UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

TECHNICAL PROJECT FOUNDATIONS OF ROBOTICS

INGEGNERIA DELL'AUTOMAZIONE E ROBOTICA

SCARA MANIPULATOR

Chiar.mo Prof. Siciliano Bruno

Patellaro Emmanuel

P38000239

Anno Accademico 2023/2024

Contents

1 INTRODUCTION

The objective of the study presented in this paper is to analyse the *kinematics*, the *dynamics* and the *manipulability* of the *SCARA manipulator*, followed by *trajectory planning* and the design of various *control techniques*.

The *SCARA manipulator* (Figure 1.1) is a very common industrial robot. The acronym stands for *Selective Compliance Assembly Robot Arm,* and it can be realized with *two revolute joints* and *one prismatic joint* (they are actuated by *electric motors*) in such a way that all the axes of motion are parallel. This type of manipulator is a *4 DOFs* manipulator and it offers high stiffness to vertical loads and compliance to horizontal loads. The typical workspace is illustrated in Figure 1.2.

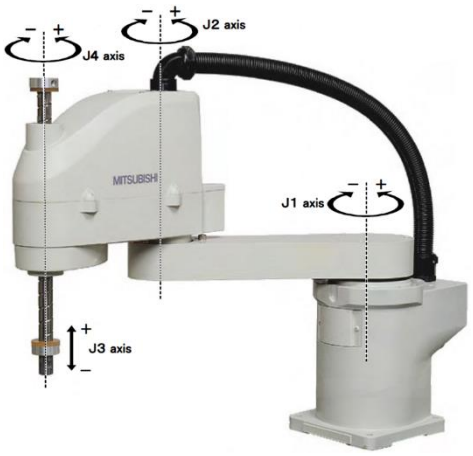

Figure 1.1: SCARA Manipulator

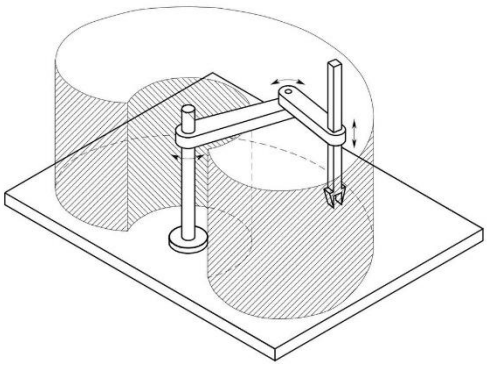

Figure 1.2: SCARA Manipulator Workspace

2 MODELLING

To begin with, the calculation of the *Kinematic Model* and the *Dynamic Model* of this manipulator is required. These models will subsequently be used to achieve the objectives described in the previous chapter.

2.1 KINEMATIC MODEL

Let's start with the *Kinematic Model*. In general, *Kinematics* describes the analytical relationship between the joint positions and the end-effector position and orientation, while *Differential Kinematics* describes the analytical relationship between the joint velocities and the end-effector velocities thanks to the use of the *JACOBIAN* [1](*Robotics is the land of Jacobians*). All of this allows for the study of two important problems:

- *Direct Kinematic Problem*: it concerns the determination of a method to describe the end-effector motion as a function of the joint motion
- *Inverse Kinematic Problem*: it concerns the inverse problem in such a way to transform the desired end-effector motion into the corresponding joint motion

It is always possible to solve the *Direct Kinematics Problem*, unlike the *Inverse* one which may not have an analytical solution. In fact, if the *joint variables* are specified, the complexity of the robotic system does not matter, as the resulting pose of the end-effector can always be determined.

The computation of *Direct Kinematics* through a *geometric approach* is possible only for very simple structure (for example a Two(or Three)-link planar arm). So, it is useful to adopt a systematic approach based on the *Denavit-Hartenberg Convention* [2]. By applying the standard rules of this convention, it is possible to arrange the *coordinate frames* in the figure in such a way to define the relative *position* and *orientation* of two consecutive links (Figure 2.1).

[1] The Jacobian matrix of a vector-valued function of several variables is the matrix of all its first-order partial derivatives. The Jacobian is one of the most important concepts in Robotics (very powerful)
[2] To describe the relative position between two frames, ideally, 6 parameters are to be used (3 for position and 3 for orientation). Using this convention, it is possible to do this with only 4 parameters

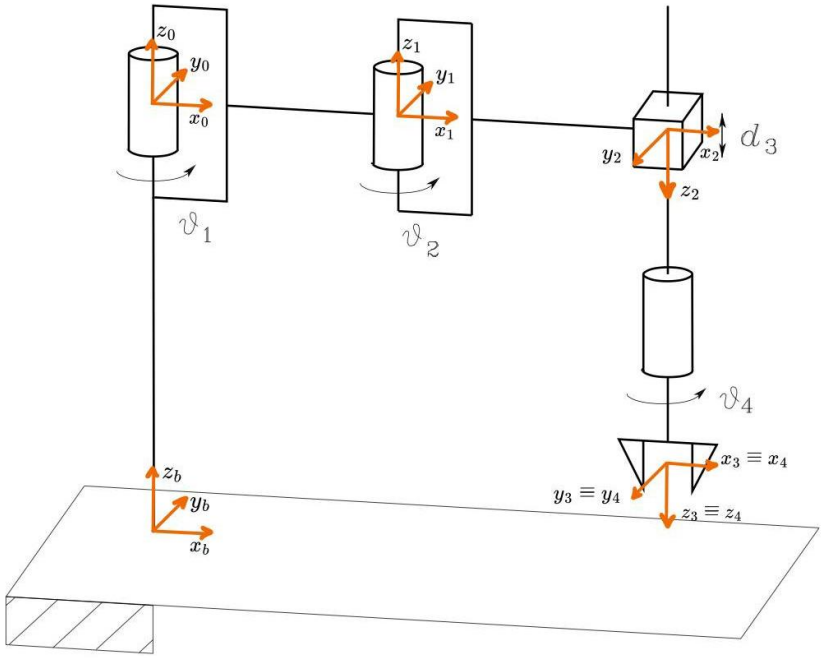

Figure 2.1: Denavit-Hartenberg Convention for SCARA Manipulator

Therefore, it is possible to define the *DH parameters* table for the SCARA manipulator in this way:

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

Table 2.1: DH parameters for the SCARA Manipulator

As is well known, the pose of a body with respect to a reference frame $O_b - x_b y_b z_b$ is described by the position vector of the origin and the unit vectors of a frame attached to the body. So, the *Direct Kinematics Function* is expressed by the *homogeneous transformation*[3] matrix:

$$T_e^b(q) = \begin{bmatrix} n_e^b(q) & s_e^b(q) & a_e^b(q) & p_e^b(q) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[3] The *homogeneous representation* of a generic vector p is $\tilde{p} = \begin{bmatrix} p \\ 1 \end{bmatrix}$ and the generic *homogeneous transformation matrix* is $A_i^j = \begin{bmatrix} R_i^j & o_i^j \\ 0^T & 1 \end{bmatrix}$ which is a $4x4$ matrix

where n_e, s_e e a_e[4] are the *unit vectors* of a frame attached to the end-effector, and p_e is the position vector of the origin of such a frame with respect to the origin of the base frame.

For an *open-chain manipulator* (like the SCARA), it is possible to write the coordinate transformation which describes the position and orientation of Frame n with respect to Frame 0 in this way:

$$T_n^0(q) = A_1^0(q_1)A_2^1(q_2) \dots A_n^{n-1}(q_n)$$

and, more generally:

$$T_e^b(q) = T_0^b T_n^0(q) T_e^n$$

with T_0^b and T_e^n constant homogeneous transformations describing the position and orientation of Frame 0 with respect to the base frame, and of the end-effector frame with respect to Frame n, respectively.

To compute $T_n^0(q)$ it is useful remember that:

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

For the mechanical structure of a manipulator, each DOFs is typically associated with a joint articulation and constitutes a *joint variable*. The vector of joint variables, in this case, is denoted as:

$$q = \begin{bmatrix} \theta_1 \\ \theta_2 \\ d_3 \\ \theta_4 \end{bmatrix}$$

Once the Frames have been defined and using the relations, the *Kinematic Model* is provided, and only simple matrix computations remain to be performed.

So, the equation is:

$$T_4^0(q) = A_1^0(q_1)A_2^1(q_2)A_3^2(q_3)A_4^3(q_4)$$

For this purpose, it is possible to use MATLAB and the script *Kinematic_Model.m*.

[4] a_e is chosen in the *approach* direction to the object, s_e is chosen normal to a_e in the *sliding* plane of the jaws, and n_e is chosen *normal* to the other two in such a way to form a right-handed frame

```
%Kinematic Model
syms theta1 theta2 d3 theta4 a1 a2

A01 = [cos(theta1),-sin(theta1),0,a1*cos(theta1);
    sin(theta1),cos(theta1),0,a1*sin(theta1);
    0,0,1,0;
    0,0,0,1];

A12 = [cos(theta2),-sin(theta2)*cos(pi),0,a2*cos(theta2);
    sin(theta2),cos(theta2)*cos(pi),0,a2*sin(theta2);
    0,0,-1,0;
    0,0,0,1];

A23 = [1,0,0,0;
    0,1,0,0;
    0,0,1,d3;
    0,0,0,1];

A34 = [cos(theta4),-sin(theta4),0,0;
    sin(theta4),cos(theta4),0,0;
    0,0,cos(pi),0;
    0,0,0,1];

T04 = A01*A12*A23*A34;
T04 = simplify(T04);
```

The result is:

```
T04 =

[cos(theta1 + theta2 - theta4),  sin(theta1 + theta2 - theta4), 0, a2*cos(theta1 + theta2) + a1*cos(theta1)]
[sin(theta1 + theta2 - theta4), -cos(theta1 + theta2 - theta4), 0, a2*sin(theta1 + theta2) + a1*sin(theta1)]
[                           0,                             0, 1,                                     -d3]
[                           0,                             0, 0,                                       1]
```

that is[5]:

$$T_4^0(q) = \begin{bmatrix} c_{12(-4)} & s_{12(-4)} & 0 & a_2c_{12} + a_1c_1 \\ s_{12(-4)} & -c_{12(-4)} & 0 & a_2s_{12} + a_1s_1 \\ 0 & 0 & -1 & -d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For completeness, it is possible to consider $T_0^b = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and $T_e^n = I$ in such a way to write, in conclusion:

$$T_e^b(q) = \begin{bmatrix} c_{12(-4)} & s_{12(-4)} & 0 & a_2c_{12} + a_1c_1 \\ s_{12(-4)} & -c_{12(-4)} & 0 & a_2s_{12} + a_1s_1 \\ 0 & 0 & -1 & -d_3 + d_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

But the *Direct Kinematics Equation* can be written in another form:

$$x_e = k(q)$$

which $k(q)$ allows computation of the *operational space variables* from the knowledge of the *joint space variables*. Since the SCARA is a simple structure, it is possible to compute all directly in this form:

[5] A compact form for the sinusoidal functions has been used. For example, $c_{12(-4)} = \cos(\theta_1 + \theta_2 - \theta_4)$ or $s_{12(-4)} = \sin(\theta_1 + \theta_2 - \theta_4)$

$$x_e = k(q) = \begin{bmatrix} a_2c_{12} + a_1c_1 \\ a_2s_{12} + a_1s_1 \\ -d_3 + d_0 \\ \theta_1 + \theta_2 - \theta_4 \end{bmatrix}$$

As mentioned at the beginning of this chapter, to describes the analytical relationship between the joint velocities and the end-effector velocities the *JACOBIAN* is needed. The fundamental equation of *Differential Kinematics* is:

$$\dot{x}_e = \begin{bmatrix} \dot{p}_e \\ \dot{\phi}_e \end{bmatrix} = \begin{bmatrix} J_p(q) \\ J_\phi(q) \end{bmatrix} \dot{q} = J_A(q)\dot{q}$$

where $J_A(q)$ is the *Analytical Jacobian* and it can be written as:

$$J_A(q) = \frac{\partial k(q)}{\partial q}$$

So, it is possible to write:

$$J(q) = \begin{bmatrix} -a_2s_{12} - a_1s_1 & -a_2s_{12} & 0 & 0 \\ a_2c_{12} + a_1c_1 & a_2c_{12} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix}$$

where the two null rows can be deleted in such a way to obtain a $4x4$ matrix[6]

$$J_A(q) = \begin{bmatrix} -a_2s_{12} - a_1s_1 & -a_2s_{12} & 0 & 0 \\ a_2c_{12} + a_1c_1 & a_2c_{12} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & -1 \end{bmatrix}$$

2.2 DYNAMIC MODEL

After obtaining the *Kinematic Model* in the previous section, the next step is to derive the *Dynamic Model*. The resulting model will serve as a foundation for subsequent control strategies in the following chapter. During the course, it was demonstrated, using the *Lagrange Formulation*, that the Dynamic Model can be written, without interaction forces phenomena, in the form:

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

where:

[6] In this case it is possible to establish a substantial equivalence between J and J_A because the DOFs cause rotation of the end-effector all about the same axis in space (like in the three-link planar arm)

- $B(q)$ is the *Inertia Matrix*, which is symmetric and positive definite (always invertible)

$$B(q) = B_l(q) + B_m$$

with:

$$B_l(q) = \sum_{i=1}^{n} (m_i {J_p^i}^T(q) J_p^i(q) + {J_o^i}^T(q) R_i(q) I_i^i R_i^T(q) J_o^i(q))$$

$$B_m = N_r I_m N_r$$

$$N_r = diag\{n_{r1}, \dots, n_{rn}\}^7 \qquad I_m = diag\{I_{m1}, \dots, I_{mn}\}$$

- $c(q, \dot{q})$ is the matrix that accounts the contributions of *Centrifugal* and *Coriolis* terms
- F_v is the diagonal matrix with the *viscous friction* coefficients
- $g(q)$ represents the *gravity* term
- τ is the vector of the *actuating torques*

Specifically, it is now possible to derive the Dynamic Model of the SCARA robot, having all the necessary tools at our disposal. To this end, as shown in the previous section, a MATLAB script can be utilized, which, with the proper adjustments, allows the output parameter to be the matrix B. It is important to remark that this matrix cannot depend on θ_1 because the moment of inertia cannot depend on the choice of the reference frame as it is a physical property[8]. Let's see it.

[7] In this way, the presence of mechanical couplings between joints is excluded
[8] If a different reference base frame is chosen, then θ_1 will be different

```
%Inertia Matrix
syms a1 a2 l1 l2 theta1 theta2 ml1 ml2 ml3 Il1 Il2 Il4 Im1 Im2 Im3 Im4 kr1 kr2 kr3 kr4 real

Jpl1=[-l1*sin(theta1) 0 0 0;
    l1*cos(theta1) 0 0 0;
    0 0 0 0];

Jpl2 = [-l2*sin(theta1+theta2)-a1*sin(theta1) -l2*sin(theta1+theta2) 0 0;
       l2*cos(theta1+theta2)+a1*cos(theta1) l2*cos(theta1+theta2) 0 0;
       0 0 0 0];

Jpl4 = [-a2*sin(theta1+theta2)-a1*sin(theta1) -a2*sin(theta1+theta2) 0 0;
       a2*cos(theta1+theta2)+a1*cos(theta1) a2*cos(theta1+theta2) 0 0;
       0 0 -1 0];

Jol1 = [0 0 0 0;
        0 0 0 0;
        1 0 0 0];

Jol2 = [0 0 0 0;
        0 0 0 0;
        1 1 0 0];

Jol4 = [0 0 0 0;
        0 0 0 0;
        1 1 0 -1];

Bl1 = ml1*(Jpl1'*Jpl1) + Jol1'*Il1*Jol1;
Bl2 = ml2*(Jpl2'*Jpl2) + Jol2'*Il2*Jol2;
Bl3 = ml3*(Jpl4'*Jpl4) + Jol4'*Il4*Jol4;

Bl = Bl1+Bl2+Bl3;

Nr = diag([kr1 kr2 kr3 kr4]);
Im = diag([Im1 Im2 Im3 Im4]);
Bm = Nr*Im*Nr;

B = Bl+Bm;
B = simplify(B)
```

The script where it is possible to find this MATLAB Code is *InertiaMatrix.m*. The output is difficult to shown due to the numerous elements present in the matrix itself. So, it is possible to compute and to view the matrix B running the MATLAB Script given in the folder with this Technical Project.

Now, once the inertia matrix has been calculated, all that remains is to compute the other terms present in the equation written at the beginning of this section. So, the other elements are:

$$c(q,\dot{q}) = \dot{B}(q)\dot{q} - \frac{1}{2}\left(\frac{\partial}{\partial q}(\dot{q}^T B(q)\dot{q})\right)^T$$

$$F_v = \begin{bmatrix} k_{r2}^2 F_{m2} & 0 & 0 & 0 \\ 0 & k_{r2}^2 F_{m2} & 0 & 0 \\ 0 & 0 & k_{r3}^2 F_{m3} & 0 \\ 0 & 0 & 0 & k_{r4}^2 F_{m4} \end{bmatrix}$$

In conclusion, it is possible to decompose the initial equation (without the friction term) in a compact vector format as

$$\sum_{j=1}^{n} m_{ij}(q)\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk}(q)\dot{q}_j\dot{q}_k + g_i(q) = \tau_i \qquad \forall i \in [1,\dots,n]$$

where

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

are known as *Christoffel Symbols* of the first kind.

Unlike the kinematic model, the Dynamic Model has two strong and important properties:

1. *Skew-Symmetry* of the matrix $\dot{B}(q) - 2C(q,\dot{q})$ [9]
2. *Linearity in the Dynamic Parameters*: $Y_\pi(q,\dot{q},\ddot{q})\pi = \tau$ with $\pi \in \mathbb{R}^p$ and Y_π (*Regressor Matrix* of the robot dynamic model) is a $n\ x\ p$ matrix.

$$\pi_i = \begin{pmatrix} m_i & m_i r_{di,x} & m_i r_{di,y} & m_i r_{di,z} & \bar{I}_{ixx} & \bar{I}_{ixy} & \bar{I}_{ixz} & \bar{I}_{iyy} & \bar{I}_{iyz} & \bar{I}_{izz} & \bar{I}_{mi} \end{pmatrix}^T$$

This model and these properties will be used in Chapter 6 with the implementation of the *Motion Control*.

2.3 SCARA WITH PETER CORKE'S ROBOTICS TOOLBOX

It is possible to show a 3D representation of the SCARA manipulator thanks to the *Peter Corke's Robotics Toolbox* which is very useful for the study of a robot in general (manipulability, kinematics, trajectory planning and so on). Given a general configuration and assigning the corresponding joint variables, it is possible to plot the manipulator in such a way to obtain the following image:

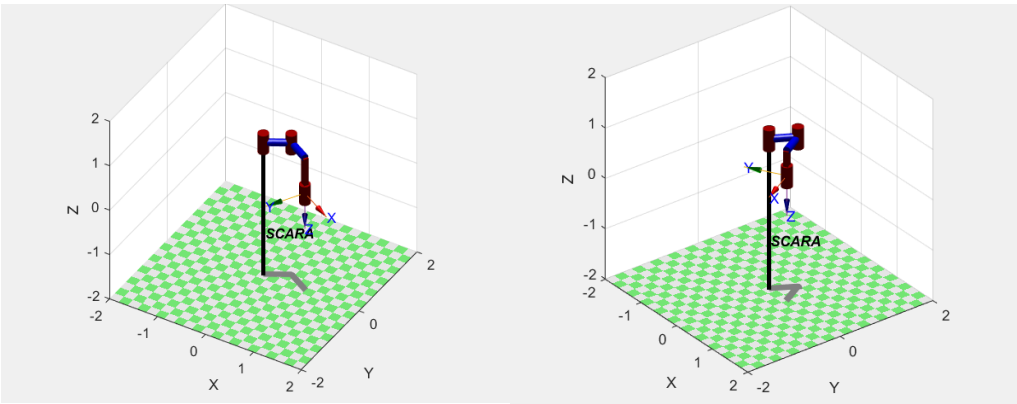

Figure 2.2: Different SCARA configurations with Peter Corke's Robotics Toolbox

[9] This is possible thanks to the use of the Christoffel Symbols

3 ROBOT MANIPULABILITY

To study certain characteristics of a generic manipulator, it is very useful to use the concept of *Manipulability Ellipsoids*. Thanks to the *Differential Kinematics Equation* (as discussed in the previous chapter) and thanks to the *Statics Equation*, it is possible to define some indices aimed at highlighting manipulator performance. The concept of *Ellipsoid* is a very important and crucial concept in Algebra and, for a pure robotics study, it can be used to reveal the manipulator's ability to exert a specific force or velocity (for the specific task) in a given configuration. Given the *kineto-statics duality*, velocity and force ellipsoids are closely related. In fact, as will be clearly shown below, these ellipsoids have the *same axes' directions*, but the *dimensions* are *inversely proportional*: where a high speed can be achieved, a small force can be exerted, and vice versa.

3.1 VELOCITY MANIPULABILTY ELLIPSOID

The *velocity manipulability ellipsoid* represents the attitude of a manipulator to arbitrarily change end-effector position and orientation. Consider the set of joint velocities of constant (unit) norm $\dot{q}^T\dot{q} = 1$ and considering the generical case $\dot{q} = J^{\dagger}(q)v_e$ and substituting with the expression of the *pseudo-inverse of* J, it is possible to write the equation of the points on the surface of an ellipsoid in the end-effector velocity space:

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

The shape and the orientation of the ellipsoid is given by the core of the *quadratic form* $J(q)J(q)^T$ [10]. In addition, it is possible to consider the *volume* of the ellipsoid as the *manipulability measure*:

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

In the direction of the ellipsoid's major axis, the end-effector can move at higher velocities, while along the minor axis, lower velocities are achieved. Moreover, the closer the ellipsoid is to a sphere, the better the end-effector can move *isotropically* in all directions within the Operational Space. For the SCARA, which lacks redundancy, the manipulability measure is given by:

[10] The eigenvectors u_i determine the direction of the principal axes and the singular values $\sigma_i = \sqrt{\lambda_i(JJ^T)}$ give their dimensions

$$w(q) = a_1 a_2 |s_2|$$

In fact, the SCARA has a very simple structure and, in this case, only the first 2 link are of interest. For this reason, the formula $w(q) = a_1 a_2 |s_2|$ is the same as the two-link planar arm.

3.2 FORCE MANIPULABILTY ELLIPSOID

It is possible to do the same for the forces, thanks to the *kineto-static duality*[11]. In fact, taking, as before, the set of joint torques with constant norm $\tau^T \tau = 1$, we obtain:

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

As already highlighted, the directions of the principal axes of the *force manipulability ellipsoid* coincide are the same of the *velocity manipulability ellipsoid*, while their dimensions are *inverse proportional*. A clear example is the following:

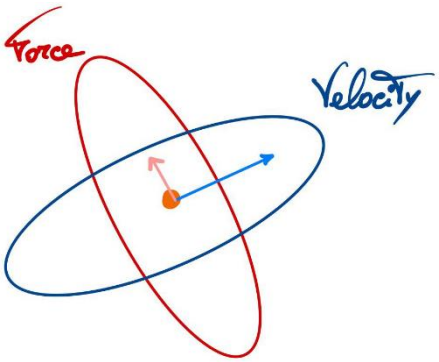

Figure 3.1: Force and Velocity Manipulability Ellipsoids

3.3 FORCE AND VELOCITY MANIPULABILITY ELLIPSOIDS FOR SCARA

In this section, the results obtained for the SCARA robot will be presented. Numerous tests have been conducted to demonstrate the robot's behaviour in terms of velocity and force in various configurations. For the reasons previously mentioned, only the first two joints of the manipulator have been considered (in the configurations, all joints are included for completeness). Additionally, the results have been obtained both using Peter Corke's Robotics

[11] There is a cross relationship (the so-called St. Andrew's Cross of the Scottish flag) due to the principle of virtual work. It is easy to demonstrate that $\tau = J^T(q)\gamma_e$, where γ_e is a $rx1$ vector and it represents the end-effector forces, and τ is a vector $nx1$, which represents the equivalent torques at joint (τ is a force if the joint is a prismatic joint or a torque for the revolute one

Toolbox[12] and through a dedicated function included in the MATLAB script *ellipsoids.m* (only the second one method is shown in the following).

Now the results.

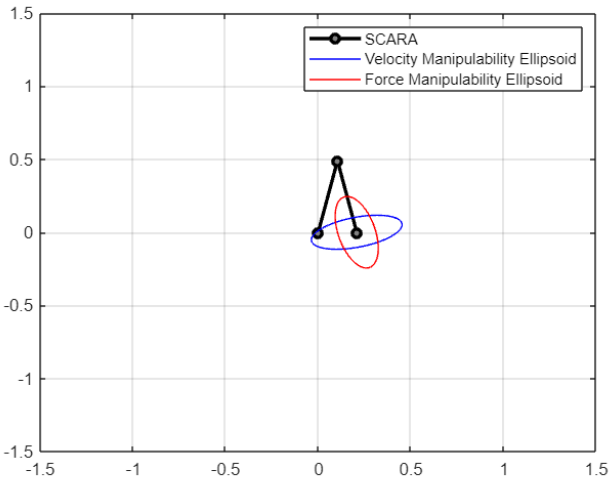

Figure 3.2: Manipulability Ellipsoids for $q = \begin{bmatrix} \frac{1.3\pi}{3} \\ -\frac{2.6\pi}{3} \\ 0.5 \\ 0 \end{bmatrix}$

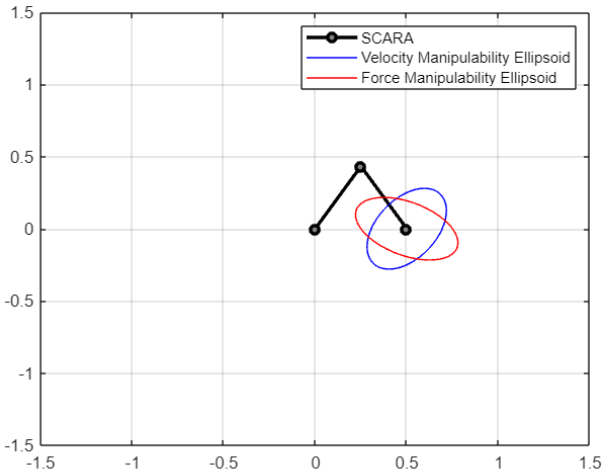

Figure 3.3: Manipulability Ellipsoids for $q = \begin{bmatrix} \frac{\pi}{3} \\ -\frac{2\pi}{3} \\ 0.5 \\ 0 \end{bmatrix}$

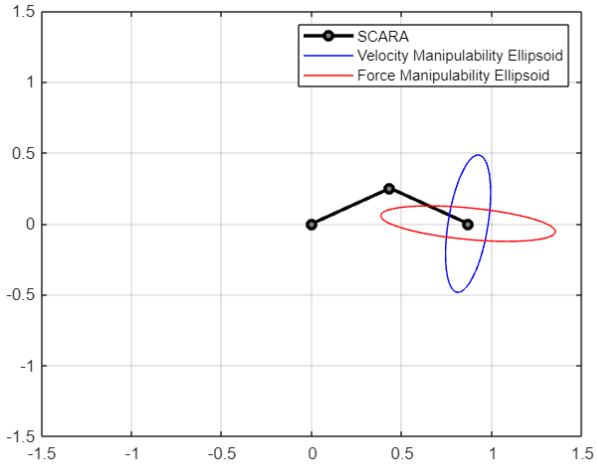

Figure 3.4: Manipulability Ellipsoids for $q = \begin{bmatrix} \frac{\pi}{6} \\ -\frac{\pi}{3} \\ 0.8 \\ 0 \end{bmatrix}$

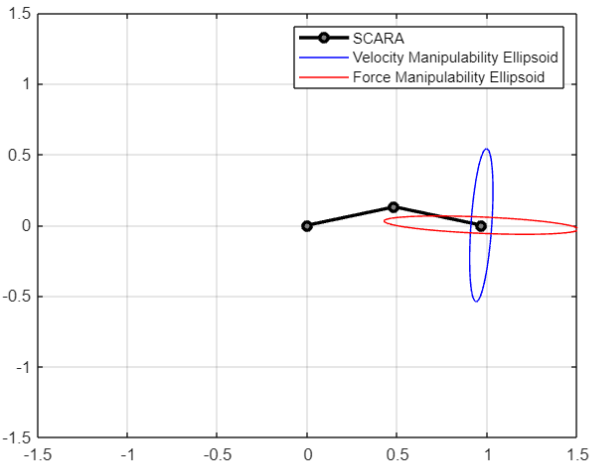

Figure 3.5: Manipulability Ellipsoids for $q = \begin{bmatrix} \frac{\pi}{12} \\ -\frac{\pi}{6} \\ 0.8 \\ 0 \end{bmatrix}$

[12] It is useful to remember that is necessary, for this toolbox, to do as first instruction the line startup_rvc

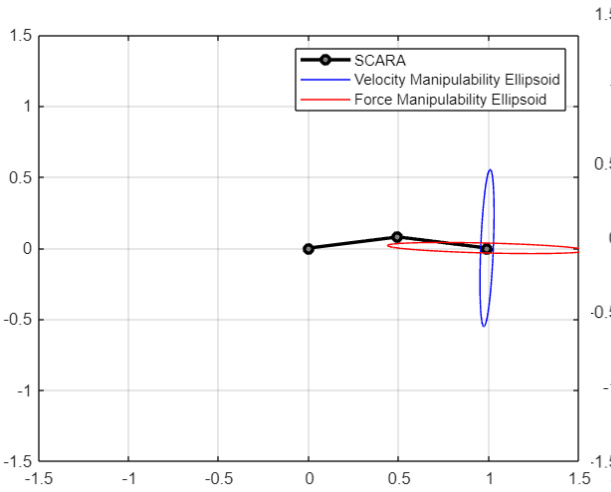

Figure 3.6: Manipulability Ellipsoids for $q = \begin{bmatrix} \frac{\pi}{20} \\ -\frac{\pi}{10} \\ 0.8 \\ 0 \end{bmatrix}$

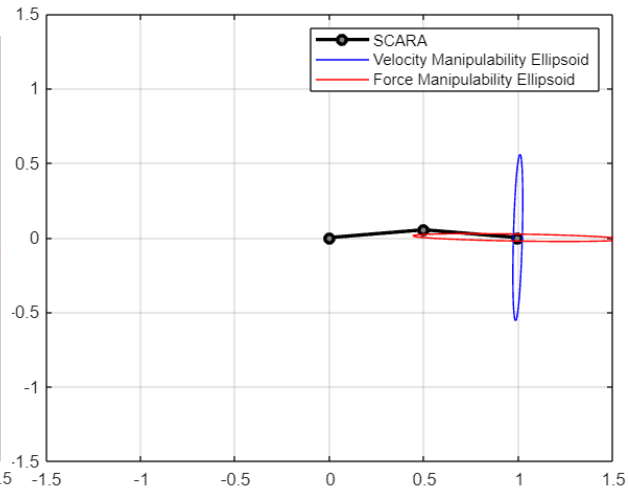

Figure 3.7: Manipulability Ellipsoids for $q = \begin{bmatrix} \frac{\pi}{30} \\ -\frac{\pi}{15} \\ 0.7 \\ 0 \end{bmatrix}$

Additionally, for each configuration, the corresponding *manipulability measures* were calculated using the script *manipulability_measure.m*, using the formula provided in the previous section.

$$w_1 = 0.1017 \quad w_2 = 0.2165 \quad w_3 = 0.2165 \quad w_4 = 0.1250 \quad w_5 = 0.0773 \quad w_6 = 0.0520$$

We can notice that the manipulability measure is bigger when the manipulator is in a medium configuration.

4 TRAJECTORY PLANNING

In the field of robotics, *trajectory planning* is of great importance and interest. As it is easy to imagine, to perform any *task* with a robot, it is necessary to provide inputs that allow it to follow a specific trajectory in space. In fact, the goal of trajectory planning is to generate the *reference inputs* to the *motion control* system which ensures that the manipulator executes the planned trajectories. The classical scheme that shows this is the following:

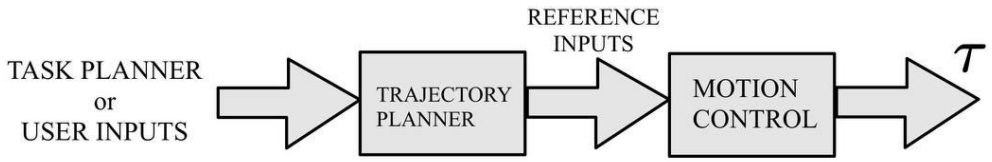

Figure 4.1: General Scheme for Trajectory Planning and Motion Control

In the case under consideration, the design specifications required to plan the trajectory along a *path* characterized by at least *20 points* within the *workspace*, in which there are at least one *straight portion* and one *circular portion* and, also, the passage for at least 3 *via point*[13]. In the following table the *points* used for the trajectory planning are shown.

Table 4.1: Trajectory Points

Point	x	y	z
1	0.1	-0.5	0
2 (VIA POINT)	0.3	-0.6	0
3 (CIRCULAR)	0.4	-0.1	0
4	0.7	-0.1	0
5 (VIA POINT)	0.7	-0.4	0
6	0.6	-0.5	0.4
7 (VIA POINT)	0.75	0	0.5
8	0.7	0.4	0.3
9(VIA POINT)	0.4	0.5	0.35
10	0.3	0.2	0.5
11(VIA POINT)	0.15	-0.1	0.7
12	0	-0.4	0.6
13(VIA POINT)	0.1	-0.45	0.55
14	0.05	-0.3	0.45
15(VIA POINT)	0	-0.15	0.4
16	0.1	0	0.5
17 (VIA POINT)	0.4	0.1	0.4
18	0.5	0	0.25
19	0.25	-0.1	0.15
20 (VIA POINT)	0	-0.25	0.1
21	0.1	-0.5	0

[13] The via point is a point in the trajectory where the robot does not pass exactly through it. In this case we refer to a trajectory with interpolating linear polynomials with parabolic blends

Using these points, the *3D trajectory* is the following:

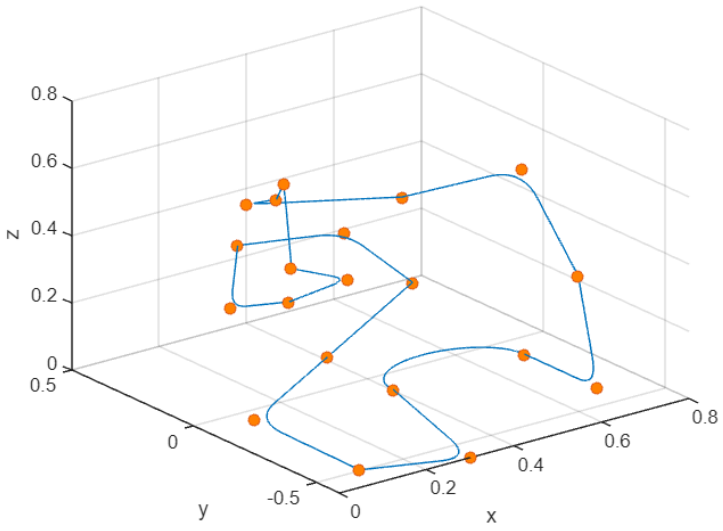

Figure 4.2: Trajectory in the Operational Space

In this example, a *trapezoidal velocity profile*[14] is assigned, which imposes a *constant acceleration* in the start phase, a *cruise velocity*, and a *constant deceleration* in the arrival phase (one for each couple of points). Additionally, the problem is formulated by assuming that the final time for each segment is assigned and it is assumed to be 2 s with a total final time $t_f = 40\ s$. Instead, for the Via Points, the time $\Delta t'$ is posed equal to 0.7 s. So, at the end of the day, it is possible to show the *time history* of *position*, *velocity* and *acceleration* associated with the desired trajectory planned.

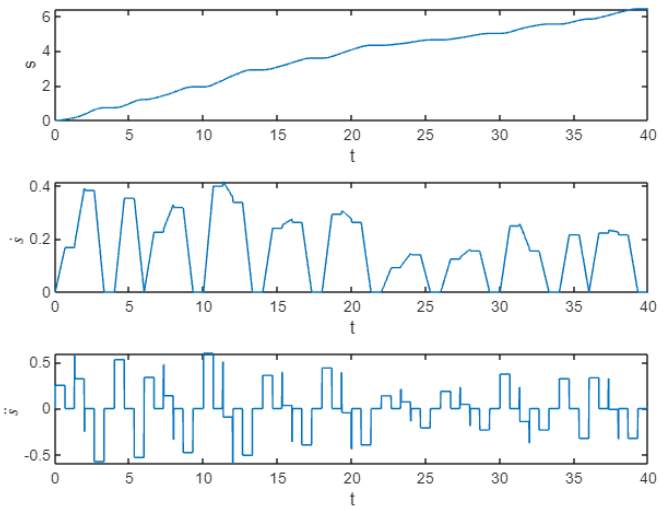

Figure 4.3: Characterization of the timing law in terms of position, velocity and acceleration

[14] This type of velocity profile is widely used in industry. This is because it allows for immediate verification of whether the velocities and accelerations imposed by such motion laws are consistent with the physical characteristics of the mechanical manipulator.

It is possible to do the same with the *orientation*. An orientation pattern of the following type has been imposed: $\phi = [0 \quad \frac{\pi}{2} \quad 0 \quad \frac{\pi}{2} \quad 0]$. In this way, the resulting time law is:

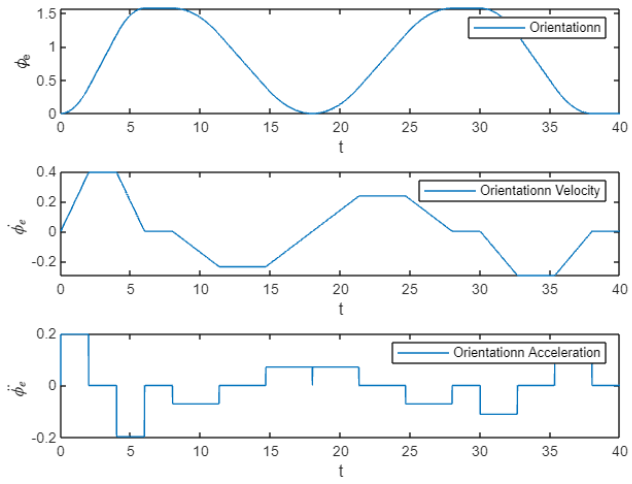

Figure 4.4: Characterization of the timing law in terms of orientation, orientation velocity and orientation acceleration

5 INVERSE KINEMATICS ALGORITHMS

As mentioned in Section 2.1, the *Inverse Kinematics Problem* can be very tedious and difficult to solve, except for geometrically simple structures for that, with *geometrical* and *analytical intuition*, it is possible to find a solution to this problem. The difficult also arises from the *non-linear mapping* between the position in the Operational Space and the position in the Joint Space. For this reason, it is useful to use the *Differential Kinematics Equation* in such a way to obtain a *linear mapping* between the velocities in the two spaces. All of this is possible thanks to the *Jacobian* which encapsulates all the nonlinearities of the model within it[15]. Doing this, it is possible to determine the *desired position* of the joints through numerical integration techniques such as *Euler integration* ($q(t_{k+1}) = q(t_k) + \dot{q}(t_k)\Delta t$). However, these solutions can lead to *Drift Phenomena* because in each interval, it is as if something is lost and, at the end of the simulation, the error between the desired final position and the real position obtained could be very large. This problem can be overcome through a solution scheme that considers the *Operational Space Error* between reached end-effector pose and the desired one thanks to a loop. Such error will be:

$$e = x_d - x_e$$

$$\dot{e} = \dot{x}_d - \dot{x}_e = \dot{x}_d - J_A(q)\dot{q}$$

The choice of $\dot{q}$ as a function of e permits finding *Inverse Kinematics Algorithms* with different features. However, these types of algorithms are called *CLIK Algorithms*, where *CLIK* stands for *Closed-Loop Inverse Kinematics*.

In the following, there will be shown different types of CLIK control scheme:

1. *CLIK Algorithm with Jacobian Inverse*
2. *CLIK Algorithm with Jacobian Transpose*
3. *CLIK Algorithm with Jacobian Pseudo-Inverse*
4. *2nd-Order CLIK Algorithm*

For all of this, the objective of the CLIK Algorithm is to find $\dot{q}(e) \rightarrow e = 0$. We will now proceed with the development of all these algorithms.

[15] This is one of the reasons why the Jacobian is so important in Robotics

5.1 CLIK ALGORITHM WITH JACOBIAN INVERSE

To implement this type of algorithm, it is necessary for the *Jacobian* to be *square* and *nonsingular*. The choice

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

leads to a *linearization* of the error dynamics such that

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

where, usually, K is a positive definite and diagonal matrix. Thanks to this choice, the error at the steady-state is zero. As will be clearly visible from the Simulink scheme in Figure 5.1, there are a *Feedback Action* using the Direct Kinematics Function and a *Feedforward Action* provided by $\dot{x}_d$ for a time-varying reference which ensures that the error is kept to zero (if $e(0) = 0$) along the whole trajectory, whatever $x_d(t)$ may be.

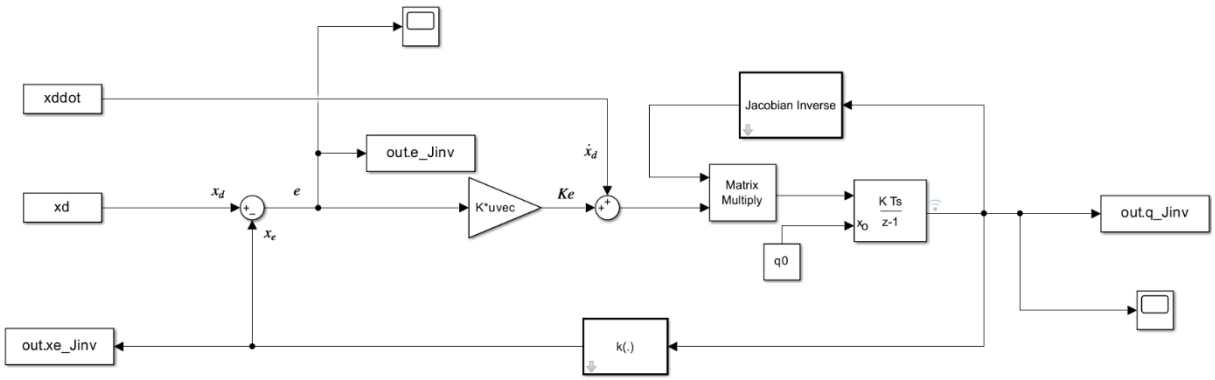

Figure 5.1: CLIK Algorithm with Jacobian Inverse Simulink Scheme

The matrix $K_{J_{inv}}$ used for the simulation was:

$$K_{J_{inv}} = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 50 \end{bmatrix}$$

For the calculation of q, the forward *Euler Integration Technique* was used with a sampling time of 1 ms and an externally assigned initial condition.

The various results obtained from the simulations conducted in MATLAB will now be presented. Additionally, to highlight the zero steady-state error and its *maintenance*,

simulations lasting 50 s were conducted, which exceed the time required to complete the trajectory.

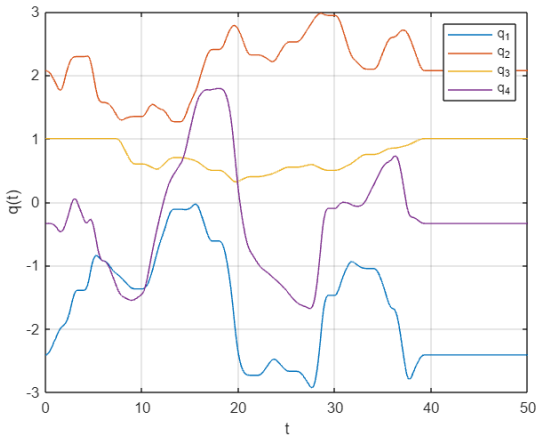

Figure 5.2: Joint Position Variables with Jacobian Inverse

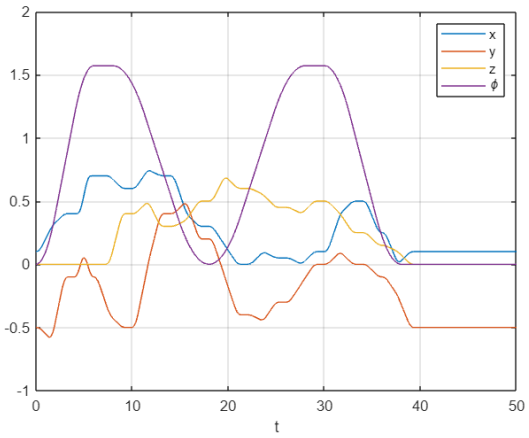

Figure 5.3: Operational Space Variables with Jacobian Inverse

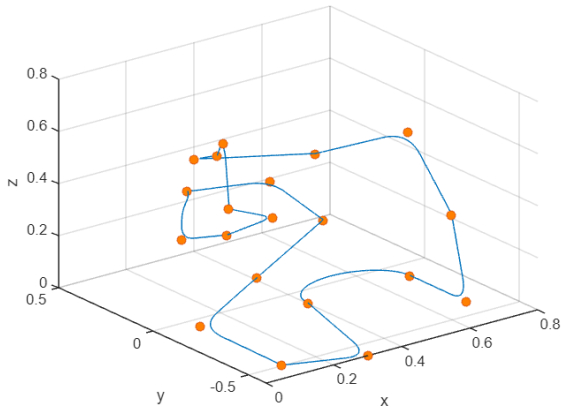

Figure 5.4: Trajectory with Jacobian Inverse

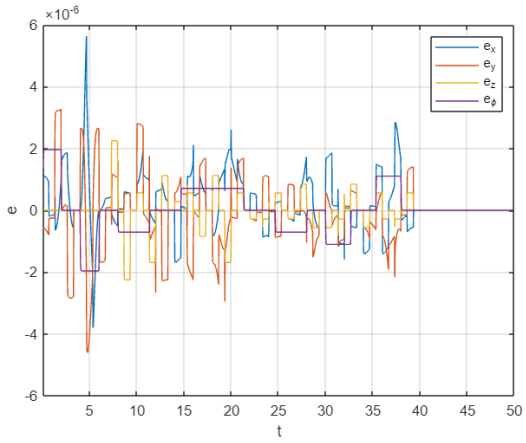

Figure 5.5: Position and Orientation Error with Jacobian Inverse

As can be seen in Figure 5.5, the maximum error is on the order of 10^{-6}, indicating that this algorithm allows for very accurate result.

5.2 CLIK ALGORITHM WITH JACOBIAN TRANSPOSE

With the *Jacobian Transpose*, it is possible to obtain a *computationally simpler algorithm* without having the clean error dynamics obtained with the linearization introduced in the previous section. However, the increased computational simplicity comes at the cost of precision during the simulation, as will be evident from the following figures. With the *Lyapunov Method*, it is easy to identify that the choice

$$\dot{q} = J_A^T(q)Ke$$

ensures the same steady-state results obtained with the previous algorithm. The resulting block scheme in Figure 5.6 shows the notable feature of the algorithm to require computation only of direct kinematics function $k(q)$ and $J_A^T(q)$.

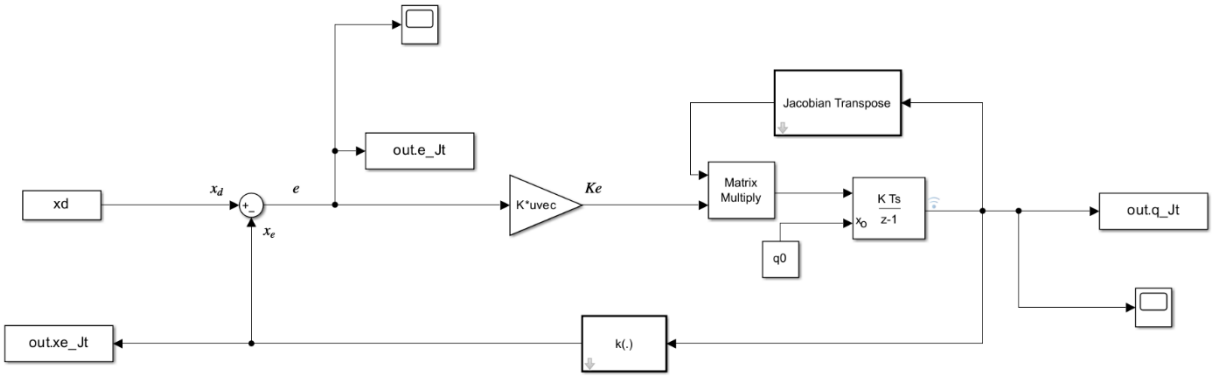

Figure 5.6: CLIK Algorithm with Jacobian Inverse Simulink Scheme

The matrix K_{J_t} used for the simulation was:

$$K_{J_t} = \begin{bmatrix} 2000 & 0 & 0 & 0 \\ 0 & 2000 & 0 & 0 \\ 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}$$

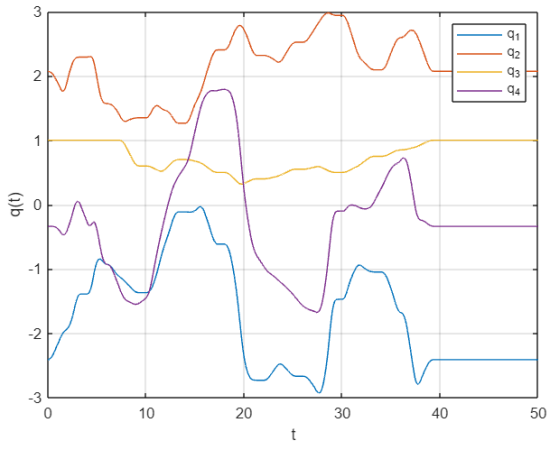

Figure 5.7: Joint Position Variables with Jacobian Transpose

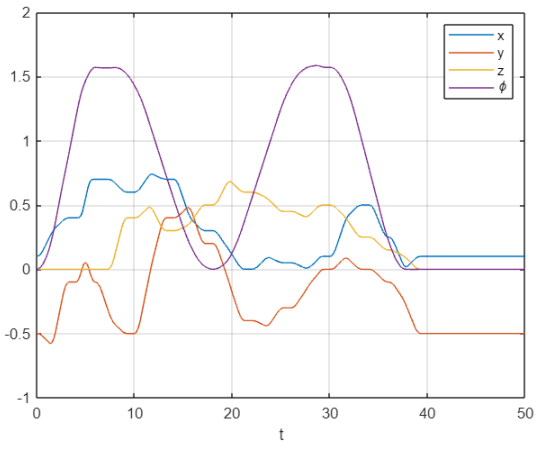

Figure 5.8: Operational Space Variables with Jacobian Transpose

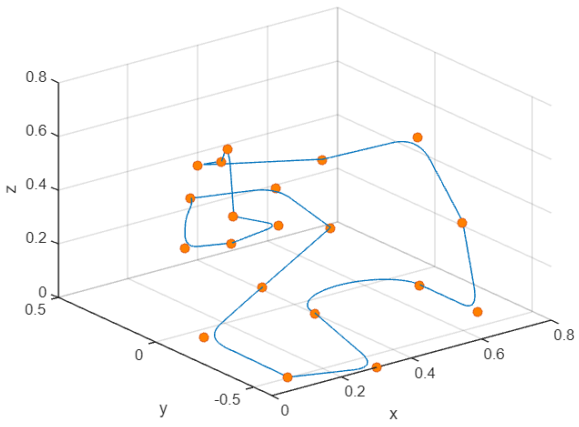

Figure 5.9: Trajectory with Jacobian Transpose

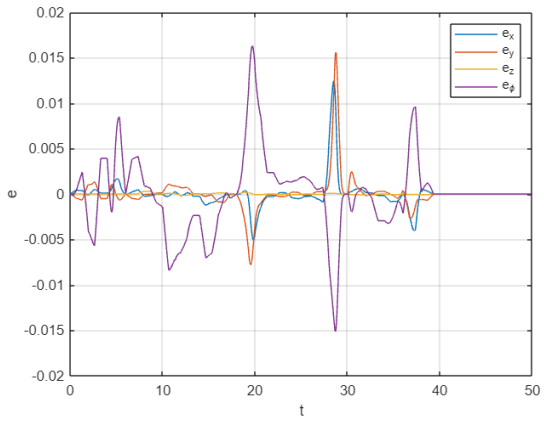

Figure 5.10: Position and Orientation Error with Jacobian Transpose

Here, the maximum error is on the order mm[16], indicating that this algorithm is *less accurate* than the previous one, but it has the significant advantage, as mentioned, of greater simplicity. So, for this algorithm, it is necessary to have a gains matrix K bigger than the matrix used for the algorithm showed in the previous section (there is still an upper limit dictated by the *sampling time*). In this way, the error during the trajectory remains limited in norm until it also becomes zero at steady-state when the trajectory is completed. This because, in this moment it is $\dot{x}_d = 0$ whatever the desired trajectory may be and so it is possible to obtain the same results at the steady-state obtained previously without inverting any matrix and avoiding *singularities*. The use of the *Jacobian Transpose* is a very *pragmatic* choice.

5.3 CLIK ALGORITHM WITH JACOBIAN PSEUDO-INVERSE

In the case of a *redundant manipulator*, the algorithm introduced in Section 5.3 can be generalized into

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

where the first term is relative to minimum norm joint velocities and the second term is projected into the *null-space* of the matrix which attempts to satisfy a *secondary additional constraint* in such a way to generate *internal motions* useful to several reasons.

[16] In many applications, it is acceptable to have errors of this magnitude.

In the case of SCARA manipulator, there isn't redundancy, but it is possible to impose it assuming to *relax an operational space component* in such a way to implement a CLIK Algorithm with *Jacobian Pseudo-Inverse* along the trajectory when optimizing a *dexterity constraint*. In this way the SCARA is now *functionally redundant*. In particular, it was imposed that the only two coordinates of interest are x and z. Doing this, only the movement on the xz plane is considered by the algorithm while the movement along y is not considered.

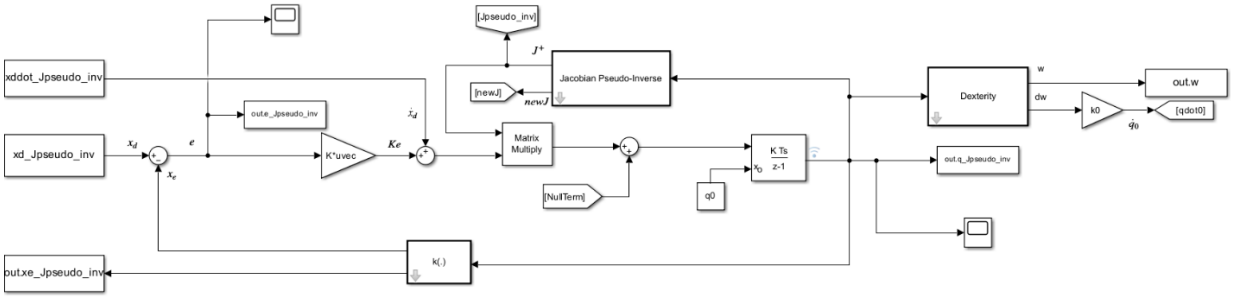

Figure 5.11: CLIK Algorithm with Jacobian Pseudo-Inverse Simulink Scheme

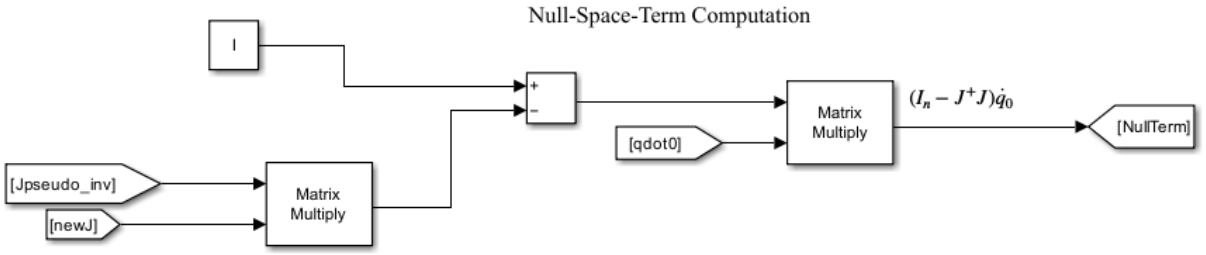

Figure 5.12: Null-Space Term Computation Simulink Scheme

The matrix $K_{J_{pinv}}$ used for the simulation was:

$$K_{J_{pinv}} = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 50 \end{bmatrix}$$

As mentioned, it is possible to exploit functional redundancy to control the internal motions of the manipulator. Let's define a simplified measure of the manipulator's dexterity:

$$w = \sin^2(\theta_2)$$

in such a way to have

$$\dot{q}_0 = k_0 \left(\frac{\partial w(q)}{\partial q} \right)^T$$

with $k_0 > 0$. In the following, it is assumed $k_0 = 10$.

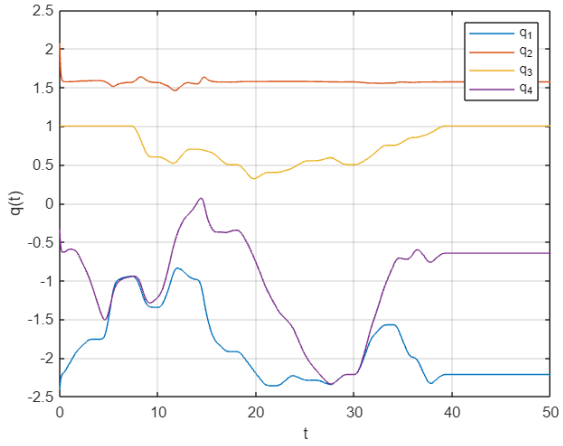

Figure 5.13: Joint Position Variables with Jacobian Pseudo-Inverse

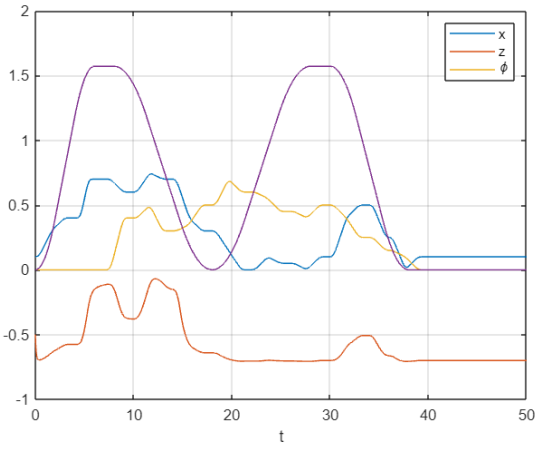

Figure 5.14: Operational Space Variables with Jacobian Pseudo-Inverse

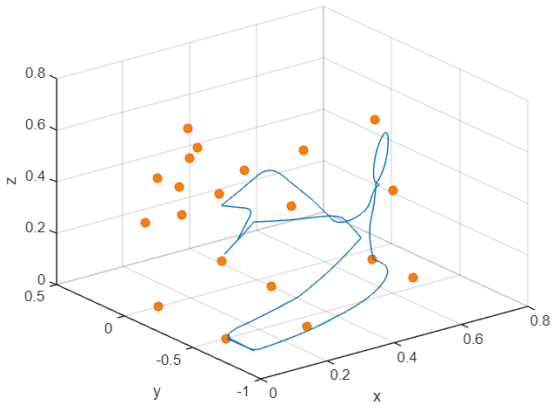

Figure 5.15: Trajectory with Jacobian Pseudo-Inverse

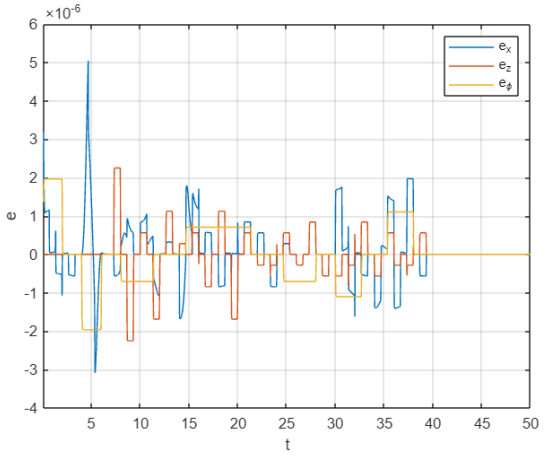

Figure 5.16: Position and Orientation Error with Jacobian Pseudo-Inverse

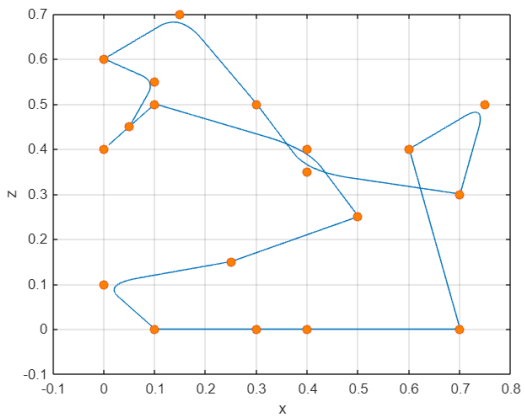

Figure 5.17: Trajectory with Jacobian Pseudo-Inverse in xz plane

As can be easily inferred, the trajectory in the xz plane is perfectly followed, while the trajectory in space is, of course, not accurate due to the assumption previously mentioned. Now it is possible to show the values of the manipulability measure obtained with this algorithm and the one with the Jacobian Inverse (Sect. 5.1). Additionally, the maximum error, as in the previous case, is on the order of 10^{-6}.

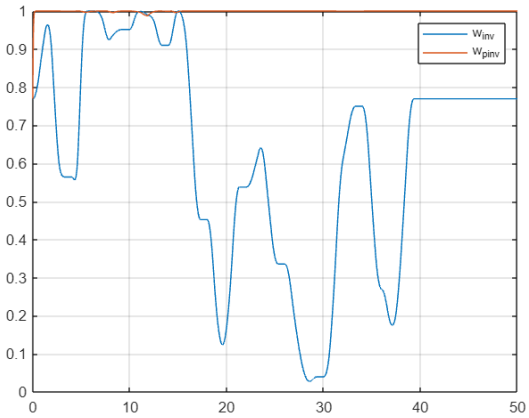

Figure 5.18: Manipulability Measure Comparison ($k_0 = 10$)

For completeness, the results obtained by setting $k_0 = 0$ are presented in the figure below.

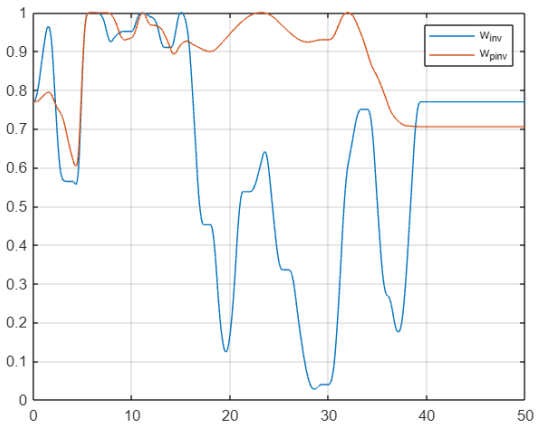

Figure 5.19: Manipulability Measure Comparison ($k_0 = 0$)

5.4 2nd-ORDER CLIK ALGORITHM

The CLIK Algorithms introduce in the previous sections can be define as *1st-order algorithms*. It is possible to define another algorithm in such a way to specify a trajectory in terms of position, velocity and, also, acceleration. It is useful not only for this but also because the manipulator, in general, is a *second-order mechanical system*. The time differentiation of the Differential Kinematics Equation leads to

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

and, under the assumption of square and non-singular matrix, it is possible to invert and to obtain the equation

$$\ddot{q} = J_A^{-1}(q)\left(\ddot{x}_e - \dot{J}_A(q,\dot{q})\dot{q}\right)$$

In this case, because of Drift Phenomena due to discretization, it is necessary to also consider the second derivative of the error:

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

It is convenient to choose the joint acceleration vector as

$$\ddot{q} = J_A^{-1}(q)(\ddot{x}_d + K_D\dot{e} + K_P e - \dot{J}_A(q,\dot{q})\dot{q})$$

with the consequential error dynamics linearization

$$\ddot{e} + K_D\dot{e} + K_P e = 0$$

which leads to a *global asymptotically stable* system if $K_D > 0$ and $K_P > 0$.

For this type of algorithm, it is impossible to consider the *Jacobian Transpose*, but it is necessary to use the *Jacobian Inverse*.

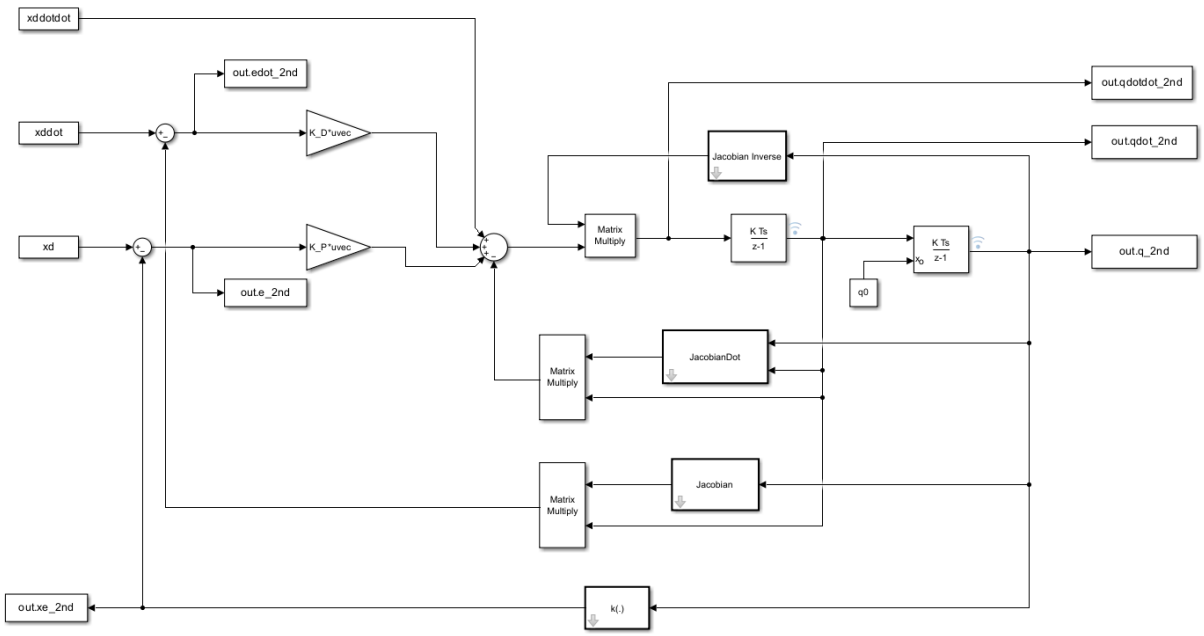

Figure 5.20: 2nd-order CLIK Algorithm Simulink Scheme

The matrices K_P and K_D used for the simulation were:

$$K_P = \begin{bmatrix} 200 & 0 & 0 & 0 \\ 0 & 200 & 0 & 0 \\ 0 & 0 & 200 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}$$

$$K_D = \begin{bmatrix} 40 & 0 & 0 & 0 \\ 0 & 40 & 0 & 0 \\ 0 & 0 & 40 & 0 \\ 0 & 0 & 0 & 40 \end{bmatrix}$$

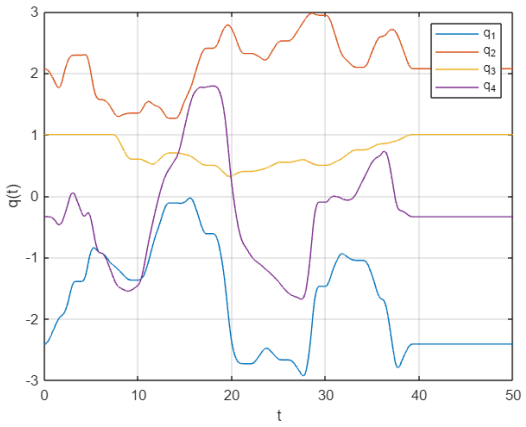

Figure 5.21: Joint Position Variables with 2nd-order CLIK

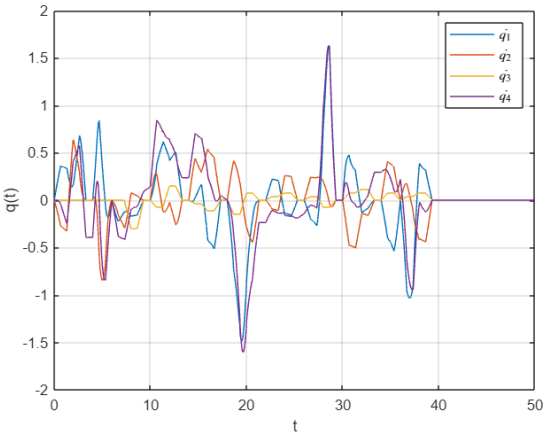

Figure 5.22: Joint Velocity Variables with 2nd-order CLIK

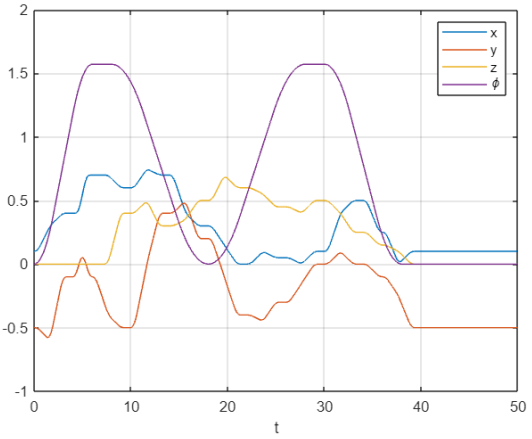

Figure 5.23: Operational Space Variables with 2nd-order CLIK

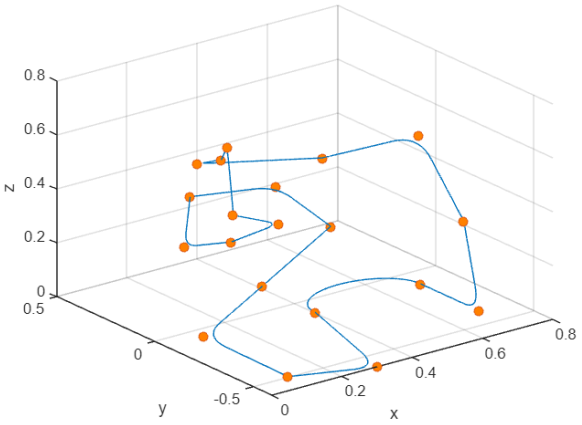

Figure 5.24: Trajectory with 2nd-order CLIK

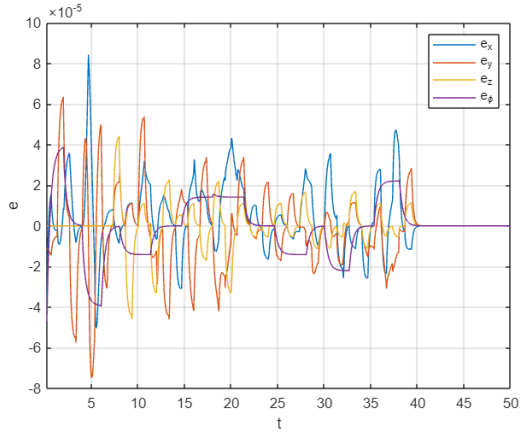

Figure 5.25: Position and Orientation Error with 2nd-order CLIK

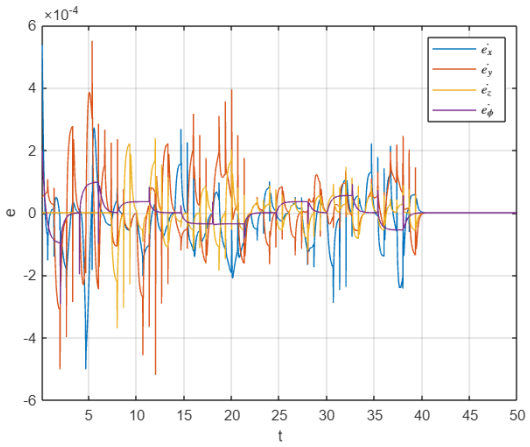

Figure 5.26: Velocity Error with 2nd-order CLIK

The trajectories generated by this algorithm will be used in the next chapter for the *Robust Control* and for the *Adaptive Control*.

6 MOTION CONTROL

In Chapter 5, trajectory planning techniques have been presented which allow the generation of the reference inputs to the motion control system. In this chapter, different control strategies in both *Joint Space* and *Operational Space* are analysed. It is possible to distinguished between *Decentralized Control Schemes* (when the single manipulator joint is controlled independently of all the others) and *Centralized Control Schemes* (when the dynamic interaction effect between joints is considered). When discussing control, it's important to distinguish between:

- *Motion Control*: the robot manipulator is moving in *Free-Space* and there is no interaction with the environment (for example, obstacles in the workspace)
- *Interaction Control*: there is an interaction between the robot and the *environment*

In this Chapter, the first one type of control is presented.

For the correct functioning of this type of control, *internal sensors*, such as encoders for joint positions and tachometers for joint velocities, are used. In Figure 6.1 and 6.2 are shown the two general scheme of *Joint Space Control* and *Operational Space Control*, useful to understand the necessary components for each of this scheme.

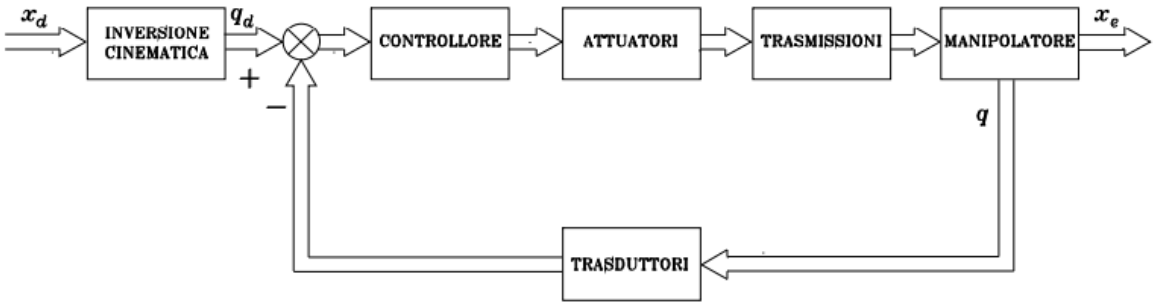

Figure 6.1: General Scheme of Joint Space Control

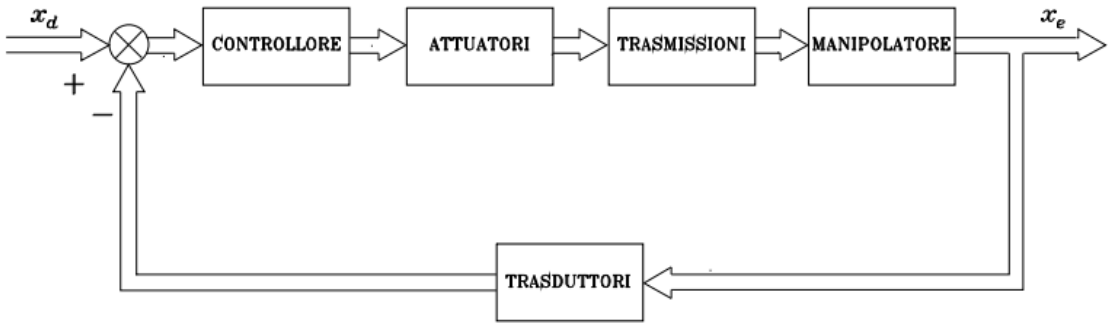

Figure 6.2: General Scheme of Operational Space Control

The *Joint Space Control Problem* is articulated in two subproblems:

1. *Manipulator Inverse Kinematics*
2. *Tracking* of the actual motion q to some reference inputs

So, we do first the *Inverse Kinematics* and then the C*ontrol*. The *Operational Space Control Problem*, instead, has a greater algorithmic complexity, since, while in the previous Joint Space Control the *Jacobian* is present in the *first stage*, here the *Jacobian* is present in the *controller*, and all is made in only one *single stage*. Additionally, the increased complexity is given also because the controller is based directly on the error between the desired end-effector trajectory and the effective one. Obviously, here the great advantage is that Operational Space variables are used, even though the measurement is not direct, since the implementation of Direct Kinematics is necessary.

So, precisely for the reasons mentioned, the primary difference between the two control schemes concerns *singularities* and *redundancy*. In the first case, it is ensured that redundancy and singularities are managed *outside the feedback loop* (due to the reasons related to the *Jacobian* discussed above), while in the second case, it is necessary to handle singularities and redundancy *inside the feedback loop*.

In this Chapter will be shown 3 control techniques considering a concentrated end-effector *payload* of about 3 Kg:

1. *A Robust Control*
2. *An Adaptive Control*
3. *An Operational Space Inverse Dynamics Control*

For all of these, it is assumed that the desired joint trajectories for the first two controllers are generated with a *2nd-Order CLIK Algorithm* which has already been shown in Sect. 5.4 and it is assumed a sampling period of 1 ms.

6.1 CENTRALIZED CONTROL

As mentioned at the beginning of the chapter, it is possible to distinguish between *Centralized Control* and *Decentralized Control*. In this section, will be shown the first one because this type of control includes *Robust Control* and *Adaptive Control* previously mentioned.

It is necessary to use these type of control schemes when large operational speeds are required, or direct drive actuation is employed. In fact, in these cases, the *nonlinear coupling* terms strongly influence system performance and the decentralized control schemes could give large tracking error. So, this leads to Centralized Control Algorithms that are based on the partial or complete knowledge of the manipulator *Dynamic Model* which is thoroughly described in Section 2.2. At this point it is necessary to find nonlinear centralized control laws (usually indicated with u) in the context of nonlinear multivariable systems.

Now, the two controls mentioned before will be introduced, but first a brief digression will be made on *Inverse Dynamics Control*.

6.1.1 INVERSE DYNAMICS CONTROL

The Dynamic Model shown in Sect. 2.2 can be rewritten as

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

where

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

are all the *nonlinear terms* of this model. The idea is to find a control vector u as a function of the system state in such a way to obtain a linear I/O relationship. Taking

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

we obtain

$$\ddot{q} = y$$

where y represent a new input vector

$$y = -K_P q - K_D\dot{q} + r$$

This leads to the to the *Globally Asymptotically Stable* system of position error

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

with $K_D > 0$ and $K_P > 0$.

Both *Robust* and *Adaptive Control* are based on *Inverse Dynamics Control* (represented in Figure 6.3)

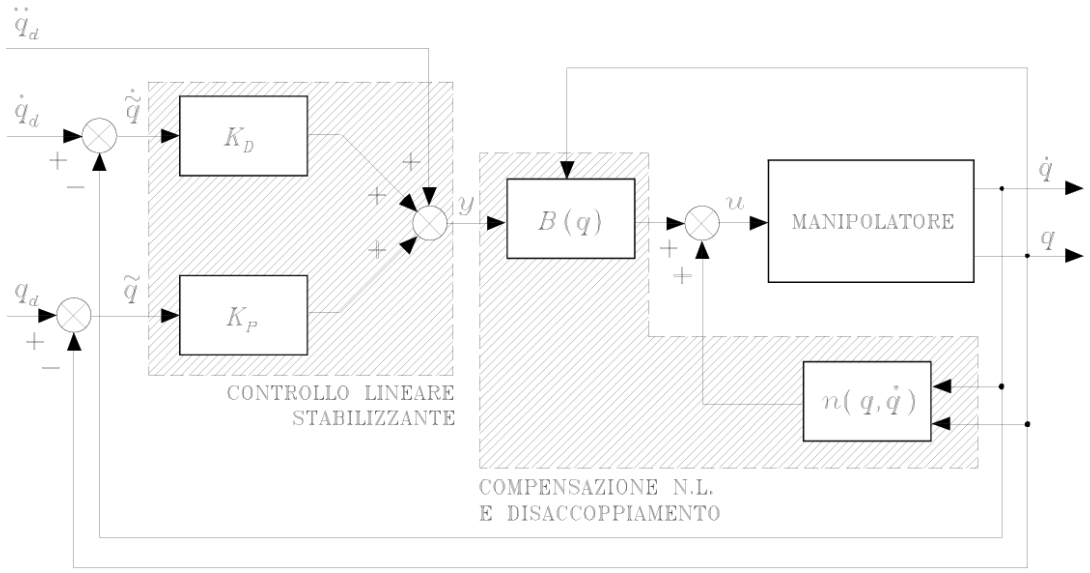

Figure 6.3: General Scheme of Joint Space Inverse Dynamics Control

6.1.2 ROBUST CONTROL

In the case of *imperfect compensation*, it is reasonable to assume a control vector expressed by

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

where $\hat{B}$ and $\hat{n}$ [17]represent the estimates of the terms in the Dynamic Model (see Sect. 2.2). The error on the estimates (uncertainty) is represented by

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

and is due to imperfect model compensation as well as to intentional simplification in Inverse Dynamics Computation. So $B\ddot{q} + n = \hat{B}y + \hat{n}$ and

$$\ddot{q} = y - \eta$$

with $\eta = \left(I - B^{-1}\hat{B}\right)y - B^{-1}\tilde{n}$.

With $y = \ddot{q}_d + K_D(\dot{q}_d - \dot{q}) + K_P(q_d - q)$ the system will be

$$\ddot{\tilde{q}} + K_D\dot{\tilde{q}} + K_P\tilde{q} = \eta$$

[17] $\hat{B}$ is chosen by considering only the diagonal terms of the inertia matrix and $\hat{n}$ is chosen neglecting the $C\dot{q}$ terms

Unlike the system shown in Sect 6.1.1, here there is η and not 0 in the right side, which could lead the system to *instability* despite the positivity of the matrices K_D and K_P. The control aims to overcome this term η. This is the essence of *Robust Control*.

Chosen

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

after all the passages using the *Lyapunov Method* with the candidate $V(\xi) = \xi^T Q\xi > 0$ with $\xi = \begin{bmatrix}\tilde{q}\\ \dot{\tilde{q}}\end{bmatrix}$ it is possible to obtain that $\dot{V} = -\xi^T P\xi + 2\xi^T QD(\eta - w)$. If $\xi \notin \mathcal{N}(D^T Q)$ the control w must be chosen to render the second term in this equation less than or equal to zero. By setting $z = D^T Q\xi$ the second term can be written as $z^T(\eta - w)$. The control

$$w = \frac{\rho}{\|z\|}z \qquad \rho > 0$$

ensures that $\dot{V}$ is less than zero along all error system trajectories. This approach has led to find a control law formed by some different contribution:

- $\hat{B}y + \hat{n}$: *approximate compensation* of nonlinear effects and joint decoupling
- $\ddot{q}_d + K_D\dot{\tilde{q}} + K_P\tilde{q}$: linear *feedforward* action and a linear *feedback* action
- $w = \frac{\rho}{\|z\|}z$: *robust contribution* for $\tilde{B}$ and $\tilde{n}$

Since $\frac{z}{\|z\|}$ is available, it is possible to refer this controller as a *Unit Vector Controller* because $\frac{z}{\|z\|}$ is precisely the unit vector of z

There is a problem. What happens when $\|z\| \to 0$? If one attempts to implement such control and if the system starts from zero error, the system *aborts*. Pragmatically, it is not possible to use this controller. So, the previous expression became:

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

in such a way to set a *threshold* because it is possible to divide by a small number but not by zero. In this latter case, the control would saturate. This control law implies that the hyperplane $z = 0$ is no longer attractive, but the error is allowed to vary within a *boundary layer* which depends on ϵ.

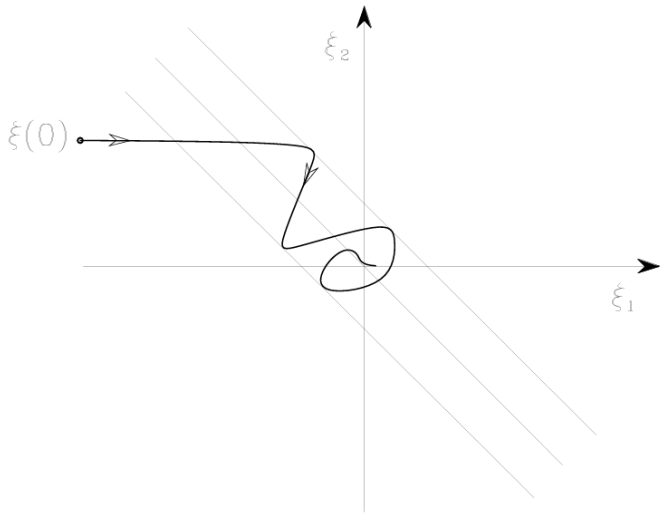

Figure 6.4: Error Trajectory with Robust Control and Chattering[18] Elimination

This is the essence of *Unit Vector Control*. It is like the controller gives a slap to the right and another to the left to the system, keeping it balanced on the desired trajectory. Referring to Figure 6.4, it is preferable for the 'tube' to be routed more from top to bottom rather than from right to left, to prioritize a *small position error* and a *large velocity error*.

Do these *oscillations* interfere with the behaviour of the robot in question? The answer is NO. This is because the concept of a *Bandwidth* comes into play. The manipulator will filter out all frequencies above a certain threshold (usually 100/200/300 Hz)[19].

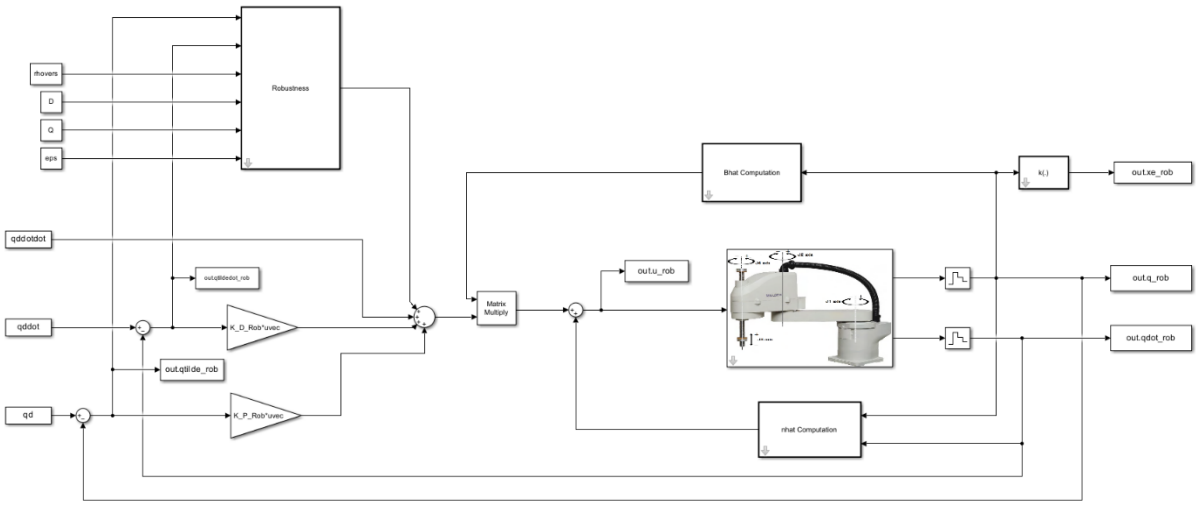

Figure 6.5: Robust Control Simulink Scheme

[18] Elimination of high-frequency oscillatory components
[19] To give a practical example, the robot is like a ferry; sudden changes in the rudder do not cause any movement of the ferry itself.

Now let's move on to the implementation of Robust Control. The parameters used for the simulation are:

$$K_{P_{Rob}} = \begin{bmatrix} 1250 & 0 & 0 & 0 \\ 0 & 1250 & 0 & 0 \\ 0 & 0 & 2000 & 0 \\ 0 & 0 & 0 & 1250 \end{bmatrix}$$

$$K_{D_{Rob}} = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 250 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}$$

$$Q = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 \end{bmatrix}$$

This choice of Q was made for the reasons just mentioned.

$$\rho = 10$$

$$\epsilon_1 = 0.01 \qquad \epsilon_2 = 0.001$$

Several tests were conducted with different values of ϵ.

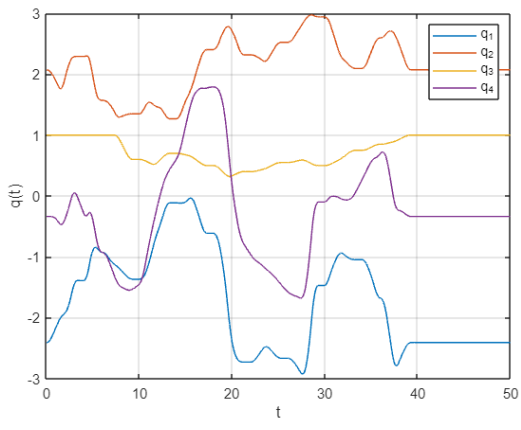

Figure 6.6: Joint Position Variables with Robust Control

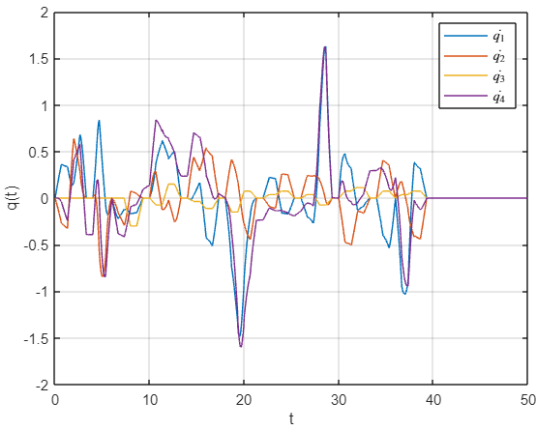

Figure 6.7: Joint Velocity Variables with Robust Control

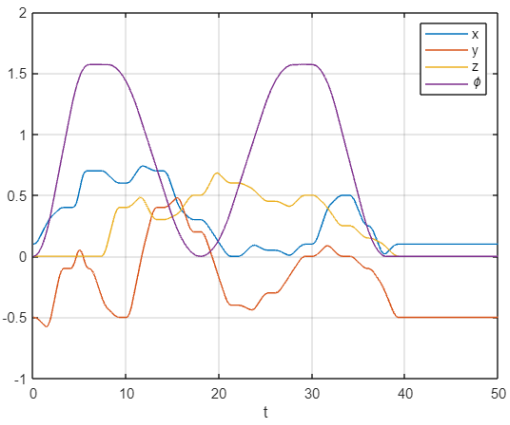

Figure 6.8: Operational Space Variables with Robust Control

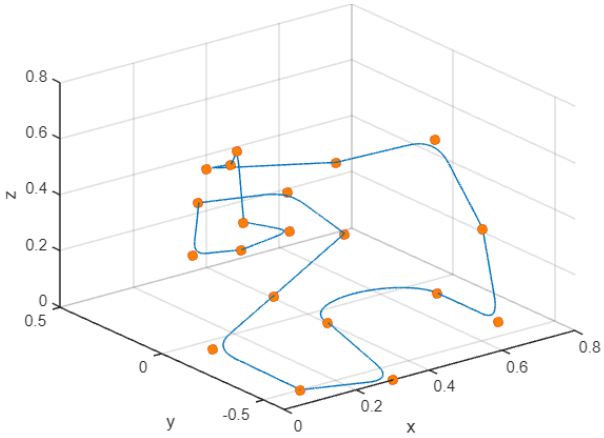

Figure 6.9: Trajectory with Robust Control

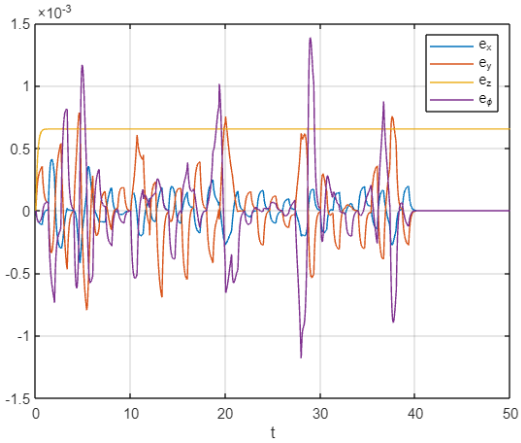

Figure 6.10: Position and Orientation Error with Robust Control ($\epsilon = 0.01$)

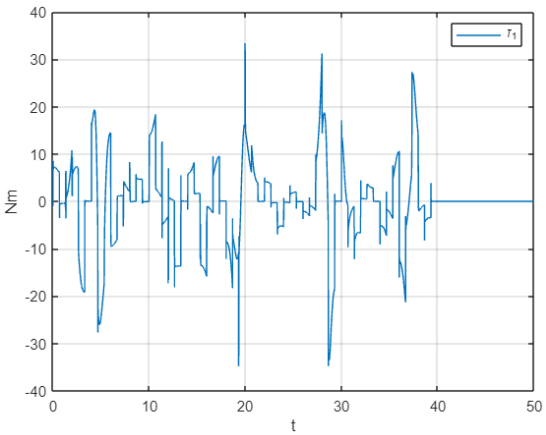

Figure 6.11: τ_1 with Robust Control ($\epsilon = 0.01$)

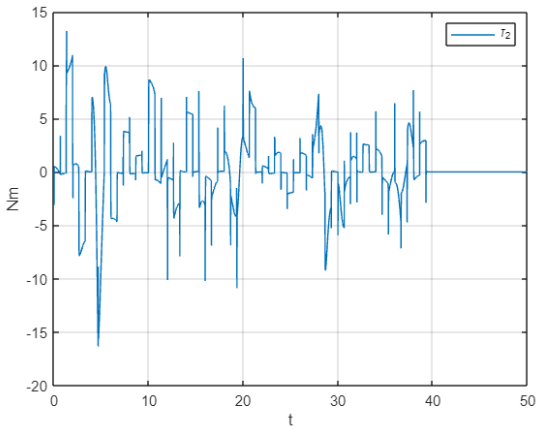

Figure 6.12: τ_2 with Robust Control ($\epsilon = 0.01$)

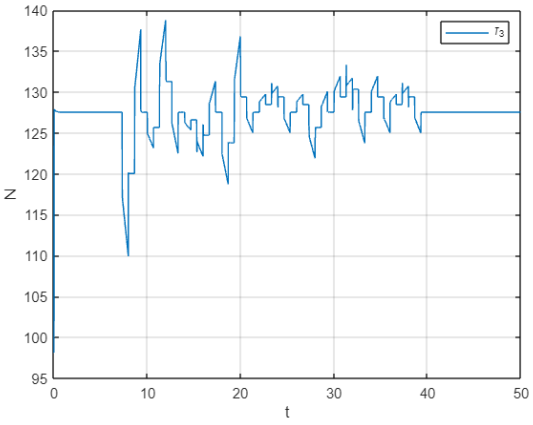

Figure 6.13: τ_3 with Robust Control ($\epsilon = 0.01$)

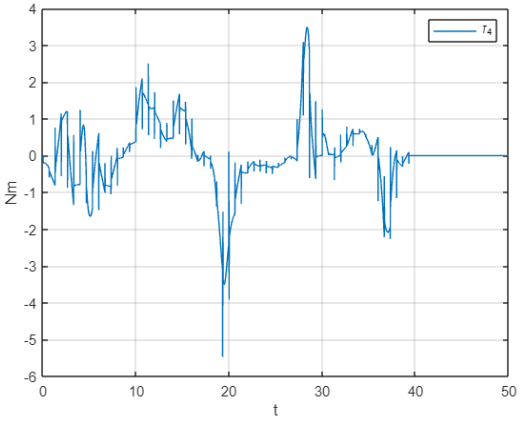

Figure 6.14: τ_4 with Robust Control ($\epsilon = 0.01$)

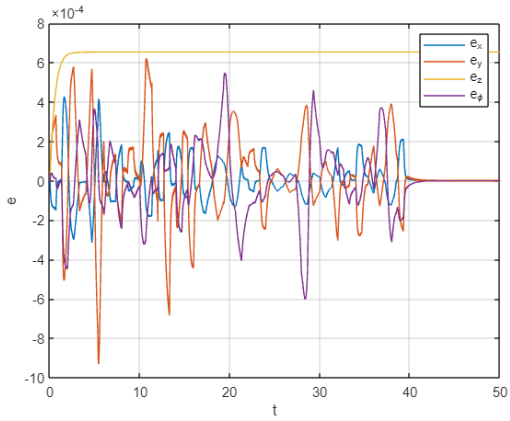

Figure 6.15: Position and Orientation Error with Robust Control ($\epsilon = 0.001$)

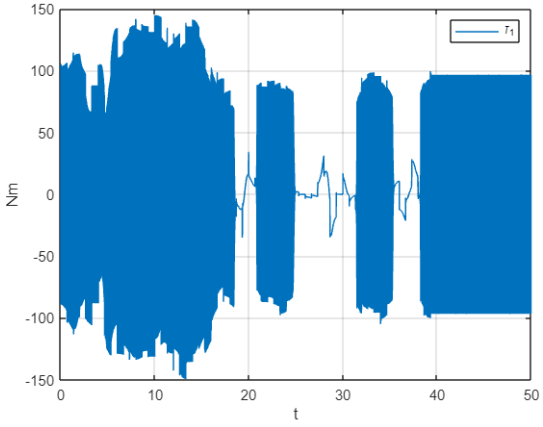

Figure 6.16: τ_1 with Robust Control ($\epsilon = 0.001$)

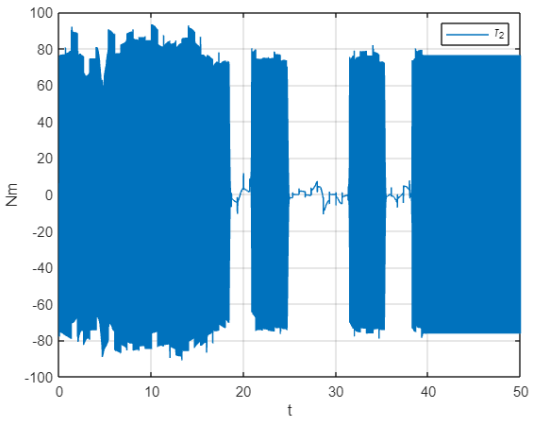

Figure 6.17: τ_2 with Robust Control ($\epsilon = 0.001$)

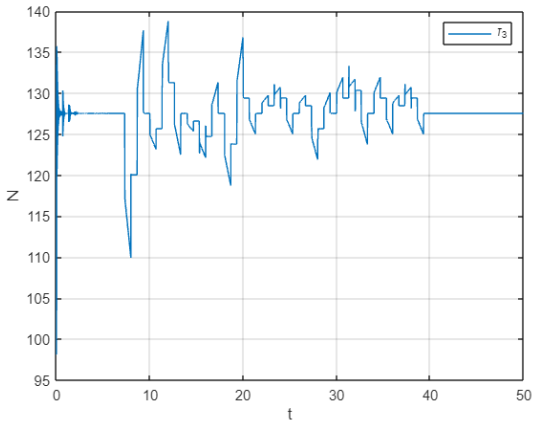

Figure 6.18: τ_3 with Robust Control ($\epsilon = 0.001$)

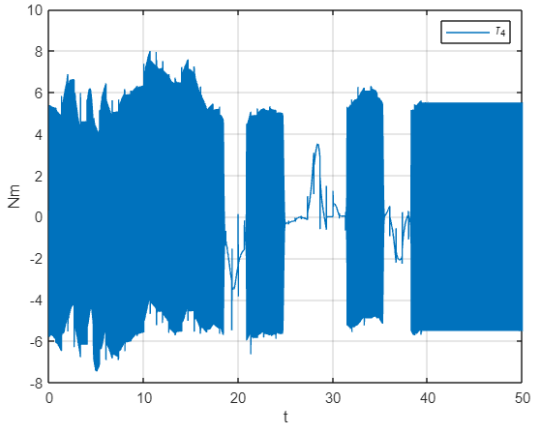

Figure 6.19: τ_4 with Robust Control ($\epsilon = 0.001$)

As can be seen from the figures above, there is a constant error in the z variable of about 0.6 mm for both cases. This because of the 3 Kg payload at the tip.

6.1.3 ADAPTIVE CONTROL

In the *Adaptive Control*, it is possible to devise solutions that allow an on-line adaptation of the computational model to the Dynamic Model. The use of this type of control is possible thanks to one of the two important properties of the Dynamic Model, in particular the property of *linearity in the parameters*. For this, it is always possible to express the nonlinear equations of motion in a linear form with respect to suitable set of constant dynamic parameters. In fact, it is possible to write the equation introduced in Sect. 2.2 in such a way to obtain

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

where Y is the *Regressor Matrix*. Consider the control law

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

with $K_D > 0$ and $\sigma = \dot{q}_r - \dot{q} = \dot{\tilde{q}} + \Lambda\tilde{q}$, where $\dot{q}_r = \dot{q}_d + \Lambda\tilde{q}$ and $\ddot{q}_r = \ddot{q}_d + \Lambda\dot{\tilde{q}}$.

After some passages, it is possible to find this equation

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

where $\tilde{\pi} = \hat{\pi} - \pi$.

With the *Lyapunov Method* and the *energy-based* candidate $V(\sigma, \tilde{q}, \tilde{\pi}) = \frac{1}{2}\sigma^T B(q)\sigma + \tilde{q}^T \Lambda K_D \tilde{q} + \frac{1}{2}\tilde{\pi}^T K_\pi \tilde{\pi} > 0 \quad \forall \sigma, \tilde{q}, \tilde{\pi} \neq 0$, it is possible to demonstrate the *Global Asymptotic Stability* of the system below. So, at this point, the control law is modified into

$$u = Y(q, \dot{q}, \dot{q}_r, \ddot{q}_r)\hat{\pi} + K_D(\dot{\tilde{q}} + \Lambda\tilde{q})$$

and the *parameter adaptive law*

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

globally asymptotically converge to $\sigma = 0$ and $\tilde{q} = 0$, which implies convergence to zero of $\tilde{q}, \dot{\tilde{q}}$, and boundedness of $\hat{\pi}$. Asymptotically it is

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

It is important to remark that this equation does not imply that $\hat{\pi}$ tends to π but it is guaranteed only the *convergence* of parameters to their true values *depending* on the structure of the matrix Y. It is like the controller is adjusting the parameters and is setting them to values that are not the true parameters but ensure that the tracking error is zero. For sure, Y is a low rectangular matrix and therefore it always has a Null-Space.

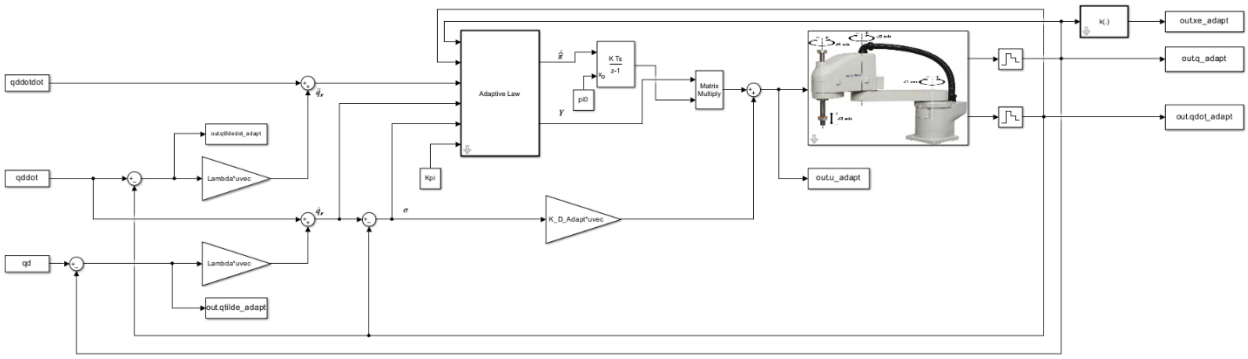

Figure 6.20: Adaptive Control Simulink Scheme

Now let's move on to the implementation of Adaptive Control. The parameters used for the simulation are:

$$\Lambda = \begin{bmatrix} 80 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 50 \end{bmatrix}$$

Considering the vector

$$\pi = [m_{l1}\ I_{l1}\ I_{m1}\ F_{m1}\ m_{l2}\ I_{l2}\ I_{m2}\ F_{m2}\ m_{l3}\ I_{m3}\ F_{m3}\ I_{l4}\ I_{m4}\ F_{m4}]^T$$

the matrix K_π is defined as

$$K_\pi = diag([1, 1, 1, 1, 1, 1, 1, 1, 0.001, 1, 1, 0.001, 1, 1])$$

The parameters of this matrix were chosen with a specific criterion. When the robot picks up the payload, the parameters related to the last link vary significantly. Therefore, assuming that the *mass* and the *inertia* of the last link change (m_{l3} and I_{l4}), it doesn't make sense to adapt all the other good parameter. Thus, it is possible to assume that the gain is unitary for all of these, while for the two terms that vary due to the payload, is assigned a value several orders of magnitude lower. This approach represents an *Anisotropic Choice* of the matrix K_π.

Then there is

$$K_{D\,Adapt} = \begin{bmatrix} 1500 & 0 & 0 & 0 \\ 0 & 1500 & 0 & 0 \\ 0 & 0 & 1500 & 0 \\ 0 & 0 & 0 & 1500 \end{bmatrix}$$

Now the result.

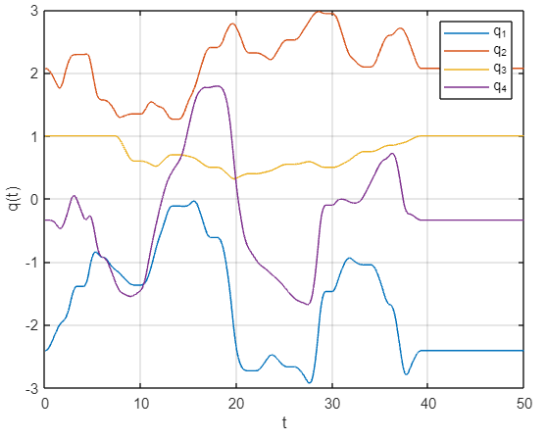

Figure 6.21: Joint Position Variables with Adaptive Control

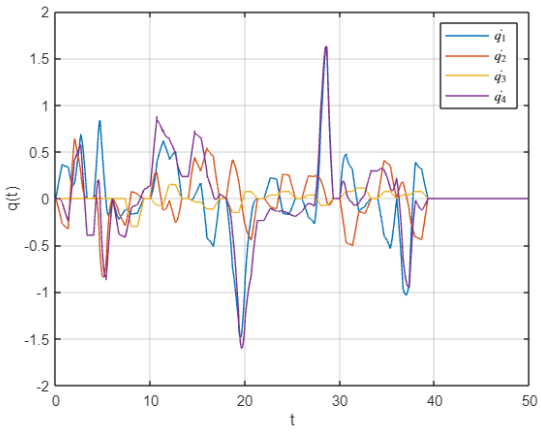

Figure 6.22: Joint Velocity Variables with Adaptive Control

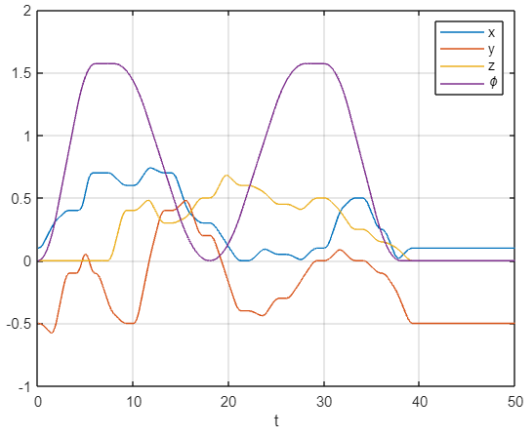

Figure 6.23: Operational Space Variables with Adaptive Control

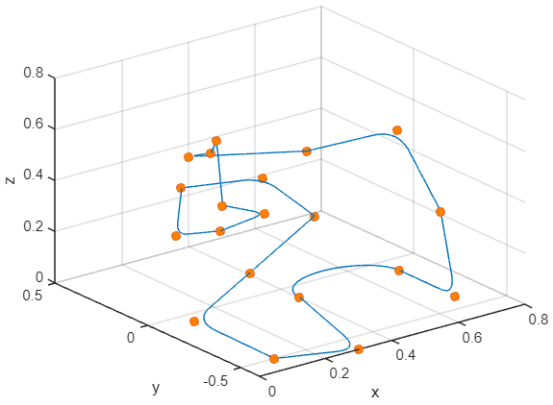

Figure 6.24: Trajectory with Adaptive Control

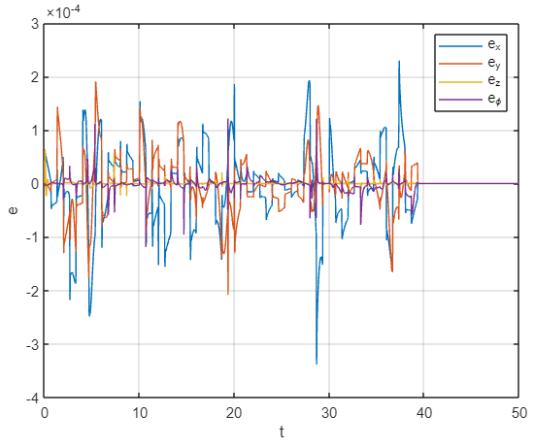

Figure 6.25: Position and Orientation Error with Adaptive Control

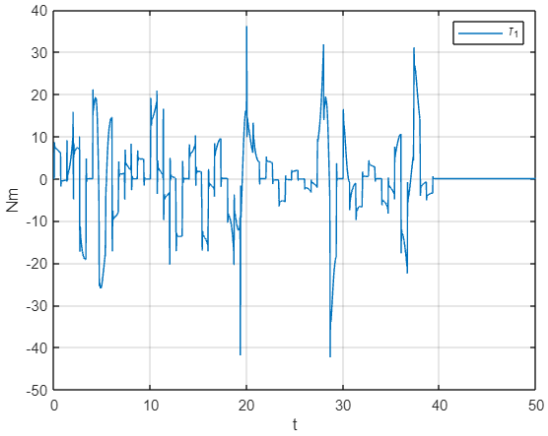

Figure 6.26: τ_1 with Adaptive Control

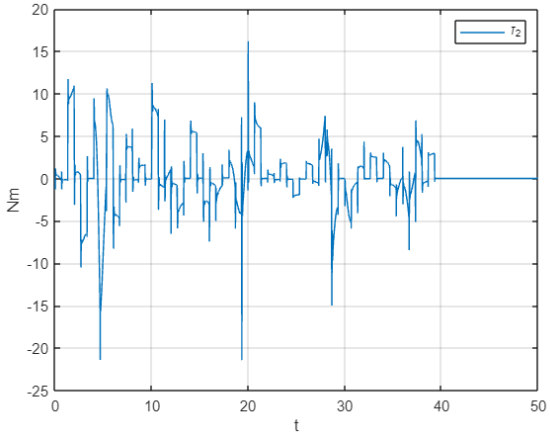

Figure 6.27: τ_2 with Adaptive Control

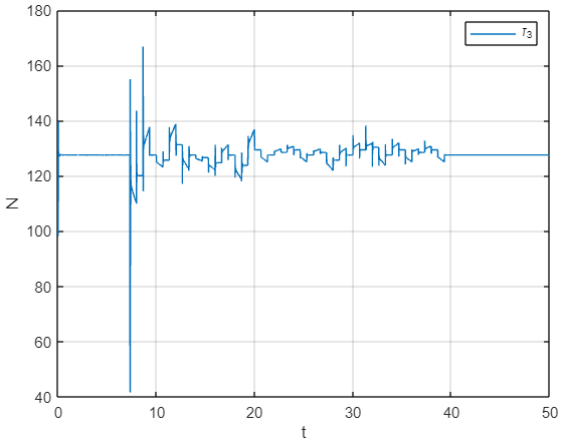

Figure 6.28: τ_3 with Adaptive Control

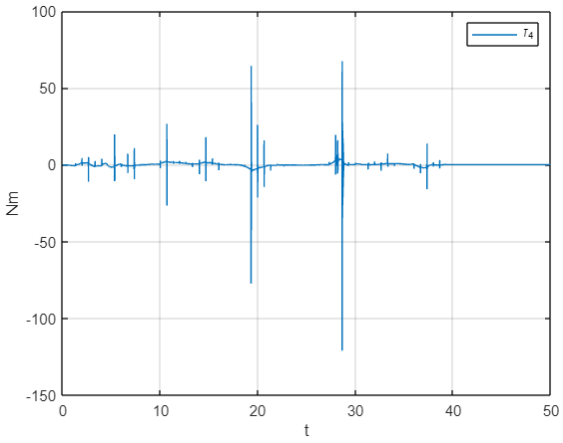

Figure 6.29: τ_4 with Adaptive Control

In this case, the error is on the order of 10^{-4} with a steady-state error of zero.

6.2 OPERATION SPACE CONTROL

In the previous controls described in Sect. 6.1, it was always assumed that the desired trajectory is available in terms of joint position, velocity and acceleration and, for this reason, also the error was expressed in the Joint Space. However, as it is easy to imagine, motion specifications are usually assigned in the Operational Space, and therefore an Inverse Kinematics Algorithm is necessary to transform the operational space references into the corresponding joint space references. This is because it is easier for us to visualize references in the Operational Space rather than in the Joint Space. Therefore, based on what has been said, it is possible to develop control schemes directly in the Operational Space in such a way as to transform the joint space variables into the corresponding operational space variables through *Direct Kinematics* relations. It is very useful and necessary when there is an *interaction* between the manipulator and the environment because a joint space control can be utilized only for *free-motion* (it is difficult to determine an obstacle in the Joint Space). The essence of this type of controller is to avoid the two-stage controller seen in the previous section. Here, it is impossible to implement a Decentralized Control like in the Joint Space because it does not make sense to design a controller for each component of x. To solve this problem, we resort, as always, to the use of the *Jacobian*, combined with a sort of *mechanical intuition*. So, the key idea is to pass from Δq to Δx. We talk about mechanical intuition because u is a torque which is proportional to Δq through a gain K. Mechanically, $K\Delta q$ is an *elastic force* for each joint that pulls q to q_d in such a way to drive $\Delta q \to 0$. Now it is possible to do the same for an Operational Space Control transforming Δx into Δq. It is also possible

to have control schemes that use the *Jacobian Transpose* (these will not be shown in the following sections). Here it is assumed to have an *elastic force* $K\Delta x$ which pulls x_e to x_d and then to obtain $\Delta x \to 0$. With the use of the *Kineto-Static Duality* and remembering that γ is a force, it is possible to use the Jacobian Transpose J_A^T in such a way to transform the end-effector forces into the equivalent joint torques. Both are conceptual schemes shown in Figure 6.30 and 6.31.

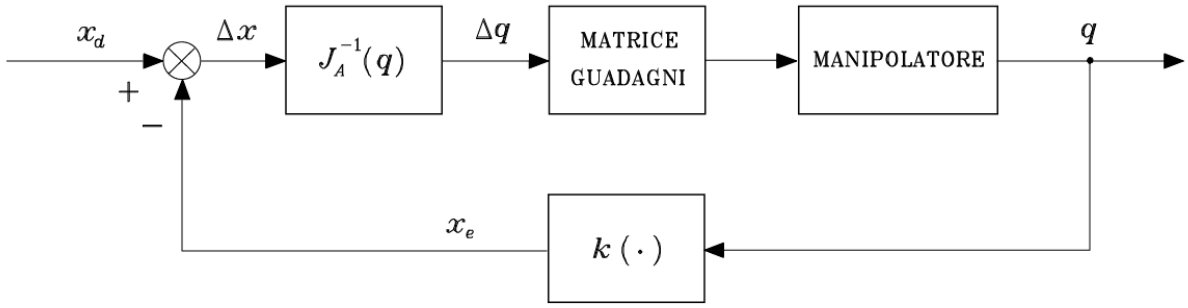

Figure 6.30: Block Scheme of Jacobian Inverse Control

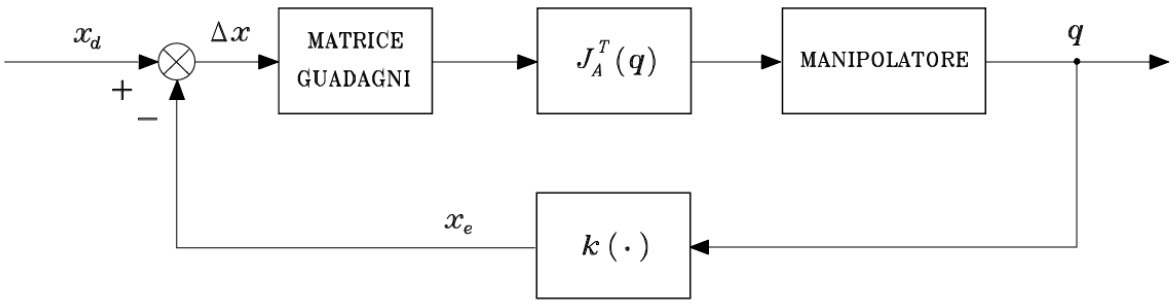

Figure 6.31: Block Scheme of Jacobian Transpose Control

6.2.1 INVERSE DYNAMICS CONTROL

In the Inverse Dynamics Control, using the manipulator Dynamic Model in the form $B(q)\ddot{q} + n(q,\dot{q}) = u$ the choice of the Inverse Dynamics *Linearizing Control*

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

leads to the system of *double integrators*

$$\ddot{q} = y$$

Using the second-order differential equation $\ddot{x}_e = J_A(q)\ddot{q} + \dot{J}_A(q,\dot{q})\dot{q}$ with the choice of the control law

$$y = J_A^{-1}(q)(\ddot{x}_d + K_D\dot{\tilde{x}} + K_P\tilde{x} - \dot{J}_A(q,\dot{q})\dot{q})$$

it is possible to obtain

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

which describes the *Operational Space Error Dynamics*. Because of the necessity to compute the *Jacobian*, it is more difficult to control a manipulator in the Operational Space then in the Joint Space, for the study of *singularities* and *redundancy*. In fact, if a singularity occurs, there could be numerous problems, such as not reaching of the desired position.

Finally, leaving that aside, an *integral action* to recover the steady-state error due to the uncompensated load has been used.

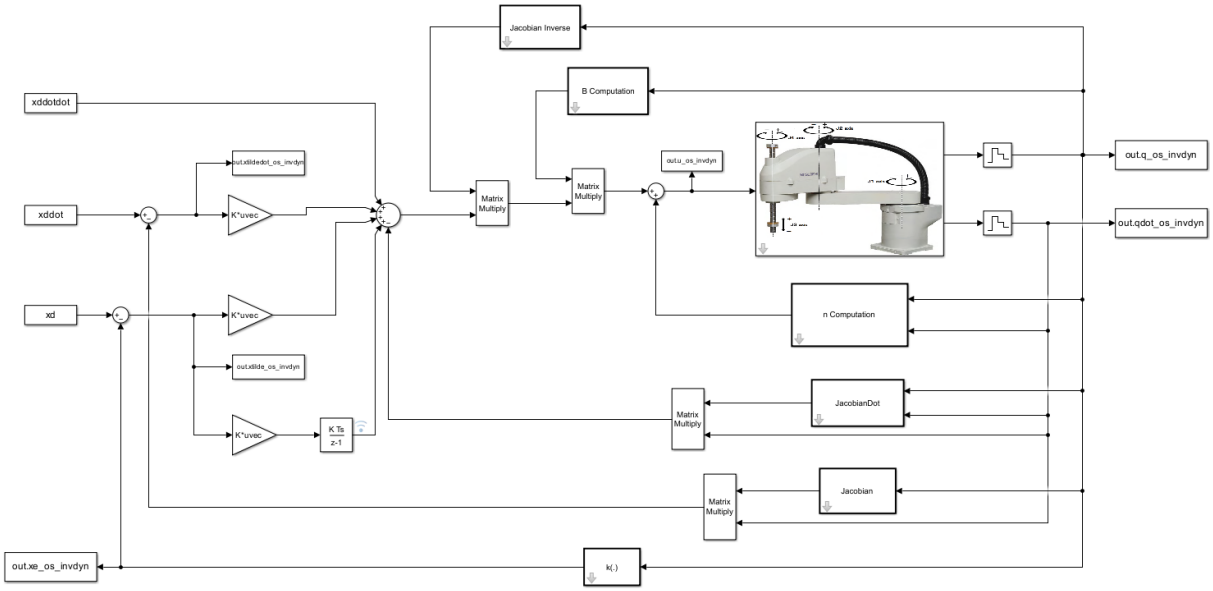

Figure 6.32: Inverse Dynamics Control Simulink Scheme

Now let's move on to the implementation of this control. The parameters used for the simulation are:

$$K_{P_{OS_{InvDyn}}} = \begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 1000 & 0 & 0 \\ 0 & 0 & 8000 & 0 \\ 0 & 0 & 0 & 200 \end{bmatrix}$$

$$K_{I_{OS_{InvDyn}}} = \begin{bmatrix} 500 & 0 & 0 & 0 \\ 0 & 500 & 0 & 0 \\ 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}$$

$$K_{D_{OS_{InvDyn}}} = \begin{bmatrix} 100 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 \\ 0 & 0 & 300 & 0 \\ 0 & 0 & 0 & 50 \end{bmatrix}$$

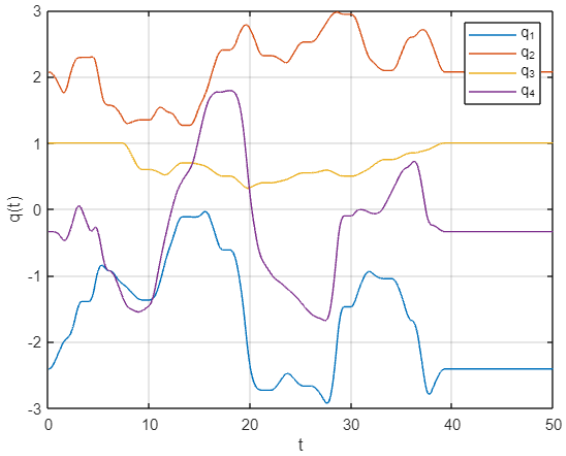

Figure 6.33: Joint Position Variables with Inverse Dynamics Control

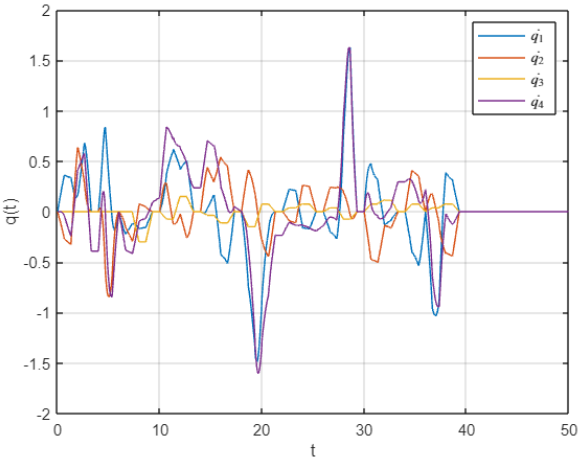

Figure 6.34: Joint Velocity Variables with Inverse Dynamics Control

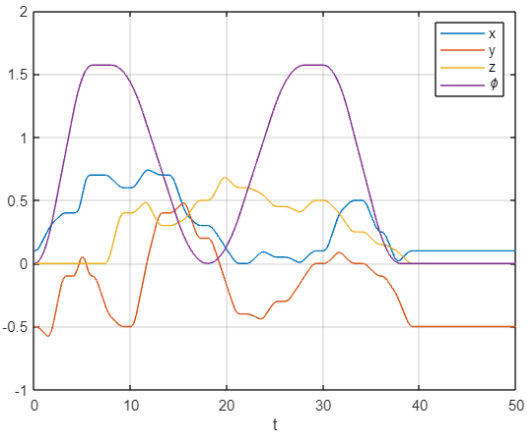

Figure 6.35: Operational Space Variables with Inverse Dynamics Control

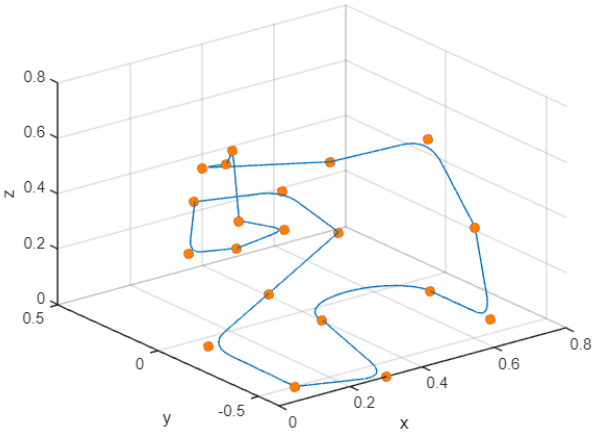

Figure 6.36: Trajectory with Inverse Dynamics Control

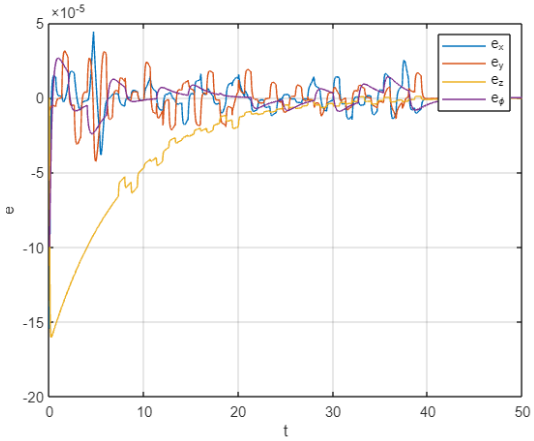

Figure 6.37: Position and Orientation Error with Inverse Dynamics Control

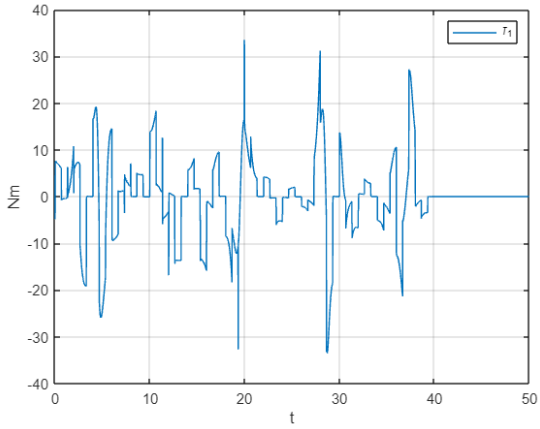

Figure 6.38: τ_1 with Inverse Dynamics Control

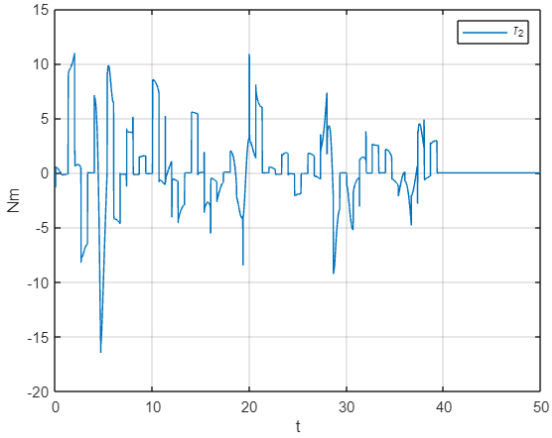

Figure 6.39: τ_2 with Inverse Dynamics Control

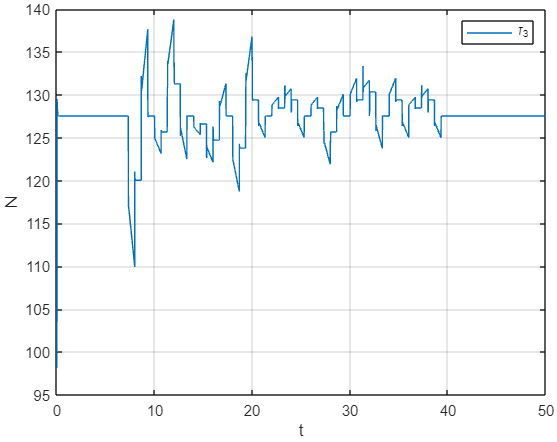

Figure 6.40: τ_3 with Inverse Dynamics Control

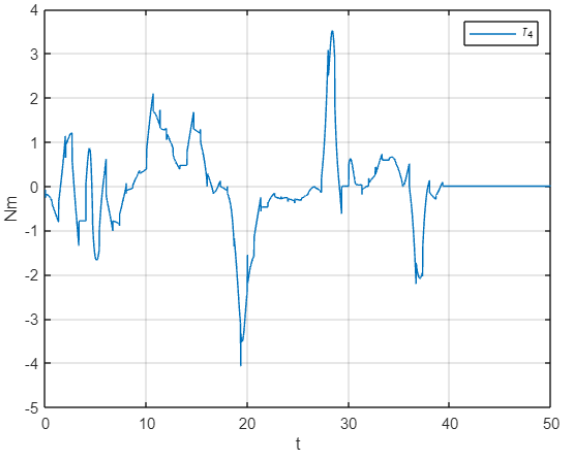

Figure 6.41: τ_4 with Inverse Dynamics Control