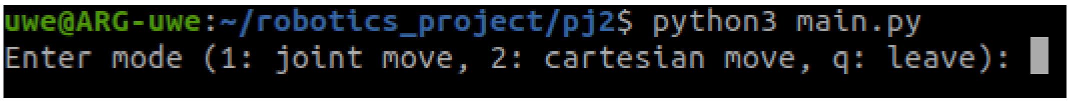
Robotics_Project2_312512063_張祐維

介面說明

Use ubuntu open terminal and enter the following commands...
Make sure your ubuntu has python3.8 above. source environment.sh to install numpy.

```
source environment.sh
python3 main.py
```

First you'll see...



```
uwe@ARG-uwe:~/robotics_project/pj2$ python3 main.py
Enter mode (1: joint move, 2: cartesian move, q: leave):
```

Enter "1" or "2" to chose joint or cartesian move.

程式架構說明

Joint Move

1. Initialization: The function starts by creating a puma560 object and initializing arrays to store joint positions (P1, P2, P3), velocities (L1, L2), and other trajectory parameters.

2. Trajectory Calculation: It divides the movement into segments (like from A to A', A' to B, and B to C) and calculates the joint positions, speeds, and accelerations for each segment based on linear interpolation. The time t is divided into intervals, and for each interval, the appropriate trajectory segment is calculated.

3. Transformation and Visualization: For each time step, the joint positions are transformed into Cartesian coordinates using the trans_pos function. These positions and orientations are then used to plot the joint movements in both joint space (angles) and Cartesian space (3D path).

Cartesian Move

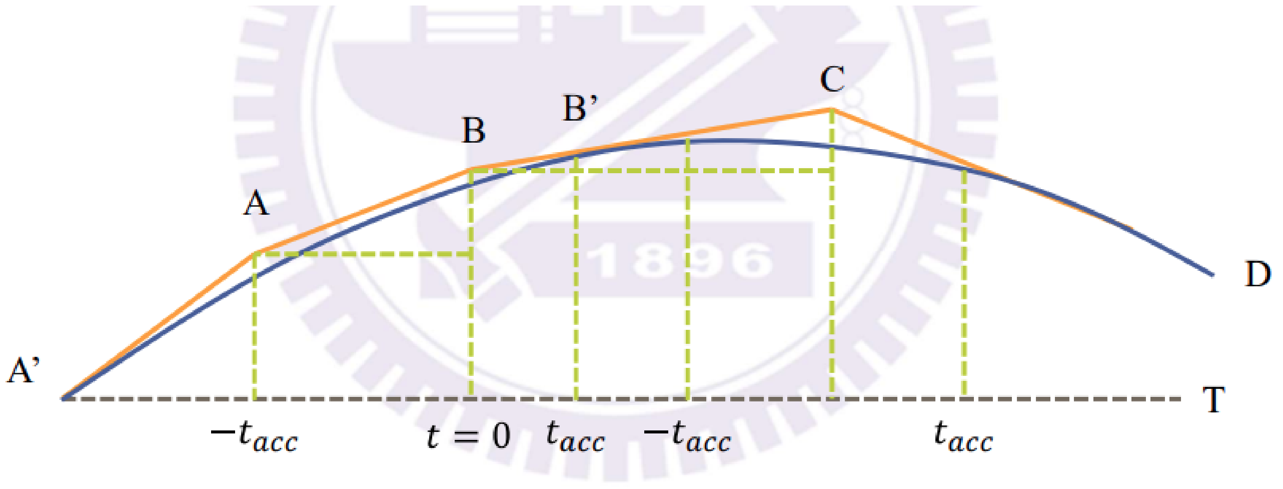
1. Trajectory Calculation Between Points: It starts by computing the transitions between points A, B, and C using the from_2_end function. This function computes the necessary changes in position and orientation (in terms of x, y, z, psi, theta, and phi) to move from one point to another. Linear and Transitional Trajectories:

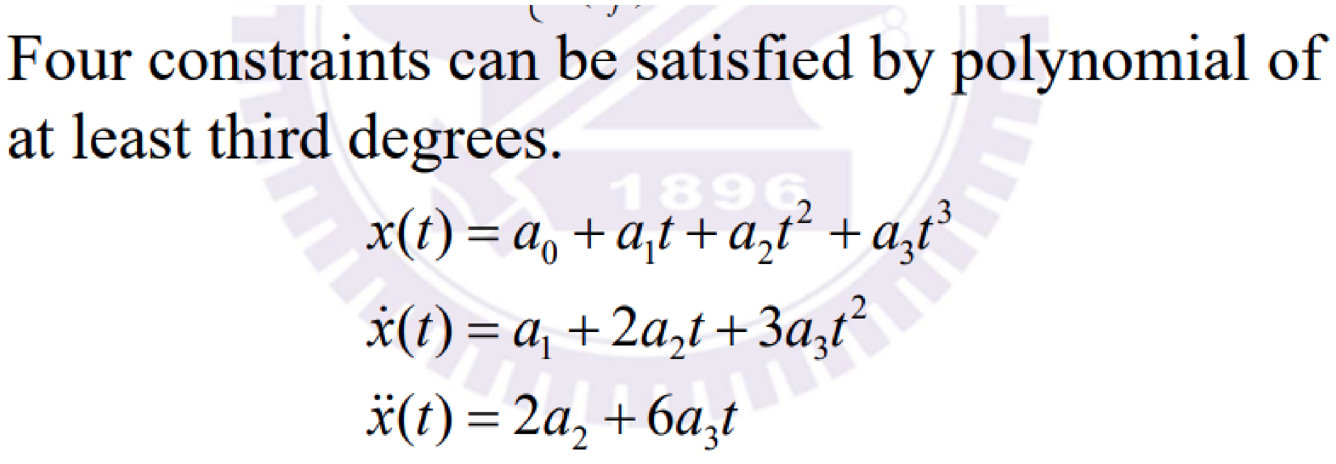
2. Linear Trajectory (A to A' and B' to C): The linear_trajectory function is used to compute the linear parts of the trajectory (from A to A' and from B' to C). It handles the straightforward movement between these points. Transitional Trajectory (A' to B'): The trans_trajectory function is then employed for the transitional phase between A' and B'. This is a more complex calculation that accounts for the smooth transition in the middle of the path, handling the changes in velocity and acceleration.

3. Combining Trajectories: The trajectories computed for each segment (A to A', A' to B', B' to C) are concatenated to form a complete path from A to C. This involves merging the positional data and the orientation data calculated for each segment.
4. Visualization: The function visualizes the entire trajectory in Cartesian space. It plots the 3D path, showing how the position of the robot's end-effector changes over time. It also plots the velocity and acceleration profiles for the movement, providing insights into the dynamics of the trajectory.
5. Orientation Handling: Special attention is given to the orientation (Euler angles: psi, theta, phi) to ensure a smooth rotational transition during the movement.

數學運算說明

Joint Move




Four constraints can be satisfied by polynomial of at least third degrees.

$$x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

$$\dot{x}(t) = a_1 + 2a_2 t + 3a_3 t^2$$

$$\ddot{x}(t) = 2a_2 + 6a_3 t$$

- Six boundary conditions from A to B'

$$q_A(t^+) = q_A(t^-) \qquad q_{B'}(t^+) = q_{B'}(t^-)$$
$$\dot{q}_A(t^+) = \dot{q}_A(t^-) \qquad \dot{q}_{B'}(t^+) = \dot{q}_{B'}(t^-)$$
$$0 = \ddot{q}_A(t^+) = \ddot{q}_A(t^-) \qquad \ddot{q}_{B'}(t^+) = \ddot{q}_{B'}(t^-) = 0$$

- Symmetry in acceleration can reduce one restriction to a 4th order which is enough

$$q(t) = a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0$$
$$\dot{q}(t) = 4a_4 t^3 + 3a_3 t^2 + 2a_2 t + a_1$$
$$\ddot{q}(t) = 12a_4 t^2 + 6a_3 t + 2a_2 \qquad -t_{acc} \le t \le t_{acc}$$

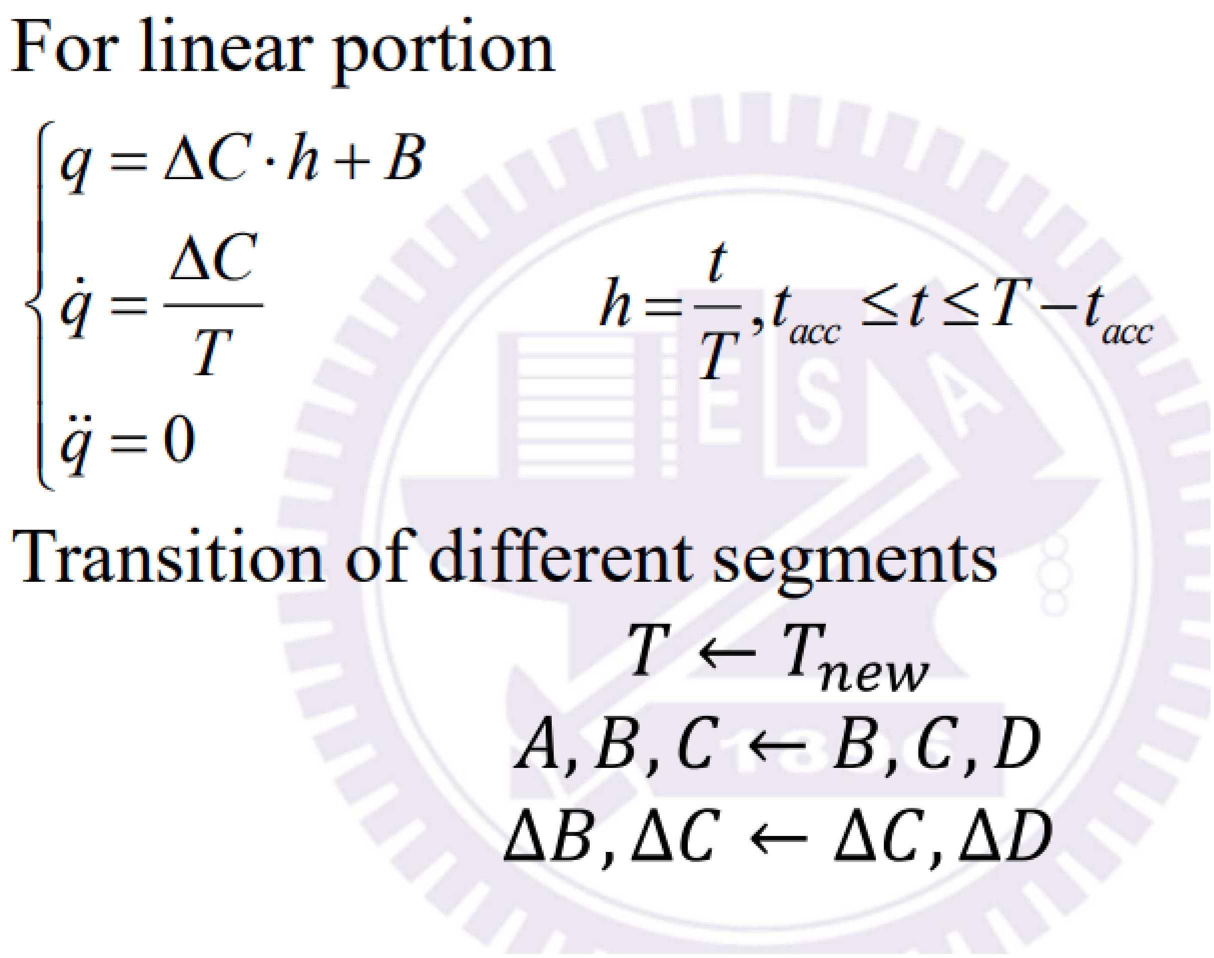
Let $\begin{cases} \Delta C = C - B \\ \Delta B = A - B \end{cases}$

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

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

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

Where $h = \dfrac{t + t_{acc}}{2t_{acc}}$ for $-t_{acc} \le t \le t_{acc}$



For linear portion

$$\begin{cases} q = \Delta C \cdot h + B \\ \dot{q} = \dfrac{\Delta C}{T} \\ \ddot{q} = 0 \end{cases} \qquad h = \frac{t}{T}, t_{acc} \le t \le T - t_{acc}$$

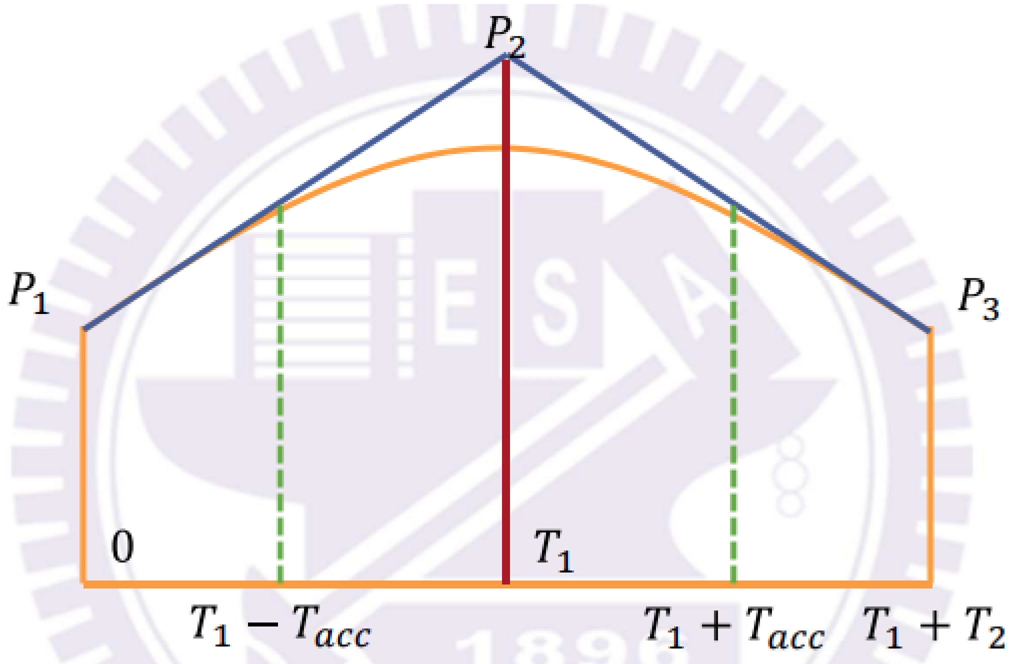
Transition of different segments

$$T \leftarrow T_{new}$$

$$A, B, C \leftarrow B, C, D$$

$$\Delta B, \Delta C \leftarrow \Delta C, \Delta D$$

Cartesian Move



$$x = {}^1n \cdot ({}^2p - {}^1p)$$
$$y = {}^1o \cdot ({}^2p - {}^1p)$$
$$z = {}^1a \cdot ({}^2p - {}^1p)$$

$$\theta = \tan^{-1}\left(\frac{\sqrt{({}^1n \cdot {}^2a)^2 + ({}^1o \cdot {}^2a)^2}}{{}^1a \cdot {}^2a}\right)$$

$$\psi = \tan^{-1}\left(\frac{{}^1o \cdot {}^2a}{{}^1n \cdot {}^2a}\right)$$

$$\text{If } |\psi_C - \psi_A| > 90^\circ\text{, then let } \begin{cases} \psi_A = \psi_A + 180^o \\ \theta_A = -\theta_A \end{cases}$$

$$r = 1, \quad D(1) = T(1) * Ra(1) * Ro(1)$$

$$T(1) = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} = D(1) * Ro(1)^{-1} * Ra(1)^{-1}$$

$$x = {}^1n \cdot ({}^2p - {}^1p)$$
$$y = {}^1o \cdot ({}^2p - {}^1p)$$
$$z = {}^1a \cdot ({}^2p - {}^1p)$$

$$K = \begin{pmatrix} C\psi & -S\psi & 0 & 0 \\ S\psi & C\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -S\psi \\ C\psi \\ 0 \\ 1 \end{pmatrix}$$

Then $Ra(r)$ represents a rotation of θ about K, which can be expressed as follows:

$$Ra(r) = \begin{pmatrix} S\psi^2 V(r\theta) + C(r\theta) & -S\psi C\psi V(r\theta) & C\psi S(r\theta) & 0 \\ -S\psi C\psi V(r\theta) & C\psi^2 V(r\theta) + C(r\theta) & S\psi S(r\theta) & 0 \\ -C\psi S(r\theta) & -S\psi S(r\theta) & C(r\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where $V(r\theta) = Vers(r\theta) = 1 - \cos(r\theta)$

$$Ro(r) = \begin{pmatrix} C(r\phi) & -S(r\phi) & 0 & 0 \\ S(r\phi) & C(r\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad D(r) = \begin{pmatrix} n_x & (\dots) & C\psi S(r\theta) & r_x \\ n_y & (\dots) & S\psi S(r\theta) & r_y \\ n_z & (\dots) & C(r\theta) & r_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$D(r) = T_r(r) * Ra(r) * Ro(r)$$

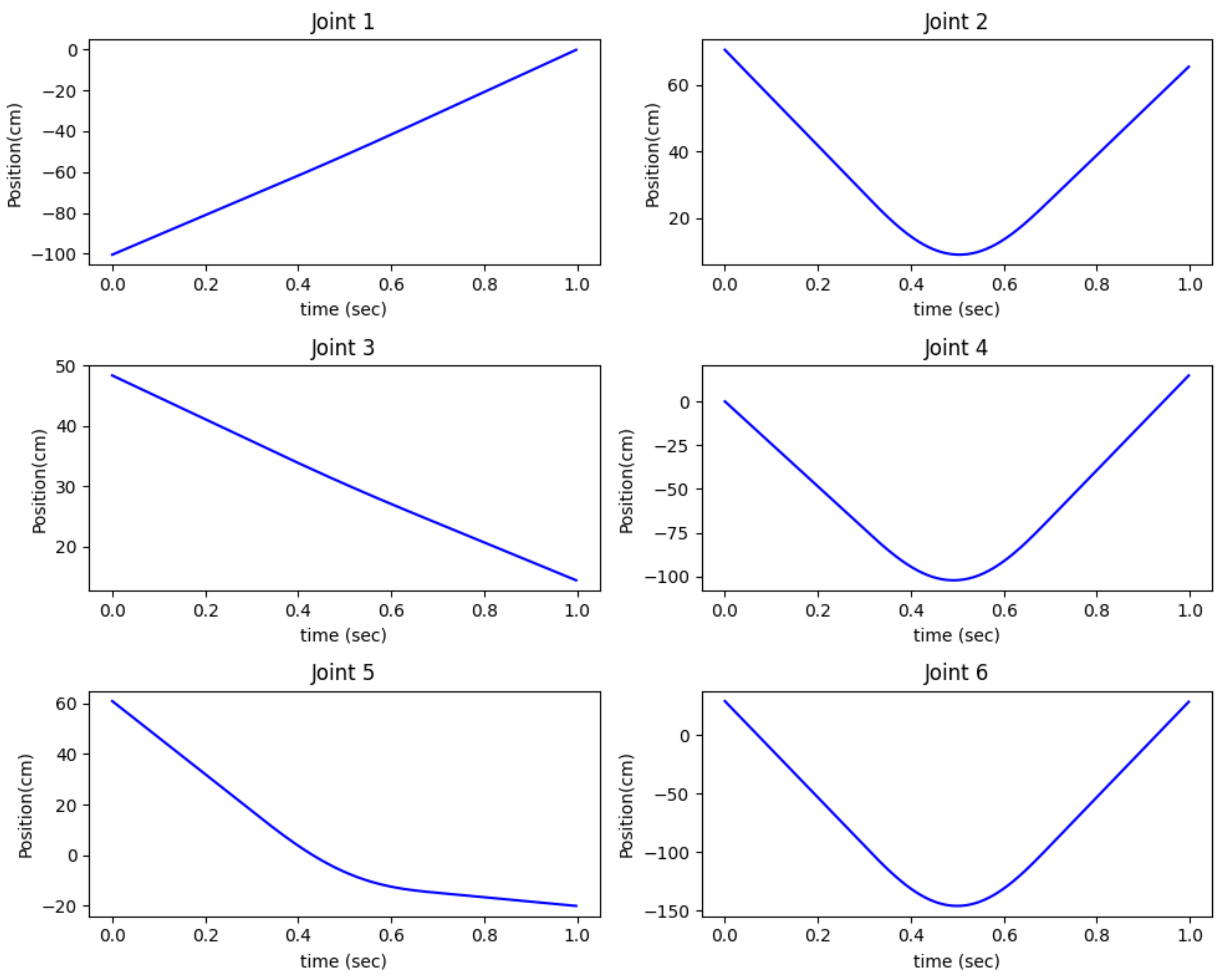
$$T_r(r) = translation\ along\ the\ line\ from\ P1\ to\ P2$$

$$T_r(r) = \begin{pmatrix} 1 & 0 & 0 & r_x \\ 0 & 1 & 0 & r_y \\ 0 & 0 & 1 & r_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

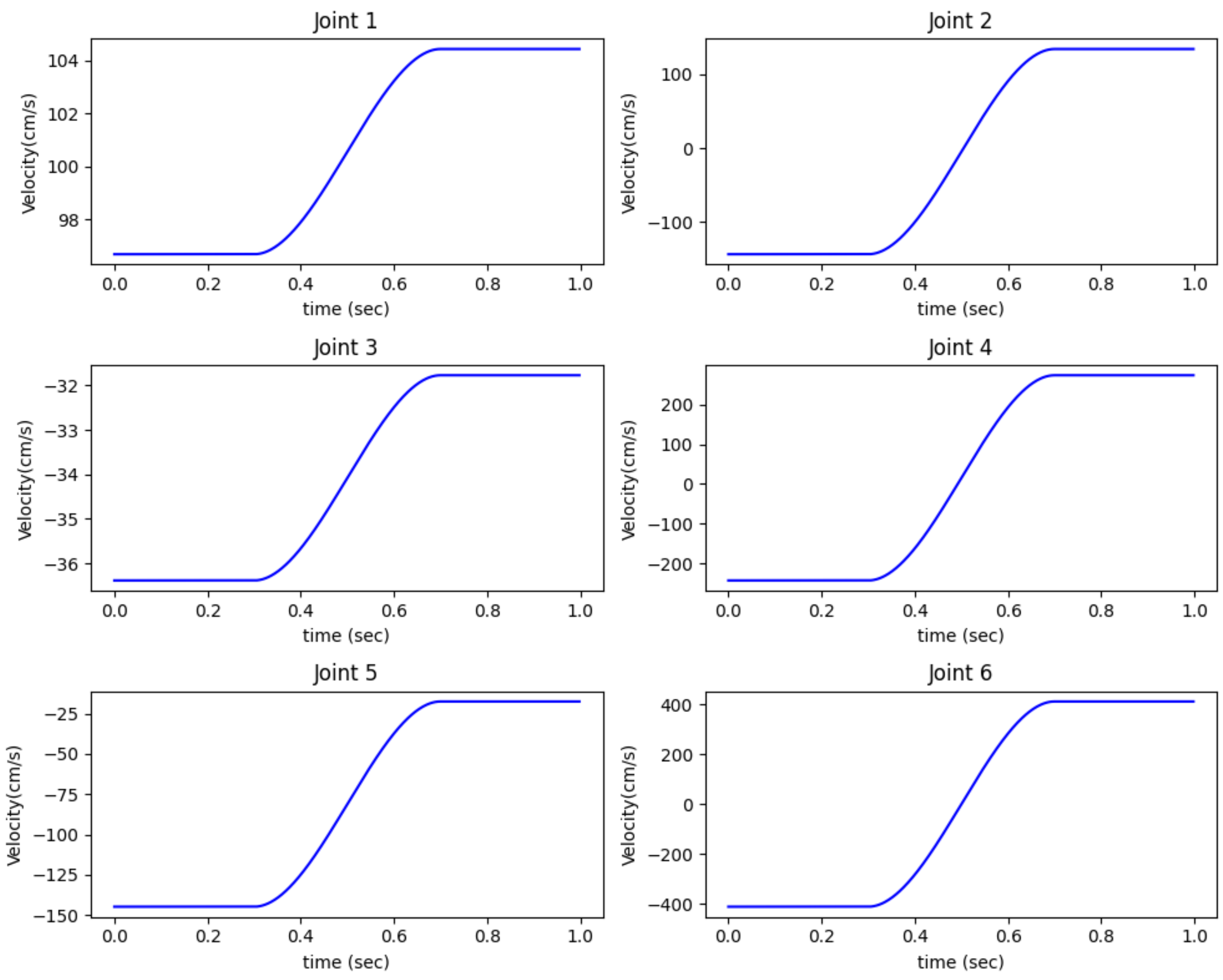
Where x, y, z are the distances between POS1 and POS2

Joint Move Result

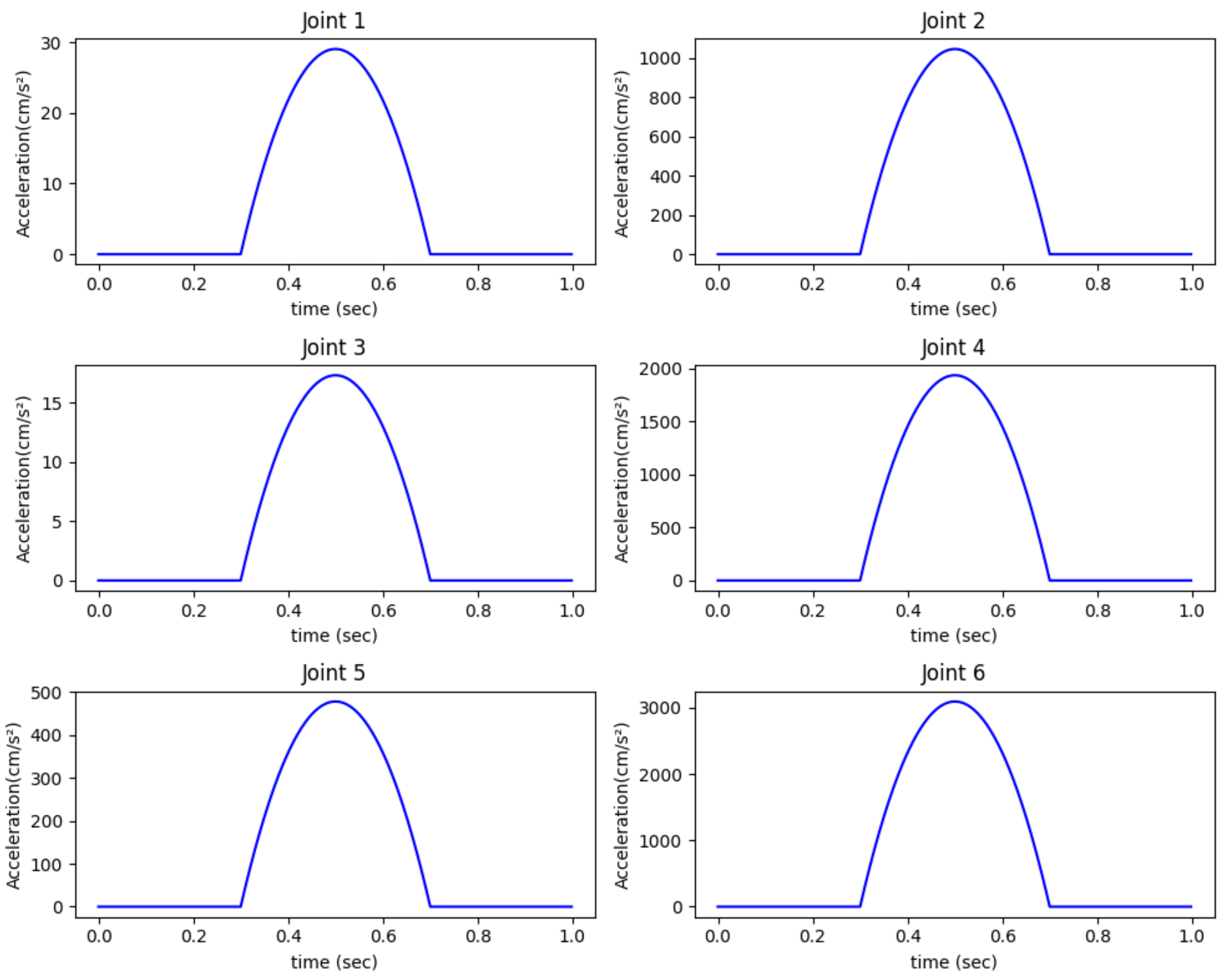
Joint angular

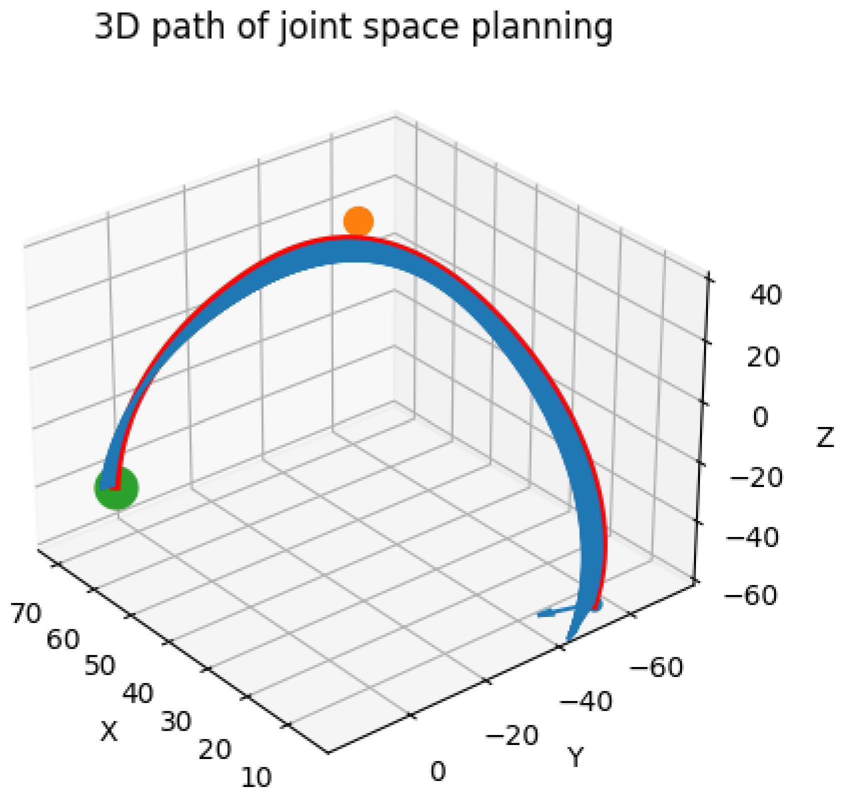


Joint angular speed



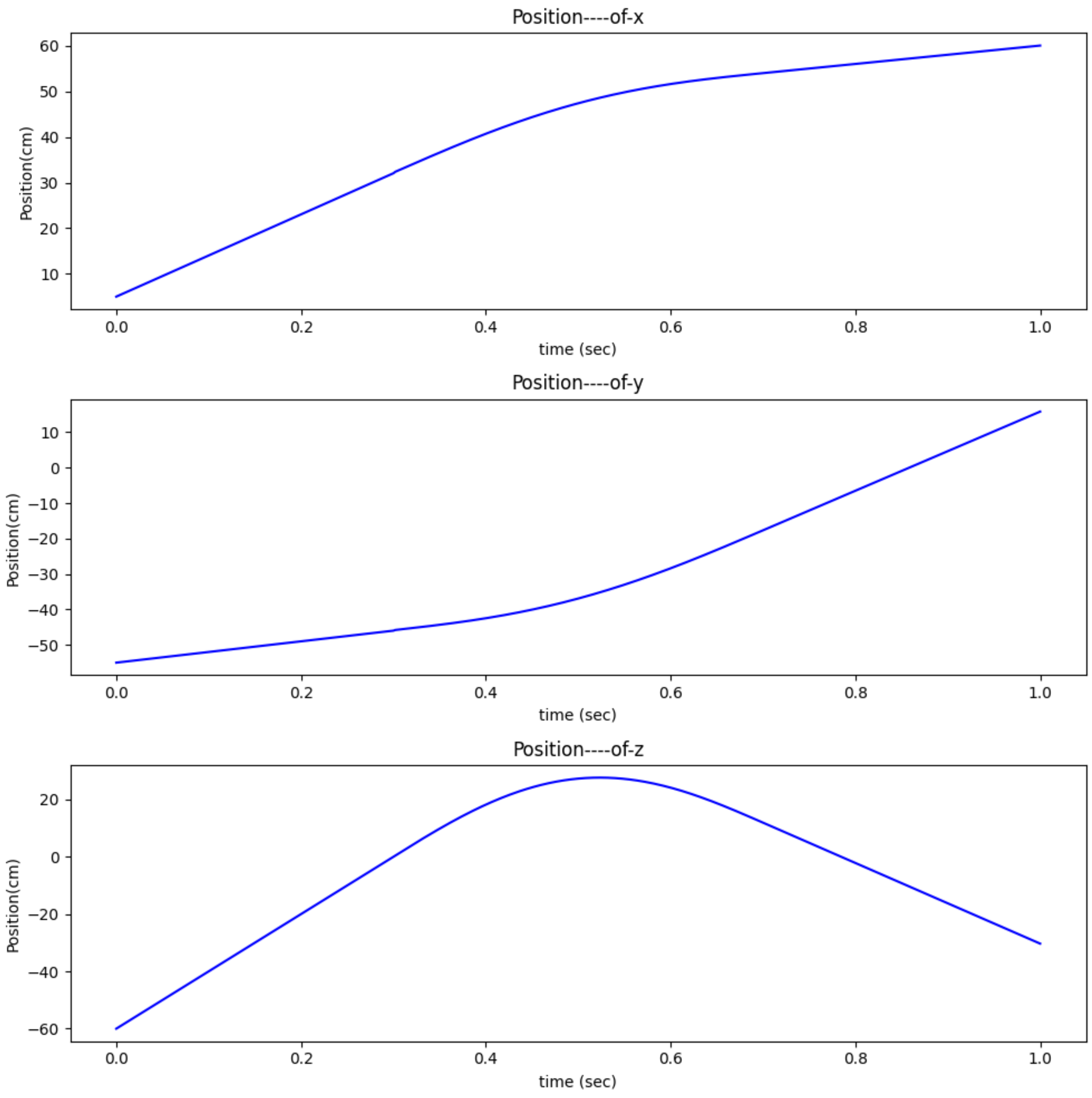
Joint angular acceleration


Joint 1
Joint 2
Joint 3
Joint 4
Joint 5
Joint 6
Acceleration(cm/s²)
time (sec)
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
200
400
600
800
1000
5
15
500
1500
2000
100
300
3000

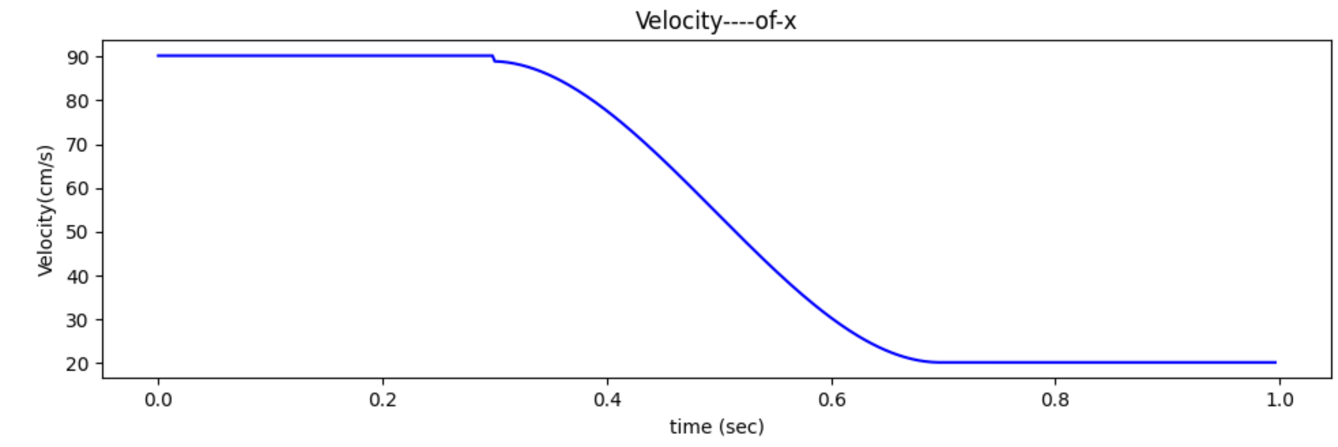


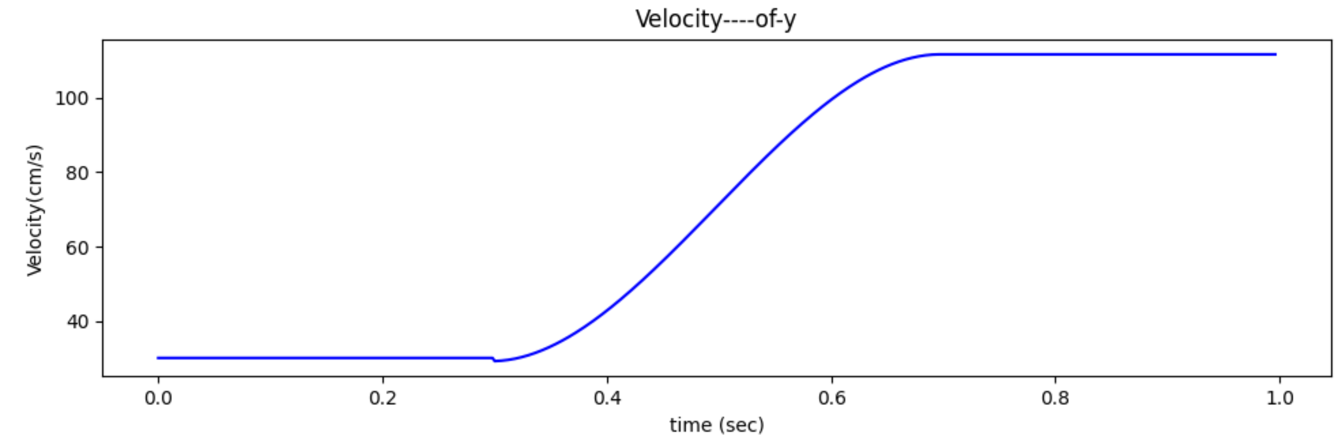
Caresian Move Result

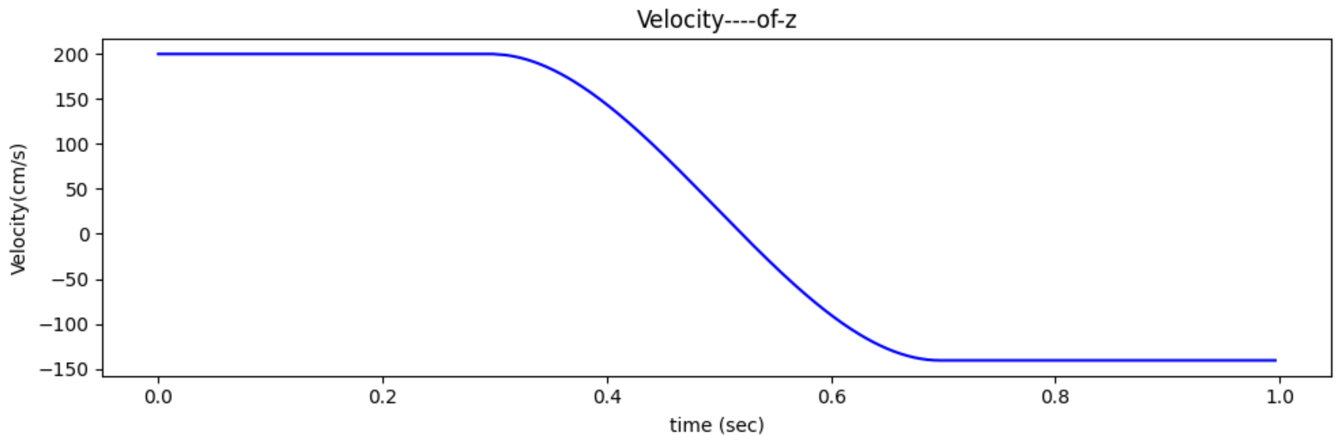
Caresian position



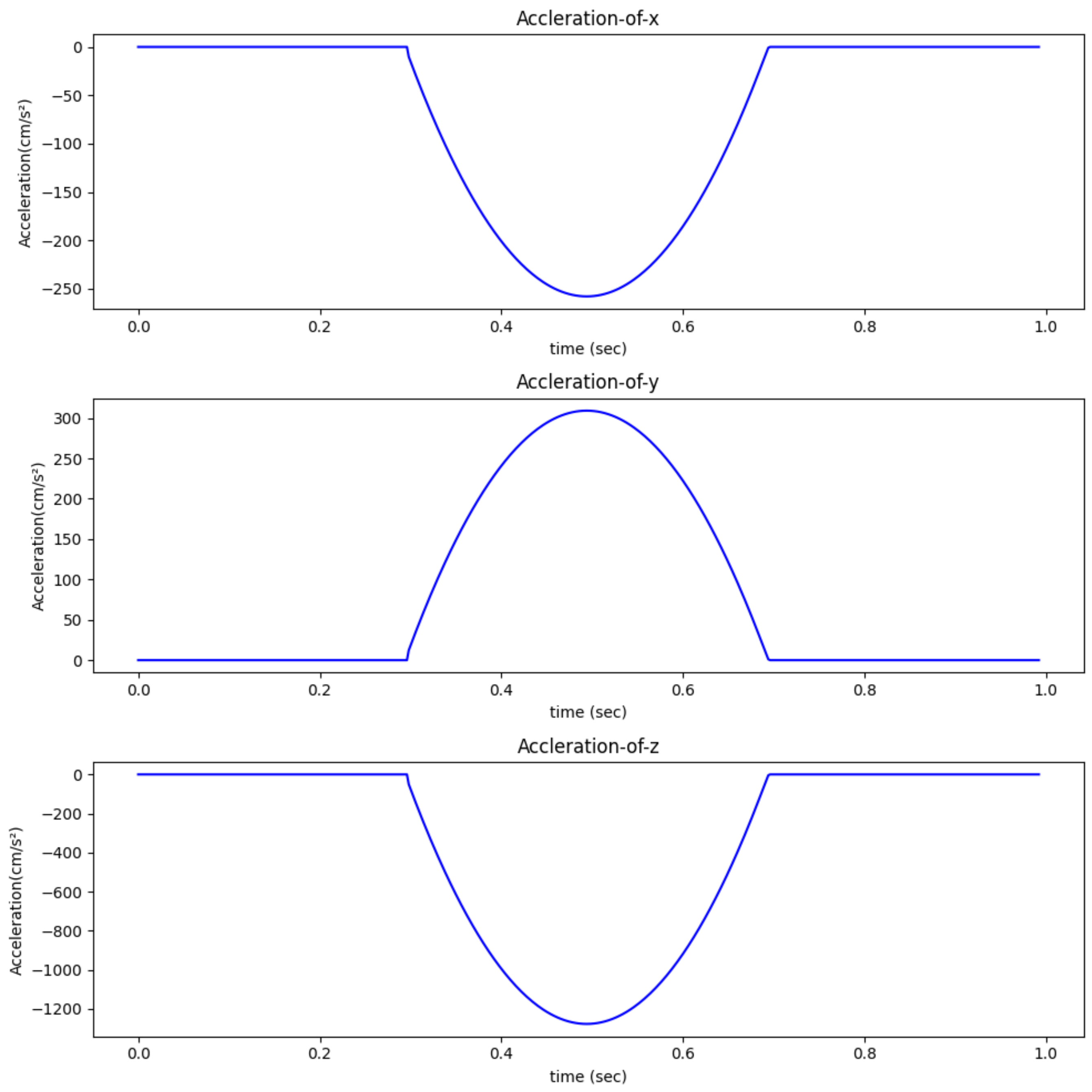
Caresian speed

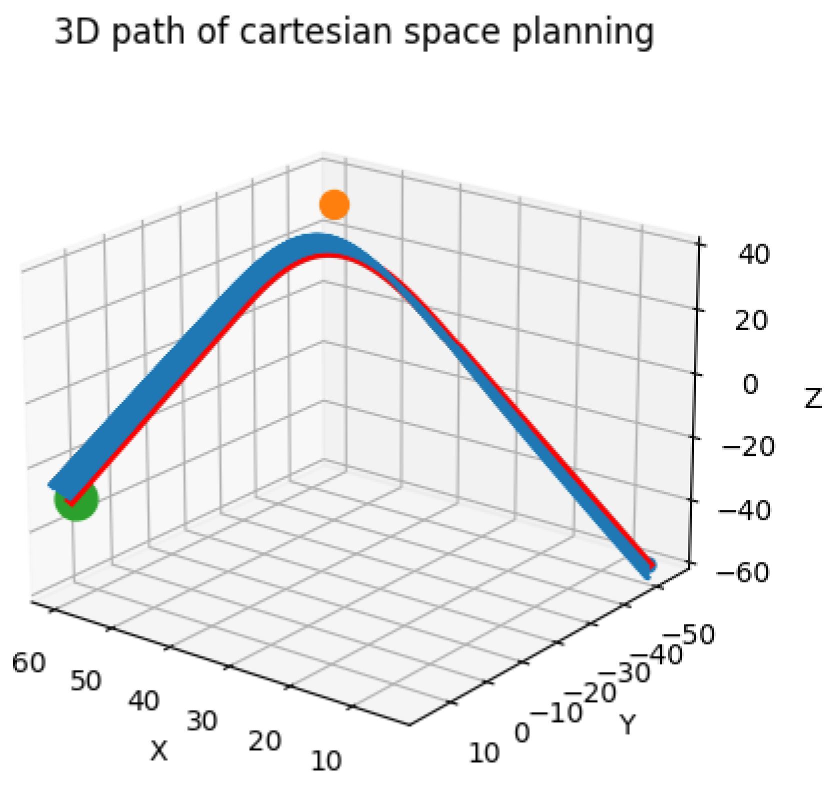






Caresian acceleration


Accleration-of-x
Acceleration(cm/s²)
0
−50
−100
−150
−200
−250
0.0
0.2
0.4
0.6
0.8
1.0
time (sec)
Accleration-of-y
Acceleration(cm/s²)
300
250
200
150
100
50
0
time (sec)
Accleration-of-z
Acceleration(cm/s²)
0
−200
−400
−600
−800
−1000
−1200
time (sec)



討論兩種軌跡規劃的優缺點、遇到奇異點如何處理

Advantages and Disadvantages:

Joint Space Planning: Easier to compute as it deals directly with the robot's actuators. However, it can be less intuitive for tasks requiring specific spatial paths. Cartesian Space Planning: More intuitive for tasks that are defined in Cartesian coordinates, like drawing or assembly tasks. The computation can be more complex due to the need for inverse kinematics.

Handling Singularities:

Singularities occur when the robot's configuration leads to a loss of degrees of freedom. They are more common in Cartesian planning due to inverse kinematics. To handle singularities, methods such as damped least squares (for inverse kinematics) or trajectory adjustment to avoid known singular configurations can be used. Advanced algorithms may also implement redundancy resolution strategies.