Advanced Control Systems: RPP manipulator

Filippo Grotto VR460638

January 3, 2022

Contents

1 Kinematics **2**
- 1.1 Direct Kinematics . . . 2
- 1.2 Inverse Kinematics . . . 3

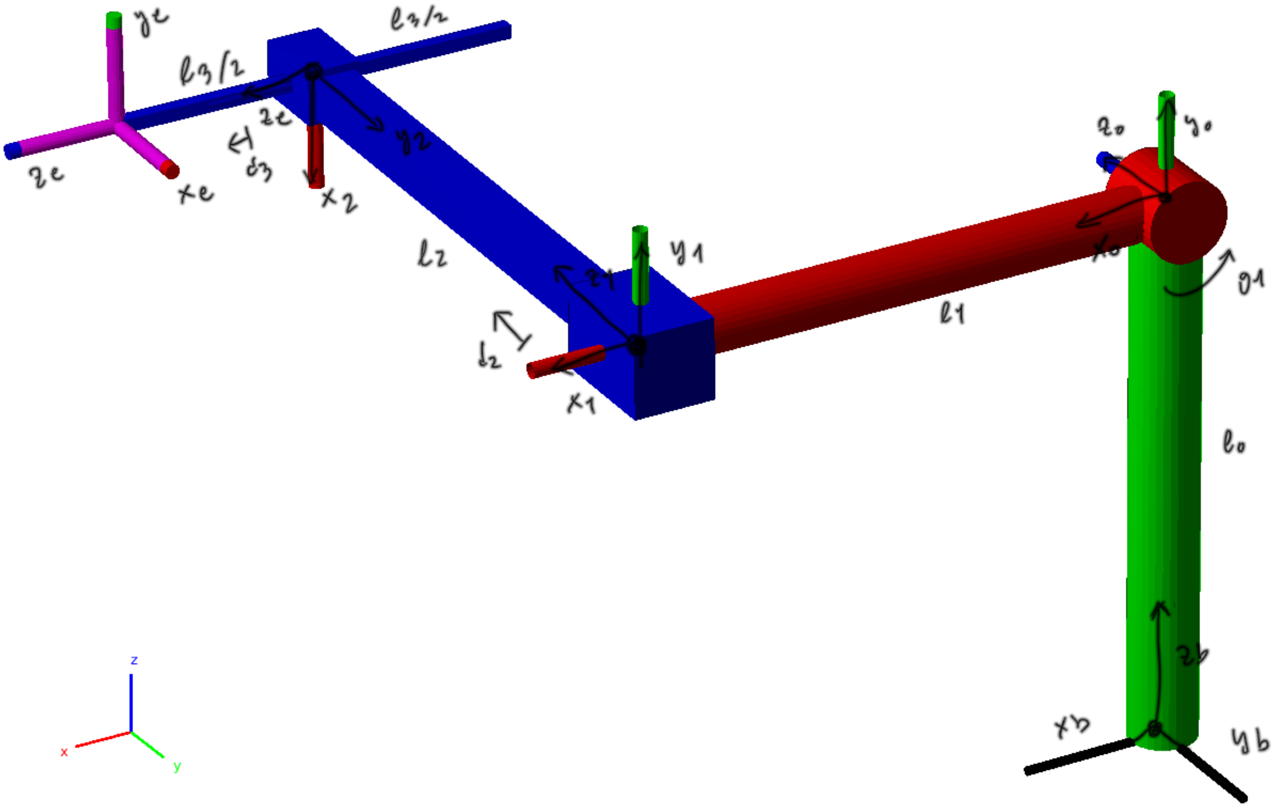
2 Differential Kinematics **3**
- 2.1 Geometric Jacobians . . . 3
- 2.2 Analytical Jacobian . . . 4

3 Lagrangian formulation **5**
- 3.1 Potential Energy . . . 5
- 3.2 Kinetic Energy . . . 5
- 3.3 Dynamic Model of the manipulator . . . 6
- 3.4 Recursive Newton Euler . . . 7

4 Control architectures **8**
- 4.1 Joint Space PD Control with Gravity Compensation . . . 8
  - 4.1.1 Without gravity compensation . . . 9
  - 4.1.2 With fixed q_d for gravity compensation . . . 10
  - 4.1.3 For the tracking problem . . . 10
  - 4.1.4 With gravity compensation and noisy reference . . . 11
- 4.2 Joint Space Inverse Dynamic PD control . . . 13
  - 4.2.1 With G, B, C different than Ĝ, B̂, Ĉ . . . 14
  - 4.2.2 Without gravity term in $n(q, \dot{q})$ . . . 14
  - 4.2.3 Extra considerations . . . 15
- 4.3 Operational Space PD control with graviy compensation . . . 16
  - 4.3.1 Without gravity . . . 16
- 4.4 Operational Space Inverse Dynamic PD control . . . 18

1 Kinematics

1.1 Direct Kinematics



Lets define the DH table for our manipulator:

Σ_i	d_i	θ_i	a_i	α_i
$b-0$	ℓ_0	0	0	$\frac{\pi}{2}$
$0-1$	0	θ_1	ℓ_1	0
$1-2$	ℓ_2+d_2	$\frac{\pi}{2}$	0	$\frac{\pi}{2}$
$2-3$	ℓ_3+d_3	$\frac{\pi}{2}$	0	0
$3-e$	0	0	0	0

The homogenous transformation is defined according to the following matrix and calculated for each row of the DH table. By multiplying $H_0^b H_1^0 H_2^1 H_3^2 H_e^3$ we obtain the final transformation

$$H_i^{i-1}(q_i) = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_0^b = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & \ell_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad H_1^0(\theta_1) = \begin{bmatrix} c_1 & -s_1 & 0 & \ell_1 c_1 \\ s_1 & c_1 & 0 & \ell_1 s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad H_2^1(d_2) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & d_2 + \ell_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_3^2(d_3) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & d_3 + l_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad H_e^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$H_e^b(\boldsymbol{q}) = \begin{bmatrix} 0 & -s_1 & c_1 & c_1(\ell_1 + \ell_3 + d_3) \\ -1 & 0 & 0 & -\ell_2 - d_2 \\ 0 & -c_1 & s_1 & s_1(\ell_1 + \ell_3 + d_3) + \ell_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.2 Inverse Kinematics

Let's consider the position of ee with respect of the base frame to calculate the value of the joints.

$$p_e^b = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} c_1(\ell_1 + \ell_3 + d_3) \\ -\ell_2 - d_2 \\ s_1(\ell_1 + \ell_3 + d_3) + \ell_0 \\ 1 \end{bmatrix}$$

It is easy to see that

$$d_2 = -\ell_2 - y$$

$$\theta_1 = Atan2(z - \ell_0, x)$$

For d_3 we can apply sum of squares and the result is:

$$d_3 = -\ell_1 \pm \sqrt{x^2 + (z - \ell_0)^2} - \ell_3$$

2 Differential Kinematics

2.1 Geometric Jacobians

The geometric jacobian is defined as follow with $\boldsymbol{q} = [\theta_1, d_2, d_3]^\top$. Note that the matlab robotic toolbox defines the angular velocities above the linear velocities:

$$\begin{bmatrix} \dot{p}_e \\ \omega_e \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \left[\begin{array}{c|c|c} J_{P_1} & J_{P_2} & J_{P_3} \\ J_{O_1} & J_{O_2} & J_{O_3} \end{array} \right] \begin{bmatrix} \dot{\theta}_1 \\ \dot{d}_2 \\ \dot{d}_3 \end{bmatrix}$$

$$J_{P_1} = z_0 \times (d_e^0 - d_0^0) = \begin{bmatrix} -s_1(\ell_1 + \ell_3 + d_3) \\ 0 \\ c_1(\ell_1 + \ell_3 + d_3) \end{bmatrix} \qquad J_{O_1} = z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$J_{P_2} = z_1 = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \qquad J_{O_2} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$J_{P_3} = z_2 = \begin{bmatrix} c_1 \\ 0 \\ s_1 \end{bmatrix} \qquad J_{O_3} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We can finally put all the pieces together and obtain the final geometric jacobian:

$$J(\boldsymbol{q}) = \begin{bmatrix} -s_1(\ell_1 + \ell_3 + d_3) & 0 & c1 \\ 0 & -1 & 0 \\ c_1(\ell_1 + \ell_3 + d_3) & 0 & s1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

2.2 Analytical Jacobian

The analytical jacobian can be easily calculated by using partial derivatives of p_e^b

$$p_e^b = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} c_1(\ell_1 + \ell_3 + d_3) \\ -\ell_2 - d_2 \\ s_1(\ell_1 + \ell_3 + d_3) + \ell_0 \\ 1 \end{bmatrix}$$

Finally we end up with the analytical jacobian

$$Ja(\boldsymbol{q}) = \begin{bmatrix} -s_1(\ell_1 + \ell_3 + d_3) & 0 & c1 \\ 0 & -1 & 0 \\ c_1(\ell_1 + \ell_3 + d_3) & 0 & s1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Another possibility is to use the relation between the geometric and analytical jacobian as follow using ZYZ:

$$\omega_e = T(\phi_e)\dot{\phi}_e \qquad T(\phi_e) = \begin{bmatrix} 0 & -s_\varphi & c_\varphi s_\theta \\ 0 & c_\varphi & s_\varphi s_\theta \\ 1 & 0 & c_\theta \end{bmatrix}$$

$$J(\boldsymbol{q}) = T_A(\phi_e) J_A(\boldsymbol{q})$$

$$T_A(\phi_e) = \begin{bmatrix} \mathbb{I}_3 & \emptyset_3 \\ \emptyset_3 & T(\phi_e) \end{bmatrix}$$

3 Lagrangian formulation

Let's calculate p_{ℓ_i} of the center of mass wrt of Σ_0. To get them let's calculate $p^i_{\ell_i}$ of the center of mass wrt of Σ_i

$$p^1_{\ell_1} = \begin{bmatrix} -\frac{\ell_1}{2} \\ 0 \\ 0 \end{bmatrix} \qquad p^2_{\ell_2} = \begin{bmatrix} 0 \\ \frac{\ell_2}{2} \\ 0 \end{bmatrix} \qquad p^3_{\ell_3} = \begin{bmatrix} 0 \\ 0 \\ -\frac{\ell_3}{2} \end{bmatrix}$$

we can express the homogenous wrt of Σ_0 using the following formula:

$$p_{\ell_i} = R^0_i p^i_{\ell_i} + d^0_i$$

3.1 Potential Energy

The potential energy is calculated according to the formula:

$$U_i = -m_{l_i} g_0^T p_{l_i} \qquad g_0 = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

The total potential energy is the sum of the 3 contributions U_1 U_2 and U_3. The total expression is reported and was calculated using the MATLAB symbolic toolbox (L_{ih} is the length of i-th link and m_i is the mass)

$$U = \frac{gsin(\theta_1)(l_1 m_1 + 2l_1 m_2 + 2l_1 m_3 + l_3 m_3 + 2d_3 m_3)}{2}$$

3.2 Kinetic Energy

The kinetic energy is calculated using the following formula:

$$\mathcal{T}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2} \dot{\boldsymbol{q}}^\top B(\boldsymbol{q}) \dot{\boldsymbol{q}}$$

$$B(\boldsymbol{q}) = \sum_{i=1}^{n} B_i(\boldsymbol{q}) = \sum_{i=1}^{n} m_{\ell_i} (J_P^{\ell_i \top} J_P^{\ell_i}) + (R_i^{0\top} J_O^{\ell_i})^\top I^i_{\ell_i} (R_i^{0\top} J_O^{\ell_i})$$

It is necessary to calculate the inertia tensors $I^i_{\ell_i}$ and the partial jacobians $J_P^{\ell_i}$ and $J_O^{\ell_i}$. We will use the steiner theorem because all frames Σ_i are translated of $p^i_{\ell_i}$ w.r.t. of the center of mass (i.e inertia tensor w.r.t. of the axis of the joint that the link is attached).

$$I^1_{\ell_1} = I^{C_1}_{\ell_1} + m_{\ell_1} S^T(r) S(r) = I^{C_1}_{\ell_1} + m_{\ell_1} (r^\top r \mathbb{I}_{3,3} - r r^\top)$$

For the inertia tensors we can use the following formulas for the cylindrical and prismatic links considering that the prismatic links have a square base.

$$I^C_{cylinder} = \frac{1}{2} \begin{bmatrix} m(a^2+b^2) & 0 & 0 \\ 0 & m(3(a^2+b^2)+h^2) & 0 \\ 0 & 0 & m(3(a^2+b^2)+h^2) \end{bmatrix}$$

$$I^C_{prismatic} = \frac{1}{12}\begin{bmatrix} m(b^2+c^2) & 0 & 0 \\ 0 & m(a^2+c^2) & 0 \\ 0 & 0 & m(a^2+b^2) \end{bmatrix}$$

Finally we need to compute the partial jacobians in order to calculate velocity of intermediate links.

$$J^{\ell_i}_{P_j} = \begin{cases} z_{j-1} & \text{prismatic joint} \\ z_{j-1} \times (p_{l_i} - p_{j-1}) & \text{revolute joint} \end{cases} \qquad J^{\ell_i}_{O_j} = \begin{cases} 0 & \text{prismatic joint} \\ z_{j-1} & \text{revolute joint} \end{cases}$$

In our case the computer partial jacobians are:

$$J^{\ell_1}_P = \begin{bmatrix} -\ell_1 sin(\theta_1)/2 & 0 & 0 \\ \ell_1 cos(\theta_1)/2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad J^{\ell_2}_P = \begin{bmatrix} \ell_2 cos(\theta_1)/2 - \ell_1 sin(\theta_1) & 0 & 0 \\ \ell_1 cos(\theta_1) + (\ell_2 sin(\theta_1))/2 & 0 & 0 \\ 0 & 1 & 0; \end{bmatrix}$$

$$J^{\ell_3}_P = \begin{bmatrix} -sin(\theta_1)(\ell_1 + \ell_3/2 + d_3) & 0 & cos(\theta_1) \\ cos(\theta_1)(\ell_1 + \ell_3/2 + d_3) & 0 & sin(\theta_1) \\ 0 & 1 & 0 \end{bmatrix} \qquad J^{\ell_1}_O = J^{\ell_2}_O = J^{\ell_3}_O = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$B(\mathbf{q}) = B_1(\mathbf{q}) + B_2(\mathbf{q}) + B_3(\mathbf{q})$$

Finally we can recover the kinetic energy using the calculated $B(\mathbf{q})$ and $\dot{\mathbf{q}}$.

3.3 Dynamic Model of the manipulator

The aim is to find an expression that describes the dynamic model of the manipulator:

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

The matrix $B(q)$ was previously calculated as a sum of the contributions of each link and $g(q)$ can be easily derived by differentiating U by the generalized positions $q = [\theta_1, d_2, d_3]$. In order to recover $C(q,\dot{q})$ some additional steps are required and described as follows:

$$\begin{aligned} \sum_{j=1}^{n} c_{ij}(q)\dot{q}_j &= \sum_{j=1}^{n}\sum_{k=1}^{n} \frac{1}{2}(\frac{\partial b_{ij}}{\partial q_k} + \frac{\partial b_{ik}}{\partial q_j} - \frac{\partial b_{jk}}{\partial q_i})\dot{q}_k\dot{q}_j \\ &= \sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}\dot{q}_k\dot{q}_j \\ &= \sum_{j=1}^{n} c_{ij}\dot{q}_j \end{aligned} \tag{1}$$

A generalized formulation for the dynamic model of the manipulator is

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

3.4 Recursive Newton Euler

The recursive newton euler has been implemented for the RPP robot. The calculations are lengthy and are not reported here. The results have been compared with the lagrangian model and using the following facts:

$$g(q) = NE(q, 0, 0, g_0)$$
$$C(q, \dot{q})\dot{q} = NE(q, \dot{q}, 0, 0)$$
$$B_i(q) = NE(q, 0, e_i, 0) \qquad e_i = \text{i-th element equal to 1}$$

4 Control architectures

4.1 Joint Space PD Control with Gravity Compensation

The joint space PD Control with gravity compensation was implemented using an S-function to define the manipulator dynamics. In fact the symbolic B,C and G matrixes were used in this context. The values for Kp and Kd has been properly selected for our robot. In Fig 1 a plot of the positions with respect of the desired positions are reported as well as the related *tau* applied to each of the three joints.

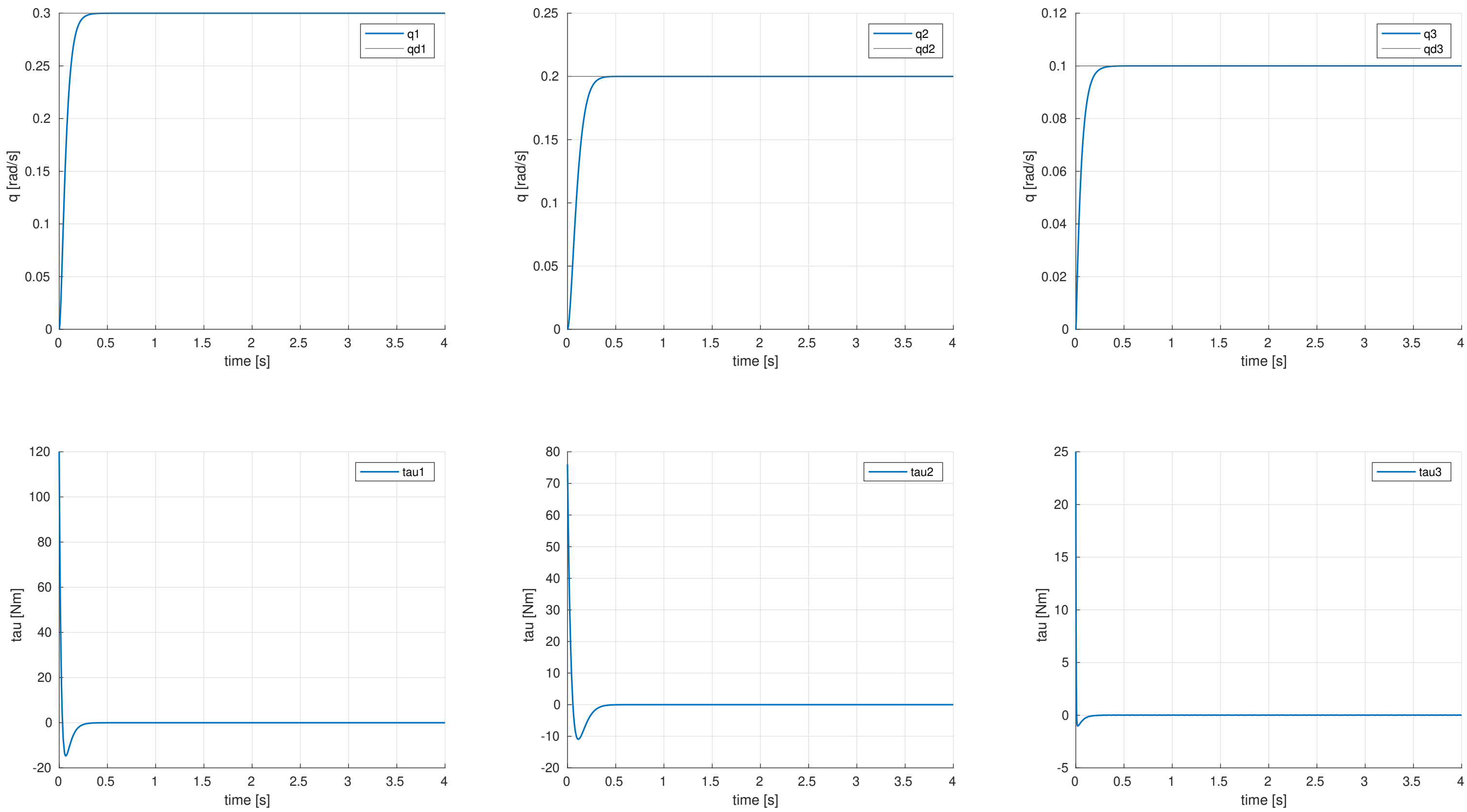


Figure 1: Joint Space PD Control with Gravity Compensation

It is reasonable to tune the 3 joints to get the same time for tracking or at least try to be at the same time. As it is visible from Fig 1 we have to find a compromise between performances and the *tau* required.

4.1.1 Without gravity compensation

Let's try without the gravity compensation, in Fig 2 it's visible that we have an offset in steady-state where the gravity plays a role, in the RPP robot it's the first joint, which is clearly visible, and on the third joint depending on the configuration (in this case the effect is really limited to to the configuration).

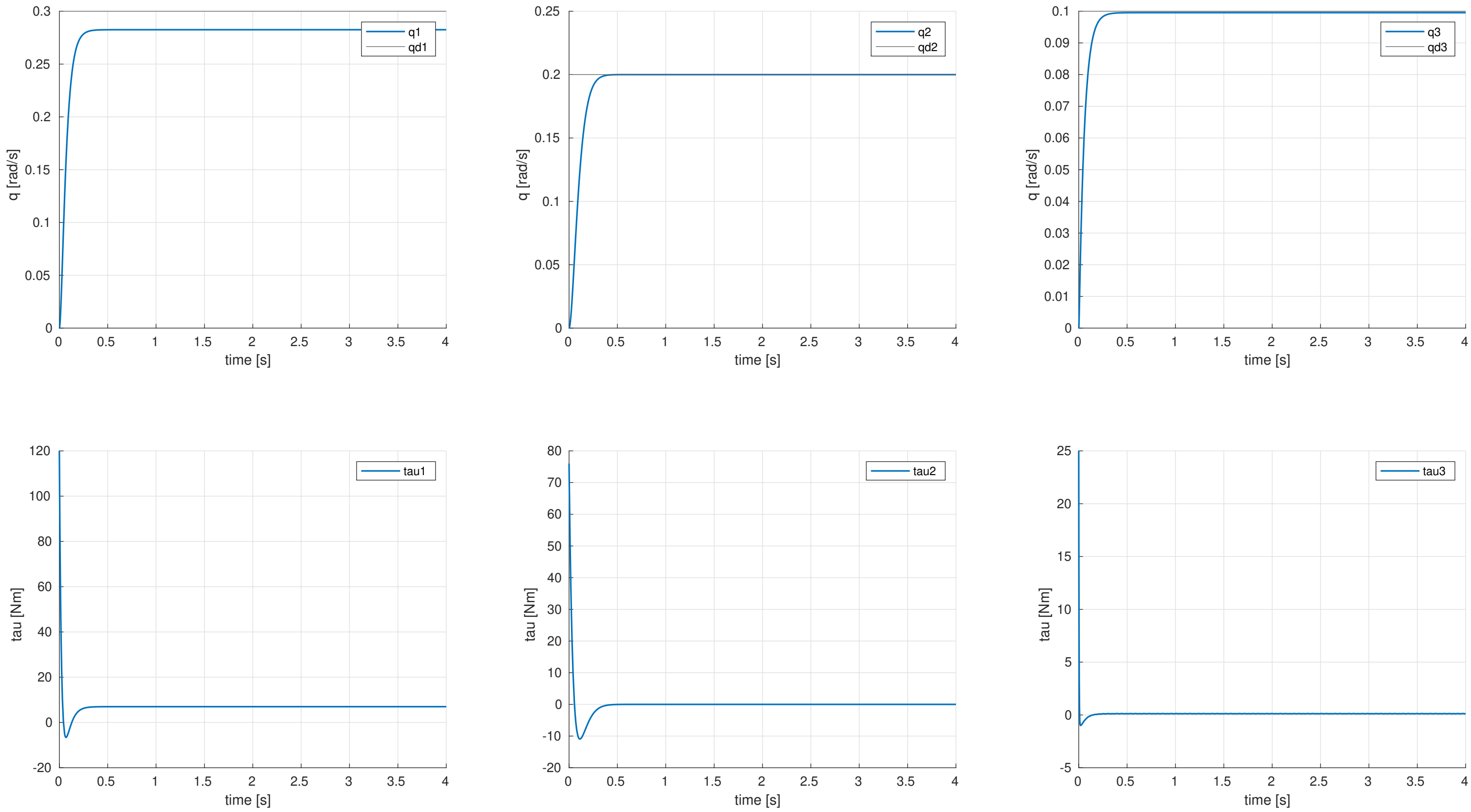


Figure 2: Joint Space PD Control without Gravity Compensation

4.1.2 With fixed q_d for gravity compensation

Finally let's compare the result obtained with gravity with fixed or time-varying q_d. In Fig 3 a small portion of the response of the first joint is reported to show the small differences using a fixed q_d for the gravity compensation vs the time-varying one.

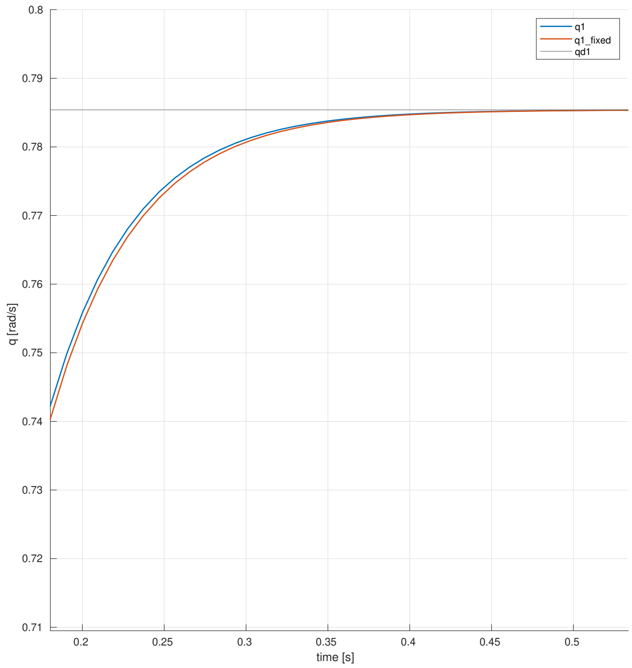


Figure 3: Joint Space PD Control with Gravity and fixed vs variable q_d

4.1.3 For the tracking problem

This control architecture can also be used, with acceptable results, for the tracking error if we don't need perfect results. The step response was achived with zero steady-state error thanks to the internal model principle (an integrator is embedded into the system thanks to the gravity compensation). This is not true, for example, for a sinusoidal function and the response is always late with respect of the reference signal. This behaviour is reported in Fig 4.

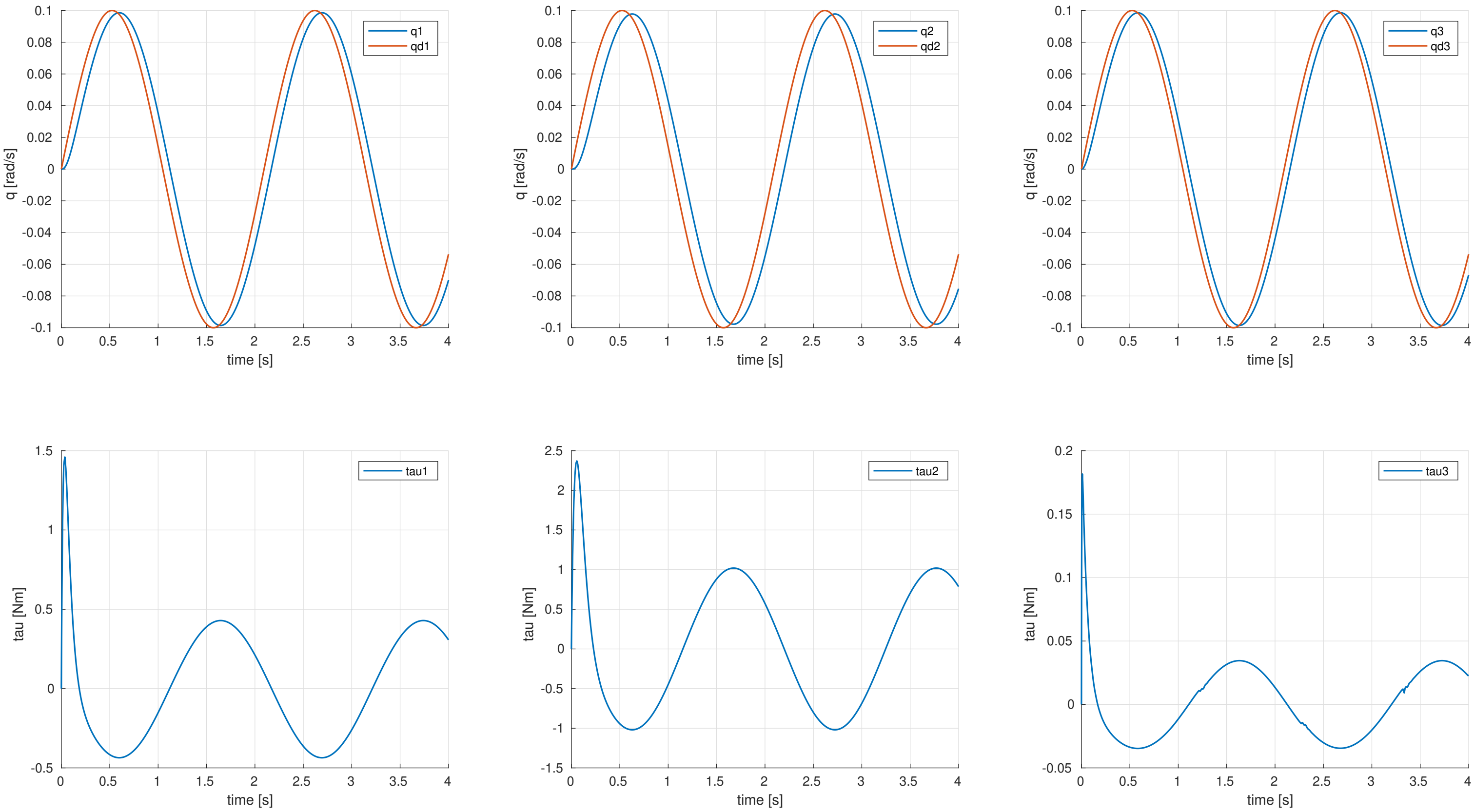


Figure 4: Joint Space PD Control with Gravity Compensation for tracking problem

4.1.4 With gravity compensation and noisy reference

Finally I can set a disturbance to the input as a sinuoidal function and see the effect on the torque of the related joints. The feedback will try to mitigate this effect and counteract this signal. This effect is reported in Fig 5.

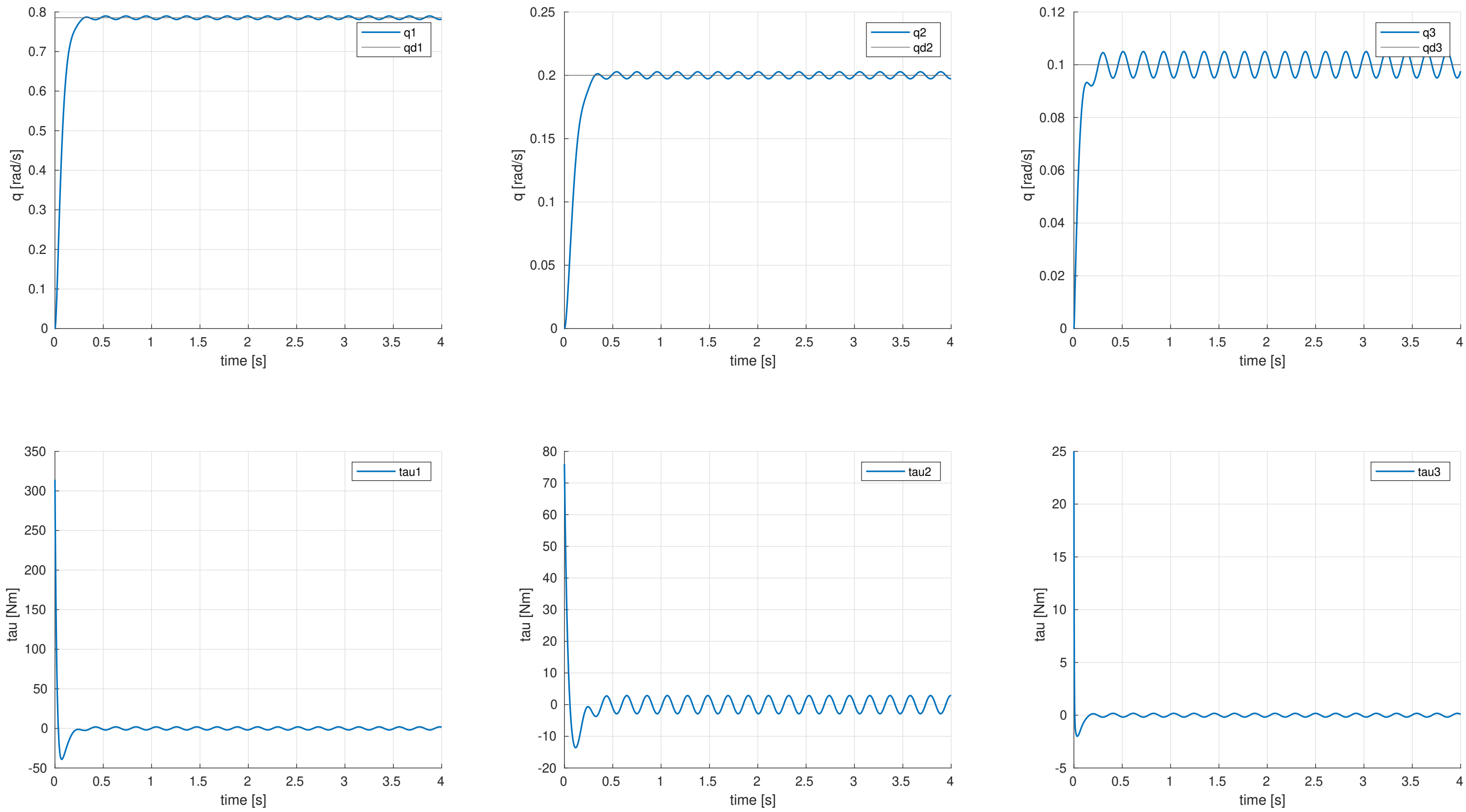


Figure 5: Joint Space PD Control with Gravity Compensation with noisy reference

4.2 Joint Space Inverse Dynamic PD control

The joint space inverse dynamics PD control architecture was introduced to solve the tracking problem using the 3DOF manipulator. In Fig 6 and example of the response obtained using a polynomial trajectory with waypoints in (0,3,5) is reported.

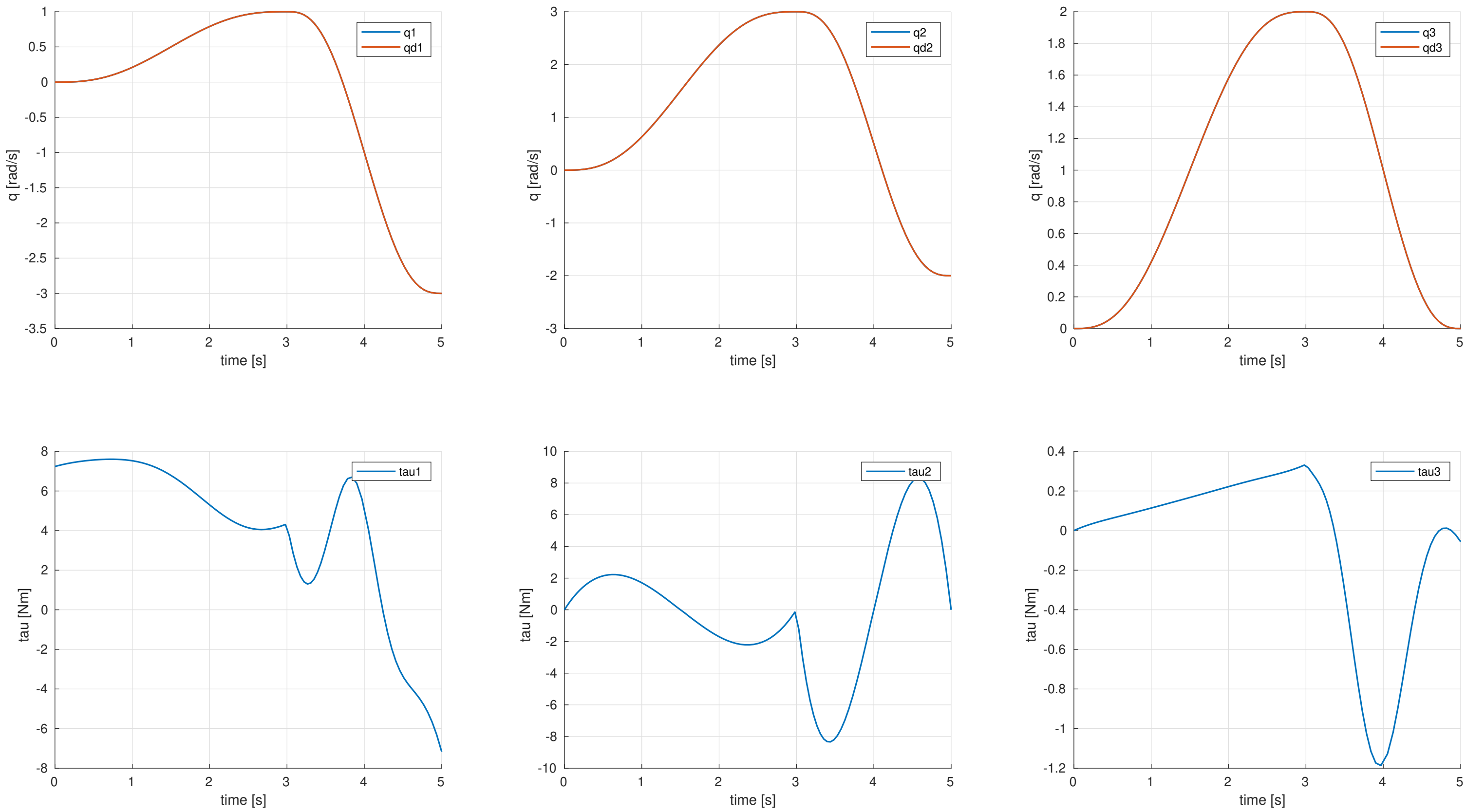


Figure 6: Inverse dynamic PD control

In order to decrease settling time we can increase K_p and K_d. Using the perfect linearization (second order system) it looks like we can obtain what we like but it's not true we need too much torque and in real life saturation limits play a big role!

We can try to introduce a saturation in *tau* with the identical saturation for each joint (in real life it's different for each joint). As a result we obtain huge overshoot and settling time is higher. (Note: we have an integrator in the plant we can't implement an antiwindup in fact we don't have it in the control side).

4.2.1 With G, B, C different than $\hat{G}$, $\hat{B}$, $\hat{C}$

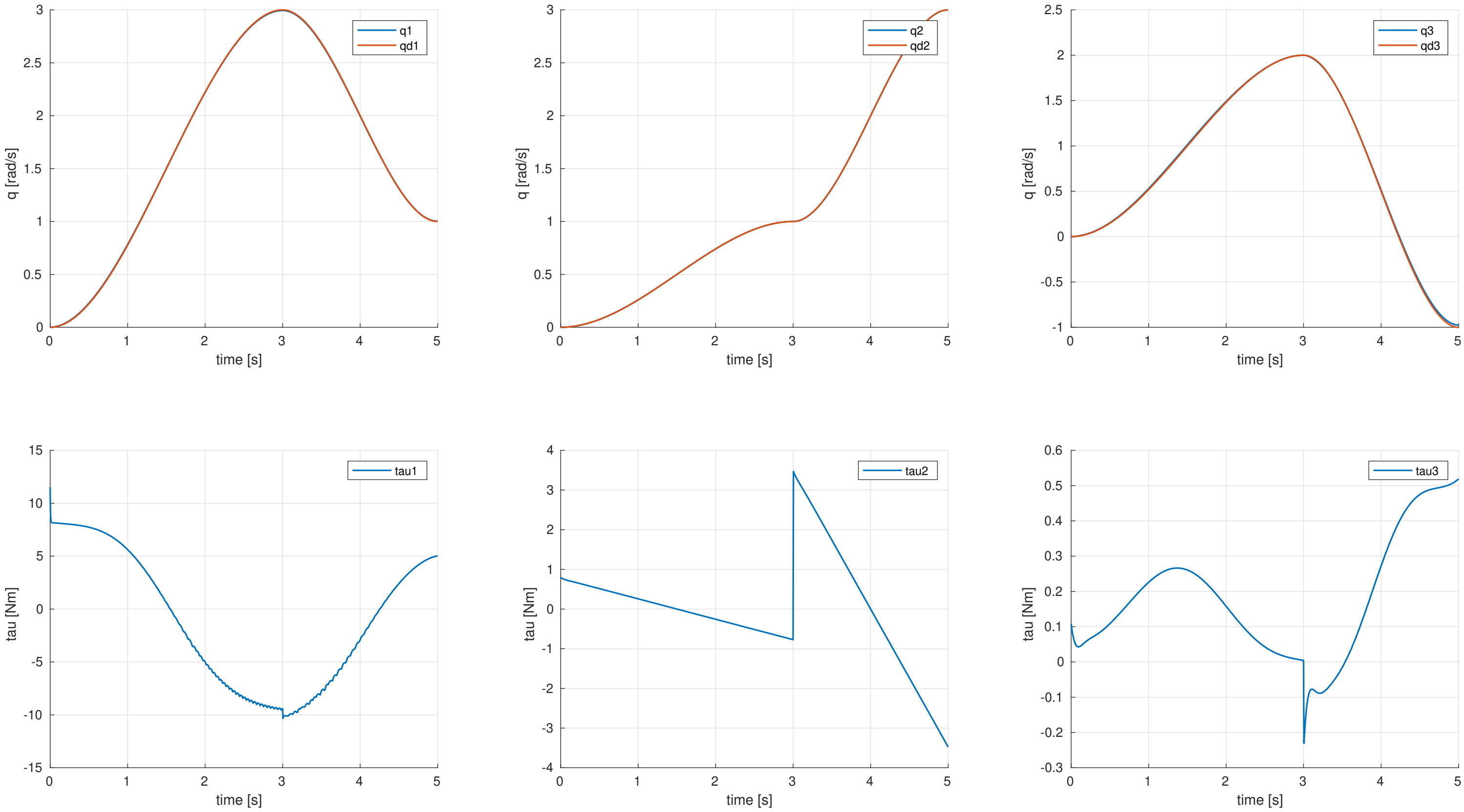


Figure 7: Inverse dynamic PD control with errors in the estimation of the dynamic parameters (m_1 and m_3 double than the real values)

From Fig 7 it is possible to see that the third joint is clearly affected by the wrong estimation.

4.2.2 Without gravity term in $n(q, \dot{q})$

In Fig 8 the gravity was removed from the $n(q, \dot{q})$ term which strongly affects the joints 1 and 3, joint 2 is not affected by gravity as it is clearly visible.

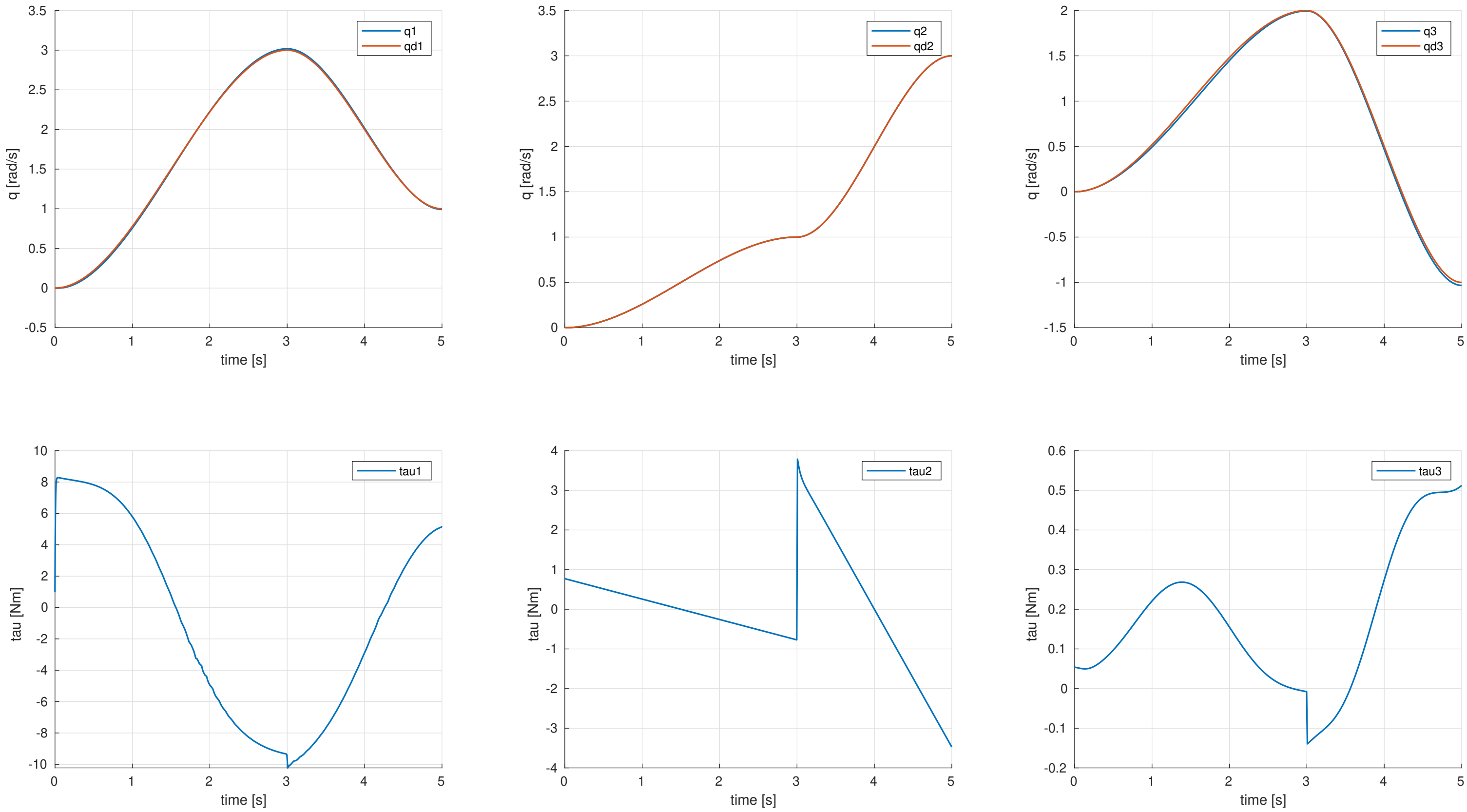


Figure 8: Inverse dynamic PD control without gravity in n term)

4.2.3 Extra considerations

- We can also try to cut $C(q,\dot{q})$ from the architecture and the result in steady state is exactly the same in fact $\dot{q} = 0$ it doesn't play any role.
- We can also try to cut $B(q)$, in this case it is responsible for the decoupling of the joints so if we apply a step to only one joint we should see that without the $B(q)$ the joints are coupled.
- n time-invariant, linear and decoupled second-order systems. we can choose

$$Kp = \left\{ w_{n1}^2, \dots \right\}$$

$$Kd = \{2\xi_1 w_{n1}, \dots\}$$

4.3 Operational Space PD control with graviy compensation

The result obtained by the PD with gravity compensation in the operational space is reported in Fig 9. The robot has 3DOF so it is not possible to force the 3 positions and 3 orientations as we want. To obtained the following results I started from a joint configuration which was translated into the operational space $q = (0.2, 0.2, 0.1)$. The results are provided wrt of frame 0.

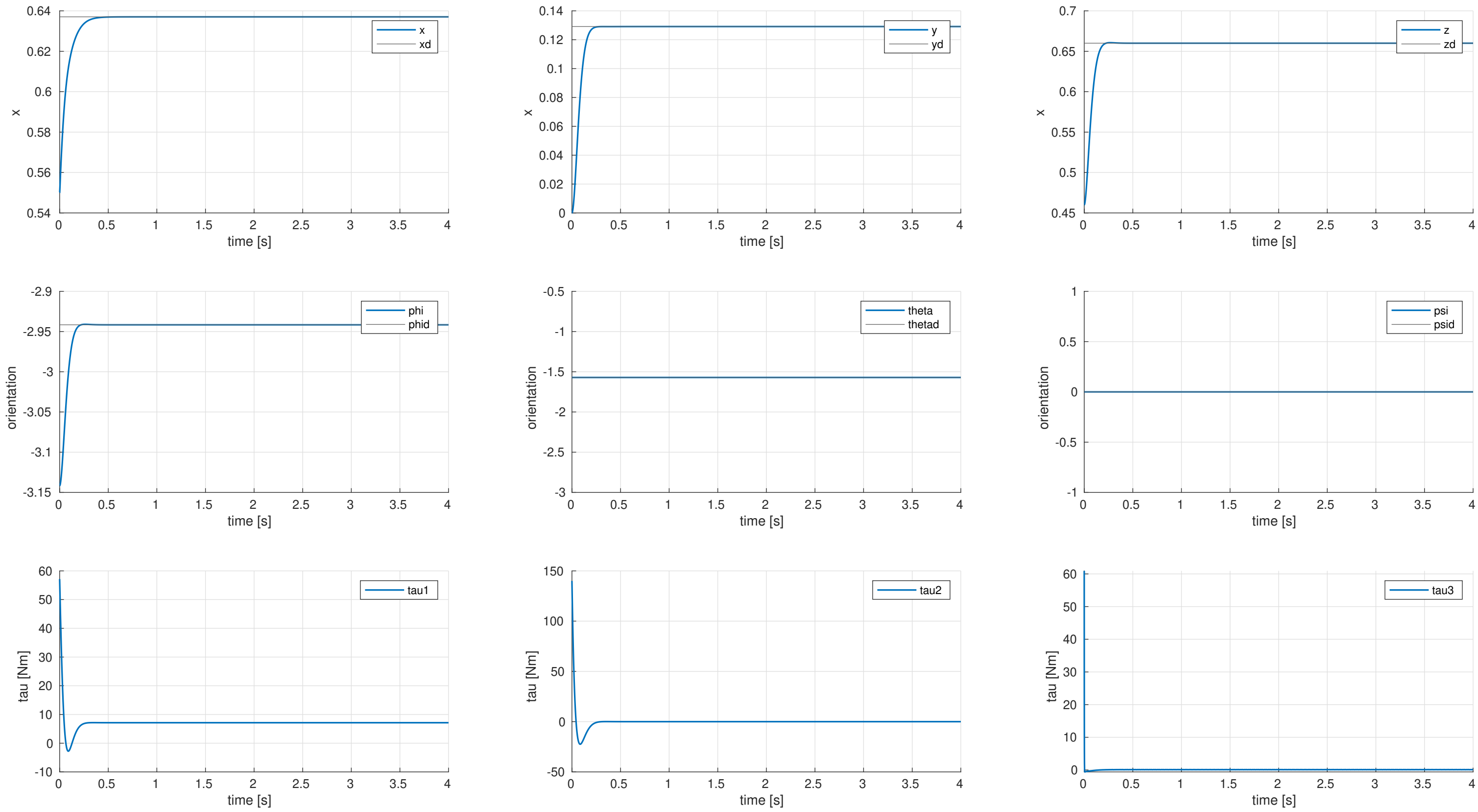


Figure 9: Operational Space PD control with graviy compensation

4.3.1 Without gravity

The desired configuration is $q = (0, 0.2, 0)$ in which the joint 1 and partially joint 3 are responsible for the gravity compensation as it is visible. As it is visible x and y don't compensate anymore (wrt of frame 0).

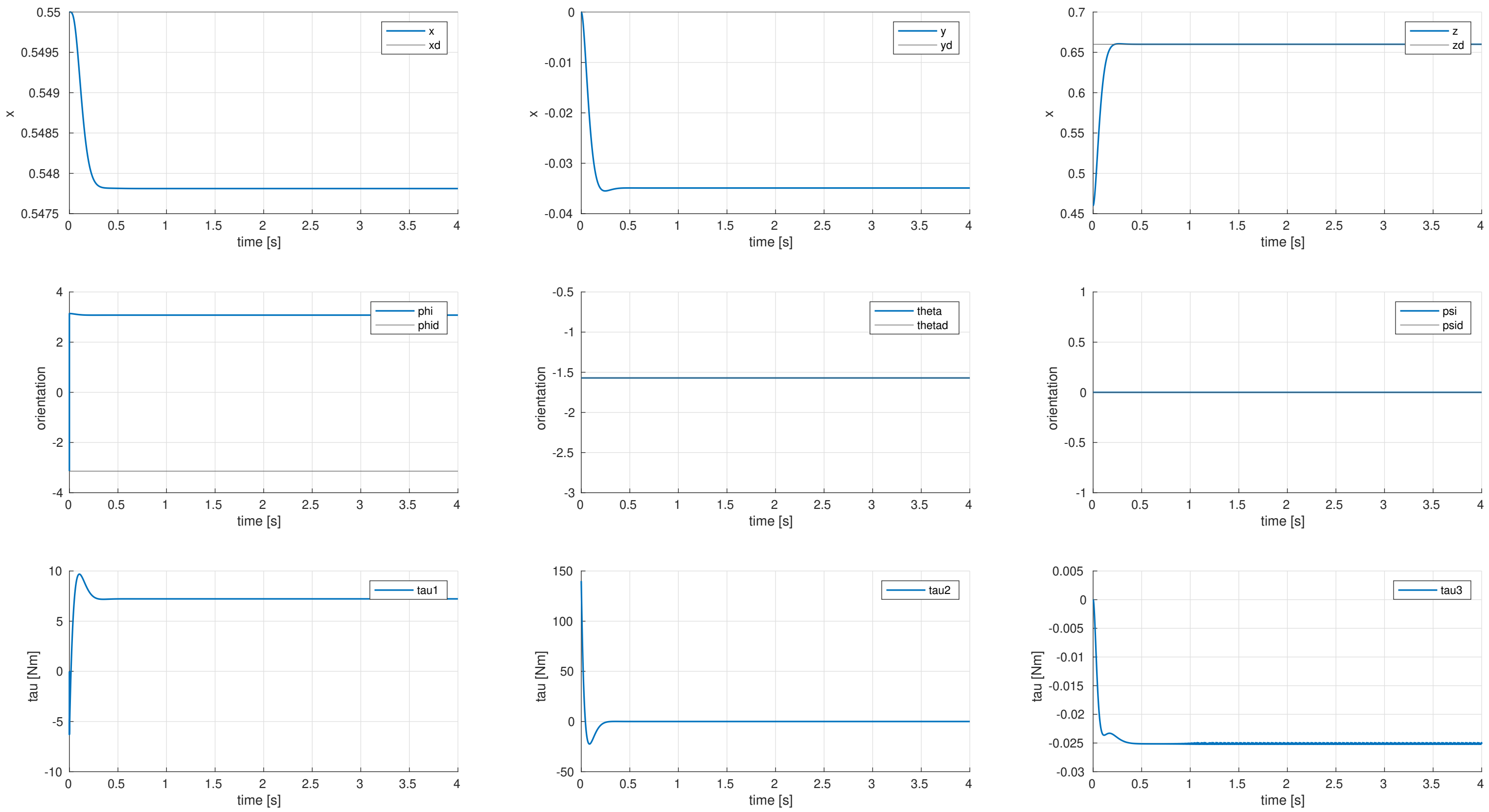


Figure 10: Operational Space PD control without graviy compensation

4.4 Operational Space Inverse Dynamic PD control

The operational space inverse dynamics PD control architecture was introduced to solve the tracking problem using the 3DOF manipulator. In Fig 11 an example of the response obtained using a polynomial trajectory. The waypoints provided are $q = [0,2,3;1,3,1;0,1,1]$ at time $t = [0,2,4]$

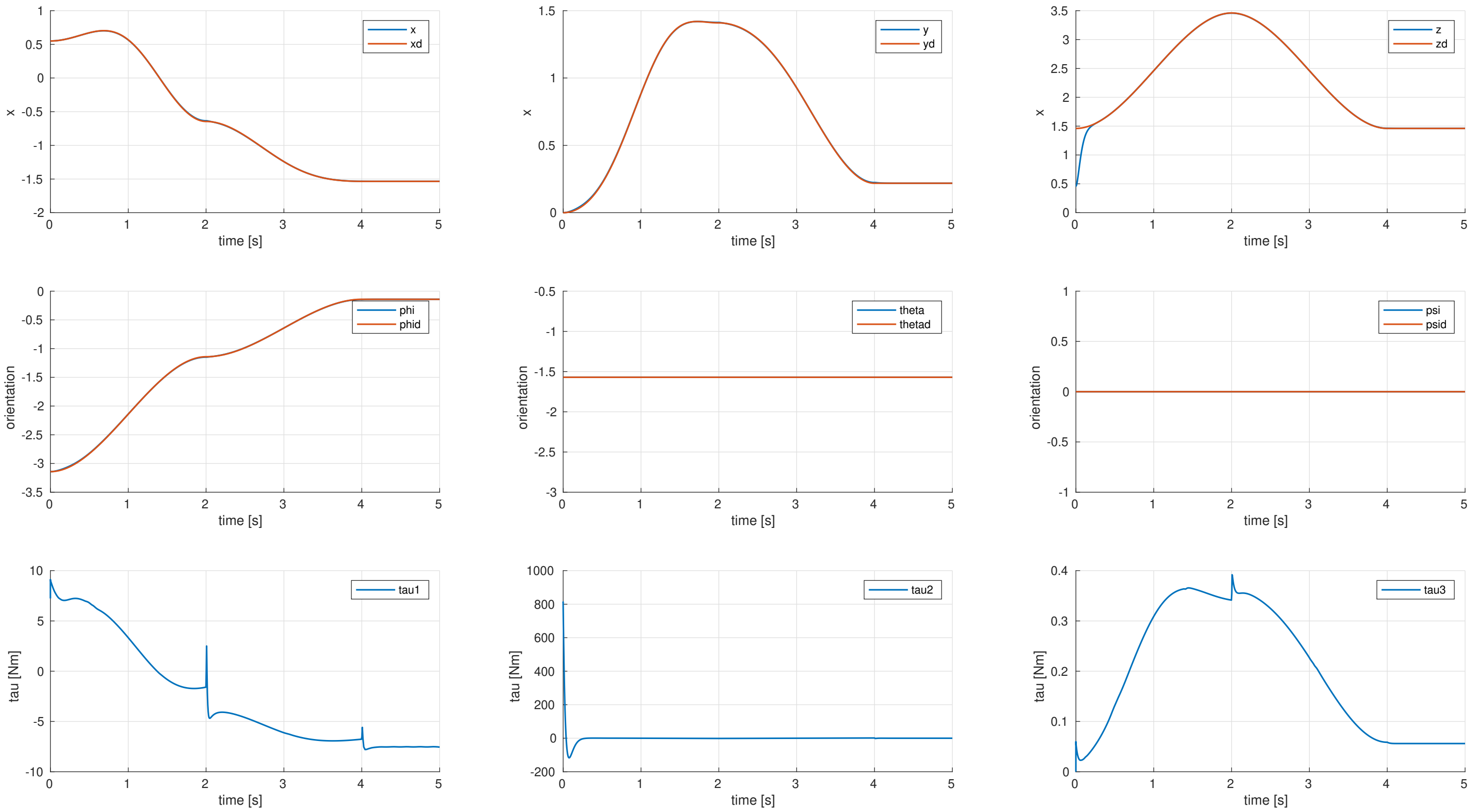


Figure 11: Operational Space Inverse dynamic PD control