Università di Genova

UNIVERSITÀ DEGLI STUDI DI GENOVA

DIBRIS

DEPARTMENT OF INFORMATICS,
BIOENGINEERING, ROBOTICS AND SYSTEM ENGINEERING

MODELLING AND CONTROL OF MANIPULATORS

Third Assignment

Jacobian Matrices and Inverse Kinematics

Author:

Spluga Matteo
Nogarin Giacomo
Tozzola Michele

Student ID:

s9185902
s8654515
s8995647

Professors:

Enrico Simetti
Giorgio Cannata

Tutors:

Luca Tarasi
Simone Borelli

December 15, 2025

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 Inverse Kinematics (IK) control of a robotic manipulator.

The third assignment consists of three exercises. You are asked to:

- Download the .zip file called from the Aulaweb page of this course.
- Implement the code to solve the exercises on MATLAB by filling in the predefined files.
- Write a report motivating your answers, following the predefined format on this document.
- **Do NOT put your code in the report!**

1.1 Exercise 1

Given the geometric model of an industrial manipulator in Figure 1, you have to add a tool frame. The tool frame is rigidly attached to the robot end-effector according to the following specifications: $\Gamma_{t/e} = [\pi/10, 0, \pi/6], {}^{e}O_t = [0.3, 0.1, 0]^\top (m)$ where $\Gamma_{t/e}$ represents the YPR values from end effector frame to tool frame.

To complete this task you should modify the class *geometricModel* by adding a new method called *getToolTransformWrtBase*.

1.2 Exercise 2

Implement an inverse kinematic control loop to control the tool of the manipulator. You should be able to complete this exercise by using the MATLAB classes implemented for the previous assignment (*geometricModel*,*kinematicModel*), and also you need to implement a new class *cartesianControl* (see the template attached). The initial joint position is $q = [\pi/2, -\pi/4, 0, -\pi/4, 0, 0.15, \pi/4]^\top$, expressed in radians and meters. The procedure can be split into the following phases

Q2.1 Compute the cartesian error between the robot end-effector frame ${}^{b}_{t}T$ and the goal frame ${}^{b}_{g}T$.

The goal frame must be defined knowing that:

- The goal position with respect to the base frame is ${}^{b}O_g = [0.2, -0.7, 0.3]^\top$ (m)
- The goal frame is rotated by the YPR angles $\Gamma_g = [0, 1.57, 0]^\top$ (rad) with respect to the base frame.

Q2.2 Compute the desired angular and linear reference velocities of the tool with respect to the base: ${}^{b}\nu^{*}_{t/b} = \begin{bmatrix} \kappa_a & 0 \\ 0 & \kappa_l \end{bmatrix} \cdot {}^{b}e$, such that $\kappa_a = 0.8$,$\kappa_l = 0.8$ is the gain.

Q2.3 Compute the desired joint velocities $\dot{\tilde{q}}$

Q2.4 Simulate the robot motion by implementing the function: *"KinematicSimulation()"* for integrating the joint velocities in time.

Q2.5 Compute the end effector and tool linear and angular velocities with respect to the base frame projected on the base frame.

Remark 1: All the methods must be implemented for a generic serial manipulator. For instance, joint types, and the number of joints should be parameters.

Remark 2: The class *plotManipulators* is provided for plot generation. You must not modify its implementation, or its calls in *main.m.*

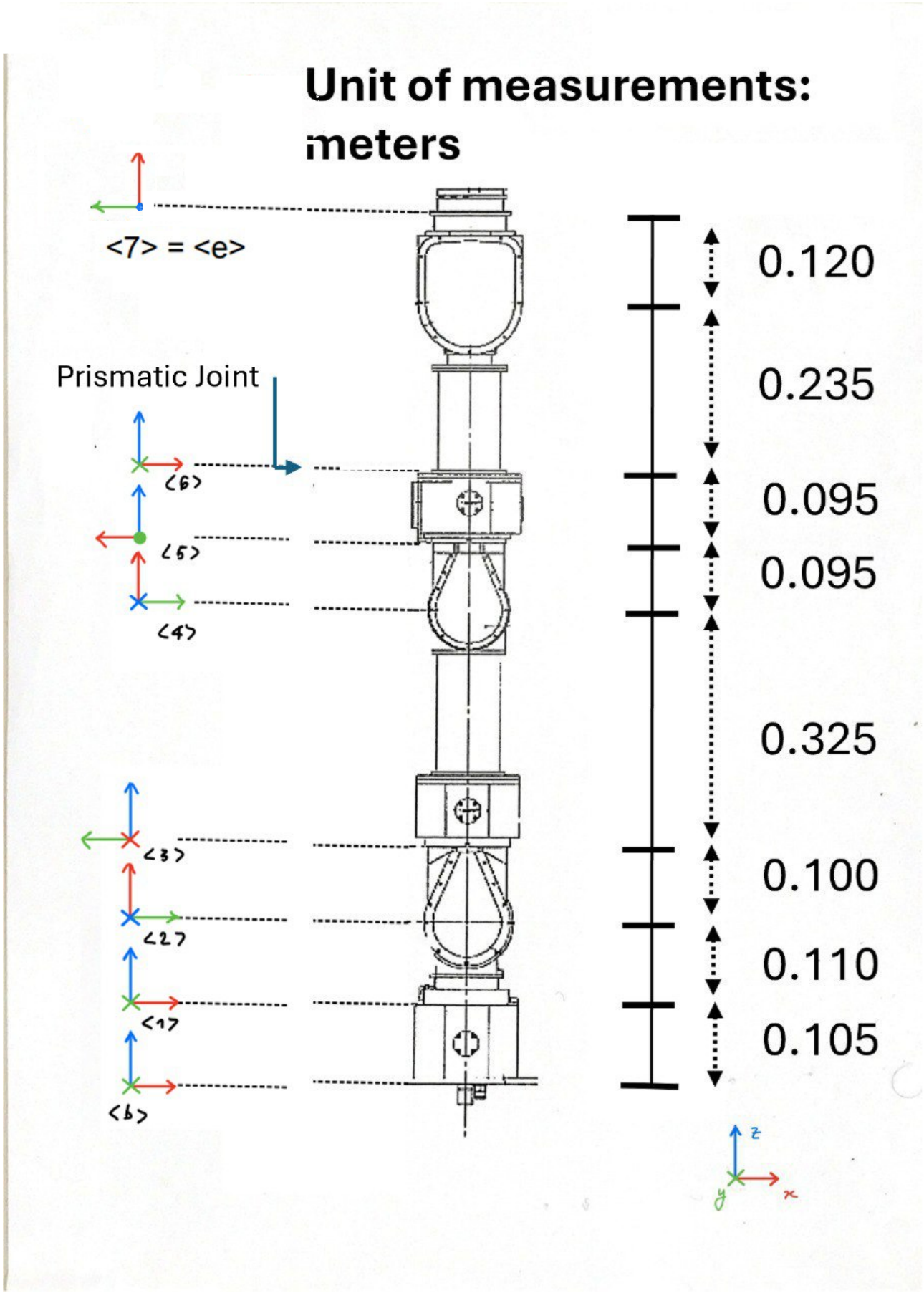


Figure 1: Manipulator CAD

2 Exercise 1

Given the CAD model of the 7 DoF robot shown in Fig. 1 considered for this assignment, the starting point of its direct kinematic analysis regards the definition of the transformation matrices of the model. This was done by filling the *BuildTree()* function. The computation of the transformation matrices was done by considering the positions and orientations already given of the reference frames of each joint of the manipulator. They also can be seen on the manipulator in Fig. 1.

The resulting transformation matrices regarding the definition of the frames in the model are the following:

$$ {}^{b}_{1}T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.1050 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{1}_{2}T = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0.1100 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}_{3}T = \begin{bmatrix} 0 & 0 & 1 & 0.1000 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{3}_{4}T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0.3250 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{4}_{5}T = \begin{bmatrix} 0 & 0 & 1 & 0.0950 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{5}_{6}T = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0.0950 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{6}_{7}T = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0.3550 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

It can be observed that, whenever the direction of the axis does not change, the rotation matrices inside the transformation ones are an identity matrix.

Now that the geometric configuration of the manipulator is identified with the computation of the transformation matrices between joint frames, the main goal is to add a tool frame, which is rigidly attached to the robot end-effector according to the following specifications:

$$ \Gamma_{t/e} = \left[\frac{\pi}{10},\ 0,\ \frac{\pi}{6}\right] \qquad {}^{e}O_{t} = \begin{bmatrix} 0.3 \\ 0.1 \\ 0 \end{bmatrix} $$

$\Gamma_{t/e}$ represents the YPR values from the end-effector frame to the tool frame. Because of this, the use of the *YPRToRot()* function was needed. This particular function converts the YPR angles to a rotation matrix which describes the rotations caused by those angles. In the *main.m* script it was indeed called the *YPRToRot()* function in order to compute this conversion, saving the result in the matrix called ${}^{e}_{t}R$.

To add the tool frame in the geometric model of the manipulator, a new method called *getToolTransformWrtBase()* was implemented. With this addition, the class *geometricModel*, which already contained the functions *updateDirectGeometry()* and *getTransformWrtBase()*, is now completed.

The*updateDirectGeometry()* function was written in order to compute the transformation matrices by taking into account the position described by vector $\underline{q}$:

$$\underline{q} = \begin{bmatrix} \frac{\pi}{2} & -\frac{\pi}{4} & 0 & -\frac{\pi}{4} & 0 & 0.15 & \frac{\pi}{4} \end{bmatrix}^\top$$

In the previous circumstance, the matrices were built considering $q = 0$, so that they could represent the transformations between the frame of one joint and the next considering their fixed geometry. In this case, the transformation matrices were calculated (based on their fixed geometry configuration) by adding their movement effect (considered with $q \neq 0$), which describes the real pose of the joints. In practice, whenever a rotational joint (rotating about the z-axis) was met, the following resulting matrix was obtained:

$${}^{i}_{i+1}T(q_i) = {}^{i}_{i+1}T(0) * T_{\text{joint}(q_i)} = {}^{i}_{i+1}T(0) \begin{bmatrix} \cos q_i & -\sin q_i & 0 & 0 \\ \sin q_i & \cos q_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For the prismatic joint, the following resulting matrix was obtained instead:

$${}^{i}_{i+1}T(q_i) = {}^{i}_{i+1}T(0) * T_{\text{joint}(q_i)} = {}^{i}_{i+1}T(0) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & q_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The *updateDirectGometry()* function utilizes parameters from the *geometricModel* constructor belonging to the *geometricModel* class.

Then, considering the *getTransformWrtBase()* function, we can describe the transformation matrices between each joint and the base frame in order to obtain the global pose for each one. More precisely, it computes the homogeneous transformation matrix ${}^{0}_{k}T$ by chaining the local transformations between consecutive joint frames. Starting from the identity matrix, each local transformation ${}^{i}_{i+1}T$ is successively multiplied, resulting in the pose of the k-th joint frame with respect to the base frame. This function relies on the values of the matrices obtained with the *updateDirectGometry()*.

The last function to take into account is finally the *getToolTransformWrtBase()* which simply represents an extension of the previous one. In fact, it only computes the transformation matrix from the manipulator base to the tool frame.

The results of this computation can be seen in different matrices subsequently shown:
Transformation matrix between the end-effector and tool frame.

$${}^{e}_{t}T = \begin{bmatrix} 0.9511 & -0.2676 & 0.1545 & 0.3000 \\ 0.3090 & 0.8236 & -0.4755 & 0.1000 \\ 0 & 0.5000 & 0.8660 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, the translational part and the rotational one are the following:

$${}^{e}O_t = \begin{bmatrix} 0.3000 \\ 0.1000 \\ 0 \end{bmatrix} \qquad {}^{e}_{t}R = \begin{bmatrix} 0.9511 & -0.2676 & 0.1545 \\ 0.3090 & 0.8236 & -0.4755 \\ 0 & 0.5000 & 0.8660 \end{bmatrix}$$

Transformation matrix between the base frame and tool frame.

$${}^{b}_{t}T = \begin{bmatrix} -0.0000 & 0.5000 & 0.8660 & -0.0000 \\ -0.4540 & 0.7716 & -0.4455 & -1.1369 \\ -0.8910 & -0.3932 & 0.2270 & 0.2327 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In this last case, the translation and rotation matrices are the following:

$${}^{b}O_t = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -0.0000 \\ -1.1369 \\ 0.2327 \end{bmatrix} \qquad {}^{b}_{t}R = \begin{bmatrix} -0.0000 & 0.5000 & 0.8660 \\ -0.4540 & 0.7716 & -0.3932 \\ -0.8910 & -0.4455 & 0.2270 \end{bmatrix}$$

In both ${}^b_tT$ and ${}^e_tT$ matrices it can be observed that the rotation matrix is not an identity matrix. This suggests that not only was the tool frame translated with respect to the end-effector or the base frame, but it was also rotated.

3 Exercise 2

In order to implement an inverse kinematic control loop, the computation of the desired tool's velocity is fundamental. In this assignment, this was performed by imposing an exponential convergence of the velocities through the use of the following formula:

$$\dot{\underline{x}}_{t/b} = \Lambda {}^b\underline{e}_{g/t} + {}^b\dot{\underline{x}}_{g/b}$$

This equation tells that the desired velocity of the tool to reach the goal position is given by the sum of the cartesian error premultiplied by Λ (the matrix of gains) and the actual goal velocity (with respect to the base frame). By expanding the terms, the following equation is obtained:

$$\begin{bmatrix} {}^b\underline{\omega}_{t/b} \\ {}^b\underline{v}_{t/b} \end{bmatrix} = \begin{bmatrix} \kappa_a I_{3\times3} & 0 \\ 0 & \kappa_l I_{3\times3} \end{bmatrix} \begin{bmatrix} {}^b\underline{\rho} \\ {}^b\underline{r} \end{bmatrix} + \begin{bmatrix} {}^b\underline{\omega}_{g/b} \\ {}^b\underline{v}_{g/b} \end{bmatrix}$$

with ${}^b\underline{\rho} = {}^b\underline{\rho}_{g/t}$, ${}^b\underline{r} = {}^b\underline{r}_{g/t}$ and κ_a, κ_l being, respectively, the angular and linear gains. In the case of this assignment the goal is not moving, therefore the last term of the equation is zero. The desired tool's velocity will then be used to compute the desired joint velocities $\dot{\underline{q}}$ to allow the tool to reach the goal position.

3.1 Q2.1

To obtain an optimal result, the computation of the cartesian error was one important yet crucial step. The first step consisted of finding the respective transformation matrix of the goal frame with respect to the base frame:

$${}^b_gT = \begin{bmatrix} {}^b_gR & {}^bO_g \\ 0 & 1 \end{bmatrix}$$

The goal's position and YPR angles were given, therefore the function *YPRToRot()* was used to convert the YPR angles to their respective rotation matrix. The following values were obtained:

$${}^bO_g = \begin{bmatrix} 0.2 \\ -0.8 \\ 0.3 \end{bmatrix} \qquad {}^b_gR = \begin{bmatrix} 0.0008 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0.0008 \end{bmatrix}$$

It was then possible to compute the cartesian error. The computation was performed in the *main.m* by calling the function *cartesianError(*${}^b_gT$*)* of the *cartesianControl* class. This function calls the *getToolTransformWrtBase()* function inside itself, which outputs the updated transformation matrix of the tool with respect to the base ${}^b_tT$. Then, the rotation matrix of the tool ${}^b_tR$ is extracted from ${}^b_tT$, the transformation matrix of the goal with respect to the tool is computed as ${}^t_gT = {}^b_tT^{-1}{}^b_gT$ and then used to calculate the rotation vector ${}^t\underline{\rho}$ by multiplying the angle axis parameters $\underline{h}$ and θ, which are obtained through the function *RotToAngleAxis(* ${}^t_gT$*)*. Finally, the linear and angular components of the cartesian error could be computed:

$$\underline{r} = {}^bO_g - {}^bO_t = \underline{r}_{g/b} - \underline{r}_{t/b} \qquad \underline{\rho} = {}^b_tR^t\underline{\rho}$$

and then:

$${}^b\underline{e}_{g/t} = \begin{bmatrix} {}^b\underline{\rho} \\ {}^b\underline{r} \end{bmatrix}$$

3.2 Q2.2

After these preliminary computations, the desired cartesian velocity could be computed. This step was performed in the *main.m* by calling the function *getCartesianReference(* ${}^b_gT$*)*. Inside it, the function *cartesianError(* ${}^b_gT$*)* was called to get the actual error, and the previously defined formula was then implemented by multiplying the gain matrix $\Lambda = \begin{bmatrix} \kappa_a I_{3\times3} & 0 \\ 0 & \kappa_l I_{3\times3} \end{bmatrix}$ with the error:

$$\dot{\underline{x}}_{t/b} = \Lambda {}^b\underline{e}_{g/t}$$

3.3 Q2.3

The final step before moving to the real end effector's velocity consisted of finding the desired joint velocities. Again, this was done in the *main* by calling the *getJacobianOfToolWrtBase* function, giving the Jacobian of the end effector as its output which allowed to link the joint space to the cartesian space. Since in this case the goal was to compute the joint velocities, it was necessary to use the Jacobian pseudoinverse. The final formula is the following:

$$\underline{\dot{q}} = J^{\#}\underline{\dot{x}}_{t/b}$$

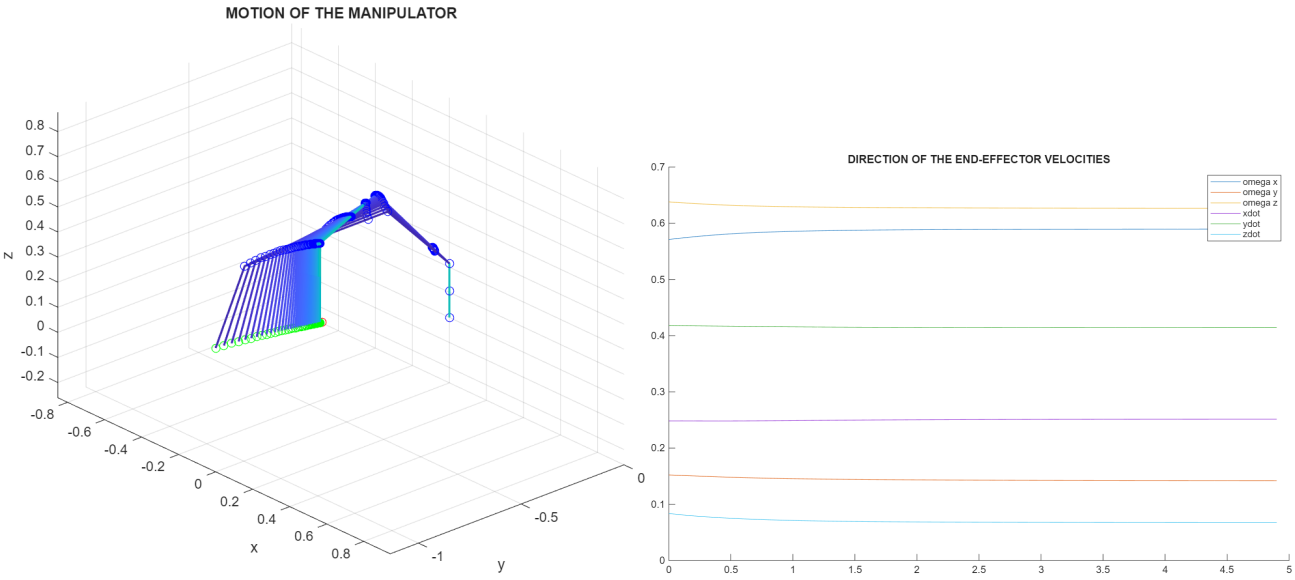
Note: since there is no risk of encountering singularities in this assignment, the MATLAB function *pinv* has been used used.

3.4 Q2.4

The *KinematicSimulation()* function performed two primary operations: it first calculated the subsequent joint configurations using the update law

$$\underline{q}_{new} = \underline{q} + \underline{\dot{q}} \times T_s$$

where T_s denotes the sample time, and subsequently saturated the output in case it was outside the lower or upper bound.



(a) Simulation (b) Direction velocity EE

Figure 2

The figure 2a shows the manipulator reaching the goal in the simulation and figure 2b shows the direction of the end effector velocities.

3.5 Q2.5

The velocity of the tool and the end effector were computed with the following formulas:

$$\underline{\dot{x}}_{t/b} = J_{b/t} * \underline{\dot{q}} \qquad \underline{\dot{x}}_{e/b} = J_{e/t} * \underline{\dot{q}}$$

where the Jacobians of tool and end effector were obtained respectively with *getJacobianOfToolWrtBase* and *getJacobianOfLinkWrtBase(gm.jointNumber)*.

Figures 3 and 4 illustrate that while the angular velocity remains identical, the linear velocities differ.

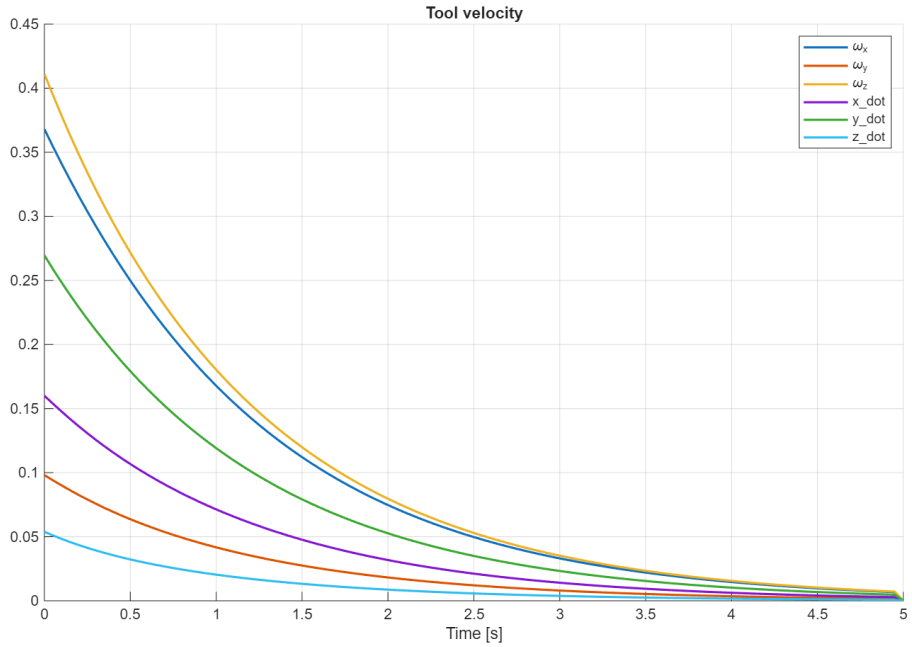


Figure 3: Velocity of the tool

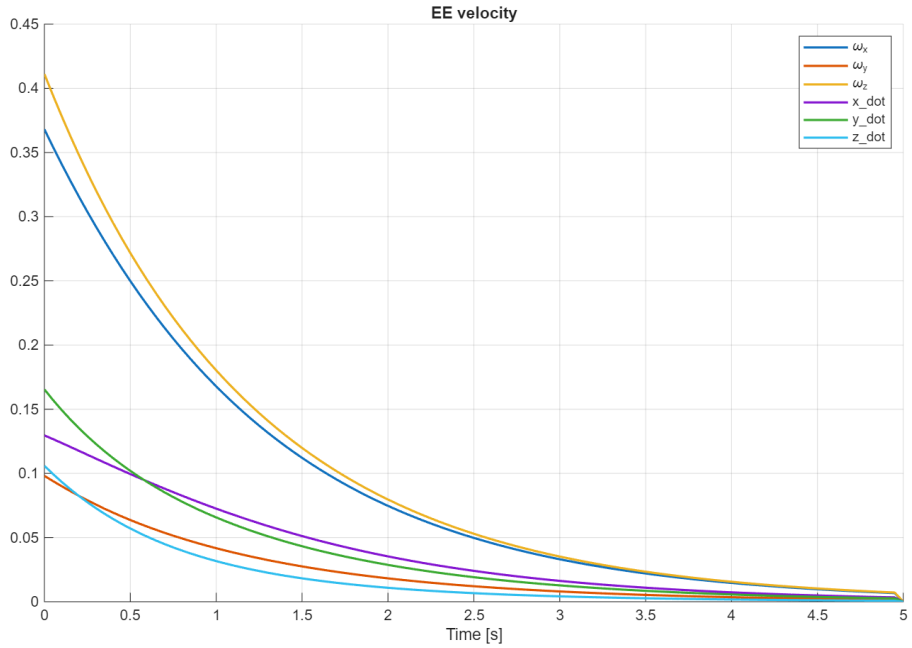


Figure 4: Velocity of the end effector