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

Professors:

Giovanni Indiveri
Enrico Simetti
Giorgio Cannata

Author:

Alberto Bono, 3962994
Andrea Chiappe, 4673275
Simone Lombardi, 6119159

Tutors:

Andrea Tiranti
Francesco Giovinazzo
George Kurshakov

January 1, 2024

Contents

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

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.

Exercise 1

The task involves computing Jacobian matrices for a robotic manipulator with seven revolute joints. The geometric model was created using the Buildtree function, establishing transformation matrices for each joint relative to the previous one. This process relied on the provided CAD model from the previous assignment.
To determine the Jacobian matrix for the manipulator at a given configuration, the GetJacobian function was implemented. The expected output is a matrix with dimension 6×7. The initial three rows represent angular motion, while the subsequent three rows correspond to linear motion along the x, y, and z axes, respectively. Each joint is represented by a column, and these are implemented using the formulas shown below.

For the linear velocity component ($J^v_{n/i}$):

$$J^v_{n/i} = \begin{cases} \underline{\mathbf{k}}_i \times \underline{\mathbf{r}}_{n/i} & \text{if joint type is revolute (R)} \\ \underline{\mathbf{k}}_i & \text{if joint type is prismatic (P)} \end{cases}$$

For the angular velocity component ($J^\omega_{n/i}$):

$$J^\omega_{n/i} = \begin{cases} \underline{\mathbf{k}}_i & \text{if joint type is revolute (R)} \\ \underline{\mathbf{0}} & \text{if joint type is prismatic (P)} \end{cases}$$

These equations represent $J^v_{n/i}$ and $J^\omega_{n/i}$, illustrating the linear and angular velocity components. Here, $\underline{\mathbf{k}}_i$ denotes the vector axis of rotation or translation for joint i in frame n, and $\underline{\mathbf{r}}_{n/i}$ represents the vector from O_i to O_n.
We provided a vector of matrices as an argument to the GetJacobian function, along with another vector specifying the types of joints, with dimensions equal to the number of joints. Specifically, our matrix vector consists of seven matrices, where each matrix represents the transformation of each joint relative to the base.
We applied a consistent approach, leveraging the first three rows of the third column in the transformation matrix, indicative of the z-axis, to compute the Jacobian matrix columns. For revolute joints, this information directly contributed to both the angular component and the linear component, computed through the cross product between the z-axis and the joint-to-end effector distance. However, in cases of prismatic joints, the angular component utilized a zero vector, while the linear component was based on the third column's z-axis information.
To determine the distances between the end effector and individual joints, we computed the differences between their respective distances from the base. These calculations relied on data extracted from the first three rows of the fourth column in the transformation matrix."

We compute the Jacobian matrix for all of these vectors that represents the different initial configurations of manipulator

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

The jacobians matrixs are:

$$\mathsf{J}_1 = \begin{bmatrix} 0 & -0.9636 & -0.0716 & -0.5061 & 0.8473 & -0.2905 & -0.2017 \\ 0 & 0.2675 & -0.2578 & -0.8231 & -0.4187 & 0.1496 & -0.9751 \\ 1.0000 & 0 & 0.9636 & -0.2578 & -0.3267 & -0.9451 & -0.0923 \\ 362.6284 & 73.5214 & 278.5715 & 40.7791 & -42.8244 & -143.1116 & 0 \\ 169.0722 & 264.8316 & 182.5777 & -160.3643 & 22.0533 & 25.0700 & 0 \\ 0 & 304.1870 & 69.5266 & 432.0064 & -139.3330 & 47.9538 & 0 \end{bmatrix}$$

$$\mathsf{J}_2 = \begin{bmatrix} 0 & -0.9738 & -0.0516 & 0.4367 & 0.8629 & 0.1484 & 0.2744 \\ 0 & -0.2272 & 0.2213 & -0.8720 & 0.4756 & 0.0849 & -0.9607 \\ 1.0000 & 0 & 0.9738 & 0.2213 & 0.1710 & -0.9853 & -0.0414 \\ -84.6839 & -112.0562 & 26.6565 & -68.7635 & 22.1181 & -145.3631 & 0 \\ 251.4504 & 480.3023 & 270.3338 & 60.8825 & 12.6505 & -40.4250 & 0 \\ 0 & -25.3391 & -60.0074 & 375.6585 & -146.8038 & -25.3846 & 0 \end{bmatrix}$$

$$\mathsf{J}_3 = \begin{bmatrix} 0 & -0.2955 & -0.1624 & -0.3880 & -0.8925 & -0.3880 & -0.0172 \\ 0 & 0.9553 & -0.0502 & 0.9215 & -0.3711 & 0.9215 & 0.0112 \\ 1.0000 & 0 & 0.9854 & -0.0170 & 0.2563 & -0.0170 & 0.9998 \\ 119.7951 & 678.7165 & 82.3672 & 237.5423 & -57.2052 & 140.9873 & 0 \\ -315.5555 & 209.9516 & -195.6046 & 107.5337 & 135.8502 & 59.4009 & 0 \\ 0 & 336.8636 & 3.6019 & 407.6740 & -2.5016 & 1.7672 & 0 \end{bmatrix}$$

$$\mathsf{J}_4 = \begin{bmatrix} 0 & 0 & -0.9950 & -0.0198 & 0.9216 & 0.1326 & 0.6340 \\ 0 & 1.0000 & 0 & 0.9801 & -0.0587 & 0.9766 & 0.0476 \\ 1.0000 & 0 & 0.0998 & -0.1977 & -0.3836 & 0.1692 & -0.7719 \\ -11.5371 & -94.2237 & -1.1518 & -277.0574 & 9.7293 & -116.5666 & 0 \\ 68.9866 & 0 & -86.8658 & -101.2317 & 71.6365 & 32.0793 & 0 \\ 0 & -68.9866 & -11.4794 & -474.1005 & 12.4132 & -93.7665 & 0 \end{bmatrix}$$

$$\mathsf{J}_5 = \begin{bmatrix} 0 & -0.8415 & -0.2919 & -0.8372 & 0.5468 & -0.4559 & -0.2954 \\ 0 & 0.5403 & -0.4546 & -0.3039 & -0.4589 & 0.5384 & -0.8427 \\ 1.0000 & 0 & 0.8415 & -0.4546 & -0.7003 & -0.7087 & -0.4501 \\ 396.0151 & 23.0076 & 313.8749 & -35.9272 & -58.6935 & -128.4543 & 0 \\ -8.7764 & 35.8322 & 5.0460 & -387.7714 & 69.3126 & 0.6435 & 0 \\ 0 & 337.9771 & 111.6171 & 325.3533 & -91.2477 & 83.1148 & 0 \end{bmatrix}$$

[Comment] For the last exercises include an image of the initial robot image of the final robot configuration

[Comment] For each exercise report the results obtained and provide an explanation of the result obtained (even though it might seem trivial). The matlab code is NOT an explanation of the algorithm.

Exercise 2

In the second and third exercises, we were tasked with creating a kinematic simulation for a seven degrees of freedom robot, considering two different scenarios:

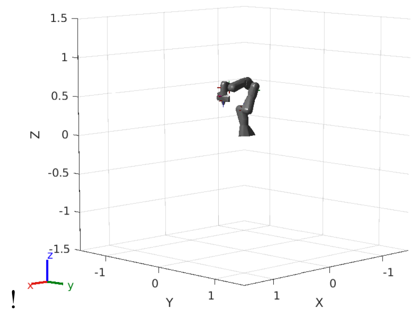


Figure 1: manipulator Franka 7DOF

- In the first one (exercise 2), the last link of the robot was considered as the end effector,
- in the second case (exercise 3), we had to add a rigidly attached tool to the last joint, with the correct specification of distance and orientation

As for the simulation, the base approach given to us was the following one, showed in Figure2

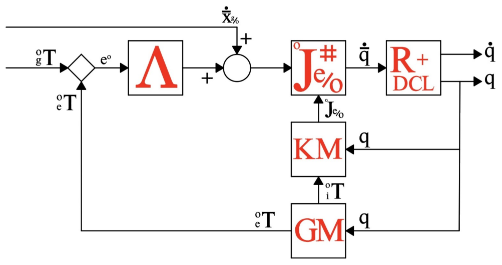


Figure 2:

We can separate the diagram into three separate parts:

1. The Cartesian space, where we compute the position and orientation error of the end effector (or tool) frame from the goal frame;
2. The 'interface,' where we carry out the computation of the desired velocities with the Jacobian pseudo-inverse
3. The 'joint space,' where we would normally carry out the calculation to obtain the acceleration, the torque, and finally the voltage to be applied to the motors to complete a task.

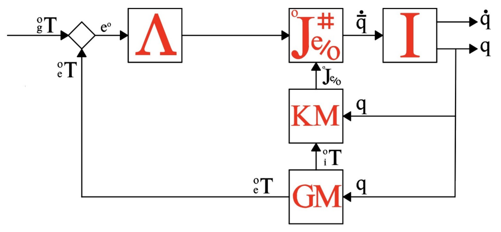
The first step was to simplify the block diagram in the following way: firstly, we considered the goal as stationary, so $\dot{\underline{x}}_g/o$ is zero at all times; secondly, the Robot + Dynamic control layer is considered to be equal to the identity. We end up with the diagram showed in Figure 3.

For the implementation, we started by initializing various elements:

- Tool frame definition:

We were given a list of instructions for the distance and orientation of the tool frame, the distance vector $\underline{eO_t} = [0, 0, 0.2104]^T$, the rotation matrix around the y-axis. The result transformation matrix is:

$$eT_t = \begin{bmatrix} \cos(\phi) & -\sin(\phi) & 0 & 0 \\ \sin(\phi) & \cos(\phi) & 0 & 0 \\ 0 & 0 & 0 & 0.2104 \end{bmatrix},$$



!

Figure 3: Enter Caption

with $\phi = -44.98°$, $\underline{{}^{e}O_t} = [0, 0, 21.04]^{\top}$(cm)

Lastly, we computed the transformation matrix from base to tool $bT_t = bT_e{\cdot}eT_t = bT_e{\cdot}[eR_t, \underline{eO_t}'; 0001]$, where the transformation matrix bT_e is computed with $getTransform(model.franka, [\underline{q\_init}', 0, 0], 'panda\_link7')$, with the vector $\underline{q\_init} = [0.0167305, -0.762614, -0.0207622, -2.34352, -0.0305686, 1.53975, 0.753872]^T$ configuration, so is equal to:

$$bT_e = \begin{bmatrix} 0.729812295678469 & -0.682403829182249 & -0.0412192551355275 & 0.314140305287370 \\ -0.683384449539698 & -0.729876840572911 & -0.0162939167369498 & -0.00471310552162817 \\ -0.0189659485354162 & 0.0400600987606159 & -0.999017257750556 & 0.693392691974274 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

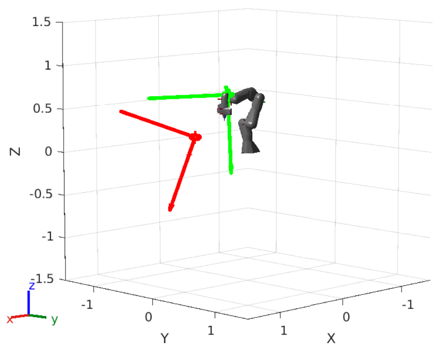
- Goal frame definition:

The specifications in both the end effector case and the tool case are the same: $\pi/6$ rad rotation around the y-axis of either end effector frame or the tool frame. In the two cases we have the distance from base to goal: $\underline{bO_g} = [0.55, -0.3, 0.2]^T$. The matrix bR_g changes in the two cases: $bR_e \cdot$ Ry$(\pi/6)$ for the e-e case or $bR_t \cdot$ Ry$(\pi/6)$ for the tool case, and in the end,

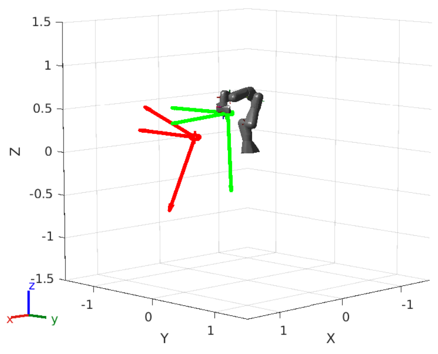
$$bT_g = \begin{bmatrix} bR_g(1,1) & bR_g(1,2) & bR_g(3,3) & eO_g(1) \text{ or } tO_g(1) \\ bR_g(2,1) & bR_g(2,2) & bR_g(3,3) & eO_g(2) \text{ or } tO_g(2) \\ bR_g(3,1) & bR_g(3,2) & bR_g(3,3) & eO_g(3) \text{ or } tO_g(3) \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

In the Fig.4, is printed the manipulator initial configuration and big frame. The red frame correspond to the goal frame, the green one correspond to the manipulators initial configuration. In the left image is center on the end effector, in the left image is center on the tool.

Figure 4: Initial configuration of manipulator



(Considering end-effector)



(Considering tool)

After the initialization, we enter the simulation part, where we have to perform all the previously mentioned calculations to obtain the desired velocities for the end effector (or tool) to reach the goal and complete the task. We will start by describing the calculations done in the case of the end effector, then discuss the differences in the case of the tool.

Starting from the robot configuration $\underline{q\_init}$, defined as the 'resting' position of the robot, we used the function 'getTransform' to retrieve the transformation matrix from the base to the end effector: bT_e. After this, we used 'geometricJacobian' to compute the geometric Jacobian J in the configuration $\underline{q}$ (these two functions were already present in the template). Now in the Cartesian space, we have to compute the error from the end effector to the goal, so we have to ensure that $\lim_{t\to\infty} e = 0$ and for each component of the Cartesian error, we have:

A distance or linear error, calculated as a vector from the center of the end effector reference frame to the center of the goal reference frame

$$\lim_{t\to\infty} \underline{\mathsf{r}} = 0,$$

An orientation or angular error between the same two frames.

$$\lim_{t\to\infty} eR_g = \mathsf{I} \Rightarrow \lim_{t\to\infty} \underline{\rho} = 0,$$

where $\underline{\rho} = \underline{v} \cdot \theta$

1. **Linear error**: since I have to modify the position of the end effector, we have to look at the derivative of each component of the Cartesian error. The distance is $\underline{\mathsf{r}} = \underline{O_g} - \underline{O_e}$ and the derivative of the distance has this form:

$$\mathcal{D}_b \underline{r} = \mathcal{D}_b(\underline{O_g} - \underline{O_v}) = \underline{{}^b v_g} - \underline{{}^b v_e}$$

since we want to nullify the distance between the goal and end effector, we need a derivative with a form that allows that to happen. So we can choose the desired derivative to be equal to: $\underline{r\_dot} = -\lambda \cdot \underline{r}$, $-\lambda \cdot \underline{r} = \underline{{}^b v_g} - \underline{{}^b v_e^*}$, where $\underline{{}^b v_e^*}$ is the desired velocity for end effector, so $\underline{{}^b v_e^*} = \underline{{}^b v_g} + \lambda \cdot \underline{r}$. In our case, $\underline{{}^b v_g}$, the velocity of goal, is supposed to be zero at any moment.

Note that this method is straightforward to implement and has very predictable results, but since the solution has to be parallel to $\underline{r}$, it uses three degrees of freedom of the robot, making them unavailable for other tasks (e.g., obstacle avoidance). To implement this behavior, we computed a variable called linear error, which is the difference in position of the goal and end effector:

$$\text{lin\_err} = bT_g(1:3,4) - bT_e(1:3,4).$$

2. **Angular error**:

we used a similar approach, considering $\underline{\rho}$ as the product of the angle-axis representation. So $\underline{\rho} = \underline{v} \cdot \theta$ We started by computing the derivative and controlling for a way to ensure that $\lim_{t\to\infty} \underline{eR_g} = I => \lim_{t\to\infty} \underline{\rho} = 0$. Where $\rho = \underline{v} \cdot \theta$,

$$\mathcal{D}_b \underline{\rho} = \mathcal{D}_e \underline{\rho} + \underline{w}_{e/b} \times \underline{\rho}$$
$$\mathcal{D}_e \underline{\rho} = \mathcal{D}_e(\underline{v}\theta) = \underline{v}\dot{\theta} + \theta \mathcal{D}_e \underline{v}$$

The implementation for this part goes as follows: Firstly, we had to compute the rotation matrix between the end effector and the goal, which we did by multiplying the inverse of the rotation matrix from the base to the end effector and the rotation matrix from the base to the goal.

$$eRg = bTe(1:3,1:3)' \cdot bTg(1:3,1:3)$$

After this, we used the function `ComputeInverseAngleAxis`, developed in the first assignment, to acquire the value of θ and $\underline{v}$ from eRg. Before assigning a value to `ang_err`, we projected the vector $\rho = \theta \cdot \underline{v}$ onto the base frame using the rotation matrix bRe, taken from the matrix bTe.

The rest of the simulation is simple as we assign the variables `angular_ref` and `linear_ref` to obtain the previously mentioned derivative of position and orientation:

$$\texttt{linear\_ref} = \texttt{linear\_gain} \cdot \texttt{linear\_err}$$

$$\underline{{}^b v_e^*} = \underline{{}^b v_g} + \lambda \cdot \underline{r}$$

$$\texttt{angular\_ref} = \texttt{angular\_gain} \cdot \texttt{ang\_err}$$

$$^{b}\underline{w}_e^* = {}^{b}\underline{w}_g + \lambda \cdot \rho$$

Unifying everything in a variable called `x_dot`, we obtained the vector of desired velocities for the end effector. We used the `pinv()` function to compute the pseudo-inverse of the geometric Jacobian for the particular configuration, and using the formula `pinv(bJe) * x_dot`, we obtained the joint velocities corresponding to our desired velocities, ant we call the variable `q_dot`.

The next step in the simulation is carried out using the function `KinematicSimulation`, which is an approximated integration with discrete time. Using the `ts` variable defined in the initialization section, we computed the new joint configuration in this way:

$$\underline{q\_new} = \underline{q\_old} + \underline{q\_dot}. \cdot \text{ts}$$

Since we are working with a real model, we also implemented a saturation control. After computing q_new, we check for each joint if the value of q_i is in the range of $q_{i,\text{min}} \le q_i \le q_{i,\text{max}}$ provided in the template. If one of these conditions occurs, we substitute the value of q_i with either $q_{i,\text{min}}$ or $q_{i,\text{max}}$, depending on the case. After this, the loop continues until the threshold is reached, or the simulation time runs out.

In the case of the tool frame, the transformation matrix used to compute all the Cartesian error is, of course, $bTt = bTe \cdot eTt$, instead of bTe. The other important difference is the use of a different geometric Jacobian to compute $\underline{q\_dot}$. The geometric Jacobian from the base to the tool has the form:

$$bJt = [S \cdot bJe, eJt]$$

Since the end effector and the tool are rigidly attached to one another, $eJt = [0\ 0\ 0\ 0\ 0\ 0]'$. The S matrix is called the rigid body Jacobian; premultiplying this matrix for the geometric Jacobian gives us a new Jacobian matrix that takes into account the displacement between the end effector and the tool frame.

The form of the rigid body Jacobian is obtained by calculating the velocity (angular and linear) for the end effector and tool. Since the angular velocity simply adds up from one frame to another, the important part is the linear velocity part:

$$\begin{aligned}
&\underline{v}_{t/b} = \underline{v}_{e/b} + \underline{w}_{e/b} \times \underline{r}_{t/e} + \underline{v}_{t/e} \\
&{}^{b}\underline{v}_{t/b} = {}^{b}J^{L}_{e/b}\dot{\underline{q}}_1 + [\underline{r}_{t/e}\times]^{T\,b}J^{A}_{e/b}\underline{q\_dot}_1 + {}^{b}_{e}R^{e}J^{L}_{t/e}\underline{q\_dot}_2 \\
&{}^{b}\underline{v}_{t/b} = [(\underline{r}_{t/e}\times)^T | I_{3\times3}]^{b}J_{e/b}\underline{q\_dot}_1 +^{b}_{e} R^{e}J^{L}_{t/e}\underline{q\_dot}_2
\end{aligned}$$

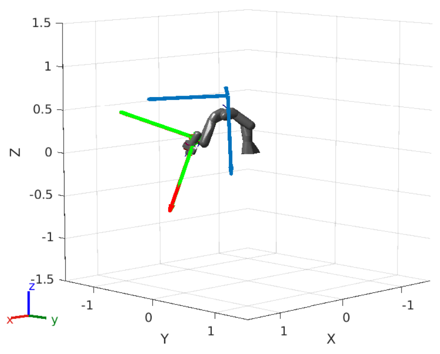
Since ${}^{t}J^{L}_{e}$ is a null vector, we needed to compute only the first part of the equation. The S matrix or rigid body Jacobian is computed in the following way:

$$eSt = \begin{bmatrix} I_{3\times3} & \mathbf{0}_{3\times3} \\ [{}^{e}\underline{r_t}\times] & I_{3\times3} \end{bmatrix}$$

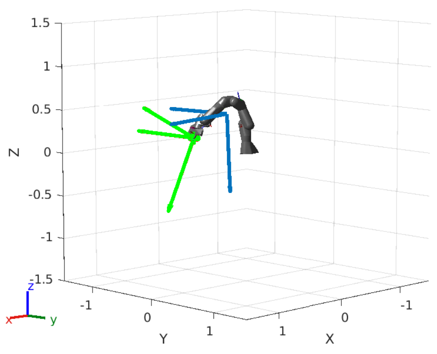
And after computing $bJt = eS_t \cdot bJe$, we used this Jacobian to compute the joint velocities instead of bJe. The progression of the simulation remains unchanged.
Now after all the passage we have reached the final configuration and reached the goal frame. In the Fig 5 there are the manipulator wher reach the goal frame. In blue is the initial frame of the respective e-e and tool. In green there is the actual frame orientation and position of e-e and tool. And in red there is the frame of the goal. In the two cases the red is not visible because is overlapped by the green frame.

In all of the two cases in example, e-e and tool, we stopped the simulation when the magnitude of x_dot is less than 0.001. In both cases we have a remain error. In specific, when we consider only end effector we have these errors: the mean of $lin\_err \approx 8 * 10^{-4}cm$ and the mean of $ang\_err \approx -0.0015rad$. In the case with the tool the $lin\_err \approx 4 * 10^{-4}cm$ and the $ang\_err \approx 0.0025rad$. The results obtained above were considered with a threshold on the magnitude of x_dot is 1×10^{-3}. We attempted to lower the threshold to a value of around 1×10^{-5} and increased the time step t to 60. The obtained results improved, achieving a precision of: $lin_err \approx 2.3 \times 10^{-5}$ and approximately similar for $ang_err \approx 2.09 \times 10^{-5}$. Therefore, we deduce that by increasing the threshold set on x_dot, we can enhance the precision of our manipulator, lowering the Cartesian error.

Figure 5: Final configuration of manipulator



(Considering end-effector)



(Considering tool)