Politecnico di Torino

Department of CONTROL AND COMPUTER ENGINEERING (DAUIN)

Master's Degree in Mechatronic Engineering 2020/2021

Robotics – Prof. Rizzo, Ing. Primatesta

Journey in robotics

29/06/2021

Rosita Delle Grazie - 290624

Francesco Lenci - 285636

Giuseppe Morello - 281726

Matteo Sartoni – 289467

Edoardo Todde – 286330

Contents

1. Objective

This report contains a study on the manipulator's kinematic. First of all the forward kinematics is analyzed, followed by an overlook on the inverse kinematics problem. For this second topic we have studied the case in which there is more than one solution, hence we have worked with a planar manipulator (Three-link planar arm) and a 3D one (Spherical robot). Then we have studied the inverse kinematics using the Jacobian matrix and how the numerical derivation can be compensated by means of a kinematic feedback. Finally we have also analyzed the redundancy for said manipulators, so how we can obtain joint velocities that do not produce any motion at the end effector.

2. Manipulators under analysis

2.1 Spherical robot

The spherical (or polar) robot, depicted in *figure 1*, is made of three joints: two rotating joints (one vertical and one horizontal axis) that can rotate 360 degrees, followed by a prismatic one, that determines the radial stroke. It follows that this manipulator has 3 DOM, where each one of them corresponds to a polar coordinate. The link offset introduced by the prismatic joint is related to the manipulator's accuracy, the smaller is this elongation the more accurate is the robot. The workspace is a portion of a hollow sphere and it can also include a portion of the supporting base, allowing the manipulation of objects on the floor.

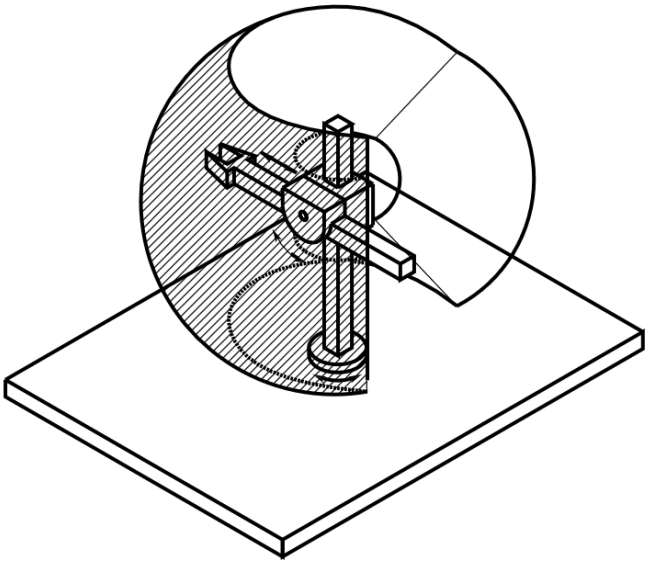

Figure 1 Spherical manipulator and its workspace

2.2 Three link manipulator

The Three Link manipulator, represented in *figure 2*, is a planar robot made by three consecuitve and parallel rotating joints. The arm consists of one fixed link and three movable ones that move within the plane. All the links are connected by revolute joints whose joint axes are all perpendicular to the plane of the links, hence it is a 3 DOM manipulator.

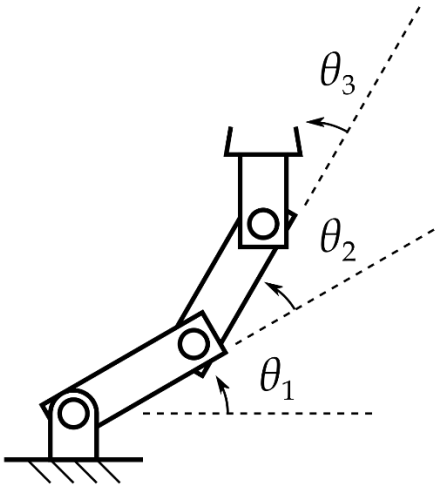

Figure 2 Three Link Manipulator

3. Direct kinematics

The direct kinematics uses linear algebra equations to compute the position and orientation of the end-effector with respect to the fixed frame, given the joint space q. With the D-H convention is possible to define a frame of reference for every link of our manipulator and to represent, with only four parameters, a geometrical transformation in a three-dimensional Euclidean space.

3.1 Direct kinematics of a Spherical Manipulator

Using the D-H convention for a spherical manipulator a suitable choice of the reference frames is represented in *figure 3*.

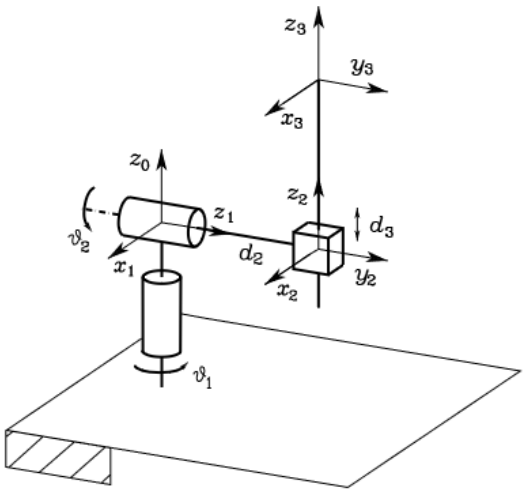

Figure 3 D-H convention for a spherical manipulator

Hence, the D-H parameters are the following:

Link	$\boldsymbol{a_i}$	$\boldsymbol{\alpha_i}$	$\boldsymbol{d_i}$	$\boldsymbol{\theta_i}$
1	0	-π/2	0	θ_1
2	0	π/2	d_2	θ_2
3	0	0	d_3	0

Table 1 D-H parameters for spherical manipulator

where it is worth to notice that the origin of frame R_0 is chosen equal to the one of R_1 in order to minimize the number of parameters ($d_1 = 0$).

From the values reported in table 1, we can define the homogeneous transformation matrices between a reference frame attached to a link and the following one:

$$A_1^0(\theta_1) = \begin{pmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad A_2^1(\theta_2) = \begin{pmatrix} c_2 & 0 & s_2 & 0 \\ s_2 & 0 & -c_2 & 0 \\ 0 & 1 & 0 & d_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

Multiplying these matrices we obtain to homogeneous transformation matrix $T_3^0(q)$ between the fixed frame R_0 and the one attached to the tool center point R_3, hence we have solved the direct kinematic problem:

$$T_3^0(q) = A_1^0(\theta_1)A_2^1(\theta_2)A_3^2(d_3) = \begin{pmatrix} c_1c_2 & -s_1 & c_1s_2 & c_1s_2d_3 - s_1d_2 \\ s_1c_2 & c_1 & s_1s_2 & s_1s_2d_3 + c_1d_2 \\ -s_2 & 0 & c_2 & c_2d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where $q = \begin{pmatrix} \theta_1 \\ \theta_2 \\ d_3 \end{pmatrix}$ is the *joint space* of the manipulator.

3.1.1 Matlab code

Here is reported the Matlab code to generate a spherical manipulator and to compute the direct kinematic, where we have set the link offset $d_2 = 5$.

```
clear all, close all, clc
%% DH Parameters
% Link length
DH.a1 = 0;
DH.a2 = 0;
DH.a3 = 0;
% Link offset
DH.d1 = 1;
DH.d2 = 5;
% Link twist
DH.alpha1 = -pi/2;
DH.alpha2 = pi/2;
DH.alpha3 = 0;
% Joint angle
DH.theta3 = 0;
%% Link definition
% Link and joint definition
L(1) = Link('d',DH.d1,'a',DH.a1,'alpha',DH.alpha1);
L(2) = Link('d',DH.d2,'a',DH.a2,'alpha',DH.alpha2);
L(3) = Link('theta',DH.theta3,'a',DH.a3,'alpha',DH.alpha3,'offset',0.5);
% Minimum and maximum values for the joint variables
L(1).qlim = [-pi pi];
L(2).qlim = [-pi pi];
L(3).qlim = [0 3];
%% Robot definition
Spherical = SerialLink(L,'name','RRP')
save Spherical Spherical
save DH DH
%% Direct kinematics
q_1 = [pi/2 pi/4 2]; % Joint space q
Spherical.plot(q_1) % Spherical manipulator
T = Spherical.fkine(q_1) % Direct kinematic for the defined q
```

Through this code we can generate a spherical manipulator, as depicted in *figure 4*, and by defining the joint space as follows $q = (\frac{\pi}{2} \ \frac{\pi}{4} \ 2)^T$ and using the command *fkine,* we have solved the forward kinematic problem, obtaining the following homogeneous transformation matrix:

```
T =
         0         -1          0         -5
    0.7071          0     0.7071     1.768
   -0.7071          0     0.7071     1.768
         0          0          0         1
```

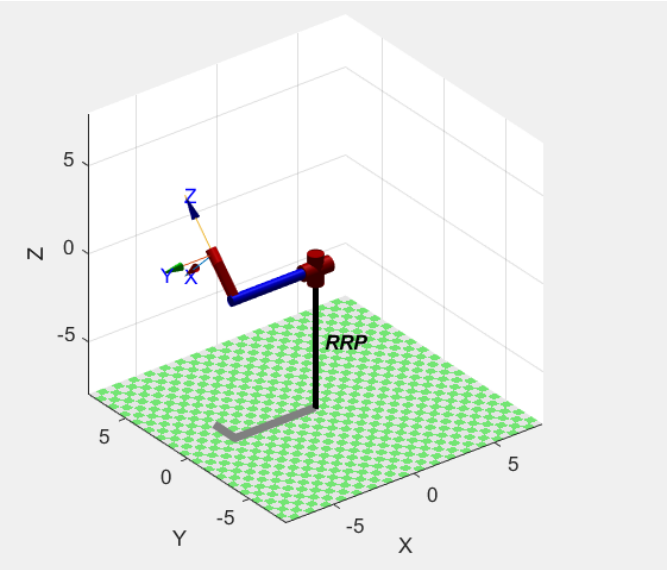

Figure 4 Spherical Manipulator

3.2 Direct kinematics of a Three-Link Planar arm

The D-H convention for a three-link planar arm is depicted in *figure 5*.

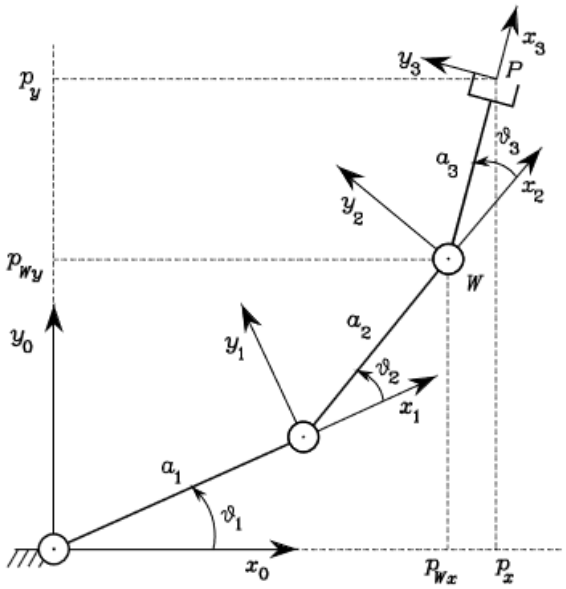

Figure 5 D-H convention for a three-link planar arm

This choice of the frame of references leads to the following D-H parameters:

Link	$\boldsymbol{a_i}$	$\boldsymbol{\alpha_i}$	$\boldsymbol{d_i}$	$\boldsymbol{\theta_i}$
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3

Table 2 D-H parameters for a three-link planar arm

Using the defined parameters, we can obtain the homogeneous transformation matrices between two consecutive links:

$$A_1^0(\theta_1) = \begin{pmatrix} c_1 & -s_1 & 0 & a_1c_1 \\ s_1 & c_1 & 0 & a_1s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad A_2^1(\theta_2) = \begin{pmatrix} c_2 & -s_2 & 0 & a_2c_2 \\ s_2 & c_2 & 0 & a_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_3^2(\theta_3) = \begin{pmatrix} c_3 & -s_3 & 0 & a_3c_3 \\ s_3 & c_3 & 0 & a_3s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

In order to solve the forward kinematic problem, we can once again multiply these matrices to compute the rototranslation matrix $T_3^0(q)$ between the fixed frame R_0 and the end-effector frame R_3.

$$T_3^0(q) = A_1^0(\theta_1)A_2^1(\theta_2)A_3^2(\theta_3) = \begin{pmatrix} c_{123} & -s_{123} & 0 & a_1c_1 + a_2c_{12} + a_3c_{123} \\ s_{123} & c_{123} & 0 & a_1s_1 + a_2s_{12} + a_3s_{123} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Where $q = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix}$ is the three-link manipulator joint space.

3.2.1 Matlab code

Here is reported the Matlab code to generate a Three Link Planar manipulator and to solve the direct kinematic problem, for which we have chosen the following joint space $q = (\frac{\pi}{3}\ \frac{\pi}{2}\ \frac{\pi}{4})^T$.

```
clear all, close all, clc
%% DH parameters
% Link length
DH.a1 = 3;
DH.a2 = 2;
DH.a3 = 1;
% Link offset
```

```
DH.alpha1 = 0;
DH.alpha2 = 0;
DH.alpha3 = 0;
% Link twist
DH.d1 = 0;
DH.d2 = 0;
DH.d3 = 0;
%% Link definition
L(1) = Link('d', DH.d1, 'a', DH.a1, 'alpha', DH.alpha1);
L(2) = Link('d', DH.d2, 'a', DH.a2, 'alpha', DH.alpha2);
L(3) = Link('d', DH.d3, 'a', DH.a3, 'alpha', DH.alpha3);
% Minimum and maximum values for the joint variables
L(1).qlim = [-pi pi];
L(2).qlim = [-pi pi];
L(3).qlim = [-pi pi];
%% Robot definition
Three_Link = SerialLink(L, 'name', 'RRR')
save Three_Link Three_Link
save DH DH
%% Direct Kinematics
q1 = [pi/3 pi/2 pi/4]; % Joint Space
Three_Link.plot(q1)
T = Three_Link.fkine(q1) % Forward Kinematics
```

By means of this code we have built the manipulator reported in *figure 6* and we have obtained the following homogeneous transformation matrix:

```
T =

   -0.9659    0.2588         0   -1.198

   -0.2588   -0.9659         0    3.339

         0         0         1         0

         0         0         0         1
```

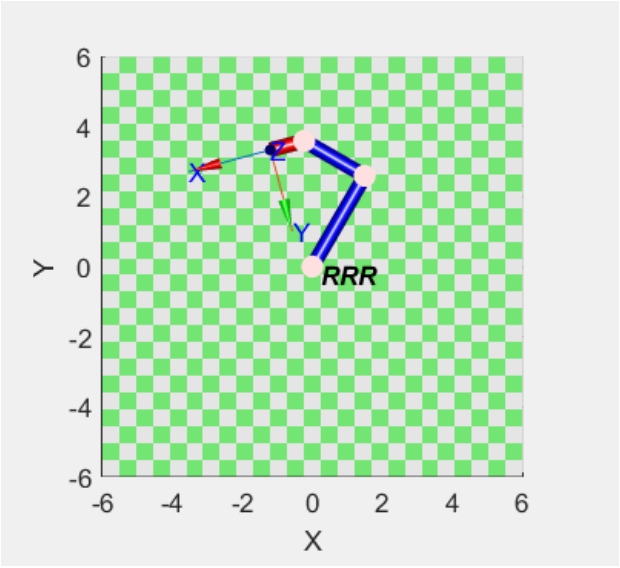

Figure 6 Three Link Planar Manipulator

4. Inverse kinematics

The aim of the inverse kinematics problem is to determine the joint variables corresponding to a desired end-effector position and orientation. While in the direct kinematics the end-effector rototranslation matrix is computed in a unique manner, once the joints are known, the inverse kinematics is much more complex since:

- It is not always possible to find a closed-form solution since the equations to be solved are in general non-linear.
- Multiple solutions may exist.
- Infinite solutions may exist, in the case of a kinematically redundant manipulator.
- There might be no admissible solutions.

Among these problems that may arise, we have focused on the second one.

The inverse kinematics may be solved through suitable numerical calculations, exploiting a kinematic feedback, needed to avoid numerical drift. Two possible implementations exist, depending on the examined case, through the transpose of the Jacobian or through its inverse. Later, they will be adapted to our manipulators.

4.1 Inverse kinematics of a Spherical Manipulator

Given the definition of the inverse kinematics problem one can think that by applying in the end of the *Matlab* code reported in 3.1.1, the command `q_inv = Spherical.ikine(T)`, we should solve this problem and obtain a joint space `q_inv` equal to the one `q_1` used for the forward kinematic. Unfortunately this does not happen and the following error occurs: "`Number of robot DOF must be >= the number of 1s in the mask matrix`".

This happens because in the spherical manipulator it possible to demonstrate that two different solutions aries for the inverse kinematics problem.

Given the desired end-effector position $p_3 = (p_{3x}\ p_{3y}\ p_{3z})^T$ and knowing that $T_3^0(q) = A_1^0 A_2^1 A_3^2$, we can define p_3 with respect to frame R_1 and consider this equation:

$$(A_1^0)^{-1} T_3^0(q) = A_2^1 A_3^2.$$

Considering the first three element of the fourth column of the matrices, we can obtain the vector p_3^1 as follows:

$$p_3^1 = \begin{pmatrix} p_{3x}c_1 + p_{3y}s_1 \\ -p_{3z} \\ -p_{3x}s_1 + p_{3y}c_1 \end{pmatrix} = \begin{pmatrix} d_3 s_2 \\ -d_3 c_2 \\ d_2 \end{pmatrix}$$

that depends only on d_3 and θ_2. Because of this, we can add a new parameter t to solve these equations

$$t = tan\frac{\theta_1}{2} \Rightarrow c_1 = \frac{1-t^2}{1+t^2} \; and \; s_1 = \frac{2t}{1+t^2}.$$

Substituting this in the third equation of p_3^1 we obtain:

$$\left(d_2 + p_{3y}\right)t^2 + 2p_{3x}t + d_2 - p_{3y} = 0$$

from which we can compute the parameter t

$$t = \frac{-p_{3x} \pm \sqrt{{p_{3x}}^2 + {p_{3y}}^2 - {d_2}^2}}{d_2 + p_{3y}}.$$

The two different solutions for the parameter t lead to two different configurations of the first joint variable θ_1

$$\theta_1 = 2Atan2\left(-p_{3x} \pm \sqrt{{p_{3x}}^2 + {p_{3y}}^2 - {d_2}^2}, d_2 + p_{3y}\right),$$

once θ_1 is defined if we square and sum the first two equations that give us the vector p_3^1 it follows that:

$$d_3^2(s_2^2 + c_2^2) = (p_{3x}c_1 + p_{3y}s_1)^2 + p_{3z}^2$$

and since a distance can only be positive, we obtain

$$d_3 = \sqrt{(p_{3x}c_1 + p_{3y}s_1)^2 + p_{3z}^2}.$$

If the parameter $d_3 \neq 0$, considering again the first two equations that give us the vector p_3^1 but dividing the first with the second one, we can easily compute the parameter θ_2, that is

$$\frac{p_{3x}c_1 + p_{3y}s_1}{-p_{3z}} = \frac{d_3 s_2}{-d_3 c_2} \xrightarrow{yields} \theta_2 = Atan2\left(p_{3x}c_1 + p_{3y}s_1, p_{3z}\right).$$

If instead $d_3 = 0$, the joint variable θ_2 cannot be determined.

It is worth to underline that the inverse kinematics problem can also be solved using the Matlab command `ikine` and defining a suitable mask matrix. This matrix is a vector of dimension 6x1 that is made only by ones and/or zeros, in which every component represents one of the six parameters of the operational space. When a parameter is set to 1 it means that when computing the inverse kinematics we want to find a joint space q that, if applied to the robot, let us obtain an end-effector pose in which that parameter (a coordinate or an angle) is achieved.

Hence by setting the mask matrix in this way:

```
q_inv = Spherical.ikine(T,'mask',[1 1 1 0 0 0])
```

we no longer have an error because the number of DOF is equal to the number of ones in the mask matrix, in fact we obtain:

```
q_inv = 1.5708     0.0000     2.0000
```

that is equal to the joint space q_1 defined in 3.1.1 .

4.2 Inverse kinematics of a Three-Link Planar arm

Even for the Three-Link Planar arm, if we add to the code reported in 3.2.1 the command `q_inv = Three_Link.ikine(T)` we face the same error obtained for the Spherical manipulator.
Again, this is due to the multiple solutions of the inverse kinematics problem for the robot under analysis. Given a desired pose of the end-effector, we want to find the three joint variables θ_1 , θ_2, θ_3 that allow us to reach that configuration. Once the orientation is defined:

$$\varphi = \theta_1 + \theta_2 + \theta_3$$

is one of the equations to be solved. Given the vector $p_2^0 = (p_{2x} \;\; p_{2y} \;\; p_{2z})^T$, i.e. the coordinates of the origin of frame R_2 with respect to R_0, if we look at the first two elements of the last column of the matrix $A_2^0 = A_1^0(\theta_1)A_2^1(\theta_2)$, where $A_1^0(\theta_1)\ and\ A_2^1(\theta_2)$ are defined in 3.2, we obtain that:

$$\begin{aligned} p_{2x} &= a_1 c_1 + a_2 c_{12} \\ p_{2y} &= a_1 s_1 + a_2 s_{12} \end{aligned}.$$

If we square and sum these two equations, it follows that:

$$p_{2x}^2 + p_{2y}^2 = a_1^2 + a_2^2 + 2a_1 a_2 c_2 \xrightarrow{yields} c_2 = \frac{p_{2x}^2 + p_{2y}^2 - a_1^2 - a_2^2}{2a_1 a_2}.$$

Obviously, the existence of the solution imposes that $-1 \leq c_2 \leq 1$. Then by means of trigonometry we can determine the joint variable θ_2, since

$$s_2 = \pm\sqrt{1 - c_2^2} \xrightarrow{yields} \theta_2 = Atan2(s_2, c_2).$$

In order to compute θ_1, we substitute θ_2 in the equations related to p_{2x} and p_{2y} and we just have to solve the following system

$$s_1 = \frac{(a_1 + a_2 c_2)p_{2y} - a_2 s_2 p_{2x}}{p_{2x}^2 + p_{2y}^2}$$
$$c_1 = \frac{(a_1 + a_2 s_2)p_{2x} + a_2 s_2 p_{2y}}{p_{2x}^2 + p_{2y}^2}$$

from which we can calculate the joint variable as

$$\theta_1 = Atan2(s_1, c_1).$$

This angle can be determined uniquely unless $a_1 = a_2$ and it is required $p_{2x} = p_{2y}$.
Given θ_1 and θ_2 the third joint variable can be easily computed:

$\theta_3 = \varphi - \theta_1 - \theta_2$.

As seen for the Spherical manipulator, even for the Three-Link one it is possible to try solving the inverse kinematics problem via Matlab defining a suitable mask matrix.
More in detail, if we add to the code reported in 3.2.1 the command
`q_inv = Three_Link.ikine(T, 'mask', [1 1 1 0 0 0])` we obtain the following joint space:
`q_inv = 1.6057  1.5781  -2.7443.`
It is worth to underline that the obtained joint space is not equal to the one defined by us as `q1`, apart from the second joint variable, because we have defined the mask matrix so that we are interested only in the position of the end-effector and not in its orientation. In fact by applying the following command `T1 = Three_Link.fkine(q_inv)` we have:

```
T1 = 0.9049   -0.4256         0    -1.198
     0.4256    0.9049         0     3.339
          0         0         1         0
          0         0         0         1
```

and it results that by using the joint space obtained in the inverse kinematics problem we have a homogeneous transformation matrix `T1` that gives us the same position of the end-effector (the first three rows of the last column are the same of `T`) but a different rotation with respect to the fixed frame.

5. Inverse kinematics with the Jacobian

The differential kinematics gives us a linear mapping between the joint velocity $\dot{q}$ and the operational space one $\dot{x}$ that changes with the robot configuration:

$$\dot{x} = J_A(q)\dot{q}\,.$$

One can think to use this relationship to solve the inverse kinematics problem, since if we consider $n = r$ the joint velocities can be easily obtained

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

and if the initial manipulator posture $q(0)$ is known, the joint positions can be computed by integrating the velocity

$$q(t) = \int_0^t \dot{q}\,(\alpha)d\alpha + q(0).$$

This integration can be performed by means of the Euler integration method: given an integration interval Δt, if the joint positions and velocities at time t_k are known, the joint positions at time $t_{k+1} = t_k + \Delta t$ are

$$q(t_{k+1}) = q(t_k) + \dot{q}(t_k)\Delta t.$$

Here a problem arises, the reconstruction of the joint space q is based on a numerical integration that involves drift phenomena of the solution, therefore the computed end-effector pose is not equal to the desired one.
This problem can be solved by means of algorithms that consider the operational space error $e = x_d - x_e$, where x_d is the desired operational space while x_e is the achieved one: the "Inverse kinematics algorithm with Jacobian inverse" and the "Inverse kinematics algorithm with the Jacobian transpose".

5.1 Inverse kinematics with Jacobian inverse

The algorithm to implement is the one depicted in *figure 7*. Here we start from the assumption that J_A is square and non-singular: $\dot{q} = J_A^{-1}(q) \cdot (\dot{x_d} + Ke)$, that leads us to the error dynamics equation: $\dot{e} + Ke = 0$ which is a first order linear dynamical system.
If $K > 0$ → $\lim_{t\to\infty} e(t) = 0 \; \forall e(0)$, i.e. the system is asymptotically stable, so the convergency depends on the eigenvalues of K (in general a diagonal matrix), more in detail the larger the eigenvalues the faster is the convergence. In the feedback loop instead $k()$ is the forward kinematics operator.
The limitation for this kind method is that, to implement kinematic control, it works for low velocities because inertia is mitigated.
In case of redundant manipulators instead, the relation that holds takes into account the pseudo-inverse Jacobian:

$$\dot{q} = J_A^{+}(q) \cdot (\dot{x_d} + ke) + (I - J_A^{+}J_A) \cdot \dot{q_o}$$

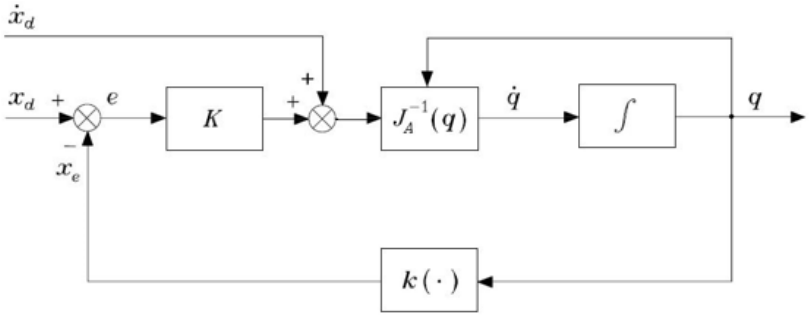

Figure 7 Inverse kinematics with Jacobian inverse

5.1.1 Inverse kinematics of a Three-Link planar arm with Jacobian inverse

In order to apply this algorithm to a Three-Link arm, a suitable Simulink model has been designed as represented in *figure 8*.

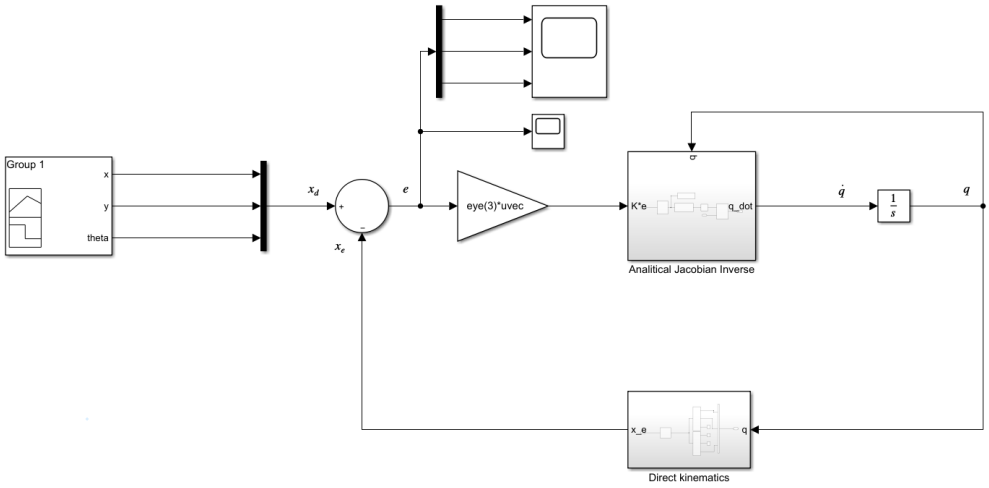

Figure 8 Simulink model of the Inverse kinematics with Jacobian Inverse algorithm

The manipulator under analysis is a planar one, so the desired operational space has only three parameters: $x_d = (0.27526 \quad 5.237 \quad 2.3562)^T$ that are related to the joint space $q = (\pi/3 \quad \pi/4 \quad \pi/6)^T$. The three operational space variables tell us what are the x and y coordinates of the origin of R_3 with respect to the fixed frame and the last one represents the angle θ, i.e. the rotation about the z axis between this two frames. Because of this, K is a 3x3 matrix but the same considerations made before hold true. The inverse of J_A is modelled using a subsystem that takes as input the joint space q step by step and computes the Jacobian with the block '*jacob0*' developed by Peter Corke using the '*rpy*' angles, but since J_A is a 6x3 matrix, to compute J_A^{-1} we have added also a "*Variable Selector*" to select only the first, the second and the fifth rows hence to obtain a square matrix. The output of this block has been multiplied by the Ke vector and an integrator is needed for computing q from $\dot{q}$.

Regarding the feedback path, given as input the joint configuration in time, the position of the end-effector is computed by simply using the homogeneous transformation matrix T_3^0 and extracting from this the position and the orientation.

One last thing worth to underline in the Simulink scheme is that from the moment that we have adopted a constant desired operational space x_d its velocity is equal to zero and so it can be neglected in this scenario.

In *figure 9* the error dynamics is reported and it's clearly visible that our system is asymptotically stable.

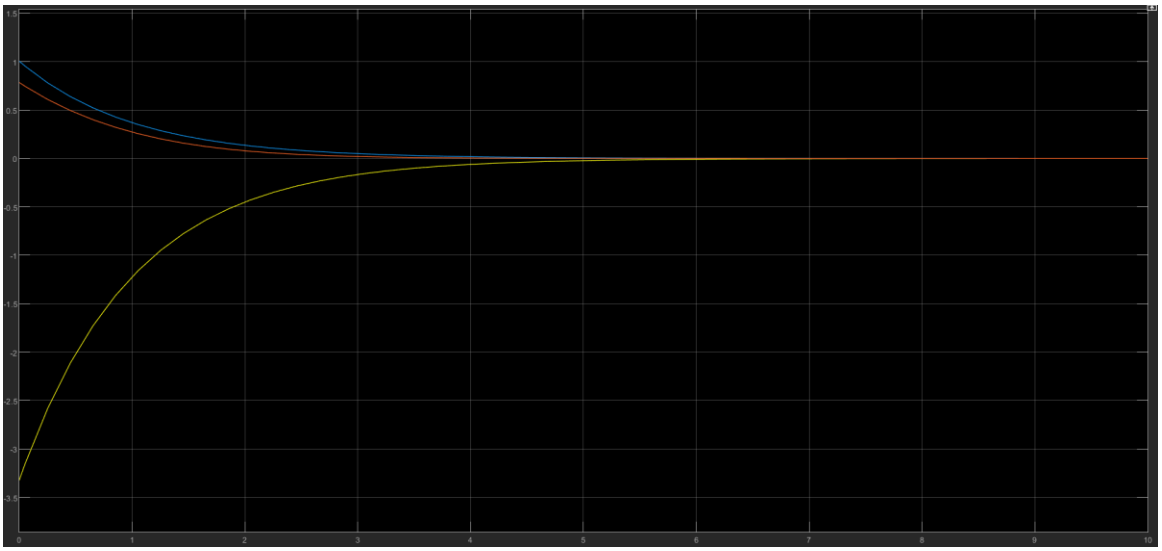

Figure 9 Error Dynamics

5.1.2 Inverse kinematic of a Stanford arm with Jacobian inverse

Given that the inverse kinematics algorithm with Jacobian inverse can only be applied if the Jacobian matrix is square and non-singular, it was not possible to test it with the spherical manipulator, since it is a manipulator working in the space that has only three joints, hence J_A is a 6x3 matrix.
To solve this problem we have decided to work with a Stanford manipulator, that is made by a spherical manipulator and a spherical wrist as reported in *figure 10*.

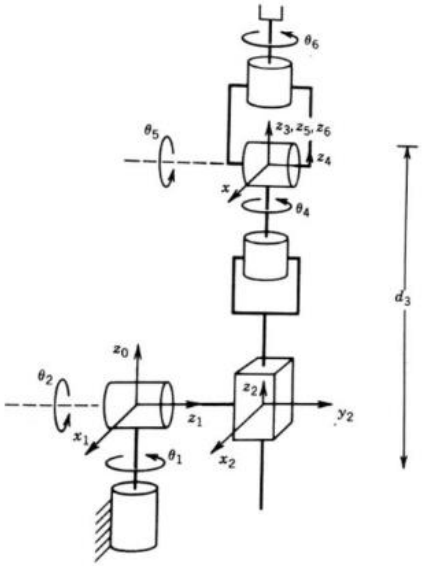

Figure 10 Stanford Manipulator

The Simulink scheme exploited for this robot is represented in *figure 11*. The main difference between the previous case and this one is that the Stanford manipulator is a robot working in the space, hence the desired joint space x_d is a 6x1 vector and the K matrix is a 6x6 one.

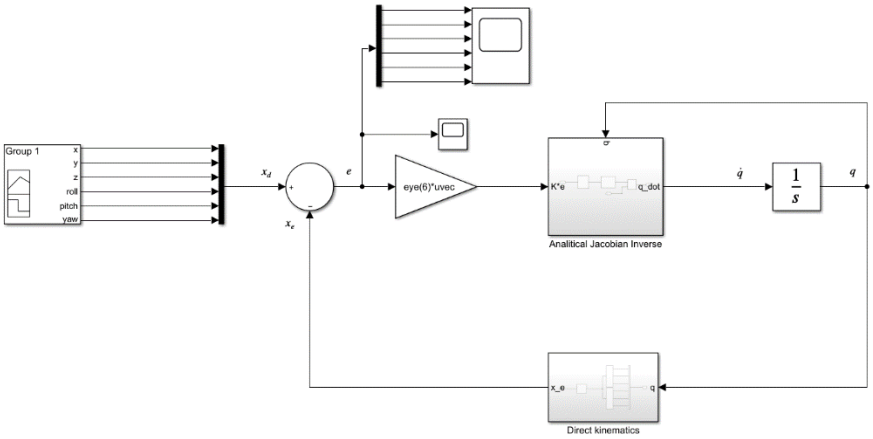

Figure 11 Simulink scheme for Stanford manipulator

The error dynamics is depicted in *figure 12*. In this analysis we have considered the joint space: $q = (\pi/2 \quad \pi/3 \quad 2 \quad \pi/4 \quad \pi/2 \quad \pi/6)^T$ that leads to the following operational space:

$$x = -7.121$$
$$y = 2.793$$
$$z = -0.8371$$
$$roll = 2.2896$$
$$pitch = 0.92221$$
$$yaw = -1.9152$$

Here it is possible to notice that out of the six variables of the operational space x only five are stable, while the yaw angle does not converge to zero, probably because of some approximation made by the compiler. It is interesting also to underline that this error sometimes changes its value between, approximately, 1.225 and -5.055 with a very high slope, so it has jump of 6.28, more or less 2π.

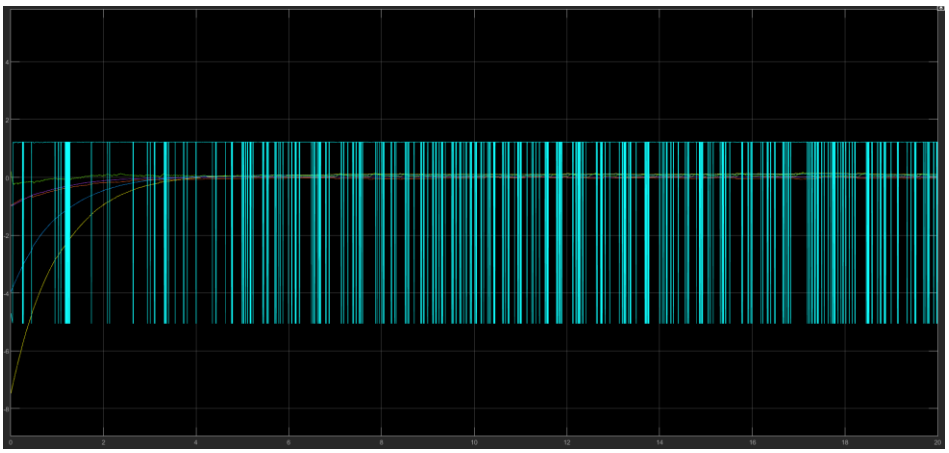

Figure 12 Error Dynamics

This means that in the computed operational space x_e, reported in *figure 13*, the yaw angle converges to a given value, around -3.14, and as previously analyzed, it changes of 2π hence the configuration does not change.

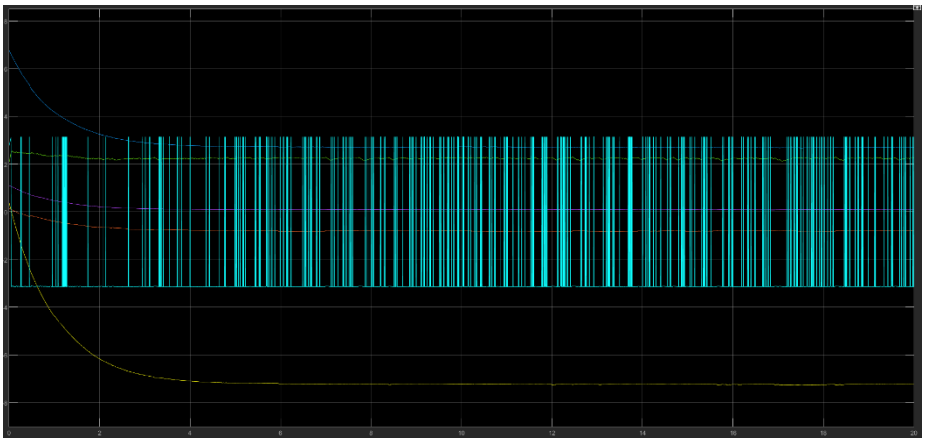

Figure 13 Operational space x_e

One last thing worth to underline is that the joint space *q* coming from the '*Integrator*' has all the variables converge to some fixed values, as depicted in *figure 14*, even though it has a noisy trend.

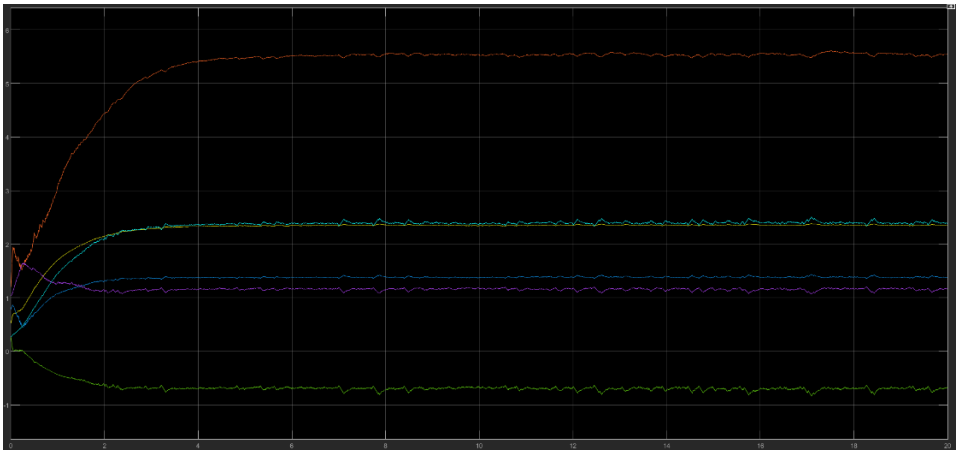

Figure 14 Joint space q

The MATLAB code developed to obtain the Stanford manipulator is here reported:

```
clear all, close all, clc
%% DH Parameters
% Link length
DH.a1 = 0;
DH.a2 = 0;
DH.a3 = 0;
DH.a4 = 0;
DH.a5 = 0;
DH.a6 = 0;
% Link offset
DH.d1 = 0;
DH.d2 = 5;
DH.d3 = 4;
DH.d4 = 0;
DH.d5 = 0;
DH.d6 = 3;
% Link twist
DH.alpha1 = -pi/2;
DH.alpha2 = pi/2;
DH.alpha3 = 0;
```

```
DH.alpha4 = -pi/2;
DH.alpha5 = pi/2;
DH.alpha6 = 0;
% Joint angle
DH.theta3 = 0;
%% Link definition
L(1) = Link('d', DH.d1, 'a', DH.a1, 'alpha', DH.alpha1);
L(2) = Link('d', DH.d2, 'a', DH.a2, 'alpha', DH.alpha2);
L(3) = Link('theta', DH.theta3, 'a', DH.a3, 'alpha', DH.alpha3);
L(4) = Link('d', DH.d4, 'a', DH.a4, 'alpha', DH.alpha4);
L(5) = Link('d', DH.d5, 'a', DH.a5, 'alpha', DH.alpha5);
L(6) = Link('d', DH.d6, 'a', DH.a6, 'alpha', DH.alpha6);
% Joint variables limit
L(1).qlim = [-pi pi];
L(2).qlim = [-pi pi];
L(3).qlim = [0 3];
L(4).qlim = [-pi/2 pi/2];
L(5).qlim = [-pi/2 pi/2];
L(6).qlim = [-pi/2 pi/2];
%% Manipulator
Stanford = SerialLink(L,'name','Stanf')
save Stanford Stanford
save DH DH
q_1 = [pi/2 pi/3 2 pi/4 pi/2 pi/6]; % Joint space
Stanford.plot(q_1)
J = Stanford.jacob0(q_1) %Jacobian matrix
T = Stanford.fkine(q_1) % Direct kinematics
```

5.2 Inverse kinematics with Jacobian transpose

Another algorithm to compute the inverse kinematics using the Jacobian can be defined by looking at a relationship between $\dot{q}$ and e that ensures asymptotical stability, without linearizing $\dot{e} = \dot{x_d} - J_A(q)\,\dot{q}$. In order to do this we can choose a suitable Lyapunov function $V(e) = \frac{1}{2} e^T K e$ where K is a symmetric positive definite matrix, $V(e) > 0$ if $e \neq 0$ and $V(0) = 0$. The asymptotical stability is reached if the derivative of the Lyapunov function with respect to the time is negative, that we can easily compute as:

$$\dot{V} = e^T K \dot{x_d} - e^T K J_A(q)\, J_A^T(\dot{q}) K e\,.$$

Where we have defined the joint velocity as follows $\dot{q} = J_A^T(q) K e$. In this case, if we consider a constant reference system i.e. $\dot{x_d} = 0$, it results that $V > 0$ and $\dot{V} < 0$ so the system is asymptotically stable.

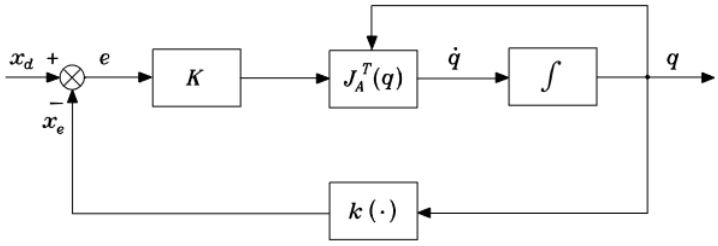

Figure 15 Inverse kinematics algorithm with Jacobian transpose

5.2.1 Inverse kinematics of a Three Link planar arm with Jacobian transpose

To apply the scheme depicted in *figure 15*, a suitable Simulink model can be designed. Since the analytical Jacobian is required it is computed with the option '*rpy*', which returns the Jacobian in analytical form with rotational part expressed in '*rpy*' angles. As a consequence, the rotation part of the final position must be expressed with the same angles.

In the forward path K is a 6x6 diagonal matrix where the entries are index of weight to assign to the related variable (6 variables are x, y, z, roll, pitch, yaw): the higher is the element on the diagonal, the faster that variable will converge to the final values. The J_A is modelled also in this case by a subsystem that takes as input q (joint space) step by step and compute the Jacobian with the block '*jacob0*' belonging to the Robotics Toolbox developed by Peter Corke, followed in cascade by a '*Transpose'* block. An integrator is once again needed for computing q from $\dot{q}$.

The feedback path, instead, is constituted by the simple direct kinematics. It has been modelled by a subsystem that, given as input the joint configuration in time, returns the position of the end-effector by simply computing the homogeneous transformation matrix T_3^0 and extracting from this the position and the orientation.

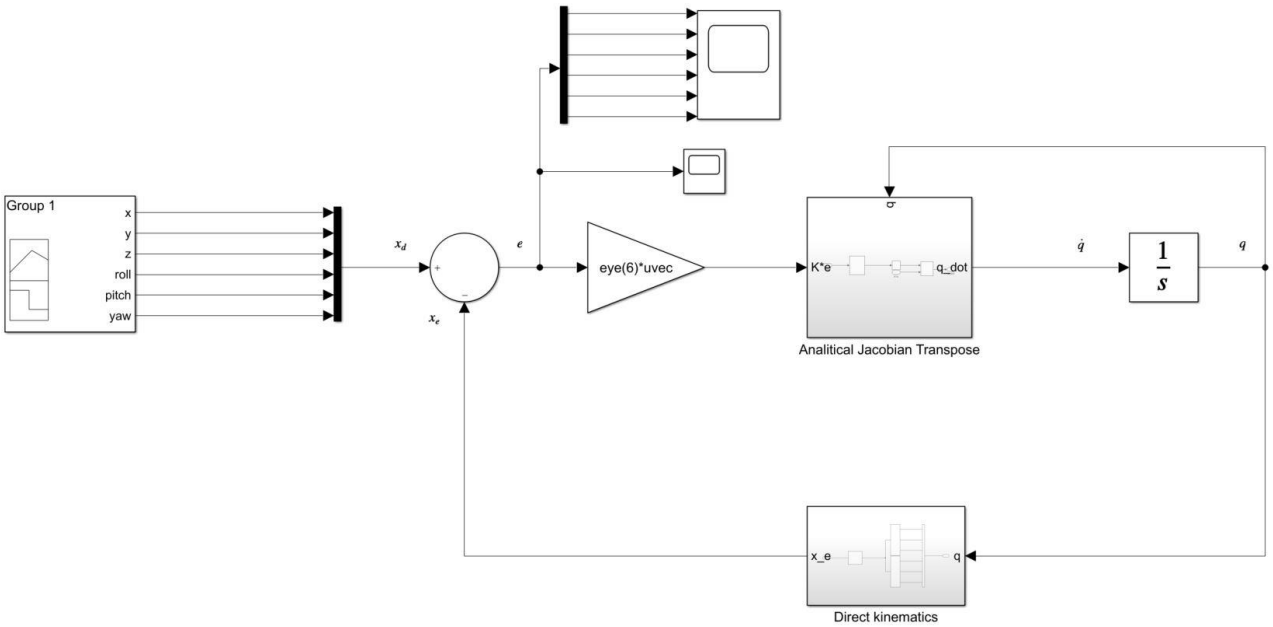

Figure 16 Inverse kinematics system with kinematic feedback.

A scope is exploited, in order to see the convergence of the error to zero.
The desired position is set in this way:

$$x = 1.9319$$
$$y = 4.7603$$
$$z = 0$$
$$roll = 0$$
$$pitch = 0$$
$$yaw = 2.3562$$

Where z, roll and pitch are obviously equal to zero because the robot stands on a 2-D plane.
These values have been obtained starting from a known configuration of the robot (in such a way that desired position is feasible for the manipulator):

$$q = \left[\frac{\pi}{4}, \frac{\pi}{6}, \frac{\pi}{3}\right]^T$$

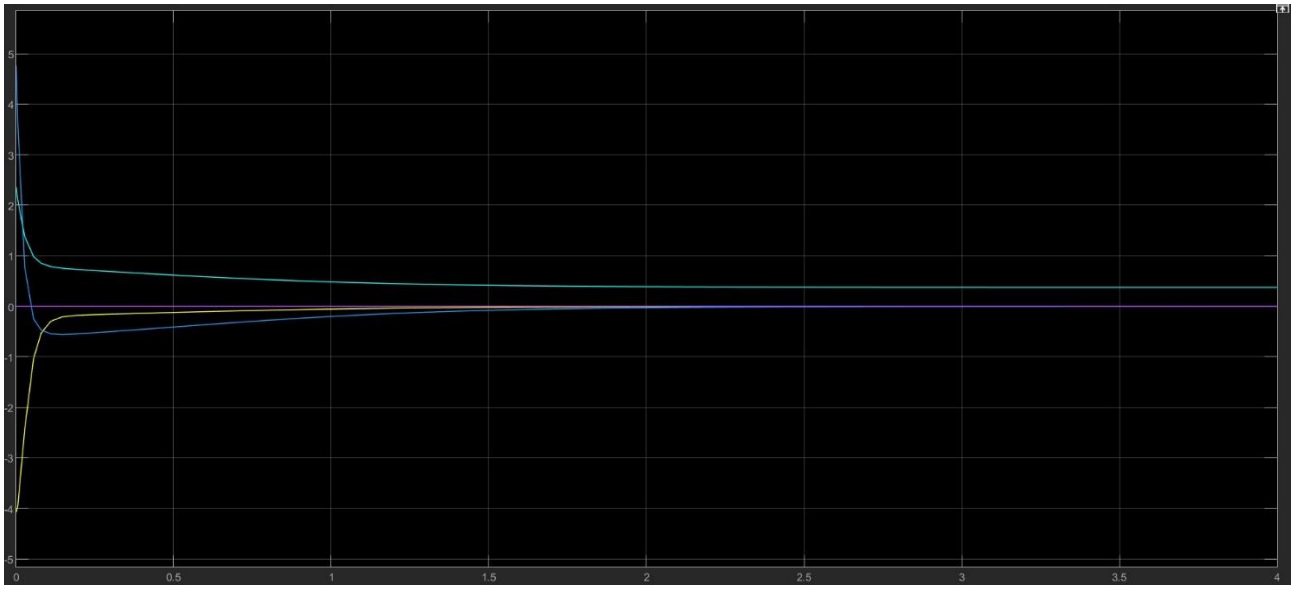

Figure 16 Errors on a scope.

K is set as identity matrix I_{6x6}. Then, a 4 second simulation is done.

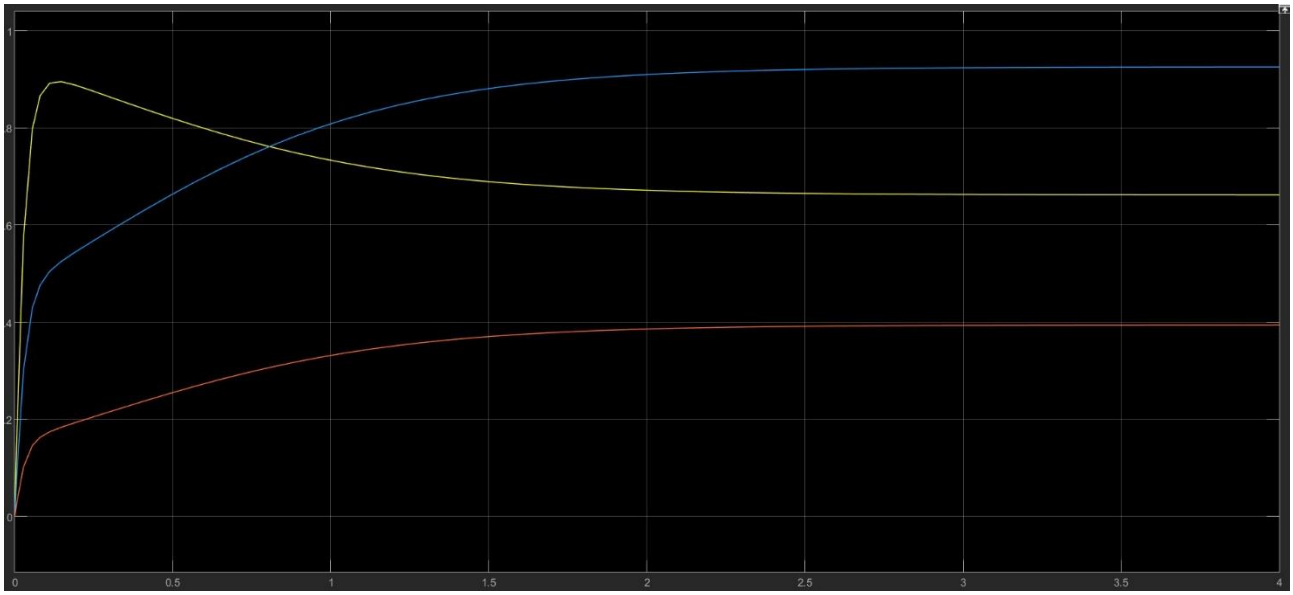

Figure 17 Joint space q(t) of the simulation.

It is evident that each variable converges to zero, with the exception of one: it is the yaw angle, that has an offset error of about 0.3739 radians.

The reason of this offset error can be easily justified: starting from a given posture, the inversion of the kinematics is done without specifying constraints, thus the solution given by simulating is not coincident with

the one we started from: the problem of multiple solution arises. A solution very near to the one specified is found, with a very little variation on the angle.
Final joint space in fact, is given by:

$$q_{final} \cong [0.662, 0.925, 0.394]^T$$

computing the transformation matrix, the position of the end-effector is the same as before, but the yaw angle is equal to:

$$yaw_{final} \cong 1.9810$$
$$\varepsilon_{yaw} \cong 2.3562 - 1.9810 = 0.3752$$

consistent with result obtained in simulation.

5.2.2 Inverse kinematics of a Spherical manipulator with Jacobian transpose

Even for the Spherical manipulator, in order to apply the algorithm of the inverse kinematics using the Jacobian transpose, we have built a suitable Simulink model as depicted in *figure 18*.

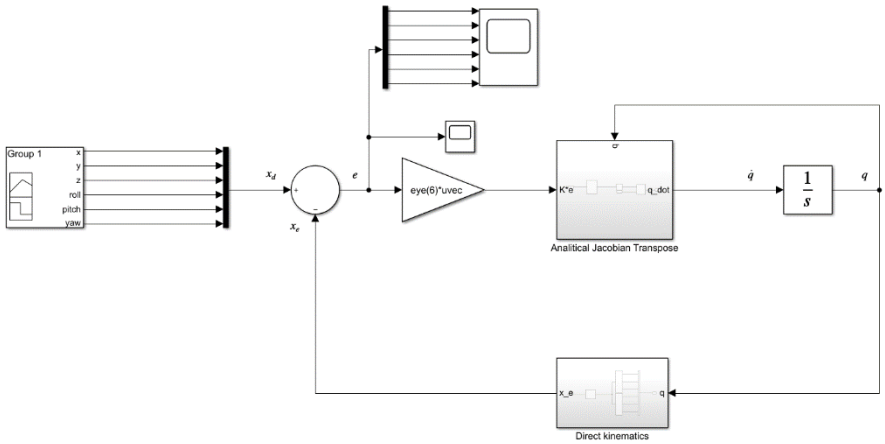

Figure 18 Inverse kinematics with Jacobian transpose scheme for Spherical manipulator

It is worth to underline that the overall structure of this system is the same of the one used for the Three-Link planar arm, where the only difference is related to the desired operational space x_d. In fact we have verified that given the following manipulator pose:

$$x = -3.623$$
$$y = 3.7247$$
$$z = 1.4142$$
$$roll = 0$$
$$pitch = 0.7854$$
$$yaw = 1.0472$$

that corresponds to the joint space $q = (\pi/3 \quad \pi/4 \quad 2)^T$, the error converges to zero as represented in *figure 19*.

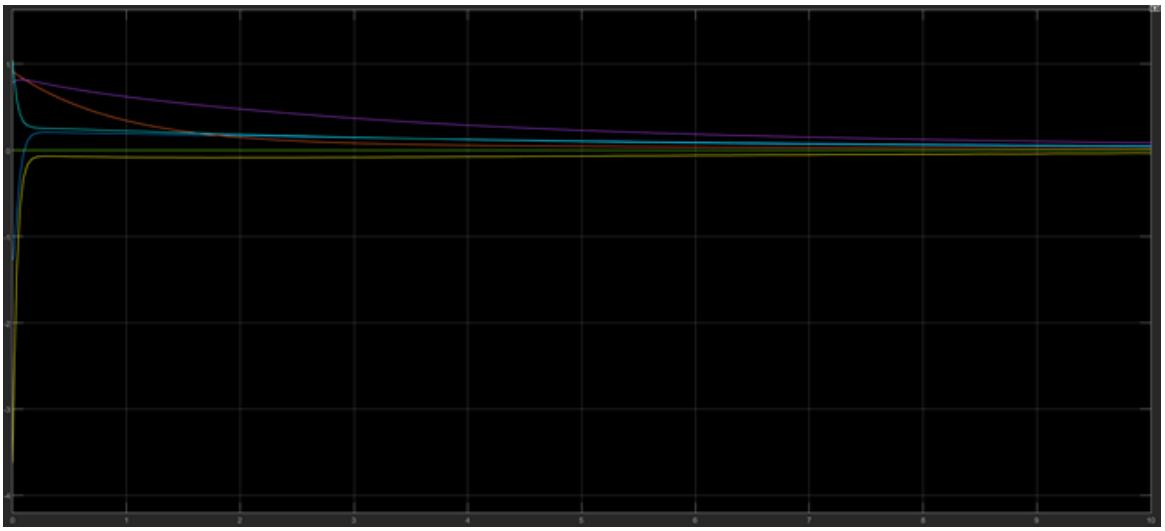

Figure 19 Error Dynamics

6. Redundancy

Redundancy of a manipulator is related with *n* the number of DOFs of the structure, *m* the number of variables describing the operational space and $r \leq m$ the number of variables needed to describe the task space.

When $n > m$ the manipulator is said intrinsically redundant, which means that the robot has DOFs more than the minimum number required to accomplish movement inside the space.

When $n > r$ the manipulator is said functionally redundant, that means the robot is redundant just for that task, but not in the entire space in which is located.

What happen is that in case of redundancy, there are $n - r$ redundant DOFs.

In order to study the redundancy, let us consider the differential kinematic equation:

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

where v_e is the (r×1) vector of end-effector velocity for a given task, J is the (r×n) Jacobian matrix that can be extracted from the geometric Jacobian and $\dot{q}$ is the (n×1) vector of joint velocities.

The Jacobian describes the linear mapping from the joint velocity space to the end-effector velocity space.

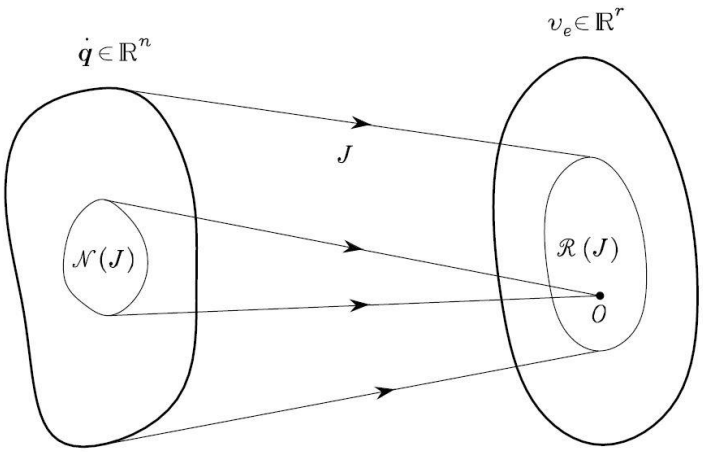

Figure 20 Jacobian mapping

The differential kinematics equation can be analyzed by means of the range and null spaces of the Jacobian matrix, in particular:
• The range space of J is the subspace R(J) in IR^r of the end-effector velocities that can be generated by the joint velocities, in the given manipulator posture.
• The null space of J is the subspace N(J) in IR^n of joint velocities that do not generate any end-effector velocity, in the given manipulator configuration.

If the Jacobian has full rank, it follows that:

$$dim(R(J)) = r \qquad dim(N(J)) = n - r$$

thus, redundant manipulators have an existing Null space, exploitable to generate internal motions with particular additional objectives.

Our purpose is to apply these reasonings to our manipulators, three-link and spherical. To do that, MATLAB and Simulink environments have been used again.

6.1 Three-Link planar arm redundancy

A three-link planar manipulator has three joints (n = 3). It is defined into a bidimensional space, the variables m needed to completely describe the end-effector frame in a plane are 3: x, y for the position and θ for the orientation with respect to the z axis (m = 3). Since n = m, this manipulator is not intrinsically redundant.

Nevertheless, if a specific task to assign is not interested in the orientation variable but on the position only, the Three-Link becomes functionally redundant (r = 2 => n > r).

Looking at the differential kinematic equation:

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

v_e is now the (2 x 1) vector of the end-effector velocity to accomplish a positioning task on the plane; J is the corresponding (2 x 3) Jacobian matrix extracted from the geometric Jacobian; $\dot{q}$ is the (3 x 1) vector of joints velocities.

When the robot is not in singular configurations (full rank Jacobian):
dim(N(J)) = n – r = 1

Exploiting the MATLAB code of the definition of the three-link arm, the geometric Jacobian can be computed by the command '`J = Three_Link.jacob0(q)`' where q is the chosen joint space:

```
q = [pi/4 -pi/4 pi/4]
```

The Jacobian associated is:

```
J =

   -2.8284   -0.7071   -0.7071
    4.8284    2.7071    0.7071
         0         0         0
         0         0         0
         0         0         0
    1.0000    1.0000    1.0000
```

where, as already said, only the two first rows must be considered for positioning task.

```
J_position =

   -2.8284   -0.7071   -0.7071
    4.8284    2.7071    0.7071
```

After the computation of the geometric Jacobian, the MATLAB command '`null(J_position)`' allows to compute a base for vectors belonging to the Null space.

Resulting null space is:

```
null =

   -0.3015
    0.3015
    0.9045
```

After the computation of the null space, we performed a simulation on Simulink, to verify the correctness of achieved results. Below, the scheme adopted for the simulation:

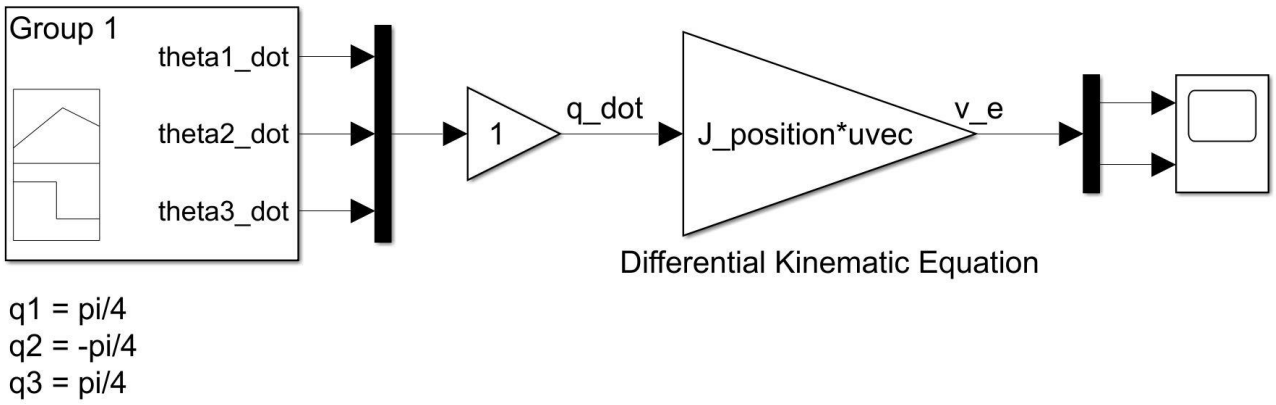

Figure 21 Simulink scheme for redundancy analysis

where input signals are the joint velocities and have been set as the null space basis computed by MATLAB

$$\dot{\vartheta} = \begin{bmatrix} -0.3015 \\ 0.3015 \\ 0.9045 \end{bmatrix}$$

The gain block '1' is simply a multiplier to verify that the behaviour is correct not just with the basis of the Null but with every its multiple.
The gain block 'Differential Kinematic Equation' performs the computation of the end-effector velocities.

By means of a scope results can be shown:

Figure 22 End-effector velocities

First plot is the velocity $\dot{x}$, the second represents $\dot{y}$.

$$v_e = \begin{bmatrix} -1.1 * 10^{-16} \cong 0 \\ 0 \end{bmatrix}$$

In this way, it is verified the presence of joint velocities that do not produce any motion at the end-effector. The same result is achieved changing the value of the unitary gain.

Trying to compute by MATLAB a base for the Null space, also considering orientation Jacobian, the result is an empty vector: this is a proof that a non-redundant manipulator has no Null.

6.2 Spherical manipulator redundancy

The spherical manipulator is not redundant since it has only three joint variables to describe a configuration in the space. Even for this case we have taken into account the position tasking only, so the first three rows of J. Because of this it follows that the $\dim(N(J)) = 0$.
In fact if we give to the robot under analysis an input joint space, for example $q = (\pi/3 \quad \pi/2 \quad 2)^T$ and we compute the command `Null_Jac = null(Jac)` Matlab gives as response `Null_Jac = 3×0 empty double matrix.`

A possible solution to have a $N(J)$ with a non-zero dimension is to consider our manipulator in a singularity configuration, because in this case we have that the $\dim(\mathrm{R(J)})$ decreases, thus following in an increasing for $\dim(\mathrm{N(J)})$, because the equation $\dim(\mathrm{R(J)}) + \dim(\mathrm{N(J)}) = n$ always holds.
By some simple mathematical operation is possible to demonstrate that the Spherical manipulator is in a singular configuration if and only if $\theta_2 = 0\,, 2\pi$.
As a result, if we give to this manipulator the joint space previously defined with the exception of $\theta_2 = 0$, we obtain that:

```
Null_Jac =

    0.3714
    0.9285
         0
```

By means of the Simulink scheme reported in *figure 23*

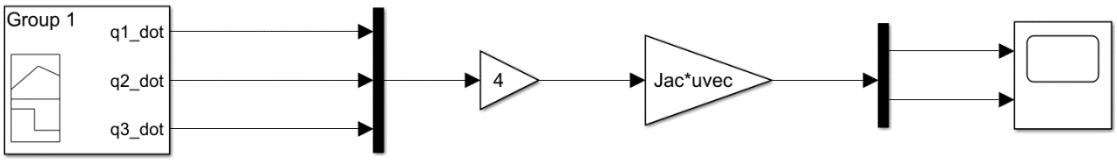

Figure 23 Redundancy analysis

if we give as input variables the velocity defined by $\mathrm{N(J)}$, we can obtain an end-effector linear velocity constantly equal to zero, as depicted in *figure 24*.

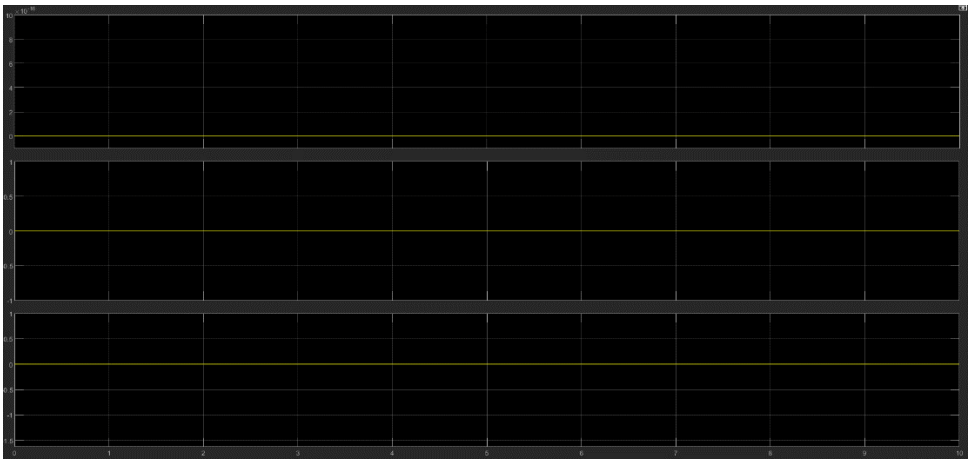

Figure 24 End-effector linear velocity

In this case is important to consider that the joint velocity given by $\mathrm{N(J)}$ is a property *only* of the spherical manipulator for the singularity configuration we have considered, it is not a velocity that can be applied for a given time interval, because if the spherical robot slightly changes its configuration, more in detail if θ_2 becomes not equal to zero, the Spherical arm is not in a singular configuration anymore and the $\dim(\mathrm{N(J)})$ becomes again equal to zero.