Foundation of Robotics

Technical Project

Supervisor

Esteemed Prof. Bruno Siciliano

Candidates

Andrea Morghen
P38000230

Salvatore Del Peschio
P38000190

Contents

Chapter 1

Kinematics

1.1 Direct kinematics

To start solving the tasks of the technical project, our initial objective is to acquire the kinematic model of the robot. At this purpose, the first step involves attaching the reference frames to the manipulator's structure. Following the Denavit-Hartenberg convention it leads to the configuration illustrated in figure:

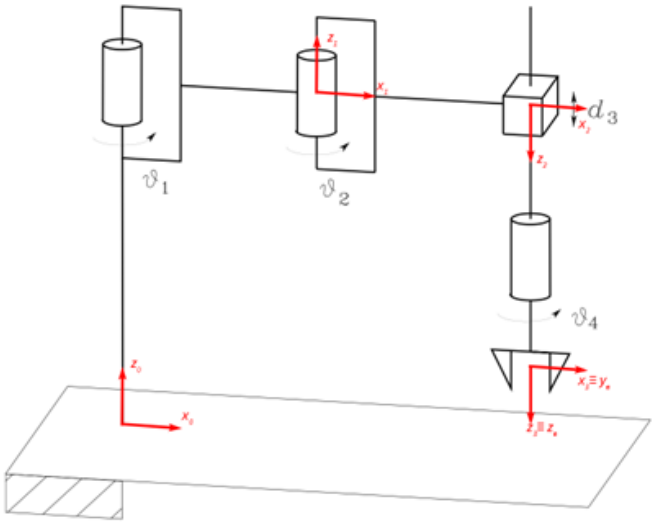


Figure 1.1: Reference frames

Where the frames are positioned to comply with the approach-sliding-normal convention. The resulting DH parameters table is shown below.

Link	a_i	α_i	d_i	θ_i
1	0.5	0	1	θ_1
2	0.5	π	0	θ_2
3	0	0	0.5+d_3	0
4	0	0	0	$\theta_4 - \frac{\pi}{2}$

Using the parameters, it is possible to determine the transformation matrix associated with the end-effector frame.

$$T_e = A_1^0 A_2^1 A_3^2 A_4^3 = \begin{bmatrix} -sin(\theta_1+\theta_2-\theta_4) & cos(\theta_1+\theta_2-\theta_4) & 0 & \frac{cos(\theta_1+\theta_2)}{2}+\frac{cos(\theta_1)}{2} \\ cos(\theta_1+\theta_2-\theta_4) & sin(\theta_1+\theta_2-\theta_4) & 0 & \frac{sin(\theta_1+\theta_2)}{2}+\frac{sin(\theta_1)}{2} \\ 0 & 0 & -1 & \frac{1}{2}-d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From this transformation matrix it is possible to extract the direct form of the kinematics equations.

$$x_e = k(q) = \begin{bmatrix} \frac{cos(\theta_1+\theta_2)}{2}+\frac{cos(\theta_1)}{2} \\ \frac{sin(\theta_1+\theta_2)}{2}+\frac{sin(\theta_1)}{2} \\ \frac{1}{2}-d_3 \\ \theta_1+\theta_2-\theta_4 \end{bmatrix}$$

1.2 Differential Kinematics

For the study of differential kinematics, we must compute the geometric Jacobian, using the following formula:

$$J = \begin{bmatrix} z_0 \times (p_4-p_0) & z_1 \times (p_4-p_1) & z_2 & z_3 \times (p_4-p_3) \\ z_0 & z_1 & 0 & z_3 \end{bmatrix}$$

The resulting Jacobian matrix is as follow:

$$J = \begin{bmatrix} -\frac{sin(\theta_1+\theta_2)}{2}-\frac{sin(\theta_1)}{2} & -\frac{sin(\theta_1+\theta_2)}{2} & 0 & 0 \\ \frac{cos(\theta_1+\theta_2)}{2}+\frac{cos(\theta_1)}{2} & \frac{cos(\theta_1+\theta_2)}{2} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix}$$

It's worth noting that the null rows correspond to the rotations around the x and y axes, and they can be eliminated obtaining, resulting in a simplified, non-singular Jacobian matrix.

$$J = \begin{bmatrix} -\frac{sin(\theta_1+\theta_2)}{2}-\frac{sin(\theta_1)}{2} & -\frac{sin(\theta_1+\theta_2)}{2} & 0 & 0 \\ \frac{cos(\theta_1+\theta_2)}{2}+\frac{cos(\theta_1)}{2} & \frac{cos(\theta_1+\theta_2)}{2} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix}$$

1.3 Kinematics singularities

Evaluating the determinant of the Jacobian we obtain:

$$det(J(q)) = \frac{sin(\theta_2)}{4}$$

Setting it to zero yields to the solution:

$$\theta_2 = k\pi \quad k \in Z$$

It's important to note that the only physical attainable singular configuration is $\theta_2 = 0$, that represent the pose when the link 2 is fully stretched so to reach the boundaries of the workspace. We can also observe that the first 2x2 block of the matrix mirrors the structure of the two-planar link manipulator. The SCARA manipulator, up to its second joint, share a comparable structure, and thus inherits its singularities.

1.4 Manipulability ellipsoids

Analyze velocity and force manipulability for the supporting structure, plotting the relative ellipsoids for a significant number of positions of the end-effector within the workspace.

Velocity and force ellipsoids play a crucial role in evaluating a manipulator's ability to generate speed or forces at its end effector in various directions. The characteristics of manipulability ellipsoids are dependent on the Jacobian matrix, particularly the orientation of their principal axes, determined by the eigenvectors, and their lengths, influenced by the eigenvalues of the Jacobian matrix. The velocity ellipsoids are defined by the following relation:

$$v_e^T \left(J\left(q\right) J\left(q\right)^T \right)^{-1} v_e = 1$$

In cases where the manipulator is redundant, we can derive a general formula utilizing the pseudoinverse. Let's consider fixed manipulator configurations as follows:

$$q_1 = \begin{bmatrix} 0 & \frac{\pi}{2} & 0 & 0 \end{bmatrix}$$

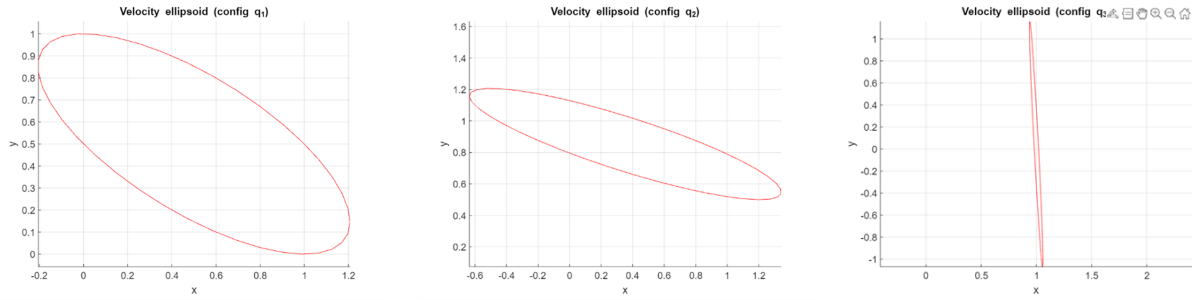
$$q_2 = \begin{bmatrix} \frac{\pi}{4} & \frac{\pi}{4} & 0.1 & \frac{\pi}{18} \end{bmatrix}$$

$$q_3 = \begin{bmatrix} 0 & \frac{5\pi}{180} & 0 & 0 \end{bmatrix}$$

The most notable configuration is the latest corresponding to the elbow almost fully extended. The variation of the angle θ_2 will vary the manipulability measure, representing the ellipsoid's volume; This measure represents a simple, albeit non-quantitative, indicator of distance from singular configurations.

$$w\left(q\right) = \left| det\left(J\left(q\right)\right) \right| = \frac{\left| sin\left(\theta_2\right) \right|}{4}$$

Now, let's visualize the manipulability ellipsoids in velocity. It can be seen how volume decreases as the manipulator approaches an outstretched posture.

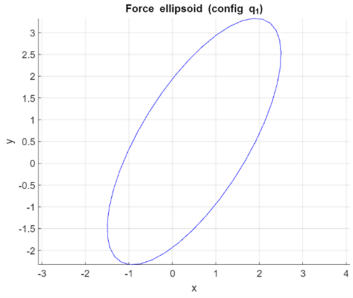


(a) Velocity ellipsoid in q_1 (b) Velocity ellipsoid in q_2 (c) Velocity ellipsoid in q_3

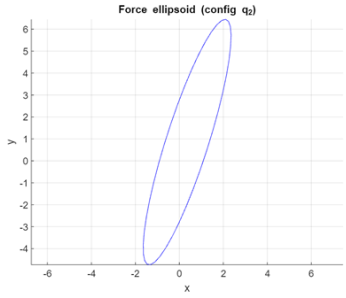
Moving on to the analysis of force ellipsoids, which are described by the equation:

$$\gamma_e^T \left(J\left(q\right) J\left(q\right)^T \right) \gamma_e = 1$$

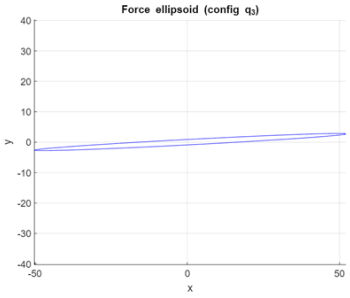
Like our previous approach, let's plot the force ellipsoids for the manipulator with the same fixed postures.



(a) Force ellipsoid in q_1



(b) Force ellipsoid in q_2



(c) Force ellipsoid in q_3

Upon examining the two equations governing velocity and force ellipsoids, it becomes clear that the central components of these equations are inversely related. This leads us to the following concept: a direction in which there is high manipulability in velocity corresponds to a direction in which there is low manipulability in force. This result is clearly visible in the third configuration.

Chapter 2

Trajectory planning

Plan the trajectory along a path characterized by at least 11 points within the workspace, in which there are at least one straight portion and one circular portion and the passage for at least 4 via points.

For the trajectory, we carefully selected a path that met all the constraints outlined in the project requirements while achieving a meaningful task. Among several possible options, we thought on a practical operation that a robotic arm, such as a SCARA (Selective Compliance Articulated Robot Arm), could perform cutting a sheet into the specific shape.The trajectory was completely generated in the operational space.
Each segment of this path was meticulously computed individually and seamlessly connected while ensuring both spatial and velocity continuity. In total, there are 16 key points in this trajectory, consisting of 5 via points, 12 straight-line segments, and 3 circular portions.
The resulting path is this funny ghost:

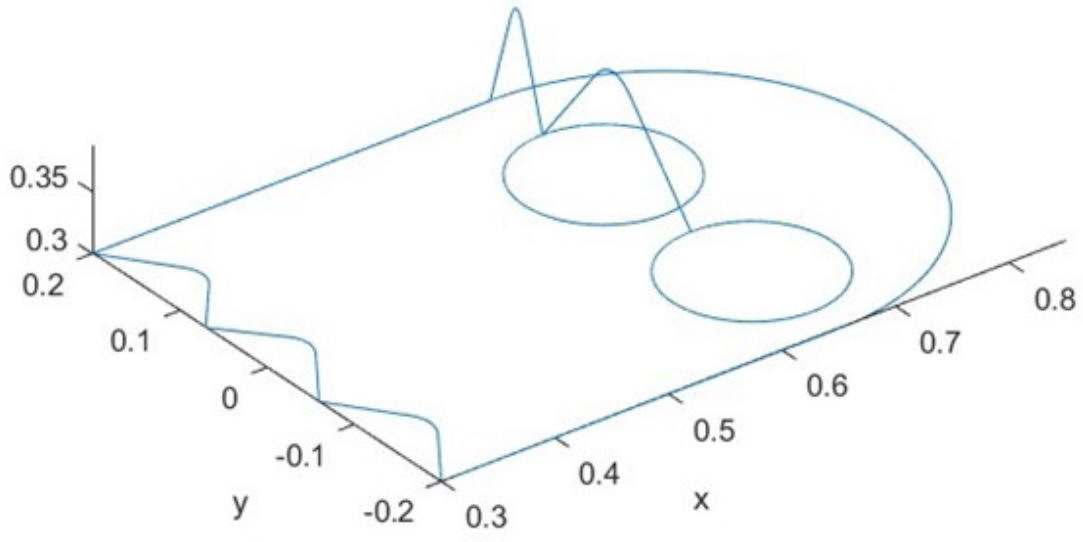


Figure 2.1: Planned path

To prevent singularities and ensure the smooth operation of the SCARA robot, we have carefully confined the entire path within an area located in front of the SCARA, with a length smaller than the radius of the SCARA arm itself. This strategic placement not only enhances the robot's performance but also minimizes the risk of reaching problematic configurations.

Since the orientation of the SCARA can be traced back to the orientation of a two-planar link manipulator, it is sufficient to define the trajectory of a single orientation variable (2d orientation). For the end-effector's orientation, a trajectory has been chosen that starts at 0, reaches π in 10 seconds, and then finishes at $\pi/2$ in twenty seconds.

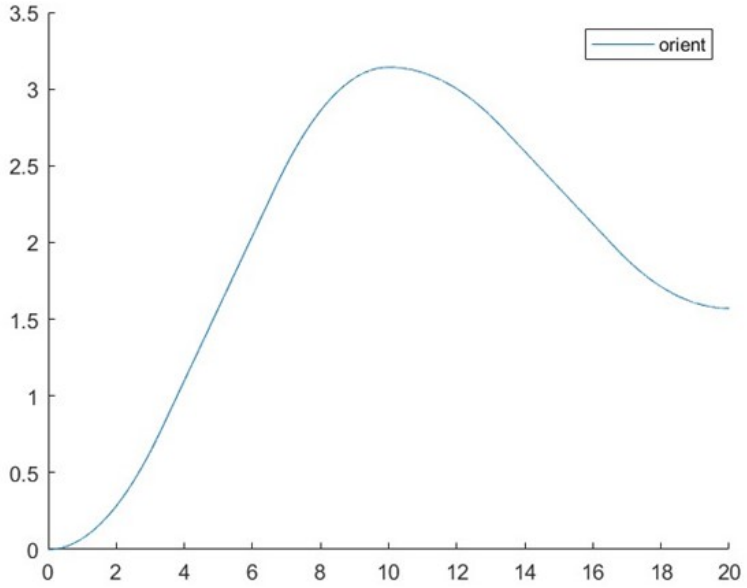
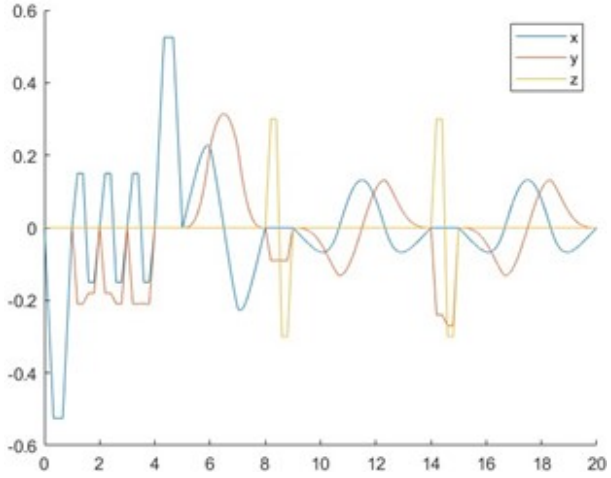


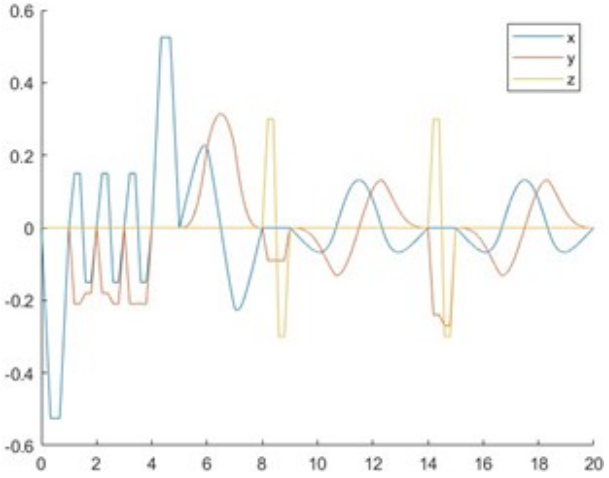
Figure 2.2: Orientation of the end-effector

For the computation of velocity and acceleration, we have employed trapezoidal profiles. These profiles enable us to precisely control the SCARA's motion by ensuring gradual acceleration, constant velocity, and gradual deceleration throughout the trajectory. This approach has a worse performance index compared to the cubic polynomial; the decrease is, however, limited.

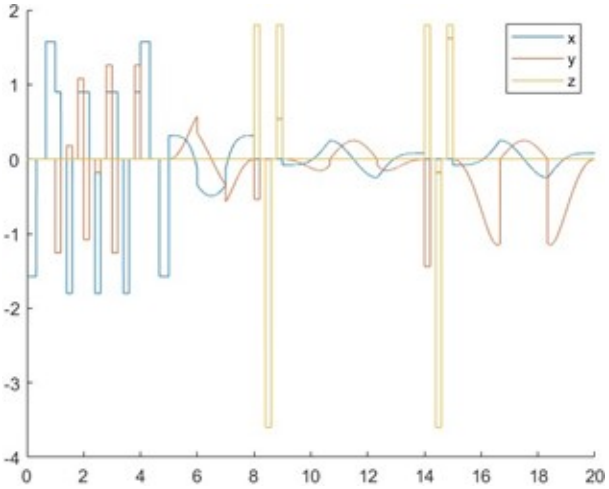
Now, let's show the trajectory we have designed:



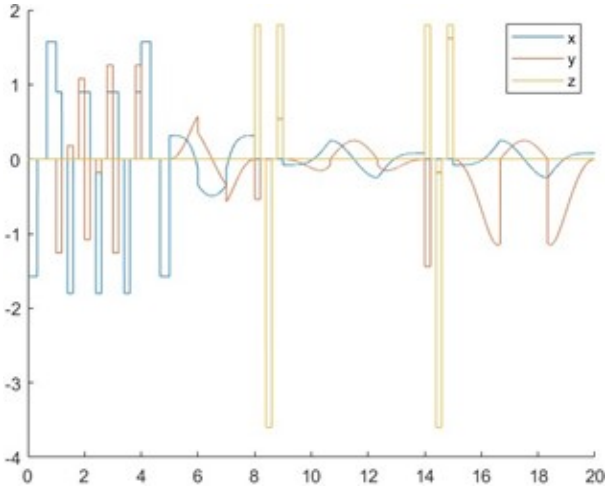
(a) Velocity profile of position



(b) Velocity profile of orientation



(a) Acceleration profile of position



(b) Acceleration profile of orientation

Chapter 3

Kinematics inversion

Implement the CLIK algorithms in MATLAB with both Jacobian inverse and transpose along the trajectory. Adopt Euler numerical integration rule with sampling period of 1 ms.

The trajectory we've chosen is defined in the operational space, specifically focusing on the position of the end-effector. This approach allows for a more intuitive control of the robot's movements, but it necessitates the use of algorithms for inverse kinematics to translate these operational space references into joint space coordinates, including positions, velocities, and even accelerations. To address this, we've implemented several inverse kinematic algorithms to handle this mapping efficiently. These include:

- CLIK with Inverse Kinematics: This algorithm computes the joint angles based on the desired end-effector positions, ensuring precise control over the robot's movements.
- CLIK with Transpose Jacobian: By utilizing the transpose of the Jacobian matrix, this method provides an alternative way to calculate the joint velocities, results in a reduction in computational complexity but results in a loss of performance.
- CLIK with Pseudo-Inverse Jacobian: This technique uses the pseudo-inverse of the Jacobian matrix to calculate joint velocities, offering a robust solution for redundancy resolution and enhanced flexibility in complex tasks.
- Second-Order CLIK: Building on the CLIK framework, this advanced algorithm not only computes joint velocities but also considers joint accelerations, allowing for even more refined control of the robot's motion, especially in dynamic environments.

3.1 Kinematic inversion with Jacobian inverse

This algorithm effectively employs the inverse matrix of the analytical Jacobian to linearize the error dynamics. Thanks to the specific design and mechanics of the SCARA manipulator, where it typically features a planar configuration with two revolute joints, the analytical Jacobian aligns perfectly with the geometric Jacobian. Employing this algorithm, you can calculate the joint velocities in the following straightforward manner:

$$\dot{q} = J_A^{-1}(q)\left(\dot{x}_d + Ke\right)$$

where the matrix J is square and non-singular. The choice made for the velocity calculation, along with the formula for calculating the derivative of the direct kinematics, leads to:

$$\dot{e} + Ke = 0$$

where e represents the error in operational space between the desired and computed pose, $\dot{e}$ is the derivative of the error, and K is a positive definite matrix, typically diagonal, employed to guarantee the convergence of the error towards 0. In our specific case

$$K = \begin{bmatrix} 500 & 0 & 0 & 0 \\ 0 & 500 & 0 & 0 \\ 0 & 0 & 700 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

Following Simulink scheme was used for inverse kinematics.

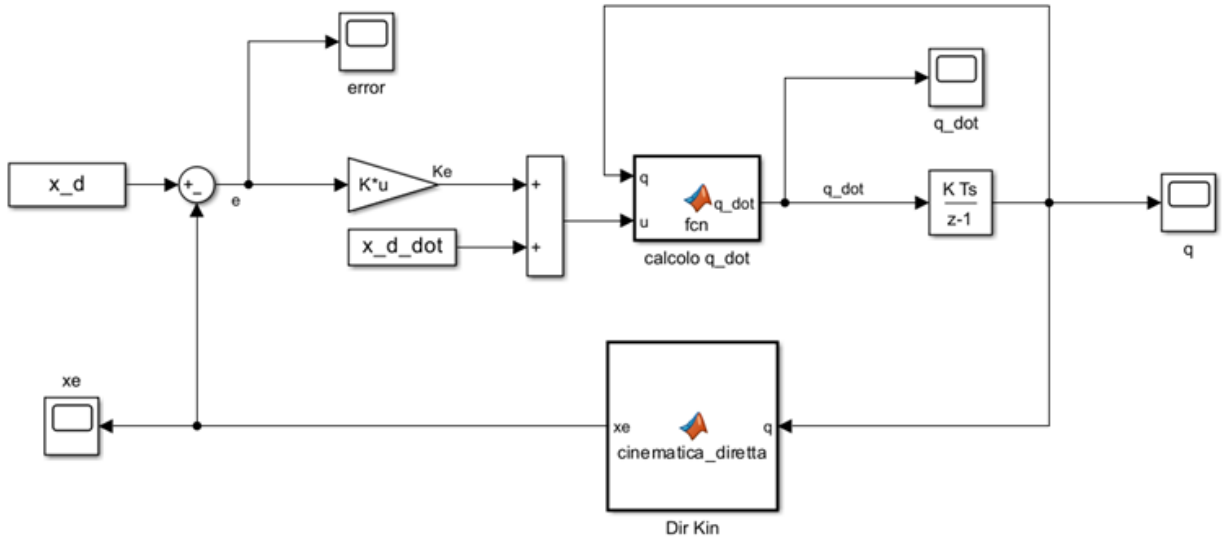


Figure 3.1: CLIK scheme with jacobian inverse

3.2 Kinematic inversion with Jacobian transpose

The algorithm utilizing the Jacobian transpose offers the advantage of reducing computational load on the computer, though it comes with a trade-off of slightly increased error between the desired and computed poses when compared to the outcomes achieved using the Jacobian inverse. This trade-off is crucial to consider when deciding the most suitable approach for a particular robotic task. The Simulink scheme implemented is structured as follows:

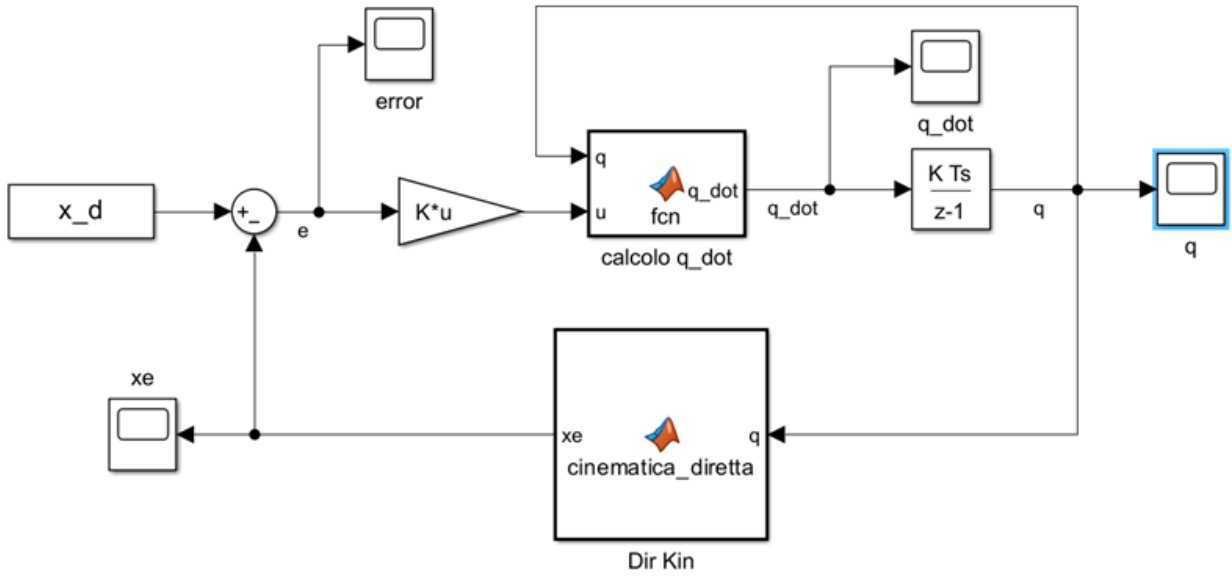
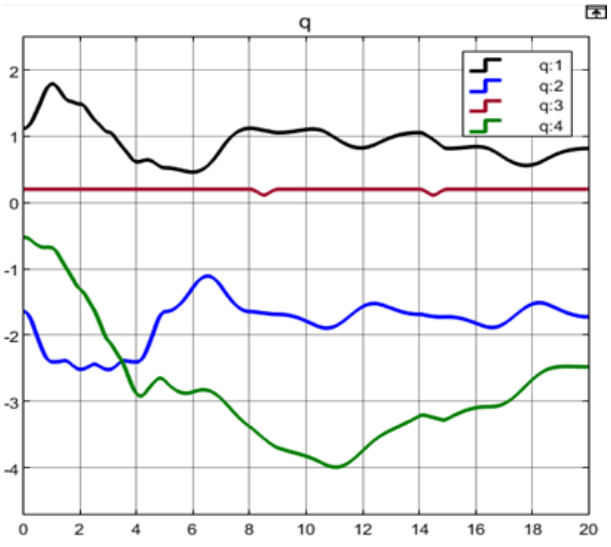


Figure 3.2: CLIK scheme with jacobian transpose

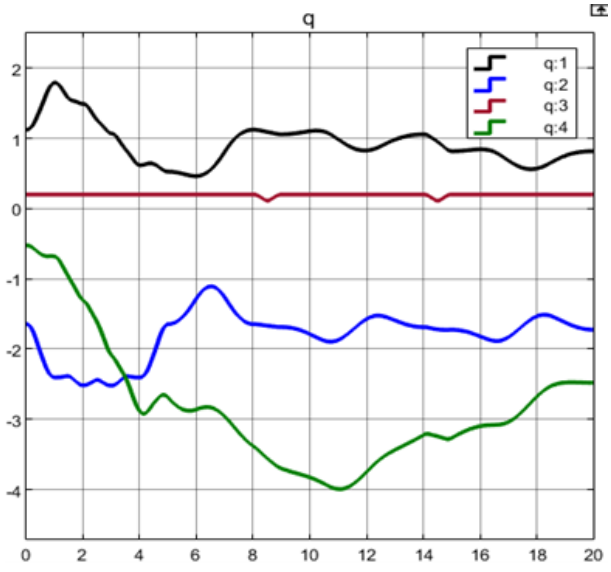
To achieve desirable results and partly comparable ones to those obtained with the inverse Jacobian, higher values of K have been chosen.

$$K_{tr} = \begin{bmatrix} 2000 & 0 & 0 & 0 \\ 0 & 1300 & 0 & 0 \\ 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 350 \end{bmatrix}$$

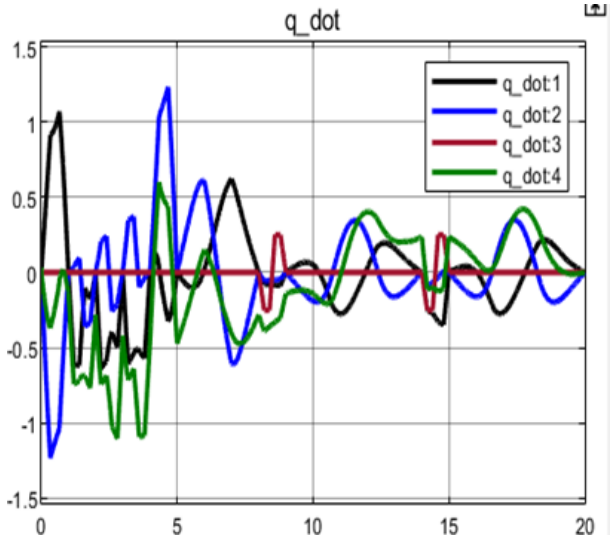
Now, let's compare the results obtained using the two algorithms:



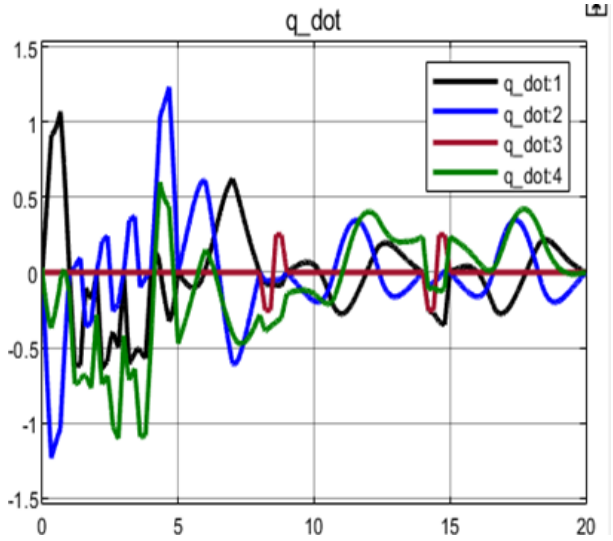
(a) q values from CLIK with jacobian inverse



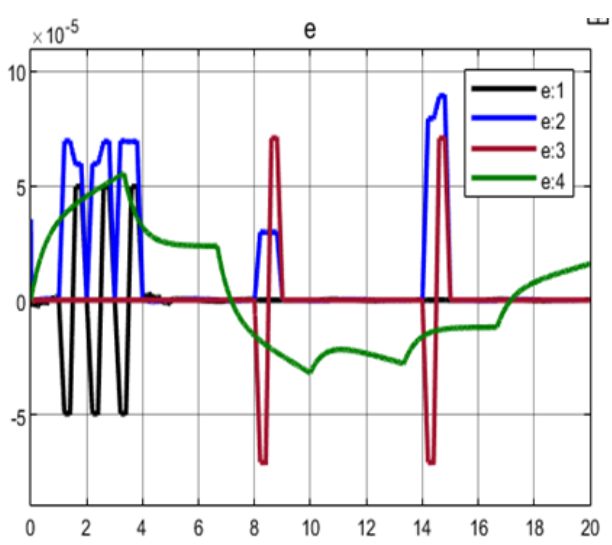
(b) q values from CLIK with jacobian transpose



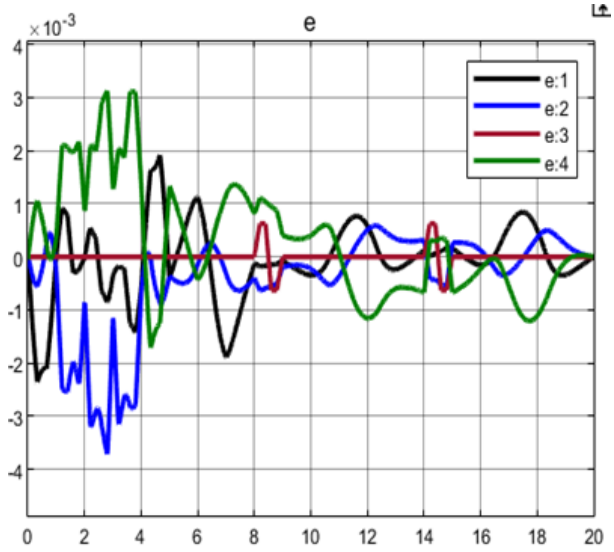
(a) $\dot{q}$ values from CLIK with jacobian inverse



(b) $\dot{q}$ values from CLIK with jacobian transpose



(a) error values from CLIK with jacobian inverse



(b) error values from CLIK with jacobian transpose

While at first glance, the results may seem practically equivalent, a more detailed analysis of the error reveals the limitations of the transpose-based algorithm. In fact, there is a difference in error of two orders of magnitude (10^{-5} for inverse against 10^{-3} for transpose).

3.3 Kinematic inversion with Jacobian Pseudo-inverse and dexterity constraint

Assuming to relax an operational space component, implement the CLIK algorithm in MATLAB with Jacobian pseudo-inverse along the trajectory when optimizing a dexterity constraint.

To introduce redundancy in the manipulator, it is essential to selectively relax one component within the operational space. For our project, we have opted to relax the x-component. As a result, we obtain the following Jacobian matrix, which is presented as a 3×4 matrix with a distinctive low rectangular shape:

$$J = \begin{bmatrix} \frac{cos(\theta_1+\theta_2)}{2} + \frac{cos(\theta_1)}{2} & \frac{cos(\theta_1+\theta_2)}{2} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix}$$

When implementing inverse kinematics for a redundant manipulator, we employ the following formula as a fundamental tool in finding optimal joint configurations:

$$\dot{q} = J_A^{\dagger}(q)(\dot{x}_d + Ke) + \left(I - J_A^{\dagger} J_A\right)\dot{q}_0$$

In the equation provided, the presence of the $\dot{q}_0$ term is significant as it undergoes a projection into the null space of the Jacobian matrix. This projection ensures that $\dot{q}_0$ does not disrupt the trajectory that has been pre-planned for the manipulator. The $\dot{q}_0$ term offers us a valuable opportunity to harness the inherent redundancy of the manipulator's degrees of freedom (DOF) to accomplish a variety of tasks. To do this, we choose the joint $\dot{q}_0$ appropriately. We consider the measure of manipulability:

$$w(q) = \sqrt{\det\left(J_A J_A^T\right)}$$

It enables us to navigate away from singular configurations. As a result, our approach involves computing 'q' by taking the gradient of the manipulability function $w(q)$

$$\dot{q}_0 = k_0 \frac{\partial w(q)}{\partial q}$$

Where $k_0 = 10$ because we want to maximize this function.

The Simulink scheme implemented is as follows:

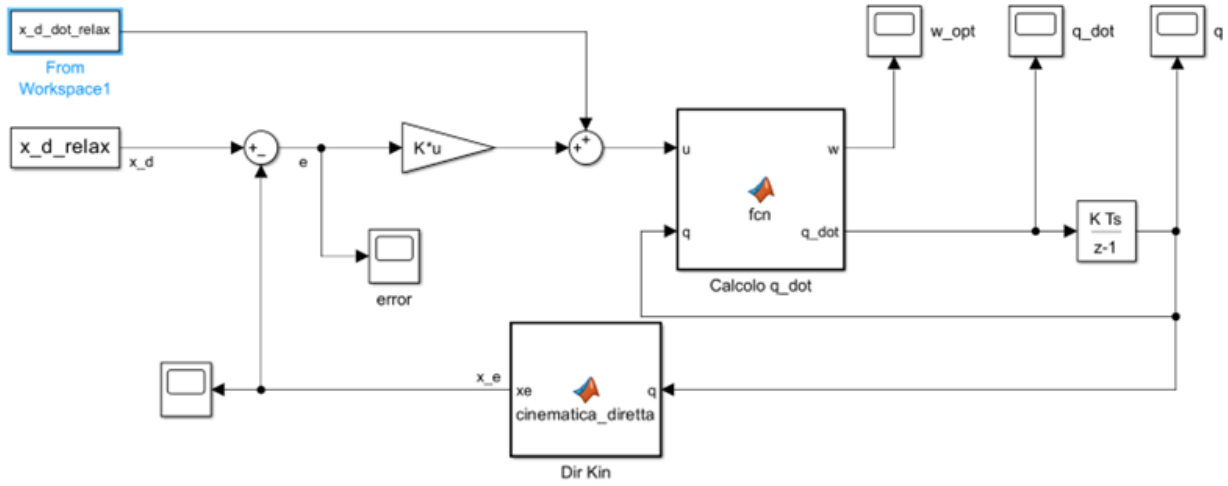
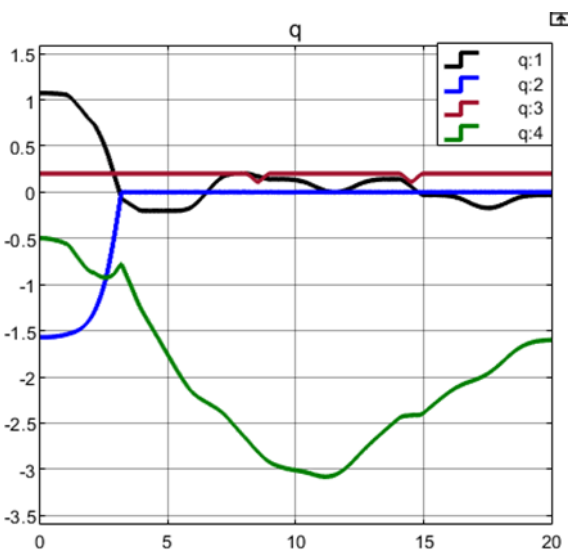


Figure 3.6: CLIK scheme with jacobian pseudo-inverse

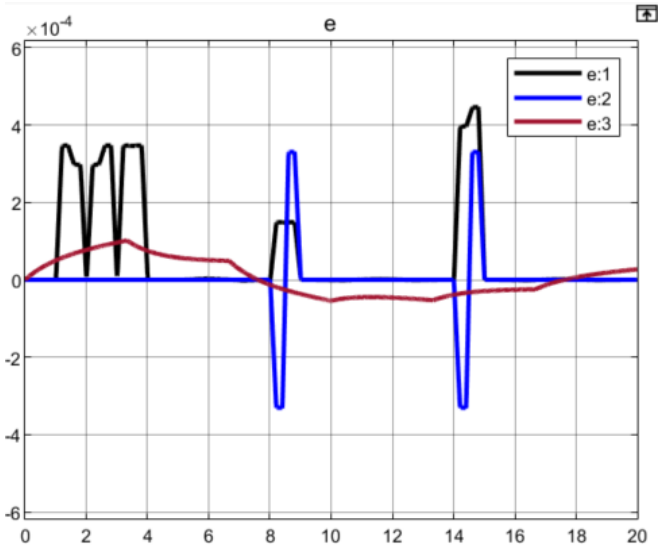
The gain matrix K is equal to:

$$K = \begin{bmatrix} 1000 & 0 & 0 \\ 0 & 1500 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

Let's see the results of the simulation:

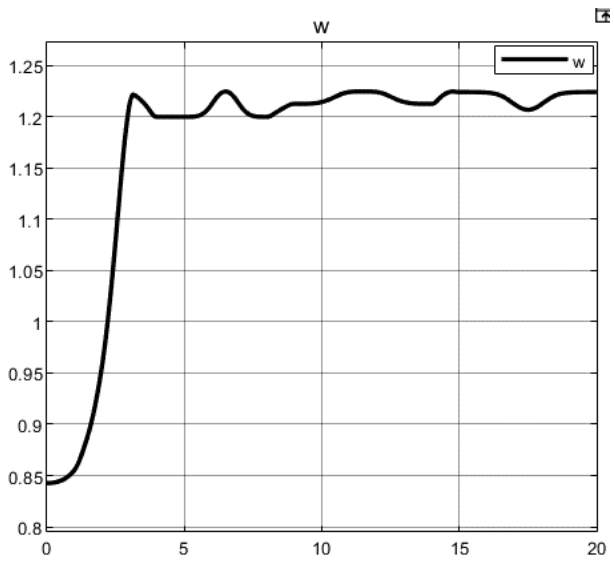


(a) q values from CLIK with pseudo-invese

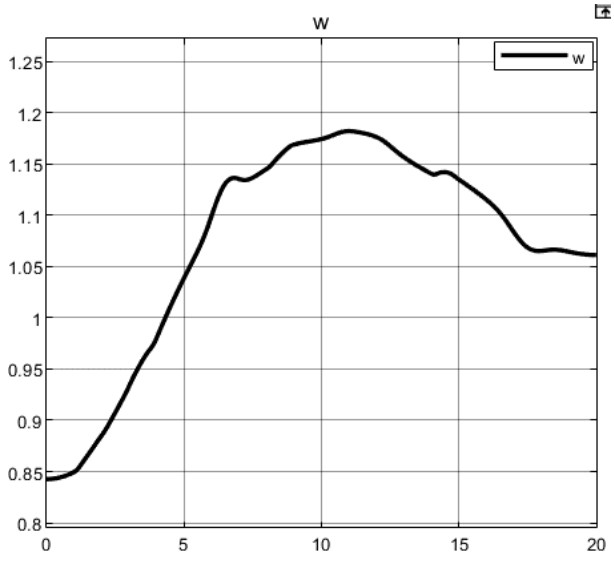


(b) error values from CLIK with pseudo-invese

Let's compare the value of w(q) when it is inserted into the objective function versus when it is not.



(a) w values with the object function



(b) w values without the object function

3.4 Closed Loop Inverse Kinematic of second order

Let's introduce the Second-Order Click, which will be fundamental for the discussion of dynamics. It is a powerful algorithm because it not only considers the path and velocity but also the desired acceleration, thus allowing the calculation of joint accelerations. To design our control schemes, we need to compute the vector of joint acceleration. To do that we can implement the CLIK algorithm of second order where:

$$\ddot{q} = JA - 1(q)(x\dot{d} + K_D\dot{x} + K_P\tilde{x} - JA(q,\dot{q})\dot{q}))$$

Choosing K_D and K_P as positive definite matrices and diagonals we obtain a linear and asymptotically stable system. Where we have chosen

$$K_P = \begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 \\ 0 & 0 & 2000 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad K_D = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The block scheme is the following:

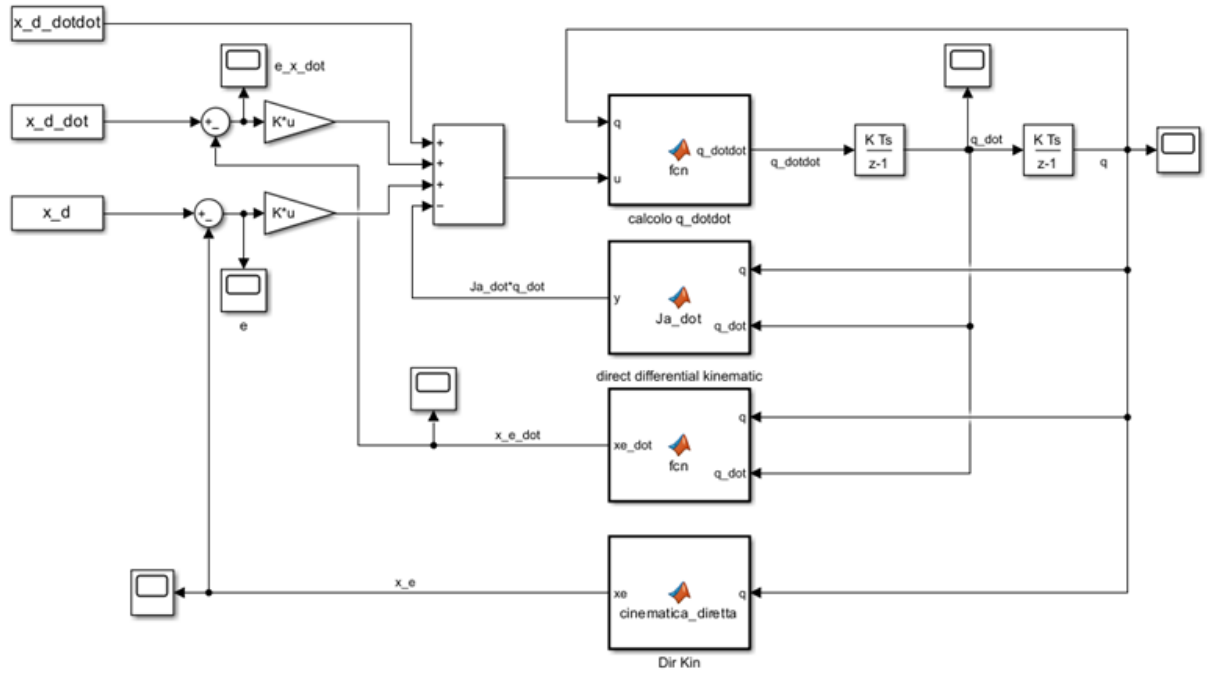


Figure 3.9: Second-Order CLIK scheme

Chapter 4

Dynamic model and control

We undertake an analysis of our robot manipulator, focusing on the development of a control strategy through the study of its dynamic equation. To derive the dynamic equation, we employ the Lagrange formulation, enabling us to express it analytically in a concise form. This analytical representation is convenient in computing various contributions that will prove useful in designing our control strategy. Assuming the absence of static friction, the model for our manipulator in joint space takes the following form:

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

$$B(q) = \sum_{i=1}^{n}\left(m_{l_i}{J_P^{(l_i)}}^T J_P^{(l_i)} + {J_o^{(l_i)}}^T R_i I_{l_i^i} R_i^T J_o^{(l_i)} + m_{m_i}{J_P^{(m_i)}}^T J_P^{(m_i)} + {J_o^{(m_i)}}^T R_{m_i} I_{m_i}^{m_i} R_{m_i}^T J_o^{(m_i)}\right)$$

This expression yields to:

$$B(q) = \begin{bmatrix} 10\cos\left(\theta_2\right) + 10763/500 & 5\cos\left(\theta_2\right) + 4383/500 & -1/4 & -51/50 \\ 5\cos\left(\theta_2\right) + 4383/500 & 4383/500 & -1/4 & -51/50 \\ -1/4 & -1/4 & 45/2 & 0 \\ -51/50 & -51/50 & 0 & 7/5 \end{bmatrix}$$

To compute the matrix $C(q,\dot{q})\dot{q}$ we will use the expression of Christoffel symbols of the first type [1].

$$\sum_{k=1}^{n} c_{ijk}q_k = c_{ij}$$

$$c_{ijk} = \frac{1}{2}\left(\frac{\partial b_{ij}}{\partial q_k} + \frac{\partial b_{ik}}{\partial q_j} + \frac{\partial b_{jk}}{\partial q_i}\right)$$

These yields:

$$C(q,\dot{q}) = \begin{bmatrix} -5\dot{\theta}_2\sin\left(\theta_2\right) & -5\dot{\theta}_1\sin\left(\theta_2\right) - 5 * \dot{\theta}_2\sin\left(\theta_2\right) & 0 & 0 \\ 5\theta_1\sin\left(\theta_2\right) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

In the end we have the viscous friction torques as a diagonal matrix F_v from the given parameters and computed the gravity term $g\left(q\right)$ obtaining the joint space dynamic model.

Consider a concentrated end-effector payload of about 5 kg. Simulate in MATLAB the motion of the controlled manipulator under the assumption that the desired joint trajectories for the first two controllers are generated with a 2nd-order CLIK algorithm. Implement discrete-time controllers with a sampling period of 1 ms.

4.1 Robust Control

This control strategy aimed at ensuring robustness in case of uncertainties within the dynamic model. Specifically, we will focus on assessing the robustness by comparing the nominal matrix to an estimated inertia matrix, denoted as

$$\widehat{B} = B - \tilde{B}$$

where $\tilde{B}$ represents the uncertainty. Similarly, we extend this consideration to the term $n(q, \dot{q})$ expressed as

$$\hat{n} = n - \tilde{n}$$

Recalling the dynamic model's expression:

$$B(q)\ddot{q} + n(q, \dot{q}) = u$$

we observe linearity concerning the control input. Leveraging the direct Lyapunov approach, we can regulate our manipulator using a control input comprising three distinct contributions:

- $\boldsymbol{u} = \widehat{\boldsymbol{B}}\boldsymbol{y} + \widehat{\boldsymbol{n}}$, approximate compensation of nonlinear effects and joint decoupling
- $\ddot{\boldsymbol{q}} + K_D\tilde{\boldsymbol{q}} + K_p\tilde{\boldsymbol{q}}$, encompassing a nonlinear term $\ddot{\boldsymbol{q}}$, a linear PD controller with gain matrices K_D and K_p acting on the error $\tilde{\boldsymbol{q}}$.
- $w = \frac{\rho}{||z||}z$, where ρ is a unit vector control that ensures convergence of the vector $\xi = [\tilde{q}\dot{\tilde{q}}]$ to the sliding space. The resulting control law is of the unit vector type, a vector of magnitude ρ aligned with the unit vector of $z = D^T Q\xi$.

Additionally, a boundary layer is introduced to mitigate chattering:

$$\mathbf{z} = \begin{cases} \frac{\rho}{\|\mathbf{z}\|}\mathbf{z} & \text{if } \|\mathbf{z}\| \geq \epsilon \\ \frac{\rho}{\epsilon}\mathbf{z} & \text{if } \|\mathbf{z}\| < \epsilon \end{cases}$$

with $\boldsymbol{\rho} = 10$ and $\boldsymbol{\epsilon} = 0.2$. The introduction of the threshold ϵ is essential because once the sliding space is reached, the resulting control law would oscillate with infinite frequency, leading to the onset of the chattering phenomenon. The other parameters used are:

$$K_P = \begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 \\ 0 & 0 & 2000 & 0 \\ 0 & 0 & 0 & 1000 \end{bmatrix} \quad K_D = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

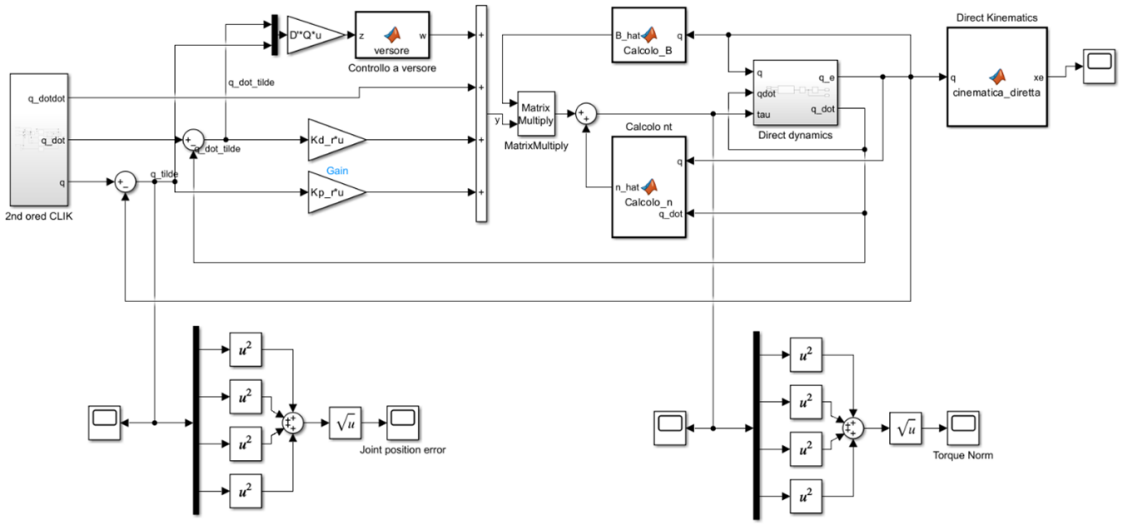
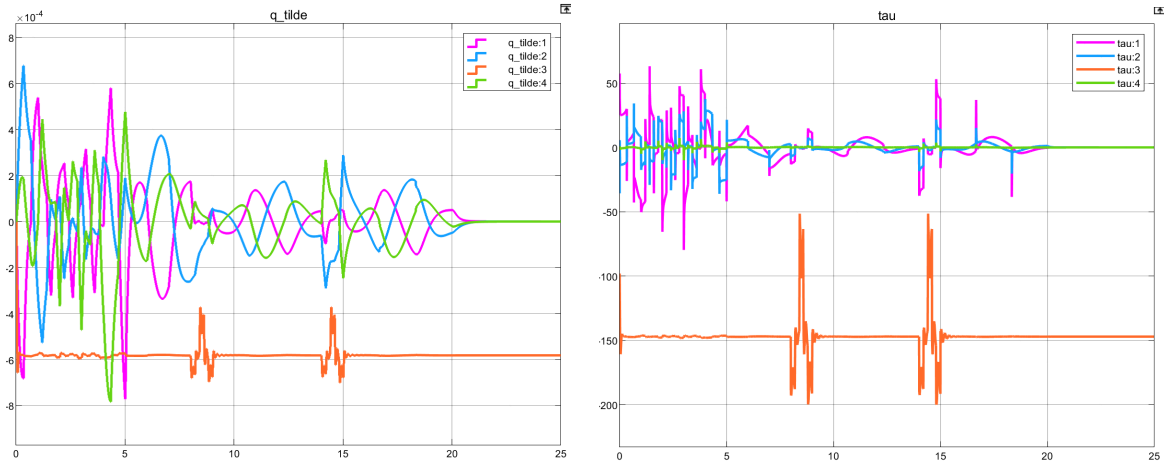


Figure 4.1: Robust control scheme



(a) q_error values of Robust control

(b) tau values of Robust control

Figure 4.2: Robust control result

We can see that it is not possible to fully compensate for the presence of the payload, so there is a steady-state error on the z component. The errors can be kept small with an appropriate choice of gains.

4.2 Adaptive Control

In this section, we explore the design of a controller capable of adapting the control input in real-time. Consider the dynamic model:

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

This model appears linear with respect to the vector of dynamic parameters π. Consequently, we rewrite the expression as:

$$\boldsymbol{Y}(\boldsymbol{q},\dot{\boldsymbol{q}},\ddot{\boldsymbol{q}})\boldsymbol{\pi} = \boldsymbol{u}$$

We then consider the following control action:

$$u = B(q)\ddot{q}_r + C(q,\dot{q})\dot{q}_r + F_V\dot{q}_r + g(q) + K_D\sigma$$

with

$$\ddot{q}_r = \ddot{q}_d + \Lambda\tilde{\dot{q}}$$
$$\dot{q}_r = \dot{q}_d + \Lambda\tilde{\mathbf{q}}$$
$$\sigma = \dot{q}_r - \dot{\boldsymbol{q}} = \tilde{\dot{q}} + \Lambda\tilde{\mathbf{q}}$$

where K_D and Λ are positive definite matrices facilitating the expression of nonlinear decoupling compensation terms. Additionally, $K_D\sigma$ is equivalent to a proportional derivative (PD) control action. Assuming knowledge of the dynamic model but uncertainty regarding the vector of dynamic parameters, we consider the parameter estimation as the difference between the nominal vector and the uncertainty:

$$\widehat{\boldsymbol{\pi}} = \pi - \tilde{\boldsymbol{\pi}}$$

Consequently, the control input becomes:

$$u = Y\left(q,\dot{q},\dot{q}_r,\ddot{\boldsymbol{q}}_r\right)\widehat{\pi} + K_D\sigma = Y\left(q,\dot{q},\dot{q}_r,\ddot{\boldsymbol{q}}_r\right)\widehat{\pi} + K_D(\tilde{\dot{q}} + \Lambda\tilde{\mathbf{q}})$$

The gradient law to update the vector of dynamic parameters is expressed as:

$$\hat{\boldsymbol{\pi}} = \mathbf{K}_{\pi}^{-1}\bar{Y}\left(\boldsymbol{q},\dot{\boldsymbol{q}}_{\boldsymbol{q}},\dot{\boldsymbol{q}}_r,\ddot{\boldsymbol{q}}_r\right)^T\boldsymbol{\sigma}$$

Here, K_π is a positive definite matrix.

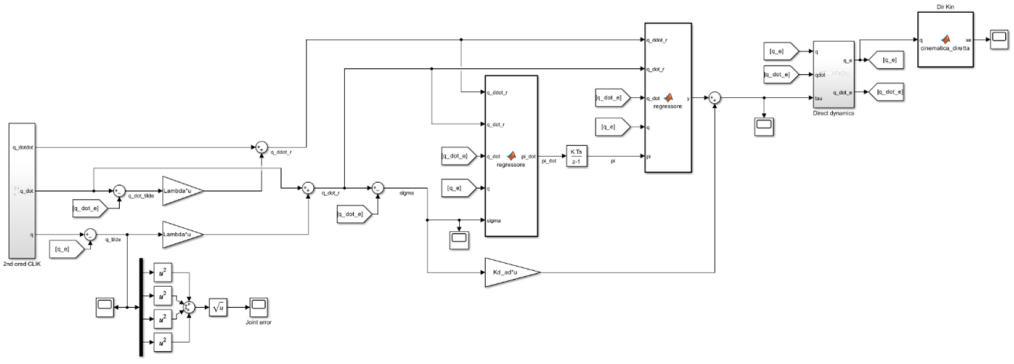


Figure 4.3: Adaptive control scheme

For our robot manipulator, we have chosen:

$$K_\pi = 10 * \text{diag}([1, 1, 0.001, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])$$

$$K_D = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \Lambda = \begin{bmatrix} 500 & 0 & 0 & 0 \\ 0 & 500 & 0 & 0 \\ 0 & 0 & 10000 & 0 \\ 0 & 0 & 0 & 500 \end{bmatrix}$$

since the third parameter in K_π represent the most uncertain parameter (the mass of link 3). Below is the plot of the mass variable of link 3, it converges in about 5 seconds thus going to compensate for the addition of the mass at the tip of the end-effector.

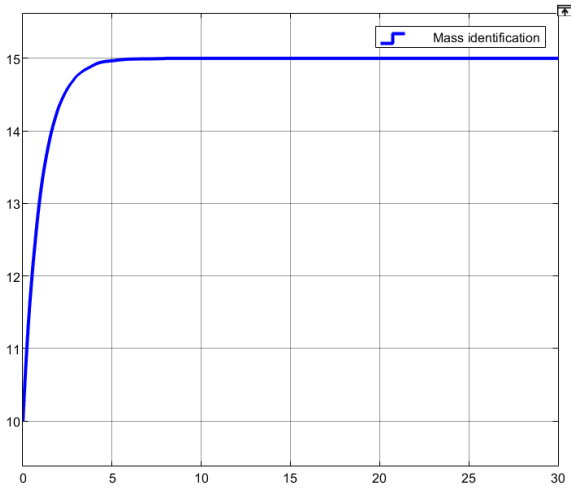
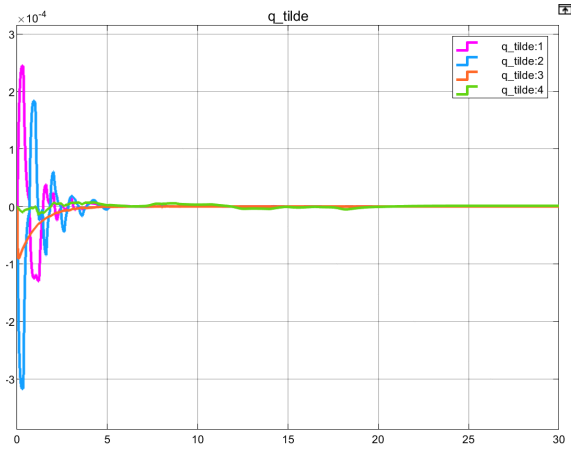
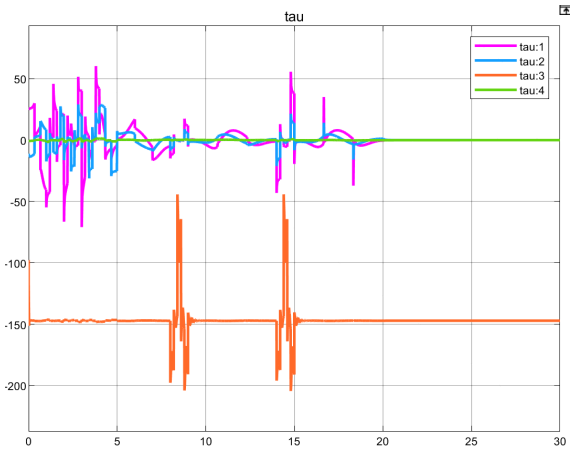


Figure 4.4: Estimated mass of link$_3$



(a) q_error values of Adaptive control



(b) tau values of Adaptive control

Figure 4.5: Adaptive control result

We can see that the payload is fully compensate, so the steady-state error on the z component converges to zero. The error, after five second, become of the order of 10^{-5} with an appropriate choice of gains.

4.3 Operational space inverse dynamics control with the adoption of an integral action to recover the steady-state error due to the uncompensated load

In this concluding section, we explore the design of a controller in operational space for our robot manipulator. While previous methods have focused on joint space control, it is important to note that an algorithm for inverse kinematics is required to obtain the reference in joint space. However, this approach can be computationally demanding, especially when considering algorithms like the second order CLIK algorithm. As an alternative, we can compute a control input directly in operational space. Recalling the dynamic model

$$B(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = u$$

where u is the control input, we can introduce a linearizing control input:

$$B(q)\mathbf{y} + \boldsymbol{n}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = u$$

leads to the system of double integrators:

$$\ddot{q} = y$$

Furthermore, leveraging the second-order relationship of differential kinematics

$$\ddot{x}_e = J_A(q)\ddot{q} + J_A(q, \dot{q})\dot{q}$$

where x_e is the end-effector position, we can choose y as:

$$y = J_A^{-1}(q)\left(\ddot{x}_d + K_D\dot{\tilde{x}} + K_P\tilde{x} + K_I\int_0^t \tilde{\mathbf{x}}(\tau)\,d\tau - \dot{J}_A(q, \dot{q})\dot{q}\right)$$

Here, K_P, K_I and K_D are positive definite matrices representing the gains of a PID controller. The chosen control input in operational space provides a means to directly influence the end-effector behavior.

For our robot manipulator, we have chosen:

$$K_P = \begin{bmatrix} 5000 & 0 & 0 & 0 \\ 0 & 5000 & 0 & 0 \\ 0 & 0 & 10000 & 0 \\ 0 & 0 & 0 & 5000 \end{bmatrix} \quad K_I = \begin{bmatrix} 20 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 2000 & 0 \\ 0 & 0 & 0 & 20 \end{bmatrix} \quad K_D = \begin{bmatrix} 50 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 50 \end{bmatrix}$$

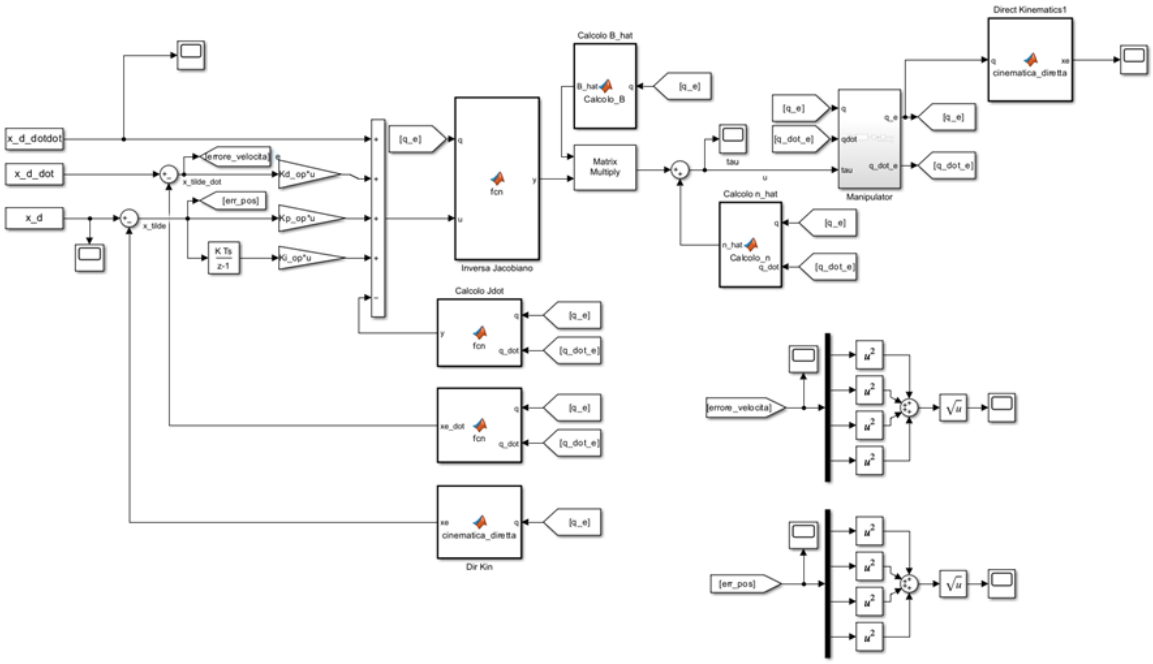
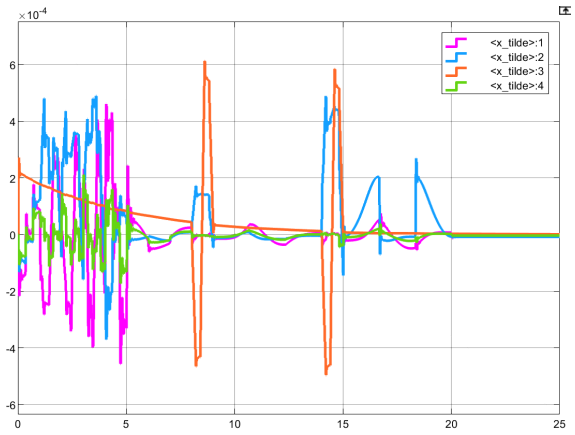
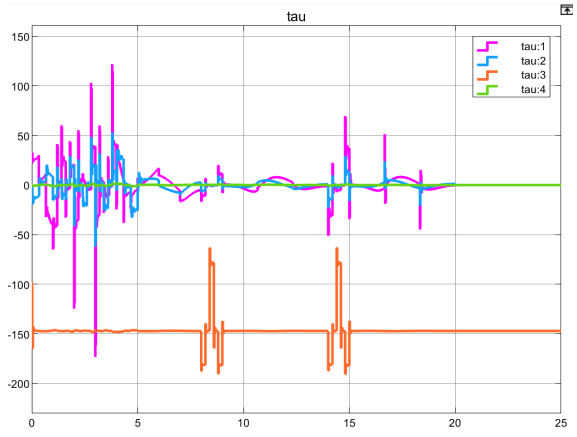


Figure 4.6: Operational space control scheme



(a) q_error values of Operational space inverse dynamic control



(b) tau values of Operational space inverse dynamic control

Figure 4.7: Operational space inverse dynamic control result

Bibliography

[1] *Robotics: Modelling, Planning and Control*. 2000.