University of the West of England

Coursework

Serial and Parallel Robot Kinematics and Mobile Robot Motion Planning

Jinhang Zhu, Wenxing Peng

UWE ID: 19044859, 19044853

supervised by

Dr. Aghil Jafari

December 12, 2019

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**
Jinhang Zhu 19044859	√	√	√	√	√
Wenxing Peng 19044853	√	√	√	√	√

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

Abstract

Kinematics concept has been the essential part of robots controlling. This report of the coursework demonstrates our understanding on robots fundamentals. Denavit-Hartenberg convention is highlighted in kinematics and is further used in the analytical approach to derive the inverse kinematics on both serial and parallel robots. Additionally, an algorithm in motion planning is implemented in simulation. Detailed discussion is given in the following parts.

Contents

List of Tables

List of Figures

Chapter 1

Kinematics of Lynxmotion Arm

1.1 Forward Kinematics

The forward kinematics problem defines that: given the joint variables of the robot, the forward kinematics calculates the position and the orientation of the end-effector. Generally, this chapter introduces the basic knowledge of kinematic chain and illustrates the parameters of the chain, which leads to the significant link frame, the basis of D-H convention. A proximal D-H convention (Craig, 1989) for Lynxmotion AL5B arm will then be presented. This model will then be simulated in MATLAB to demonstrate its usage in direct kinematics. Finally, the sampled workspace of the arm is given to present its reachable space. The corresponding MATLAB file is *FK.m*. **N.B.** Run *specifications.m* before running any MATLAB files of parts below.

1.1.1 DH Representation

Assigning the Coordinate Frames

We follow the algorithm below to assign the frames for Lynxmotion arm (seen in Figure 1.1).

1. Assigning of base frame: link 0
    - $a_0 = 0$, $\alpha_0 = 0$
    - $d_1 = 0$ because it is a revolute joint
2. Identify links and joints. Name the frames by number according to the link they are attach to. Link i has two joints axes: z_i and z_{i+1}.
3. Identify the common normal between z_i and z_{i+1}, or the origin point of intersection of common normal a_i with z_i axis

4. Assign the z_i axis along the i-th joint axis.
5. Assign x_i axis along the common normal a_i in the direction from z_i to z_{i+1}. In the case of $a_i = 0$, x_i is normal to the plane of z_i and z_{i+1} axes.
    - If two z axes are parallel, pick the common normal that is collinear with the common normal of the previous joint.
    - If the z-axes are intersecting, we assign the x-axis along a line perpendicular to the plane formed by the two axes.
6. Assign y_i based on right-hand rule
7. End-effector frame assigning. Since the joint n is revolute, x_n if along x_{n-1} when $\theta_n = 0$, origin n is chosen so that $d_n = 0$.
8. Fill link parameters table

Based on the algorithm above, we deduce the DH convention model of Lynxmotion arm as shown in Figure 1.1.

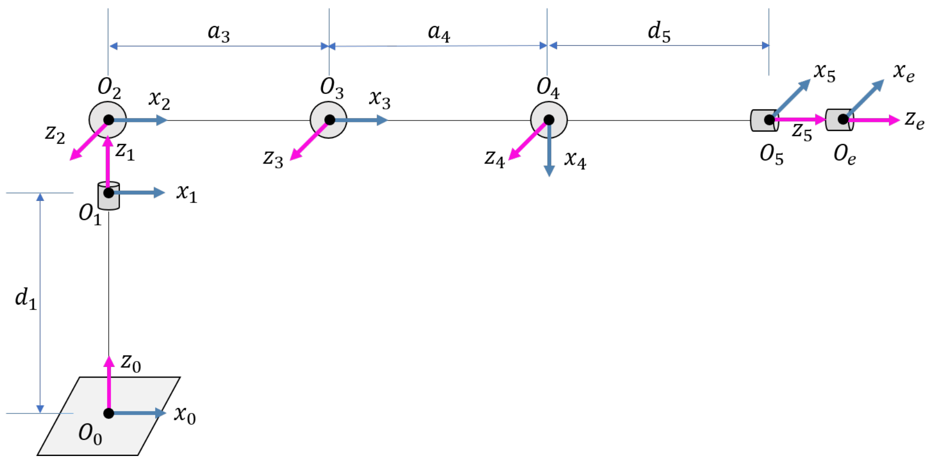


Figure 1.1: Modified D-H Convention Model

D-H Paramemeters

We define that all link lengths are 0.1 (m). Hence, based on the DH model we set up in the last section, we can fill the DH table with the relationships in the model. The DH parameters table is shown as Table 1.1.

The homogeneous transformation from link i-1 to link i is represented as a product of four basic transformations as follows:

$$^{i-1}T_i = R(x_{i-1}, \alpha_{i-1})T(x_{i-1}, a_{i-1})R(z_i, \theta_i)T(z_i, d_i) \tag{1.1}$$

Table 1.1: DH Parameters: Proximal Table

link i	α_{i-1}	a_{i-1}	θ_i	d_i
1	0	0	θ_1	d_1
2	$90°$	0	θ_2	0
3	0	a_3	θ_3	0
4	0	a_4	θ_4	0
5	$-90°$	0	θ_5	d_1

$$^{i-1}T_i = \begin{bmatrix} cos\theta_i & -sin\theta_i & 0 & a_{i-1} \\ sin\theta_i cos\alpha_{i-1} & cos\theta_i cos\alpha_{i-1} & -sin\alpha_{i-1} & -d_i sin\alpha_{i-1} \\ sin\theta_i sin\alpha_{i-1} & cos\theta_i sin\alpha_{i-1} & cos\alpha_{i-1} & d_i cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

Based on the equation 1.2of single homogeneous transformation matrix between two ajoint axes, we can derive all single homogeneous transformations of the arm.

$$^{0}T_1 = \begin{bmatrix} cos\theta_1 & -sin\theta_1 & 0 & 0 \\ sin\theta_1 & cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

$$^{1}T_2 = \begin{bmatrix} cos\theta_2 & -sin\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ sin\theta_2 & cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.4}$$

$$^{2}T_3 = \begin{bmatrix} cos\theta_3 & -sin\theta_3 & 0 & a_3 \\ sin\theta_3 & cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.5}$$

$$^{3}T_4 = \begin{bmatrix} cos\theta_4 & -sin\theta_4 & 0 & a_4 \\ sin\theta_4 & cos\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.6}$$

$$^{4}T_5 = \begin{bmatrix} cos\theta_5 & -sin\theta_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ -sin\theta_5 & -cos\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.7}$$

Since the joint variable of joint 5 θ_5 does not influence the position of the end-effector, we may assume $\theta_5 = 0$. So we have

$$
{}^4T_5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.8}
$$

The position and orientation of the end-effector is given by

$$
H =^0 T_5 =^0 T_1^1 T_2^2 T_3^3 T_4^4 T_5 \tag{1.9}
$$

where,

$$
{}^{i-1}T_i = \begin{bmatrix} {}^iR_{i-1} & {}^id_{i-1} \\ 0 & 1 \end{bmatrix} \tag{1.10}
$$

Table 1.2: Given Sets of Joint Values (in degrees)

q_1	q_2	q_3	q_4
60	-30	45	-60
70	-35	55	-55
80	-40	60	-50
90	-45	65	-45

MATLAB test: Given four sets of joint variables shown in Table 1.2, we can derive the position of the end-effector illustrated in Figure 1.2.

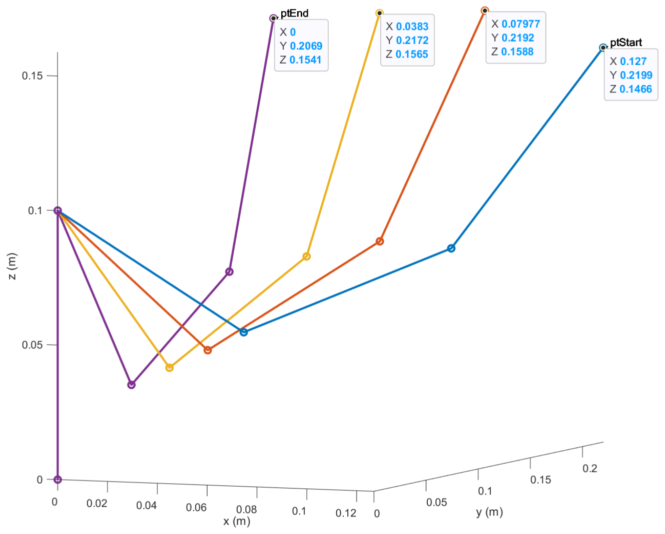


Figure 1.2: Animation of the Arm in Four Given Status

Take $q = [90°, -45°, 65°, -45°](q_5 = 0°)$ as the input joint variables set, we get the final homogeneous transformation matrix 1.11. Then the left top 3 matrix turns out to be the rotation matrix and the right top 3 vector is the translation vector.

$$
{}^0T_5 = \begin{bmatrix} 0 & -1.0000 & 0 & 0 \\ 0.9063 & 1 & 0.4226 & 0.2069 \\ -0.4226 & 0 & 0.9063 & 0.1541 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix} \tag{1.11}
$$

1.1.2 Workspace

Workspace is the space where the end-effector of the robot can reach by at least one approach. We basically are drawing the reachable workspace, where the robot may reach by more than one approach.

Table 1.3: Range of Joint Variables

Joint variables	Range of angles
q_1	[-90, 90]
q_2	[0, 180]
q_3	[-135, 45]
q_4	[-180, 0]

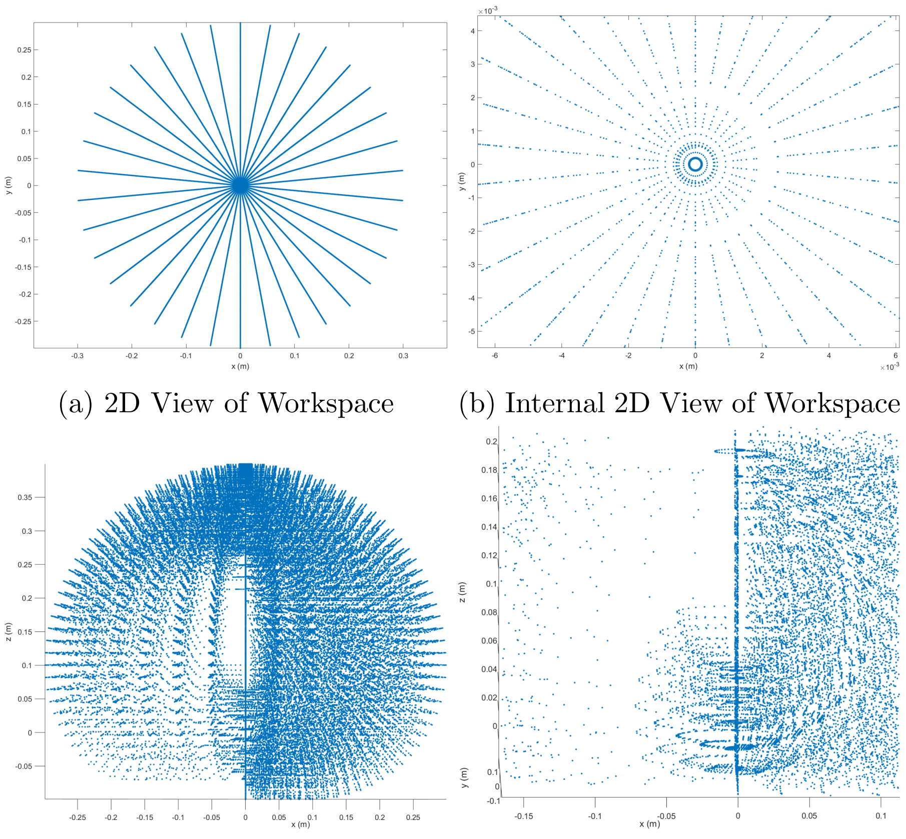
Since joint 5 does not influence the position of the end-effector, we pose limits to other four joint variables. These specifications of limits are found in Table 1.3. For the convenience of the animation, the range is devided into $numStatus = 18$ discrete angles for each revolute joint.

Applying for loop to each joint variable (q_1, q_2, q_3, q_4), making it iterate among the discrete angles, we then use forward kinematics to calculate the homogeneous transformation matrix of the arm and plot the end-effector position in Cartesian coordinate system. The final results of workspace are shown in Figure 1.3.

From the workspace animation, we can reveal that the reachable workspace is in the shape of a body of rotation. 1.3(a) demonstrates the change in joint variable q_1 while a detailed enlarged view in 1.3(b) reveals an uneven distribution of the points. 1.3 describes this phenomenon more vividly. To wrap up, the reachable points are mostly located in $x > 0$ plane.

1.2 Inverse Kinematics

Inverse kinematics defines a process of deriving the values in joint space given the information of position or orientation in end-effector's Cartesian space. The corresponding MATLAB file is $IK.m$.


(a) 2D View of Workspace (b) Internal 2D View of Workspace

(c) 3D View of Workspace (d) Internal 3D View of Workspace

Figure 1.3: Views of Workspace

1.2.1 Analytical Approach

In inverse kinematics, the known information includes the position (given as the 3-by-1 translation vector) of the end-effector and its orientation (given as the angle of elevation ψ). Generally, the given information can form a 4-by-4 homogeneous transformation matrix, seen in 1.12.

$$H = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.12}$$

While the H matrix also equals ${}^0T_5$, which is also the product of every single homogeneous transformation. So we have

$$H \times {}^4T_5{}^{-1} =^0 T_1 \times^1 T_2 \times^2 T_3 \times^3 T_4 \tag{1.13}$$

From 1.3 - 1.6 we have

$$^{0}T_4 = H \times {^{4}T_5}^{-1} \tag{1.14}$$

$$\begin{aligned}
H \times {^{4}T_5}^{-1} &= \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \\
&= \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & -d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} n_x & a_x & -o_x & p_x - a_x d_5 \\ n_y & a_y & -o_y & p_y - a_y d_5 \\ n_z & a_z & -o_z & p_z - a_z d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}$$

Then

$$\begin{cases}
n_x = (c_1c_2c_3 - c_1s_2s_3)c_4 - (c_1c_2s_3 + c_1s_2c_3)s_4 = c_1c_{234} \\
n_y = (s_1c_2c_3 - s_1s_2s_3)c_4 - (s_1c_2s_3 = s_1s_2c_3)s_4 = s_1c_{234} \\
n_z = (s_2c_3 = c_2s_3)c_4 - (s_2s_3 - c_2c_3)s_4 = s_{234} \\
a_x = -(c_1c_2c_3 - c_1s_2s_3)s_4 - (c_1c_2s_3 + c_1s_2c_3)c_4 = -c_1s_{234} \\
a_y = -(s_1c_2c_3 - s_1s_2s_3)s_4 - (s_1c_2s_3 + s_1s_2c_3)c_4 = -s_1s_{234} \\
a_z = -(s_2c_3 + c_2s_3)s_4 - (s_2s_3 - c_2c_3)c_4 = c_{234} \\
-o_x = s_1 \\
-o_y = -c_1 \\
-o_z = 0 \\
p_x - a_xd_5 = (c_1c_2c_3 - c_1s_2s_3)a_4 + c_1c_2a_3 = c_1(c_{23}a_4 + c_2a_3) \\
p_y - a_yd_5 = (s_1c_2c_3 - s_1s_2s_3)a_4 + s_1c_2a_3 = s_1(c_23a_4 + c_2a_3) \\
p_z - a_zd_5 = (s_2c_3 + c_2s_3)a_4 + s_2a_3 + d_1 = s_23a_4 + s_2a_3 + d_1
\end{cases} \tag{1.15}$$

Where $c_i = cos(q_i), c_{ijk} = cosq_i + q_j + q_k$ and this principle is same for s_i, s_{ijk}.

It is known that the given information is the position and orientation of the end-effector, i.e. $[p_x, p_y, p_z]$ and $\psi = q_1 + q_2 + q_3$.

From the geometrical information in p_x, p_y we know that the first joint decides the whole arm's projection in the $x - y$ plane. Thus, we can derive q_1 first:

$$q_1 = atan2(p_y, p_x) \tag{1.16}$$

From the last three equations in 1.15:

$$q_2 = \begin{cases} 2 * atan2(n \pm \sqrt{m^2 + n^2 - e^2}, m + e) & \text{if } m + e \neq 0 \\ 180^\circ & \text{if } m + e = 0 \end{cases} \tag{1.17}$$

then we can derive q_3 seen in 1.18. An significant idea is that in the following tasks, the angle of elevation is constant at 0°, which also means $\psi = q_2 + q_3 + q_4 = -90^\circ$ in our DH convention.

$$q_3 = atan2(k_2, k_1) - q_2 \tag{1.18}$$

$$q_4 = \psi - q_2 - q_3 \tag{1.19}$$

where m, n, e, k_1, k_2 are defined as below:

$$\begin{cases} m &= \frac{p_x - a_x d_5}{c_1} \\ n &= p_z - a_z d_5 - d_1 \\ e &= \frac{m^2 + n^2 + a_3^2 - a_4^2}{2a_3} \\ k_1 &= \frac{m - a_3 s_2}{a_4} \\ k_2 &= \frac{n - a_3 s_2}{a_4} \end{cases} \tag{1.20}$$

1.2.2 Discussion of Solutions

Although the representation of the joint angles is unique, there is a $\pm$ symbol in the equation of q_2, seen in 1.17. Hence, for a given position and orientation, there may be two solutions of q_2 but no more than two solutions. This multiple-solution situation is illustrated in Figure 1.4.

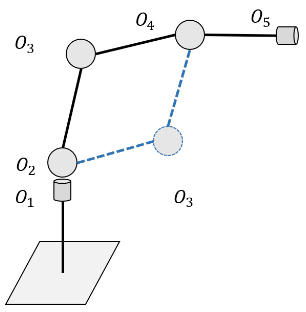


Figure 1.4: Situation with 2 Sets Solutions

For each value of q_2, q_3 and q_4 are unique. Therefore, the number solution for joint variable set q is no more than 2. If the end-effector is outside the workspace, there is no solution. The overall range of the number of solution sets is shown as below.

$$0 \leq \text{number of solutions} \leq 2 \tag{1.21}$$

1.3 Trajectory Planning

1.3.1 Task Planning

We define the five viapoints which the arm has access to with 0 pitch angle as shown in Table 1.4. **N.B.** The angle of elevation is zero when $\psi = -90°$ in our convention. The corresponding MATLAB file is $FK.m$.

Table 1.4: Positions of Viapoints

Viapoints	p_x	p_y	p_z	ψ
1	0.2	0.1	0.1	$-90°$
1	0.2	0.05	0.15	$-90°$
1	0.2	0	0.1	$-90°$
4	0.2	-0.05	0.15	$-90°$
5	0.2	-0.1	0.1	$-90°$

Applying inverse kinematics, we can now draw the animation of the arm in these viapoints, shown in Figure 1.5.

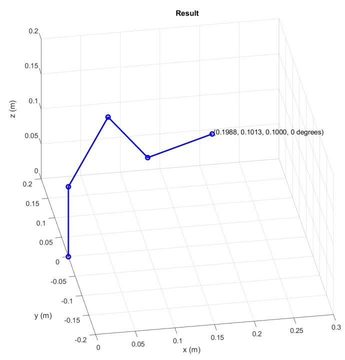
1.3.2 Implement Trajectories

Free Motion: Quintic Polynomial Trajectory

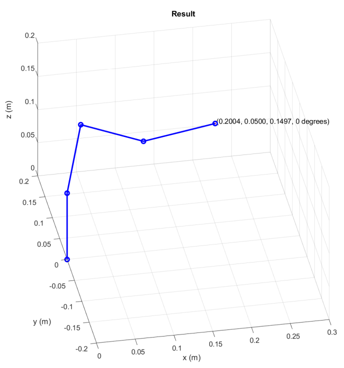
The motion we plan is a quintic polynomial line. Basically, the number of sample points between two ajoint viapoints is $M = 60$, and time interval between every points (including sample points) is 0.01s. The highest power of the polynomial regression is 5. After the polynomial curve fitting of the viapoints, we plot the arm at every point and display the animation shown in Figure 1.6. The corresponding MATLAB file is $FreeMotion.m$.

Trajectory with Linear Segments

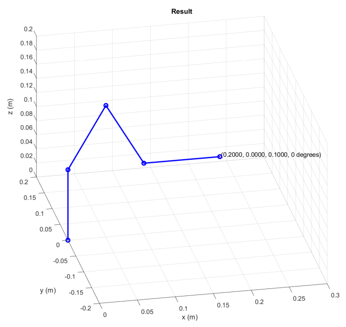
We apply a acceleration-constant-deceleration process to model the straight line trajectory. From Figure 1.7 (a), we define $t_1 = 1/rt_2$. Specify the constant speed



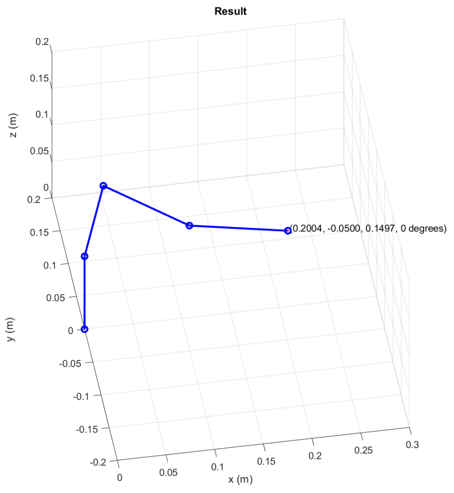
(a) [0.2,0.1,0.1]



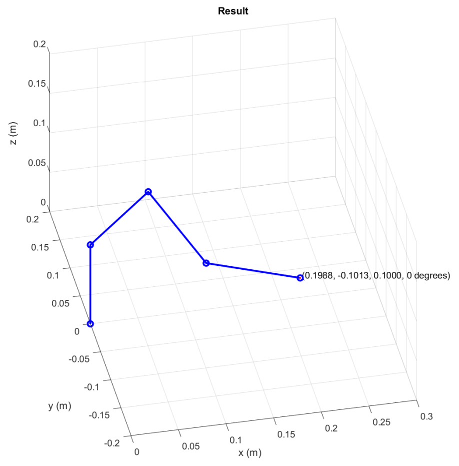
(b) [0.2,0.05,0.15]



(c) [0.2,0,0.1]



(d) [0.2,-0.05,0.15]



(e) [0.2,-0.1,0.1]

Figure 1.5: Animation at Viapoints

is u. Hence $a = u/t_1$. $dist$ is the distance between two viapoints. The corresponding MATLAB file is $StraightTrajectory.m$.

$$s_{distance} \begin{cases} \frac{1}{2}at^2 & = \frac{p_x - a_x d_5}{c_1} \\ \frac{1}{2}at_1^2 + u(t - t_1) & = p_z - a_z d_5 - d_1 \\ dist - \frac{1}{2}a(t_total - t)^2 & = \frac{m^2+n^2+a_3^2-a_4^2}{2a_3} \end{cases} \tag{1.22}$$

Then, the animation and the change in joint values or end-effector valocity is shown in Figure 1.7.

Linear Trajectory with Obstacle Avoidance

We insert spheres between the viapoints. And the arm tries to go to another point to avoid the obstacle before reaching the next viapoint. The animation is shown in Figure 1.8. The corresponding MATLAB file is $ObstacleAvoidance.m$.

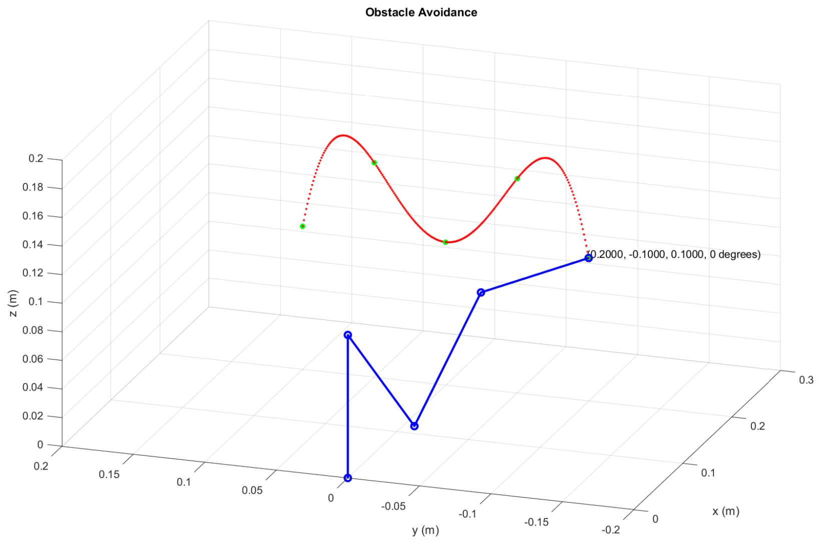
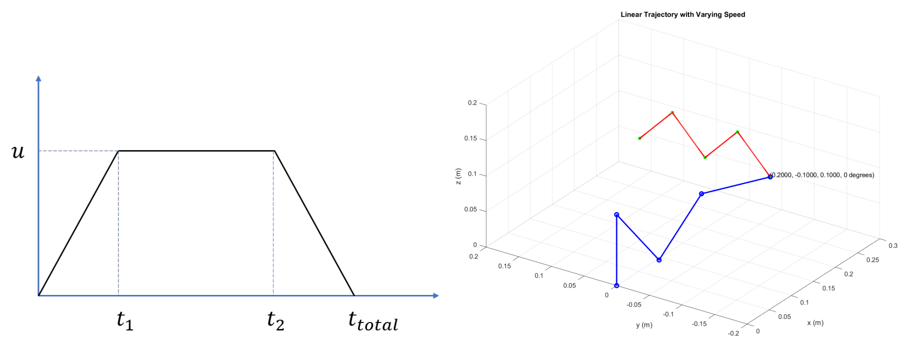
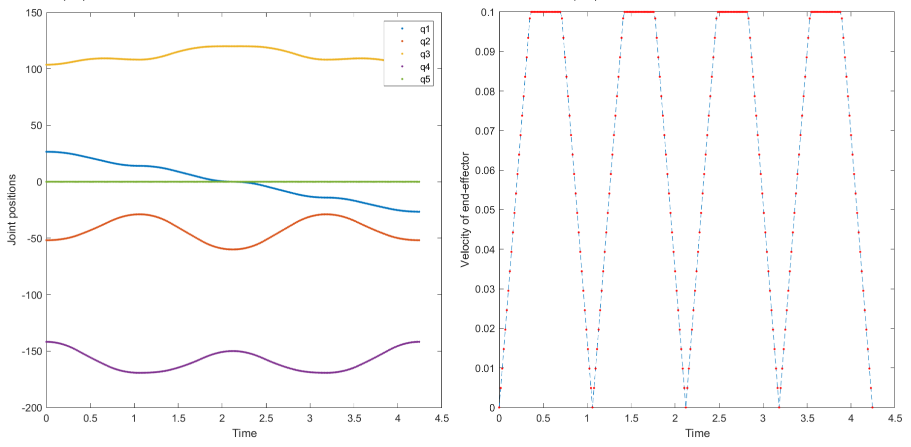


Figure 1.6: Free Motion

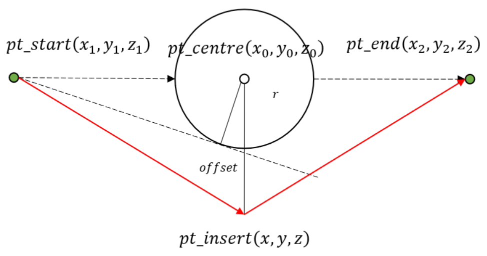


(a) Varied and Even Speed (b) Linear Motion

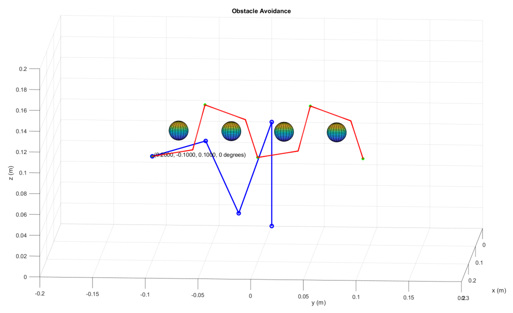


(c) Joint Values (d) Velocity of End-effector

Figure 1.7: Linear Motion Tracjetory



(a) Insert a Point



(b) Avoid the Sphere

Figure 1.8: Obstacle Avoidance

Chapter 2

Kinematics of Planar Parallel Robot

2.1 Inverse Kinematics of Parallel Robot

The parallel robot can be seen in Figure 2.1 as three serial robots with their end-effector coupled by triangular geometry relationship. These serial robots have three arms. The last arm is the triangle platform, which can be seen as a link between PP_i and the center of the triangle. The corresponding MATLAB file is $2position.m$.

The flow chart of the inverse kinematics (Abo-Shanab, 2014) is shown in Figure 2.2.

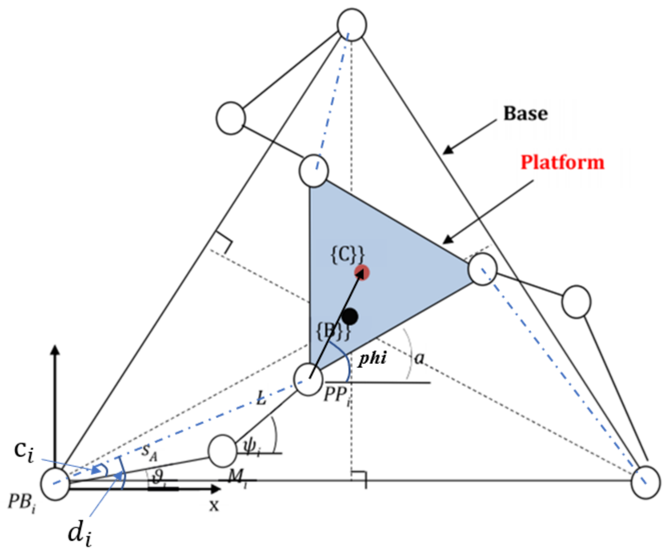


Figure 2.1: Planar Parallel Robot Kinematic Model

1. First, we should specify the design parameter of the parallel robot. See

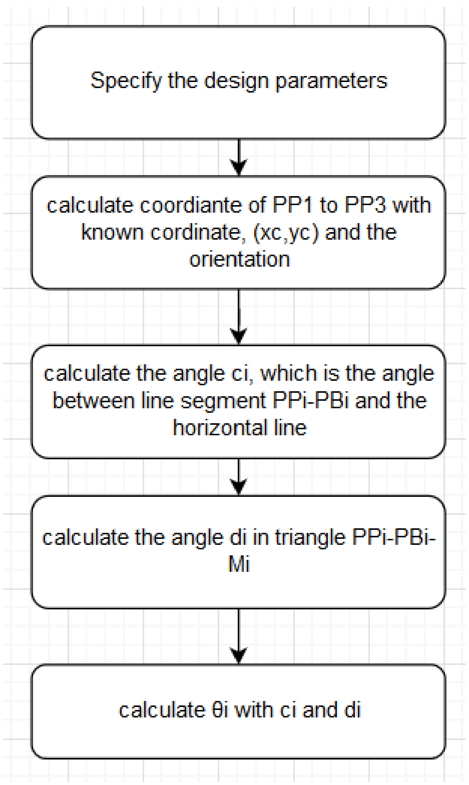

Specify the design parameters
calculate coordiante of PP1 to PP3 with known cordinate, (xc,yc) and the orientation
calculate the angle ci, which is the angle between line segment PPi-PBi and the horizontal line
calculate the angle di in triangle PPi-PBi-Mi
calculate θi with ci and di

Figure 2.2: Inverse Kinematics of Planar Parallel Robot

Table 2.1. Based on the parameters, we can get the coordinate of the vertex of the base.

$$PB_1 = [0, 0] \tag{2.1}$$
$$PB_2 = [R, 0] \tag{2.2}$$
$$PB_3 = [\frac{R}{2}, \frac{\sqrt{3}R}{2}] \tag{2.3}$$

Table 2.1: Design Parameter of Planar Parallel Robot

Parameter— Data Geometry	Value(mm)
S_A, S_B, S_C	170
L	130
r (joint circle radius)	130
R (side length of base)	$290\sqrt{3}$

2. Assume we know the Cartesian coordinate of the center of the platform and the orientation of the platform. With basic knowledge of triangle geometry, we can calculate the vertex of the triangle platform, which is PP_i where $i = 1 : 3$.

$$PP_1 = [x_c - r * cos(\phi_1), y_c - r * sin(\phi_1)] \tag{2.4}$$
$$PP_2 = [x_c + r * cos(\phi_2 - \pi), y_c + r * sin(\phi_2 - \pi)] \tag{2.5}$$
$$PP_3 = [x_c - r * cos(2\pi - \phi_3), y_c + r * sin(2\pi - \phi_3)] \tag{2.6}$$

where $\phi_1 = \alpha + \frac{\pi}{6}, \phi_2 = \alpha + \frac{5\pi}{6}, \phi_3 = \alpha + \frac{3\pi}{2}$.

3. Calculate the angle between the line segment $PP_i - PB_i$ and the horizontal line as follow:

$$c_1 = atan2(y_c - r * sin(\phi_1), x_c - r * cos(\phi_1)) \tag{2.7}$$
$$c_2 = atan2(y_c + r * sin(\phi_2 - \pi)) - R, x_c + r * cos(\phi_2 - \pi) \tag{2.8}$$
$$c_3 = atan2(y_c + r * sin(2\pi - \phi_3) - \frac{\sqrt{3}R}{2} \tag{2.9}$$
$$, x_c - r * cos(2\pi - \phi_3) - \frac{R}{2}) \tag{2.10}$$

4. Calculate the angle d_i in the triangle $\triangle PP_iPB_iM_i$.

$$d_1 = acos(\frac{S_A^2 + L^2 - |PP_1PB_1|^2}{2 * S_A|PP_1PB_1|}) \tag{2.11}$$
$$d_2 = acos(\frac{S_B^2 + L^2 - |PP_2PB_2|^2}{2 * S_B|PP_2PB_2|}) \tag{2.12}$$
$$d_3 = acos(\frac{S_C^2 + L^2 - |PP_3PB_3|^2}{2 * S_C|PP_3PB_3|}) \tag{2.13}$$

where $|PP_iPB_i|$ is the distance between PP_i and PB_i.

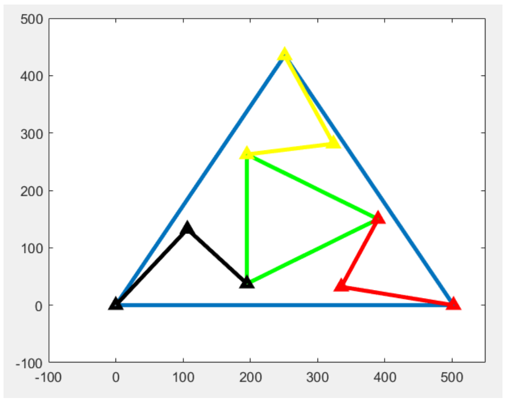
5. After calculating c_i and d_i, we can calculate θ as: $\theta_i = c_i + d_i$ or $\theta_i = c_i - d_i$.

The MATLAB code of implementing this inverse kinematic is attached in Appendix. Solution of two different positions is shown in Figure.

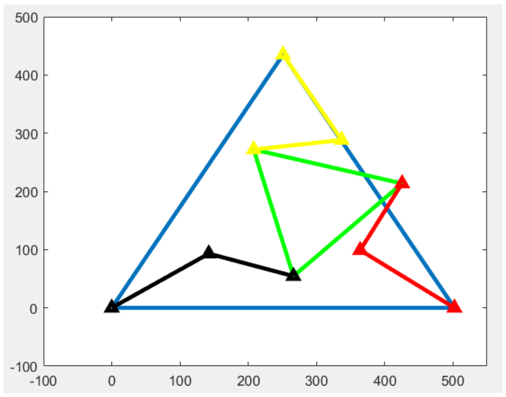
2.2 Workspace for a Given Orientation

When the platform of the robot is in the workspace for a given orientation, the distance between PP_i and PB_i is in the interval $[S - L, S + L]$, where $S = S_A$ or S_B or S_C. Using this judgement condition, we can let (x_c, y_c) traverse in a square which contains the workspace. If (x_c, y_c) satisfy the condition, we plot this point. The corresponding MATLAB file is *workspace.m*.

Using MATLAB to realize this algorithm, the result is as follow:

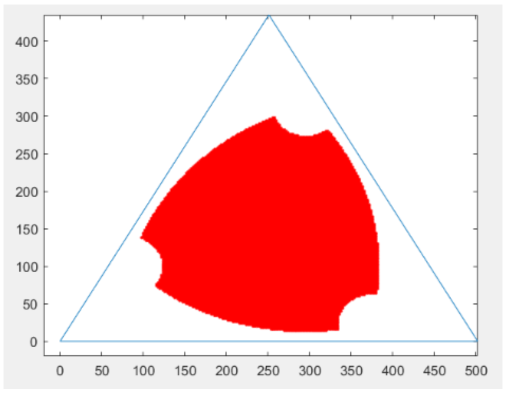


(a) $(x_c, y_c\alpha) = (260, 150, \pi/6)$

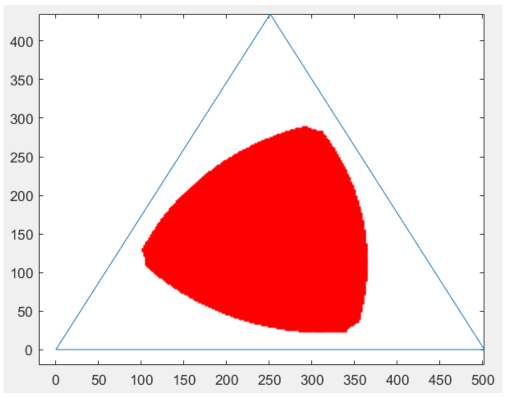


(b) $(x_c, y_c\alpha) = (300, 180, \pi/4)$

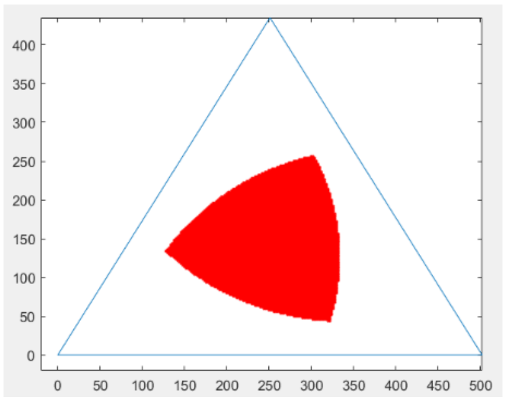
Figure 2.3: Two Position of Parallel Robot



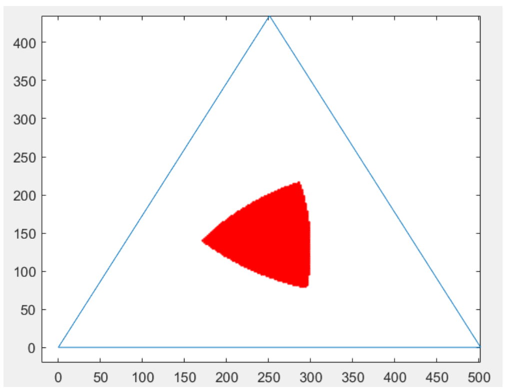
(a) $\alpha = \pi/9$



(b) $\alpha = \pi/6$



(c) $\alpha = \pi/4$



(d) $\alpha = \pi/3$

Figure 2.4: Workspace of Parallel Robot under Different Orientation

Chapter 3

Tangent Bug Algorithm for Motion Planning

3.1 Tangent Bug Algorithm

Tanget Bug is an algorithm for mobile robot to avoid obstacles and to plan a path in the environment, serving as an improvement to the Bug2 algorithm (Choset et al., 2007). The agent in the environment uses a range sensor with certain sensing range to detect any obstacle around it. The sensing curriculum is a 360° scene. When the robot is at position x, the sensing field is represented by $\rho(x, \theta)$, and we have

$$\begin{aligned} \rho(x,\theta) &= \min_{\lambda\in[0,\infty]} d(x, x+\lambda[cos\theta, sin\theta]^T) \\ &\text{such that } x+\lambda[cos\theta, sin\theta]^T \in \bigcup_i \mathcal{WO} \end{aligned} \tag{3.1}$$

However, in real cases, the sensing range is always limited to a upper value R, meaning that the obstacles outside the range will not be detected, i.e. detected at infinite distance. Formally,

$$\rho_R(x,\theta) = \begin{cases} \rho(x,\theta) & \text{, if } \rho(x,\theta) \leq R \\ \infty & \text{, otherwise.} \end{cases} \tag{3.2}$$

The range sensor can detect discontinuities in the field by assigning range distance values to certain angles. For an $interval of continuity$, the points are connected on the boundary of the obstacle. The points where $\rho_R(x, \theta)$ loses its continuity are called endpoints, denoted as O_i, which are just the ends of the detected curves. Seeing Figure 3.1, we define the thick solid curves are the connected points detected within the sensing range, while at other angles the distance is ∞.

Following the convention of Bug1 and Bug2 algorithms, Tangent Bug also iterates between two modes of motion: *motion-to-goal* and *boundary-following*. Under motion-to-goal mode, the robot runs directly to the goal or the endpoint in a straight line. Consider the O_i where $d(O_i, q_{goal} < d(x, q_{goal})$. The robot moves toward the O_i which maximally decreases the heuristic distance $(d(x, O_i) + d(O_i, q_{goal}))$ to the goal. And the robot keeps undergoing motion-to-goal mode until it can no longer decrease $d(x, q_{goal})$. The robot terminates the boundary-following mode when the $d(x, q_{goal}) - R$ is less than $d(O_i, q_{goal})$ for any O_i.

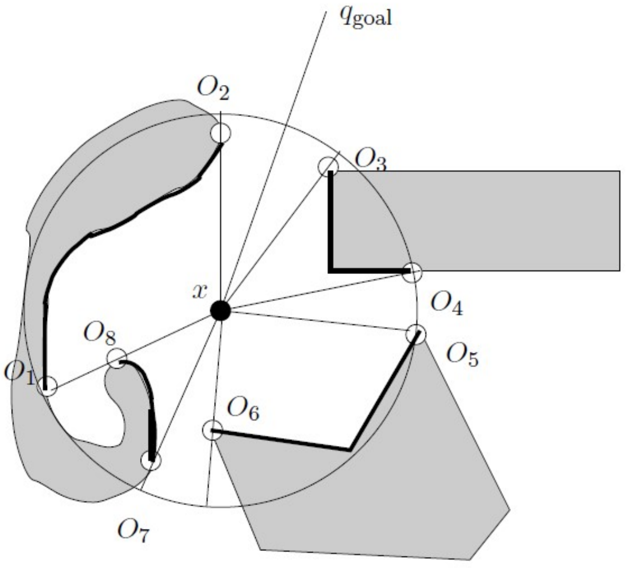


Figure 3.1: The Endpoints of Detected Curves

3.2 Implementation

The process is illustrated in the flowchart below. This section will then introduce the detailed process of implementing the algorithm in MATLAB. The corresponding MATLAB file is *tangentbug.m*.

3.2.1 Specifications of the Environment

Obstacles are created in such a rectangle field which is determined by the start point and the goal point. The point $(min(startx, goalx))$ defines the left bottom and the point $(max(startx, goalx))$ defines the right top. Points are created to

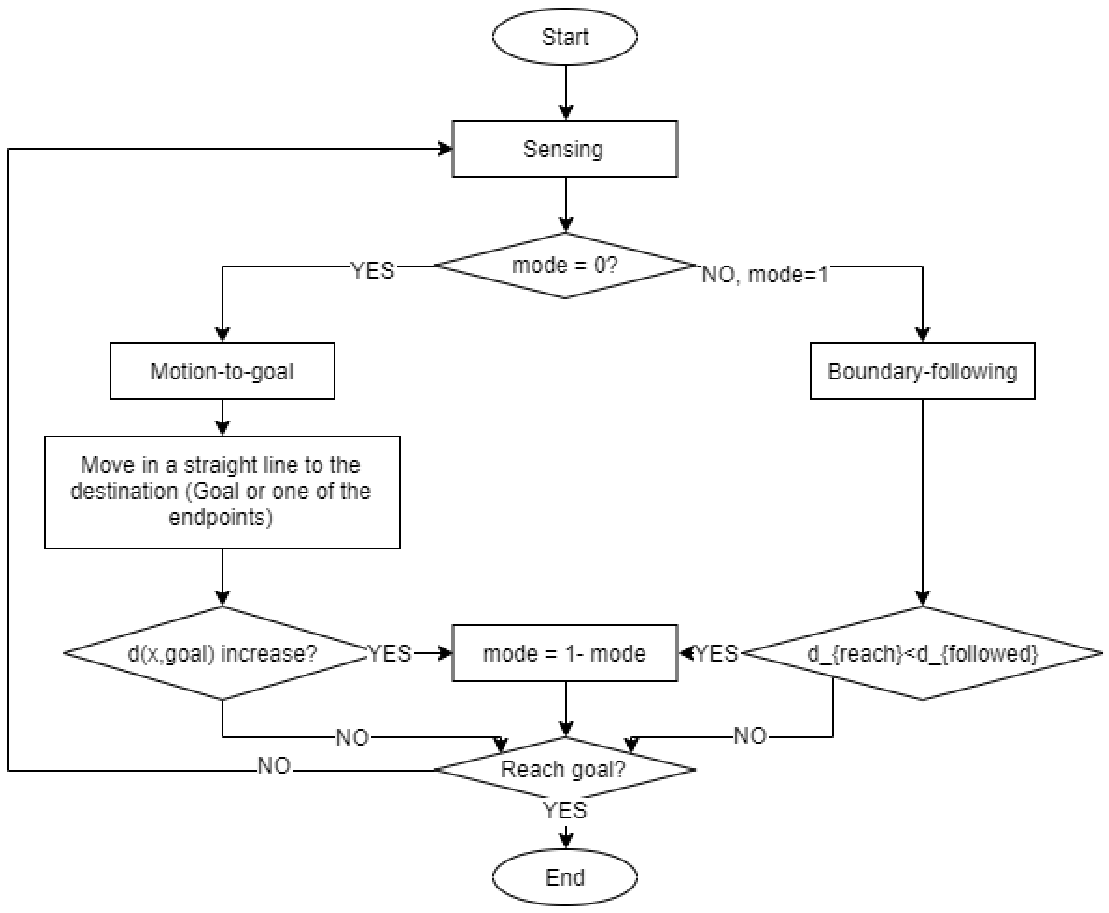


Figure 3.2: Algorithm of Tangent Bug

create some sets of points to form several obstacles in the field, seen in Figure 3.3. (This function is seen in appendix as *createobstacles.m*)

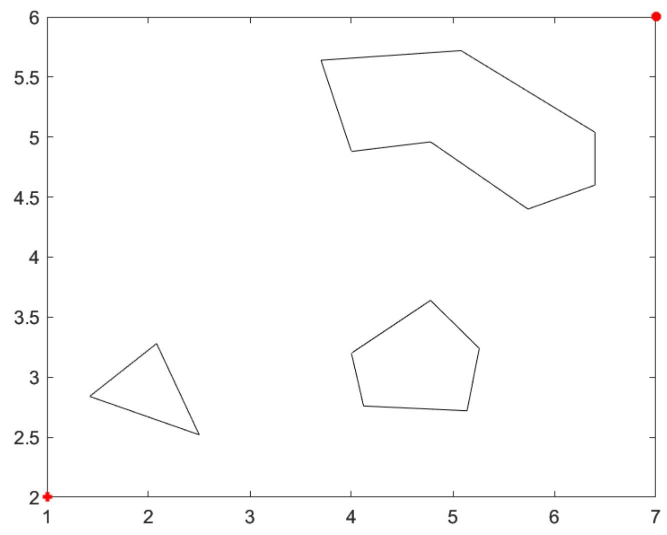


Figure 3.3: Obstacles in the Environment

3.2.2 Agent with Range Sensor

The robot has a range sensor which detects the surroundings by iterating the angles in $[0, 2\pi]$. At each angle, the sensor finds the point with the shortest distance to the robot. If the distance is less than sensing range, then it is stored as part of the continuous curve. If it is not with the range, the corresponding index points to ∞.

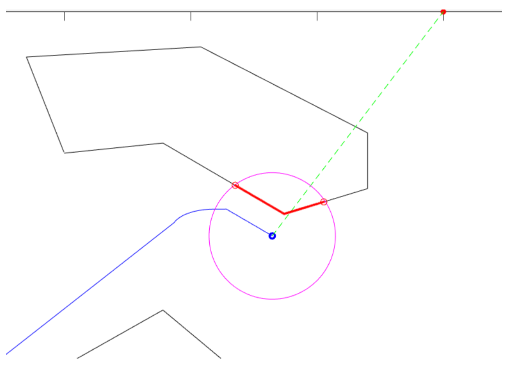
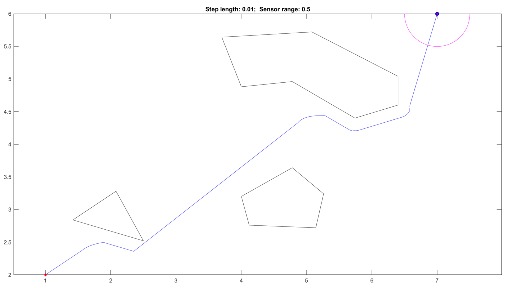


Figure 3.4: Curve with Endpoints Detected

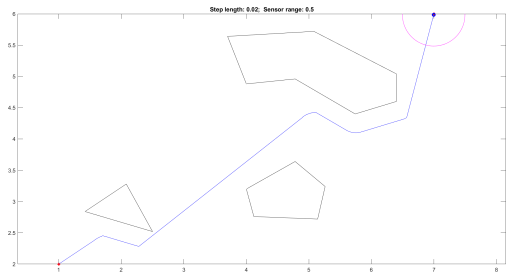
After the iteration of the angles, the robot use the detected curves to determine the endpoints and which one of them is to be the destination at the current status. Particularly, the robot only use the curve which intersects with $m - line$, which is the connected line between start point and goal point. One animation is displayed in Figure 3.4. (This function is seen in appendix as *getcurve.m*)

3.3 Experiments and Results

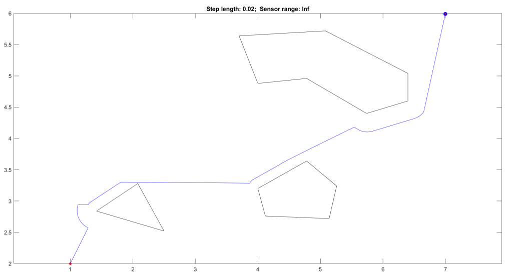
The experiments will cover two sections: the experiments on different sensing range and the evaluation, the experiments on different running steps and the evaluation. The experiments all specify the start point at $(1, 2)$ and the goal point at $(7, 6)$. The results are demonstrated in Figure 3.5.



(a) Sensing range:0.5,step:0.01



(b) Sensing range:0.5,step:0.02



(c) Sensing range:∞,step:0.02

Figure 3.5: Tangent Bug Experiments on Sensing Range and Running Step

From the animations of the Tangent Bug motion planning, we now can demonstrate that Tangent Bug create more smooth path than what Bug1 or Bug2 does. The robot in our design can successfully directly pass by the obstacles which are not blocking it.

One issue in this system lies in the range sensor. Since the robot cannot fully eliminate the points from the **back** of the obstacles due to the discreteness of obstacle walls in MATLAB, the robot cannot really highlight the core of Tangent Bug algorithm: **Discontinuity and Continuity**. That's to say, the curve detected is not really connected due to some points from the behind. However, this issue can be easily solved in real cases because the sensor on the robot only detect what it can see.

Chapter 4

Conclusion

This report has three parts, providing detailed simulation-based designs of robotic kinematics. First, we adopt $Denavit - Hartenberg$ convention to connect the joint space and the Cartesian space of a serial robot called Lynxmotion robot. We then plan the trajectory of the robot end-effector by three methods: polynomial fitting, linear segments and obstacle avoidance.

Second, parallel robots can be seen as the combination of several serial robots with the same end-effector. So the inverse kinematics of parallel robots can be decomposed into several simple serial roots inverse kinematics. When plotting the workspace of parallel robot, there are also alternative conditions that can determine whether the point is in the workspace. For example, we can set the state that whether the length of the second arm is 130. If it is, then the point is in the workspace, Vice versa.

Finally, we implement an essential principle of robot motion called $TangentBug$ algorithm in a simulation-based environment. The agent, i.e. the mobile robot, uses the range sensor to detect an obstacle within the sensing circumstance and takes actions to avoid the obstacle. The $TangentBug$ principle, compared to basic Bug1 or Bug2 algorithms, tries to minimise the travelling distance and smooth the path to a greater extent.

Bibliography

Abo-Shanab, R., 2014. An efficient method for solving the direct kinematics of parallel manipulators following a trajectory. *Journal of automation and control engineering*, 2 (), pp.228–233. Available from: `https://doi.org/10.12720/joace.2.3.228-233`.

Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., and Thrun, S., 2007. Principles of robot motion: theory, algorithms, and implementation errata!!!! 1, pp.7–19.

Craig, J.J., 1989. *Introduction to robotics: mechanics and control*. 2nd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.