2023 Robotics Project2

Student ID：312512061　Name：劉郁昇

1. Project Request

PUMA 560 D-H parameters

Joint	d(m)	a(m)	α	θ
1	0	0	-90°	θ_1
2	0	0.432	0°	θ_2
3	0.149	-0.02	90°	θ_3
4	0.433	0	-90°	θ_4
5	0	0	90°	θ_5
6	0	0	0°	θ_6

Angle limit

$$-160° \le \theta_1 \le 160° \quad , \quad -125° \le \theta_2 \le 125°$$
$$-135° \le \theta_3 \le 135° \quad , \quad -140° \le \theta_4 \le 140°$$
$$-100° \le \theta_5 \le 100° \quad , \quad -260° \le \theta_6 \le 260°$$

Write a program to plan a path for (1) Joint move and (2) Cartesian move. The start, via, and end points for the path are

$$A = \begin{bmatrix} 0.64 & 0.77 & 0 & 5 \\ 0.77 & -0.64 & 0 & -55 \\ 0 & 0 & -1 & -60 \\ 0 & 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 0.87 & -0.1 & 0.48 & 50 \\ 0.29 & 0.9 & -0.34 & -40 \\ -0.4 & 0.43 & 0.81 & 40 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$
$$C = \begin{bmatrix} 0.41 & 0.29 & 0.87 & 60 \\ 0.69 & 0.71 & -0.09 & 15 \\ -0.6 & 0.64 & 0.49 & -30 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The length unit above is **cm**.

The time to move from point A to B is $t_{AB} = 0.5$ sec, and the time from point B to C is $t_{BC} = 0.5$ sec. The acceleration time for the transition portion is $t_{acc} = 0.2$ sec, and the sampling time is $t_s = 0.002$ sec.

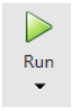
Result express

(1) Joint move：Curve chart of planning trajectory，Six axis angle ($\theta_1 \sim \theta_6$)、velocity、acceleration change

(2) Cartesian move：Curve chart of planning trajectory，end point (x,y,z)position、velocity、acceleration change

2. Platform

MATLAB is used as the development platform, mainly because MATLAB provides convenient 3D drawing functions (plot3). (One-click) Execute **main.m** to display the project results.



press “run” in **main.m**

3.Path planning illustration

The relationship between time and location is assumed to be as follows：

t=-0.5($=-t_{AB}$) → at position A

t=-0.2($=-t_{acc}$) → at position A’ (A’ is on line segment AB)

t=0 → At the midpoint of the transition from position A’ to C’

t=0.2($=t_{acc}$) → At position B’ (C’ is on line segment BC)

t=0.5($=t_{BC}$) → at position C

(1) Joint move

(a) Use the inverse kinematics of project1 to calculate the angles of each axis at three points A, B, and C.

q_A = [-100.4577 70.6108 48.3997 0 60.9896 29.2746]

q_B = [-52.1158 -1.4358 30.2060 -121.3834 -11.4781 -178.4572]

q_C = [0.0955 65.7969 14.3196 15.3377 -20.1730 30.0401]

(b) separate to three segment(I)、(II)、(III) Perform polynomial path planning for each axis angle (q:angle、$\dot{q}$:angular velocity、$\ddot{q}$:angular acceleration)

Set $T = 0.5 = t_{AB} = t_{BC}$、$t_{acc} = 0.2$

(I) Position from A to A’ (Linear portion)

$t_{AA'} = -T \sim -t_{acc}$

$h = \frac{-t_{AA'}}{T}$

$q = (q_B - q_A)h + q_A$

$\dot{q} = \frac{(q_B - q_A)}{T}$

$\ddot{q} = 0$

(II) Position from A’ to C’ (Transition portion)

$t_{A'C'} = -t_{acc} \sim t_{acc}$

$h = \frac{t_{A'C'} + t_{acc}}{2t_{acc}}$

$q'_A = (q_B - q_A)\left(\frac{-t_{acc}+T}{T}\right) + q_A$

$\Delta B = q'_A - q_B$

$$\Delta C = q_C - q_B$$

$$q = \left[\left(\Delta C \frac{t_{acc}}{T} + \Delta B\right)(2-h)h^2 - 2\Delta B\right]h + B + \Delta B$$

$$\dot{q} = \left[\left(\Delta C \frac{t_{acc}}{T} + \Delta B\right)(1.5-h)2h^2 - \Delta B\right]\frac{1}{t_{acc}}$$

$$\ddot{q} = \left(\Delta C \frac{t_{acc}}{T} + \Delta B\right)(1-h)\frac{3h}{{t_{acc}}^2}$$

(III) Position from C' to C (Linear portion)

$$t_{C'C} = t_{acc} \sim T$$

$$h = \frac{t_{C'C}}{T}$$

$$q = (q_C - q_B)h + q_B$$

$$\dot{q} = \frac{(q_C - q_B)}{T}$$

$$\ddot{q} = 0$$

(c) Based on the polynomial of the three-segment path planning, calculate the six-axis angle, angular velocity, and angular acceleration of the path from A to C (t=-T~T, ts=0.002 interval), and use the forward kinematics of project1 to calculate the [n o a p] matrix (The three-axis direction and position of the end point)

(2) Cartesian move

(a) In Cartesian planning, drive transformation (consisting of one translation and two rotations) is used to move the position and coordinate system of the points on the cassette coordinates.

(I)At t=0，The (initial) position and three-axis direction ([n o a p] matrix) are$POS_1 = POS_1 * D(0)$；At t=T0，The (final) position and three-axis direction ([n o a p] matrix) are $POS_2 = POS_1 * D(1)$；When t=t, the position and coordinate system are $POS_1 * D(r = t/T)$。

$$POS_1 = \begin{bmatrix} {}^{P1}n & {}^{P1}o & {}^{P1}a & {}^{P1}p \\ 0 & 0 & 0 & 1 \end{bmatrix}、POS_2 = \begin{bmatrix} {}^{P2}n & {}^{P2}o & {}^{P2}a & {}^{P2}p \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$POS_2 = POS_1 * D(1) \rightarrow D(1) = {POS_1}^{-1} * POS_2$$

$$\rightarrow D(1) = \begin{bmatrix} {}^{P1}n \cdot {}^{P2}n & {}^{P1}n \cdot {}^{P2}o & {}^{P1}n \cdot {}^{P2}a & {}^{P1}n \cdot ({}^{P2}p - {}^{P1}p) \\ {}^{P1}o \cdot {}^{P2}n & {}^{P1}o \cdot {}^{P2}o & {}^{P1}o \cdot {}^{P2}a & {}^{P1}o \cdot ({}^{P2}p - {}^{P1}p) \\ {}^{P1}a \cdot {}^{P2}n & {}^{P1}a \cdot {}^{P2}o & {}^{P1}a \cdot {}^{P2}a & {}^{P1}a \cdot ({}^{P2}p - {}^{P1}p) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(II) drive transformation The matrix D(r) consists of a translation T(r) and two rotations Ra(r) and Ro(r). $D(r) = T(r) * Ra(r) * Ro(r)$，$0 \le r = t/T \le 1$

$$\rightarrow T(r) = \begin{bmatrix} 1 & 0 & 0 & rx \\ 0 & 1 & 0 & ry \\ 0 & 0 & 1 & rz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\rightarrow Ra(r) = \begin{bmatrix} s\psi^2 V(r\theta) + c(r\theta) & -s\psi c\psi V(r\theta) & c\psi s(r\theta) & 0 \\ -s\psi c\psi V(r\theta) & c\psi^2 V(r\theta) + c(r\theta) & s\psi s(r\theta) & 0 \\ -c\psi s(r\theta) & -s\psi s(r\theta) & c(r\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\rightarrow Ra(r) = \begin{bmatrix} c(r\phi) & -s(r\phi) & 0 & 0 \\ s(r\phi) & c(r\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Simple express： $s\psi = \sin\psi$、$c\psi = \cos\psi$、$s(r\theta) = \sin(r\theta)$ 、$c(r\theta) = \cos(r\theta)$ 、$V(r\theta) = 1 - \cos(r\theta)$、$s\phi = \sin\phi$、$c\phi = \cos\phi$

$D(r) = T(r) * Ra(r) * Ro(r) = [D_n(r) \quad D_o(r) \quad D_a(r) \quad D_p(r)]$

$\rightarrow D_n(r) = D_o(r) \times D_a(r)$

$$\rightarrow D_o(r) = \begin{bmatrix} -s(r\phi)[s\psi^2 V(r\theta) + c(r\theta)] + c(r\phi)[-s\psi c\psi V(r\theta)] \\ -s(r\phi)[-s\psi c\psi V(r\theta)] + c(r\phi)[c\psi^2 V(r\theta) + c(r\theta)] \\ -s(r\phi)[-c\psi s(r\theta)] + c(r\phi)[-s\psi s(r\theta)] \\ 0 \end{bmatrix}$$

$$\rightarrow D_a(r) = \begin{bmatrix} c\psi s(r\theta) \\ s\psi s(r\theta) \\ c(r\theta) \\ 0 \end{bmatrix}$$

$$\rightarrow D_p(r) = \begin{bmatrix} rx \\ ry \\ rz \\ 1 \end{bmatrix}$$

(III) The parameters in D(r) are derived from (I) and (II) (x、y、z、ψ、θ、ϕ)

by $D_p(1)$

$\rightarrow x = {}^{P1}n \cdot ({}^{P2}p - {}^{P1}p)$

$\rightarrow y = {}^{P1}o \cdot ({}^{P2}p - {}^{P1}p)$

$\rightarrow z = {}^{P1}a \cdot ({}^{P2}p - {}^{P1}p)$

$$\text{by } D_a(1) = \begin{bmatrix} c\psi s(\theta) \\ s\psi s(\theta) \\ c(\theta) \\ 0 \end{bmatrix} = \begin{bmatrix} {}^{P1}n \cdot {}^{P2}a \\ {}^{P1}o \cdot {}^{P2}a \\ {}^{P1}a \cdot {}^{P2}a \\ 0 \end{bmatrix}$$

$$\rightarrow \psi = \tan^{-1}\left(\frac{{}^{P1}o \cdot {}^{P2}a}{{}^{P1}n \cdot {}^{P2}a}\right)$$

$$\text{by } D_a(1) \rightarrow \theta = \tan^{-1}\left(\frac{s\theta}{c\theta}\right) = \tan^{-1}\left(\frac{[(c\psi s\theta)^2 + (s\psi s\theta)^2]^{\frac{1}{2}}}{c\theta}\right)$$

$$\rightarrow \theta = \tan^{-1}\left(\frac{[({}^{P1}o\cdot{}^{P2}a)^2+({}^{P1}n\cdot{}^{P2}a)^2]^{\frac{1}{2}}}{{}^{P1}a\cdot{}^{P2}a}\right)$$

by $D_n(1)$、$D_o(1)$

$$\rightarrow \phi = \tan^{-1}\left(\frac{(-s\psi c\psi V\theta)({}^{P1}n\cdot{}^{P2}n)+(c\psi^2V\theta+c\theta)({}^{P1}o\cdot{}^{P2}n)+(-s\psi s\theta)({}^{P1}a\cdot{}^{P2}n)}{(-s\psi c\psi V\theta)({}^{P1}n\cdot{}^{P2}o)+(c\psi^2V\theta+c\theta)({}^{P1}o\cdot{}^{P2}o)+(-s\psi s\theta)({}^{P1}a\cdot{}^{P2}o)}\right)$$

(b) Divide the path into three sections (I), (II), and (III) for path planning

Set $T = 0.5 = t_{AB} = t_{BC}$、$t_{acc} = 0.2$

(I) Position from A to A' (Linear portion)

$t_{AA'} = -T \sim -t_{acc}$

$r = \frac{T+t_{AA'}}{T}$

from A(r=0)、B(r=1)deduced transformation matrix parameters$(x, y, z, \psi, \theta, \phi)$，Then perform polynomial programming on the parameters as follows：

$rP = P \cdot r$，$P = x$、 y、z、θ、ϕ

$T_{AA'} = A * D(\mathbf{rx}, \mathbf{ry}, \mathbf{rz}, \boldsymbol{\psi}, \mathbf{r\theta}, \mathbf{r\phi})$

(II) Position from A' to C' (Transition portion)

$t_{A'C'} = -t_{acc} \sim t_{acc}$

$r = \frac{t_{A'C'}+t_{acc}}{2t_{acc}}$

First calculate the transformation matrix parameters from B (r=0) to A (r=1)$(x_{\Delta B}, y_{\Delta B}, z_{\Delta B}, \psi_{\Delta B}, \theta_{\Delta B}, \phi_{\Delta B})$and transformation matrix parameters from B(r=0) to C(r=1)$(x_{\Delta C}, y_{\Delta C}, z_{\Delta C}, \psi_{\Delta C}, \theta_{\Delta C}, \phi_{\Delta C})$，Then perform polynomial programming on the parameters as follows：

If $|\psi_{\Delta C} - \psi_{\Delta B}| > \frac{\pi}{2}$，then $\psi_{\Delta B} = \psi_{\Delta B} + \pi$、$\theta_{\Delta B} = -\theta_{\Delta B}$

$$rP = \left[\left(P_{\Delta C}\frac{t_{acc}}{T} + P_{\Delta B}\right)(2-r)r^2 - 2P_{\Delta B}\right]r + P_{\Delta B}$$，$P = x$、 y、z、θ、ϕ

$r\psi = (\psi_{\Delta C} - \psi_{\Delta B})r + \psi_{\Delta B}$

$T_{A'C'} = B * D(\mathbf{rx}, \mathbf{ry}, \mathbf{rz}, \mathbf{r\psi}, \mathbf{r\theta}, \mathbf{r\phi})$

(II) Position from C' to C (Linear portion)

$t_{C'C} = t_{acc} \sim T$

$r = \frac{t_{C'C}}{T}$

from B(r=0)、C(r=1)deduced transformation matrix parameters$(x, y, z, \psi, \theta, \phi)$，Then perform polynomial programming on the parameters as follows：

$rP = P \cdot r$，$P = x$、 y、z、θ、ϕ

$T_{C'C} = B * D(\mathbf{rx}, \mathbf{ry}, \mathbf{rz}, \boldsymbol{\psi}, \mathbf{r\theta}, \mathbf{r\phi})$

(c) Obtain the corresponding planning based on the three-section path drive transformation to calculate the end point [n o a p] matrix($T_{AA'}$、$T_{A'C'}$、$T_{C'C}$)

4. Program illustration

(1) start program from 'main.m'. The [n o a p] matrix of the starting point A, the [n o a p] matrix of the transition point B, the [n o a p] matrix of the end point C and the sampling time will be set. After calling JointMotion and CartesianMotion to do path planning, pass the relevant calculation results to plot_joint_motion and plot_cartesian_motion to draw the change diagram of the end point angle, angular velocity, and angular acceleration, as well as the change diagram of the end point position, speed, and acceleration, as well as the corresponding 3D trajectory diagram.

(2) JointMotion.m : The program for axis coordinate path planning will first call "inverse" to calculate the angles of each axis corresponding to the starting point, transition point, and end point, and then plan the path in segments (linear, transitional, linear). The input is the [n o a p] matrix of the starting point A, the [n o a p] matrix of the transition point B, and the [n o a p] matrix of the end point C. The output is the angle, angular velocity, and angular acceleration of all six axes of the path from A to C.

(a) Polynomial path planning from location A to A' (Linear portion)

```
% path A to A'
count = 1;
for t = -0.5 : SamplingTime : -0.2
    h = (t+0.5)/0.5;
    q1(:,count) = (thetaB - thetaA) * h + thetaA; % angle
    dq1(:,count) = (thetaB - thetaA)/0.5;         % angular velocity
    d2q1(:,count) = zeros(6,1);                   % angular acceleration
    count = count + 1;
end
```

(b) Polynomial path planning from location A' to C' (Transition portion)

```
% path A' to C'
count = 1;
thetaA_p = (thetaB - thetaA) * ((-0.2+0.5)/0.5) + thetaA;
dB = thetaA_p - thetaB;
dC = thetaC - thetaB;
for t = (-0.2+SamplingTime) : SamplingTime : (0.2 - SamplingTime)

    h = (t+0.2)/0.4;

    q2(:,count) = ((dC*0.2/0.5+dB)*(2-h)*(h^2)-2*dB)*h+thetaB+dB; % angle
    dq2(:,count) = ((dC*0.2/0.5+dB)*(1.5-h)*2*(h^2)-dB)/0.2;      % angular velocity
    d2q2(:,count) = (dC*0.2/0.5+dB)*(1-h)*3*h/(0.2^2);            % angular acceleration
    count=count+1;
end
```

(c) Polynomial path planning from location C to C' (Linear portion)

```
% path C' to C
count = 1;
for t = 0.2 : SamplingTime : 0.5
    h = t/0.5;
    q3(:,count) = dC*h+thetaB;      % angle
    dq3(:,count) = dC/0.5;          % angular velocity
    d2q3(:,count) = zeros(6,1);     % angular acceleration
    count=count+1;
end
```

(3) "CartesianMotion.m" is a program for cartesian coordinate path planning. It will call "LinearDrive.m" and "TransitionDrive.m" according to segments (linear, transition, linear) for path planning. The input is the [n o a p] matrix of the starting point A, the [n o a p] matrix of the transition point B, and the [n o a p] matrix of the end point C. The output is the position, velocity, acceleration and approach direction vector of all end points of the path from A to C.

(a) Path planning from location A to A' (Linear portion)

```
for t = -0.5 : SamplingTime : -0.2  % path AA'
    h = (t+0.5)/0.5;
    LD1 = LinearDrive(A,B,h);   % drive matrix
    m_A_Ap(:,:,count) = A*LD1;  % [n o a p] matrix

    x1(:,count) = m_A_Ap(1,4,count);
    y1(:,count) = m_A_Ap(2,4,count);
    z1(:,count) = m_A_Ap(3,4,count);
    a1(:,count) = m_A_Ap(1:3,3,count);

    count = count+1;
end
```

(b) Polynomial path planning from location A' to C' (Transition portion)

```
count = 1;
for t = (-0.2+SamplingTime) : SamplingTime : (0.2-SamplingTime)   % path A'C'
    h = (t+0.2)/(2*(0.2));
    TD = TransitionDrive(Ap,B,C,h); % drive matrix
    m_Ap_Cp(:,:,count) = B*TD;     % [n o a p] matrix

    x2(:,count) = m_Ap_Cp(1,4,count);
    y2(:,count) = m_Ap_Cp(2,4,count);
    z2(:,count) = m_Ap_Cp(3,4,count);
    a2(:,count) = m_Ap_Cp(1:3,3,count);

    count = count+1;
end
```

(c) Path planning from location C to C' (Linear portion)

```
count=1;
for t = 0.2 : SamplingTime : 0.5  % path C'C
    h = t/(0.5);
    LD3 = LinearDrive(B,C,h);     % drive matrix
    m_Cp_C(:,:,count) = B*LD3;  % [n o a p] matrix

    x3(:,count) = m_Cp_C(1,4,count);
    y3(:,count) = m_Cp_C(2,4,count);
    z3(:,count) = m_Cp_C(3,4,count);
    a3(:,count) = m_Cp_C(1:3,3,count);

    count = count+1;
end
```

(4) "LinearDrive.m" calculates the drive transformation of the linear section. Its input is the [n o a p] matrix of the starting point, the [n o a p] matrix of the end point and the time ratio (r), and the output is the matrix D(r) of the drive transformation.

(a) First find the drive transformation matrix parameters$(x, y, z, \psi, \theta, \phi)$

```
x = dot( nA , (pB - pA) ); y = dot( oA , (pB - pA) ); z = dot( aA , (pB - pA) ); psi = atan2( dot(oA,aB) , dot(nA,aB) );
theta = atan2( sqrt( dot(oA,aB)^2 + dot(nA,aB)^2 ) , dot(aA,aB) );
Vtheta = 1-cos(theta);
sin_phi = -sin(psi)*cos(psi)*Vtheta*dot(nA,nB)+...
    ( (cos(psi)^2)*Vtheta+cos(theta) )*dot(oA,nB)-sin(psi)*sin(theta)*dot(aA,nB);
cos_phi = -sin(psi)*cos(psi)*Vtheta*dot(nA,oB)+...
    ( (cos(psi)^2)*Vtheta+cos(theta) )*dot(oA,oB)-sin(psi)*sin(theta)*dot(aA,oB);
phi = atan2(sin_phi,cos_phi);
```

(b) Then perform polynomial programming on the parameters

```
%固定PSI
rx=x*r;   ry=y*r;   rz=z*r;   rtheta=theta*r;   rphi=phi*r;
```

(c) Bring in new parameters and calculate the drive transformation matrix

```
S_psi=sin(psi);  C_psi=cos(psi);  S_rtheta=sin(rtheta);  C_rtheta=cos(rtheta);
V_rtheta=1-C_rtheta;  S_rphi=sin(rphi);  C_rphi=cos(rphi);

% compute D_r (D_r = T_r*Ra_r*Ro_r)
T_r = [1 0 0 rx;    0 1 0 ry;      0 0 1 rz;      0 0 0 1];
Ra_r = [(S_psi^2)*V_rtheta+C_rtheta,       -S_psi*C_psi*V_rtheta, C_psi*S_rtheta, 0;
             -S_psi*C_psi*V_rtheta, (C_psi^2)*V_rtheta+C_rtheta, S_psi*S_rtheta, 0;
                  -C_psi*S_rtheta,             -S_psi*S_rtheta,       C_rtheta, 0;
                                0,                           0,               0, 1];
Ro_r = [C_rphi, -S_rphi, 0, 0;  S_rphi,  C_rphi, 0, 0;    0,   0, 1, 0;    0,   0, 0, 1];
D_r = T_r*Ra_r*Ro_r;
```

(5) “TransitionDrive.m” calculates the drive transformation of the transition section. Its input is the [n o a p] matrix of the starting point, the [n o a p] matrix of the transition target point, the [n o a p] matrix of the end point and the time ratio (r). Its output is the drive transformation. The matrix D(r).

(a)First compute drive transformation parameters$(x_{\Delta B}, y_{\Delta B}, z_{\Delta B}, \psi_{\Delta B}, \theta_{\Delta B}, \phi_{\Delta B})$ of B (r=0) to A(r=1) and drive transformation parameters$(x_{\Delta C}, y_{\Delta C}, z_{\Delta C}, \psi_{\Delta C}, \theta_{\Delta C}, \phi_{\Delta C})$ B(r=0) to C(r=1)

```
x1 = dot( nB , (pA - pB) );  y1 = dot( oB , (pA - pB) );
z1 = dot( aB , (pA - pB) );  psi1 = atan2( dot(oB,aA) , dot(nB,aA) );
theta1 = atan2( sqrt( dot(oB,aA)^2 + dot(nB,aA)^2 ) , dot(aB,aA) );
Vtheta1 = 1-cos(theta1);
sin_phi1 = -sin(psi1)*cos(psi1)*Vtheta1*dot(nB,nA)+...
    ( (cos(psi1)^2)*Vtheta1+cos(theta1) )*dot(oB,nA)-sin(psi1)*sin(theta1)*dot(aB,nA);
cos_phi1 = -sin(psi1)*cos(psi1)*Vtheta1*dot(nB,oA)+...
    ( (cos(psi1)^2)*Vtheta1+cos(theta1) )*dot(oB,oA)-sin(psi1)*sin(theta1)*dot(aB,oA);
phi1 = atan2( sin_phi1 , cos_phi1 );
% input parameter
x2 = dot( nB , (pC - pB) );  y2 = dot( oB , (pC - pB) );  z2 = dot( aB , (pC - pB) );
psi2 = atan2( dot(oB,aC) , dot(nB,aC) );
theta2 = atan2( sqrt( dot(oB,aC)^2 + dot(nB,aC)^2 ) , dot(aB,aC) );
Vtheta2 = 1-cos(theta2);
sin_phi2 = -sin(psi2)*cos(psi2)*Vtheta2*dot(nB,nC)+...
    ( (cos(psi2)^2)*Vtheta2+cos(theta2) )*dot(oB,nC)-sin(psi2)*sin(theta2)*dot(aB,nC);
cos_phi2 = -sin(psi2)*cos(psi2)*Vtheta2*dot(nB,oC)+...
    ( (cos(psi2)^2)*Vtheta2+cos(theta2) )*dot(oB,oC)-sin(psi2)*sin(theta2)*dot(aB,oC);
phi2 = atan2( sin_phi2 , cos_phi2 );
```

(b) Then modify the parameters and perform polynomial programming

```
if abs(psi2-psi1)>pi/2   % of psi,phi)-> let |psi2-psi1| < pi/2
    psi1=psi1+pi;
    theta1=-theta1;
end
rx=((x2*0.2/0.5+x1)*(2-r)*(r^2)-2*x1)*r+x1;   ry=((y2*0.2/0.5+y1)*(2-r)*(r^2)-2*y1)*r+y1;
rz=((z2*0.2/0.5+z1)*(2-r)*(r^2)-2*z1)*r+z1;
rpsi=(psi2-psi1)*r+psi1;
rtheta=((theta2*0.2/0.5+theta1)*(2-r)*(r^2)-2*theta1)*r+theta1;
rphi=((phi2*0.2/0.5+phi1)*(2-r)*(r^2)-2*phi1)*r+phi1;
```

(c) Bring in new parameters and calculate the drive transformation matrix

```
S_rpsi=sin(rpsi);  C_rpsi=cos(rpsi);  S_rtheta=sin(rtheta);  C_rtheta=cos(rtheta);
V_rtheta=1-C_rtheta;  S_rphi=sin(rphi);  C_rphi=cos(rphi);
T_r = [1 0 0 rx;        0 1 0 ry;        0 0 1 rz;      0 0 0 1];% compute D_r (D_r = T_r*Ra_r*Ro_r)
Ra_r = [(S_rpsi^2)*V_rtheta+C_rtheta,      -S_rpsi*C_rpsi*V_rtheta, C_rpsi*S_rtheta, 0;
            -S_rpsi*C_rpsi*V_rtheta, (C_rpsi^2)*V_rtheta+C_rtheta, S_rpsi*S_rtheta, 0;
                  -C_rpsi*S_rtheta,             -S_rpsi*S_rtheta,        C_rtheta, 0;
                                 0,                            0,              0, 1];
Ro_r = [C_rphi, -S_rphi, 0, 0;   S_rphi,  C_rphi, 0, 0;    0,       0, 1, 0;     0,   0, 0, 1];
D_r = T_r*Ra_r*Ro_r;
```

(6) “forward.m” is a program for calculating forward kinematics. It calculates the [n o a p] matrix of the end point based on the input six-axis angles. Its input is the six-axis angle, and its output is the (x, y, z) position of the end point, (ψ, θ, ϕ) and approach vector (ax, ay, az)

(7) “inverse.m” is a program for calculating inverse kinematics. It calculates the six-axis angle based on the [n o a p] matrix of the input end point. Its input is the [n o a p] matrix of the end point, and the output is the six-axis angle (θ1~θ6)。

(8)”plot_joint_motion.m” is the result of drawing axis planning, which in order is the six-axis angle, angular velocity, angular acceleration change diagram of the end point, 3D trajectory diagram

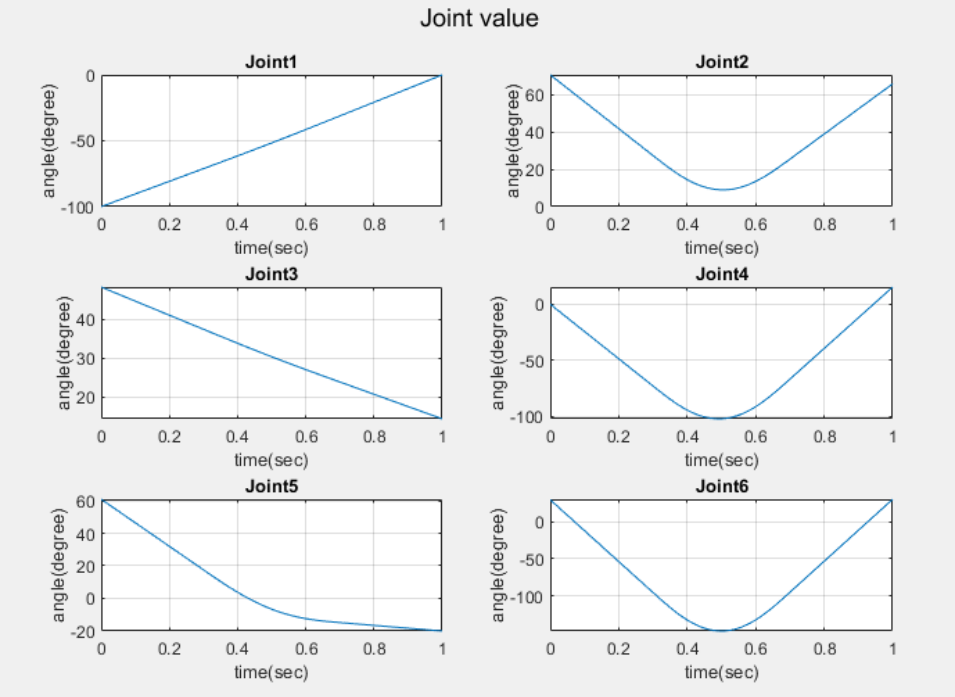
(9) “plot_cartesian_motion.m” is the result of drawing the cassette planning, which is the change diagram of the position, velocity and acceleration of the end point, the 3D trajectory diagram

5.Results

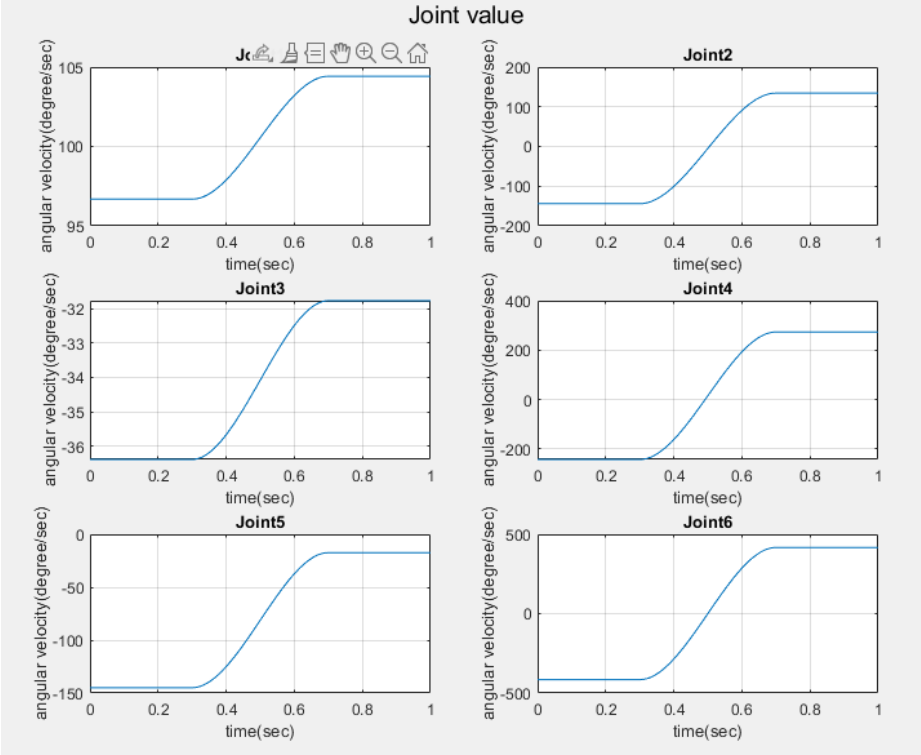
(1) Joint Motion：

(a)

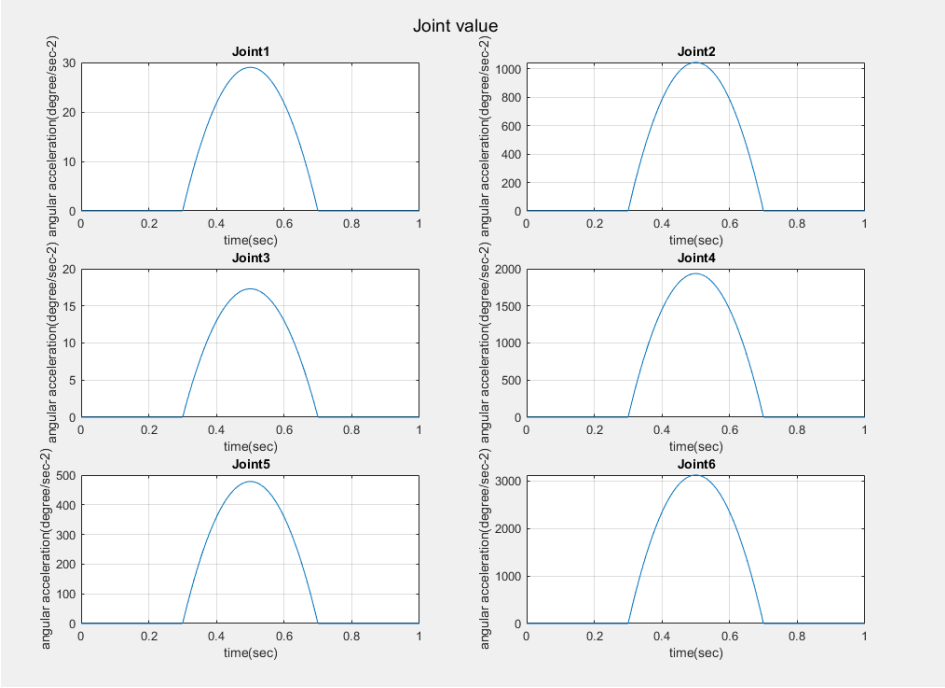
- The angle changes of the end point corresponding to the six axes



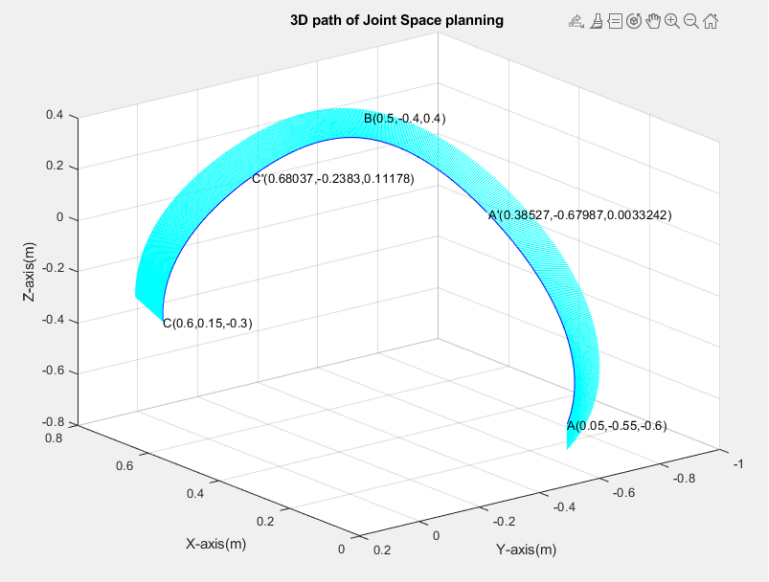
- The angular velocity changes of the end point corresponding to the six axes



- The angular acceleration changes of the end point corresponding to the six axes



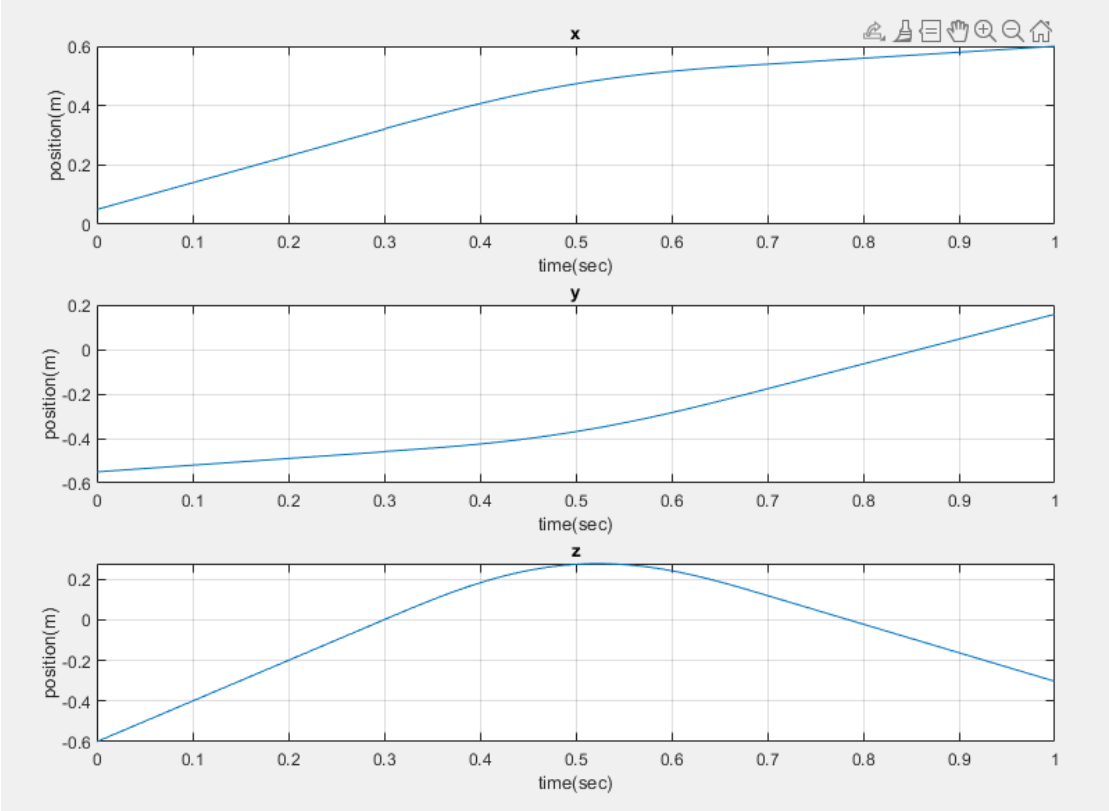
(b) 3D trajectory map



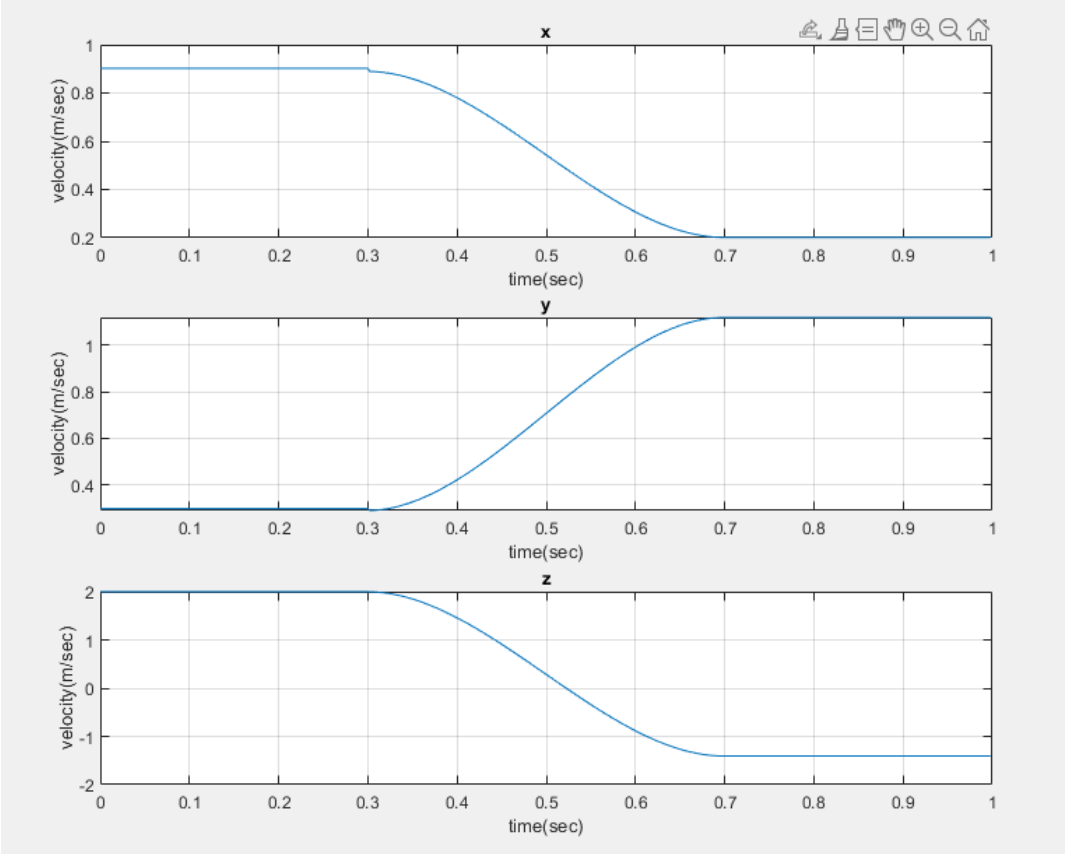
(2) Cartesian Motion：

(a)

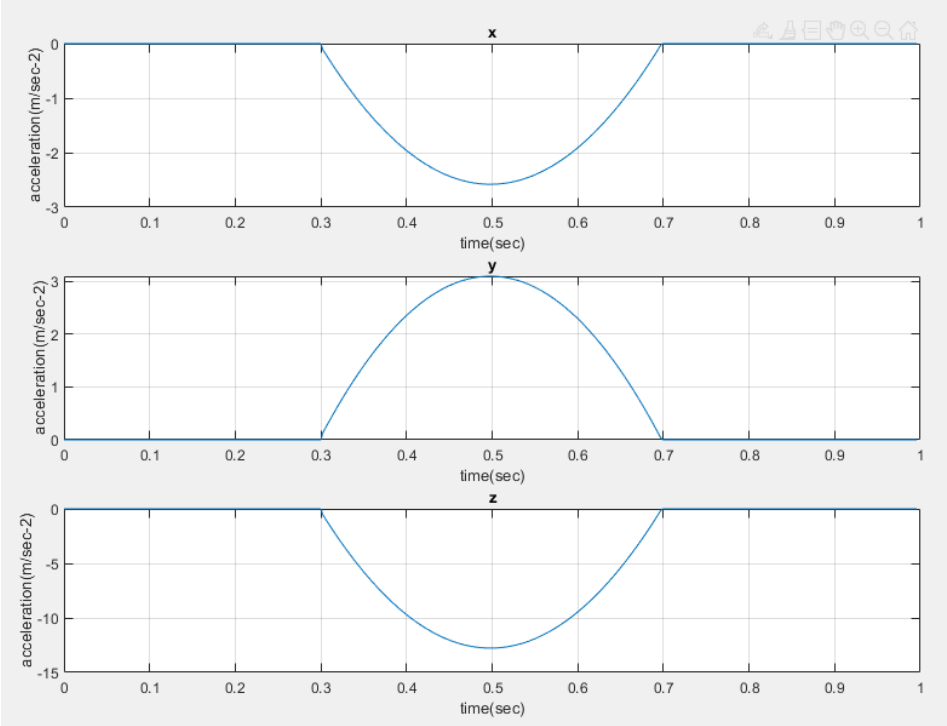
- The angle changes of the end point corresponding to the six axes



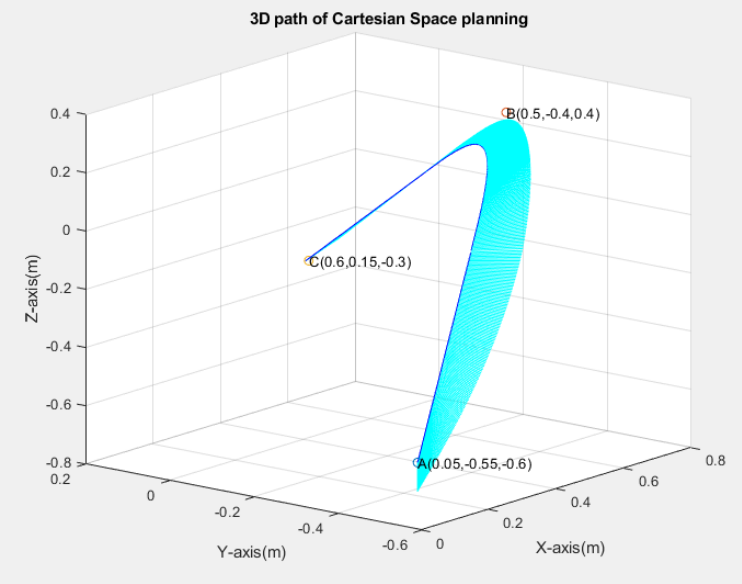
- The angular velocity changes of the end point corresponding to the six axes



- The angular acceleration changes of the end point corresponding to the six axes



(b) 3D trajectory map



6. Advantages and disadvantages of two types of trajectory planning How to solve it when meet singularity?

(1) **Joint Move (Joint Space Planning)**：

Advantages:

1. Direct Control: Offers direct control over each joint, which can be simpler for certain tasks, especially those involving joint limits.
2. Computational Efficiency: Often computationally less complex since it deals directly with the robot's actuators.
3. Kinematic Simplicity: Easier to implement for robots with complex kinematics where Cartesian planning might be more challenging.

Disadvantages:

1. Non-Intuitive Path: The path of the end-effector in workspace might not be intuitive or easy to control.
2. Obstacle Avoidance: More challenging to implement effective obstacle avoidance in the workspace.
3. End-Effector Precision: Achieving precise end-effector positioning and orientation can be more difficult.

Singularity Issues:

- Joint space planning is less prone to kinematic singularities compared to Cartesian planning.
-

(2) Cartesian Move (Cartesian Space Planning)

Advantages:

1. Intuitive End-Effector Control: More intuitive for controlling the end-effector's path in the workspace.
2. Task-Oriented: Easier to plan paths for tasks naturally described in Cartesian coordinates, like straight lines or specific geometric shapes.
3. Obstacle Avoidance: Simplifies implementing collision and obstacle avoidance strategies.

Disadvantages:

1. Inverse Kinematics Complexity: Requires solving the inverse kinematics, which can be complex and may not have a unique solution.
2. Computational Intensity: Generally more computationally intensive than joint space planning.
3. Precision and Stability: Achieving high precision and stability can be challenging, especially near singular configurations.

Singularity Issues:

- Cartesian planning is more susceptible to kinematic singularities. These occur when the Jacobian matrix (relating joint velocities to end-effector velocities) becomes singular, leading to issues like loss of control or infinite joint speeds.
- Common examples include "wrist singularities" in robotic arms and "gimbal lock" in orientation control.
- In these singular configurations, small changes in end-effector position can require large or undefined joint movements.