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

Advanced Control System
Final Report

Luca Ponti

February 2025

Contents

1 First homework: DH table, forward kinematics, inverse kinematics and Jacobian

The robot that we are going to analyze is a 3-DOF robot manipulator with the following parameters:

- $l_1 = 4$ m
- $l_3 = 0.24$ m
- $d_2 = 0.3$ m (from -0.3 to 0)
- $l_b = 0.15$ m

The robot manipulator has three joints: θ_1, θ_2 and θ_3. A figure that shows the robot manipulator is shown in Figure 1.

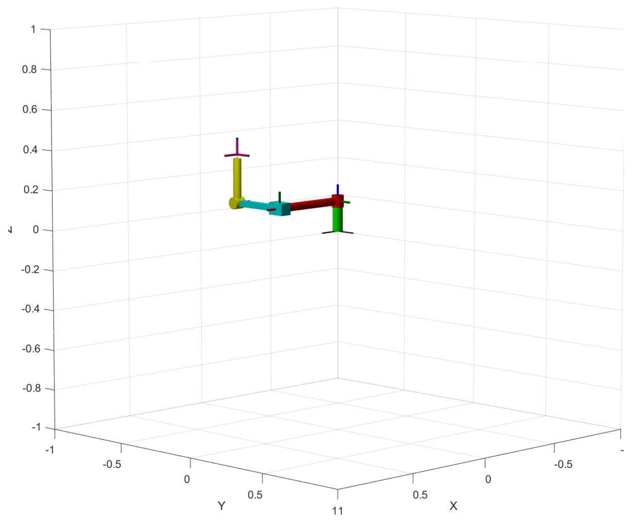

Figure 1: 3-DOF Robot Manipulator

1.1 DH table

The Denavit-Hartenberg (DH) parameters are used to describe the kinematics of a robot manipulator. The DH parameters are four values that describe the relationship between two consecutive links in a robot manipulator. The four values are: a_i, d_i, α_i and θ_i. The a_i and d_i values are the lengths of the common normal and the link offset, respectively. The α_i and θ_i values are the twist angle and the joint angle, respectively. The DH table is a matrix that contains the DH parameters for each link in the robot manipulator.

Table 1: DH Table

Frame	Link	a_i	α_i	d_i	θ_i
$b \rightarrow 0$		0	0	l_b	0
$0 \rightarrow 1$	1	l_1	$\frac{\pi}{2}$	0	θ_1
$1 \rightarrow 2$	2	0	$-\frac{\pi}{2}$	$d_2 + 0.3$	$\frac{\pi}{2}$
$2 \rightarrow 3$	3	l_3	0	0	$\theta_3 - \frac{\pi}{2}$
$3 \rightarrow ee$		0	$\frac{\pi}{2}$	0	$\frac{\pi}{2}$

1.2 Forward kinematics

The forward kinematics is the process of determining the position and orientation of the end-effector of a robot manipulator given the joint angles. The forward kinematics can be calculated using the DH parameters and the homogeneous transformation matrices. The homogeneous transformation matrix is a 4x4 matrix that describes the relationship between two consecutive links in a robot manipulator.

The homogeneous transformation matrix between two consecutive links i and $i+1$ is given by:

$$T_{i,i+1} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

The forward kinematics can be calculated by multiplying the homogeneous transformation matrices for each link in the robot manipulator. The forward kinematics can be calculated using the following equation:

$$T_{b,ee} = T_{b,0} \times T_{0,1} \times T_{1,2} \times T_{2,3} \times T_{3,ee} \tag{2}$$

Where $T_{b,ee}$ is the homogeneous transformation matrix between the base frame and the end-effector frame (accordingly to the nsa convention):

$$T_{b,ee} = \begin{pmatrix} -\sin(\theta_1)\sin(\theta_3) & -\cos(\theta_1) & \cos(\theta_3)\sin(\theta_1) & \frac{2\cos(\theta_1)}{5} + \frac{3\sin(\theta_1)}{10} + \frac{6\cos(\theta_3)\sin(\theta_1)}{25} + \theta_2\sin(\theta_1) \\ \cos(\theta_1)\sin(\theta_3) & -\sin(\theta_1) & -\cos(\theta_1)\cos(\theta_3) & \frac{2\sin(\theta_1)}{5} - \frac{3\cos(\theta_1)}{10} - \frac{6\cos(\theta_1)\cos(\theta_3)}{25} - \theta_2\cos(\theta_1) \\ \cos(\theta_3) & 0 & \sin(\theta_3) & \frac{6\sin(\theta_3)}{25} + \frac{3}{20} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

For all the others matrices see the matlab code.

1.3 Inverse kinematics

The inverse kinematics is the process of determining the joint angles of a robot manipulator given the position and orientation of the end-effector. The inverse kinematics can be calculated using the DH parameters and the homogeneous transformation matrices, and we can start from the following equation:

$$T_e^b = \begin{bmatrix} \frac{2}{5}\cos\theta_1 + \frac{3}{10}\sin\theta_1 + \frac{6}{25}\cos\theta_3\sin\theta_1 + \theta_2\sin\theta_1 \\ \frac{2}{5}\sin\theta_1 - \frac{3}{10}\cos\theta_1 - \frac{6}{25}\cos\theta_3\cos\theta_1 - \theta_2\cos\theta_1 \\ \frac{6}{25}\sin\theta_3 + \frac{3}{20} \end{bmatrix} \tag{4}$$

From the last row of the matrix, we can find the value of θ_3:

$$\theta_3 = \arcsin\left(\frac{z - 0.15}{0.24}\right) \tag{5}$$

Then, if we apply the technique of summing the squares of the first two rows, we can find the value of θ_2:

$$\theta_2 = \frac{-0.6 \pm \sqrt{-0.6^2 - 4(\frac{53}{50} - x^2 - y^2)}}{2} \tag{6}$$

Finally, we can find the value of θ_1 by substituting the values of θ_2 and θ_3 in the first row of the matrix and recalling the Weierstrass substitution formulas:

$$\sin x = \frac{2t}{1+t^2} \tag{7}$$

$$\cos x = \frac{1-t^2}{1+t^2} \tag{8}$$

$$x = \tan\frac{x}{2} \tag{9}$$

$$\theta_1 = \frac{2A \pm \sqrt{4A^2 - 4(x-2)^2}}{2(x-2)} \tag{10}$$

where A is:

$$A = 0.3 + \frac{6}{25}\cos\theta_3 + \theta_2 \tag{11}$$

1.4 Geometric Jacobian

The geometric Jacobian matrix is a 6x3 matrix that describes the relationship between the joint velocities and the end-effector velocities of a robot manipulator.

The geometric Jacobian matrix can be calculated using the following equation:

$$J_{geometric} = \begin{pmatrix} \frac{3\cos(\theta_1)}{10} - \frac{2\sin(\theta_1)}{5} + \frac{6\cos(\theta_1)\cos(\theta_3)}{25} + \theta_2\cos(\theta_1) & \sin(\theta_1) & -\frac{6\sin(\theta_1)\sin(\theta_3)}{25} \\ \frac{2\cos(\theta_1)}{5} + \frac{3\sin(\theta_1)}{10} + \frac{6\cos(\theta_3)\sin(\theta_1)}{25} + \theta_2\sin(\theta_1) & -\cos(\theta_1) & \frac{6\cos(\theta_1)\sin(\theta_3)}{25} \\ 0 & 0 & \frac{6\cos(\theta_3)}{25} \\ 0 & 0 & -\cos(\theta_1) \\ 0 & 0 & -\sin(\theta_1) \\ 1 & 0 & 0 \end{pmatrix} \tag{12}$$

1.5 Analytical Jacobian

The analytical Jacobian matrix is a 6x3 matrix and it's calculated as the derivative of the forward kinematics with respect to the joint angles. The analytical Jacobian matrix can be calculated using the following equation:

$$J_{analytical} = \begin{pmatrix} \frac{3\cos(\theta_1)}{10} - \frac{2\sin(\theta_1)}{5} + \frac{6\cos(\theta_1)\cos(\theta_3)}{25} + \theta_2\cos(\theta_1) & \sin(\theta_1) & -\frac{6\sin(\theta_1)\sin(\theta_3)}{25} \\ \frac{2\cos(\theta_1)}{5} + \frac{3\sin(\theta_1)}{10} + \frac{6\cos(\theta_3)\sin(\theta_1)}{25} + \theta_2\sin(\theta_1) & -\cos(\theta_1) & \frac{6\cos(\theta_1)\sin(\theta_3)}{25} \\ 0 & 0 & \frac{6\cos(\theta_3)}{25} \\ 1 & 0 & 0 \\ 0 & 0 & -\frac{\cos(\theta_3)}{|\cos(\theta_3)|} \\ 0 & 0 & 0 \end{pmatrix} \tag{13}$$

2 Second homework: Kinetic and potential energy

2.1 Kinetic energy

The kinetic energy of a robot manipulator is the energy due to the motion of the robot. Before calculating the kinetic energy, we need to calculate the partial jacobian for every joint, as shown in the following equation:

$$\sum_{i=1}^{n} \left(m_{\ell_i} \left(J_P^{\ell_i} \right)^T J_P^{\ell_i} + \left(J_O^{\ell_i} \right)^T R_i I_{\ell_i}^i R_i^T J_O^{\ell_i} \right) \tag{14}$$

In particular, the partial jacobian for the end effector is given by:
positional part:

$$j_{P_j}^{l_i} = \begin{cases} Z_{j-1}, & \text{prismatic joint} \\ Z_{j-1} \times (p_{l_i} - p_{j-1}), & \text{revolute joint} \end{cases} \tag{15}$$

rotational part:

$$j_{Oj}^{l_i} = \begin{cases} 0, & \text{prismatic joint} \\ Z_{j-1}, & \text{revolute joint} \end{cases} \tag{16}$$

where

- p_{j-1} is the position vector of the origin of Frame Σ_{j-1} w.r.t. Σ_0
- z_{j-1} is the unit vector of axis z of Frame Σ_{j-1} w.r.t. Σ_0

Now, we need and additional matrix, the inertia matrix for each link, wrt the frame attached to the current link.
To compute this matrix we also use **Steiner's theorem**, which states that the inertia matrix of a body with respect to a point P is equal to the inertia matrix of the body with respect to its center of mass G plus the mass of the body times the square of the distance between G and P times the identity matrix.

for prismatics joints the inertia matrix wrt to the CoM is the following:

$$I_C = \begin{bmatrix} \frac{1}{12}m(b^2+c^2) & 0 & 0 \\ 0 & \frac{1}{12}m(a^2+c^2) & 0 \\ 0 & 0 & \frac{1}{12}m(a^2+b^2) \end{bmatrix} \tag{17}$$

Remapping the inertia matrix to the frame of the link, we obtain:

$$I = I_C + m\left(\begin{bmatrix} -\frac{a}{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} -\frac{a}{2} \\ 0 \\ 0 \end{bmatrix} I_{3\times3} - \begin{bmatrix} -\frac{a}{2} \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} -\frac{a}{2} & 0 & 0 \end{bmatrix}\right) =$$

$$= \begin{bmatrix} 0.0151 & 0 & 0 \\ 0 & 0.0601 & 0 \\ 0 & 0 & 0.0453 \end{bmatrix} \tag{18}$$

for revolute joints:

$$I_C = \begin{bmatrix} \frac{1}{2}m(a^2+b^2) & 0 & 0 \\ 0 & \frac{1}{2}m(3(a^2+b^2)^2+h^2) & 0 \\ 0 & 0 & \frac{1}{2}m(3(a^2+b^2)^2+h^2) \end{bmatrix} \tag{19}$$

Remapping the inertia matrix to the frame of the link, we obtain:

$$I = I_C + m\left(\begin{bmatrix}-\frac{h}{2} & 0 & 0\end{bmatrix}\begin{bmatrix}-\frac{h}{2}\\0\\0\end{bmatrix} I_{3\times3} - \begin{bmatrix}-\frac{h}{2}\\0\\0\end{bmatrix}\begin{bmatrix}-\frac{h}{2} & 0 & 0\end{bmatrix}\right) =$$

$$= \begin{bmatrix}0.0004 & 0 & 0\\0 & 0.2400 & 0\\0 & 0 & 0.2400\end{bmatrix} and \begin{bmatrix}0.0004 & 0 & 0\\0 & 0.0864 & 0\\0 & 0 & 0.0864\end{bmatrix} \tag{20}$$

Finally, we can calculate the kinetic energy of the robot manipulator using the following equation:

All the computations are made with symbolic toolbox in Matlab, and to obtain a numerical value we need to substitute the values for the joints and velocities. In this specific case, we choose the following values:

jointValues $= [\frac{\pi}{2}, -0.2, \frac{\pi}{3}]$

velocityValues $= [5, 2, 8]$

accelerationValues $= [4, 1, 2]$

$$K = \frac{1}{2}\dot{q}^T\left[\sum_{i=1}^{n}\left(m_{l_i}(J_P^{l_j})^T J_P^{l_i} + (J_O^{l_i})^T R_i I_{l_i}^i R_i^T J_O^{l_i}\right)\right]\dot{q} = 8.8520J \tag{21}$$

Where the part inside the square brakets is the inertia matrix of the robot, and it's called B.

2.2 Potential energy

Regarding the potential energy, it's way easier to compute, since it's just the sum of the potential energy of each link, which is given by:

$$\mathcal{U} = \sum_{i=1}^{n}\mathcal{U}_i = -\sum_{i=1}^{n} m_{l_i} g_0^T \rho_{l_i} = 10.8680J \tag{22}$$

Recalling that the gravity vector is $g_0 = [0, 0, -9.81]$.

3 Third homework: Equations of motion

3.1 τ with derivatives

The lagrangian of a system is defined as the sum between the kinetic energy and the potential energy of the system.

$$\mathcal{L}(q,\dot{q}) = \mathcal{T}(q,\dot{q}) - \mathcal{U}(q) = \frac{1}{2}\dot{q}^T B(q)\dot{q} + \sum_{i=1}^{n} m_{l_i} g_0^T \rho_{l_i} \tag{23}$$

Now, to obtain the *tau*, the torques to be imparted to the robot's motors, we must calculate the derivative of the Lagrangian, first with respect to *dotq* and then with respect to q.

$$\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right)^T = B(q)\dot{q} \tag{24}$$

$$\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right)^T = \frac{1}{2}\left(\frac{\partial}{\partial \dot{q}}\left(\dot{q}^T B(q)\dot{q}\right)\right)^T - \left(\frac{\partial \mathcal{U}}{\partial q}\right)^T \tag{25}$$

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right)^T = B(q)\ddot{q} + \dot{B}(q)\dot{q} \tag{26}$$

If we substitute the values for the joints, elocities and acceleration, we obtain the following values for the torques:

$$\tau = \begin{bmatrix} 4.6117 \\ -15.9957 \\ 3.2947 \end{bmatrix} \tag{27}$$

3.2 τ without derivatives

If we don's want to compute all this derivatives, we have an other way to compute the torques, using the Coriolis and centrifugal matrices and the gravity vector. As we'll see the results for the torques are the same.

$$\sum_{j=1}^{n} b_{ij}(q)\ddot{q}_j + \sum_{j=1}^{n} c_{ij}(q,\dot{q})\dot{q}_j + g_i(q) = \tau_i \tag{28}$$

where the gravity vector is given by:

$$\sum_{j=1}^{n} m_{l_j} g_0^T J_{Pi}^{l_j}(q) \tag{29}$$

and the $c_i,_j$ terms is given by:

$$\sum_{j=1}^{n}\sum_{k=1}^{n} h_{ijk}(q)\dot{q}_k\dot{q}_j \tag{30}$$

After all the computations in Matlab, that are too long to be written here, we obtain the following values for the torques:

$$\tau = \begin{bmatrix} 4.6117 \\ -15.9957 \\ 3.2947 \end{bmatrix} \tag{31}$$

that are exactly the same as the ones obtained with the derivatives of the Lagrangian.

4 Fourth homework: Newton-Euler formulation

The NE formulation is based on the balance of all the forces acting on the generic link of the manipulator. It's divided in two steps: the first one is the forward dynamics, where we compute the linear and angular velocities and accelerations of the links, and the second one is the backward dynamics, where we compute the torques to be imparted to the motors. It's important to remember that the NE formulation is an algorithmic and numerical method, and it's not as precise as the Lagrangian formulation, but it's much faster and easier to compute.

Before starting with the computations, we need to understand what is the augmented link, which is the link that we are studying in the current iteration of the algorithm plus the mass of the motor and the inertia.

4.1 Forward dynamics

The forward recursion is performed for propagating link velocities and accelerations from i = 0 to i = n. For all the code see the Matlab code.

4.2 Backward dynamics

The backward recursion for propagating forces from i = n to i = 0. For all the code see the Matlab code.

Finally, as expected, torques computed with the NE formulation are the same as the ones computed with the Lagrangian formulation. In particular, we have the following values for the torques:

$$\tau = \begin{bmatrix} 4.6117 \\ -15.9957 \\ 3.2947 \end{bmatrix} \tag{32}$$

when the joint values, velocities and accelerations are the following:

$\text{jointValues} = [\frac{\pi}{2}, -0.2, \frac{\pi}{3}]$

$\text{velocityValues} = [5, 2, 8]$

$\text{accelerationValues} = [4, 1, 2]$

5 Control Scheme

5.1 PD control law with gravity compensation in joint space

$$\mathbf{B}(q)\ddot{q} + \mathbf{C}(q,\dot{q})\dot{q} + \mathbf{F}\dot{q} + \mathbf{g}(q) = \mathbf{g}(q) + \mathbf{K}_P\tilde{q} - \mathbf{K}_D\dot{q} \tag{33}$$

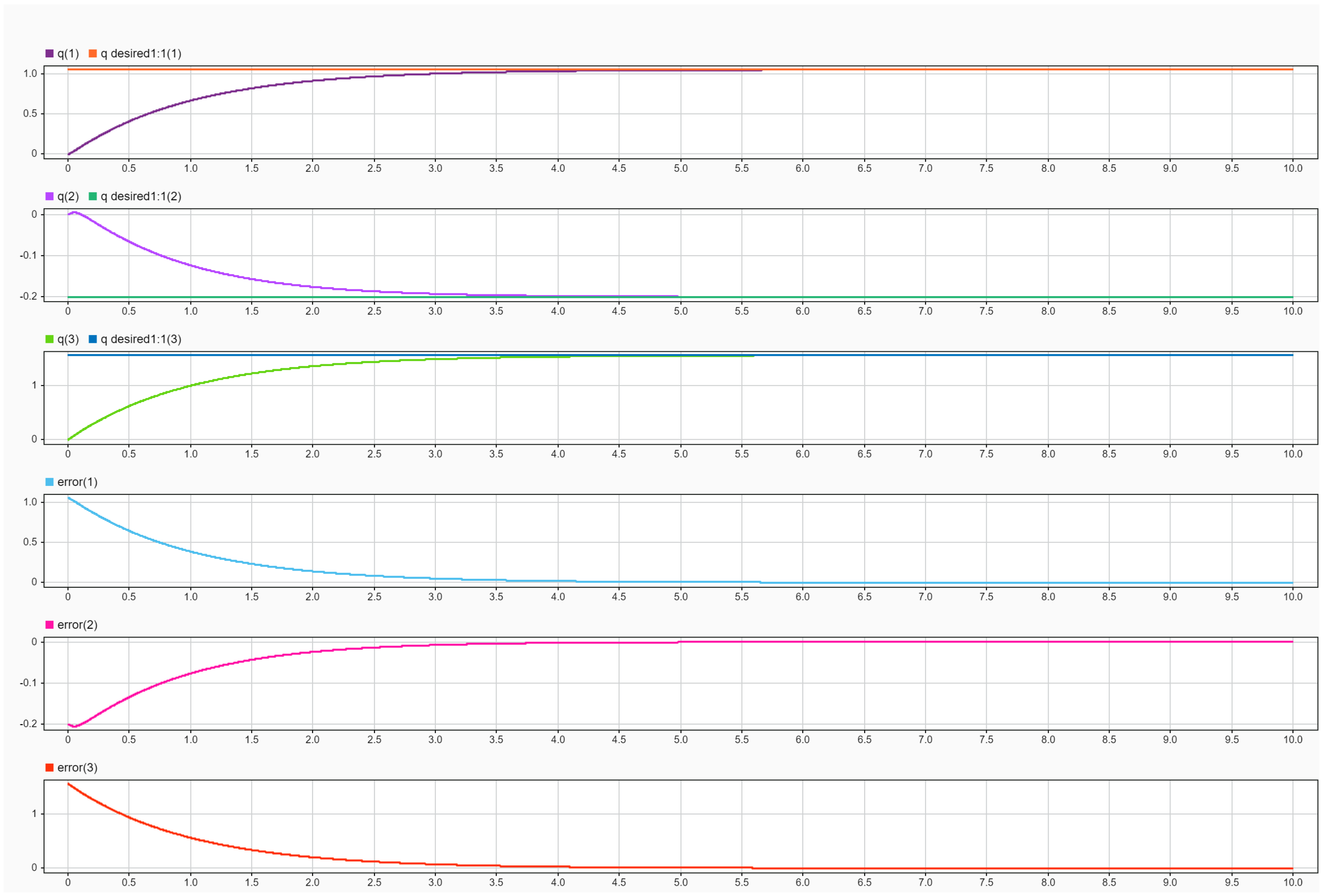

Figure 2: PD control law with gravity compensation in joint space

5.2 Inverse dynamic control law in joint space

$$\tau = \mathbf{B}(q)[\ddot{q}_d + K_D(\dot{q}_d - \dot{q}) + K_P(q_d - q)] + n(q, \dot{q}) \tag{34}$$

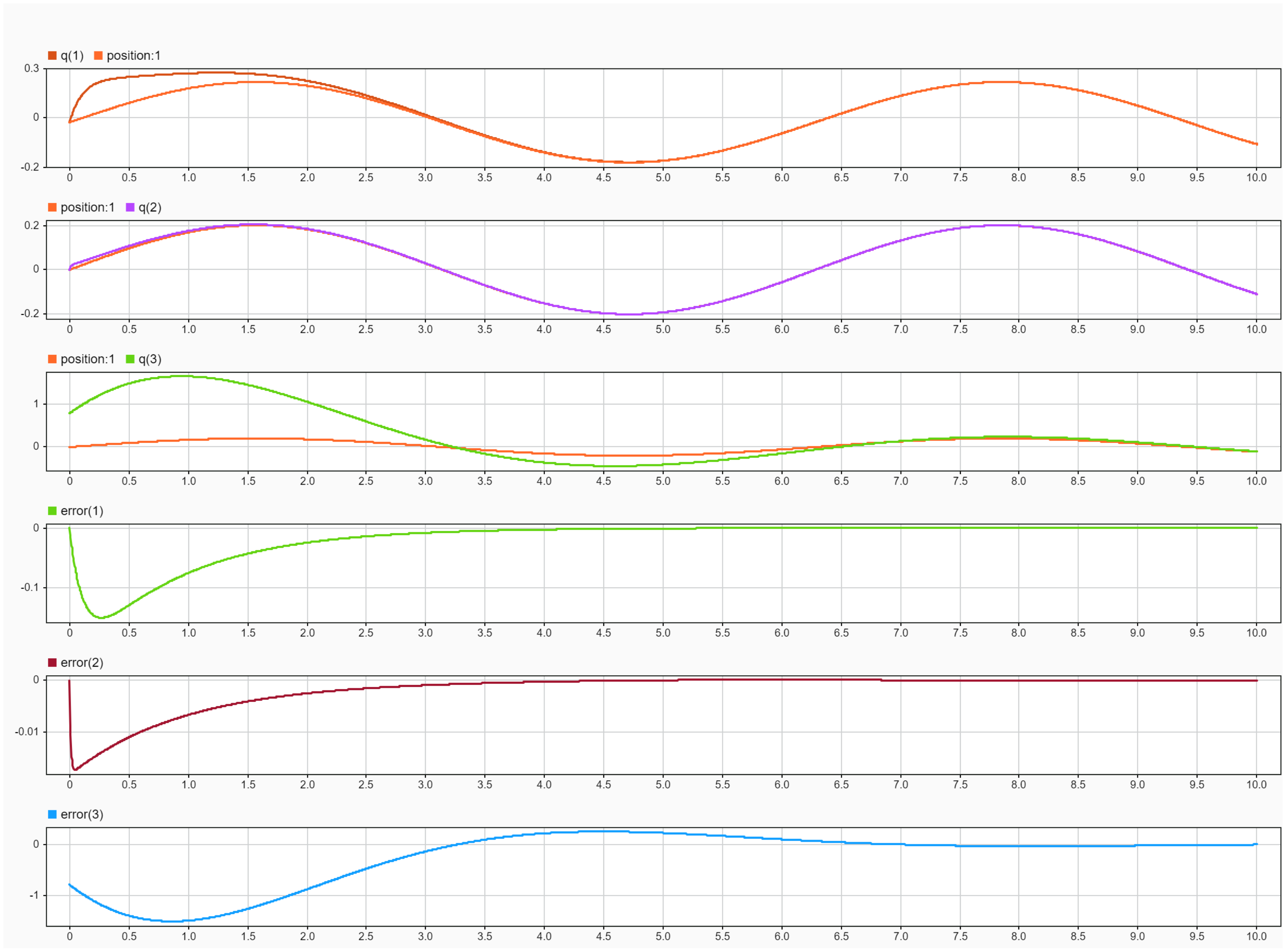

Figure 3: Inverse dynamic control law in joint space

5.3 PD control law with gravity compensation in the operational space

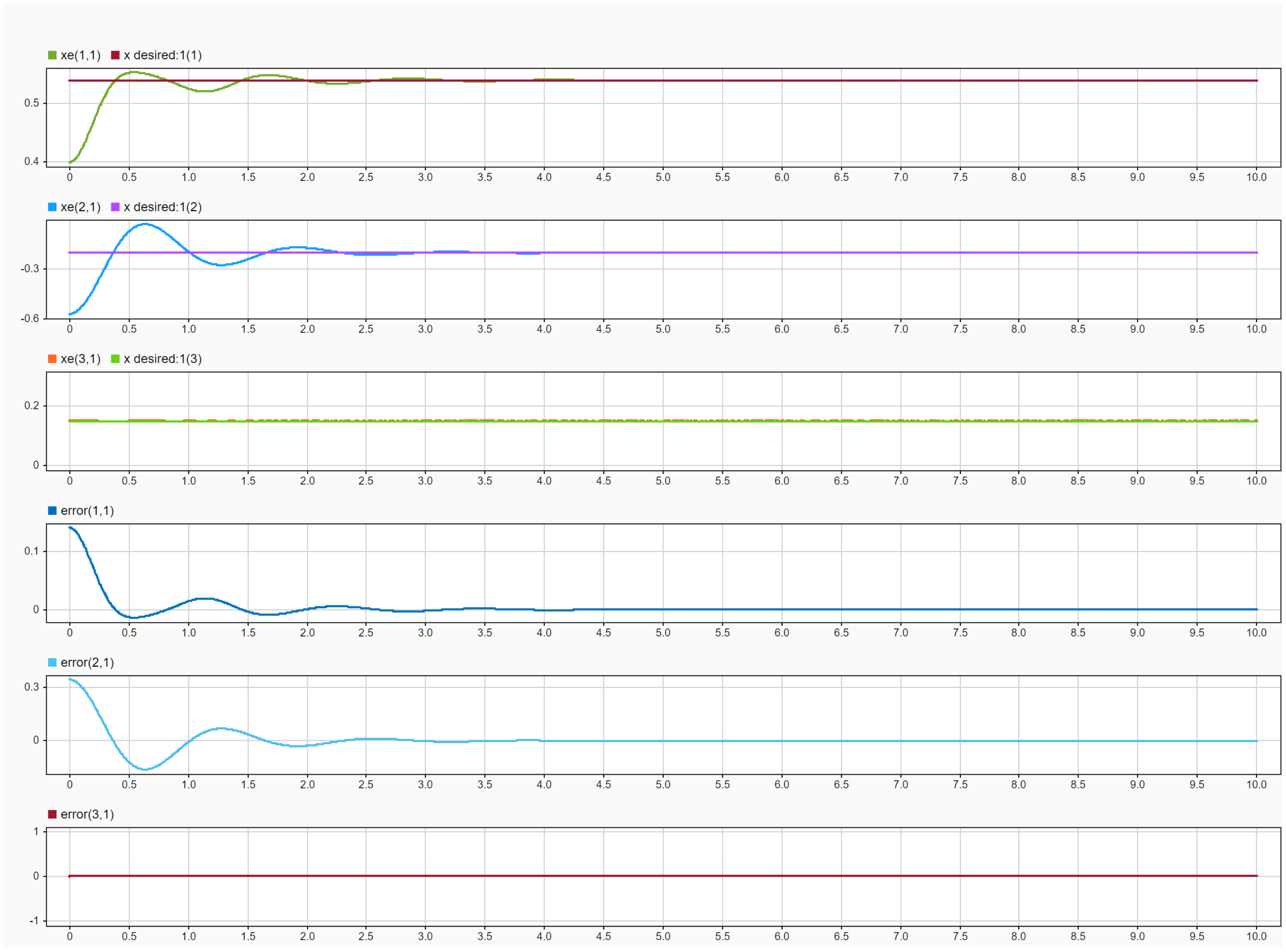

Figure 4: PD control law with gravity compensation in the operational space

5.4 Inverse dynamic control law in the operational space

Here I study two different cases: the first one is the case where the robot can reach the configuration, and the second one is the case where the robot cannot reach the configuration.

In the first one we have:

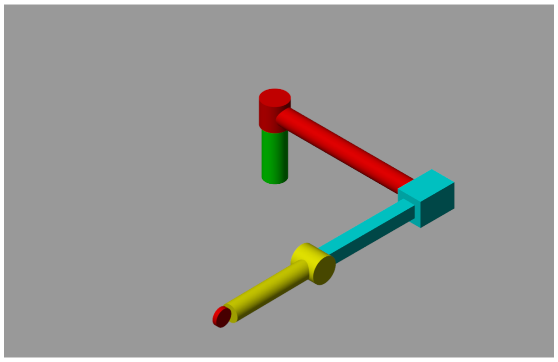
Figure 5: The robot can reach the position

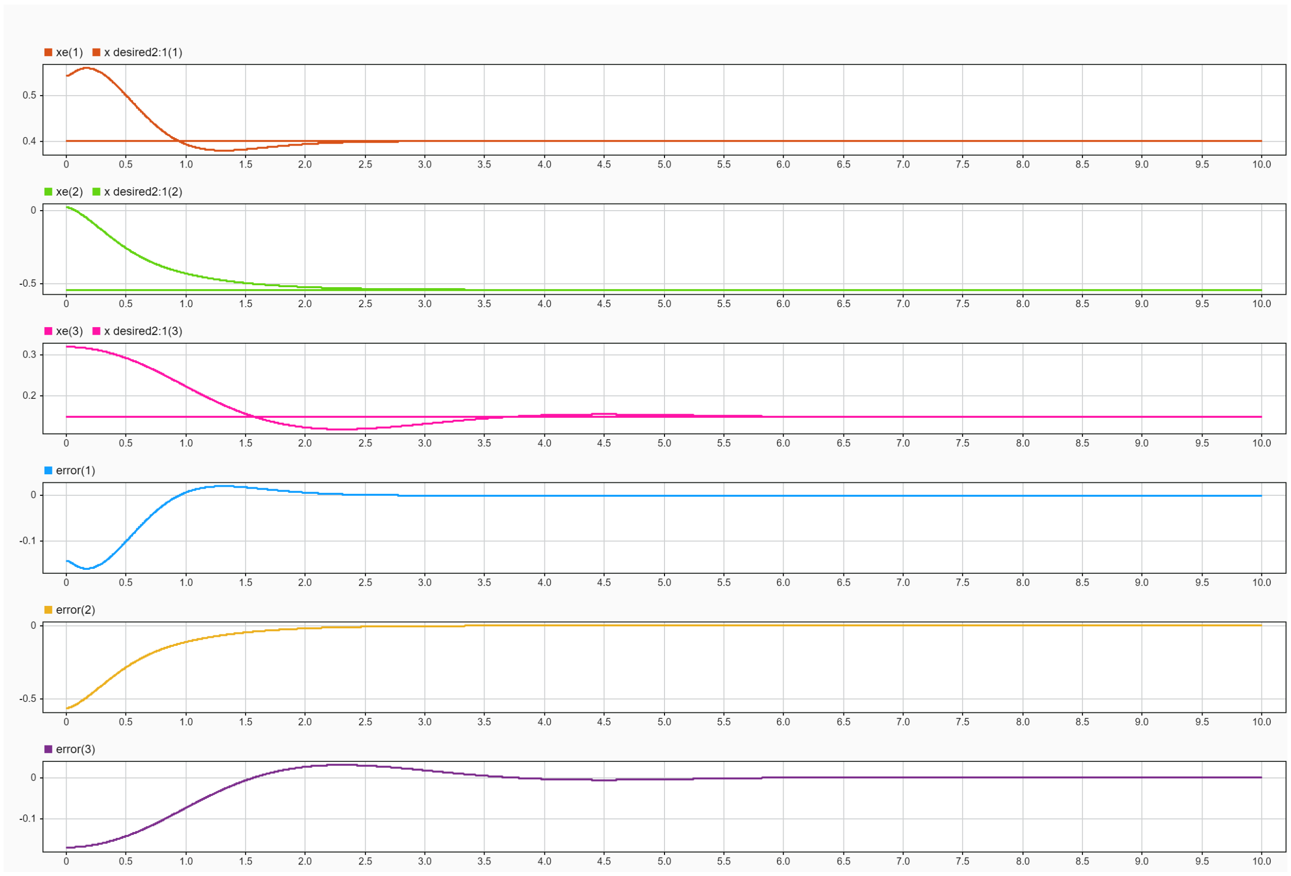

Figure 6: The robot can reach the position and the rror goes to zero

In the second one we have:

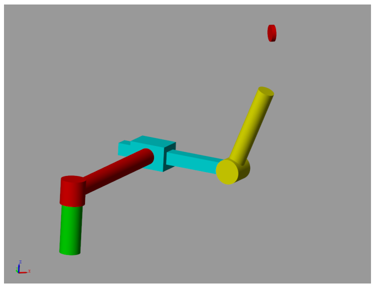

Figure 7: The robot can't reach the position

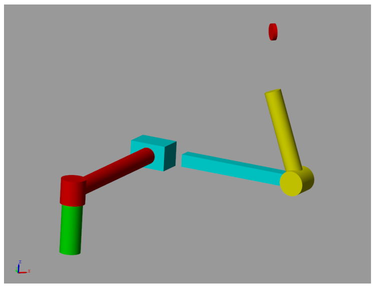

Figure 8: The robot can't reach the position

where we can note that the robot if can't reach the position, it will obscillate but it does not reach the zero steady state error.

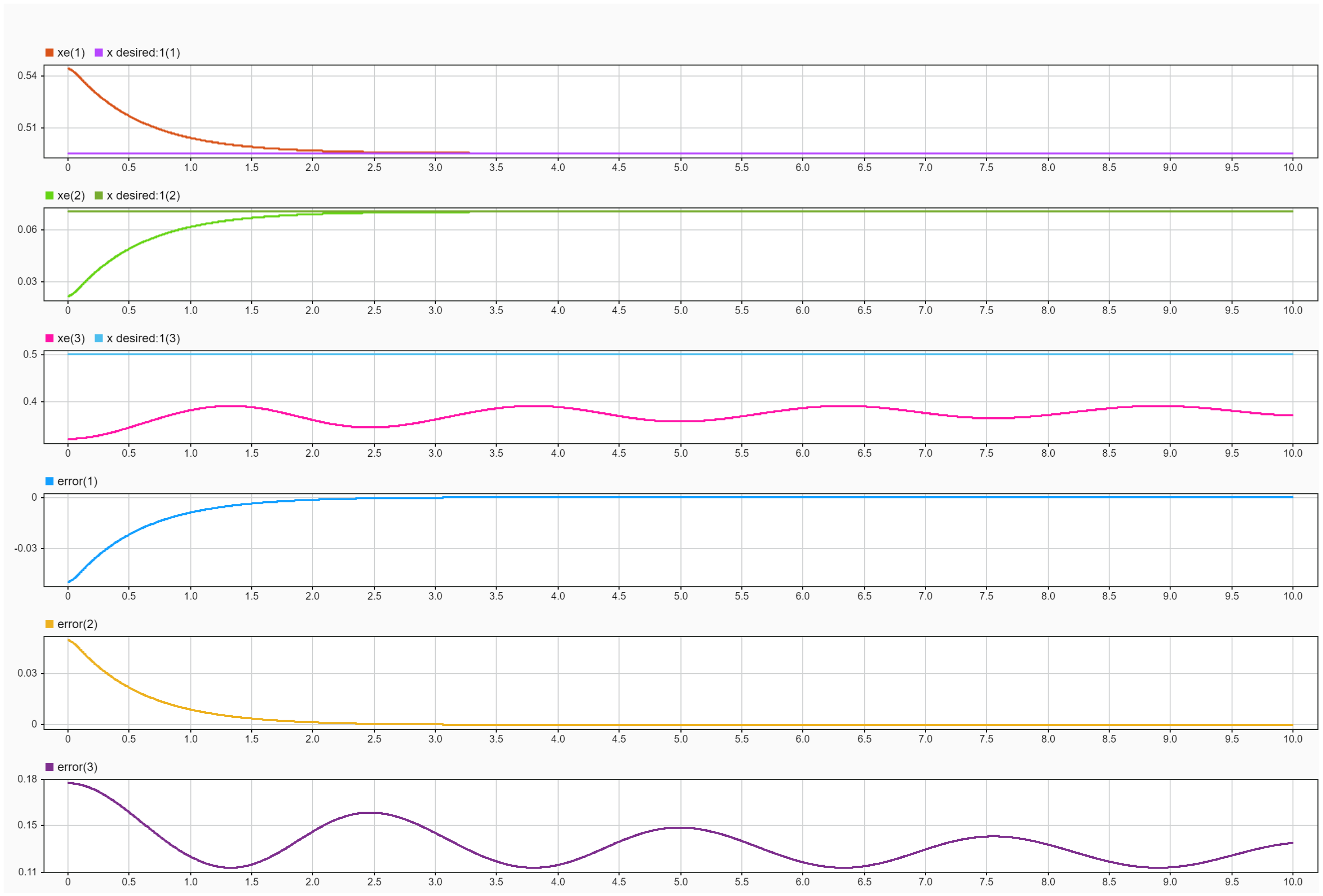

Figure 9: The robot can't reach the position and the error NOT goes to zero

5.5 Compliance control

In this control scheme we modify the PD with gravity compensation control scheme.

Also in the compliance control we have 2 distinct cases. The first on is the passive compliance control, where the wall acts like a spring. Off course the robot in both cases will not reach the desired position, due to the wall that is pushing it back.

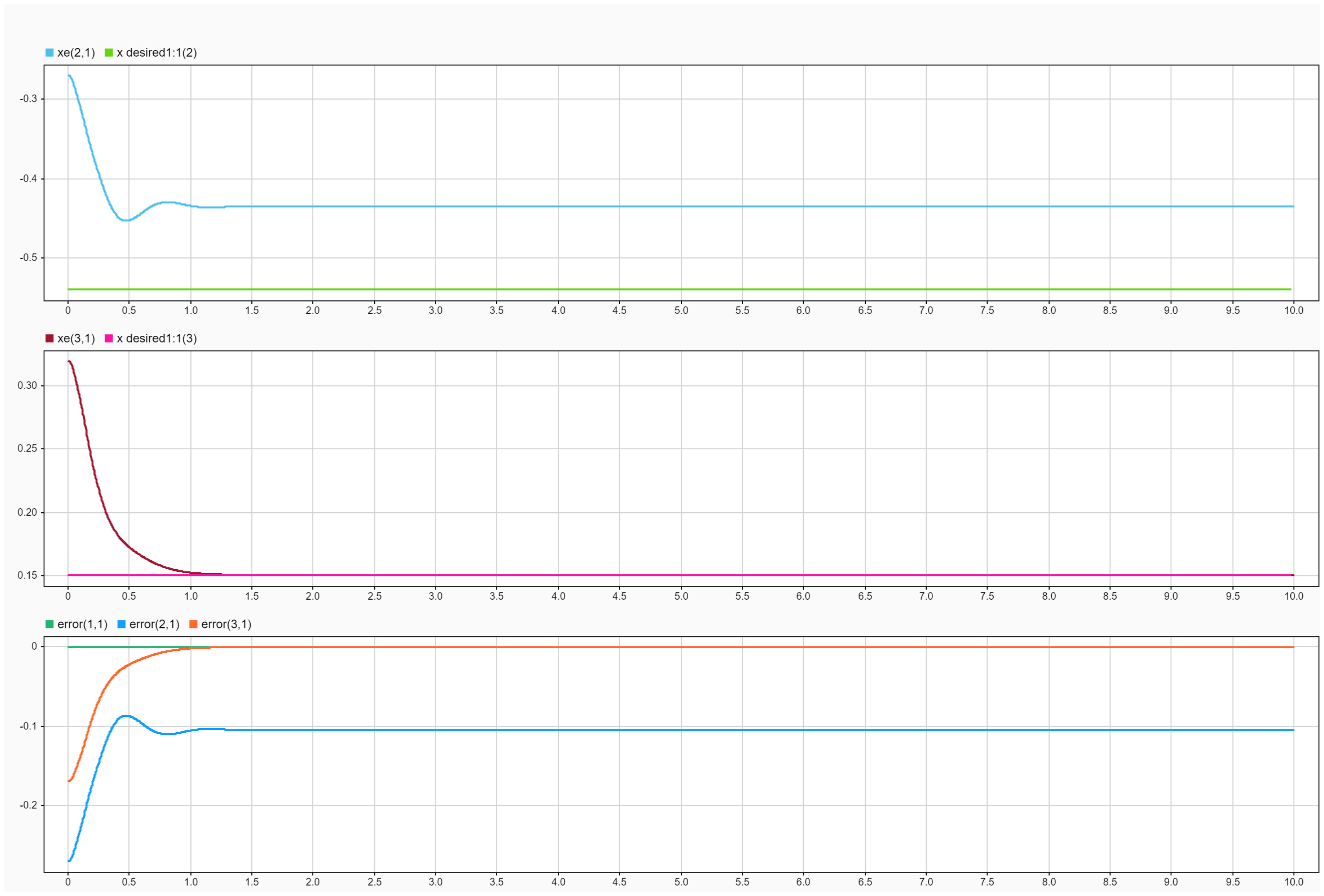

Figure 10: passiveCompliance

in the second case, that is very different from the first one, we have an active compliace control, where, with an s-function, we compute dinamicly the error between the desired position and the actual one, and we use this error to compute the torques to be imparted to the motors.

In particular we use this particular Jacobian, derivative from this formula:

$$\dot{\tilde{x}} = -T_A^{-1}(\phi_{d,e}) \begin{bmatrix} R_d^T & O \\ O & R_d^T \end{bmatrix} J(q)\dot{q} = -J_{Ad}(q, \tilde{x})\dot{q} \tag{35}$$

In super important to understand that the T matrix, is the transformation matrix from the end effector to the desired position, as reported in the matlab code below:

```
Td = [eye(3), desired_position;
      0 0 0 1];

% end effector frame
Te = A_b_ee;

% desired position in the end effector frame
T_D_e = [Td(1:3, 1:3)' * Te(1:3, 1:3), Td(1:3, 1:3)' *
    (Te(1:3, 4) - Td(1:3, 4));
            0 0 0 1];

phi = atan2(T_D_e(2, 3), T_D_e(1, 3));
gamma = acos(T_D_e(3, 3));
% psi = atan2(A_b_ee(3, 2), -A_b_ee(3, 1));

T_zyz = [0, -sin(phi), cos(phi) * sin(gamma);
            0, cos(phi), sin(phi) * sin(gamma);
            1, 0, cos(gamma)];

Ta = blkdiag(eye(3), T_zyz);

Jad = inv(Ta) * blkdiag(Td(1:3, 1:3)', Td(1:3, 1:3)')
   * J_analytical;

block.OutputPort(1).Data = Jad(1:3, 1:3);
```

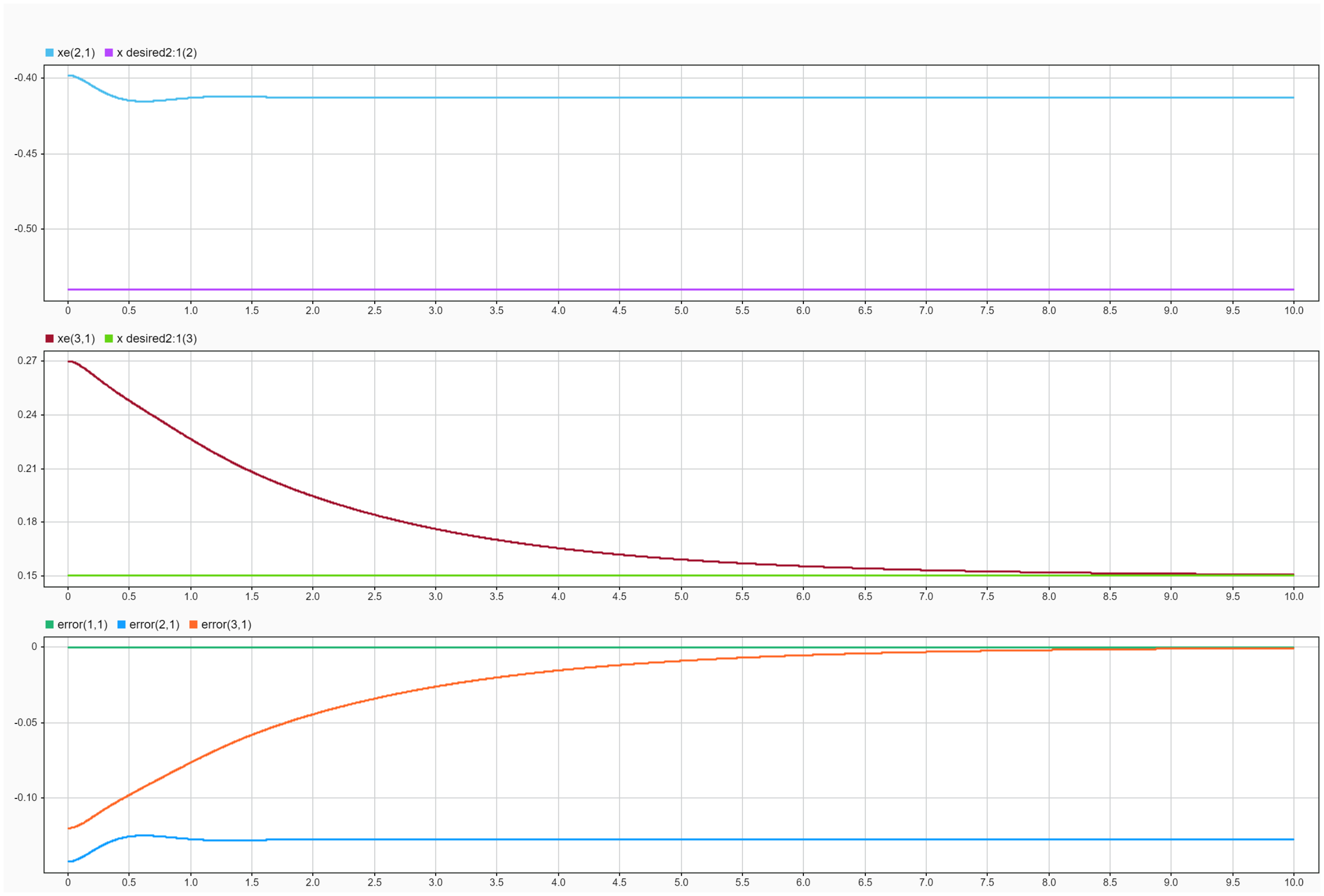

Figure 11: activeCompliance

As expected we have an error on the second joint, where the "spring" of the wall is acting, and the robot is not able to reach the desired position.

5.6 Force control

5.6.1 Block scheme of force control with inner position loop

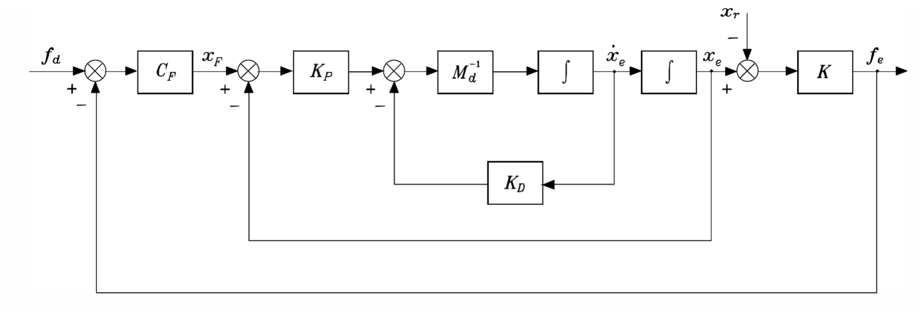

Figure 12: Block scheme of force control with inner position loop.

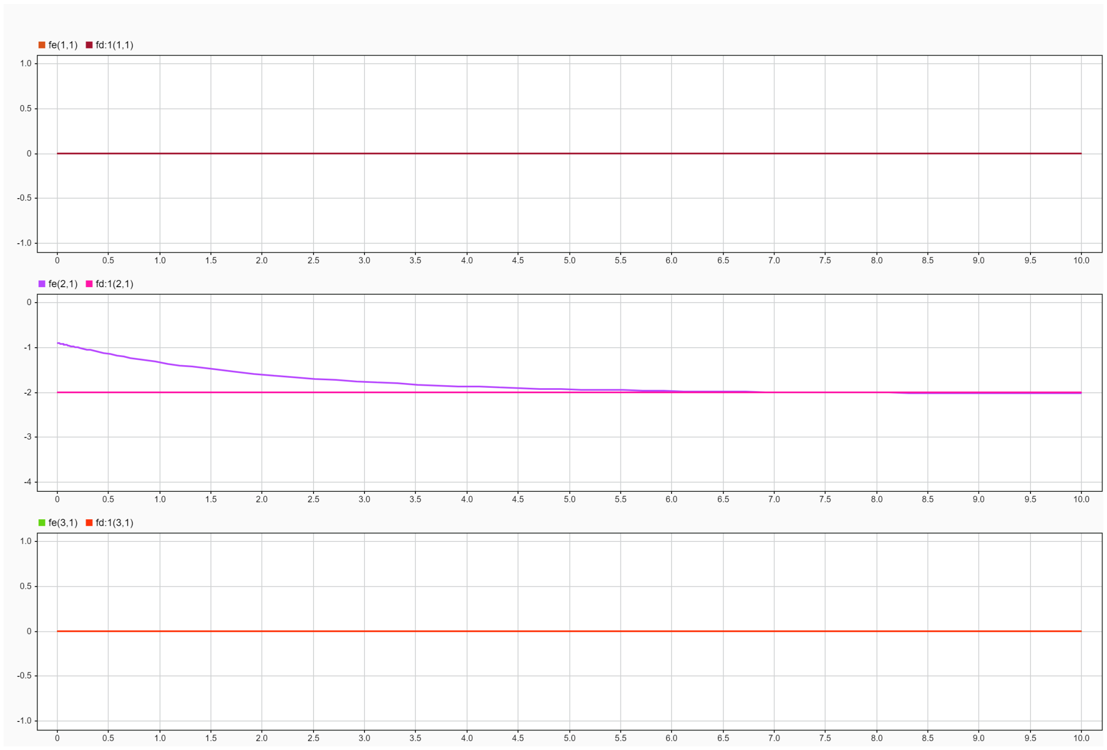

Figure 13: force control inner position

In the control scheme, if we assume Cf as a constant matrix we cannot reach the zero steady state error. But if we use an integrator, we will have the zero error.

$$C_F(s) = K_F + K_i \frac{1}{s} \tag{36}$$

5.6.2 Block scheme of force control with inner velocity loop

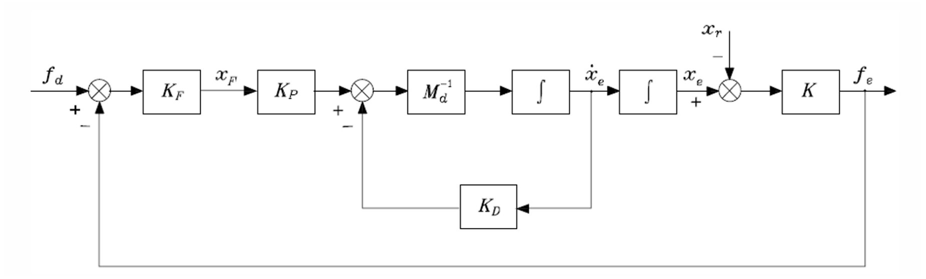

Figure 14: Block scheme of force control with inner velocity loop.

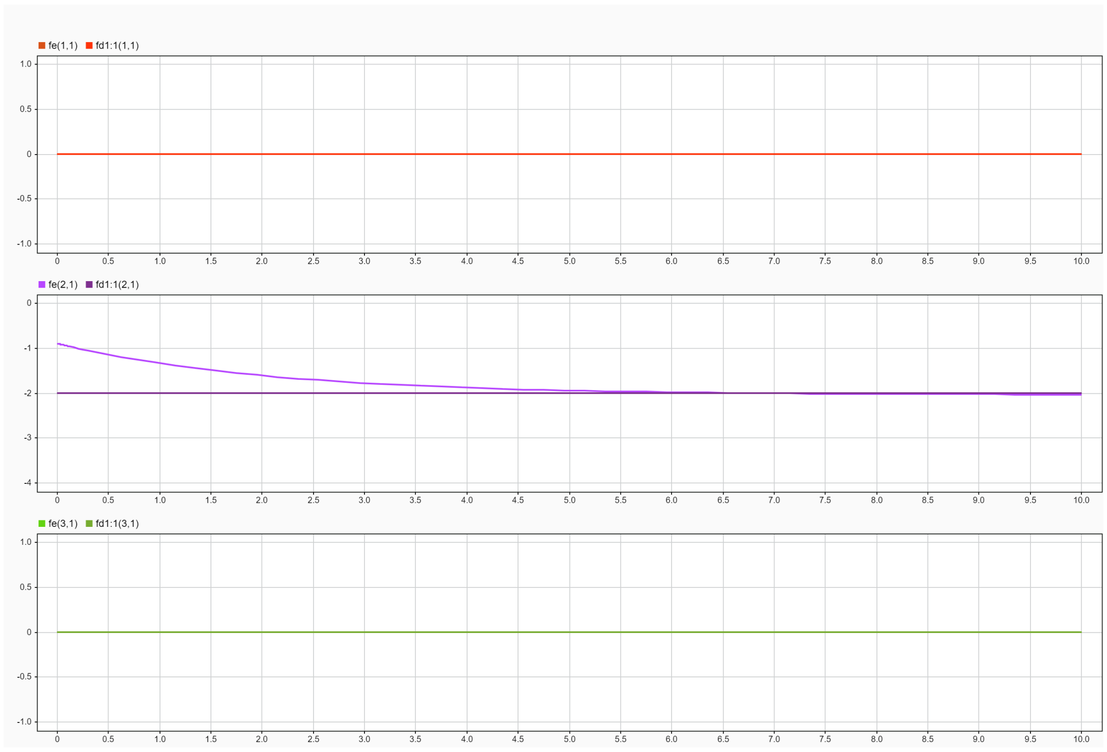

Figure 15: force control inner velocity loop

5.6.3 Block scheme of parallel force/position control

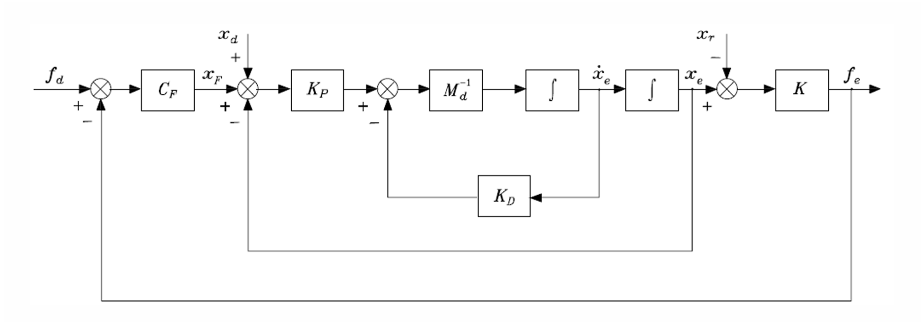

Figure 16: Block scheme of parallel force/position control.

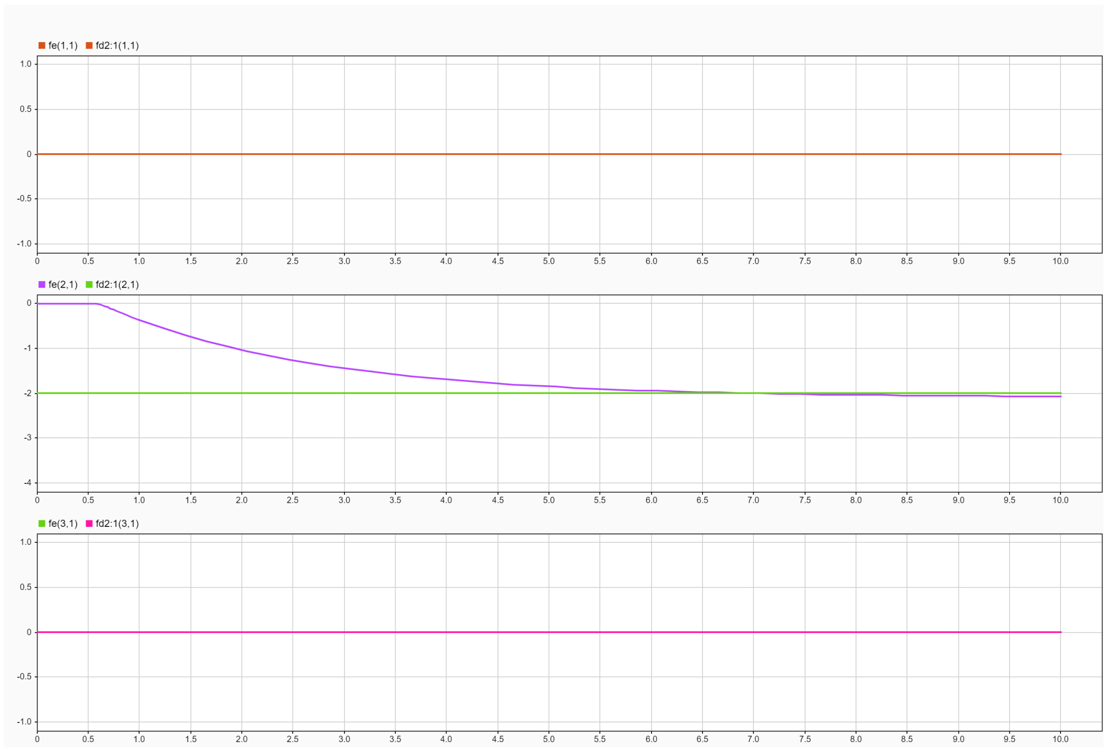

Figure 17: parallel force/position control

5.7 Adaptive Control

Adaptive control represents an advanced control strategy that addresses the challenge of system uncertainty by estimating unknown parameters during operation. This control approach is particularly valuable when dealing with systems whose dynamic parameters are not precisely known or may vary over time.

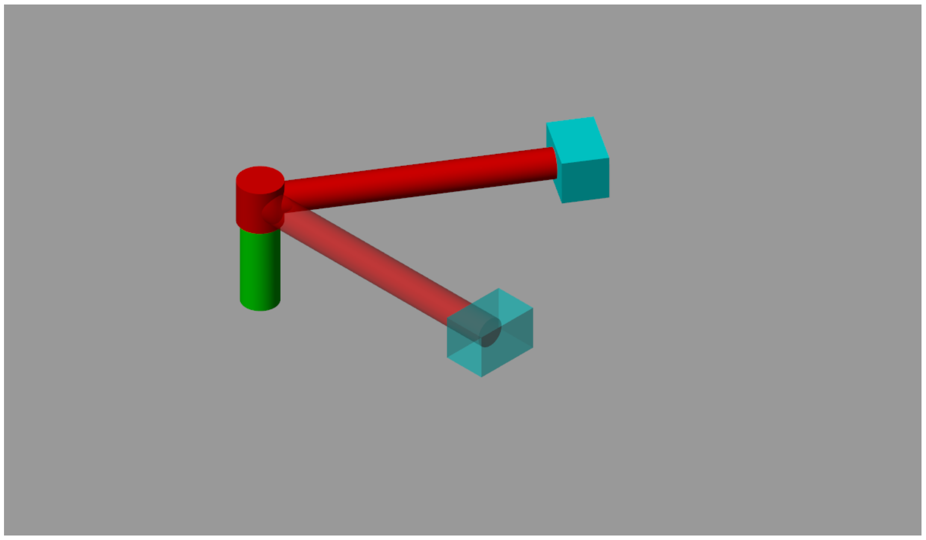

Figure 18: Adaptive control scheme

Traditional control schemes, such as the inverse dynamic control law in joint space:

$$\tau = B(q)\ddot{q}_d + C(q,\dot{q})\dot{q}_d + g(q) + F\dot{q}_d + K_D\dot{\tilde{q}} + K_P\tilde{q} \tag{37}$$

require precise knowledge of the dynamic parameters contained in the matrices $B(q)$, $C(q,\dot{q})$, and $g(q)$. However, in practice, **these parameters are often uncertain or imperfectly known**, which can significantly degrade control performance.

The fundamental objective of adaptive control is to estimate the unknown dynamic parameters of a system while simultaneously minimizing the tracking error along a desired trajectory. This dual purpose makes adaptive control particularly suited for robotic applications where model uncertainties are common.

Parameter Estimation Strategy

When the exact dynamic parameters are unknown, we can introduce parameter estimation into the control law. Consider the modified control equation where we replace the unknown parameters with their estimates:

$$\tau = B(q)\ddot{q}_r + C(q,\dot{q})\dot{q}_r + F\dot{q}_r + g(q) + K_D\sigma = Y(q,\dot{q},\dot{q}_r,\ddot{q}_r)\hat{\Theta} + K_D\sigma \tag{38}$$

where:

- $\hat{\Theta}$ represents the vector of estimated parameters
- $Y(q, \dot{q}, \dot{q}_r, \ddot{q}_r)$ is the regressor matrix that relates the parameters to the dynamics
- $\sigma = \dot{\tilde{q}} + \Lambda\tilde{q}$ is a filtered tracking error
- $\dot{q}_r = \dot{q}_d - \Lambda\tilde{q}$ and $\ddot{q}_r = \ddot{q}_d - \Lambda\dot{\tilde{q}}$ are reference signals

Parameter Update Law

The key innovation in adaptive control lies in the parameter update mechanism. The parameter estimation error is defined as:

$$\tilde{\Theta} = \Theta - \hat{\Theta} \tag{39}$$

where Θ represents the true (but unknown) parameter vector.

The adaptive update law is designed to minimize this parameter error while ensuring system stability:

$$\begin{cases} \tau = Y(q, \dot{q}, \dot{q}_r, \ddot{q}_r)\hat{\Theta} + K_D\sigma \\ \dot{\hat{\Theta}} = \Gamma Y^T(q, \dot{q}, \dot{q}_r, \ddot{q}_r)\sigma \end{cases} \tag{40}$$

where $\Gamma = \Gamma^T \succ 0$ is a positive definite gain matrix that controls the rate of parameter convergence.

The adaptive control law ensures that:

1. The tracking error $\tilde{q}$ converges to zero as $t \to \infty$
2. The parameter estimates $\hat{\Theta}$ remain bounded
3. The closed-loop system is globally stable

The parameter update law acts as a gradient descent algorithm on the tracking error, effectively adjusting the parameter estimates to minimize the discrepancy between the actual and desired system behavior.

The adaptive control approach provides a systematic framework for handling parameter uncertainty in robotic systems, making it particularly valuable for applications where precise system identification is challenging or where parameters may vary during operation.

To illustrate the concept, we consider a simple example of a 1 DOF arm with the following equations:

$$I\ddot{q} + F\dot{q} + mgd\sin q = \tau \tag{41}$$

where $G = mgd\sin q$ represents the gravitational term.

This can be rewritten in the regressor matrix form as:

$$\tau = \begin{bmatrix} \ddot{q} & \dot{q} & \sin q \end{bmatrix} \begin{bmatrix} I \\ F \\ G \end{bmatrix} \tag{42}$$

where:

- $Y(q, \dot{q}, \ddot{q}) = \begin{bmatrix} \ddot{q} & \dot{q} & \sin q \end{bmatrix}$ is the regressor matrix
- $\Theta = \begin{bmatrix} I \\ F \\ G \end{bmatrix}$ is the parameter vector containing inertia, friction, and gravity terms

In this case, the Θ vector is already linear, which simplifies the parameter estimation process. The adaptive control law can directly estimate these physical parameters using the update rule previously described.

Substituting in the equation of the error signal:

$$\tau = I\ddot{q}_r + F\dot{q}_r + mgd\sin q + K_D\sigma \tag{43}$$

where $G = mgd\sin q$ represents the gravitational term.

For the first experiment, a sinusoidal trajectory was used. The real parameters $[I, F, G]$ are in order: $[3, 2, -4.164]$, starting from an initial estimation (initial condition of the integral) $[1, 1, -9.81]$.

After an initial discrepancy, the parameters settle down to an almost steady number, and the position and velocity errors approach zero. Note that the parameters may not converge to the real ones, but are inside the null space of $Y(q, \dot{q}, \ddot{q}_r, \ddot{q}_r)$.

In this case, the Y function is missing the dependence on $\dot{q}$, because in our model there is no parameter multiplied by $\dot{q}$.

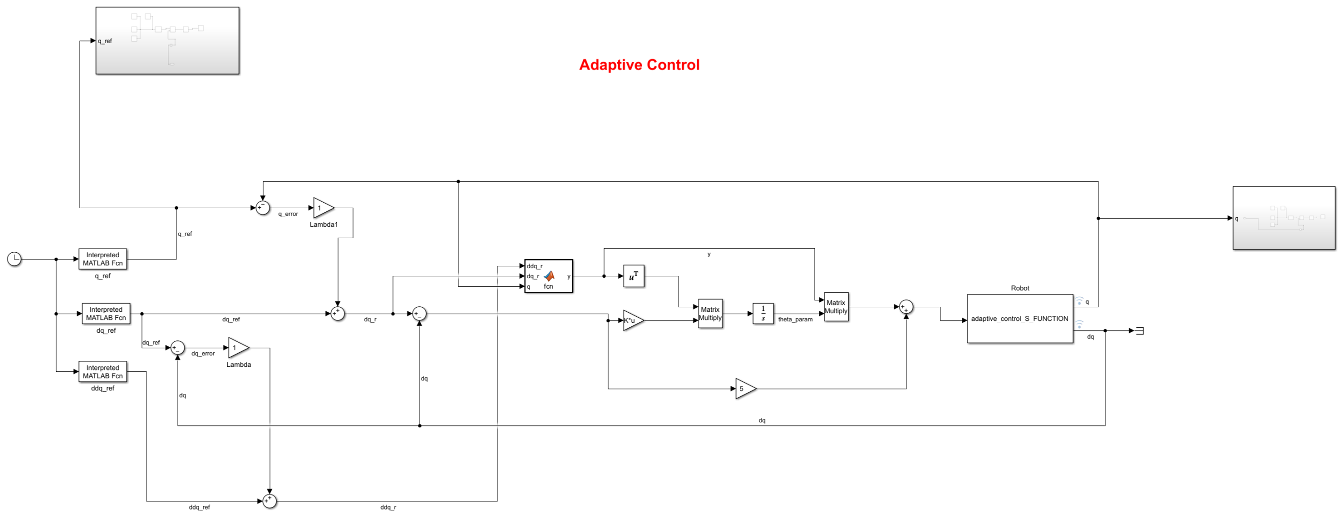

Figure 19: Adaptive control scheme

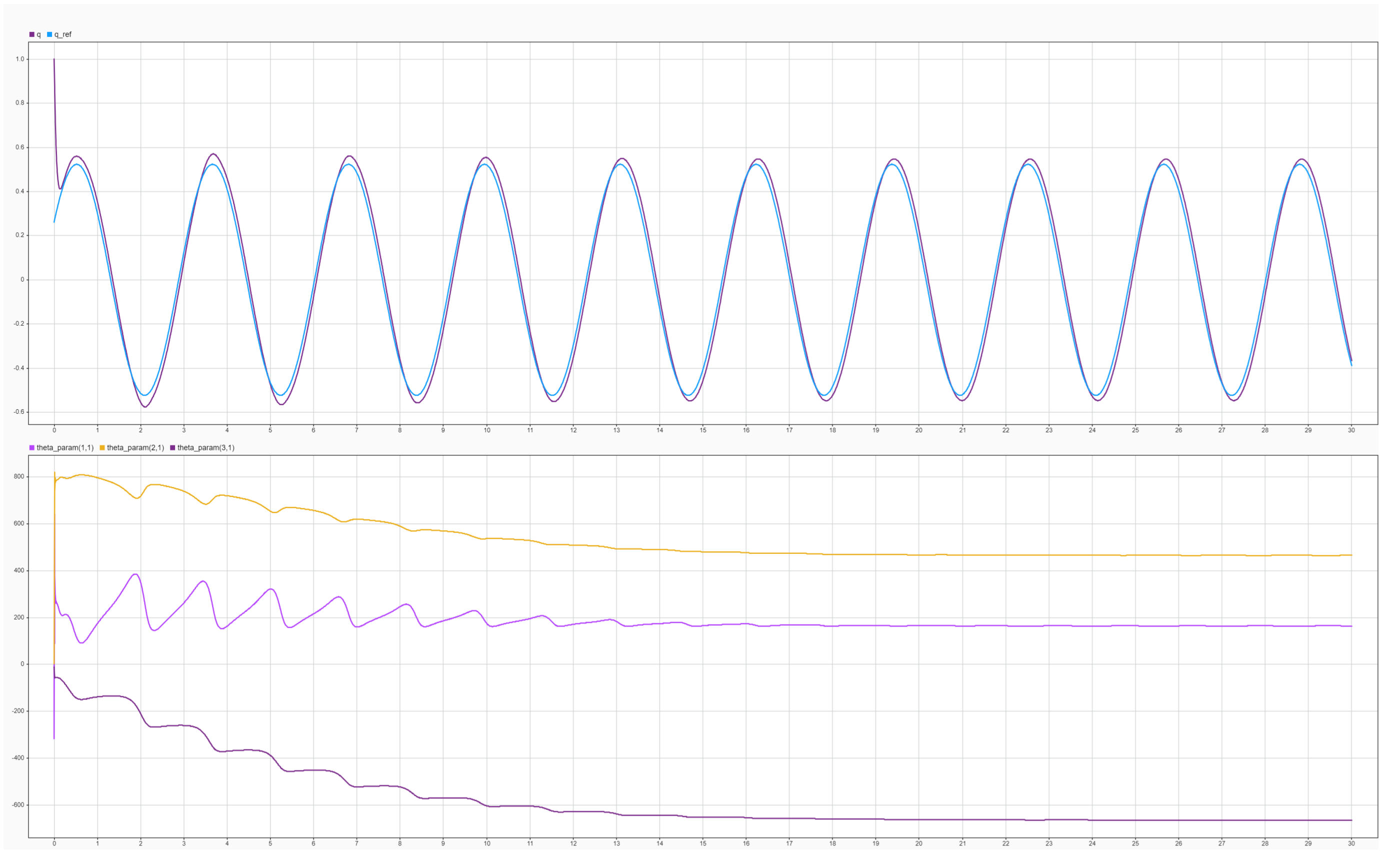

Figure 20: Adaptive control scheme

5.8 Admittance Control

This scheme is used to achieve a compliant behavior of the manipulator, when there is no direct access to torque control. Basically, we measure the force at the tip, and compute a compliant frame x_t starting from a desired frame x_d. We can control the stiffness of the robot against the environment by acting on the gain of the impedance control, that are different from the ones of the robot.

In this way we can choose how the robot interacts with the environment, if the arm is in free-motion, the control is equal to an inverse dynamic control, where $x_t = x_d$.

The compliance control is done by solving this equation:

$$M_t\ddot{Z} + K_{Dt}\dot{Z} + K_{Pt}Z = h_e^d \tag{44}$$

where

$$Z = x_d - x_t = -\begin{bmatrix} O_{d,t} \\ \phi_{d,t} \end{bmatrix} \tag{45}$$

so by rearranging the terms:

$$\ddot{x}_t = M_t^{-1}(-h_e^d + K_d\dot{z} + k_p\dot{z} + M_t\ddot{x}_d) \tag{46}$$

then we just need to double integrate to have $\dot{x}_t$ and x_t

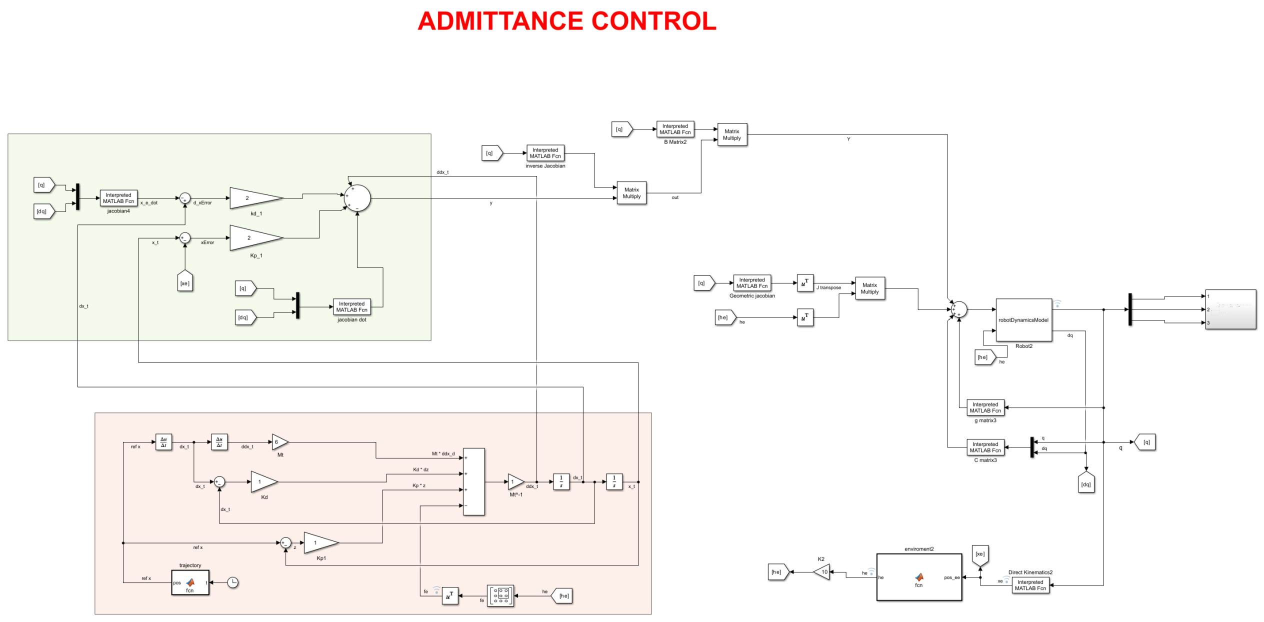

Figure 21: Admittance control scheme