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

Author:

Galvagni Gianluca

Student ID:

s5521188

Professors:

Giovanni Indiveri
Enrico Simetti
Giorgio Cannata

Tutors:

Andrea Tiranti
Francesco Giovinazzo

December 23, 2022

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.

The second assignment is **mandatory** and consists of one exercise. You are asked to:

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

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()*, *GetBasicVectgorWrtBase()*

Q1.4 Given the following starting and ending configuration:

- $\mathbf{q}_i = [1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3]$ and $\mathbf{q}_f = [2, 2, 2, 2, 2, 2, 2]$
- $\mathbf{q}_i = [1.3, 0, 1.3, 1.7, 1.3, 0.8, 1.3]$ and $\mathbf{q}_f = [2, 0, 1.3, 1.7, 1.3, 0.8, 1.3]$
- $\mathbf{q}_i = [1.3, 0.1, 0.1, 1, 0.2, 0.3, 1.3]$ and $\mathbf{q}_f = [2, 2, 2, 2, 2, 2, 2]$

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.

2 Exercise 1

In this second assignment, I implemented a software to do that it calculates the matrices to manipulate an anthropomorphic robot with six axis. And then calculate the geometry and direct kinematics to simulated a movement from a start position and a end position.
The goal of the kinematic analysis of manipulator is to evaluate how, in a certain configuration q, a given set of velocities in the joint space $\dot{q}$ will effect the total velocity $\dot{x_{e/o}}$ of the end-effector. In the first programs' rows, I defined some useful values and then I started with the exercise.

2.1 Q1.1

In the first point, I filled up the function *Buildtree()* with all values to build the tree of frames for the chosen manipulator. In this function there are seven "big arrays" like this one:

$$ {}^a_bT = \begin{bmatrix} {}^a_bR & {}^ar_b \\ 000 & 1 \end{bmatrix} $$

It is the pose of a frame with respect to another can be represented in a more compact way by proving a 4x4 matrix, called the transformation matrix from frame $< a >$ to frame $< b >$ (${}^a_bT$). When we working with transformation matrix we need to switch to the *homogeneous coordinates of a projected point*, which are obtained simply by adding a fourth one-component to the point:

$$ {}^aP = \begin{vmatrix} x & y & z & 1 \end{vmatrix}^T \in \Re^4 $$

Once that we have computed the transformation matrix ${}^a_bT$, the coordinate change from $< a >$ to $< b >$ is given by:

$$ {}^bP = {}^a_b T \, {}^aP $$

Before to do that, I drew on the CAD model the reference frames for each link to represent in this function (Figure 2) with the following relation between links:

- $< 0 > \rightarrow < 1 >$: no rotation;
- $< 1 > \rightarrow < 2 >$: -90° rotation on the x-axis;
- $< 2 > \rightarrow < 3 >$: -90° rotation on the y-axis;
- $< 3 > \rightarrow < 4 >$: +90° rotation on the y-axis;
- $< 4 > \rightarrow < 5 >$: -90° rotation on the y-axis;
- $< 5 > \rightarrow < 6 >$: -90° rotation on the y-axis;
- $< 6 > \rightarrow < 7 >$: +90° rotation on the y-axis;

They are all elementary rotations, I can define all the rotation matrices using the $ElementaryRotation.m$ function developed in the first assignment or exploiting the rule that each column of the rotation matrix describes the orientation of each axis of the second frame on the first.
To fill the last column of the transformation matrices, I need to know also the position of each frame with respect to the previous: using the dimensions provided to us in the CAD model, I can develop these vectors, so I can finally fill the $BuildTree()$ function and building all the transformation matrices:

2.2 Q1.2

In this point, I used two functions *DirectGeometry()* and *GetDirectGeometry()* to calculate, at the end the ${}^iTj_q$ vector of matrices containing the transformation matrices from link $< i >$ to link $< j >$ for the input q.
All the rotation are done to respect the *z axis* and it makes the pure rotation matrix along it self, and we need only this matrix to calculate the future rotations (up to rotational joints).

$$ Rz = \begin{bmatrix} \cos(q_i) & -\sin(q_i) & 0 \\ \sin(q_i) & \cos(q_i) & 0 \\ 0 & 0 & 0 \end{bmatrix} $$

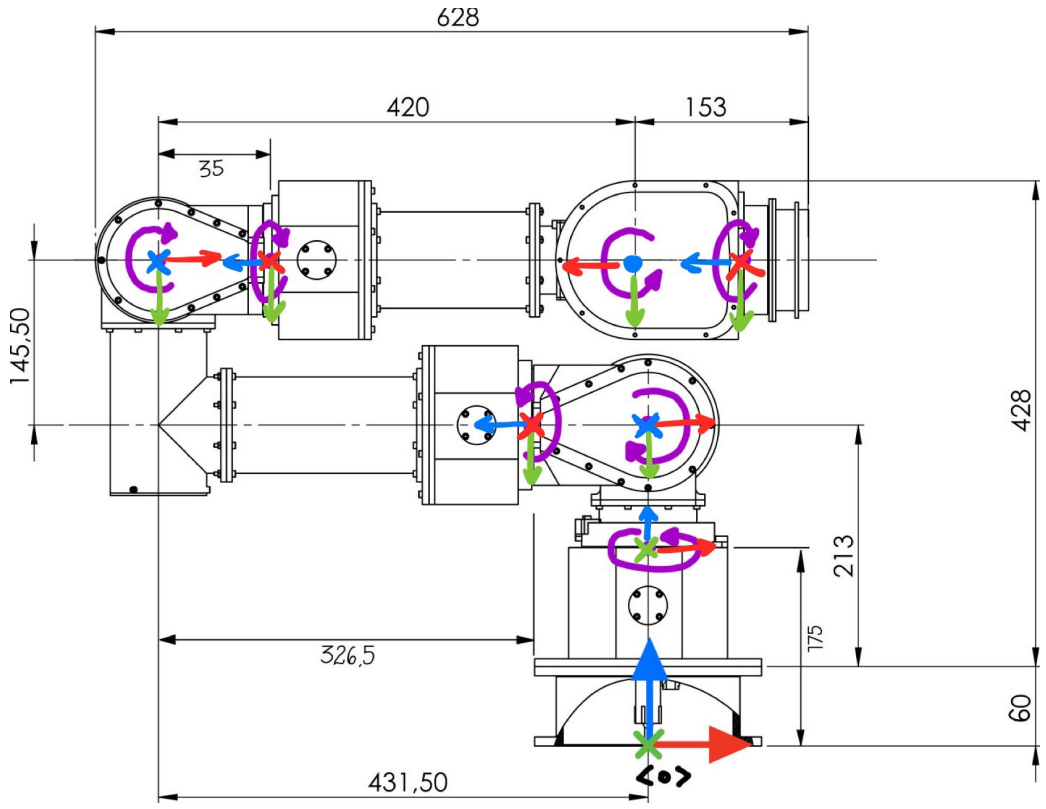

Figure 1: CAD model which represents all frames with the corresponding coordinates to respect into the purple line.

```
val(:,:,1) =

     1     0     0     0
     0     1     0     0
     0     0     1   175
     0     0     0     1


val(:,:,2) =

     1     0     0     0
     0     0     1     0
     0    -1     0    98
     0     0     0     1


val(:,:,3) =

     0     0    -1  -105
     0     1     0     0
     1     0     0     0
     0     0     0     1


val(:,:,4) =

         0         0    1.0000         0
         0    1.0000         0  -145.5000
   -1.0000         0         0   326.5000
         0         0         0    1.0000


val(:,:,5) =

     0     0    -1    35
     0     1     0     0
     1     0     0     0
     0     0     0     1


val(:,:,6) =

     0     0    -1     0
     0     1     0     0
     1     0     0  -385
     0     0     0     1


val(:,:,7) =

     0     0     1  -153
     0     1     0     0
    -1     0     0     0
     0     0     0     1
```

Figure 2: Matrix configuration of our robot

```
val(:,:,1) =

    0.2675   -0.9636         0         0
    0.9636    0.2675         0         0
         0         0    1.0000  175.0000
         0         0         0    1.0000

val(:,:,2) =

    0.2675   -0.9636         0         0
         0         0    1.0000         0
   -0.9636   -0.2675         0   98.0000
         0         0         0    1.0000

val(:,:,3) =

         0         0   -1.0000 -105.0000
    0.9636    0.2675         0         0
    0.2675   -0.9636         0         0
         0         0         0    1.0000

val(:,:,4) =

         0         0    1.0000         0
    0.9636    0.2675         0 -145.5000
   -0.2675    0.9636         0  326.5000
         0         0         0    1.0000

val(:,:,5) =

         0         0   -1.0000   35.0000
    0.9636    0.2675         0         0
    0.2675   -0.9636         0         0
         0         0         0    1.0000

val(:,:,6) =

         0         0   -1.0000         0
    0.9636    0.2675         0         0
    0.2675   -0.9636         0 -385.0000
         0         0         0    1.0000

val(:,:,7) =

         0         0    1.0000 -153.0000
    0.9636    0.2675         0         0
   -0.2675    0.9636         0         0
         0         0         0    1.0000
```

Figure 3: Model matrices given q configuration equal to [1.3,1.3,1.3,1.3,1.3,1.3,1.3]

All the translation are done to respect the *z axis* and it makes the pure translation matrix along it self, and we calculated it but we are not going to use it in this exercise (up to prismatic joints).

$$Tz = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

How you can see it is the identical matrix (I_{3x3}).

The *DirectGeometry()* calculates the transformation matrix from link i to link $i+1$ for the input q_i taking into account the actual rotation/translation of the joint. And the function *GetDirectGeometry()* is useful to make for each links the vector of matrices containing the transformation matrices for all links in relation to the input q. The size of this matrix is equal to [4,4, link's number].

2.3 Q1.3

For the third task, I only need to do some mathematical processing of the data we already have.
To calculate the final transformation matrix between two or more serial links, I have to multiply the single transformation matrix of each joint:

$$^{0}_{n}H = ^{0}_{1}H \, ^{1}_{2}H \, ... \, ^{n-1}_{n}H \tag{1}$$

So, in the $GetFrameWrtFrame()$ function I have two joint numbers and I want to know the transformation matrix in between these links, so we multiply all the transformation matrices (given by the $DirectGeometry()$ function) in this range of joints using two for loops.

In the $GetTransformationWrtBase()$ function I do the same thing but in this case, the first link is always the base; so it is exactly the same function $GetFrameWrtFrame()$ but the first link number is always 1.
Finally, for filling the $GetBasicVectgorWrtBase()$ function we need the last column of the matrix developed in these steps, so I call the $GetFrameWrtFrame()$ function passing the link number desired and I extract the first three rows of the last column.

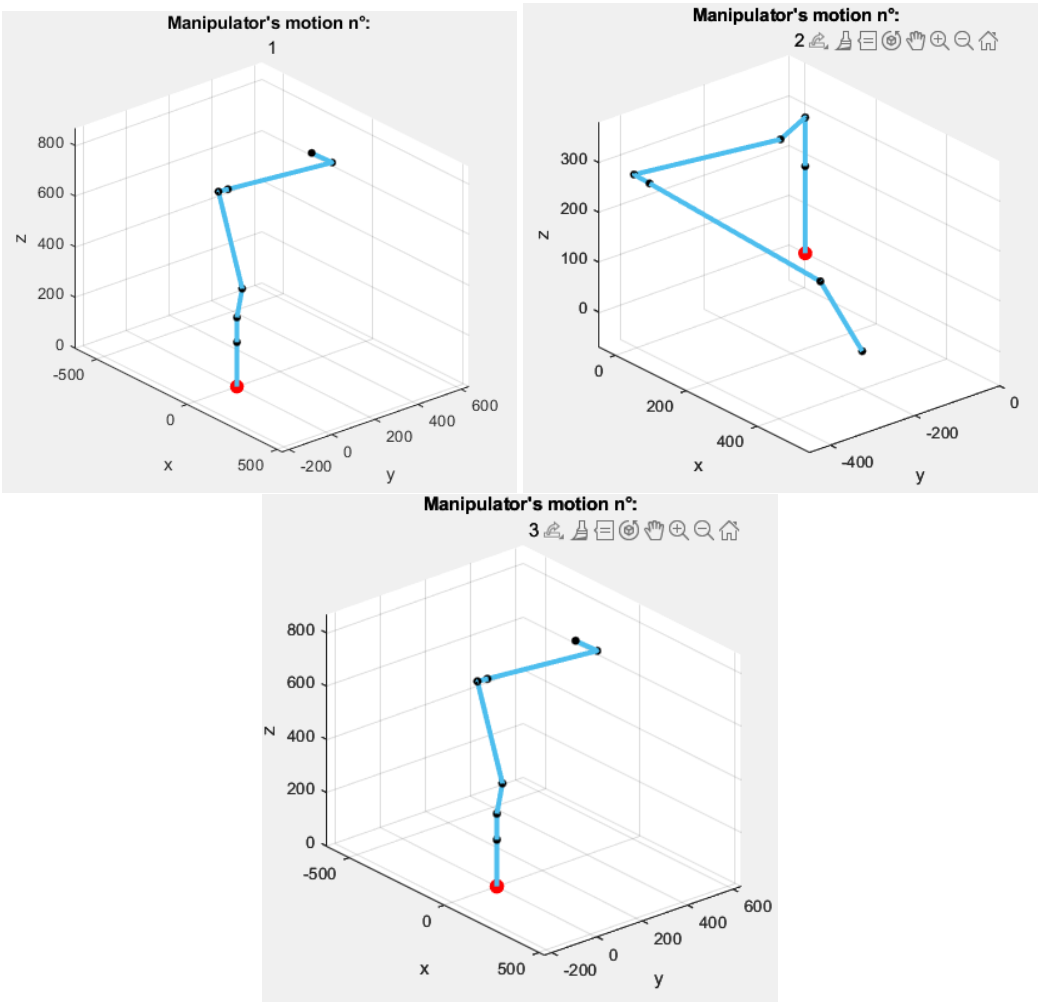

Figure 4: The red dot is the frame $< 0 >$, the blue lines are links and the dark dots are the joints

2.4 Q1.4

In this task, I created q_i and q_f like that two vectors with three different configuration to move our robot and i chose the number of step to go from q_i to q_f. Then I made the $PlotConfigurationDirectKinematic()$, it is a function to plot in 3D the robots' links and the robots' joints during the movement from q_i to q_f. It use the function we have seen before at point 2 and 3.
I can show you only the end configuration of the simulation but you can run the $main.m$ program and see all movements.
Manipulator's motion number :

1. q_i = [1.3,1.3,1.3,1.3,1.3,1.3,1.3] and q_f = [2, 2, 2, 2, 2, 2, 2];
2. q_i = [1.3, 0, 1.3, 1.7, 1.3, 0.8, 1.3] and q_f = [2, 0, 1.3, 1.7, 1.3, 0.8, 1.3];
3. 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].

2.5 Q1.5

At the end, I create a while loops to allow the user to choose the joint number to change its configuration and to decide the future configuration of the chosen joint. The robot starts from a initial configuration (q_i = [2, 2, 2, 2, 2, 2, 2], my random selection) and the user picks the joint (in this case from 1 to 7) and selects its future configuration. For instance, if I choose "joint 3" with configuration "1.3", the robot moves from q_i to q_f = [2, 2, 1.3, 2, 2, 2, 2] and then I can choose to change the fifth joint position with the configuration "2.6" then, the robot starts from q_i = [2, 2, 1.3, 2, 2, 2, 2] (the old q_f) from the new q_f = [2, 2, 1.3, 2, 2.6, 2, 2] and go on like this.
$\rightarrow$ launch the main.m to see that.

3 Appendix

[Comment] Add here additional material (if needed)

3.1 Appendix A

3.2 Appendix B