UNIVERSITÀ DEGLI STUDI DI GENOVA

DIBRIS

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

MODELLING AND CONTROL OF MANIPULATORS

Second Assignment

Manipulator Geometry and Direct Kinematics

Professors:

Enrico Simetti
Giorgio Cannata

Author:

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

Tutors:

Andrea Tiranti
Francesco Giovinazzo
George Kurshakov

December 1, 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 second assignment of Modelling and Control of Manipulators focuses on manipulators geometry and direct kinematics.

- Download the .zip file called MOCOM-LAB2 from the Aulaweb page of this course.
- Implement the code to solve the exercises on MATLAB by filling the predefined files called "*main.m*", "*BuildTree.m*", "*GetDirectGeometry.m*", "*DirectGeometry.m*", "*GetTransformationWrtBase.m*", "*GetBasicVectorWrtBase.m*" and "*GetFrameWrtFrame.m*".
- Write a report motivating your answers, following the predefind format on this document.

1.1 Exercise 1

Given the following CAD model of an industrial 7 dof manipulator:

Q1.1 Define all the model matrices, by filling the structures in the *BuildTree()* function. Be careful to define the z-axis coinciding with the joint rotation axis, and such that the positive rotation is the same as showed in the CAD model you received. Draw on the CAD model the reference frames for each link and insert it into the report.

Q1.2 Implement a function called *DirectGeometry()* which can calculate how the matrix attached to a joint will rotate if the joint rotates. Then, develop a function called GetDirectGeometry() which returns all the model matrices given the following joint configurations:

- **q** $= [0, 0, 0, 0, 0, 0, 0]$.
- **q** $= [0, 0, 0, 0, 0, \pi/2, 0]$.
- **q** $= [0, \pi/2, 0, -\pi/2, 0, 0, 0]$.
- **q** $= [\pi/4, \pi/2, -\pi/8, -\pi/2, \pi/4, 2/3\pi, 0]$.

Comment the results obtained.

Q1.3 Calculate all the transformation matrices between any two links, between a link and the base and the corresponding distance vectors, filling respectively: *GetFrameWrtFrame()*, *GetTransformationWrtBase()*, *GetBasicVectorWrtBase()*

Q1.4 Given the following starting and ending configuration:

- **q**$_i = [0, 0, 0, 0, 0, 0, 0]$ and **q**$_f = [\pi/4, \pi/2, -\pi/8, -\pi/2, \pi/4, 2/3\pi, 0]$
- **q**$_i = [0, \pi/2, 0, -\pi/2, 0, 0, 0]$ and **q**$_f = [0, 0, 0, 0, 0, 0, 0]$
- **q**$_i = [1.3, 0.1, 0.1, 1, 0.2, 0.3, 1.3]$ and **q**$_f = [2, 2, 2, 2, 2, 2, 2]$

Plot the intermediate link positions in between the two configurations (you can use the plot3() or line() functions) and comment the results obtained.

Q1.5 Test your algorithm by changing one joint position at the time and plot the results obtained for at least 3 configurations.

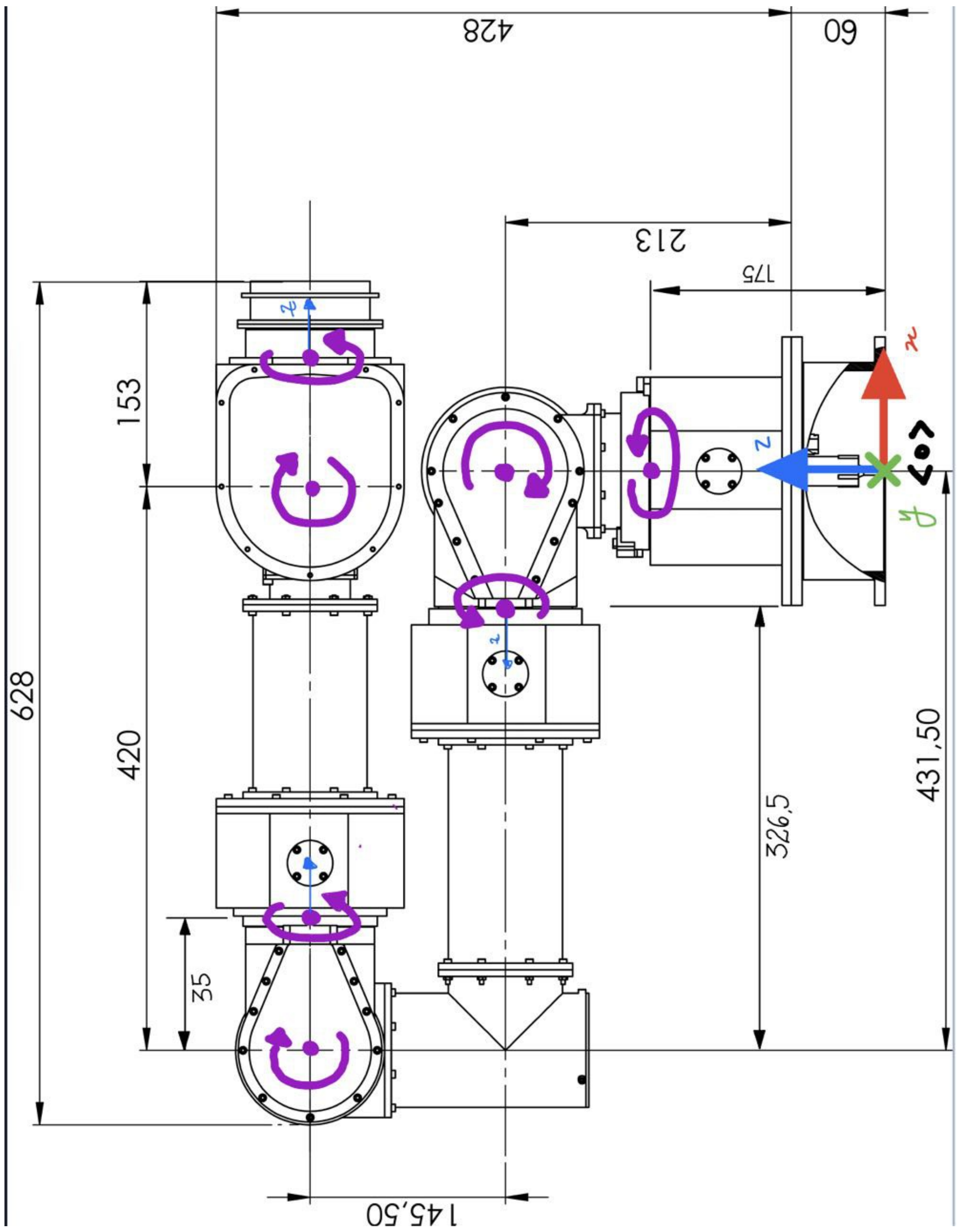
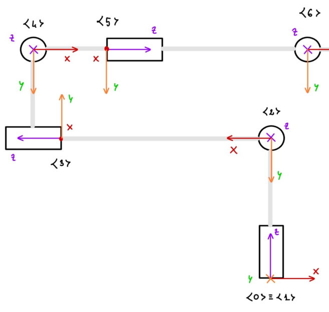


Figure 1: CAD model of the robot

2 Exercise 1

We started by defining how we wanted the frame to be positioned and oriented.

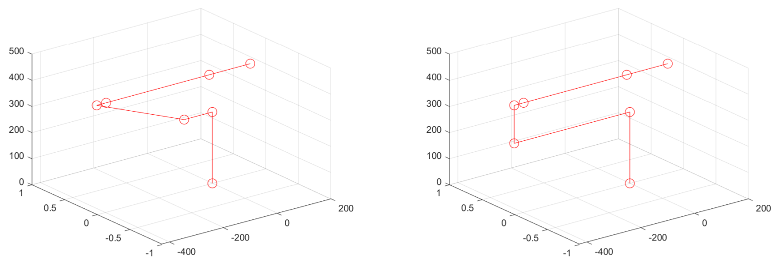
Figure 2: Geometry model with reference frame



//

For all the joints we defined the reference frame z-axis coincident with the joint rotation axis. For the third frame we opted to shift it along its own Z-axis, by doing so we were able to better match the geometry of the robot, as shown in the picture below.

Figure 3: Difference in positioning about reference frame link 3



Original position 3rd frame translated along z-axis

For the others frame we did not apply any changes from the model that was provided to us. After this we created by inspection the $BuildTree()$ function, that is used to create the geometric model of the robot. To be more specific the function return an array of matrix where each element is a 4 by 4 transformation matrix. Starting from the first, the transformation from the base frame to the first, from the first to the second and so on.

$$0T1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad 1T2 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 273 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$2T3 = \begin{bmatrix} 0 & 0 & 1.0000 & 431.5000 \\ 1.0000 & 0 & 0 & 0 \\ 0 & 1.0000 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} \qquad 3T4 = \begin{bmatrix} 0 & -1.0000 & 0 & 145.5000 \\ 0 & 0 & 1.0000 & 0 \\ -1.0000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

$$4T5 = \begin{bmatrix} 0 & 0 & 1 & 35 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad 5T6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 385 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$6T7 = \begin{bmatrix} 0 & 0 & 1 & 153 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can see the geometric model as the standard configuration of the robot, in this position all the joint variable are set to zero. After the geometric model we developed the function used to compute the transformation matrices for a generic configuration. The function we were asked to implement were two: $DirectGeometry()$ and $GetDirectGeometry()$. The core of the computation is carried out in the first one that with a transformation matrix from the geometric model, a variable that explain the type of joint we have at the specific link and the value of the joint variable is able to compute a new transformation matrix that takes into consideration the value of the joint variable.

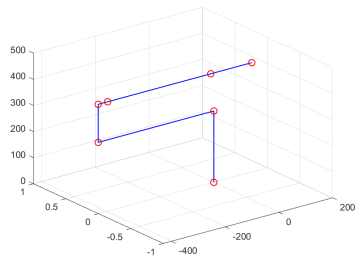
$$i-1Ri = \begin{cases} Ri & \text{if } \Gamma = \text{prismatic} \\ Ri * Rz(qi) & \text{if } \Gamma = \text{rotational} \end{cases} i-1ri = \begin{cases} ri + \hat{z} * qi & \text{if } \Gamma = \text{prismatic} \\ ri & \text{if } \Gamma = \text{rotational} \end{cases} \tag{1}$$

were Rz is the generic rotation around the z axis, and z is the translation axis of the joint.

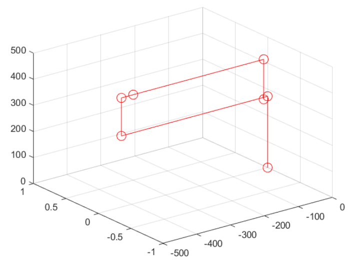
$$R_z = \begin{bmatrix} \cos(q_i) & -\sin(q_i) & 0 \\ \sin(q_i) & \cos(q_i) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

After we reconstruct the new Transformation matrix considering the joint rotation of qi and return the result to the user. The second function is used to cycle through the complete geometric model matrix to compute all the matrix needed for a specific configuration. Now we test the function with four different **q** vector.

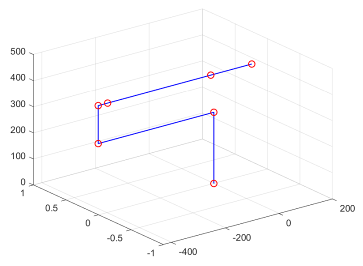
Figure 4: Difference geometry according with different **q**



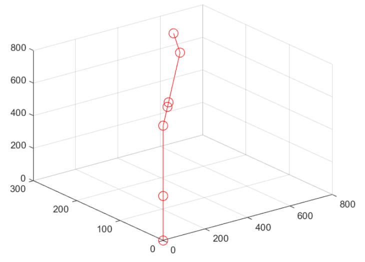
q = [0, 0, 0, 0, 0, 0, 0]



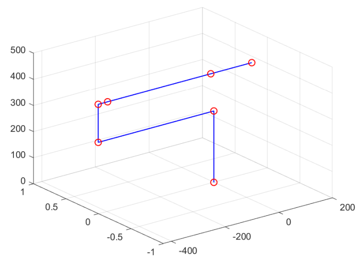
q $= [0, 0, 0, 0, 0, \pi/2, 0]$



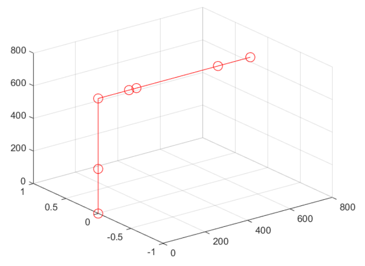
q = [0, 0, 0, 0, 0, 0, 0]



q $= [\pi/4, \pi/2, -\pi/8, -\pi/2, \pi/4, 2/(3*\pi), 0]$



q = [0, 0, 0, 0, 0, 0, 0]



q $= [0, \pi/2, 0, -\pi/2, 0, 0, 0]$

For the second part of this assignment we had to plot a series of motion, we were given a set of couples of joint variable vector each rapresenting an initial or a final position.

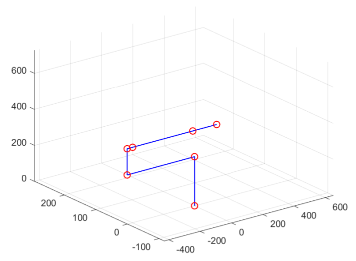
To achieve this goal we developed a series of function: $GetTransformationWrtBase()$, $GetBasicVectorWrtBase()$, $GetFrameWrtFrame()$. The first function is used to calculate the transformation matrix from the base frame to every other frame, we need this calculation since to be able to plot coherent images we need to project every body and its motion in one single frame(the base frame is chosen for ease of representation). For calculate the transformation matrix from the base to one general frame we multiply all the transformation matrix starting from the base. So consider N = 7, total number of link and $n < N$ a general frame of the chain

$$0Tn = 0T1 * 1T2 * ... * n - 1Tn$$

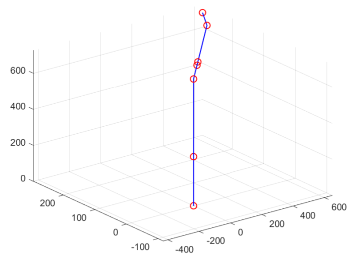
The second function simply uses the array of matrix obtained by $GetTransformationWrtBase()$, to extract the vector that represent the link of the robot now projected in the base frame. The use of the function to calculate all the configuration from an initial position to a final one goes as follows: Firstly we take the 2 joint variable vector and we divide their difference from the number of "steps" or "frame" we want to have (in a real environment this number is very important since the computational load needed scale up very quickly in function of this number, higher number grant better control but at a cost) in our motion, we will call this quantity Δq. Than for each steps:

1. Compute the direct geometry – $GetDirectGeometry()$
2. Compute the projection on the base frame - $GetTransformationWrtBase()$
3. Extract the projected link from the transf. matrix - $GetBasicVectorWrtBase()$
4. Plot all the link in a certain configuration – $plot3()$
5. Add the increment to the initial configuraiton q = q + Δq

Figure 5: Moving from **qi** $= [0, 0, 0, 0, 0, 0, 0]$ to **q** $= [\pi/4, \pi/2, -\pi/8, -\pi/2, \pi/4, 2/3\pi, 0]$

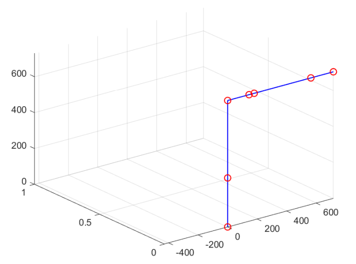


Initial Configuration

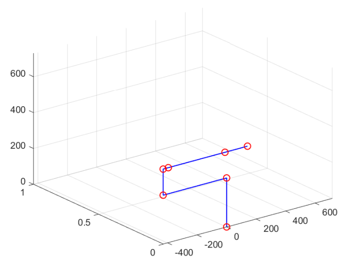


Final Configuration

Figure 6: Moving from **q** $= [0, \pi/2, 0, -\pi/2, 0, 0, 0]$ to **qf** $= [0, 0, 0, 0, 0, 0, 0]$

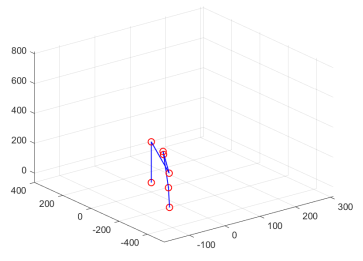


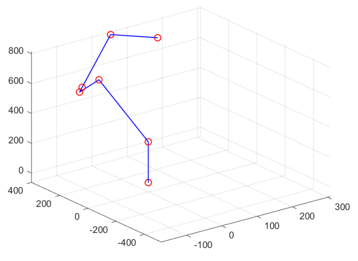
Initial Configuration



Final Configuration

Figure 7: Moving from $\mathbf{qi} = [1.3, 0.1, 0.1, 1, 0.2, 0.3, 1.3] to \mathbf{qf} = [2, 2, 2, 2, 2, 2, 2]$


Initial Configuration

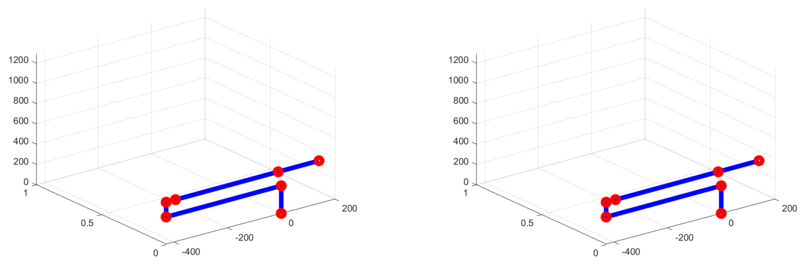

Final Configuration

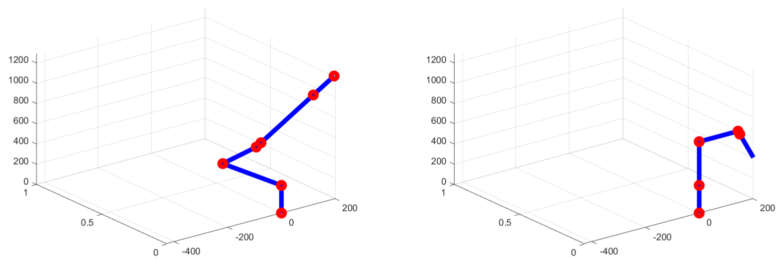
For the last section of the assignment we executed the animation of part 1.4 but moving one joint at a time(we decided to use the same couples of position of section 1.4 to more easily check the correctness of the execution). We implemented an algorithm that execute the same code of section 1.4 but using as initial and final position two matrix that progressively added more joint to the movement, heres how the iteration would look like for the first animation:

- $\mathbf{qi}_1 = [0, 0, 0, 0, 0, 0, 0]$ and $\mathbf{qf}_1 = [\pi/4, 0, 0, 0, 0, 0, 0]$
- $\mathbf{qi}_2 = [\pi/4, 0, 0, 0, 0, 0, 0]$ and $\mathbf{qf}_2 = [\pi/4, \pi/2, 0, 0, 0, 0, 0]$
- $\mathbf{qi}_3 = [\pi/4, \pi/2, 0, 0, 0, 0, 0]$ and $\mathbf{qf}_3 = [\pi/4, \pi/2, -\pi/8, 0, 0, 0, 0]$
- $\mathbf{qi}_4 = [...]$ and $\mathbf{qf}_4 = [...]$

and so on. The result that we obtained are equal to the one in section 1.4, but it is important to notice the different space usage that we encountered in the two version. In the first case on average the space used during the movement is smaller than in the second case, this difference would be something to keep in mind for obstacle avoidance and general safety around the robot. Following an example that shows this difference:

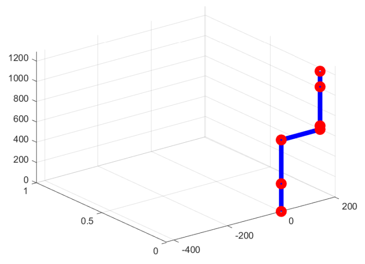
Figure 8: Moving from $\mathbf{qi} = [0, 0, 0, 0, 0, 0, 0]$ to $\mathbf{qf} = [0, \pi/2, 0, -\pi, 0, 0, 0]$

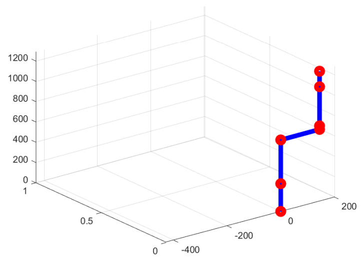

Initial Configuration for both the section



As we observed in the first instance any step of motion is done inside the boundaries shown in the images, instead in the second images as we see some link and joint are not visible because they are outside the boundaries.

Lastly we implemented the function GetFrameWrtFrame(), this function has the same function of GetFrameWrtBase() the difference is that in the second one the the base frame is fixed as the projection frame.





but as expected the final configuration is the same.

It was possible to use this function to reduce the number of calculation needed to resolve section 1.5, at each iteration we would have simulated a robot that started from the last moved joint. By doing so after moving one joint it would not have been necessary to compute its movement again.

3 Appendix

[Comment] Add here additional material (if needed)

3.1 Appendix A

3.2 Appendix B