INTERNSHIP REPORT

Arvin Mohammadi
student number: 810698303

September 17, 2022

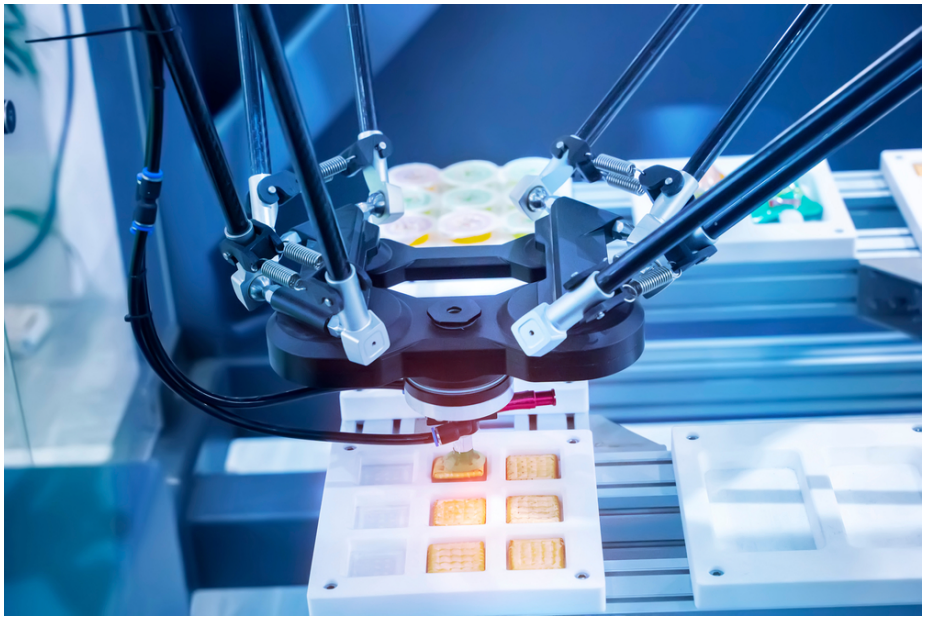
Contents

Chapter 1

INTRODUCTION

Delta Parallel Robots now-a-days are globally used for industrial purposes because of their precision and speed of operation. They are mostly used for pick and place operations in different industries. The design of Delta Robot is quite unique compared to other industrial robots. The gripper (or End-Effector) is connected to long, slender mechanical linkages. These links lead up to three or four large motors (depending on whether the EE rotates or not). The slender and lightweight structure is what allows the Delta Robot to have the speed wanted for high production rate. (see Figure 1.1)

Figure 1.1: Delta Robot



Delta Parallel Robots are used in industrial applications for their speed and precision

1.1 Applications

The Delta robot is commonly deployed in a few applications that take advantage of its unique design. Including the following:

1. Pick and place
2. Assembly
3. Disassembly
4. Packaging
5. Sorting

1.2 Strengths

As said before, the main reason you would find Delta robots in these applications is their unparalleled speed. in other words, the placement of the motors and the lightweight arms allow the Delta to reach speed that is unmatched through out the different range of industrial robots. in the applications mentioned, speed and precision is top priority, hence Delta robot is the obvious choice for all of them. An often- over-looked benefit of Deltas is in their efficient use of floor space. Most stationary robots are mounted on the ground near the workspace. The robot may have to be surrounded by a safety cage and other equipment. This type of setup may take up dozens of square feet. The Delta, however, is mounted over the top of the workspace. Because of this feature in its design, the Delta can take advantage of the commonly unused vertical space in manufacturing facilities. Better use of vertical space frees up more floor space for extra equipment and storage.

1.3 Limitations

Of course, the Delta robot, as any man-made machine, has its drawbacks. for example the payload is really low due to the lightweight structure, most Deltas will be maxed out at around only a few kilograms. Another issue of the Delta is the workspace volume or the range of reach. Again because of its mechanical construction, the range of motion is significantly limited. the work envelope is cone-shaped. This means that the farther down the Delta must reach the less side-to-side motion it can achieve. All of these limitations mean you can use Delta robot for small-space and lightweight materials and loads which is a huge hit to the range of applications it can attend.

1.4 About The company - Tavan Ressan

Tavan Ressan started out as the exclusive trade and technical representative of the German company Lenze in the year 1369 (1990 in Gregorian calendar). Most of the activity at the beginning was in field of industrial power transmission in two subsections of electrical and mechanical. This company was the first in Iran to import inverter and servo motors. Tavan Ressan has five core services:

- Marketing, counseling and sale (local and global)
- Production
- Automation and project implementation
- After-sale services
- Training

In the recent years Tavan Ressan is one of the best company in Iran when it comes to Servo motors. It offers up to thirty different models of servo motors from the Lenze company, and also other products like servo controllers, complementary systems like gear-less and direct drive, different kinds of gearbox, DC injection brakes, etc.

1.5 My assignment

This section is about what the assignment. The problem was built around the concept of controlling the End-Effector of Delta robot with a discretized velocity profile with the help of motion planning.

1.5.1 Problem

The problem, as said, is about motion planning. the algorithm previously used for generating the velocity profile, had the problem of unbounded jerk (derivative of acceleration) at the start and end of the path. meaning the robot, vibrates every-time it starts and stops moving, even though in the interpolated path, it had no signs of being uncontrollable.

1.5.2 Solution

In this part the different methods, algorithms and python implementations tested for solving the problem is introduced.

1. Point-to-point movement
   (a) 3-4-5 Polynomial interpolation
   (b) 4-5-6-7 Polynomial interpolation
   (c) Trapezoidal point-to-point movement
   (d) S-curve point-to-point movement
   (e) Jacobian point-to-point movement
2. Multi-point movement
   (a) Cubic spline interpolation
   (b) B-spline interpolation
   (c) Trapezoidal through a sequence of points
   (d) Jacobian through a sequence of points

Figure 1.2: Delta Robot

Delta Parallel Robots used in the Tavan Ressan Company

Chapter 2

KINEMATICS

This chapter is about the calculation of inverse and forward kinematics of the Delta parallel robot. IK and FK are the answers to two very fundamental questions in robotic:

1. FK: If the joint angles are given, then what are the coordinates for position of the End-Effector?

2. IK: If the coordinates for position of the EE are given, then what are the joint angles of the robot?

Moving forwards, the topics such as Jacobian, IK and FK calculation are explained in detail.

2.1 Jacobian

Jacobian matrix is a way to relate the velocity of the End-Effector, to the velocity of the actuated joints. with the formula $J_\theta \dot{\vec{\theta}} = J_P \vec{v}$ if the vector $\vec{v}$ (EE velocity vector) is given, it is possible to calculate the $\dot{\vec{\theta}}$ (actuated joints, angular velocity) in a rather simple manner.

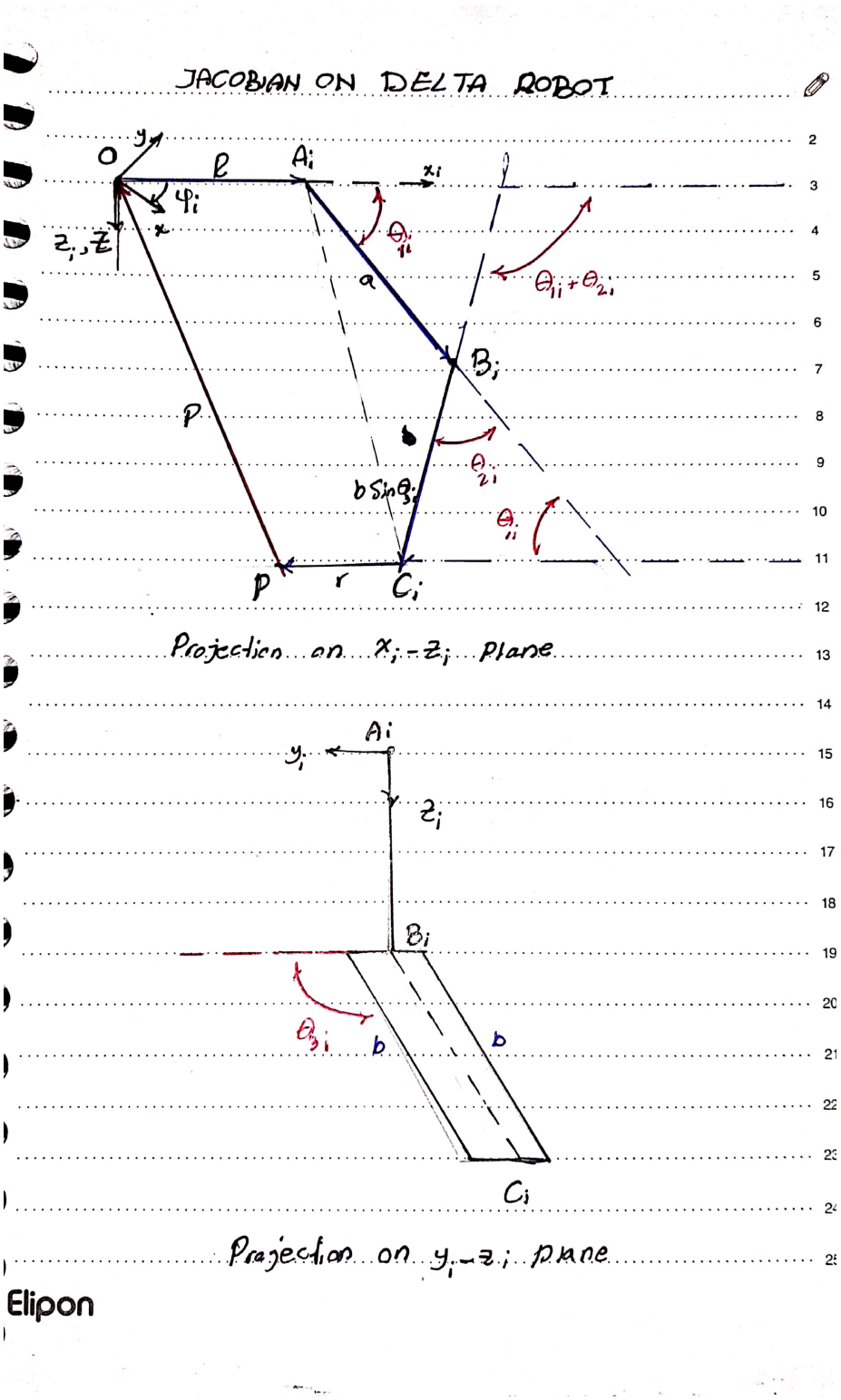
2.1.1 Theory

Looking at Figure (2.1), the most relevant loop should be picked up for the intended Jacobian analysis. So let $\vec{\theta}_{1i}$ be the angle of actuated joint, and $\vec{p}$ be the position vector of the moving platform (EE). Then:

$$\vec{\theta} = \theta_{1i} = \begin{bmatrix} \theta_{11} \\ \theta_{12} \\ \theta_{13} \end{bmatrix}$$

$$\vec{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

Figure 2.1: Kinematics



Kinematic parameters of the Delta robot

The Jacobian matrix will be derived by differentiating the appropriate loop closure equation and rearranging in the following form:

$$J_\theta \dot{\vec{\theta}} = J_P \dot{\vec{v}} \tag{2.1}$$

that:

$$J_\theta = \begin{bmatrix} J_{1x} & J_{1y} & J_{1z} \\ J_{2x} & J_{2y} & J_{2z} \\ J_{3x} & J_{3y} & J_{3z} \end{bmatrix} \tag{2.2}$$

$$J_P = \begin{bmatrix} \sin(\theta_{21})\sin(\theta_{31}) & 0 & 0 \\ 0 & \sin(\theta_{22})\sin(\theta_{32}) & 0 \\ 0 & 0 & \sin(\theta_2)\sin(\theta_{33}) \end{bmatrix} \tag{2.3}$$

with considering *Equation* 2.1 we can see that for each point in space-time (x_0, y_0, z_0, t_0) we can relate the velocity of EE to angular velocity of actuated joint. The loop needed for acquiring the correct equations is:

$$\vec{OP} = \vec{OA_i} + \vec{A_iB_i} + \vec{B_iC_i} + \vec{C_iP} \tag{2.4}$$

According to Figure (2.1), the Equation (2.4) can be re-written as:

$$\begin{bmatrix} p_x\cos(\phi_i) - p_y\sin(\phi_i) \\ p_x\sin(\phi_i) + p_y\cos(\phi_i) \\ p_z \end{bmatrix} = \begin{bmatrix} R - r \\ 0 \\ 0 \end{bmatrix} + a \begin{bmatrix} \cos(\theta_{1i}) \\ 0 \\ \sin(\theta_{1i}) \end{bmatrix} + b \begin{bmatrix} \sin(\theta_{3i})\cos(\theta_{1i} + \theta_{2i}) \\ \cos(\theta_{3i}) \\ \sin(\theta_{3i})\sin(\theta_{1i} + \theta_{2i}) \end{bmatrix} \tag{2.5}$$

from Equation (2.5) we can calculate J_θ elements as followed:

$$\begin{aligned} J_{ix} &= \sin(\theta_{3i})\cos(\theta_{1i} + \theta_{2i})\cos(\phi_i) + \cos(\theta_{3i})\sin(\phi_i) \\ J_{iy} &= -\sin(\theta_{3i})\cos(\theta_{1i} + \theta_{2i})\sin(\phi_i) + \cos(\theta_{3i})\cos(\phi_i) \\ J_{iz} &= \sin(\theta_{3i})\sin(\theta_{1i} + \theta_{2i}) \end{aligned} \tag{2.6}$$

2.1.2 Python Implementation

- INPUTS: angles of actuated joints, velocity of the End-Effector $(\vec{\theta},\ \vec{v})$
- OUTPUTS: angular velocity of actuated joints $(\dot{\vec{\theta}})$

Back tracking the theory explained above, it can be said that for calculating a Jacobian matrix for Delta robot:

1. Receiving θ_{ij} (actuated joint angles) as input
2. Calculate J_{ij} from Equation (2.6)
3. Use J_{ij} and θ_{ij} to construct J_P and J_θ
4. Use Equation (2.1) for finding $\dot{\vec{\theta}}$ from $\vec{v},\ J_{ij},\ \theta_{ij}$

2.2 Inverse Kinematics

Inverse Kinematics is dependent to geometry of the robot. With inverse kinematics, one is able to find the angle of the actuator joints in the robot, given the information about position of End-Effector.

2.2.1 Theory

For calculating inverse kinematics it is needed to calculate θ_{ij} (for $i, j = 1, 2, 3$). it is done so, from solving the system of 3 equations of Equation (2.5). firstly we have:

$$\theta_{3i} = \arccos(\frac{p_x \sin(\phi_i) + p_y \cos(\phi_i)}{b})$$

The other two equations are:

$$\begin{aligned} p_x \cos(\phi_i) - p_y \sin(\phi_i) - R + r &= a \cos(\theta_{1i}) + b \sin(\theta_{3i}) \cos(\theta_{1i} + \theta_{2i}) \\ p_z &= a \sin(\theta_{1i}) + b \sin(\theta_{3i}) \sin(\theta_{1i} + \theta_{2i}) \end{aligned}$$

The above equations can be re-written as:

$$\begin{aligned} A &= x_1 + x_2 \\ B &= x_3 + x_4 \\ x_1^2 + x_3^2 &= a^2 \\ x_2^2 + x_4^2 &= (b \sin(\theta_{3i}))^2 \end{aligned}$$

where:

$$\begin{aligned} A &= p_x \cos(\phi_i) - p_y \sin(\phi_i) - R + r, & B &= p_z \\ x_1 &= a \cos(\theta_{1i}), & x_3 &= a \sin(\theta_{1i}) \\ x_2 &= b \sin(\theta_{3i}) \cos(\theta_{1i} + \theta_{2i}), & x_2 &= b \sin(\theta_{3i}) \sin(\theta_{1i} + \theta_{2i}) \end{aligned}$$

then we can substitute the equations as:

$$\begin{aligned} (b \sin(\theta_{3i}))^2 &= (A - x_1)^2 + (B - x_3)^2 \\ &= A^2 + B^2 + x_1^2 + x_3^2 - 2(Ax_1 + Bx_3) \end{aligned}$$

since $x_1^2 + x_3^2 = 1$:

$$\begin{aligned} A \cos(\theta_{1i}) + B \sin(\theta_{1i}) &= \frac{A^2 + B^2 + a^2 - (b \sin(\theta_{3i}))^2}{2a} = -M \\ &\rightarrow A \cos(\theta_{1i}) + B \sin(\theta_{1i}) + M = 0 \end{aligned}$$

Then with a simple variable transformation of $t_i = \tan(\theta_{1i}/2)$ the above equation can be solved. In conclusion:

$$\begin{cases} \theta_{3i} = \arccos(\frac{p_x \sin(\phi_i) + p_y \cos(\phi_i)}{b}) \\ \theta_{1i} = 2\arctan(t) \\ \theta_{2i} = \arcsin(\frac{p_z - a\sin(\theta_{1i})}{b\sin(\theta_{3i})}) - \theta_{1i} \end{cases} \tag{2.7}$$

where:

$$\begin{aligned} t &= \frac{-B \pm \sqrt{B^2 + A^2 - M^2}}{M - A} \\ A &= p_x \cos(\phi_i) - p_y \sin(\phi_i) - R + r \\ B &= p_z \\ M &= -\frac{A^2 + B^2 + a^2 - b^2 \sin^2(\theta_{3i})}{2a} \end{aligned}$$

2.2.2 Python Implementation

- INPUTS: position of EE ($\vec{p}$)
- OUTPUTS: angles of the actuated joints ($\vec{\theta}$)

This is how the python code calculates the angle of actuator joints:

1. Takes the geometry of the Delta robot and position of EE as input.
2. $\phi = [0, 120, 240]$ degree by default
3. Initializes a 3 by 3 matrix for θ_{ij} and for each motor begins calculating $\theta_{1i}, \theta_{2i}, \theta_{3i}$ by making use of Equation (2.7)

2.3 Forward Kinematics

forward kinematics is more straight forward because it's a linear system of equations and the answer to it is:

$$\begin{cases} p_x = b\cos(\theta_{3i})\sin(\phi_i) + (R - r + a\cos(\theta_{1i}) + b\sin(\theta_{3i})\cos(\theta_{1i} + \theta_{2i}))\cos(\phi_i) \\ p_y = b\cos(\theta_{3i})\cos(\phi_i) - (R - r + a\cos(\theta_{1i}) + b\sin(\theta_{3i})\cos(\theta_{1i} + \theta_{2i}))\sin(\phi_i) \\ p_z = a\sin(\theta_{1i}) + b\sin(\theta_{3i})\sin(\theta_{1i} + \theta_{2i}) \end{cases} \tag{2.8}$$

2.4 Summary

The *chapter* 2 was about kinematics and how to calculate the Jacobian matrix, Inverse kinematics and Forward kinematics. In other words:

- Given the coordination of EE, the angle of actuated joints can be calculated.
- Given the angle of actuated joints, coordination of EE can be calculated.

- Given the velocity of the EE, angular velocity of the actuated joints can be calculated
- Given the angular velocity of actuated joints, the velocity of the EE can be calculated

Chapter 3

MOTION PLANNING - POINT TO POINT MOVEMENT

In this chapter of the report, the topic is algorithms and ways to move the EE from point A to point B in space with a motion as smooth as possible. meaning the velocity and acceleration profiles need to be continuous and jerk is bounded. Some of the different methods discussed are polynomial interpolation, Trapezoidal and S-curves.

3.1 3-4-5 Polynomial Interpolation

In order to represent each joint motion, a fifth-order polynomial $s(\tau)$ is used here. meaning that, if θ_0 and θ_1 and the corresponding time instants of t_0 and t_1 are given, the path between them can be interpolated by use of a fifth order polynomial that outputs a predicted $\theta_{pred}(t)$ that $t \in [t_0,\ t_1]$

3.1.1 Theory

The polynomial is shown in the following form:

$$s(\tau) = a\tau^5 + b\tau^4 + c\tau^3 + d\tau^2 + e\tau + f$$

such that

$$0 \leq s \leq 1 \quad 0 \leq \tau \leq 1$$

and

$$\tau = \frac{t}{T}$$

we will thus aim at a normal polynomial that, upon scaling both its argument and the polynomial itself, will allow us to represent each of the joint variables

θ_j throughout its range of motion, so that

$$\theta(t) = \theta_I + (\theta_F - \theta_I)s(\tau) \tag{3.1}$$

$$\dot{\theta}(t) = (\theta_F - \theta_I)\frac{1}{T}s'(\tau) \tag{3.2}$$

$$\ddot{\theta}(t) = \frac{1}{T^2}(\theta_F - \theta_I)s''(\tau) \tag{3.3}$$

$$\dddot{\theta}(t) = \frac{1}{T^3}(\theta_F - \theta_I)s'''(\tau) \tag{3.4}$$

The boundary conditions are as followed

$$s(0) = 0, \quad s'(0) = 0, \quad s''(0) = 0, s(1) = 1, \quad s'(1) = 0, \quad s''(1) = 0$$

Solving the equation for the coefficients results in the following:

$$s(\tau) = 6\tau^5 - 15\tau^4 + 10\tau^3 \tag{3.5}$$

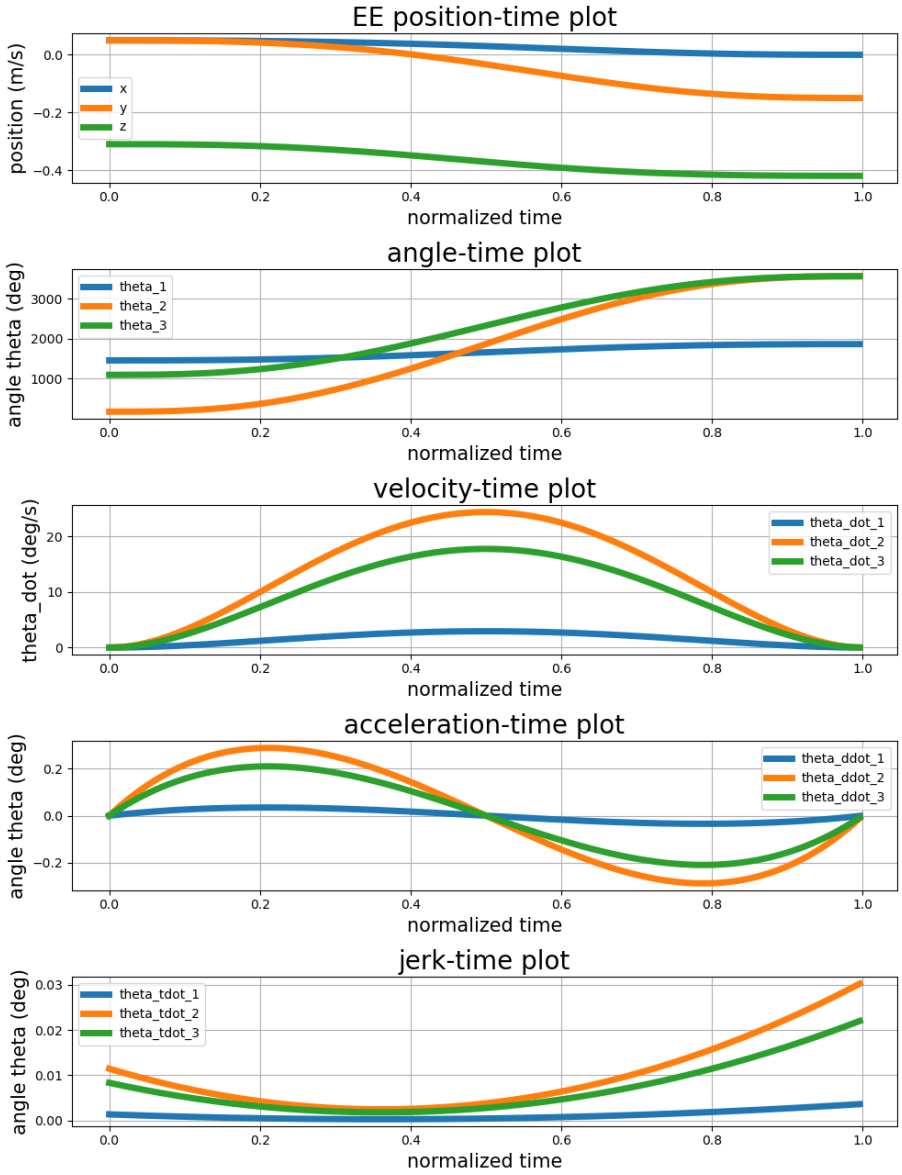
3.1.2 Python Implementation

- INPUTS: Initial and final position of EE
- OUTPUTS: Position, velocity, acceleration and jerk profile between initial and final points of movement

The algorithm for implementing 3-4-5 polynomial interpolation:

1. Taking the initial and final position of EE as input. ($\vec{p}_{initial}$ and $\vec{p}_{final}$)
2. calculating the inverse kinematics for initial and final position results in initial and final angles of actuated joints ($\theta_{initial}$ and θ_{final})
3. interpolate the initial and final actuated angles with the help of 3-4-5 polynomial, using Equations (3.5) and (3.1). This results in the polynomial $s(\tau)$ and $\theta(t)$
4. calculate the first, second and third derivatives of the predicted path for velocity, acceleration and jerk plot profiles. This results in $\dot{\theta}$, $\ddot{\theta}$, $\dddot{\theta}(t)$

Figure 3.1: 3-4-5 interpolation



Interpolation of the angle of motors,
Note that this Delta robot has a gear-ratio = 50 and offset= 47.2 *degree*

3.2 4-5-6-7 Polynomial Interpolation

The problem with 3-4-5 polynomial is that the jerk at the boundary can't be set to zero, for this problem we use a higher degree of polynomial, namely, seventh-degree polynomial.

3.2.1 Theory

with the steps similar to the last section and the added condition that $s'''(0) = 0, \quad s'''(1) = 0$ we solve the system of equation generated from the boundary conditions. The resultant polynomial is as followed:

$$s(\tau) = -20\tau^7 + 70\tau^6 - 84\tau^5 + 35\tau^4 \tag{3.6}$$

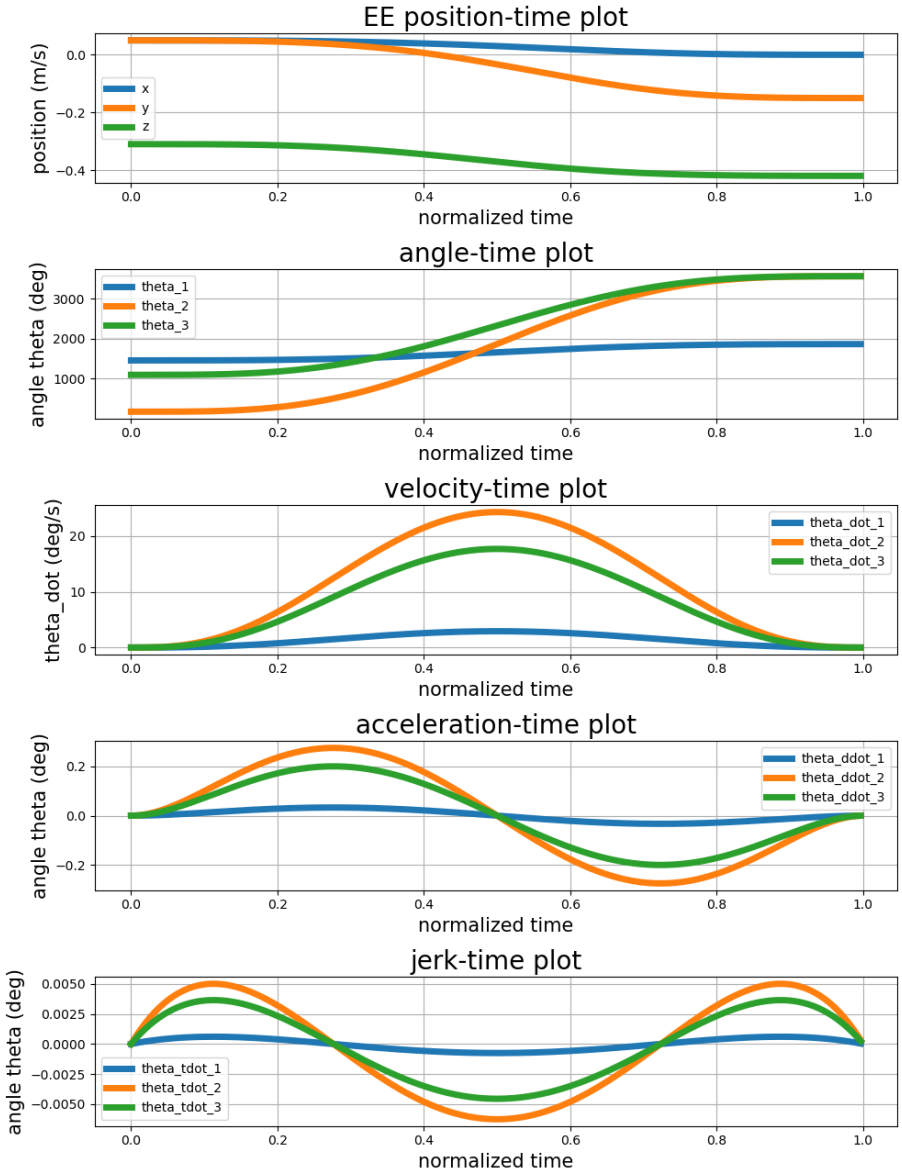
3.2.2 Python Implementation

- INPUTS: Initial and final position of EE
- OUTPUTS: Position, velocity, acceleration and jerk profile between initial and final points of movement

the algorithm is the same as *part* 3.1.2

1. Taking the initial and final position of EE as input. ($\vec{p}_{initial}$ and $\vec{p}_{final}$)
2. calculating the inverse kinematics for initial and final position results in initial and final angles of actuated joints ($\theta_{initial}$ and θ_{final})
3. interpolate the initial and final actuated angles with the help of 3-4-5 polynomial, using Equations (3.6) and (3.1). This results in the polynomial $s(\tau)$ and $\theta(t)$
4. calculate the first, second and third derivatives of the predicted path for velocity, acceleration and jerk plot profiles. This results in $\dot{\theta}$, $\ddot{\theta}$, $\dddot{\theta}\,(t)$

Figure 3.2: 4-5-6-7 interpolation



Interpolation of the angle of motors,
Note that this Delta robot has a gear-ratio = 50 and offset= 47.2 *degree*

3.3 Trapezoidal

In trapezoidal point to point movement the velocity has three phases as described the following subsection, from start of *zero* to v_{max} and then back to *zero* again.

3.3.1 Theory

The velocity profile mathematical expression is as followed:

$$v(t) = \begin{cases} at & for \quad 0 \leq t \leq T_a \\ v_{max} & for \quad T_a \leq t \leq T - T_a \\ a(T-t) & for \quad T - T_a \leq t \leq T \end{cases} \tag{3.7}$$

and since, $\dot{p} = v \rightarrow p = \int_{p_{intial}}^{p_{final}} v\, dt$, so it results:

$$p(t) = \begin{cases} at^2/2 + p_0 & for \quad 0 \leq t \leq T_a \\ aT_a^2/2 + v_{max}(t - T_a) + p_0 & for \quad T_a \leq t \leq T - T_a \\ aT_a^2/2 + v_{max}(T - 2T_a) + a(T - T_a)^2/2 - aT(T - T_a) & \\ \qquad + aTt - at^2/2 + p_0 & for \quad T - T_a \leq t \leq T \end{cases} \tag{3.8}$$

acceleration a can easily be calculated as: $a = V_{max}/T_a$. Also the relation between T and the other parameters is as followed

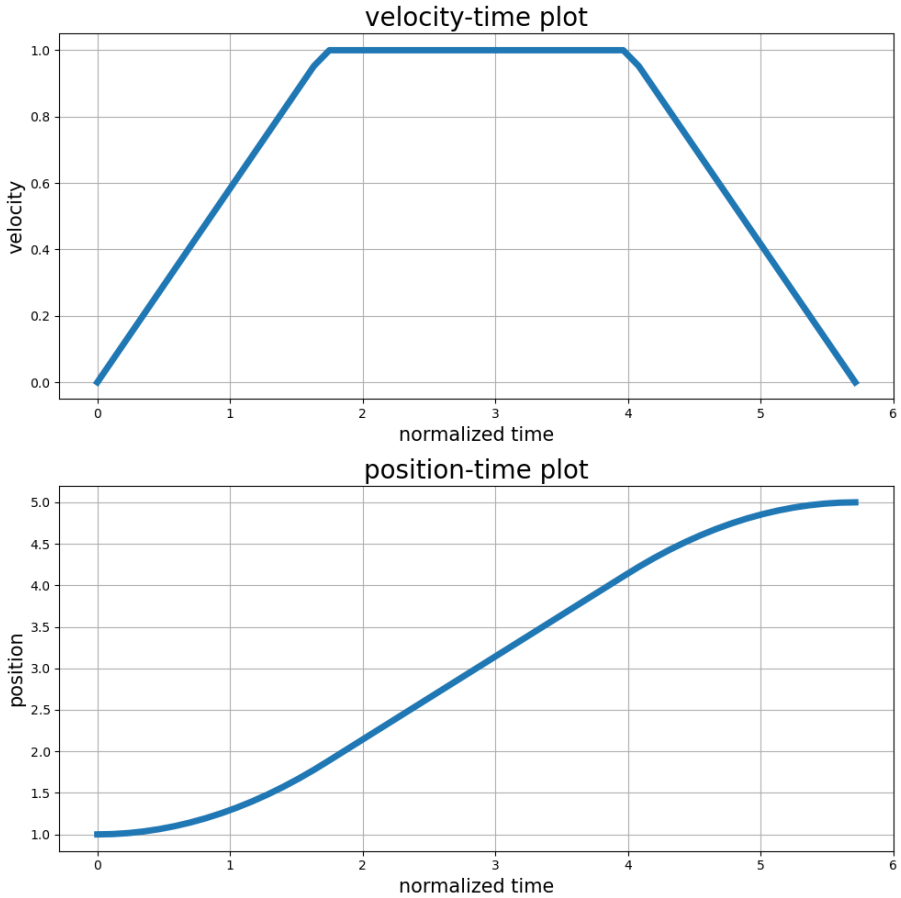
$$T = \frac{(p_{final} - p_{initial})}{v_{max}(1-k)} \qquad that \quad k = \frac{T_a}{T} \tag{3.9}$$

The above relation results from $p(T) = p_{final}$ that $p(T)$ comes from Equation (3.8)

3.3.2 Python Implementation

The Trapezoidal diagram is pretty simple to plot. but the important part of this diagram is that it's discrete. so at the points of changing acceleration, it might behave differently than of a continuous plot.

Figure 3.3: Point-to-Point Trapezoidal



The 3 phases of velocity, in the Trapezoidal method

algorithm used for programming this discrete plot includes the following steps:

1. Takes $p_{initial}$, p_{final}, v_{max} as input and calculates the needed duration for the whole motion
2. With T and k as input, calculates T_a
3. discretize time profile and calculating position and velocity profiles from the Equations 3.7 and 3.8.

3.4 S-Curve

As a concept, S-curve is the better version of trapezoidal. strictly speaking, it has seven phases instead of three. as explained below in mathematical terms:

$$a(t) = \begin{cases} Jt & for \quad T_0 \leq t \leq T_1 \\ a_{max} & for \quad T_1 \leq t \leq T_2 \\ J(T_3 - t) & for \quad T_2 \leq t \leq T_3 \\ 0 & for \quad T_3 \leq t \leq T_4 \\ -J(t - T_4) & for \quad T_4 \leq t \leq T_5 \\ -a_{max} & for \quad T_5 \leq t \leq T_6 \\ -J(T_7 - t) & for \quad T_6 \leq t \leq T_7 \end{cases} \tag{3.10}$$

And the rest is simple algebra with $a(t) = \dot{v}(t) = \ddot{p}(t)$.

3.5 Summary

In chapter 3, the point-to-point movement of the Delta robot End-Effector has been discussed. the methods mentioned are as followed:

- 3-4-5 Polynomial movement is not suitable for real-life modeling because of unbounded jerk at the beginning of the motion.
- 4-5-6-7 is suitable for real-life modeling because of continuous acceleration and bounded jerk. (this has been tested by the Delta robot)
- Trapezoidal method is not acceptable as a solution because of discontinuity in acceleration.
- S-curve is a valid solution because it solves the problem of discontinuity in the Trapezoidal method

Chapter 4

MOTION PLANNING - MULTI-POINT MOVEMENT

THE fourth chapter of this report is about moving the EE through a sequence of predefined points in space. Last chapter discussed the way of moving between two points in space, so the EE can move in a straight line by the methods mentioned before. But what if, the task is moving the EE through a curve in space? This chapter is about answering this exact question. various methods have been discussed such as polynomial interpolation, splines, Trapezoidal and Jacobian methods. Like the last chapter some of this methods are not suited for real-life use, but they give us a better understanding of how the End-Effector behaves in different situations.

4.1 Cubic Spline

One way of interpolating a path of $n+1$ points in space, is by a polynomial of degree n. This way might work for 3 or 4 points, but for higher degree polynomials it can be very computationally expensive. Another way which is much better in terms of computational power needed, is using n polynomials of degree p that $p \ll n$. The overall function $s(t)$ defined in this manner is called spline of degree p. The value of p is chosen according to the desired degree of continuity of the spline for instance, in order to obtain the continuity of velocity and acceleration at the same time instants t_k, where the transition between two consecutive segments occurs, it is possible to assume a polynomial of degree 3

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

The overall function is given by:

$$\begin{gathered} s(t) = q_k(t), t \in [t_k, t_(k+1)], k = 0, ..., n-1 \\ \text{that} \\ q_k(t) = a_{k0} + a_{k1}(t - t_k) + a_{k2}(t - t_k)^2 + a_{k3}(t - t_k)^3 \end{gathered} \tag{4.1}$$

According to 4.1, total of 4n coefficients are needed for this to work.

- $2n$ conditions for the interpolation of the given points.
- $n-1$ conditions for the continuity of the velocity at transition points.
- $n-1$ conditions for the continuity of the acceleration at transition points

These conditions add up to $4n-2$, there are still two conditions missing. These two conditions can be chosen from these options:

1. The initial and final velocity $\dot{s}(t_0) = v_0,\ \dot{s}(t_n) = v_n$
2. The initial and final acceleration $\ddot{s}(t_0) = a_0,\ \ddot{s}(t_n) = a_n$
3. The conditions $\dot{s}(t_0) = \dot{s}(t_n),\ \ddot{s}(t_0) = \ddot{s}(t_n)$
4. The continuity of the jerk at time instant $t_1,\ t_{n-1}$

4.1.1 Computation of the coefficient for assigned initial and final velocities

For the definition of a trajectory for an automatic machine the condition on the continuity of the velocity profile is of fundamental importance. for this reason, a typical choice for the computation of the spline is to assign the initial and final velocities $v_0,\ v_f$.

Theory

Given the points $(t_k,\ q_k)$ for $k = 0, ..., n$ the final goal is to determine the function shown in Equation (4.1) with the conditions as explained below:

$$\begin{aligned}
q_k(t_k) &= q_k, & q_k(t_{k+1}) &= q_{k+1} & k &= 0, 1, ..., n-1 \\
\dot{q_k}(t_{k+1}) &= \dot{q_{k+1}}(t_{k+1}) = v_{k+1} & \ddot{q_k}(t_{k+1}) &= \ddot{q_{k+1}}(t_{k+1}) & k &= 0, 1, ..., n-2 \\
\dot{q_0}(t_0) &= v_0 & \dot{q_{n+1}}(t_n) &= v_n
\end{aligned}$$

The coefficients $a_{k,i}$ can be found with the following formula

$$\begin{cases}
a_{k,0} = q_k \\
a_{k,1} = v_k \\
a_{k,2} = \frac{1}{T_k}\left[\frac{3(q_{k+1}-q_k)}{T_k} - 2v_k - v_{k+1}\right] \\
a_{k,3} = \frac{1}{T_k^2}\left[\frac{2(-q_{k+1}+q_k)}{T_k} + v_k + v_{k+1}\right]
\end{cases} \tag{4.2}$$

if $T_k = t_{k+1} - t_k$ for $k = 0, 1, ..., n-2$. In this stage all of the polynomials which interpolate the path of EE are known (Note that q can be in $x,\ y$ *and* z direction). With the polynomials known, the velocity profile can be calculated with the below equation:

$$Av = c \tag{4.3}$$

That v is a vector of velocities at points $\vec{q}$

$$\vec{v} = \begin{bmatrix} v_0 \\ v_1 \\ \vdots \\ v_{n-2} \\ v_{n-1} \end{bmatrix}$$

$$A = \begin{bmatrix} 2(T_0 + T_1) & T_0 & 0 & \dots & 0 \\ T_2 & 2(T_1 + T_2) & T_1 & 0 & \vdots \\ 0 & \ddots & & & 0 \\ \vdots & & T_{n-2} & 2(T_{n-3} + T_{n-2}) & T_{n-3} \\ 0 & \dots & 0 & T_{n-1} & 2(T_{n-2} + T_{n-1}) \end{bmatrix}$$

$$c = \begin{bmatrix} \frac{3}{T_0 T_1}[T_0^2(q_2 - q_1) + T_1^2(q_1 - q_0)] - T_1 v_0 \\ \frac{3}{T_1 T_2}[T_1^2(q_3 - q_2) + T_2^2(q_2 - q_1)] \\ \vdots \\ \frac{3}{T_{n-3} T_{n-2}}[T_{n-3}^2(q_{n-1} - q_{n-2}) + T_{n-2}^2(q_{n-2} - q_{n-3})] \\ \frac{3}{T_{n-2} T_{n-1}}[T_{n-2}^2(q_n - q_{n-1}) + T_{n-1}^2(q_{n-1} - q_{n-2})] - T_{n-2} v_n \end{bmatrix}$$

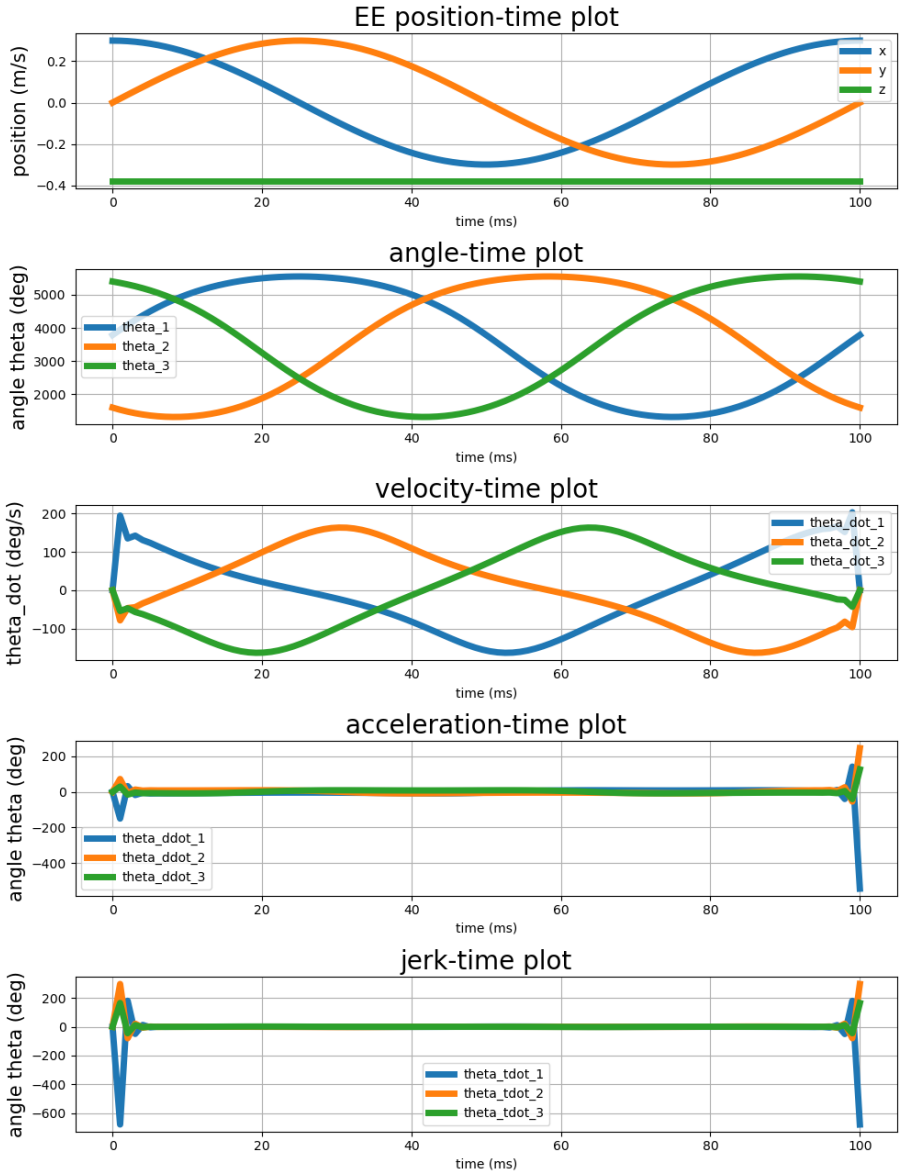
So with that all of the polynomial parameters $a_{ij}, for i = 0, \dots, n-1, j = 0, 1, 2, 3$ can be calculated.

Python Implementation

Cubic-spline code, follows the steps explained below:

1. Generates the position movement for EE
2. Calculates the IK for generated positions, to find θ_i
3. Interpolates θ_i with cubic-spline method. Outputs the polynomial coefficients of the interpolation.
4. uses the resulted coefficients in the previous step to re-calculate θ_i in a more precise time-step
5. uses the resulted coefficients in step 3 to find angular velocity profile of the motors
6. uses the resulted coefficients in step 3 to find angular acceleration profile of the motors
7. uses the resulted coefficients in step 3 to find angular jerk profile of the motors

Figure 4.1: Cubic-spline method



In cubic spline the problem is the high acceleration change (discontinuity in velocity profile) in the boundary conditions, in other words, there is an unbounded jerk, at the start and end of motion

4.1.2 Computation of the coefficient for assigned initial and final velocities and accelerations

The difference between This section and the previous section is that we will have two more boundary conditions. meaning:

$$v(0) = v_i,\ v(1) = v_f,\ a(0) = a_i,\ a(1) = a_f$$

If there are $n-1$ points to be interpolated, then $n-2$ polynomial are needed to interpolate the via points, that results in $4n-8$ unknown parameters (polynomial parameters $a_{k0},\ a_{k1},\ a_{k2},\ a_{k3}$) The boundary conditions are:

- position at each point. ($2n-2$ equations)
- continuity of velocity. ($n-2$ equations)
- continuity of acceleration ($n-2$ equations)
- velocity and acceleration at the beginning and end of the motion (4 equations)

this adds up to $4n-2$ equations, which does not make a linear system of equations with the $4n-8$ unknowns. The way around this problem is to add two points, which give us 10 more unknowns (their position and polynomial parameters) and 4 more continuity equations. In other words, with adding two more points in between the path, number of unknowns and equations match. The rest of it is like before:

Theory

The given points are as followed:

$$q = [q_0,\ q_2,\ q_3,\ \ldots,\ q_{n-3},\ q_{n-2},\ q_n]^T$$

with the respective time instants:

$$t = [t_0,\ t_2,\ t_3,\ \ldots,\ t_{n-3},\ t_{n-2},\ t_n]^T$$

After adding the in-between points (as explained before) of q_1 and q_{n-1} the new set of points will be:

$$q_{new} = [q_0,\ \bar{q}_1,\ q_2,\ q_3,\ \ldots,\ q_{n-3},\ q_{n-2},\ \bar{q}_{n-1},\ q_n]^T$$

at the time instants:

$$t_{new} = [t_0,\ \bar{t}_1,\ t_2,\ t_3,\ \ldots,\ t_{n-3},\ t_{n-2},\ \bar{t}_{n-1},\ t_n]^T$$

that $\bar{t}_1 = \frac{t_0+t_2}{2}$ and $\bar{t}_{n-1} = \frac{t_{n-2}+t_n}{2}$

Since $\bar{q}_1$ and $\bar{q}_{n-1}$ are unknown, it is important to write them as a function of the other parameters. (position, velocity and acceleration at the first/last points ($q_0,\ q_n,\ v_0,\ v_n\ a_0,\ a_n$) and velocity at the new points, ($\omega_1,\ \omega_{n-1}$))

$$q_1 = q_0 + T_0 v_0 + T_0^2 a_0/3 + T_0^2 \omega_1/6 \tag{4.4}$$

$$q_{n-1} = q_n - T_{n-1} v_n + T_{n-1}^2 \omega_n/3 + T_{n-1}^2 a_{n-1}/6 \tag{4.5}$$

like the Equation (4.3), it can be said:

$$A\omega = c \tag{4.6}$$

that

$$\omega = [\omega_0,\ \omega_1,\ \ldots,\ \omega_{n-1},\ \omega_n]^T$$

$$A = \begin{bmatrix} 2T_1 + T_0\left(3 + \frac{T_0}{T_1}\right) & T_1 & 0 & \dots & & & 0 \\ T_1 - \frac{T_0^2}{T_1} & 2(T_1 + T_2) & T_2 & & & & \vdots \\ 0 & T_2 & 2(T_2 + T_3) & T_3 & & & \\ \vdots & & & \vdots & & & 0 \\ & & & & T_{n-3} & 2(T_{n-3} + T_{n-2}) & T_{n-2} - \frac{T_{n-1}^2}{T_{n-2}} \\ 0 & \dots & & & T_{n-2} & & 2T_{n-2} + T_{n-1}\left(3 + \frac{T_{n-1}}{T_{n-2}}\right) \end{bmatrix}$$

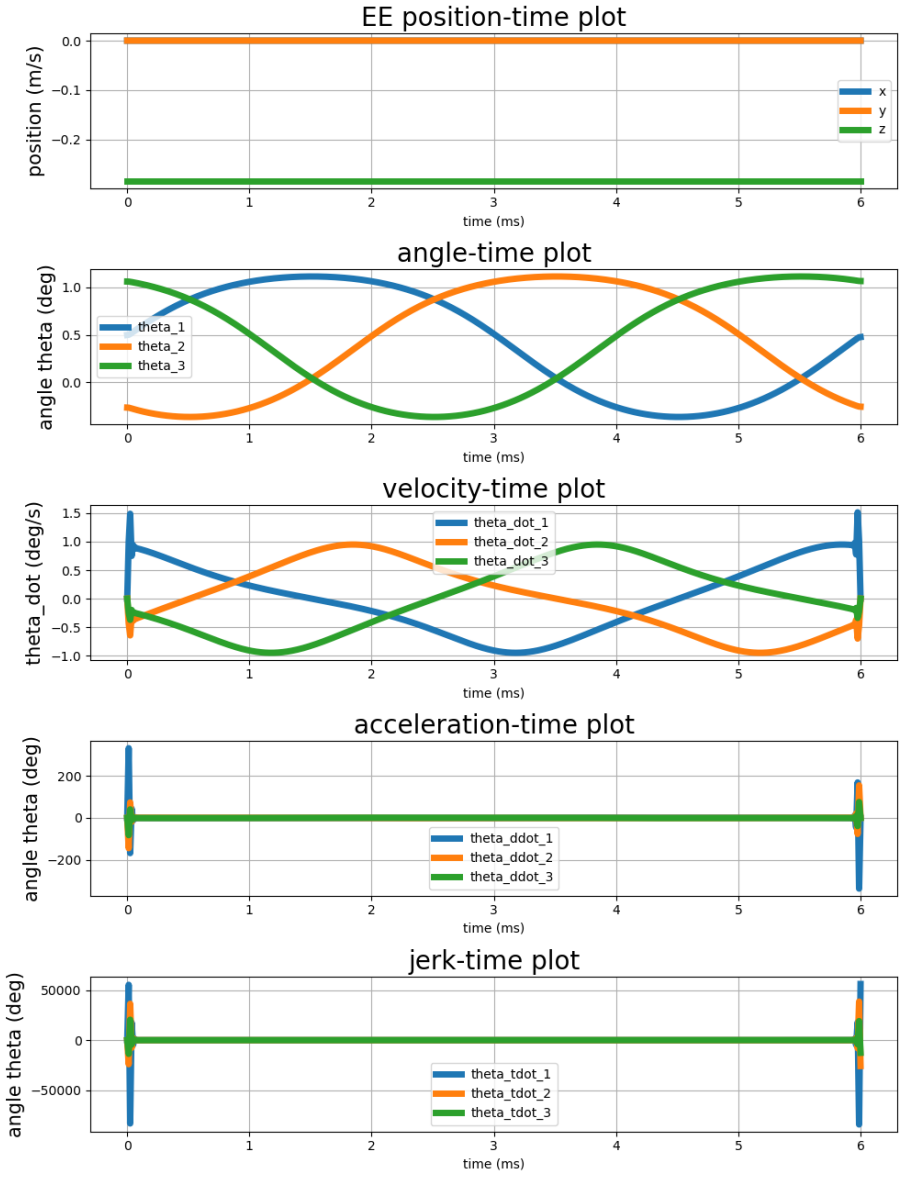
$$c = 6 \begin{bmatrix} \frac{q_2 - q_0}{T_1} - v_0\left(1 + \frac{T_0}{T_1}\right) - a_0\left(0.5 + \frac{T_0}{3T_1}\right)T_0 \\ \frac{q_3 - q_2}{T_2} - \frac{q_2 - q_0}{T_1} + v_0\frac{T_0}{T_1} + a_0\frac{T_0^2}{3T_1} \\ \frac{q_4 - q_3}{T_3} - \frac{q_3 - q_2}{T_2} \\ \vdots \\ \frac{q_{n-2} - q_{n-3}}{T_{n-3}} - \frac{q_{n-3} - q_{n-4}}{T_{n-4}} \\ \frac{q_n - q_{n-2}}{T_{n-2}} - \frac{q_{n-2} - q_{n-3}}{T_{n-3}} - v_n\frac{T_{n-1}}{T_{n-2}} + a_n\frac{T_{n-1}^2}{3T_{n-2}} \\ \frac{q_{n-2} - q_n}{T_{n-2}} + v_n\left(1 + \frac{T_{n-1}}{T_{n-2}}\right) - a_n\left(0.5 + \frac{T_{n-1}}{3T_{n-2}}\right)T_{n-1} \end{bmatrix}$$

So with that all of the polynomial parameters $a_{ij},\ for\ i = 0,\ \dots,\ n-1,\ j = 0,\ 1,\ 2,\ 3$ can be calculated.

Python Implementation

The algorithm is same as section 4.1.2.

Figure 4.2: Cubic-spline method

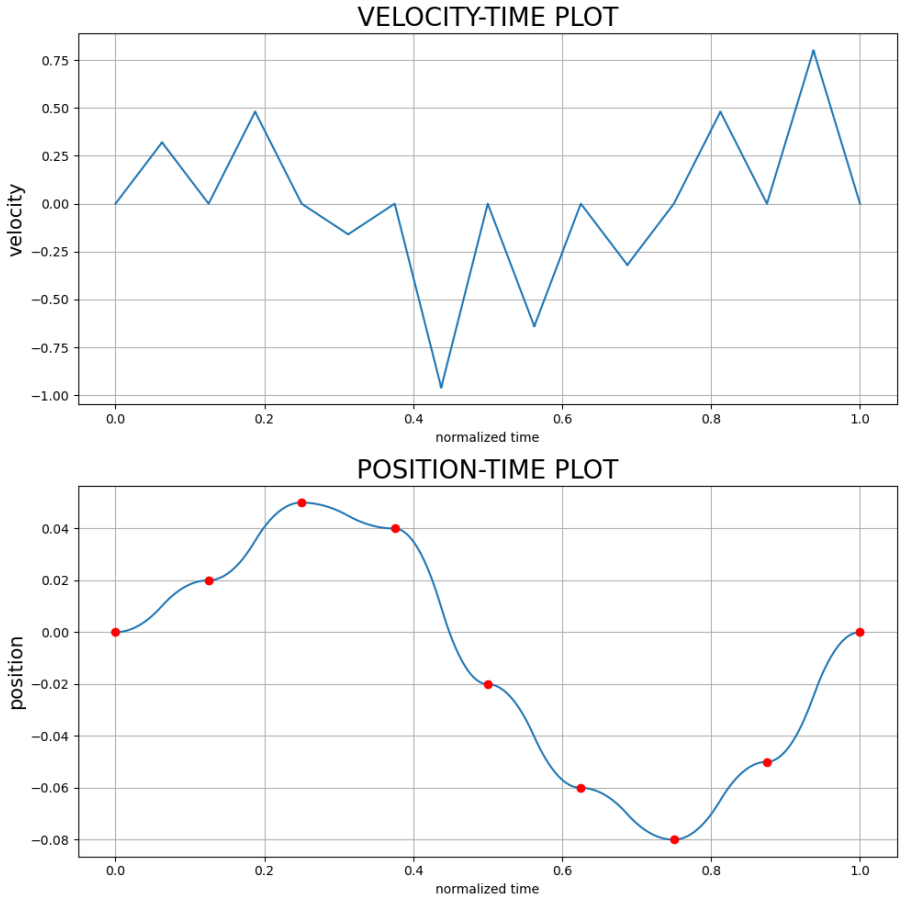


Initial and final velocities and accelerations set to zero. even though this may work great in theory, it doesn't do so well in practice because all it does is to set the first acceleration point to zero but the second acceleration point is still far away from being smooth

4.2 Trapezoidal - Through a Sequence of Points

If the method from section 3.3 is used directly for multiple points, the velocity becomes zero between each two points, which is not acceptable. (see Figure 4.3)

Figure 4.3: Trapezoidal Through a Sequence of points



The constant back and forth in velocity profile is something that is extremely unrealistic and unoptimized. (k = 0.5)

There is a better way of constructing the velocity profile, than it hitting zero in each point of movement. This new method is built on the previous one. It is explained in the following section.

4.2.1 Trapezoidal - Through a Sequence of Points - Modified Velocity Profile

Another way of using Trapezoidal method is not to use section 3.3 directly, but rather modify the velocity profile to be more smooth than the method used in section 4.2. The Algorithm for making the velocity profile smoother is explained in the following steps:

Step 1:

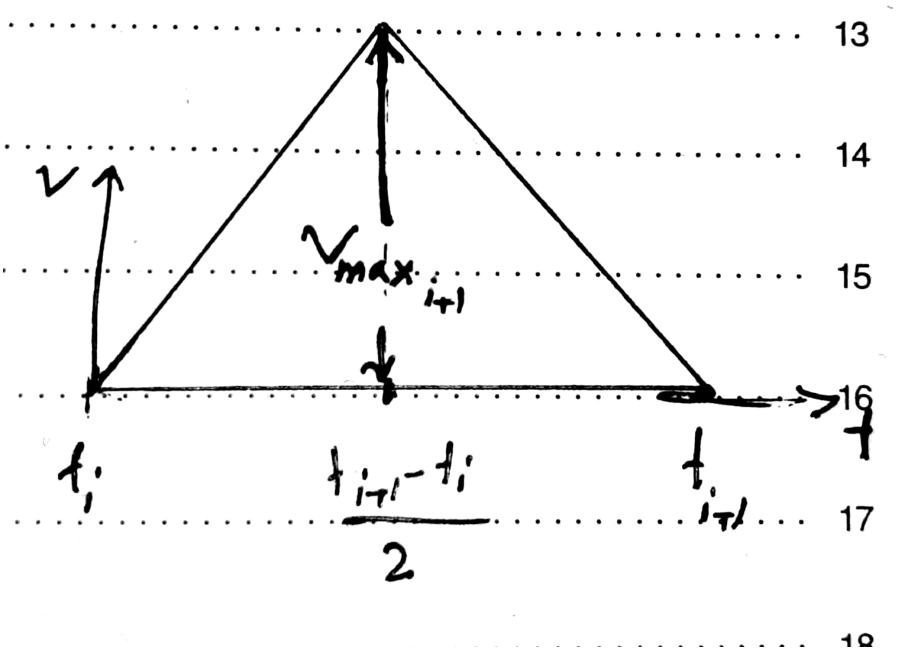
Given the positions of EE as the vector $\vec{p} = [p_0,\ \ldots,\ p_n]^T$ we define the corresponding normalized time vector as $\vec{t} = [t_0,\ t_1,\ \ldots,\ t_{n-1},\ t_n]$ that

$$\begin{aligned} t_0 &= 0 \\ t_n &= T = 1 \\ t_{i+1} - t_i &= \frac{T}{n+1},\ \text{for } i = 0,\ \ldots,\ n-1 \end{aligned}$$

Step 2:

Calculating v_{max_i} by considering $k = 0.5$ (if you're not familiar with the term k see section 3.3)

Figure 4.4: Trapezoidal with $k = 0.5$



v_{max_i} calculation (k = 0.5)

Considering Figure (4.4) it can be said that:

$$v_{max_i} = \frac{2(p_{i+1} - p_i)}{t_{i+1} - t_i} \quad \text{for} \quad i = 0,\ \ldots,\ n-1$$

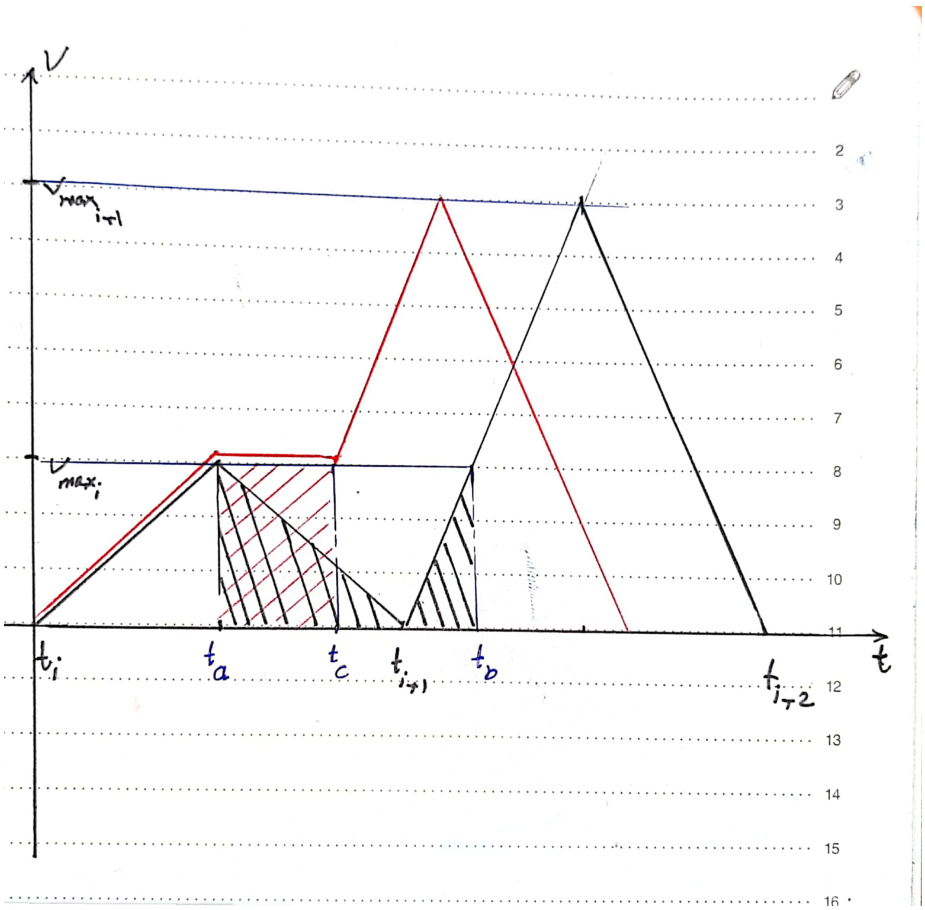
and so:

$$\vec{v}_{max} = [v_{max_0},\ \ldots,\ v_{max_{n-1}}] \tag{4.7}$$

Step 3:

Consider Figure (4.3) as reference velocity-time profile. Define v' and t' as the modified velocity and time profile respectively.

Figure 4.5: Modified Trapezoidal



Modified velocity profile is red and the old velocity profile is black. The area under red diagram from t_a to t_c is equal to area under black diagram from t_a to t_b that $t_c = \frac{t_a+t_b}{2}$

Assume v_{max_i} and $v_{max_{i+1}}$ both are greater than zero and $v_{max_{i+1}} > v_{max_i}$. from Figure (4.5) it can be seen that

$$\forall v \in [v_{max_i},\ v_{max_{i+1}}], i \in [0,\ n-1] : \big(\text{if } v_{max_i}.v_{max_{i+1}} < 0 \rightarrow v' = v\big) \tag{4.8}$$

and

$$\forall v \in [v_{max_i},\ v_{max_{i+1}}], i \in [0,\ n-1] : \text{if } v_{max_i}.v_{max_{i+1}} > 0 \rightarrow$$
$$\begin{cases} v'(t') = v(t) & \text{for} \quad t' = t \in [t_i,\ t_a] \\ v'(t') = v_{max_i} & \text{for} \quad t' \in [t_a,\ t_c] \\ v'(t') = v(t) & \text{for} \quad t' \in [t_c,\ t_{i-2} - t_c] \quad \text{and} \quad t \in [t_b,\ t_{i-2}] \\ v'(t') = 0 & \text{for} \quad t' \in [t_{i+2} - t_c,\ t_{i-2}] \end{cases} \tag{4.9}$$
$$(\text{where} \quad t_c = \frac{t_a + t_b}{2})$$

In general if $v_{max_i}.v_{max_{i+1}} > 0$ then for $t' \in [t_a,\ t_c]$, it can be said:

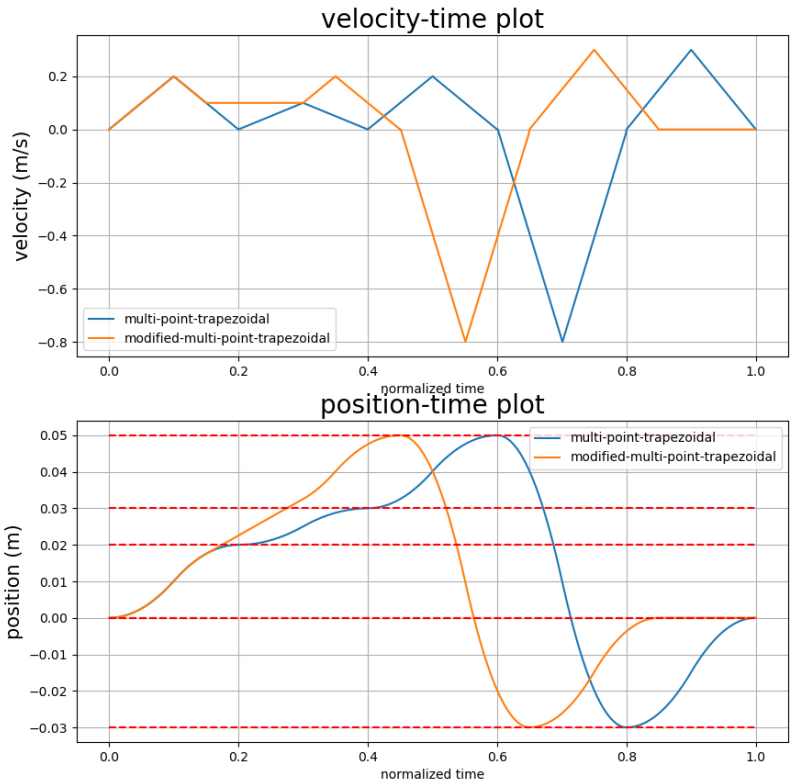
$$v'(t') = sign(v_{max_i}).\min(|v_{max_i}|,\ |v_{max_{i+1}}|)$$

Python Implementation

Following steps 1 through 3 in the theory section the python implementation steps can be listed like below:

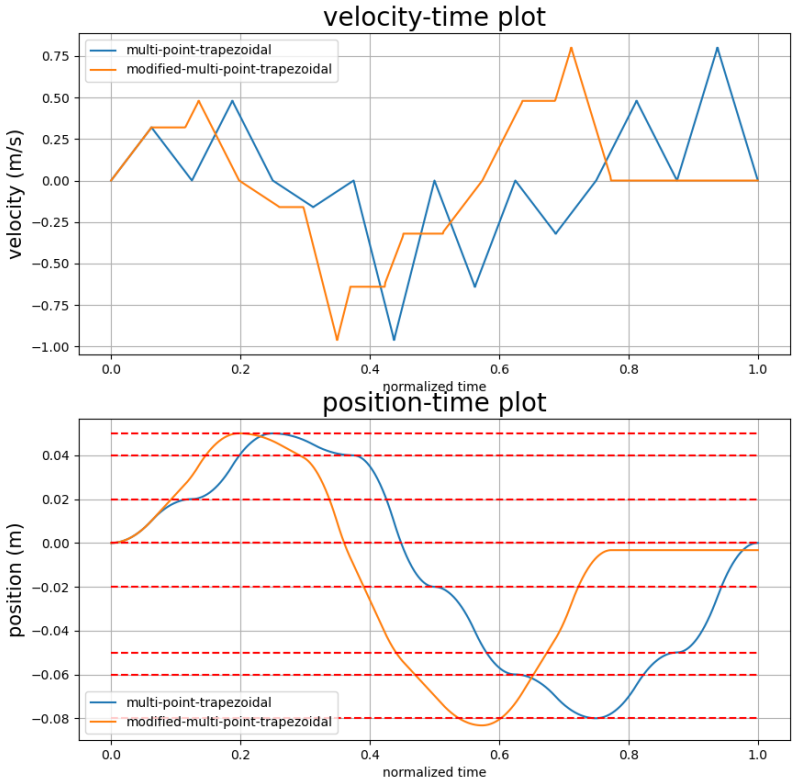
1. Take *position* vector as input ($\vec{p} = [p_0,\ \ldots,\ p_n]$)
2. Initialize *time* instance vector with equal intervals based on *position* vector. ($\vec{t} = [t_0,\ \ldots,\ t_n]$, that $t_n = T$ and $t_{i+1} - t_i = T/(n+1)$)
3. Calculate $\vec{v}_{max}$ vector from Equation (4.7)
4. construct the new time profile ($\vec{t'}$) based on time vector ($\vec{t}$) as $\vec{t'} = [t'_0,\ \ldots,\ t'_{m.n+1}]$ that m is the segmentation number. It should be that ($t'_{i+1} - t'_i = minimum\ timestep$).
5. Construct velocity profile between each two consecutive points according to Equation (3.7) that $k = 0.5$.
6. Modify the velocity profile according to part 4.2.1

Figure 4.6: Modified Trapezoidal



The red dashed line is the places that the EE is supposed to cross

Figure 4.7: Modified Trapezoidal



The red dashed line is the places that the EE is supposed to cross

Of course this method is not prone to error and the more points introduced, the more extreme this error will get. The reason for this error is that time profile is not actually continuous in the implementation, so the discretization causes small errors in-between each point, and the more the points get, this errors add up and mainly will be uncontrollable after a while.

Review

This method is not actually built for solving real-world problems, but rather to give us insight into how does the S-curve method works. The reason it won't work is that there are multiple discontinuities in the acceleration profile, which produces unbounded jerk at the points of transition. See section 4.3 for more information of how to fix the problem of multi-point trapezoidal.

4.3 S-Curve - Through a Sequence of Points

This section is a continuation of section 4.2.1. According to section 3.4 S-curve has 7 phases, but for the sake of simplicity, it can be changed to 6 phases as t_3 and t_4.

4.3.1 Acceleration, Velocity and Position Profiles

In the 6 phases are explained as the following cases:

Acceleration

Acceleration is the first part of the kinematics that's defined, after that velocity and position profiles are integrated from the acceleration profile.

$$a = \begin{cases} Jt & for \quad t_0 \leq t \leq t_1 \\ a_{max} & for \quad t_1 \leq t \leq t_2 \\ J(\frac{T}{2} - t) & for \quad t_2 \leq t \leq t_4 \\ -a_{max} & for \quad t_4 \leq t \leq t_5 \\ J(t - T) & for \quad t_5 \leq t \leq t_6 \end{cases} \tag{4.10}$$

Velocity

Since velocity is calculated as $v = \int_{t_i}^{t_f} a dt$, hence the velocity profile is integrated from Equation (4.10):

$$v = \begin{cases} \frac{Jt^2}{2} & for \quad t_0 \leq t \leq t_1 \\ \frac{JT^2}{72} + a_{max}(t - t_1) & for \quad t_1 \leq t \leq t_2 \\ \frac{4JT^2}{72} - \frac{J(T-t)^2}{2} & for \quad t_2 \leq t \leq t_4 \\ v_2 - a_{max}(t - t_4) & for \quad t_4 \leq t \leq t_5 \\ \frac{J(T-t)^2}{2} & for \quad t_5 \leq t \leq t_6 \end{cases} \tag{4.11}$$

that:

$$\begin{aligned} v_0 &= 0 \\ v_1 &= \frac{JT^2}{72} \\ v_2 &= \frac{JT^2}{72} + a_{max}\frac{T}{6} \\ v_3 &= \frac{4JT^2}{72} \\ v_4 &= v_2 \\ v_5 &= v_1 \\ v_6 &= 0 \end{aligned}$$

Position

For calculating position profile all is need to be done is integrating Equation (4.11). but since that is not needed for the implementation it won't be discussed in this part of the report. Though the same cannot be said about the equation $p(T) = p_f$ as it is needed for calculating the relation between $p_i,\ p_f,\ T\ and\ J$:

$$\begin{aligned} p_f - p_i &= \int_0^{T/6} \frac{Jt^2}{2} dt \\ &+ \int_{T/6}^{T/3} [\frac{JT^2}{72} + a_{max}(t - \frac{T}{6})] dt \\ &+ \int_{T/3}^{2T/3} [\frac{JT^2}{18} - \frac{J(\frac{T}{2} - t)^2}{2}] dt \\ &+ \int_{2T/3}^{5T/6} [v_2 - a_{max}(t - t_4)] dt \\ &+ \int_{5T/6}^{T} \frac{J(T-t)^2}{2} dt \\ &= 0.050926 JT^3 \end{aligned}$$

Summary

As input of this algorithm we have normalized time intervals plus initial and final positions. In other words $a_{max},\ v_{max}\ and\ J$ are unknown. Finding Three unknowns requires three equations, so the below equations are introduced as such:

$$\begin{aligned} a(t_1) &= a_{max} \\ v(t_3) &= v_{max} \\ p_f - p_i &= \int_0^T v(t) dt \end{aligned}$$

With the previously discussed relations and calculations, the three above equations simplify to the following forms:

$$a_{max} = \frac{JT}{6} \tag{4.12}$$

$$v_{max} = \frac{JT^2}{18} \tag{4.13}$$

$$J = \frac{p_f - p_i}{0.050926 T^3} \tag{4.14}$$

4.4 Jacobian

As previously debated over what the Jacobian matrix does in Section (2.1), in this part the method of using it for trajectory planning is going to be studied.

4.4.1 Algorithm Steps

Referring to Figure (2.1), as a reminder of what has already been said:

- Theta vector is the angles of actuated joints ($\vec{\theta} = \overrightarrow{\theta_{1i}} = \begin{bmatrix} \theta_{11} \\ \theta_{12} \\ \theta_{13} \end{bmatrix}$)
- Position vector refers to the position of End-Effector relative to the global origin ($\vec{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$)
- Fro one point in space-time at ($x_0,\ y_0,\ z_0,\ t_0$) we can relate the velocity of EE to angular velocity of actuated joints. with the following formula:

  $$J_\theta \overrightarrow{\dot{\theta}} = J_P \overrightarrow{v}$$

  (For complete explanation refer to section (2.1))

Step 1: Generating the Trajectory of EE

The first thing that is needed, is the desired trajectory of the EE from $t = 0$ to $t = T$.

Example - A circular movement in 3D space is described as below:

$$x = R\cos(\frac{t}{T}2\pi)$$
$$y = R\sin(\frac{t}{T}2\pi)$$
$$z = C_0(constant)$$

That $0 \leq t \leq T$

Step 2: Velocity Profile Calculation

Derive the velocity profile of the movement from the Trajectory of EE by differentiating the actual equation of movement as a function of time.

Example - Continuing on the line of previous example we can derive the velocity profile:

$$\dot{x} = \frac{2\pi}{T}R(-\sin(\frac{t}{T}2\pi))$$
$$\dot{y} = \frac{2\pi}{T}R\cos(\frac{t}{T}2\pi)$$
$$\dot{z} = 0$$

Step 3: IK of the multi-point Trajectory

In this step, angles of θ_{1i} or the actuated joints will be calculated by performing IK (see section 2.2 for more information) on the acquired points in space-time. Strictly speaking, with the position coordinates of EE in each time-instance, angles of actuated joints can be calculated in the same instance.

Step 4: Using Jacobian matrix

This step, like the previous one, information about kinematics of actuated joints is calculated from End-Effector. Specifically Jacobian matrix is used to calculate angular velocity profile of actuated joints from the velocity profile of the EE. (see section 2.1 for more information)

4.4.2 EE Velocity Profile

From previous section (4.4.1) it can be said that if a "Good" velocity profile or trajectory can be found for End-Effector, then the rest of information about kinematics can be simply calculated. The challenge for my assignment is to find the perfect trajectory for a circular movement.

Since the circle formula is $(x-x_0)^2+(y-y_0)^2=R^2$, then either $x(t)$ or $y(t)$ is independent and the other one is dependent. if $x(t)$ is considered independent, then

$$y(t)=\sqrt{R^2-(x-x_0)^2}+y_0$$

differentiating the circle formula results in:

$$\dot{x}(x-x_0)+\dot{y}(y-y_0)=0$$

and as said before, $x(t)$ is independent of $y(t)$, hence $\dot{x}(t)$ is independent of $\dot{y}(t)$ and:

$$\dot{y}=-\frac{x-x_0}{y-y_0}\dot{x}$$

So the task is to find a "Good" $x(t)$ or $\dot{x}(t)$ that results in a "Good" $y(t)$ and $\dot{y}(t)$, while defining p as "Good" like:

- $\dot{p}(0)=\dot{p}(T)=0$
- $\dot{p}$ profile is smooth and does not have any discontinuity.