Robot Design and Implementation

Design of a 2dof Parallel Robot (5-Bar Linkage)

Candidate No. 181395

Word Count: 2654

Individual Contributions:

Kinematic Design: 181395

CAD Design: 168291

Trajectory Tracking: 181395

Abstract

The purpose of this project was to design and simulate a 5-bar parallel robot with 2 degrees of freedom. This involved the forwards and inverse kinematic design of the robot as well as a computer aided design model, made on Solidworks. Using the kinematic designs, the robot's movement was simulated on MATLAB and trajectories were plotted for the robot to move a certain way. This was a group project between two members where my partner did the CAD models and I did the forwards and inverse kinematics as well as the MATLAB trajectory tracking.

Table of Contents

1 Introduction **4**
2 Kinematic Design **5**
2.1 Forward Kinematics 5
2.2 Inverse Kinematics 8
3 CAD Model **9**
4 Trajectory Tracking **10**
5 Discussion **13**
6 Conclusion **13**
7 References **14**
8 Appendix **15**

1 Introduction

This project will show the processes of designing a robot, including the mathematical designs, the CAD models and the code used to programme the robot. The robot in question, is a 5-bar parallel robot with 2 degrees of freedom. Parallel robots are popular in industry as they have many benefits such as their flexibility, strength, and high loading capacity. However, parallel robots often have smaller workspaces as well as more singularities than serial robots. Like most robots, the performance will depend on its application.

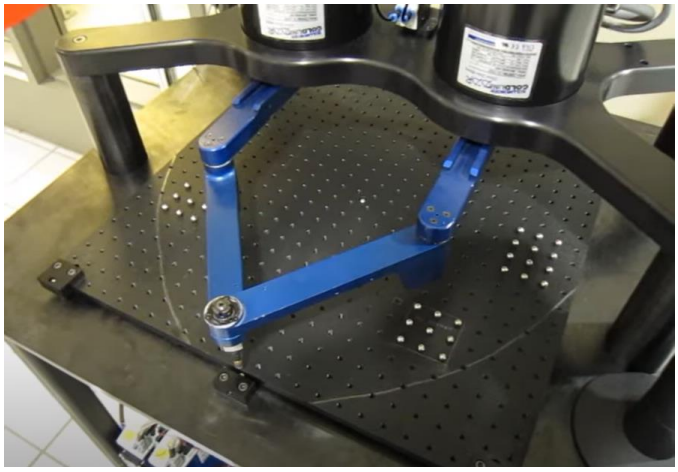


Figure 1: DexTAR 5-Bar Parallel Robot Video Screenshot [1]

The robot designed for this report was structured symmetrically so that the corresponding links on either side were equal as seen in figure 2. L_1 was used for both lower links on the left and right sides and L_2 for upper links on both sides. Two active joints at points O and B represent the joints controlled by motors while the other three joints are passive, the movement of the robot is controlled entirely by changing angles θ_1 and θ_4, i.e. actuating the two motors. Other specifications included how the robot must be mounted on a horizontal platform, it must be able to house and manoeuvre a tool of our choice and the workspace of the end-effector should cover a 10x10 cm rectangle.

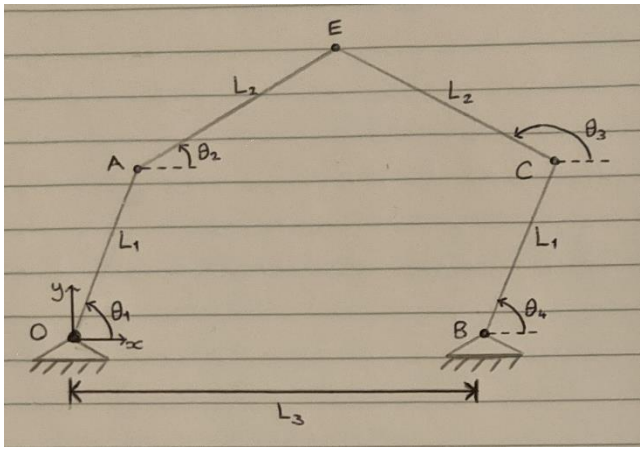


Figure 2: Diagram of 5-bar Parallel Robot

2 Kinematic Design

Kinematics is a description of the robot's motion based on the position and orientation of the robot. For forward kinematics, the position (co-ordinates) of the end effector is calculated based on the links of the robot as well as the angles at each joint. Figure 2 shows us that there are two routes to end effector point E from the origin, through points O to A to E or through points O to B to C to E.

$$\overrightarrow{OE} = \overrightarrow{OA} + \overrightarrow{AE} = \overrightarrow{OB} + \overrightarrow{BC} + \overrightarrow{CE} \qquad (1)$$

Based on trigonometric functions, we can find the x and y co-ordinates of the end-effector from the link lengths and angles at each joint.

$$x = L_1 cos(\theta_1) + L_2 cos(\theta_2) \qquad (2)$$

or

$$x = L_3 + L_1 cos(\theta_4) + L_2 cos(\theta_3) \qquad (3)$$

$$y = L_1 sin(\theta_1) + L_2 sin(\theta_2) \qquad (4)$$

or

$$y = L_1 sin(\theta_4) + L_2 sin(\theta_3) \qquad (5)$$

From equations (2) to (5) we can see that two solutions exist for the x and y co-ordinates of the end-effector position. This is in-line with the relationship seen in equation (1) in that there are two routes from the origin to the end-effector.

2.1 Forward Kinematics

There are multiple methods of obtaining the forward kinematic equations of the robot and there is no optimal solution but many different solutions. One approach is to obtain the transform operator of the robot from point O to end effector E. This is the translations or rotations required at each point to reach the end effector from the origin. To achieve this convention, a configuration space approach should be taken so we use angles q_1, q_4 instead of θ_1 and θ_4, as seen in figure 3.

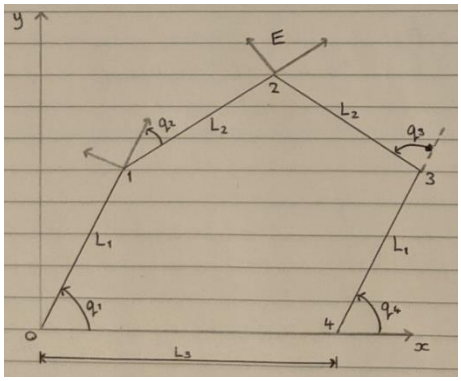


Figure 3: Diagram of 5-bar parallel robot, showing C-Space approach

$$ {}^{0}_{2}A = {}^{0}_{1}A\,{}^{1}_{2}A = \begin{bmatrix} cos(q_1+q_2) & -sin(q_1+q_2) & 0 & L_1cos(q_1) + L_2cos(q_1+q_2) \\ sin(q_1+q_2) & cos(q_1+q_2) & 0 & L_1sin(q_1) + L_2sin(q_1+q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6) $$

$$ {}^{0}_{2}A = {}^{0}_{4}A\,{}^{4}_{3}A\,{}^{3}_{2}A = \begin{bmatrix} cos(q_4+q_3) & -sin(q_4+q_3) & 0 & L_3 + L_1cos(q_4) + L_2cos(q_4+q_3) \\ sin(q_4+q_3) & cos(q_4+q_3) & 0 & L_1sin(q_4) + L_2sin(q_4+q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7) $$

Another way to find the transformation matrix from point O to end-effector E is through Denavit-Hartenberg Parameters. We define 4 parameters a, α, d and θ for each transformation:

Link	α	a	d	θ
1	L_1	0	0	q_1
2	L_2	0	0	q_2

Link	α	a	d	θ
4	L_3	0	0	0
3	L_1	0	0	q_4
2	L_2	0	0	q_3

Our robot design is 2D, therefore there is no movement of the z-axis and the z-axis for each joint are parallel to each other. This means there is no link twist or offset values (α and d). Using the tables with parameter values, we can substitute these parameters into a matrix which describes the transformation for each link, ${}^{i-1}_{i}T$.

$$ {}^{i-1}_{i}T = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_ic_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_is_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}_{1}T = \begin{bmatrix} cos(q_1) & -sin(q_1) & 0 & L_1cos(q_1) \\ sin(q_1) & cos(q_1) & 0 & L_1sin(q_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{1}_{2}T = \begin{bmatrix} cos(q_2) & -sin(q_2) & 0 & L_2cos(q_2) \\ sin(q_2) & cos(q_2) & 0 & L_2sin(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}_{2}T = {}^{0}_{1}T\,{}^{1}_{2}T = \begin{bmatrix} cos(q_1+q_2) & -sin(q_1+q_2) & 0 & L_1cos(q_1) + L_2cos(q_1+q_2) \\ sin(q_1+q_2) & cos(q_1+q_2) & 0 & L_1sin(q_1) + L_2sin(q_1+q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6) $$

$$ {}^{0}_{4}T = \begin{bmatrix} 1 & 0 & 0 & L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^{4}_{3}\mathrm{T} = \begin{bmatrix} cos(q_4) & -sin(q_4) & 0 & L_1 cos(q_4) \\ sin(q_4) & cos(q_4) & 0 & L_1 sin(q_4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{3}_{2}T = \begin{bmatrix} cos(q_3) & -sin(q_3) & 0 & L_2 cos(q_3) \\ sin(q_3) & cos(q_3) & 0 & L_2 sin(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{0}_{2}T = {}^{0}_{4}T\ {}^{4}_{3}T\ {}^{3}_{2}T = \begin{bmatrix} cos(q_4 + q_3) & -sin(q_4 + q_3) & 0 & L_3 + L_1 cos(q_4) + L_2 cos(q_4 + q_3) \\ sin(q_4 + q_3) & cos(q_4 + q_3) & 0 & L_1 sin(q_4) + L_2 sin(q_4 + q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7) $$

As you can observe, we ended with equal solutions for the transformation matrices using the Denavit-Hartenberg method and the normal transform operator method. We can use these matrices to derive the x and y co-ordinates for the end effector, however these equations will not actuate the robot the way we require for this particular case, since θ_1 and θ_4 are both active joints but θ_2 and θ_3 are not. The equations previously calculated use all four angles but the solution we require uses just θ_1 and θ_4 to actuate the robot since these are the values related to the motors that actuate the robot.

By eliminating θ_1 and θ_4 from equations (2) to (5), we algebraically obtain equations (8) and (9) which relate the end effector position to the angles θ_1 and θ_4 and lengths L_1, L_2 and L_3. [2]

$$ x = L_3 + L_1 cos(\theta_4) + L_2 cos\left[2arctan\left(\frac{-B \pm \sqrt{A^2 + B^2 - C^2}}{C - A}\right)\right] \quad (8) $$

$$ y = L_1 sin(\theta_4) + L_2 sin\left[2arctan\left(\frac{-B \pm \sqrt{A^2 + B^2 - C^2}}{C - A}\right)\right] \quad (9) $$

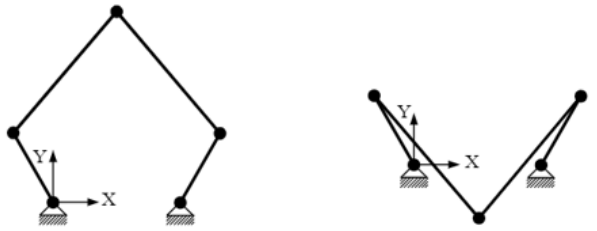
Where:

$$ A = 2L_2(L_3 + L_1 cos(\theta_4 - \theta_1)) $$

$$ B = 2L_1 L_2(sin(\theta_4) - sin(\theta_1)) $$

$$ C = L_3^2 + 2L_1^2 + 2L_3 L_1 cos(\theta_4) - 2L_3 L_1 cos(\theta_1) - 2L_1^2 cos(\theta_4 - \theta_1) $$

From these equations it is clear to see that two sets of solutions for the x and y co-ordinates are calculated due to the "±" symbol. This is due to there being two configurations for the robot, either facing up or down, shown in figure 4. The robot is facing up when the subtract option is used in either equation, whereas the robot is facing down when addition is used.



"Elbow-Up" "Elbow-Down"

Figure 4: Diagram of Elbow up and down configurations of robot [2]

2.2 Inverse Kinematics

Since the forward kinematic equations have been calculated, the inverse kinematics can be derived. We calculated θ_1 and θ_4 based on co-ordinates of end effector x and y.

$$\left[\theta_1 = 2arctan\left(\frac{-B\pm\sqrt{A^2+B^2-C^2}}{C-A}\right)\right] \quad (10)$$

$$\left[\theta_4 = 2arctan\left(\frac{-B\pm\sqrt{D^2+B^2-E^2}}{E-D}\right)\right] \quad (11)$$

Where:

$$A = -2L_1x$$

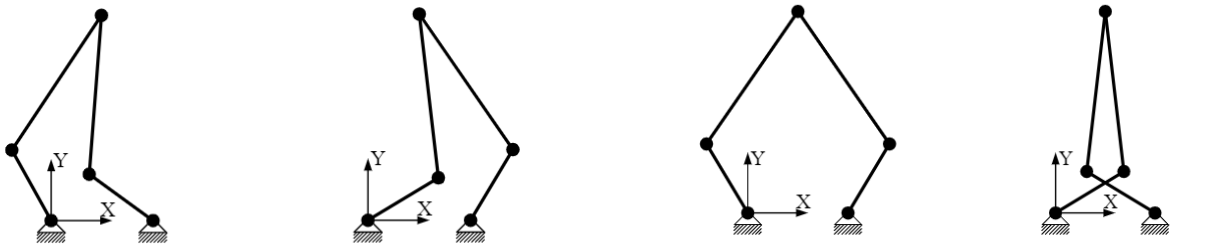
$$B = -2L_1y$$

$$C = L_1^2 - L_2^2 + x^2 + y^2$$

$$D = -2L_1(-x + L_3)$$

$$E = L_3^2 + L_1^2 - L_2^2 + x^2 + y^2 - 2L_3x$$

Much like the forwards kinematics equations, there are again multiple sets of solutions calculated due to the "±" symbol. This is due to four different types of configuration for the robot at a desired end-effector position shown in figure 5.



a) '+' – '+' combination b) '-' – '-' combination c) '+' – '-' combination d) '-' – '+' combination

Figure 5: Diagrams of four different configurations from the kinematics solution [2]

3 CAD Model

A Computer Aided Design model was made through SOLIDWORKS, to observe the behaviours and capabilities of the robot. This CAD model gave us an understanding of the constraints in the workspace of the robot as well as how the motors at points O and B effected the other points of the robot. The assembled robot can be seen in figures 6 and 7, and all individual part drawings can be found in appendix a.

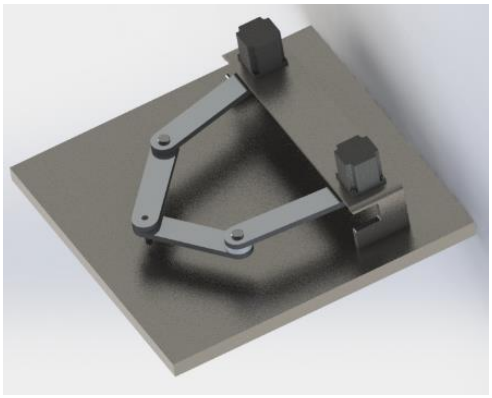


Figure 6: Rendered View of CAD Model

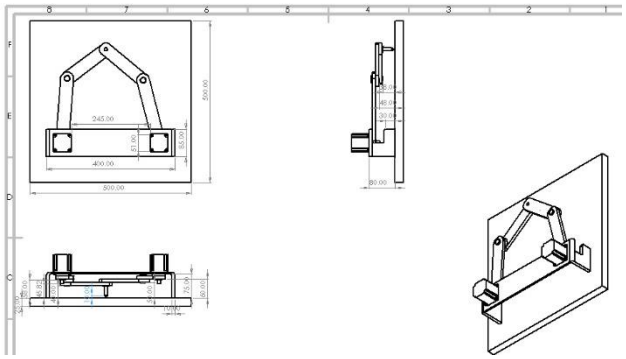


Figure 7: Assembly Drawing of Robot

The model consists of many individual components welded and assembled together to function as it would in the real world. Each component is made from particular materials that suit the function of that part. The base and frame of the robot model is made from steel as the material is extremely sturdy and can withstand high amounts of loads. The links are each made from aluminium as this material is relatively lightweight compared to other metals, yet still strong. The robot arms would be able to undergo a large amount of cycles and repetitions without any deformation. Stepper motors at point O and B are made from plastic, therefore cheap and easy to replace. The motors are based on designs from a previous university module "Mechanics of Mechanisms and Robots". [3]

In order to design the robot model, we had to assign the lengths to each link based on the specifications of the robot. The robot workspace had to cover a 10x10 cm rectangle, so we drew a rectangle of the same dimensions in the software plane and observed how long the links needed to be under singularity, when there was no more movement in a certain direction (end effector could not move any further), to reach the corners of the rectangle. We rounded these measurements up to allow some room for error and assigned the following lengths: L_1 = 15 cm, L_2 = 20 cm, L_3 = 30 cm.

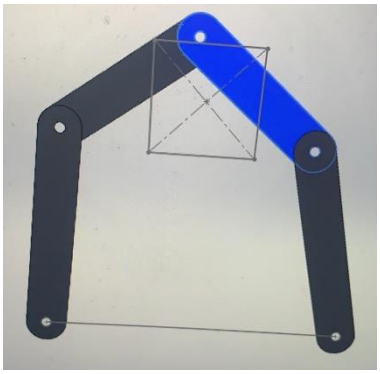


Figure 8: Screenshot of CAD model showing desired 10x10cm workspace.

It was at this stage that we chose the tool located at the end-effector, a pen tool that could be used to write or draw figures. This has many applications and is a simple yet effective mechanism and it works particularly well for this project with the trajectory tracking tasks. The pen tool can be seen in figure 9 below.

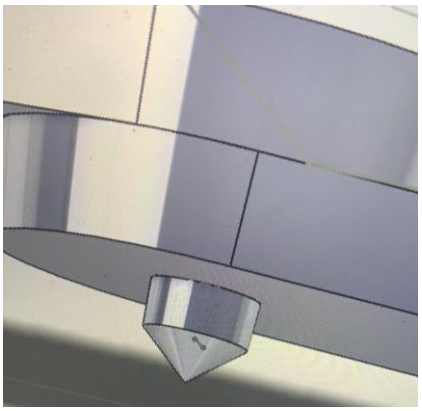


Figure 9: Close-up of end-effector with pen tool

When designing the CAD model, bumpers were placed on the base of the robot to stop the robot at certain points and limit the workspace of the end effector to the space in front of the robot as seen in figure 6. Although this makes the workspace smaller, the rest of the workspace is not needed for our pen tool application.

4 Trajectory Tracking

Since the computer aided design had been modelled and we could observe how the robot looked and behaved, a simulation that plots the robot's trajectory and movement could be built through MATLAB based on the kinematic designs previously calculated. The simulation will be achieved using MATLAB scripts, filled with code used to plot the robot on a graph. MATLAB Scripts can be found from appendix B to E in this report.

Firstly, functions fwdKin.m and inverseKin.m were made using the forward and inverse kinematics equations previously obtained. The forward kinematics function when called, takes in lengths L_1, L_2 and L_3, and angles q_1 and q_2, and output the x and y co-ordinates of the end effector. The function calculates and outputs two sets of co-ordinates but the answer of the function is set as the elbow-up configuration co-ordinates as we are only modelling the robot in that configuration and will use the answer from this function in a later test. The inverse kinematics function, as expected, takes in lengths L_1, L_2 and L_3, and the end-effector co-ordinates x and y and outputs the angles q_1 and q_2 required for the end effector to be located at those x and y co-ordinates. For this function, only the angles corresponding to the elbow-up configuration, are calculated as the other configurations are undesired in this scenario.

```
>> fwdKin(15,20,30,45,45)

x_elbowDown =

   25.6066


y_elbowDown =

   -2.6222


x_elbowUp =

   25.6066


y_elbowUp =

   23.8354
```

Figure 10: MATLAB Command window showing fwdKin.m function

```
>> inverseKin(15,20,30,25.6066,23.8354)

q4 =

   45.0001


ans =

   44.9984
```

Figure 11: MATLAB Command window showing inverseKin.m function

The next function to be made was the plotUpdate.m function. This function plots the robot on a graph using link lengths L_1 and L_3, the two angles as well as a previous trajectory value. This previous trajectory is used in the trajectory tracking scripts later on and it plots each configuration of the robot to see the movement it has made. Calling this function in the command window and entering the required constants produces a graph showing the robot's position.

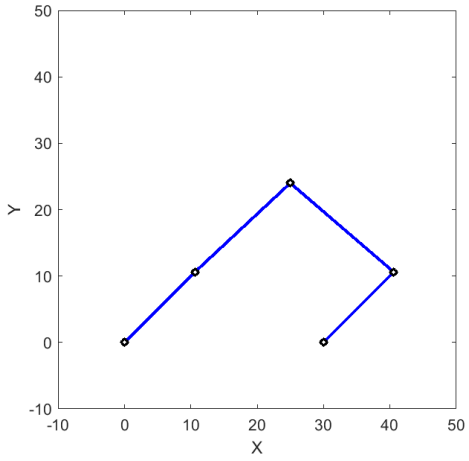


Figure 12: Graph of robot's position from function plotUpdate.m

Three functions for finding the co-ordinates and angles required, and for plotting the graph had been made so now an algorithm for simulating the movement of the robot could be made. Three scripts were made called plotCircle.m, plotSquare.m, and plotTriangle.m which moved the end-effector in a circle, square or triangle path respectively.

The circle plotting script states the three link lengths as well as the location and radius of the circle. The co-ordinates for each point in the circle are calculated and using the inverse kinematics function, the angles required to reach each point in the circle are calculated. These angles are placed into the forwards kinematics function and the end-effector position is found. This repeats in a loop for each point in the circle. The plot update function is then called to plot the robot configuration at every point in the circle as well as the previous trajectory which allows a circle to be plotted.

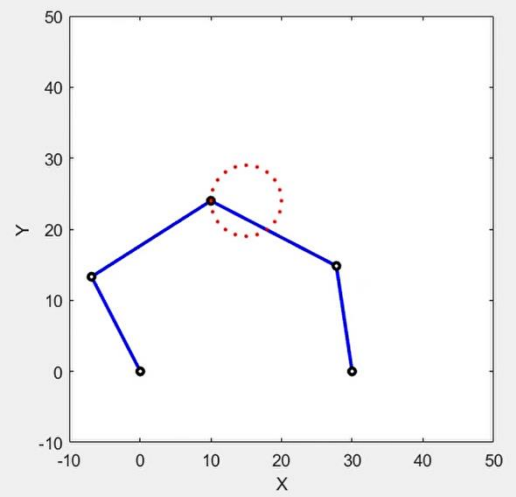


Figure 13: Graph showing circle plot from plotCircle.m script

The square plotting script followed a similar structure with the inverse kinematics function being used to calculate the angles required for the end-effector to reach certain points and the forward kinematics function calculating the end-effector position to be used in the plot update function to plot the graph. The MATLAB function linspace() was used to uniformly distribute points between two co-ordinates and these points are where the end-effector position will reach. There are 4 instances of this, creating 4 lines making a 10x10cm square. The triangle plotting script mimics the square plotting script except the linspace() values were at different points making the end-effector follow a triangle instead of a square.

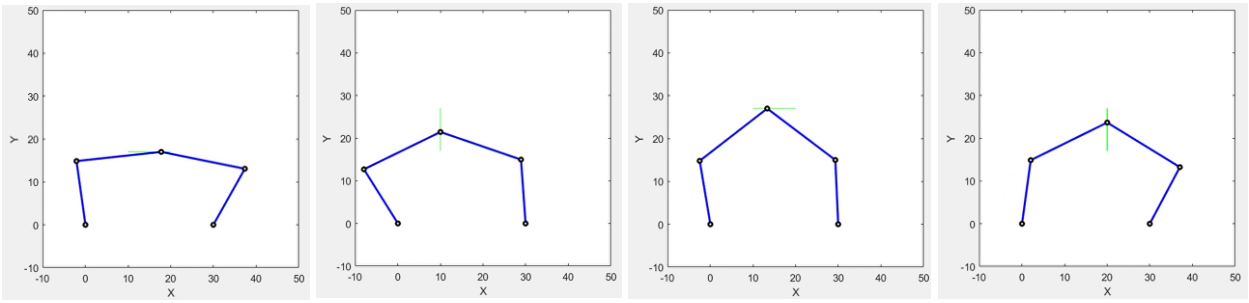


Figure 14: Graphs showing robot following 10x10cm square trajectory

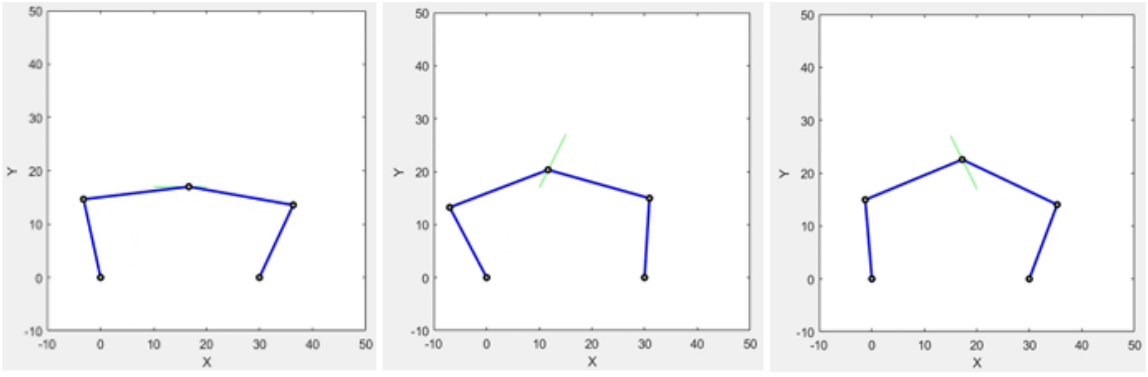


Figure 15: Graphs showing robot following triangle trajectory

5 Discussion

A combination of the three plotting scripts could hypothetically make the end-effector follow any pre-designated path within the workspace, therefore an algorithm could be made that programmes the robot to write letters and numbers rather than shapes. This would be especially helpful to our application with the pen tool as an end-effector.

The robot designed in this project has many potential uses with the most obvious use being for text writing applications. This could be used for automation purposes where the robot would perform faster than humans at filling out forms for example. Perhaps the most useful task the robot could perform would be to help people with disability. A program could be created based on the kinematics and code used in MATLAB, that could write any letter based on an input given by the user, much like an automated typewriter. This would be useful for both children and adults with disability or who are going through rehabilitation.

The MATLAB code used in this project meets the specifications tasked in designing the robot, however, if the group had more time to complete the project, a user interface could have been completed, asking the user to type any letter or number and the robot would follow the pattern of that input given. The previous trajectory variable in the plotUpdate.m function would trace the previous trajectory as though it was writing on paper. Furthermore, a script that plotted all possible co-ordinates of the end-effector was attempted that would show the entire workspace of the end-effector. However, this is not crucial as we made the decision to place bumpers on the robot base to stop the end-effector going to the opposite side of the end-effector as this was not a desired function for our robot application. Other features like computer vision allowing the robot to track an object from a camera could have been implemented to improve the scope and expand the possibilities of our robot design.

6 Conclusion

Over the course of this project, various mathematical models were designed representing the robot kinematics. An algebraic model was chosen that matched the specifications and constraints of the robot by relating the end-effector co-ordinates with the two input angles to be controlled by motors at joints O and B. This model was proven to be correct when simulated in MATLAB as we were able to program the robot to follow particular shapes and paths.

In addition, a computer aided design model was created showcasing the robot's relative size and movement possibilities. Materials were assigned to each individual part considering practical applications if the robot was to be used in real life. These designs allowed us to think about potential applications for the robot, having many useful real-world executions.

7 References

[1] CoRoETS, "*DexTAR:: The fastest five-bar parallel robot*", [Online] Available from: https://www.youtube.com/watch?v=dnixuCu49o4 [Accessed 27/04/21]

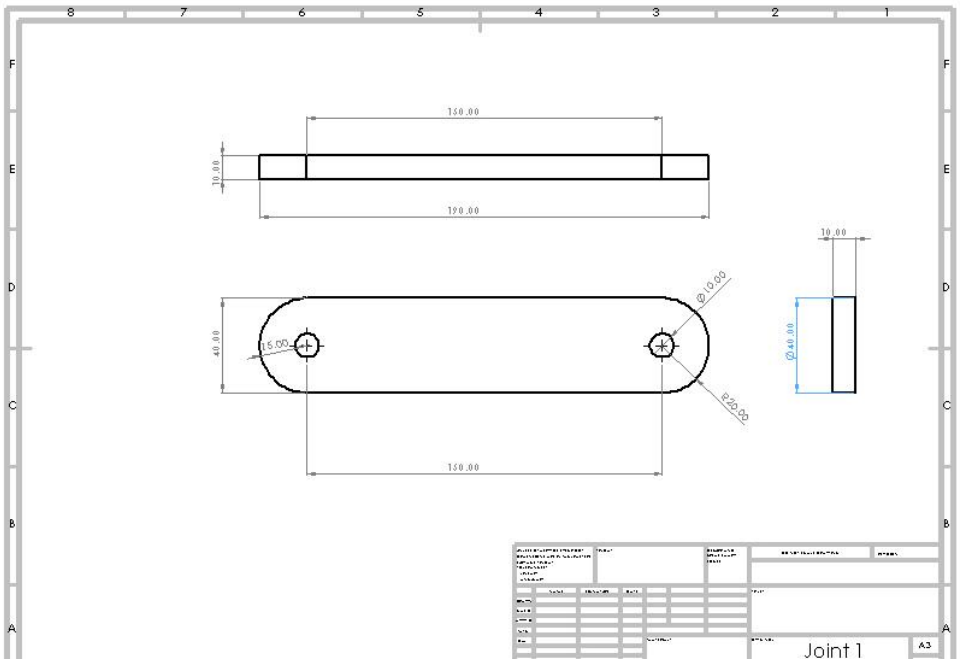
[2] Tuong Manh Hoang, Trung Tin Vuong, Bang Cong Pham, "*Study and Development of Parallel Robots Based On 5-Bar Linkage*", October 2015, National Conference on Machines and Mechanisms, 2015, pp 3 – 4.

[3] Yanan Li, *"Jaw Stepper.SLDPRT"*, Mechanics of Mechanisms and Robots SOLIDWORKS file

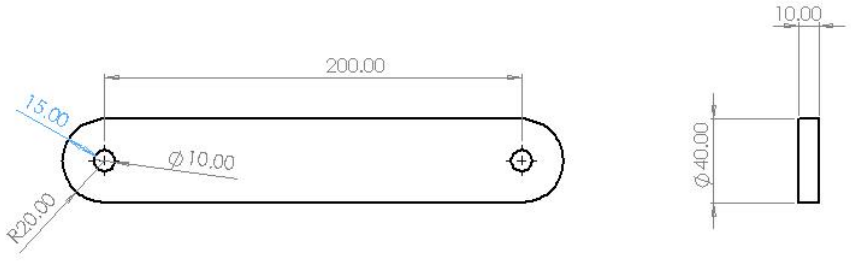
8 Appendices

Appendix A: CAD Assembly Drawings of Individual Robot Components

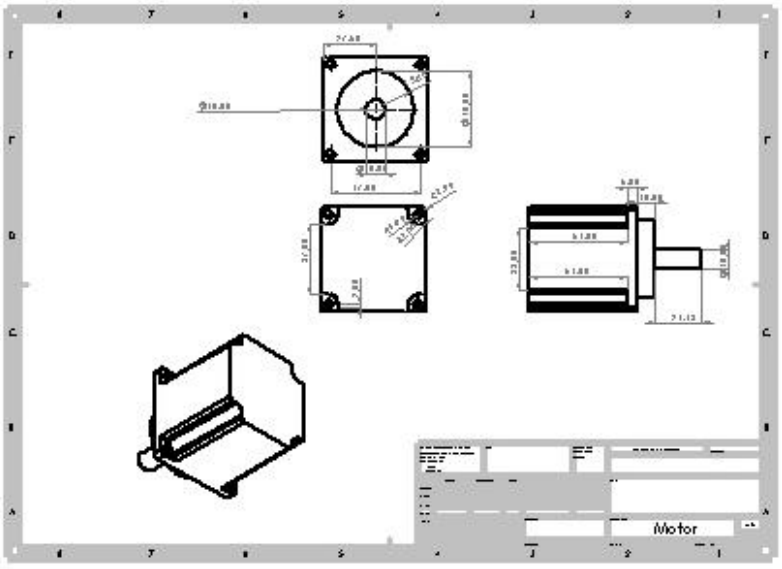
Link 1 (L1):



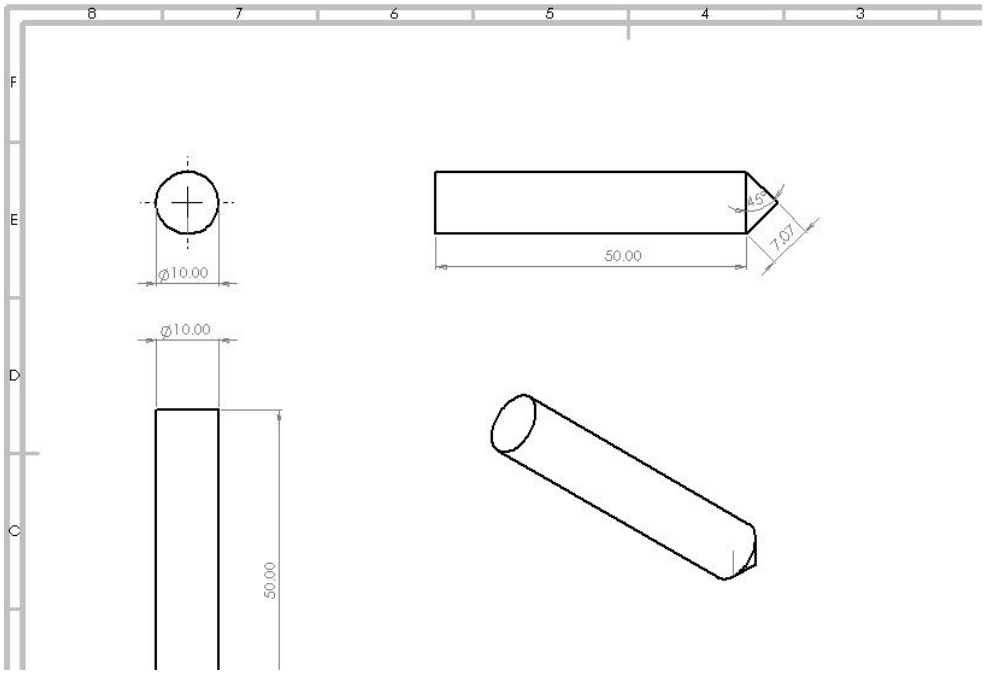
Link 2 (L2):



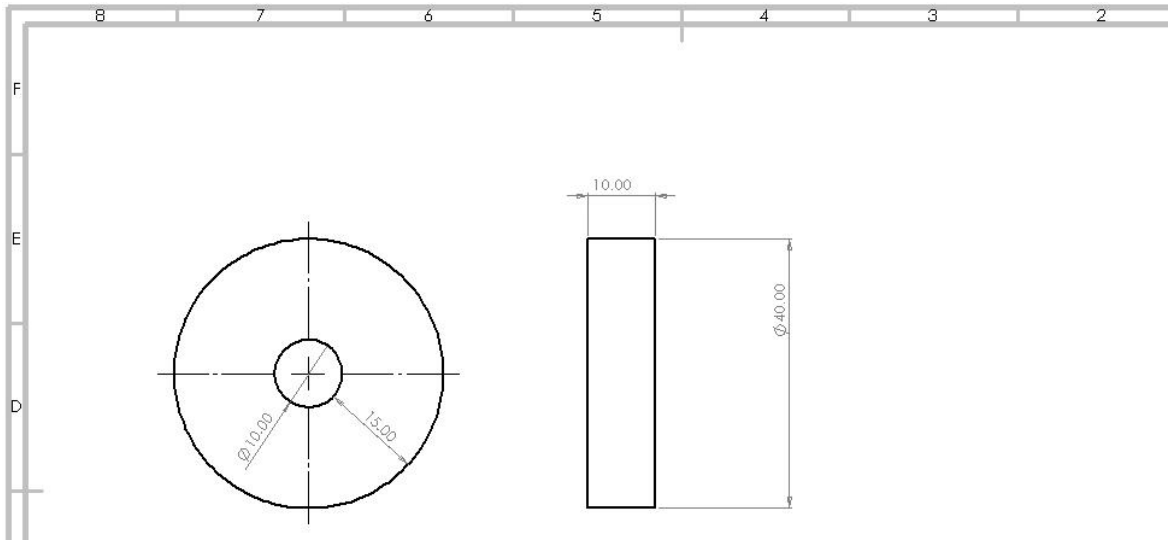
Motor:



Pen Tool:



Joint:



Appendix B: MATLAB Script for fwdKin.m function

```
% Forward Kinematics - Calculating position of end effector
based on
% angles q1 and q4, and lengths of joints L1, L2 and L3.

function [pos1,pos2] = fwdKin(L1,L2,L3,q1,q4)

% A, B and C are variables used to simplify equation for
co-ordinates
A = 2*L2*(L3+L1*(cosd(q4)-cosd(q1)));
B = 2*L1*L2*(sind(q4)-sind(q1));
C = L3^2+2*L1^2+2*L3*L1*cosd(q4)-2*L3*L1*cosd(q1)-
2*L1^2*cosd(q4-q1);


% Two sets of co-ordinates based on elbow up or elbow down
% co-ordinates when the robot is "elbow down"
x_elbowDown = L3+L1*cosd(q4)+L2*cosd(2*atand((-
B+sqrt(A^2+B^2-C^2))/(C-A)))
y_elbowDown = L1*sind(q4)+L2*sind(2*atand((-B+sqrt(A^2+B^2-
C^2))/(C-A)))

% co-ordinates when the robot is "facing up"
x_elbowUp = L3+L1*cosd(q4)+L2*cosd(2*atand((-B-
sqrt(A^2+B^2-C^2))/(C-A)))
y_elbowUp = L1*sind(q4)+L2*sind(2*atand((-B-sqrt(A^2+B^2-
C^2))/(C-A)))

pos1 = x_elbowUp;
pos2 = y_elbowUp;
```

Appendix C: MATLAB Script for inverseKin.m function

```
function [q1,q4] = inverseKin(L1,L2,L3,x,y)

A = -2*L1*x;
B = -2*L1*y;
C = L1^2-L2^2+x^2+y^2;
D = 2*L1*(-x+L3);
E = L3^2+L1^2-L2^2+x^2+y^2-2*L3*x;

q1 = 2*atand((-B+sqrt(A^2+B^2-C^2))/(C-A));
q4 = 2*atand((-B-sqrt(D^2+B^2-E^2))/(E-D))
```

Appendix D: MATLAB Script for plotUpdate.m function

```
% Plot Update - Plotting robot on chart

function plotUpdate(x_pos,y_pos,prev_traj,L1,L3,q1,q4)

% x and y co-ordinates of L1 joints (a is left arm, b is
right arm)
L1a_pos = [L1*cosd(q1) L1*sind(q1)];
L1b_pos = [L3+L1*cosd(q4) L1*sind(q4)];

% x and y co-ordinates of end effector
X = [0, L1a_pos(1), x_pos, L1b_pos(1), L3];
Y = [0, L1a_pos(2), y_pos, L1b_pos(2), 0];


% Plot robot on XY graph
plot(X,Y,'LineWidth',2,'Color','Blue','Marker','o','MarkerE
dgeColor','black','MarkerFaceColor','White','MarkerSize',4)
;
xlabel('X');
ylabel('Y');
hold on;

axis equal;
```

```
plot(prev_traj(:,1),prev_traj(:,2),'LineStyle','None','Line
Width',1,'Color','Red','Marker','.');
xlim([-10,50]);
ylim([-10,50]);

end
```

Appendix C: MATLAB Script for plotCircle.m

```
% Plotting a Circle

% Circle with radius 10cm and centre co-ordinates of
(15,24)
r = 10;
xCentre = 15;
yCentre = 24;

% Lengths of links
L1=15;
L2=20;
L3=30;

traj_q=[]; %Variable for storing joint angles
traj_xy=[];   %Variable for storing end-effector co-
ordinates


while 1
    for theta_circle = 0:pi/10:2*pi    % loop repeats from
0 to 2 pi in increments of 1/10 pi

        % calculate xy coordinates for every point on the
circle
        x_circle = xCentre + r*cos(theta_circle);
        y_circle = yCentre + r*sin(theta_circle);


        % calc joint angles q1, q2 with inverse kinematics
        [q1,q4] = inverseKin(L1, L2, L3, x_circle,
y_circle);

        % calc XY co-ordinates with forward kinematics
        [pos1,pos2]=fwdKin(L1,L2,L3,q1,q4);
```

```
        % store joint angles q1,q2
        q = [q1,q4];
        traj_q = [traj_q ; q];

        % Store position of end effector
        pos = [pos1, pos2];
        traj_xy = [traj_xy;pos];

        clf;

        %plot the desired trajectory
        plot(x_circle,y_circle,'green');
        hold on;

        %plot the current configuration and completed
trajectory
        plotUpdate(pos1,pos2,traj_xy,L1,L3,q1,q4);
        pause(0.05);

        hold on;

    end
end
```

Appendix E: MATLAB Script for plotSquare.m

```
% Plotting an 10x10cm square

% Lengths of links
L1=15;
L2=20;
L3=30;

traj_q = [];    %Variable for storing joint angles
traj_xy = [];   %Variable for storing end-effector co-
ordinates

while 1

    % Draw line from x = 10 to x = 20 with y = 17 ( 1st
line)
    x = linspace(10,20,10);
    y = linspace(27,27,10);
```

```
    for i = 1:1:10   % start loop at 1, finish at 10 with
increments of 1

        xy=[x(i) y(i)];

        % calc joint angles q1, q2 with inverse kinematics
        [q1,q4]=inverseKin(L1, L2, L3, xy(1), xy(2));

        % calc XY co-ordinates with forward kinematics
        [pos1,pos2]=fwdKin(L1,L2,L3,q1,q4);

        % store joint angles q1,q2
        q = [q1,q4];
        traj_q=[traj_q; q];

        % Store position of end effector
        traj_xy=[traj_xy; pos2];

        clf;

        % Plot the desired trajectory
        plot(x,y,'green');
        hold on;

        % Plot the current configuration and completed
trajectory
        plotUpdate(pos1,pos2,traj_xy,L1,L3,q1,q4);
        pause(0.05); % Pause and see the figure

    end

    % draw line from y = 27 to y = 17 with x = 20 (2nd
line)
    x=linspace(20,20,10);
    y=linspace(17,27,10);

    for i=10:-1:1

      xy=[x(i) y(i)];


        % calc joint angles q1, q2 with inverse kinematics
        [q1,q4]=inverseKin(L1, L2, L3, xy(1), xy(2));
```

```
        % calc XY co-ordinates with forward kinematics
        [pos1,pos2]=fwdKin(L1,L2,L3,q1,q4);

        % store joint angles q1,q2
        q = [q1,q4];
        traj_q=[traj_q; q];

        % Store position of end effector
        traj_xy=[traj_xy; pos2];

        clf;

        %plot the desired trajectory
        plot(x,y,'green');
        hold on;

        %plot the current configuration and completed
trajectory
        plotUpdate(pos1,pos2,traj_xy,L1,L3,q1,q4);
        pause(0.05); % Pause and see the figure
    end

    % draw line from x = 20 to x = 10 with y = 17 (3rd
line)
    x=linspace(10,20,10);
    y=linspace(17,17,10);

    for i=10:-1:1

        xy=[x(i) y(i)];

        % calc joint angles q1, q2 with inverse kinematics
        [q1,q4]=inverseKin(L1, L2, L3, xy(1), xy(2));

        % calc XY co-ordinates with forward kinematics
        [pos1,pos2]=fwdKin(L1,L2,L3,q1,q4);

        % store joint angles q1,q2
        q = [q1,q4];
        traj_q=[traj_q; q];

        % Store position of end effector
        traj_xy=[traj_xy; pos2];

        clf;
```

```
        %plot the desired trajectory
        plot(x,y,'green');
        hold on;

        %plot the current configuration and completed
trajectory
        plotUpdate(pos1,pos2,traj_xy,L1,L3,q1,q4);
        pause(0.05); % Pause and see the figure
    end

    % draw line from y = 17 to y = 27 with x = 10 (4th line)
    x=linspace(10,10,10);
    y=linspace(17,27,10);

    for i= 1:1:10

       xy=[x(i) y(i)];

        % calc joint angles q1, q2 with inverse kinematics
        [q1,q4]=inverseKin(L1, L2, L3, xy(1), xy(2));

        % calc XY co-ordinates with forward kinematics
        [pos1,pos2]=fwdKin(L1,L2,L3,q1,q4);

        % store joint angles q1,q2
        q = [q1,q4];
        traj_q=[traj_q; q];

        % Store position of end effector
        traj_xy=[traj_xy; pos2];

        clf;

        %plot the desired trajectory
        plot(x,y,'green');
        hold on;

        %plot the current configuration and completed
trajectory
        plotUpdate(pos1,pos2,traj_xy,L1,L3,q1,q4);
        pause(0.05); % Pause and see the figure
    end
end
```

Appendix F: MATLAB Script for plotTraingle.m

```
% Plotting a Triangle

% Lengths of links
L1=15;
L2=20;
L3=30;

traj_q = [];     %Variable for storing joint angles
traj_xy = [];    %Variable for storing end-effector co-
ordinates

while 1

    % Draw line from x = 10 to x = 20 with y = 17 (1st
line)
    x = linspace(10,15,10);
    y = linspace(17,27,10);

    for i = 1:1:10   % start loop at 1, finish at 10 with
increments of 1

        xy=[x(i) y(i)];

        % calc joint angles q1, q2 with inverse kinematics
        [q1,q4]=inverseKin(L1, L2, L3, xy(1), xy(2));

        % calc XY co-ordinates with forward kinematics
        [pos1,pos2]=fwdKin(L1,L2,L3,q1,q4);

        % store joint angles q1,q2
        q = [q1,q4];
        traj_q=[traj_q; q];

        % Store position of end effector
        traj_xy=[traj_xy; pos2];

        clf;

        % Plot the desired trajectory
        plot(x,y,'green');
        hold on;
```

```
        % Plot the current configuration and completed
trajectory
        plotUpdate(pos1,pos2,traj_xy,L1,L3,q1,q4);
        pause(0.05); % Pause and see the figure

    end

    % draw line from y = 27 to y = 17 with x = 20 (2nd
line)
    x=linspace(15,20,10);
    y=linspace(27,17,10);

    for i=1:1:10

      xy=[x(i) y(i)];


        % calc joint angles q1, q2 with inverse kinematics
        [q1,q4]=inverseKin(L1, L2, L3, xy(1), xy(2));

        % calc XY co-ordinates with forward kinematics
        [pos1,pos2]=fwdKin(L1,L2,L3,q1,q4);

        % store joint angles q1,q2
        q = [q1,q4];
        traj_q=[traj_q; q];

        % Store position of end effector
        traj_xy=[traj_xy; pos2];

        clf;

        %plot the desired trajectory
        plot(x,y,'green');
        hold on;

        %plot the current configuration and completed
trajectory
        plotUpdate(pos1,pos2,traj_xy,L1,L3,q1,q4);
        pause(0.05); % Pause and see the figure
    end

    % draw line from x = 20 to x = 10 with y = 17 (3rd
line)
    x=linspace(10,20,10);
```

```
    y=linspace(17,17,10);

    for i=10:-1:1

        xy=[x(i) y(i)];

        % calc joint angles q1, q2 with inverse kinematics
        [q1,q4]=inverseKin(L1, L2, L3, xy(1), xy(2));

        % calc XY co-ordinates with forward kinematics
        [pos1,pos2]=fwdKin(L1,L2,L3,q1,q4);

        % store joint angles q1,q2
        q = [q1,q4];
        traj_q=[traj_q; q];

        % Store position of end effector
        traj_xy=[traj_xy; pos2];

        clf;

        %plot the desired trajectory
        plot(x,y,'green');
        hold on;

        %plot the current configuration and completed
trajectory
        plotUpdate(pos1,pos2,traj_xy,L1,L3,q1,q4);
        pause(0.05); % Pause and see the figure
    end
end
```