Rohit Raibagkar

ECE 5425

Fall 2016

Report of Project Assignment 3

College of Engineering

Declaration : The project and the report are my own work

Contents

Result

Table of Timing tests using tic-toc.

Details	mtraj@tpoly Interpolation	mtraj@lspb Interpolation	Only tpoly Interpolation	Only lspb Interpolation	No Interpolation
D* Algorithm	48.663746 sec	52.347124 sec	19.87564 sec	25.63249 sec	14.7741 sec
PRM Algorithm	95.6642 sec	89.22415 sec	62.5547 sec	46.35487 sec	32.4659 sec

Time mentioned above is in seconds.

The time is always liable to change because PRM and D* planning takes time that is not fixed. Sometimes it may take long time, it may take short time depending upon distance between nodes.

From the discussion above, we can conclude that mtraj takes longest time, with both D* and PRM.

But, trajectory drawn by mtraj are much smoother.

Figure below shows the trajectory generated in PRM using mtraj@tpoly and mtraj@lspb.

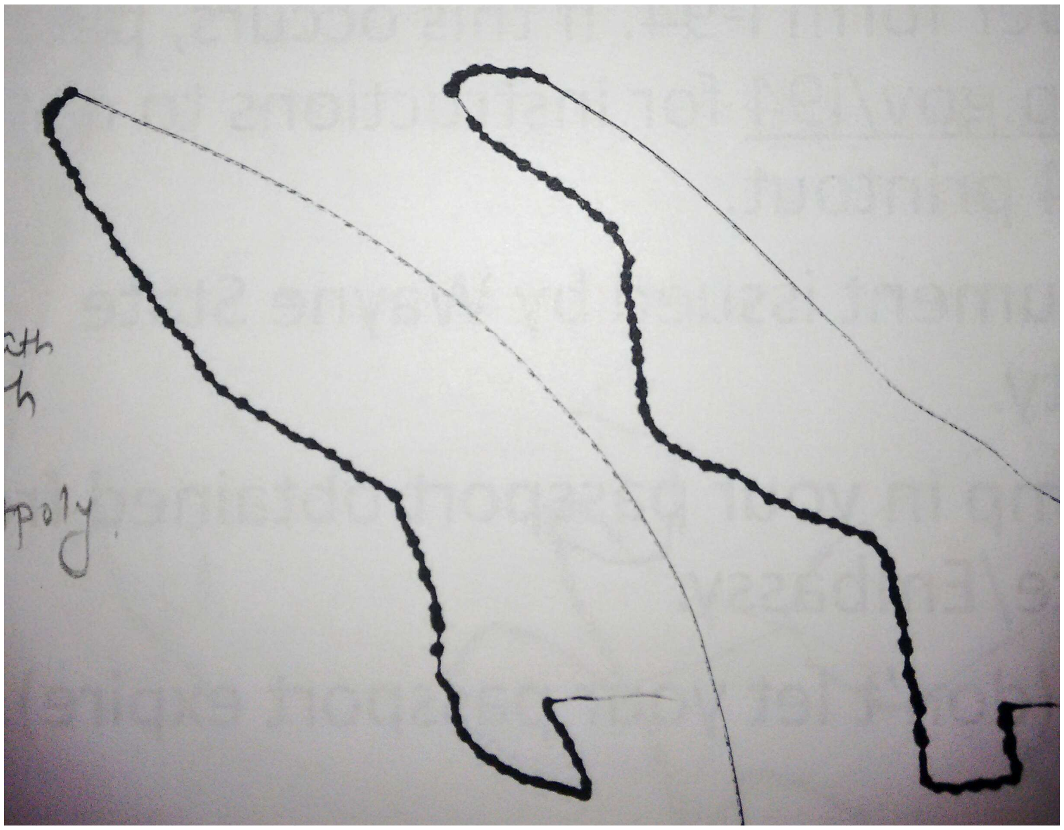
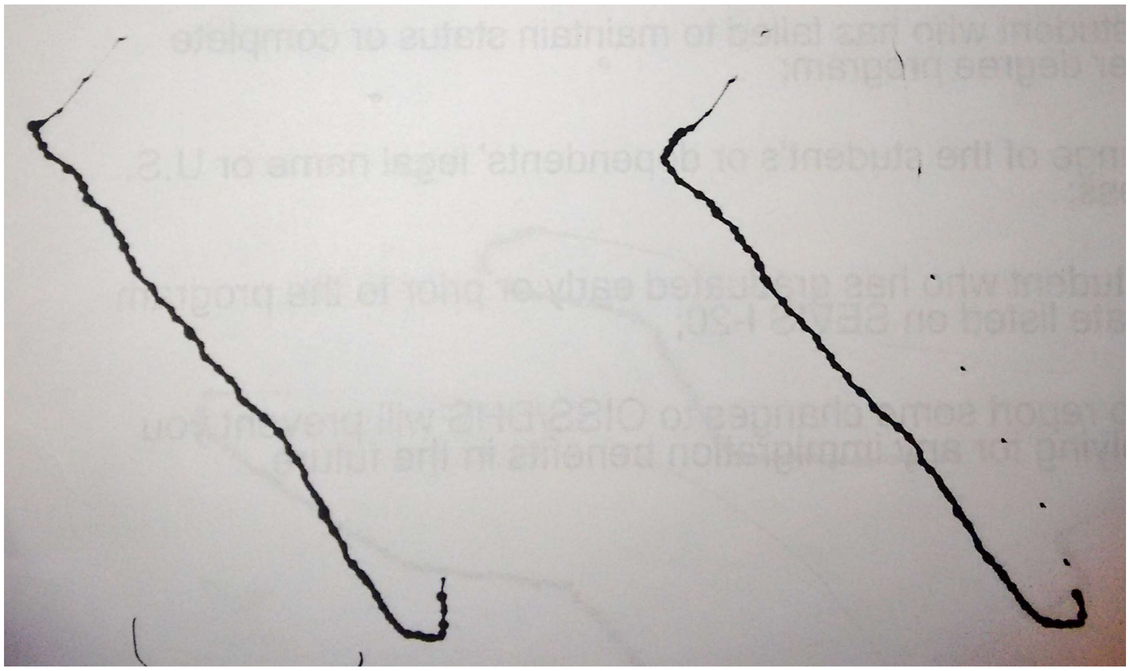
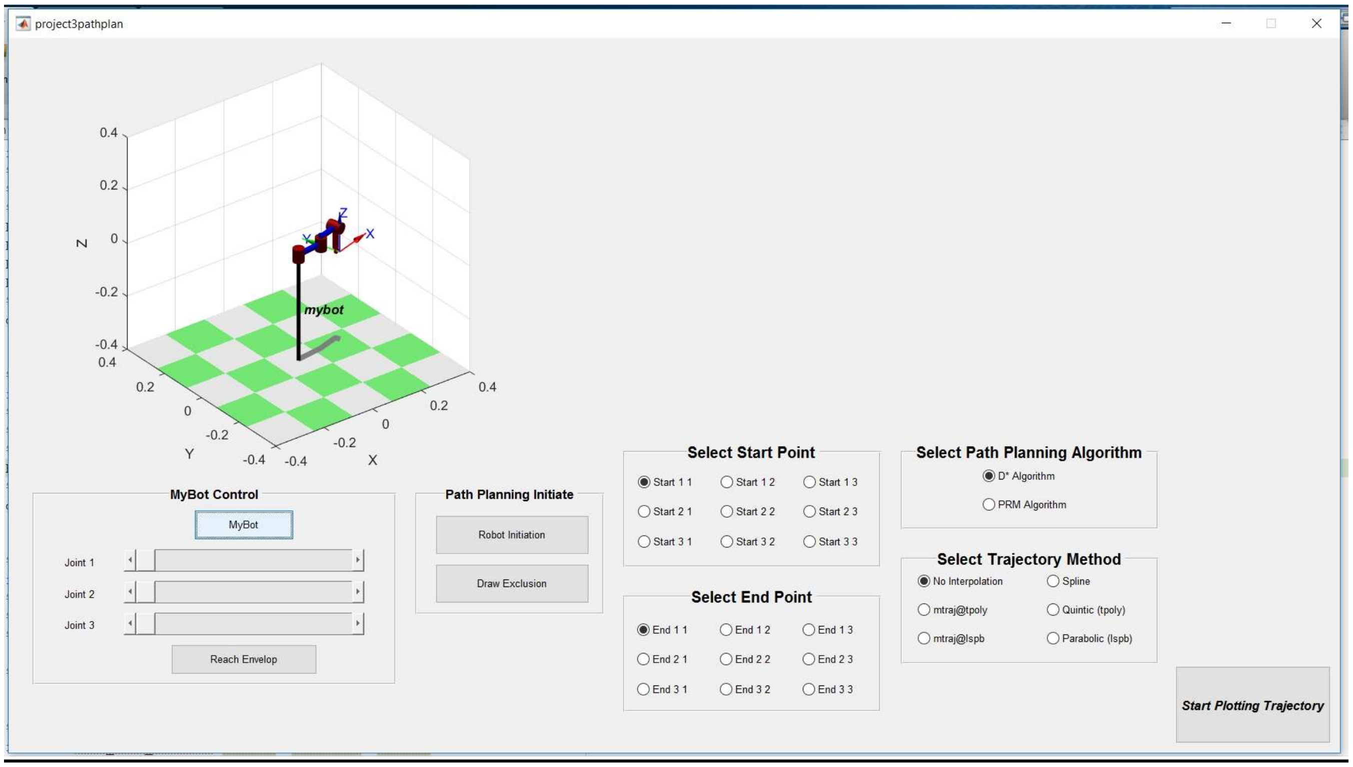


Figure below shows the trajectory plotted using D* Algorithm with mtraj interpolation method.



GUI For PathPlanning() class.



Select Start Point: [✓ Start 1 1] Start 1 2, Start 1 3, Start 2 1, Start 2 2, Start 2 3, Start 3 1, Start 3 2, Start 3 3

Select End Point: [✓ End 1 1] End 1 2, End 1 3, End 2 1, End 2 2, End 2 3, End 3 1, End 3 2, End 3 3

Select Path Planning Algorithm: [✓ D* Algorithm] PRM Algorithm

Select Trajectory Method: [✓ No Interpolation] Spline, mtraj@tpoly, Quintic (tpoly), mtraj@lspb, Parabolic (lspb)

Appendix

A. PathPlanning() class.

The coding for pathplanning class is below..

```
classdef PathPlanning
    % PathPlanning Class structure for Project Assignment 3 - Path
    % Planning... A moving experience..

    %   Path Planning is a class created to move from one point to another, with
    % obstacles in the way. The path created using D* and PRM Algorithm
    % avoids the obstacles. D* and PRM(Probabilistic Road Maps) are used to
    % trace path from one point to another.
    % Further Interpolation techniques are required to generate the
    % trajectory. The class also contains all interpolation functions.

    properties % properties (Global Variables) of PathPlanning class...
        MyBot % name of my robot.
        a % 'a' is a variable to be assigned for arduino communication.
        s1 % variable for communication with servo 1, for the joint 1 of the
robot..
        s2 % variable for communication with servo 2, for the joint 2 of the
robot..
        s3 % variable for communication with servo 3, for the end effecter of
robot..

        p11 = [15;75] % center point, p11, corresponding to center of box in 1st
row,1st column..
        p12 = [45;75] % center point, p12, corresponding to center of box in 1st
row,2nd column..
        p13 = [75;75] % center point, p13, corresponding to center of box in 1st
row,3rd column..
        p21 = [15;45] % center point, p21, corresponding to center of box in 2nd
row,1st column..
        p22 = [45;45] % center point, p22, corresponding to center of box in 2nd
row,2nd column..
        p23 = [75;45] % center point, p23, corresponding to center of box in 2nd
row,3rd column..
        p31 = [15;15] % center point, p31, corresponding to center of box in 3rd
row,1st column..
        p32 = [45;15] % center point, p32, corresponding to center of box in 3rd
row,2nd column..
        p33 = [75;15] % center point, p33, corresponding to center of box in 3rd
row,3rd column..

        M = [1 1 1 0 0 0] % define mask matrix for the 3DOF system...

        % Transformation Matrix of cartesian co-ordinates..
        T00 = transl(-.045, -.105, .0244) % Transformation matrix of co-ordinates
for center point of Grid...
```

```
        map % global Variable where exclusion map is stored..
        xy % a global variable where the co-ordinates of the path/trajectory
generated and stored....
        time % a global variable where time elapsed during an operation is
stored...
        start % global variabe to store start point on the map...
        goal % global variable to store end point on map where robot EE position
will be..
    end

    methods % The functions to execute D* and PRM(Probabilistic Road Map) Algorithm
are listed below...

        function obj = connectArd(obj) % The function connects the arduino with
MATLAB and starts communication..
            % with Arduino Hardware..
            obj.a = arduino('COM3','UNO'); % variable 'a' is assigned in workspace
to communicate with SerialPort COM3,
            % to which Arduino Hardware is connected..
        end % Attached Arduino with MATLAB...

        function obj = attachServo(obj) % The function Connects Joint Servo Motors
to arduino and communicates
            % through the port variable assigned 'a'...serially..
            obj.s1 = servo(obj.a, 'D9', 'MinPulseDuration', 575e-6,
'MaxPulseDuration', 2460e-6); % s1 = communication port variable with servo 1 for
joint 1
            obj.s2 = servo(obj.a, 'D10', 'MinPulseDuration', 575e-6,
'MaxPulseDuration', 2460e-6); % s2 = communication port variable with servo 2 for
joint 2
            obj.s3 = servo(obj.a, 'D3', 'MinPulseDuration', 640e-6,
'MaxPulseDuration', 2250e-6); % s3 = communication port variable with servo 3 for
end effecter
            writePosition(obj.s3, .25); % making pen up for smooth movement
        end % Attached joint angle servoes with MATLAB...

        function obj = createBot(obj) % The function creates a three-link, 3
Degrees-of-freedom robot with D-H parameters of the hardware..
            L(1) = Link('revolute', 'd', 0, 'a', .1035592, 'alpha', 0, 'offset',
.1303761, 'qlim', [0 -3.534291735288517]); % Link 1 of robot
            L(2) = Link('revolute', 'd', .024384, 'a', .0929005, 'alpha', pi/2,
'offset',.2265527692, 'qlim', [0 -3.534291735288517]); % Link 2 of robot
            L(3) = Link('revolute', 'd', .02017933333, 'a', 0, 'alpha', 0,
'offset', 0,'qlim', [0 2.879793265790644]); % Link 3 of robot
            %T=(r2t(rotx(-pi/2)))*transl([0 0 -.09]); % Tool or EE of the
Robot...which is pen in this case...
            obj.MyBot = SerialLink(L, 'name', 'ProBot'); % serial link creates the
robot 'MyBot', which is used to path planning algorithm.
        end % Created 3 Degrees-of-freedom robot...

        function obj = bring2CentrPos(obj) % The function brings end-effecter of 3-
link Robot to the center of grid. This will be initial position or
            % initial real-world cartesian co-ordinates for further
            % calculations and considerations...

            q00 = [-pi/3 -pi/3 0]; %1.590431280879833 Give initial joint angles as
input, which are closest to the center of grid..
```

```
            writePosition(obj.s1, abs(q00(:,1)/3.534291735288517)) % write angle 1
to servo 1 of joint 1
            writePosition(obj.s2, abs(q00(:,2)/3.534291735288517)) % write angle 2
to servo 2 of joint 2
            pause(1)
            %T = obj.MyBot.fkine(q00); % make fkine of the joint angles set q to
get the current position co-ordinates
            %M = [1 1 1 0 0 0]; % define mask matrix for the 3DOF system
            %q0 = obj.MyBot.ikine(T, q00, obj.M, 'pinv', 'ilimit', 150000, 'tol',
.0244); % initial joint angles
            %writePosition(obj.s1, abs(q0(:,1)/3.534291735288517)) % write angle 1
to servo 1 of joint 1
            %writePosition(obj.s2, abs(q0(:,2)/3.534291735288517)) % write angle 2
to servo 2 of joint 2

            % Note that, here and here onwards, z = .0244m, which is
            % permanent elevation of EE of robot, in positive z direction.
            % Since, our robot is moving only in two directions, only x and y
            % co-ordinates are changing, z is constant.

            q = obj.MyBot.ikine(obj.T00, q00, obj.M, 'pinv', 'ilimit', 150000,
'tol', .0244); % angle set corresponding to
            % center of map..
            % note that here pseudo-inverse option is set to get ikine of
            % the transformation matrix. Further, iteration limit is set to
            % 150000 using 'ilimit' option and tolerance is set to .0244
            % radians to avoid iteration warnings.
            pause(1) % delay to stabilize robot
            writePosition(obj.s1, abs(q(:,1)/3.534291735288517)); % writing joint
angle to servo 1 of joint 1
            writePosition(obj.s2, abs(q(:,2)/3.534291735288517)); % writing joint
angle to servo 2 of joint 2
            % note that max run each servo of the joint is 202.5 degree.
            % Therefore, max angle = 202.5*(pi/180) = 3.534291735288517 rad
            % Servo position must be from 0 to +1. hence we have to scale
            % generated joint angle from 0 to 1. Hence from hereonwards, we
            % will be dividing every generated joint angle by
            % 3.534291735288517 and taking its absolute value..
            pause(1) % delay to stabilize motors...
        end % End Effecter of Robot is at center of grid...

        function obj = BoxCenter(obj, xyz, q) % function to bring the EE of the
robot to the center of the Box
            % with xyz = x,y,z co-ordinates and q = joint angles as the
            % input arguments...

            writePosition(obj.s1, abs(q(:,1)/3.534291735288517)) % write
angle(Position) 1 to servo 1 of joint 1
            writePosition(obj.s2, abs(q(:,2)/3.534291735288517)) % write
angle(position) 2 to servo 2 of joint 2
            pause(1)
            T = transl(xyz(1), xyz(2), xyz(3)); % Defining center point p11 of box
11
            q = obj.MyBot.ikine(T, q, obj.M, 'pinv', 'ilimit', 150000, 'tol',
.0244); % ikine to get joint angles set corresponding to p11
            writePosition(obj.s1, abs(q(:,1)/3.534291735288517)); % writing joint
angle to servo 1 of joint 1
```

```
            writePosition(obj.s2, abs(q(:,2)/3.534291735288517)); % writing joint
angle to servo 2 of joint 2
            pause(1) % delay to stabilize the hardware...
        end % function to calculation of joint angles for center of box ends...

        function obj = bring2p11(obj) % The function is used to bring EE of robot
to center point, p11, of box in 1st row,1st column..

            xyz = [-.015 -.135 .0244]; % Defining enter of box 11 (box in 1st row,
1st column)..
            q = [-pi/3 -pi/3 0]; % Give initial joint angles as input, which are
closest to the center of grid..
            obj.BoxCenter(xyz, q); % calling function for calculating and writing
joint angles..
        end % EE of the robot is at the center of the Box 11 of map..

        function obj = bring2p12(obj) % The function is used to bring EE of robot
to center point, p12, of box in 1st row,2nd column..

            xyz = [-.045 -.135 .0244]; % Defining enter of box 12 (box in 1st row,
2nd column)..
            q = [-pi/3 -pi/3 0]; % Give initial joint angles as input, which are
closest to the center of grid..
            obj.BoxCenter(xyz, q); % calling function for calculating and writing
joint angles..
        end % EE of the robot is at the center of the Box 12 of map..

        function obj = bring2p13(obj) % The function is used to bring EE of robot
to center point, p13, of box in 1st row,3rd column..

            xyz = [-.075 -.135 .0244]; % Defining enter of box 13 (box in 1st row,
3rd column)..
            q = [-pi/3 -pi/3 0]; % Give initial joint angles as input, which are
closest to the center of grid..
            obj.BoxCenter(xyz, q); % calling function for calculating and writing
joint angles..
        end % EE of the robot is at the center of the Box 13 of map..

        function obj = bring2p21(obj) % The function is used to bring EE of robot
to center point, p21, of box in 2nd row,1st column..

            xyz = [-.015 -.105 .0244]; % Defining enter of box 21 (box in 2nd row,
1st column)..
            q = [-pi/3 -pi/3 0]; % Give initial joint angles as input, which are
closest to the center of grid..
            obj.BoxCenter(xyz, q); % calling function for calculating and writing
joint angles..
        end % EE of the robot is at the center of the Box 21 of map..

        function obj = bring2p22(obj) % The function is used to bring EE of robot
to center point, p22, of box in 2nd row,2nd column..

            xyz = [-.045 -.105 .0244]; % Defining enter of box 21 (box in 2nd row,
2nd column)..
            q = [-pi/3 -pi/3 0]; % Give initial joint angles as input, which are
closest to the center of grid..
            obj.BoxCenter(xyz, q); % calling function for calculating and writing
joint angles..
```

```
        end % EE of the robot is at the center of the Box 22 of map..

        function obj = bring2p23(obj) % The function is used to bring EE of robot
to center point, p23, of box in 2nd row,3rd column..

            xyz = [-.075 -.105 .0244]; % Defining enter of box 21 (box in 2nd row,
3rd column)..
            q = [-pi/3 -pi/3 0]; % Give initial joint angles as input, which are
closest to the center of grid..
            obj.BoxCenter(xyz, q); % calling function for calculating and writing
joint angles..
        end % EE of the robot is at the center of the Box 23 of map..

        function obj = bring2p31(obj) % The function is used to bring EE of robot
to center point, p31, of box in 3rd row,1st column..

            xyz = [-.015 -.075 .0244]; % Defining enter of box 31 (box in 3rd row,
1st column)..
            q = [-pi/3 -pi/3 0]; % Give initial joint angles as input, which are
closest to the center of grid..
            obj.BoxCenter(xyz, q); % calling function for calculating and writing
joint angles..
        end % EE of the robot is at the center of the Box 31 of map..

        function obj = bring2p32(obj) % The function is used to bring EE of robot
to center point, p32, of box in 3rd row,2nd column..

            xyz = [-.045 -.075 .0244]; % Defining enter of box 31 (box in 3rd row,
2nd column)..
            q = [-pi/2 -pi/2 0]; % Give initial joint angles as input, which are
closest to the center of grid..
            obj.BoxCenter(xyz, q); % calling function for calculating and writing
joint angles..
        end % EE of the robot is at the center of the Box 32 of map..

        function obj = bring2p33(obj) % The function is used to bring EE of robot
to center point, p32, of box in 3rd row,3rd column..

            xyz = [-.075 -.075 .0244]; % Give initial joint angles as input, which
are closest to the center of grid..
            q = [-pi/2 -pi/2 0]; % Give initial joint angles as input, which are
closest to the center of grid..
            obj.BoxCenter(xyz, q); % calling function for calculating and writing
joint angles..
        end % EE of the robot is at the center of the Box 33 of map..

        function obj = Exclusion(obj, xyz1, xyz2, xyz3, xyz4) % The function draws
Exclusion(obstacles) on the board, with four corner
            % points xyz1, xyz2, xyz3, and xyz4 as input arguments....

            q = [-pi/2 -pi/2 0]; % Given the initial joint angles as input for the
ikine..
            T = transl(xyz1(1),xyz1(2), xyz1(3)); % Creating the homogeneous
transform matrix of co-ordinates..
            q = obj.MyBot.ikine(T, q, obj.M, 'pinv', 'ilimit', 150000, 'tol',
.0244); % joint angles set corresponding to start of Exclusion of Box
            writePosition(obj.s1, abs(q(:,1)/3.534291735288517)); % writing joint
angle to servo 1 of joint 1
```

```
            writePosition(obj.s2, abs(q(:,2)/3.534291735288517)); % writing joint
angle to servo 2 of joint 2
            pause(1);

            writePosition(obj.s3, .54); % Making pen 'down' to write exclusion on
the map..

            T1 = transl(xyz2(1), xyz2(2),xyz2(3)); % Creating the homogeneous
transform matrix of co-ordinates..
            traj = ctraj(T, T1, 20); % creating cartesian trajectory with 20 points
on it..
            q1 = obj.MyBot.ikine(traj, q, obj.M, 'pinv', 'ilimit', 150000, 'tol',
.0244); % joint angles set corresponding to 1st line of  Exclusion of Box
            for n = 1:1:numrows(q1) % creating loop to write joint angles to
servos....
                writePosition(obj.s1, abs(q1(n,1)/3.534291735288517)); % writing
joint angle to servo 1 of joint 1
                writePosition(obj.s2, abs(q1(n,2)/3.534291735288517)); % writing
joint angle to servo 2 of joint 2
                pause(.05); % Delay to stabilize the servos of joints..
            end % Loop for writing the joint angles to servoes is complete..

            T2 = transl(xyz3(1), xyz3(2), xyz3(3)); % Creating the homogeneous
transform matrix of co-ordinates..
            traj = ctraj(T1, T2, 20); % creating cartesian trajectory with 20
points on it..
            q2 = obj.MyBot.ikine(traj, q1(20,:), obj.M, 'pinv', 'ilimit', 150000,
'tol', .0244); % joint angles set corresponding to 2nd line ofExclusion of Box
            for n = 1:1:numrows(q2) % creating loop to write joint angles to
servos....
                writePosition(obj.s1, abs(q2(n,1)/3.534291735288517)); % writing
joint angle to servo 1 of joint 1
                writePosition(obj.s2, abs(q2(n,2)/3.534291735288517)); % writing
joint angle to servo 2 of joint 2
                pause(.05); % Delay to stabilize the servos of joints..
            end % Loop for writing the joint angles to servoes is complete..

            T3 = transl(xyz4(1), xyz4(2), xyz4(3)); % Creating the homogeneous
transform matrix of co-ordinates..
            traj = ctraj(T2, T3, 20); % creating cartesian trajectory with 20
points on it..
            q3 = obj.MyBot.ikine(traj, q2(20,:), obj.M, 'pinv', 'ilimit', 150000,
'tol', .0244); % joint angles set corresponding to 3rd line of Exclusion of Box
            for n = 1:1:numrows(q3) % creating loop to write joint angles to
servos....
                writePosition(obj.s1, abs(q3(n,1)/3.534291735288517)); % writing
joint angle to servo 1 of joint 1
                writePosition(obj.s2, abs(q3(n,2)/3.534291735288517)); % writing
joint angle to servo 2 of joint 2
                pause(.05); % Delay to stabilize the servos of joints..
            end % Loop for writing the joint angles to servoes is complete..
            writePosition(obj.s3, .25); % Making pen up for easy movement..
        end % function for exclusion map drawing is complete..

        function obj = DrawExclusion(obj) % The function calls Exclusion(obj) to
draw exclusion map on board with predefined joint angles..
```

```
            % Exclusion for Box11
            xyz1 = [-.0075 -.1435 .0244]; xyz2 = [-.0235 -.1435 .0244]; % Providing
corner points
            xyz3 = [-.0235 -.1265 .0244]; xyz4 = [-.0075 -.1265 .0244]; % as input
arguments....
            obj.Exclusion(xyz1, xyz2, xyz3, xyz4); % calling Exclusion(obj) to draw
exclusion for box 11

            % Exclusion for Box12
            xyz1 = [-.0365 -.1425 .0244]; xyz2 = [-.0365 -.1265 .0244]; % Providing
corner points
            xyz3 = [-.0535 -.1265 .0244]; xyz4 = [-.0535 -.1425 .0244]; % as input
arguments....
            obj.Exclusion(xyz1, xyz2, xyz3, xyz4); % calling Exclusion(obj) to draw
exclusion for box 12

            % Exclusion for Box13
            xyz1 = [-.0825 -.1435 .0244]; xyz2 = [-.0665 -.1435 .0244]; % Providing
corner points
            xyz3 = [-.0665 -.1265 .0244]; xyz4 = [-.0825 -.1265 .0244]; % as input
arguments....
            obj.Exclusion(xyz1, xyz2, xyz3, xyz4); % calling Exclusion(obj) to draw
exclusion for box 13

            % Exclusion for Box23
            xyz1 = [-.0665 -.1125 .0244]; xyz2 = [-.0665 -.0965 .0244]; % Providing
corner points
            xyz3 = [-.0835 -.0965 .0244]; xyz4 = [-.0835 -.1125 .0244]; % as input
arguments....
            obj.Exclusion(xyz1, xyz2, xyz3, xyz4); % calling Exclusion(obj) to draw
exclusion for box 23

            % Exclusion for Box33
            xyz1 = [-.0665 -.0675 .0244]; xyz2 = [-.0665 -.0835 .0244]; % Providing
corner points
            xyz3 = [-.0835 -.0835 .0244]; xyz4 = [-.0835 -.0675 .0244]; % as input
arguments....
            obj.Exclusion(xyz1, xyz2, xyz3, xyz4); % calling Exclusion(obj) to draw
exclusion for box 33

            % Exclusion for Box32
            xyz1 = [-.0525 -.0665 .0244]; xyz2 = [-.0365 -.0665 .0244]; % Providing
corner points
            xyz3 = [-.0365 -.0835 .0244]; xyz4 = [-.0525 -.0835 .0244]; % as input
arguments....
            obj.Exclusion(xyz1, xyz2, xyz3, xyz4); % calling Exclusion(obj) to draw
exclusion for box 32

            % Exclusion for Box31
            xyz1 = [-.0065 -.0675 .0244]; xyz2 = [-.0065 -.0835 .0244]; % Providing
corner points
            xyz3 = [-.0235 -.0835 .0244]; xyz4 = [-.0235 -.0675 .0244]; % as input
arguments....
            obj.Exclusion(xyz1, xyz2, xyz3, xyz4); % calling Exclusion(obj) to draw
exclusion for box 31

            % Exclusion for Box21
            xyz1 = [-.0225 -.1135 .0244]; xyz2 = [-.0065 -.1135 .0244]; % Providing
corner points
            xyz3 = [-.0065 -.0965 .0244]; xyz4 = [-.0225 -.0965 .0244]; % as input
arguments....
            obj.Exclusion(xyz1, xyz2, xyz3, xyz4); % calling Exclusion(obj) to draw
exclusion for box 21

            % Exclusion for Box22
            xyz1 = [-.0365 -.0975 .0244]; xyz2 = [-.0365 -.1135 .0244]; % Providing
corner points
            xyz3 = [-.0535 -.1135 .0244]; xyz4 = [-.0535 -.0975 .0244]; % as input
arguments....
            obj.Exclusion(xyz1, xyz2, xyz3, xyz4); % calling Exclusion(obj) to draw
exclusion for box 21
        end % Drawing of exclusion on board is complete...

        function obj = NoInterPol(obj, xy) % The function is to plot the path
wothout any interpolation methods with 'xy' co-ordinates
            % generated from either PRM or D* algorithm, as input arguments....

            xy = obj.xy; % exchanging/storing/retrieving xy co-ordinates with
global variable xy..
            for n = 1:1:numrows(xy) % setting loop for updating xy co-ordinates
array, to the real world co-ordinates..
                xy(n,1) = -xy(n,1)/1000; % Transforming x co-ordinate of exclusion
map to the real world map x co-ordinate...
                xy(n,2) = -xy(n,2)/1000 - .06; % Transforming y co-ordinate of
exclusion map to the real world map y co-ordinate...
                xy(n,3) = 0.0244; % Setting z co-ordinate at the constant value...
            end % co-ordinates of xy array are updated to real world co-ordinates..
            q = [-pi/2 -pi/2 0]; % providing initial joint angle set as input
argument for ikine...
            writePosition(obj.s3, .54); % Making pen 'Down' to draw trajectory on
board..
            for n = 1:1:numrows(xy) % Loop for getting joint angles for the xy co-
ordinates..
                T = transl(xy(n,1), xy(n,2), xy(n,3)); % getting transformation
matrix...
                q = obj.MyBot.ikine(T, q, obj.M, 'pinv', 'ilimit', 150000, 'tol',
.0244) % making ikine to get joint angle set..
                writePosition(obj.s1, abs(q(1,1))/3.534291735288517) % writing
joint angle to servo 1..
                writePosition(obj.s2, abs(q(1,2))/3.534291735288517) % writing
joint angle to servo 2..
                pause(.005) % delay to stabilize servoes..
            end % Loop for writing the joint angles is ended..
            writePosition(obj.s3, .25); % making pen up for easy movement and to
avoid unwanted lines on map..
        end % function of PathPlanning class to draw path without interpolation is
ended....

        function obj = quinticTpoly(obj, xy) % Function of PathPlanning class to
draw trajectory with quintic interpolation (mtarj@tpoly)...

            xy = obj.xy; % exchanging/storing/retrieving xy co-ordinates with
global variable xy..
```

```
            for n = 1:1:numrows(xy) % setting loop for updating xy co-ordinates
array, to the real world co-ordinates..
                xy(n,1) = -xy(n,1)/1000; % Transforming x co-ordinate of exclusion
map to the real world map x co-ordinate...
                xy(n,2) = -xy(n,2)/1000 - .06; % Transforming y co-ordinate of
exclusion map to the real world map y co-ordinate...
                %xy11(n,3) = .0244; % Setting z co-ordinate at the constant
value...
            end % loop for updating co-ordinates of xy array ended and co-ordinates
are updated to real world co-ordinates..

            q = [-pi/2 -pi/2 0]; % providing initial joint angle set as input
argument for ikine...
            writePosition(obj.s3, .54); % Making pen 'Down' to draw trajectory on
board..

            for n = 1:1:numrows(xy) % Loop for getting joint angles for the xy co-
ordinates..
                if n == numrows(xy) % checking value of n to avoid 'index
dimensions exceed' error
                    break % break operation to avoid 'index dimensions exceed
error'...
                elseif n < numrows(xy) % Proceed for interpolation if n<numrows(xy)
                    traj = mtraj(@tpoly, xy(n,:), xy(n+1,:), 10); % making mtraj of
delta points obtained from PathPlanning algorithm...
                    for n = 1:1:numrows(traj) % Loop for making transformation
matrix of interpolated co-ordinates..
                        T =transl(traj(n,1), traj(n,2), .0244); % Transformation
matrix of interpolated co-ordinates..
                        q = obj.MyBot.ikine(T, q, obj.M, 'pinv', 'ilimit', 150000,
'tol', .0244); % joint angle set corresponding to interpolated points..
                        writePosition(obj.s1, abs(q(1,1))/3.534291735288517) %
writing joint angle to sero 1..
                        writePosition(obj.s2, abs(q(1,2))/3.534291735288517) %
writing joint angle to sero 2..
                        pause(.005) % delay to stabilize servoes..
                    end % nested Loop for interpolation of two points and writing
joint angles is ended..
                end % condition check loop for interpolation & writing joint angles
ended...
            end % Loop for interpolation of all co-ordinates and and writing joint
angles ended..
            writePosition(obj.s3, .25); % Making pen up for smooth movement and
avoiding unwanted lines on board..
        end % function for mtraj@tpoly interpolation ended..

        function obj = parabolicBlend(obj, xy) % Function of PathPlanning class to
draw trajectory with parabolic interpolation (mtraj@lspb)...

            xy = obj.xy; % exchanging/storing/retrieving xy co-ordinates with
global variable xy..
            for n = 1:1:numrows(xy) % setting loop for updating xy co-ordinates
array, to the real world co-ordinates..
                xy(n,1) = -xy(n,1)/1000; % Transforming x co-ordinate of exclusion
map to the real world map x co-ordinate...
                xy(n,2) = -xy(n,2)/1000 - .06; % Transforming y co-ordinate of
exclusion map to the real world map y co-ordinate...
```

```
                %xy11(n,3) = .0244; % Setting z co-ordinate at the constant 
value...
            end % loop for updating co-ordinates of xy array ended and co-ordinates 
are updated to real world co-ordinates..
            q = [-pi/2 -pi/2 0]; % providing initial joint angle set as input 
argument for ikine...
            writePosition(obj.s3, .54); % Making pen 'Down' to draw trajectory on 
board..

            for n = 1:1:numrows(xy) % Loop for getting joint angles for the xy co-
ordinates..
                if n == numrows(xy) % checking value of n to avoid 'index 
dimensions exceed' error
                    break % break operation to avoid 'index dimensions exceed 
error'...
                elseif n < numrows(xy) % Proceed for interpolation if n<numrows(xy)
                    traj = mtraj(@lspb, xy(n,:), xy(n+1,:), 10); % making mtraj of 
delta points obtained from PathPlanning algorithm...
                    for n = 1:1:numrows(traj) % Loop for making transformation 
matrix of interpolated co-ordinates..
                        T =transl(traj(n,1), traj(n,2), .0244); % Transformation 
matrix of interpolated co-ordinates..
                        q = obj.MyBot.ikine(T, q, obj.M, 'pinv', 'ilimit', 150000, 
'tol', .0244); % joint angle set corresponding to interpolated points..
                        writePosition(obj.s1, abs(q(1,1))/3.534291735288517) % 
writing joint angle to sero 1..
                        writePosition(obj.s2, abs(q(1,2))/3.534291735288517) % 
writing joint angle to sero 2..
                        pause(.005) % delay to stabilize servoes..
                    end % nested Loop for interpolation of two points and writing 
joint angles is ended..
                end % condition check loop for interpolation & writing joint angles 
ended...
            end % Loop for interpolation of all co-ordinates and and writing joint 
angles ended..
            writePosition(obj.s3, .25); % Making pen up for smooth movement and 
avoiding unwanted lines on board..
            pause(1) % delay to stabilize servoes..
        end % function for mtraj@lspb interpolation ended..

        function obj = onlyLspb(obj, xy) % Function of PathPlanning class to draw 
trajectory with parabolic interpolation (only lspb)...
            xy = obj.xy; % exchanging/storing/retrieving xy co-ordinates with 
global variable xy..
            for n = 1:1:numrows(xy) % setting loop for updating xy co-ordinates 
array, to the real world co-ordinates..
                xy(n,1) = -xy(n,1)/1000; % Transforming x co-ordinate of exclusion 
map to the real world map x co-ordinate...
                xy(n,2) = -xy(n,2)/1000 - .06; % Transforming y co-ordinate of 
exclusion map to the real world map y co-ordinate...
                %xy11(n,3) = .0244; % Setting z co-ordinate at the constant 
value...
            end % loop for updating co-ordinates of xy array ended and co-ordinates 
are updated to real world co-ordinates..

            q = [-pi/2 -pi/2 0]; % providing initial joint angle set as input 
argument for ikine...
```

```
            writePosition(obj.s3, .54) % Making pen 'Down' to draw trajectory on
board..
            for n = 1:1:numrows(xy) % Loop for getting joint angles for the xy co-
ordinates..
                if n==numrows(xy) % checking value of n to avoid 'index dimensions
exceed' error
                    break % break operation to avoid 'index dimensions exceed
error'...
                elseif n < numrows(xy) % Proceed for interpolation if n<numrows(xy)
                    xy0(:,1) = lspb(xy(n,1), xy(n+1,1), 10) % making
interpolation(lspb) of two consecutive x co-ordinates and storing to column array1
xy0...
                    xy0(:,2) = lspb(xy(n,2), xy(n+1,2), 10) % making
interpolation(lspb) of two consecutive y co-ordinates and storing to column array2
xy0...

                    for n2 = numrows(xy0) % Loop for making transformation matrix
of interpolated co-ordinates..
                        T = transl(xy0(n2,1), xy0(n2,2), .0244) % Transformation
matrix of interpolated co-ordinates..
                        q = obj.MyBot.ikine(T, q, obj.M, 'pinv', 'ilimit', 150000,
'tol', .0244); % joint angle set corresponding to interpolated points..
                        writePosition(obj.s1, abs(q(1,1))/3.534291735288517) %
writing joint angle to sero 1..
                        writePosition(obj.s2, abs(q(1,2))/3.534291735288517) %
writing joint angle to sero 2..
                        pause(.005) % delay to stabilize servoes..
                    end % nested Loop for interpolation of two points and writing
joint angles is ended..
                end % condition check loop for interpolation & writing joint angles
ended...
            end % Loop for interpolation of all co-ordinates and and writing joint
angles ended..
            writePosition(obj.s3, .25) % Making pen up for smooth movement and
avoiding unwanted lines on board..
        end % function for lspb interpolation ended..

        function obj = onlyTpoly(obj, xy) % Function of PathPlanning class to draw
trajectory with quintic interpolation (only tpoly)...
            xy = obj.xy; % exchanging/storing/retrieving xy co-ordinates with
global variable xy..
            for n = 1:1:numrows(xy) % setting loop for updating xy co-ordinates
array, to the real world co-ordinates..
                xy(n,1) = -xy(n,1)/1000; % Transforming x co-ordinate of exclusion
map to the real world map x co-ordinate...
                xy(n,2) = -xy(n,2)/1000 - .06; % Transforming y co-ordinate of
exclusion map to the real world map y co-ordinate...
                %xy11(n,3) = .0244; % Setting z co-ordinate at the constant
value...
            end % loop for updating co-ordinates of xy array ended and co-ordinates
are updated to real world co-ordinates..

            q = [-pi/2 -pi/2 0]; % providing initial joint angle set as input
argument for ikine...
            writePosition(obj.s3, .54) % Making pen 'Down' to draw trajectory on
board..
```

```
            for n = 1:1:numrows(xy) % Loop for getting joint angles for the xy co-
ordinates..
                if n==numrows(xy) % checking value of n to avoid 'index dimensions
exceed' error
                    break % break operation to avoid 'index dimensions exceed
error'...
                elseif n < numrows(xy) % Proceed for interpolation if n<numrows(xy)
                    xy0(:,1) = tpoly(xy(n,1), xy(n+1,1), 10); % making
interpolation(tpoly) of two consecutive x co-ordinates and storing to column array1
xy0...
                    xy0(:,2) = tpoly(xy(n,2), xy(n+1,2), 10); % making
interpolation(tpoly) of two consecutive y co-ordinates and storing to column array2
xy0...

                    for n2 = numrows(xy0) % Loop for making transformation matrix
of interpolated co-ordinates..
                        T = transl(xy0(n2,1), xy0(n2,2), .0244); % Transformation
matrix of interpolated co-ordinates..
                        q = obj.MyBot.ikine(T, q, obj.M, 'pinv', 'ilimit', 150000,
'tol', .0244); % joint angle set corresponding to interpolated points..
                        writePosition(obj.s1, abs(q(1,1))/3.534291735288517) %
writing joint angle to sero 1..
                        writePosition(obj.s2, abs(q(1,2))/3.534291735288517) %
writing joint angle to sero 2..
                        pause(.005) % delay to stabilize servoes..
                    end % nested Loop for interpolation of two points and writing
joint angles is ended..
                end % condition check loop for interpolation & writing joint angles
ended...
            end % Loop for interpolation of all co-ordinates and and writing joint
angles ended..
            writePosition(obj.s3, .25) % Making pen up for smooth movement and
avoiding unwanted lines on board..
        end % function for tpoly interpolation ended..

        function obj = onlySpline(obj, xy) % Function of PathPlanning class to draw
trajectory with quintic interpolation (only spline)...
            xy = obj.xy; % exchanging/storing/retrieving xy co-ordinates with
global variable xy..
            for n = 1:1:numrows(xy) % setting loop for updating xy co-ordinates
array, to the real world co-ordinates..
                xy(n,1) = -xy(n,1)/1000; % Transforming x co-ordinate of exclusion
map to the real world map x co-ordinate...
                xy(n,2) = -xy(n,2)/1000 - .06; % Transforming y co-ordinate of
exclusion map to the real world map y co-ordinate...
                %xy11(n,3) = .0244; % Setting z co-ordinate at the constant
value...
            end % loop for updating co-ordinates of xy array ended and co-ordinates
are updated to real world co-ordinates..

            q = [-pi/2 -pi/2 0]; % providing initial joint angle set as input
argument for ikine...
            writePosition(obj.s3, .54) % Making pen 'Down' to draw trajectory on
board..
            for n = 1:1:numrows(xy) % Loop for getting joint angles for the xy co-
ordinates..
```

```
                if n==numrows(xy) % checking value of n to avoid 'index dimensions
exceed' error
                    break % break operation to avoid 'index dimensions exceed
error'...
                elseif n < numrows(xy) % Proceed for interpolation if n<numrows(xy)
                    xy0(:,1) = spline(xy(n,1), xy(n+1,1), 10); % making
interpolation(tpoly) of two consecutive x co-ordinates and storing to column array1
xy0...
                    xy0(:,2) = spline(xy(n,2), xy(n+1,2), 10); % making
interpolation(tpoly) of two consecutive y co-ordinates and storing to column array2
xy0...

                    for n2 = numrows(xy0) % Loop for making transformation matrix
of interpolated co-ordinates..
                        T = transl(xy0(n2,1), xy0(n2,2), .0244); % Transformation
matrix of interpolated co-ordinates..
                        q = obj.MyBot.ikine(T, q, obj.M, 'pinv', 'ilimit', 150000,
'tol', .0244); % joint angle set corresponding to interpolated points..
                        writePosition(obj.s1, abs(q(1,1))/3.534291735288517) %
writing joint angle to sero 1..
                        writePosition(obj.s2, abs(q(1,2))/3.534291735288517) %
writing joint angle to sero 2..
                        pause(.005) % delay to stabilize servoes..
                    end % nested Loop for interpolation of two points and writing
joint angles is ended..
                end % condition check loop for interpolation & writing joint angles
ended...
            end % Loop for interpolation of all co-ordinates and and writing joint
angles ended..
            writePosition(obj.s3, .25) % Making pen up for smooth movement and
avoiding unwanted lines on board..
        end % function for spline interpolation ended..

        function obj = loadMap(obj) % function to load exclusion map ...

            load map2; % saved exclusion map in .mat format is loaded...
            obj.map = map2; % exclusion map is stored in global variable...
        end % function to load exclusion map is ended..

        function obj = createMap(obj) % function to create exclusion map...

            load map2; % saved exclusion map in .mat format is loaded...
            obj.map = map2; % exclusion map is stored in global variable...
        end % function to load exclusion map is ended..

        function obj = dStarAlgo(obj, start, goal, trajgen) % the function performs
D* pathplanning operation
            % It requires start, goal and trajgen function as input
            % argument. The start, goal and trajectory generating
            % algorithms are selected from GUI. The function gets values of
            % handles.start, handles.end, handles.ppalg & handles.trajgen
            % from GUI to check these variables and then performs actions
accordingly.

            tic; % Timer clock is set to to calculate time elapsed during
operation..
            switch start % getting value of start point selected from GUI.
```

```
                case 11 % if values of handles.start = 11.....
                    startpoint = obj.p11; % then start point = obj.p11, which is
defined in properties..
                case 12 % if values of handles.start = 12.....
                    startpoint = obj.p12; % then start point = obj.p12, which is
defined in properties..
                case 13 % if values of handles.start = 13.....
                    startpoint = obj.p13; % then start point = obj.p13, which is
defined in properties..
                case 21 % if values of handles.start = 21.....
                    startpoint = obj.p21; % then start point = obj.p21, which is
defined in properties..
                case 22 % if values of handles.start = 22.....
                    startpoint = obj.p22; % then start point = obj.p22, which is
defined in properties..
                case 23 % if values of handles.start = 23.....
                    startpoint = obj.p23; % then start point = obj.p23, which is
defined in properties..
                case 31 % if values of handles.start = 31.....
                    startpoint = obj.p31; % then start point = obj.p31, which is
defined in properties..
                case 32 % if values of handles.start = 32.....
                    startpoint = obj.p32; % then start point = obj.p32, which is
defined in properties..
                case 33 % if values of handles.start = 33.....
                    startpoint = obj.p33; % then start point = obj.p33, which is
defined in properties..
            end % switch for getting value of start point is ended..

            switch goal % getting value of goal point selected from GUI.
                case 11 % if values of handles.end = 11.....
                    goalpoint = obj.p11; % then goal point = obj.p11, which is
defined in properties..
                case 12 % if values of handles.end = 12.....
                    goalpoint = obj.p12; % then goal point = obj.p12, which is
defined in properties..
                case 13 % if values of handles.end = 13.....
                    goalpoint = obj.p13; % then goal point = obj.p13, which is
defined in properties..
                case 21 % if values of handles.end = 21.....
                    goalpoint = obj.p21; % then goal point = obj.p21, which is
defined in properties..
                case 22 % if values of handles.end = 22.....
                    goalpoint = obj.p22; % then goal point = obj.p22, which is
defined in properties..
                case 23 % if values of handles.end = 23.....
                    goalpoint = obj.p23; % then goal point = obj.p23, which is
defined in properties..
                case 31 % if values of handles.end = 31.....
                    goalpoint = obj.p31; % then goal point = obj.p31, which is
defined in properties..
                case 32 % if values of handles.end = 32.....
                    goalpoint = obj.p32; % then goal point = obj.p32, which is
defined in properties..
                case 33 % if values of handles.end = 33.....
                    goalpoint = obj.p33; % then goal point = obj.p33, which is
defined in properties..
```

```
            end % switch for getting value of goal point is ended..

            load map2; % loading exclusion map which is saved in .mat format..
            obj.map = map2; % saving loaded exclusion map in global variable map
which is defined in properties..
            ds = Dstar(obj.map); % Planning loaded exclusion map for goalpoint with
D* algorithm....
            ds.plan(goalpoint); % Planning loaded exclusion map for goalpoint with
D* algorithm....
            xy = ds.path(startpoint); % Getting x & y co-ordinates with minimum
distance from start to goal..
            obj.xy = xy; % storing generated xy co-ordinates to the global variable
xy, which is defined in properties..

            switch trajgen % getting value of trajectory generation (interpolation)
method selected from GUI...
                case 1 % if value of handles.trajgen = 1...
                    obj.NoInterPol(xy); % then select No Interpolation function,
defined in methods...
                case 2 % if value of handles.trajgen = 2...
                    obj.quinticTpoly(xy); % then select mtraj@tpoly function,
defined in methods...
                case 3 % if value of handles.trajgen = 3...
                    obj.parabolicBlend(xy); % then select mtraj@lspb function,
defined in methods...
                case 4 % if value of handles.trajgen = 4...
                    obj.onlyTpoly(xy); % then select only tpoly function, defined
in methods...
                case 5 % if value of handles.trajgen = 5...
                    obj.onlyLspb(xy); % then select only lspb function, defined in
methods...
                case 6 % if value of handles.trajgen = 6...
                    obj.onlySpline(xy); % then select only spline function, defined
in methods...
            end % switch for getting value of trajectory generation method is
ended..
            toc; % timer for calculation of elapsed time is ended..
            t = toc; % elapsed time value is stored in local variable...
            t = obj.time; % elapsed time value is stored in global variable...
            disp(obj.time); % Elapsed time is displayed in command window..
        end % function for D* path planning algorithm and trajectory plotting is
ended...

        function obj = prmAlgo(obj, start, goal, trajgen) % the function performs
Probabilistic( Voronoi) Road Map (PRM) pathplanning operation
            % It requires start, goal and trajgen function as input
            % argument. The start, goal and trajectory generating
            % algorithms are selected from GUI. The function gets values of
            % handles.start, handles.end, handles.ppalg & handles.trajgen
            % from GUI to check these variables and then performs actions
accordingly.

            tic; % Timer clock is set to to calculate time elapsed during
operation..
            switch start % getting value of start point selected from GUI.
                case 11 % if values of handles.start = 11.....
```

```
                    startpoint = obj.p11; % then start point = obj.p11, which is
defined in properties..
                case 12 % if values of handles.start = 12.....
                    startpoint = obj.p12; % then start point = obj.p12, which is
defined in properties..
                case 13 % if values of handles.start = 13.....
                    startpoint = obj.p13; % then start point = obj.p13, which is
defined in properties..
                case 21 % if values of handles.start = 21.....
                    startpoint = obj.p21; % then start point = obj.p21, which is
defined in properties..
                case 22 % if values of handles.start = 22.....
                    startpoint = obj.p22; % then start point = obj.p22, which is
defined in properties..
                case 23 % if values of handles.start = 23.....
                    startpoint = obj.p23; % then start point = obj.p23, which is
defined in properties..
                case 31 % if values of handles.start = 31.....
                    startpoint = obj.p31; % then start point = obj.p31, which is
defined in properties..
                case 32 % if values of handles.start = 32.....
                    startpoint = obj.p32; % then start point = obj.p32, which is
defined in properties..
                case 33 % if values of handles.start = 33.....
                    startpoint = obj.p33; % then start point = obj.p33, which is
defined in properties..
            end % switch for getting value of start point is ended..

            switch goal % getting value of goal point selected from GUI.
                case 11 % if values of handles.end = 11.....
                    goalpoint = obj.p11; % then goal point = obj.p11, which is
defined in properties..
                case 12 % if values of handles.end = 12.....
                    goalpoint = obj.p12; % then goal point = obj.p12, which is
defined in properties..
                case 13 % if values of handles.end = 13.....
                    goalpoint = obj.p13; % then goal point = obj.p13, which is
defined in properties..
                case 21 % if values of handles.end = 21.....
                    goalpoint = obj.p21; % then goal point = obj.p21, which is
defined in properties..
                case 22 % if values of handles.end = 22.....
                    goalpoint = obj.p22; % then goal point = obj.p22, which is
defined in properties..
                case 23 % if values of handles.end = 23.....
                    goalpoint = obj.p23; % then goal point = obj.p23, which is
defined in properties..
                case 31 % if values of handles.end = 31.....
                    goalpoint = obj.p31; % then goal point = obj.p31, which is
defined in properties..
                case 32 % if values of handles.end = 32.....
                    goalpoint = obj.p32; % then goal point = obj.p32, which is
defined in properties..
                case 33 % if values of handles.end = 33.....
                    goalpoint = obj.p33; % then goal point = obj.p33, which is
defined in properties..
            end % switch for getting value of goal point is ended..
            load map2; % loading exclusion map which is saved in .mat format..
            obj.map = map2; % saving loaded exclusion map in global variable map
which is defined in properties..
            prm = PRM(obj.map, 'npoints', 200); % Planning loaded exclusion map for
goalpoint with PRM(probabilistic road map) algorithm....
            prm.plan; % Planning loaded exclusion map for goalpoint with PRM
algorithm with 200 numpoints....
            xy = prm.path(startpoint,goalpoint); % Getting x & y co-ordinates from
start to goal by connecting node points..
            obj.xy = xy; % storing generated xy co-ordinates to the global variable
xy, which is defined in properties..

            switch trajgen % getting value of trajectory generation (interpolation)
method selected from GUI...
                case 1 % if value of handles.trajgen = 1...
                    obj.NoInterPol(xy); % then select No Interpolation function,
defined in methods...
                case 2 % if value of handles.trajgen = 2...
                    obj.quinticTpoly(xy); % then select mtraj@tpoly function,
defined in methods...
                case 3 % if value of handles.trajgen = 3...
                    obj.parabolicBlend(xy); % then select mtraj@lspb function,
defined in methods...
                case 4 % if value of handles.trajgen = 4...
                    obj.onlyTpoly(xy); % then select only tpoly function, defined
in methods...
                case 5 % if value of handles.trajgen = 5...
                    obj.onlyLspb(xy); % then select only lspb function, defined in
methods...
                case 6 % if value of handles.trajgen = 6...
                    obj.onlySpline(xy); % then select only spline function, defined
in methods...
            end % switch for getting value of trajectory generation method is
ended..
            toc; % timer for calculation of elapsed time is ended..
            t = toc; % elapsed time value is stored in local variable...
            t = obj.time; % elapsed time value is stored in global variable...
            disp(obj.time); % Elapsed time is displayed in command window..
        end % function for PRM(probabilistic road map) path planning algorithm and
rajectory plotting is ended...
    end % All the methods for path planning algorithm are ende hereby......
end % Finally, class for Path Planning project assignment 3 is ended..
```

B. Code for Path Planning GUI

```
function varargout = project3pathplan(varargin)
% PROJECT3PATHPLAN MATLAB code for project3pathplan.fig
%      PROJECT3PATHPLAN, by itself, creates a new PROJECT3PATHPLAN or raises the
existing
%      singleton*.
%
%      H = PROJECT3PATHPLAN returns the handle to a new PROJECT3PATHPLAN or the
handle to
%      the existing singleton*.
%
%      PROJECT3PATHPLAN('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PROJECT3PATHPLAN.M with the given input
arguments.
%
%      PROJECT3PATHPLAN('Property','Value',...) creates a new PROJECT3PATHPLAN or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before project3pathplan_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to project3pathplan_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help project3pathplan

% Last Modified by GUIDE v2.5 11-Dec-2016 00:47:17

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @project3pathplan_OpeningFcn, ...
                   'gui_OutputFcn',  @project3pathplan_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before project3pathplan is made visible.
function project3pathplan_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to project3pathplan (see VARARGIN)

% Choose default command line output for project3pathplan
% ---------------------------------------------------------------------------
% Defining handles structure for MyBot and it's Joints
% ---------------------------------------------------------------------------
handles.output = hObject; %handles for GUI Output..
handles.mybot=[];% Handles of robot...
handles.ja=0; % initiate handles for joint angle 1
handles.jb=0; % initiate handles for joint angle 2
handles.jc=0; % initiate handles for joint angle 3
handles.b=0;
handles.pp = PathPlanning(); % Handles structure to store class 'PathPlanning'
handles.start = 0; % handles structure to get start point
handles.end = 0; % handles structure variable to store end point
handles.ppalg = 0; % handles structure to store Path Planning algorithm
handles.trajgen = 0; % handles structure to store trajectory generation algorithm..
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes project3pathplan wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = project3pathplan_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% ---------------------------------------------------------------------------
% Creating MyBot's Link in GUIDE using D-H parameters of MyBot
% All Dimensions are in 'Meter', All angles are in 'Radians'
% ---------------------------------------------------------------------------
L(1) = Link('revolute', 'd', 0, 'a', .1035582, 'alpha', 0, 'offset', .1303761,
'qlim', [0 3.665191429188092]); % Link 1 of robot..
L(2) = Link('revolute', 'd', .024384, 'a', .08287, 'alpha', pi/2, 'offset',
.1717404, 'qlim', [0 3.665191429188092]); % link 2 of the robot..
L(3) = Link('revolute', 'd', .016435667, 'a', 0, 'alpha', 0, 'offset', 0, 'qlim',
[0 2.879793265790644]); % Link 3 of the Robot..
T=(r2t(rotx(-pi/2)))*transl([0 0 -.09]);% Tool of the ee of the robot..

% Creating Three-link MyBot using D-H Parameters, one set per joint
handles.mybot = SerialLink(L, 'tool', T, 'name', 'mybot');

% Plotting MyBot
```

```
handles.mybot.plot([handles.ja, handles.jb, handles.jc]); % Plotting  the robot
with joint angle handles as variable..
zoom on; % zoom on...

guidata(hObject,handles); % update handles structure..

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.ja=get(hObject, 'Value');
t=handles.mybot.fkine([handles.ja, handles.jb, handles.jc]);
p=t(1:3,4);
hold on;
plot3(p(1),p(2),p(3),'--bs');
handles.mybot.plot([handles.ja, handles.jb, handles.jc]);
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.jb=get(hObject, 'Value');
t=handles.mybot.fkine([handles.ja, handles.jb, handles.jc]);
p=t(1:3,4);
hold on;
plot3(p(1),p(2),p(3),'--rs');
handles.mybot.plot([handles.ja, handles.jb, handles.jc]);
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
```

```
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function slider3_Callback(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
handles.jc=get(hObject, 'Value');
handles.mybot.plot([handles.ja, handles.jb, handles.jc]);
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function slider3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in reach_envelop.
function reach_envelop_Callback(hObject, eventdata, handles)
% hObject    handle to reach_envelop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
for x=0:.1:3.577924966588375
    for y=0:.1:3.577924966588375
        v=handles.mybot.fkine([x,y,0]);
        u=v(1:3,4);
        hold on;
        handles.mybot.plot([x,y,0]);
        plot3(u(1),u(2),u(3),'--rs');
    end
end
hold off


% --- Executes on button press in start_1_1.
function start_1_1_Callback(hObject, eventdata, handles)
% hObject    handle to start_1_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of start_1_1


% --- Executes on button press in start_1_2.
```

```
function start_1_2_Callback(hObject, eventdata, handles)
% hObject    handle to start_1_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of start_1_2

% --- Executes on button press in start_1_3.
function start_1_3_Callback(hObject, eventdata, handles)
% hObject    handle to start_1_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of start_1_3

% --- Executes on button press in start_2_1.
function start_2_1_Callback(hObject, eventdata, handles)
% hObject    handle to start_2_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of start_2_1

% --- Executes on button press in start_2_2.
function start_2_2_Callback(hObject, eventdata, handles)
% hObject    handle to start_2_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of start_2_2

% --- Executes on button press in start_2_3.
function start_2_3_Callback(hObject, eventdata, handles)
% hObject    handle to start_2_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of start_2_3

% --- Executes on button press in start_3_1.
function start_3_1_Callback(hObject, eventdata, handles)
% hObject    handle to start_3_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of start_3_1

% --- Executes on button press in start_3_2.
function start_3_2_Callback(hObject, eventdata, handles)
% hObject    handle to start_3_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of start_3_2

% --- Executes on button press in start_3_3.
function start_3_3_Callback(hObject, eventdata, handles)
% hObject    handle to start_3_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of start_3_3


% --- Executes on button press in end_1_1.
function end_1_1_Callback(hObject, eventdata, handles)
% hObject    handle to end_1_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of end_1_1

% --- Executes on button press in end_1_2.
function end_1_2_Callback(hObject, eventdata, handles)
% hObject    handle to end_1_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of end_1_2

% --- Executes on button press in end_1_3.
function end_1_3_Callback(hObject, eventdata, handles)
% hObject    handle to end_1_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of end_1_3

% --- Executes on button press in end_2_1.
function end_2_1_Callback(hObject, eventdata, handles)
% hObject    handle to end_2_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of end_2_1

% --- Executes on button press in end_2_2.
function end_2_2_Callback(hObject, eventdata, handles)
% hObject    handle to end_2_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of end_2_2

% --- Executes on button press in end_2_3.
function end_2_3_Callback(hObject, eventdata, handles)
% hObject    handle to end_2_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of end_2_3

% --- Executes on button press in end_3_1.
function end_3_1_Callback(hObject, eventdata, handles)
% hObject    handle to end_3_1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of end_3_1

% --- Executes on button press in end_3_2.
function end_3_2_Callback(hObject, eventdata, handles)
% hObject    handle to end_3_2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of end_3_2

% --- Executes on button press in end_3_3.
function end_3_3_Callback(hObject, eventdata, handles)
% hObject    handle to end_3_3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of end_3_3


% --- Executes on button press in d_star_algorithm.
function d_star_algorithm_Callback(hObject, eventdata, handles)
% hObject    handle to d_star_algorithm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of d_star_algorithm


% --- Executes on button press in prm_algorithm.
function prm_algorithm_Callback(hObject, eventdata, handles)
% hObject    handle to prm_algorithm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of prm_algorithm


% --- Executes on button press in no_interpolation.
function no_interpolation_Callback(hObject, eventdata, handles)
% hObject    handle to no_interpolation (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of no_interpolation


% --- Executes on button press in mtraj_tpoly.
function mtraj_tpoly_Callback(hObject, eventdata, handles)
% hObject    handle to mtraj_tpoly (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mtraj_tpoly


% --- Executes on button press in mtraj_lspb.
function mtraj_lspb_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to mtraj_lspb (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of mtraj_lspb


% --- Executes when selected object is changed in select_start_point.
function select_start_point_SelectionChangedFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in select_start_point
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
    case 'start_1_1' % changes on radio button press start 11
        display('Start Point : Start 11') %  displays value..
        handles.start = 11; % Stores value in handles.start to send to Pathplanning
Class
    case 'start_1_2' % changes on radio button press start 12
        display('Start Point : Start 12') %  displays value..
        handles.start = 12; % Stores value in handles.start to send to Pathplanning
Class
    case 'start_1_3' % changes on radio button press start 13
        display('Start Point : Start 13') %  displays value..
        handles.start = 13; % Stores value in handles.start to send to Pathplanning
Class
    case 'start_2_1' % changes on radio button press start 21
        display('Start Point : Start 21') %  displays value..
        handles.start = 21; % Stores value in handles.start to send to Pathplanning
Class
    case 'start_2_2' % changes on radio button press start 22
        display('Start Point : Start 22') %  displays value..
        handles.start = 22; % Stores value in handles.start to send to Pathplanning
Class
    case 'start_2_3' % changes on radio button press start 23
        display('Start Point : Start 23') %  displays value..
        handles.start = 23; % Stores value in handles.start to send to Pathplanning
Class
    case 'start_3_1' % changes on radio button press start 31
        display('Start Point : Start 31') %  displays value..
        handles.start = 31; % Stores value in handles.start to send to Pathplanning
Class
    case 'start_3_2' % changes on radio button press start 32
        display('Start Point : Start 32') %  displays value..
        handles.start = 32; % Stores value in handles.start to send to Pathplanning
Class
    case 'start_3_3' % changes on radio button press start 33
        display('Start Point : Start 33') %  displays value..
        handles.start = 33; % Stores value in handles.start to send to Pathplanning
Class
end
% Update handles structure
guidata(hObject, handles);

% --- Executes when selected object is changed in select_end_point.
function select_end_point_SelectionChangedFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in select_end_point
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
    case 'end_1_1' % changes on radio button press end 11
        display('End Point : End 11') %  displays value..
        handles.end = 11; % Stores value in handles.start to send to Pathplanning
Class
    case 'end_1_2' % changes on radio button press end 12
        display('End Point : End 12') %  displays value..
        handles.end = 12; % Stores value in handles.start to send to Pathplanning
Class
    case 'end_1_3' % changes on radio button press end 13
        display('End Point : End 13') %  displays value..
        handles.end = 13; % Stores value in handles.start to send to Pathplanning
Class
    case 'end_2_1' % changes on radio button press end 21
        display('End Point : End 21') %  displays value..
        handles.end = 21; % Stores value in handles.start to send to Pathplanning
Class
    case 'end_2_2' % changes on radio button press end 22
        display('End Point : End 22') %  displays value..
        handles.end = 22; % Stores value in handles.start to send to Pathplanning
Class
    case 'end_2_3' % changes on radio button press end 23
        display('End Point : End 23') %  displays value..
        handles.end = 23; % Stores value in handles.start to send to Pathplanning
Class
    case 'end_3_1' % changes on radio button press end  31
        display('End Point : End 31') %  displays value..
        handles.end = 31; % Stores value in handles.start to send to Pathplanning
Class
    case 'end_3_2' % changes on radio button press end 32
        display('End Point : End 32') %  displays value..
        handles.end = 32; % Stores value in handles.start to send to Pathplanning
Class
    case 'end_3_3' % changes on radio button press end 33
        display('End Point : End 33') %  displays value..
        handles.end = 33; % Stores value in handles.start to send to Pathplanning
Class
end
% Update handles structure
guidata(hObject, handles);

% --- Executes when selected object is changed in select_path_plan_algorithm.
function select_path_plan_algorithm_SelectionChangedFcn(hObject, eventdata,
handles)
% hObject    handle to the selected object in select_path_plan_algorithm
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
    case 'd_star_algorithm' % when D* algorithm selection button i pressed...
        display('D* Algorithm Selected')
        handles.ppalg = 1; % store value to exchange with class
    case 'prm_algorithm' % when PRM algorithm selection button i pressed...
        display('PRM Algorithm Selected')
        handles.ppalg = 2; % store value to exchange with class
end
% Update handles structure
```

```
guidata(hObject, handles);

% --- Executes when selected object is changed in traj_gen_method.
function traj_gen_method_SelectionChangedFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in traj_gen_method
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%no_interpolation
switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
    case 'no_interpolation' % When no interpolation is selected...
        display('No Interpolation Method Selected...!!')
        handles.trajgen = 1; % Store the value in trajgen..
    case 'mtraj_tpoly' % When mtraj@tpoly interpolation is selected...
        display('mtraj with tpoly selected...')
        handles.trajgen = 2; % Store the value in trajgen..
    case 'mtraj_lspb'% When mtraj @ lspb interpolation is selected...
        display('mtraj with lspb selected...')
        handles.trajgen = 3; % Store the value in trajgen..
    case 'only_tpoly' % When only tpoly interpolation is selected...
        display('Only tpoly selected...')
        handles.trajgen = 4; % Store the value in trajgen..
    case 'only_lspb' % When only lspb interpolation is selected...
        display('Only lspb selected...')
        handles.trajgen = 5; % Store the value in trajgen..
    case 'only_spline' % When only spline interpolation is selected...
        display('Only spline selected...')
        handles.trajgen = 6; % Store the value in trajgen..
end
% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in start_plotting_tajectory.
function start_plotting_tajectory_Callback(hObject, eventdata, handles)
% hObject    handle to start_plotting_tajectory (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if handles.ppalg == 1 % if handles.ppalg value is one,
    % D* algorithm is used...
    handles.pp.dStarAlgo(handles.start, handles.end, handles.trajgen); % value is
given to class Pathplanning
elseif handles.ppalg == 2 % if handles.ppalg value is two..
    % PRM Algorithm is used..
    handles.pp.prmAlgo(handles.start, handles.end, handles.trajgen); % value is
given to class..
end


% --- Executes on button press in robot_initiate.
function robot_initiate_Callback(hObject, eventdata, handles)
% hObject    handle to robot_initiate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.pp = handles.pp.connectArd; % updating handles.pp to connect arduino
function(obj)
```

```
handles.pp = handles.pp.attachServo; % updating handles.pp to attach sevo
function(obj)
handles.pp = handles.pp.createBot; % updating handles.pp to createbot function(obj)
handles.pp = handles.pp.bring2CentrPos; % updating handles.pp to Bring to centerpos
function(obj)
% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in draw_exclusion.
function draw_exclusion_Callback(hObject, eventdata, handles)
% hObject    handle to draw_exclusion (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.pp = handles.pp.DrawExclusion; % updates handles.pp structure with
DrawExclusion(obj) and draws exclusion
% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in only_tpoly.
function only_tpoly_Callback(hObject, eventdata, handles)
% hObject    handle to only_tpoly (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of only_tpoly


% --- Executes on button press in only_lspb.
function only_lspb_Callback(hObject, eventdata, handles)
% hObject    handle to only_lspb (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of only_lspb


% --- Executes on button press in only_spline.
function only_spline_Callback(hObject, eventdata, handles)
% hObject    handle to only_spline (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of only_spline
```

C. Video Link

https://youtu.be/Hbo7bJH3JI0

https://www.youtube.com/watch?v=Hbo7bJH3JI0&feature=youtu.be

End of the Document

End of Assignment 3 : Path Planning