

UNIVERSITY OF MARYLAND

ENPM 662 - FINAL REPORT

Coloring and Drawing using Kinova Jaco

Pranali Desai

116182935



Contents

Introduction	4
Motivation	4
Application	4
Specific Application for the Simulation	4
Description of the Robot	5
Assumptions	8
Scope Of Achievements	9
Replicability	11
Validation of Project	29
Conclusion and Future Scope	30
References	31

List of Figures

1	Dimensions of JACO (dimensions in mm)	5
2	Techical details of Jaco	6
3	Workspace of Jaco	6
4	Boundary Singularity	7
5	Wrist over base singularity	7
6	Wrist alignment Singularity	7
7	Jaco Manipulator Imported in Matlab	9
8	selected frames	11
9	lengths of the links	11
10	lengths of the links	12
11	dh table of Jaco	12
12	lines of the dh table in matlab	13
13	$R = Seriallink(l)$ in MATLAB	13
14	Reprenstation of manipulator	14
15	Forward kinematics Implemention	15
16	Inverse kinematicsof Jaco	16
17	Mathematical Calculation on Inverse Kinematics	17
18	Code for Trajectory Generation	18
19	Trajectory Drwan	18
20	shows code to draw a circle	19
21	drawn circle	20
22	code to draw a square	20
23	drawn square	21
24	code to colour a circle	22
25	coloured circle	22
26	code to colour a square	23
27	coloured square	23
28	dialoge box for GUI	24
29	Drag and drop of different tools	24
30	Example of .m file generated	25
31	forward kinematics GUI	25
32	inverse kinematics GUI	26
33	Drawing and Colouring Circle GUI	26
34	Drawing and Colouring Square GUI	27
35	Complete GUI	27
36	T_n^0 from Inverse kinematics	28

37	T_n^0 from Forward kinematics	29
----	---	----

Introduction

Kinova's Jaco manipulator's simulation is the central aim of the project. The simulation is being implemented in MATLAB's robotics toolbox named Peter Corke.

Motivation

Robots that assist impaired human beings in day to day life is an impeccable advancement in the field of robotics. Wheelchair robotics is growing at a great slope in the field of rehabilitation/medical robotics. Jaco is currently being used as the manipulator attached to wheelchairs to assist people. Jaco's role in this field of wheelchair robotics was the basic reason for me to select it as the manipulator of this project. While assisting impaired humans, the key important factor is to follow a trajectory to reach the end goal, since the environment around is not the same for all time and thus to avoid collisions, if the trajectory is specified the goal can be reached easily.

Application

Since this manipulator is easily compatible with robotic wheelchairs there are various applications which can assist humans to do task which they are not able to complete. The generalized application of this manipulator is to follow a particular trajectory specified in its workspace directed by the human. This application can range from cleaning a surface, or to pick up a phone, or to even write.

Specific Application for the Simulation

The specific application which my simulation will cover is a sub section of the general application of it following a trajectory. Jaco will follow a particular trajectory to draw/colour specific 2D/3D shapes. Thus to draw shapes trajectory generation is of key importance. A minimum jerk trajectory is planned to be calculated using the cost function. Thus drawing and colouring 2D/3D shape will help us realize the accuracy it can achieve to follow the general applications.

Description of the Robot

Jaco is a 6 DOF manipulator which is facilitated by the movement of the shoulder, elbow and wrist. It is of the 'RRRRRR' configuration. Being light weighted of 5.2 kg, weather resistant and made up of carbon fibre helps for easy attachment with wheelchairs. The maximum payload of the manipulator is 1.6 kg. The link lengths and basic geometry is defined in Figure 1.

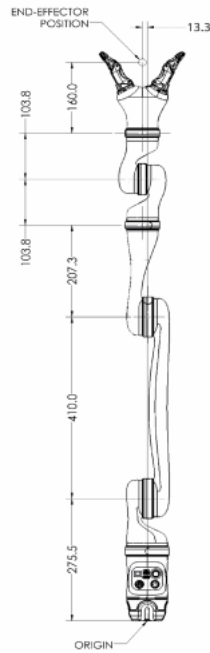


Figure 1: Dimensions of JACO (dimensions in mm)

The other few details about the manipulator are depicted in Figure (2).

Materials	Carbon fiber (links), Aluminum (actuators)
Joint range (software limitation)	± 27.7 turns
Maximum linear arm speed	20 cm / s
Power supply voltage	18 to 29 VDC
Average power	25 W (5 W in standby)
Peak power	100W
Water resistance	IPX2
Operating temperature	-10 °C to 40 °C *

Figure 2: Technical details of Jaco

The workspace of the jaco manipulator can be depicted from Figure(3).

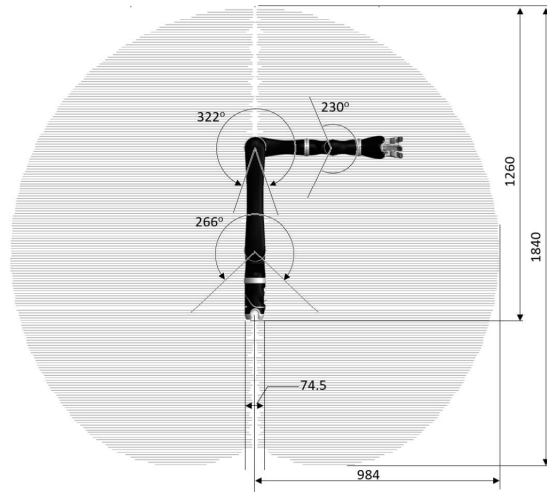


Figure 3: Workspace of Jaco

It also represents the maximum angular reachable limits of the joint. The singularities that in the physical model are depicted below.

1. Boundary Singularity- Not possible to bring elbow at 180 degree as shown in Figure(4).



Figure 4: Boundary Singularity

2. Wrist over base singularity-Not possible to bring the wrist inside a virtual cylinder of radius 15cm as show in Figure(5).



Figure 5: Wrist over base singularity

3. Wrist alignment Singularity- Not really a singularity but robot might not move as commanded 15cm as show in Figure(6).



Figure 6: Wrist alignment Singularity

Assumptions

- All the links are rigid bodies.
- Manipulator is attached to fixed base.
- All the revolute joints are at zero degrees at start of the simulation.
- The base of the manipulator is assumed to be at the origin and thus all the calculations have been done accordingly with respect to the base of the manipulator(origin).
- The trajectory to be calculated by the manipulator will always be in its workspace.
- The environment is assumed to be a 3-D plot to depict the correctness of the dimensions and location of the shapes which are being drawn and coloured.
- End effector has an object it can draw with, i.e. painting tool is attached at the end-effector.
- There is no singular condition in the robot, i.e. singularity is not analyzed in Peter corke's model and it is assumed it can reach to any joint angle which is specified.
- There is no joint limit, so all the joints can rotate freely and have full 360 degrees rotation.
- Friction is not considered in the simulation, so it is assumed that there is no friction in the system implying no friction between the links.
- Masses of the links have been not considered during simulation since the application required only kinematics, thus Center of mass does not come into play.

Scope Of Achievements

The general milestones achieved while simulating the Kinovaa Jaco Manipulator are listed and explained as follows,

1. Importing the Jaco manipulator in MATLAB.
 - The DH parameters which are derived are inserted into MATLAB's Peter Corke's toolbox.
 - The Peter Corke's toolbox uses the DH table inserted to build up the Jaco manipulator in MATLAB, as shown in figure(7)

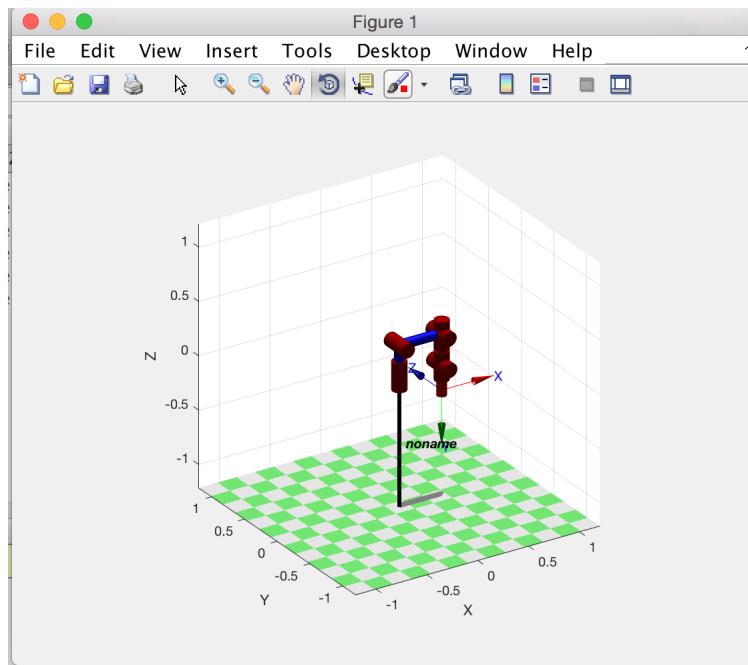


Figure 7: Jaco Manipulator Imported in Matlab

2. Deriving the Forward and Inverse Kinematics for the manipulator and implementing the same in MATLAB.
 - The DH table which is derived is used to find the T_0^n matrix thus giving us the forward kinematics.

- It had also been found using the direct function of peter corke to find forward kinematics. (The used of this has been explained in replicability and validation)
- Inverse Kinematics has been found using the kinematic decoupling method.
- Just like Forward kinematics, inverse kinematics has been found using the direct function in MATLAB's peter corke.

3. Trajectory generation

- A smoothest jerk trajectory has been generated for the manipulator to move around in the work-space.
- A cartesian trajectory is also generated but is not implemented since the application did not require it.

4. Drawing of 2D shapes with the use of trajectory generated.

- The trajectory generated is then implemented and analyzed to draw.
- The trajectory generated is implemented on the equation of the shape to be drawn.

5. Colouring of 2D shapes

- Different types of colouring methods have been found which are currently being used in the industry.
- The trajectory generation code further more is implemented on the equations derived for coloring shapes.

6. GUI is made in MATLAB to show Forward/Inverse kinematics and 2D drawing and colouring.

- A graphical user interface takes in the values of joint angles and gives out position of the manipulator(forward kinematics).
- It also takes in position and orientation and gives out manipulator joint angles(inverse kinematic).
- The GUI created helps the user to select whether to draw or colour, which shape to draw, its also its dimensions.

Replicability

The steps involved to replicate the simulation are listed below:-

1. Finding the DH table of the manipulator.
 - (a) The selected frames in order to derive the dh table are shown in figure(8).

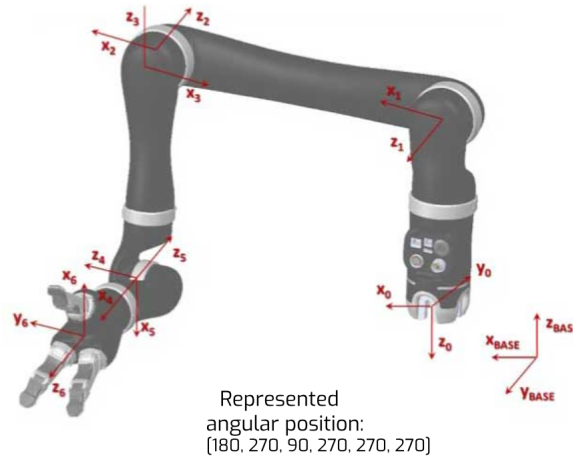


Figure 8: selected frames

- (b) The lengths of the links are depicted in Figure(10) and Figure (9)

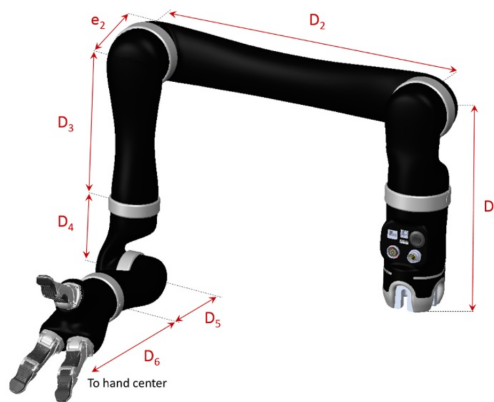


Figure 9: lengths of the links

Parameter	Description	Length (m)
D1	Base to shoulder	0.2755
D2	Upper length (shoulder to elbow)	0.4100
D3	Forearm length (elbow to wrist)	0.2073
D4	First wrist length	0.0741
D5	Second wrist length	0.0741
D6	Wrist to center of the hand	0.1600
e2	Joint 3-4 lateral offset	0.0098

Figure 10: lengths of the links

(c) Thus the derived dh table is shown in figure (11).

LINK	θ	D	α	a
1	$\theta_1 + 180$	0.2755	90	0
2	$\theta_2 - 90$	0	180	0.41
3	$\theta_3 - 90$	-0.0098	90	0
4	θ_4	-0.2814	90	0
5	θ_5	0	90	0
6	$\theta_6 + 90$	-0.2341	180	0

Figure 11: dh table of Jaco

2. Installation of Peter Corke's toolbox in MATLAB

- Download the *peter – corke* toolbox as a zip file from the official website of peter corke.
- Unpack the zip, this will create the directory (folder) *rvctools*, and which is within that the directories *robot*, *simulink*, and *common*.
- Adjust your matlab path to include *rvctools*.
- Run the startup file *rvctools/startupprvc.m* and this will place the correct directories in your MATLAB path.

3. DH table is fed into MATLAB.

- The $l(n) = \text{Link}([\theta, d, a, \alpha, offset])$ is the code to enter the dh table in MATLAB.

- (b) $l(n)$ represents the first n th row of the dh table, i.e. dh line of the the link n .
- (c) The first coloumn of the $l(n)$ is the θ value. the next is d , then the third is a , fourth is α and the last one represents the offset.
- (d) Now all the lines of the dh table has been added in MATLAB.
- (e) Figure (12) shows the same.

```

1- l(1) = Link([pi, 0.2755, 0, pi/2, 0]); %values frome 1st row of dh table
2- l(2) = Link([-pi/2, 0, 0.41, pi, 0]); %values frome 2nd row of dh table
3- l(3) = Link([-pi/2, -0.0098, 0, pi/2, 0]); %values frome 3rd row of dh table
4- l(4) = Link([0, -0.2841, 0, pi/2, 0]); %values frome 4th row of dh table
5- l(5) = Link([0, 0, 0, pi/2, 0]); %values frome 5th row of dh table
6- l(6) = Link([pi, 0.2341, 0, pi/2, 0]); %values frome 6th row of dh table
7-
8-
9-
10-
11-
12-
13-
14-

```

Figure 12: lines of the dh table in matlab

- (f) Note: When we enter the values of θ 's we only add the constant part since it is a revolute joint. The θ_n s will be input as 0 if they dont have any constant part added/subtracted from them.
4. The dh table is used to generate the stick model of the manipulator.
- (a) The code $R = Seriallink(l)$ generates a manipulator which will be referenced as R and is made using the dh table created above using $l(n)$ values as shown in Figure(13)

```

1- l(1) = Link([pi, 0.2755, 0, pi/2, 0]); %values frome 1st row of dh table
2- l(2) = Link([-pi/2, 0, 0.41, pi, 0]); %values frome 2nd row of dh table
3- l(3) = Link([-pi/2, -0.0098, 0, pi/2, 0]); %values frome 3rd row of dh table
4- l(4) = Link([0, -0.2841, 0, pi/2, 0]); %values frome 4th row of dh table
5- l(5) = Link([0, 0, 0, pi/2, 0]); %values frome 5th row of dh table
6- l(6) = Link([pi, 0.2341, 0, pi/2, 0]); %values frome 6th row of dh table
7- R = SerialLink(l);
8- q=[0, 0, 0, 0, 0, 0];
9- R.plot(q);
10-
11-
12-
13-
14-

```

Figure 13: $R = Seriallink(l)$ in MATLAB

5. Plotting the manipulator, using joint angles.
 - (a) $R.plot(q)$ function, which is an inbuilt function of peter corke is used to plot the manipulator at various joint angles.
 - (b) R represent the manipulator.
 - (c) q is a 1x6 matrix which represent the values of joint angles. It has 6 coloums since there are 6 joints of the manipulator.
 - (d) The figure(14) represent the manipulator when all the joint angles are zero. This is the start point of the simulation.

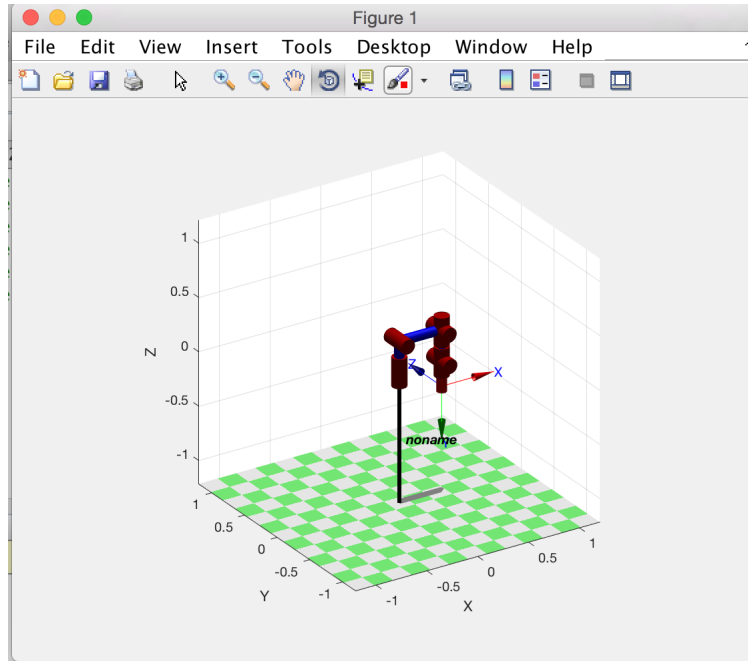
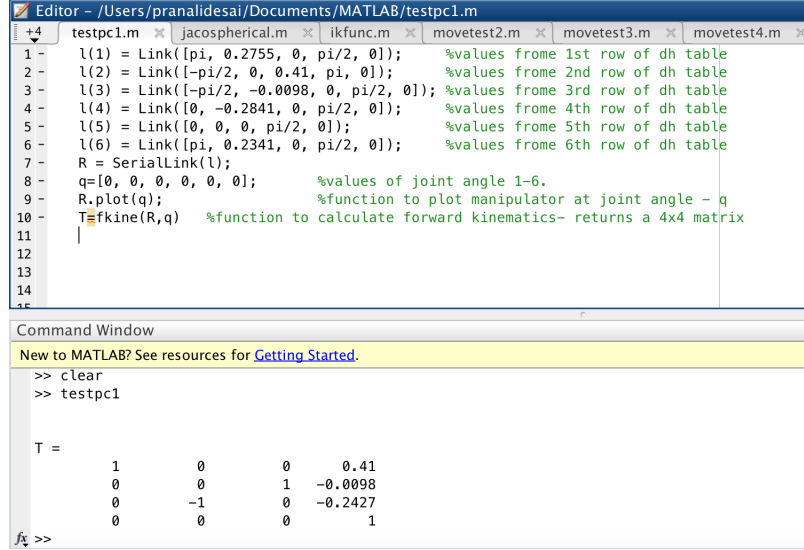


Figure 14: Reprenstation of manipulator

6. Forward Kinematics function is now implemented.
 - (a) We use the $fkine(R, q)$ MATLAB function - This function is an inbuilt peter corke function which takes in the matrix that represents joint angles(explained above) and returns the T_n^0 matrix.
 - (b) This T_n^0 matrix's last column represents the position of end effector and the rotational matrix represents the orientation of end-effector.

- (c) To test the function, we take the values of joint angles $q = 0$ and feed in the values to the `fkine` function.
- (d) Figure(15) represents the matrix T_n^0 which we get after running the code.



The image shows a MATLAB Editor window with a script named `testpc1.m`. The script defines six links using the `Link` function with DH parameters, serializes them into a robot object `R`, and sets joint angles `q` to zero. It then calls `R.plot(q)` and `fkine(R,q)`. The Command Window shows the output of `fkine`, which is a 4x4 transformation matrix T .

```

1- l(1) = Link([pi, 0.2755, 0, pi/2, 0]); %values from 1st row of dh table
2- l(2) = Link([-pi/2, 0, 0.41, pi, 0]); %values from 2nd row of dh table
3- l(3) = Link([-pi/2, -0.0098, 0, pi/2, 0]); %values from 3rd row of dh table
4- l(4) = Link([0, -0.2841, 0, pi/2, 0]); %values from 4th row of dh table
5- l(5) = Link([0, 0, 0, pi/2, 0]); %values from 5th row of dh table
6- l(6) = Link([pi, 0.2341, 0, pi/2, 0]); %values from 6th row of dh table
7- R = SerialLink(l);
8- q=[0, 0, 0, 0, 0, 0]; %values of joint angle 1-6.
9- R.plot(q); %function to plot manipulator at joint angle - q
10- T=fkine(R,q) %function to calculate forward kinematics- returns a 4x4 matrix
11-
12-
13-
14-
15-

```

Command Window

New to MATLAB? See resources for [Getting Started](#).

```

>> clear
>> testpc1

T =

     1     0     0     0.41
     0     0     1    -0.0098
     0    -1     0    -0.2427
     0     0     0     1

```

Figure 15: Forward kinematics Implementation

7. Inverse kinematics function in matlab.

- (a) The `R.ikunc(T)` is the inbuilt matlab function to calculate the joint angles when we feed in the T_n^0 matrix.
- (b) Figure (16) represents the use of `ikunc(T)` function.

The image shows a MATLAB Editor window with a script named `testpc1.m`. The script defines six links for a robotic arm using the Denavit-Hartenberg (DH) convention. The links are defined as follows:

- Link 1: `l(1) = Link([pi, 0.2755, 0, pi/2, 0]);` (values from 1st row of DH table)
- Link 2: `l(2) = Link([-pi/2, 0, 0.41, pi, 0]);` (values from 2nd row of DH table)
- Link 3: `l(3) = Link([-pi/2, -0.0098, 0, pi/2, 0]);` (values from 3rd row of DH table)
- Link 4: `l(4) = Link([0, -0.2841, 0, pi/2, 0]);` (values from 4th row of DH table)
- Link 5: `l(5) = Link([0, 0, 0, pi/2, 0]);` (values from 5th row of DH table)
- Link 6: `l(6) = Link([pi, 0.2341, 0, pi/2, 0]);` (values from 6th row of DH table)

The script then creates a serial link model `R = SerialLink(l);`, sets initial joint angles `q=[0, 0, 0, 0, 0, 0];`, and defines functions for plotting the manipulator (`R.plot(q);`), calculating forward kinematics (`T=fkine(R,q);`), and calculating inverse kinematics (`q1=R.ikunc(T);`).

The Command Window shows the execution of `testpc1`, resulting in the joint angles `q1 = [0, 0, 0, 0, 0, 0]`.

Figure 16: Inverse kinematics of Jaco

(c) Also note if we feed in the same T matrix back to *ikunc* we get the same joint angles which we had input to find the forward kinematics.

8. Mathematical Calculation of Inverse Kinematics

- Function $ik = pos(x, y, z, r, p, w, R)$ is generated in MATLAB.
- x, y, z represent the position of the end effector.
- r, p, w represent the roll, pitch and yaw angles of the manipulator.
- R represents the manipulator whose inverse needs to be calculated.
- With the help of roll, pitch and yaw the rotational part of T_n^0 is calculated, since it is the Euler angle rotation in x, y and z axis. RYW is the 3×3 matrix which is the multiplication of Euler angles rotation in z, y , and x axis.
- x, y, z represent the translation part of T_n^0 . Matrix P is the 3×1 matrix which represents the same.
- Thus the calculated

$$T_n^0 = \begin{bmatrix} RYW & P \\ 0 & 1 \end{bmatrix}$$

- (h) T_n^0 is now fed to the *ikunc* function to find out the joint angles.
- (i) Figure (17) represents the code written in matlab for the same.

```

Editor - /Users/pranalidesai/Documents/MATLAB/testpc1.m
testpc1.m  jacospherical.m  ikfunc.m  movetest2.m  movetest3.m  movetest4.m
1- l(1) = Link([pi, 0.2755, 0, pi/2, 0]);
2- l(2) = Link([-pi/2, 0, 0.41, pi, 0]);
3- l(3) = Link([-pi/2, -0.0098, 0, pi/2, 0]);
4- l(4) = Link([0, -0.2841, 0, pi/2, 0]);
5- l(5) = Link([0, 0, 0, pi/2, 0]);
6- l(6) = Link([pi, 0.2341, 0, pi/2, 0]);
7- R = SerialLink(l);
8- q=pos(0.5,0,0,0,0,0,R) % example values passed in the function
9- R.plot(q); %plots the manipulator at the joint values found
10- function ik = pos(x,y,z,r,p,w,R)
11- t1=r*(pi/180); %conversion of degree to radian
12- t2=p*(pi/180); %conversion of degree to radian
13- t3=w*(pi/180); %conversion of degree to radian
14- rx=rotx(t3); % euler ange rotation in x - yaw
15- ry=roty(t2); % euler ange rotation in y - pitch
16- rz=rotz(t1); % euler ange rotation in z - roll
17- ryw=rx*ry*rz; % multiplication or rotations
18- o = [x;y;z]; % position vector
19- T = [ryw o; 0 0 0 1] % T matrix
20- ik=R.ikunc(T);

Command Window
New to MATLAB? See resources for Getting Started.
T =

    1.0000         0         0    0.5000
         0    1.0000         0         0
         0         0    1.0000         0
         0         0         0    1.0000

q =

fx -0.4201 -0.0734 -0.6013 -1.2025 -1.7777 -1.0816

```

Figure 17: Mathematical Calculation on Inverse Kinematics

9. Trajectory generation peter corke.

- The inbuilt function `jtraj(q1,q2,N)` is used to generate the smoothest trajectory between two given joint angle matrices - q_1 and q_2 .
- N represents the number of steps it takes to move from one joint angle to the other.
- This function returns a $N \times 6$ matrix representing the iterating values of joint angles.
- The rows of the $N \times 6$ matrix are fed one by one in the forward kinematics function in order to extract the position and orientation of the manipulator by using a for loop.
- With the help of the positions and joint angles achieved, the manipulator is made to draw the `path(trajectory)`.

(f) Figure(18) represents the code for the same.

```

Editor - /Users/pranalidesai/Documents/MATLAB/movetest2.m*
testpc1.m x jacospherical.m x ikfunc.m x movetest2.m* x movetest3.m x movetest4.m x
1 - l(1) = Link([pi, 0.2755, 0, pi/2, 0]);
2 - l(2) = Link([-pi/2, 0, 0.41, pi, 0]);
3 - l(3) = Link([-pi/2, -0.0098, 0, pi/2, 0]);
4 - l(4) = Link([0, -0.2841, 0, pi/2, 0]);
5 - l(5) = Link([0, 0, 0, pi/2, 0]);
6 - l(6) = Link([pi, 0.2341, 0, pi/2, 0]);
7 - R = SerialLink(l);
8 - q1=[0,0,0,0,0,0]*(pi/180); % initial angle
9 - q2=[0,0,-180,0,0,0]*(pi/180); % final angle
10 - T1=fkine(R,q1); % initial position T matrix
11 - T2=fkine(R,q2); % final position T matrix
12 - q=jtraj(q1,q2,20); % finds the iterating angles to generate trajectory
13 - for i=1:20
14 -     T=fkine(R,q(i,1:6)); %extracts the T matrix
15 -     plot3(T.t(1),T.t(2),T.t(3),'*r'); %plots the trajectory points
16 -     hold on;
17 -     R.plot(q(i,1:6)); % plots the manipulator at iterating angles
18 - end

```

Figure 18: Code for Trajectory Generation

(g) Figure(19) shows trajectory generated for code shown in figure(18).

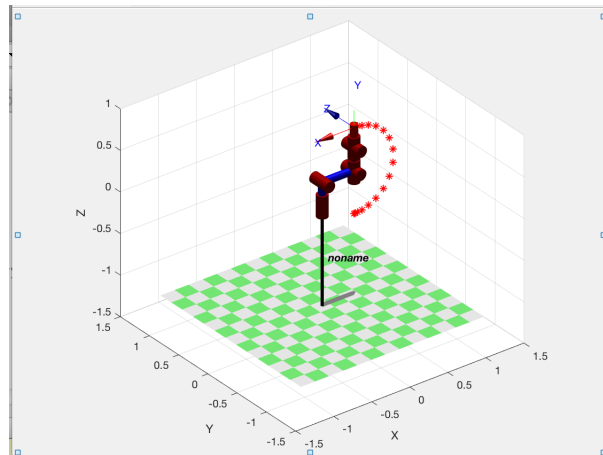
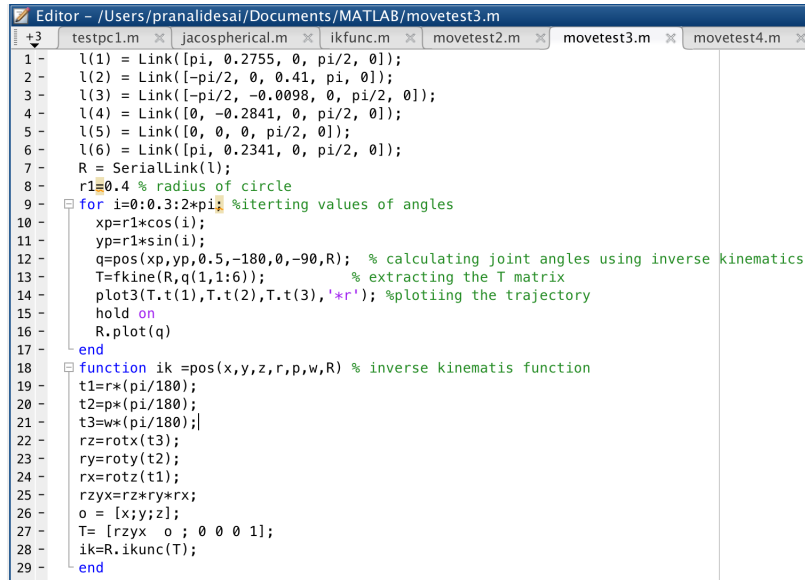


Figure 19: Trajectory Drwan

(h) Clearly from the figure we observe that only one joint angle is moved and if we notice the input only the third joint had been moved by -180 degrees

10. To drawa circle/square.

- (a) The concept used to draw comes from trajectory generation.
- (b) The equation of a circle(shape) is used to realize the various points needed for the manipulator to hover on.
- (c) These points are then fed into the inverse kinematics function which outputs the joint angles.
- (d) Thus for each and every point joint angles for the manipulator is calculated, giving out Mx6 matrix, where M is the number of points.
- (e) Figure(20) shows code to draw a circle of radius = r1



```

Editor - /Users/pranalidesai/Documents/MATLAB/movetest3.m
+3 testpc1.m x jacospherical.m x ikfunc.m x movetest2.m x movetest3.m x movetest4.m x
1 l(1) = Link([pi, 0.2755, 0, pi/2, 0]);
2 l(2) = Link([-pi/2, 0, 0.41, pi, 0]);
3 l(3) = Link([-pi/2, -0.0098, 0, pi/2, 0]);
4 l(4) = Link([0, -0.2841, 0, pi/2, 0]);
5 l(5) = Link([0, 0, 0, pi/2, 0]);
6 l(6) = Link([pi, 0.2341, 0, pi/2, 0]);
7 R = SerialLink(l);
8 r1=0.4 % radius of circle
9 for i=0:0.3:2*pi %iterating values of angles
10     xp=r1*cos(i);
11     yp=r1*sin(i);
12     q=pos(xp,yp,0.5,-180,0,-90,R); % calculating joint angles using inverse kinematics
13     T=fkine(R,q(1,1:6)); % extracting the T matrix
14     plot3(T.t(1),T.t(2),T.t(3),'*r'); %plotting the trajectory
15     hold on
16     R.plot(q)
17 end
18 function ik =pos(x,y,z,r,p,w,R) % inverse kinematics function
19 t1=r*(pi/180);
20 t2=p*(pi/180);
21 t3=w*(pi/180);
22 rz=rotz(t3);
23 ry=roty(t2);
24 rx=rotz(t1);
25 rzyx=rz*ry*rx;
26 o = [x;y;z];
27 T= [rzyx o ; 0 0 0 1];
28 ik=R.ikunc(T);
29 end

```

Figure 20: shows code to draw a circle

- (f) Figure(21) shows the drawn circle.

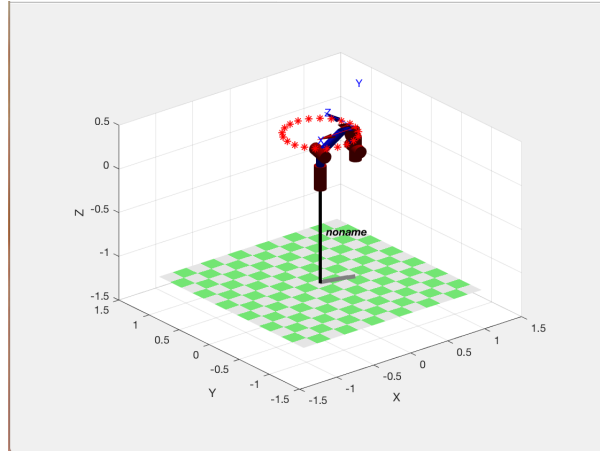


Figure 21: drawn circle

(g) Figure(22) shows code to draw a square of side = s1

```

Editor - /Users/pranlidesai/Documents/MATLAB/movetest4.m*
testpcl.m  jacospherical.m  ikfunc.m  movetest2.m  movetest3.m  movetest4.m*  colourtest1
8  s1=0.8 % length of side of a square
9  x1=s1/2; x2=-s1/2; % calculating x the coordinates of square
10 y1=s1/2; y2=-s1/2; % calculating y the coordinates of square
11 n=round((y1-y2)/0.1+(1)); % number of iterations
12 xp=x1:-0.1:x2; yp=y1:-0.1:y2;
13 for i=1:n
14     q=pos(xp(i),yp(i),0.8,-180,0,-90,R); %calculating joint angles using inverse kinematics
15     T=fkine(R,q(1,1:6)); %extracting the T matrix
16     plot3(T.t(1),T.t(2),T.t(3),'r*'); %plotting the trajectory
17     hold on;
18     R.plot(q);
19 end
20 for i=1:n
21     q=pos(xp(n),yp(i),0.8,-180,0,-90,R); %calculating joint angles using inverse kinematics
22     T=fkine(R,q(1,1:6)); %extracting the T matrix
23     plot3(T.t(1),T.t(2),T.t(3),'r*'); %plotting the trajectory
24     hold on;
25     R.plot(q);
26 end
27 for i=1:n
28     q=pos(xp(n+1-i),yp(n),0.8,-180,0,-90,R); %calculating joint angles using inverse kinematics
29     T=fkine(R,q(1,1:6)); %extracting the T matrix
30     plot3(T.t(1),T.t(2),T.t(3),'r*'); %plotting the trajectory
31     hold on;
32     R.plot(q);
33 end
34 for i=1:n
35     q=pos(xp(1),yp(n+1-i),0.8,-180,0,-90,R); %calculating joint angles using inverse kinematics
36     T=fkine(R,q(1,1:6)); %extracting the T matrix
37     plot3(T.t(1),T.t(2),T.t(3),'r*'); %plotting the trajectory
38     hold on;
39     R.plot(q);
40 end

```

Figure 22: code to draw a square

(h) Figure(23) shows the drawn square.

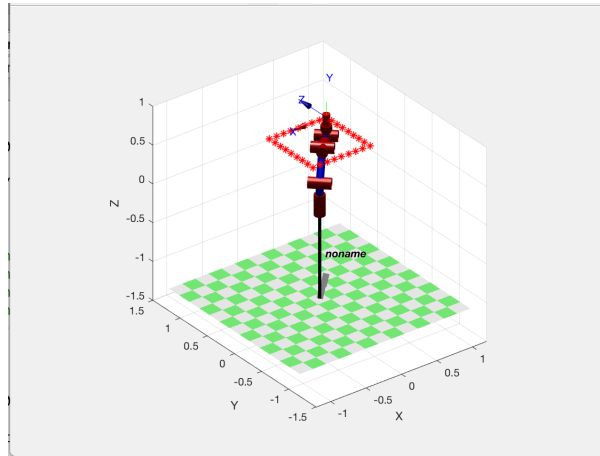


Figure 23: drawn square

11. To colour a shape

- (a) The concept used to colour the shape comes from decreasing order of the dimensions of the shape.
- (b) Thus it starts with the given radius or side length and then decreases by minute value and runs in the loop to draw for that dimension.
- (c) So the manipulator colours the shape by drawing multiples of the shape of decreasing dimension.
- (d) Figure(24) shows code to colour a circle of radius = $r1$

```

Editor - /Users/pranlidesai/Documents/MATLAB/colourtest1.m
testpc1.m x jacospherical.m x ikfunc.m x movetest2.m x movetest3.m x movetest4.m x colc
1 - l(1) = Link([pi, 0.2755, 0, pi/2, 0]);
2 - l(2) = Link([-pi/2, 0, 0.41, pi, 0]);
3 - l(3) = Link([-pi/2, -0.0098, 0, pi/2, 0]);
4 - l(4) = Link([0, -0.2841, 0, pi/2, 0]);
5 - l(5) = Link([0, 0, 0, pi/2, 0]);
6 - l(6) = Link([pi, 0.2341, 0, pi/2, 0]);
7 - R = SerialLink(l);
8 - r2=0.4 % radius of circle
9 - for r1=r2:-0.05:0 % decreasing values of radiue
10 - for i=0:0.2:2*pi % iterting values of angles
11 -     xp=r1*cos(i)
12 -     yp=r1*sin(i)
13 -     q=pos(0.5,yp,xp,180,0,90,R) % calculating joint angles using inverse kinematics
14 -     T=fkine(R,q(1:6)); % extracting the T matrix
15 -     plot3(T.t(1),T.t(2),T.t(3),'*r'); %plotiing the trajectory
16 -     hold on
17 -     R.plot(q) % plots the manipulator at iterating angles
18 - end
19 - end
20 - R.plot([0,0,0,0,0,0]);
21 - function ik =pos(x,y,z,r,p,w,R) % inverse kinematis function
22 -     t1=r*(pi/180);
23 -     t2=p*(pi/180);
24 -     t3=w*(pi/180);
25 -     rz=rotx(t3);
26 -     ry=roty(t2);
27 -     rx=rotz(t1);
28 -     rzyx=rz*ry*rx;
29 -     o = [x;y;z];
30 -     T= [rzyx o; 0 0 0 1];
31 -     ik=R.ikunc(T);
32 - end

```

Figure 24: code to colour a circle

(e) Figure(25) shows the coloured circle.

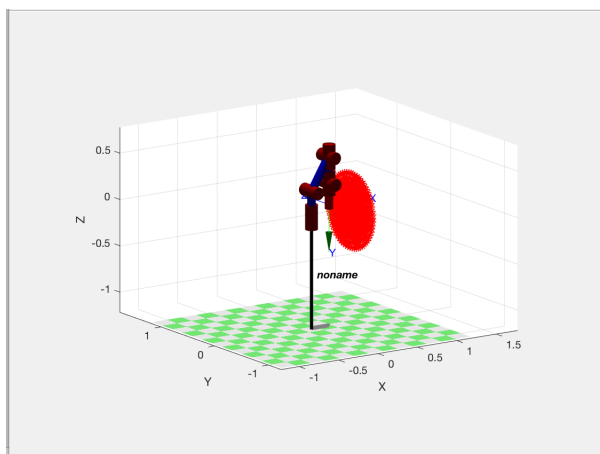


Figure 25: coloured circle

(f) Figure(26) shows code to colour a square of side = s1

```

Editor - /Users/pranalidesai/Documents/MATLAB/colourtest2.m*
+3 testpc1.m x jacospherical.m x ikfunc.m x movetest2.m x movetest3.m x movetest4.m x colourtest2.m
9 - for s1=s1:-1:0
10 - x1=s1/2;x2=-s1/2; % calculating x the coordinates of square
11 - y1=s1/2;y2=-s1/2; % calculating y the coordinates of square
12 - n=round((y1-y2)/1+(1)); % number of iterations
13 - xp1=x1:-1:x2; xp = xp1/10;
14 - yp1=y1:-1:y2; yp = yp1/10;
15 - for i=1:1:n % to draw the 1st side
16 - q=pos(xp(i),yp(1),0.8,-180,0,-90,R); %calculating joint angles using inverse kinematics
17 - T=fkine(R,q(1,1:6)); %extracting the T matrix
18 - plot3(T.t(1),T.t(2),T.t(3),'r*');
19 - hold on;
20 - R.plot(q);
21 - end
22 - for i=1:1:n % to draw the 2nd side
23 - q=pos(xp(n),yp(i),0.8,-180,0,-90,R); %calculating joint angles using inverse kinematics
24 - T=fkine(R,q(1,1:6)); %extracting the T matrix
25 - plot3(T.t(1),T.t(2),T.t(3),'r*');
26 - hold on;
27 - R.plot(q);
28 - end
29 - for i=1:1:n % to draw the 3rd side
30 - q=pos(xp(n+1-i),yp(n),0.8,-180,0,-90,R); %calculating joint angles using inverse kinematics
31 - T=fkine(R,q(1,1:6)); %extracting the T matrix
32 - plot3(T.t(1),T.t(2),T.t(3),'r*');
33 - hold on;
34 - R.plot(q);
35 - end
36 - for i=1:1:n % to draw the 4th side
37 - q=pos(xp(1),yp(n+1-i),0.8,-180,0,-90,R); %calculating joint angles using inverse kinematics
38 - T=fkine(R,q(1,1:6)); %extracting the T matrix
39 - plot3(T.t(1),T.t(2),T.t(3),'r*');
40 - hold on;R.plot(q);

```

Figure 26: code to colour a square

(g) Figure(27) shows the coloured square.

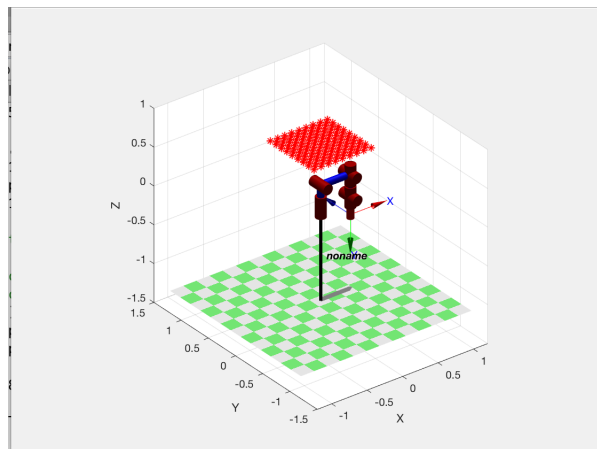


Figure 27: coloured square

12. Creating a GUI in matlab

(a) Run the "guide" code in the command line of MATLAB.

- (b) This will open up a dialog box from which you can either open a window to create a new GUI, or open an existing GUI as shown in Figure ().

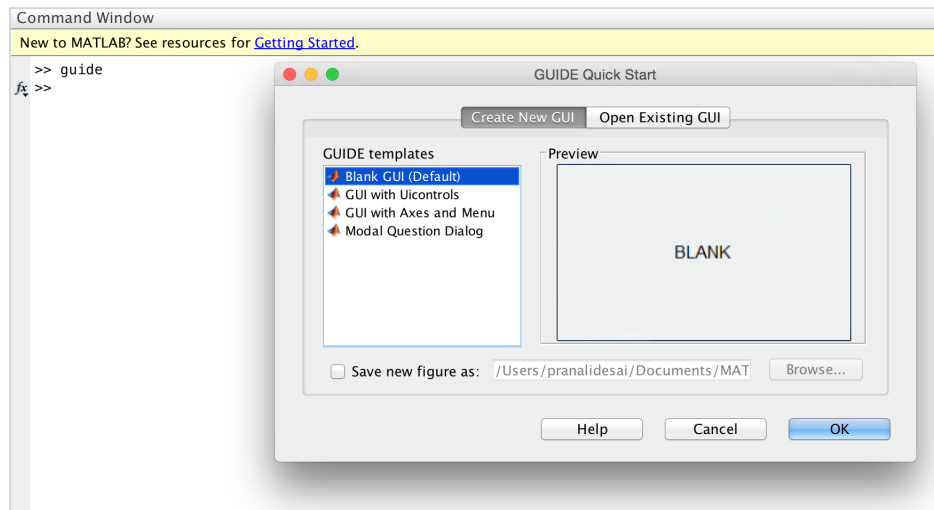


Figure 28: dialog box for GUI

- (c) Once you open to create a new GUI, you can simply drag different tools directly and save the GUI as shown in figure(28).

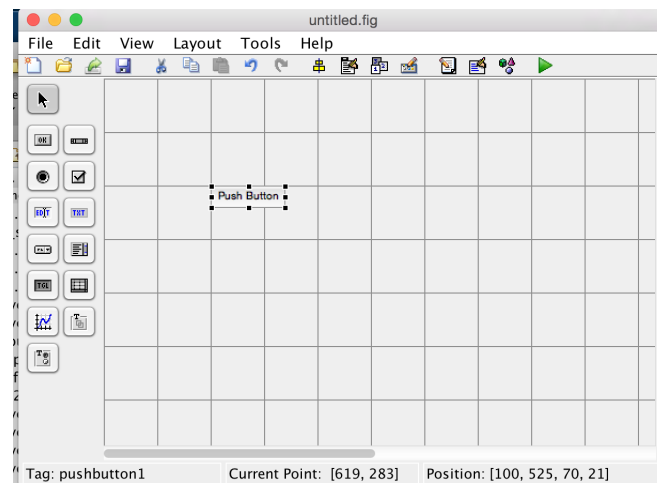


Figure 29: Drag and drop of different tools

- (d) Once the GUI is saved its .m file is directly created by matlab.
- (e) This is where the code as to what the tool is supposed to do goes.
- (f) Example: If the GUI has a button, the the callback function in the .m file states what does the button do, as shown in figure (30).

```

75 % --- Executes on button press in pushbutton1.
76 function pushbutton1_Callback(hObject, eventdata, handles)
77 % hObject    handle to pushbutton1 (see GCBO)
78 % eventdata  reserved - to be defined in a future version of MATLAB
79 % handles    structure with handles and user data (see GUIDATA)
80
81 %% code which executes on push button goes here%
82
83

```

Figure 30: Example of .m file generated

- (g) This means that whenever the button is pressed the codes in the callback function run thus executing the need of the button.

13. Created GUI for this project.

- (a) Forward Kinematics - There are 6 text box, which take in values of joint angles and present the robot at those values in its work space.
 - i. The code includes calculating the forward kinematics and plotting the manipulator at those angles as shown in above steps.
 - ii. The code is simply put under the button for forward kinematics as shown in Figure (31).

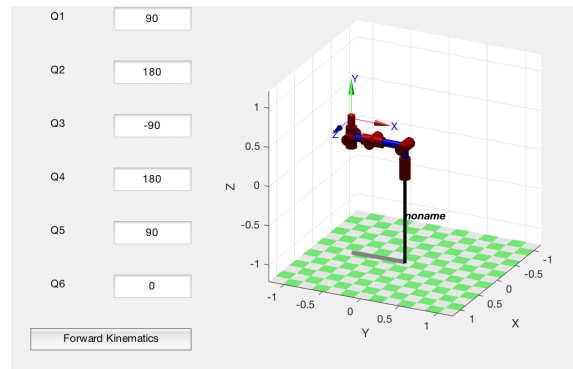
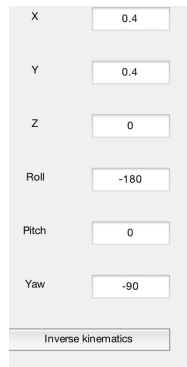


Figure 31: forward kinematics GUI

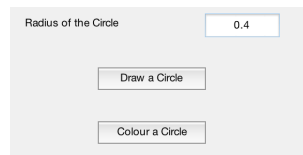
- (b) Inverse Kinematics - There are 6 text box, those take in values of position and orientation of the end effector which calculate the respective joint angles of manipulator and present the robot at those values in its work space.
- The code includes calculating the inverse kinematics and plotting the manipulator at those values as shown in above steps.
 - The code is simply put under the button for inverse kinematics as shown in Figure (32).



X	0.4
Y	0.4
Z	0
Roll	-180
Pitch	0
Yaw	-90
Inverse kinematics	

Figure 32: inverse kinematics GUI

- (c) Drawing/Colouring a shape - There is one text box which takes in the value of the dimension for the shape.
- The code includes calculating the inverse kinematics for the particular points found from the equation of the shape.
 - The code is simply put under the button for drawing/colouring as shown in Figure (33) and Figure(34).



Radius of the Circle	0.4
Draw a Circle	
Colour a Circle	

Figure 33: Drawing and Colouring Circle GUI

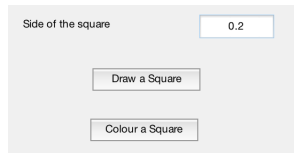


Figure 34: Drawing and Colouring Square GUI

- (d) Figure(35) Represents the full GUI created for calculating various parts of the project.

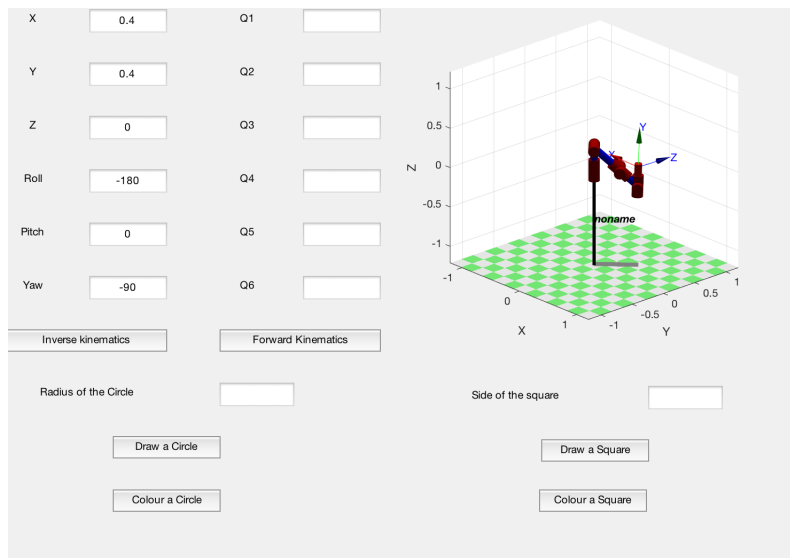
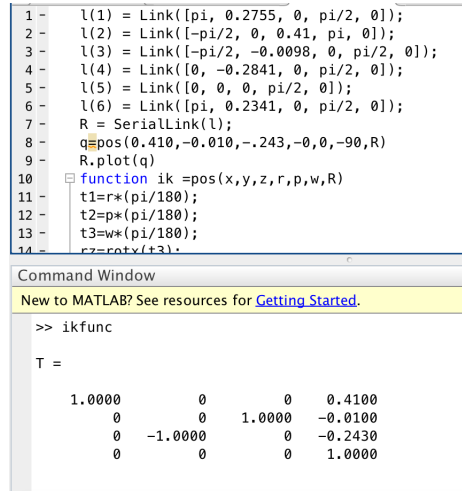


Figure 35: Complete GUI

Validation of Project

1. The idea behind the validation of my project is by checking T_n^0 matrix.
 - (a) We check the T_n^0 matrix which we calculated mathematically in MATLAB and compare it with the T_n^0 matrix which MATLAB generates by using the fkine function.
 - (b) We feed in values for the manipulator position and orientation and achieved the T_n^0 as explained in 8.(g) (Note: This is a mathematical calculation and not a matlab inbuilt function)



```

1 - l(1) = Link([pi, 0.2755, 0, pi/2, 0]);
2 - l(2) = Link([-pi/2, 0, 0.41, pi, 0]);
3 - l(3) = Link([-pi/2, -0.0098, 0, pi/2, 0]);
4 - l(4) = Link([0, -0.2841, 0, pi/2, 0]);
5 - l(5) = Link([0, 0, 0, pi/2, 0]);
6 - l(6) = Link([pi, 0.2341, 0, pi/2, 0]);
7 - R = SerialLink(l);
8 - q=pos(0.410,-0.010,-.243,-0,0,-90,R);
9 - R.plot(q)
10 - function ik =pos(x,y,z,r,p,w,R)
11 - t1=r*(pi/180);
12 - t2=p*(pi/180);
13 - t3=w*(pi/180);
14 - rz=roty(t3);

```

Command Window

New to MATLAB? See resources for [Getting Started](#).

```

>> ikfunc

T =

    1.0000         0         0     0.4100
         0         0     1.0000    -0.0100
         0    -1.0000         0    -0.2430
         0         0         0     1.0000

```

Figure 36: T_n^0 from Inverse kinematics

- (c) The matlab inbuilt function iknine helps us to achieved the angles from the T_n^0 matrix.
- (d) So we feed in a value of angles which the manipulator needs to reach from the above point and through the fkine function we get back the T_n^0 as explained in 6.(b).

The image shows a MATLAB script editor with four tabs: testpcl.m, jacospherical.m, ikfunc.m, and exam26i. The script in testpcl.m defines six links and a serial chain. The Command Window shows the output of the ikfunc function, which is a 4x4 transformation matrix T.

```

1 - l(1) = Link([pi, 0.2755, 0, pi/2, 0]);
2 - l(2) = Link([-pi/2, 0, 0.41, pi, 0]);
3 - l(3) = Link([-pi/2, -0.0098, 0, pi/2, 0]);
4 - l(4) = Link([0, -0.2841, 0, pi/2, 0]);
5 - l(5) = Link([0, 0, 0, pi/2, 0]);
6 - l(6) = Link([pi, 0.2341, 0, pi/2, 0]);
7 - R = SerialLink(l);
8 - q2=[0, 0, 0, 0, 0, 0]*(pi/180);
9 - T1=fkine(R,q2)
10 - %R.teach()
11
12
13

```

Command Window

New to MATLAB? See resources for [Getting Started](#).

```

>> ikfunc

T =

    1.0000         0         0     0.4100
         0         0     1.0000    -0.0100
         0    -1.0000         0    -0.2430
         0         0         0     1.0000

```

Figure 37: T_n^0 from Forward kinematics

- (e) We now notice that both the T matrix which we get from inverse kinematics and forward kinematics are same.
- (f) Thus validating the project.

Conclusion and Future Scope

Thus as proposed we see that the manipulator reaches the position when the angles have been prescribed under forward kinematics. It also works under inverse kinematics when the position and orientation has been fed in. The colouring and drawing of shapes in MATLAB workspace has also been achieved.

The future scope for the project is implementing the same into VREP works-pace and implementing on the Jaco manipulator in the live feed. This can be achieved by working on the assumptions made and implementing them. Few controllers which convert the m script to c code for implementation can also be used to make the system work in live environment.

References

- [1] KINOVA™ Ultra lightweight robotic arm user guide.
- [2] Peter Corke Introduction.
Available at: <http://petercorke.com/wordpress/toolboxes/robotics-toolbox>.
- [3] P.I. Corke, “A Robotics Toolbox for MATLAB”, IEEE Robotics and Automation Magazine, Volume 3(1), March 1996, pp. 24-32.
- [4] P.I. Corke, A computer tool for simulation and analysis: the Robotics Toolbox for MATLAB, Proceedings of the 1995 National Conference of the Australian Robot Association, Melbourne, Australia, pp 319-330, July 1995.
- [5] P.I. Corke, “MATLAB toolboxes: robotics and vision for students and teachers”, IEEE Robotics and Automation Magazine, Volume 14(4), December 2007, pp. 16-17.