PROJECT REPORT

i) System and Task Description

Important Points

- Determine only the motion of the manipulator in the horizontal plane
- Robot can be modeled as a simple 2R manipulator
- Coordinates of robot base $(x, y) = (0, 0)$
- Link lengths; L1 = 0.5m, L2 = 0.4m

ii) Definition and Representation of Free Space and Obstacle Space

Important Points

- Wall runs parallel to $x - axis$; distance from robot base is 0.1m
- Circle 1 has radius $B = 0.2m,$ center $(xc, yc) = (-0.6, 0.7)$
- Circle 2 has radius $B = 0.2m,$ center$(xc, yc) = (0.6, 0.7)$
- Angular motion of link 1 is limited by presence of wall
- Angular motion of link 2; $\pm 90\ degrees$ with respect to configuration in which it is perfectly aligned with $\theta_2 \in [-\pi/2, \pi/2]$ Link 1

iii) Tasks

Important Points

- Plan a motion from any given initial configuration (where object is picked) to any final configuration (where object is placed), chosen in the free space, avoiding any collision during robot motion
- Represent Robot Configuration Space, free and obstacle space.
- Representation of start and goal configurations, generated tree, and shortest path found by RRT and RRT* (each on separate figure)
- Step validation d = pi/50
- $Radius\ of\ Cgoal\ circle = pi/25$
- $Qstart = [3, -0.8]\ \ and\ Qgoal = [0,1]$

Code Explanation

```
L_1 = 0.5;          %link length L1
L_2 = 0.4;          %link length L2
R_obstacle = 0.2;   %Radius of obstacles
x_obst1 = -0.6;     %x-coordinate of obstacle 1
y_obst1 = 0.7;      %y-coordinate of obstacle 1
x_obst2 = 0.6;      %x-coordinate of obstacle 2
y_obst2 = 0.7;      %y-coordinate of obstacle 2

x_start = 3;        %x-coordinate of q_start
y_start = -0.8;     %y-coordinate of q_start
x_goal = 0;         %x-coordinate of q_goal
y_goal = 1;         %y-coordinate of q_goal
```

This part of the code declares the parameters of the link lengths, obstacle radius, the coordinates of obstacles (x, y), and the (x, y) start and goal coordinates to be planned. Since the robot is a 2R Planar robot, we work with 2-dimensional Euclidean, (x, y).

```
%Create Cspace.
figure(1)
[theta1, theta2] = meshgrid(-pi/2:0.02:3*pi/2,-pi/2:0.02:3*pi/2);

cond_1 = L_1.*sin(theta1)>-0.1;
cond_2 = (L_1.*cos(theta1) + 1./4.*L_2.*cos(theta1+theta2)-x_obst1).^2 + (L_1.*sin(theta1) + 1./4.*L_2.*sin(theta1+theta2
cond_3 = (L_1.*cos(theta1) + 3./4.*L_2.*cos(theta1+theta2)-x_obst1).^2 + (L_1.*sin(theta1) + 3./4.*L_2.*sin(theta1+theta2
cond_4 = (L_1.*cos(theta1) + 1./4.*L_2.*cos(theta1+theta2)-x_obst2).^2 + (L_1.*sin(theta1) + 1./4.*L_2.*sin(theta1+theta2
cond_5 = (L_1.*cos(theta1) + 3./4.*L_2 *cos(theta1+theta2)-x_obst2).^2 + (L_1.*sin(theta1) + 3./4.*L_2.*sin(theta1+theta2

conditions = cond_1&cond_2&cond_3&cond_4&cond_5;    %combines the conditions to create cspace
cspace = conditions;

%Create a state space.
```

This section of the code creates a figure (Figure 1) for visualizing the configuration space (C-space). The meshgrid function is used to create a grid of theta1 and theta2 values, which represent the joint angles of the robot arm. The created grid covers a range of values of both joint angles.

Afterwards, several conditions (cond_1 through cond_5) which represent constrains on the robot's movement are defined. These conditions check for collisions with obstacles and boundaries.

The conditions are then combined to create a binary map (c-space) of the configuration space. This map represents where the robot can and cannot move.

Explanation of Conditions 1 – 5

*cond_1 = L_1.*sin(theta1) > -0.1;*

This condition checks for a collision with the wall located at y = -0.1 in the configuration space. It is a simple condition based on the sin(theta1) component of the robot's arm position. If the sine component is greater than -0.1, it indicates that the arm does not collide with this imaginary obstacle.

*cond_2 = (L_1.*cos(theta1) + 1./4.*L_2.*cos(theta1+theta2)-x_obst1).^2 + (L_1.*sin(theta1) + 1./4.*L_2.*sin(theta1+theta2)-y_obst1).^2 > (1./4.*L_2 + R_obstacle).^2;*

This condition evaluates whether the robot's arm collides with the first obstacle located at (x_obst1, y_obst1).

It calculates the distance between the end effector of the robot (specified by L_1.*cos(theta1) + 1./4.*L_2.*cos(theta1+theta2) and L_1.*sin(theta1) + 1./4.*L_2.*sin(theta1+theta2)) and the obstacle.

It then checks if this distance is greater than the sum of the radii of the obstacle (1./4.*L_2 + R_obstacle). If it is greater, the configuration is collision-free.

*cond_3 = (L_1.*cos(theta1) + 3./4.*L_2.*cos(theta1+theta2)-x_obst1).^2 + (L_1.*sin(theta1) + 3./4.*L_2.*sin(theta1+theta2)-y_obst1).^2 > (1./4.*L_2 + R_obstacle).^2;*

This condition is similar to cond_2 but checks for collisions with the same obstacle when the robot's arm is extended by 3/4 of the second link length. It ensures that the entire arm's movement is considered for obstacle avoidance.

*cond_4 = (L_1.*cos(theta1) + 1./4.*L_2.*cos(theta1+theta2)-x_obst2).^2 + (L_1.*sin(theta1) + 1./4.*L_2.*sin(theta1+theta2)-y_obst2).^2 > (1./4.*L_2 + R_obstacle).^2;*

This condition checks for collisions with the second obstacle located at (x_obst2, y_obst2) in a similar manner as cond_2.

*cond_5 = (L_1.*cos(theta1) + 3./4.*L_2.*cos(theta1+theta2)-x_obst2).^2 + (L_1.*sin(theta1) + 3./4.*L_2.*sin(theta1+theta2)-y_obst2).^2 > (1./4.*L_2 + R_obstacle).^2;*

This condition is similar to cond_4 but checks for collisions with the second obstacle when the robot's arm is extended by 3/4 of the second link length.

```
%Create a state space.
ss = stateSpaceSE2;
%Create an occupanyMap-based state validator using the created state space.
sv = validatorOccupancyMap(ss);

%Create an occupany map from an example map and set map resolution as 10 cells/meter.
map = occupancyMap(flip(1-cspace),52.5);
sv.Map = map;
figure(1)
show(map);
title('Free and Obstacle Space');

%Set validation distance for the validator.
sv.ValidationDistance = pi/50;
%Update state space bounds to be the same as map limits.
ss.StateBounds = [map.XWorldLimits;map.YWorldLimits; [-pi pi]];
```

This section of the code creates the state space ***SS*** for the robot. The **stateSpaceSE2 object (from MATLAB)** represents the state space for 2D rigid body motion, which is suitable for this robotic arm. The line ***sv=validatorOccupancyMap(ss)*** creates the state validator ***sv*** for the state space. The validator checks if the state is valid based on the occupancy map and the conditions defined.

The line map=occupancyMap(flip(1-cspace), 52.5) creates the occupancy map (map) based on the configuration space. The occupancy map is used to represent the obstacles and free space in the environment. ***sv.Map = map*** is used to set the ***validatorOccupancyMap*** to the occupancy map created.

Figure 1 shows the occupancy map in Figure 1 for visualization. ***sv.ValidationDistance*** sets the validation distance for the state validator ***(sv.ValidationDistance)*** and update the state space bounds to match the map limits.

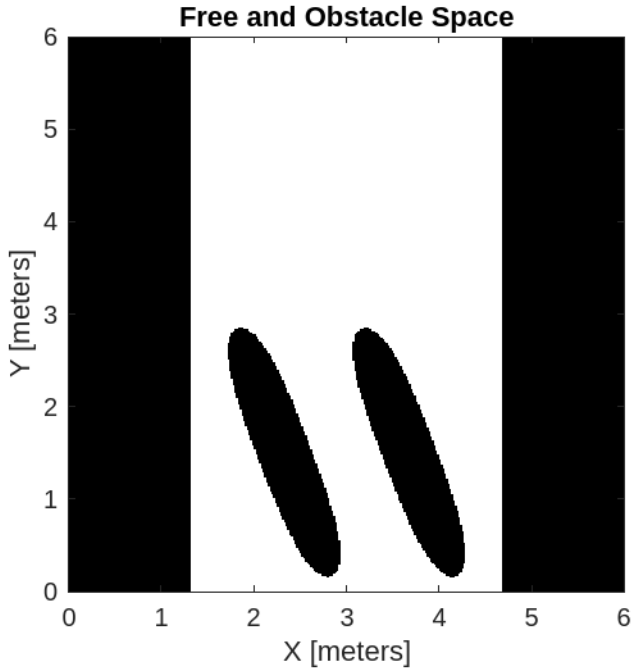

Figure 1: Free and Obstacle Space

```
%Set the start and goal states.
start = [x_start+1.5, y_start+1.5, 0];
goal = [x_goal+1.5, y_goal+1.5, 0];

%RRT algortim using plannerRRT

%Create the path planner and increase max connection distance.
planner = plannerRRT(ss,sv);
planner.MaxIterations = 6000;
planner.MaxConnectionDistance = pi/50;
%Plan a path with default settings.
[pthObj,solnInfo] = plan(planner,start,goal);
%Visualize the results.
figure(2)
show(map);
title('RRT');
start_text = text(start(1),start(2),'Start');
start_text.FontSize = 15;
start_text.Color = 'r';
goal_text = text(goal(1),goal(2),'Goal');
goal_text.FontSize = 15;
goal_text.Color = 'r';

hold on
plot(solnInfo.TreeData(:,1),solnInfo.TreeData(:,2),'.-'); % tree expansion
plot(pthObj.States(:,1),pthObj.States(:,2),'r-','LineWidth',2) % draw path
```

This section of the code sets the start and goal states for the path planning. The start and end goals are shifted by 1.5 to increase the bound of the map. The states are represented as 3D points (x, y) and theta. Theta is set to 0 in both start and goal states.

A path planner (planner) is created using RRT algorithm (using the ***plannerRRT*** function from MATLAB. The plannerRRT function takes as input, the ***ss*** and ***sv*** created earlier. ***ss*** and ***sv*** are the state space and state validation) for the given state space and state validator. The maximum connection distance is increased using (***planner.MaxConnectionDistance***), as required in the project Task to use ***d = pi/50*** for the RRT algorithm. The iteration is set using the iteration function in planner RRT library. It is set to a maximum of 6000. The purpose of the iteration is to show optimized and better improvement.

The path is planned using the RRT planner (plan) and the path object and solution information is returned.

Figure 2 is created for visualizing the RRT path planning results. It shows the occupancy map and marks the start and goal points. Additionally, the tree expansion and the planned path is displayed using the plot functions.

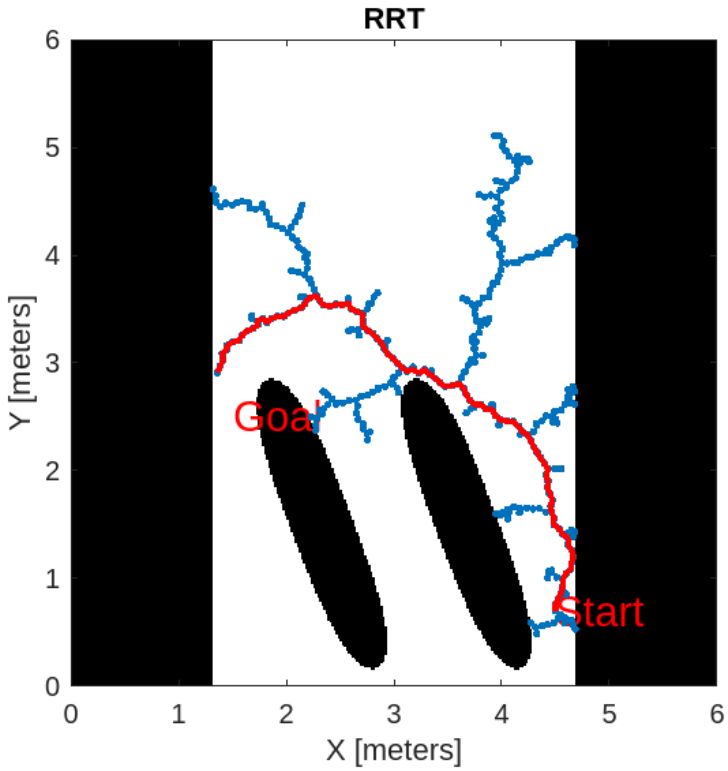

Figure 2: RRT Algorithm

RRT* Algorithm

```
%RRT* algortim using plannerRRTstar

%Create RRT* path planner and allow further optimization after goal is reached. Reduce the maximum iterations and increa
planner2 = plannerRRTStar(ss,sv);
planner2.ContinueAfterGoalReached = true; % Allow further optimization
planner2.MaxIterations = 6000; % Number of max iterations
planner2.MaxConnectionDistance = pi/50; % Increase max connection distance

%Plan a path with default settings.
[pthObj,solnInfo] = plan(planner2,start,goal);

%Visualize the results.
figure(3)
map.show;
title('RRT*')
start_text = text(start(1),start(2),'Start');
start_text.FontSize = 15;
start_text.Color = 'r';
goal_text = text(goal(1),goal(2),'Goal');
goal_text.FontSize = 15;
goal_text.Color = 'r';

hold on
plot(solnInfo.TreeData(:,1),solnInfo.TreeData(:,2),'.-') % Tree expansion
plot(pthObj.States(:,1),pthObj.States(:,2),'r-','LineWidth',2) % Draw path
```

A second path planner (planner2) is implemented using the RRT* algorithm. This algorithm is an improved version of RRT and allows further optimization after the goal is reached. The parameters are set in the same way I set them for RRT planner.

The path is planned using the RRT* planner library from Matlab. It returns the path object and solution information.

Figure 3 is created for visualizing the RRT* path planning results. It shows the occupancy map and marks the start and goal points. The tree expansion and optimized path are displayed

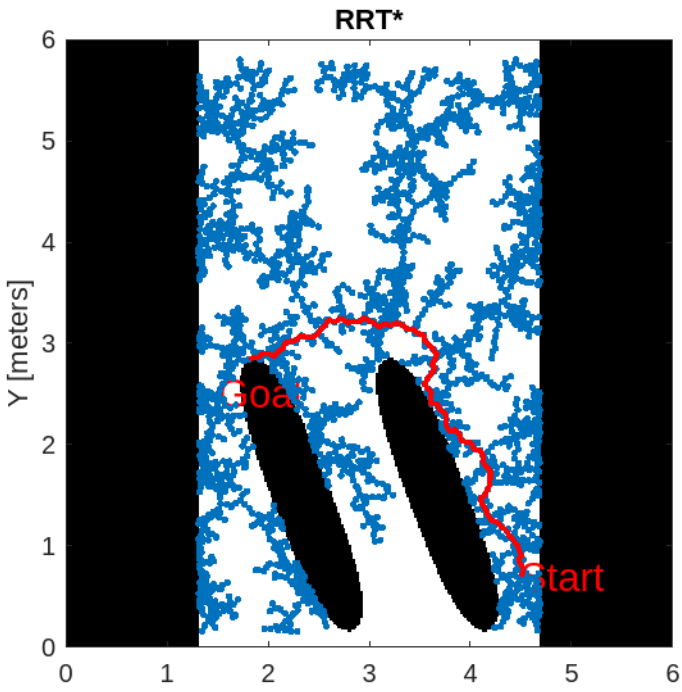

*Figure 3: RRT**