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:

Triki Karim

Student ID:

s5528602

Professors:

Giovanni Indiveri
Enrico Simetti
Giorgio Cannata

Tutors:

Andrea Tiranti
Francesco Giovinazzo

January 19, 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 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 is **mandatory** and 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 the predefined files. In particular, you will find two different main files: *"ex1.m"* for the first exercise and *"ex2.m"* for the second and third exercises.
- Write a report motivating your answers, following the predefind format on this document.

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:

- $\mathbf{q}_1 = [1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3]$
- $\mathbf{q}_2 = [1.3, 0.4, 0.1, 0, 0.5, 1.1, 0]$
- $\mathbf{q}_3 = [1.3, 0.1, 0.1, 1, 0.2, 0.3, 1.3]$
- $\mathbf{q}_4 = [2, 2, 2, 2, 2, 2, 2]$

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 ${}^b_{ge}T$.
${}^b_{ge}T$ must be defined knowing that:

- The goal position with respect to the base frame is ${}^bO_g = [0.6, 0.4, 0.4]^T$
- The goal frame is rotated of $\theta = -\pi/4$ around the z-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 transformation matrix:

$$eTt = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Q3.1 Compute the cartesian error between the robot tool frame ${}^b_tT$ and the goal frame ${}^b_{gt}T$.
${}^b_{gt}T$ must be defined knowing that:

- The goal position with respect to the base frame is ${}^bO_g = [0.6, 0.4, 0.4]^T$
- The goal frame is rotated of $\theta = -\pi/4$ around the z-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.

Q3.6 Test the algorithm for a new tool goal, knowing that the transformation matrix of the goal with respect to the robot base is:

$$bTg = \begin{bmatrix} 0.7071 & 0 & -0.7071 & 0.6 \\ -0.7071 & 0 & -0.7071 & 0.5 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2 Exercise 1

In this exercise we want to compute *The Jacobian Martices* for the manipulator for the joint configurations.

But what do we mean *The Jacobian Martices*? and why we called *Jacobian*?

From my previous undergraduate studies, I know the origin of the word *"Jacobian"* The name *Jacobian* attributed to the German mathematician **Carl Gustav Jacobi**, where the first research was on the ***Jacobian variety*** but in this exersice we are talking about **Jacobian Matrices** which is completely different from the **Jacobian variety**

Simply Jacobian is Matrix in robotics which provides the relation between joint velocities (***q***) and end effector velocities (***X***) of a robot manipulator. If the joints of the robot move with certain velocities then we might want to know with what velocity the end effector would move. Here is where Jacobian comes to our help. The relation between joint velocities and end effector velocities is given as below,

$$X = Jq$$

Where :

-***q*** is the column matrix representing the joint velocities. Size of the this matrix is n x 1. 'n' is the number of joints of the robot.

-***X*** is the column matrix representing the end-effector velocities. Size of this matrix is m x 1. 'm' is 3 for a planar robot and 6 for a spatial robot.

-***J*** is the Jacobian matrix which is a function of the current pose . Size of jacobian matrix is m x n.

$$\begin{bmatrix} x \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix}_{6\times1} = \begin{bmatrix} J_{11} & J_{12} & \cdot & \cdot & \cdot & J_{1n} \\ J_{21} & J_{22} & \cdot & \cdot & \cdot & J_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdot & \cdots & \cdot & \cdot & \cdot & \cdot \\ J_{61} & \cdots & \cdot & \cdot & \cdot & J_{6n} \end{bmatrix}_{6\times n} * \begin{bmatrix} q_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \cdot \\ \vdots \\ \dot{q}_n \end{bmatrix}_{n\times1}$$

And this is the definition of the Jacobian matrix

How it's works?

The code is working like the previous one of the 2nd Assignment except that we add the Jacobian matrices

The code is available in Appendix

3 Exercise 2

3.1 Q2.1

The second exercise we have model of a Panda robot by Franka Emika

Picture of Panda robot by Franka Emika

we want to compute the cartesian error between the robot end effector frame ${}^{b}_{e}T$ and the goal frame ${}^{b}_{ge}T$.

In the beginning we need to make the premiere configuration in the MATLAB like define the end effector, putting the initial Joints configuration according to panda model, in the other part we introduce the goal position with its frame in order to calculate the cartesian error

THE OPERATIONS

we are going to compute the cartestian error until we reach the goal so we use the transformation matrices and the jacobian matrix for rigid body.

All of that are necessary to find the error.

RIGID BODY TRANSFORM

Consider a geometric object modeled as a set of points in Euclidean space. By definition, a rigid body transform is a mapping from this set to another subset of the Euclidean space,such that the Euclidean distances between points are preserved.

After we know what does mean a Rigid Body Transform, we can calculate the Error and get the Angular error and Linear Error

THE CODE IS AVAILABLE IN APPENDIX

3.2 Q2.2

In this exercise we want to calculate the desired angular and linear reference velocities

COMPUTE THE REFERENCE VELOCITIES

according to the rule

ANGULAR VELOCITY = ANGULAR ERROR X ANGULAR GAIN

LINEAR VELOCITY = LINEAR ERROR X LINEAR GAIN

velocity = angular velocity x linear velocity

THE CODE IS AVAILABLE IN APPENDIX

3.3 Q2.3

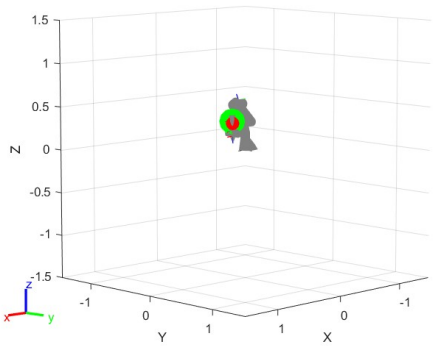
COMPUTE DESIRED JOINT VELOCITIES

We have multiply the velocity and the Inverse matrix of the transformation Or the Inverse Jacobian Matrix

THE CODE IS AVAILABLE IN APPENDIX

3.4 Q2.4

In this Question We want to make the Simulation so we use the function plot and this kind of simulation we do it in order to see how the robot is going to move in other part not to lose the robot



THE CODE IS AVAILABLE IN APPENDIX

4 Exercise 3

in this exersice We have use the same code of Exersice 2

4.1 Q3.1

the goal position and the goal frame are different so we need to add them without this everything looks the same as Exersice 2

4.2 Q3.2

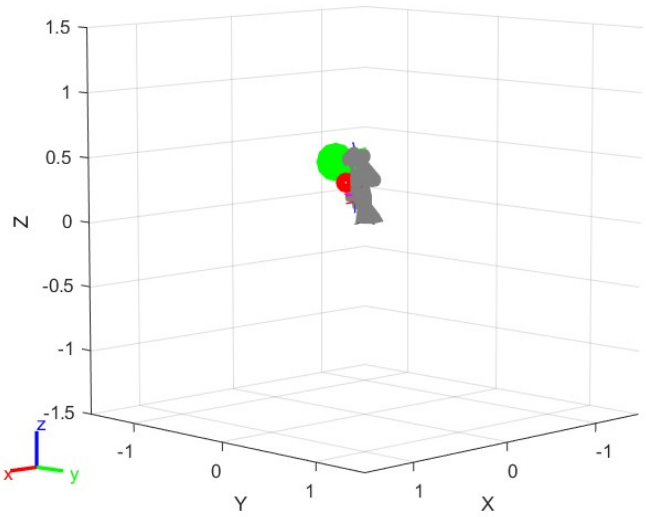
here we use the same code of Exersice 2 , the gain is 0.2

4.3 Q3.3

the deired joint velocities as I mention in *Q2.3*

4.4 Q3.4

simulation:



4.5 Q3.5

the difference between them is the movement of the joints and the end effector

4.6 Q3.6

in this question we have different goal with respect to robot base and this made the robot so we need to define this to the code in order to know the new tool goal but at least all the algorithm stay the same

THE CODE IS AVAILABLE IN APPENDIX

5 Appendix

```
%% Modelling and Control of Manipulator assignment 3 - Exercise 1: Jacobian matrix
clc;
clear;
close all;
addpath('include');
% The same model of assignment 2
geom_model = BuildTree();
numberOfLinks = size(geom_model,3); % number of manipulator's links.
linkType = zeros(numberOfLinks,1); % specify two possible link type: Rotational, Prismatic.
bTi = zeros(4,4,numberOfLinks);% Trasformation matrix i-th link w.r.t. base
%disp(numberOfLinks);
% joint configurations : to chose a configurations just uncomment.
q1 = [1.3,1.3,1.3,1.3,1.3,1.3,1.3];
q2 = [1.3, 0.4, 0.1, 0, 0.5, 1.1, 0];
q3 = [1.3, 0.1, 0.1, 1, 0.2, 0.3, 1.3];
q4 = [2, 2, 2, 2, 2, 2, 2];

% Compute direct geometry for each configuration: for specific
% configuration,uncomment.
biTei1 = GetDirectGeometry(q1,geom_model,linkType);
biTei2 = GetDirectGeometry(q2,geom_model,linkType);
biTei3 = GetDirectGeometry(q3,geom_model,linkType);
biTei4 = GetDirectGeometry(q4,geom_model,linkType);

% Compute the transformation w.r.t. the base

for i =1:numberOfLinks
    bTi1(:,:,i)= GetTransformationWrtBase(biTei1,i);
    bTi2(:,:,i)= GetTransformationWrtBase(biTei2,i);
    bTi3(:,:,i)= GetTransformationWrtBase(biTei3,i);
    bTi4(:,:,i)= GetTransformationWrtBase(biTei4,i);
end

% computing end effector jacobian :  to chose a configurations uncomment.
    j1=GetJacobian(biTei1, bTi1,linkType);
    j2=GetJacobian(biTei2, bTi2,linkType);
    j3=GetJacobian(biTei3, bTi3,linkType);
    j4=GetJacobian(biTei4, bTi4,linkType);
```

Code of the Exercise 1

```
%% Modelling and Control of Manipulator assignment 3 - Exercise 2 and 3: Inverse Kinematic Control
addpath('include')
model = load("panda.mat"); % don't worry about the warnings
% Simulation Parameters
ts = 0.5;
t_start = 0.0;
t_end = 30.0;
t = t_start:ts:t_end;

% Initial Joints configuration
q_init = [0.0167305,-0.762614,-0.0207622,-2.34352,-0.0305686,1.53975,0.753872]';
% Joint limits
qmin = [-2.8973;-1.7628;-2.8973;-3.0718;-2.8973;-0.0175;-2.8973];
qmax = [2.8973;1.7628;2.8973;-0.0698;2.8973;3.7525;2.8973];
% Initial transformation from <base> to <e-e>
bTe = getTransform(model.franka,[q_init',0,0],'panda_link7'); %useful for save initial end-effector orientation w.r.t robot base
```

the premiere references Exercise 2

```
%% END-EFFECTOR Goal definition
bOge = [0.6; 0.4; 0.4];
edv=-pi/4;
Rotz = [cos(edv) -sin(edv) 0;sin(edv)  cos(edv) 0;0 0 1 ];% Rotation around the z axis with angle edv
 eRge = Rotz;% rotation matrix from end effector to goal effector
bRe(:,:)= bTe(1:3,1:3);% rotation matrix from base to end effector
bTge = [bRe*eRge bOge;0 0 0 1];% transformation matrix from base to goal effector
disp("bTge= ");disp(bTge);
```

END-EFFECTOR Goal definition Exercise 2

```
%% TOOL Goal definition
bOgt = [0.6; 0.4; 0.4];
bdg=[0,0,0.2]';
eTt = [eye(3) bdg; 0 0 0 1];

% Transformation from te base to the tool
bTt = bTe*eTt;
disp("bTt= ");disp(bTt);
tRg = bTt(1:3,1:3)*Rotz;% rotation matrix from tool to goal
disp("tRg= ");disp(tRg);
bTgt = [tRg bOgt; 0 0 0 1];% transformation matrix from base to goal tool
disp("bTgt= ");disp(bTgt);
% Control Proportional Gain
angular_gain = 0.2;
linear_gain = 0.2;
% Preallocation variables
x_dot = zeros(6,1);
error_linear = zeros(3,1);
error_angular = zeros(3,1);
% Start the inverse kinematic control
tool = false; % change to true for using the tool
q = q_init;
```

TOOL Goal definition Exercise 2

```
for i = t
    %% Compute the cartesian error to reach the goal
    if tool == true %compute the error between the tool frame and goal frame

        % Computing transformation matrix from base to end effector
        bTe = getTransform(model.franka,[q',0,0],'panda_link7'); %DO NOT EDIT
        tmp = geometricJacobian(model.franka,[q',0,0],'panda_link7'); %DO NOT EDIT
        bJe = tmp(1:6,1:7); %DO NOT EDIT


        % computing tRgt rotation matrix from tool to goal tool
        tRgt = transpose(bTt(1:3,1:3)) * bTgt(1:3,1:3);

        % angular and linear error computation
        error_angular = (edv)*bTgt(1:3,3)-(edv)*bTt(1:3,3);
        error_linear = bTgt(1:3,4)-bTt(1:3,4);
        % linear velocity is the difference between vector from base to
        % goal tool and base to tool

    else % compute the error between the e-e frame and goal frame
        % Computing transformation matrix from base to end effector
        bTe = getTransform(model.franka,[q',0,0],'panda_link7'); %DO NOT EDIT
        % Computing end effector jacobian w.r.t. base
        tmp = geometricJacobian(model.franka,[q',0,0],'panda_link7'); %DO NOT EDIT
        bJe = tmp(1:6,1:7); %DO NOT EDIT

        % computing the rotation eRg from "ee" to goal"
        eRge = transpose(bTe(1:3,1:3))*bTge(1:3,1:3);

        % angular and linear error computation
        error_angular = (edv)*bTge(1:3,3)-(edv)*bTe(1:3,3);
        error_linear = bTge(1:3,4)-bTe(1:3,4);
        % linear velocity is the difference between vector from base to
        % goal and base to end-effector
    end
```

Compute the cartesian error to reach the goal code Exercise 2

```
%% Compute the reference velocities

% angular velocity
angular_velocity = angular_gain*error_angular;

% linear velocity
linear_velocity = linear_gain*error_linear;

x_dot = [angular_velocity; linear_velocity];
```

Compute desired joint velocities code Exercise 2

```
%% Compute desired joint velocities
if tool == true
    % transformation from "ee" to "tool"
    rot = bTe(1:3,1:3) * eTt(1:3,4);
    skew =[0 -rot(3) rot(2); rot(3) 0 -rot(1); -rot(2) rot(1) 0];
    ps = [eye(3) zeros(3);-skew eye(3)];
    bJt = ps * bJe;% The jacobian from base to tool
    % joint velocity
    q_dot = pinv(bJt)*x_dot;
else
    q_dot = pinv(bJe)*x_dot;
end
```

Compute the reference velocities Exercise 2

```
%% Simulate the robot - implement the function KinematicSimulation()
q = KinematicSimulation(q(1:7), q_dot,ts, qmin, qmax);

% DO NOT EDIT - plot the robot moving
show(model.franka,[q',0,0],'visuals','on');%switch visuals to off for seeing only the frames
hold on
if tool == true
    plot3(bTt(1,4),bTt(2,4),bTt(3,4),'go','LineWidth',15);
    plot3(bOgt(1),bOgt(2),bOgt(3),'ro','LineWidth',5);
else
    plot3(bTe(1,4),bTe(2,4),bTe(3,4),'go','LineWidth',15);
    plot3(bOge(1),bOge(2),bOge(3),'ro','LineWidth',5);
end
drawnow
if(norm(x_dot) < 0.01)
    disp('REACHED THE REQUESTED GOAL POSITION !!!!!!!')
    break
end
if i < t_end
    % function that clean the window.
    cla();
end
```

Implement the function KinematicSimulation Exercise 2

```
%% Exercice 3
% Q3.6

% New tool goal
bTgt = [0.9986 -0.0412 -0.0335 0.6;

         0.0329 -0.0163 0.9993 0.4;

        -0.0417 -0.9990 -0.0149 0.4;

        0 0 0 1];

bOgt = bTgt(1:3,4);
q = q_init;
```

Exercise 3 question Q3.6

```
for i = t
    %% Compute the cartesian error to reach the goal

    % Computing transformation matrix from base to end effector
    bTe = getTransform(model.franka,[q',0,0],'panda_link7'); %DO NOT EDIT
    tmp = geometricJacobian(model.franka,[q',0,0],'panda_link7'); %DO NOT EDIT
    bJe = tmp(1:6,1:7); %DO NOT EDIT
    % ...
    bTt = bTe * eTt;

    % computing tRgt rotation matrix from tool to goal tool
    tTgt = transpose(bTt)*bTg*eTt;
    tRgt = tTgt(1:3,1:3);

    % angular and linear error computation
    error_angular = (edv)*bTgt(1:3,3)-(edv)*bTt(1:3,3);
    error_linear = bTgt(1:3,4)-bTt(1:3,4);
    % linear velocity is the difference between vector from base to
    % goal tool and base to tool

    % Compute the reference velocities

    % angular velocity
    angular_velocity = angular_gain*error_angular;

    % linear velocity
    linear_velocity = linear_gain*error_linear;

    x_dot = [angular_velocity; linear_velocity];

    % Compute desired joint velocities
    % transformation from "ee" to "tool"
    rotz = bTe(1:3,1:3) * eTt(1:3,4);
    skew =[0 -rotz(3) r_et(2); rotz(3) 0 -rotz(1); -rotz(2) rotz(1) 0];
    PS = [eye(3) zeros(3);-skew eye(3)];

    % The jacobian from base to tool is
    bJt = PS * bJe;

    % joint velocity
    q_dot = pinv(bJt)*x_dot;

    % Simulate the robot - implement the function KinematicSimulation()
    q = KinematicSimulation(q(1:7), q_dot,ts, qmin, qmax);

    % DO NOT EDIT - plot the robot moving
    show(model.franka,[q',0,0],'visuals','on');%switch visuals to off for seeing only the frames
    hold on

    plot3(bTt(1,4),bTt(2,4),bTt(3,4),'go','LineWidth',15);
    plot3(bOgt(1),bOgt(2),bOgt(3),'ro','LineWidth',5);

    drawnow
    if(norm(x_dot) < 0.03)
        disp('REACHED THE REQUESTED GOAL POSITION')
        break
    end
    if i < t_end
        % function that clean the window.
        cla();
    end
```

Compute the cartesian error to reach the goal Exercise 3