Kinematic Simulation of a 5 DOF Lynx Motion Robot Arm and a Parallel Planar Robot

Rowland T

(Dated: January 15, 2023)

Introduction

This report will document the process of simulating the kinematics of a 5 DOF lynx motion robot arm and a parallel planar robot. The software used for this simulation will be MATLAB as it was deemed to be the most appropriate due to the large amount of matrix multiplication involved in the kinematic calculations which MATLAB was designed for. The project is split into several sections; part 1.a will focus on deriving the correct representation of the robot arm by forming a DH table, the robot workspace and the forward and inverse kinematic models. Part 1.b will explore planning a task with the robot arm whereby the end effector moves between some predefined coordinates, implemented with different trajectories. A parallel planar robot will be modelled in Part 2 where the construction of the inverse kinematic representation will be simulated followed by a discussion about the workspace of a parallel planar robot.

CONTENTS

I. PART 1 - MODELLING THE LYNX MOTION ROBOT ARM

A. Part A

1. *Question 1 - Formulating a DH table*

Derive a DH representation of the forward kinematics for the Lynxmotion arm. Use MATLAB, lecture material and further reading. Include all your investigations and report this

Within robotics, kinematics is the study of the relationship between a robots geometry the robot in joint space, and the robot in cartesian space. Which can be written simply as: how joint angles will effect the robots position in 3D cartesian space.

The Denavit-Hartenberg (DH) tables are a method by which the kinematic relationships between the links of a robot can be defined. The table is constructed with a series of rows, representing the different joints, and columns that contain the parameters a_n, $alpha_n$, d_n and $theta_n$. These parameters model the transformation between coordinate frames throughout the robotic manipulator.

The lynxmotion robot arm under investigation is a 5 degree of freedom (DOF) robot meaning that it has five axes of motion that will have to be isolated to prevent any movement from the robot. The robot is constructed with 5 revolute joints in a $R \perp R \parallel R \parallel R \perp R$ configuration The DH table will therefore contain 5 rows, representing the links between the base and the end effector. Populating the DH table requires a simplified representation of the robot, including each joint and link from the base to the manipulator. A reference frame will have to be defined for each link and joint to establish their relationship between each other. Using this diagram, the table can be completed. The following rules can be used to complete the DH table. Z_n must lie along the axis of rotation, X_n must be normal to Z_n and $Z_n - 1$ and Y_n completed the right hand rule [2].

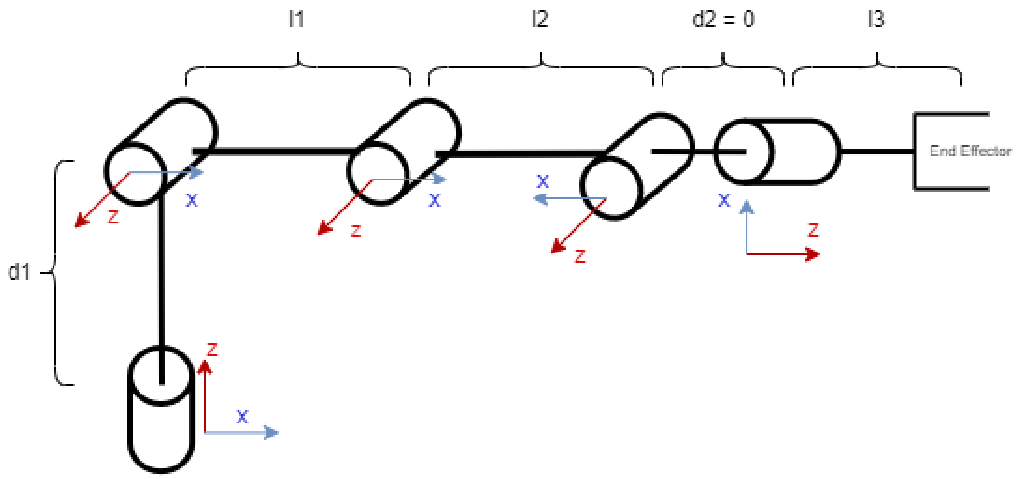


FIG. 1. The simplified drawing of the 5 DOF lynxmotion robot arm. The direction of Z_n and X_n can be seen from the red and blue arrows respectively. For the simplicity of the diagram Y_n has been omitted as it isn't necessary for the DH table.

This report will focus on the Distal form of the DH table as shown below in Table I.

Link	a_n	α_n	d_n	θ_n
1	0	90	d	$q1$
2	$l1$	0	0	$q2$
3	$l2$	0	0	$q3$
4	0	90	0	$q4$
5	0	0	$l3$	$q5$

TABLE I. A Table showing the DH parameters defined from Fig.3

A homogeneous transformation matrix can be used to determine the position and orientation of each link in the robot arm. A homogeneous transformation is a 4x4 matrix that combines the rotation matrix and translation matrix to describe how the a links frame with respect to another frame. By using transformation matrices for each link a description of the whole arm can be established by multiplying successive homogeneous transformation matrices together. You can see the individual transformation matrices below of the lynxmotion robot arm when combined with the information from the Distal table, table I.

```
%Indiviual transpose matrices from the distal table
T01 = [cos(q1) 0 sin(q1) 0; sin(q1) 0 cos(q1) 0; 0 1 0 d; 0 0 0 1];
T12 = [cos(q2) -sin(q2) 0 l1*cos(q2); sin(q2) cos(q2) 0 l1*sin(q2); 0 0 1 0; 0 0 0 1];
T23 = [cos(q3) -sin(q3) 0 l2*cos(q3); sin(q3) cos(q3) 0 l2*sin(q3); 0 0 1 0; 0 0 0 1];
T34 = [cos(q4) 0 sin(q4) 0; sin(q4) 0 -cos(q4) 0; 0 1 0 0; 0 0 0 1];
T45 = [cos(q5) -sin(q5) 0 0; sin(q5) cos(q5) 0 0; 0 0 1 l3; 0 0 0 1];
```

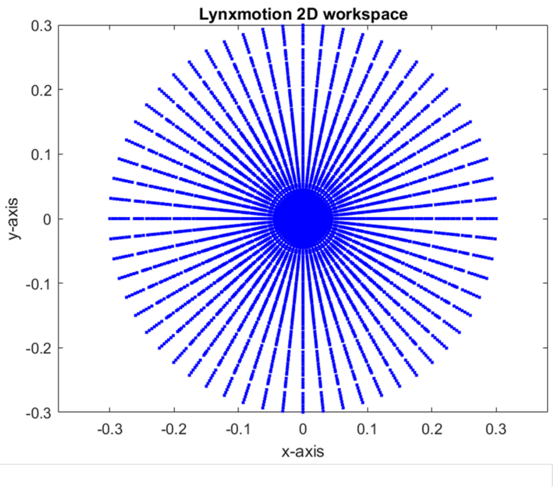
By multiplying the above transformation matrices a transformation matrix that describes the whole system can be formed as shown below in Eq.1.

$$T05 = T01 * T12 * T23 * T34 * T45; \tag{1}$$

2. Question 2

Analyse the workspace of the centre of the wrist (5th joint) when each preceding joint moves through its range of motion and plot the 2D and 3D views of the workspace.

The workspace of a robot is the area with which the end effector can move and interact with its environment. As such it is defined by the geometry and construction of the robot. By approximating the link lengths $l1$, $l2$, $l3$ and $d1$ to 0.1m and using the previously developed forward kinematic transformation matrices can be used to develop a simulation of the robot arms workspace in MATLAB by using a series of for loops to iterate each joint through its range of motion.



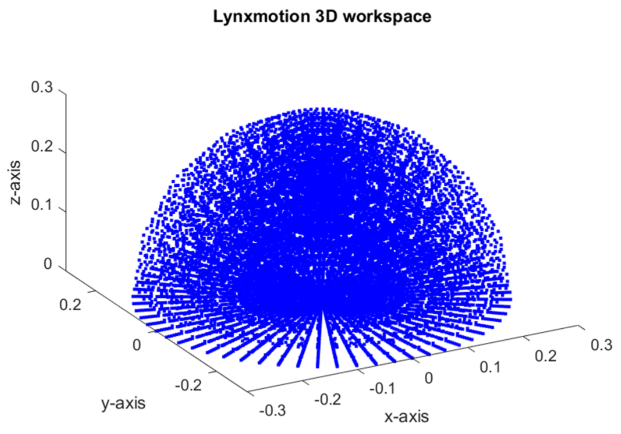


FIG. 2. Two representations of the lynxmotion robot arms workspace. The left image shows a 2 dimensional 'top down' view of the workspace and the right image shows the 3 dimensional simulation.

Fig.2 shows workspace from 2 different view points, a 2 dimensional view of the workspace where just the x and y axes have been included and a 3D view where the z axis has been added. Contrary to the left image, the true workspace of the robot is everything inside of the circle except from the centre where the robots base sits. It does not have a segmented workspace as shown above, that is a result of the step size when iterating through joint q1's range of motion. The radius of the workspace is the summation of the link length from q2 to the end effector.

The right hand image, showing the 3D workspace has a flat bottom. This image has been simulated with the assumption that the robot is likely sat on a flat and level surface such as a table, and hence the workspace can not go negative in the z axis.

3. Question 3

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

The inverse kinematics of a robot are used to determine what joint angles are required to position the end effector at a specified position. The forward kinematics conversely have been used to determine the position and orientation of the end effector given as prescribed set of joint angles [1]. This is a very important part of robot control and motion planning as it essentially determines what inputs are required for a certain output. The inverse kinematics of a robotic system can be solved iteratively or analytically. The iterative method involves repeatedly changing the joint angles of the robot until the end effector gets to the desired position. This method is should always converge to a solution but it is very computationally expensive. The analytical method use mathematical equations with the geometry of the robot to determine the solution. This method is more efficient than than the iterative solution but can be complicated to derive the required equations.

With the bottom joint, q1, being a parallel revolute joint the spatial representation of the robot arm can be simplified to a planar one. With the angle of q1 being pre-defined the robot manipulator will now only work in a slice of of the z plane in the x - y frame. The last joint can also be omitted from initial calculations as it is also a parallel revolute joint and does not effect the position of the end effector, only the orientation of the manipulators grippers. The planar representation of joints 2-3-4 are shown below in FIG.3.

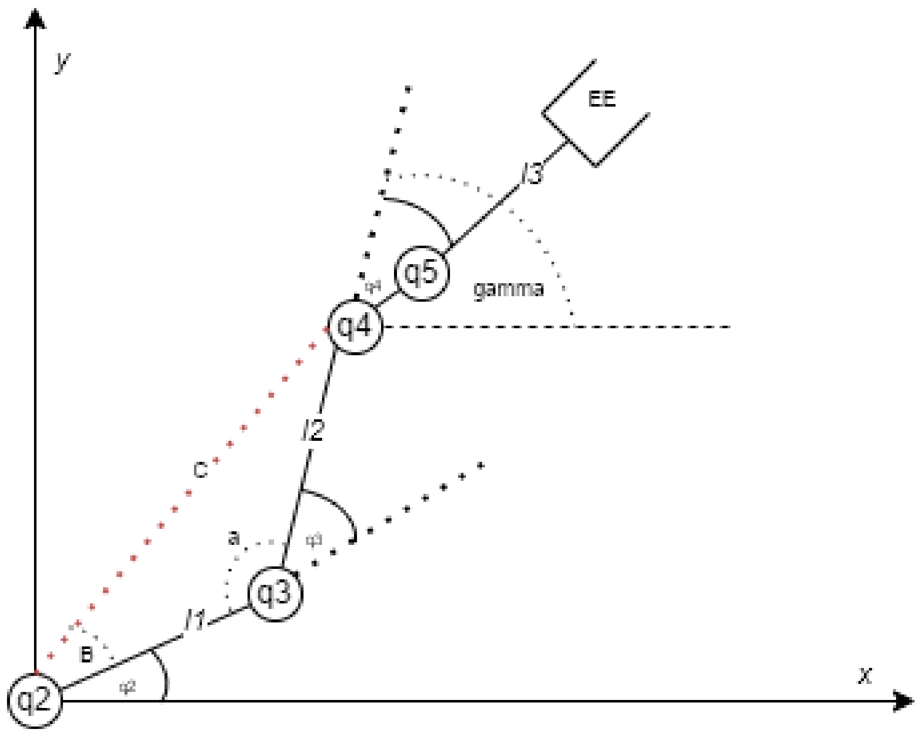


FIG. 3. A planar view of the lynxmotion arm.

With a predefined position for the end effector and known length of $l3$ the x, y, and z, of joint q4 can be deduce using Eq.2. The angle of $l3$ from q4 to the end effector is denoted as γ in the Fig.3.

$$\begin{bmatrix} x_4 \\ y_4 \end{bmatrix} = \begin{bmatrix} X_{ee} - (l3 * cos(\gamma)) \\ Y_{ee} - (l3 * sin(\gamma)) \end{bmatrix} \tag{2}$$

Using Pythagoras, the distance between q2 and q4 can be established as can be shown by the red line labelled . This then allows the calculation of the second and third joints using the law of cosines. First Q2 can be found by:

$$\arccos{(l1^2 + C^2 - l2^2)/(2l1C)}$$

The third joint is found by first determining:

$$\arccos{(l1^2 + l2^2 - C^2)/(2l1l2)}$$

and then subtracting it from 180. As all of the joint angles 2,3 and 4 will sum together to give angle γ we can subtract the angle of 2 and 3 to give the angle of q4.

When calculating the inverse kinematics for a robot there can be many solutions. This is due to their being many possible joint configurations that may lead to the end effector being in the correct place. The program that has been developed gives 2 sets of possible joint values with the second and third joints that will end with the end effector ending in the required position.

Below in Fig.5 you can see the plot that has been created from the joint angles calculated from a defined end effector position. You can see that the calculations give 2 possible configurations that position the end effector at the desired cartesian coordinates which are $(0.15, 0.15, 0.05)$. The elbow up arm path is coloured in green in FIG.5. The output from the program with the defined joint angles are is shown below:

Elbow-DOWN: The joint 2, 3 and 4 angles are (-21.326260,107.399839, -26.073578).
Elbow-UP: The joint 2, 3 and 4 angles are (86.073578,-107.399839, 81.326260).

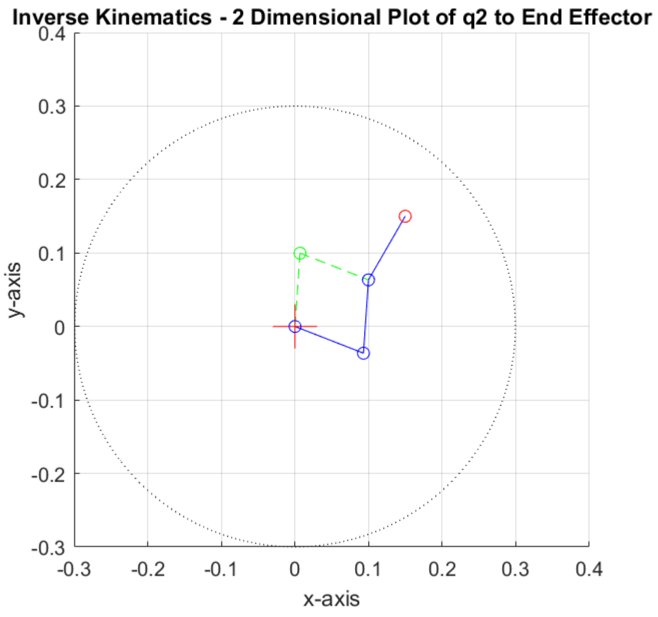


FIG. 4. The 2D representation of the robot manipulator inverse kinematic plot.

B. Part B

1. *Question 1*

Plan a task* in MATLAB with at least 5 positions. This process should give you at least 5 sets of Cartesian coordinates specifying the end-effector position and orientation in 3D space.

The planned shape was a simple square based pyramid with the robot manipulator in the centre. The coordinates of the shape are as follows:

Corner1 = [-0.15, 0.15, 0.05];
Corner2 = [0.15, 0.15, 0.05];
Corner3 = [0.15, -0.15, 0.05];
Corner4 = [-0.15, -0.15, 0.05];
Top = [0.0, 0.0, 0.25];

This shape makes the robot arm move to four corner points in both positive and negative regions of the x and y axis. As the robot will be sitting on a flat surface only the positive z axis has been modelled with the Top coordinate. Comparison with the forward kinematic model developed previously showed different results which will require further work to correct.

2. *Question 2*

Solve the Inverse Kinematics for these positions in 3D space and obtain sets of Joint Coordinates. Create an appropriate plot/animation in MATLAB for the motion of the robot.

With the end effector positions specified, a program was written that would cycle through the corners of the pyramid in a numerical and clockwise direction and then to the top coordinate. The output of the program will again show two sets of answers for the perpendicular revolute joints 2, 3, and 4. Below in Fig.5, 3 of the positions are shown, $Corner1$, $Corner4$ and Top. Working with the assumption that the robot would be placed on a level flat surface the green rectangle represents that surface which is plotted at z = 0. It can be seen that the robot manipulator orientates its joints in 2 configurations as to position the end effector at the x. The link lengths have not been specified explicitly in the plot, only the positions of each joint. the path between consequtive joints has been plotted and these represent the links in the robot arm. By roughly measuring the link lengths against the backing grid the system seems to be the correct length of 0.1 for lengths $l1$, $l2$ and $l3$. A more definite method of calculating the link lengths with the absolute magnitude between the two points could easily be implemented.

The joint positions are shown below:

Corner1 = [-0.15, 0.15, 0.0];

Elbow-DOWN: The joint 2, 3 and 4 angles are (74.920633,111.299806, -46.220439).
Elbow-UP: The joint 2, 3 and 4 angles are (186.220439,-111.299806, 65.079367).

Corner2 = [0.15, 0.15, 0.0];

Elbow-DOWN: The joint 2, 3 and 4 angles are (-21.326260,107.399839, -26.073578).
Elbow-UP: The joint 2, 3 and 4 angles are (86.073578,-107.399839, 81.326260).

Corner3 = [0.15, -0.15, 0.0];

Elbow-DOWN: The joint 2, 3 and 4 angles are (-101.777265,57.773971, 54.003294).
Elbow-UP: The joint 2, 3 and 4 angles are (-44.003294,-57.773971, 111.777265).

Corner4 = [-0.15, -0.15, 0.0];

Elbow-DOWN: The joint 2, 3 and 4 angles are (-135.996706,57.773971, 248.222735).
Elbow-UP: The joint 2, 3 and 4 angles are (-78.222735,-57.773971, 305.996706).

Top = [0.0, 0.0, 0.25];

Elbow-DOWN: The joint 2, 3 and 4 angles are (-110.000000,120.000000, 120.000000).
Elbow-UP: The joint 2, 3 and 4 angles are (10.000000,-120.000000, 240.000000).

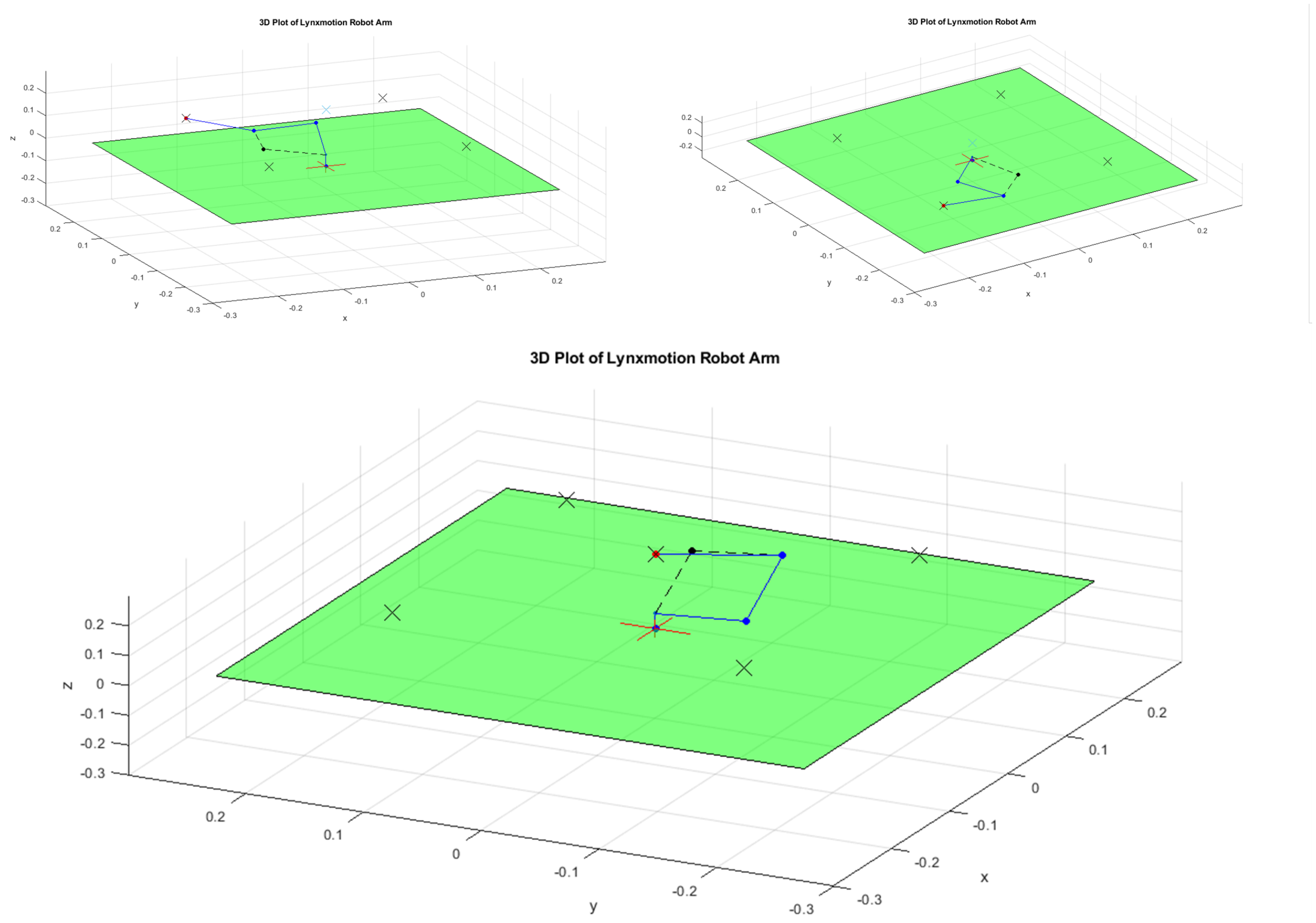


FIG. 5. Three examples of the inverse kinematic plots of 3 end effector positions.

3. Question 2

Implement 3 different trajectories between the Cartesian Points identified above and create an appropriate plot to demonstrate them):

A : Implement free motion between the points
Free motion between points is where the manipulator moves naturally though the points in any direction and at any speed, it is comparable to how a human might move his arms with each joint moving independently from each other.

Via points can be used to give the robot smoother [3], less jerky motion when making corners but this will result in the end effector cutting a corner so the application would need to allow for that and applications where precision is deemed a high priority this may not be suitable.

To implement free motion code would have to be written that iterates through a range of positions and joint orientations to give the manipulator an unrestricted path between the corners.

B : Implement straight line trajectory between the points
Straight line trajectory is a lot simpler than free motion. As the name suggests the end effector of the manipulator will move in a straight line between the corners and will reach every defined coordinate. The direct path between

the corners can be split into distinct steps whereby the end effector position and the orientation is specified and the inverse kinematic calculations can be carried out to discern the joint angles of the manipulators. The route can then be interpolated between the points.

A program was created that utilised the previously written inverse kinematics to plot the robot at each point. Each side of the shape was split into 10 points which was then iterated over to simulate the robot moving between points. The code does show the end effector moving across the shape along a straight line between points. It does however jump at a few points, and a more efficient algorithm should be implemented to determine the correct value of γ for each step.

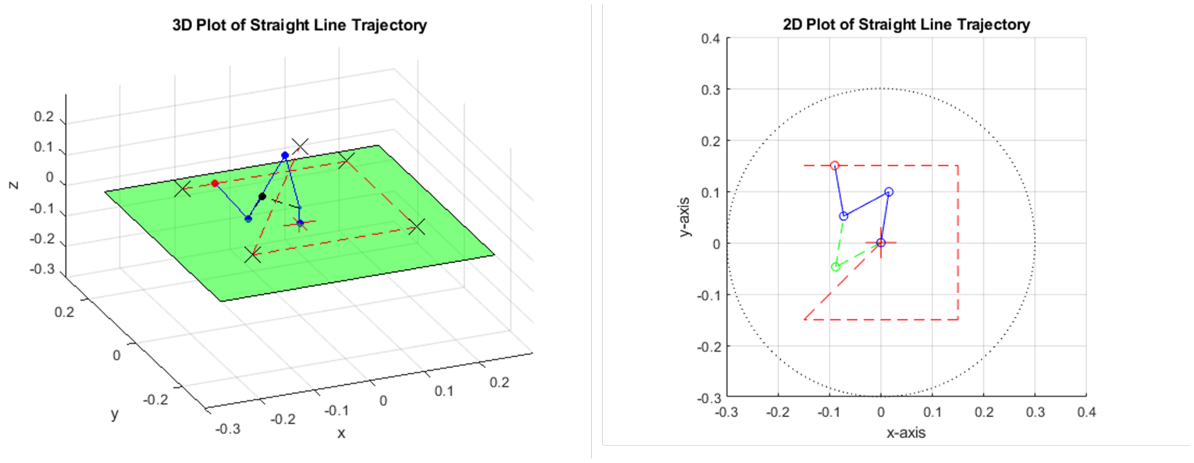


FIG. 6. 3D and 2D plots showing the straight line trajectory of the robot arm moving between corners.

FIG. 6 shows 2D and 3D representations of the path taken during during the straight line trajectory with the robot manipulator part way through moving from Corner1 to Corner2.

C : Set an obstacle between any two points (e.g. a cylinder between point 3 and 4) and implement an object avoidance trajectory.

Often when motion planning within robotics object, avoidance algorithms may be utilised to enable the manipulator to reach a desired goal despite an obstacle [6]. Examples of these include the Backward and Forward Unification Gradient (BUG) algorithms. These aim to combine both forward and backward motion to reach the goal.

BUG algorithms use gradient-based method to search for a path in the configuration space of the manipulator [4]. They have been developed since the late 80's and provide a simple, low computation set of instruction that allow the manipulator to avoid obstacles whilst still working towards the desired location. There are 3 iterations of this algorithm, BUG0, BUG1 and BUG2. These are briefly summarised below [4]:

- BUG0: Will move in the direction whenever it can, which may cause it to get trapped.
- BUG1: Will do a full loop of the obstacle and will leave towards its goal when the magnitude between the robot and the goal is the least. This can cause the robot to take bvery long paths.
- BUG2: An imaginary line is drawn between the start point and the goal, if an obstacle is encountered the robot will loop in a direction until it reached the imaginary line again whereby it will carry on following it.

C. Part 2 - Planar Parallel Robot

1. *Question 1 - Inverse Kinematics and Plot*

Parallel robots generally are made of both moving and non-moving parts. They can offer some advantages over single sided robots, with multiple bases they can be more stable, which in turn can increase the precision and speed with which they can be used [5]. With increased precision they lend themselves well to applications within surgery where they may outperform a surgeon with some tasks. For a robot to be operating in so close proximity to humans it is crucial that kinematic analysis is carried out so that the end effector will reliably be where indented. With the increased limbs the complexity of the design and control of parallel robots is greater and they generally have a smaller workspace than that of a serial robot .

Plotting the kinematics of this parallel planar robot can be achieved in a similar fashion to the singular robot manipulator. By specifying the position of the end effector the chain back to the base can be established. The design of this robot specifies that the end effector sits in the middle a platform in the shape of an equilateral triangle. From the geometry of the triangle you can establish that the platform link connections will sit 120 degrees from each other at distance $r_{platform}$ from the end effector. By then calculating the difference between the base point and the platform point an inner triangle can be formed and the angles for the joint calculated using the cosine rule.

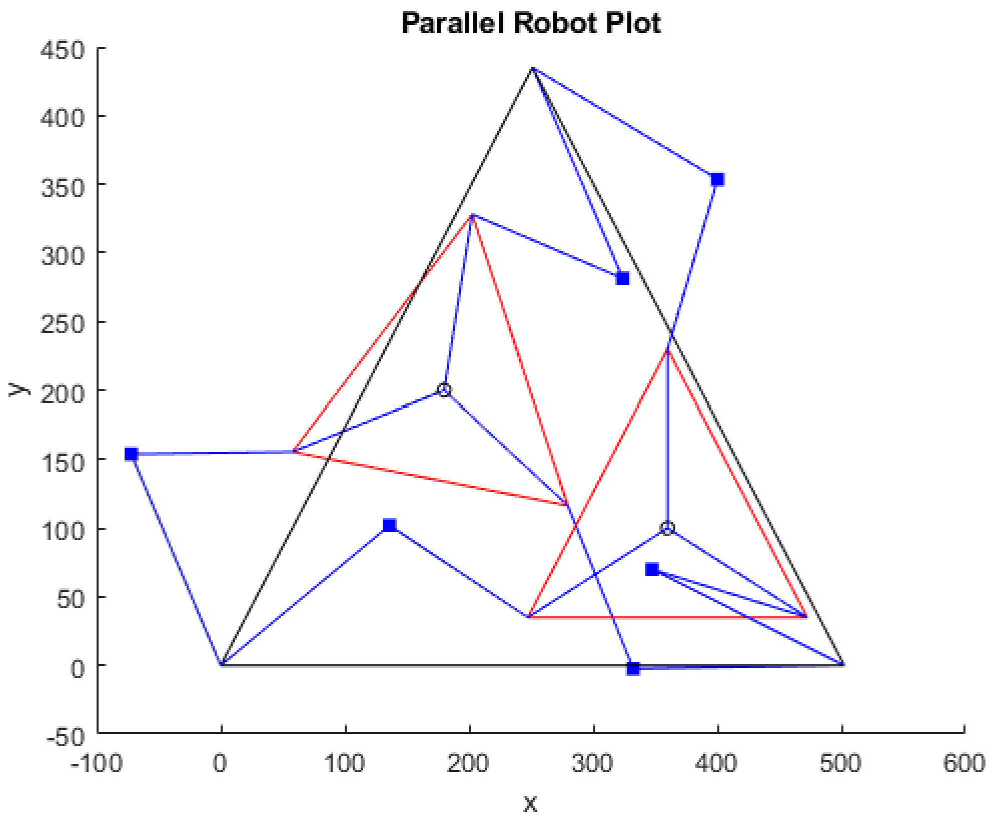


FIG. 7. A figure of the parallel robot in 2 different end effector position and orientations.

Fig. 7 shows the plot of the parallel robot in 2 different positions and orientations. You can see that the multiple robot arms attached to the platform allow the end effector to be positioned at different points within the workspace and can allow for a twist of the platform relative to the base.

2. Question 2 - Plotting the workspace

A way to plot the workspace for this parallel planar robot would involve iterating through for loops to position the end effector in every possible position allowed by the robots construction. A program was attempted to iterate over a range of x and y positions and then filter out points that were not in the workspace of the robot. The base and the platform are connected by 3 links and the platform rotates around the centre of the platform triangle. The combination of these two would produce a hexagonal workspace.

D. Part 3 - Investigation into the Filtered Inverse Approach to Singularities

A singularity is where a robot manipulator is operated at the limit of its workspace. This will cause the result of kinematic calculations to become either infinite or undefined which can then lead to unexpected and unpredictable movements by the robot [7] .

There are many algorithms that have been developed to minimise the risk caused by this problem, one such algorithm is the filtered inverse approach. The filter gets applies to the inverse solutions so that they always remain defined even when the robot would normally be in a singularity. The filter that gets applied is an modified jacobian matrix. There are several filters that may be applied to the jacobian such as the pseudo-inverse or a damping factor. These prevent the jacobian from ever becoming singular and therefore avoids the singularity condition and enables predictable control of the robot.

[1] Aristidou, A. and Lasenby, J. [2009], 'Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver'.

[2] Jafari, A. [2022*a*], 'Week 3 - forward kinematics - denavit hartenberg', Blackboard.

[3] Jafari, A. [2022*b*], 'Week 8 - trajectories i', Blackboard.

[4] Mcguire, K., Croon, G. and Tuyls, K. [2019], 'A comparative study of bug algorithms for robot navigation', *Robotics and Autonomous Systems* **121**, 103261.

[5] Park, F. [2014], *Parallel Robots*, pp. 1–8.

[6] Petric, T., Gams, A., Likar, N. and Zlajpah, L. [2015], *Obstacle Avoidance with Industrial Robots*, Vol. 29, pp. 113–145.

[7] Vargas, L. V., Leite, A. C. and Costa, R. R. [2014], 'Overcoming kinematic singularities with the filtered inverse approach', *IFAC Proceedings Volumes* **47**(3), 8496–8502. 19th IFAC World Congress.
URL: *https://www.sciencedirect.com/science/article/pii/S147466701642954X*

QUESTION 1

```
syms c a alpha dN theta l1 l2 l3 q1 q2 q3 q4 q5 d;

Link = [1;2;3;4;5];
aN = [0;l1;l2;0;0];
alphaN = [90;0;0;90;0];
DN = [d;0;0;0;l3];
thetaN = [q1;q2;q3;q4;q5];

% Table of value calulated from the drawing of the Lynx Robot Arm
Distal_Table = table(Link, aN, alphaN, DN, thetaN)




%Indiviual transpose matrices from the distal table
T01 = [cos(q1) 0 sin(q1) 0; sin(q1) 0 cos(q1) 0; 0 1 0 d; 0 0 0 1];
T12 = [cos(q2) -sin(q2) 0 l1*cos(q2); sin(q2) cos(q2) 0 l1*sin(q2); 0 0 1 0; 0 0 0 1];
T23 = [cos(q3) -sin(q3) 0 l2*cos(q3); sin(q3) cos(q3) 0 l2*sin(q3); 0 0 1 0; 0 0 0 1];
T34 = [cos(q4) 0 sin(q4) 0; sin(q4) 0 -cos(q4) 0; 0 1 0 0; 0 0 0 1];
T45 = [cos(q5) -sin(q5) 0 0; sin(q5) cos(q5) 0 0; 0 0 1 l3; 0 0 0 1];

% Get the transpose from the origin to the end effector by multiplying all
% of the individual transposes together

T05 = T01*T12*T23*T34*T45;

xt = T05(1,4)          % End position in x

yt = T05(2,4)          % End position in y

zt = T05(3,4)          % End position in z

pt = [ xt yt zt ] ;
```

———————————— WORKSPACE ————————————

```
%% Part 1.1

clear all;
clc;

syms c a alpha dN theta l1 l2 l3 q1 q2 q3 q4 q5 d;

Link = [1;2;3;4;5];
aN = [0;l1;l2;0;0];
alphaN = [90;0;0;90;0];
DN = [d;0;0;0;l3];
thetaN = [q1;q2;q3;q4;q5];

% Table of value calulated from the drawing of the Lynx Robot Arm
Distal_Table = table(Link, aN, alphaN, DN, thetaN)
```

```
%Indiviual transpose matrices from the distal table


T01 = [cos(q1) 0 sin(q1) 0; sin(q1) 0 cos(q1) 0; 0 1 0 d; 0 0 0 1];
T12 = [cos(q2) -sin(q2) 0 l1*cos(q2); sin(q2) cos(q2) 0 l1*sin(q2); 0 0 1 0; 0 0 0 1];
T23 = [cos(q3) -sin(q3) 0 l2*cos(q3); sin(q3) cos(q3) 0 l2*sin(q3); 0 0 1 0; 0 0 0 1];
T34 = [cos(q4) 0 sin(q4) 0; sin(q4) 0 -cos(q4) 0; 0 1 0 0; 0 0 0 1];
T45 = [cos(q5) -sin(q5) 0 0; sin(q5) cos(q5) 0 0; 0 0 1 l3; 0 0 0 1];

% Get the transpose from the origin to the end effector by multiplying all
% of the individual transposes together

T05 = T01*T12*T23*T34*T45;


%% ----------------------------- Part 1.2 -----------------------------------------------------------


%% Links Lengths
l1 = 0.1 ;
l2 = 0.1 ;
l3 = 0.1 ;
d = 0.05;




%% Workspace
q1 = 0:pi/30:2*pi-pi/180 ;    %pi/180  = the step distance  <-------------------- These look to
q2 = 0:pi/10:pi-pi/180 ;
q3 = 0:pi/10:pi-pi/180 ;
q4 = 0:pi/10:pi-pi/180 ;
q5 = 0:pi/10:2*pi-pi/180 ;


%% Plot the workspace of the robot
 figure (3)
 set(3,'position',[1243 190 560 420])

Xwork = [];
Ywork = [];
Zwork = [];
%


 for i = 1:length(q1)    % for q1
     for j = 1:length(q2)   % for q2
         for k = 1:length(q3) % for q3
             for l = 1:length(q4) % for q4
                 xwork = l3*(cos(q4(l))*(cos(q1(i))*cos(q2(j))*sin(q3(k)) + cos(q1(i))*cos(q3(l
                 ywork = l3*(cos(q4(l))*(cos(q2(j))*sin(q1(i))*sin(q3(k)) + cos(q3(k))*sin(q1(
                 zwork = d + l1*sin(q2(j)) - l3*(cos(q4(l))*(cos(q2(j))*cos(q3(k)) - sin(q2(j)
                 zwork = zwork - d;
                  if zwork < 0
                     zwork = 0;
                 end
                 Xwork = [Xwork xwork];
                 Ywork = [Ywork ywork];
```

```
                Zwork = [Zwork zwork];


            end
        end
    end
end

figure(3)
plot3(Xwork,Ywork,Zwork, '.', 'color', 'b')
title('Lynxmotion 3D workspace')
xlabel('x-axis')
ylabel('y-axis')
zlabel('z-axis')
axis equal

figure(4)
plot(Xwork, Ywork,'.', 'color', 'b')
title('Lynxmotion 2D workspace')
xlabel('x-axis')
ylabel('y-axis')
axis equal




% %Inverse Kinematic test
% %q1 = 0:pi/30:2*pi-pi/180 ;
% q1 = pi/4;
% %q2 = 0:pi/10:pi-pi/180 ;
% q2 = deg2rad(-21.326260);
% %q3 = 0:pi/10:pi-pi/180 ;
% q3 = deg2rad(107.399839);
% %q4 = 0:pi/10:pi-pi/180 ;
% q4 = deg2rad(-26.073578);
% q5 = 0;
%
% Xwork = [];
% Ywork = [];
% Zwork = [];
%
% for i = 1:length(q1)  % for q1
%     for j = 1:length(q2)   % for q2
%         for k = 1:length(q3) % for q3
%             for l = 1:length(q4) % for q4
%                 xwork = l3*(cos(q4(l))*(cos(q1(i))*cos(q2(j))*sin(q3(k)) + cos(q1(i))*cos(q
%                 ywork = l3*(cos(q4(l))*(cos(q2(j))*sin(q1(i))*sin(q3(k)) + cos(q3(k))*sin(q
%                 zwork = d + l1*sin(q2(j)) - l3*(cos(q4(l))*(cos(q2(j))*cos(q3(k)) - sin(q2(
%                 zwork = zwork - d;
%                 if zwork < 0
%                     zwork = 0;
%                 end
%                 Xwork = [Xwork xwork];
%                 Ywork = [Ywork ywork];
%                 Zwork = [Zwork zwork];
%
%
%             end
%         end
%     end
% end
%
%
% figure(5)
% plot3(Xwork,Ywork,Zwork, '.', 'color', 'b')
% title('IK test')
% xlabel('x-axis')
% ylabel('y-axis')
% zlabel('z-axis')
% axis equal
```

INVERSE KINEMATICS

```
clear all;
clc;

% Corner coordinates [x,y,z]
Corner1 = [-0.15, 0.15, 0.05];  % fine
Corner2 = [0.15, 0.15, 0.05];  % fine
Corner3 = [0.15, -0.15, 0.05];  %fine
Corner4 = [-0.15, -0.15, 0.05];  % fine
Top = [0.00, 0.00, 0.25]; % fine


%for psi = 0:10:180;
 %disp(psi);
 pause(2);
 psi = 140;
 ikinematics(0.1,0.1,0.1, Corner1(1), Corner1(2), Corner1(3),psi);  %yes
 pause(2);
 psi = 60;
 ikinematics(0.1,0.1,0.1, Corner2(1), Corner2(2), Corner2(3),psi); %yes
 pause(2)
 psi = 10;
 ikinematics(0.1,0.1,0.1, Corner3(1), Corner3(2), Corner3(3),psi);  % yes
 pause(2);
 psi = 170;
 ikinematics(0.1,0.1,0.1, Corner4(1), Corner4(2), Corner4(3),psi);  %no
 pause(2);
 psi = 140;
 ikinematics(0.1,0.1,0.1, Corner1(1), Corner1(2), Corner1(3),psi);  %yes
 pause(2);
 psi = 130;
 ikinematics(0.1,0.1,0.1, Top(1), Top(2), Top(3),psi);   %Yes
 pause(2);
%end




 function ikinematics(links1, links2, links3, positionx, positiony, positionz, gamma)

% Inputs:
% links1,links2,links3 are length of each links in the robotic arm
% base is the length of base of the robotic arm
% positionx,positiony are joint angles in reference to x-axis
% gamma is the orientation
```

```
format compact
format short



l1 = links1;
l2 = links2;
l3 = links3;
d1 = 0.05;
Xee = positionx;
Yee = positiony;
Zee = positionz;
g = gamma;

%position P4
x4 = Xee-(l3*cosd(g));
y4 = Yee-(l3*sind(g));
z4 = Zee - (d1*sind(g));
C = sqrt(x4^2 + y4^2);

if (l1+l2) > C
    %angle a and B
    a = acosd((l1^2 + l2^2 - C^2 )/(2*l1*l2));
    B = acosd((l1^2 + C^2 - l2^2 )/(2*l1*C));

    %joint angles elbow-down
    J2a = atan2d(y4,x4)-B;
    J3a = 180-a;
    J4a = g - J2a -J3a;

    %joint angles elbow-up
    J2b = atan2d(y4,x4)+B;
    J3b = -(180-a);
    J4b = g - J2b - J3b;

    fprintf('Elbow-DOWN: The joint 2, 3 and 4 angles are (%f,%f, %f).\n',J2a,J3a,J4a)
    fprintf('Elbow-UP: The joint 2, 3 and 4 angles are (%f,%f, %f).\n',J2b,J3b,J4b)
else
    disp('      Dimension error!')
    disp('      End effecter is outside the workspace.')
    return
end

clf;
x2a = l1*cosd(J2a);
y2a = l1*sind(J2a);
z2a = l1*sind(g) + d1;

x2b = l1*cosd(J2b);
y2b = l1*sind(J2b);
z2b = l1*sind(g) + d1;


figure(1)
clf
r = l1 + l2 + l3;
daspect([1,1,1])
rectangle('Position',[-r,-r,2*r,2*r],'Curvature',[1,1],...
    'LineStyle',':')
line([0 x2a], [0 y2a],'Color','b')
line([x2a x4], [y2a y4],'Color','b')
line([x4 Xee], [y4 Yee],'Color','b')
line([0 x2b], [0 y2b],'Color','g','LineStyle','--')
line([x2b x4], [y2b y4],'Color','g','LineStyle','--')
line([x4 Xee], [y4 Yee],'Color','b','LineStyle','--')
%line([0 xe], [0 ye],'Color','r')
line([0 0], [-r/10 r/10], 'Color', 'r')
line([-r/10 r/10], [0 0], 'Color', 'r')
hold on
plot([0 x2a x4],[0 y2a y4],'o','Color','b')
plot([x2b],[y2b],'o','Color','g')
plot([Xee],[Yee],'o', 'Color', 'r')
grid on
xlabel('x-axis')
ylabel('y-axis')
title('Inverse Kinematics - 2 Dimensional Plot of q2 to End Effector')



Corner1 = [-0.15, 0.15, 0.05];  % fine
Corner2 = [0.15, 0.15, 0.05];  % fine
Corner3 = [0.15, -0.15, 0.05];  %fine
Corner4 = [-0.15, -0.15, 0.05];  % fine
Top = [0.00, 0.00, 0.25]; % fine







 % Plot the points
figure(3)
 clf
 hold on
 plot3(0, 0, d1, 'marker', '.', 'markersize', 10)
 plot3(Corner1(1), Corner1(2), Corner1(3), 'marker', 'x', 'markersize', 15, 'Color', 'k')
 plot3(Corner2(1), Corner2(2), Corner2(3), 'marker', 'x', 'markersize', 15, 'Color', 'k')
 plot3(Corner3(1), Corner3(2), Corner3(3), 'marker', 'x', 'markersize', 15, 'Color', 'k')
 plot3(Corner4(1), Corner4(2), Corner4(3), 'marker', 'x', 'markersize', 15, 'Color', 'k')
 plot3(Top(1), Top(2), Top(3), 'marker', 'x', 'markersize', 15, 'Color', 'k')
 line([0 0], [-r/10 r/10], [0 0], 'Color', 'r')
 line([-r/10 r/10], [0 0], [0 0], 'Color', 'r')
 line([0 0], [0 0], [-r/10 r/10], 'Color', 'r')
 axis([-0.3 0.3 -0.3 0.3 -0.3 0.3]);     % Axis dimensions
 % Option A path
 line([0 0], [0 0], [0 d1],'Color','b')
 line([0 x2a], [0 y2a], [d1 z2a],'Color','b')   % Origin to first joint q3
 line([x2a x4], [y2a y4], [z2a z4],'Color','b')   % q3 to q4
 line([x4 Xee], [y4 Yee ], [z4 Zee],'Color','b')  % q4 to ee
 % Option B path
 line([0 0], [0 0], [0 d1],'Color','b')
 line([0 x2b], [0 y2b], [d1 z2b],'Color','k','LineStyle','--')
 line([x2b x4], [y2b y4], [z2b z4],'Color','k','LineStyle','--')
 line([x4 Xee], [y4 Yee], [z4 Zee],'Color','b','LineStyle','--')
 plot3([Xee], [Yee], [Zee],'.', 'Color', 'r', 'markersize', 15) % end effector
```

```
  plot3([0 x2a x4],[0 y2a y4],[0 z2a z4],'.','Color','b', 'markersize', 15)
  plot3([x2b],[y2b],[z2b],'.','Color','k', 'markersize', 15)
%  rectangle('Position',[-0.15,-0.15,0.3,0.3],'FaceColor','g');
x = [-0.25 0.25 0.25 -0.25];
y = [-0.25 -0.25 0.25 0.25];
z = [0 0 0 0];
fill3(x, y, z, 'g', 'FaceAlpha', 0.5)
  grid on
  hold off

 % Add labels and a title
  xlabel('x')
  ylabel('y')
  zlabel('z')
  title('3D Plot of Lynxmotion Robot Arm')

 % Set the view
  view(-23, 38)


pause(2);
end
```

—————————————— TRAJECTORIES ——————————————

```
Corner1 = [-0.15, 0.15, 0.05];  % fine
Corner2 = [0.15, 0.15, 0.05];  % fine
Corner3 = [0.15, -0.15, 0.05];  %fine
Corner4 = [-0.15, -0.15, 0.05];  % fine
Top = [0.00, 0.00, 0.25]; % fine


straightLine(Corner1, Corner2, Corner3, Corner4, Top);



function ikinematics(links1, links2, links3, positionx, positiony, positionz, gamma)
% Inputs:
% links1,links2,links3 are length of each links in the robotic arm
% base is the length of base of the robotic arm
% positionx,positiony are joint angles in reference to x-axis
% gamma is the orientation

format compact
format short

Corner1 = [-0.15, 0.15, 0.05];  % fine
Corner2 = [0.15, 0.15, 0.05];  % fine
Corner3 = [0.15, -0.15, 0.05];  %fine
Corner4 = [-0.15, -0.15, 0.05];  % fine
Top = [0.00, 0.00, 0.25]; % fine



l1 = links1;
l2 = links2;
```

```
l3 = links3;
d1 = 0.05;
Xee = positionx;
Yee = positiony;
Zee = positionz;
g = gamma;

%position P4
x4 = Xee-(l3*cosd(g));
y4 = Yee-(l3*sind(g));
z4 = Zee - (d1*sind(g));
C = sqrt(x4^2 + y4^2);

if (l1+l2) > C
    %angle a and B
    a = acosd((l1^2 + l2^2 - C^2 )/(2*l1*l2));
    B = acosd((l1^2 + C^2 - l2^2 )/(2*l1*C));

    %joint angles elbow-down
    J2a = atan2d(y4,x4)-B;
    J3a = 180-a;
    J4a = g - J2a -J3a;

    %joint angles elbow-up
    J2b = atan2d(y4,x4)+B;
    J3b = -(180-a);
    J4b = g - J2b - J3b;

    fprintf('Elbow-DOWN: The joint 2, 3 and 4 angles are (%f,%f, %f).\n',J2a,J3a,J4a)
    fprintf('Elbow-UP: The joint 2, 3 and 4 angles are (%f,%f, %f).\n',J2b,J3b,J4b)
else
    disp('      Dimension error!')
    disp('      End effecter is outside the workspace.')
    return
end

clf;
x2a = l1*cosd(J2a);
y2a = l1*sind(J2a);
z2a = l1*sind(g) + d1;

x2b = l1*cosd(J2b);
y2b = l1*sind(J2b);
z2b = l1*sind(g) + d1;


figure(1)
clf
r = l1 + l2 + l3;
daspect([1,1,1])
rectangle('Position',[-r,-r,2*r,2*r],'Curvature',[1,1],...
    'LineStyle',':')
line([0 x2a], [0 y2a],'Color','b')
line([x2a x4], [y2a y4],'Color','b')
line([x4 Xee], [y4 Yee],'Color','b')
line([0 x2b], [0 y2b],'Color','g','LineStyle','--')
line([x2b x4], [y2b y4],'Color','g','LineStyle','--')
line([x4 Xee], [y4 Yee],'Color','b','LineStyle','--')
```

```
 line([Corner1(1) Corner2(1)], [Corner1(2)  Corner2(2)], 'Color', 'r', 'LineStyle', '--')
 line([Corner2(1) Corner3(1)], [Corner2(2)  Corner3(2)], 'Color', 'r', 'LineStyle', '--')
 line([Corner3(1) Corner4(1)], [Corner3(2)  Corner4(2)], 'Color', 'r', 'LineStyle', '--')
 line([Corner4(1) Top(1)], [Corner4(2)  Top(2)], 'Color', 'r', 'LineStyle', '--')
%  plot(Corner1(1), Corner1(2), 'marker', 'x', 'markersize', 15, 'Color', 'k')
%  plot(Corner2(1), Corner2(2), 'marker', 'x', 'markersize', 15, 'Color', 'k')
%  plot(Corner3(1), Corner3(2), 'marker', 'x', 'markersize', 15, 'Color', 'k')
%  plot(Corner4(1), Corner4(2), 'marker', 'x', 'markersize', 15, 'Color', 'k')
%  plot(Top(1), Top(2), 'marker', 'x', 'markersize', 15, 'Color', 'k')
%line([0 xe], [0 ye],'Color','r')
line([0 0], [-r/10 r/10], 'Color', 'r')
line([-r/10 r/10], [0 0], 'Color', 'r')
hold on
plot([0 x2a x4],[0 y2a y4],'o','Color','b')
plot([x2b],[y2b],'o','Color','g')
plot([Xee],[Yee],'o', 'Color', 'r')
grid on
xlabel('x-axis')
ylabel('y-axis')
title('2D Plot of Straight Line Trajectory')
```

```
 % Plot the points
figure(3)
 clf
 hold on
 plot3(0, 0, d1, 'marker', '.', 'markersize', 10)
 plot3(Corner1(1), Corner1(2), Corner1(3), 'marker', 'x', 'markersize', 15, 'Color', 'k')
 plot3(Corner2(1), Corner2(2), Corner2(3), 'marker', 'x', 'markersize', 15, 'Color', 'k')
 plot3(Corner3(1), Corner3(2), Corner3(3), 'marker', 'x', 'markersize', 15, 'Color', 'k')
 plot3(Corner4(1), Corner4(2), Corner4(3), 'marker', 'x', 'markersize', 15, 'Color', 'k')
 plot3(Top(1), Top(2), Top(3), 'marker', 'x', 'markersize', 15, 'Color', 'k')
 line([Corner1(1) Corner2(1)], [Corner1(2)  Corner2(2)], [Corner1(3) Corner2(3)], 'Color', 'r
 line([Corner2(1) Corner3(1)], [Corner2(2)  Corner3(2)], [Corner2(3) Corner3(3)], 'Color', 'r
 line([Corner3(1) Corner4(1)], [Corner3(2)  Corner4(2)], [Corner3(3) Corner4(3)], 'Color', 'r
 line([Corner4(1) Top(1)], [Corner4(2)  Top(2)], [Corner4(3) Top(3)], 'Color', 'r', 'LineStyle
 line([0 0], [-r/10 r/10], [0 0], 'Color', 'r')
 line([-r/10 r/10], [0 0], [0 0], 'Color', 'r')
 line([0 0], [0 0], [-r/10 r/10], 'Color', 'r')
 axis([-0.3 0.3 -0.3 0.3 -0.3 0.3]);     % Axis dimensions
 % Option A path
 line([0 0], [0 0], [0 d1],'Color','b')
 line([0 x2a], [0 y2a], [d1 z2a],'Color','b')   % Origin to first joint q3
 line([x2a x4], [y2a y4], [z2a z4],'Color','b')   % q3 to q4
 line([x4 Xee], [y4 Yee ], [z4 Zee],'Color','b')  % q4 to ee
 % Option B path
 line([0 0], [0 0], [0 d1],'Color','b')
 line([0 x2b], [0 y2b], [d1 z2b],'Color','k','LineStyle','--')
 line([x2b x4], [y2b y4], [z2b z4],'Color','k','LineStyle','--')
 line([x4 Xee], [y4 Yee], [z4 Zee],'Color','b','LineStyle','--')
 plot3([Xee], [Yee], [Zee],'.', 'Color', 'r', 'markersize', 15) % end effector
 plot3([0 x2a x4],[0 y2a y4],[0 z2a z4],'.','Color','b', 'markersize', 15)
 plot3([x2b],[y2b],[z2b],'.','Color','k', 'markersize', 15)
%  rectangle('Position',[-0.15,-0.15,0.3,0.3],'FaceColor','g');
x = [-0.25 0.25 0.25 -0.25];
y = [-0.25 -0.25 0.25 0.25];
z = [0 0 0 0];
fill3(x, y, z, 'g', 'FaceAlpha', 0.5)
 grid on
 hold off

 % Add labels and a title
 xlabel('x')
 ylabel('y')
 zlabel('z')
 title('3D Plot of Straight Line Trajectory')

 % Set the view
 view(-23, 38)


pause(2);
end


function straightLine(Corner1, Corner2, Corner3, Corner4, Top)


% Calculate the differences in x, y, and z between the corners.
delta12 = [(Corner2(1)-Corner1(1)), (Corner2(2)-Corner1(2)), (Corner2(3)-Corner1(3))];
delta23 = [(Corner3(1)-Corner2(1)), (Corner3(2)-Corner2(2)), (Corner3(3)-Corner2(3))];
delta34 = [(Corner4(1)-Corner3(1)), (Corner4(2)-Corner3(2)), (Corner4(3)-Corner3(3))];
delta45 = [(Top(1)-Corner4(1)), (Top(2)-Corner4(2)), (Top(3)-Corner4(3))];


disp(delta12);
disp(delta23);
disp(delta34);
disp(delta45);


% Divide the difference by the total no. of steps - Lets initially say 10

increment12 = delta12 / 10;
increment23 = delta23 / 10;
increment34 = delta34 / 10;
increment45 = delta45 / 10;


%Starting Positions
path12 = Corner1;
path23 = Corner2;
path34 = Corner3;
path45 = Corner4;

ikinematics(0.1,0.1,0.1, Corner1(1), Corner1(2), Corner1(3),140);
```

```
for n = 0:10
    path12 = path12 + increment12;
    ikinematics(0.1,0.1,0.1, path12(1), path12(2), path12(3),100);
end


%ikinematics(0.1,0.1,0.1, Corner2(1), Corner2(2), Corner2(3),140);

for n = 0:10
    path23 = path23 + increment23;
    ikinematics(0.1,0.1,0.1, path23(1), path23(2), path23(3),20);
end
% pause(1);
% ikinematics(0.1,0.1,0.1, Corner3(1), Corner3(2), Corner3(3),20);
% disp("1");
% pause(1);
% ikinematics(0.1,0.1,0.1, Corner3(1), Corner3(2), Corner3(3),30);
% disp("2");
% pause(1);
% ikinematics(0.1,0.1,0.1, Corner3(1), Corner3(2), Corner3(3),45);
% disp("3");
% pause(1);
% ikinematics(0.1,0.1,0.1, Corner3(1), Corner3(2), Corner3(3),120);
% disp("4");
% pause(1);
% ikinematics(0.1,0.1,0.1, Corner3(1), Corner3(2), Corner3(3),50);
% disp("5");
% pause(1);

for n = 0:3
    path34 = path34 + increment34;
    ikinematics(0.1,0.1,0.1, path34(1), path34(2), path34(3),20);
    pause(0.5);
end

for n = 0:2
    path34 = path34 + increment34;
    ikinematics(0.1,0.1,0.1, path34(1), path34(2), path34(3),0);
    pause(0.5);
end

for n = 0:5
    path34 = path34 + increment34;
    ikinematics(0.1,0.1,0.1, path34(1), path34(2), path34(3),160);
    pause(0.5);
end



for n = 0:9
    path45 = path45 + increment45;
    ikinematics(0.1,0.1,0.1, path45(1), path45(2), path45(3),120);
    pause(0.5);
end


end
```

——————————— PARALLEL ———————————

```
clear all;
clc;
clf;


% Defined robot dimensions
Sa=170;
L=130;
Rplatform = 130;
Rbase =290;



% Define the base positions
xbase1 = 0;   % Bottom left
ybase1 = 0;

xbase2 = Rbase*sqrt(3);  % Bottom Right
ybase2 = 0;

xbase3 = xbase2/2;            % Top
ybase3 = xbase2*sind(60);

% X and Y end effector positions
x_centre = [180 360];
y_centre = [200 100];


% Platform orientation
alpha = [-10 0];


for n = 1:2    % 2 as specified in question

    a = deg2rad(alpha(n));



% angle between ee-base point & x axis
psi1= a + (pi/6);
psi2 = a +(5*pi/6);
psi3 = (a - (pi/2));


xplat1 = x_centre(n) -Rplatform*cos(psi1);  % btm left
yplat1 = y_centre(n) -Rplatform*sin(psi1);

xplat2 = x_centre(n) -Rplatform*cos(psi2); % btm right
yplat2 = y_centre(n) -Rplatform*sin(psi2);

xplat3 = x_centre(n) -Rplatform*cos(psi3); % Top
yplat3 = y_centre(n) -Rplatform*sin(psi3);

% Link1 - Bottom Left
```

```
%Difference in x and y postions between plat and base
 delta_y1 = yplat1 - ybase1;
 delta_x1 = xplat1 - xbase1;

 angle_base_ee1 = atan2(delta_y1,delta_x1);

 %Cosine law on inner triangle
 angle_ee_joint1 = ((Sa^2 - L^2+delta_x1^2+delta_y1^2)/(2*Sa*sqrt(delta_x1^2+delta_y1^2)));
 angle_ee_joint1 = acos(angle_ee_joint1);

 q11= angle_base_ee1+angle_ee_joint1;
 q12 = angle_base_ee1-angle_ee_joint1;
 q11 = rad2deg(q11);
 q12 = rad2deg(q12);

 % Link2 - Bottom Right
 delta_y2 = yplat2-ybase2;
 delta_x2 = xplat2 -xbase2;

 angle_base_ee2 = atan2(delta_y2,delta_x2);
 angle_ee_joint2 = ((Sa^2 - L^2+delta_x2^2+delta_y2^2)/(2*Sa*sqrt(delta_x2^2+delta_y2^2)));
 angle_ee_joint2 = acos(angle_ee_joint2);

 q21= angle_base_ee2+angle_ee_joint2;
 q22 = angle_base_ee2-angle_ee_joint2;
 q21 = rad2deg(q21);
 q22 = rad2deg(q22);


 % Link 3 - Top
 delta_y3 = ybase3-yplat3;
 delta_x3 = xbase3 - xplat3;

 angle_base_ee3 = atan2(delta_y3,delta_x3) -pi;
 angle_ee_joint3 = ((Sa^2 - L^2+delta_x3^2+delta_y3^2)/(2*Sa*sqrt(delta_x3^2+delta_y3^2)));
 angle_ee_joint3 = acos(angle_ee_joint3);

 q31 = (angle_base_ee3+angle_ee_joint3);
 q32 = (angle_base_ee3-angle_ee_joint3);
 q31 = rad2deg(q31);
 q32 = rad2deg(q32);

 %Draw Links
 x_joint_11 = xbase1 + Sa*cosd(q11);
 y_joint_11 = ybase1 + Sa*sind(q11);
 x_joint_21 = xbase2 + Sa*cosd(q21);
 y_joint_21 = ybase2 + Sa*sind(q21);
 x_joint_31 = xbase3 + Sa*cosd(q31);
 y_joint_31 = ybase3 + Sa*sind(q31);


 %Draw Platform Triangle
 line([xplat1 xplat2], [yplat1 yplat2],'Color','r')
 line([xplat2 xplat3], [yplat2 yplat3],'Color','r')
 line([xplat3 xplat1], [yplat3 yplat1],'Color','r')
 hold on

 %Draw Base Triangle
 line([xbase1 xbase2], [ybase1 ybase2],'Color','k')
 line([xbase2 xbase3], [ybase2 ybase3],'Color','k')
 line([xbase3 xbase1], [ybase3 ybase1],'Color','k')

%Draw Links
 line([xbase1 x_joint_11], [ybase1 y_joint_11],'Color','b')
 line([x_joint_11 xplat1], [y_joint_11 yplat1],'Color','b')
 line([xplat1 x_centre(n)], [yplat1 y_centre(n)],'Color','b')


 line([xbase2 x_joint_21], [ybase2 y_joint_21],'Color','b')
 line([x_joint_21 xplat2], [y_joint_21 yplat2],'Color','b')
 line([xplat2 x_centre(n)], [yplat2 y_centre(n)],'Color','b')


 line([xbase3 x_joint_31], [ybase3 y_joint_31],'Color','b')
 line([x_joint_31 xplat3], [y_joint_31 yplat3],'Color','b')
 line([xplat3 x_centre(n)], [yplat3 y_centre(n)],'Color','b')


 plot([x_joint_11 x_joint_21 x_joint_31], [y_joint_11 y_joint_21 y_joint_31],'.','Color','b',
 plot([x_centre(n)], [y_centre(n)],'o','Color','k', MarkerSize= 5);


  % Add labels and a title
    xlabel('x')
    ylabel('y')

 title('Parallel Robot Plot')


 end
```

———————————— INCOMPLETE PARALLEL WORKSPACE CODE ————————————

```
clc;
clf;


%  Defined robot dimensions
Sa=170;
L=130;
Rplatform = 130;
Rbase =290;




% Define the base positions
xbase1 = 0;    % Bottom left
ybase1 = 0;

xbase2 = Rbase*sqrt(3);  % Bottom Right
ybase2 = 0;

xbase3 = xbase2/2;              % Top
ybase3 = xbase2*sind(60);
```

```
% X and Y end effector positions
x_range = 0:10:500;
y_range = 0:10:500;

% Platform orientation
alpha = 0;

%Joint angles and ee positions
joint_angles = zeros(3, length(x_range), length(y_range));
ee_positions = zeros(2, length(x_range), length(y_range));

% for n = 1:2    % 2 as specified in question

    a = deg2rad(alpha);



% angle between ee-base point & x axis
psi1= a + (pi/6);
psi2 = a +(5*pi/6);
psi3 = (a - (pi/2));




% Link1 - Bottom Left

for i = 1:length(x_range)
   for j = 1:length(y_range)
      x_centre = x_range(i);
      y_centre = y_range(j);

        xplat1 = x_centre -Rplatform*cos(psi1);  % btm left
        yplat1 = y_centre -Rplatform*sin(psi1);

        xplat2 = x_centre -Rplatform*cos(psi2); % btm right
        yplat2 = y_centre -Rplatform*sin(psi2);

        xplat3 = x_centre -Rplatform*cos(psi3); % Top
        yplat3 = y_centre -Rplatform*sin(psi3);




      % Calculate angles for joint 1
      delta_y1 = yplat1 - ybase1;
      delta_x1 = xplat1 - xbase1;
      angle_base_ee1 = atan2(delta_y1,delta_x1);
      angle_ee_joint1 = ((Sa^2 - L^2+delta_x1^2+delta_y1^2)/(2*Sa*sqrt(delta_x1^2+delta_y1^2)
      angle_ee_joint1 = acos(angle_ee_joint1);
      q11= angle_base_ee1+angle_ee_joint1;
      q12 = angle_base_ee1-angle_ee_joint1;
      q11 = rad2deg(q11);
      q12 = rad2deg(q12);
      % Store the joint angles
      joint_angles(1,i,j) = q11;
      % Calculate angles for joint 2
    delta_y2 = yplat2 - ybase2;
    delta_x2 = xplat2 - xbase2;
    angle_base_ee2 = atan2(delta_y2,delta_x2);
    angle_ee_joint2 = ((Sa^2 - L^2+delta_x2^2+delta_y2^2)/(2*Sa*sqrt(delta_x2^2+delta_y2^2)))
    angle_ee_joint2 = acos(angle_ee_joint2);
    q21= angle_base_ee2+angle_ee_joint2;
    q22 = angle_base_ee2-angle_ee_joint2;
    q21 = rad2deg(q21);
    q22 = rad2deg(q22);
    % Store the joint angles
    joint_angles(2,i,j) = q21;

    % Calculate angles for joint 3
    delta_y3 = ybase3-yplat3;
    delta_x3 = xbase3 - xplat3;
    angle_base_ee3 = atan2(delta_y3,delta_x3) -pi;
    angle_ee_joint3 = ((Sa^2 - L^2+delta_x3^2+delta_y3^2)/(2*Sa*sqrt(delta_x3^2+delta_y3^2)))
    angle_ee_joint3 = acos(angle_ee_joint3);
    q31 = (angle_base_ee3+angle_ee_joint3);
    q32 = (angle_base_ee3-angle_ee_joint3);
    q31 = rad2deg(q31);
    q32 = rad2deg(q32);
    % Store the joint angles
    joint_angles(3,i,j) = q31;

      % Store the end-effector positions
      ee_positions(1,i,j) = x_centre;
      ee_positions(2,i,j) = y_centre;
   end
end


figure;
%scatter3(ee_positions(1,:),ee_positions(2,:),joint_angles(1,:));
scatter3(ee_positions(1,:), ee_positions(2,:), zeros(1, length(x_range)*length(y_range)), '.'
xlabel('X');
ylabel('Y');
zlabel('Joint 1 angle');

 %Draw Links
 x_joint_11 = xbase1 + Sa*cosd(q11);
 y_joint_11 = ybase1 + Sa*sind(q11);
 x_joint_21 = xbase2 + Sa*cosd(q21);
 y_joint_21 = ybase2 + Sa*sind(q21);
 x_joint_31 = xbase3 + Sa*cosd(q31);
 y_joint_31 = ybase3 + Sa*sind(q31);


%  %Draw Platform Triangle
%  line([xplat1 xplat2], [yplat1 yplat2],'Color','r')
%  line([xplat2 xplat3], [yplat2 yplat3],'Color','r')
%  line([xplat3 xplat1], [yplat3 yplat1],'Color','r')
%  hold on

 %Draw Base Triangle
 line([xbase1 xbase2], [ybase1 ybase2],'Color','k')
 line([xbase2 xbase3], [ybase2 ybase3],'Color','k')
```

```
  line([xbase3 xbase1], [ybase3 ybase1],'Color','k')

%
%
%  plot([x_joint_11 x_joint_21 x_joint_31], [y_joint_11 y_joint_21 y_joint_31],'.','Color','b
%  plot([x_centre(n)], [y_centre(n)],'o','Color','k', MarkerSize= 5);


   % Add labels and a title
     xlabel('x')
     ylabel('y')

  title('Parallel Robot Plot')


%  end
```