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:

Gavagna Veronica

Student ID:

s5487110

Professors:

Giovanni Indiveri
Enrico Simetti
Giorgio Cannata

Tutors:

Andrea Tiranti
Francesco Giovinazzo

January 15, 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

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 is **mandatory** and 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 the predefined files. In particular, you will find two different main files: *"ex1.m"* for the first exercise and *"ex2.m"* for the second and third exercises.
- Write a report motivating your answers, following the predefind format on this document.

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:

- $\mathbf{q}_1 = [1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3]$
- $\mathbf{q}_2 = [1.3, 0.4, 0.1, 0, 0.5, 1.1, 0]$
- $\mathbf{q}_3 = [1.3, 0.1, 0.1, 1, 0.2, 0.3, 1.3]$
- $\mathbf{q}_4 = [2, 2, 2, 2, 2, 2, 2]$

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 ${}^b_{ge}T$.
${}^b_{ge}T$ must be defined knowing that:

- The goal position with respect to the base frame is ${}^bO_g = [0.6, 0.4, 0.4]^T$
- The goal frame is rotated of $\theta = -\pi/4$ around the z-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 transformation matrix:

$$eTt = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

- The goal position with respect to the base frame is ${}^bO_g = [0.6, 0.4, 0.4]^T$
- The goal frame is rotated of $\theta = -\pi/4$ around the z-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.

Q3.6 Test the algorithm for a new tool goal, knowing that the transformation matrix of the goal with respect to the robot base is:

$$bTgt = \begin{bmatrix} 0.9986 & -0.0412 & -0.0335 & 0.6 \\ 0.0329 & -0.0163 & 0.9993 & 0.4 \\ -0.0417 & -0.9990 & -0.0149 & 0.4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2 Exercise 1

In the previous assignment, a CAD model of an open kinematic chain with 7 dof and the joints are all revolute joints (RJ) was given. So the same functions implemented before are used in this assignment:

- "*BuildTree()*" function: this function builds the tree of frames for the manipulator given by returning the transformation matrix between the i-th and (i+1)-th frames.
- "*GetDirectGeometry.m*" function: it returns a 3D matrix containing how the transformation matrices attached to the joint will rotate if the joint rotates, given the joint configuration q_i, the transformation matrix and the joint type. These parameters are the inputs of the "*DirectGeometry.m*" function, and the computation of the transformation matrix depends on the joint type.
- "*GetTransformationWrtBase()*" function: From the transformation matrices given if the joint rotates, I can compute the transformation w.r.t. the base by multiplying the transformation matrix of the i-th link with the one of the previous one until you get to the base.

These functions are necessary to compute the Jacobian matrices for the manipulator.

2.1 Q1.1 - Solution

Given the following joint configurations:

- $\mathbf{q}_1 = [1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3]$
- $\mathbf{q}_2 = [1.3, 0.4, 0.1, 0, 0.5, 1.1, 0]$
- $\mathbf{q}_3 = [1.3, 0.1, 0.1, 1, 0.2, 0.3, 1.3]$
- $\mathbf{q}_4 = [2, 2, 2, 2, 2, 2, 2]$

Through "*GetDirectGeometry.m*", I can compute the transformation matrix attached to the joint if the joint rotates (biTei).
After that, I can compute the transformation matrix between the base and the i-th joint (bTi) for each joint using "*GetTransformationWrtBase()*".
This matrix is used for the function "*GetJacobian()*" to obtain the Jacobian matrix given the bTi matrix, the joint type and the number of links.

The Jacobian matrices are transformations from joint space velocities to cartesian space angular and linear velocities. It provides the relation between joint velocities and the end-effector velocities of a robot manipulator. Columns of the Jacobian matrix are associated with the joints of the robot. Each column in the Jacobian matrix represents the effect on end-effector velocities due to variation in each joint velocity. Similarly, rows of the Jacobian matrix can also be split into two parts:

- The first three rows are associated with linear velocities of the end-effector.
- The last three rows are associated with the angular velocities of the end-effector due to changes in velocities of all the joints combined.

$$J_{e/0} = \begin{bmatrix} J^A_{e/0} \\ \\ J^L_{e/0} \end{bmatrix}$$

The Jacobian matrix is, in our case, a 6x7 because we have 7 joints and the result depends on the type of the joint:

- Revolute joint (RJ): The first three rows are corresponding to the vector representing the coordinates of the (i)-th joint w.r.t. the base. The last three rows are the cross product between the vector representing the coordinates of the (i)-th joint w.r.t. the base and the vector representing the coordinates of the (i)-th joint w.r.t. the end-effector.

  $$J^A_{e/0} = \begin{bmatrix} \underline{k_1} & \underline{k_2} & \underline{k_3} & \dots & \underline{k_e} \end{bmatrix}$$

  Where $\underline{k_i}$ is the rotation vector of the i-th joints.

  $$J^L_{e/0} = \begin{bmatrix} \underline{k_1} \wedge \underline{r_{e/1}} & \underline{k_2} \wedge \underline{r_{e/2}} & \underline{k_3} \wedge \underline{r_{e/3}} & \dots & \underline{k_e} \wedge \underline{r_{e/3}} \end{bmatrix}$$

  Where $\wedge$ stands for cross product and $\underline{r_{e/i}}$ is the distance from the i-th joints and the end-effector.

- Prismatic joint (PJ): The first three rows are the vector null because there is no rotation. The last three rows are the vector representing the coordinates of the (i)-th joint w.r.t. the end-effector.

$$J^A_{e/0} = \begin{bmatrix} 0_{3xn} \end{bmatrix}$$

$$J^L_{e/0} = \begin{bmatrix} \underline{k_1} & \underline{k_2} & \underline{k_3} & ... & \underline{k_e} \end{bmatrix}$$

Where n is the number of joints.

The resulting Jacobian Matrices for each configuration are:

- $\mathbf{q}_1 = [1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3]$

$$J_1 = 1.0e+05 * \begin{bmatrix} 0 & 0 & -0.0001 & -0.0016 & -0.0013 & 0.0020 & 0.0030 \\ 0 & 0 & -0.0003 & -0.0004 & -0.0005 & -0.0021 & -0.0010 \\ 0.0018 & 0.0027 & 0.0037 & 0.0070 & 0.0069 & 0.0056 & 0.0055 \\ 0.5523 & -0.1633 & 1.0332 & 0.3291 & 0.7914 & 0.0000 & 0.0000 \\ -0.0256 & 0.7377 & 0.5439 & 3.0012 & 0.5378 & 0.8598 & 0.0000 \\ 0 & 0 & 0.0601 & 0.2347 & 0.1862 & 0.3266 & 0.0000 \end{bmatrix}$$

- $\mathbf{q}_2 = [1.3, 0.4, 0.1, 0, 0.5, 1.1, 0]$

$$J_2 = 1.0e+05 * \begin{bmatrix} 0 & 0 & -0.0003 & -0.0011 & -0.0010 & -0.0000 & 0.0008 \\ 0 & 0 & -0.0009 & -0.0032 & -0.0029 & 0.0005 & 0.0014 \\ 0.0018 & 0.0027 & 0.0031 & 0.0057 & 0.0056 & 0.0041 & 0.0050 \\ 0.0675 & -0.7381 & -0.6805 & 0.8996 & -0.5699 & 0.0000 & 0.0000 \\ 0.2776 & -0.1581 & 0.1631 & 2.7422 & 0.4390 & 0.6286 & 0.0000 \\ 0 & 0 & -0.0077 & 1.7152 & 0.1317 & -0.0734 & 0.0000 \end{bmatrix}$$

- $\mathbf{q}_3 = [1.3, 0.1, 0.1, 1, 0.2, 0.3, 1.3]$

$$J_3 = 1.0e+05 * \begin{bmatrix} 0 & 0 & -0.0003 & -0.0012 & -0.0012 & -0.0004 & 0.0003 \\ 0 & 0 & -0.0010 & -0.0040 & -0.0038 & -0.0022 & -0.0008 \\ 0.0018 & 0.0027 & 0.0028 & 0.0046 & 0.0043 & 0.0009 & 0.0009 \\ 0.0811 & 0.5199 & 0.6010 & 0.7221 & -1.1478 & 0 & 0.0000 \\ -0.1284 & 0.1491 & 0.0627 & 2.2180 & 0.5357 & 0.1340 & 0.0000 \\ 0 & 0 & 0.0815 & 2.1045 & 0.1608 & 0.3390 & 0.0000 \end{bmatrix}$$

- $\mathbf{q}_4 = [2, 2, 2, 2, 2, 2, 2]$

$$J_4 = 1.0e+05 * \begin{bmatrix} 0 & 0 & -0.0002 & -0.0017 & -0.0015 & 0.0005 & -0.0010 \\ 0 & 0 & 0.0004 & 0.0006 & 0.0009 & 0.0040 & 0.0042 \\ 0.0018 & 0.0027 & 0.0037 & 0.0069 & 0.0070 & 0.0078 & 0.0077 \\ 0.1536 & -0.4895 & 0.7785 & -0.4734 & 1.2500 & -0.0000 & 0.0000 \\ 0.7459 & 1.7265 & 0.5620 & 2.2429 & 0.4927 & 1.1985 & 0.0000 \\ 0 & 0 & -0.0222 & -0.3069 & 0.2133 & -0.6123 & 0 \end{bmatrix}$$

As we were aspected all the Jacobian matrices are 6x7, and the last three elements of the last column are 0 because the distance between the end-effector and the end-effector is null.

3 Exercise 2

This exercise focuses on the problem of inverse kinematics, so how to compute velocity joint vectors to achieve a desired configuration (called goal position).
To complete this exercise, some of the functions implemented in the first assignment are used and updated:

- "*ComputeAngleAxis()*" function: this function returns the rotation matrix using the Rodriguez formula given the angle of rotation and the axis vector. This function also calls the function "*skeDim3()*" to compute the skew-symmetric matrix of a vector of dimension 3.
  This function is used to compute the rotation matrix of the goal w.r.t. the end-effector. This matrix is used to get the initial rotation matrix of the goal w.r.t. the base just by multiplying the rotation matrix of the end-effector w.r.t. the base and the rotation matrix of the goal w.r.t. the end-effector.
- "*ComputeInverseVersorLemma()*" function: it is the updated version of the function "*ComputeInverseAngleAxis()*" that returns the angle of rotation and the axis vector. In the updated version, two matrices of rotation are given as inputs and the angle of rotation θ is computed by the difference between the two angles of rotation associated with the two matrices, obtained by the following formula:

  $$\theta = \arccos \frac{(tr(R) - 1)}{2}$$

  After computing the angle of rotation θ, three different situations are considered:
  - in the case of $\theta = 0$, the axis vector is [0; 0; 0], so no rotation occurs
  - in the case of $0 < \theta < \pi$, the axis vector is computed by the difference between the axis vector associated with the first matrix and the axis vector associated with the second matrix, using the following formulas:

    $$\mathbf{v} = \frac{vex(R)}{\sin\theta}$$

    and

    $$vex(R) = \frac{1}{2}\begin{bmatrix} R_{3,2} - R_{2,3} \\ R_{1,3} - R_{3,1} \\ R_{2,1} - R_{1,2} \end{bmatrix}$$

  - and by exclusion, in the case of $\theta = \pi$, the axis vector is equal to the column vector of the sum of the two matrices which is not zero.

Since Jacobian gives a direct relation between end-effector velocities ($\dot{X}$) and joint velocities ($\dot{q}$), the solution to the inverse kinematics problem is to compute the end-effector velocities and Jacobian pseudo-inverse. Finally, to find the joint velocities using the following formula:

$$\dot{q} = pinv({}^{b}_{e}J) * \dot{X}$$

Once we find the joint velocities, the end-effector will reach the desired goal pose in a given time. Once the goal pose is reached the robot stops.

3.1 Q2.1/Q2.2/Q.2.3 - Solution

To compute the joint velocities ($\dot{q}$), ${}^{b}_{e}J$ and $\dot{x}$ are needed:

- ${}^{b}_{e}J$ is the Jacobian matrix of the end-effector w.r.t. the base, returned from the Matlab function *"geometricJacobian()"*
- $\dot{X}$ is desired angular and linear reference velocities of the end-effector w.r.t. the base and can be calculated by multiplying the gain (given in the text) for the cartesian error.

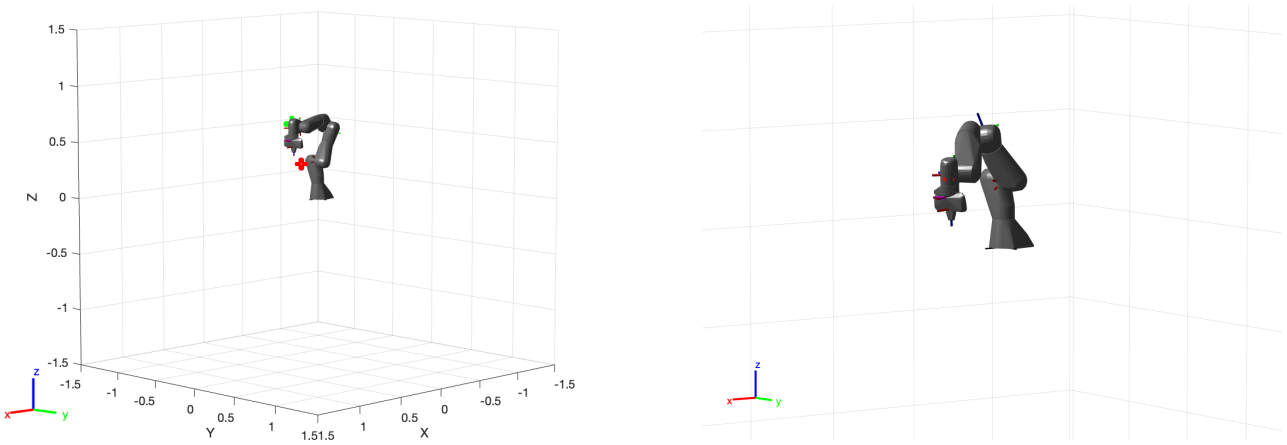
The cartesian error can be obtained by computing the angular error and the linear error between the goal and the end-effector:

- The angular error with respect to the base is obtained by multiplying the rotation matrix from the end-effector to the base for the rotation vector ρ. The rotation vector ρ is computed by multiplying the angle of rotation θ and the versor of rotation h returned by the function *"ComputeInverseVersorLemma()"*, which has as inputs the rotation matrix of the goal w.r.t. the base (which is constant) and the rotation matrix of the end-effector w.r.t. the base (which changes every time the robot moves and it is given by the Matlab function *"getTransform()"*).
- The linear error w.r.t. the base is obtained by subtracting the vector of the end-effector w.r.t. the base from the vector of the goal w.r.t. the base (that is constant because the goal does not move).

3.2 Q2.4 - Solution

To simulate the motion of the robot, it is necessary to update the position of the robot given the joints velocities $\dot{q}$, the actual configuration of the robot, the sample time and the upper and lower joints bounds. In the function *"KinematicSimulation()"* the new configuration of the robot is computed using the formula $q_{new} = q + \dot{q} * ts$ where q is the actual joints configuration, $\dot{q}$ is the joints velocity and ts is the time sample fixed to 0.5. After computing the new robot configuration, the result is checked to be within the limits and the correct value is returned.
The simulation ends when $\dot{X}$ is less than 0.01 so that means that the goal position is reached.

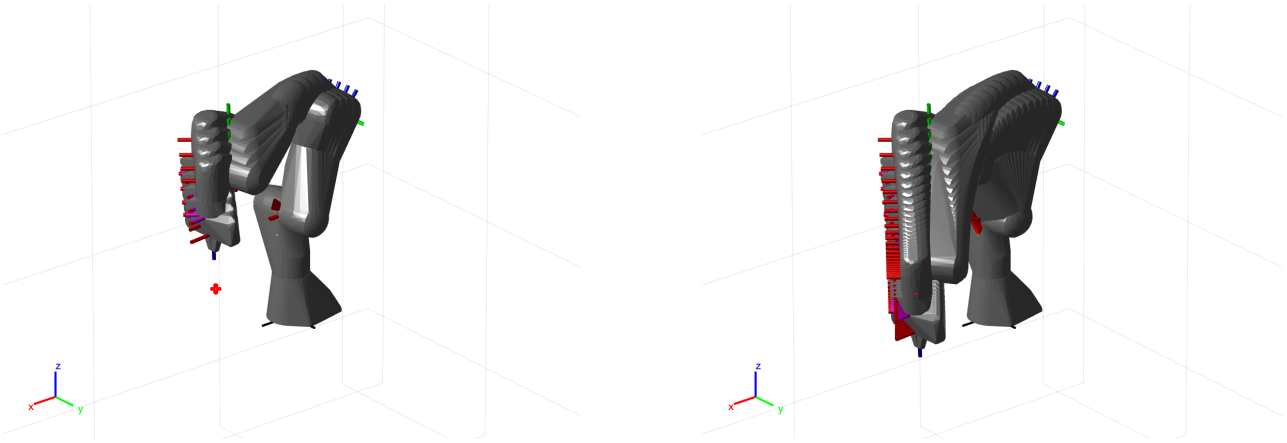
Graphics of the simulation are shown in Figure 1 and Figure 2.



(a) Initial configuration

(b) Final configuration

Figure 1: Graphs Exercise 2 - Initial and final configuration



(a) Intermediate configuration

(b) Final configuration zoomed

Figure 2: Graphs Exercise 2 - Intermediate and final configuration zoomed with motion

As shown in the graphs, the goal position is reached by the end-effector, so the red cross is in the middle of the end-effector.

4 Exercise 3

Also, this exercise focuses on the problem of inverse kinematics, so how to compute velocity joint vectors to achieve a desired configuration (called goal position).
To complete this exercise, some of the functions implemented in the first assignment are used and updated:

- "*ComputeAngleAxis()*" function: this function returns the rotation matrix using the Rodriguez formula given the angle of rotation and the axis vector. This function also calls the function "*skeDim3()*" to compute the skew-symmetric matrix of a vector of dimension 3.
  This function is used to compute the rotation matrix of the goal w.r.t. the tool. This matrix is used to get the initial rotation matrix of the goal w.r.t. the base just by multiplying the rotation matrix of the tool w.r.t. the base and the rotation matrix of the goal w.r.t. the tool (given by the text).
- "*ComputeInverseVersorLemma()*" function: it is the updated version of the function "*ComputeInverseAngleAxis()*" that returns the angle of rotation and the axis vector. In the updated version, two matrices of rotation are given as inputs and the angle of rotation θ is computed by the difference between the two angles of rotation associated with the two matrices, obtained by the following formula:

  $$\theta = \arccos \frac{(tr(R) - 1)}{2}$$

  After computing the angle of rotation θ, three different situations are considered:
  - in the case of $\theta = 0$, the axis vector is [0; 0; 0], so no rotation occurs
  - in the case of $0 < \theta < \pi$, the axis vector is computed by the difference between the axis vector associated with the first matrix and the axis vector associated with the second matrix, using the following formulas:

    $$\mathbf{v} = \frac{vex(R)}{\sin\theta}$$

    and

    $$vex(R) = \frac{1}{2}\begin{bmatrix} R_{3,2} - R_{2,3} \\ R_{1,3} - R_{3,1} \\ R_{2,1} - R_{1,2} \end{bmatrix}$$

  - and by exclusion, in the case of $\theta = \pi$, the axis vector is equal to the column vector of the sum of the two matrices which is not zero.

Since Jacobian gives a direct relation between tool velocities ($\dot{X}$) and joint velocities ($\dot{q}$), the solution to the inverse kinematics problem is to compute the tool velocities and Jacobian pseudo-inverse. Finally, to find the joint velocities using the following formula:

$$\dot{q} = pinv({}^{b}_{t}J) * \dot{X}$$

Once we find the joint velocities, the tool will reach the desired goal pose in a given time. Once the goal pose is reached the robot stops.

4.1 Q3.1/Q3.2/Q3.3 - Solution

To compute the joint velocities ($\dot{q}$), ${}^{b}_{t}J$ and $\dot{x}$ are needed:

- ${}^{b}_{t}J$ is the Jacobian matrix of the tool w.r.t. the base, is given by multiplying the static Jacobian matrix of the tool w.r.t. the base ${}^{e}_{t}S$ and the Jacobian matrix of the end-effector ${}^{b}_{e}J$ returned from the Matlab function *"geometricJacobian()"*.
  The static Jacobian matrix of the tool w.r.t. the base ${}^{e}_{t}S$ is given by the following formula:

  $${}^{e}_{t}S = \begin{bmatrix} I_{3x3} & 0_{3x3} \\ skew({}^{b}_{e}R * {}^{e}_{t}r)^{T} & I_{3x3} \end{bmatrix}$$

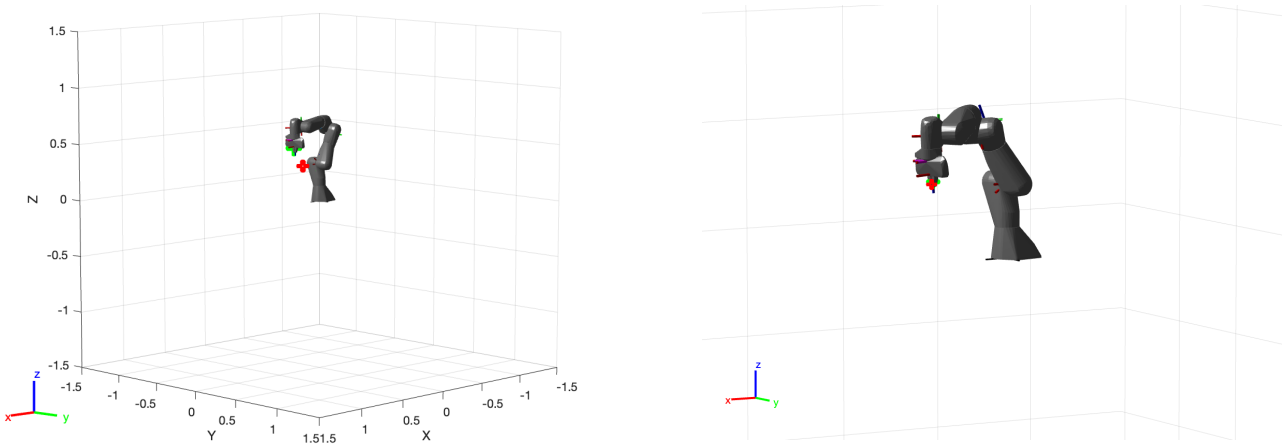
  Where ${}^{b}_{e}R$ is the rotation matrix of the end-effector w.r.t. the base, given by the transformation matrix ${}^{b}_{e}T$ returned by the Matlab function *"getTransform()"*. And ${}^{e}_{t}r$ is the distance vector between the tool and the end-effector, given by the transformation matrix ${}^{e}_{t}T$.
- $\dot{X}$ is desired angular and linear reference velocities of the tool w.r.t. the base and can be calculated by multiplying the gain (given in the text) for the cartesian error.

The cartesian error can be obtained by computing the angular error and the linear error between the goal and the tool:

- The angular error with respect to the base is obtained by multiplying the rotation matrix of the tool w.r.t. the base for the rotation vector ρ.
  The rotation vector ρ is computed by multiplying the angle of rotation θ and the axis of rotation h returned by the function *"ComputeInverseVersorLemma()"*, which has as inputs the rotation matrix of the goal w.r.t. the base (which is constant) and the rotation matrix of the tool w.r.t. the base (which changes every time the robot moves and it is given as explained before).
- The linear error w.r.t. the base is obtained by subtracting the vector of the tool w.r.t. the base from the vector of the goal w.r.t. the base (that is constant because the goal does not move).

4.2 Q3.4 - Solution

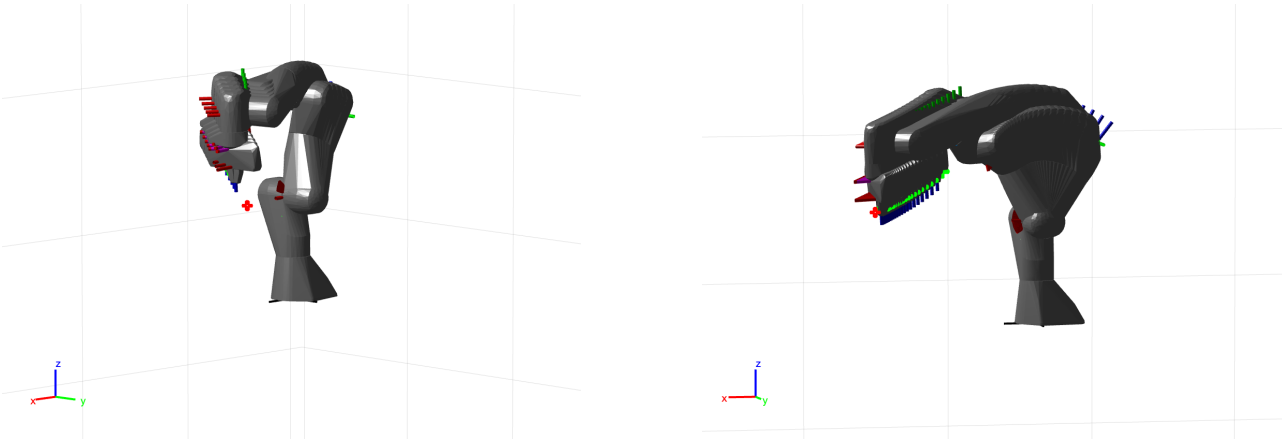
To simulate the motion of the robot, it is necessary to update the position of the robot given the joints velocities $\dot{q}$, the actual configuration of the robot, the sample time and the upper and lower joints bounds. In the function *"KinematicSimulation()"* the new configuration of the robot is computed using the formula $q_{new} = q + \dot{q} * ts$ where q is the actual joints configuration, $\dot{q}$ is the joints velocity and ts is the time sample fixed to 0.5. After computing the new robot configuration, the result is checked to be within the limits and the correct value is returned.
The simulation ends when $\dot{X}$ is less than 0.01 so that means that the goal position is reached.
Graphics of the simulation are shown in Figure 3 and Figure 4.



(a) Initial configuration

(b) Final configuration

Figure 3: Graphs Exercise 3.4 - Initial and final configuration



(a) Intermediate configuration

(b) Final configuration zoomed

Figure 4: Graphs Exercise 3.4 - Intermediate and final configuration zoomed with motion

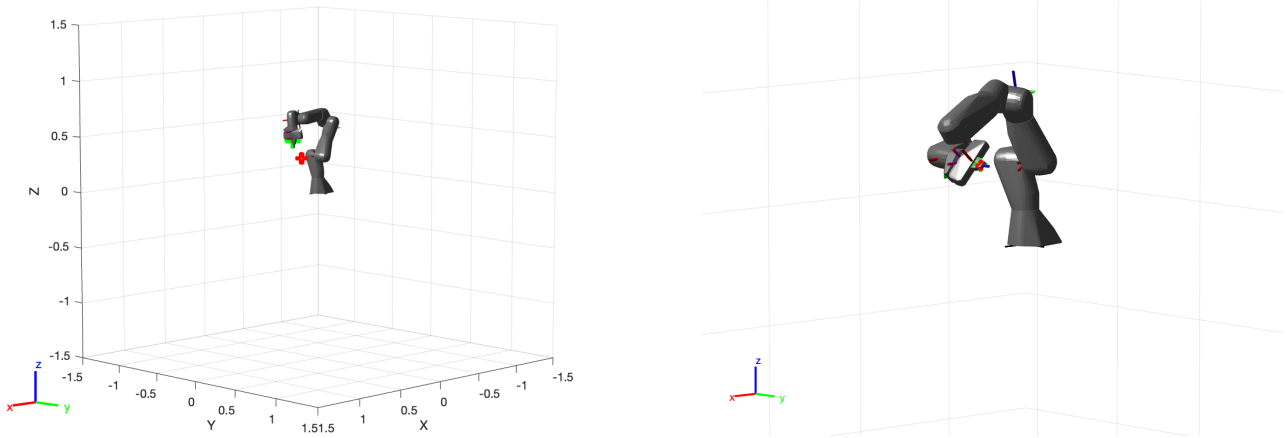
As shown in the graphs, the goal position is reached by the tool, so the red cross is very close to the tool.

4.3 Q3.5 - Solution

In comparison with the previous exercise, the goal position is reached by the tool and not by the end-effector. The only difference in terms of the transformation matrix is the given by the distance vector because the tool is rigidly connected to the end-effector at a distance of 0.2 along the z-axis of the end-effector, so the rotation movement is the same as the previous exercise.
In terms of realistic representation, the second one represents better a real experiment because the goal position has to be reached by the tool, which represents the grabber, and not by the end-effector.

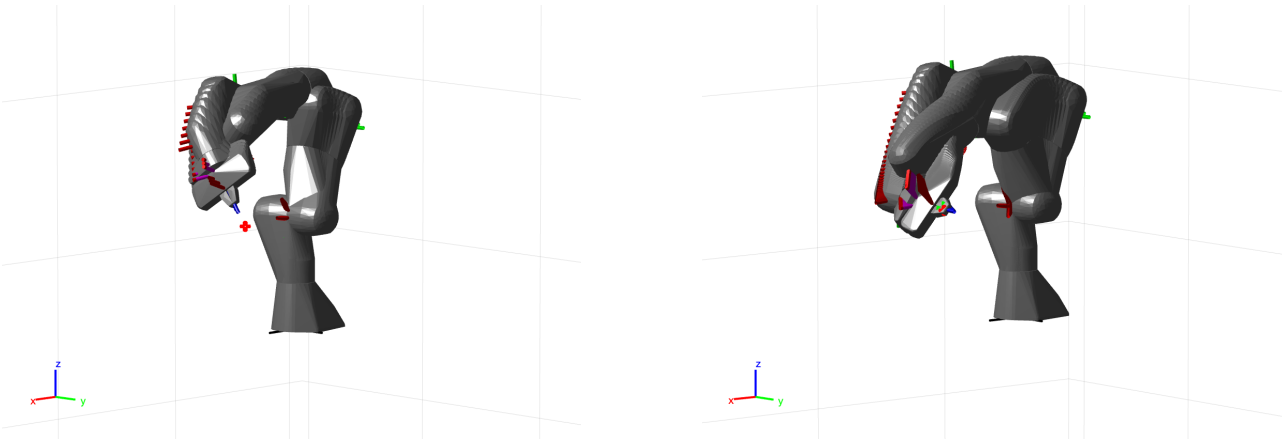
4.4 Q3.6 - Solution

In this part of the third exercise, the matrix of the goal with respect to the robot base is changed and given by the text.
The steps and calculations are the same as in exercise 3.1, so graphics of the simulation are shown in Figure 5 and Figure 6.



(a) Initial configuration (b) Final configuration

Figure 5: Graphs Exercise 3.6 - Initial and final configuration



(a) Intermediate configuration (b) Final configuration zoomed

Figure 6: Graphs Exercise 3.6 - Intermediate and final configuration zoomed with motion

As shown in the graphs, the goal position is reached by the tool, so the red cross is in the middle of the tool. In this exercise, it is necessary to consider the limit cases (when $\theta = 0$ or $\theta = \pi$) because otherwise, the robot will never reach the goal position.

5 List of figures

List of Figures