Robotics Project: Part 2

一、介面說明(開發平台、如何執行 ...)

1. 上傳 .ipynb 檔案到 Google Colab 後，按 Run All 或 鍵盤 Shift+Enter
2. 或是終端機執行 python3 robotics_project2_309605008.py

二、程式架構說明(程式運行流程、核心程式碼說明 …)

1. Joint Motion

(1) 將 A,B,C 帶入上次作業的逆運動學，算出 theta

```
A = np.array([[ 0,   1,   0,   0.20 ],
              [-1,   0,   0,   0.30 ],
              [ 0,   0,   1,   0.20 ]])
B = np.array([[ 0,   0,  -1,  -0.10 ],
              [-1,   0,   0,   0.15 ],
              [ 0,   1,   0,   0.30 ]])
C = np.array([[ 1,   0,   0,  -0.25 ],
              [ 0,  -1,   0,   0.10 ],
              [ 0,   0,  -1,  -0.20 ]])

# by inverse kinematics
thetaA = np.array([31.9007,    32.4750,  -34.6102,         0,    2.1352, -121.9007])
thetaB = np.array([-0.5687,   -39.9083,  -44.4259,    5.7417,   -5.6942,   -95.7135])
thetaC = np.array([124.5999,  -28.2193, -127.9886,         0,  -23.7921,   -55.4001])
```

(2) 帶入公式計算變化量 delta

```
tacc = 0.2
T = 0.5
r = float((T-tacc)/T)
#find the point at which start to accelerate
thetaAp = thetaA + (thetaB-thetaA)*r

#calculate deltaB and deltaC
deltaC = thetaC-thetaB
deltaB = thetaAp-thetaB
```

(3) 帶入公式並用 for loop 反覆計算出各取樣時間點時，theta 的 Value, 速度和加速度。

```
#start motion from A to C pass nearby B and find their z-axis
sampling_rate = 0.002
for t_ in range(int(-0.5/sampling_rate),int(0.5/sampling_rate),1):
  t = float(t_*sampling_rate)
  if t < -0.2 :
    if t_==int(-0.5/sampling_rate):
      theta_p = (thetaB-thetaA)*((t+0.5)/T) + thetaA
      theta_v = (thetaB-thetaA)/T
      theta_a = np.zeros(6)
    else:
      theta_p = np.vstack((theta_p, (thetaB-thetaA)*((t+0.5)/T) + thetaA))
      theta_v = np.vstack((theta_v, (thetaB-thetaA)/T))
      theta_a = np.vstack((theta_a ,np.zeros(6) ))
  elif t > 0.2 :
    theta_p = np.vstack((theta_p ,deltaC*t/T+thetaB))
    theta_v = np.vstack((theta_v ,deltaC/T))
    theta_a = np.vstack((theta_a ,np.zeros(6) ))
  else :
    h = float((t+tacc)/(2*tacc))

    theta_p = np.vstack((theta_p, ((deltaC*tacc/T+deltaB)*(2-h)*h**2-2*deltaB)*h + deltaB + thetaB))
    theta_v = np.vstack((theta_v, ((deltaC*tacc/T+deltaB)*(1.5-h)*2*h**2-deltaB)/tacc))
    theta_a = np.vstack((theta_a, (deltaC*tacc/T+deltaB)*(1-h)*3*h/tacc**2))
```

(4) 作圖

```
t = [-0.5+i*sampling_rate for i in range(int( (0.5-(-0.5))/sampling_rate) )]
t1 = [-0.5+i*sampling_rate for i in range(int( (-0.2-0.002-(-0.5))/sampling_rate) )]
t2 = [-0.2+i*sampling_rate for i in range(int( (0.2-(-0.5))/sampling_rate) )]
t3 = [(0.2+0.002)+i*sampling_rate for i in range(int( (0.5-(0.2+0.002))/sampling_rate) )]


fig = plt.figure(figsize=(20,10))
for i in range(6):

  ax = fig.add_subplot(7,3,1+3*(i))
  ax.plot(t,theta_p[:,i])
  if i==0:
    ax.set_title('Joint Value')
  bx = fig.add_subplot(7,3,2+3*(i))
  bx.plot(t,theta_v[:,i])
  if i==0:
    bx.set_title('Velocity')
  cx = fig.add_subplot(7,3,3+3*(i))
  cx.plot(t,theta_a[:,i])
  if i==0:
    cx.set_title('Acceleration')
```

(5) 帶入上次作業的正運動學並用 for loop 計算各取樣時間點的 joint pose

```
def forward_kinematic(theta_input):
  #setting
  d=[0,0,0.149,0.433,0,0]
  a=[0,0.432,-0.02,0,0,0]
  alpha=[-0.5*pi, 0, 0.5*pi, -0.5*pi, 0.5*pi, 0]
  theta=np.radians(theta_input)

  #transform matrix(frame i+1 relate to i)
  A_matrix = [ np.array([[cos(theta[i])  ,  -sin(theta[i])*cos(alpha[i])  ,  sin(theta[i])*sin(alpha[i])   ,  a[i]*cos(theta[i])  ],
                         [sin(theta[i])  ,  cos(theta[i])*cos(alpha[i])   ,  -cos(theta[i])*sin(alpha[i])  ,  a[i]*sin(theta[i])  ],
                         [0              ,  sin(alpha[i])                 ,  cos(alpha[i])                 ,  d[i]                ],
                         [0              ,  0                             ,  0                             ,  1                   ]]) for i in range(6) ]


  #iterate to fin the transform matrix(frame 6 relate to 1)
  Cartesian_point=np.identity(4)
  for i in range(0,6):
    Cartesian_point=np.dot(Cartesian_point,A_matrix[i])

  #Computation of the Orientation Angles and Position
  x=Cartesian_point[0][3]
  y=Cartesian_point[1][3]
  z=Cartesian_point[2][3]
  phi = atan(Cartesian_point[1][2]/Cartesian_point[0][2])+pi
  thetaa = atan((cos(phi)*Cartesian_point[0][2] + sin(phi)*Cartesian_point[1][2])/Cartesian_point[2][2])+pi
  psi = atan((-sin(phi)*Cartesian_point[0][0] + cos(phi)*Cartesian_point[1][0])/(-sin(phi)*Cartesian_point[0][1] + cos(phi)*Cartesian_point[1][1]))+pi
  output=np.array([x,y,z, phi*180/pi, thetaa*180/pi, psi*180/pi])

  return Cartesian_point
```

```
for i in range(np.shape(theta_p)[0]):
    T = forward_kinematic(theta_p[i])
    temp = T.dot([0,0, 0.1, 1])
    if i==0:
      p = T[0:3,3]
      zAxis = temp[0:3]
    else :
      p = np.vstack((p,T[0:3,3] ))
      zAxis = np.vstack((zAxis, temp[0:3] ))
```

(6) 作圖

```
ax = fig.add_subplot(111, projection='3d')
fig.suptitle('3D path of Joint Motion')
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')
ax.scatter(0.20,0.30,0.20)
ax.scatter(-0.10,0.15,0.30)
ax.scatter(-0.25,0.10,-0.20)
ax.text(0.20,0.30,0.20, '(0.2,0.3,0.2)')
ax.text(-0.10,0.15,0.30, '(-0.1,0.15,0.3)')
ax.text(-0.25,0.10,-0.20, '(-0.25,0.1,-0.2)')
Ax = A*[0.1, 0, 0, 1]
Ay = A*[0, 0.1, 0 ,1]
Az = A*[0, 0, 0.1, 1]
for i in range(len(zAxis)):
  ax.plot([p[i,0] ,zAxis[i,0]] , [p[i,1],zAxis[i,1]], zs=[p[i,2] ,zAxis[i,2]] ,color='r')
Ax = A.dot([0.1, 0   ,0   ,1])
Ay = A.dot([0   , 0.1 ,0   ,1])
Az = A.dot([0   , 0   ,0.1 ,1])
Bx = B.dot([0.1, 0   ,0   ,1])
By = B.dot([0   , 0.1 ,0   ,1])
Bz = B.dot([0   , 0   ,0.1 ,1])
Cx = C.dot([0.1, 0   ,0   ,1])
Cy = C.dot([0   , 0.1 ,0   ,1])
Cz = C.dot([0   , 0   ,0.1 ,1])
ax.plot([A[0,3], Ax[0]], [A[1,3] ,Ax[1]], zs=[A[2,3] ,Ax[2]] ,color='r')
ax.plot([A[0,3], Ay[0]], [A[1,3] ,Ay[1]], zs=[A[2,3] ,Ay[2]] ,color='g')
ax.plot([A[0,3], Az[0]], [A[1,3] ,Az[1]], zs=[A[2,3] ,Az[2]] ,color='b')
ax.plot([B[0,3], Bx[0]], [B[1,3] ,Bx[1]], zs=[B[2,3] ,Bx[2]] ,color='r')
ax.plot([B[0,3], By[0]], [B[1,3] ,By[1]], zs=[B[2,3] ,By[2]] ,color='g')
ax.plot([B[0,3], Bz[0]], [B[1,3] ,Bz[1]], zs=[B[2,3] ,Bz[2]] ,color='b')
ax.plot([C[0,3], Cx[0]], [C[1,3] ,Cx[1]], zs=[C[2,3] ,Cx[2]] ,color='r')
ax.plot([C[0,3], Cy[0]], [C[1,3] ,Cy[1]], zs=[C[2,3] ,Cy[2]] ,color='g')
ax.plot([C[0,3], Cz[0]], [C[1,3] ,Cz[1]], zs=[C[2,3] ,Cz[2]] ,color='b')
plt.grid(True)
plt.show()
```

2. Cartesian Motion

(1) 將 A,B,C 帶入公式算出 x,y,z,phi,theta,psi

```
A = np.array([[ 0,   1,   0,   20 ],
              [-1,   0,   0,   30 ],
              [ 0,   0,   1,   20 ]])
B = np.array([[ 0,   0,  -1,  -10 ],
              [-1,   0,   0,   15 ],
              [ 0,   1,   0,   30 ]])
C = np.array([[ 1,   0,   0,  -25 ],
              [ 0,  -1,   0,   10 ],
              [ 0,   0,  -1,  -20 ]])

nA=np.array([A[0,0],A[1,0],A[2,0]])
oA=np.array([A[0,1],A[1,1],A[2,1]])
aA=np.array([A[0,2],A[1,2],A[2,2]])
pA=np.array([A[0,3],A[1,3],A[2,3]])
nB=np.array([B[0,0],B[1,0],B[2,0]])
oB=np.array([B[0,1],B[1,1],B[2,1]])
aB=np.array([B[0,2],B[1,2],B[2,2]])
pB=np.array([B[0,3],B[1,3],B[2,3]])
nC=np.array([C[0,0],C[1,0],C[2,0]])
oC=np.array([C[0,1],C[1,1],C[2,1]])
aC=np.array([C[0,2],C[1,2],C[2,2]])
pC=np.array([C[0,3],C[1,3],C[2,3]])

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

(2) 在 for loop 中計算 t=-0.5~t=-0.2 的路徑規劃

```
dataA = 0; # the index of the data of the matrix
xA_B,yA_B,zA_B = [],[],[]
for t_ in range(int(-0.5/sampling_rate),int(-0.2/sampling_rate),1):
  t = float(t_*sampling_rate)
  h = float((t-(-0.5))/0.5)
  dx = float(x*h)
  dy = float(y*h)
  dz = float(z*h)
  dsi = float(psi)
  dtheta = float(theta*h)
  dphi = float(phi*h)

  S_psi=sin(psi);
  C_psi=cos(psi);
  S_theta=sin(dtheta);
  C_theta=cos(dtheta);
  V_theta=1-C_theta;
  S_phi=sin(dphi);
  C_phi=cos(dphi);

  #find Dr with Dr=Tr*Rar*Ror
  Tr = np.array([[1,  0,  0,  dx],
                 [0,  1,  0,  dy],
                 [0,  0,  1,  dz],
                 [0,  0,  0,  1 ]])
  Rar = np.array([[(S_psi**2)*V_theta+C_phi, -S_psi*C_psi*V_theta    , C_psi*S_phi, 0 ],
                  [-S_psi*C_psi*V_theta    , (C_psi**2)*V_theta+C_phi, S_psi*S_phi, 0 ],
                  [-C_psi*S_phi            , -S_psi*S_phi            , C_phi      , 0 ],
                  [0                       , 0                       , 0          , 1 ]])
  Ror = np.array([[C_theta, -S_theta, 0, 0 ],
                  [S_theta, C_theta , 0, 0 ],
                  [0      , 0       , 1, 0 ],
                  [0      , 0       , 0, 1 ]])
  Dr = (Tr.dot(Rar)).dot(Ror)

  if t_ == int(-0.5/sampling_rate) : pA_B = np.array([A.dot(Dr)])
  else : pA_B = np.vstack((pA_B,[A.dot(Dr)]))

  xA_B.append(pA_B[dataA,0,3])
  yA_B.append(pA_B[dataA,1,3])
  zA_B.append(pA_B[dataA,2,3])

  dataA += 1
```

(3) 紀錄 A’點，帶入公式算出新的 x,y,z,phi,theta,psi

```python
A2 = pA_B[-1].copy()        #A'位置
nA2 = np.array([A2[0,0],A2[1,0],A2[2,0]])
oA2 = np.array([A2[0,1],A2[1,1],A2[2,1]])
aA2 = np.array([A2[0,2],A2[1,2],A2[2,2]])
pA2 = np.array([A2[0,3],A2[1,3],A2[2,3]])
xA = (nB.T).dot((pA2-pB))
yA = (oB.T).dot((pA2-pB))
zA = (aB.T).dot((pA2-pB))
psiA = atan2((oB.T).dot(aA2),(nB.T).dot(aA2))
thetaA = atan2(sqrt(((nB.T).dot(aA2))**2+((oB.T).dot(aA2))**2), (aB.T).dot(aA2))
SphiA = -sin(psiA)*cos(psiA)*(1-cos(thetaA))*((nB.T).dot(nA2))+((cos(psiA))**2*(1-cos(thetaA))+cos(thetaA))*((oB.T).dot(nA2))-sin(psiA)*sin(thetaA)*((aB.T).do
CphiA = -sin(psiA)*cos(psiA)*(1-cos(thetaA))*((nB.T).dot(oA2))+((cos(psiA))**2*(1-cos(thetaA))+cos(thetaA))*((oB.T).dot(oA2))-sin(psiA)*sin(thetaA)*((aB.T).do
phiA=atan2(SphiA,CphiA);

xC = (nB.T).dot((pC-pB))
yC = (oB.T).dot((pC-pB))
zC = (aB.T).dot((pC-pB))
psiC = atan2((oB.T).dot(aC),(nB.T).dot(aC))
thetaC = atan2(sqrt(((nB.T).dot(aC))**2+((oB.T).dot(aC))**2),(aB.T).dot(aC))
SphiC = -sin(psiC)*cos(psiC)*(1-cos(thetaC))*((nB.T).dot(nC))+((cos(psiC))**2*(1-cos(thetaC))+cos(thetaC))*((oB.T).dot(nC))-sin(psiC)*sin(thetaC)*((aB.T).dot
CphiC = -sin(psiC)*cos(psiC)*(1-cos(thetaC))*((nB.T).dot(oC))+((cos(psiC))**2*(1-cos(thetaC))+cos(thetaC))*((oB.T).dot(oC))-sin(psiC)*sin(thetaC)*((aB.T).dot
phiC = atan2(SphiC,CphiC)

if abs(psiC-psiA)>pi/2:
    psiA = psiA+pi
    thetaA = -thetaA
```

(4) 在 for loop 中計算 t=-0.2~t=0.2 的路徑規劃

```python
dataB = 0
x_B,y_B,z_B = [],[],[]
#從 A' 到 C' 曲線部分的路徑規劃 (-0.2s~0.2s)
for t_ in range(int(-0.2/sampling_rate),int(0.2/sampling_rate),1):
  t = float(t_*sampling_rate)
  h = float((t+tacc)/(2*tacc))
  dx_B = float(((xC*0.2/0.5+xA)*(2-h)*h**2-2*xA)*h+xA )
  dy_B = float(((yC*0.2/0.5+yA)*(2-h)*h**2-2*yA)*h+yA )
  dz_B = float(((zC*0.2/0.5+zA)*(2-h)*h**2-2*zA)*h+zA )
  dpsi_B = float((psiC-psiA)*h+psiA)
  dtheta_B = float(((thetaC*0.2/0.5+thetaA)*(2-h)*h**2-2*thetaA)*h+thetaA)
  dphi_B = float(((phiC*0.2/0.5+phiA)*(2-h)*h**2-2*phiA)*h+phiA)

  S_psi=sin(dpsi_B)
  C_psi=cos(dpsi_B)
  S_theta=sin(dtheta_B)
  C_theta=cos(dtheta_B)
  V_theta=1-C_theta
  S_phi=sin(dphi_B)
  C_phi=cos(dphi_B)

  Tr = np.array([[1, 0, 0, dx_B ],
                 [0, 1, 0, dy_B ],
                 [0, 0, 1, dz_B ],
                 [0, 0, 0, 1    ]])
  Rar = np.array([[S_psi**2*V_theta+C_phi, -S_psi*C_psi*V_theta   , C_psi*S_phi, 0 ],
                  [-S_psi*C_psi*V_theta  , C_psi**2*V_theta+C_phi , S_psi*S_phi, 0 ],
                  [-C_psi*S_phi          , -S_psi*S_phi           , C_phi      , 0 ],
                  [0                     , 0                      , 0          , 1 ]])
  Ror = np.array([[C_theta, -S_theta, 0, 0 ],
                  [S_theta, C_theta , 0, 0 ],
                  [0      , 0       , 1, 0 ],
                  [0      , 0       , 0, 1 ]])

  Dr_B = (Tr.dot(Rar)).dot(Ror)
  if t_ == int(-0.2/sampling_rate) : p_B = np.array([B.dot(Dr_B)])
  else : p_B = np.vstack((p_B,[B.dot(Dr_B)]))
  x_B.append(p_B[dataB,0,3])
  y_B.append(p_B[dataB,1,3])
  z_B.append(p_B[dataB,2,3])

  dataB += 1
```

(5) 在 for loop 中計算 t=0.2~t=0.5 的路徑規劃

```python

    # % find Dr with Dr=Tr*Rar*Ror
    Tr = np.array([[1, 0, 0, dx_C ],
                   [0, 1, 0, dy_C ],
                   [0, 0, 1, dz_C ],
                   [0, 0, 0, 1    ]])
    Rar = np.array([[(S_psi**2)*V_theta+C_phi, -S_psi*C_psi*V_theta     , C_psi*S_phi, 0 ],
                    [-S_psi*C_psi*V_theta  , (C_psi**2)*V_theta+C_phi    , S_psi*S_phi, 0 ],
                    [-C_psi*S_phi          , -S_psi*S_phi              , C_phi      , 0 ],
                    [0                     , 0                         , 0          , 1 ]])
    Ror = np.array([[C_theta, -S_theta, 0, 0 ],
                    [S_theta, C_theta , 0, 0 ],
                    [0      , 0       , 1, 0 ],
                    [0      , 0       , 0, 1]])


    Dr_C = (Tr.dot(Rar)).dot(Ror)


    if t_ == int(0.2/sampling_rate) : p_C = np.array([B.dot(Dr_C)])
    else : p_C = np.vstack((p_C,[B.dot(Dr_C)]))

    x_C.append(p_C[dataC,0,3])
    y_C.append(p_C[dataC,1,3])
    z_C.append(p_C[dataC,2,3])

    dataC += 1
```

(6) 3D 作圖

```python
fig = plt.figure(figsize=(20,10))
ax = fig.add_subplot(111, projection='3d')
fig.suptitle('3D path of Cartesian Motion')
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')
ax.scatter(20,30,20)
ax.scatter(-10,15,30)
ax.scatter(-25,10,-20)
ax.text(20,30,20, '(20,30,20)')
ax.text(-10,15,30, '(-10,15,30)')
ax.text(-25,10,-20, '(-25,10,-20)')
for i in range(len(pA_B)):
  if i==0 or i==(len(pA_B)-1):
    ax.plot([pA_B[i][0,3], pA_B[i][0,3]+pA_B[i][0,1]*5], [pA_B[i][1,3],pA_B[i][1,3]+pA_B[i][1,1]*5],zs=[pA_B[i][2,3],pA_B[i][2,3]+pA_B[i][2,1]*5],color='g')
    ax.plot([pA_B[i][0,3], pA_B[i][0,3]+pA_B[i][0,2]*5], [pA_B[i][1,3],pA_B[i][1,3]+pA_B[i][1,2]*5],zs=[pA_B[i][2,3],pA_B[i][2,3]+pA_B[i][2,2]*5],color='b')
  ax.plot([pA_B[i][0,3], pA_B[i][0,3]+pA_B[i][0,0]*5], [pA_B[i][1,3],pA_B[i][1,3]+pA_B[i][1,0]*5],zs=[pA_B[i][2,3],pA_B[i][2,3]+pA_B[i][2,0]*5],color='r')

for i in range(len(p_B)):
  if i==0 or i==(len(p_B)-1):
    ax.plot([p_B[i][0,3], p_B[i][0,3]+p_B[i][0,1]*5], [p_B[i][1,3],p_B[i][1,3]+p_B[i][1,1]*5],zs=[p_B[i][2,3],p_B[i][2,3]+p_B[i][2,1]*5],color='g')
    ax.plot([p_B[i][0,3], p_B[i][0,3]+p_B[i][0,2]*5], [p_B[i][1,3],p_B[i][1,3]+p_B[i][1,2]*5],zs=[p_B[i][2,3],p_B[i][2,3]+p_B[i][2,2]*5],color='b')
  ax.plot([p_B[i][0,3], p_B[i][0,3]+p_B[i][0,0]*5], [p_B[i][1,3],p_B[i][1,3]+p_B[i][1,0]*5],zs=[p_B[i][2,3],p_B[i][2,3]+p_B[i][2,0]*5],color='r')

for i in range(len(p_C)):
  if i==0 or i==(len(p_C)-1):
    ax.plot([p_C[i][0,3], p_C[i][0,3]+p_C[i][0,1]*5], [p_C[i][1,3],p_C[i][1,3]+p_C[i][1,1]*5],zs=[p_C[i][2,3],p_C[i][2,3]+p_C[i][2,1]*5],color='g')
    ax.plot([p_C[i][0,3], p_C[i][0,3]+p_C[i][0,2]*5], [p_C[i][1,3],p_C[i][1,3]+p_C[i][1,2]*5],zs=[p_C[i][2,3],p_C[i][2,3]+p_C[i][2,2]*5],color='b')
  ax.plot([p_C[i][0,3], p_C[i][0,3]+p_C[i][0,0]*5], [p_C[i][1,3],p_C[i][1,3]+p_C[i][1,0]*5],zs=[p_C[i][2,3],p_C[i][2,3]+p_C[i][2,0]*5],color='r')

plt.grid(True)
plt.show()
```

(7) 計算微分(速度),二階微分(加速度)並作圖

```
X = np.concatenate((xA_B, x_B ,x_C))
Y = np.concatenate((yA_B, y_B ,y_C))
Z = np.concatenate((zA_B, z_B ,z_C))
t = [-0.5+i*sampling_rate for i in range(len(X))]


plt.plot(t,X)
plt.xlim(-0.5,0.5)
plt.ylim(X.min(),X.max())
plt.title('position of x')
plt.xlabel('time (sec)')
plt.ylabel('position (cm)')
plt.grid(True)
plt.show()


plt.plot(t,Y)
plt.xlim(-0.5,0.5)
plt.ylim(Y.min(),Y.max())
plt.title('position of y')
plt.xlabel('time (sec)')
plt.ylabel('position (cm)')
plt.grid(True)
plt.show()

plt.plot(t,Z)
plt.xlim(-0.5,0.5)
plt.ylim(Z.min(),Z.max())
plt.title('position of z')
plt.xlabel('time (sec)')
plt.ylabel('position (cm)')
plt.grid(True)
plt.show()

dX = np.diff(X)/sampling_rate
dY = np.diff(Y)/sampling_rate
dZ = np.diff(Z)/sampling_rate
dt = [-0.5+i*sampling_rate for i in range(1,len(X))]


plt.plot(dt,dX)
plt.xlim(-0.5,0.5)
plt.ylim(-60,-30)
plt.title('velocity of x')
plt.xlabel('time (sec)')
plt.ylabel('velocity (cm/sec)')
plt.grid(True)
plt.show()

plt.plot(dt,dY)
plt.xlim(-0.5,0.5)
plt.ylim(-30,-10)
plt.title('velocity of y')
plt.xlabel('time (sec)')
plt.ylabel('velocity (cm/sec)')
plt.grid(True)
plt.show()

plt.plot(dt,dZ)
plt.xlim(-0.5,0.5)
plt.ylim(-100,dZ[0])
plt.title('velocity of z')
plt.xlabel('time (sec)')
plt.ylabel('velocity (cm/sec)')
plt.grid(True)
plt.show()

```

```
ddX = np.diff(dX)/sampling_rate;
ddY = np.diff(dY)/sampling_rate;
ddZ = np.diff(dZ)/sampling_rate;
ddt = [-0.5+i*sampling_rate for i in range(2,len(X))]

plt.plot(ddt,ddX)
plt.xlim(-0.5,0.5)
plt.ylim(0,ddX[int(1/(2*sampling_rate))])
plt.title('acceleration of x')
plt.xlabel('time (sec)')
plt.ylabel('acceleration (cm/sec^2)')
plt.grid(True)
plt.show()

plt.plot(ddt,ddY)
plt.xlim(-0.5,0.5)
plt.ylim(0,ddY[int(1/(2*sampling_rate))])
plt.title('acceleration of y')
plt.xlabel('time (sec)')
plt.ylabel('acceleration (cm/sec^2)')
plt.grid(True)
plt.show()

plt.plot(ddt,ddZ)
plt.xlim(-0.5,0.5)
plt.ylim(ddZ[int(1/(2*sampling_rate))],0)
plt.title('acceleration of z')
plt.xlabel('time (sec)')
plt.ylabel('acceleration (cm/sec^2)')
plt.grid(True)
plt.show()
```

三、數學運算說明

Joint motion 於 transition 過程之運動路徑 $\theta(t)$ 速度 $\omega(t)$ 加速度 $a(t)$ 分別表示為

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

$$\dot{\theta}(t)=\omega(t)=\frac{1}{T}\left[\left(\Delta B+\Delta C\right)\left(3-2h(t)\right)h^2(t)-\Delta B\right]$$

$$\ddot{\theta}(t)=a(t)=\frac{3}{t_{acc}T}\left[\left(\Delta B+\Delta C\right)\left(1-h(t)\right)h(t)\right]$$

其中

$$\Delta B=\theta(t_A)-\theta(t_B),\quad \Delta C=\theta(t_C)-\theta(t_B)$$

且

$$h(t)=\frac{t+t_{acc}}{2t_{acc}},\quad -T\le t\le +T,\quad t_{acc}=0.2,\quad T=0.5$$

Cartesian motion 中，將各 drive 變數於於 transition 過程之運動路徑 $\theta(t)$ 速度 $\omega(t)$ 加速度 $a(t)$ 分別表示為

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

$$\dot{q}(t)=\frac{1}{T}\Big[\big(\Delta B+\Delta C\big)\big(3-2h(t)\big)h^2(t)-\Delta B\Big]$$

$$\ddot{q}(t)=\frac{3}{t_{acc}T}\Big[\big(\Delta B+\Delta C\big)\big(1-h(t)\big)h(t)\Big]$$

其中

$$\Delta B=q(t_A)-q(t_B),\quad \Delta C=q(t_C)-q(t_B)$$

且

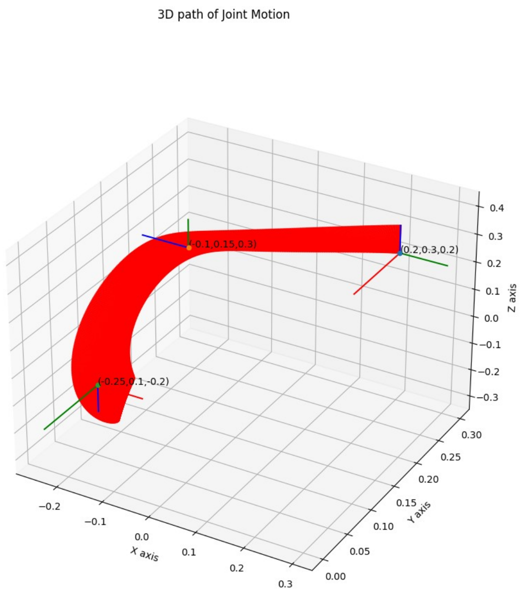
$$q(t_A)=x_A,y_A,z_A,\theta_A,\phi_A,\quad q(t_B)=B=0,\quad q(t_C)=x_C,y_C,z_C,\theta_C,\phi_C$$

另外

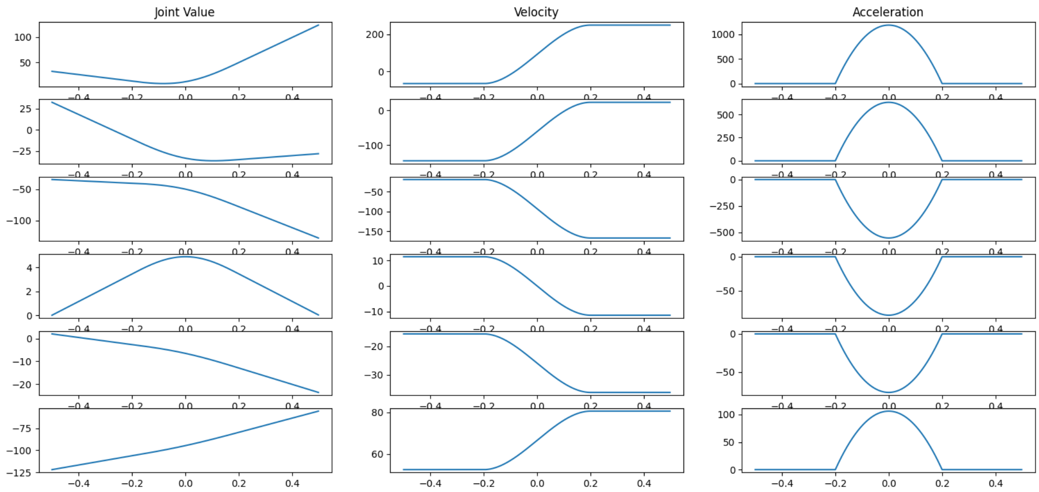
$$h(t)=\frac{t+t_{acc}}{2t_{acc}},\quad -T\le t\le +T,\quad t_{acc}=0.2,\quad T=0.5$$

四、軌跡規劃曲線圖結果

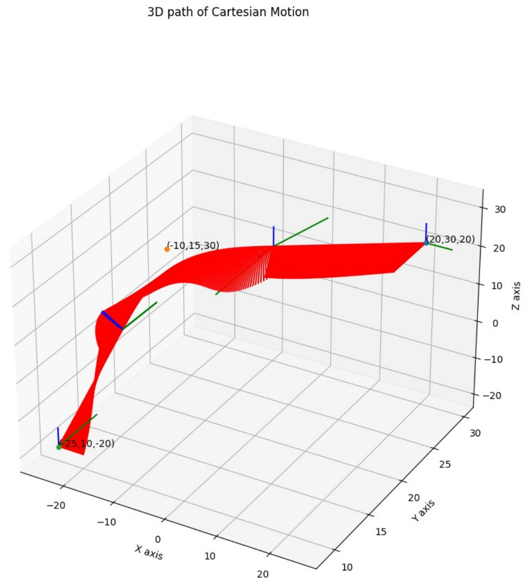
1.軸座標軌跡規劃曲線圖



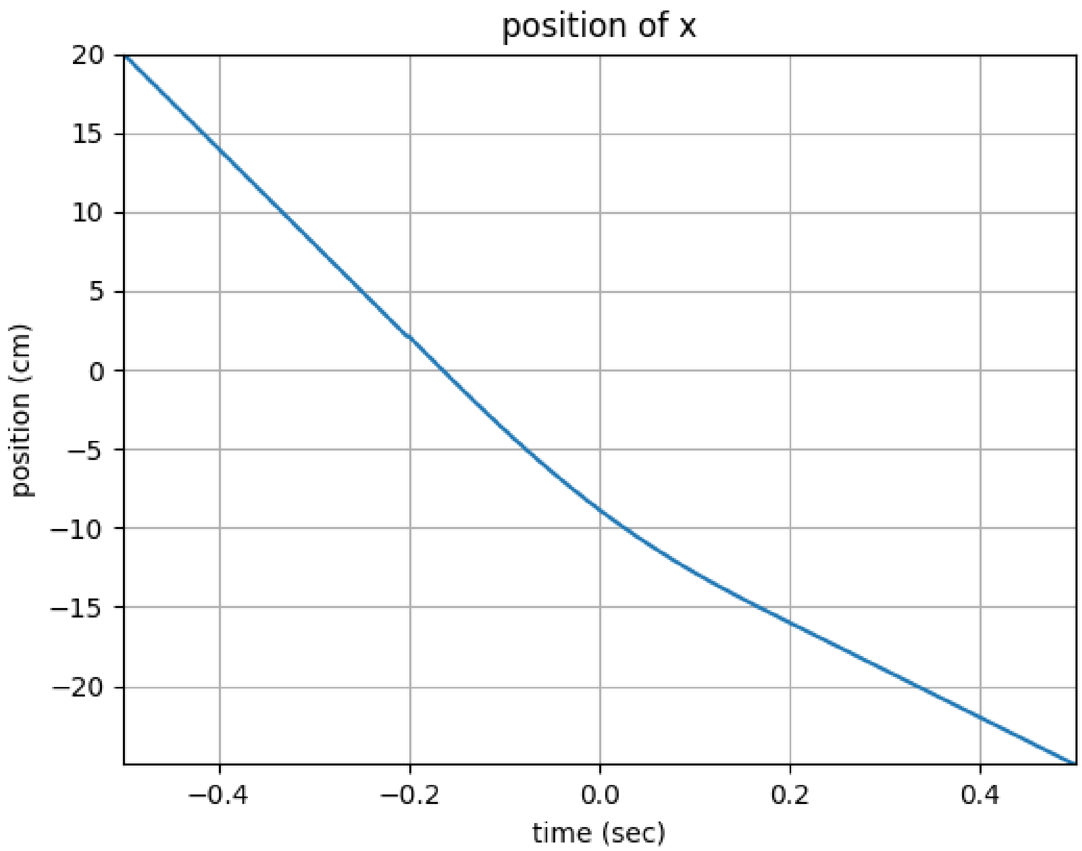
2.六軸軸變數、速度、加速度之變化圖

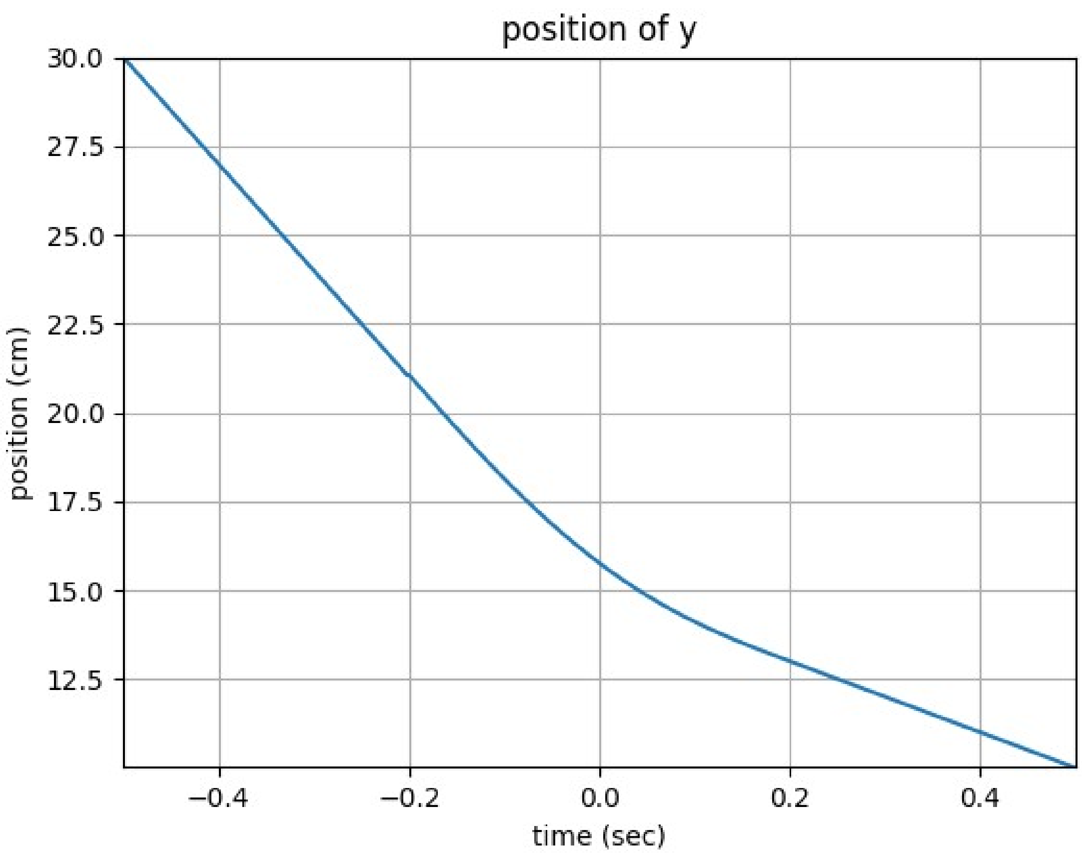


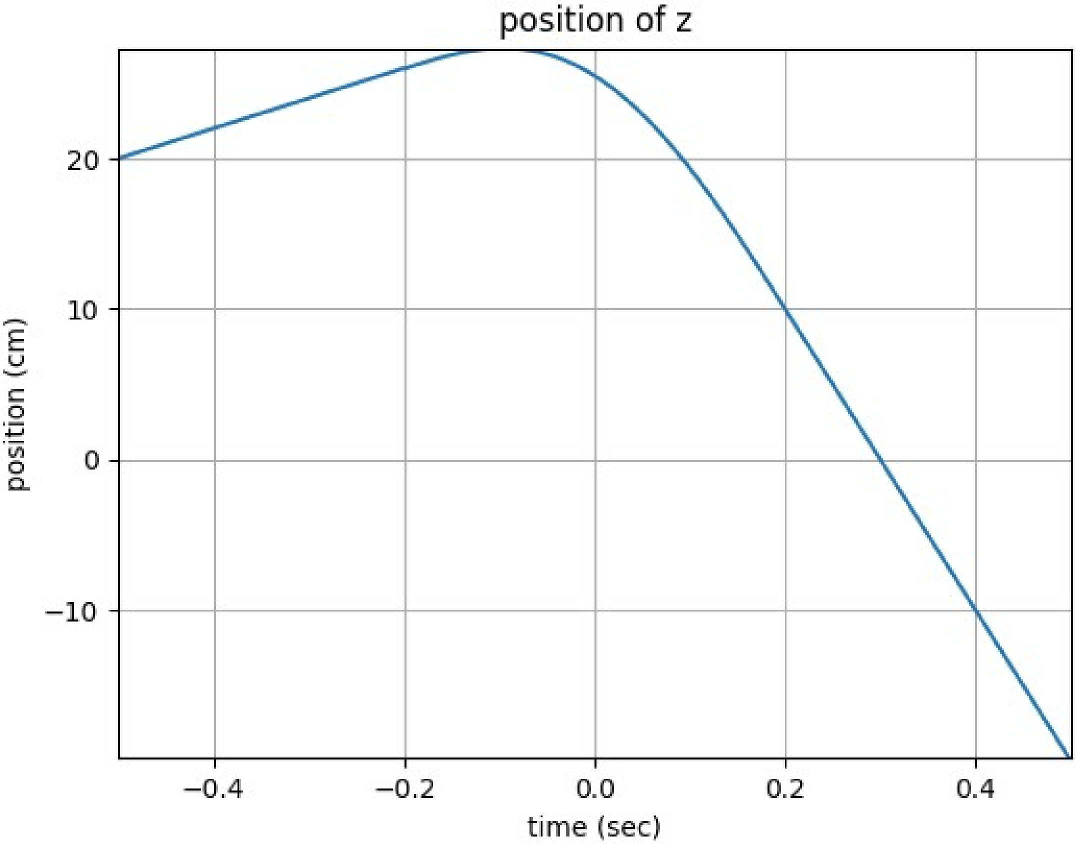
3.卡式座標軌跡規劃曲線圖

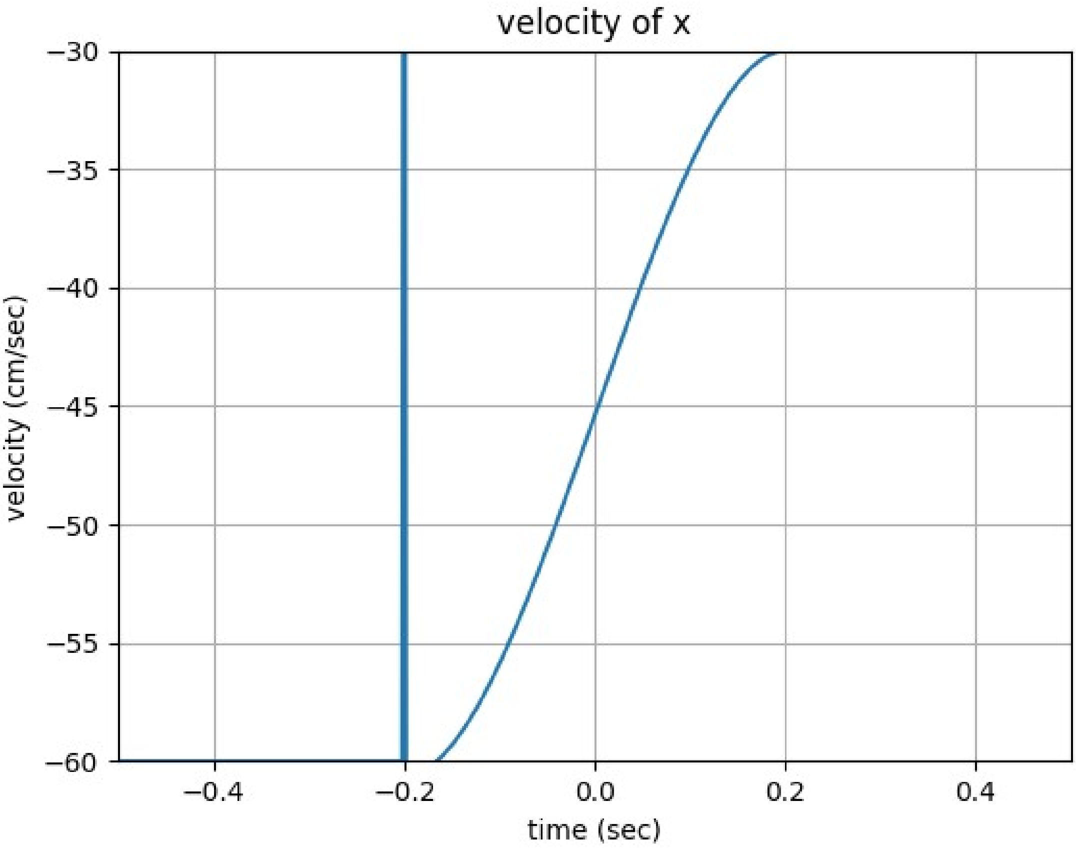


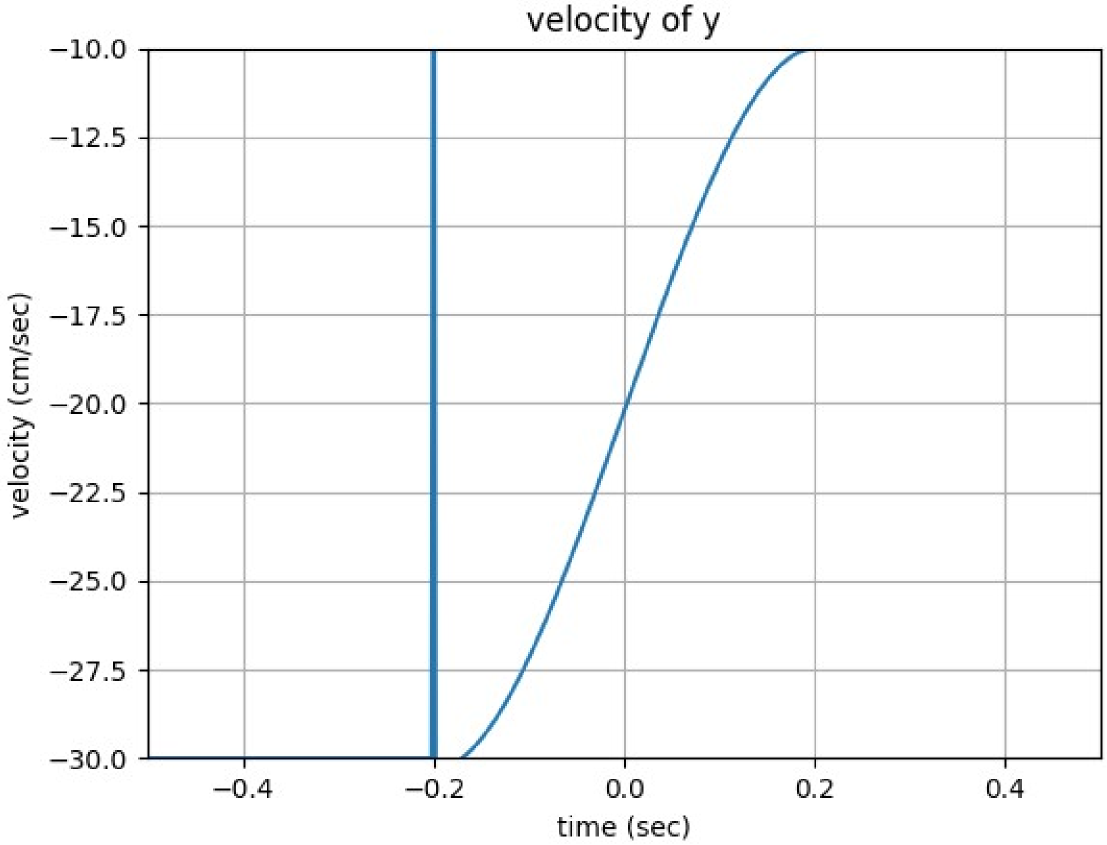
4.末端點位置、速度、加速度之變化

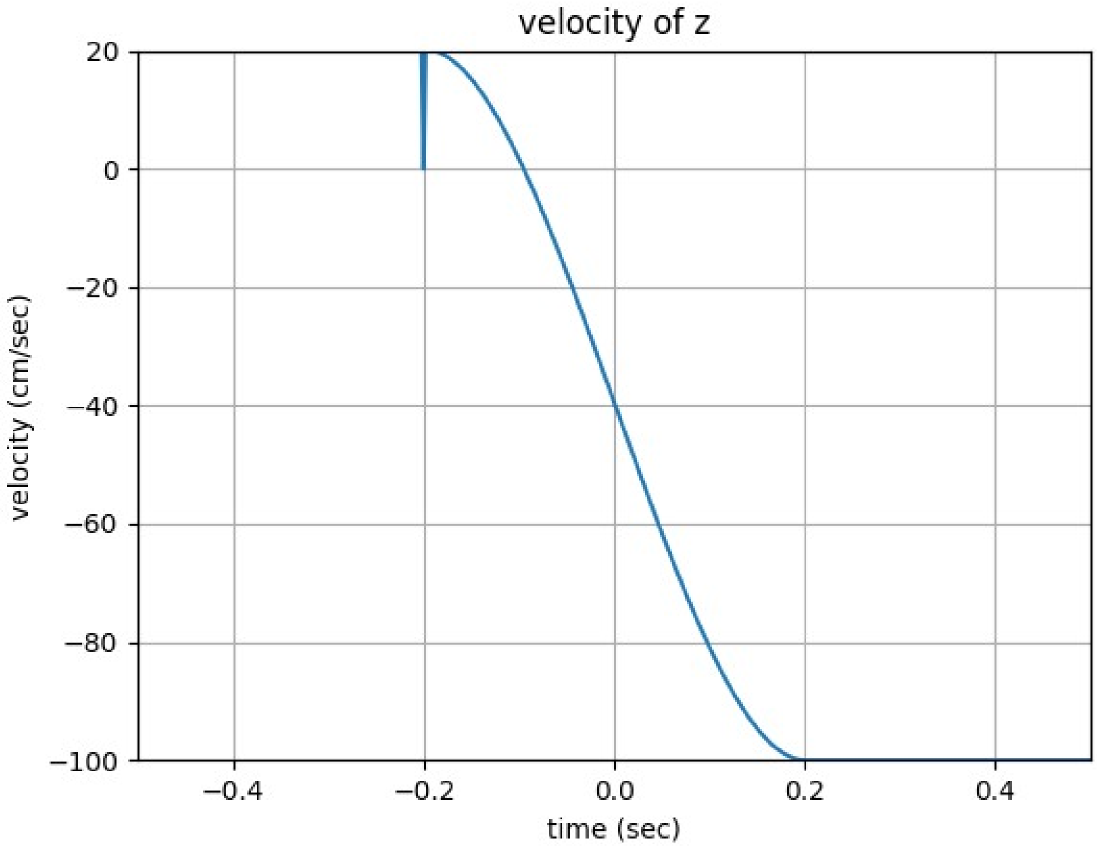


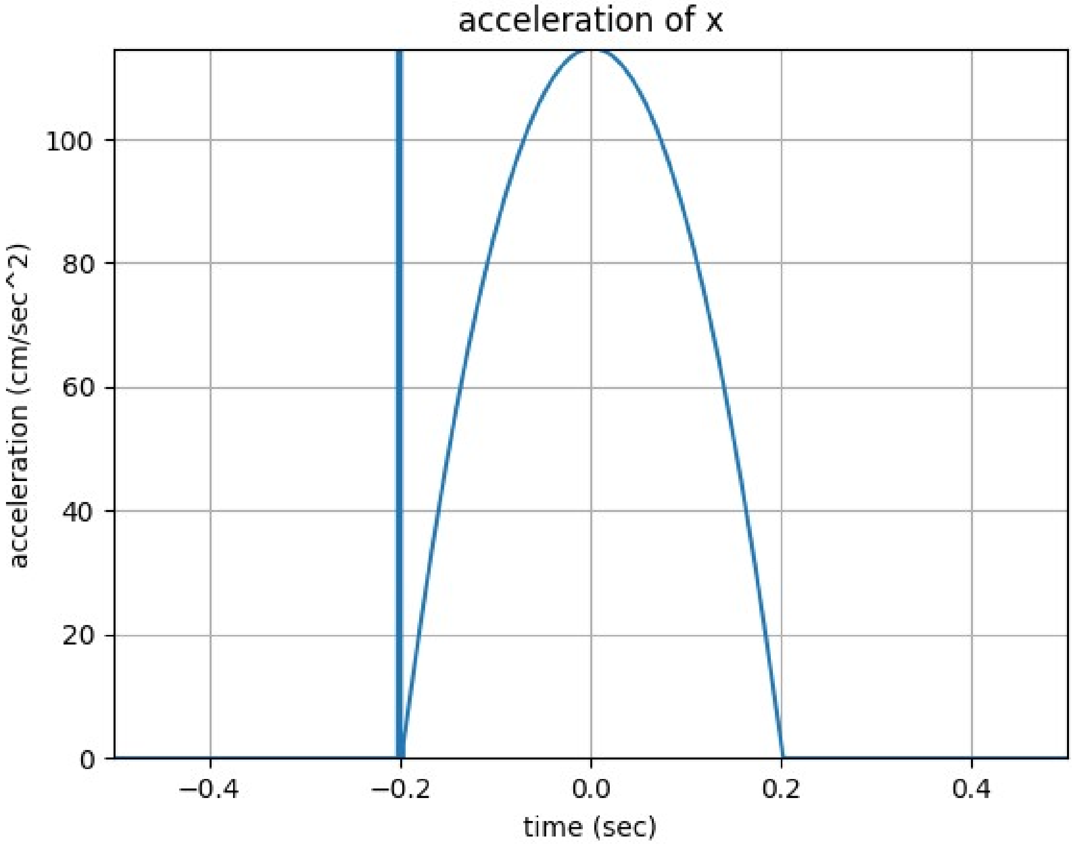


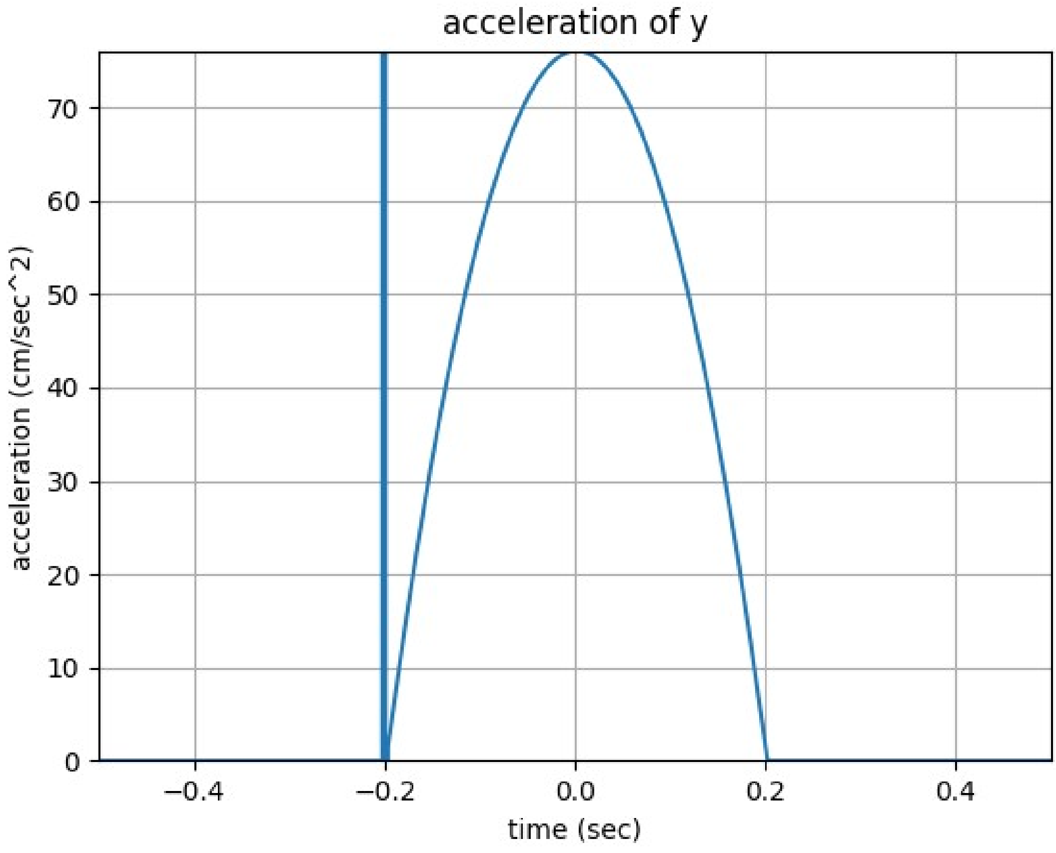

position of z
position (cm)
time (sec)
20
10
0
−10
−0.4
−0.2
0.0
0.2
0.4

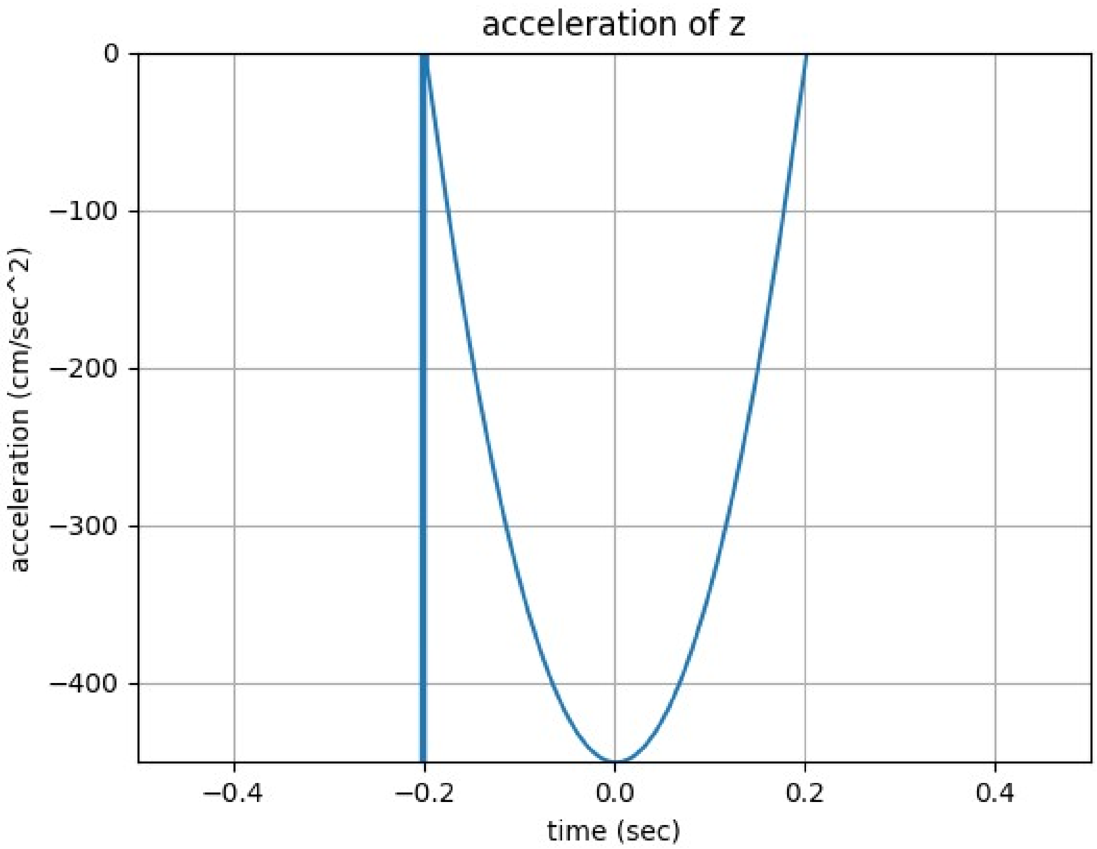

velocity of x
velocity (cm/sec)
time (sec)
−30
−35
−40
−45
−50
−55
−60
−0.4
−0.2
0.0
0.2
0.4


velocity of y
−10.0
−12.5
−15.0
−17.5
−20.0
−22.5
−25.0
−27.5
−30.0
velocity (cm/sec)
−0.4
−0.2
0.0
0.2
0.4
time (sec)


velocity of z
20
0
−20
−40
−60
−80
−100
velocity (cm/sec)
−0.4
−0.2
0.0
0.2
0.4
time (sec)


acceleration of x
acceleration (cm/sec^2)
0
20
40
60
80
100
−0.4
−0.2
0.0
0.2
0.4
time (sec)


acceleration of y
acceleration (cm/sec^2)
0
10
20
30
40
50
60
70
−0.4
−0.2
0.0
0.2
0.4
time (sec)



五、討論兩種軌跡規劃的優缺點

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: (1)Computational load is high. (2)The motion breaks down when singularity occurs , J is not invertible. (3)不知道為什麼用 python 撰寫的程式，在 t=-0.2 出現了一個不連續點。同樣作法的程式在 matlab 沒有這個問題。