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:

Condorelli Valentina
Meschini Marco
Piacenza Enrico

Student ID:

s4945679
s6273938
s4878469

Professors:

Giovanni Indiveri
Enrico Simetti
Giorgio Cannata

Tutors:

Andrea Tiranti
Francesco Giovinazzo
George Kurshakov

January 9, 2024

Contents

1 Assignment description **3**
1.1 Exercise 1 . . . 3
1.2 Exercise 2 . . . 3
1.3 Exercise 3 . . . 3

2 Introduction **5**

3 Exercise 1 **5**

4 Exercise 2 **7**
4.1 Q2.1 . . . 7
4.2 Q2.2 . . . 8
4.3 Q2.3 . . . 8
4.4 Q2.4 . . . 8

5 Exercise 3 **9**
5.1 Q3.1 . . . 9
5.2 Q3.2 . . . 9
5.3 Q3.3 . . . 10
5.4 Q3.4 . . . 10
5.5 Q3.5 . . . 10

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 ${}^b_gT$.
${}^b_{ge}T$ 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:

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

Q3.1 Compute the cartesian error between the robot tool frame ${}^b_tT$ and the goal frame ${}^b_gT$.
${}^b_{gt}T$ 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 Introduction

In the third assignment, we addressed the *Inverse Kinematic problem*, which consists of computing the desired joint velocities, knowing the end-effector velocity. To do this, we used the following formula:

$$\underline{\dot{q}} = J^{\#} \cdot \underline{\dot{x}}^{*} \tag{1}$$

To this purpose, the first step was to compute the Jacobian *J*, which is a matrix of dimension $6 \times n$, where *n* is the number of joints of the robot. With this matrix, it is possible to obtain the joint velocities by means of its pseudoinverse $J^{\#}$, which is necessary to invert a non-square matrix and overcome eventual singularities. For this computation, we used the Matlab function *pinv()* which implements the *Moore-Penrose Pseudoinverse*.
Once the Jacobian was computed, we obtained the desired linear and angular velocities for the end-effector as:

$$\underline{\dot{x}} = \begin{bmatrix} {}^0\underline{\omega}_{e/0} \\ {}^0\underline{v}_{e/0} \end{bmatrix} = \Lambda \cdot \begin{bmatrix} {}^0\underline{\rho} \\ {}^0\underline{r} \end{bmatrix} + \begin{bmatrix} {}^0\underline{\omega}_{g/0} \\ {}^0\underline{v}_{g/0} \end{bmatrix} \tag{2}$$

In this equation, $\underline{\rho}$ is the angular error, while $\underline{r}$ is the linear error. On the other hand, Λ is the gain matrix composed by:

$$\Lambda = \begin{bmatrix} \lambda_1 \cdot \mathbb{I}_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & \lambda_2 \cdot \mathbb{I}_{3\times3} \end{bmatrix} \tag{3}$$

3 Exercise 1

In the first exercise, it was requested to build the Jacobian for the 7-DOF manipulator analyzed in the previous laboratory.

Firstly, we created the geometric model of the manipulator using the function *BuildTree()*, which returns a vector of matrices containing, for each frame, the corresponding transformation matrix with respect to the previous one.
Secondly, we initialized some variables useful for the next computations:

- The number of links (*numberOfLinks*), initialized to 7.
- The joints type (*JointType*) which, in the configuration provided, were always rotational.
- *bTi*, a vector of matrices containing the transformation matrix of the i-th frame with respect to the base.

Before computing the Jacobian, we used the function *GetDirectGeometry()* to get a vector of transformation matrices between each i-th frame and the previous one, in the current joint configuration. Then, to calculate the Jacobian matrix, we implemented the function *GetJacobian()*, which takes as inputs:

- biTei: a vector of matrices containing the transformation matrices from the base to the i-th joint for the current joint configuration, which was computed using the function *GetTransformationWrtBase()*.
- bTe: the transformation matrix from base to the end effector.
- jointType: a vector identifying the joint type, 0 for revolute and 1 for prismatic.

The function's output is the robot's Jacobian which, for the given manipulator, was ${}^b_e J$, as the last joint coincided with the end-effector and there was no rigid tool attached to it.
For this computation, we first need to compute the *angular Jacobian*, with the following formulas for the i-th joint:

	R	P
$\underline{J}^A_{e/i}$	$\underline{k}_i$	$\underline{0}$

Table 2: R: Revolute joint; P: Prismatic joint

In Table 2, the vector $\underline{k}_i$ indicates the axis of motion.

After that, we can compute the *linear Jacobian*, with the following formulas for the i-th joint:

	R	P
$\underline{J}^L_{e/i}$	$\underline{k}_i \times \underline{r}_{e/i}$	$\underline{k}_i$

Table 3: R: Revolute joint; P: Prismatic joint

To compute the complete robot Jacobian we combined the just obtained angular and linear Jacobian, as shown in Equation 4:

$$ {}^b_e J = \begin{bmatrix} \underline{J}^A_{e/b} \\ ---- \\ \underline{J}^L_{e/b} \end{bmatrix} \tag{4} $$

To test the correctness of the function we defined 5 different initial joint configurations on which we computed the corresponding Jacobian:

- q0 = [1.3,1.3,1.3,1.3,1.3,1.3,1.3]
- q1 = [1.8,1.8,1.8,1.8,1.8,1.8,1.8]
- q2 = [0.3,1.4,0.1,2.0,0,1.3,0]
- q3 = [0,0.1,0.2,0.3,0.4,0.5,0]
- q4 = [1,1,1,1,1,1,1]

Obtaining the following results:

$$ {}^b_e J(q0) = \begin{bmatrix} 0 & -0.9636 & -0.0716 & -0.5061 & 0.8473 & -0.2905 & -0.2017 \\ 0 & 0.2675 & -0.2578 & -0.8231 & -0.4187 & 0.1496 & -0.9751 \\ 1.0000 & 0 & 0.9636 & -0.2578 & -0.3267 & -0.9451 & -0.0923 \\ 0.3626 & 0.0735 & 0.2786 & 0.0408 & -0.0428 & -0.1431 & 0 \\ 0.1691 & 0.2648 & 0.1826 & -0.1604 & 0.0221 & 0.0251 & 0 \\ 0 & 0.3042 & 0.0695 & 0.4320 & -0.1393 & 0.0480 & 0 \end{bmatrix} \tag{5} $$

$$ {}^b_e J(q1) = \begin{bmatrix} 0 & -0.9738 & -0.0516 & 0.4367 & 0.8629 & 0.1484 & 0.2744 \\ 0 & -0.2272 & 0.2213 & -0.8720 & 0.4756 & 0.0849 & -0.9607 \\ 1.0000 & 0 & 0.9738 & 0.2213 & 0.1710 & -0.9853 & -0.0414 \\ -0.0847 & -0.1121 & 0.0267 & -0.0688 & 0.0221 & -0.1454 & 0 \\ 0.2515 & 0.4803 & 0.2703 & 0.0609 & 0.0127 & -0.0404 & 0 \\ 0 & -0.0253 & -0.0600 & 0.3757 & -0.1468 & -0.0254 & 0 \end{bmatrix} \tag{6} $$

$$ {}^b_e J(q2) = \begin{bmatrix} 0 & -0.2955 & -0.1624 & -0.3880 & -0.8925 & -0.3880 & -0.0172 \\ 0 & 0.9553 & -0.0502 & 0.9215 & -0.3711 & 0.9215 & 0.0112 \\ 1.0000 & 0 & 0.9854 & -0.0170 & 0.2563 & -0.0170 & 0.9998 \\ 0.1198 & 0.6787 & 0.0824 & 0.2375 & -0.0572 & 0.1410 & 0 \\ -0.3156 & 0.2100 & -0.1956 & 0.1075 & 0.1359 & 0.0594 & 0 \\ 0 & 0.3369 & 0.0036 & 0.4077 & -0.0025 & 0.0018 & 0 \end{bmatrix} \tag{7} $$

$$ {}^b_e J(q3) = \begin{bmatrix} 0 & 0 & -0.9950 & -0.0198 & 0.9216 & 0.1326 & 0.6340 \\ 0 & 1.0000 & 0 & 0.9801 & -0.0587 & 0.9766 & 0.0476 \\ 1.0000 & 0 & 0.0998 & -0.1977 & -0.3836 & 0.1692 & -0.7719 \\ -0.0115 & -0.0942 & -0.0012 & -0.2771 & 0.0097 & -0.1166 & 0 \\ 0.0690 & 0 & -0.0869 & -0.1012 & 0.0716 & 0.0321 & 0 \\ 0 & -0.0690 & -0.0115 & -0.4741 & 0.0124 & -0.0938 & 0 \end{bmatrix} \tag{8} $$

$$ {}^{b}_{e}J(q4) = \begin{bmatrix} 0 & -0.8415 & -0.2919 & -0.8372 & 0.5468 & -0.4559 & -0.2954 \\ 0 & 0.5403 & -0.4546 & -0.3039 & -0.4589 & 0.5384 & -0.8427 \\ 1.0000 & 0 & 0.8415 & -0.4546 & -0.7003 & -0.7087 & -0.4501 \\ 0.3960 & 0.0230 & 0.3139 & -0.0359 & -0.0587 & -0.1285 & 0 \\ -0.0088 & 0.0358 & 0.0050 & -0.3878 & 0.0693 & 0.0006 & 0 \\ 0 & 0.3380 & 0.1116 & 0.3254 & -0.0912 & 0.0831 & 0 \end{bmatrix} \tag{9} $$

4 Exercise 2

4.1 Q2.1

For the first task of the second exercise, it was requested to compute the cartesian error between the robot end-effector frame, described with the transformation matrix ${}^{b}_{e}T$, and the goal frame ${}^{b}_{g}T$. To this purpose, two quantities were provided:

- ${}^{b}O_{g} = [0.55, -0.3, 0.2]^{T}$ (m) as the position of the goal w.r.t. the base frame.
- $\theta = \frac{\pi}{6}$ as the rotation of the goal frame around the y-axis of the robot end-effector initial configuration. In order to not confuse this value with the result of the *ComputeInverseAngleAxis()* used later in the code, this angle was initialized as α.

Firstly, the angle α was used to define the rotation matrix of the goal frame w.r.t. the base as follows:

$$ {}^{e}_{g}R = R\_y = \begin{bmatrix} cos(\alpha) & 0 & sin(\alpha) \\ 0 & 1 & 0 \\ -sin(\alpha) & 0 & cos(\alpha) \end{bmatrix} \tag{10} $$

Then, the rotation matrix of the goal frame w.r.t. the base was found by using the composition property of rotation matrices:

$$ {}^{b}_{g}R = {}^{b}_{e}R \cdot {}^{e}_{g}R \tag{11} $$

From it, the transformation matrix of the goal frame w.r.t. the base was found as:

$$ {}^{b}_{g}T = \begin{bmatrix} {}^{b}_{g}R & {}^{b}O_{g} \\ \mathbb{O}_{1\times 3} & 1 \end{bmatrix} \tag{12} $$

The goal was then to compute ang_err and lin_err as the elements of the cartesian error.
lin_err was computed with the following steps:

1. Find the transformation matrix between the end-effector and the goal frames as ${}^{e}_{g}T = {}^{b}_{e}T^{-1} \cdot {}^{b}_{g}T$, where ${}^{b}_{e}T$ was obtained with the already defined *getTransform()* Matlab function
2. Extract the rotation matrix ${}^{b}_{e}R$ from ${}^{e}_{g}T$ as its first three rows and columns
3. Compute the linear error as the vector ${}^{e}_{g}r$, obtained by extracting the first three elements of the last column of ${}^{e}_{g}T$, pre-multiplied by the rotation matrix ${}^{b}_{e}R$ to project it onto the base frame.
   So $lin\_err = {}^{b}_{e}R \cdot {}^{e}_{g}r$

On the other hand, ang_err was computed with the following steps:

1. Extract ${}^{e}_{g}R$ from ${}^{e}_{g}T$
2. Compute the correspondent angle-axis representation with the *ComputeInverseAngleAxis()* function, already implemented in the first assignment of this course. As a consequence, two values were obtained:
   - θ as the angle of rotation of the goal frame w.r.t. the end-effector frame
   - $\underline{v}$ as the axis of rotation
3. Compute the angular error as $ang\_err = {}^{b}_{e}R \cdot (\theta \cdot \underline{v})^{T}$, so the unique vector representing the angle-axis pre-multiplied by the rotation matrix ${}^{b}_{e}R$ to project it onto the base frame. The vector was transposed just because Matlab gave it as a row vector in the results.

4.2 Q2.2

For the second task of the second exercise, it was requested to compute the desired angular and linear reference velocities of the robot end-effector w.r.t. the base such that $\alpha = 0.2$ was the gain.[1]

Firstly, considering that the goal was not moving, the reference velocities can be computed as:

$$\begin{cases} \underline{\omega}^*_{e/b} = \alpha \cdot ang\_err \\ \underline{v}^*_{e/b} = \alpha \cdot lin\_err \end{cases} \tag{13}$$

4.3 Q2.3

For the third task of the second exercise, it was requested to compute the desired joint velocities. To this purpose, the vector containing both the angular and the linear desired velocities of the end-effector was built as:

$$\underline{\dot{x}} = \begin{bmatrix} \underline{\omega}^*_{e/b} \\ \underline{v}^*_{e/b} \end{bmatrix} \tag{14}$$

Then, the desired joint velocities were computed as:

$$\underline{\dot{q}} = {}^b_e J^{\#} \cdot \underline{\dot{x}} \tag{15}$$

In this equation, the Jacobian ${}^b_e J$ was obtained before the computation of lin_err and ang_err by using the already defined Matlab function *geometricJacobian()*. This function was particularly useful in the simulation loop so that the Jacobian could be updated at each step.

4.4 Q2.4

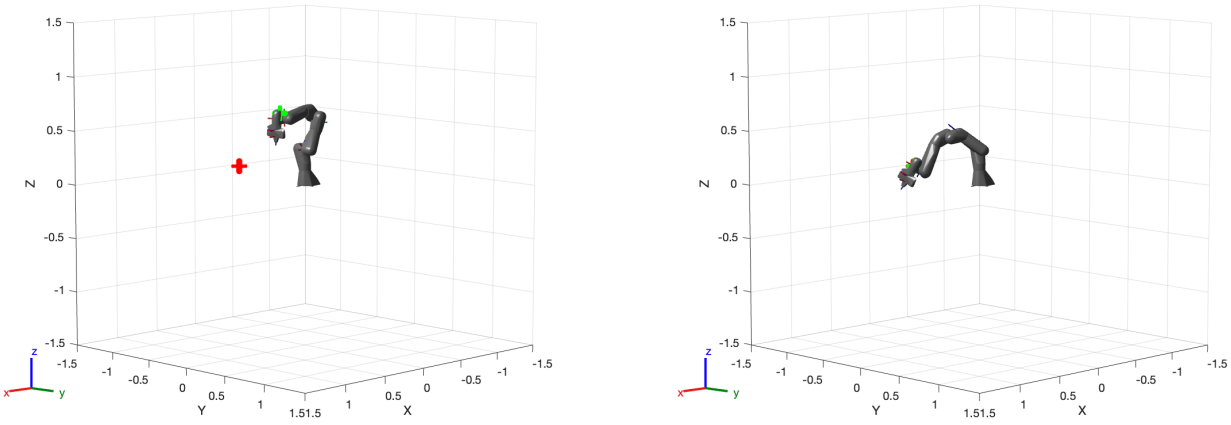
For the fourth and last task of the second exercise, it was required to implement the function *KinematicSimulation()* so that the robot motion could be simulated.

The function was implemented considering two main steps:

1. Updating q: the new joint configuration q, which was the output of the function, needed to be updated by taking into consideration the sample time *ts* and the joints velocity $\underline{\dot{q}}$ so that: $\underline{q} = \underline{q} + ts \cdot \underline{\dot{q}}$.
2. Saturating the joint positions: considering that a lower joints bound *q_min* and an upper joints bound *q_max* were provided for the new joints positions, each element of $\underline{q}$ needed to be compared with both *q_min* and *q_max*. Particularly, for the i-th element of $\underline{q}$:
   - If $\underline{q}(i) < q\_min(i)$, then $\underline{q}(i) = q\_min(i)$, to avoid going below the lower joint bound.
   - If $\underline{q}(i) > q\_max(i)$, then $\underline{q}(i) = q\_max(i)$, to prevent surpassing the upper joint bound.

With the *KinematicSimulation()* function implemented, the behaviour of the robot could finally be simulated. As it can be seen in Fig.1, at the beginning the robot is in its initial standard configuration, with the red cross as the goal to reach. In the final configuration, the end-effector of the robot reaches the goal, overlapping with it. This behaviour was expected as, for the second exercise, the tool was not considered.
To see the whole simulation, please click on the figures.

[1] Please note that this α is not the same quantity as the angle $\pi/6$.



(a) (b)

Figure 1: Initial and final configuration of the robot

5 Exercise 3

5.1 Q3.1

For the first task of the third exercise, it was requested to compute the cartesian error between the tool frame and the goal frame. The tool was rigidly attached to the end-effector of the Panda model, already implemented in the second exercise.
Initially, we built the transformation matrices between the base frame and the tool frame, as well as between the base frame and the goal frame. Then, the transformation matrix and the Jacobian between the base frame and the end-effector frame were computed. Finally, the rigid Jacobian, which allows the control of the velocities of the tool frame, was implemented as:

$$ {}^{0}S_{t/e} = \begin{bmatrix} \mathbb{I}_{3\times3} & \mathbb{O}_{3\times3} \\ [{}^{0}r_{t/e}\times]^{T} & \mathbb{I}_{3\times3} \end{bmatrix} \tag{16} $$

In this matrix, $\mathbb{I}_{3\times3}$ represents a 3 by 3 identity matrix, $\mathbb{O}_{3\times3}$ represents a 3 by 3 null matrix and $[{}^{0}r_{t/e}\times]^{T}$ represents the transpose of the skew-symmetric matrix built up on the vector containing the distance between the tool frame and the end-effector frame.

The following step, involved the computation of the Jacobian between the base frame and the tool frame. This was done as:

$$ {}^{b}_{t}J = {}^{0}S_{t/e} \cdot {}^{b}_{e}T \tag{17} $$

Finally, using the transformation matrix between the base frame and the tool frame, the linear and angular errors were computed. In particular:

$$ {}^{t}_{g}T = {}^{b}_{t}T^{-1} \cdot {}^{b}_{g}T \tag{18} $$

and then, the linear error is given by:

$$ lin\_err = {}^{b}_{t}R \cdot {}^{t}_{g}r \tag{19} $$

On the other hand, for the angular error, the angle-axis representation of the matrix ${}^{t}_{g}R$ was computed, by using the function *ComputeInverseAngleAxis()*, obtaining θ and $\underline{v}$ (defined as in Q2.1) and the angular error as:

$$ ang\_err = {}^{b}_{t}R \cdot (\theta \cdot \underline{v})^{T} \tag{20} $$

5.2 Q3.2

For the second task of the third exercise, it was requested to compute the desired angular and linear reference velocities of the robot end-effector w.r.t. the base such that $\alpha = 0.2$ was the gain (just as in Q2.2). Considering that the goal is not moving, the computation of reference velocities was given by:

$$ \begin{cases} \underline{\omega}^{*}_{t/b} = \alpha \cdot ang\_err \\ \underline{v}^{*}_{t/b} = \alpha \cdot lin\_err \end{cases} \tag{21} $$

5.3 Q3.3

To calculate the desired joint velocities, the vector with reference velocities was constructed, and then the velocities of each joint were determined using the Jacobian matrix.

$$\underline{\dot{x}} = \begin{bmatrix} \omega^*_{t/b} \\ v^*_{t/b} \end{bmatrix} \tag{22}$$

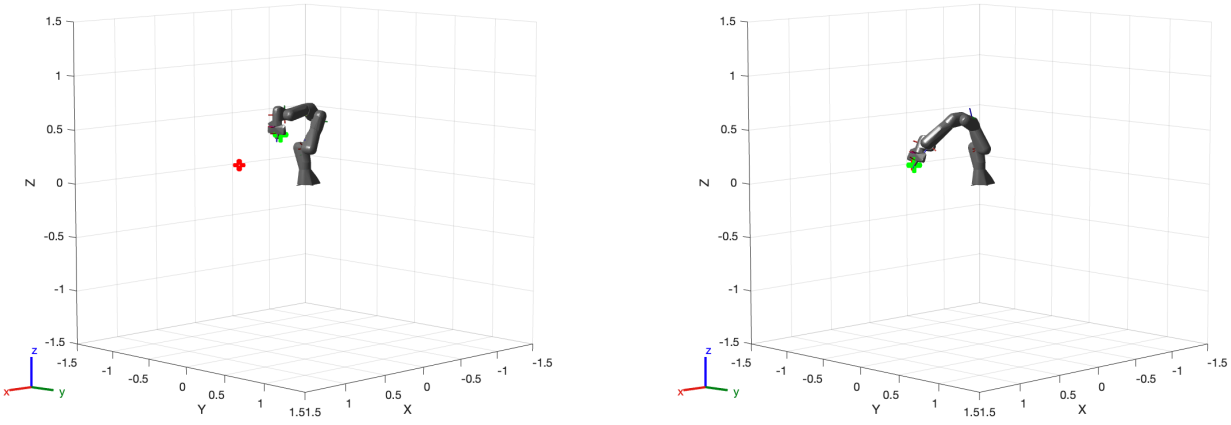
$$\dot{q} = {}^b_t J^{\#} \cdot \underline{\dot{x}} \tag{23}$$

Again, the procedure for this task was analogue to its correspondent task in exercise 2, so Q2.3

5.4 Q3.4

The implementation of the *KinematicSimulation()* function is the same as in the corresponding task of the previous exercise (in Section Q2.4). Therefore, please refer to that explanation.

As it can be seen in Fig.2, the initial configuration of the robot and the goal to reach, represented respectively by the green and by the red cross, are the same as in the second exercise (Fig.1). However, since in this case the tool was taken into consideration, the final configuration was different. Indeed, it was the tool that reached the goal, overlapping with it as expected.
To see the whole simulation, please click on the figures.



(a) (b)

Figure 2: Initial and final configuration of the robot

5.5 Q3.5

The computation remains consistent for the Jacobian of the joints up to the end-effector. The only difference is given by the tool in the third exercise, which introduces the rigid Jacobian. As a consequence, the behaviour of the robot is a bit different: in the first case, the robot reached the goal by overlapping its last joint with it, as shown in Fig.1. Whereas, in the second case, the robot reached the goal by overlapping the tool with it, as shown in Fig.2.