UNIVERSITÀ DEGLI STUDI DI GENOVA

DIBRIS

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY, BIOENGINEERING, ROBOTICS AND SYSTEM ENGINEERING

MODELLING AND CONTROL OF MANIPULATORS

Third Assignment

Jacobian Matrices and Inverse Kinematics

Author:

Colomba Ilaria
Ierardi Ambra
Malatesta Federico

Student ID:

s4829201
s4843302
s4803603

Professors:

Giovanni Indiveri
Enrico Simetti
Giorgio Cannata

Tutors:

Andrea Tiranti
Francesco Giovinazzo
George Kurshakov

December 28, 2023

Contents

Mathematical expression	Definition	MATLAB expression
$< w >$	World Coordinate Frame	w
${}^{a}_{b}R$	Rotation matrix of frame $< b >$ with respect to frame $< a >$	aRb
${}^{a}_{b}T$	Transformation matrix of frame $< b >$ with respect to frame $< a >$	aTb
${}^{a}O_{b}$	Vector defining frame $< b >$ wit respect to frame $< a >$	aOb

Table 1: Nomenclature Table

1 Assignment description

The third assignment of Modelling and Control of Manipulators focuses on the definition of the Jacobian matrices for a robotic manipulator and the computation of its inverse kinematics.
The third assignment consists of three exercises. You are asked to:

- Download the .zip file called MOCOM-LAB3 from the Aulaweb page of this course.
- Implement the code to solve the exercises on MATLAB by filling in the predefined files. In particular, you will find two different main files: *"ex1.m"* for the first exercise and *"ex2_ex3.m"* for the second and third exercises.
- Write a report motivating your answers, following the predefined format on this document.
- **Putting code in the report is not an explanation!**

1.1 Exercise 1

Given the CAD model of the robotic manipulator from the previous assignment and using the functions already implemented:
Q1.1 Compute the Jacobian matrices for the manipulator for the following joint configurations:

- $\mathbf{q}_1 = [1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8]$
- $\mathbf{q}_2 = [0.3, 1.4, 0.1, 2.0, 0, 1.3, 0]$
- $\mathbf{q}_3 = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.]$
- $\mathbf{q}_4 = [1, 1, 1, 1, 1, 1, 1]$

1.2 Exercise 2

In the second exercise the model of a Panda robot by Franka Emika is provided. The robot geometry and jacobians can be easily retrieved by calling the following built-in functions: *"getTransform()"* and *"geometricJacobian()"*.
Q2.1 Compute the cartesian error between the robot end-effector frame ${}^b_eT$ and the goal frame ${}^b_gT$.
${}^b_gT$ must be defined knowing that:

- The goal position with respect to the base frame is ${}^bO_g = [0.55, -0.3, 0.2]^\top (m)$
- The goal frame is rotated of $\theta = \pi/6$ around the y-axis of the robot end-effector initial configuration.

Q2.2 Compute the desired angular and linear reference velocities of the end-effector with respect to the base: ${}^b\nu^*_{e/0} = \alpha \cdot \begin{bmatrix} \omega^*_{e/0} \\ v^*_{e/0} \end{bmatrix}$, such that $\alpha = 0.2$ is the gain.
Q2.3 Compute the desired joint velocities. (Suggested matlab function: *"pinv()"*).
Q2.4 Simulate the robot motion by implementing the function: *"KinematicSimulation()"*.

1.3 Exercise 3

Repeat the Exercise 2, by considering a tool frame rigidly attached to the robot end-effector according to the following specifications:

$$eRt = \begin{bmatrix} cos(\phi) & -sin(\phi) & 0 & 0 \\ sin(\phi) & cos(\phi) & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \phi = -44.98(deg), {}^e O_t = [0, 0, 21.04]^\top (cm)$$

Q3.1 Compute the cartesian error between the robot tool frame ${}^b_tT$ and the goal frame ${}^b_gT$.
${}^b_gT$ must be defined knowing that:

- The goal position with respect to the base frame is ${}^bO_g = [0.55, -0.3, 0.2]^\top (m)$
- The goal frame is rotated of $\theta = \pi/6$ around the y-axis of the robot tool frame initial configuration.

Q3.2 Compute the angular and linear reference velocities of the tool with respect to the base:
${}^{b}\nu^{*}_{e/0} = \alpha \cdot \begin{bmatrix} \omega^{*}_{e/0} \\ v^{*}_{e/0} \end{bmatrix}$, such that $\alpha = 0.2$ is the gain.
Q3.3 Compute the desired joint velocities. (Suggested matlab function: *"pinv()"*).
Q3.4 Simulate the robot motion by implementing the function: *"KinematicSimulation()"*.
Q3.5 Comment the differences with respect to Exercise2.

2 Exercise 1

2.1 Q1.1

The purpose of the first exercise is to compute the Jacobian matrices for the manipulator for four different given joints configurations, given the CAD model of the robotic manipulator from the previous assignment. The model of the robot is the following:

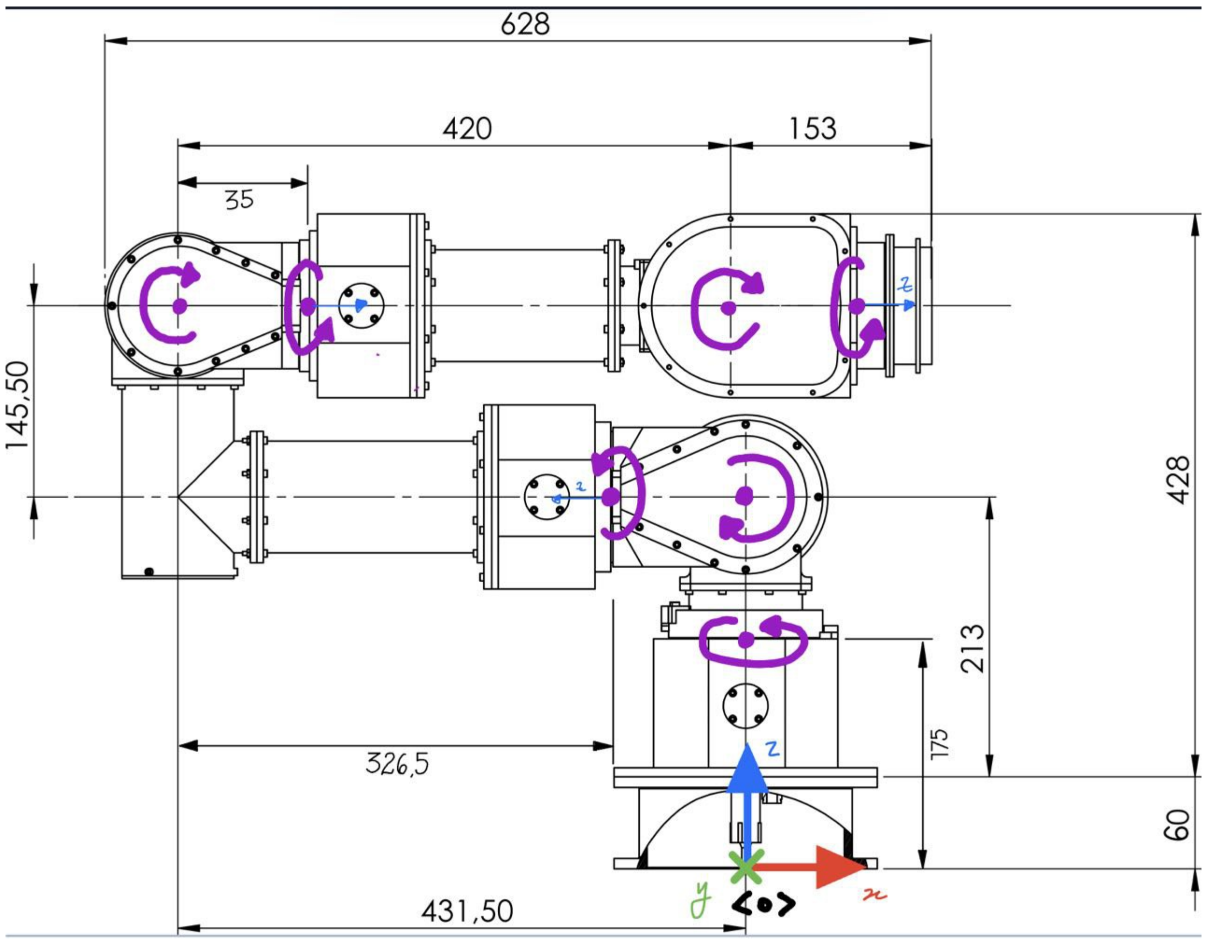


Figure 1: CAD model of the robot

The Jacobian matrix describes the relationship between the velocity of the end-effector and the velocity of the joints: it determines how a change in the joints velocities affects the end-effector velocity.
The given joint configurations are:

- $\mathbf{q}_1 = [1.8, 1.8, 1.8, 1.8, 1.8, 1.8, 1.8]$
- $\mathbf{q}_2 = [0.3, 1.4, 0.1, 2.0, 0, 1.3, 0]$
- $\mathbf{q}_3 = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.]$
- $\mathbf{q}_4 = [1, 1, 1, 1, 1, 1, 1]$

In order to compute the Jacobian, some functions already implemented during the second assignment were used:

- The function **BuildTree.m**, which builds the tree of frames for the chosen manipulator. The output of the function is ${}^i_jT$, a 3 dimensional matrix, suitable for defining tree of frames. ${}^i_jT$ represents the transformation matrix between the *i-th* and *j-th* frames;
- The function **DirectGeometry.m**, which returns ${}^i_jT_q$. The output of the function is a transformation matrix ${}^i_jT_q$ which represents the transformation between the base of the joint *i* and its following frame, taking into account the actual rotation/translation of the joint itself;

- The function **GetDirectGeometry.m**: the output of the function is the vector of matrices ${}^{i}_{j}T_q$ containing the transformation matrices from link *i* to link *j* for the input of joint configurations **q**. These transformation matrices, between each link, are computed calling the function **DirectGeometry.m**;
- The function **GetTransformationWrtBase.m** returns ${}^{b}_{i}T$. It is the transformation matrix from the manipulator base to the *i-th* joint.

In order to compute the Jacobian matrix, the function **GetJacobian.m** was implemented. This function takes as inputs:

- ${}^{b_i}_{e_i}T$ which are the transformation matrices from *i-1* to joint *i* for the current configuration of joints;
- ${}^{b}_{e}T$ which is the current transformation matrix from base to the end-effector;
- *joinType* which is the vector identifying the joint types: 0 if a joint is a revolute, 1 if it is a prismatic one.

This function returns the end-effector Jacobian matrix J for the manipulator, whose current transformation is described by ${}^{b}_{e}T$.

The Jacobian matrix can be split into two parts: the *linear Jacobian matrix* and the *angular Jacobian matrix*. The first one puts in relation the linear velocities of the end-effector to the joint velocities, while the second one relates the angular velocities of the end-effector to the joint velocities. Both parts have the number of rows equal to three and the number of columns equal to the number of joints in the robot.

The computation of each part of the Jacobian matrix depends on the joint's type:

- if the joint is a revolute: ${}^{0}J_i^A$ = $\mathbf{k_i}$ and ${}^{0}J_i^L$ = $\mathbf{k_i}$ x $\mathbf{r_{n/i}}$
- if the joint is prismatic: ${}^{0}J_i^A$ = 0 and ${}^{0}J_i^L$ = $\mathbf{k_i}$

where $\mathbf{r_{n/i}}$ is the translation vector of the joint *n*, which corresponds to the end-effector in this case, with respect to the frame *i-th* and $\mathbf{k_i}$ is the vector which represents the rotational axis of the joint *i-th*.

The steps to compute the Jacobian matrices are the following:

1. for each initial configuration, the transformation matrices from link *i* to link *j* is computed using the function **GetDirectGeometry.m**;
2. for each link, the transformation matrix with respect to the base is computed by the function **GetTransformationWrtBase.m**;
3. using the previous results obtained it is possible to compute the Jacobian matrix in the function **GetJacobian.m**.

The results obtained are the following:

$$J_1 = \begin{bmatrix} 0 & -0.9738 & -0.0516 & 0.4367 & 0.8629 & 0.1484 & 0.2744 \\ 0 & -0.2272 & 0.2213 & -0.8720 & 0.4756 & 0.0849 & -0.9607 \\ 1 & 0 & 0.9738 & 0.2213 & 0.1710 & -0.9853 & -0.0414 \\ -0.0847 & -0.1121 & 0.0267 & -0.0688 & 0.0221 & -0.1454 & 0 \\ 0.2515 & 0.4803 & 0.2703 & 0.0609 & 0.0127 & -0.0404 & 0 \\ 0 & -0.0253 & -0.0600 & 0.3757 & -0.1468 & -0.0254 & 0 \end{bmatrix}$$

$$J_2 = \begin{bmatrix} 0 & -0.2955 & -0.1624 & -0.3880 & -0.8925 & -0.3880 & -0.0172 \\ 0 & -0.9553 & -0.0502 & 0.9215 & -0.3711 & 0.9215 & 0.0112 \\ 1 & 0 & 0.9854 & -0.0170 & 0.2563 & -0.0170 & 0.9998 \\ 0.1198 & 0.6787 & 0.0824 & 0.2375 & -0.0572 & 0.1410 & 0 \\ -0.3156 & 0.2100 & -0.1956 & 0.1075 & 0.1359 & 0.0594 & 0 \\ 0 & 0.3369 & 0.0036 & 0.4077 & -0.0025 & 0.0018 & 0 \end{bmatrix}$$

$$J_3 = \begin{bmatrix} 0 & 0 & -0.9950 & -0.0198 & 0.9216 & 0.1326 & 0.6340 \\ 0 & 1 & 0 & 0.9801 & -0.0587 & 0.9766 & 0.0476 \\ 1 & 0 & 0.0998 & -0.1977 & -0.3836 & 0.1692 & -0.7719 \\ -0.0115 & -0.0942 & -0.0012 & -0.2771 & 0.0097 & -0.1166 & 0 \\ 0.0690 & 0 & -0.0869 & -0.1012 & 0.0716 & -0.0321 & 0 \\ 0 & -0.0690 & -0.0115 & -0.4741 & 0.0124 & -0.0938 & 0 \end{bmatrix}$$

$$J_4 = \begin{bmatrix} 0 & -0.8415 & -0.2919 & -0.8372 & 0.5468 & -0.4559 & -0.2954 \\ 0 & 0.5403 & -0.4546 & -0.3039 & -0.4589 & 0.5384 & -0.8427 \\ 1 & 0 & 0.8415 & -0.4546 & -0.7003 & -0.7087 & -0.4501 \\ 0.3960 & 0.0230 & 0.3139 & -0.0359 & -0.0587 & -0.1285 & 0 \\ 0.0088 & 0.0358 & 0.0050 & -0.3878 & 0.0693 & 6.4350e-04 & 0 \\ 0 & -0.3380 & 0.1116 & 0.3254 & -0.0912 & 0.0831 & 0 \end{bmatrix}$$

each one corresponding to the relative joint configuration.
It is possible to notice that the last three components of the last column of each Jacobian matrix are all zeros, due to the fact that the angular Jacobian is, as previously mentioned, the cross product of $\mathbf{k}_n$ with $\mathbf{r}_{n/n}$, which is identically zero, so the product itself will be a vector of zeros. This is a useful point to check whether the algorithm was correctly implemented.

3 Exercise 2

In the second exercise it was required to deal with a model of **Panda robot** by Franka Emika, in particular in the situation where the end-effector was considered free, meaning that the tool connected was ignored. The end-effector of the robot is located in the last joint of the manipulator. The robot geometry and jacobian can be easily retrieved by calling the following built-in functions: **getTransform()** and **geometricJacobian()**.

3.1 Q2.1

The first goal of the second exercise is to compute the **cartesian error** between the robot end-effector frame ${}^b_eT$ and the goal frame ${}^b_gT$ knowing that:

- The goal position with respect to the base frame is ${}^bO_g = [0.55, -0.3, 0.2]^\top (m)$
- The goal frame is rotated of $\theta = \pi/6$ around the y-axis of the robot end-effector initial configuration.

The Cartesian error is a measure of how far the end-effector is from the desired position, and it is used to determine the necessary adjustments to do on the joint angles and positions in order to move the end-effector to the goal position.

The cartesian error is composed by two components: the linear error and the angular error.

The current position of the end-effector is represented by the transformation matrix ${}^b_eT$ (from the base frame to the end-effector frame), and the goal position is represented by the transformation matrix ${}^b_gT$ (from the base frame to the goal frame).

To calculate the position, or linear, error, first it's necessary to calculate the transformation matrix from the end-effector frame to the goal frame, which is ${}^e_gT$. This is done by multiplying the pseudo-inverse of ${}^b_eT$ (obtained using `pinv` function in order to avoid computation problems if there were not full rank matrices) with ${}^b_gT$.

The linear error is then the translational part of this transformation matrix ${}^e_gT$, which is the fourth column of the matrix. This gives you a vector that represents the difference in x, y, and z coordinates between the current position of the end-effector and the goal position. As last point, it is necessary to project it on the base frame, by premultiplying it with ${}^b_eR$, the rotation matrix extracted by ${}^b_eT$

The orientation, or angular, error is the error between the current orientation of the robot's end-effector and the desired (goal) one.

It can be computed from ${}^e_gT$, calculated in the previous step. The angular error is obtained by the rotational part of this transformation matrix ${}^e_gT$, which can be represented in an angle-axis form, where the direction of the vector represents the axis of rotation, and the angle represents how much it is rotated.

This vector can be calculated from the rotational part of ${}^e_gT$ using the function **ComputeInverseAngleAxis.m**. This angle gives a measure of how much the end-effector has to rotate (and around which axis) to match the goal orientation.

3.2 Q2.2

In the second part, it is required to compute the desired angular and linear velocities of the end-effector with respect to the base: ${}^b\nu^*_{e/0} = \alpha \cdot \begin{bmatrix} \omega^*_{e/0} \\ v^*_{e/0} \end{bmatrix}$, such that $\alpha = 0.2$ is the gain (scalar value that determines the speed at which the end-effector will move).

The sought-after angular and linear reference velocities of the end-effector with respect to the base are the velocities that the end-effector has to move at. These velocities are used to control the movement of the end-effector such that it will reach the goal position, and this is a key element to achieve a specific task.

The desired angular reference velocity of the end-effector with respect to the base refers to the angular velocity that the end-effector is wanted to rotate, about its own axes with respect to the base frame. The desired linear reference velocity of the end-effector with respect to the base is the velocity that the end-effector is required to move with respect to the base frame, as in the previous case. It is important to note that both linear and angular velocities are defined with respect to the base frame: this is because the inverse kinematics problem is to find the joint velocities that will result in the desired end-effector velocities, and the base frame is the reference frame of the robot.

Supposing the goal to be steady in its position, it is possible to compute the desired angular and linear velocities as following:

$$\omega^*_{\mathbf{e/0}} = \alpha \cdot \mathbf{e_a}$$

$$\mathbf{v}^*_{\mathbf{e/0}} = \alpha \cdot \mathbf{e_l}$$

where $\mathbf{e_a}$ is the angular error, and $\mathbf{e_l}$ is the linear one.

So, it is possible to define the column matrix $\dot{x}$ as:

$$\dot{\mathbf{x}} = \begin{bmatrix} \omega^*_{\mathbf{e/0}} \\ \mathbf{v}^*_{\mathbf{e/0}} \end{bmatrix}$$

3.3 Q2.3

In the third part of the second exercise, the aim is to compute the desired joint velocities with the suggested Matlab function *pinv()*.
The desired joint velocities are important elements in inverse kinematics because they allow an efficient control of the end-effector movement and can be used to achieve a specific task and improve the performance of the robot.

The desired joint velocities, called $\dot{q}$, are computed using the column vector $\dot{x}$ previously obtained, and the pseudo-inverse of the Jacobian matrix of the end-effector with respect to the base using the Matlab function *pinv(^b_eJ)*. It is necessary to compute the pseudo-inverse of the Jacobian because this matrix is not always full rank, thus it is not always invertible.

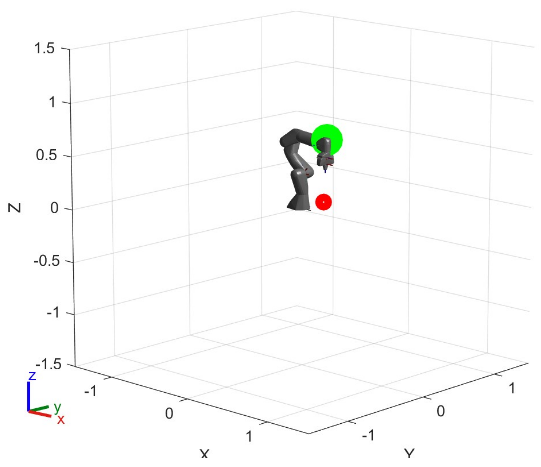
3.4 Q2.4

In the last part of the second exercise, it was asked to simulate the robot motion implementing the function **KinematicSimulation.m**. This function takes as input:

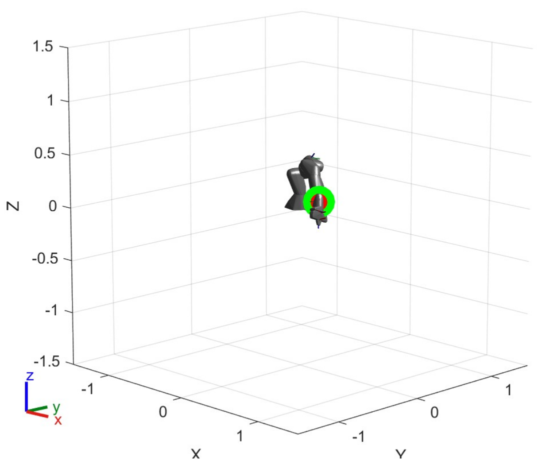
- the current robot configuration;
- the joints velocity;
- the sampling time;
- the lower joints bound, that is the lower bound in the range of positions which the joints can assume;
- the upper joints bound, that is the opposite of the lower bound.

and it returns the new joint configuration by updating the current robot configuration and saturating the joints positions, in case the new computed positions exceed the bounds.

The starting and the ending robot configurations are the following:



(a) Starting configuration



(b) Ending configuration

Figure 2: Q2.4

In order to have a good look at the superposition of the green frame, of the end-effector, and the red one, of the goal, the cartesian frames were rotated with respect to the ones proposed by the Matlab plot.
As it is possible to see in the figures, the robot reaches the goal position and its motion is well controlled.

4 Exercise 3

In the last exercise, it was proposed the same model of **Panda robot**, this time in the situation in which a tool rigidly attached to the end-effector was taken into account.
The Rotation matrix between the end-effector and the tool is the following one:

$$
{}^{e}_{t}R = \begin{bmatrix} cos(\phi) & -sin(\phi) & 0 \\ sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

where ϕ is -44.98 degrees, and the distance vector between the two is ${}^{e}O_t = [0, 0, 0.2104]^\top (m)$, which is constantly of the same length, since the tool is a rigid body.

4.1 Q3.1

As first thing, the **cartesian error** between the robot tool frame and the goal frame was computed, knowing that the position of the goal, with respect to the base frame, was exactly as in the previous point, and the rotation between the goal and the tool frames was of π/6 on the *y axis*. To compute then the transformation matrix of the goal with respect to the base, it was necessary to compute the following matrix product, in order to get the *rotation matrix* of the goal with respect to the base frame, instead of the tool frame, which was meaningless in the calculation of the cartesian error:

$$
{}^{b}_{g}R =^{b}_{t} R *^{t}_{g} R
$$

Then, the transformation matrix was composed as:

$$
{}^{b}_{g}T = \begin{bmatrix} {}^{b}_{g}R & {}^{b}_{g}O \\ 0_{1x3} & 1 \end{bmatrix}
$$

At this point, it was required to compute the error, both the **angular** and the **linear** one. In order to do so, as in the previous point, is was necessary to refresh at each time step the actual position of the tool and its transformation matrix with respect to the base.
Then, the **Jacobian matrix of the tool with respect to base** ${}^{b}_{t}J$, used later on in the computation of the desired joint velocities, was computed as the multiplication of the **rigid body Jacobian S** by the **Jacobian of the end-effector with respect to the base** ${}^{b}_{e}J$:

$$
S = \begin{bmatrix} 1_{3x3} & 0_{3x3} \\ A & 1_{3x3} \end{bmatrix}
$$

$$
{}^{b}_{t}J = S *^{b}_{e} J
$$

where A is the *skew-symmetric* matrix computed starting from the vector product operator applied on the distance vector between the end-effector and the tool:

$$
A = \begin{bmatrix} 0 & -{}^{e}_{t}O_3 & {}^{e}_{t}O_2 \\ {}^{e}_{t}O_3 & 0 & -{}^{e}_{t}O_1 \\ -{}^{e}_{t}O_2 & {}^{e}_{t}O_1 & 0 \end{bmatrix}
$$

being each subscript the corresponding component of the vector.
At this point, the transformation matrix from the tool to the goal is computed as:

$$
{}^{t}_{g}T = pinv({}^{b}_{t}T) *^{b}_{g} T
$$

where *pinv* stands for pseudo-inverse.
At this point, the errors are computed projecting the actual position and rotation of the tool with respect to the goal: the **linear error** is computed as the distance between the tool and the goal projected on the base frame:

$$
\mathbf{e_l} =^{b}_{t} R * {}^{\mathbf{t}}_{\mathbf{g}}\mathbf{O}
$$

being ${}^{b}_{t}R$ the rotation matrix extracted from the transformation matrix from base to tool, and ${}^{t}_{g}O$ the distance vector from tool to goal extracted by the transformation matrix just computed.
Then, the **angular error** is computed as follows:

$$
\mathbf{e_a} =^{b}_{t} R * \theta * \mathbf{v}
$$

where θ is the angle and **v** is the vector obtained exploiting the **ComputeInverseAngleAxis.m** function, which, staring from a rotation matrix, in this case the one from the tool to the goal, computes the corresponding couple of angle and axis of rotation.

4.2 Q3.2

The reference velocities of the tool with respect to base are in the following vector: ${}^{\mathbf{b}}\nu^*_{\mathbf{t/0}} = \alpha \cdot \begin{bmatrix} \omega^*_{\mathbf{t/0}} \\ \mathbf{v}^*_{\mathbf{t/0}} \end{bmatrix}$ with α, that is the gain, equal to 0.2. $\omega^*_{\mathbf{t/0}}$ is computed as the multiplication of α and the angular error, while $\mathbf{v}^*_{\mathbf{t/0}}$ is obtained multiplying α with the linear error.

4.3 Q3.3

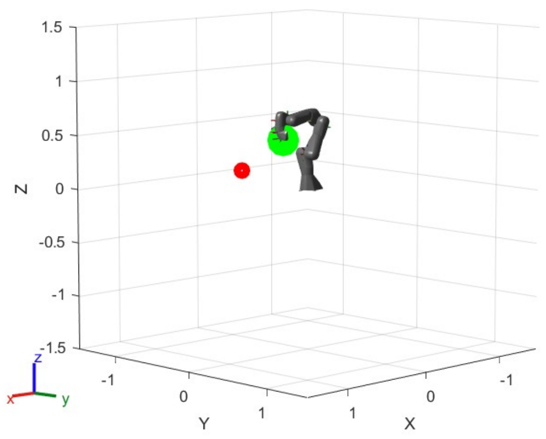
The desired joint velocities, in the case of the use of the tool, are computed as:

$$\dot{\mathbf{q}} = pinv({}^{b}_{t}J) * {}^{\mathbf{b}}\nu^*_{\mathbf{t/0}}$$

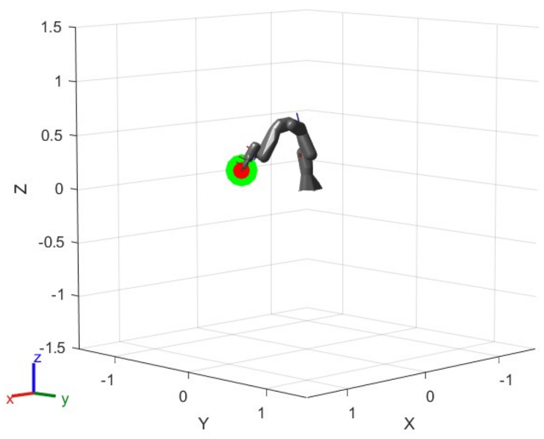
being *pinv* again the pseudo-inverse, and ${}^{b}_{t}J$ the Jacobian of the tool with respect to base, computed in the section **Q3.1**.

4.4 Q3.4

The initial and final configuration of the robot are the following ones: it is well visible how the tool frame (in green) and the goal frame (in red) are overlapped in the second picture.



(a) Starting configuration



(b) Ending configuration

Figure 3: Q3.4

4.5 Q3.5

In the exercise 3, since the tool was considered, the final configuration of the robot is with the tool oriented and positioned as described by the goal; while, concerning the case of the exercise 2, the tool is ignored, and so the end-effector is going to reach the goal position and orientation, instead of the tool as in the previous case. Also, since the time interval is the same, but the length of the considered part of the robot (*i.e.* from the base to the end-effector, or from the base to the tool) is different, according to the chosen goal position and orientation, the desired velocities of the joints will be different.