ROB315-TP

Kai ZHANG, Mengyu PAN

December 2020

1 Q1: find the successive links of the robot according to the MDH convention

We draw arrows in the figure 1 according to the MDH convention.

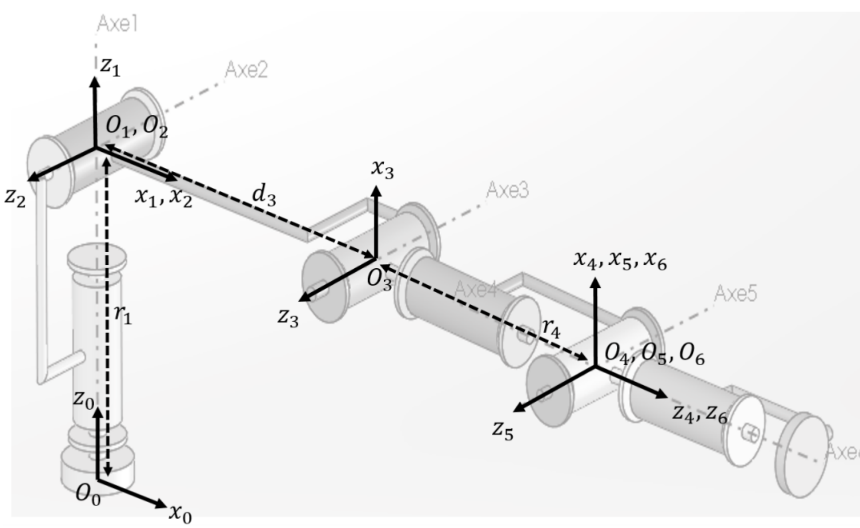

Figure 1: The Description of the robot's geometry

2 Q2: MDH Table

The parameters that we are interested in are shown below:

- α_i is the angle that axe x_{i-1} turns.
- d_i is the distance that axe x_{i-1} turns.
- θ_i is the angle that axe z_{i-1} turns.

i	α_i	d_i	θ_i	z_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 1: MDH Table

- z_i is the distance that axe z_{i-1} turns.

In the table 1, we have completed all part in the MDH table.

3 Q3: Computation of the direct geometric model

3.1 Q3a: Computation of homogeneous transform matrix

The homogeneous transform matrix is:

$$\bar{g}_{sd} = \begin{bmatrix} R_{sd} & p_{sd} \\ 0_{1*3} & 1 \end{bmatrix}$$

The code TransformMatElem(α_i, d_i, θ_i, r_i) is implemented below:

```
function g=TransformMatElem(alpha,d,theta,r)
    g = [cos(theta), -sin(theta), 0, d;
         cos(alpha)*sin(theta), cos(alpha)*cos(theta), -sin(alpha), -r*sin(alpha);
         sin(alpha)*sin(theta), sin(alpha)*cos(theta), cos(alpha), r*cos(alpha);
         0, 0, 0, 1;];
end
```

3.2 Q3b: Computation of the direct geometric model

We compute the direct geometric model by calculating homogeneous transform matrix for each link with TransformMatElem(α_i, d_i, θ_i, r_i). The function ComputeDGM(α, d, θ, r) is shown below:

```
function g=ComputeDGM(alpha,d,theta,r)
    g = diag([1,1,1,1]);
    i = 0;
    while i < length(alpha)
        i = i + 1;
        g1 = TransformMatElem(alpha(i),d(i),theta(i),r(i));
        g = g * g1;
```

```
    end
end
```

3.3 Q3c: Computation of the direct geometric model for the Robot

Here is our test function:

```
function testQ3
    alpha = [0,pi/2,0,pi/2,-pi/2,pi/2];
    d = [0,0,0.7,0,0,0];
    theta = [0,0,pi/2,0,0,0];
    r = [0.5,0,0,0.2,0,0];
    ComputeDGM(alpha,d,theta,r)*ComputeDGM(0,0,0,0.1)
end
```

Based on the MDH Table in Q2 and supposing the q is zero, the result is below:

$$ans = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4 Q4: Computation of the direct geometric model for the Robot

Based on the function in Q3, we can use it to calculate the direct geometric model for initial state and final state. Here is the code:

```
function testQ4
    alpha = [0,pi/2,0,pi/2,-pi/2,pi/2];
    d = [0,0,0.7,0,0,0];
    theta = [0,0,pi/2,0,0,0];
    r = [0.5,0,0,0.2,0,0];
    qi = [-pi/2,0,-pi/2,-pi/2,-pi/2,-pi/2];
    qf = [0,pi/4,0,pi/2,pi/2,0];
    %% initial state
    gi = ComputeDGM(alpha,d,theta+qi,r)*ComputeDGM(0,0,0,0.1)
    theta_i = atan2(0.5*sqrt(pow2(gi(3,2)-gi(2,3))+pow2(gi(1,3)-gi(3,1)) ...
    +pow2(gi(2,1)-gi(1,2))),0.5*(gi(1,1)+gi(2,2)+gi(3,3)-1))
    %% final state
    gf = ComputeDGM(alpha,d,theta+qf,r)*ComputeDGM(0,0,0,0.1)
    theta_f = atan2(0.5*sqrt(pow2(gf(3,2)-gf(2,3))+pow2(gf(1,3)-gf(3,1))...
    +pow2(gf(2,1)-gf(1,2))),0.5*(gf(1,1)+gf(2,2)+gf(3,3)-1))
end
```

For the initial state, the joint configuration is

$$q_i = [-\pi/2, 0, -\pi/2, -\pi/2, -\pi/2, -\pi/2]^t$$

Its position is :

$$[P_x, P_y, P_z] = [-0.1, -0.7, 0.3]$$

and its rotation angle is :

$$\theta_i = 2.0944$$

For the final state, the joint configuration is

$$q_i = [0, \pi/4, 0, \pi/2, \pi/2, 0]^t$$

Its position is :

$$[P_x, P_y, P_z] = [0.6364, -0.1, 1.1364]$$

and its rotation angle is :

$$\theta_i = 2.3126$$

5 Q5: Visualize the position and rotation of end-effector

To visualize its position, we need to know the position of the joint.

$$p(t) = R_{0i}(t)p$$

With the help of homogeneous transform matrix, we can know the rotation matrix.

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

For $q = q_i$, the result is shown in the figure 2.For $q = q_f$, the result is shown in the figure 3.

The code is shown below.

```
% Q5
clear
close all
qi=[-pi/2,0,-pi/2,-pi/2,-pi/2,-pi/2];
PlotFrame(qi);
```

```
function PlotFrame(q)
% parameters
alpha=[0,pi/2,0,pi/2,-pi/2,pi/2];
d=[0,0,0.7,0,0,0];
```

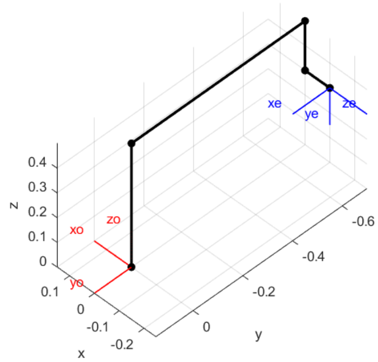

Figure 2: visualization of the position and the orientation of the end-effector frame with $q = q_i$

```
r=[0.5,0,0,0.20,0,0];
t=[0,0,pi/2,0,0,0];
theta=q+t;
re=0.1;

% transformation matrix
M_0e=ComputeDGM(alpha,d,theta,r)*TransformMatElem(0,0,0,re);

% origin point
origin=[0;0;0];
init_x=[1;0;0];
init_y=[0;1;0];
init_z=[0;0;1];
plot_scale=0.1;

% initial rotation matrix
M_00=eye(3);

% plot the links and joints
joint_position=[origin];
scatter3(origin(1),origin(2),origin(3),'k','filled');
hold on;
for i =1:length(alpha)
   M_0i=ComputeDGM(alpha(1:i),d(1:i),theta(1:i),r(1:i));
   position=origin+M_00*M_0i(1:3,4);
   joint_position=[joint_position position];
end
% end effector
position=origin+M_00*M_0e(1:3,4);
```

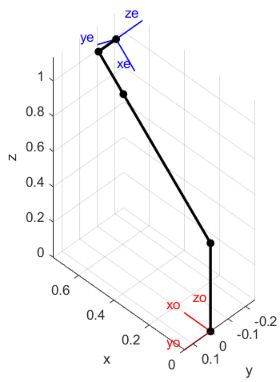

Figure 3: visualization of the position and the orientation of the end-effector frame with $q = q_f$

```
joint_position=[joint_position position];
% plot position
plot3(joint_position(1,:),joint_position(2,:),joint_position(3,:),'-ok','LineWidth',2);
hold on;
scatter3(position(1),position(2),position(3),'r','filled');
hold on;

% plot x,y,z axis
% origin
x_extend=plot_scale.*M_00*init_x;
y_extend=plot_scale.*M_00*init_y;
z_extend=plot_scale.*M_00*init_z;
PlotAxis(origin,x_extend,y_extend,z_extend,'o');


% end effector
x_extend=plot_scale.*M_0e(1:3,1:3)*init_x;
y_extend=plot_scale.*M_0e(1:3,1:3)*init_y;
z_extend=plot_scale.*M_0e(1:3,1:3)*init_z;
PlotAxis(position,x_extend,y_extend,z_extend,'e');

title("Follow the trajectory");
xlabel('x');
ylabel('y');
zlabel('z');
grid on;
axis equal;
end
```

6 Q6: the Jacobin matrix

For this question, we have

$$V_{O,E} = \begin{bmatrix} V_{O,E}(OE) \\ \omega_{O,E} \end{bmatrix} = \begin{bmatrix} J_V(q) \\ J_\omega \end{bmatrix} * \dot{q} \quad = J(q) * \dot{q}$$

- R_{0i} is the rotation matrix from R_0 to R_i.
- Z_i is the vector $[0, 0, 1]^T$.
- p_{iN} is the translation vector from R_i to R_N.

The code is shown below:

```
function J=ComputeJac(alpha,d,theta,r)
    J = zeros(6, length(theta));
    i = 0;
    gF = ComputeDGM(alpha,d,theta,r);
    Z = [0,0,1];
    while i < length(theta)
        i = i +1;
        j = 0;
        g = diag([1,1,1,1]);
        while j < i
           j = j +1;
           g1 = TransformMatElem(alpha(j),d(j),theta(j),r(j));
           g = g * g1;
        end
        giN = (gF - g);
        piN = giN(1:3,4);
        J(1:6,i) = [cross(g(1:3,1:3) * Z',piN); g(1:3,1:3) * Z'];
    end
end
```

We have already known the joint velocity

$$\dot{q} \quad = [0.5, 1, -0.5, 0.5, 1, -0.5]^t$$

For $q = q_i$, the value of the twists at O_E is

$$V_{O,E} = [0.35, -0.1, 0.6, 0, -1, 0]^t$$

For $q = q_f$, the value of the twists at O_E is

$$V_{O,E} = [0.5510, 0.3182, 0.4596, 1.0607, 0, 0.1464]^t$$

7 Q7: analyze the transmission of velocity

In this question, we need to calculate the preferred direction to transmit velocity in the task space with different configurations of the manipulator. Besides, the velocity manipulabilities are required to be specified.

The preferred direction can be obtained by calculating the eigenvector with the maximum eigenvalue σ_i. The velocity manipulability can be obtained by

$$w = \prod_{i=1}^{r} \sigma_i \geq 0$$

.

The corresponding code is shown as follows.

```
% Q7
J=ComputeJac(alpha,d,theta,r);
% translation and rotation matrix
JRT=J(1:3,:);
% decomposition in sigular values
[U,S,V]=svd(JRT*JRT');
% the preferred direction
[m,inx]=max(diag(S));
direction=V(:,idx);
% calculate the manipubility
sigma=sqrt(diag(S));
manipulability=prod(sigma);
% calculate the transformation matrix
M_0e=ComputeDGM(alpha,d,theta,r)*TransformMatElem(0,0,0,re);
PlotEllipse(JRT,qf,M_0e)
```

7.1 Configuration qi

When we choose qi as the configuration, the calculated the preferred direction can be expressed as vector $V = [0.3659; -0.3263; 0.8715]$. The manipulability is $W = 0.1116$.

The matrix of eigenvalues is

$$diag(\sigma) = \begin{bmatrix} 0.7432 & 0 & 0 \\ 0 & 0.7013 & 0 \\ 0 & 0 & 0.2140 \end{bmatrix}$$

The velocity episode is shown in Figure 4

7.2 Configuration qf

When we choose qf as the configuration, the calculated preferred direction can be expressed as vector $V = [-0.7141; -0.0975; 0.6959]$. The manipulability is $W = 0.0590$.

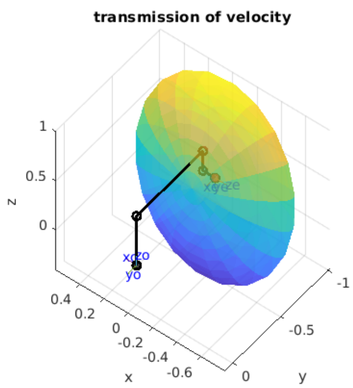

Figure 4: The transmission of velocity when configuration is qi

The matrix of eigenvalues is

$$diag(\sigma) = \begin{bmatrix} 0.9324 & 0 & 0 \\ 0 & 0.6369 & 0 \\ 0 & 0 & 0.0994 \end{bmatrix}$$

The velocity episode is shown in Figure 5

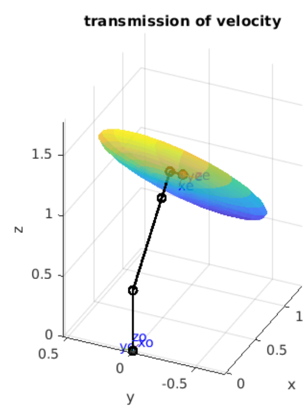

Figure 5: The transmission of velocity when configuration is qf

7.3 Code

```
function PlotEllipse(JRT,q,M_0e)
% decomposition in sigular values
```

```
[U,S,V]=svd(JRT*JRT');
% the preferred direction
[m,inx]=max(diag(S));
direction=V(:,inx);
% calculate the manipubility
sigma=sqrt(diag(S));
manipulability=prod(sigma);

PlotFrame(q);

[ang,ax]=AngleFromRMatrix(V);
pf=M_0e(1:3,4);
[x,y,z]=ellipsoid(pf(1),pf(2),pf(3),sigma(1),sigma(2),sigma(3));
ellip=surf(x,y,z);
alpha 0.7;
ellip.EdgeColor="none";
rotate(ellip,ax,radtodeg(ang),pf);
end
```

8 Q8: Inverse geometric model

This question asks us to develop an iterative method to calculate the position of each joint according to the task position and initial condition.

The corresponding code is described as follows:

```
function q_star=ComputeIGM(Xd,q0,kmax,epsx)
% parameters
re=0.1;
r=[0.5,0,0,0.20,0,0];
d=[0,0,0.7,0,0,0];
alpha=[0,pi/2,0,pi/2,-pi/2,pi/2];
t=[0,0,pi/2,0,0,0]'; % change of theta

for k=1:kmax
    theta=q0+t;
    % compute jacobian matrix
    J=ComputeJac(alpha,d,theta,r);
    JRT=J(1:3,:);

    M_0e=ComputeDGM(alpha, d,theta,r)*TransformMatElem(0,0,0,re);%M_06*M6e
    % translation vector
    X=M_0e(1:3,4);

    %calculate the difference
    delta_X=Xd-X;
    %update q
```

```
    q0=q0+pinv(JRT)*delta_X;
    % calculate error
    e=norm(delta_X,2);

    if e<epsx
        break;
    end
end
q_star=q0;
end
```

8.1 Case 1

When

$$X_d = X_{d_i} = (0.1, 0.7, 0.3)^t$$
$$q0 = [1.57, 0.00, 1.47, 1.47, 1.47, 1.47]$$
$$k_{max} = 100, \epsilon_x = 1mm$$

, we can calculate q^*.

```
% Q8
clear;

Xd=[-0.1,-0.7,0.3]';
kmax=100;
epsx=0.001;
q0=[-1.57,0,-1.47,-1.47,-1.47,-1.47]';
q_star=ComputeIGM(Xd,q0,kmax,epsx)
```

We can get

$$q^* = [-1.5725, 0.0132, -1.5232, -1.4452, -1.4819, -1.4700]$$

To evaluate the accuracy, we use the calculated q^* to compute the position of end-effector and compare it with the X_d.

```
alpha=[0,pi/2,0,pi/2,-pi/2,pi/2];
d=[0,0,0.7,0,0,0];
r=[0.5,0,0,0.20,0,0];
re=0.1;
t=[0,0,pi/2,0,0,0]';
q_1=q_star+t;
M_0e=ComputeDGM(alpha,d,q_1,r)*TransformMatElem(0,0,0,re);
diff1=abs(Xd-M_0e(1:3,4))
```

The difference is $diff = 1.0e-07 * [0.0024, 0.5096, 0.0165]$ which means our result is correct.

8.2 Case 2

When $X_d = X_{d_i} = (0.64, 0.10, 1.14)^t, q0 = [0, 0.80, 0, 1, 2, 0], k_{max} = 100, \epsilon_x = 1mm$, we can calculate q^*.

```
% case 2
Xd=[0.64, -0.10, 1.14]';
kmax=100;
epsx=0.001;
q0=[0, 0.80, 0, 1.0, 2.0, 0.0]';
q_star=ComputeIGM(Xd,q0,kmax,epsx)
```

We can get

$$q^* = [-0.0246, 0.7643, -0.1782, 1.0017, 1.5693, 0.0000]$$

To evaluate the accuracy, we use the calculated q^* to compute the position of end-effector and compare it with the X_d.

```
alpha=[0,pi/2,0,pi/2,-pi/2,pi/2];
d=[0,0,0.7,0,0,0];
r=[0.5,0,0,0.20,0,0];
re=0.1;
t=[0,0,pi/2,0,0,0]';
q_2=q_star+t;
M_0e=ComputeDGM(alpha,d,q_2,r)*TransformMatElem(0,0,0,re);
diff2=abs(Xd-M_0e(1:3,4))
```

The difference is $diff = 1.0e - 07 * [0.2770, 0.4613, 0.5327]$ which means our result is correct.

9 Q9: Inverse kinematic model

This question asks to calculate the configuration when moving the end-effector from position X_{d_i} to position X_{d_f}. To achieve the objective, we sampled the line from Xdi to Xdf into many units according to the time intervals. Therefore, this question can be simplified as the calculation of configuration at each time instant kT_e.

The code is described as follows:

```
% Q9
clear;
Xdi=[-0.1,-0.7,0.3]';
Xdf=[0.64,-0.10,1.14]';
```

```
V=1;
Te=0.001;

q0=[-1.57,0,-1.47,-1.47,-1.47,-1.47]';
[qdk,Xdk]=computeIKM(Xdi,Xdf,V,Te,q0);

plot3(Xdk(1,:),Xdk(2,:),Xdk(3,:),"--m",'LineWidth',1);
hold on

samples=6
step=round(length(qdk)/samples);
for i =1:step:length(qdk)
    PlotFrame(qdk(:,i)');
    hold on;
end
```

The corresponding result of following the trajectory is demonstrated in Figure 6. From the figure, we can see that the position of end-effector is exacted in the line from X_{d_i} to X_{d_f} and we can conclude that the computed configuration is correct.

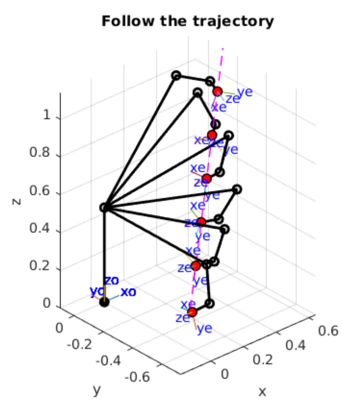

Figure 6: Follow the trajectory(6 samples)

```
function [qdk,Xdk]=computeIKM(Xdi,Xdf,V,Te,qi)
% calculate the action to move from Xdi to Xdf
% qdk: corresponding action
% Xdk: sampled points values

% parameters
kmax=100;
epsx=0.001;
```

```

% calculate the transformation vector from Xdi to Xdf
dist=norm(Xdf-Xdi,2);
step=V*Te;
step_vector=step*(Xdf-Xdi)/dist;
total_T=dist/V; % required time for movement

% initialize the parameters for iteration
Xdk=Xdi;
Xd=Xdi;
qdk=qi;
qd=qi;
t=0;
while(t<total_T)
    Xd=Xd+step_vector;
    % compute configuration
    qd=ComputeIGM(Xd,qd,kmax,epsx);

    % add sampled points
    Xdk=[Xdk,Xd];
    % add calculated configuration
    qdk=[qdk,qd];

    t=t+Te;
end
end
```

10 Q10: Plot the temporal evolution

This question asks to plot the calculated q in Q9 and two limits of q, q_{min} and q_{max}.

10.1 Results

The results are illustrated as follows.

```
% Q10
clear;
Xdi=[-0.1,-0.7,0.3]';
Xdf=[0.64,-0.10,1.14]';
V=1;
Te=0.001;
qmin = [-pi,-pi/2 ,-pi,-pi, -pi/2 , -pi];
qmax = [0,pi/2 ,0,pi/2, pi/2 ,pi/2];

q0=[-1.57,0,-1.47,-1.47,-1.47,-1.47]';
```

```
[qdk,Xdk]=computeIKM(Xdi,Xdf,V,Te,q0);

plotEvolution(qdk,qmin,qmax)
```

As shown in Figure 7, we can see that some configuration, for example q_5 does not always meet the constraints, which indicates the configuration is not suitable. To overcome this problem, we need to take the limits into consideration during calculating the configuration of the joints.

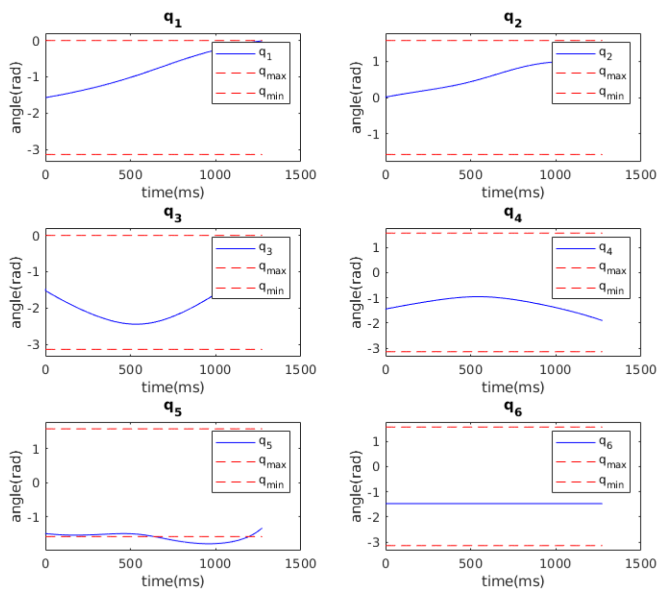

Figure 7: Configuration and limits

10.2 Code

```
function plotEvolution(q,qmin,qmax)
% plot configuration q
for i=1:6
   subplot(3,2,i);
   plot(q(i,:),"b-");
   hold on

   % plot the boundary
   plot(ones(1,size(q,2)).*qmax(i),'--r');
   plot(ones(1,size(q,2)).*qmin(i),'--r');
```

```

    max_y=max(max(q(i,:)),qmax(i));
    min_y=min(min(q(i,:)),qmin(i));
    ylim([min_y-0.2,max_y+0.2]);

    legend(sprintf('q_{%d}',i),'q_{max}','q_{min}');
    title(sprintf('q_{%d}',i));
    xlabel('time(ms)');
    ylabel('angle(rad)');
end
end
```

11 Q11:Compute configuration with limits

This question asks us to compute the configuration with the limits proposed in Q10. We use the minimize mean square error(MMSE) algorithm to find the most appropriate configuration under the constraints.

The code is described as follows:

```
% Q11
clear;
Xdi=[-0.1,-0.7,0.3]';
Xdf=[0.64,-0.10,1.14]';
V=1;
Te=0.001;
% q0=[-1.57,0,-1.47,-1.47,-1.47,-1.47]';

q0=[-pi/2,0,-pi/2,-pi/2,-pi/2,-pi/2]';
q_min=[-pi,-pi/2,-pi,-pi,-pi/2,-pi]';
q_max=[0,pi/2,0,pi/2,pi/2,pi/2]';
[qdk,Xdk]=ComputeIKMlimits(Xdi,Xdf,V,Te,q0,q_min,q_max);

plotEvolution(qdk,q_min,q_max)

figure(2)
plot3(Xdk(1,:),Xdk(2,:),Xdk(3,:),"--m",'LineWidth',1);
hold on

samples=6;
step=round(length(qdk)/samples);
for i =1:step:length(qdk)
    PlotFrame(qdk(:,i)');
    hold on;
end
```

We can obtain the trajectory of the configuration, shown in Figure 8. From the generated trajectory, we can verify the correctness of the configurations. From Figure 9, we can find the improvement comparing to Q9. All the configurations meet their constraints and generate the target trajectory.

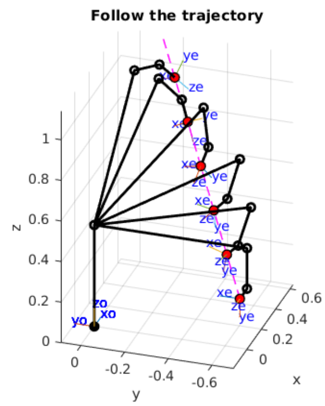

Figure 8: The followed trajectory

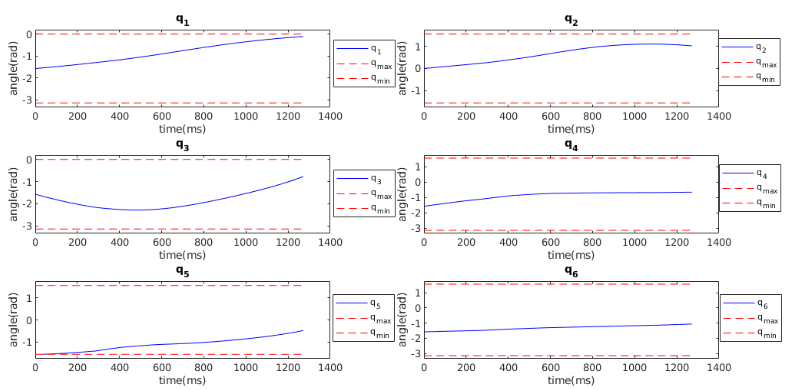

Figure 9: Configuration and the limits

The corresponding code for calculating the configurations using MMSE method is described as follows.

```
function [qdk,Xdk]=ComputeIKMlimits(Xdi,Xdf,V,Te,qi,qmin,qmax)
% follow the trajectory from Xdi to Xdf

% parameters
kmax=100;
```

```
epsx=0.001;

% calculate the transformation vector from Xdi to Xdf
dist=norm(Xdf-Xdi,2);
step=V*Te;
step_vector=step*(Xdf-Xdi)/dist;
total_T=dist/V; % required time for movement

% initialize the parameters for iteration
Xdk=Xdi;
Xd=Xdi;
qdk=qi;
qd=qi;
t=0;

while(t<total_T)
    Xd=Xd+step_vector;
    % compute configuration
    qd=ComputeIGMLimit(Xd,qd,kmax,epsx,qmin,qmax);

    % add sampled points
    Xdk=[Xdk,Xd];
    % add calculated configuration
    qdk=[qdk,qd];

    t=t+Te;
end
end
```

```
function q_star=ComputeIGMLimit(Xd,q0,kmax,epsx,qmin,qmax)

% parameters
re=0.1;
r=[0.5,0,0,0.20,0,0];
d=[0,0,0.7,0,0,0];
alpha=[0,pi/2,0,pi/2,-pi/2,pi/2];
t=[0,0,pi/2,0,0,0]'; % change of theta

for k=1:kmax
    theta=q0+t;
    % compute jacobian matrix
    J=ComputeJac(alpha,d,theta,r);
    JRT=J(1:3,:);

    M_0e=ComputeDGM(alpha, d,theta,r)*TransformMatElem(0,0,0,re);%M_06*M6e
    % translation vector
    X=M_0e(1:3,4);
```

```
    %calculate the difference
    delta_X=Xd-X;

    q_bar=(qmax-qmin)/2;

    H=2*((q0-q_bar)./(qmax-qmin).^2);
    %update q with MSE
    q_delta=pinv(JRT)*delta_X+(eye(size(JRT,2))-(pinv(JRT)*JRT))*(-0.001.*H);
    q0=q0+q_delta;
    % calculate error
    e=norm(delta_X,2);

    if e<epsx
        break;
    end
end
q_star=q0;

end
```

12 Q12:Jacobian matrix

This question aims to determine the velocity of the center of mass and the rotation speed of all the grid bodies in the frame. We need to calculate the Jacobian matrices according to the following functions.

$$^0V_{G_i} =^0 J_{v_{G_i}}(q)\dot{q}$$

$$^0w_{G_i} =^0 J_{v_{w_i}}(q)\dot{q}$$

12.1 Result

The code to calculate the Jacobian matrices is shown as follows.

```
%Q12
% parameters
x_G=[0,0.35,0,0,0,0];
y_G=[0,0,-0.1,0,0,0];
z_G=[-0.25,0,0,0,0,0];
alpha=[0,pi/2,0,pi/2,-pi/2,pi/2];
d=[0,0,0.7,0,0,0];
r=[0.5,0,0,0.20,0,0];
t=[0,0,pi/2,0,0,0];
q0=[-1.57,0,-1.47,-1.47,-1.47,-1.47];
theta=q0+t;
```

```
q_point = [0.5 1 -0.5 0.5 1 -0.5]';
% calculate the jacobien matrix
[OJv_Gi,OJ_wi]=ComputeJacGi(alpha,d,theta,r,x_G,y_G,z_G);
% calculate the speed V and W
OVGi = [];
OwGi = [];
for i = 1:6
OVGi = [OVGi OJv_Gi(:,:,i) * q_point];
OwGi = [OwGi OJ_wi(:,:,i) * q_point];
end
OVGi
OwGi
```

We test the function using $q_0 = [-1.57, 0, -1.47, -1.47, -1.47, -1.47]$;,$\dot{q} = [0.51-0.50.51-0.5]'$; and we get the results:

$$^0V_{G_i} \begin{bmatrix} 0 & 0.1750 & 0.3551 & 0.3601 & 0.3601 & 0.3601 \\ 0 & 0.0001 & -0.0495 & -0.0992 & -0.0992 & -0.0992 \\ 0 & 0.3500 & 0.7050 & 0.7101 & 0.7101 & 0.7101 \end{bmatrix}$$

$$^0w_{G_i} = \begin{bmatrix} 0 & -1.0000 & -0.5000 & -0.5000 & -0.5998 & -0.1048 \\ 0 & -0.0008 & -0.0004 & -0.0507 & -1.0407 & -1.0850 \\ 0.5000 & 0.5000 & 0.5000 & 0.0025 & 0.1027 & 0.1577 \end{bmatrix}$$

12.2 Code

The function ComputeJacGi is:

```
function [OJv_Gi,OJ_wi]=ComputeJacGi(alpha,d,theta,r,x_G,y_G,z_G)

    J = ComputeJac(alpha, d, theta, r);

    OJ_Oi = zeros(size(J,1), size(J,2), size(J,2));
    OJ_Gi = zeros(size(J,1), size(J,2), size(J,2));
    OJv_Gi = zeros(3, size(J,2), size(J,2));
    OJ_wi = zeros(3, size(J,2), size(J,2));
    g_elem = zeros(4,4);
    ROi = zeros(3,3,size(J,2));
    g_0i = eye(4);

    % compute the transoformation matrix from O to E
    g_06 = ComputeDGM(alpha, d, theta, r);
    rE = 0.1;
    g_6E = TransformMatElem(0,0,0,rE);
    g_0E = g_06 *g_6E;
    OOOE = g_0E(1:3, 4); %translation
    OEOO = -OOOE;
```

```

    000i = zeros(3,1,6);

    %Boucle sur les corps
    for i = 1:size(J,2)
        %Construction de toutes les matrices jacobiennes des corps Ci
        %exprimee au centre Oi du repere Ri attache a Ci
        OJ_Oi(:,1:i,i) = J(:,1:i);
        %Vecteur OiGi exprimé dans Ri
        iOiGi = [x_G(i) y_G(i) z_G(i)]';
        %Matrice de rotation R0i du repère R0 à Ri
        g_elem = TransformMatElem(alpha(i), d(i), theta(i), r(i));
        %Matrice de transformation élémentaire de passage de C(i-1)
        g_0i = g_0i*g_elem;
        R0i(:,:,i) = g_0i(1:3,1:3);
        %Vecteur OEGi dans R0 à  calculer: OEGi = OE0i + OiGi
        %Vecteur OiGi exprimé dans R0
        0OiGi = R0i(:,:,i)*iOiGi;
        %Vecteur OE0i exprimé dans R0: OE0i = OE00 + 000i
        000i(:,:,i) = g_0i(1:3,4);
        0E0i = 0E00 + 000i(:,:,i);
        0OEGi = 0E0i + 0OiGi;
        0PreproduitVect_OEGi = [0 -0OEGi(3) 0OEGi(2);...
                                0OEGi(3) 0 -0OEGi(1);...
                                -0OEGi(2) 0OEGi(1) 0];
        %Formule de Varignon
        0J_Gi(:,:,i) =  [eye(3) -0PreproduitVect_OEGi;...
                         zeros(3,3) eye(3)]*0J_Oi(:,:,i);

        %Resultats
        0Jv_Gi(:,:,i) = 0J_Gi(1:3,:,i);
        0J_wi(:,:,i) = 0J_Gi(4:6,:,i);
    end
end
```

13 Q13: the inertia matrix

This question aims to calculate the inertia matrix $A(q)$ of the robot. According to the slide p169, we can find that

$$A(q) = \sum_{i=1}^{N}(m_i^0 J_{v_{G_i}}^t(q)^0 J_{v_{G_i}}(q) +^0 J_{v_{w_i}}^t(q)^0 I_i^0 J_{wi}(q))$$

13.1 Result

We use $q_0 = [-pi/2, 0, -pi/2, -pi/2, -pi/2, -pi/2]'$; and the algorithm is shown in below.

```
% Q13
% slide p169
clear
q0=[-pi/2,0,-pi/2,-pi/2,-pi/2,-pi/2]';
ComputeMatInert(q0)
```

The result is:

$$A(q) = \begin{bmatrix} 6.4350 & 0.0000 & 0.0000 & -0.0700 & -0.0000 & 0.0000 \\ 0.0000 & 7.1650 & 0.9100 & 0.0000 & 0.0000 & 0.0100 \\ 0.0000 & 0.9100 & 1.0100 & 0.0000 & 0.0000 & 0.0100 \\ -0.0700 & 0.0000 & 0.0000 & 0.1190 & 0.0000 & 0.0000 \\ -0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0690 & 0.0000 \\ 0.0000 & 0.0100 & 0.0100 & 0.0000 & 0.0000 & 0.0590 \end{bmatrix}$$

13.2 Code

The definition of ComputeMatInert is shown as follows.

```
function A=ComputeMatInert(q)
% parameters
iI=zeros(3,3,6);
iI(:,:,1)=[0.80 0 0.05 ; 0 0.80 0 ; 0.05 0 0.10];
iI(:,:,2)=[0.10 0 0.10; 0 1.50 0 ; 0.10 0 1.50];
iI(:,:,3)=[0.05 0 0 ; 0 0.01 0 ; 0 0 0.05];
iI(:,:,4)=[0.50 0 0 ; 0 0.50 0 ; 0 0 0.05];
iI(:,:,5)=[0.01 0 0 ; 0 0.01 0 ; 0 0 0.01];
iI(:,:,6)=[0.01 0 0 ; 0 0.01 0 ; 0 0 0.01];

x_G=[0,0.35,0,0,0,0];
y_G=[0,0,-0.1,0,0,0];
z_G=[-0.25,0,0,0,0,0];

alpha=[0,pi/2,0,pi/2,-pi/2,pi/2];
d=[0,0,0.7,0,0,0];
r=[0.5,0,0,0.20,0,0];
t=[0,0,pi/2,0,0,0];
theta=q'+t;

Rred = [100 100 100 70 70 70];
Jm = 1e-5*[1 1 1 1 1 1];
m=[15,10,1,7,1,0.5];
```

```
A=zeros(6,6);
M_Oi=eye(4); % initialize transformation matrix

% calculate the Jacobien matrix
[OJv_Gi,OJ_wi]=ComputeJacGi(alpha,d,theta,r,x_G,y_G,z_G);

for i=1:length(q)
    OPreproduitVect_OiGi=[0 -z_G(i) y_G(i);z_G(i) 0 -x_G(i); -y_G(i) x_G(i) 0];
    iI_Gi=iI(:,:,i)+m(i)*OPreproduitVect_OiGi*OPreproduitVect_OiGi;

    M_i=TransformMatElem(alpha(i),d(i),theta(i),r(i));
    M_Oi=M_Oi*M_i;
    R_Oi=M_Oi(1:3,1:3); % rotation matrix

    OI_Gi=R_Oi*iI_Gi*R_Oi';
    % inertia matrix
    A=A+((m(i)*OJv_Gi(:,:,i)')*OJv_Gi(:,:,i) + (OJ_wi(:,:,i)')*OI_Gi*OJ_wi(:,:,i));
end
A=A+diag(Rred.^2.*Jm);
end
```

14 Q14: The bounds of inertia matrix

The question aims to calculate the lower and upper bound of the inertia matrix. According to the slide p178, we can find that

$$\mu_1 \leq A(q) \leq \mu_2$$

We pick 1000 samples from the range between q_{di} and q_{df} and calculate the maximum and minimum value of A based on these samples.

14.1 Result

We use the following algorithm and set the number of samples at 1000.

```
% Q14
% slide P178
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';

mu1=inf; % lower
mu2=0; % upper

% take samples
numSamples=1000;
```

```
dq=(qmax-qmin)/numSamples;
qCur=qmin;

for i=1:numSamples
    % compute the current q
    qCur=qCur+dq;
    A=ComputeMatInert(qCur);
    maxEig=max(eig(A));
    minEig=min(eig(A));
    % choose the limit value
    if maxEig>mu2
        mu2=maxEig;
    end
    if minEig<mu1
        mu1=minEig;
    end
end

mu1
mu2
```

Finally, we get the results:

$$\mu_1 = 0.0574$$

$$\mu_2 = 10.1985$$

.

15 Q15: The gravity torque

The question aims to obtain a vector of joint torques due to the gravity $G(q)$. According to the tutorial, we know that

$$G(q) = -({}^0J^t_{v_{G_1}} m_1 g + ... +{}^0 J^t_{v_{G_6}} m_6 g)$$

15.1 Result

We set $q_0 = [-pi/2, 0, -pi/2, -pi/2, -pi/2, -pi/2]$; and test the algorithm, which is shown as follows.

```
% Q15
clear
q0=[-pi/2,0,-pi/2,-pi/2,-pi/2,-pi/2]';
G=ComputeGravTorque(q0)
```

Finally, we can obtain the results:

$$G = \begin{bmatrix} 0 \\ 99.5715 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

15.2 Code

The function of ComputeGravTorque is defined as follows:

```
function G=ComputeGravTorque(q)
% parameters
m=[15,10,1,7,1,0.5];
g=[0,0,-9.81]';
x_G=[0,0.35,0,0,0,0];
y_G=[0,0,-0.1,0,0,0];
z_G=[-0.25,0,0,0,0,0];
alpha=[0,pi/2,0,pi/2,-pi/2,pi/2];
d=[0,0,0.7,0,0,0];
r=[0.5,0,0,0.20,0,0];
t=[0,0,pi/2,0,0,0]';
theta=ones(6,1); % simulink
theta=q+t;

% calculate the javobien matrix

[OJv_Gi,~]=ComputeJacGi(alpha,d,theta,r,x_G,y_G,z_G);

% calculate G
G=zeros(length(q),1);
for i=1:length(m)
    G=G-OJv_Gi(:,:,i)'.*m(i)*g;
end

end
```

16 Q16: The upper bound of gravity torque

This question aims to calculate the upper bound g_b of the joint torques G. From slide P180 and the tutorial, we know that:

$$\forall q \in [q_{min}, q_{max}], ||G(q)||_1 \leq g_b$$

16.1 Result

We pick 1000 samples from the range $[q_{min}, q_{max}]$, calculate corresponding $||G||_1$ and choose the maximum value as the uppder bound g_b.

```
% Q16
% slide P180
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';

gb=0; % upper

% take samples
numSamples=1000;
dq=(qmax-qmin)/numSamples;
qCur=qmin;

for i=1:numSamples
   % compute the current q
   qCur=qCur+dq;
   G=ComputeGravTorque(qCur);
   normG=norm(G,1);
   if normG>gb
       gb=normG;
   end

end
gb
```

Finally, we get the $gb = 117.3237$

17 Q17: Simulation of dynamic model

In the part, we build a block in the Simulink for simulation of dynamic model in the system. We use the provided function ComputeCCTorques and ComputeMatInert. Then we create the function ComputeFrictionTorque:

$$\tau_{f_i}(\dot{q}_i) = diag(\dot{q}_i) * F_v i$$

```
function Frott = ComputFrictionTorque(dq_dt)
    %parametres
    Fv = 10*ones(6,1);

    %Fonction donnée à l'interieure du TP
    Frott = diag(dq_dt)*Fv;
```

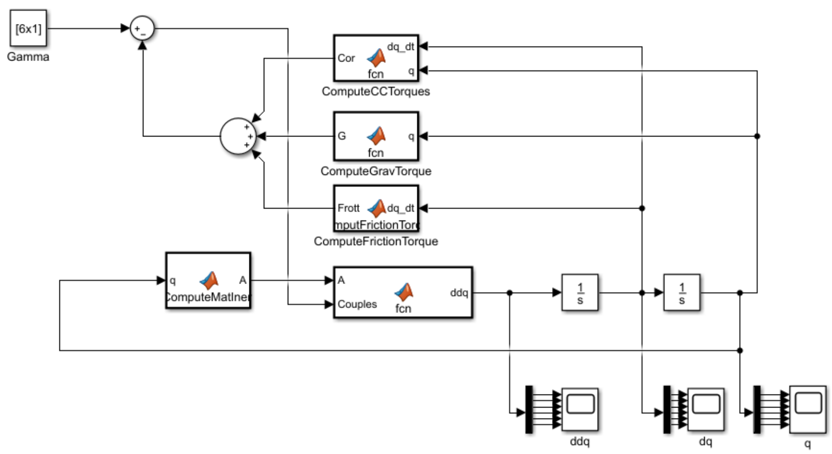

Figure 10: block design for the dynamic model

```

end
```

So, we can build a simulink block like the figure 10. We set the Gamma as a constant like [0.2,0.2,0.2,0.2,0.2,0.2]' and the result is like the figure 11, 12 and 13. There is the fluctuation in the beginning but it gets stable in the end.

18 Q18: the final minimum time in each joint

In the part, the minimal q and maximal q are given like in Q14. The time of each joint is :

$$t_{fi} = max(\frac{15|D_i|}{8k_v i}, \sqrt{\frac{10|D_i|}{\sqrt{3}k_{ai}}})$$

where $D = q_{max} - q_{min}$. As we only know k_{ai}, so the time is:

$$t_{fi} = \sqrt{\frac{10|D_i|}{\sqrt{3}k_{ai}}}$$

```
%set parameter
qmin = [-pi -pi/2 -pi -pi -pi/2 -pi]';
qmax = [0 pi/2 0 pi/2 pi/2 pi/2]';
qdi = [-1 0 -1 -1 -1 -1]';
qdf = [0 1 0 0 0 0]';
Te = 0.001;
```

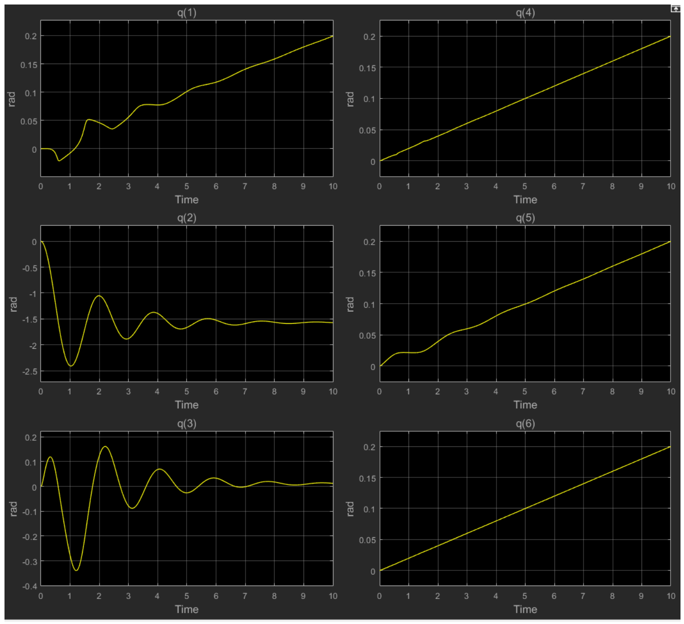

Figure 11: Simulation Result for the dynamic model: q

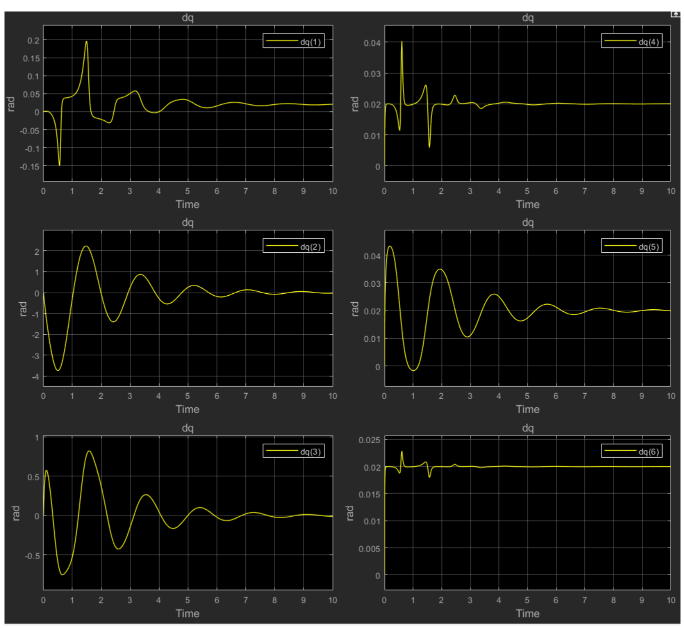

Figure 12: Simulation Result for the dynamic model: dq

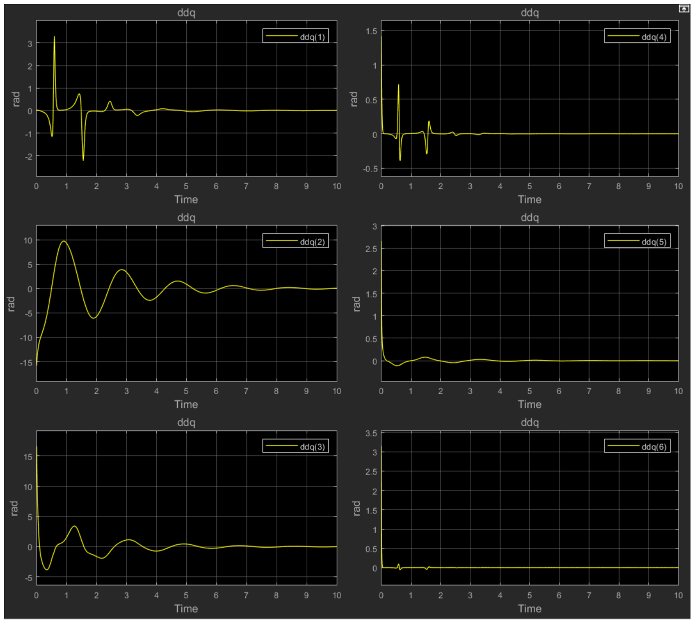

Figure 13: Simulation Result for the dynamic model: ddq

```
Rred = [100 100 100 70 70 70];
tau = 5;

%use the code of Q14
[number1,number2] = FindScalar(qmin,qmax);
%calculation of parameter
Distance = abs(qdf - qdi);
k = ((Rred * tau) / number1)';

%calculation of time
t = (sqrt((10/sqrt(3))*Distance./k))
t = max(t);
```

The result is :

$$t = [0.340.340.340.410.410.41]'$$

19 Q19: generate the desired joint trajectory point

We use the user-defined block to generate the desired joint trajectory point in the figure 14. The code is shown blew and the result is like the figure 15.

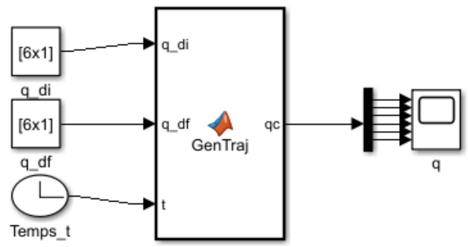

Figure 14: block design to generate the desired joint trajectory point

```
function qc = GenTraj(q_di, q_df, t)

    tf = 500; % ms
    D = (q_df - q_di);
    r = 10 * (t/tf)^3 - 15 * (t/tf)^4 + 6 * (t/tf)^5;
    qc = q_di + r*D;
end
```

20 Q20: Joint Control Law

In this part, we build the whole model of join control and calculate the error in the simulation. The whole model is shown in the figure 16. In the whole model, we use the subsystem that we create in Q17 and Q20. For the subsystem of position command, it is shown in the figure 17. At last, we calculate the error between qc and q in the figure 18. Although it is quite big in the beginning it goes down quickly and it is smaller than 0.05 after 50ms.

References

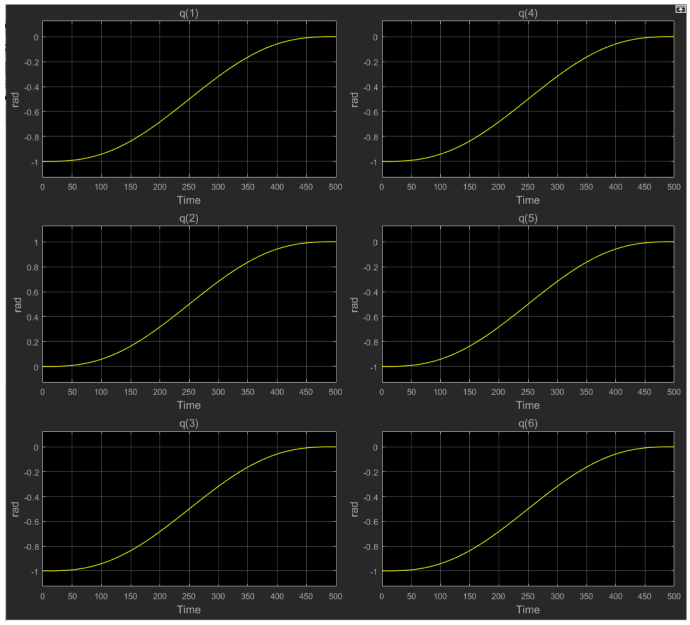

Figure 15: Simulation Result to generate the desired joint trajectory point

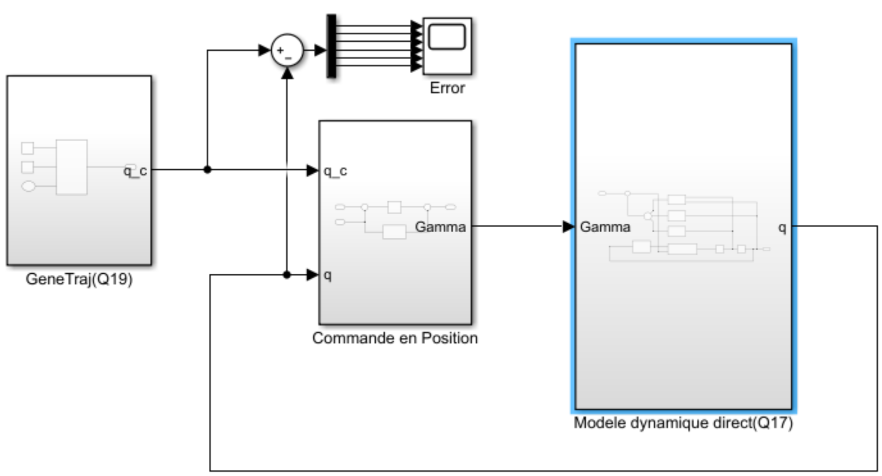

Figure 16: block design for the whole model

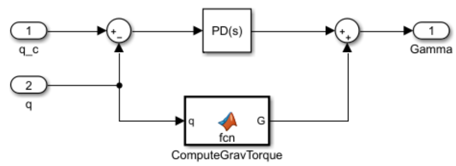

Figure 17: block design for position command

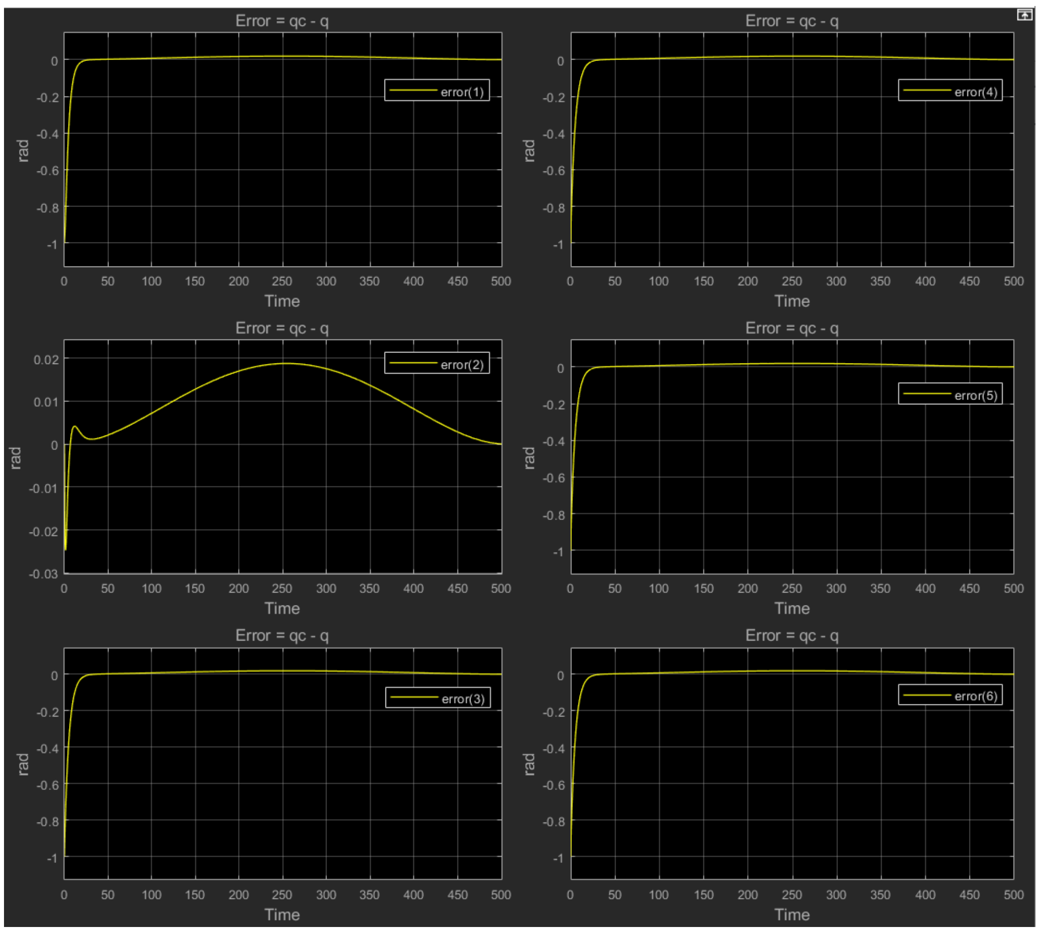

Figure 18: Result of the simulation