UFMF4X-15-M

Robotic Fundamentals

Coursework

Serial and Parallel Robot Kinematics

Authors :

Ziniu Wu
Student.No.: 22071766

Tunwu Li
Student.No.: 22071758

Instructor:
Dr Aghil Jafari

University of Bristol

University of the West of England UWE Bristol

brl Bristol Robotics Laboratory

January 9, 2023

Coursework Report - Group Cover Page

Coursework: Serial and Parallel Robot Kinematics

Student name and number	PART I.A	PART I.B	PART II.1	PART II.2	PART III
Ziniu Wu 22071766	✔	✔	✔	✔	✔
Tunwu Li 22071758	✔	✔	✔	✔	✔

Note: Students must tick the sections that they have been undertaking or helped with.

Abstract

This report is the coursework for Robotic Fundamentals and is divided into three main sections: Lynxmotion Arm, Planar Parallel Robots and Optimisation-based Trajectory Generation.

In Section 1, we first define a forward kinematic model of the five-degree-of-freedom Lynxmotion Arm by Modified DH representation and analyse the reachable workspace of its wrist. We then generate an inverse kinematic model of the Lynxmotion Arm by geometric method. Finally, we apply the obtained forward and inverse kinematic models to plan three trajectories between five positions (free motion, straight line and obstacle avoidance).

In Section 2, we first derive the inverse kinematic model of the planar parallel robot, which is similar to the delta robot, by the vector method. We then use the algebraic method to solve the expression for its workspace and the root formula for the quadratic function to determine whether it is a real solution.

Inspired by the trajectory planning in Section 1, we introduce in Section 3 a minimum-snap trajectory generation scheme for robot arm in the geometric space that allows the selection of the optimal coefficients of the polynomial as a quadratic planning problem. A numerical example is given to validate the method in one dimension.

It ends with a conclusion and references. Almost all the codes in the paper can be found in appendices.

Contents

List of Figures

List of Tables

1 Part I——Lynxmotion Arm

In this section, we firstly derive a Denavit Hartenberg (DH) representation of forward kinematics for the Lynxmotion arm as shown in Fig. 1. Secondly, we analyze the workspace of the center of the wrist (5th joint) when each preceding joint moves through its range of motion. Next, we plot the visualization of the workspace in both 2D and 3D. Thirdly, we derive the inverse kinematics model for the manipulator in closed form. Fourth, we achieve task planning for the manipulator and validate the method in different scenarios.

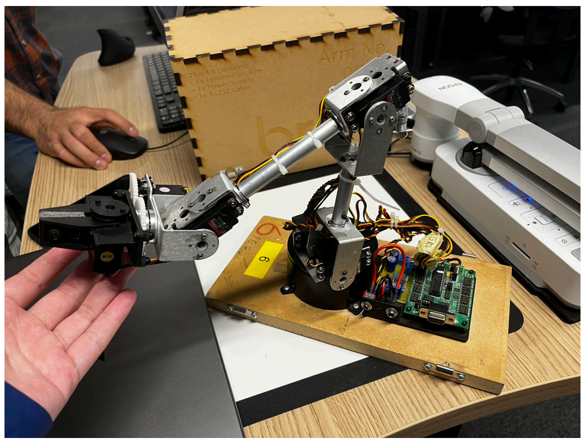


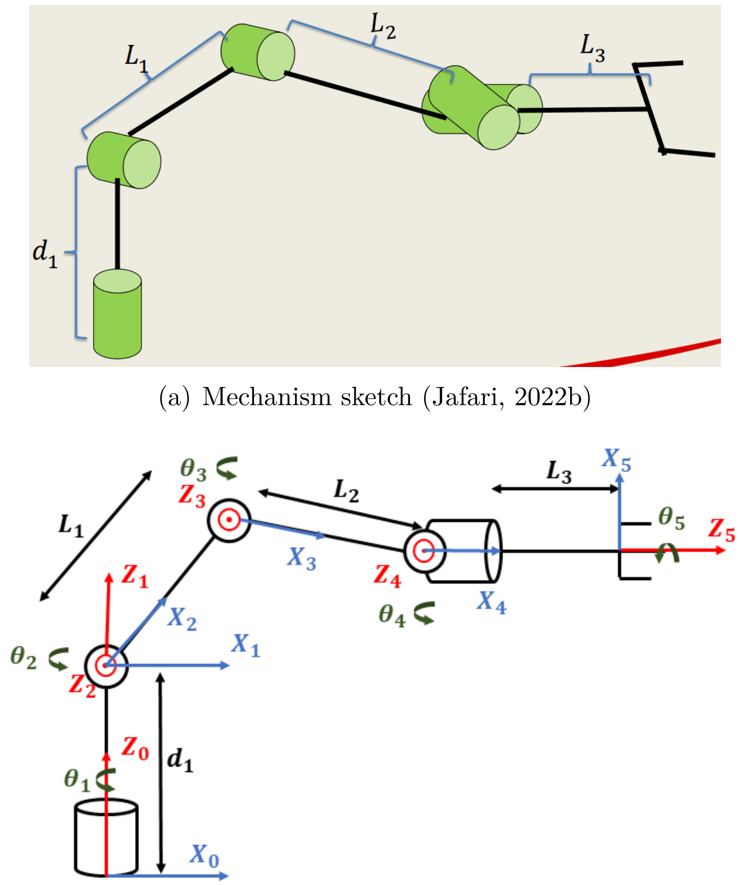
Figure 1: Lynxmotion Robot.

1.1 Reference Frames

The sketch of the mechanism of the Lynxmotion arm is shown in Fig. 2(a). We take the view in the XZ plane and simplify it to Fig. 2(b) and lay out the coordinates. First we indicate the Z-axis of each joint in red, the X-axis in blue and the angle of rotation θ around the Z-axis in green. Subsequently, we use $\odot$ to indicate that the axis is perpendicular to the paper facing outwards. All coordinates are set to satisfy the right-hand law.

1.2 Forward Kinematics

First, calculate the degrees of freedom of the Lynxmotion arm to determine the number of parameters that represent the end-effector. As shown in Fig. 2, the Lynxmotion arm is a spatial mechanism, so $d = 6$, the sum of the number of links and the number of bases $n = 5$——4 links and 1 base, the total number of joints $n = 5$——4 links and 1 base, and the total number of joints $g = 4$ ——3 revolutes and 1 universal.



(a) Mechanism sketch (Jafari, 2022b)

(b) Layout of frames

Figure 2: Kinematic model of the Lynxmotion arm (MDH)——(a) mechanism sketch and (b) layout of frames

$g = 4$——3 revolutes and 1 universal, the sum of the degrees of freedom of each joint $\sum_{i=0}^{g} f_i = 3 \times 1 + 2 = 5$, so that the degrees of freedom of the Lynxmotion arm are as follows (Jafari, 2022d), i.e. 5 parameters are needed to describe the end-effector.

$$DoF = d(n - g - 1) + \sum_{i=0}^{g} f_i = 6 \times (5 - 4 - 1) + 5 = 5 \tag{1}$$

Based on the frame in the previous subsection, we list the DH table (Tab. 1) for the Lynxmotion arm in Proximal method (MDH).

The homogeneous transformations of frame n in frame $n - 1$ for Distal is written as below (Jafari, 2022f), where cos and sin are abbreviated as c and s:

Table 1: DH parametres of the Lynxmotion arm –Proximal approach

Joint n	a_{n-1}	α_{n-1}	d_n	θ_n
1	0	0	d_1	θ_1
2	0	90°	0	θ_2
3	L_1	0	0	θ_3
4	L_2	0	0	θ_4
5	0	90°	L_3	θ_5

$$
{}^{n-1}_{n}T = \begin{bmatrix} c\theta_n & -s\theta_n & 0 & a_{n-1} \\ s\theta_n c\alpha_{n-1} & c\theta_n c\alpha_{n-1} & -s\alpha_{n-1} & -s\alpha_{n-1} d_n \\ s\theta_n s\alpha_{n-1} & c\theta_n s\alpha_{n-1} & c\alpha_{n-1} & c\alpha_{n-1} d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}
$$

The position and orientation of 5^{th} joint (the wrist) can be computed using compound transformation in frame 0. The compound transformation from frame 0 to frame 5 is denoted as (Craig, 2005):

$$
{}^{0}_{5}T = {}^{0}_{1}T \, {}^{1}_{2}T \, {}^{2}_{3}T \, {}^{3}_{4}T \, {}^{4}_{5}T \tag{3}
$$

where

$$
{}^{0}_{1}T = \begin{bmatrix} c\theta_1 & -s\theta_1 & 0 & 0 \\ s\theta_1 & c\theta_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{1}_{2}T = \begin{bmatrix} c\theta_2 & -s\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s\theta_2 & c\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{2}_{3}T = \begin{bmatrix} c\theta_3 & -s\theta_3 & 0 & L_1 \\ s\theta_3 & c\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{3}_{4}T = \begin{bmatrix} c\theta_4 & -s\theta_4 & 0 & L_2 \\ s\theta_4 & c\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{4}_{5}T = \begin{bmatrix} c\theta_5 & -s\theta_5 & 0 & 0 \\ 0 & 0 & -1 & L3 \\ s\theta_5 & c\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Therefore, the position of the end-effector in Cartesian coordinate ${}^{0}_{eff}P$ with respect to joint parameters is written as:

$$
{}^{0}P_{ORG} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T \tag{4}
$$

$$
{}^{0}_{eff}P = {}^{0}_{5}T \; P_{ORG} \tag{5}
$$

Also because the end-effector has degrees of freedom of rotation $O_\psi = \theta_2 + \theta_3 + \theta_4$ around Z_2, Z_3 and Z_4, and degrees of freedom of rotation $O_\mu = \theta_5$ around Z_5. Thus, the Cartesian coordinates of the end-effector are expressed as:

$$\begin{cases} P_x = & 150c\theta_1(s(\theta_2+\theta_3+\theta_4)+2c(\theta_2+\theta_3)+2c\theta_2) \\ P_y = & 150s\theta_1(s(\theta_2+\theta_3+\theta_4)+2c(\theta_2+\theta_3)+2c\theta_2) \\ P_z = & 300s(\theta_2+\theta_3)-150c(\theta_2+\theta_3+\theta_4)+300s\theta_2+100 \\ O_\psi = & \theta_2+\theta_3+\theta_4 \\ O_\phi = & \theta_5 \end{cases} \tag{6}$$

1.3 Workspace Analysis

The length of each link of the Lynxmotion arm and the range of rotation of each joint are:

$$\begin{cases} d_1 = 100\ mm \\ L_1 = 300\ mm \\ L_2 = 400\ mm \\ L_3 = 150\ mm \end{cases} \qquad \begin{cases} \theta_1 \in [0:360°] \\ \theta_2 \in [0:180°] \\ \theta_3 \in [0:360°] \\ \theta_4 \in [0:360°] \end{cases}$$

The reachable workspace of the Lynxmotion arm is shown in Fig. 3.

To be aware of the safe operation of the robot, it is critical to find its reachable workspace. The safe sign and fence should be placed around its workspace to avoid collision with humans and objects.

1.4 Inverse Kinematics

Inverse kinematics calculates the angle of each joint for a known position and orientation of the end-effector (Eq. 7) with respect to the origin (Craig, 2005).

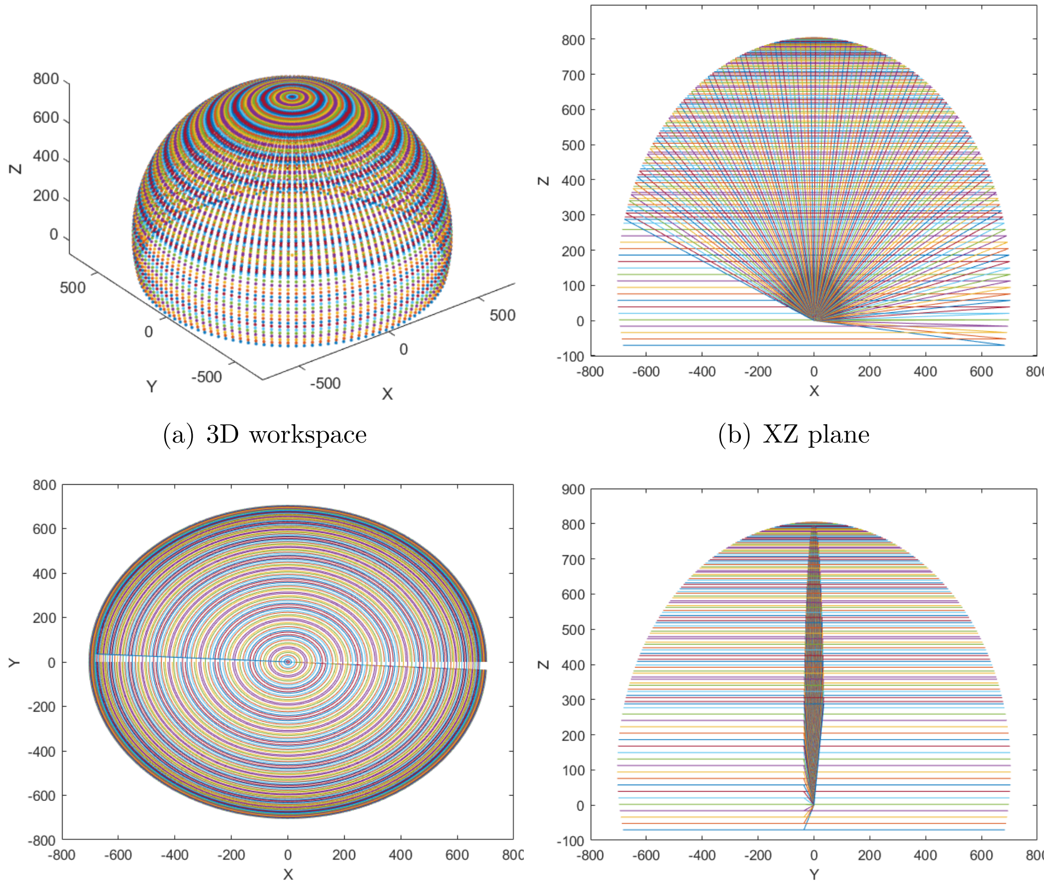
$$[P_x, P_y, P_z, O_\psi, O_\mu]^T \tag{7}$$

Step 1, as shown in Fig. 4, establish the XZ coordinate system with q_4 as the reference point. The coordinates (X_e, Z_e) of the end-effector are:

$$\begin{cases} X_e = \sqrt{P_x^2 + P_y^2} \\ Z_e = P_z \end{cases} \tag{8}$$

Step 2, solve for θ_3 and θ_2. As shown in Fig. 5, establish the XZ coordinate system with q_2 as the reference point, then the coordinates (X_4, Y_4) of q_4 are:

$$\begin{cases} X_4 = X_e - L_3cO_\psi = \sqrt{P_x^2 + P_y^2} - L_3cO_\psi \\ Z_4 = Z_e - L_3sO_\psi = P_z - L_3sO_\psi \end{cases} \tag{9}$$



(a) 3D workspace

(b) XZ plane

(c) XY plane

(d) YZ plane

Figure 3: Reachable workspace of the Lynxmotion arm——(a) 3D workspace, (b) XZ plane, (c) XY plane and (d) YZ plane

The square of the distance from q_2 to q_4 is s^2 :

$$
\begin{aligned}
s^2 &= X_4^2 + (Z_4 - d_1)^2 = L_1^2 + L_2^2 - 2L_1L_2c\theta_C \\
c\theta_C &= \frac{L_1^2 + L_2^2 - (X_4^2 + (Z_4 - d_1)^2)}{2L_1L_2} \\
&= \frac{L_1^2 + L_2^2 - (\sqrt{P_x^2 + P_y^2} - L_3cO_\psi)^2 + ((P_z - L_3sO_\psi - d_1)^2)}{2L_1L_2}
\end{aligned}
\tag{10}
$$

Calculate θ_C with inverse cosine function:

$$
\theta_C = arccos(\frac{L_1^2 + L_2^2 - (\sqrt{P_x^2 + P_y^2} - L_3cO_\psi)^2 + ((P_z - L_3sO_\psi - d_1)^2)}{2L_1L_2}) \tag{11}
$$

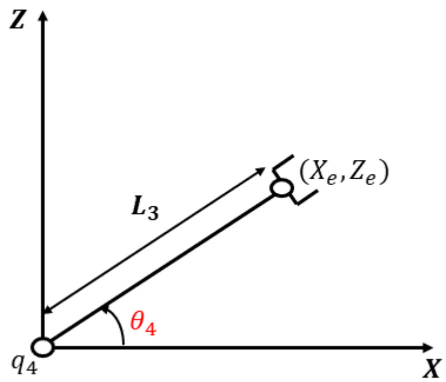
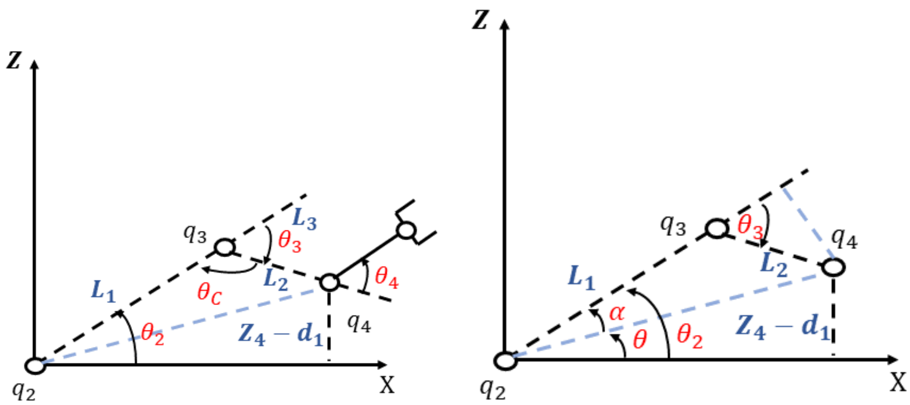


Figure 4: XZ coordinate system with q_4 as the origin



(a) Solve for θ_3 (b) Solve for θ_2

Figure 5: XZ coordinate system with q_2 as the origin

Also, since θ_C and θ_3 are complementary, i.e. $\theta_3 = \pi - \theta_C$, it follows that:

$$\theta_3 = \pi - arccos(\frac{L_1^2 + L_2^2 - (\sqrt{P_x^2 + P_y^2} - L_3 cO_\psi)^2 + ((P_z - L_3 sO_\psi - d_1)^2)}{2L_1 L_2}) \quad (12)$$

As shown in Fig. 5(b), decomposing the rotation of q_2 into θ and α and making a vertical line from the point q_4 to the extension of L_1, we get:

$$\begin{aligned} tan\theta &= \frac{Z_4 - d_1}{X_4} = \frac{P_z - L_3 sO_\psi - d_1}{\sqrt{P_x^2 + P_y^2} - L_3 cO_\psi} \\ tan\alpha &= \frac{L_2 s\theta_3}{L_1 + L_2 c\theta_3} \end{aligned} \quad (13)$$

Hence, the rotation of q_2 is:

$$\theta_2 = arctan(\frac{P_z - L_3 sO_\psi - d_1}{\sqrt{P_x^2 + P_y^2} - L_3 cO_\psi}) + arctan(\frac{L_2 s\theta_3}{L_1 + L_2 c\theta_3}) \tag{14}$$

The rotation of q_4 is:

$$\begin{aligned} \theta_4 &= O_\psi - \theta_3 - \theta_2 \\ &= O_\psi - (\pi - arccos(\frac{L_1^2 + L_2^2 - (\sqrt{P_x^2 + P_y^2} - L_3 cO_\psi)^2 + ((P_z - L_3 sO_\psi - d_1)^2)}{2L_1L_2})) \\ &- (arctan(\frac{P_z - L_3 sO_\psi - d_1}{\sqrt{P_x^2 + P_y^2} - L_3 cO_\psi}) + arctan(\frac{L_2 s\theta_3}{L_1 + L_2 c\theta_3})) \end{aligned} \tag{15}$$

From Eq. 6 we know:

$$\theta_5 = O_\mu \tag{16}$$

Step 3, solve for θ_1. As shown in Fig. 6, establish the XY coordinate system by top-down view of q_1. Then, the rotation of q_1 is:

$$\theta_1 = arctan(\frac{P_y}{\sqrt{P_x^2 + P_y^2}}) \ or \ arctan(\frac{P_y}{\sqrt{P_x^2 + P_y^2}}) + \pi \tag{17}$$

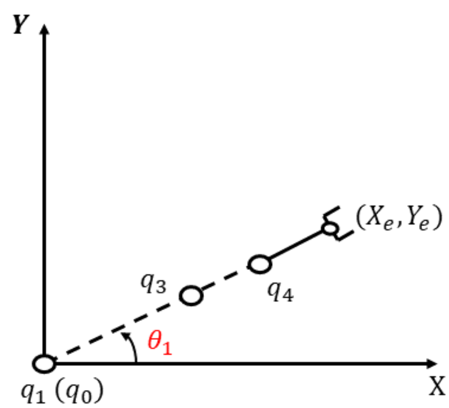


Figure 6: XY coordinate system with q_1 as the origin

In summary, the IK model for the lynxmotion arm is:

$$\begin{cases} \theta_1 &= arctan(\frac{P_y}{\sqrt{P_x^2+P_y^2}}) \ or \ arctan(\frac{P_y}{\sqrt{P_x^2+P_y^2}}) + \pi \\ \theta_2 &= arctan(\frac{P_z - L_3 sO_\psi - d_1}{\sqrt{P_x^2+P_y^2} - L_3 cO_\psi}) + arctan(\frac{L_2 s\theta_3}{L_1 + L_2 c\theta_3}) \\ \theta_3 &= \pi - arccos(\frac{L_1^2 + L_2^2 - (\sqrt{P_x^2+P_y^2} - L_3 cO_\psi)^2 + ((P_z - L_3 sO_\psi - d_1)^2)}{2L_1L_2}) \\ \theta_4 &= O_\psi - \theta_3 - \theta_2 \\ \theta_5 &= O_\mu \end{cases} \tag{18}$$

In the following trajectory planning task, we define L_3 as 0. The above expression thereby simplifies to:

$$\begin{cases} \theta_1 &= arctan(\frac{P_y}{P_x}) \\ \theta_2 &= arctan(\frac{P_z-d_1}{\sqrt{P_x^2+P_y^2}}) - arccos(\frac{L_1^2+P_x^2+P_y^2+(P_z-d_1)^2-L_2^2}{2L_1\sqrt{P_x^2+P_y^2+(P_z-d_1)^2}}) \\ \theta_3 &= arccos(\frac{P_x^2+P_y^2+(P_z-d_1)^2-L_1^2-L_2^2)}{2L_1L_2}) \\ \theta_4 &= O_\psi - \theta_2 - \theta_3 \\ \theta_5 &= O_\mu \end{cases} \tag{19}$$

In addition, we implement the $arctan$ function by using the $atan2$ function in MATLAB. It is a important detail to implement the $arctan$ function safely in order to avoid the numerical issue.

1.5 Part B: Task Planning

1.5.1 Task Planning

The state vector of i^{th} key-frames is denoted as:

$$\mathbf{K}_i = [x_i, y_i, z_i, \psi_i, \phi_i]^T \in \mathbb{R}^5 \tag{20}$$

The task $\mathcal{K}$ is defined as a set of key-frames that satisfies:

$$\mathcal{K} = \{\mathbf{K}_0, \mathbf{K}_1, \mathbf{K}_2, \ldots, \mathbf{K}_N | N \in \mathbb{Z}^+, \mathbf{K}_i \in \mathcal{W}\} \tag{21}$$

where $\mathbf{K}_0$ is the initial state of the robot arm. $\mathbf{K}_i$ is the state vector of the i^{th} key-frame. N is the number of key-frames that needs to be reached. $\mathcal{W}$ is the workspace of the robot arm. Each key-frame $\mathbf{K}_i$ is required to be a subset of the workspace $\mathcal{W}$.

The task $\mathcal{K}$ represents, firstly, the robot arm initializes its position at the beginning (i.e. the current state of the robot arm). Then, the robot arm is required to reach key-frames from $\mathbf{K}_1$ to $\mathbf{K}_N$ subsequently. It is essential to ensure each key-frames is reachable in its workspace.

A numerical example of a planned task is given by Tab. 2:

1.5.2 IK Test and Animation

We implement the Inverse Kinematics (IK) in MATLAB to calculate sets of joint configurations using **IKtest.m**, as shown in the Tab. 3:

Table 2: An instance of task planning.

	x_i	y_i	z_i	ψ_i	ϕ_i
$\mathbf{K}_0$	500	250	100	0	0
$\mathbf{K}_1$	350	300	250	0	0
$\mathbf{K}_2$	200	400	300	0	0
$\mathbf{K}_3$	150	200	150	0	0
$\mathbf{K}_4$	400	0	100	0	0

Table 3: IK test result

	θ_1	θ_2	θ_3	θ_4	θ_5
$\mathbf{K}_0$	0.4636	-0.3717	0.7435	-0.3717	0
$\mathbf{K}_1$	0.7086	-0.3155	1.2603	-0.9447	0
$\mathbf{K}_2$	1.1071	-0.1949	1.2310	-1.0360	0
$\mathbf{K}_3$	0.9273	-0.9345	2.2638	-1.3293	0
$\mathbf{K}_4$	0	-0.8411	1.6821	-0.8411	0

From the result, we ensure the value are reachable in the joint space for all keyframes. Then, we use the Forward Kinematics (FK) to validate the state of the end effector using **FKtest.m**. The result is shown in Tab. 4.

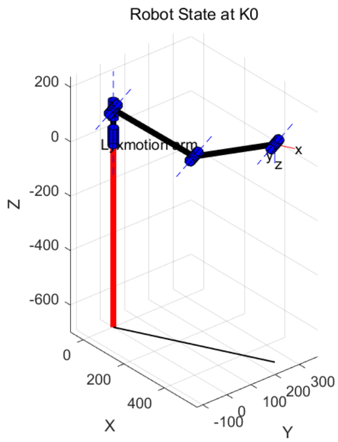
Table 4: FK test result.

	x_i	y_i	z_i	ψ_i	ϕ_i
$\mathbf{K}_0$	500.0109	249.9757	100.0280	0.0001	0
$\mathbf{K}_1$	349.9978	299.9821	250.0264	0.0001	0
$\mathbf{K}_2$	200.0107	399.9728	300.0265	0.0001	0
$\mathbf{K}_3$	150.0068	200.0111	150.0037	0.0000	0
$\mathbf{K}_4$	400.0083	-0.0000	99.9800	-0.0001	0

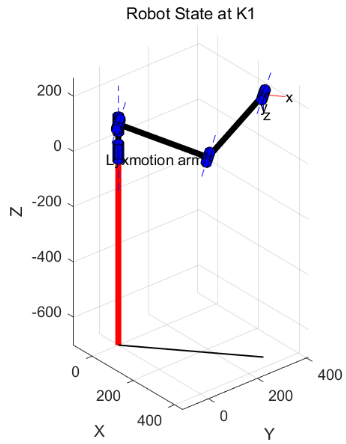
The numerical value of the FK test result is not exactly the same as the planned value due to float-point calculation. However, the maximum difference is 0.0280, which is acceptable to safely manipulate the robot arm. As shown in Fig. 7, we visualize the robot state (including the joint configurations and the state of the end-effector) at each key-frame using the Robotics Toolbox by **visualization.m**:

1.5.3 Trajectories

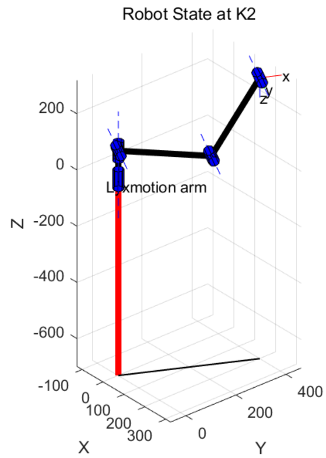
Free Motion: The trajectory can be represented by 3^{rd} order polynomial parameterised by time t, which is denoted as:



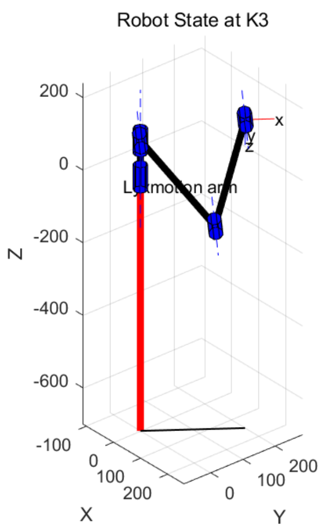
(a) $\mathbf{K}_0$



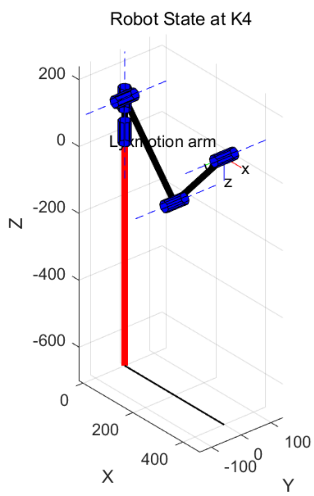
(b) $\mathbf{K}_1$



(c) $\mathbf{K}_2$



(d) $\mathbf{K}_3$



(e) $\mathbf{K}_4$

Figure 7: Visualization of the lynxmotion arm at each key-frame

$$\theta_{i,k}(t) = \sum_{j=0}^{3} p_{i,j,k}t^j = p_{i,3,k}t^3 + p_{i,2,k}t^2 + p_{i,1,k}t + p_{i,0,k}, \;\; t_0 < t < t_T \tag{22}$$

where i is the index of each piece of the polynomial. θ_i is the trajectory that connects the initial state and the end state. $p_{i,j}$ is the coefficient of the polynomial. The interval $[t_0, t_T]$ is the allocated time duration. To simplify the time allocation process, we apply equal time allocation for each segment, which means, for example, $t_T - t_0 = 2$. To express in a compact way, $\mathbf{p}_{i,k} = [p_{i,3,k}, p_{i,2,k}, p_{i,1,k}, p_{i,0,k}]^T$ denotes the vector of coefficients of k^{th} piece trajectory of θ_i.

Then the joint velocity trajectory is represented by the first-order derivative of the joint angle trajectory, which is denoted as:

$$\dot{\theta}_{i,k}(t) = 3p_{i,3,k}t^2 + 2p_{i,2,k}t + p_{i,1,k}, \;\; t_0 < t < t_T \tag{23}$$

The free motion-based trajectory generation problem is to solve the following system of equations and find the coefficients of polynomials:

$$\theta_{i,k}(0) = \mathbf{K}_{k-1}.\theta_i \tag{24}$$

$$\theta_{i,k}(T) = \mathbf{K}_k.\theta_i \tag{25}$$

$$\dot{\theta}_{i,k}(0) = \dot{\theta}_{i,k}(T) = 0 \tag{26}$$

For example, considering the first piece of the trajectory of θ_2 is denoted as $\theta_{2,1}(t)$, the start condition is $\mathbf{K}_0.\theta_2$ at $t = 0$ and the end condition is $\mathbf{K}_1.\theta_2$ at $t = T$, similarly, then we apply inverse kinematics to find the desired θ_i in the two conditions. At each keyframe, the desired velocity of the trajectory is 0:

$$\theta_{2,1}(0) = p_0 = -0.3717 \tag{27}$$

$$\theta_{2,1}(2) = 8p_3 + 4p_2 + 2p_1 + p_0 = -0.3155 \tag{28}$$

$$\dot{\theta}_{2,1}(0) = p_1 = 0 \tag{29}$$

$$\dot{\theta}_{2,1}(2) = 12p_3 + 4p_2 + p1 = 0 \tag{30}$$

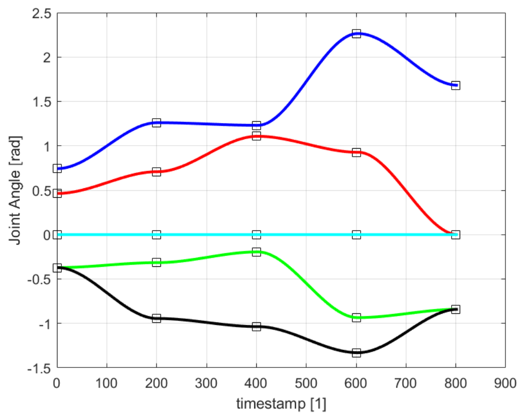
Then, solving the system of equations, we have coefficients of the $\theta_{2,1}$ trajectory, which is $\mathbf{p}_{2,1} = [-0.01405, 0.04215, 0, -0.3717]^T$. Similarly, all trajectories in the joint space and task space are calculated and shown below using **freeMotion.m**.

Straight Line Trajectory: The straight line trajectory indicates the end-effector moves from point A to point B along a straight line in the geometric space. We assume the end-effector moves at a constant speed during the trajectory. It is necessary to calculate the direction of the movement and the time duration between two keyframes. The direction of the movement can be calculated as the unit vector pointing from the current keyframe to the subsequent keyframe, which is denoted as:

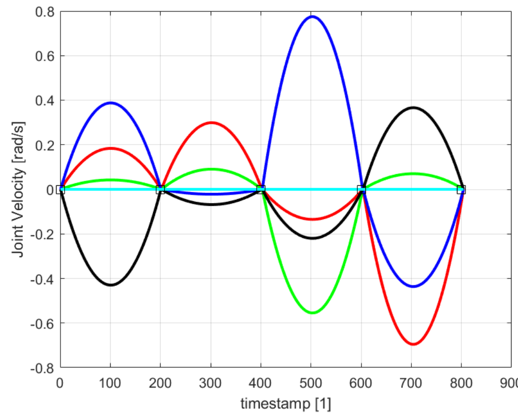
$$\mathbf{d_i} = \frac{\mathbf{K}_{i+1} - \mathbf{K}_i}{\|\mathbf{K}_{i+1} - \mathbf{K}_i\|} \tag{31}$$

and the time duration of i^{th} segment is written as:

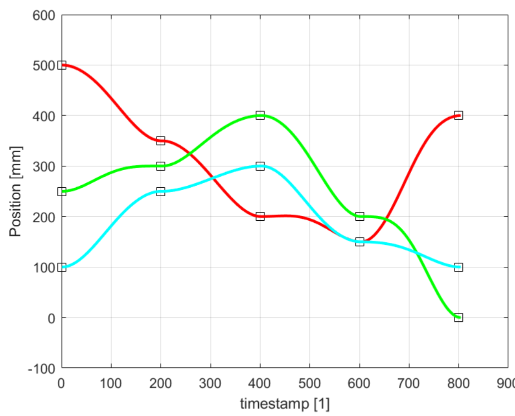
$$\mathbf{t}_i = \frac{\|\mathbf{d_i}\|}{v} \tag{32}$$



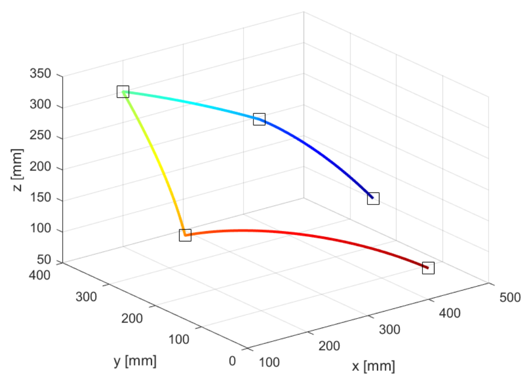
(a) Trajectories of joint angle. Red is θ_1. Red is θ_1. Green is θ_2. Blue is θ_3. Black is θ_4. Cyan is θ_5.



(b) Trajectories of joint velocity. Red is θ_1. Green is θ_2. Blue is θ_3. Black is θ_4. Cyan is θ_5.



(c) Trajectories of position. Red is x axis. Green is y axis. Cyan is z axis.

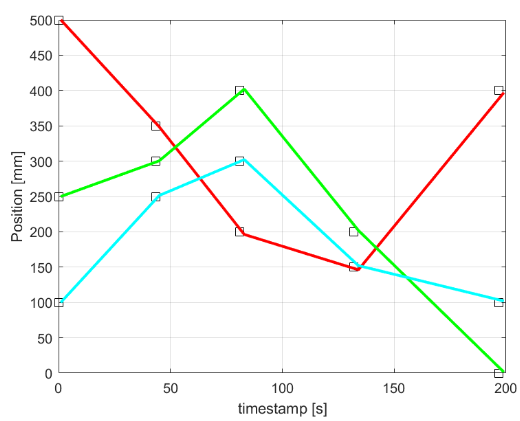


(d) 3D visualization of position trajectory. Green represents the position trajectory of the end-effector. Makers are the planned keyframes.

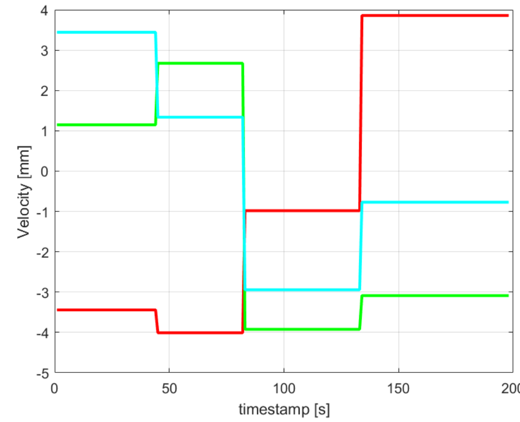
Figure 8: Free motion trajectory. One timestamp represents 0.01 second.

where v is the user-defined constant velocity of the end-effector. Then, the scalar $\mathbf{d_i}v$ represents the desired velocity of the end-effector. And the $\sum_k^{\mathbf{t}_i} \mathbf{d_i}v$ represents the desired position in each axis. Finally, the desired command in the joint space can be derived by the IK. The planned straight-line trajectory is shown in Fig. 9 using **straightTrajectory.m**.

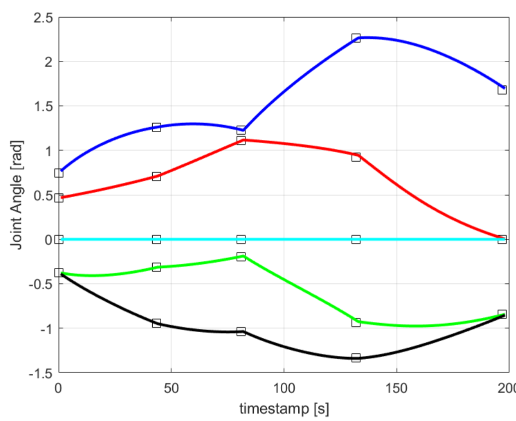
Obstacle Avoidance: The obstacle avoidance trajectory planned requires finding a collision-free trajectory to move the robot from point A to B. One approach is to interpolate waypoints based safe distance criterion. We insert the keyframe to ensure the minimum distance between the end-effector and the obstacle are above the minimum safe distance. The planned obstacle avoidance trajectory is shown in



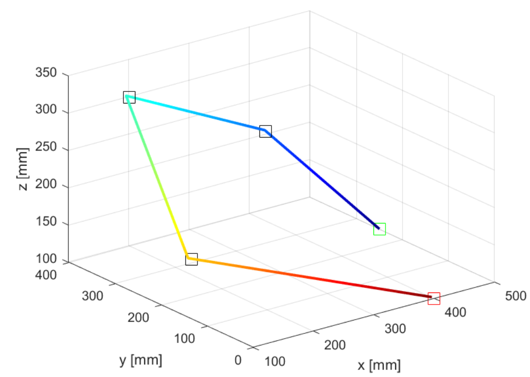
(a) Trajectories of end effector position. Red is x axis. Green is y axis. Cyan is z axis.



(b) Trajectories of end effector velocity. Red is x axis. Green is y axis. Cyan is z axis.



(c) Trajectories of joint angle. Red is θ_1. Green is θ_2. Blue is θ_3. Black is θ_4. Cyan is θ_5.



(d) 3D visualization of position trajectory. Green represents the position trajectory of the end-effector. Makers are the planned keyframes.

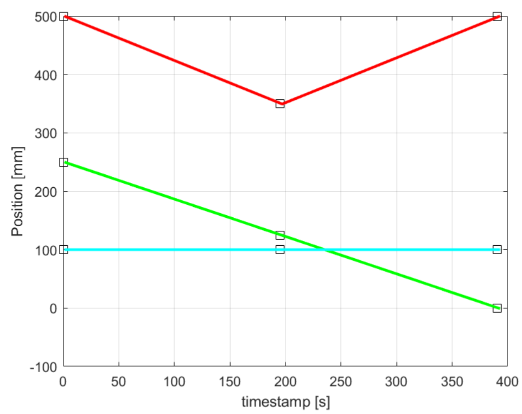
Figure 9: Straight line trajectory.

Fig. 10 using **avoidance.m**.

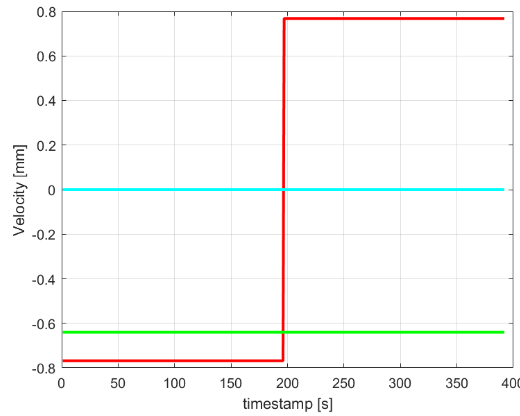
2 Part II——Planar Parallel Robot

2.1 Inverse Kinematics

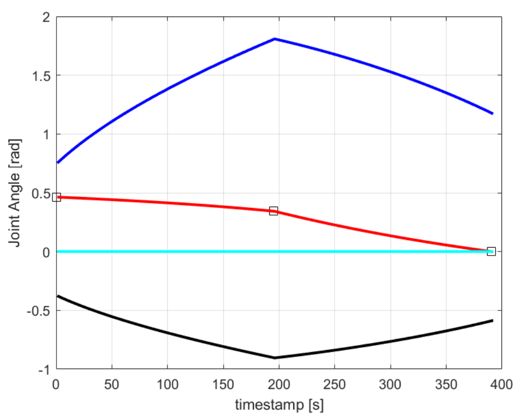
The sketch of the mechanism of the parallel robot and the layout of the frames are shown in Fig. 11. $\{B\}$ and $\{C\}$ are the origins of the base and platform, i.e. the centre of gravity of the equilateral triangles. We set the radius of the external circles



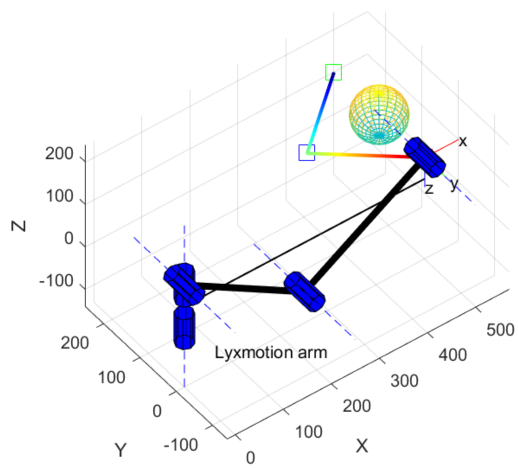
(a) Trajectories of end effector position. Red is x axis. Green is y axis. Cyan is z axis.



(b) Trajectories of end effector velocity. Red is x axis. Green is y axis. Cyan is z axis.



(c) Trajectories of joint angle. Red is θ_1. Green is θ_2. Blue is θ_3. Black is θ_4. Cyan is θ_5.



(d) 3D visualization of position trajectory. Green represents the position trajectory of the end-effector. Makers are the planned keyframes. The meshed sphere is the obstacle.

Figure 10: Straight line trajectory.

of the base and platform to be R and r respectively. The design parameters are summarised in Tab. 5.

It is known that the translation and rotation of $\{C\}$ with respect to $\{B\}$ is represented by X_c, Y_c and α:

$$\overrightarrow{BC} = \begin{bmatrix} X_c \\ Y_c \\ \alpha \end{bmatrix} \tag{33}$$

The coordinates of the point PP_i in $\{B\}$ are the combination of the transformation

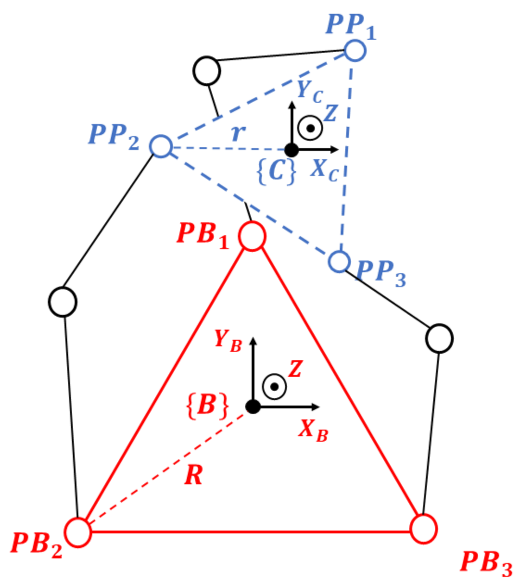


Figure 11: Kinematic model of the parallel robot (Jafari, 2022g)

Table 5: Parameters of parallel robot (Jafari, 2022a)

Dimension	Value/mm
Length of upper section S_A	170
Length of lower section L	130
Radius of outer circle of base R	290
Radius of outer circle of platform r	130

of $\{C\}$ with respect to $\{B\}$ and the coordinates of PP_i in $\{C\}$ (Jafari, 2022c):

$$\overrightarrow{BPP_i} = \overrightarrow{BC} + \overrightarrow{CPP_i} \quad where\ i = 1, 2, 3 \tag{34}$$

Express $\overrightarrow{CPP_i}$ as:

$$\overrightarrow{CPP_i} = {}^C T_{PP_i} = Trans(X, r)^T \cdot Rot(Z, \phi_i) \cdot Rot(Z, \frac{\pi}{2} + (i-1)\frac{2}{3}\pi) \quad where\ i = 1, 2, 3 \tag{35}$$

where ϕ_i is the rotation of each end-effector around the Z-axis:

$$\begin{cases} \phi_1 & = \alpha + \frac{3}{2}\pi \\ \phi_2 & = \alpha + \frac{\pi}{6} \\ \phi_3 & = \alpha + \frac{5}{6}\pi \end{cases} \tag{36}$$

The rotation matrix $Rot(Z, \phi_i)$ around the Z-axis is:

$$Rot(Z, \phi_i) = \begin{bmatrix} c\phi_i & -s\phi_i & 0 \\ s\phi_i & c\phi_i & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad where\ i = 1, 2, 3 \tag{37}$$

Similarly, the point PB_i in the reference system $\{B\}$ is:

$$\overrightarrow{BPB_i} =^B T_{PB_i} = Trans(X, R)^T \cdot Rot(Z, \frac{\pi}{2} + (i-1)\frac{2}{3}\pi) \tag{38}$$

Also:

$$\overrightarrow{PB_iPP_i} = \overrightarrow{BPP_i} - \overrightarrow{BPB_i} \tag{39}$$

Thus, compute $\overrightarrow{PB_iPP_i}$ as:

$$\overrightarrow{PB_1PP_1} = \begin{bmatrix} X_C + rc\alpha \\ Y_C + R - rs\alpha \\ \alpha \end{bmatrix} \quad \overrightarrow{PB_2PP_2} = \begin{bmatrix} X_C + \frac{\sqrt{3}}{2}R - rc(\alpha + \frac{\pi}{3}) \\ Y_C - \frac{R}{2} + rs(\alpha + \frac{\pi}{3}) \\ \alpha \end{bmatrix}$$

$$\overrightarrow{PB_3PP_3} = \begin{bmatrix} X_C - \frac{\sqrt{3}}{2}R - rc(\alpha - \frac{\pi}{3}) \\ Y_C - \frac{R}{2} + rc(\alpha + \frac{\pi}{6}) \\ \alpha \end{bmatrix}$$

Substitute the values in Tab. 5 to obtain:

$$\overrightarrow{PB_1PP_1} = \begin{bmatrix} X_C + 130 \cdot c\alpha \\ Y_C - 130 \cdot s\alpha + 290 \\ \alpha \end{bmatrix}$$

$$\overrightarrow{PB_2PP_2} = \begin{bmatrix} X_C - 130 \cdot c(\alpha + \frac{\pi}{3}) + 145\sqrt{3} \\ Y_C + 130 \cdot s(\alpha + \frac{\pi}{3}) - 145 \\ \alpha \end{bmatrix}$$

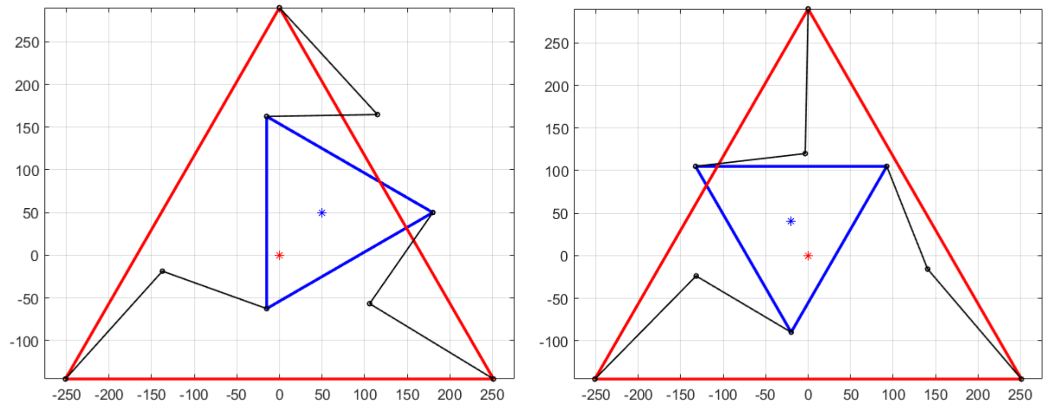
$$\overrightarrow{PB_3PP_3} = \begin{bmatrix} X_C - 130 \cdot c(\alpha - \frac{\pi}{3}) - 145\sqrt{3} \\ Y_C - 130 \cdot c(\alpha + \frac{\pi}{6}) - 145 \\ \alpha \end{bmatrix}$$

Therefore, the angle θ_i for each leg is:

$$\theta_i = arctan(\overrightarrow{PB_iPP_{iy}}, \overrightarrow{PB_iPP_{ix}}) \quad where\ i = 1, 2, 3$$

$$\begin{cases} \theta_1 = arctan(\frac{Y_C - 130 \cdot s\alpha + 290}{X_C + 130 \cdot c\alpha}) \\ \theta_2 = arctan(\frac{Y_C + 130 \cdot s(\alpha + \frac{\pi}{3}) - 145}{X_C - 130 \cdot c(\alpha + \frac{\pi}{3}) + 145\sqrt{3}}) \\ \theta_3 = arctan(\frac{Y_C - 130 \cdot c(\alpha + \frac{\pi}{6}) - 145}{X_C - 130 \cdot c(\alpha - \frac{\pi}{3}) - 145\sqrt{3}}) \end{cases} \tag{40}$$

The kinematic model for α angles at two positions is shown in Fig. 12.



(a) $\alpha = 30°, X_C = 50, Y_C = 50$

(b) $\alpha = 60°, X_C = -20, Y_C = 40$

Figure 12: Simulated motion in two positions——(a) $\alpha = 30°, X_C = 50, Y_C = 50$ and (b) $\alpha = 60°, X_C = -20, Y_C = 40$

2.2 Workspace for a Given Orientation

To draw the workspace of the robot, we need to calculate its FK model first. As shown in Fig. 13, the coordinates of the point PP_i are:

$$\begin{cases} PP_{ix} = S_A \cdot c\theta_i + L \cdot c\psi_i \\ PP_{iy} = S_A \cdot s\theta_i + L \cdot s\psi_i \end{cases} \tag{41}$$

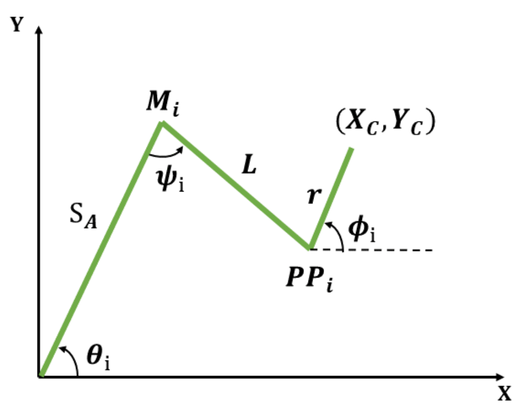


Figure 13: Analytic forward kinematic model for the parallel robot

Add the square of the first row of Eq. 41 to the square of the second row to obtain:

$$PP_{ix}^2 + PP_{iy}^2 - 2 \cdot S_A(PP_{ix} \cdot c\theta + PP_{iy} \cdot s\theta) + S_A^2 - L^2 = 0 \tag{42}$$

Simplified above equation with e_1, e_2, e_3 yields:

$$\begin{cases} e_1 &= \quad -2PP_{iy} \cdot S_A \\ e_2 &= \quad -2PP_{ix} \cdot S_A \\ e_3 &= PP_{ix}^2 + PP_{iy}^2 + S_A^2 - L^2 \end{cases} \tag{43}$$

$$e_1 s\theta + e_2 c\theta + e_3 = 0 \tag{44}$$

Let $t = tan(\frac{\theta}{2})$, then:

$$\begin{cases} s\theta = \frac{2t}{1+t^2} \\ c\theta = \frac{1-t^2}{1+t^2} \end{cases} \tag{45}$$

Substitute into Eq. 44 to get:

$$(e_3 - e_2)t^2 + 2e_1 t + e_2 + e_3 = 0 \tag{46}$$

Apply the root formula for the quadratic equation:

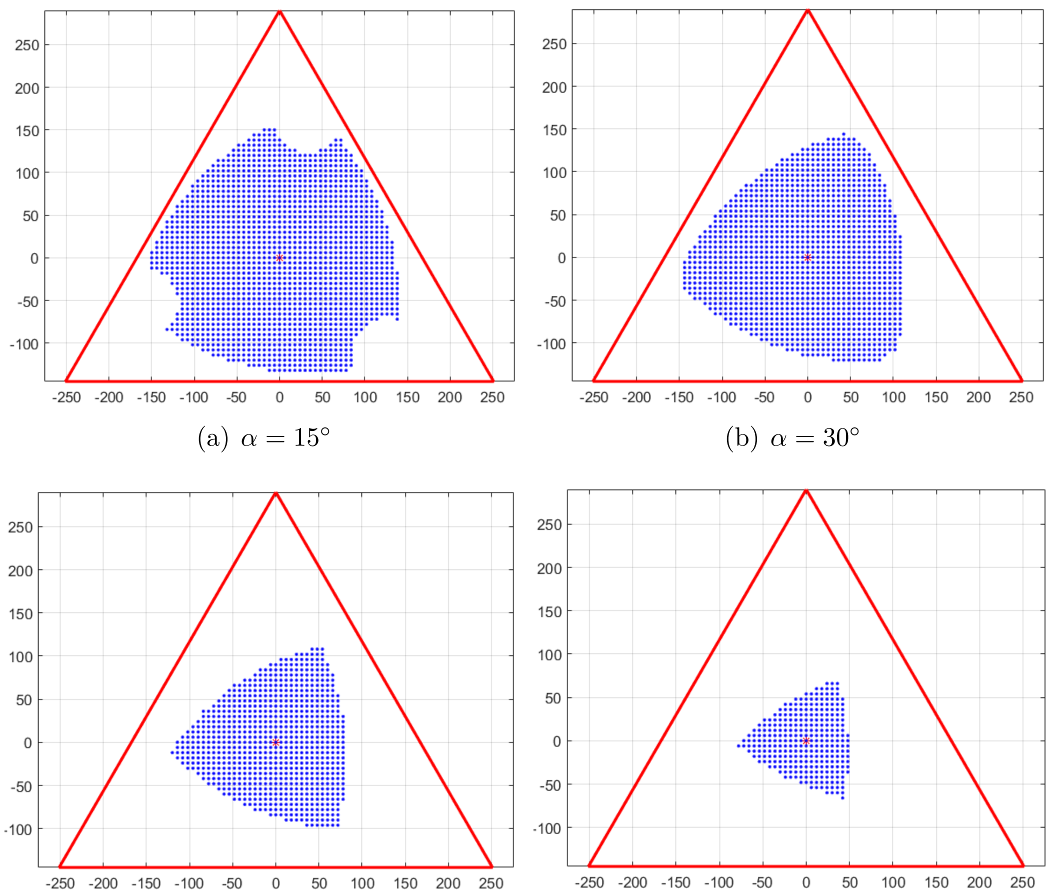
$$\begin{aligned} & t = \frac{-e_1 \pm \sqrt{e_1^2 + e_2^2 + e_3^2}}{e_3 - e_2} \quad discard\ imaginary\ numbers \\ & \theta_i = 2 \cdot arctan(t) \end{aligned} \tag{47}$$

In MATLAB, we first define $\{C\}$ as known, apply IK to solve for all possible angles θ_i, then discard the imaginary numbers and the set of retained real numbers that form the workspace of the parallel robot. The workspaces for given orientations are shown in Fig. 14.

3 Part III——Optimization-based Trajectory Generation

3.1 Optimization Problem

Kyriakopoulos and Saridis (1988) presented the minimum-jerk trajectory generation in 1988. To ensure the higher order derivative of the trajectory is continuous, and the jerk through the trajectory is minimized. Subsequently, the minimum-snap trajectory based on piece-wise polynomials is formulated as the optimization problem to optimize the energy consumption (Mahony, Kumar, & Corke, 2012). The problem of minimum-snap trajectory is denoted as:



(a) $\alpha = 15°$ (b) $\alpha = 30°$

(c) $\alpha = 45°$ (d) $\alpha = 60°$

Figure 14: Workspace of the parallel robot for given orientations——(a) $\alpha = 15°$, (b) $\alpha = 30°$, (c) $\alpha = 45°$ and (d) $\alpha = 60°$

$$
\begin{aligned}
\min_{\mathbf{a}} \quad & J(p) = \sum_{i=0}^{M-1} \int_0^{\Delta T_i} (p_i^{(4)}(t))^2 dt \\
\text{s.t.} \quad & \left\{ \begin{array}{llll}
p_0^{(k)}(T_0) & = d_0^{(k)} & , k \in [0,3] & \textit{Initial State Constraint} \\
p_{M-1}^{(k)}(T_M) & = d_{T_M}^{(k)} & , k \in [0,3] & \textit{Final State Constraint} \\
p_i^{(k)}(T_i) & = p_{i+1}^{(k)}(T_i) & , i \in [0, M-2], k \in [0,3] & \textit{Continuity Constraint}
\end{array} \right.
\end{aligned} \tag{48}
$$

where the objective function is to minimize the L-2 norm of the fourth derivative of the desired trajectory. The first constraint requires meeting the initial state constraint. The second constraint requires meeting the initial state constraint. The last constraint requires achieving the continuity constraint between keyframes.

A sequence of key-frames can be represented by the M segments of piece-wise trajectory:

$$p(t) = \begin{cases} p_1(t) = \sum_{i=0}^{n} a_{1,i} t^i, & T_0 \leq t \leq T_1 \\ p_2(t) = \sum_{i=0}^{n} a_{2,i} t^i, & T_1 \leq t \leq T_2 \\ \vdots & \vdots \\ p_M(t) = \sum_{i=0}^{n} a_{M,i} t^i, & T_{M-1} \leq t \leq T_M \end{cases} \tag{49}$$

where $p_k(t)$ is the k^{th} segment polynomial trajectory. The graphical representation in one axis is shown in Fig. 15. $a_{k,i}$ is the coefficients of the k^{th} segment. As discussed in the free motion trajectory part, the trajectory is parameterized by time t. Therefore, the time duration for each segment should be known. We could allocate equal time allocation for each segment or adaptive allocate time duration based on the Euclidean distance between two key-frames. This trajectory is formulated in the task space. Inverse kinematics is necessary to transform the geometric command into joint-level commands.

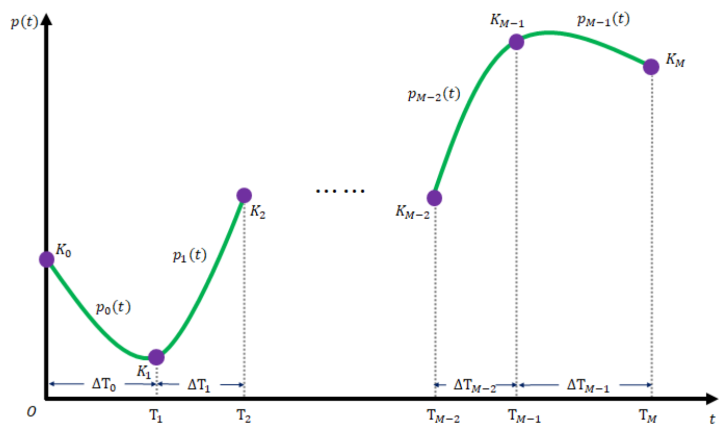


Figure 15: Piece-wise polynomial parameterized by time t in one axis

Then the compact form of the optimization problem is denoted as:

$$\begin{aligned} \min_{\mathbf{a}} \quad & \mathbf{a}^T \mathbf{Q}(\Delta T_i) \mathbf{a} \\ s.t. \quad & \mathbf{A}_{eq} \mathbf{a} = \mathbf{d}_{eq} \end{aligned} \tag{50}$$

In mathematics, it is formulated as a quadratic programming problem (QP) with equality constraints. Polynomial coefficients are decision variables. Quadratic program solvers, such as MATLAB quadprog() and fmincon(), can be implemented to find the optimal solution of the coefficients.

3.2 Numerical Example

In this subsection, we examine the minimum-snap trajectory generation with 5-th-order polynomial curves between two keyframes on one dimension with time, with a numerical example.

Let a 5-th order polynomial curve represent the trajectories:

$$x(t) = \sum_{i=0}^{5} a_i t^i = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0 \tag{51}$$

By calculating its derivatives, the trajectory of velocity, acceleration, jerk and snap are written as:

$$velocity = \dot{x}(t) = \sum_{i=1}^{5} i a_i t^{i-1} = 5a_5 t^4 + 4a_4 t^3 + 3a_3 t^2 + 2a_2 t + a_1 \tag{52}$$

$$accelaration = \ddot{x}(t) = \sum_{i=2}^{5} i(i-1) a_i t^{i-2} = 20a_5 t^3 + 12a_4 t^2 + 6a_3 t + 2a_2 \tag{53}$$

$$jerk = x^{(3)}(t) = \sum_{i=3}^{5} i(i-1)(i-2) a_i t^{i-3} = 60a_5 t^2 + 24a_4 t + 6a_3 \tag{54}$$

$$snap = x^{(4)}(t) = \sum_{i=4}^{5} i(i-1)(i-2)(i-3) a_i t^{i-4} = 120a_5 t + 24a_4 \tag{55}$$

Jerk measures how fast the acceleration change (Kyriakopoulos & Saridis, 1988). Differential jerk is called snap, which represents to the change of energy of the end-effector in the geometric space (Mahony et al., 2012). For the robot arm, snap represents the differential acceleration in the workspace. So far this part has focused on the translation motion of a robot arm in one dimension. The orientation of the end-effector could be ignored since it is irrelevant to geometric planning. Therefore, we assume the orientation of the end-effector is constant.

Boundary conditions represent the desired initial and final states. For example, the initial position is x_0. the initial velocity is v_0, and the initial acceleration is a_0. At the time T, the expected position is x_T, the velocity is v_T and the acceleration is a_T.:

Using the polynomial above, we can rewrite the boundary conditions in matrix form, where A_{eq} is the Hessian matrix, $\mathbf{a}$ is the coefficients of the polynomial, $\mathbf{d}_{eq}$ is the vector of initial and final states:

$$\mathbf{A}_{eq}\mathbf{a} = \mathbf{d}_{eq}$$

Table 6: Boundary conditions

	x	**v**	**a**
t=0	x_0	v_0	a_0
t=T	x_T	v_T	a_T

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ T^5 & T^4 & T^3 & T^2 & T & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 5T^4 & 4T^3 & 3T^2 & 2T & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 20T^3 & 12T^2 & 6T & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_5 \\ p_4 \\ p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_T \\ v_0 \\ v_T \\ a_0 \\ a_T \end{bmatrix}$$

We denoted the objective function as minimizing the snap, which refers to plan the energy-efficient trajectory in task space:

$$\begin{aligned} J(T) &= \int_0^T ||x^{(4)}(t)||_2 dt \\ &= \int_0^T (120p_5 t + 24p_4)^2 dt \\ &= \int_0^T (14400p_5^2 t^2 + 2880p_4 p_5 t + 576p_4^2) dt \\ &= [4800p_5^2 t^3 + 1440p_4 p_5 t^2 + 576p_4^2 t]|_{t=0}^{t=T} \\ &= 4800p_5^2 T^3 + 1440p_4 p_5 T^2 + 576p_4^2 T \\ &:= \mathbf{a}^T \mathbf{Q} \mathbf{a} \end{aligned}$$

where

$$\mathbf{a} = \begin{bmatrix} p_5 \\ p_4 \\ p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix}, \mathbf{Q} = \begin{bmatrix} 4800T^3 & 720T^2 & 0 & 0 & 0 & 0 \\ 720T^2 & 576T & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The exact value are given in the following table to validate the result:

Therefore, the parametric matrix:

Table 7: Boundary conditions

	x	**v**	**a**
t=0	0	1	0
t=1	1	0	0

$$\mathbf{Q} = \begin{bmatrix} 4800 & 720 & 0 & 0 & 0 & 0 \\ 720 & 576 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \mathbf{A}_{eq} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 5 & 4 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 \\ 20 & 12 & 6 & 2 & 0 & 0 \end{bmatrix}, \mathbf{d}_{eq} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The MATLAB codes for solving the quadratic programming problem are listed in Appendix H. Then, we found the optimal coefficients using the **quadprog()** solver in the MATLAB:

$$\mathbf{a}^* = \begin{bmatrix} 3 & -7 \; 4 \; 0 \; 1 \; 0 \end{bmatrix}^T$$

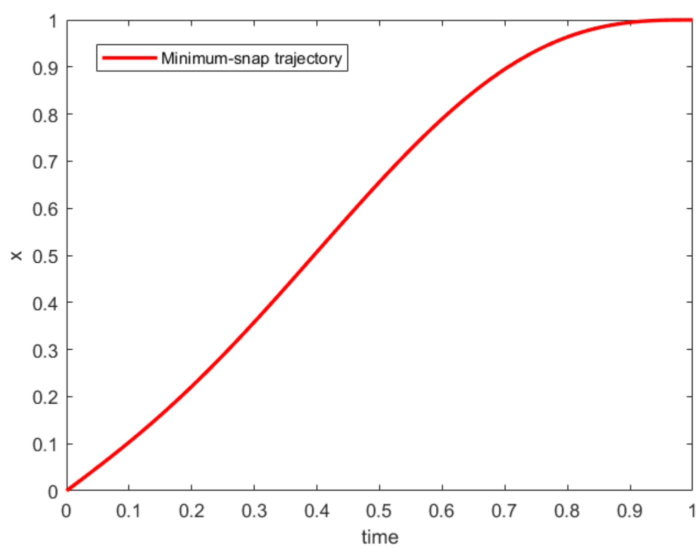


Figure 16: 1D Minimum-snap Trajectory Result

As shown in Fig. 16, the resultant polynomial satisfied the proposed condition and parameters. The velocity, jerk, and snap can be easily calculated by taking the derivative of the resultant polynomial. Further, we can calculate the trajectory in the y-axis and z-axis in the same way. The desired control command in joint space is to achieve a position and higher derivatives at a specific time with respect to the polynomial with a transformation using inverse kinematics (IK).

4 Conclusion

In conclusion, we implemented forward and inverse kinematic models for the Lynxmotion Arm and plotted its 2D and 3D reachable workspace. After verifying its correctness, we selected five points in this workspace for three different trajectory planning. As for the planar parallel robot, we first plotted the platform's poses in different positions and rotations by the derived inverse kinematic model and then solved its workspace by the algebraic method. Lastly, we introduced and implemented a minimum-snap trajectory in the task space, successfully finding the optimal coefficients of the quadratic programming problem, resulting in an energy-efficient trajectory.

References

Craig, J. J. (2005). *Introduction to robotics : mechanics and control* (3rd ed.). Pearson/Prentice Hall.

Jafari, A. (2022a). *Coursework: Serial and Parallel Robot Kinematics.*

Jafari, A. (2022b). *Lynxmotion Robot.*

Jafari, A. (2022c). *Parallel Robots II - IK Answers.*

Jafari, A. (2022d). *Week 1 - Introduction to kinematic concepts.*

Jafari, A. (2022e). *Week 2 - Homogenous Transformation.*

Jafari, A. (2022f). *Week 3 - Forward Kinematics - Denavit Hartenberg .*

Jafari, A. (2022g). *Week 7 - Parallel II.*

Kyriakopoulos, K., & Saridis, G. (1988). Minimum jerk path generation. In *Proceedings. 1988 ieee international conference on robotics and automation* (p. 364-369 vol.1). doi: 10.1109/ROBOT.1988.12075

Mahony, R., Kumar, V., & Corke, P. (2012). Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics and Automation Magazine*, *19*(3), 20–32. doi: 10.1109/MRA.2012.2206474

Appendix A MATLAB Codes of FK & Workspace for Lynxmotion Arm

```
%% Forward Kinematics
% A general expression for a homogeneous transformation matrix
% Author: Tunwu Li

%%
clear
clc
close all

%% Initialization

syms d_1 t_1 L_1 t_2 L_2 t_3 t_4 L_3 t_5

%%
T_01 = Distal_val(0, 90, d_1, t_1);

T_01 = [cos(t_1), 0,  sin(t_1),   0;
         sin(t_1), 0, -cos(t_1),   0;
        0,         1,  0,          d_1;
        0,         0,  0,           1];

R_01 = T_01(1:3, 1:3);
P_01 = T_01(1:3, 4);

T_12 = Distal_val(L_1, 0, 0, t_2);

T_12 = [cos(t_2), -sin(t_2), 0, L_1*cos(t_2);
        sin(t_2),  cos(t_2), 0, L_1*sin(t_2);
        0,         0,        1, 0;
        0,         0,        0, 1];

R_12 = T_12(1:3, 1:3);
P_12 = T_12(1:3, 4);

T_23 = Distal_val(L_2, 0, 0, t_3);

T_23 = [cos(t_3), -sin(t_3), 0, L_2*cos(t_3);
        sin(t_3),  cos(t_3), 0, L_2*sin(t_3);
        0,         0,        1, 0;
        0,         0,        0, 1];

R_23 = T_23(1:3, 1:3);
P_23 = T_23(1:3, 4);

T_34 = Distal_val(0, 90, 0, t_4);

```

```
T_34 = [cos(t_4), 0,  sin(t_4), 0;
        sin(t_4), 0, -cos(t_4), 0;
        0,        1,  0,        0;
        0,        0,  0,        1];

R_34 = T_34(1:3, 1:3);
P_34 = T_34(1:3, 4);

T_45 = Distal_val(0, 0, L_3, t_5);

T_45 = [cos(t_5), -sin(t_5), 0,   0;
        sin(t_5),  cos(t_5), 0,   0;
        0,         0,        1, L_3;
        0,         0,        0,   1];

R_45 = T_45(1:3, 1:3);
P_45 = T_45(1:3, 4);

% Compound transformation
% T_05
T_05 = T_01 * T_12 * T_23 * T_34 * T_45;
T_sim = simplify(T_05);

%% Forward Kinematics Function for Distal Approach
% To calculate value
% Author: Tunwu Li

%%
function Distal_val = Distal_val(a, alpha, d, t)
% A general expression for a homogeneous transformation matrix
% Degree system
% Input parametres: a, alpha, d, t

Distal_val = [cosd(t)  -cosd(alpha)*sind(t)  sind(alpha)*sind(t)  a*cosd(t);
              sind(t)   cosd(alpha)*cosd(t) -sind(alpha)*cosd(t)  a*sind(t);
              0         sind(alpha)          cosd(alpha)          d;
              zeros(1, 3)   1];
%%

% Cartesian coordinates of end-effector
X = simplify(T_05(1, 4))
Y = simplify(T_05(2, 4))
Z = simplify(T_05(3, 4))

% End-effector orientation
ef_pitch = t_3 + t_4 + t_5;
ef_roll  = t_5;

%% Workspace
% Degree system
t_1 = 0 : 1 : 359;
```

```
t_2 = 0 : 1/2 :  179.5;
t_3 = 0 : 1 :  359;
t_4 = 0 : 1 :  359;
t_5 = 0 : 1 :  359;

d_1 = 100; % 100 mm
L_1 = 300;
L_2 = 400;
L_3 = 150;

%% 3D reachable workspace
figure (1)

x_work = zeros(360, 360); % reserving space for the variables, because
y_work = zeros(360, 360); % otherwise they would be created later within a loop.
z_work = zeros(360, 360);

for i = 1 : 360 % for theta1
    for j = 1 : 360   % for theta2
        for k = 1 : 360 % for theta3
            for q = 1 : 360 % for theta4

                x_work(i, j) = cosd(t_1(i))*(L_2*cosd(t_2(j) + t_3(k)) + L_1*cosd(t_2(j)) + L_3*sind(t_2(j) + t_3(k) + t_4(q)));
                y_work(i, j) = sind(t_1(i))*(L_2*cosd(t_2(j) + t_3(k)) + L_1*cosd(t_2(j)) + L_3*sind(t_2(j) + t_3(k) + t_4(q)));
                z_work(i, j) = d_1 + L_2*sind(t_2(j) + t_3(k)) + L_1*sind(t_2(j)) - L_3*cosd(t_2(j) + t_3(k) + t_4(q));

            end
        end
    end
end

plot3(x_work, y_work, z_work, '.')
xlabel('X'); ylabel('Y'); zlabel('Z');
axis equal
```

Appendix B MATLAB Codes of Free Motion Trajectory

```matlab

%% Free Motion Trajectory
% Author: Ziniu Wu
clc
clear all

N = 4; % number of segments
tstep = 0.01;
ts = [2 2 2 2]; % Time allocation (simple equal allocation)

% Joint values
theta = [0.4636   -0.3717   0.7435   -0.3717        0;
         0.7086   -0.3155   1.2603   -0.9447        0;
         1.1071   -0.1949   1.2310   -1.0360        0;
         0.9273   -0.9345   2.2638   -1.3293        0;
              0   -0.8411   1.6821   -0.8411        0];
keyframe_theta = transpose(theta);
disp(keyframe_theta);

% Get polynomial coefficients
poly_coef_theta_1 = [];
poly_coef_theta_2 = [];
poly_coef_theta_3 = [];
poly_coef_theta_4 = [];
poly_coef_theta_5 = [];

for i = 1:N

    poly_coef_theta_1_temp = getCoeff(keyframe_theta(1,i),keyframe_theta(1,i+1));
    poly_coef_theta_2_temp = getCoeff(keyframe_theta(2,i),keyframe_theta(2,i+1));
    poly_coef_theta_3_temp = getCoeff(keyframe_theta(3,i),keyframe_theta(3,i+1));
    poly_coef_theta_4_temp = getCoeff(keyframe_theta(4,i),keyframe_theta(4,i+1));
    poly_coef_theta_5_temp = getCoeff(keyframe_theta(5,i),keyframe_theta(5,i+1));

    poly_coef_theta_1 = [poly_coef_theta_1 poly_coef_theta_1_temp];
    poly_coef_theta_2 = [poly_coef_theta_2 poly_coef_theta_2_temp];
    poly_coef_theta_3 = [poly_coef_theta_3 poly_coef_theta_3_temp];
    poly_coef_theta_4 = [poly_coef_theta_4 poly_coef_theta_4_temp];
    poly_coef_theta_5 = [poly_coef_theta_5 poly_coef_theta_5_temp];

end

% Generate control sequence
theta_1_n = [];
theta_2_n = [];
theta_3_n = [];
```

```
theta_4_n = [];
theta_5_n = [];
theta_dot_1_n = [];
theta_dot_2_n = [];
theta_dot_3_n = [];
theta_dot_4_n = [];
theta_dot_5_n = [];
pos_x = [];
pos_y = [];
pos_z = [];
vel_x = [];
vel_y = [];
vel_z = [];

k = 1;
for i = 0:N-1
    p1 = poly_coef_theta_1(1+4*i:4*i+4);
    p2 = poly_coef_theta_2(1+4*i:4*i+4);
    p3 = poly_coef_theta_3(1+4*i:4*i+4);
    p4 = poly_coef_theta_4(1+4*i:4*i+4);
    p5 = poly_coef_theta_5(1+4*i:4*i+4);

    for t = 0:tstep:ts(i+1)
        theta_1_n(k) = polyval(p1, t);
        theta_2_n(k) = polyval(p2, t);
        theta_3_n(k) = polyval(p3, t);
        theta_4_n(k) = polyval(p4, t);
        theta_5_n(k) = polyval(p5, t);

        theta_dot_1_n(k) = polyval(polyder(p1), t);
        theta_dot_2_n(k) = polyval(polyder(p2), t);
        theta_dot_3_n(k) = polyval(polyder(p3), t);
        theta_dot_4_n(k) = polyval(polyder(p4), t);
        theta_dot_5_n(k) = polyval(polyder(p5), t);

        % Get FK
        [pos_x(k),pos_y(k),pos_z(k)] = getFK(theta_1_n(k),theta_2_n(k),theta_3_n(k),theta_4_n(k),theta_5_n(k));
        [vel_x(k),vel_y(k),vel(k)] = getFK(theta_dot_1_n(k),theta_dot_2_n(k),theta_dot_3_n(k),theta_dot_4_n(k),theta_dot_5_n(k));
        k = k + 1;
    end
end

figure(1);
% Draw keyframes
plot(0,keyframe_theta(:,1), 'marker','square','color','k',MarkerSize=8); hold on;
plot(200,keyframe_theta(:,2), 'marker','square','color','k',MarkerSize=8); hold on;
plot(400,keyframe_theta(:,3), 'marker','square','color','k',MarkerSize=8); hold on;
plot(600,keyframe_theta(:,4), 'marker','square','color','k',MarkerSize=8); hold on;
plot(800,keyframe_theta(:,5), 'marker','square','color','k',MarkerSize=8); hold on;
```

```
% Draw control sequence
plot(theta_1_n,'DisplayName','theta_1_n','color','r',LineWidth=2);hold on;
plot(theta_2_n,'DisplayName','theta_2_n','color','g',LineWidth=2);
plot(theta_3_n,'DisplayName','theta_3_n','color','b',LineWidth=2);
plot(theta_4_n,'DisplayName','theta_4_n','color','k',LineWidth=2);
plot(theta_5_n,'DisplayName','theta_5_n','color','cyan',LineWidth=2);hold off;

xlabel('timestamp [1]');
ylabel('Joint Angle [rad]');
grid on;

% 2D Position Trajectory
figure(2);
% x
plot(0,500, 'marker','square','color','k',MarkerSize=8); hold on;
plot(200,350, 'marker','square','color','k',MarkerSize=8); hold on;
plot(400,200, 'marker','square','color','k',MarkerSize=8); hold on;
plot(600,150, 'marker','square','color','k',MarkerSize=8); hold on;
plot(800,400, 'marker','square','color','k',MarkerSize=8); hold on;

% y
plot(0,250, 'marker','square','color','k',MarkerSize=8); hold on;
plot(200,300, 'marker','square','color','k',MarkerSize=8); hold on;
plot(400,400, 'marker','square','color','k',MarkerSize=8); hold on;
plot(600,200, 'marker','square','color','k',MarkerSize=8); hold on;
plot(800,0, 'marker','square','color','k',MarkerSize=8); hold on;

% z
plot(0,100, 'marker','square','color','k',MarkerSize=8); hold on;
plot(200,250, 'marker','square','color','k',MarkerSize=8); hold on;
plot(400,300, 'marker','square','color','k',MarkerSize=8); hold on;
plot(600,150, 'marker','square','color','k',MarkerSize=8); hold on;
plot(800,100, 'marker','square','color','k',MarkerSize=8); hold on;


plot(pos_x,'DisplayName','theta_1_n','color','r',LineWidth=2);hold on;
plot(pos_y,'DisplayName','theta_2_n','color','g',LineWidth=2);
plot(pos_z,'DisplayName','theta_5_n','color','cyan',LineWidth=2);hold off;

xlabel('timestamp [1]');
ylabel('Position [mm]');
grid on;


% 3D Position Trajectory
figure(3);
plot3(500,250,100, 'marker','square','color','k',MarkerSize=12);hold on;
plot3(350,300,250, 'marker','square','color','k',MarkerSize=12);hold on;
plot3(200,400,300, 'marker','square','color','k',MarkerSize=12);hold on;
plot3(150,200,150, 'marker','square','color','k',MarkerSize=12);hold on;
```

```matlab
plot3(400,0,100, 'marker','square','color','k',MarkerSize=12);hold on;

% plot trajectory
p = plot3(pos_x,pos_y,pos_z,'color','g',LineWidth=2);
% set color
int=size(pos_x',1); % get number of rows
cd = [uint8(jet(int)*255) uint8(ones(int,1)) ].';
drawnow
set(p.Edge, 'ColorBinding','interpolated', 'ColorData',cd)

% plot3(pos_x,pos_y,pos_z,'color','g',LineWidth=2);
xlabel('x [mm]');
ylabel('y [mm]');
zlabel('z [mm]');
grid on;

% Joint velocity
figure(4);


% Draw control sequence
plot(theta_dot_1_n,'DisplayName','theta_1_n','color','r',LineWidth=2);hold on;
plot(theta_dot_2_n,'DisplayName','theta_2_n','color','g',LineWidth=2);hold on;
plot(theta_dot_3_n,'DisplayName','theta_3_n','color','b',LineWidth=2);hold on;
plot(theta_dot_4_n,'DisplayName','theta_4_n','color','k',LineWidth=2);hold on;
plot(theta_dot_5_n,'DisplayName','theta_5_n','color','cyan',LineWidth=2);hold on;

% % Draw keyframes
plot(0,0, 'marker','square','color','k',MarkerSize=8); hold on;
plot(200,0, 'marker','square','color','k',MarkerSize=8); hold on;
plot(400,0, 'marker','square','color','k',MarkerSize=8); hold on;
plot(600,0, 'marker','square','color','k',MarkerSize=8); hold on;
plot(800,0, 'marker','square','color','k',MarkerSize=8); hold on;

xlabel('timestamp [1]');
ylabel('Joint Velocity [rad/s]');
grid on;
```

Appendix C MATLAB Codes of Straight Line Trajectory

```
%% Straight Motion Trajectory
% Author: Ziniu Wu
clc;
clear all;


keyframe = [500,250,100;
            350,300,250;
            200,400,300;
            150,200,150;
            400,0,100];
% Joint values
theta = [0.4636   -0.3717   0.7435   -0.3717         0;
         0.7086   -0.3155   1.2603   -0.9447         0;
         1.1071   -0.1949   1.2310   -1.0360         0;
         0.9273   -0.9345   2.2638   -1.3293         0;
              0   -0.8411   1.6821   -0.8411         0];
keyframe_theta = transpose(theta);

v = 5; % straight line velocity
seg = 4; % segments
tstep = 1; % time steps
k = 1;
t_sum = 0; % init

ts = [];
direction = [];


pos_traj = [keyframe(1,:)];
vel_traj = [];
theta_1_n = [];
theta_2_n = [];
theta_3_n = [];
theta_4_n = [];
theta_5_n = [];
theta_1= [];
theta_2= [];
theta_3= [];
theta_4= [];
theta_5= [];

last_pos = keyframe(1,:);
timestamp = [];

for i=1:1:seg
```

```
    diffVec  = keyframe(i+1,:) - keyframe(i,:);
    direction_temp = diffVec/norm(diffVec);
    direction  = [direction; direction_temp];
    distance  = norm(diffVec);
    ts(i)  = distance/v;
    t_sum = t_sum + ts(i);
end


% t_sum = 0;
for i=1:1:seg

    for t  = 0:tstep:ts(i)
        incremental = direction(i,:)*v;
        last_pos = last_pos + incremental;
        current_vel = incremental;
        pos_traj = [pos_traj;last_pos];
        vel_traj  = [vel_traj;current_vel];

        [theta_1_n,theta_2_n,theta_3_n,theta_4_n,theta_5_n] = getIK(last_pos(1),last_pos
         (2),last_pos(3),0,0);
        theta_1 = [theta_1;theta_1_n];
        theta_2 = [theta_2;theta_2_n];
        theta_3 = [theta_3;theta_3_n];
        theta_4 = [theta_4;theta_4_n];
        theta_5 = [theta_5;theta_5_n];


    end

end

pos_traj = pos_traj';
vel_traj  = vel_traj';

%% 3D Position Trajectory
figure(1);
plot3(500,250,100, 'marker','square','color','g',MarkerSize=12);hold on;
plot3(350,300,250, 'marker','square','color','k',MarkerSize=12);hold on;
plot3(200,400,300, 'marker','square','color','k',MarkerSize=12);hold on;
plot3(150,200,150, 'marker','square','color','k',MarkerSize=12);hold on;
plot3(400,0,100, 'marker','square','color','r',MarkerSize=12);hold on;

% plot trajectory
p = plot3(pos_traj(1,:),pos_traj(2,:),pos_traj(3,:),'color','g',LineWidth=2);hold on;
% set color
int=size(pos_traj',1); % get number of rows
cd = [uint8(jet(int)*255) uint8(ones(int,1))].';
drawnow
set(p.Edge, 'ColorBinding','interpolated', 'ColorData',cd)

```

```
% plot3(pos_x,pos_y,pos_z,'color','g',LineWidth=2);
xlabel('x [mm]');
ylabel('y [mm]');
zlabel('z [mm]');
grid on;

%% 2D Position Trajectory
figure(2);
% x
plot(0,500, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1),350, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2),200, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2)+ts(3),150, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2)+ts(3)+ts(4),400, 'marker','square','color','k',MarkerSize=8); hold on;

% y
plot(0,250, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1),300, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2),400, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2)+ts(3),200, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2)+ts(3)+ts(4),0, 'marker','square','color','k',MarkerSize=8); hold on;

% z
plot(0,100, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1),250, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2),300, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2)+ts(3),150, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2)+ts(3)+ts(4),100, 'marker','square','color','k',MarkerSize=8); hold on;


plot(pos_traj(1,:),'DisplayName','theta_1_n','color','r',LineWidth=2);hold on;
plot(pos_traj(2,:),'DisplayName','theta_2_n','color','g',LineWidth=2);
plot(pos_traj(3,:),'DisplayName','theta_5_n','color','cyan',LineWidth=2);hold off;

xlabel('timestamp [s]');
ylabel('Position [mm]');
grid on;

%% 2D Velocity Trajectory
figure(3);
% % x
% plot(0,500, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1),350, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2),200, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3),150, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3)+ts(4),400, 'marker','square','color','k',MarkerSize=8); hold on;
%
% % y
% plot(0,250, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1),300, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2),400, 'marker','square','color','k',MarkerSize=8); hold on;
```

```
% plot(ts(1)+ts(2)+ts(3),200, 'marker','square',' color ',' k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3)+ts(4),0, 'marker','square',' color ',' k',MarkerSize=8); hold on;
%
% % z
% plot(0,100, 'marker','square',' color ',' k',MarkerSize=8); hold on;
% plot(ts(1),250, 'marker','square',' color ',' k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2),300, 'marker','square',' color ',' k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3),150, 'marker','square',' color ',' k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3)+ts(4),100, 'marker','square',' color ',' k',MarkerSize=8); hold on;


plot(vel_traj(1,:),'DisplayName','theta_1_n','color','r',LineWidth=2);hold on;
plot(vel_traj(2,:),'DisplayName','theta_2_n','color','g',LineWidth=2);
plot(vel_traj(3,:),'DisplayName','theta_5_n','color','cyan',LineWidth=2);hold off;

xlabel('timestamp [s]');
ylabel('Velocity [mm]');
grid on;

% Draw theta
figure(4);
% Draw keyframes
plot(0,keyframe_theta(:,1), 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1),keyframe_theta(:,2), 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2),keyframe_theta(:,3), 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2)+ts(3),keyframe_theta(:,4), 'marker','square','color','k',MarkerSize=8);
        hold on;
plot(ts(1)+ts(2)+ts(3)+ts(4),keyframe_theta(:,5), 'marker','square','color','k',MarkerSize
        =8); hold on;

% Draw control sequence
plot(theta_1,'DisplayName','theta_1_n','color','r',LineWidth=2);hold on;
plot(theta_2,'DisplayName','theta_2_n','color','g',LineWidth=2);
plot(theta_3,'DisplayName','theta_3_n','color','b',LineWidth=2);
plot(theta_4,'DisplayName','theta_4_n','color','k',LineWidth=2);
plot(theta_5,'DisplayName','theta_5_n','color','cyan',LineWidth=2);hold off;

xlabel('timestamp [s]');
ylabel('Joint Angle [rad]');
grid on;
```

Appendix D MATLAB Codes of Avoidance Trajectory

```
%% Avoidance Trajectory
% Author: Ziniu Wu
clc;
clear all;

% avoidance planner
obstacle = [500,125,100];
R = 50; % radius of the obstacle
D = 100; % safe distance

L = sqrt((200-150)^2+(400-200)^2);
H = 300 - 150;

look_ahead_point = [500-R-D,125,100];

keyframe = [500,250,100;
            look_ahead_point;
             500,0,100];



% Joint values
theta = [getIK(500,250,100,0,0);
         getIK(500-R-D,125,100,0,0);
         getIK(500,0,100,0,0) ];
keyframe_theta = transpose(theta);

v = 1; % straight line velocity
seg = 1+1; % segments
tstep = 1; % time steps
k = 1;
t_sum = 0; % init

ts = [];
 direction  = [];


pos_traj = [keyframe(1,:)];
vel_traj = [];
theta_1_n = [];
theta_2_n = [];
theta_3_n = [];
theta_4_n = [];
theta_5_n = [];
theta_1= [];
theta_2= [];
```

```
theta_3= [];
theta_4= [];
theta_5= [];

last_pos = keyframe(1,:);
timestamp = [];



for i=1:1:seg
    diffVec = keyframe(i+1,:) - keyframe(i,:);
    direction_temp = diffVec/norm(diffVec);
    direction = [direction; direction_temp];
    distance = norm(diffVec);
    ts(i) = distance/v;
    t_sum = t_sum + ts(i);
end


% t_sum = 0;
for i=1:1:seg

    for t = 0:tstep:ts(i)
        incremental = direction(i,:)*v;
        last_pos = last_pos + incremental;
        current_vel = incremental;
        pos_traj = [pos_traj;last_pos];
        vel_traj = [vel_traj;current_vel];

        [theta_1_n,theta_2_n,theta_3_n,theta_4_n,theta_5_n] = getIK(last_pos(1),last_pos
         (2),last_pos(3),0,0);
        theta_1 = [theta_1;theta_1_n];
        theta_2 = [theta_2;theta_2_n];
        theta_3 = [theta_3;theta_3_n];
        theta_4 = [theta_4;theta_4_n];
        theta_5 = [theta_5;theta_5_n];


    end

end

pos_traj = pos_traj';
vel_traj = vel_traj';

%% 3D Position Trajectory
figure(1);
plot3(500,250,100, 'marker','square','color','g',MarkerSize=12);hold on;
plot3(500,0,100, 'marker','square','color','r',MarkerSize=12);hold on;
plot3(look_ahead_point(1),look_ahead_point(2),look_ahead_point(3), 'marker','square','
        color','b',MarkerSize=12);hold on;
```

```
drawObstacle(obstacle(1),obstacle(2),obstacle(3),R);hold on;

% plot trajectory
p = plot3(pos_traj(1,:),pos_traj(2,:),pos_traj(3,:),'color','g',LineWidth=2);hold on;
% set color
int=size(pos_traj',1); % get number of rows
cd = [uint8(jet(int)*255) uint8(ones(int,1)) ].';
drawnow
set(p.Edge, 'ColorBinding','interpolated', 'ColorData',cd)

% plot3(pos_x,pos_y,pos_z,'color','g',LineWidth=2);
xlabel('x [mm]');
ylabel('y [mm]');
zlabel('z [mm]');
grid on;

%% 2D Position Trajectory
figure(2);
% x
plot(0,500, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1),look_ahead_point(1), 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2),500, 'marker','square','color','k',MarkerSize=8); hold on;

% y
plot(0,250, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1),look_ahead_point(2), 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2),0, 'marker','square','color','k',MarkerSize=8); hold on;

% z
plot(0,100, 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1),look_ahead_point(3), 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2),100, 'marker','square','color','k',MarkerSize=8); hold on;


plot(pos_traj(1,:),'DisplayName','theta_1_n','color','r',LineWidth=2);hold on;
plot(pos_traj(2,:),'DisplayName','theta_2_n','color','g',LineWidth=2);
plot(pos_traj(3,:),'DisplayName','theta_5_n','color','cyan',LineWidth=2);hold off;

xlabel('timestamp [s]');
ylabel('Position [mm]');
grid on;

%% 2D Velocity Trajectory
figure(3);
% % x
% plot(0,500, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1),350, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2),200, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3),150, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3)+ts(4),400, 'marker','square','color','k',MarkerSize=8); hold on;
%
```

```matlab
% % y
% plot(0,250, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1),300, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2),400, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3),200, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3)+ts(4),0, 'marker','square','color','k',MarkerSize=8); hold on;
%
% % z
% plot(0,100, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1),250, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2),300, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3),150, 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3)+ts(4),100, 'marker','square','color','k',MarkerSize=8); hold on;


plot(vel_traj(1,:),'DisplayName','theta_1_n','color','r',LineWidth=2);hold on;
plot(vel_traj(2,:),'DisplayName','theta_2_n','color','g',LineWidth=2);
plot(vel_traj(3,:),'DisplayName','theta_5_n','color','cyan',LineWidth=2);hold off;

xlabel('timestamp [s]');
ylabel('Velocity [mm]');
grid on;

% Draw theta
figure(4);
% Draw keyframes
plot(0,keyframe_theta(:,1), 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1),keyframe_theta(:,2), 'marker','square','color','k',MarkerSize=8); hold on;
plot(ts(1)+ts(2),keyframe_theta(:,3), 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3),keyframe_theta(:,4), 'marker','square','color','k',MarkerSize=8); hold on;
% plot(ts(1)+ts(2)+ts(3)+ts(4),keyframe_theta(:,5), 'marker','square','color','k',MarkerSize=8); hold on;

% Draw control sequence
plot(theta_1,'DisplayName','theta_1_n','color','r',LineWidth=2);hold on;
plot(theta_2,'DisplayName','theta_2_n','color','g',LineWidth=2);
plot(theta_3,'DisplayName','theta_3_n','color','b',LineWidth=2);
plot(theta_4,'DisplayName','theta_4_n','color','k',LineWidth=2);
plot(theta_5,'DisplayName','theta_5_n','color','cyan',LineWidth=2);hold off;

xlabel('timestamp [s]');
ylabel('Joint Angle [rad]');
grid on;
```

Appendix E MATLAB Codes of Utility Functions in Part A

```

%% Test for FK for Lynvmotion arm
% Author: Ziniu Wu

% Forward kinematics
clc
clear all
syms theta1 theta2 theta3 theta4 theta5 a2 a3 d1 d5

% setup DH-table (modified DH)
alpha0 = 0; % link twist angle
a0 = 0; % link legth
% theta1 = 0; % link rotation angle
d1 = 100; % link offset distance

alpha1 = pi/2;
a1 = 0;
% theta2 = -0.7854;
d2 = 0;

alpha2 = 0;
a2 = 300;
% theta3 = 0;
d3 = 0;

alpha3 = 0;
a3 = 300;
% theta4 = 0;
d4 = 0;

alpha4 = pi/2;
a4 = 0;
% theta5 = 0;
d5 = 0;

% Homogeneous transformation (MDH)
T01 = [cos(theta1), -sin(theta1), 0, a0;
       sin(theta1)*cos(alpha0), cos(theta1)*cos(alpha0), -sin(alpha0), -sin(alpha0)*d1;
       sin(theta1)*sin(alpha0), cos(theta1)*sin(alpha0), cos(alpha0), cos(alpha0)*d1;
       0, 0, 0, 1];
T12 = [cos(theta2), -sin(theta2), 0, a1;
       sin(theta2)*cos(alpha1), cos(theta2)*cos(alpha1), -sin(alpha1), -sin(alpha1)*d2;
       sin(theta2)*sin(alpha1), cos(theta2)*sin(alpha1), cos(alpha1), cos(alpha1)*d2;
       0, 0, 0, 1];
T23 = [cos(theta3), -sin(theta3), 0, a2;
       sin(theta3)*cos(alpha2), cos(theta3)*cos(alpha2), -sin(alpha2), -sin(alpha2)*d3;
```

```
        sin(theta3)*sin(alpha2), cos(theta3)*sin(alpha2), cos(alpha2), cos(alpha2)*d3;
        0,  0,  0,  1];
T34 = [cos(theta4), -sin(theta4), 0, a3;
        sin(theta4)*cos(alpha3), cos(theta4)*cos(alpha3), -sin(alpha3), -sin(alpha3)*d4;
        sin(theta4)*sin(alpha3), cos(theta4)*sin(alpha3), cos(alpha3), cos(alpha3)*d4;
        0,  0,  0,  1];
T45 = [cos(theta5), -sin(theta5), 0, a4;
        sin(theta5)*cos(alpha4), cos(theta5)*cos(alpha4), -sin(alpha4), -sin(alpha4)*d5;
        sin(theta5)*sin(alpha4), cos(theta5)*sin(alpha4), cos(alpha4), cos(alpha4)*d5;
        0,  0,  0,  1];


% Compound transformation
T = T01*T12*T23*T34*T45;


% End-effector position
end_effector_x = T(1,4);
end_effector_y = T(2,4);
end_effector_z = T(3,4);

% End-effector orientation
end_effector_pitch = theta2+theta3+theta4;
end_effector_roll = theta5;

 effector_state  = [end_effector_x end_effector_y end_effector_z end_effector_pitch end_effector_roll];

% Joint values
theta_K0 = [0.4636   -0.3717    0.7435   -0.3717         0];
theta_K1 = [0.7086   -0.3155    1.2603   -0.9447         0];
theta_K2 = [1.1071   -0.1949    1.2310   -1.0360         0];
theta_K3 = [0.9273   -0.9345    2.2638   -1.3293         0];
theta_K4 = [     0   -0.8411    1.6821   -0.8411         0];

% K0
% theta1=theta_K0(1);
% theta2=theta_K0(2);
% theta3=theta_K0(3);
% theta4=theta_K0(4);
% theta5=theta_K0(5);
%
% effector_state = eval(effector_state);
% disp(effector_state);

% K1
% theta1=theta_K1(1);
% theta2=theta_K1(2);
% theta3=theta_K1(3);
% theta4=theta_K1(4);
% theta5=theta_K1(5);
```

```
%
% effector_state = eval(effector_state);
% disp(effector_state);

% K2
% theta1=theta_K2(1);
% theta2=theta_K2(2);
% theta3=theta_K2(3);
% theta4=theta_K2(4);
% theta5=theta_K2(5);
%
% effector_state = eval(effector_state);
% disp(effector_state);


% K3
% theta1=theta_K3(1);
% theta2=theta_K3(2);
% theta3=theta_K3(3);
% theta4=theta_K3(4);
% theta5=theta_K3(5);
%
% effector_state = eval(effector_state);
% disp(effector_state);


% K4
theta1=theta_K4(1);
theta2=theta_K4(2);
theta3=theta_K4(3);
theta4=theta_K4(4);
theta5=theta_K4(5);

effector_state = eval(effector_state);
disp(effector_state);
```

```

%% Test for IK for Lynvmotion arm
% Author: Ziniu Wu
clc
clear all

syms theta1 theta2 theta3 theta4 theta5 L1 L2 L3 d1
syms x y z psi phi

theta1 = atan2(y,x);
theta2 = atan2((z-d1),(sqrt(x^2+y^2)))-acos((L1^2+x^2+y^2+(z-d1)^2-L2^2)/(2*L1*sqrt(x^2+y^2+(z-d1)^2)));
theta3 = acos((x^2+y^2+(z-d1)^2-L1^2-L2^2)/(2*L1*L2));
theta4 = psi-(atan2((z-d1),(sqrt(x^2+y^2)))-acos((L1^2+x^2+y^2+(z-d1)^2-L2^2)/(2*L1*
```

```
        sqrt(x^2+y^2+(z-d1)^2))))-acos((x^2+y^2+(z-d1)^2-L1^2-L2^2)/(2*L1*L2));
theta5 = phi;

L1 = 300;
L2 = 300;
L3 = 0;
d1 = 100;

% off commit one of the following

% K0
% x = 500;
% y = 250;
% z =  100;
% psi =  0;
% phi = 0;

% K1
% x = 350;
% y = 300;
% z =  250;
% psi =  0;
% phi = 0;

% K2
% x = 200;
% y = 400;
% z =  300;
% psi =  0;
% phi = 0;

% K3
% x = 150;
% y = 200;
% z =  150;
% psi = 0;
% phi = 0;

% K4
x = 400;
y = 0;
z =  100;
psi  = 0;
phi = 0;

theta = [theta1 theta2 theta3 theta4 theta5];
theta = eval(theta);
disp(theta);
```

```
%% FK Utility Function for Lynvmotion arm
```

```matlab
% Author: Ziniu Wu
function [x,y,z] = getFK(theta1, theta2, theta3, theta4, theta5)
%GETFK 此处显示有关此函数的摘要
%   此处显示详细说明
% setup DH-table (modified DH)
alpha0 = 0; % link twist angle
a0 = 0; % link legth
% theta1 = 0; % link rotation angle
d1 = 100; % link offset distance

alpha1 = pi/2;
a1 = 0;
% theta2 = -0.7854;
d2 = 0;

alpha2 = 0;
a2 = 300;
% theta3 = 0;
d3 = 0;

alpha3 = 0;
a3 = 300;
% theta4 = 0;
d4 = 0;

alpha4 = pi/2;
a4 = 0;
% theta5 = 0;
d5 = 0;


% Homogeneous transformation (MDH)
T01 = [cos(theta1), -sin(theta1), 0, a0;
        sin(theta1)*cos(alpha0), cos(theta1)*cos(alpha0), -sin(alpha0), -sin(alpha0)*d1;
        sin(theta1)*sin(alpha0), cos(theta1)*sin(alpha0), cos(alpha0), cos(alpha0)*d1;
        0, 0, 0, 1];
T12 = [cos(theta2), -sin(theta2), 0, a1;
        sin(theta2)*cos(alpha1), cos(theta2)*cos(alpha1), -sin(alpha1), -sin(alpha1)*d2;
        sin(theta2)*sin(alpha1), cos(theta2)*sin(alpha1), cos(alpha1), cos(alpha1)*d2;
        0, 0, 0, 1];
T23 = [cos(theta3), -sin(theta3), 0, a2;
        sin(theta3)*cos(alpha2), cos(theta3)*cos(alpha2), -sin(alpha2), -sin(alpha2)*d3;
        sin(theta3)*sin(alpha2), cos(theta3)*sin(alpha2), cos(alpha2), cos(alpha2)*d3;
        0, 0, 0, 1];
T34 = [cos(theta4), -sin(theta4), 0, a3;
        sin(theta4)*cos(alpha3), cos(theta4)*cos(alpha3), -sin(alpha3), -sin(alpha3)*d4;
        sin(theta4)*sin(alpha3), cos(theta4)*sin(alpha3), cos(alpha3), cos(alpha3)*d4;
        0, 0, 0, 1];
T45 = [cos(theta5), -sin(theta5), 0, a4;
        sin(theta5)*cos(alpha4), cos(theta5)*cos(alpha4), -sin(alpha4), -sin(alpha4)*d5;
        sin(theta5)*sin(alpha4), cos(theta5)*sin(alpha4), cos(alpha4), cos(alpha4)*d5;
```

```
        0,  0,  0,  1];


% Compound transformation
T = T01*T12*T23*T34*T45;


% End-effector position
x = T(1,4);
y = T(2,4);
z = T(3,4);


end
```

```

%% IK Utility Function for Lynvmotion arm
% Author: Ziniu Wu
function [theta1,theta2,theta3,theta4,theta5] = getIK(x, y, z, psi, phi)


L1 = 300;
L2 = 300;
L3 = 0;
d1 = 100;

theta1 = atan2(y,x);
theta2 = atan2((z-d1),(sqrt(x^2+y^2)))-acos((L1^2+x^2+y^2+(z-d1)^2-L2^2)/(2*L1*sqrt(
    x^2+y^2+(z-d1)^2)));
theta3 = acos((x^2+y^2+(z-d1)^2-L1^2-L2^2)/(2*L1*L2));
theta4 = psi-(atan2((z-d1),(sqrt(x^2+y^2)))-acos((L1^2+x^2+y^2+(z-d1)^2-L2^2)/(2*L1*
    sqrt(x^2+y^2+(z-d1)^2))))-acos((x^2+y^2+(z-d1)^2-L1^2-L2^2)/(2*L1*L2));
theta5 = phi;

theta = [theta1 theta2 theta3 theta4 theta5];

end
```

```

%% Closed form solution of the coefficients of Free Motion Trajectory
% Author: Ziniu Wu

function p = getCoeff(theta_prev,theta_i)

p3 = (theta_prev-theta_i)/4;
p2 = 3*(theta_i-theta_prev)/4;
p1 = 0;
p0 = theta_prev;

```

```
p = [p3 p2 p1 p0];

end
```

```
%% drawsphere
% Author: Ziniu Wu
function drawObstacle(a,b,c,R)
% (a,b,c) is center, R is radius

    [x,y,z] = sphere(20);

    x = R*x;
    y = R*y;
    z = R*z;

    x = x+a;
    y = y+b;
    z = z+c;

%     figure;
    axis equal;
    mesh(x,y,z);
%
%     figure;
%     axis equal;
%     surf(x,y,z);
end
```

Appendix F MATLAB Codes of IK for Parallel Robot

```matlab
%% Part 2a. Parallel Robot
% Inverse Kinematics
% Degree system
% Author: Tunwu Li

%%
clear
clc
close all

%% Initialization
% mm
SA = 170;
L = 130;
R_plat = 130;
R_base = 290;

%Input the orientation a of the robot and the centre point of the platform
alpha = input ('Orientation of the platform: ');
x_c = input('X-coordinate of {C}: ');
y_c = input('Y-coordinate of {C}: ');

%% Points of the platform (CPP_i)
% Degree system
Platform = zeros(2, 3); % row1: X, row2: Y
for i=1:3
    Platform(1, i) = x_c - R_plat * cosd(alpha + 270 + 120*(i-1));
    Platform(2, i) = y_c - R_plat * sind(alpha + 270 + 120*(i-1));
end

%% Points of the base (BPB_i)
% Degree system
Base = zeros(2, 3); % row1: X, row2: Y
for i=1:3
    Base(1, i) = -R_base * cosd(90 + (i-1)*120);
    Base(2, i) = -R_base * sind(90 + (i-1)*120);
end

%% PB_iPP_i
PBPP = zeros(2, 3); % row1: X, row2: Y
for i=1:3
    PBPP(1, i) = Base(1, i) + Platform(1, i);
    PBPP(2, i) = Base(2, i) + Platform(2, i);
end

%% The joints connect upper and lower section (e_i)
e1 = zeros(1, 3);
e2 = zeros(1, 3);
```

```
e3 = zeros(1, 3);
theta = zeros(1, 3);

for i=1:3
    theta(i) = atan2d(PBPP(2, i), PBPP(1, i));
    e1(i) = -SA * 2 * PBPP(2, i);
    e2(i) = -SA * 2 * PBPP(1, i);
    e3(i) = PBPP(1,i)^2 + PBPP(2, i)^2 + SA^2 - L^2;

    % Degree system
    theta(i) = 2 * atan2d(-e1(i) + sqrt(e1(i)^2 + e2(i)^2 - e3(i)^2), e3(i) - e2(i));
end

%% Calculate the positions of the  joints
% Degree system
Joints = zeros(2,3); % row1: X, row2: Y
for i=1:3
    Joints(1, i) = SA * cosd(theta(i)) - Base(1, i);
    Joints(2, i) = SA * sind(theta(i)) - Base(2, i);
end

%% Assign the platform
platform = [Platform(1, :) Platform(1, 1);
            Platform(2, :) Platform(2, 1)];

base = [-Base(1, :) -Base(1, 1);
        -Base(2, :) -Base(2, 1)];

% Links
link_1 = [-Base(1, 1) Joints(1, 1) platform(1, 1);
          -Base(2, 1) Joints(2, 1) platform(2, 1)];

link_2 = [-Base(1, 2) Joints(1, 2) platform(1, 2);
          -Base(2, 2) Joints(2, 2) platform(2, 2)];

link_3 = [-Base(1, 3) Joints(1, 3) platform(1, 3);
          -Base(2, 3) Joints(2, 3) platform(2, 3)];

%% Plot the kinematic model in two positions
% Platform---blue
% Base---red
% Links---black

plot(x_c, y_c, 'blue*') % {C}
hold on % keep the drawing
plot(0, 0, 'red*') % {B}

line(platform(1, :), platform(2, :), 'Color', 'blue', 'linewidth', 2) % enclose platform
line(base(1, :), base(2, :), 'Color', 'red', 'linewidth', 2) % enclose base

% Plot links
```

```
plot(link_1(1, :), link_1(2, :), 'k-o', 'MarkerSize', 3, 'linewidth', 1)
plot(link_2(1, :), link_2(2, :), 'k-o', 'MarkerSize', 3, 'linewidth', 1)
plot(link_3(1, :), link_3(2, :), 'k-o', 'MarkerSize', 3, 'linewidth', 1)

grid on
axis equal
```

Appendix G MATLAB Codes of Workspace for Parallel Robot

```matlab
%% Part 2b. Parallel Robot
% Plot workspace for a given orientation alpha
% Degree system
% Author: Tunwu Li

%%
clear
clc
close all

%% Initialization
% mm
SA = 170;
L = 130;
r = 130;
R = 290;

% Input the orientation of the platform (alpha)
a = input ('Orientation of the platform: ');

%% Points of the base (BPB_i)
% Radian system
Base = zeros(2, 3); % row1: X, row2: Y
for i=1:3
    Base(1, i) = -R * cosd(90 + (i-1)*120);
    Base(2, i) = -R * sind(90 + (i-1)*120);
end

%% Coordinates of {C}
x_c = -150 : 6 : 150;
y_c = -150 : 6 : 150;

%% Preallocate space
platform = zeros(2,3);
e_1 = zeros(1,3);
e_2 = zeros(1,3);
e_3 = zeros(1,3);
t = zeros(1,3);
theta = zeros(1,3);
PBPP = zeros(2,3);

n = 1;
points = zeros(2, 2*180);

%% Calculate possible angles theta_i
% Discard the imaginary number angle
```

```
for j=1:length(x_c)
    for k=1:length(y_c)
        for i=1:3
            platform(1, i) = x_c(j) - r * cosd(270 + a + 120*(i-1));
            platform(2, i) = y_c(k) - r * sind(270 + a + 120*(i-1));

            PBPP(1, i) = Base(1, i) + platform(1, i);
            PBPP(2, i) = Base(2, i) + platform(2, i);

            e_1(i) = -2 * PBPP(2, i) * SA;
            e_2(i) = -2 * PBPP(1, i) * SA;
            e_3(i) = PBPP(1, i)^2 + PBPP(2, i)^2 + SA^2 - L^2;

            t(i) = (-e_1(i) - sqrt(e_1(i)^2 + e_2(i)^2 - e_3(i)^2)) / (e_3(i)-e_2(i));
            theta(i) = 2 * atand(t(i));
        end

        % Determines whether theta is a real number
        if abs(imag(theta)) == 0
            points(1, n) = x_c(j);
            points(2, n) = y_c(k);
            n = n + 1;
        end
    end
end

%% Plot workspace
plot(0, 0, 'red*') % {B}
hold on

base=[-Base(1, :) -Base(1, 1);
      -Base(2, :) -Base(2, 1)];
line(base(1,:),base(2,:), 'Color', 'red', 'linewidth', 2); % enclose base

% workspace
if points ==0
else
    scatter(points(1, :), points(2, :),'b.')
end

grid on
axis equal
```

Appendix H MATLAB Codes of Minimum Snap Trajectory

```
clc;
clear;
H = [4800 720 0 0 0 0 ;
720 576 0 0 0 0  ;
0 0 0 0 0 0 ;
0 0 0 0 0 0 ;
0 0 0 0 0 0 ;
0 0 0 0 0 0  ;];

A = [0 0 0 0 0 1  ;
1 1 1 1 1 1 ;
0 0 0 0 1 0 ;
5 4 3 2 1 0 ;
0 0 0 2 0 0 ;
20 12 6 2 0  0];

d = [0 ;  1  ;  1  ;  0  ;  0;  0];

p_star = quadprog(H, [],  [],   [],   A, d);  % Quadratic Program Solver

t  =  0:0.01:1;
x   = polyval(p_star, t);
axis equal;
plot(t,  x  ,  'r', 'linewidth',2);hold on;
```