UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

DIETI UNINA

Master's Course in Automation and Robotics Engineering

Modelling, Planning and Control of a SCARA Manipulator

Foundations of Robotics Technical Project

Professor: Bruno Siciliano

Student: Salvatore Granata

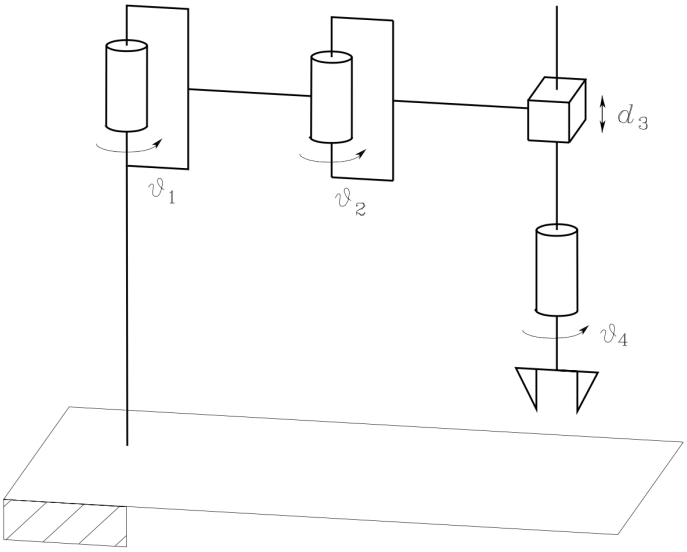
ID Number: P38000219

Academic Year: 2022/2023

Contents

Project Topic

Consider the SCARA manipulator in the figure:



with the following data:

$$d_0 = 1\,\text{m} \quad a_1 = a_2 = 0.5\,\text{m} \quad \ell_1 = \ell_2 = 0.25\,\text{m}$$

$$m_{\ell_1} = m_{\ell_2} = 20\,\text{kg} \quad m_{\ell_3} = 10\,\text{kg}$$

$$I_{\ell_1} = I_{\ell_2} = 4\,\text{kg}\cdot\text{m}^2 \quad I_{\ell_4} = 1\,\text{kg}\cdot\text{m}^2$$

$$k_{r1} = k_{r2} = 1 \quad k_{r3} = 50\,\text{rad/m} \quad k_{r4} = 20$$

$$I_{m_1} = I_{m_2} = 0.01\,\text{kg}\cdot\text{m}^2 \quad I_{m_3} = 0.005\,\text{kg}\cdot\text{m}^2 \quad I_{m_4} = 0.001\,\text{kg}\cdot\text{m}^2$$

$$F_{m1} = F_{m2} = 0.00005\,\text{N}\cdot\text{m}\cdot\text{s/rad} \quad F_{m3} = 0.01\,\text{N}\cdot\text{m}\cdot\text{s/rad} \quad F_{m4} = 0.005\,\text{N}\cdot\text{m}\cdot\text{s/rad}$$

- 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.

- 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 also the passage for at least 4 via points.

- 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.

- 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.

- Consider a concentrated end-effector payload of about 5 kg. Then, design: 1) a robust control; 2) an adaptive control; 3) an operational space inverse dynamics control with the adoption of an integral action to recover the steady-state error due to the uncompensated load. 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.

Summary

In the following work, a four-axis SCARA manipulator was considered, and its manipulability in terms of velocity and force was analyzed using the respective ellipsoids. Subsequently, given a desired trajectory in the operational space, various inverse kinematics algorithms were employed to achieve it in the joint space.

Finally, with the dynamic model defined to represent the manipulator, three algorithms were used to compute the joint torques in order to execute the desired trajectory. These algorithms include adaptive control, robust control, and control with operational space inverse and integral action.

The simulations were conducted using the MATLAB and Simulink development environment, with the assistance of the ”Robotics Toolbox for Matlab” developed by Peter Corke.

Chapter 1

Robot Kinematics

1.1 Introduction to Robot Kinematics

The kinematic analysis of a robot's mechanical structure involves the description of its motion with respect to a fixed Cartesian frame without considering the forces and moments that cause the motion. Kinematics and differential kinematics are distinguished from each other. In the context of a robot manipulator, kinematics establishes the analytical relationship between the joint positions and the position and orientation of the end-effector, while differential kinematics establishes the analytical relationship between the joint motion and the end-effector motion in terms of velocities, using the manipulator Jacobian.

The formulation of kinematic relationships allows the study of two fundamental problems in robotics: the direct kinematics problem and the inverse kinematics problem. The direct kinematics problem focuses on finding a systematic and general method to describe the motion of the end-effector as a function of the joint motion using linear algebra tools. The inverse kinematics problem deals with finding the solution to the inverse problem, which is crucial for transforming the desired motion of the end-effector, specified in the workspace, into the corresponding joint motion.

The kinematic model of a robot primarily describes the instantaneous motions considering the constraints. On the other hand, the dynamic model takes into account the reaction forces and describes the relationship between the aforementioned motions and the generalized forces acting on the robot.

The robot manipulator on which it's been studied both the kinematics and the dynamics is the SCARA.

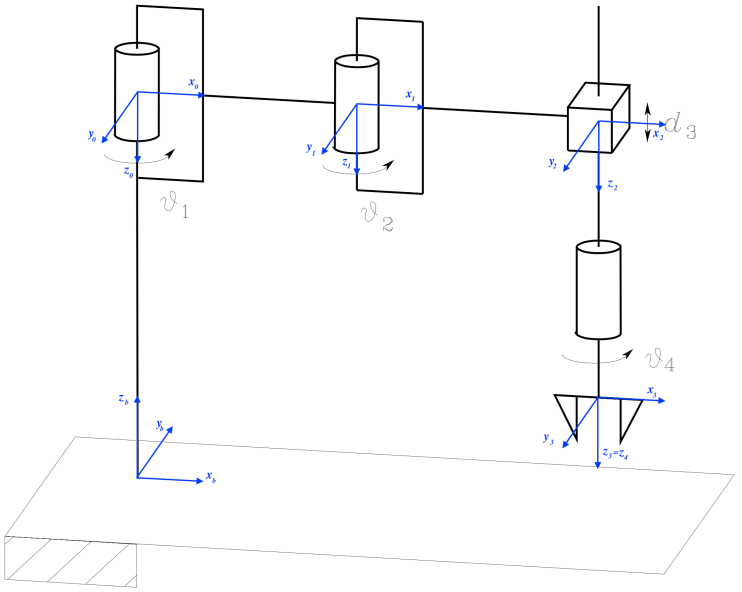
The acronym SCARA stands for *Selective Compliance Assembly Robot Arm* and characterizes the mechanical features of a structure offering high stiffness to vertical loads and compliance to horizontal loads. As such, the SCARA structure is well-suited to vertical assembly tasks. The correspondence between the DOFs and Cartesian space variables is maintained only for the vertical component of a task described in Cartesian coordinates. Wrist positioning accuracy decreases as the distance of the wrist from the first joint axis increases.

A manipulator can be schematized from a mechanical viewpoint as a kinematic chain consisting of rigid bodies (arms) connected through revolute or prismatic joints, which represent the degrees of freedom of the structure. One end of the chain is constrained to a base, while the other end is connected to an end effector. The overall motion of the structure is achieved by composing elementary motions of each arm with respect to the previous one. In order to manipulate an object in space, therefore, it is necessary to describe the position and orientation (pose) of the end effector.

1.2 Frame Positioning with the DH Convention

In order to compute the direct kinematics equation for an open-chain manipulator, a systematic and general method has to be derived to define the relative position and orientation of two consecutive links; the problem is that to determine two frames attached to the two links and compute the coordinate transformations between them. In general, the frames can be arbitrarily chosen as long as they are attached to the link they are referred to. Nevertheless, it is convenient to set some rules also for the definition of the link frames. These rules are expressed by the Denavit-Hartengerg convention, introduced, as the name suggest, by the two scientists Jacques Denavit e Richard S. Hartenberg.

SCARA manipulator is composed by 3 revolute joints and 1 prismatic joint:



Furthermore, the reference frames chosen according to the Denavit-Hartenberg convention are shown in blue in the figure. Therefore, the first step to perform in the study of a manipulator is the positioning of the reference frames connected to each link. Using the aforementioned convention, the following choices have been made:

- The base frame has its origin at point (0,0,0) with the z-axis aligned with the axis of the first joint;
- The zero frame has its z-axis directed downwards along the direction of the base frame, the x-axis chosen arbitrarily and the y axis to complete the left-handed triads;
- The frame 1 has its origin at the second joint, with the z-axis pointing downwards, the x-axis is in the direction from the first to the second joint, and the y-axis completes the left-handed triads;
- The frame 2 has its origin on the third joint with the z-axis facing downwards, the x axis in the direction that goes from the second to the third joint and the y axis completing the left-handed triads;
- The frame 3 is similar to frame 2 but with its origin at the fourth joint;
- The frame 4 has its z-axis coinciding with the z-axis of frame 3;
- The end effector frame has the n-axis coinciding with the y-axis of frame 4, the a-axis coinciding with the z-axis of frame 4, and the s-axis completing the left-handed frame.

With the tripletes positioned in the represented way it is possible to derive the following table of parameters:

Link	a_i	α_i	d_i	θ_i
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2
3	0	0	d_3	0
4	0	0	0	θ_4

Table 1.1: D-H Table

All the variables in D-H Table are already known. Therefore, the manipulator has 4 degrees of freedom (DOFs) to which the following joint variables are associated:

$$q = [q_1, q_2, q_3, q_4] = [\theta_1, \theta_2, d_3, \theta_4]$$

These joint variables are useful for the computation of Jacobian and then the direct kinematics in the section 1.3.

1.3 Direct Kinematics Computation

Since this section purpose is the computation of the direct Kinematics, *Homogeneous Transformations* facilitate the conversion between different coordinate systems in robotics. They allow transformations between the robot's local coordinate frames, world coordinates, and coordinates of external objects or sensors. The computed Homogenous Matrices can be represented as:

$$A_0^b = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & d_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_1^0(q_1) = \begin{bmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2^1(q_2) = \begin{bmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^2(q_3) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4^3(q_4) = \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_e^4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From here, it's possible to link all homogeneous matrices in order to express the rotation of a joint from a particolar frame (in particular the end-effector one in this case), with respect to the base frame:

$$T_e^b = A_0^b A_1^0(q_1) A_2^1(q_2) A_3^2(q_3) A_4^3(q_4) A_e^4$$

$$T_e^b = \begin{bmatrix} cos(\theta_1+\theta_2+\theta_4) & -sin(\theta_1+\theta_2+\theta_4) & 0 & a_2cos(\theta_1+\theta_2)+a_1cos(\theta_1) \\ -sin(\theta_1+\theta_2+\theta_4) & -cos(\theta_1+\theta_2+\theta_4) & 0 & a_2sin(\theta_1+\theta_2)+a_1sin(\theta_1) \\ 0 & 0 & -1 & d_0-d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_e^b indicates thet transformation matrix that solves the problem of direct kinematics, i.e. determining the pose of the end-effector given the joint variables.

The Jacobian matrix is closely related to the direct kinematics of a robot manipulator. Direct kinematics refers to the computation of the position and orientation (pose) of the end-effector based on the joint angles or joint displacements.

In particular, through the knowledge of transformation matrices, it is possible to compute the geometric Jacobian from the core of this relationship (i.e., the matrices that depend on the joint configuration $T_4^0(q)$). This computation involves considering

where: $s_i = sin(\theta_i)$, $c_i = cos(\theta_i)$ and $c_{ij} = cos(\theta_i + \theta_j)$

the generic position vectors p_0, p_1, p_2, p_3 for each frame, computed using the aforementioned rotation matrices, and the vector z of the frames, which, as represented, results in $z_0 = z_1 = z_2 = z_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ Once the position vectors and unit vectors of all the frames are known, it's possible to derive the geometric Jacobian of the whole structure using:

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

So, thanks to transformation and rotation matrices, by substituting the parameters, it's possible to compute the geometric Jacobian directly as:

$$J = \begin{bmatrix} a_2 s_{12} - a_1 s_1 & -a_2 s_{12} & 0 & 0 \\ a_2 c_{12} + a_1 c_1 & a_2 c_{12} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

The Jacobian matrix provides a mathematical representation of the relationship between the joint velocities and the end-effector velocity.

Note: for this manipulator structure, the geometric Jacobian is equal to the analitic one computed as $J_A(q) = \frac{\partial k(q)}{\partial q}$.

In summary, the Jacobian matrix provides a linear mapping between joint variables and end-effector velocities, and by extension, it can be used to relate joint displacements to end-effector displacements, which is a key component of direct kinematics. It enables the determination of the end-effector's position and orientation based on the joint angles or displacements of a robot manipulator.

Given the current joint angles or displacements, multiplying the Jacobian matrix by the vector of joint velocities or displacements yields the corresponding end-effector velocity or displacement. This relationship allows for the direct computation of the end-effector's pose from the joint variables using the Jacobian matrix. The pose of the end-effector will be expressed as:

$$x_e = k(q) = \begin{pmatrix} p_e \\ \Phi \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ \Phi \end{pmatrix} = \begin{pmatrix} a_2 c_{12} + a_1 c_{12} \\ a_2 s_{12} + a_2 s_2 \\ d_0 - d_3 \\ \theta_1 + \theta_2 + \theta_4 \end{pmatrix}$$

It's also worth noting the decoupling between the position and the orientation of the end-effector. This because, in a robot manipulator, there are typically multiple joints or DOFs that contribute to the overall motion of the end-effector. However, due to the mechanical structure and linkages of the robot, these DOFs can be interconnected, resulting in coupling effects. Kinematic decoupling aims to minimize or eliminate these coupling effects and enable independent control of each DOF. By achieving kinematic decoupling, the control of a specific DOF can be isolated from the other DOFs, allowing for more precise and efficient control of the robot's motion. This decoupling can simplify the control algorithms and enhance the overall performance of the robot manipulator.

Chapter 2

Manipulability: Force and Velocity Ellipsoides

The differential kinematics equation $v_e = J(q)\dot{q}$ and the statics equation $\tau = J^T(q)\gamma_e$, together with the duality property (*St. Andrew Cross*), allow the definition of indices for the evaluation of manipulator performance. Such indices can be helpful both for mechanical manipulator design and for determining suitable manipulator postures to execute a given task in the current configuration. First, it is desired to represent the attitude of a manipulator to arbitrarily change end-effector position and orientation. This capability is described in an effective manner by the velocity manipulability ellipsoid.

A global representative measure of manipulation ability can be obtained by considering the volume of the ellipsoid. This volume is proportional to the quantity:

$$w(q) = \sqrt{det(J(q)J^T(q))}$$

As evident from the Jacobian, the joint variables that affect the manipulability measure are θ_1 and θ_2. Therefore, to analyze the manipulability in terms of velocity and force of the manipulator, the chosen configurations to represent the corresponding ellipsoids are obtained by varying only θ_2, as varying θ_1 does not influence the manipulability of the robot but results in a simple change of reference.

The chosen configurations are as follows:

- $q_1 = \begin{bmatrix} 0 & -\pi/2 & 0.5 & 0 \end{bmatrix}$
- $q_2 = \begin{bmatrix} 0 & \pi/4 & 0.5 & 0 \end{bmatrix}$
- $q_3 = \begin{bmatrix} 0 & \pi/30 & 0.5 & 0 \end{bmatrix}$
- $q_4 = \begin{bmatrix} 0 & \frac{98\pi}{100} & 0.5 & 0 \end{bmatrix}$
- $q_5 = \begin{bmatrix} 0 & -\frac{99.99\pi}{100} & 0.5 & 0 \end{bmatrix}$

For each configuration the velocity and force ellipsoids can be computed in MATLAB and represented as:

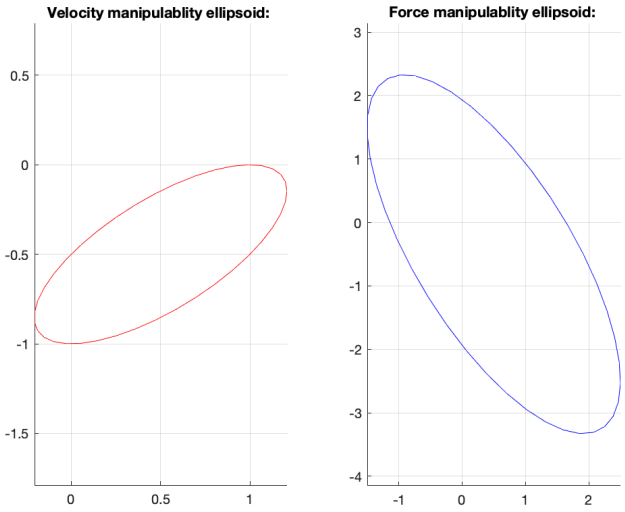


Figure 2.1: Velocity and Force Ellipsoids in q_1 configuration

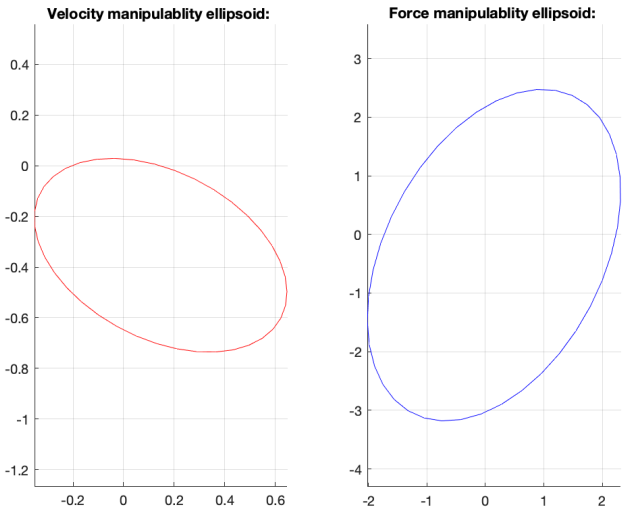


Figure 2.2: Velocity and Force Ellipsoids in q_2 configuration

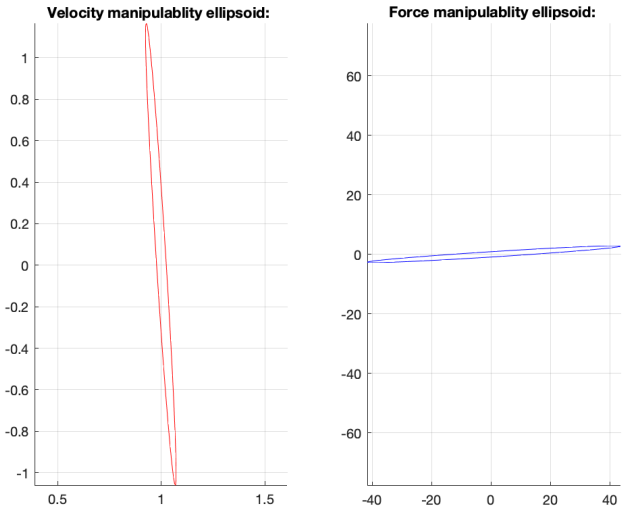


Figure 2.3: Velocity and Force Ellipsoids in q_3 configuration

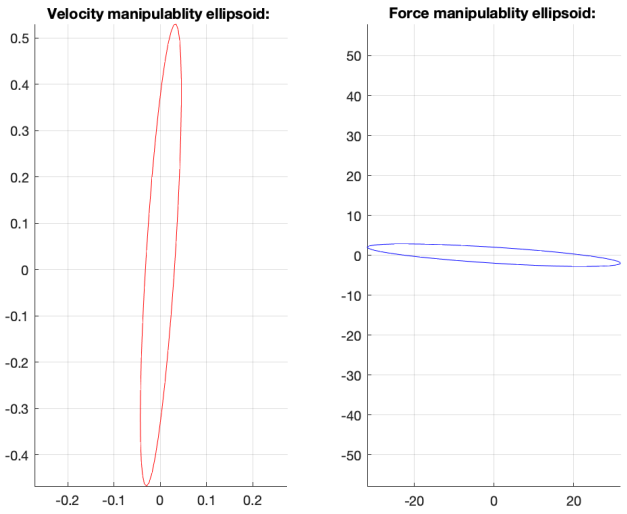


Figure 2.4: Velocity and Force Ellipsoids in q_4 configuration

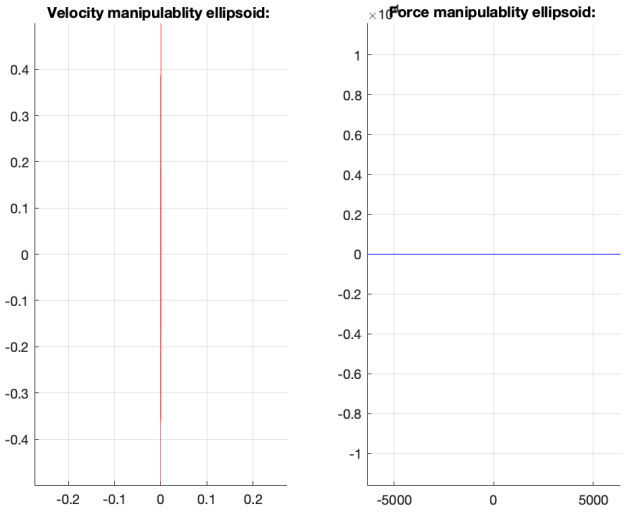


Figure 2.5: Velocity and Force Ellipsoids in q_5 configuration

From the results above it's clear to see that configurations q_1 and q_2 are far from singularity (and give a manipulability value of $w_1 = 0.25$, $w_2 = 0.177$) so the manipulator can perform a task smoothly. The configurations q_3, q_4 and q_5, instead, are close to the singularities ($w = 0.026$), while for q_4 and q_5 it holds $\theta_4 \approx \pi$ and $\theta_5 \approx \pi$ the arm is almost completely extended or folded on itself (in these cases: $w_4 = 0.016$, $w_5 = 0.00$).

In fact if a generic configuration has $q_i = 0$ or $q_i = \pi$, the manipulator would be in singularity therefore the ellipsoids would degenerate into straight lines and the manipulator would not be able to supply the end-effector with arbitrary velocities. As we can see in the case of q_5 in fact we practically have the divergence of the ellipsoids in straight lines because we are practically on the singularity. Indeed, with respect to the case of q_4 we are much closer to the singularity and the difference between the two cases is evident.

Chapter 3

Path Planning

When carrying out a particular task with a robot, it is important to take into account the key aspects of motion planning algorithms. The objective of trajectory planning is to create the reference inputs for the motion control system, ensuring that the manipulator follows the planned trajectories. Typically, the user defines various parameters to describe the desired trajectory. The planning process involves generating a time sequence of values based on an interpolating function, often a polynomial, that represents the desired trajectory.

During the path planning, it's useful to set some via points. In robotics, via points refer to intermediate positions or waypoints that a robot must pass through while executing a trajectory or a path and allow a better control of the robot's motion. They can be strategically placed to ensure the robot follows a specific route, avoids obstacles, or achieves certain configurations or constraints. They are additional positions specified along the desired trajectory to define specific locations or key moments during the robot's motion. In practice, via points can be specified using Cartesian coordinates, joint angles, or other relevant parameters depending on the robot's kinematic model and the requirements of the task. The trajectory planning algorithm then generates a smooth path that passes through these via points while considering the robot's dynamics and constraints. By including via points, the robot can perform complex movements, navigate obstacles, or achieve specific poses or orientations as required by the task at hand.

Consider the case where, to simplify trajectory generation algorithms, it is desired to connect N waypoints $q_1, ..., q_N$ specified at times $t_1, ..., t_N$ using linear functions. To avoid discontinuities in the first derivative at the time instants t_k, the function $q(t)$ must have a parabolic behavior in the vicinity of t_k. Therefore, the overall trajectory consists of a sequence of first and second-degree polynomials, allowing for discontinuity in the second derivative of $q(t)$.

3.1 Path Points Selection

A series of points were chosen to create a trajectory in the X-Y-Z plane, essentially forming a path with one side containing a semicircle to fulfill the requirement of a circular segment. Thus, the chosen trajectory consists of twelve points, four of which are waypoints. The positions for the end-effector to reach were specified in the operational space coordinates. The trajectory is formed by the following points:

$$\mathbf{p} = [\, p_0 \quad p_1 \quad p_2 \quad p_3 \quad p_4 \quad p_5 \quad p_6 \quad p_7 \quad p_8 \quad p_9 \quad p_{10} \quad p_{11}]$$

where:

$$\begin{aligned}
\boldsymbol{p}_0 &= \begin{bmatrix} 0.18 & 0.2 & 0.5 \end{bmatrix} \\
\boldsymbol{p}_1 &= \begin{bmatrix} 0.14 & 0.2 & 0.5 \end{bmatrix} \\
\boldsymbol{p}_2 &= \begin{bmatrix} 0.1 & 0.2 & 0.2 \end{bmatrix} \\
\boldsymbol{p}_3 &= \begin{bmatrix} 0.05 & 0.15 & 0.5 \end{bmatrix} \\
\boldsymbol{p}_4 &= \begin{bmatrix} 0.1 & 0.1 & 0.5 \end{bmatrix} \\
\boldsymbol{p}_5 &= \begin{bmatrix} 0.225 & 0.1 & 0.5 \end{bmatrix} \\
\boldsymbol{p}_6 &= \begin{bmatrix} 0.35 & 0.1 & 0.5 \end{bmatrix} \\
\boldsymbol{p}_7 &= \begin{bmatrix} 0.7 & 0.15 & 0.3 \end{bmatrix} \\
\boldsymbol{p}_8 &= \begin{bmatrix} 0.35 & 0.2 & 0.5 \end{bmatrix} \\
\boldsymbol{p}_9 &= \begin{bmatrix} 0.30 & 0.2 & 0.5 \end{bmatrix} \\
\boldsymbol{p}_{10} &= \begin{bmatrix} 0.28 & 0.2 & 0.5 \end{bmatrix} \\
\boldsymbol{p}_{11} &= \begin{bmatrix} 0.18 & 0.2 & 0.5 \end{bmatrix}
\end{aligned}$$

$$\boldsymbol{C} = \begin{bmatrix} 0.23 & 0.2 & 0.5 \end{bmatrix}; \qquad \boldsymbol{r} = 0.05$$

where, the points p_3, p_4, p_5 e p_8 are the via points, while the points p_0 e p_{10} are the extreme points of the segment of circular trajectory having radius r and center C. The trajectory has an estimate time of 70s expressed in a T vector of times, while the via points have been obtained anticipating the next trajectory of a coefficient ΔT defined as:

$$\Delta T = [\, 0 \quad 0 \quad 0 \quad 1500 \quad 2000 \quad 1500 \quad 0 \quad 0 \quad 1600 \quad 0 \quad 0 \quad 0 \,]$$

By using the vector points $\boldsymbol{p}$ and the vector of times $\boldsymbol{T}$ (which indicates the time required to reach the next point from the current one), it's been possible to generate a trajectory. This is achieved using an end-effector timing law, allowing it to precisely follow the predetermined points in the operational space.

3.2 Operational Space Trajectory Computation

The algorithm used in joint space trajectory planning for a manipulator generates a time sequence of joint variable values to move the manipulator from the initial to the final configuration through intermediate configurations. However, the resulting end-effector motion is challenging to predict due to nonlinear effects introduced by direct

kinematics. To achieve a desired path in the operational space, trajectory planning can be done by interpolating a sequence of prescribed path points or generating analytical motion primitives and relative trajectories.

The time sequence of operational space variable values is used to obtain corresponding joint space variable values through an inverse kinematics algorithm. Computational complexity in the operational space trajectory generation sets a limit on the maximum sampling rate for generating the sequences. To enhance dynamic performance, linear microinterpolation is typically used to increase the update frequency of reference inputs.

When the path does not need to be followed exactly, its characterization can be done through N points specifying the end-effector pose in the operational space at given time instants. This is what has been done in this project and the trajectory is generated by interpolating a smooth function between the path points using various interpolation techniques, where. for a prescribed motion trajectory, the geometric features of the path and timing laws are expressed analytically using motion primitives and time primitives, respectively.

In particular, generating a trajectory in the operational space means to determine a function $x_e(t)$ taking the end-effector frame from the initial to the final pose in a time t_f along a given path with a specific motion timing law. In order to find an analytic expression for *s(t)*, there are many techniques for joint trajectory generation that can be employed. In this case, a sequence of linear segments with parabolic blends can be chosen for *s(t)*, but an alternative could have been the choice of a cubic polynomial.

Because of the discontinuity of the first derivative at the path points between two non-aligned segments, the manipulator will have to stop and then go along the direction of the following segment, showing a "rusty" behavior. Assumed a relaxation of the constraint to pass through the path points, it is possible to avoid a manipulator stop by connecting the segments near the above points, which will then be named operational space via points so as to guarantee, at least, continuity of the first derivative. As already illustrated for planning of interpolating linear polynomials with parabolic blends passing by the via points in the joint space, the use of trapezoidal velocity profiles for the arc lengths allows the development of a rather simple planning algorithm. In detail, it will be sufficient to properly anticipate the generation of the single segments, before the preceding segment has been completed. This leads to following timing law of the trajectory:

$$s_j(t) = \begin{cases} 0 & 0 \le t \le t_{j-1} - \Delta t_j \\ s'_j(t + \Delta t_j) & t_{j-1} - \Delta t_j < t < t_j - \Delta t_j \\ \|\boldsymbol{p}_j - \boldsymbol{p}_{j-1}\| & t_j - \Delta t_j \le t \le t_f - \Delta t_N \end{cases}$$

where, for the computation of $s'_j(t+\Delta t_j)$ it's been used the MATLAB command "*lspb*" as to compute the position, velocity and acceleration with a timing law of interpolating linear polynomials with parabolic blends obtained by anticipating the generation of the second segment of trajectory.

3.3 End-Effector Motion

From the $s_j(t)$ computation, it's possible to compute the end-effector position, velocity and acceleration in the operational space with the following relationships:

$$\boldsymbol{p}_e = \boldsymbol{p}_0 + \sum_{j=1}^{N} \frac{s_j}{\|\boldsymbol{p}_j - \boldsymbol{p}_{j-1}\|}(\boldsymbol{p}_j - \boldsymbol{p}_{j-1})$$

$$\dot{\boldsymbol{p}}_e = \frac{\dot{s}}{\|\boldsymbol{p}_f - \boldsymbol{p}_i\|}(\boldsymbol{p}_f - \boldsymbol{p}_i) = \dot{s}\boldsymbol{t}$$

$$\ddot{\boldsymbol{p}}_e = \frac{\ddot{s}}{\|\boldsymbol{p}_f - \boldsymbol{p}_i\|}(\boldsymbol{p}_f - \boldsymbol{p}_i) = \ddot{s}\boldsymbol{t}$$

From the p_e it's possible to get the end-effector trajector in the plane and in the space:

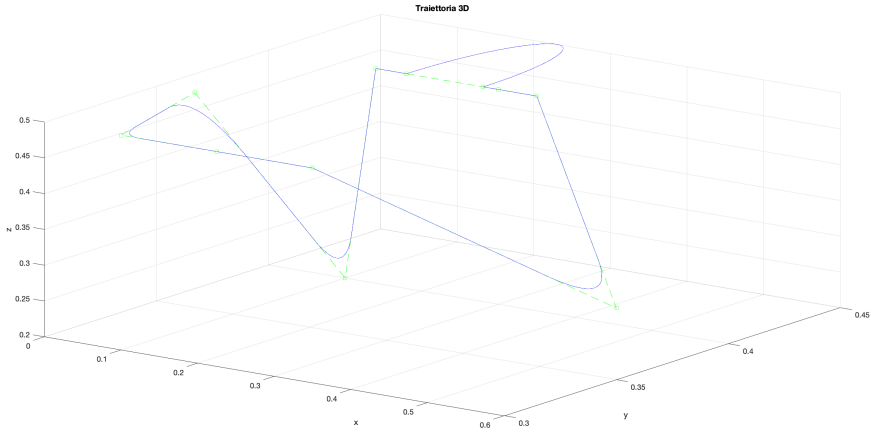


Figure 3.1: 3D Trajectory

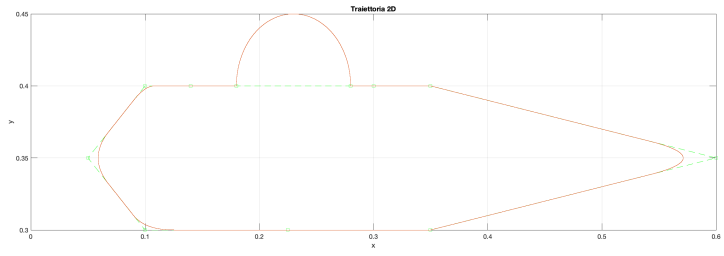


Figure 3.2: 2D Trajectory

The overall behavior of the position, speed and acceleration along the x,y,z axes shows a trapezoidal velocity profile and it is:

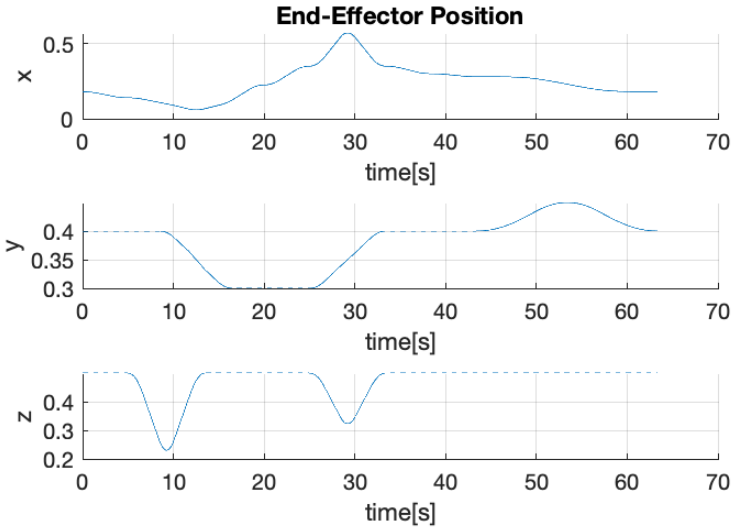


Figure 3.3: End-Effector Position Profile in X,Y,Z

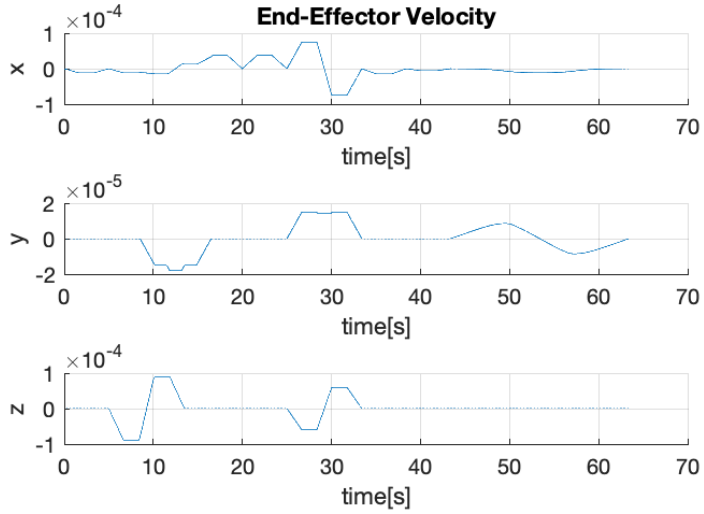


Figure 3.4: End-Effector Velocity Profile in X,Y,Z

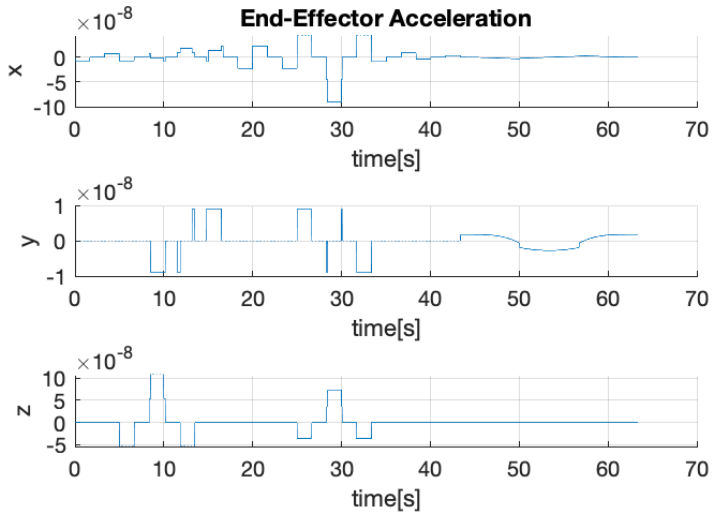


Figure 3.5: End-Effector Acceleration Profile in X,Y,Z

Chapter 4

Kinematic Inversion Algorithm: CLIK

4.1 CLIK Algorithm with Jacobian Inverse

In the previous chapter, an operational space trajectory was specified for the manipulator. As mentioned before, assigning the trajectory in the operational space is generally a better solution than interpolating path points directly in the joint space. Nevertheless, in order to use such trajectory as a reference for a joint space control scheme, the end-effector's positions and orientations must be converted to joint space coordinates. Inverting the kinematics function analytically is a difficult task, which requires geometric and algebraic intuition, and is not always feasible. A great solution to this problem consists of using the pre-mentioned differential kinematics, in particular the Jacobian which estabilishes a *linear mapping* between the vector of joint velocity $\dot{q}$ and the vector of end-effector velocity v_e (Twist) with the following relationship:

$$v_e = J(q) \cdot \dot{q}$$

From which:

$$\dot{q} = J^{-1}(q) \cdot v_e$$

For reconstructing joint variables *q* it's possible to do a numerical integration of inverse differential kinematics equation (in discrete time) and get:

$$q(t_{k+1}) = q(t_k) + \dot{q}(t_k)\Delta t \qquad \textit{in D.T.}$$

Note: this represents the Euler rule for the integration and that's an approximation of the C.T. integral; hence an error called *drift* will be cumulated out for each computation. This error may be unacceptable if the manipulator is moving into a constraint domain. However, to avoid drift phenomena of the solution, computation of $\dot{q}$ cannot be limited to $J^{-}1(q) \cdot v_e$, but it must account an operational space error $e = x_d - x_e$. The goal of the control system will be finding a $\dot{q} : f(e)/e \to 0$

Thus, in the following, only Closed Loop Inverse Kinematics (CLIK) algorithms will

be presented, while the open-loop solution above will be ignored. There can be 2 kind of CLIK Algorithms:

1 CLIK Algorithm based on J_A^{-1}

2 CLIK Algorithm based on J_A^T

4.1.1 CLIK Algorithm based on J_A^{-1}

The term $J_A \cdot \dot{q}$ is linear in $\dot{q}$ but overall it's NL! It's not possible to use the theory of classic control. One technique used for a wide class of NL systems which are linear in control input is the *Feedback Linearization* and it regards the possibility to compesante the linear term and linearize the system and decouple it. Since the SCARA manipulator is not intrinsically redundant, its Jacobian is square. This means that, when not at a singular configuration, the Jacobian can be inverted without resorting to its right pseudo-inverse. It is straightforward to prove that the choice of $\dot{q}$ as:

$$\dot{q} = J_A^{-1} \cdot (\dot{x_d} + K \cdot e)$$

This choice of $\dot{q}$ leads to a linearization of the error dynamics as:

$$\dot{e} + K \cdot e = 0$$

From which:

$$e(t) = e^{-Kt} \cdot e(0)$$

Being K chosen as $K > 0$ is a positive defined matrix, the system is ansintotically stable and this proves that this control algorithm works well.
The control scheme used is the following:

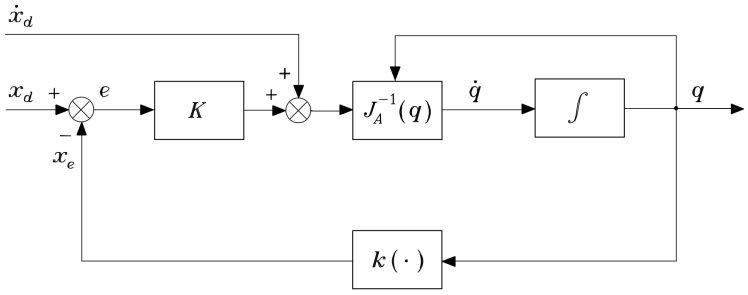


Figure 4.1: Inverse kinematics algorithm with Jacobian inverse

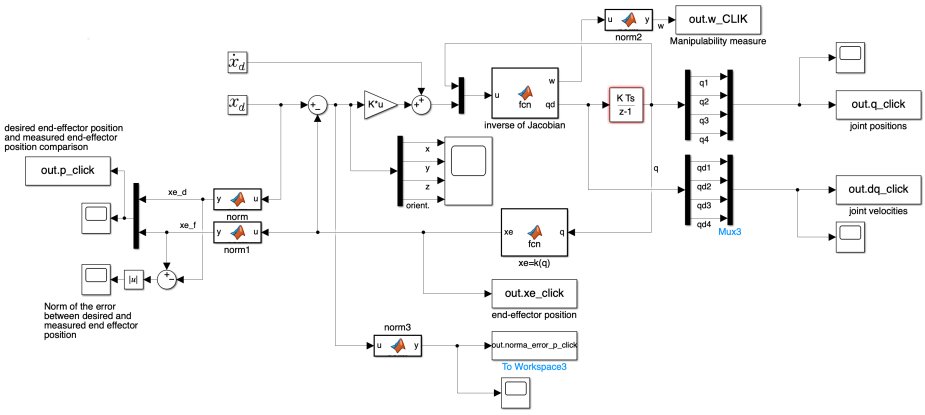


Figure 4.2: Inverse kinematics algorithm with Jacobian inverse (MATLAB-Simulink Implementation)

As a rule of thumb, a good choice of the gain matrix K is by picking it as a diagonal matrix which parameters are at least, as an upper limit, equal to $\frac{1}{\Delta t}$ and its determine the settling times of the error.

$$K = \begin{bmatrix} 400 & 0 & 0 & 0 \\ 0 & 400 & 0 & 0 \\ 0 & 0 & 400 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}$$

Also, the block that contains the Inverse Kinematics relationship also implements a *Damped Least Square* solution since, in case of the Manipulator were close to a singularity, the computation of the Jacobian would be ill-conditioned, because $det(J) \approx 0$.

Thanks to that solution, the Jacobian becomes defined as a new form:

$$J^* = J^T(JJ^T + k^2)^{-1}$$

where k is the damping factor.
Thanks to that the Jacobian is well-conditioned even if we're close to a singularity. The drawback of using the Damped Least Square solution is the loss of the possibility of inverting the exact mapping since $JJ^* \neq I$ (the error may increase), but at the advantage of reducing the joint velocity. From the same function block, it's been computed the manipulability value too, just by computing the determinant of the Jacobian J (since $J = J^T$).
In the feedback loop there's the NL function $k(\cdot)$ which allows the computation of x_e from the q .
The trend of the joints position and velocity during the motion is:

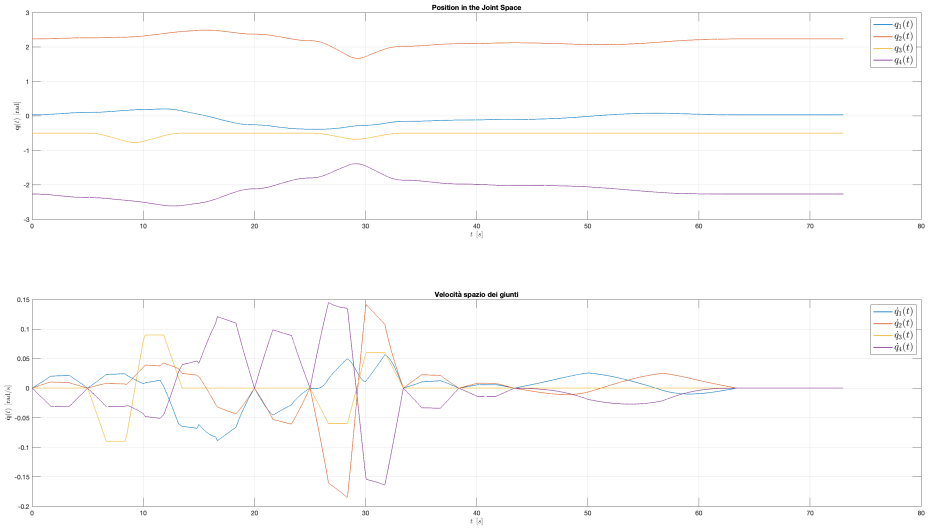


Figure 4.3: Positions and Velocities in the Joint Space

To evaluate the CLIK algorithm it's worth considering the position and orientation errors.

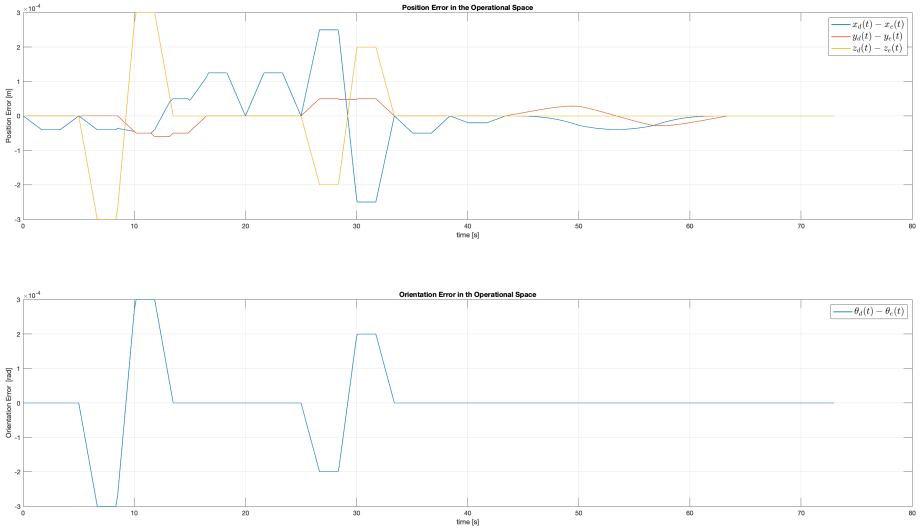


Figure 4.4: Position and Orientation Error in the Operational Space

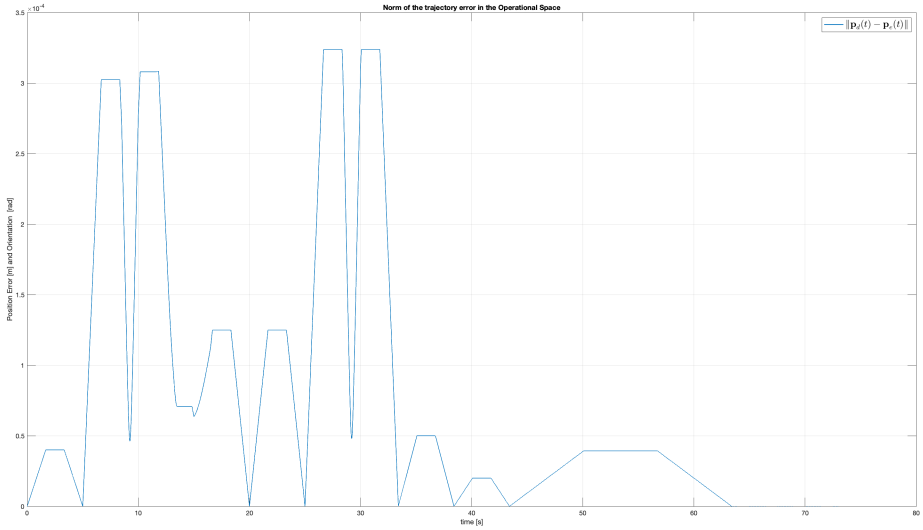


Figure 4.5: Norm of the Trajectory Error in the Operational Space

As it's possible to see, with the chosen value of K, the magnitude of the error is in the order of thenths of millimeters, which is acceptable for most of SCARA applications.

4.1.2 CLIK Algorithm based on J_A^T

The use of transpose of Jacobian in CLIK algorithm allows to avoid situations where the control performance gets worse if the manipulator is close to a singularity. The Jacobian transpose algorithm allows error convergence without linearizing error dynamics, and thus without resorting to any computationally expensive matrix inversion. Using the Lyapunov direct method, it can be demonstrated that the best choice of $\dot{q}$ is:

$$\dot{q} = J_A^T(q) \cdot K \cdot e$$

where matrix $K > 0$, chosen the same of before, guarantees the error system to be asymptotically stable when $\dot{x}_d = 0$.

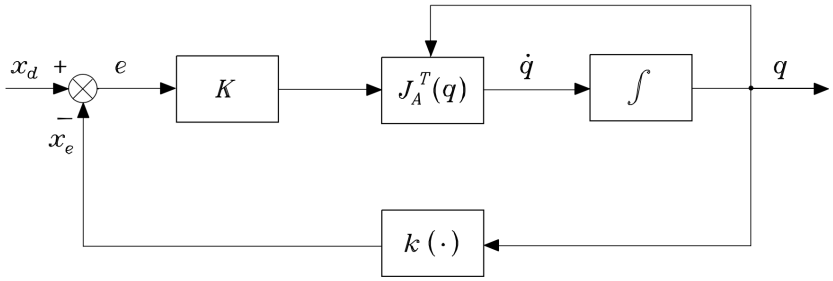


Figure 4.6: Block scheme of the inverse kinematics algorithm with Jacobian transpose

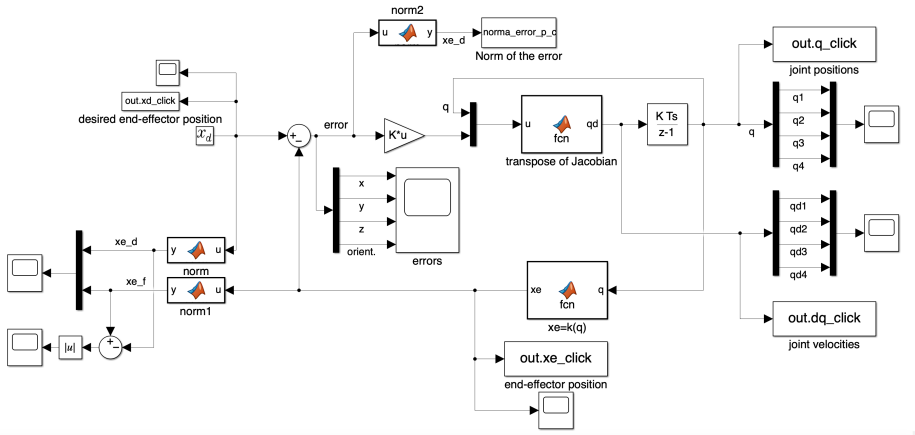


Figure 4.7: Block scheme of the inverse kinematics algorithm with Jacobian transpose (MATLAB-Simulink Implementation)

Note: the choice of K in this algorithm isn't as easy as in the J_A^{-1}. This because we'd be tempted to increase it, since it appears twice in the derivate of Lyapunov function, to make it more negative. Seems to be that the greater is K and the faster is the convergence. In reality it's not really like this because the error term is quadratic! This means that the effect of discretization can make the system instable!
The trend of the joints position and velocity during the motion is:

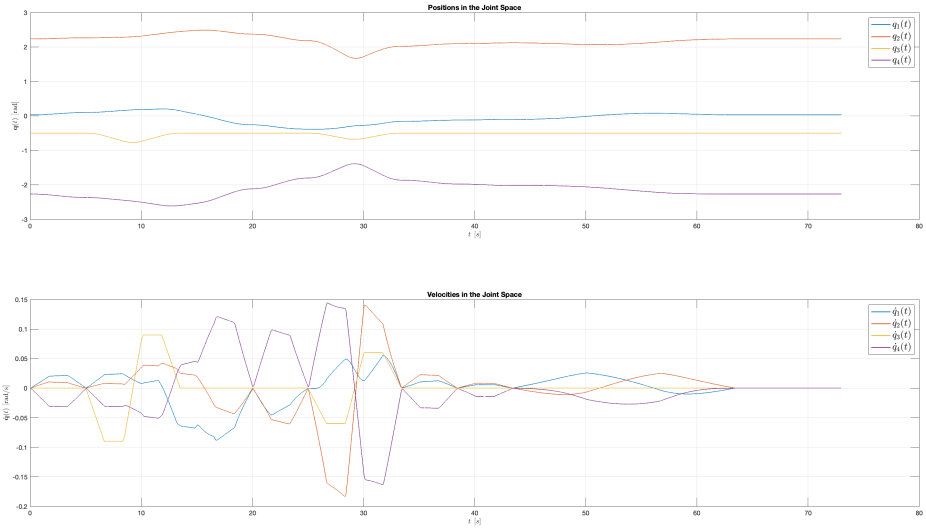


Figure 4.8: Positions and Velocities in Joint Space

To evaluate this CLIK algorithm performance it's worth considering the position and

orientation errors.

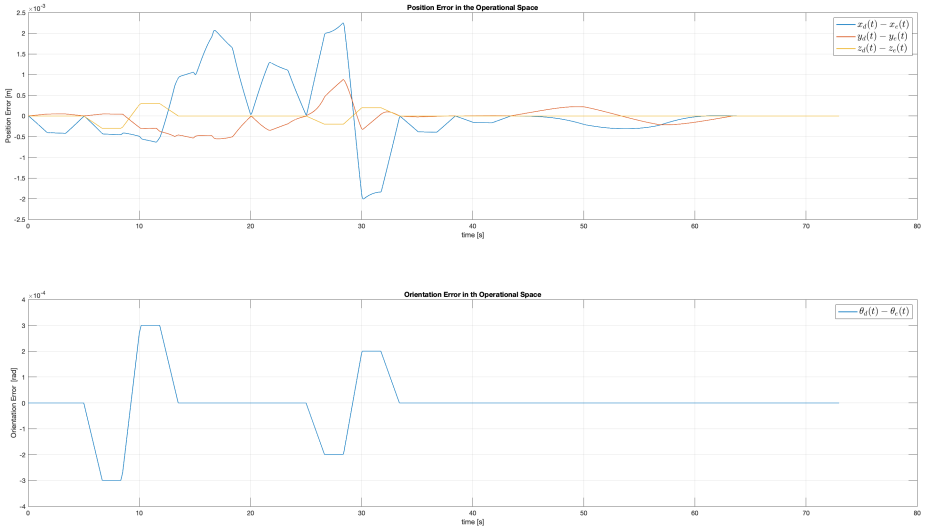


Figure 4.9: Position and Orientation Error in the Operational Space

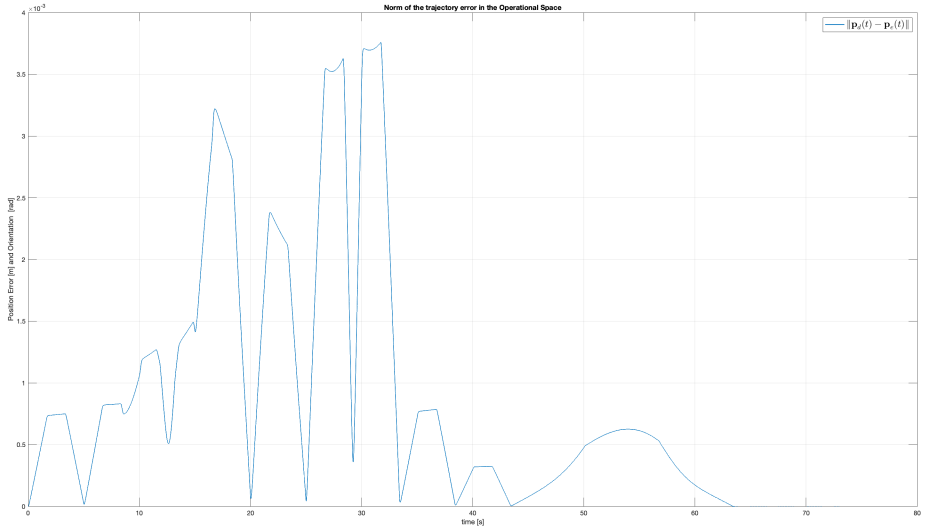


Figure 4.10: Norm of the Trajectory Error in the Operational Space

Nonetheless, it can be seen from figures above that performance is worse than what was achieved with the Jacobian inverse algorithm. Using the same Kp gain matrix indeed, it can be seen in the position error graph that the error is greater than in the previous algo-

rithm. For this reason this algorithm is particularly recommended when x_d is constant since convergence is guaranteed as long as x_d is reachable by the manipulator.

4.2 CLIK algorithm with Jacobian pseudo-inverse

The manipulator, although nonredundant, can be made functionally redundant by planning a trajectory that involves only a reduced number of operational space components. Redundancy is beneficial in avoiding singular configurations or obstacles. In the case of kinematic redundancy, the Jacobian matrix is no longer square but rectangular. To handle this situation, a variant of the algorithm used for the simple Jacobian inverse is employed. The pseudo-inverse of the Jacobian is used instead of the regular inverse (since it couldn't be computed), and to utilize the available degrees of freedom, a cost function optimization is performed.
For testing the algorithm, the x-component of the operational space is relaxed by eliminating the corresponding row from the Jacobian, resulting in:

$$J(q) = \begin{bmatrix} a_2c_{12} + a_1c_1 & a_2c_{12} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

This will allow the manipulability measure $w(q) = sin^2(\theta_2)$ to be maximized, while still satisfying the differential kinematics constraint $\dot{x}_e = J_A(q)\dot{q}$. The resulting joint space trajectory will steer clear of singular configurations $\theta_2 = 0, \pi$. Nevertheless, by ignoring component x, the corresponding operational space trajectory will evolve uncontrolled. The implemented solution is:

$$\dot{q} = J_A^{\dagger}(\dot{x}_d + Ke) + (I_n - J_A^{\dagger}J)\dot{q}_0$$

where:

$$\dot{q}_0 = (\tfrac{\partial w(q)}{\partial q})^T$$

This last quantity has to be computed on-line, with a feedback on joint positions. The control scheme used for this algorithm is:

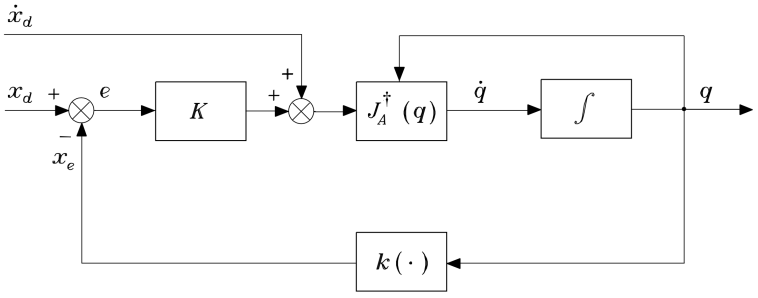


Figure 4.11: Block scheme of the inverse kinematics algorithm with Jacobian pseudo-inverse

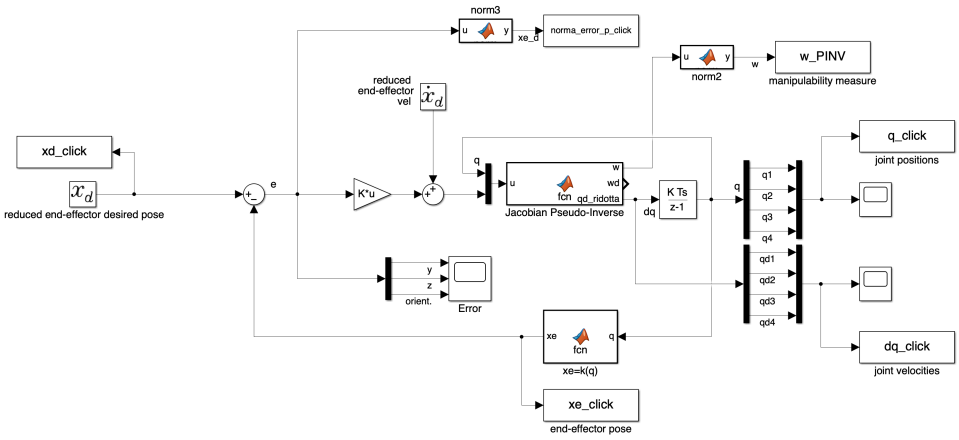


Figure 4.12: Block scheme of the inverse kinematics algorithm with Jacobian pseudo-inverse (MATLAB-Simulink Implementation)

The trend of the joints position and velocity during the motion is:

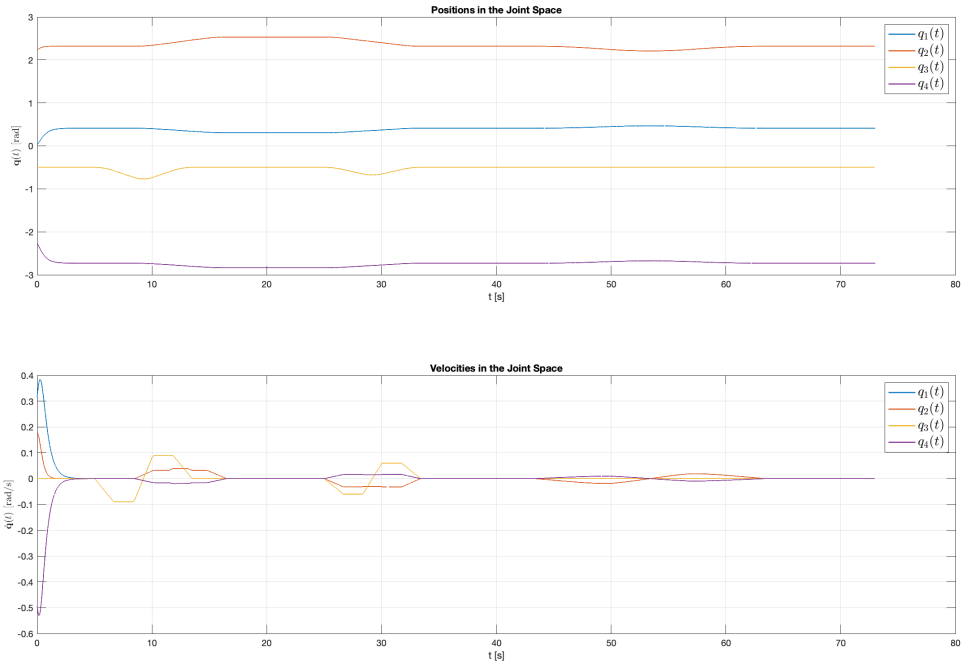


Figure 4.13: Positions and Velocities in Joint Space

To evaluate this algorithm performance it's worth considering the position and orientation errors:

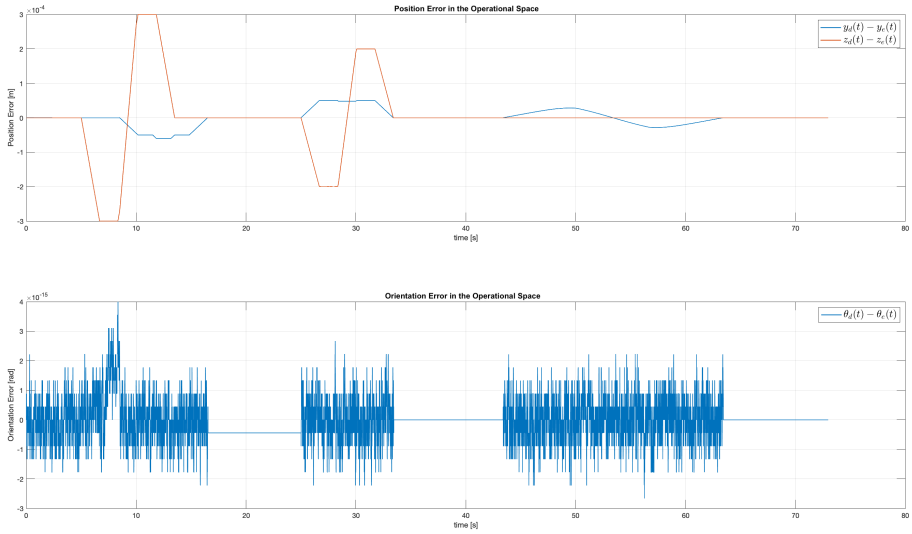


Figure 4.14: Position and Orientation Error in the Operational Space

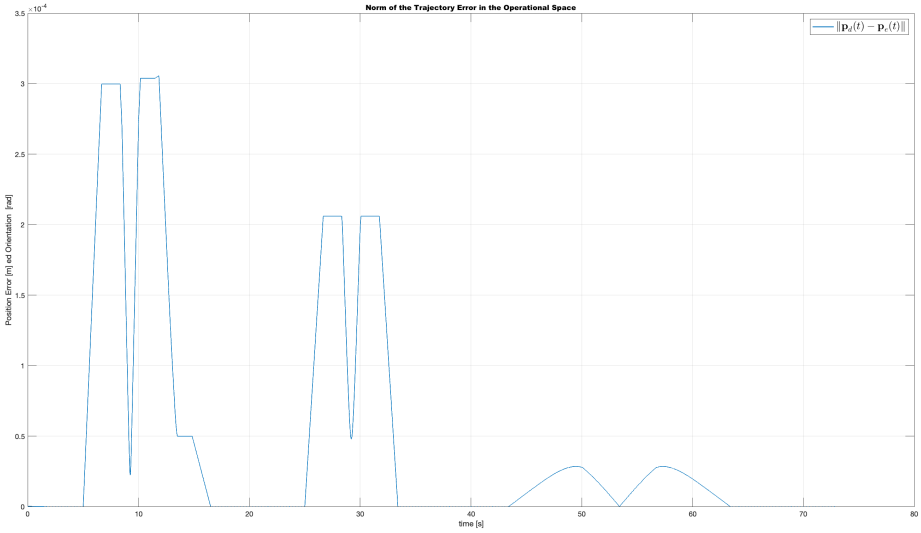


Figure 4.15: Norm of the Trajectory Error in the Operational Space

As it's possible to see, the manipulator now avoids singular configurations affecting joint coordinate θ_2

Note: thanks to this feature added in the control function, it's worth comparing the

manipulability index obtained in the case of the CLIK with the inverse and the one obtained with the pseudo-inverse. This analysis will be done both for a case of relaxation of x.

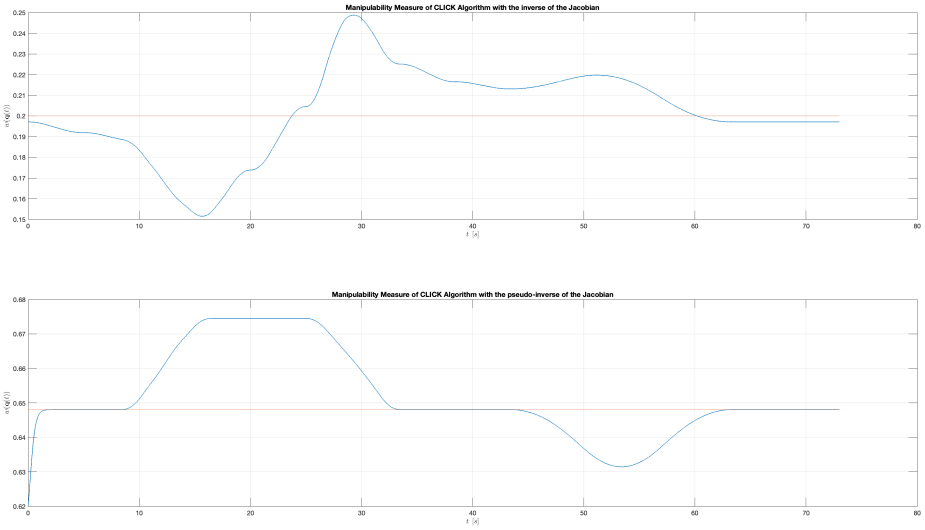


Figure 4.16: Norm of the Trajectory Error in the Operational Space

From the figures it's possible to see that the CLICK with pseudo-inverse algorithm there is an improvement in the mean value of the measure of manipulability, as expected.

Chapter 5

Dynamics and Motion Control

In this final chapter, the results of implementing two centralized motion control algorithms will be presented, specifically the adaptive and robust control schemes. Firstly, an analysis of the manipulator's dynamics will be conducted.
The desired trajectory in operational space will be transformed into joint space using the second-order CLICK algorithm.

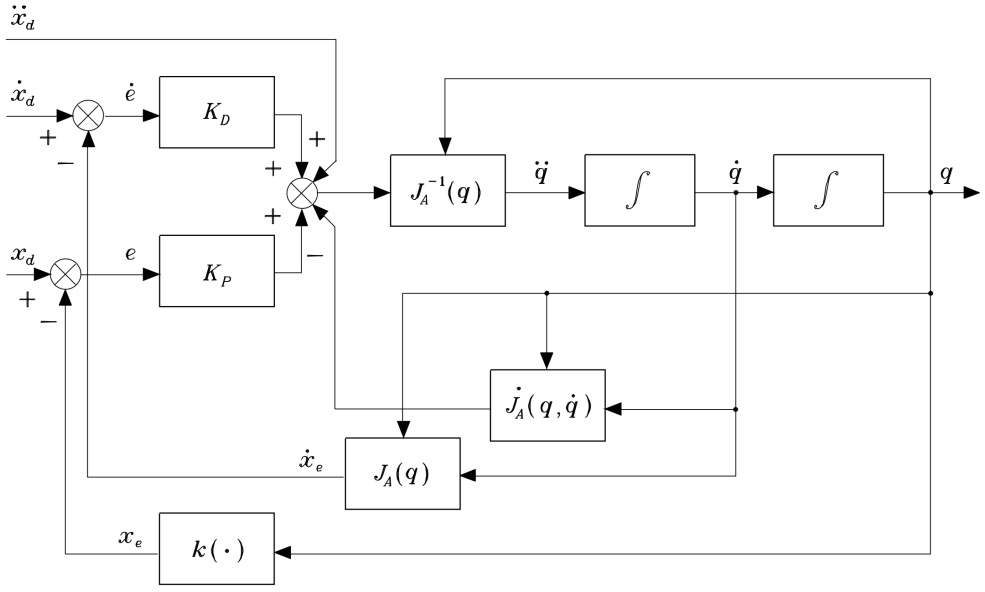


Figure 5.1: Block scheme of the second-order inverse kinematics algorithm with Jacobian inverse

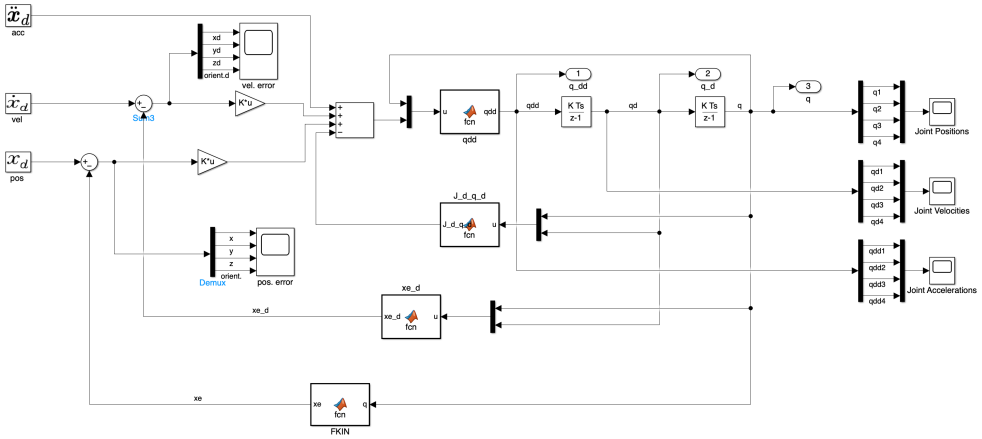


Figure 5.2: Block scheme of the second-order inverse kinematics algorithm with Jacobian inverse

It is important to notice that the desired acceleration in joint space is also required to utilize the considered control algorithms.

5.1 Manipulator Dynamics

The abovementioned control algorithms require the use of the manipulator's dynamic model. Thanks to the Lagrange formulation it's been possible to compute it, obtaining the following relationship expressed in terms of the joint variables (which serve as generalized coordinates):

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

where:

- $B(q)$ is the inertia matrix, which is symmetric and positive definite, expressed as:

$$\boldsymbol{B(q)} = \sum_{i=1}^{4} \left(m_{li}\boldsymbol{J}_P^{(li)T}\boldsymbol{J}_P^{(li)} + \boldsymbol{J}_O^{(li)T}\boldsymbol{R}_i\boldsymbol{I}_{li}^i\boldsymbol{R}_i^T\boldsymbol{J}_O^{(li)} + m_{mi}\boldsymbol{J}_P^{(mi)T}\boldsymbol{J}_P^{(mi)} + \boldsymbol{J}_0^{(mi)T}\boldsymbol{R}_{mi}\boldsymbol{I}_{mi}^{mi}\boldsymbol{R}_{mi}^T\boldsymbol{J}_O^{(mi)} \right)$$

  where:

  - m_{li} is the mass of link i
  - m_{mi} is the mass of motor i
  - I_{li}^i is the inertia tensor of link i
  - I_{mi}^i is the inertia tensor of rotor i
  - J_P^{li} is the position Jacobian relative to link i

  - J_P^{mi} is the position Jacobian relative to rotor i
  - J_O^{li} is the orientation Jacobian relative to link i
  - J_O^{mi} is the orientation Jacobian relative to rotor i.

- $C(q, \dot{q})\dot{q}$ accounts for centrifugal and Coriolis forces. The choice of this matrix is non-unique. It can be shown that by conveniently choosing:

$$c_{ij} = \sum_{k=1}^{4} c_{ijk}\dot{q}_k \qquad\qquad c_{ijk} = \frac{1}{2}\left(\frac{\partial b_{ij}}{\partial q_k} + \frac{\partial b_{ik}}{\partial q_j} - \frac{\partial b_{jk}}{\partial q_i}\right)$$

  which represent the Christoffel first order terms in order to make the matrix $N(q, \dot{q}) = \dot{B}(q) - 2C(q, \dot{q})$ become *skew-symmetric*.

- F_v denotes the diagonal matrix of viscous friction coefficients for each joint, expressed as:

$$\boldsymbol{F}_v = \begin{bmatrix} F_{m1}k_{r1}^2 & 0 & 0 & 0 \\ 0 & F_{m2}k_{r2}^2 & 0 & 0 \\ 0 & 0 & F_{m3}k_{r3}^2 & 0 \\ 0 & 0 & 0 & F_{m3}k_{r3}^2 \end{bmatrix}$$

- $g(q)$ represents the moments generated at each joint by the presence of gravity in the current configuration and it's expressed as:

$$g = \begin{bmatrix} 0 \\ 0 \\ 9.81(m_l + m_{l3}) \\ 0 \end{bmatrix}$$

  and depends only on the load and link masses, and influencing only joint 3 in terms of torque balance.

- τ is the array of actuation torques.

After deriving the dynamic model of the manipulator, defining the desired end-effector trajectory (Chapter 1) and converting this trajectory from operational space to joint space (Chapter 4), the SCARA manipulator can now be controlled directly in the joint space. This control considers a 5 kg mass load at the end-effector.
Inverse dynamics control is a straightforward technique for joint space control. It achieves a precise global linearization of the system dynamics through the use of non-linear state feedback, commonly referred to as feedback linearization. This approach involves expressing the control torques, denoted as "*u*" in the form:

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

with:

$$n(q, \dot{q}) = C(q, \dot{q})\dot{q} + F\dot{q} + g(q)$$

and

$$y = \ddot{q}_d + K_D(\dot{q_d} - \dot{q}) + K_P(q_d - q)$$

These equations lead to leads to the following joint error dynamics:

$$\ddot{\tilde{\boldsymbol{q}}} + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{q}}} + \boldsymbol{K}_P\tilde{\boldsymbol{q}} = \boldsymbol{0}.$$

where $\tilde{q} = q_d - q$
Nevertheless, this only works if the terms $B(q)$ and $n(q, \dot{q})$ are accurately known, in order to obtain a perfect cancellation. Such a condition is not realistic, as the compensation may be intentionally partial, or affected by uncertainty. *Robust* and *Adaptive* Control are two techniques aimed at counteracting the effects of an imperfect compensation. Both control schemes are centralized, and require the reference trajectory to be specified not only as regards joint positions and velocities, but also for accelerations. Consequently, it is useful to use a second-order CLIK algorithm: the second-order Jacobian inverse algorithm (Fig. 5.1)

5.2 Robust Control

Robust control allows the control of the manipulator despite not having the exact dynamic model, which is practically always true in real applications. In such situations, an estimated dynamic model is assumed to be known, allowing for an approximate compensation of nonlinear dynamic effects and decoupling of the joints. Consequently, the control action is utilized:

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

where $\hat{B}(q)$ and $\hat{n}(q, \dot{q})$ represent estimates of the actual dynamic model matrices
In this control, the y is chosen as:

$$y = \ddot{q_d} + K_D\dot{\tilde{q}} + K_P\tilde{q} + w$$

where the first three terms offer a linear feedback and feedforward action whose purpose is to stabilize the dynamics of the error, while the term w serves to contrast the effect of the uncertainty on the dynamic model, which makes the sliding subspace attractive. The latter is a region of the state space and is an attractive subspace given that in it the control law w ideally switches at an infinite frequency and all the error components tend to 0 with a matrix-dependent transient K_P ,K_D and Q . It's been operated only on the gain matrices K_P and K_D. Therefore, once the state is finalized to belong to this subspace, the convergence of the error is guaranteed and this control law is defined as:

$$\boldsymbol{w} = \begin{cases} \frac{\rho}{\|\boldsymbol{z}\|}\boldsymbol{z} & \text{per } \|\boldsymbol{z}\| \geq \epsilon \\ \frac{\rho}{\epsilon}\boldsymbol{z} & \text{per } \|\boldsymbol{z}\| < \epsilon \end{cases}, \quad \rho \geq ||\boldsymbol{\eta}||$$

with $\rho > 0$ and it's the greater as the larger is the uncertainty on the model, and $\epsilon > 0$ which defines the width of the boundary layer (therefore the greater is ϵ and the less is the presence of chattering in the control action, but a bigger error is allowed). It should be pointed out that w's nonlinear structure in the relation above is motivated by the necessity of avoiding high frequency commutations in the control signal (a phenomena known as *chattering*).
In the following simulations $\hat{B}$ is considered to be the a matrix obtained by considering only an estimate of the diagonal terms of the matrix B, in fact the diagonal terms are the ones with the greatest weight. This choice tends to underline how the robust control allows to avoid the computation of all the terms of the matrix B showing lower computational load. $\hat{n}$ instead, is obtained by neglecting the Coriolis and centrifugal force terms.
The realized scheme of the discussed Robust Control is:

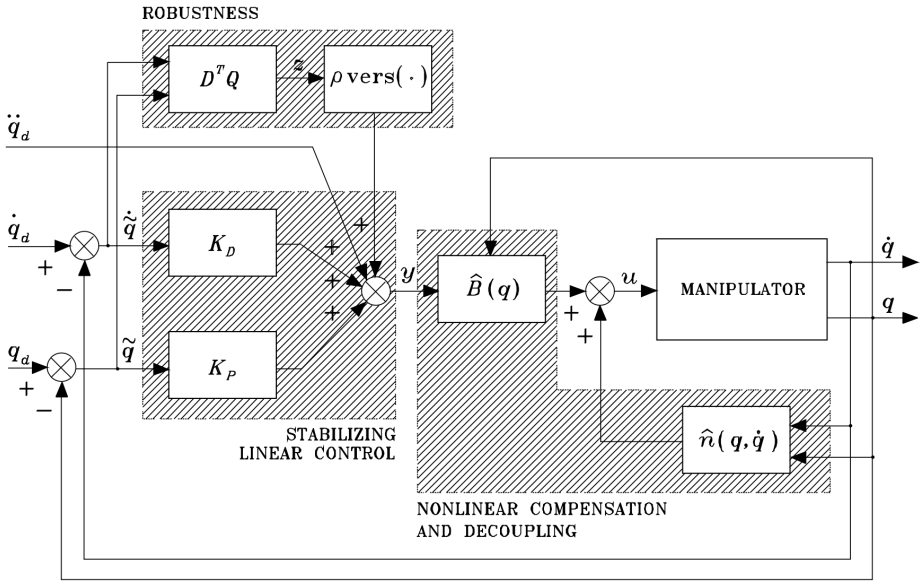


Figure 5.3: Block scheme of joint space robust control

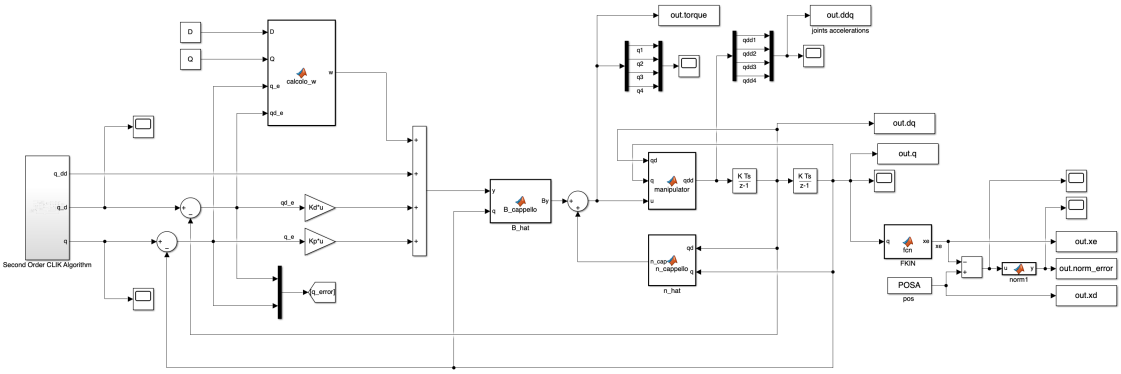


Figure 5.4: Block scheme of joint space robust control (MATLAB-Simulink Implementation)
where ρ, ϵ, K_p, K_D values have been chosen experimentally with the aim of improving the manipulator dynamics. Plotting the motion and the control values, results:

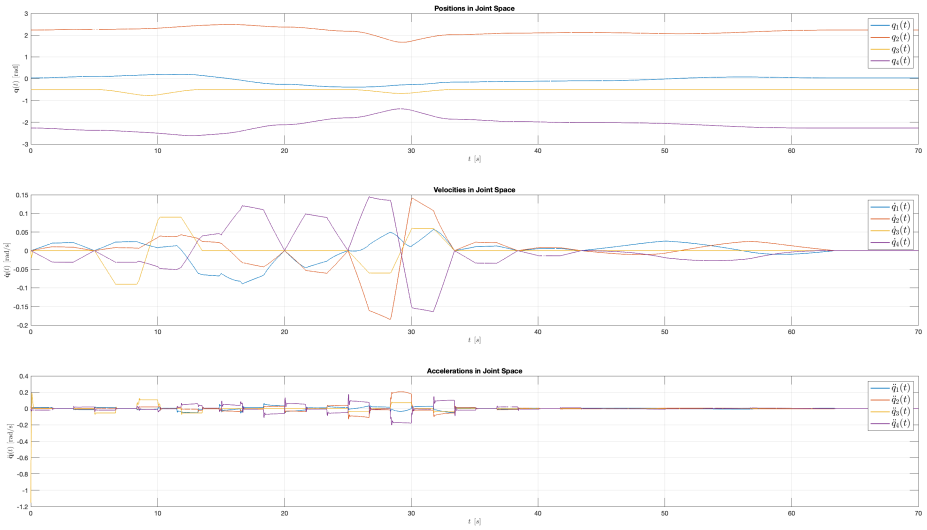


Figure 5.5: Positions, Velocities and Accelerations in the Joint Space

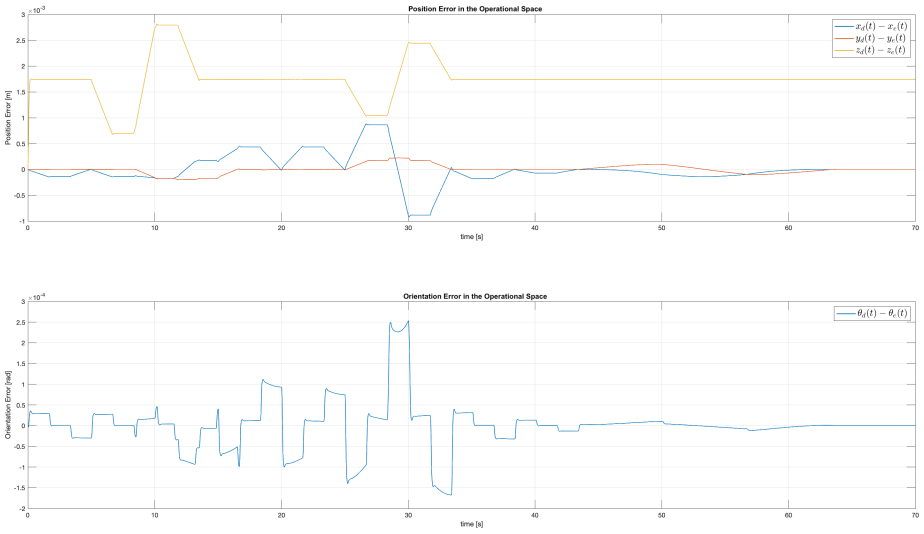


Figure 5.6: Norm of the Trajectory Error in the Operational Space

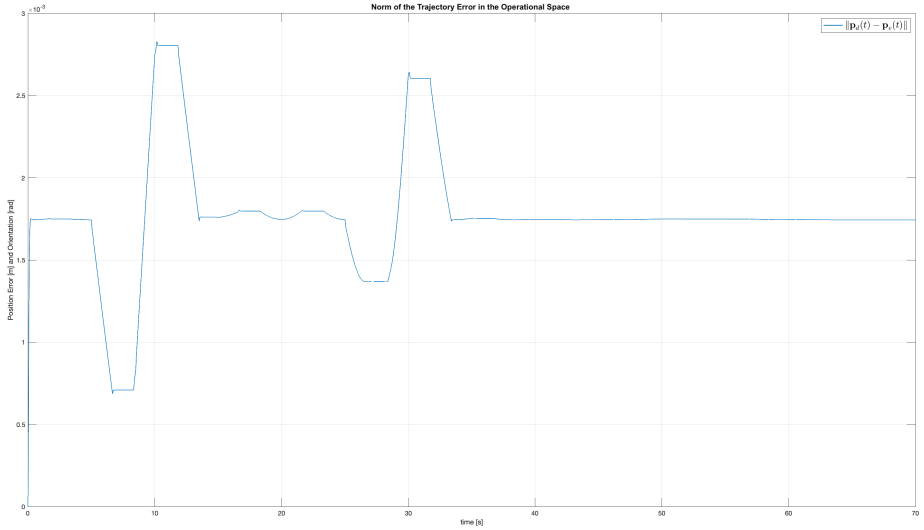


Figure 5.7: Norm of the Trajectory Error in the Operational Space

The value of the torque provided for each joint is:

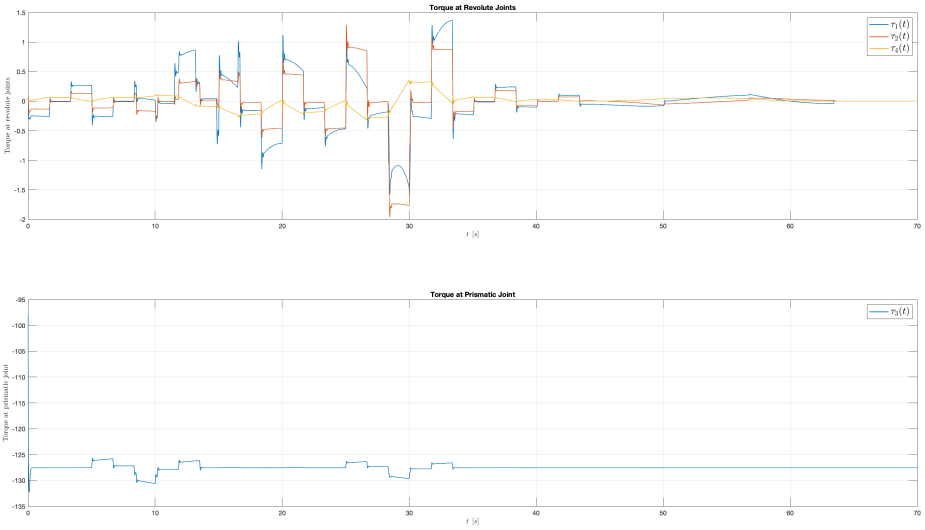


Figure 5.8: Torque Values

5.3 Adaptive Control

Adaptive control is useful in controlling the motion of a manipulator since, in general, the true dynamic model of the manipulator has the same structure as the computational model with the equation seen in the previous paragraph. The discrepancy between the real and computational models will occur at the parametric level. Therefore, an online adaptation of the computational model's parameters to those of the real model is performed, taking advantage of the linearity of the dynamic model with respect to the parameters. In particular, it is possible to rewrite the equation of the dynamic model as:

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

where Y is a matrix called *regressor*, and π is vector of *manipulator parameters*, defined as:

$$\boldsymbol{\pi}^T = \begin{pmatrix} m_{l_1} & I_{l_1} & I_{m_1} & F_{m1} & m_{l_2} & I_{l_2} & I_{m_2} & F_{m2} & m_{l_3} & I_{l_3} & I_{m_3} & F_{m3} & m_{l_4} & I_{l_4} & I_{m_4} & F_{m4} \end{pmatrix}$$

So thanks to that, the dynamic model of a manipulator is linear in its parameters. If an uncertainty in the estimates of such parameters is assumed, this property can be exploited to derive the control law:

$$\begin{aligned} \boldsymbol{u} &= \widehat{\boldsymbol{B}}(\boldsymbol{q})\ddot{\boldsymbol{q}}_r + \widehat{\boldsymbol{C}}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}_r + \widehat{\boldsymbol{F}}\dot{\boldsymbol{q}}_r + \widehat{\boldsymbol{g}} + \boldsymbol{K}_D\boldsymbol{\sigma} \\ &= \boldsymbol{Y}\left(\boldsymbol{q}, \dot{\boldsymbol{q}}, \dot{\boldsymbol{q}}_r, \ddot{\boldsymbol{q}}_r\right)\widehat{\boldsymbol{\pi}} + \boldsymbol{K}_D\boldsymbol{\sigma} \end{aligned}$$

where π represents the available parameter estimates, and:

$$\dot{q}_r = \dot{q} + \Sigma\tilde{q}$$
$$\ddot{q}_r = \ddot{q}_d + \Sigma\dot{\tilde{q}}$$
$$\sigma = \dot{q}_r - \dot{q} = \dot{\tilde{q}} + \Sigma\tilde{q}$$

From here, using Lyapunov's direct method, it can be demonstrated that if the estimates $\hat{\pi}$ are updated with the adaptive law:

$$\dot{\hat{\pi}} = K_\pi^{-1} Y^T(q, \dot{q}, \dot{q}_r, \ddot{q}_r)$$

then errors $\dot{\tilde{q}}$ and $\tilde{q}$ converge to zero and is bounded. It also hold that, aymptotically:

$$Y^T(q, \dot{q}, \dot{q}_r, \ddot{q}_r)(\hat{\pi} - \pi) = 0$$

but this does not imply that $\hat{\pi} \to \pi$, because this indirect adaptive law only aims at stabilizing error dynamics, without actually finding the true parameters.
The implemented control scheme is:

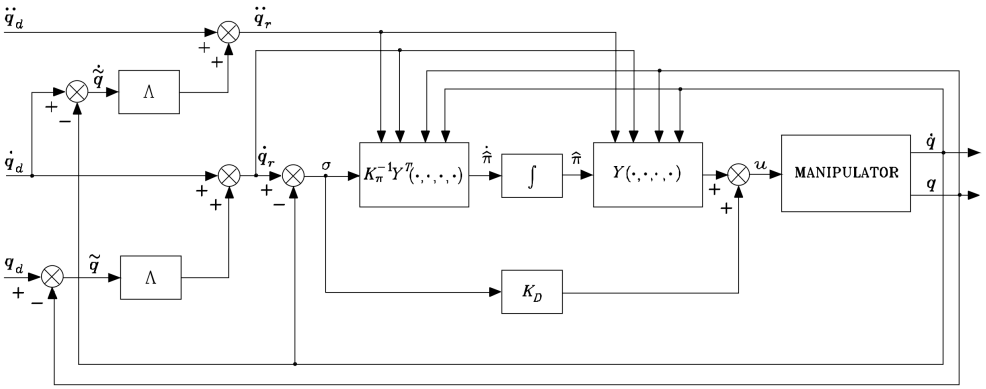


Figure 5.9: Block scheme of joint space adaptive control

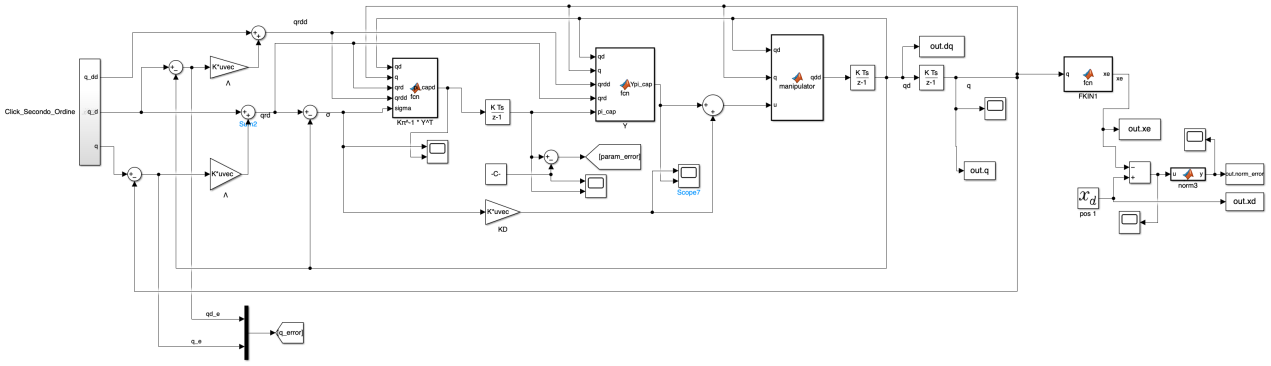


Figure 5.10: Block scheme of joint space adaptive control (MATLAB-Simulink Implementation)

Matrix K_π was chosen this way in order to concentrate the adaptive law on the only parameter affected by uncertainty, which is the mass of the last link. Control characteristics and performance can be seen by plotting the errors, joints positions, velocities, accelerations and torques:

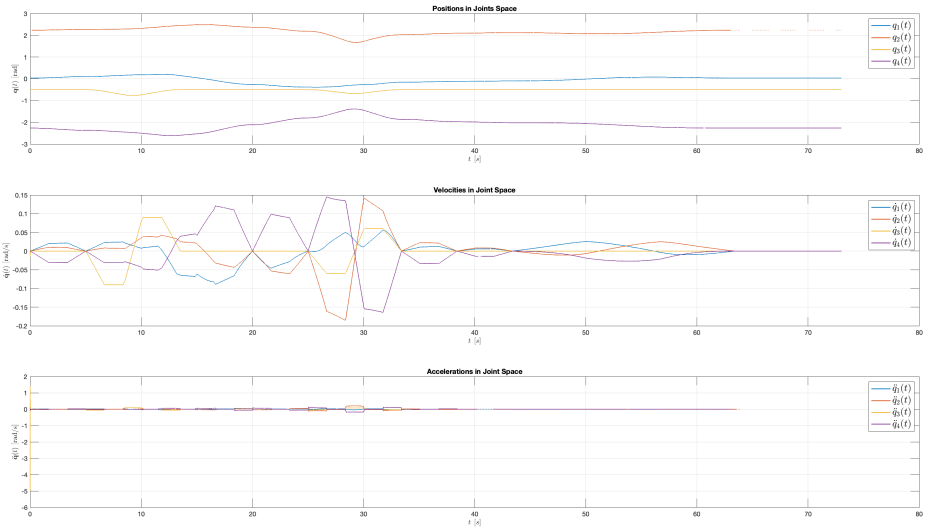


Figure 5.11: Positions, Velocities and Accelerations in the Joint Space

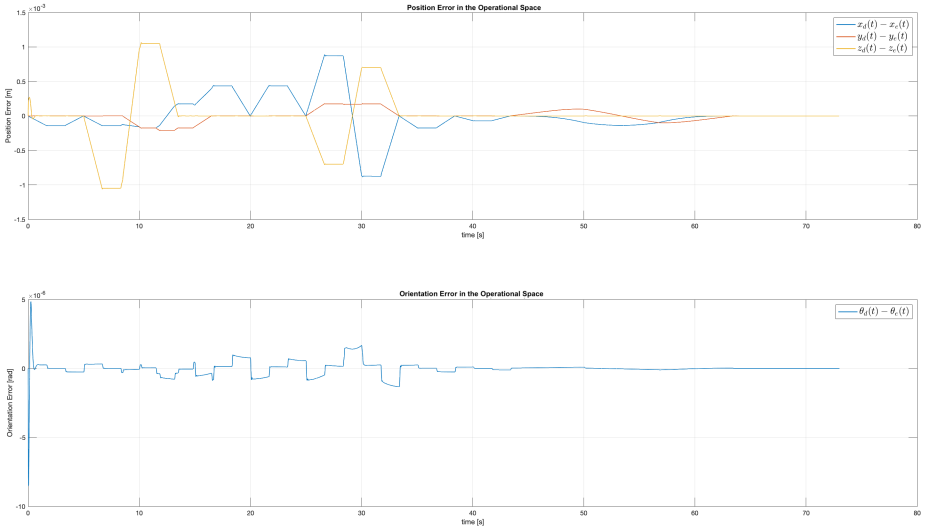


Figure 5.12: Position and Orientation error in the Operational Space

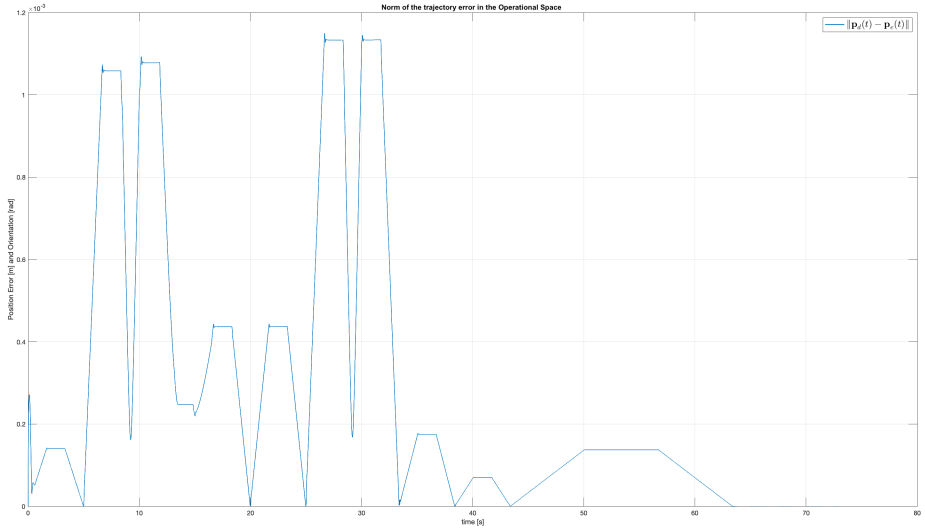


Figure 5.13: Norm of the Trajectory Error in the Operational Space

The value of the torque provided for each joint is:

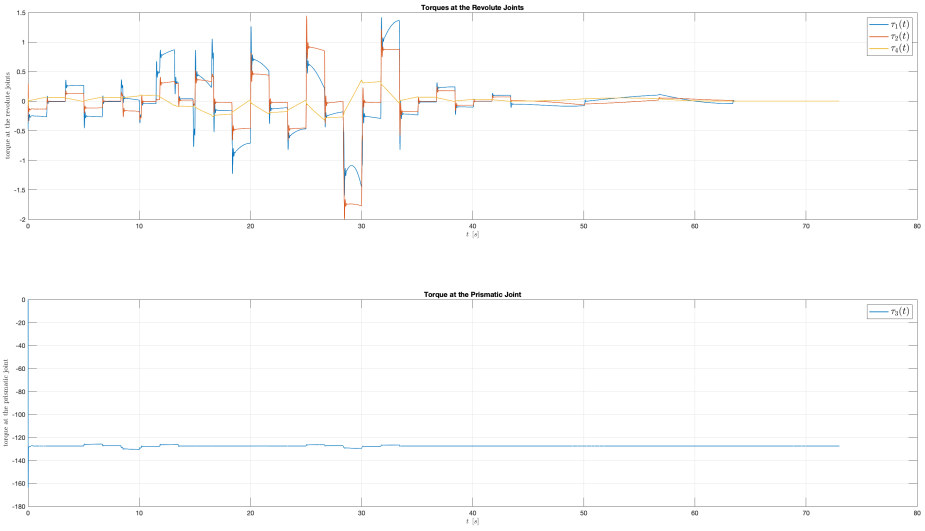


Figure 5.14: Torque Values

As can be seen, the initial estimate for such parameter does not consider the 5 kg mass of the manipulator's load, but converges to after a short transient. This occurrence is

due to the fact that only one parameter's estimate is actually adapted by the algorithm.

5.4 Operational Space Inverse Dynamics Control

The control schemes discussed so far assumed the desired trajectory in joint space, while motion specifications are often given in operational space. This necessitates an inverse kinematics algorithm to transform operational space references into joint space references, leading to increased computational load. Operational space control schemes directly consider the operational space, avoiding extensive kinematic inversion. These schemes become necessary when controlling interactions between the manipulator and the environment, as joint space control is suitable only for motion control in free space. In situations where the manipulator's end-effector is constrained by the environment, operational space control allows controlling both positions and contact forces effectively. However, operational space control schemes have considerable computational requirements and may affect overall control system performance due to demanding computational loads. Operational space control schemes involve directly comparing the specified operational space trajectories with the measurements of the manipulator outputs. To achieve this, the control system requires actions to transform the error from operational space, where it is specified, to the joint space, where control generalized forces are applied.
A possible dynamics control algorithm in operational space is called *Inverse Dynamics Control*.
Recalling the manipulator dynamic model:

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

the choice of the *inverse dynamics linearizing control*:

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

leads to the system of double integrators $\ddot{q} = y$.
The new control input y is to be designed so as to yield tracking of a trajectory specified by $x_d(t)$. To this end, the second-order differential equation:

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

suggests, for a nonredundant manipulator, the choice of the following control law:

$$\boldsymbol{y} = \boldsymbol{J}_A^{-1}(\boldsymbol{q})\big(\ddot{\boldsymbol{x}}_d + \boldsymbol{K}_D\dot{\tilde{\boldsymbol{x}}} + \boldsymbol{K}_P\tilde{\boldsymbol{x}} - \dot{\boldsymbol{J}}_A(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}}\big)$$

with K_P and K_D positive definite (diagonal) matrices. This choice, leads to:

$$\ddot{\tilde{x}} + K_D\dot{\tilde{x}} + K_P\tilde{x} = 0$$

which describes the operational space error dynamics, with K_P and K_D determining the error convergence rate to zero. Since measurements of x_e and $\dot{x}_e$ are indirect, the controller must compute the direct kinematics functions $k(q)$ and $J_A(q)$ on-line, which is a lot more heavy computationally.

The control scheme of Inverse Dynamics Control is:

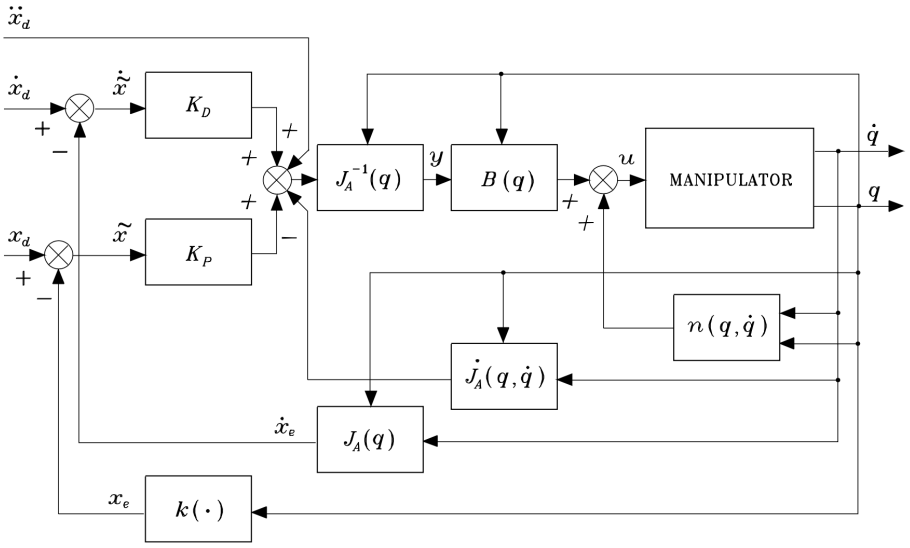


Figure 5.15: Block scheme of operational space inverse dynamics control

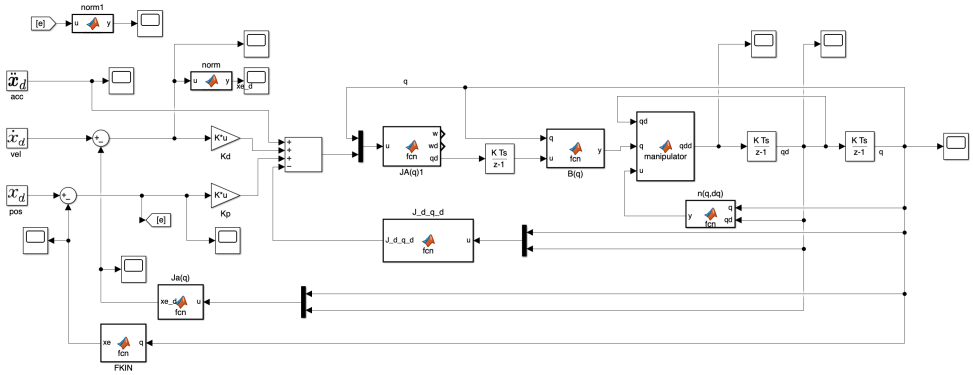


Figure 5.16: Block scheme of operational space inverse dynamics control (MATLAB-Simulink Implementation)

By plotting the graphs of the quantities of interest it is possible to evaluate the control performances:

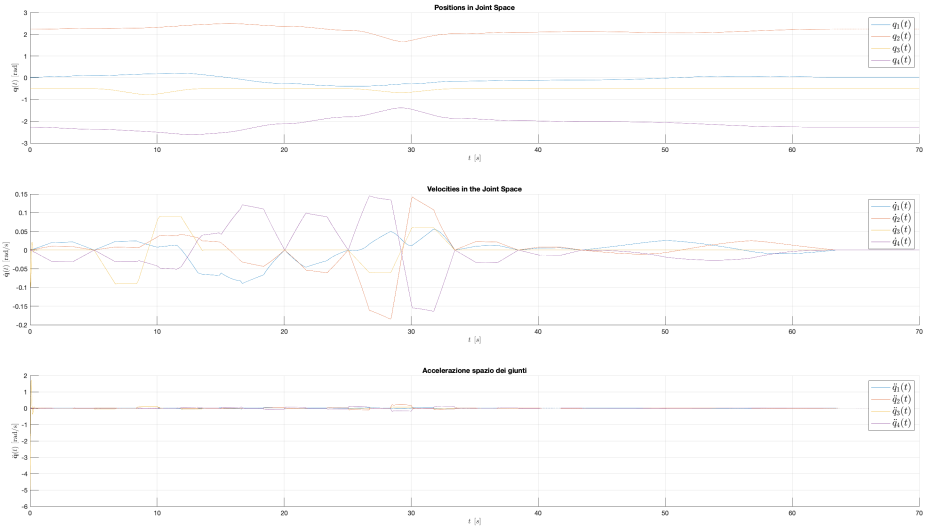


Figure 5.17: Positions, Velocities and Accelerations in the Joint Space

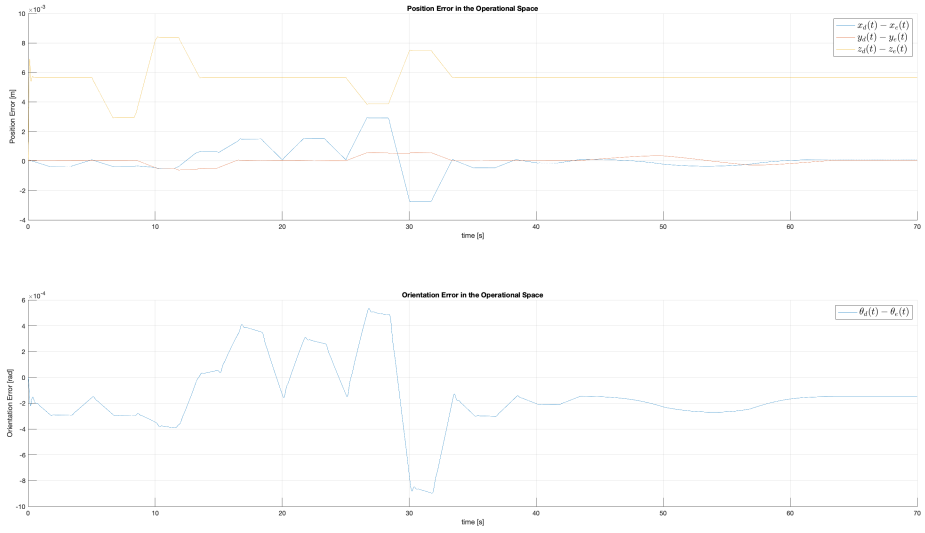


Figure 5.18: Position and Orientation error in the Operational Space

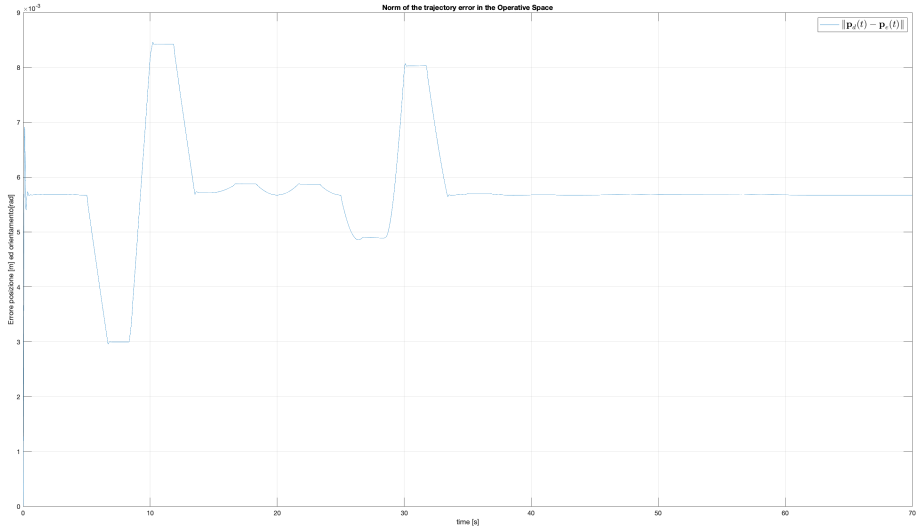


Figure 5.19: Norm of the Trajectory Error in the Operational Space

The value of the torque provided for each joint is:

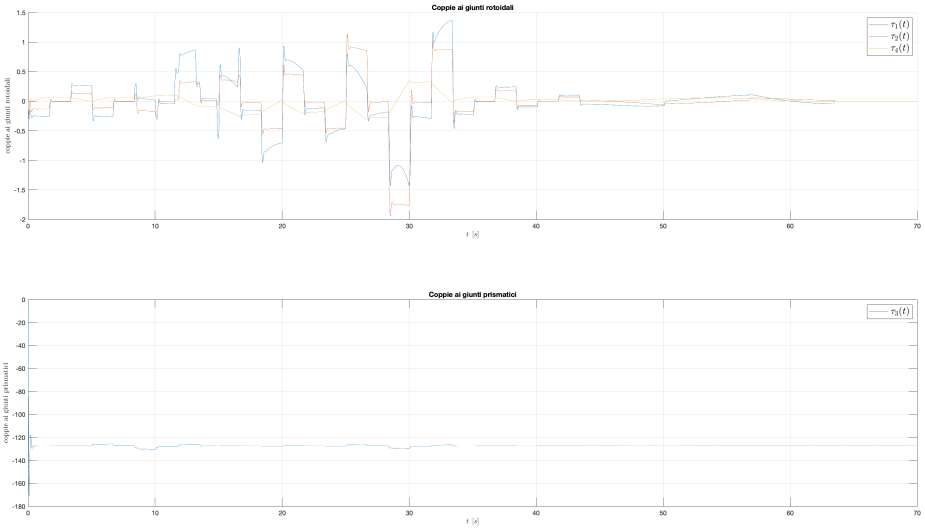


Figure 5.20: Torque Values

References

Corke, P. (2017). *Robotics, vision and control*. Springer International Publishing.

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009). *Robotics: Modeling, planning and control*. Springer London.

In photo, the student Salvatore Granata and Professor Bruno Siciliano, both smiling, at the Engineering University headquarters in Via Claudio. It's a lively scene, with a captivating view of the famous Diego Armando Maradona Stadium located behind them

The proposed work is the result of the knowledge acquired during the Master's course in Robotics Engineering "Foundations Of Robotics" taught by Professor Bruno Siciliano, in the academic year 2022-2023. The report has been entirely written by Salvatore Granata. The project was developed with MATLAB and SIMULINK software. The scripts were realized by Salvatore Granata, using the Robotics System Toolbox and Robotics Toolbox and referring to the course textbook.