ROB 315 - TP1: Direct and inverse kinematics

Yan Chen & Dajing Gu

December 2020

1 Introduction

In this practical work (TP), in order to have a better idea of the **geometric** and **kinematic** modeling, a manipulator arm developed by the Interactive Robotics Laboratory of the CEA List is studied. The robot arm used in this practical has 6 revolute joints (j_i with $i = 1, ..., 6$), and the kinematic chain is of serial type, $\boldsymbol{MATLAB}$ is used to perform the tutorial.

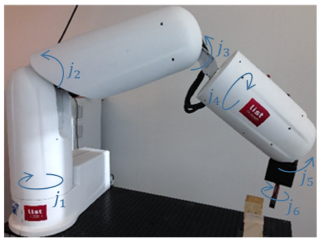

Figure 1.1: Prototype of the robotic manipulation of CEA-LIST

2 Direct geometric model

2.1 Q1

According to the MDH convention, we firstly describe precisely the geometric condition of the robot arm, in which the frames attached to the successive links, the axis names and geometric distances are included.

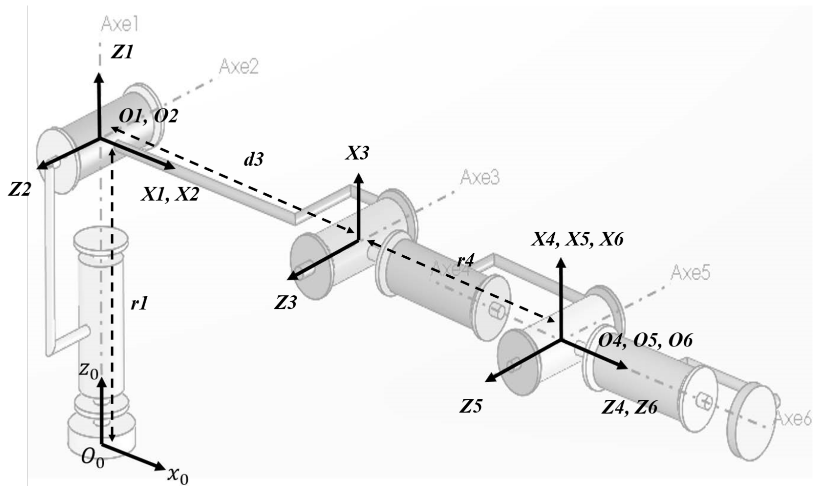

Figure 2.1: Description of the robot's geometry

2.2 Q2

Then we fill in the table with the geometric parameters of the robot, with the parameters presented below:

- α_i : angle between axes Z_{i-1} and Z_i around axis X_{i-1};
- d_i : distance between axes Z_{i-1} and Z_i around axis X_{i-1};
- θ_i : angle between axes X_{i-1} and X_i around axis Z_i;
- r_i : distance between axes X_{i-1} and X_i around axis Z_i;

i	α_i	d_i	θ_i	r_i
1	0	0	q_1	r_1
2	$\pi/2$	0	q_2	0
3	0	d_3	$q_3+\pi/2$	0
4	$\pi/2$	0	q_4	r_4
5	$-\pi/2$	0	q_5	0
6	$\pi/2$	0	q_6	0

Table 2.1: Geometric parameters of the robot

2.3 Q3

In this section, we define several functions to compute the direct geometric model of the robot.

`TransformMatElem(`$\alpha i, di, \theta i, ri$`)` : A generic function whose output argument is the homogeneous transform matrix $\overline{g}$ between two successive frames. The transformation matrix is defined as follows :

$$\overline{g}_{(i-1)i} = \begin{bmatrix} R_{(i-1)i} & P_{(i-1)i} \\ 0_{1\times3} & 1 \end{bmatrix} \tag{2.1}$$

```
function [g] = TransformMatElem(alphai, di, thetai, ri)
g = [[cos(thetai),              -sin(thetai),             0,             di];
     [cos(alphai)*sin(thetai), cos(alphai)*cos(thetai), -sin(alphai), -ri*sin(alphai)];
     [sin(alphai)*sin(thetai), sin(alphai)*cos(thetai), cos(alphai),  ri*cos(alphai)];
     [0,                        0,                        0,             1]];
end
```

`ComputeDGM(`α, d, θ, r`)` : A function which computes the direct geometric model of any robot with series open kinematic chain. The open kinematic chain is calculated as fallows :

$$\overline{g}_{0N}(q) = \overline{g}_{01}(q_1)...\overline{g}_{(i-1)i}(q_i)...\overline{g}_{(N-1)N}(q_N) \tag{2.2}$$

```
function [g_06, g_elem] = ComputeDGM(alpha, d, theta, r)
    g_06 = eye(4);
    N = size(alpha,2);
    g_elem = zeros(size(g_06,1), size(g_06,2), N);
    for i = 1:N
        g_elem(:, :, i) = TransformMatElem(alpha(i), d(i), theta(i), r(i));
        g_06 = g_06 *  g_elem(:, :, i);
    end
end
```

We compute the homogeneous transform matrix $\overline{g}_{0E}$ which gives the position and the orientation of the frame $\mathcal{R}_E$ attached to the end-effector of the robot, expressed in the base frame $\mathcal{R}_0$

```
g0E = ComputeDGM(alpha, d, qi, r)*TransformMatElem(0,0,0,rE);
```

We test the MGD with two joint configurations q_i and q_f defined in Q4.

For q_i, we have :

$$\overline{g}_{0E} = \begin{bmatrix} 0 & 0 & -1 & -0.1 \\ 1 & 0 & 0 & -0.7 \\ 0 & -1 & 0 & 0.3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

For q_f, we have :

$$\overline{g}_{0E} = \begin{bmatrix} -0.71 & 0.71 & 0 & 0.64 \\ 0 & 0 & -1 & -0.1 \\ -0.71 & -0.71 & 0 & 1.14 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

2.4 Q4

We define two joint configurations here $q_i = [-\frac{\pi}{2}, 0, -\frac{\pi}{2}, -\frac{\pi}{2}, -\frac{\pi}{2}, -\frac{\pi}{2}]^t$ and $q_f = [0, \frac{\pi}{4}, 0, -\frac{\pi}{2}, -\frac{\pi}{2}, 0]^t$. Given the two joint configurations, we calculate the values of positions P_x, P_y, P_z and the parameters related to the orientation R_{nq}. Suppose that the rotation matrix is represented by Rodrigues formula. We can retrieve the axis and angle parameters by a inverse internship as follows.

Consider the rotation matrix $R \in SO(3)$ given by :

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.5}$$

The identification of the quantities w and $\theta (0 \leq \theta \leq \pi)$ such that $R = R_{w,\theta}$ is given by :

$$\theta = atan2\left(\frac{1}{2}\sqrt{(r_{32}-r_{23})^2 + (r_{13}-r_{31})^2 + (r_{21}-r_{12})^2},\ \frac{1}{2}(r_{11}+r_{22}+r_{33}-1)\right) \tag{2.6}$$

$$w = \begin{bmatrix} \frac{r_{32}-r_{23}}{2S\theta} & \frac{r_{13}-r_{31}}{2S\theta} & \frac{r_{21}-r_{12}}{2S\theta} \end{bmatrix} \quad \text{if } S\theta \neq 0 \tag{2.7}$$

We define a function `RotMat_to_Angle` which retrieves the axis and angle parameters from a given rotation matrix, which can also be replaced by the pre-defined function `tform2axang` in *MATLAB*.

```
function [P, theta, w] = RotMat_to_Angle(g0E)

% Get the positio
P = g0E(1:3, 4);

% Get the rotation matrix
R = g0E(1:3, 1:3);

% Retrieving the angle parameters from a given rotation matrix
theta = atan2(0.5*sqrt((R(3,2)-R(2,3))^2 + (R(1,3)-R(3,1))^2 + (R(2,1)-R(1,2))^2), ...
    0.5*(R(1,1)+R(2,2)+R(3,3)-1));
```

```

% Retrieving the axis parameters from a given rotation matrix
w = [(R(3,2)-R(2,3))/(2*sin(theta)), (R(1,3)-R(3,1))/(2*sin(theta)), ...
    (R(2,1)-R(1,2))/(2*sin(theta))];

end
```

Define these variables about geometric parameters of robot

```
alpha = [0, pi/2, 0, pi/2, -pi/2, pi/2];
d = [0, 0, 0.7, 0, 0, 0];
qi = [-pi/2, 0, -pi/2, -pi/2, -pi/2, -pi/2];
qf = [0, pi/4, 0, pi/2, pi/2, 0];
r = [0.5, 0, 0, 0.2, 0, 0];
rE = 0.1;
theta = qi;
theta(3) = theta(3) + pi/2;
```

The code as follows is aimed to calculate the values of positions, the angle of rotation θ and axis of rotaion w.

```
% Get the homogeneous transform matrix
g0E = ComputeDGM(alpha, d, theta, r)*TransformMatElem(0,0,0,rE);
% Get the position, the rotation angle and rotation axis
[P, theta, w] = RotMat_to_Angle(g0E);
% Or use directly tform2axang
w_theta = tform2axang(g_0E);
w = w_theta(1:3);
theta = w_theta(4;)
```

For the joint configuration q_i, we have :

$$\begin{aligned} P &= [P_x, P_y, P_z]^t = [-0.1, -0.7, 0.3]^t \\ w &= [-0.5774, -0.5774, 0.5774] \\ \theta &= 2.0944 \end{aligned} \tag{2.8}$$

For the joint configuration q_f, we have :

$$\begin{aligned} P &= [P_x, P_y, P_z]^t = [0.6364, -0.1, 1.1364]^t \\ w &= [0.2811, 0.6786, -0.6786] \\ \theta &= 2.5936 \end{aligned} \tag{2.9}$$

2.5 Q5

The function `PlotFrame(q)` is written to provide a visualization of the position and the orientation of the end-effector frame RE with respect to the base frame R_0 for the joint configurations qi and qf, it provides us also with the visualization of the joints and the links of a robot arm.

Listing 1: Code of the function `PlotFrame`

```
function PlotFrame(q)
    % Parametres
    r4 = 0.2; r1 = 0.5; rE = 0.1;
    r = [r1 0 0 r4 0 0];
    d3 = 0.7; d = [0 0 d3 0 0 0];
    alpha = [0 pi/2 0 pi/2 -pi/2 pi/2];
    theta = q;
    theta(3) = theta(3) + pi/2;

    % DGM of the robot
    [g_06, g_elem]= ComputeDGM(alpha, d, theta, r);
    g_6E = TransformMatElem(0, 0, 0, rE);
    g_0E = g_06 * g_6E;
    origin = [0;0;0] ;
    RotN = eye(3) ;
    hold on

    % Plot R0
    plot_coordinate(origin, RotN, 'green', 'O');
    % Plot the robot body
    for i = 1:length (g_elem)
        position = origin + RotN*g_elem(1:3,4,i);
        RotN = RotN*g_elem(1:3,1:3, i);
        plot3([origin(1) position(1)],[origin(2) position(2)],[origin(3) position(3)], ...
            'color',[96 96 96]/255, 'LineWidth',3);
        scatter3(origin(1),origin(2),origin(3),'black','filled');
        origin = position;
    end
    % Plot the end-effector
    position = g_0E(1:3,4);
    RotN = RotN*g_6E(1:3,1:3);
    plot3([origin(1) position(1)],[origin(2) position(2)],[origin(3) ...
        position(3)],'color',[96 96 96]/255, 'LineWidth',3);
    scatter3(position(1),position(2),position(3),'red','filled');
    %Plot the end-effector RE
    plot_coordinate(position, RotN, 'red', 'E')

    xlabel('x'); ylabel('y'); zlabel('z'); grid on;
    axis equal;
    hold on;
end
```

The function `plot_coordinat(q)` gives us the initial and ending coordinate system for visualization.

Listing 2: Code of the function `plot_coordinate`

```
function plot_coordinate(point, RotN, color, sign)
    XYZ = [1 0 0; 0 1 0; 0 0 1];
    str = ['X','Y','Z'];
    for i = 1:3
        P = 0.1 * RotN * XYZ(i,:)';
        plot3([point(1), point(1)+ P(1)],  [point(2), point(2) + P(2)], ...
            [point(3),point(3)+ P(3)],'Color',  color, 'LineWidth',2);
        text(0.04+point(1)+P(1), 0.04+point(2)+P(2), 0.04+point(3)+P(3), ...
            strcat(str(i)',sign), 'Color', color);
    end
    scatter3(point(1),point(2),point(3), color, 'filled');
end
```

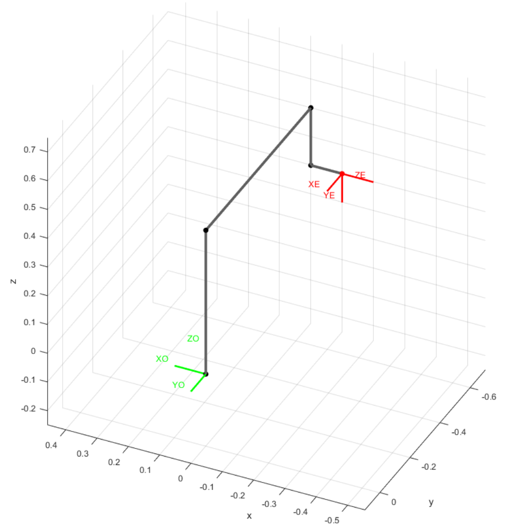

Figure 2.2: Frame visualization for q_i

For q_i, we obtain :

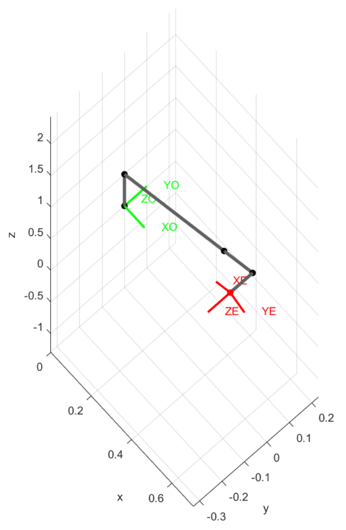

Figure 2.3: Frame visualization for q_f

For q_f, we obtain :

3 Direct kinematic model

The direct kinematic model can help us to express the position and the orientation X of the end-effector as a function of joint variables q, in which the Jocobian matrix is needed.

3.1 Q6

3.1.1 Calculation of the Jacobian matrix ${}^0J$

In using the method of elocities composition, we write a function `ComputeJac`(α, d, θ, r) whose output is the Jacobian matrix ${}^0J$.

The principal of the code is based on the equation (3.1)

$$ {}^0J_i(q) = \begin{bmatrix} R_{0i}\,(Z_i \times p_{iN}) \\ R_{0i}\,(Z_i) \end{bmatrix} \dot{q}_i \tag{3.1} $$

- R_{0i} is the rotation matrix from $\mathscr{R}_0$ to $\mathscr{R}_i$
- Z_i the unit vector of the i^{th} joint $[0\ 0\ 1]^T$
- p_{iN} the translation vector from $\mathscr{R}_i$ to $\mathscr{R}_N$

Listing 3: Code of the function `ComputeJac`

```matlab
function J = ComputeJac(alpha, d, theta, r)
    rE = 0.1;
    [g_06, g_elem] = ComputeDGM(alpha, d, theta, r);
    g_6E  = TransformMatElem(0, 0, 0, rE);
    g_0E = g_06 * g_6E;

    N = size(alpha, 2);

    % Initialisation of the Jacobian matrix
    J = zeros(6,N);
    % unit vector of the ith joint in frame Ri
    z_i = [0 0 1]';

    for i=1:N
        p_iE = zeros(3,1);
        g_0i = eye(4);
        for j=1:i
            g_0i = g_0i*g_elem(:,:,j);
        end

        % Calculate R_0i
        R_0i = g_0i(1:3,1:3);

        % Calculate p_iE (translation vecteur from R_i to R_E)
        p_0E = g_0E(1:3,4);
        p_0i = g_0i(1:3,4);
        p_iE = p_0E - p_0i;

        % Calculate Ji in reference R_0
        oJi = [cross(R_0i*z_i,p_iE) ; R_0i*z_i];

    %Concatenation of J
        J(1:6,i) = oJi;
    end
end
```

⋆ For the joint configuration q_i, we have :

$$ {}^0J = \begin{bmatrix} 0.70 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ -0.10 & -0.20 & -0.20 & 0.10 & 0.0000 & -0.00 \\ 0.00 & 0.70 & -0.00 & -0.00 & -0.1000 & -0.00 \\ 0.00 & -1.00 & -1.00 & -0.00 & -0.0000 & -1.00 \\ 0.00 & -0.00 & -0.00 & -0.00 & -1.0000 & -0.00 \\ 1.0000 & 0.00 & 0.00 & -1.00 & -0.0000 & 0.00 \end{bmatrix} \tag{3.2} $$

⋆ For the joint configuration, we have q_f :

$$
{}^0J = \begin{bmatrix} 0.1000 & -0.6364 & -0.1414 & 0.0707 & -0.0707 & 0.0000 \\ 0.6364 & 0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.6364 & 0.1414 & -0.0707 & -0.0707 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.7071 & 0.7071 & 0.0000 \\ 0.0000 & -1.0000 & -1.0000 & -0.0000 & -0.0000 & -1.0000 \\ 1.0000 & 0.0000 & 0.0000 & 0.7071 & -0.7071 & 0.0000 \end{bmatrix} \tag{3.3}
$$

3.1.2 Calculation of the twist ${}^0\mathscr{V}_{i,N}$

In using the script for Q6, whose principal is based on ${}^0\mathscr{V}_{i,N}\,(O_E) = {}^0J_i(q)\dot{q}$, we can calculate the twist values of the end-effector velocity given in base frame, with the joint velocities $\dot{q} = [0.5, 1.0, -0.5, 0.5, 1.0, -0.5]^t$

Listing 4: Code of for Q6

```
% Q6: Calculate the Jacobian matrix and the twist values

% Parameters
...

% Speed  of joints
q_point = [0.5 1 -0.5 0.5 1 -0.5]';

% Calculate the Jacobian matrix
J = ComputeJac(alpha, d, theta, r)
V = J * q_point
```

⋆ For the joint configuration q_i, we have :

$$
{}^0\mathscr{V}_{i,N}\,(O_E) = [0.35\ \ -0.10\ 0.60\ \ -0.00\ \ -1.00]^t \tag{3.4}
$$

⋆ For the joint configuration q_f , we have :

$$
{}^0\mathscr{V}_{i,N}\,(O_E) = [-0.5510\ 0.3182\ 0.4596\ 1.0607\ \ -0.0000\ 0.1464]^t \tag{3.5}
$$

3.2 Q7

In the rest of the study, the analysis of operational end-effector velocities is restricted to translational velocities via ${}^0J_v(q)$, which is the first three lines of ${}^0J$.

In order to find the preferred direction to transmit velocity in the task space, we can analyze the singular values of ${}^0J$, which are equal to the eigenvalues λ_i of the product J^tJ. The eigen-vector associated with the biggest eigenvalue is the preferred direction that we want to calculate.

The velocity transmission performance in task space is given by the quadratic form, which is presented as an ellipsoid. The ellpsoid's volume indicates the velocity manipulability, given by $\mathscr{W} = \prod_{i=1}^{r} \sigma_i \geq 0$.

Listing 5: Code of for Q7

```
%% Q7: Velocity transmissions
J_v = J(1:3,:)

% Singular value decomposition
[U,S,V] = svd(J_v*J_v.')

% Direction_preferred
[Eigenvalue_max, idx] = max(diag(S));
direction_preferred = V(:,idx)

% Manipulabilty
```

```
eigenvalues = sqrt(diag(S))
manipulabilite = prod(eigenvalues)

% Rotation Axe and angle of rotation
[g_06, g_elem]= ComputeDGM(alpha_, d, theta, r);
g_6E = TransformMatElem(0, 0, 0, rE);
g_0E = g_06 * g_6E;

% Visualisation of the robot
PlotFrame(q)
hold on;

pf = g_0E(1:3,4);
w_theta = tform2axang(g_0E);
w = w_theta(1:3);
theta_ = w_theta(4);
% [theta, w] = RotMattoAngle(g_0E);

% Velocities Ellipsoides
R = sqrt(diag(S));
[x,y,z] = ellipsoid(pf(1),pf(2),pf(3),R(1),R(2),R(3));
Ellipsoide = surf(x,y,z);
alpha 0.2;
Ellipsoide.EdgeColor = 'black';
rotate(Ellipsoide, w, radtodeg(theta_), pf);
title('Velocities Ellipsoides');
```

3.2.1 Configuration q_i

- Singular values

$$\Sigma = \begin{bmatrix} 0.5524 & 0 & 0 \\ 0 & 0.4918 & 0 \\ 0 & 0 & 0.0458 \end{bmatrix} \tag{3.6}$$

- Eigen vectors

$$V = \begin{bmatrix} 0.3660 & 0.9186 & 0.1489 \\ -0.3263 & -0.0232 & 0.9450 \\ 0.8715 & -0.3944 & 0.2913 \end{bmatrix} \tag{3.7}$$

- Preferred direction

$$\begin{bmatrix} 0.3660 & -0.3263 & 0.8715 \end{bmatrix} \tag{3.8}$$

- Velocity manipulabilities

$$\mathscr{W} = 0.1116$$

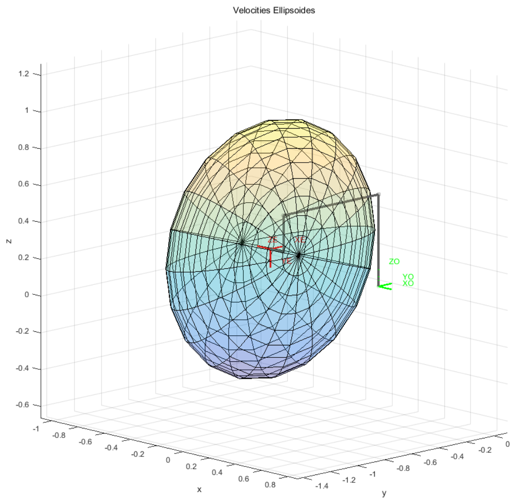

Figure 3.1: Frame visualization for q_f

3.2.2 Configuration q_f

- Singular values

$$\Sigma = \begin{bmatrix} 0.8695 & 0 & 0 \\ 0 & 0.4057 & 0 \\ 0 & 0 & 0.0099 \end{bmatrix} \tag{3.9}$$

- Eigen vectors

$$V = \begin{bmatrix} -0.7114 & 0.0103 & -0.7027 \\ -0.0975 & 0.9888 & 0.1132 \\ 0.6959 & 0.1490 & -0.7025 \end{bmatrix} \tag{3.10}$$

- Preferred direction

$$\begin{bmatrix} -0.7114 & -0.0975 & 0.6959 \end{bmatrix} \tag{3.11}$$

- Velocity manipulabilities

$$\mathscr{W} = 0.0590$$

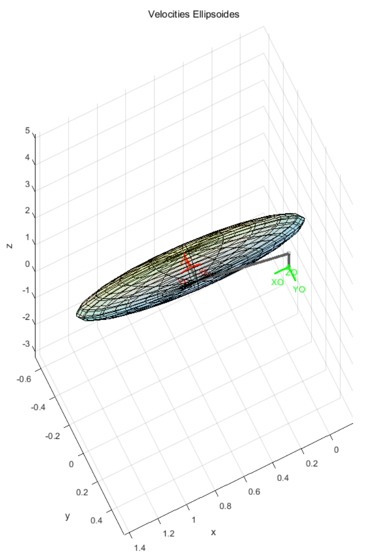

Figure 3.2: Frame visualization for q_f

4 Inverse geometric model

4.1 Q8

In this section, we use Newton-Raphson method to compute inverse geometric model. The function $q^* = ComputeIGM(X_d, q_0, k_{max}, \epsilon_x)$ is defined. X_d is the input argument the desired task position and q_0 is the initial condition. k_{max} is the maximum number of iterations ans $||X_d - f(q_k)|| < \epsilon_x$ is the norm of the tolerated Cartesian error. The iterative optimisation procedure based on Newton-Raphson method as follows :

Algorithm 1 Iterative optimisation procedure

```
1: $k \leftarrow 0$
2: **while** $||J^t(q_k)|| > \epsilon$ **do**
3:     $k \leftarrow k + 1$
4:     ▷ Case of the Gradient-based method
5:     $q_k \leftarrow q_{k-1} + \alpha J^t(q_{k-1})[X_d - f(q_{k-1})]$
6:     ▷ Case of Newton-Raphson-based method
7:     $q_k \leftarrow q_{k-1} + J^{-1}(q_{k-1})[X_d - f(q_{k-1})]$
8: **end while**
9: $q^* \leftarrow q_{k+1}$
10: return $(q^*)$
```

where, the stop criteria is Cartesian error $||X_d - f(q_k)|| < \epsilon_x$ and iteration number $k < k_{max}$. The code as follow :

Listing 6: Code of the function `ComputeIGM`

```
function [q_star] = ComputeIGM(Xd, q0, k_max, epsilon_x)

alpha = [0, pi/2, 0, pi/2, -pi/2, pi/2];
d = [0, 0, 0.7, 0, 0, 0];
```

```
rE = 0.1;
r = [0.5, 0, 0, 0.2, 0, 0];

k = 0;
q = q0';

for k = 1:k_max
    theta = [q(1), q(2), q(3)+pi/2, q(4), q(5), q(6)];
    [g_06, g_elem] = ComputeDGM(alpha, d, theta, r);
    g_0E = g_06*TransformMatElem(0,0,0,rE);
    fq = g_0E(1:3, 4);

    % compute cartesian error
    e = Xd - fq;
    J = ComputeJac(alpha, d, theta, r);
    Ji = J(1:3, :);

    % Newton-Raphson method
    q = q + pinv(Ji)*e;
    if (norm(e,2) < epsilon_x)
        break
    end
end
q_star = q;
end
```

We define the two configurations as follows :

- $X_d = X_{d_i} = (-0.1, -0.7, 0.3)^t$, $q_0 = [-1.57, 0.00, -1.47, -1.47, -1.47, -1.47, -1.47]$, $k_{max} = 100$, $\epsilon_x = 1mm$
- $X_d = X_{d_f} = (0.64, -0.10, 1.14)^t$, $q_0 = [0, 0.80, 0.00, 1.00, 2.00, 0.00]$, $k_{max} = 100$, $\epsilon_x = 1mm$

Listing 7: Test code for `ComputeIGM`

```
% compute the qstar
% first conficuration
% Xd = [-0.1, -0.7, 0.3]';
% q0 = [-1.57, 0.00, -1.47, -1.47, -1.47, -1.47];
% k_max = 100;
% epsilon_x = 0.001;

% second configuration
 Xd = [0.64, -0.1, 1.14]';
 q0 = [0, 0.8, 0, 1, 2, 0];
 k_max = 100;
 epsilon_x = 0.001;

q_star1 = ComputeIGM(Xd, q0, k_max, epsilon_x);

% Compute the error
theta_veri = [q_star1(1), q_star1(2), q_star1(3)+pi/2, q_star1(4), q_star1(5), q_star1(6)];
[g_06,g_element] = ComputeDGM(alpha, d, theta_veri, r);
g_0E =g_06*TransformMatElem(0,0,0,rE);
diff = abs(Xd-g_0E(1:3, 4));
```

For the first configuration, we have :

$$q* = [-1.5725, 0.0132, -1.5232, -1.4452, -1.4819, -1.4700] \tag{4.1}$$

$$Error = 10^{-7} * [0.0024, 0.5096, 0.0165] \tag{4.2}$$

For the second configuration, we have :

$$q* = [-0.0246, 0.7643, -0.1782, 1.0017, 1.5693, 0.0000] \tag{4.3}$$
$$Error = 10^{-7} * [0.2770, 0.4613, 0.5327] \tag{4.4}$$

According to the small error, we can see that the function ComputeIGM has great accuracy to compute inverse geometric model.

5 Inverse kinematic model

5.1 Q9

In this section, we define two functions. The first one is `ComputeIKM(`X_{di}`, `X_{df}`, `V`, `T_e`, `qi`)`. X_{di} is the initial position and X_{df} is the final position. V is the speed of the rectilinear motion and T_e is the sampled period. q_i is the initial configuration of the robot. The second one is `PlotTrajec(`X_d`, `q`)` that can plot the trajectory of robot, position and the orientation of the end-effector.

Listing 8: Code of the function `ComputeIKM`

```matlab
function [Xd, q] = ComputeIKM(Xdi, Xdf, V, Te, qi)

kmax = 100;
epsilon_x = 0.001;

% compute the distance between initial point and final point
distance = norm(Xdf-Xdi, 2);

% compute step length every Te
step_length = V*Te*(Xdf-Xdi)/distance;

%compute the total times
times= distance/V;

% Initiation
Xd = Xdi;
Xdk = Xdi;
q = qi;
qk = qi;
t = 0;

while(t≤temps)
    Xdk = Xdk + step_length;
    qk = ComputeIGM(Xdk, qk', kmax, epsilon_x);
    Xd = [Xd, Xdk];
    q = [q qk];
    t = t+Te;
end
end
```

Listing 9: Code of the function `PlotTrajec`

```matlab
function PlotTrajec(Xd,q)
    figure
    set(gcf,'Name','Plot trajectory')
    N = size(q,2);
    for i = 1:round(N/4):N
        PlotFrame(q(:,i)');
        hold on;
    end
    plot3(Xd(1,:), Xd(2,:), Xd(3,:), 'm--', 'color', 'magenta');
end
```

Listing 10: Code of `Test`

```
% plot trajectoire
Xdi = [-0.1, -0.7, 0.3];
Xdf = [0.64, -0.1, 1.14];
V = 1;
Te = 0.001;

[Xd, q] = ComputeIKM(Xdi, Xdf, V, Te, qi);
PlotTrajec(Xd, q);
```

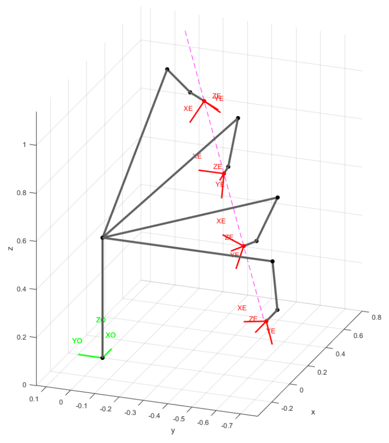

Figure 5.1: Trajectory of end-effector

In the figure 5.1, the dotted line is the trajectory and these sampled points are red. We can see that the robot follows the trajectory perfectly. Robot's joint configurations are presented very well.

5.2 Q10

In this section, the temporal evolution of the joint variables q_1 to q_6 calculated in the previous question is plotted. There are limitations to the value minimum and maximum for each joint variable,

$$q_{\min} = \left[-\pi, -\frac{\pi}{2}, -\pi, -\pi, -\frac{\pi}{2}, -\pi\right] \tag{5.1}$$

and

$$q_{\max} = \left[0, \frac{\pi}{2}, 0, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}\right] \tag{5.2}$$

Listing 11: Code of the function `PlotEvolution`

```
function PlotEvolution(q, qmin, qmax)
    figure;
    set(gcf,'Name','Evolution of q');
    for i=1:6
        % Plot evolution
        subplot(2,3,i);
        hold on;
        plot(q(i,:));
        % Plot the value extremum
        plot(ones(1,length(q)).*qmax(i), '--', 'color', 'red');
        plot(ones(1,length(q)).*qmin(i), 'red');
        % Tune the y axis
        y_max = max(qmax(i), max(q(i,:)));
        y_min = min(qmin(i), min(q(i,:)));
        ylim([y_min-0.2 y_max+0.2]);
        % legend and label
        grid on
        xlim([0 length(q)]);
        title(sprintf('q_{%d}',(i)));
        xlabel('time (ms)') ;
        ylabel('angle (rad)');
        legend(sprintf('q_{%d}',(i)),'q_{max}', 'q_{min}');
    end
```

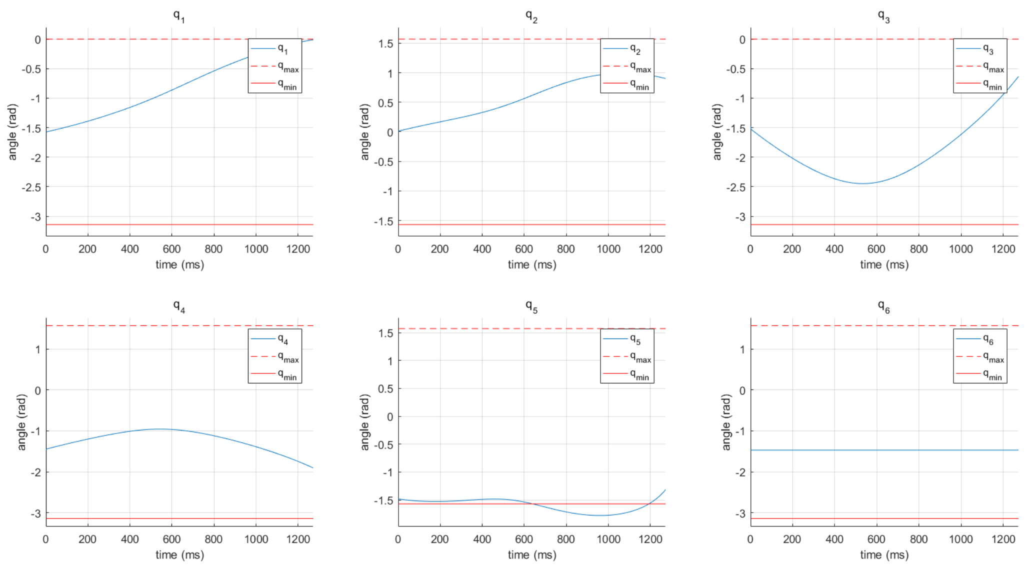

Figure 5.2: Evolution of q

Even though the end effector has followed the reference trajectory, the limitations on the extreme values on each joint are not obeyed. According to the figure 5.2, sometimes the value of q_6 is smaller than the corresponding minimum value, which can not cater to our satisfaction. This phenomenon of out of boundary may cause some problems in the control of the robot in real life.

In order to avoid this kind of problem, the algorithm used in Q9 need to be modified, which will be finished in the next question. A secondary task aiming at keeping some distance from the articular stops q_{min} and q_{max} will be considered.

5.3 Q11

In this section, a secondary task aiming at keeping some distance from the articular stops q_{min} and q_{max} is considered. Based on functions already written in the previous questions, two new functions are created to solve the problem of out of boundary : `ComputeIKMlimits`$(X_{d_i}, X_{d_f}, V, T_e, q_i, q_{min}, q_{max})$ and `ComputeIGMlimits`$(X_{d_i}, X_{d_f}, V, T_e, \alpha_H, q_{min}, q_{max})$.

The general solution of the non-weighted norm problem is

$$\dot{q}^* = J^{\#}\dot{X}_d + \left(I_n - J^{\#}J\right)\dot{q}_0 \tag{5.3}$$

In using the *projected gradient technique*, we have

$$\dot{q}_0 = -\alpha\nabla_q H(q) = -\alpha \begin{pmatrix} \frac{\partial H}{\partial q_1} \\ \vdots \\ \frac{\partial H}{\partial q_n} \end{pmatrix} \tag{5.4}$$

With H the following potential function considered to be minimized:

$$H_{lim}(q) = \sum_{i=1}^{n} \left(\frac{q_i - \bar{q}_i}{q_{\max} - q_{\min}} \right)^2 \quad \text{where } \bar{q}_i = \frac{q_{\max} - q_{\min}}{2}$$

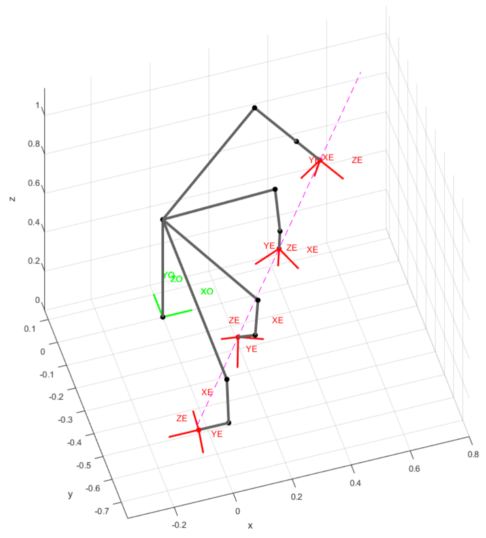

Figure 5.3: Trajectory of end-effector

It can be seen from the figure 5.3 that the end-effector is in good control to move along the reference rtrajectory. Then according to the figure 5.4, the values of all the joint angles are between the two extremes in the whole procedure of movement, which means the secondary task has succeeded in keeping the joints angles from the articular stops q_{min} and q_{max}.

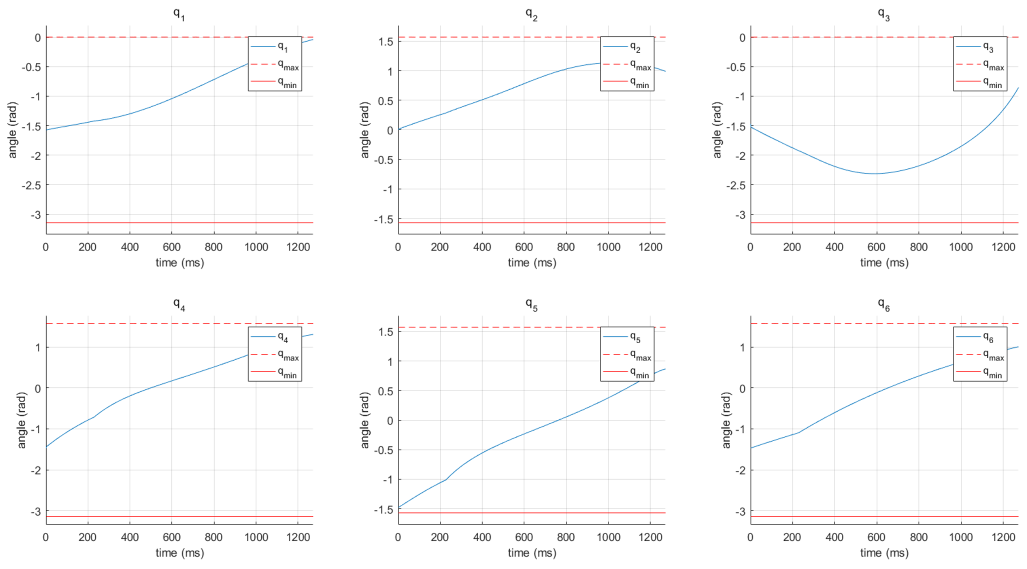

Figure 5.4: Evolution of q

The codes below are the codes used in this question.

Listing 12: Code of the function `ComputeIGMlimites`

```
function [q_star] = ComputeIGMlimites(Xd, q0, k_max, epsilon_x, aH, qmin, qmax)

% Parameters
alpha = [0, pi/2, 0, pi/2, -pi/2, pi/2];
d = [0, 0, 0.7, 0, 0, 0];
rE = 0.1;
r = [0.5, 0, 0, 0.2, 0, 0];
k = 0;
q = q0';

for k = 1:k_max
    theta = [q(1), q(2), q(3)+pi/2, q(4), q(5), q(6)];
    [g_06, g_elem] = ComputeDGM(alpha, d, theta, r);
    g_0E = g_06*TransformMatElem(0,0,0,rE);
    fq = g_0E(1:3, 4);

    % compute cartesian error
    e = Xd - fq;

    J = ComputeJac(alpha, d, theta, r);
    Ji = J(1:3, :);

    % Compute the potential function
    q_= (qmax - qmin)/2;
    H = [];
    for i = 1:size(q_,2)
        H = [H;2*(q(i)-q_(i))/(qmax(i) - qmin(i))^2];
    end

    %  Solution of the quadratic optimization
    q = q + pinv(Ji)*e + (eye(size(Ji,2))-(pinv(Ji)*Ji))*(-aH * H')';

    if (norm(e,2) < epsilon_x)
        break
    end
end
q_star = q;
end
```

Listing 13: Code of the function `ComputeIKMlimites`

```
function [Xd, q] = ComputeIKMlimites(Xdi, Xdf, V, Te, qi, qmin, qmax)

% Parameters
kmax = 100;
epsilon_x = 0.001;
aH = 0.005;

% compute the distance between initial point and final point
distance = norm(Xdf-Xdi, 2);

% compute step length every Te
step_length = V*Te*(Xdf-Xdi)/distance;

%compute the total time
times = distance/V;

% Initiation
Xd = Xdi;
Xdk = Xdi;
q = qi;
qk = qi;
```

```
t = 0;

while(t≤times)
    Xdk = Xdk + step_length;
    qk = ComputeIGMlimites(Xdk, qk', kmax, epsilon_x, aH, qmin, qmax);
    Xd = [Xd, Xdk];
    q = [q, qk];
    t = t+Te;
end

end
```

Listing 14: Test code for Q11

```
% Q11
% Parameters
Xdi = [-0.1 -0.7 0.3]';
Xdf = [0.64 -0.1 1.14]';
V = 1;
Te = 1e-3;
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi];
qmax = [0 pi/2 0 pi/2 pi/2 pi/2];

q0 = [-1.57, 0.00, -1.47, -1.47, -1.47, -1.47]';
%  q0 = [0, 0.8, 0, 1, 2, 0];

[XTot, qTot] = ComputeIKMlimites(Xdi, Xdf, V, Te, q0, qmin, qmax);

PlotTrajec(XTot, qTot);

PlotEvolution(qTot, qmin, qmax)
```