UFMF4X-15-M Robotic Fundamentals
Lecturer: Dr Aghil Jafari

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
Shukai Liu 19044825	✓	✓	✓	✓	✓
Zongyu Wang 19044834	✓	✓	✓	✓	✓

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

Table of Contents

Coursework: Serial and Parallel Robot Kinematics

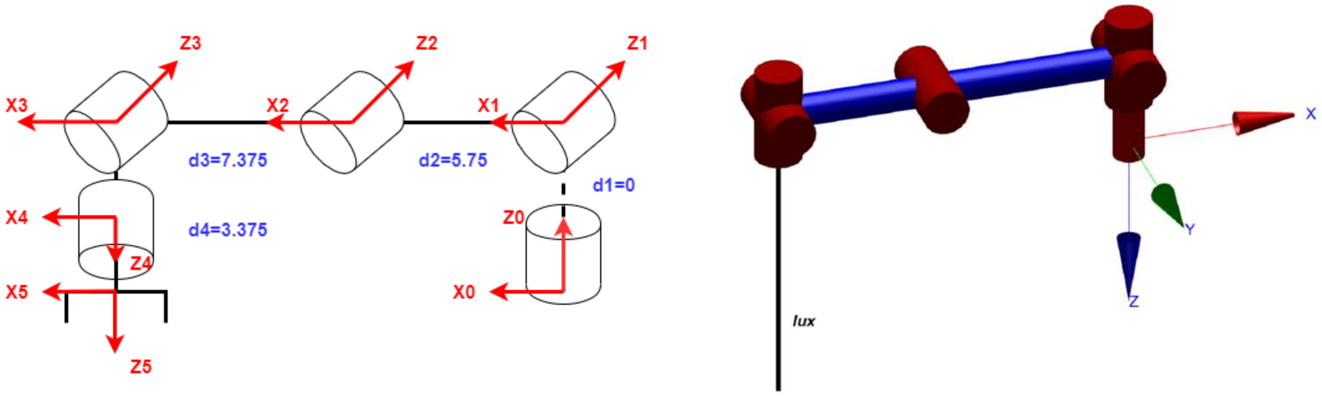
PART I:

A (1)

1.1 FK and IK for the Lynxmotion arm.

1.1.1 Forward Kinematics

From the lecture note and the observation of the Lynxmotion arm. The arm could simplify to the graph as shown below:



Graph 1: structure view of the Lynxmotion arm

The Denavit-Hartenberg parameters for the kinematic scheme presented in Table 1.

N	a_i	α_i	d_n	Θ_i
1	0	90°	d_1	q_1
2	d_2	0	0	q_2
3	d_3	0	0	q_3
4	0	90°	0	q_4
5	0	0	d_4+d_5	q_5

Table 1: Distal Table

1. Link length a_i: offset distance from oi to the intersection of the z_{i-1} and xi axes along x_i
2. Link twist α_i: angle about xi from z_{i-1} axis to the z_i
3. Link offset d_i: distance from $o_{i-1}$to the intersection of z_{i-1} with x_i along z_{i-1}
4. Joint angle θ_i: angle about z_{i-1} from x_{i-1} to x_i

From the data on the website, the detail length of Lynxmotion arm was:

$d1 = 0inch$
$d2 = 5.75\ inch$
$d3 = 7.375\ inch$
$d4 + d5 = 3.375\ inch$

To reduce the complexity of the calculation, we substitute the $d4 + d5$ with $d5$ in the following report.

Besides, there are some ranges of limitation for q.

$q_1: 0°\sim180°$
$q_2: 0°\sim150°$
$q_3: -150°\sim0°$
$q_4: 0°\sim180°$
$q_5: 0°\sim180°$

These parameters could calculate the relative position and orientation of links by using the transition matrixes show below:

$$ {}^{n-1}_{n}T = \begin{bmatrix} c_n & -c_{\alpha n}\cdot s_n & s_{\alpha n}\cdot s_n & a_n\cdot c_n \\ s_n & c_{\alpha n}\cdot c_n & -s_{\alpha n}\cdot c_n & a_n\cdot s_n \\ 0 & s_{an} & c_{\alpha n} & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1) $$

where $c_n = cos\theta i$
$s_n = sin\theta i$
$c\alpha_n = cos\alpha n$
$s\alpha_n = sin\alpha n$

From the data in table 1:

$$ {}^{0}_{1}T = \begin{bmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2) $$

$$ {}^{1}_{2}T = \begin{bmatrix} c_2 & -s_2 & 0 & d_2\times c_2 \\ s_2 & c_2 & 0 & d_2\times s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3) $$

$$ {}^{2}_{3}T = \begin{bmatrix} c_3 & -s_3 & 0 & d_3\times c_3 \\ s_3 & c_3 & 0 & d_3\times s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4) $$

$$ {}^{3}_{4}T = \begin{bmatrix} c_4 & 0 & -s_4 & 0 \\ s_4 & 0 & c_4 & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5) $$

$$ {}^{4}_{5}T = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6} $$

The homogeneous transformation matrix ${}^{0}_{5}T$ can be calculated:

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

$$ {}^{0}_{5}T = \begin{bmatrix} c_1c_{234}c_5 + s_1s_5 & -c_1c_{234}s_5 + s_1c_5 & c_1s_{234} & c_1(d_5s_{234} + d_3c_{23} + d_2c_2) \\ c_1c_{234}c_5 - s_1s_5 & -s_1c_{234}s_5 - c_1c_5 & s_1s_{234} & s_1(d_5s_{234} + d_3c_{23} + d_2c_2) \\ -s_{234}c_5 & -s_{234}s_5 & -c_{234} & d_2s_2 + d_3s_{23} - d_4c_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7} $$

Where $c_{ijk} = \cos(q_i + q_j + q_k)$, and $s_{ijk} = \sin(q_i + q_j + q_k)$.

MATLAB and FK check:
*We wrote our forward kinematic function named FowKi for lynxmotion arm in Matlab, the input is the angle of five joints q_1 , q_2 , q_3 , q_4 and q_5 , and the output is its corresponding transformation matrix T, the form of the equation is T = FowKi(q_1, q_2, q_3, q_4, q_5). For details, please see FowKi.m
Our forward kinematic function FowKi can be checked by comparing with the answer calculated by robotics toolbox, as shown in Table 2. And our forward kinematic function perform well in the following sessions.

```
>> lynx.fkine([3,2,-0.2,1,0])

ans =
    0.9328    0.1411   -0.3316     2.908
   -0.1330    0.9900    0.0473   -0.4146
    0.3350         0    0.9422     15.59
         0         0         0         1
>> FowKi(3,2,-0.2,1,0)

ans =

    0.9328    0.1411   -0.3316    2.9085
   -0.1330    0.9900    0.0473   -0.4146
    0.3350         0    0.9422   15.5906
         0         0         0    1.0000
```

```
>> lynx.fkine([0.5,1.3,-0.2,0.4,1])

ans =
    0.4370    0.2068    0.8754      7.24
   -0.7201   -0.5027    0.4782     3.955
    0.5389   -0.8394   -0.0707     11.87
         0         0         0         1
>> FowKi(0.5,1.3,-0.2,0.4,1)

ans =

    0.4370    0.2068    0.8754    7.2400
   -0.7201   -0.5027    0.4782    3.9552
    0.5389   -0.8394   -0.0707   11.8744
         0         0         0    1.0000
```

Table 2: FK check

A (2)

1.1.2 Workspace of Lynxmotion arm

The method of generating the workspace of lynxmotion arm in Matlab is: Dividing the range of each joint by a specific interval, then go through the range of each joint and generate a series of joint angle sets q= [q1, q2,q3,q4,q5]. Next to calculating forward kinematic of each joint angle set to obtain their transformation matrix. Plotting the last column of the transformation matrix, which is the x, y, z position. For the best display effect, we divide the range of q1 by $\pi/60$, and q2, q3, q4 by $\pi/15$. We set q5=0 because it doesn't affect the end-effector position. For details see workspace.m

2D view of workspace:

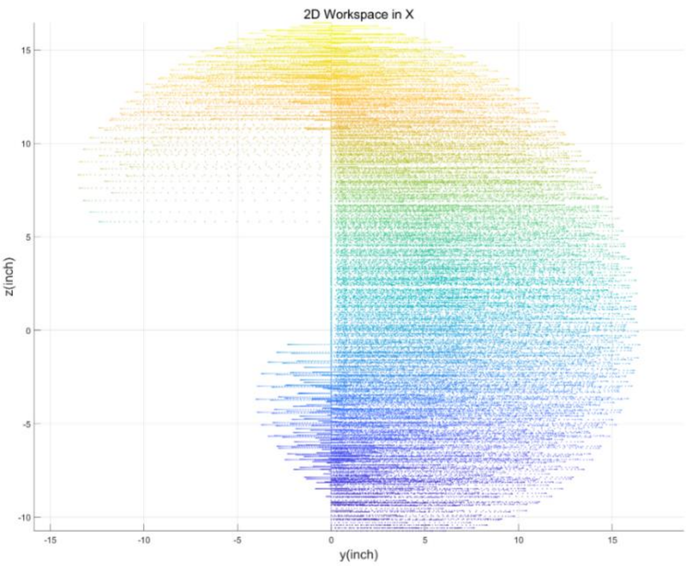


Figure 2: 2D workspace in X.

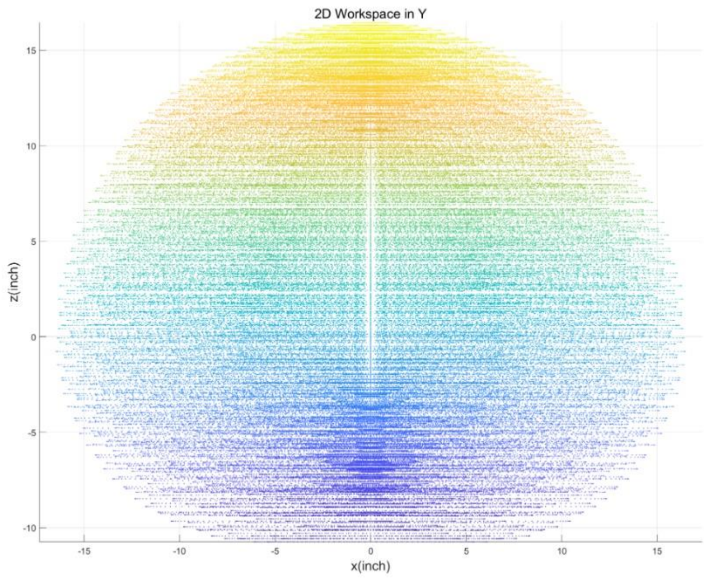


Figure 3: 2D workspace in Y

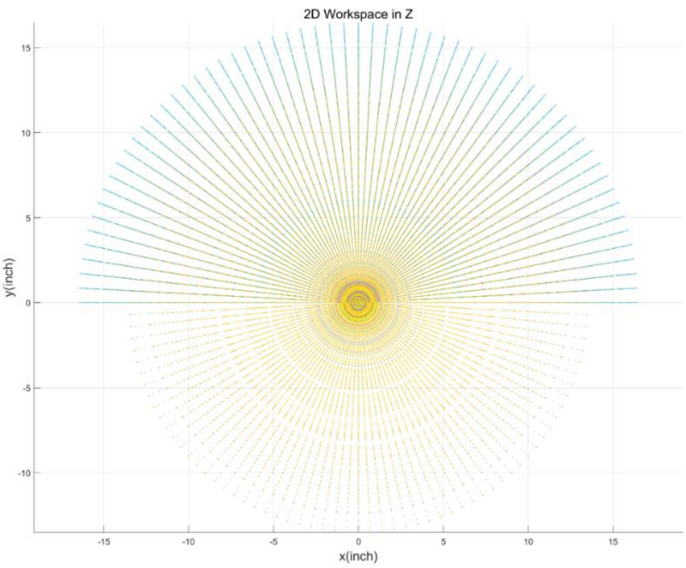


Figure 4: 2D workspace in Z

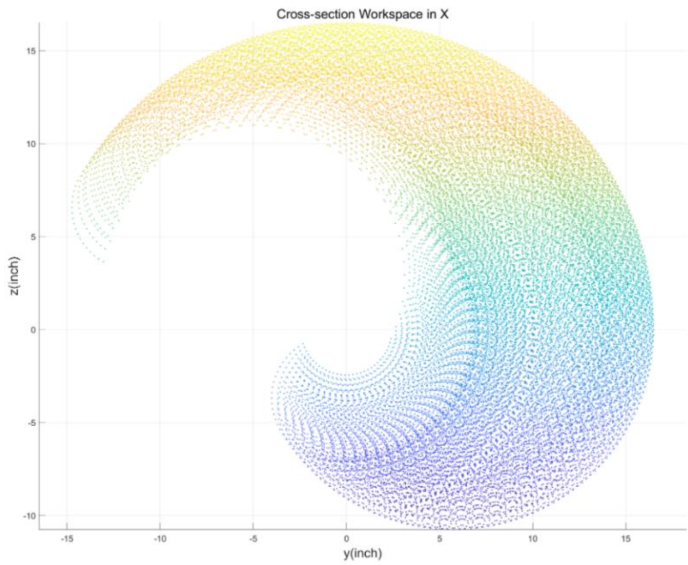


Figure 5: cross-section in X direction

3D view of workspace:

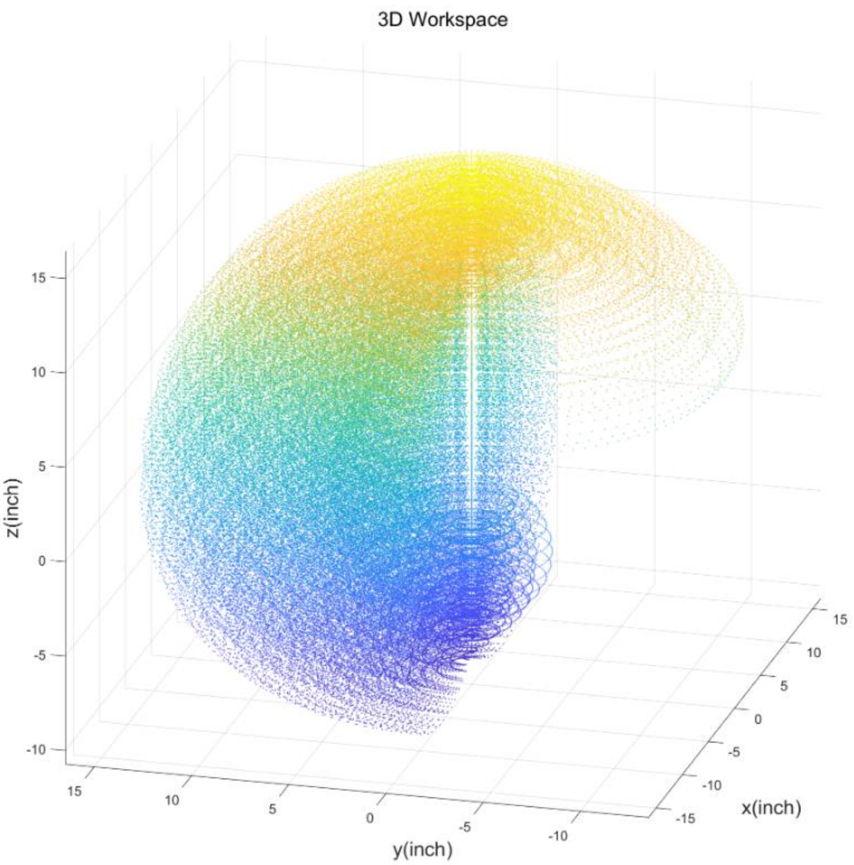


Figure 6: side view of 3D workspace

A (3)

1.1.3 Derive the inverse kinematics model for the manipulator (analytical solution).

To calculated the $q_1, q_2, q_3, q_4, and\ q_5$, set a transition matrix shown blew:

$$ {}^5_0T = \begin{bmatrix} n_x & o_x & a_x & x_e \\ n_y & o_y & a_y & y_e \\ n_z & o_z & a_z & z_e \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The q_1, q_5 can be calculated according to geometric relationship:

$$ q_1 = atan2(y_e, x_e) $$

Where (y_e, x_e) is the endpoint coordinated of the arm.
But there is a special case that, when the manipulator has a very large elevation angle, it will make the y_e<0, which make the calculation result of q_1 and a π less than it is, so q_1 should be:

$$ q1 = \begin{cases} atan2(ye, xe), & ye \geq 0 \\ atan2(ye, xe) + \pi, & ye < 0 \end{cases} $$

q_5 can be calculated by:

$$ q_5 = atan2\left(o_x p_y - n_y p_x, o_x p_y - o_y p_x\right) $$

To solve q_2 and q_3, ${}^4_0T$ equal to:

$$ {}^4_0T = {}^5_0T \times {}^5_4T^{-1} = \begin{bmatrix} n_x c_5 - o_x s_5 & o_x c_5 + n_x s_5 & \boxed{a_x} & \boxed{x_e - a_x d_4} \\ n_y c_5 - o_y s_5 & o_y c_5 + n_y s_5 & a_y & y_e - a_y d_4 \\ n_z c_5 - o_z s_5 & o_z c_5 + n_z s_5 & a_z & z_e - a_z d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8) $$

And ${}^4_0T$ also can be expressed as:

$$ {}^4_0T = {}^0_1T \times {}^1_2T \times {}^2_3T \times {}^3_4T = \begin{bmatrix} c_{234} c_1 & s_1 & s_{234} c_1 & c_1(d_3 c_{23} + d_2 c_2) \\ c_{234} s_1 & -c_1 & s_{234} s_1 & s_1(d_3 c_{23} + d_2 c_2) \\ s_{234} & 0 & -c_{234} & d_3 s_{23} + d_2 c_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (9) $$

The equation in blue circles should be the same, and we can get:

$$ x_e - a_x d_4 = c_1(d_3 c_{23} + d_2 c_2) $$
$$ y_e - a_y d_4 = s_1(d_3 c_{23} + d_2 c_2) $$
$$ z_e - a_z d_4 = d_3 s_{23} + d_2 c_2 $$

So

$$ d_3 \cos(q_2 + q_3) = \frac{x_e - a_x d_4}{cosq_1} - d_2 cosq_2 $$
$$ d_3 \sin(q_2 + q_3) = z_e - a_z d_4 - d_2 sinq_2 $$

To reduce the complexity of the equation, we define two intermediate parameters:

$$Z_4 = z_e - a_z d_4$$

$$D_4 = \frac{x_e - a_x d_4}{cosq_1}$$

So the equation becomes:

$$d_3 \cos(q_2 + q_3) = D_4 - d_2 cosq_2 \quad (10)$$

$$d_3 \sin(q_2 + q_3) = Z_4 - d_2 sinq_2 \quad (11)$$

Firstly for q_2, use the above equation, $(10)^2 + (11)^2$, we can obtain:

$$d_3^2 = (D_4 - d_2 cosq_2)^2 + (Z_4 - d_2 sinq_2)^2$$

Then applying the auxiliary angle method:

$$D_4 \times cosq_2 + Z_4 \times sinq_2 = \sqrt{Z_4^2 + D_4^2} \times \sin\left(q_2 + atan2(D_4, Z_4)\right)$$

There would be two solutions for q_2:

$$\boldsymbol{q_{2(1)}} = \arcsin\left(\frac{Z_4^2 + D_4^2 + d_2^2 + d_3^2}{2d_2\sqrt{Z_4^2 + D_4^2}}\right) - atan2(D_4, Z_4)$$

$$\boldsymbol{q_{2(2)}} = \pi - q_{2(1)}$$

Secondly, to solve q_3, divide Eqn(11) by Eqn(10):

$$\tan(q_2 + q_3) = \frac{Z_4 - d_2 sinq_2}{D_4 - d_2 cosq_2}$$

And q_3 could present as:

$$\boldsymbol{q_3} = atan2(Z_4 - d_2 sinq_2, D_4 - d_2 cosq_2) - q_2$$

Finally to solve q_4, ${}_1^5T$ equal to:

$${}_1^5T = {}_0^1T^{-1} \times {}_0^5T = \begin{bmatrix} n_x c_1 + n_y s_1 & o_x c_1 + o_y s_1 & a_x c_1 + a_y s_1 & x_e c_1 + y_e s_1 \\ n_z & o_z & a_z & z_z \\ n_x s_1 - n_y c_1 & o_x s_1 - o_y c_1 & a_x s_1 - a_y c_1 & x_e s_1 - y_e c_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (12)$$

And ${}_1^5T$ also can be expressed as:

$${}_1^5T = {}_2^1T \times {}_3^2T \times {}_4^3T \times {}_4^5T = \begin{bmatrix} c_{234}c_5 & -c_{234}s_5 & s_{234} & d_3c_{23} + d_2c_2 + d_4s_{234} \\ s_{234}c_5 & -s_{234}s_5 & -c_{234} & d_3c_{23} + d_2c_2 - d_4s_{234} \\ s_5 & c5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (13)$$

The equation in blue circles should be the same, and we can get:

$$c_{234}c_5 = n_x c_1 + n_y s_1 \tag{14}$$

$$s_{234}c_5 = n_z \tag{15}$$

Divide eqn13 by eqn12:

$$tan_{234} = \frac{n_z}{n_x c_1 + n_y s_1}$$

So q4 can be obtained as:

$$\boldsymbol{q_4} = atan2(n_z, n_x c_1 + n_y s_1) - q_2 - q_3 \tag{16}$$

because there are **two solutions** for q_2, so there are also two corresponding solutions for q_3, q_4 and q_5. But because of the limit range of each joint angle, every position and orientation of lynxmotion arm only have one set of corresponding joints angle, so we need to screen out the correct solution from two sets of joint angle solution, and according to the range of joint angle, the screening condition is: "in a set of solution, if q_2<0 or q_3>0, this solution is incorrect, and the other solution is correct. "

MATLAB and FK check:
We wrote our inverse kinematic function named InvKi for lynxmotion arm in Matlab, the input is the transformation matrix and the output is the joints angle q_1, q_2, q_3, q_4 and q_5. Because q_5 doesn't affect the position of end effector, we let $q_5 = 0$ in out function. For details, please see InvKi.m. and we also write a checking script for our inverse kinematic function InvKi, it will randomly generate a set of q_1-q_5, and after the process of forward kinematic and inverse kinematic, check whether the answer is equal to the input q_1-q_5. Our function has gone through more than 100000 times of checking and perform well, for details, please see IKRandomCheck.m.

```
>> InvKi(FowKi(3,2,-0.2,1,0))

ans =

    3.0000    2.0000   -0.2000    1.0000         0
```

```
>> InvKi(FowKi(1,2,-2,0,0))

ans =

     1     2    -2     0     0
```

```
>> InvKi(FowKi(0.2,2.5,-2,1,0))

ans =

    0.2000    2.5000   -2.0000    1.0000         0
```

```
>> InvKi(FowKi(0.2,0.3,-0.8,0.23,0))

ans =

    0.2000    0.3000   -0.8000    0.2300         0
```

Table 5: IK check

B. Complete the following:

1.2 Trajectory planning of Lynxmotion arm.

In this part, we will infer the function of free motion and the function of straight-line motion. And every motion between 2 points (both free motion and straight-line motion) will consist of three parts: acceleration part, constant velocity part and deceleration part. After obtaining the function of free motion and straight-line motion, we will operate the manipulator to do a specific task in Matlab.

1.2.1 Linear function with parabolic blends

For the motion with constant velocity (the velocity could be the joint angle velocity, spatial position velocity, etc.), the corresponding path is linear. However, if the motion has more than one segment, at each endpoint, there will be a discontinuity of velocities. To avoid this discontinuity, we must accelerate and decelerate the motion. And we use the parabolic path at the transition parts as figure 7 shown.

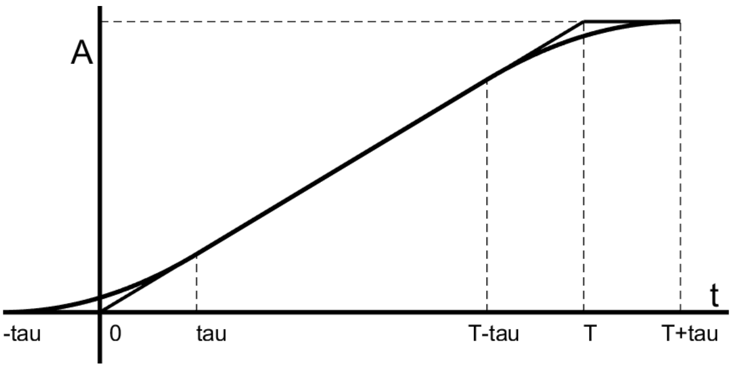


Figure 7: parabolic path at the transition parts

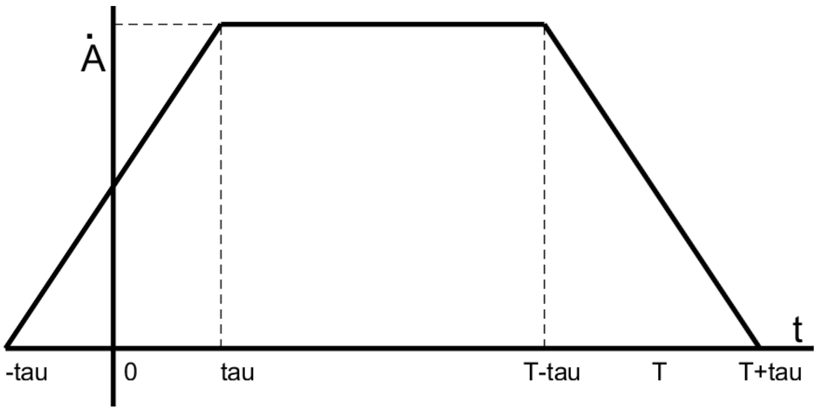


Figure 8: parabolic path velocities at the transition parts

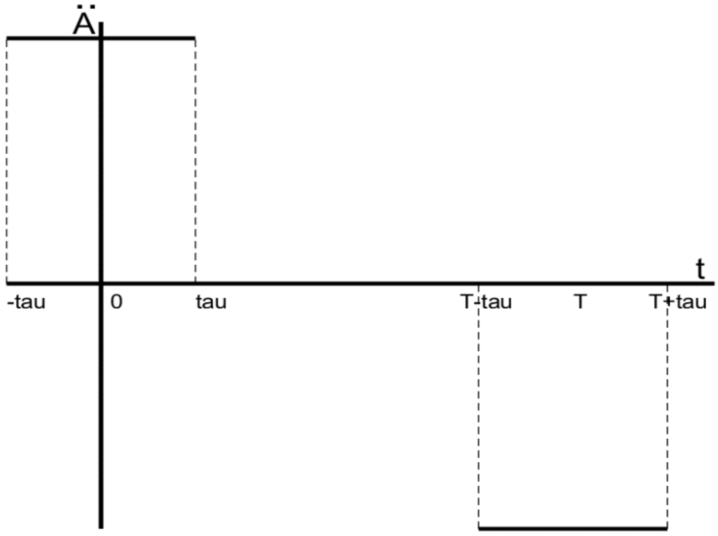


Figure 9: parabolic path acceleration at the transition parts

In a single motion segment between two points, we assume it takes time 2τ to accelerate from zero velocity to the constant velocity and takes 2τ as well to decelerate from constant velocity to zero.
The total time of the motion we assume is $T + 2\tau$.
The path, velocity and acceleration for $-\tau < t < \tau$:

$$\begin{gathered} A = \Delta A \frac{(t+\tau)^2}{4T\tau} \\ \dot{A} = \Delta A \frac{t+\tau}{2T\tau} \\ \ddot{A} = \Delta A \frac{1}{2T\tau} \end{gathered} \qquad (17)$$

The path, velocity and acceleration for $\tau < t < T - \tau$:

$$\begin{gathered} A = \Delta A \frac{t}{T} \\ \dot{A} = \Delta A \frac{1}{T} \\ \ddot{A} = 0 \end{gathered} \qquad (18)$$

The path, velocity and acceleration for $T - \tau < t < T + \tau$:

$$\begin{gathered} A = \Delta A \frac{t}{T} - \Delta A \frac{(t-T+\tau)^2}{4T\tau} \\ \dot{A} = \Delta A \frac{1}{T} - \Delta A \frac{t-T+\tau}{2T\tau} \\ \ddot{A} = -\Delta A \frac{1}{2T\tau} \end{gathered} \qquad (19)$$

In which, A could be the position, joint angle and orientation:

$$A = x, y, z, q1, q2, q3, q4, q5, \theta$$

1.2.2 Free Motion (FMTraj.m)

In free motion, there is no a specific requirement for the shape and path of trajectory, so we choose the easiest way for motors to make the free motion: Make all the joint angles move from the current angles to the target angles with a constant velocity, but at the beginning and the end of the motion we need to accelerate and decelerate respectively, and the acceleration is a constant value.

Applying the linear function with parabolic blends we mentioned above, we can obtain the function of joint angles q with time:

$$q(t) = \begin{cases} qs + (qe - qs)\dfrac{(t+\tau)^2}{4T\tau} & , -\tau < t < \tau \\ qs + (qe - qs)\dfrac{t}{T} & , \tau < t < T - \tau \\ qs + (qe - qs)\left(\dfrac{t}{T} - \dfrac{(t - T + \tau)^2}{4T\tau}\right) & , T - \tau < t < T + \tau \end{cases} \quad (20)$$

$$q = [q1, q2, q3, q4, q5]$$

In which, the $q(t)$ is the five joint angles vector of the lynxmotion arm at the time t, qs and qe are the vectors of joint angles at the initial point and endpoint of the motion respectively. The acceleration and deceleration time are 2τ, and the total time of motion is T+2τ.

We wrote our free motion function named FMTraj in Matlab. The form is: [] = FMTraj(qs,qe,tt,ta) , the input qs is the vector of initial joint angles, qe is the vector of final joint angles, tt is the total motion time and ta is the acceleration as well as deceleration time. The function will output the animation plot of free motion. Here is an example:

Entering FMTraj ([0,0,0,0,0], [1.2,0.9,-0.68,1,0],3,1) and we can get:

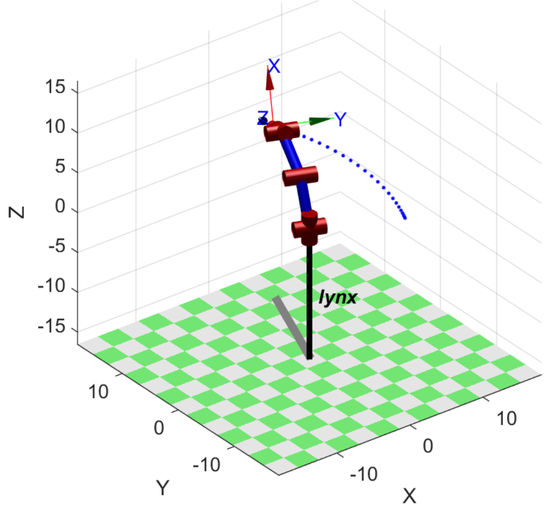


Figure 10: Free motion

And the relationship between joint angle and time (take q_2 as an example):

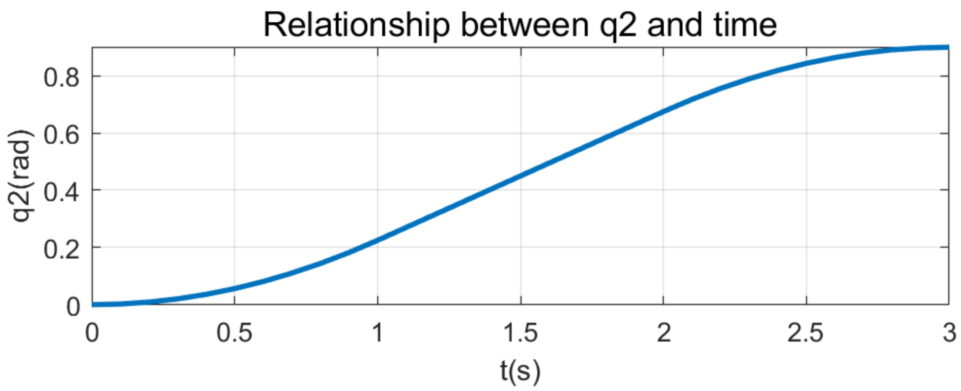


Figure 11: the relationship between q_2 and time

1.2.3 Straight-line Motion (SLTraj.m)

The straight-line motion can generate the shortest path between two points. Unlike the free motion, it is hard to generate the straight-line trajectory by manipulating in joint space directly. So we divided the process of straight-line motion into two parts: position and orientation.

Firstly we inferring the relationship between position and time, the position information is in the first three rows of the last column in the transformation matrix. In this part, the end effector will move from one point to another point with constant spatial velocity (x,y,z velocity) and accelerate at the beginning and decelerate at the end of the motion. Applying the linear function with parabolic blends, we can obtain the function of end effector position (x,y,z) with time as follow:

$$p(t)=\begin{cases} ps+(pe-ps)\dfrac{(t+\tau)^2}{4T\tau} & ,-\tau<t<\tau \\ ps+(pe-ps)\dfrac{t}{T} & ,\tau<t<T-\tau \\ ps+(pe-ps)\left(\dfrac{t}{T}-\dfrac{(t-T+\tau)^2}{4T\tau}\right) & ,T-\tau<t<T+\tau \end{cases} \quad (21)$$

$$p=[x,y,z]^T$$

In which, the p(t) is the position vector of the end effector at time t, ps and pe are the position vectors at the initial point and endpoint of the motion, respectively. The acceleration and deceleration time are 2τ, and the total time of motion is T+2τ.

Secondly, we were inferring the relationship between orientation and time. The orientation information is in the first three rows and the first three columns of the transformation matrix, which named a rotation matrix. In the motion process, the orientation of the end effector also changes, but we cannot directly interpolate the rotation matrix to control the orientation. But if we transfer the rotation matrix into Euler angles and then transfer into quaternions, which means we use four parameters in quaternions to represent the 3×3 rotation matrix, according to the properties of quaternions, we can interpolate the quaternions to achieve manipulating the orientation. The form of a quaternion is:

$$Q=a+bi+cj+dk$$

We interpolate quaternions also by applying the linear function with parabolic blends, the function of orientation represented by quaternions with time is:

$$Q(t)=\begin{cases} Qs+(Qe-Qs)\dfrac{(t+\tau)^2}{4T\tau} & ,-\tau<t<\tau \\ Qs+(Qe-Qs)\dfrac{t}{T} & ,\tau<t<T-\tau \\ Qs+(Qe-Qs)\left(\dfrac{t}{T}-\dfrac{(t-T+\tau)^2}{4T\tau}\right) & ,T-\tau<t<T+\tau \end{cases} \quad (22)$$

In which, $Q(t)$ is the orientation of end effector represented by quaternion at time t, Qs and Qe are the quaternion-represented orientation of end effector at the initial point and endpoint of the motion respectively.

So the whole process of generating the straight-line trajectory can be shown as:

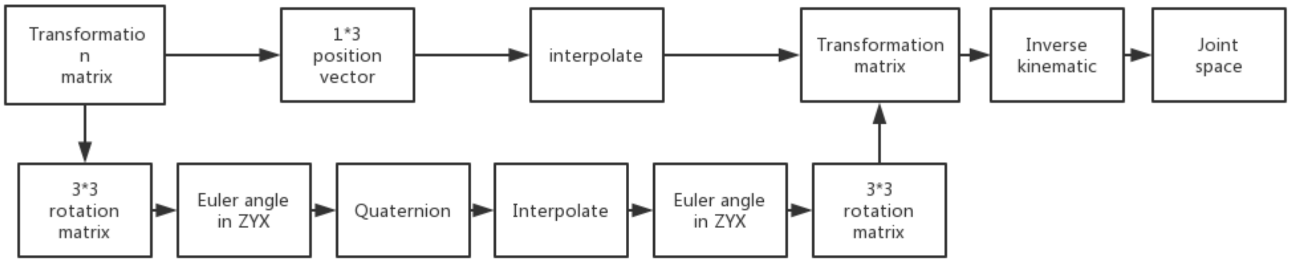


Figure 12: Flow chart of the straight-line motion

We wrote our straight-line motion function named SLTraj in Matlab, the form is: [] = SLTraj(Ts,Te,tt,ta) , the input Ts is the transformation matrix of initial posture, Te is the transformation matrix of final posture, tt is the total motion time and ta is the acceleration as well as deceleration time. The function will output the animation plot of straight-line motion. Here is an example: Letting Ts=FowKi(2*pi/3,0,0,0,0), Te=FowKi(75*pi/180,pi/6,-pi/6,0,0), and then enter SLtraj(Ts,Te,5,2), we can get:

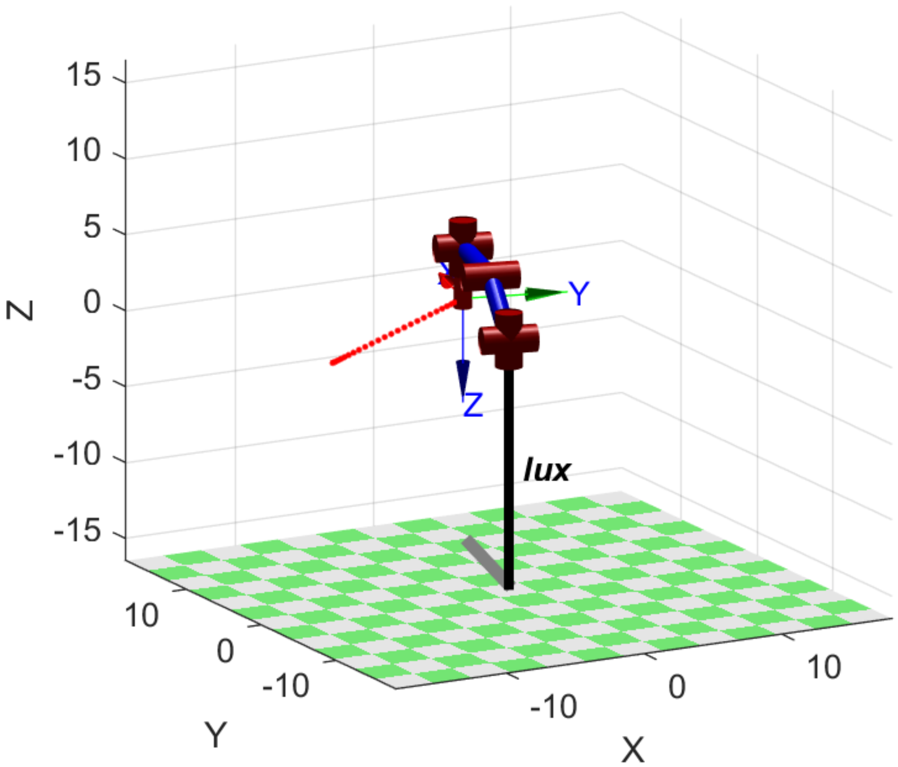


Figure 13: Straight-line Motion

And the relationship between position and time (take x as an example):

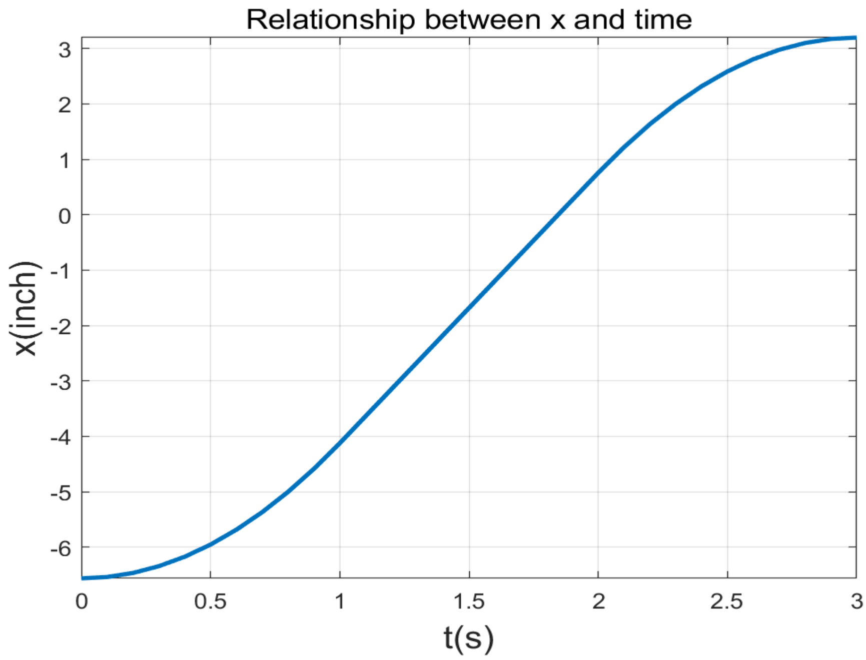


Figure 14: the relationship between x and time

We also infer another different way to generate a straight-line trajectory, but using quaternion is clearer and quicker in Matlab, we will introduce the other method in the appendix.

1.3: Trajectory test: writing 'OK' and avoid 2 obstacles.

1.3.1: 'OK' task planning.

The task we plan is: Using the lynxmotion arm to write a letter "OK" in a slant plane, and after finishing the writing, the manipulator will move back to the initial position and orientation ($[q_1, q_2, q_3, q_4, q_5]$=0). The plane is in the y-direction, and the angle between the plane and the XOY plane is 20 degree to the base coordinate. To write an "OK", the manipulator needs to move through 9 specific points. And in the task the manipulator needs to avoid two obstacles, one is between the word 'O' and 'K', and the other is in the path when the manipulator moves back to the initial position and orientation. We made a model in Solidworks software to make a clearer display, as shown in figure 14.

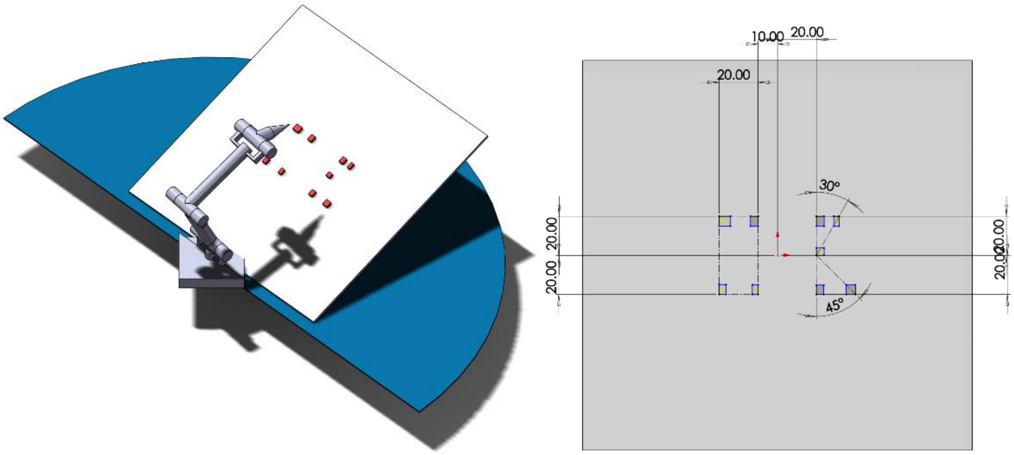


Figure 15: Solidworks modal for the Lynxmotion arm

In figure 14, the white plane is the writing plane, and the red points represent the word "O" and "K", the blue plane represents the range of q1. The geometric relationship of the 9 points, as shown in figure x+1. In the task, the writing part will be consisting of straight line trajectories, and the obstacle avoidance part will be consists of free motion trajectories, the details will discussed in the last section of Part 1 after deriving the free motion function and straight-line function.

1.3.2: Trajectory generation

To write the word "O", the end effector needs to go through four-points: O_1, O_2, O_3 and O_4. To write the work "K", the end effector needs to go through 5 points: K_1, K_2, K_3, K_4 and K_5. After writing "OK", the end effector will back to the initial position and orientation I. To avoid the two obstacles, the end effector needs to go through two intermediate points: I_1 and I_3, and to write the word "K", we also need an intermediate point I_2. According to the measurement in Solidworks, the transformation matrixes of these points are:

$$T_{O1}=\begin{bmatrix}-0.1906 & 0.9703 & -0.1489 & -3.0434\\ 0.7646 & 0.2419 & 0.5974 & 12.2604\\ 0.6157 & 0 & -0.7880 & 1.9054\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{O2}=\begin{bmatrix}-0.3196 & 0.9455 & -0.0621 & -2.8966\\ 0.9281 & 0.3256 & 0.1804 & 8.4123\\ 0.1908 & 0 & -0.9816 & 0.5472\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{O3}=\begin{bmatrix}-0.1192 & 0.9925 & -0.0253 & -1.0430\\ 0.9709 & 0.1219 & 0.2064 & 8.4946\\ 0.2079 & 0 & -0.9781 & 0.6217\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{O4}=\begin{bmatrix}-0.0696 & 0.9962 & -0.0525 & -1.0695\\ 0.7956 & 0.0872 & 0.5995 & 12.2243\\ 0.6018 & 0 & -0.7986 & 1.8983\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{K1}=\begin{bmatrix}0.1249 & 0.9877 & 0.0941 & 1.9196\\ 0.7888 & -0.1564 & 0.5944 & 12.1199\\ 0.6018 & 0 & -0.7986 & 1.8983\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{K2}=\begin{bmatrix}0.2088 & 0.9770 & 0.0425 & 1.8137\\ 0.9574 & -0.2140 & 0.1948 & 8.3183\\ 0.1994 & 0 & -0.9799 & 0.5533\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{K3}=\begin{bmatrix}0.1946 & 0.9681 & 0.1576 & 3.1613\\ 0.7524 & -0.2504 & 0.6093 & 12.2240\\ 0.6293 & 0 & -0.7771 & 1.9421\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{K4}=\begin{bmatrix}0.1769 & 0.9816 & 0.0715 & 2.0132\\ 0.9101 & -0.1908 & 0.3677 & 10.3571\\ 0.3746 & 0 & -0.9272 & 1.3752\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{K5}=\begin{bmatrix}0.4238 & 0.9041 & 0.0550 & 3.9994\\ 0.8966 & -0.4274 & 0.1164 & 8.4608\\ 0.1288 & 0 & -0.9917 & 0.6096\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{I1}=\begin{bmatrix}0 & 1 & 0 & 0\\ 0.5 & 0 & 0.8660 & 6.6103\\ 0.8660 & 0 & -0.5 & 10.4494\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$T_{I2} = \begin{bmatrix} 0.0866 & 0.9770 & 0.1946 & 1.6476 \\ 0.3974 & -0.2140 & 0.8926 & 7.5568 \\ 0.9135 & 0 & -0.4067 & 9.0305 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{I3} = \begin{bmatrix} 0 & 0.5 & 0.8660 & 6.1163 \\ 0 & -0.8660 & 0.5 & 3.5312 \\ 1 & 0 & 0 & 12.1369 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using our inverse kinematic function InvKi, we can obtain the joint angles of these points, the motion sequence, motion type and joint angles are shown in the table:

No.	Point	q_1 (degree)	q_2 (degree)	q_3 (degree)	q_4 (degree)	q_5 (degree)	Motion type (i to i+1)
1	O_1	104	57	-59	40	0	Straight line
2	O_2	109	79	-93	25	0	Straight line
3	O_3	97	83	-97	26	0	Straight line
4	O_4	95	60	-63	40	0	Straight line
5	O_1	104	57	-59	40	0	Free motion
6	I_1	90	90	-30	0	0	Free motion
7	K_1	81	60	-63	40	0	Straight line
8	K_2	77.7	83	-97.5	26	0	Free motion
9	I_2	77.7	100	-60	26	0	Free motion
10	K_3	75.5	57	-59	41	0	Straight line
11	K_4	79	70	-77	29	0	Straight line
12	K_5	64.7	72.7	-84.7	19.4	0	Free motion
13	I_3	30	90	-30	30	0	Free motion
14	I	0	0	0	0	0	

The trajectory of the whole task is shown as follow, the red trajectories represent the straight-line motion, and the blue trajectories represent free motion:

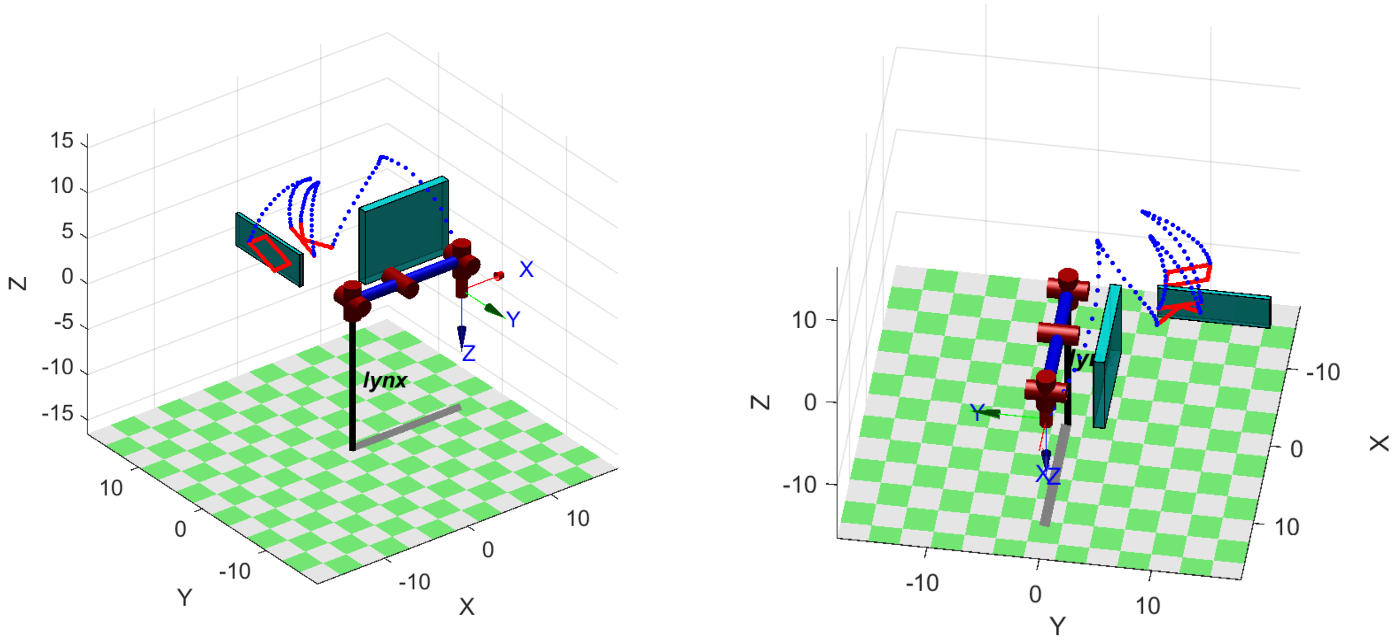


Figure 16: side view of "OK" motion task

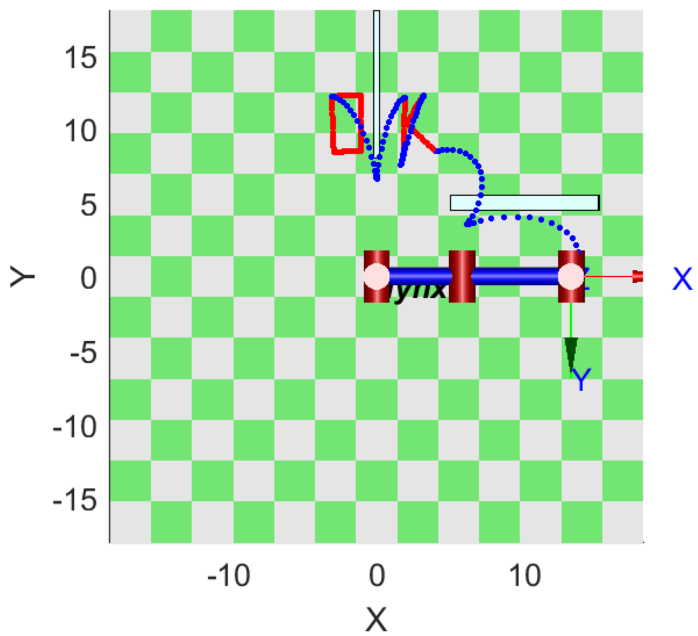


Figure 17: top view of "OK" motion task

To operate the robot arm safe, we should set appropriate velocity and acceleration. In our task, we should set appropriate motion total time tt and acceleration time ta when we input the function FMTraj and SLTraj.

PART II:
(1)

2.1: Inverse Kinematics for parallel robot.

2.1.1: Inverse Kinematics

The base has 3 points: PA, PB, PC.
the platform has 3 points: A, B, C.

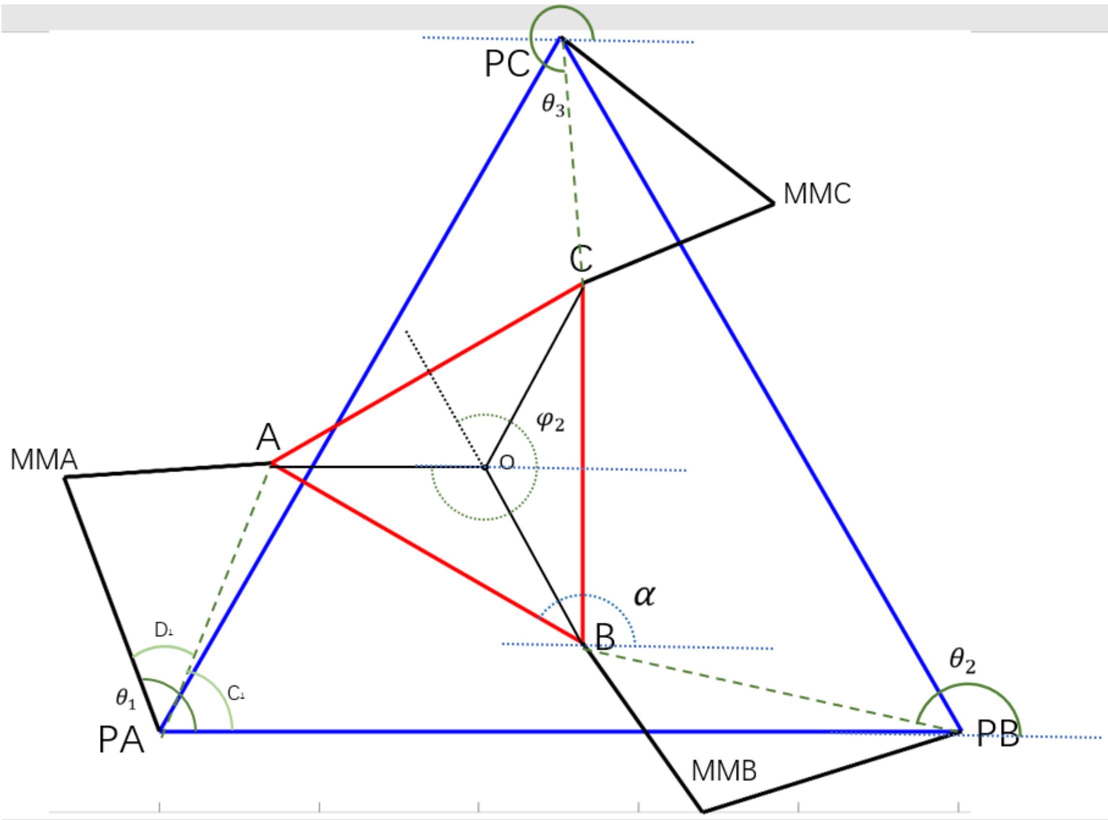


Figure 18: Planar Parallel robot kinematic model

As shown in the figure, $\boldsymbol{\theta_i}$ is the angle between the positive direction of the horizontal line and the first rod (SA) at three corners the triangle.
$\boldsymbol{\alpha}$ is the angle between the positive direction of the horizontal line and the bottom side of the plate triangle.
$\boldsymbol{\varphi_i}$ is the angle between the horizontal line and the extension line for the line between the centre and the corner of the plate triangle.

$$\varphi_1 = \alpha + \frac{\pi}{6}$$

$$\varphi_2 = \alpha + \frac{5\pi}{6} \tag{23}$$

$$\varphi_3 = \alpha + \frac{3\pi}{2}$$

Solve and implement parallel robot Inverse Kinematics

Solution 1

From figure 18,

Connect the line between PA and A. We get the triangle (PA, A, MMA)

<u>For point A</u>, θ_1 can be present:

$$\theta_1 = c_1 \pm d_1$$

$$\varphi_1 = \alpha + \frac{\pi}{6}$$

$$c_1 = tan^{-1}\left(\frac{y_c - r_p \times sin\varphi_1}{x_c - r_p \times cos\varphi_1}\right) \tag{24}$$

Use the law of cosines for the triangle (PA, A, MMA)

$$\theta_c = cos^{-1}(\frac{a^2 + b^2 - c^2}{2ab})$$

$$d_1 = cos^{-1}\left(\frac{\boxed{S_A^2} + \boxed{\left(x_c - r_p cos\varphi_1\right)^2 + \left(y_c - r_p sin\varphi_1\right)^2} - \boxed{L^2}}{2S_A\sqrt{\left(x_c - r_p cos\varphi_1\right)^2 + \left(y_c - r_p sin\varphi_1\right)^2}}\right) \tag{25}$$

<u>For point B</u>:

Connect the line between PA and A. We get the triangle (PB, B, MMB)

$$\theta_2 = c_2 \pm d_2$$

$$\varphi_2 = \alpha + \frac{5\pi}{6}$$

The x-coordinate for PB is $\sqrt{3} * 290$, besides, θ_2 is an obtuse angle, so the c_2 can be express as:

$$c_2 = tan^{-1}\left(\frac{y_c - r_p \times \sin(\pi - \varphi_2)}{x_c + r_p \times \cos(\pi - \varphi_2) - \sqrt{3}r_b}\right) \tag{26}$$

Use the law of cosines for the triangle (PB, B, MMB)

$$d_2 = cos^{-1}\left(\frac{S_A^2 + \boxed{\left(x_c + r_p \times \cos(\pi - \varphi_2) - \sqrt{3}r_b\right)^2 + \left(y_c - r_p \times \sin(\pi - \varphi_2)\right)}^2 - L^2}{2S_A\sqrt{\left(x_c + r_p \times \cos(\pi - \varphi_2) - \sqrt{3}r_b\right)^2 + \left(y_c - r_p \times \sin(\pi - \varphi_2)\right)^2}}\right) \tag{27}$$

For point C:
Connect the line between PA and A. We get the triangle (PC, C, MMC)

$$\theta_3 = c_3 \pm d_3$$
$$\varphi_3 = \alpha + \frac{3\pi}{2}$$

The x-coordinate for Pc is $(\frac{\sqrt{3}}{2} * 290)$, and y-coordinate is $\left(\frac{3}{2} * 290\right)$ besides, θ_2 is the angle in the third and fourth quadrant, so the c_3 can be express as:

$$c_3 = tan^{-1}\left(\frac{y_c + r_p \times \sin(2\pi - \varphi_3) - \frac{3}{2} r_b}{x_c - r_p \times \cos(2\pi - \varphi_3) - \frac{\sqrt{3}}{2} r_b}\right) \tag{28}$$

Use the law of cosines for the triangle (PC, C, MMC)

$$d_3 = cos^{-1}\left(\frac{S_A^2 + \left(x_c - r_p \times \cos(2\pi - \varphi_2) - \frac{\sqrt{3}}{2} r_b\right)^2 + \left(y_c + r_p \times \sin(2\pi - \varphi_3) - \frac{3}{2} r_b\right)^2 - L^2}{2S_A\sqrt{\left(x_c - r_p \times \cos(2\pi - \varphi_2) - \frac{\sqrt{3}}{2} r_b\right)^2 + \left(y_c + r_p \times \sin(2\pi - \varphi_3) - \frac{3}{2} r_b\right)^2}}\right. \tag{29}$$

We write our own parallel inverse kinematic function named ParallelIKPlot. The form of this function is theta = ParallelIKPlot(xc, yc, alpha). The input xc and yc are the coordinate of the needle, and alpha is the angle between the platform and horizontal direction. And the function will output a 8×3 matrix, its each line is a set of solution of three active angles. The function will also output the plot of eight postures. The example will be shown in next section, and for details please see ParallelIKPlot.m.

*There is another way, which use the algebraic method to figure out the θ_i, the process of the solution was in the appendix.
Also the FK for parallel robot was in the appendix.

2.1.2: Kinematic model in two different positions.

From the solution of the inverse kinematics for parallel robot mentioned before. Every θ would have two solutions. So one set of the parameters (x, y, α) would have 8 $(2 \times 2 \times 2)$ different postures.
Applying our Parallel inverse kinematic function ParallelIKPlot:
Position 1 :

$x = 260mm \quad y = 180mm \quad \alpha = 30°$

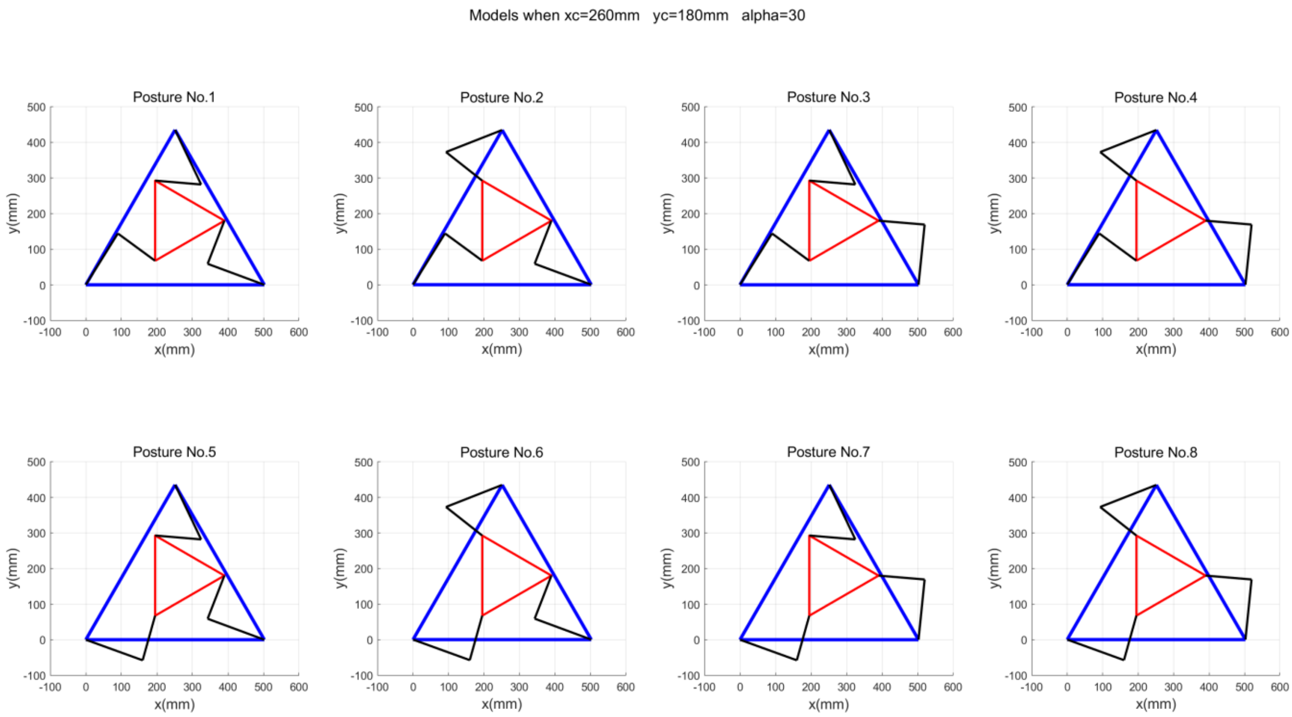


Figure 19: 8 different postures

```
>> ParallelIKPlot(260, 180, 30)
```

ans :

θ_1	θ_2	θ_3
58.0030	159.7404	-64.4231
58.0030	159.7404	-158.6103
58.0030	84.1763	-64.4231
58.0030	84.1763	-158.6103
-19.8598	159.7404	-64.4231
-19.8598	159.7404	-158.6103
-19.8598	84.1763	-64.4231
-19.8598	84.1763	-158.6103

Table 4: Different θ for 8 postures

Position 2:

$x = 213mm \quad y = 185mm \quad \alpha = -10°$

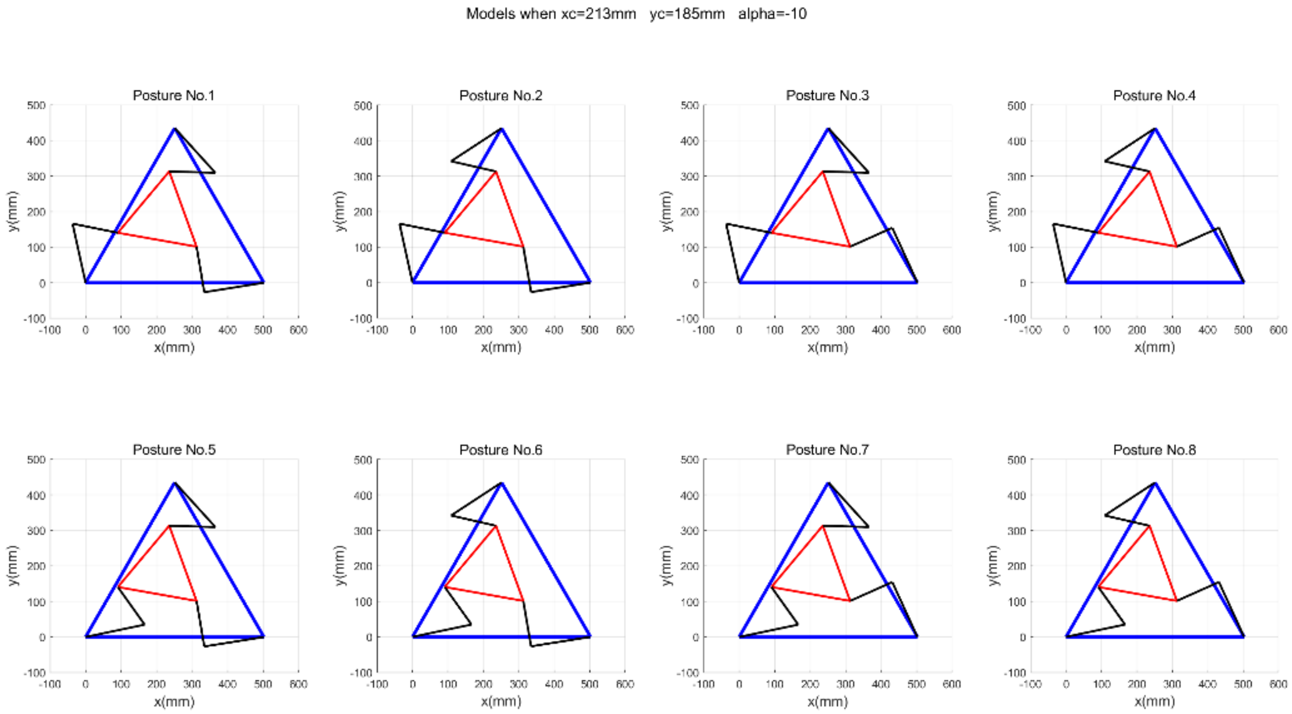


Figure 20: 8 different postures

```
>> ParallelIKPlot(213,185,-10)

ans =
```

θ_1	θ_2	θ_3
102.4470	189.0423	-47.7186
102.4470	189.0423	-146.8331
102.4470	114.6908	-47.7186
102.4470	114.6908	-146.8331
11.7977	189.0423	-47.7186
11.7977	189.0423	-146.8331
11.7977	114.6908	-47.7186
11.7977	114.6908	-146.8331

Table 4: Different θ for 8 postures

2.2 parallel robots' workspace for a given orientation.

In this section, we detect the workspace of parallel robot at specific angle α by using our inverse kinematic function ParallelIK (ParallelIK and ParallelIKPlot are basically the same, the only difference is the function ParallelIK doesn't output the plot). The basic idea is, First determining a big range of x and y which include all the needle possible position. Then, given alpha, applying the function ParallelIK to every point in the range, and when the function can find correct inverse kinematic solutions, record the needle position x and y. And after go through the whole range, the recorded set of needle positions is the workspace for a specific α.
We wrote our own workspace function named ParallelWorkspace for parallel robot, the form of function is: [] = ParallelWorkspace(alpha). The input is the angle of platform α, and the function will output the plot of workspace.
Here are two examples of parallel robot workspace for specific α.

$\alpha = 20°$

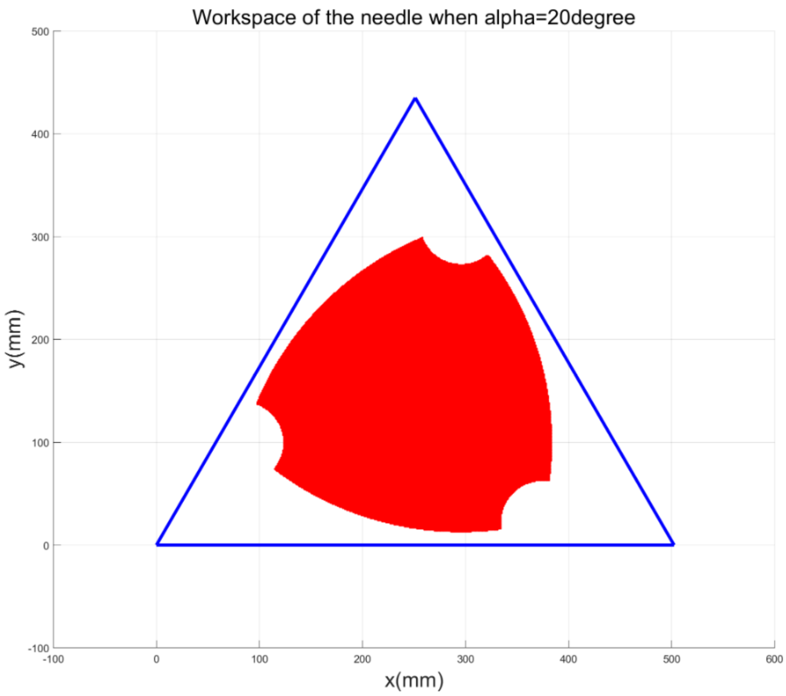


Figure 21: workspace when $\alpha = 20°$

°

$\alpha = -10^{\circ}$

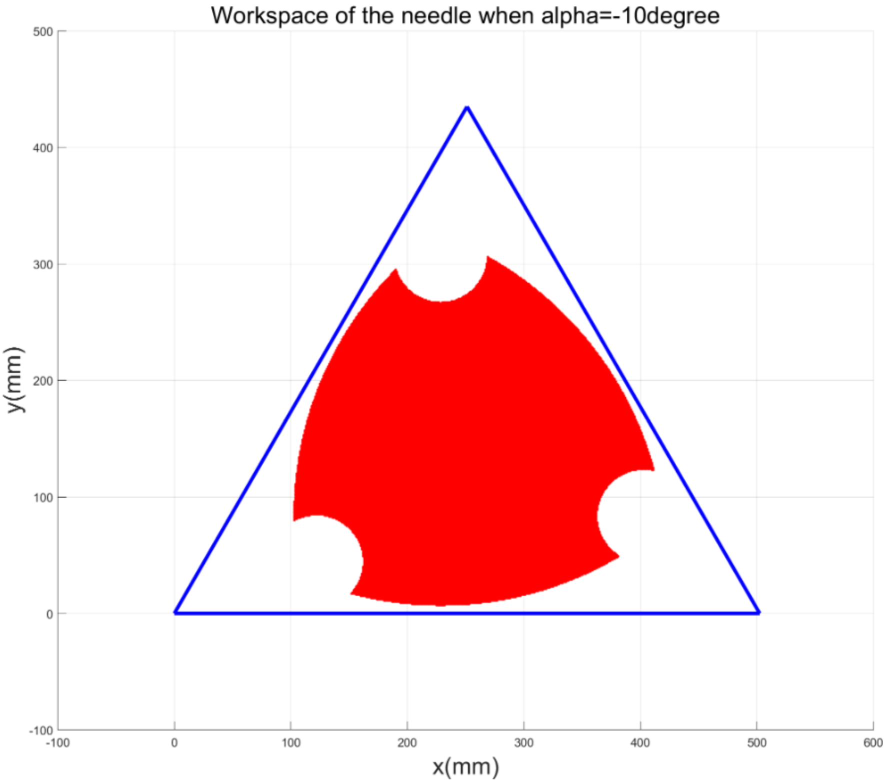


Figure 22: workspace when $\alpha = -10^{\circ}$

PART III:

the methods and algorithms that can be used to avoid problems when operating close to a singularity in robots' workspace

3.1 methods to avoid singularity point.

This is an issue that remains current and an open problem for the world of industry. Singularity is the point; the mobility of a manipulator is reduced. Usually, the arbitrary motion of the manipulator in a Cartesian direction is lost. This is referred to as "Losing a DOF". They are usually caused by a full extension of a joint and asking the manipulator to move beyond where it can be positioned. Typically, this is trying to reach out of the workspace at the farthest extent of the workspace.

Methods:
Solution 1: the robot had been programmed with a maximum speed for each joint. When the wrist joint was commanded "to infinity", this caused the software to reduce the velocity of the tip. The robot slowed down when it reached the middle of the line. When it passed the singularity, it was able to continue doing the rest of the line at the correct velocity. The paint job still would have been ruined, but the robot nevertheless functions correctly and doesn't get stuck.

Solution 2: add more axes that a robot has more possibilities for singularities. This is because there are more axes which can line up with each other. However extra axes can also reduce the effect of singularities by allowing alternative positions to reach the same point.

Solution 3: This technique is still a good way to avoid singularities. Mounting a spray-painting gun at a very slight angle (5-15 degrees) can sometimes ensure that a robot avoids singularities completely.

3.2 Algorithms to avoid singularity.

Task Reconstruction Method:
Step:
1. Define the singularity measure,
2. Calculate the vector describing the contour of the measure using the task space rather than the
joint space,
3. Eliminate the degenerate component on-line, and
4. Add restoring action.

The normal Jacobian is

$$\dot{\boldsymbol{X}} = J(q)\dot{\boldsymbol{q}} \tag{30}$$

and the $\delta\boldsymbol{q}$ can be express as:

$$\delta\boldsymbol{q} = J^{+}(q)\delta X + \left(I_n - J(p)J^{+}(q)\right)\boldsymbol{y} \quad (m \leq n) \tag{31}$$

$J^{+}(q)$ is the Moore-Penrose pseudo-inverse of $J(p)$, $\boldsymbol{y}$ is an arbitrary vector.
Assume $J(p)$ Always has a full row rank, that is, it never meets singular points. This somewhat strong assumption will always be satisfied unless the TR-method fails to work correctly.
If $\boldsymbol{y} = \boldsymbol{0}$, $\delta\boldsymbol{q} = J^{+}(q)\delta r$.
We define $m(q)$ is the singularity point. To reconstruct the desired task using the singularity measure, using the task variables to define constant surfaces of the singularity measure. The small variation of the measure can acquire it.

$$\delta m(q) = \frac{\partial m(q)}{\partial q}\delta q = \frac{\partial m(q)}{\partial q}J^{+}\delta r \tag{32}$$

In order to have $m(q) = 0$, let:

$$d = \left(\frac{\partial m(q)}{\partial q}J^{+}\right)^{T}$$

Let n_m be the unitary vector orthogonal,

$$n_m = \frac{d}{||d||}$$

The component approaching singularity should be:

$$\delta r_p = \delta r - (\delta r \times n_m)n_m = \delta r - (n_m \times {n_m}^{T})\delta r \tag{33}$$

$$= (I_m - n_m \times {n_m}^{T})\delta r$$

And then insert a shape function k_1.

$$\delta r_p = (I_m - k_1(m)n_m \times {n_m}^{T})\delta r \tag{34}$$

For shape function k_1

$$k_1(m) = \begin{cases} 0, & \bar{m} + \sigma < m \\ 2\left(\frac{m - \bar{m}}{\sigma}\right)^3 - 3\left(\frac{m - \bar{m}}{\sigma}\right)^2 + 1, & \bar{m} < m < \bar{m} + \sigma \\ x, & \bar{m} \geq m \end{cases} \tag{35}$$

Besides, to ensure that the trajectory leaves the surface by acting the task correction only when the scalar product $\delta r \times n_m$ is negative.

$$\delta r_p = \left(I_m - \frac{1 - (\delta r \times n_m)}{2} k_1 (n_m \times {n_m}^T)\right) \delta r \tag{36}$$

That when $m(q)$ is already smaller than the value on the surface, the equation mentioned before does not guarantee an escape from within the volume enclosed by the same. To avoid this situation:

$$\delta r_p = \left(I_m - \frac{1 - (\delta r \times n_m)}{2} k_1 (n_m \times {n_m}^T)\right) \delta r + k_2 n_m$$

$$k_2(m) = \begin{cases} 0, & \bar{m} < m \\ k_r(\bar{m} - m), & \bar{m} \geq m \end{cases} \tag{37}$$

Finally, the reconstructed task trajectory can be obtained as:

$$\delta r_p = TR - process(J, m(p), \delta r) \tag{38}$$

For the equation before:
$\bar{m}$: the threshold value for safety.
σ: the acting range for $TR - process$.
k_r: the escape gain.
Each parameter should be adjusted to find optimal values by the trial-and-error method. The guideline for $\bar{m}$ and σ is 3%-5% of the maximum workspace.

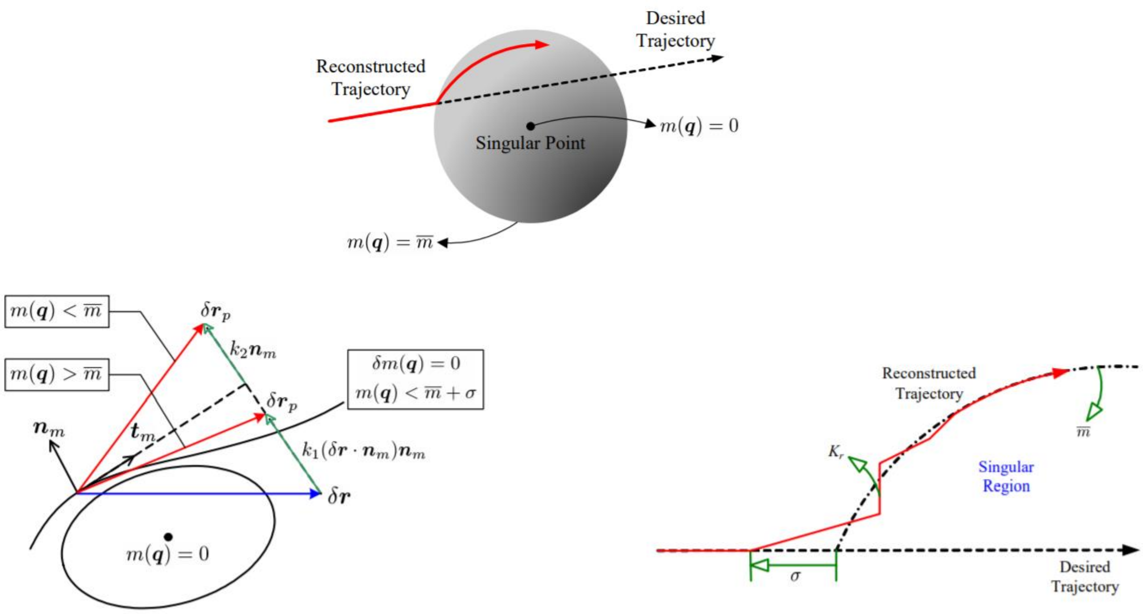


Figure 23: schematic graph of TR-process

Reference

Ii, R. and Shelley, B. (1997) *INVERSE KINEMATICS FOR PLANAR PARALLEL MANIPULATORS* [online]. Available from: https://pdfs.semanticscholar.org/45fe/2c86b6d5e882aaf9e9424ae43d803d645032.pdf [Accessed 12 December 2019].

Kim, J., Marani, G., Chung, W.K. and Yuh, J. (2006) Task reconstruction method for real-time singularity avoidance for robotic manipulators. *Advanced Robotics*. 20 (4), pp. 453–481. doi:10.1163/156855306776562233 [Accessed 12 December 2019].

Manjaree, S., Nakra, B.C. and Agarwal, V. (2015) Comparative Analysis For Kinematics Of 5-DOF Industrial Robotic Manipulator. *Acta Mechanica et Automatica* [online]. 9 (4), pp. 229–240. Available from: https://www.degruyter.com/downloadpdf/j/ama.2015.9.issue-4/ama-2015-0037/ama-2015-0037.pdfdoi:10.1515/ama-2015-0037 [Accessed 12 December 2019].

Paul, R. (1979) Manipulator Cartesian Path Control. *IEEE Transactions on Systems, Man, and Cybernetics* [online]. 9 (11), pp. 702–711. Available from: https://ieeexplore.ieee.org/abstract/document/4310109doi:10.1109/tsmc.1979.4310109 [Accessed 12 December 2019].

Xu, D., Acosta Calderon, C.A., Gan, J.Q., Hu, H. and Tan, M. (2005) An analysis of the inverse kinematics for a 5-DOF manipulator. *International Journal of Automation and Computing*. 2 (2), pp. 114–124. doi:10.1007/s11633-005-0114-1 [Accessed 11 December 2019].

Unknow. Retrieved August, 2019, Rotation matrix, Euler angle, quaternion theory and its transformation, available from: https://www.cnblogs.com/flyinggod/p/8144100.html, [Accessed 2 December 2019].

Appendix

4.1 Solution 2 to solve IK for the parallel robot.

For point B:

$$x_B = x_A + h \times cos\varphi$$
$$y_B = y_A + h \times sin\varphi$$

Hence Point C:

$$x_c = x_A + h \times \cos{(\varphi + \frac{\pi}{3})}$$

$$y_c = y_A + h \times \sin{(\varphi + \frac{\pi}{3})}$$

And the x_i and y_i can be present as:

Set PA is (0,0), so PB is (502.29,0). PC is (251.15,435)

$$x_i = x_{Pi} + r_{SA} \times \cos(\theta_1) + L \times \cos(\theta_1 + \theta_2) \quad (39)$$

$$y_i = y_{Pi} + r_{SA} \times \sin(\theta_1) + L \times \sin(\theta_1 + \theta_2) \quad (40)$$

Combine these two equations:

$$x_A^2 + y_A^2 - 2 \times r_{SA} \times (x_A \times cos\theta_1 + y_A \times \sin\theta_1) + r_{SA}^2 - L^2 = 0 \quad (41)$$

To reduce the complexity of this equation

$$e_1 = -2 \times y_A \times r_{SA}$$
$$e_2 = -2 \times x_A \times r_{SA}$$
$$e_3 = x_A^2 + y_A^2 + r_{SA}^2 - L^2$$

So the equation becomes:

$$e_1 \times sin\theta_1 + e_2 \times cos\theta_1 + e_3 = 0$$

Use the tangent half-angle substitution:

$$t = \tan{(\frac{\theta}{2})}$$

$$\sin\theta = \frac{2t}{1+t^2}$$

$$\cos\theta = \frac{1-t^2}{1+t^2}$$

And the equation could become:

$$(e_3 - e_2) \times t^2 + 2 \times e_1 \times t + (e_2 + e_3) = 0 \tag{42}$$

The solution of the equation could be:

$$\theta_1 = 2tan^{-1}\left(\frac{-e_1 \pm \sqrt{e_1^2 + e_2^2 - e_3^2}}{e_3 - e_2}\right) \tag{43}$$

4.2 FK for parallel robot

$$(A_x - M_{1x})^2 + (A_y - M_{1y})^2 = L^2$$

$$(B_x - M_{2x})^2 + \left(B_y - M_{2y}\right)^2 = L^2 \qquad (44)$$

$$(C_x - M_{3x})^2 + (C_y - M_{3y})^2 = L^2$$

From the graph, where:

$$A_x = s_A \times cos\theta_1 + L \times \cos(\theta_1 + \theta_{12})$$
$$M_{1x} = s_A \times cos\theta_1$$
$$A_y = s_A \times sin\theta_1 + L \times \sin(\theta_1 + \theta_{12})$$
$$M_{1y} = s_A \times sin\theta_1$$

And also

$$B_x = A_x + \sqrt{3} \times r_p \times \cos(a)$$

$$B_y = A_y + \sqrt{3} \times r_p \times \sin(a)$$

$$C_x = A_x + \sqrt{3} \times r_p \times \cos(a + \frac{\pi}{3})$$

$$C_y = A_y + \sqrt{3} \times r_p \times \sin(a + \frac{\pi}{3})$$

And the equation can become:

$$(A_x - M_{1x})^2 + (A_y - M_{1y})^2 = L^2$$

$$(A_x + \sqrt{3} \times r_p \times \cos(a) - M_{2x})^2 + (A_y + \sqrt{3} \times r_p \times \sin(a) - M_{2y})^2 = L^2$$

$$\left(A_x + \sqrt{3} \times r_p \times \cos\left(a + \frac{\pi}{3}\right) - M_{3x}\right)^2 + \left(A_y + \sqrt{3} \times r_p \times \sin\left(a + \frac{\pi}{3}\right) - M_{3y}\right)^2 = L^2 \qquad (45)$$

And

$$M_{1x} = s_A \times cos\theta_1$$
$$M_{1y} = s_A \times sin\theta_1$$

$$M_{2x} = s_A \times cos\theta_2 + \sqrt{3} \times 290 \qquad (46)$$

$$M_{2y} = s_A \times sin\theta_2$$

$$M_{3x} = s_A \times cos\theta_1 + \frac{\sqrt{3}}{2} \times 290 \qquad (47)$$

$$M_{3y} = s_A \times sin\theta_1 + \frac{3}{2} \times 290$$

4.3 Second way to generate straight line trajectory.

The position and orientation of the manipulator is defined by a homogeneous transformation 4*4 matrix as shown below, the first 3 columns*3 rows known as the rotation matrix, it contains the orientation information; The forth column is the position vector, contains the x y z information of the end effector.

$$P(h) = \begin{bmatrix} nx & ox & ax & px \\ ny & oy & ay & py \\ nz & oz & az & pz \\ 0 & 0 & 0 & 1 \end{bmatrix} = [n\ o\ a\ p] \tag{48}$$

In this method, we decomposed the motion between two points into one translation and two rotations. The translation indicates the position changes, and for two rotations, one is to align the tool in the required final direction and the second rotation means the rotation of the final joint. And the process of the straight-line motion between Point A and Point B can be expressed in an intermediate matrix D(h) as

$$D(h) = P(h) \cdot Ra(h) \cdot Ro(h) \tag{49}$$

where h = t/T, t is the time since the motion start and T is the whole time of the motion. P(h),Ra(h),Ro(h) indicate the process of one translation and two rotations as mentioned above in the order respectively. And when h=1 means the motion complete and the end effector has arrived point B from point A, it can be expressed as

$$T_2 = T_1 \cdot D(1) \tag{50}$$

Where T2 and T1 are the homogeneous transformation matrix in point2 and point1 respectively. And the P, Ra and Ro have the following form:

$$P(h) = \begin{bmatrix} 1 & 0 & 0 & hx \\ 0 & 1 & 0 & hy \\ 0 & 0 & 1 & hz \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{51}$$

$$Ra(h) = \begin{bmatrix} s\varphi^2 v(h\theta) + c(h\theta) & -s\varphi c\varphi v(h\theta) & c\varphi s(h\theta) & 0 \\ -s\varphi c\varphi v(h\theta) & c\varphi^2 v(h\theta) + c(h\theta) & s\varphi s(h\theta) & 0 \\ -c\varphi s(h\theta) & -s\varphi s(h\theta) & c(h\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{52}$$

$$Ro(h) = \begin{bmatrix} c(h\gamma) & -s(h\gamma) & 0 & 0 \\ s(h\gamma) & c(h\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

And

$$v(h\theta) = 1 - \cos(h\theta) \tag{53}$$

Ra(h) represents a rotation about a unit vector which is the orientation vector of A, the rotation angle is θ, and φ is the angle indicating the A_O rotating about the approach vector of A. Ro(h) expresses a rotation of γ about the approach vector of the end effector.

By combining the equation (2)(3) with equations (4)(5)(6) we can obtain:

$$\begin{cases} x = An' \cdot (Bp - Ap) \\ y = Ao' \cdot (Bp - Ap) \\ z = Aa' \cdot (Bp - Ap) \end{cases} \tag{54}$$

$$tan\varphi = \frac{Ao' \cdot Ba}{An' \cdot Ba}$$

$$tan\theta = \frac{\sqrt{(An' \cdot Ba)^2 + (Ao' \cdot Ba)^2}}{Aa' \cdot Ba} \tag{55}$$

$$sin\gamma = -s\varphi c\varphi v(h\theta)(An' \cdot Bn) + \left(c\varphi^2 v(h\theta) + c(h\theta)\right) \cdot (Ao' \cdot Bn) - s\varphi s(h\theta) \cdot (Aa' \cdot Bn)$$

$$cos\gamma = -s\varphi c\varphi v(h\theta)(An' \cdot Bo) + \left(c\varphi^2 v(h\theta) + c(h\theta)\right) \cdot (Ao' \cdot Bo) - s\varphi s(h\theta) \cdot (Aa' \cdot Bo)$$

$$tan\gamma = \frac{sin\gamma}{cos\gamma} \tag{56}$$

We test this method in Matlab, for details see SLTraj2.m

4.4 MatLab code for the coursework

We wrote a total program for the coursework, includes both the examples of Lynxmotion arm and parallel robot, it will be shown in the end of this section.

1.1 FowKi.m (function, solve the forward kinematic)

```
%%%%%%%%Function of Forward Kinematic %%%%%%%%%%%%%%
% The input (q1,q2,q3,q4,q5) is the 5 joint angle in the order
% The output is the 4*4 transformation matrix
% The range of the input angle is:
% q1: 0----pi;
% q2: 0----5*pi/6
% q3: -5*pi/6---0
% q4: 0------pi
% q5: 0------pi
% Example: T = FowKi(2*pi/3,pi/3,-pi/4,pi/4,0)
% Example: T = FowKi(1.2,0.5,-0.8,1,0)
function [T] = FowKi(q1,q2,q3,q4,q5)
T = [ sin(q1)*sin(q5) + cos(q2 + q3 + q4)*cos(q1)*cos(q5),      cos(q5)*sin(q1) -
cos(q2 + q3 + q4)*cos(q1)*sin(q5), sin(q2 + q3 + q4)*cos(q1),
(cos(q1)*(27*sin(q2 + q3 + q4) + 59*cos(q2 + q3) + 46*cos(q2)))/8;
cos(q2 + q3 + q4)*cos(q5)*sin(q1) - cos(q1)*sin(q5), - cos(q1)*cos(q5) -
cos(q2 + q3 + q4)*sin(q1)*sin(q5), sin(q2 + q3 + q4)*sin(q1),
(sin(q1)*(27*sin(q2 + q3 + q4) + 59*cos(q2 + q3) + 46*cos(q2)))/8;
                                          sin(q2 + q3 + q4)*cos(q5),
-sin(q2 + q3 + q4)*sin(q5),             -cos(q2 + q3 + q4),     (59*sin(q2 + q3))/8 -
(27*cos(q2 + q3 + q4))/8 + (23*sin(q2))/4;
                                                                                0,
0,                                      0,
1];
```

1.2 Workspace.m (script, solve the workspace)

```
%% Caculate the position
i = 0;
xwork=zeros(1,100048);
ywork=zeros(1,100048);
zwork=zeros(1,100048);
for q1=0:pi/60:pi
    for q2=0:pi/15:5*pi/6
        for q3=-5*pi/6:pi/15:0
            for q4=0:pi/15:pi
                i=i+1;
                T0e = FowKi(q1,q2,q3,q4,0);
                xwork(i) = T0e(1,4);
                ywork(i) = T0e(2,4);
```

```
                    zwork(i) = T0e(3,4);
                end
            end
        end
    end

%% 3D plot
c=zwork;
figure
scatter3(xwork,ywork,zwork,6,c,'.')
title('3D Workspace','Fontsize',15)
xlabel('x(inch)','Fontsize',15)
ylabel('y(inch)','Fontsize',15)
zlabel('z(inch)','Fontsize',15)
grid on
axis equal
view(-75,20);

%% 2D plot in Z direction
figure
scatter(xwork,ywork,6,c,'.')
title('2D Workspace in Z','Fontsize',15)
xlabel('x(inch)','Fontsize',15)
ylabel('y(inch)','Fontsize',15)
grid on
axis equal

%% 2D plot in Y direction
figure
scatter(xwork,zwork,6,c,'.')
title('2D Workspace in Y','Fontsize',15)
xlabel('x(inch)','Fontsize',15)
ylabel('z(inch)','Fontsize',15)
grid on
axis equal

%% 2D plot in X direction
figure
scatter(ywork,zwork,6,c,'.')
title('2D Workspace in X','Fontsize',15)
xlabel('y(inch)','Fontsize',15)
ylabel('z(inch)','Fontsize',15)
grid on
axis equal
```

1.3 InvKi.m (function, solve the inverse kinematic)

```
%%%%%%%%Function of Inverse Kinematic %%%%%%%%%%%%%%
% The input is the 4*4 transformation matirx of the manipulator
% The output is the vector contains 5 joint angle in the order of: [q1 q2
% q3 q4 q5]
% Example1: InvKi([      0.1492      0.4794      0.8648     10.5954;
%      0.0815     -0.8776      0.4724      5.7883;
%     0.9854              0     -0.1700      9.0159;
%              0              0              0      1.0000;])
% Example2: InvKi(FowKi(0.5,1,-0.3,0.7,0))
% i1,i2,i3,i4,i5 correspond to q1,q2,q3,q4,q5 respectively
% There are 2 sets of solution, but only one sets is correct


function ik = InvKi(T0e)
d2=5.75; d3=7.375; d4=3.375;

nx = T0e(1,1); ny = T0e(2,1); nz = T0e(3,1);
ox = T0e(1,2); oy = T0e(2,2); oz = T0e(3,2);
ax = T0e(1,3); ay = T0e(2,3); az = T0e(3,3);
xe = T0e(1,4); ye = T0e(2,4); ze = T0e(3,4);

%% Solving q1
i1 = atan2(ye,xe);
if i1<0                  % When the elevation angle of the manipulator is very
large and makes ye<0, atan2(ye,xe) will smaller than 0.
      i1 = i1+pi;       % At this situation, we need to modify q1
end

%% Solving q2
Z4 = ze-az*d4; D4 = (xe-ax*d4)/cos(i1);
if D4 == 0
      D4 = ((xe-ax*d4)^2+(ye-ay*d4)^2)^0.5;
end
beta = atan2(D4,Z4);

i2 = zeros(1,2); % There are two solution for q2
s2beta = (Z4^2+D4^2+d2^2-d3^2)/(2*d2*(Z4^2+D4^2)^0.5);
c2beta1 = (1-s2beta^2)^0.5;
c2beta2 = -(1-s2beta^2)^0.5;
i2(1) = atan2(s2beta,c2beta1)-beta; % the first solution of q2
i2(2) = atan2(s2beta,c2beta2)-beta; % the second solution for q2
```

```
%% Solving q3
i3 = zeros(1,2); % Because q3 is related to q2, so q3 also has two solutions
i3(1) = atan2(Z4-d2*sin(i2(1)),D4-d2*cos(i2(1)))-i2(1); % the first solution for
q3
i3(2) = atan2(Z4-d2*sin(i2(2)),D4-d2*cos(i2(2)))-i2(2); % the second solution
for q3

%% Solving q4
i4 = zeros(1,2); % Because q4 is related to q2 and q3, so q4 also has two
solutions
i4(1) = atan2(nz,nx*cos(i1)+ny*sin(i1))-i2(1)-i3(1); % the first solution for q4
i4(2) = atan2(nz,nx*cos(i1)+ny*sin(i1))-i2(2)-i3(2); % the second solution for
q4


if i2(1)<0 || i3(1)>0 % According to the range of the joint angle, the correct
solution should follow q2>0 and q3<0
    ik = [i1 i2(2) i3(2) i4(2) 0];
else
    ik = [i1 i2(1) i3(1) i4(1) 0];
end

if ik(4)<0
    ik(4) = ik(4)+2*pi;
elseif ik(4)>pi
    ik(4) = ik(4)-pi;
end
```

1.4 IKRandomCheck.m (script, IK function self-inspection)

```
% This programm is for check the correctness of the inverse kinematic
function: InvKi
% It randomly generate 100000 sets of joint angle (q1,q2,q3,q4,q5) in their
range
% And check the answer of InvKi(FowKi(q1,q2,q3,q4,q5)) whether is equal to
% (q1,q2,q3,q4,q5)
% If the answer doesn't equal to (q1,q2,q3,q4,q5), the programm will break
% and show that answer and its correspond (q1,q2,q3,q4,q5)
for i = 1:100000
q1 = rand()*pi; %Randomly generate q1,q2,q3,q4,q5
q2 = rand()*2*pi/3;
q3 = -rand()*2*pi/3;
q4 = rand()*pi;
q5=0;
```

```
ik = InvKi(FowKi(q1,q2,q3,q4,q5));
i

if abs(ik(1)-q1)>10e-10 || abs(ik(2)-q2)>10e-10 || abs(ik(3)-q3)>10e-10 ||
abs(ik(4)-q4)>10e-10 % Checking
        ik
        [q1 q2 q3 q4 q5]
        break
end
end
```

1.5 FMTraj.m (function, generate free motion trajectory)

```
%%%%%%%%%%%% Function of Free Motion%%%%%%%%%%%%
% This function will output the anime plot of the free motion, and the free
% motion process consists of three part: acceleration section, constant
% velocity section and decceleration section
% This function has 5 input:
%       qs is the joint angles at the starting point of motion, it should be a
%       1*5 vector [q1.q2,q3,q4,q5]
%       qe is the joint angles at the ending point of motion, it should be a
%       1*5 vector [q1,q2,q3,q4,q5]
%       tt is the total time of the motion
%       ta is the time of acceleration as well as decceleration
% The input parameters tt and ta should obey this relation: tt >= 2*ta
  % The color of free motion trajectory is blue

% Example: FMTraj([0,0,0,0,0],[1.2,0.9,-0.68,1,0],3,1)

function [] = FMTraj(qs,qe,tt,ta)
tau=ta/2;T=tt-2*tau;
m=10*(T+2*tau)+1;
qi = cell(1,m);
TMi = cell(1,m);

%% Define the lux manipulator in robotics toolbox (in order to plot by toolbox)
L1=Link('a',0,'d',0,'alpha',pi/2);
L2=Link('a',5.75,'d',0,'alpha',0);
L3=Link('a',7.375,'d',0,'alpha',0);
L4=Link('a',0,'d',0,'alpha',pi/2);
L5=Link('a',0,'d',3.375,'alpha',0);
lynx = SerialLink([L1 L2 L3 L4 L5], 'name', 'lynx');

%% The acceleration section
```

```
for t = -tau:0.1:tau
	k = round(10*(t+tau)+1);
	qi{k} = qs+(qe-qs).*(t+tau)^2./(4*T*tau);
	TMi{k} = FowKi(qi{k}(1),qi{k}(2),qi{k}(3),qi{k}(4),qi{k}(5));
end

%% The constant velocity section
for t = tau+0.1:0.1:T-tau
	k = round(10*(t+tau)+1);
	qi{k} = qs+(qe-qs).*t./T;
	TMi{k} = FowKi(qi{k}(1),qi{k}(2),qi{k}(3),qi{k}(4),qi{k}(5));
end

%% The decceleration section
for t = T-tau+0.1:0.1:T+tau
	k = round(10*(t+tau)+1);
	j = k-10*T;
	qi{k} = 2.*qs+(qe-qs).*t./T-qi{j};
	TMi{k} = FowKi(qi{k}(1),qi{k}(2),qi{k}(3),qi{k}(4),qi{k}(5));
end

%% plotting
for k = 1:m
	lynx.plot(qi{k},'tilesize',3); % Plot the manipulator by robotics toolbox
	Te = TMi{k};
	hold on
	plot3(Te(1,4),Te(2,4),Te(3,4),'b.') % Plot the trajectory point
	hold off
	pause(0.01);
end
```

1.6 SLTraj.m (function, generate straight-line trajectory)

```
%%%%%%%% Function of straight line trajectory %%%%%%%%%%%%%%
% This function will output the anime plot of the straight line motion, and
% the the straight line motion process consists of three part: acceleration
% section, constant velocity section and deceleration section
% This function has 4 input:
%	T1 is the transformation matrix of the starting point of motion, it should
be a
%	4*4 matrix
%	T2 is the transformation matrix of the ending point of motion, it should
be a
%	4*4 matrix
%	tt is the total time of the motion
%	ta is the time of acceleration as well as decceleration
```

```
% The input parameters tt and ta should obey this relation: tt >= 2*ta

% The color of straight line motion trajectory is red

%Example:
% Ts=FowKi(2*pi/3,0,0,0,0);
% Te=FowKi(75*pi/180,pi/6,-pi/6,0,0);
% SLTraj(Ts,Te,3,1)
function [] = SLTraj(Ts,Te,tt,ta)
tau=ta/2; T = tt-2*tau;
m=10*(T+2*tau)+1;
Roti = cell(1,m);
Posi = cell(1,m);
Ti = cell(1,m);
qi = cell(1,m);
TMi = cell(1,m);
pos1 = Ts(1:3,4);
pos2 = Te(1:3,4);
rot1 = Ts(1:3,1:3);
rot2 = Te(1:3,1:3);

%% Define the lux manipulator in robotics toolbox (in order to plot by toolbox)
L1=Link('a',0,'d',0,'alpha',pi/2);
L2=Link('a',5.75,'d',0,'alpha',0);
L3=Link('a',7.375,'d',0,'alpha',0);
L4=Link('a',0,'d',0,'alpha',pi/2);
L5=Link('a',0,'d',3.375,'alpha',0);
lynx = SerialLink([L1 L2 L3 L4 L5], 'name', 'lynx');

%% Transfer the rotation matrix to Euler angle(ZYX), and then transfer the
Euler angle to quaternion
eul1 = rotm2eul(rot1);
eul2 = rotm2eul(rot2);
quat1 = quaternion(eul1,'eulerd','ZYX','frame');
quat2 = quaternion(eul2,'eulerd','ZYX','frame');
InterpoQuat = quaternion();

%% The acceleration section
for t = -tau:0.1:tau
    i = round(10*(t+tau)+1);
    InterpoQuat(i) = quat1+(quat2-quat1).*(t+tau)^2./(4*T*tau);
    Posi{i} = pos1+(pos2-pos1)*(t+tau)^2/(4*T*tau);
end
```

```
%% The constant velocity section
for t = tau+0.1:0.1:T-tau
    i = round(10*(t+tau)+1);
    InterpoQuat(i) = quat1+(quat2-quat1).*t./T;
    Posi{i} = pos1+(pos2-pos1)*t/T;
end

%% The decceleration section
for t = T-tau+0.1:0.1:T+tau
    i = round(10*(t+tau)+1);
    j = i-10*T;
    InterpoQuat(i) = 2.*quat1+(quat2-quat1).*t./T-InterpoQuat(j);
    Posi{i} = 2.*pos1+(pos2-pos1)*t/T-Posi{j};
end

%% Generate the transformation matrix and joint angle for each point
q = eulerd(InterpoQuat,'ZYX','frame'); % transfer all the quaternion to euler angle

for k = 1:m
    try
    Roti{k} = eul2rotm(q(k,:)); % transfer the euler angle to rotation matrix
    Ti{k} = [Roti{k}, Posi{k};0 0 0 1]; % combine the ratation matrix and position vector to generate the transformation matrix
    qi{k} = InvKi(Ti{k}); % Use the function of inverse kinematic to generate the joint angle [q1 q2 q3 q4 q5]
    TMi{k} = FowKi(qi{k}(1),qi{k}(2),qi{k}(3),qi{k}(4),qi{k}(5)); % Use the function of foward kinematic to generate the transformation matrix corresponding to joint angle qi
    end
end

%% Plotting
for k = 1:m
    try
    lynx.plot(qi{k},'tilesize',3); % Plot the manipulator by robotics toolbox
    Te = TMi{k};
    hold on
    plot3(Te(1,4),Te(2,4),Te(3,4),'r.') % Plot the trajectory point
    hold off
    pause(0.01);
    end
end
```

1.7 TrajectoryTask.m (script, doing the “OK” task)

```
% This programm will complete the coursework trajectory task.
% in our task, the manipulator will writing a letter "OK" in a clined plane
% There are an obstacle between the word "O" and "K", and an obstacle in
% the returning path, the manipulator will avoid them
%% Define the task points
clear
T1=FowKi(104*pi/180,57*pi/180,-59*pi/180,40*pi/180,0);
T2=FowKi(109*pi/180,79*pi/180,-93*pi/180,25*pi/180,0);
T3=FowKi(97*pi/180,83*pi/180,-97*pi/180,26*pi/180,0);
T4=FowKi(95*pi/180,60*pi/180,-63*pi/180,40*pi/180,0);
q1 = [104*pi/180 57*pi/180 -59*pi/180 40*pi/180 0];
q2 = [pi/2 pi/2 -pi/6 0 0];
q3 = [81*pi/180 60*pi/180 -63*pi/180 40*pi/180 0];
T5=FowKi(81*pi/180,60*pi/180,-63*pi/180,40*pi/180,0);
T6=FowKi(77.7*pi/180,83*pi/180,-97.5*pi/180,26*pi/180,0);
q4 = [77.7*pi/180 83*pi/180 -97.5*pi/180 26*pi/180 0];
q5 = [77.7*pi/180 100*pi/180 -60*pi/180 26*pi/180 0];
q6 = [75.5*pi/180 57*pi/180 -59*pi/180 41*pi/180 0];
T7=FowKi(75.5*pi/180,57*pi/180,-59*pi/180,41*pi/180,0);
T8=FowKi(79*pi/180,70*pi/180,-77*pi/180,29*pi/180,0);
T9=FowKi(64.7*pi/180,72.7*pi/180,-84.7*pi/180,19.4*pi/180,0);
q7=[64.7*pi/180 72.7*pi/180 -84.7*pi/180 19.4*pi/180 0];
q8=[30*pi/180 90*pi/180 -30*pi/180 30*pi/180 0];
q9=[0 0 0 0 0];

%% Plotting
plotcube([0.4 10 3.5],[-0.2 8 -2],.8,[0 1 1]) % generate a cube obstacle
plotcube([10 1 8],[5 4.5 -2],.8,[0 1 1]) % generate a cube obstacle
SLTraj(T1,T2,2,0.5);
SLTraj(T2,T3,2,0.5);
SLTraj(T3,T4,2,0.5);
SLTraj(T4,T1,2,0.5);
FMTraj(q1,q2,2,0.5);
FMTraj(q2,q3,2,0.5);
SLTraj(T5,T6,2,0.5);
FMTraj(q4,q5,2,0.5);
FMTraj(q5,q6,2,0.5);
SLTraj(T7,T8,2,0.5);
SLTraj(T8,T9,2,0.5);
FMTraj(q7,q8,2,0.5);
FMTraj(q8,q9,2,0.5);
```

1.7 SLTraj2.m (script, second way to generate straight-line)

```
clear
clc
L1=Link('a',0,'d',0,'alpha',pi/2);
L2=Link('a',5.75,'d',0,'alpha',0);
L3=Link('a',7.375,'d',0,'alpha',0);
L4=Link('a',0,'d',0,'alpha',pi/2);
L5=Link('a',0,'d',3.375,'alpha',0);
robot = SerialLink([L1 L2 L3 L4 L5], 'name', 'my robot');


T1=FowKi(2*pi/3,0,0,0,0);
T2=FowKi(75*pi/180,pi/6,-pi/6,0,0);

An = T1(:,1); Ao = T1(:,2); Aa = T1(:,3); Ap = T1(:,4);
Bn = T2(:,1); Bo = T2(:,2); Ba = T2(:,3); Bp = T2(:,4);

x = An'*(Bp-Ap); y = Ao'*(Bp-Ap); z = Aa'*(Bp-Ap);


fai = atan2(Ao'*Ba,An'*Ba);
theta = atan2(((An'*Ba)^2+(Ao'*Ba)^2)^0.5,Aa'*Ba);
sf = sin(fai); cf = cos(fai); st = sin(theta); ct = cos(theta); vt = 1-cos(theta);
sg = -sf*cf*vt*(An'*Bn)+((cf^2)*vt+ct)*(Ao'*Bn)-sf*st*(Aa'*Bn);
cg = -sf*cf*vt*(An'*Bo)+((cf^2)*vt+ct)*(Ao'*Bo)-sf*st*(Aa'*Bo);
gama = atan2(sg,cg);

Ti = cell(1,11);
qi = cell(1,11);
TMi = cell(1,11);
Ti{1} = T1;
qi{1} = InvKi(T1);
TMi{1} = FowKi(qi{1}(1),qi{1}(2),qi{1}(3),qi{1}(4),qi{1}(5));

for k=0.1:0.1:1
    vtk = 1-cos(k*theta);
    ctk = cos(k*theta);
    stk = sin(k*theta);
    cgk = cos(k*gama);
    sgk = sin(k*gama);

Th = [1 0 0 x*k;0 1 0 y*k;0 0 1 z*k; 0 0 0 1];
```

```
Rah = [(sf^2)*vtk+ctk, -sf*cf*vtk, cf*stk, 0;
        -sf*cf*vtk, (cf^2)*vtk+ctk, sf*stk, 0;
        -cf*stk, -sf*stk, ctk, 0;
        0 0 0 1];


Roh = [cgk -sgk 0 0;sgk cgk 0 0; 0 0 1 0;0 0 0 1];

i = 10*k+1;
Ti{i} = T1*Th*Rah*Roh;
try
    qi{i} = InvKi(Ti{i});
    TMi{i} = FowKi(qi{i}(1),qi{i}(2),qi{i}(3),qi{i}(4),qi{i}(5));
end
end

for m = 1:11
     try
     robot.plot(qi{m},'tilesize',3);
     Te = TMi{m};
     hold on
     plot3(Te(1,4),Te(2,4),Te(3,4),'r.')
     hold off
     pause(0.3);
     end
end
```

2.1 ParallelIKPlot.m (function, solve parallel IK and plot)

```
% This function will solve the inverse kinematic problem of the coursework
parallel robot.
% The input xc and yc are the coordinate of the needle, and alpha is the angle
%between the platform and horizontal direction. And the function will output a
8¡Á3 matrix,
%its each line is a set of solution of three active angles.
%The function will also output the plot of eight postures.
function theta = ParallelIKPlot(xc,yc,alpha)
theta1=zeros(1,2); theta2=zeros(1,2); theta3=zeros(1,2);
q1 = zeros(1,2);    q2 = zeros(1,2); q3 = zeros(1,2);
alpha=alpha*pi/180;
rp=130; rb=290; S=170; L=130;
f1 = alpha+pi/6; f2 = alpha+5*pi/6; f3 = alpha+3*pi/2;

%% theta1
```

```
p1= atan2(yc-rp*sin(f1),xc-rp*cos(f1));
cq1 = (S^2-L^2+(xc-rp*cos(f1))^2+(yc-rp*sin(f1))^2)/(2*S*((xc-
rp*cos(f1))^2+(yc-rp*sin(f1))^2)^0.5);
sq1 = (1-cq1^2)^0.5;
q1(1) = atan2(sq1,cq1);
q1(2) = atan2(-sq1,cq1);
p1 = p1.*180./pi;
q1 = q1.*180./pi;
theta1(1) = p1+q1(1); theta1(2) = p1+q1(2);

%% theta2
p2 = atan2(yc-rp*sin(pi-f2),xc+rp*cos(pi-f2)-(3^0.5)*rb);
cq2 = (S^2-L^2+(xc+rp*cos(pi-f2)-(3^0.5)*rb)^2+(yc-rp*sin(pi-
f2))^2)/(2*S*((xc+rp*cos(pi-f2)-(3^0.5)*rb)^2+(yc-rp*sin(pi-f2))^2)^0.5);
sq2 = (1-cq2^2)^0.5;
q2(1) = atan2(sq2,cq2);
q2(2) = atan2(-sq2,cq2);
p2 = p2.*180./pi;
q2 = q2.*180./pi;
theta2(1) = p2+q2(1); theta2(2) = p2+q2(2);

% if alpha < 0
%       theta2 = 360+theta2;
% end

%% theta3
p3 = atan2(yc+rp*sin(2*pi-f3)-3*rb/2, xc-rp*cos(2*pi-f3)-(3^0.5)*rb/2);
cq3 = (S^2-L^2+(xc-rp*cos(2*pi-f3)-(3^0.5)*rb/2)^2+(yc+rp*sin(2*pi-f3)-
3*rb/2)^2)/(2*S*((xc-rp*cos(2*pi-f3)-(3^0.5)*rb/2)^2+(yc+rp*sin(2*pi-f3)-
3*rb/2)^2)^0.5);
sq3 = (1-cq3^2)^0.5;
q3(1) = atan2(sq3,cq3);
q3(2) = atan2(-sq3,cq3);
p3 = p3.*180./pi;
q3 = q3.*180./pi;
theta3(1) = p3+q3(1); theta3(2) = p3+q3(2);

%% solution
theta = [theta1(1) theta2(1) theta3(1); theta1(1) theta2(1) theta3(2); theta1(1)
theta2(2) theta3(1); theta1(1) theta2(2) theta3(2);
            theta1(2) theta2(1) theta3(1); theta1(2) theta2(1) theta3(2);
theta1(2) theta2(2) theta3(1); theta1(2) theta2(2) theta3(2)];

%% Plotting
```

```
thetai = theta.*pi./180;
bax=0; bay=0; bbx=290*3^0.5; bby=0; bcx=145*3^0.5; bcy=435;
pax=xc-rp*cos(f1); pay=yc-rp*sin(f1);
pbx=xc+rp*cos(pi-f2); pby=yc-rp*sin(pi-f2);
pcx=xc-rp*cos(2*pi-f3); pcy=yc+rp*sin(2*pi-f3);

for kk = 1:8
        max = 170*cos(thetai(kk,1)); may = 170*sin(thetai(kk,1));
        mbx = 170*cos(thetai(kk,2))+290*3^0.5; mby = 170*sin(thetai(kk,2));
        mcx = 170*cos(thetai(kk,3))+145*3^0.5; mcy =
170*sin(thetai(kk,3))+435;
        subplot(2,4,kk);
        hold on
        plot([pax,pbx,pcx,pax],[pay,pby,pcy,pay],'r','Linewidth',2)
plot([bax,bbx],[bay,bby],'b','Linewidth',3)
plot([bax,bcx],[bax,bcy],'b','Linewidth',3)
plot([bbx,bcx],[bby,bcy],'b','Linewidth',3)
plot([bax,max],[bay,may],'k','Linewidth',2)
plot([bbx,mbx],[bby,mby],'k','Linewidth',2)
plot([bcx,mcx],[bcy,mcy],'k','Linewidth',2)
plot([max,pax],[may,pay],'k','Linewidth',2)
plot([mbx,pbx],[mby,pby],'k','Linewidth',2)
plot([mcx,pcx],[mcy,pcy],'k','Linewidth',2)
xlabel('x(mm)','Fontsize',13)
ylabel('y(mm)','Fontsize',13)
title(['Posture No.',num2str(kk)],'Fontsize',13)
axis equal
axis([-100 600 -100 500]);
grid on
hold off
end
suptitle(['Models when xc=',num2str(xc),'mm    yc=',num2str(yc),'mm
alpha=',num2str(alpha*180/pi)]);
```

2.2 ParallelWorkspace.m (function, generate the workspace)

```
% This function will plot the workspace for a given orientation alpha
function []=ParallelWorkspace(alpha)
%% Recording the workspace
i=0;
x = zeros(1,10000); y = zeros(1,10000);
for xc = 70:1:420
    for yc = 0:1:320
        try
            ParallelIK(xc,yc,alpha);
```

```
            i = i+1;
            x(i) = xc;
            y(i) = yc;
        end
    end
end
x = x(x~=0); y = y(y~=0);

%% Plotting
bax=0; bay=0; bbx=290*3^0.5; bby=0; bcx=145*3^0.5; bcy=435;
hold on
plot([bax,bbx],[bay,bby],'b','Linewidth',3)
plot([bax,bcx],[bax,bcy],'b','Linewidth',3)
plot([bbx,bcx],[bby,bcy],'b','Linewidth',3)
plot(x,y,'r.')
xlabel('x(mm)','Fontsize',20)
ylabel('y(mm)','Fontsize',20)
title(['Workspace of the needle when
alpha=',num2str(alpha),'degree'],'Fontsize',20)
axis equal
axis([-100 600 -100 500]);
grid on
hold off
```

Coursework.m (Total program of our coursework)

```
clear
clc
%% Part 1
disp('Part 1');
%% Forward kinematic
disp('A forward kinematic example q1=[2*pi/3,pi/3,-pi/4,pi/4,0]');
T1 = FowKi(2*pi/3,pi/3,-pi/4,pi/4,0)
disp('Another forward kinematic example q2=[1.2,0.5,-0.8,1,0] ');
T2 = FowKi(1.2,0.5,-0.8,1,0)
input('Foward kinematic done. Press enter to continue \n');
%% Inverse kinematic
disp('Using InvKi to solve the inverse kinematic of T1');
q1 = InvKi(T1)
disp('Using InvKi to solve the inverse kinematic of T2');
q2 = InvKi(T2)
disp('Another inverse kinematic example q=[0.5,1,-0.3,0.7,0]');
q3 = InvKi(FowKi(0.5,1,-0.3,0.7,0))
input('Inverse kinematic done. Press enter to continue \n');
%% Workspace
```

```
clear
disp('The workspace of Lynxmotion arm')
i = 0;
xwork=zeros(1,100048);
ywork=zeros(1,100048);
zwork=zeros(1,100048);
for q1=0:pi/60:pi
    for q2=0:pi/15:5*pi/6
        for q3=-5*pi/6:pi/15:0
            for q4=0:pi/15:pi
                    i=i+1;
                    T0e = fk(q1,q2,q3,q4,0);
                    xwork(i) = T0e(1,4);
                    ywork(i) = T0e(2,4);
                    zwork(i) = T0e(3,4);
            end
        end
    end
end

c=zwork;
figure('Position', [30,550,560,420]);
scatter3(xwork,ywork,zwork,6,c,'.')
title('3D Workspace','Fontsize',15)
xlabel('x(inch)','Fontsize',15)
ylabel('y(inch)','Fontsize',15)
zlabel('z(inch)','Fontsize',15)
grid on
axis equal
view(-75,20);

figure('Position', [830,550,560,420]);
scatter(xwork,ywork,6,c,'.')
title('2D Workspace in Z','Fontsize',15)
xlabel('x(inch)','Fontsize',15)
ylabel('y(inch)','Fontsize',15)
grid on
axis equal

figure('Position', [30,50,560,420]);
scatter(xwork,zwork,6,c,'.')
title('2D Workspace in Y','Fontsize',15)
xlabel('x(inch)','Fontsize',15)
ylabel('z(inch)','Fontsize',15)
```

```
grid on
axis equal

figure('Position', [830,50,560,420]);
scatter(ywork,zwork,6,c,'.')
title('2D Workspace in X','Fontsize',15)
xlabel('y(inch)','Fontsize',15)
ylabel('z(inch)','Fontsize',15)
grid on
axis equal
input('Workspace done. Press enter to continue \n');
close
close
close
close
%% Free motion
disp('A free motion trajectory example')
figure
FMTraj([0,0,0,0,0],[1.2,0.9,-0.68,1,0],3,1)
input('Free motion done. Press enter to continue \n');
close
%% Straight-line motion
disp('A straight-line trajectory example')
figure
Ts=FowKi(2*pi/3,0,0,0,0);
Te=FowKi(75*pi/180,pi/6,-pi/6,0,0);
SLTraj(Ts,Te,3,1)
input('Straight-line motion done. Press enter to continue \n');
close
%% Trajectory task
disp('The trajectory task: writing a "OK" and avoid the obstacles')
clear
T1=FowKi(104*pi/180,57*pi/180,-59*pi/180,40*pi/180,0);
T2=FowKi(109*pi/180,79*pi/180,-93*pi/180,25*pi/180,0);
T3=FowKi(97*pi/180,83*pi/180,-97*pi/180,26*pi/180,0);
T4=FowKi(95*pi/180,60*pi/180,-63*pi/180,40*pi/180,0);
q1 = [104*pi/180 57*pi/180 -59*pi/180 40*pi/180 0];
q2 = [pi/2 pi/2 -pi/6 0 0];
q3 = [81*pi/180 60*pi/180 -63*pi/180 40*pi/180 0];
T5=FowKi(81*pi/180,60*pi/180,-63*pi/180,40*pi/180,0);
T6=FowKi(77.7*pi/180,83*pi/180,-97.5*pi/180,26*pi/180,0);
q4 = [77.7*pi/180 83*pi/180 -97.5*pi/180 26*pi/180 0];
q5 = [77.7*pi/180 100*pi/180 -60*pi/180 26*pi/180 0];
q6 = [75.5*pi/180 57*pi/180 -59*pi/180 41*pi/180 0];
```

```
T7=FowKi(75.5*pi/180,57*pi/180,-59*pi/180,41*pi/180,0);
T8=FowKi(79*pi/180,70*pi/180,-77*pi/180,29*pi/180,0);
T9=FowKi(64.7*pi/180,72.7*pi/180,-84.7*pi/180,19.4*pi/180,0);
q7=[64.7*pi/180 72.7*pi/180 -84.7*pi/180 19.4*pi/180 0];
q8=[30*pi/180 90*pi/180 -30*pi/180 30*pi/180 0];
q9=[0 0 0 0 0];

figure
plotcube([0.4 10 3.5],[-0.2 8 -2],.8,[0 1 1]) % generate a cube obstacle
plotcube([10 1 8],[5 4.5 -2],.8,[0 1 1]) % generate a cube obstacle
SLTraj(T1,T2,2,0.5);
SLTraj(T2,T3,2,0.5);
SLTraj(T3,T4,2,0.5);
SLTraj(T4,T1,2,0.5);
FMTraj(q1,q2,2,0.5);
FMTraj(q2,q3,2,0.5);
SLTraj(T5,T6,2,0.5);
FMTraj(q4,q5,2,0.5);
FMTraj(q5,q6,2,0.5);
SLTraj(T7,T8,2,0.5);
SLTraj(T8,T9,2,0.5);
FMTraj(q7,q8,2,0.5);
FMTraj(q8,q9,2,0.5);
input('Task done. Press enter to continue \n');
close
%% Part 2
disp('Part 2');
%% inverse kinematic parallel robot
disp('A inverse kinematic example of parallel robot xc=260mm, yc=180mm,
alpha=30degree');
theta1 = ParallelIKPlot(260,180,30)
input('For better display please magnify the figure. Press enter to continue
\n');
close

%% Workspace space of parallel robot
disp('A workspace example of parallel robot alpha=-10degree');
ParallelWorkspace(-10)
input('For better display please magnify the figure. Press enter to finish \n');
disp('All done');
close
```