the plots are pretty busy, we can observe in one view that the overall error tends to zero, while $\tilde{\boldsymbol{x}}$ doesn't, because of the presence of the environment. The forces' scope hasn't been shown because no forces were desired and the robot is moving freely, without any interaction.

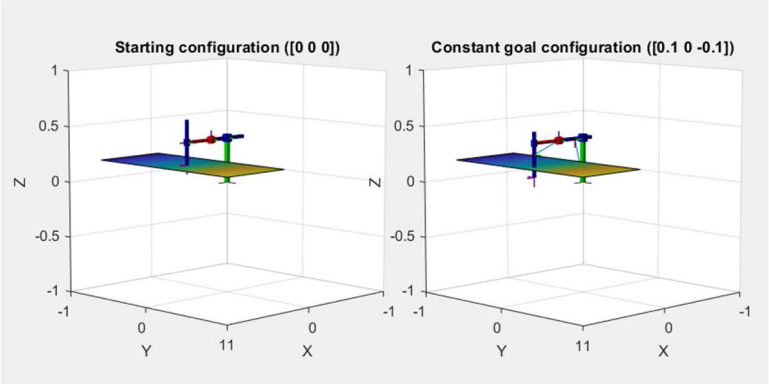
Regulation performance interacting with the environment (desired position + force):

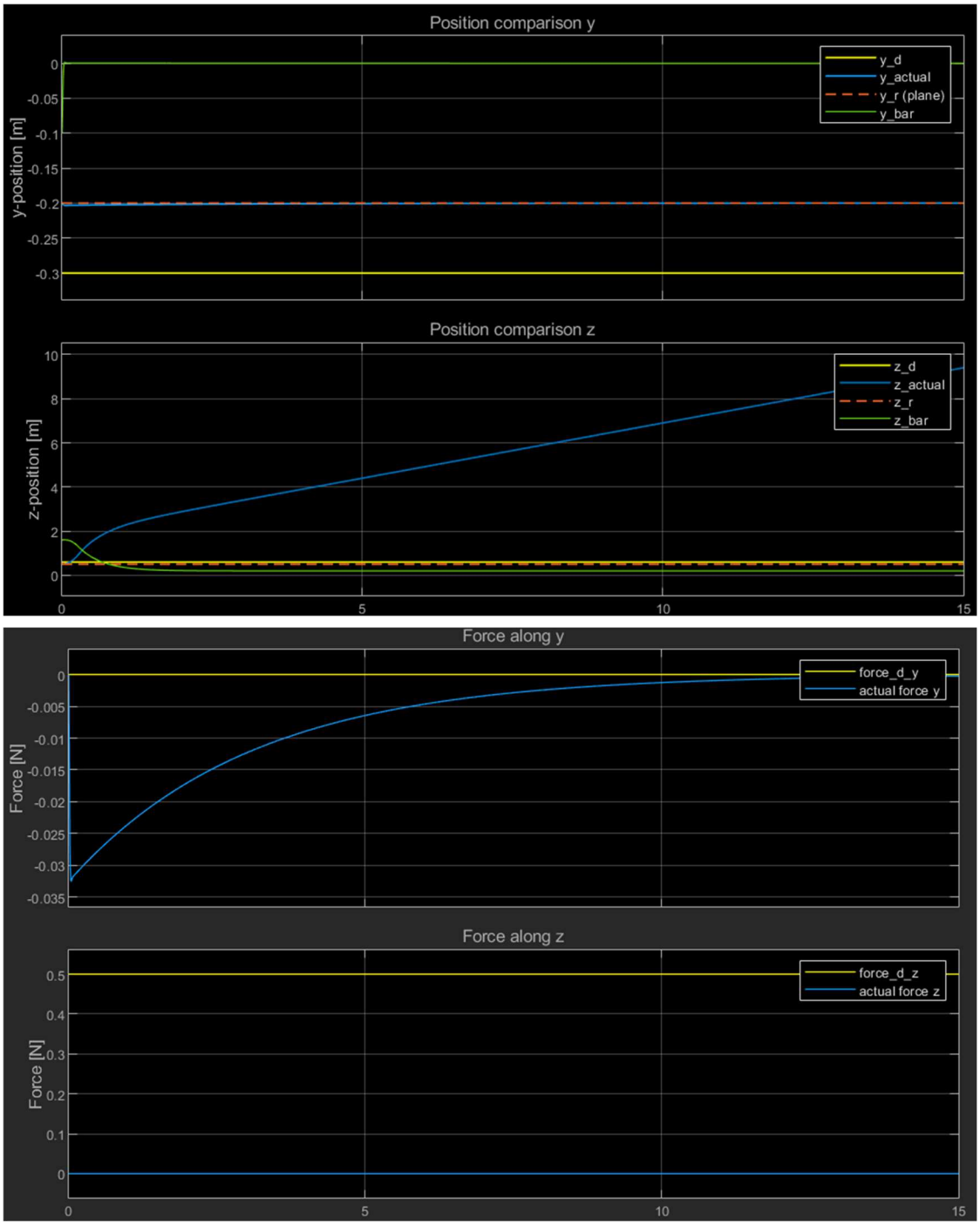




Only the non-zero plots are shown; as can be seen, along z there isn't a force (friction, for instance, isn't modeled), so the position is easily controlled. Along y, the desired position isn't reached, but the overall error (that also accounts for the environment contributes) reaches the zero. On the other hand, thanks to the *integral action*, the zero steady-state force error is reached for the vertical desired force.

Trying to impose a desired force (along z) that is outside the image of K:





The above plots clearly show that the manipulator can't reach the desired force along z. Trying to satisfy the requirements, the first link extends to the point of divergence, "breaking" the manipulator and going to non-feasible configurations.