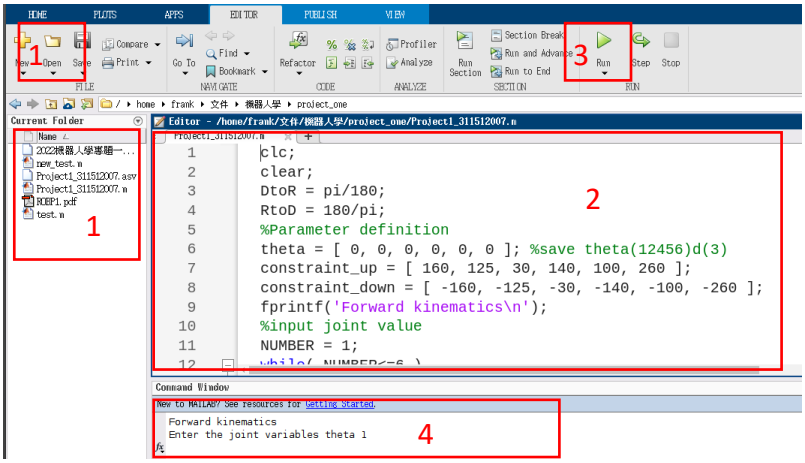
Project 2

311512007 蔡馥宇

一. 介面說明

a. Matlab

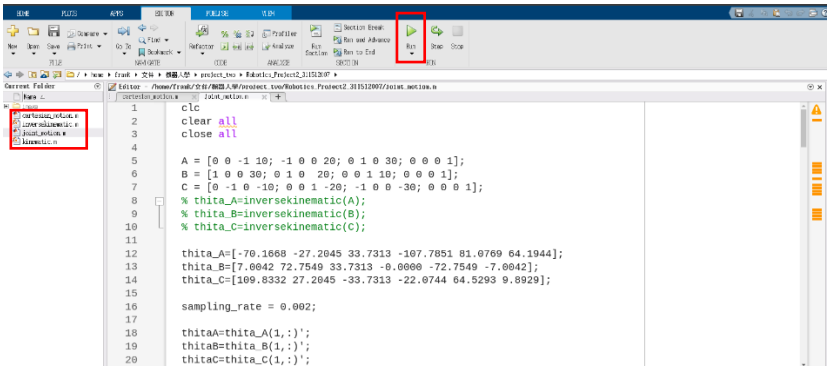


1. 將程式碼檔案(.m)打開
2. 程式碼的編輯視窗
3. 執行程式碼
4. Matlab 終端機可以進行操作

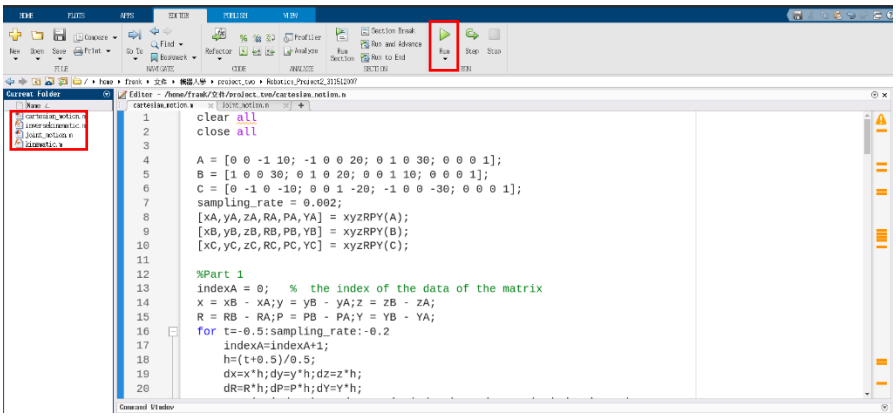
b. 如何操作我的程式

這次作業的操作方式，打開資料夾後，資料夾內會有 4 份.m 檔，並且可以看到以下畫面，直接按 run 就可以看到解果

1. cartesian_motio.m



2. joint_motion.m



```
clear all
close all

A = [0 0 -1 10; -1 0 0 20; 0 1 0 30; 0 0 0 1];
B = [1 0 0 30; 0 1 0 20; 0 0 1 10; 0 0 0 1];
C = [0 -1 0 -10; 0 0 1 -20; -1 0 0 -30; 0 0 0 1];
sampling_rate = 0.002;
[xA,yA,zA,RA,PA,YA] = xyzRPY(A);
[xB,yB,zB,RB,PB,YB] = xyzRPY(B);
[xC,yC,zC,RC,PC,YC] = xyzRPY(C);

%Part 1
indexA = 0;   %  the index of the data of the matrix
x = xB - xA;y = yB - yA;z = zB - zA;
R = RB - RA;P = PB - PA;Y = YB - YA;
for t=-0.5:sampling_rate:-0.2
    indexA=indexA+1;
    h=(t+0.5)/0.5;
    dx=x*h;dy=y*h;dz=z*h;
    dR=R*h;dP=P*h;dY=Y*h;
```

二.　程式架構說明

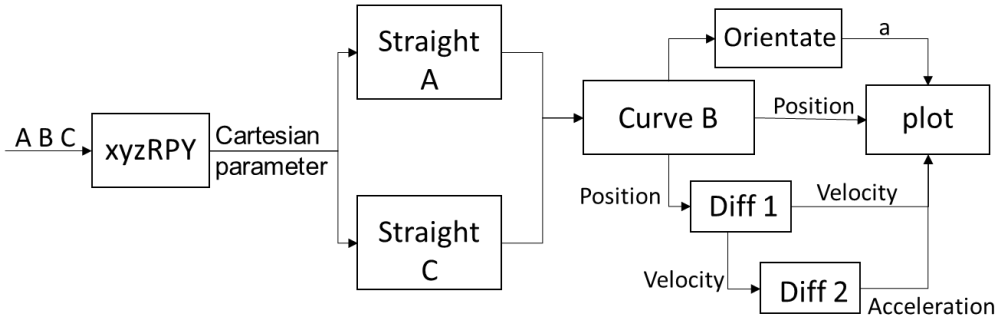
這次作業我的主要內容分是 cartesian_motio.m 和 joint_motion.m，分別代表在卡式座標下與軸座標下的路徑規劃，除此之外因為在規劃或是畫圖的時候會需要再順向逆向運動學之間做轉換，因此還有兩份作為函式使用的檔案 kinematic.m 與 inversekinematic.m 在需要的時候會呼叫它們來得到卡式與軸座標的資訊，但因為兩次 project 有單位的不同，我會把 kinematic.m 與 inversekinematic.m 中出現 d2=6.375 的部分都乘以 2.54，那以下分別來介紹卡式與軸標下的路徑規劃。

1. Cartesian move:

卡式座標路徑規劃的想法是我會將整個路徑分成三段，分別是直線、曲線、直線，那我會先用 A-B 找出第一段直線的終點，用 B-C 找出第三段直線的起點，而這兩個點分別會帶表第二段曲線的起點與終點，那再套用數學公式，將點求出來。

在講解程式碼之前，以下是 Cartesian move 的程式 flow chart，其中 A、B、C 代表的是題目所給的三個座標資訊，利用 xyzRPY，找到其 x、y、z、roll、pitch、yaw，這些資訊統稱為 Cartesian parameter，那這些參數會做路徑規劃，規劃方式就是用推導出來的數學式來得到每個時間點的座標，那這些座標會用 list 的形式存下來，最後再畫出來。

當我求出 x、y、z、roll、pitch、yaw 之後會會將後三項放到 Orientate 的函式裡面，最後可以得到每個時間點的 z 軸方向也就是 a，其他 list(x,y,z)分別去經過 Diff1 和 Diff2 的方塊可以得到速度與加速度資訊。



以下會針對不同模組來講解程式碼:

xyzRPY:

```
function [x,y,z,phi,theta,psi] = xyzRPY(T6)
    RtoD = 180/pi;
    nx = T6(1,1); ny = T6(2,1); nz = T6(3,1);
    ox = T6(1,2); oy = T6(2,2); oz = T6(3,2);
    ax = T6(1,3); ay = T6(2,3); az = T6(3,3);
    px = T6(1,4); py = T6(2,4); pz = T6(3,4);
    x = px; y = py; z = pz;
    phi = atan2(ay, ax) * RtoD;
    theta = atan2(sqrt(ax^2 + ay^2), az) * RtoD;
    psi = atan2(oz, -nz) * RtoD;
    % p = [ x, y, z, phi,theta, psi];
end
```

單純透過講義公式將座標資訊從矩陣的形式轉換為 x、y、z、roll、pitch、yaw。

StraightA & StraightC & Orientate:

```
%Straight A
indexA = 0;   %  the index of the data of the matrix
x = xB - xA;y = yB - yA;z = zB - zA;
R = RB - RA;P = PB - PA;Y = YB - YA;
for t=-0.5:sampling_rate:-0.2
    indexA=indexA+1;
    h=(t+0.5)/0.5;
    dx=x*h;dy=y*h;dz=z*h;
    dR=R*h;dP=P*h;dY=Y*h;
    xA_B(:,indexA)=xA+dx;yA_B(:,indexA)=yA+dy;zA_B(:,indexA)=zA+dz;
    R_A(:,indexA) = RA+dR;P_A(:,indexA) =PA+dP;Y_A(:,indexA) =YA+dY;
end
for i = 1:indexA
    ori_A(:,i) = Orientate(R_A(:,i),P_A(:,i),Y_A(:,i));
end
```

我在一開始將座標 A 與座標 B 所有的參數相減，以此可以得到所有 x、y、z、roll、pitch、yaw 的差，並將其帶到直線方程式裡面，再將所有資料用對應的 list 存下來，其中最後一部分的 Orientate 是利用每個時間點的 roll、pitch、yaw 來得到每個時間點的 z 方向。

```
function ori = Orientate(R,P,Y)
    R = R*pi/180;
    P = P*pi/180;
    Y = Y*pi/180;
    ori = [cos(R)*sin(P) sin(R)*sin(P) cos(P)];
end
```

Orientate 的內容也是以講義中轉換的公式來撰寫。

```
%Straight C
indexC = 0;   %  the index of the data of the matrix
x = xC - xB;y = yC - yB;z = zC - zB;
R = RC - RB;P = PC - PB;Y = YC - YB;
for t=0.2:sampling_rate:0.5
    indexC=indexC+1;
    h=(t)/0.5;
    dx=x*h;dy=y*h;dz=z*h;
    dR=R*h;dP=P*h;dY=Y*h;
    x_C(:,indexC)=xB+dx;y_C(:,indexC)=yB+dy;z_C(:,indexC)=zB+dz;
    R_C(:,indexC) =RB+dR;P_C(:,indexC) =PB+dP;Y_C(:,indexC) =YB+dY;
end
for i = 1:indexC
    ori_C(:,i) = Orientate(R_C(:,i),P_C(:,i),Y_C(:,i));
end
```

那在 StraightC 與 StraightA 的編寫方式其實大同小異，這兩個路徑規劃的方式最大重點不同在於時間點 A 部分是沒有走完後 0.2 秒，而 C 部分是沒有走前 0.2 秒，因為這些部分是 transition portion，會在 CurveB 去做處理。

Curve B:

```
%Curve B
%The end point of Straight A is the start point of curve B
xA = xA_B(:,indexA)-xB;yA = yA_B(:,indexA)-yB;zA = zA_B(:,indexA)-zB;
RA = R_A(:,indexA)-RB;PA = P_A(:,indexA)-RB;YA = Y_A(:,indexA)-RB;
%The atart point of Straight C is the end point of curve B
xC = x_C(:,1)-xB;yC = y_C(:,1)-yB;zC = z_C(:,1)-zB;
RC = R_C(:,1)-RB;PC = P_C(:,1)-PB;YC = Y_C(:,1)-YB;
indexB=0;
for t=(-0.2+sampling_rate):sampling_rate:(0.2-sampling_rate)
    indexB=indexB+1;
    h=(t+0.2)/(2*0.2);
    x_B(:,indexB)=xB + ((xC+xA)*(2-h)*h^2-2*xA)*h+xA;
    y_B(:,indexB)=yB + ((yC+yA)*(2-h)*h^2-2*yA)*h+yA;
    z_B(:,indexB)=zB + ((zC+zA)*(2-h)*h^2-2*zA)*h+zA;
    R_B(:,indexB)=RB + ((RC+RA)*(2-h)*h^2-2*RA)*h+RA;
    P_B(:,indexB)=PB + ((PC+PA)*(2-h)*h^2-2*PA)*h+PA;
    Y_B(:,indexB)=YB + ((YC+YA)*(2-h)*h^2-2*YA)*h+YA;
end
for i = 1:indexB
    ori_B(:,i) = Orientate(R_B(:,i),P_B(:,i),Y_B(:,i));
end
```

如圖中註解，Curve B 的起點是 StraightA 的終點，CurveB 的終點是 StraightC 的起點，然後模式也是跟上面兩個 part 相同，只要帶入曲線方程即可，並且在最後放入朝向的函式中來得到 z 的方向，時間點在-0.2 和 0.2 會和前一部份和後一部分重疊，因此頭尾各減少一個時點。

Diff 1 & Diff 2:

```
% XYZ的速度變化情形
%Diff 1
dt=t(2:501);
dX=diff(X)/sampling_rate;
dY=diff(Y)/sampling_rate;
dZ=diff(Z)/sampling_rate;
```

```
% XYZ的加速度變化情形
%Diff 2
dt2=t(3:501);
dX2=diff(dX)/sampling_rate;
dY2=diff(dY)/sampling_rate;
dZ2=diff(dZ)/sampling_rate;
```

這次作業裡我沒有使用在講義裡求速度與加速度的公式，而是直接對 list 用 matlab 函數 diff()

說明

Y = diff(X)計算沿大小不等於1 的第一個數組維度的 X 相鄰元素之間的差分：

- 如果 X 是長度為 m 的向量，則 Y = diff(X) 返回長度為 m-1 的向量。Y的元素是 X 相鄰元素之間的差分。

  Y = [X(2)-X(1) X(3)-X(2) ... X(m)-X(m-1)]

根據 matlab 官網可以知道，diff 就是前面減調後面的差，在除以 sampling_rate 可知其趨近於微分，那速度與加速度就是用這個方式求得。

Plot:

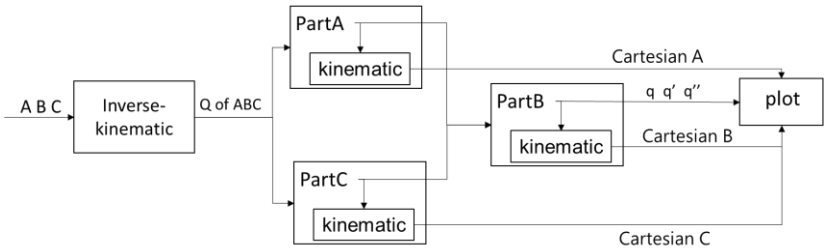
```
% 3D路徑圖
figure(1);
x_all = [xA_B x_B x_C]; y_all = [yA_B y_B y_C]; z_all = [zA_B z_B z_C];
ori_all = [ori_A ori_B ori_C];
quiver3(x_all,y_all,z_all,ori_all(1,:),ori_all(2,:),ori_all(3,:));
xlabel('x(cm)');ylabel('y(cm)');zlabel('z(cm)');
text(10,20,30,'A(10,20,30)');
text(30,20,10,'B(30,20,10)');
text(-10,-20,-30,'C(-10,-20,-30)');
title('Cartesion Motion')
```

針對軌跡圖來說明 plot 的部分，在一開始先將每個不同 part 的 x、y、z 合併，而這些點就是軌跡上的每一個點，為了在圖中表示在不同時間點其朝向，我會再把前面求出的不同時間點的 z 全部合併，最後利用 quiver3()把圖片做出來。

2. Joint move:

Joint space 的規劃方始其實跟卡式雷同，都是用推導出來的直線方程和曲線方程來規劃，並且將 PartA 的終點作為 PartB 的起點，PartC 的起點作為 PartB 的終點。

在一開始會先用 inversekinematic 來找到在 A、B、C 座標下的 6 軸角度 Q of ABC，並且送到 PartABC 去做路徑規劃，但這邊我們為了要把最後的卡式圖給畫出來，因次每次在執行 PartABC 的路徑規劃時會用 kinematic 來存下 x、y、z、a，之後再畫出來。



Inversekinematic:

```
A = [0 0 -1 10; -1 0 0 20; 0 1 0 30; 0 0 0 1];
B = [1 0 0 30; 0 1 0  20; 0 0 1 10; 0 0 0 1];
C = [0 -1 0 -10; 0 0 1 -20; -1 0 0 -30; 0 0 0 1];

% thita_A=inversekinematic(A);
% thita_B=inversekinematic(B);
% thita_C=inversekinematic(C);

thita_A=[-70.1668 -27.2045 33.7313 -107.7851 81.0769 64.1944];
thita_B=[7.0042 72.7549 33.7313 -0.0000 -72.7549 -7.0042];
thita_C=[109.8332 27.2045 -33.7313 -22.0744 64.5293 9.8929];
thitaA=thita_A(1,:)';thitaB=thita_B(1,:)';thitaC=thita_C(1,:)';
```

逆向運動學會有八組解，我直接利用其解出的八組解來選出 thita_ABC 並且做 transform 以便後面處理。

PartA & PartC & kinematic:

```
%~~~~~~~~~~~~~~~~~~(partA)
s1=0;
for t=-0.5:sampling_rate:-0.2
    s1=s1+1;
    h=(t+0.5)/0.5;
    dthitaA(:,s1)=thitaA+(thitaB-thitaA)*h;
    domegaA(:,s1)=(thitaB-thitaA)/0.5;
    dalphaA(:,s1)=[0;0;0;0;0;0];
    p1=kinematic(dthitaA(:,s1)');
    x1(s1)=p1(1,4);y1(s1)=p1(2,4);z1(s1)=p1(3,4);
    ori_1(:,s1) = [p1(1,3) p1(2,3) p1(3,3)];
end
%~~~~~~~~~~~~~~~~~~(partC)
s3=0;
for t=0.2:sampling_rate:0.5;
    s3=s3+1;
    h=t/0.5;
    dthitaC(:,s3)=thitaB+(thitaC-thitaB)*h;
    domegaC(:,s3)=(thitaC-thitaB)/0.5;
    dalphaC(:,s3)=[0;0;0;0;0;0];
    p3=kinematic(dthitaC(:,s3)');
    x3(s3)=p3(1,4);y3(s3)=p3(2,4);z3(s3)=p3(3,4);
    ori_3(:,s3) = [p3(1,3) p3(2,3) p3(3,3)];
end
```

PartA 和 PartC 都是帶入直線方程式來處理角度、角速度(domega)、角加速度(dalpha)，並且呼叫寫 kinematic，最主要是回傳座標矩陣，也就是我可以得到每個時間點的 noap，我會存下 a 和 p。

PartB:

```
%~~~~~~~~~~~~~~~~~~~(partB)
dB=dthitaA(:,s1)-thitaB;
dC=dthitaC(:,1)-thitaB;
s2=0;
for t=(-0.2+sampling_rate):sampling_rate:(0.2-sampling_rate)
    s2=s2+1;
    h=(t+0.2)/0.4;
    dthitaB(:,s2)=((dC+dB)*(2-h)*h^2-2*dB)*h+dB+thitaB;
    domegaB(:,s2)=((dC+dB)*(1.5-h)*2*h^2-dB)/0.2;
    dalphaB(:,s2)=(dC+dB)*(1-h)*3*h/0.2^2;
    p2=kinematic(dthitaB(:,s2)');
    x2(s2)=p2(1,4);y2(s2)=p2(2,4);z2(s2)=p2(3,4);
    ori_2(:,s2) = [p2(1,3) p2(2,3) p2(3,3)];
end
```

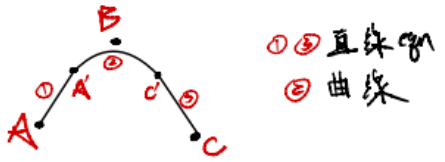
PartB 就是應用曲線方程式，其中時間點在-0.2 和 0.2 會和前一部份和後一部分重疊，因此頭尾各減少一個時點。

Plot:

```
figure(4)% Cartesian
x_all = [x1 x2 x3]; y_all = [y1 y2 y3]; z_all = [z1 z2 z3];
ori_all = [ori_1 ori_2 ori_3];
% plot3(x_all,y_all,z_all);
quiver3(x_all,y_all,z_all,ori_all(1,:),ori_all(2,:),ori_all(3,:));
xlabel('x(cm)');ylabel('y(cm)');zlabel('z(cm)');
text(10,20,30,'A(10,20,30)');
text(30,20,10,'B(30,20,10)');
text(-10,-20,-30,'C(-10,-20,-30)');
title('Cartesion Motion')
```

Plot 的部分也是將軌跡顯示的部分拿出來講，其內容與 Cartesian move 部分的 plot 基本相同。

三. 數學運算說明



① $\begin{cases} q = (A'-A)h + A \\ q' = (A'-A) \\ q'' = 0 \end{cases}$

$h = \frac{t+0.5}{0.5}$, $-0.5 \le t \le -0.2$

② $\begin{cases} q = (C-C')h + C' \\ q' = (C-C') \\ q'' = 0 \end{cases}$

$h = \frac{t}{0.5}$, $-0.2 \le t \le 0.5$

③

$q = a_4 h^4 + a_3 h^3 + a_2 h^2 + a_1 h + a_0$

$q' = 4a_4 h^3 + 5a_3 h^2 + 2a_2 h + a_1$

$q'' = 12a_4 h^2 + 6a_3 h + 2a_2$

$\Delta C = C' - B$

$\Delta B = A' - B$

$q(0) = a_0 = A'$ → $a_0 = A'$ #

$q(1) = a_4 + a_3 + a_2 + a_1 + a_0 = C'$ → $a_4 - 2a_4 + a_1 + a_0 = C' \cdots ①$

$q'(0) = a_1 = \frac{A'-B}{\frac{1}{2}}$ → $a_1 = -2\Delta B$ #

$q'(1) = 4a_4 + 3a_3 + 2a_2 + a_1 = \frac{C'-B}{\frac{1}{2}}$ → $4a_4 - 6a_4 + a_1 = 2\Delta C \cdots ②$

$q''(0) = 2a_2 = 0$ → $a_2 = 0$ #

$q''(1) = 12a_4 + 6a_3 = 0$ → $-2a_4 = a_3$

② $-2a_4 - 2\Delta B = 2\Delta C$

$a_4 = -(\Delta B + \Delta C)$ #

$a_3 = 2(\Delta B + \Delta C)$ #

$$q = -(\Delta B + \Delta C)h^4 + 2(\Delta B + \Delta C)h^3 - 2\Delta B h + A'$$

$$\text{得：} q = \left[(\Delta B + \Delta C)(2-h)h^2 - 2\Delta B\right]h + B + \Delta B$$

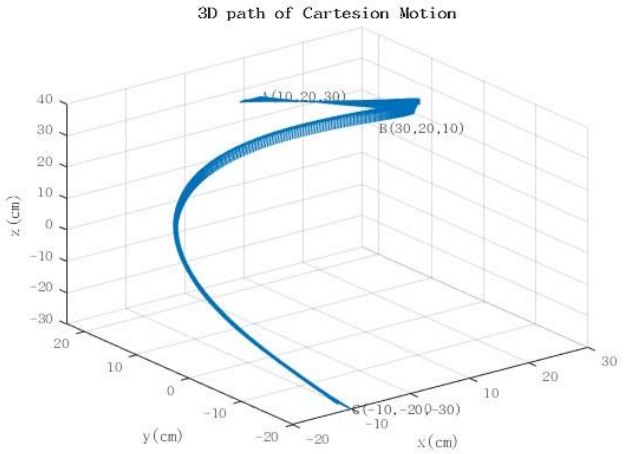
$$q' = \left[(\Delta C + \Delta B)(1.5-h)2h^2 - \Delta B\right]/0.2$$

$$q'' = (\Delta C + \Delta B)(1-h)\frac{3h}{(0.2)^2}$$

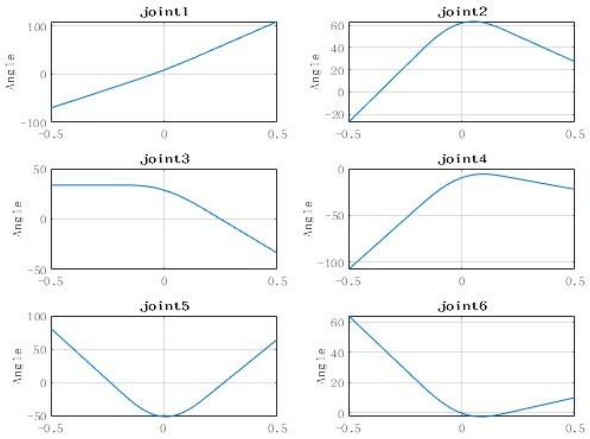
且 $h = \frac{t+0.2}{2\cdot 0.2}$, $-0.2 < t < 0.2$ #

四. 軌跡規劃曲線圖結果

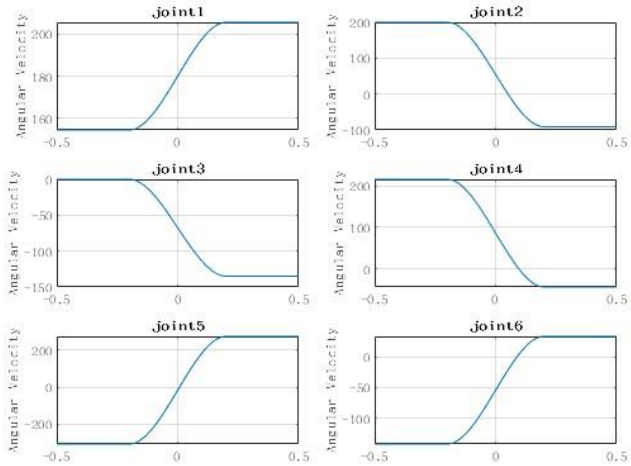
Joint move－3D plot:



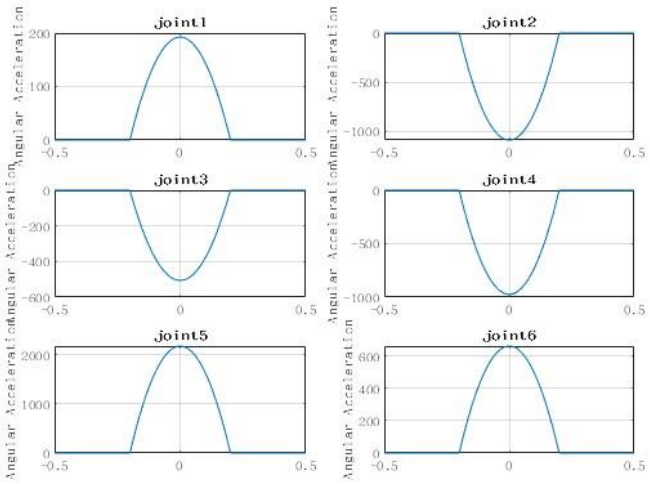
Joint move - angle:



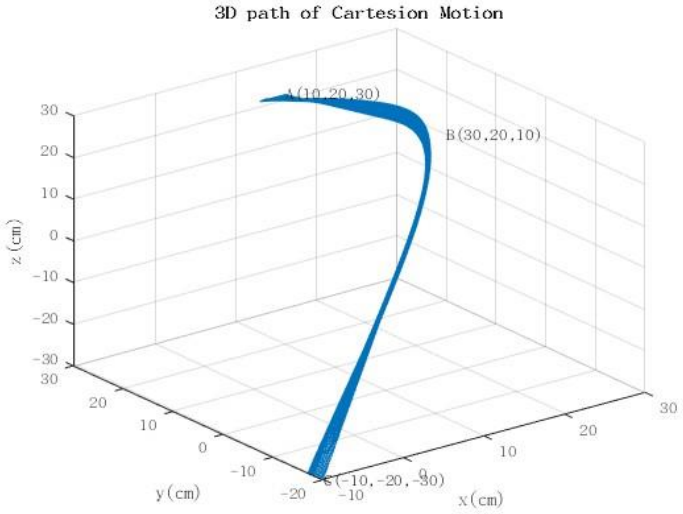
Joint move - angular velocity:



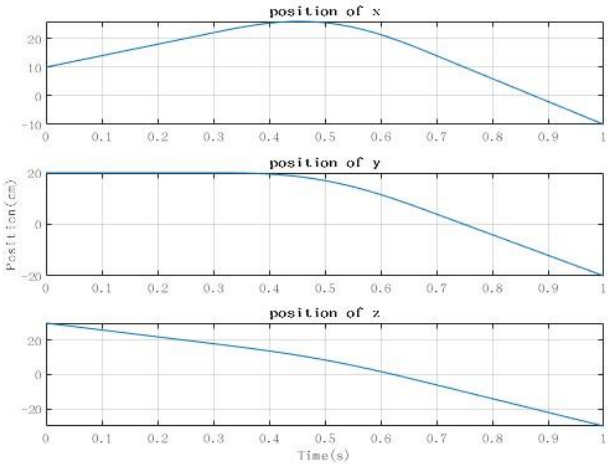
Joint move - angular acceleration:



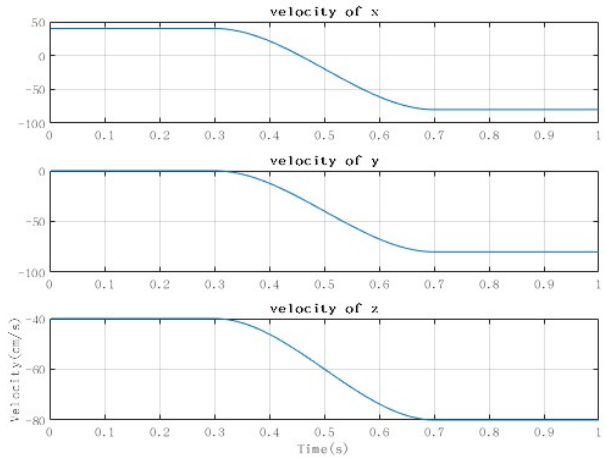
Cartesian move:



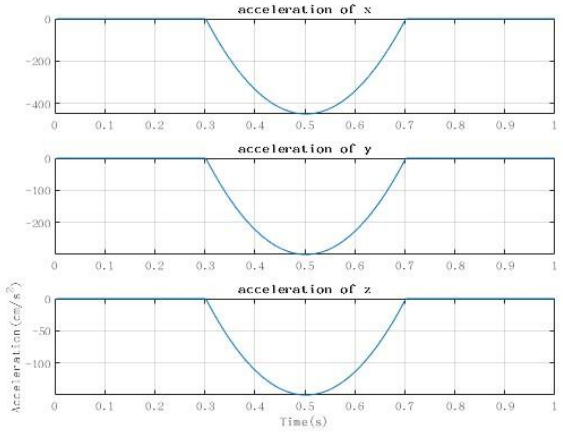
Cartesian move-position:



Cartesian move-veloicty:



Cartesian move-acceleration:



五. 加分題

Joint Motion:

- Advantage: efficient in computation, no singularity problem, no configuration problem, minimum time planning.
- Disadvantage: the corresponding Cartesian locations may be complicated.

Cartesian Motion:

- Advantage: motion between path segments and points is well defined. Different constraints, such as smoothness and shortest path, etc., can be imposed upon.
- Disadvantage: Computational load is high The motion breaks down when singularity occurs

Singularity Problem:

- 在接近奇異點時使速度變很慢很慢