Abstract:

In this project, I implemented a holonomic point-robot in a 2D plane with couple of 2D obstacles. I utilized Rapidly exploring Random Tree (RRT) as my motion planning algorithm and I used occupancy check for the point robot to prevent collision. I also used a predefined algorithm in MATLAB to find the shortest path between the source node and the target node. I did the coding in MATLAB and I also used MATLAB for the visualization part. The goal was to define the start point and end point and expand the tree from its root (start point) till it reaches to the end point using RRT algorithm and finally finding the shortest path between these two nodes using predefined MATLAB algorithm.

Next, I am going to explain all the steps that I took for this project in details:

Creating the Map and Obstacles:

The first step was to create the map in MATLAB. For this purpose, I created a function in the MATLAB which creates the map, its obstacles, starting point and the end point.

```
function map = plainmap(S,E)
```

Where S represents the Start point and E represents the End point. Next, we need to assign a value to each of these sections so we can recognize them on the map. First, I created a 100 by 100, 2D map and I assigned a 0 value to each node of this map.

```
x_axis = 100;
y_axis = 100;

   for m=1:x_axis
        for n=1:y_axis
            map(m,n)=0;
        end
    end
```

Next, I defined the starting point and the end point in the map, and I assigned a 5 value to each of them. The value itself does not matter. It only helps us to recognize the points on the map.

```
    map(Start(1),Start(2))=5;

    map(End(1),End(2))=5;
```

Now we need to define one more thing on the map, the obstacles. I have defined five different polyhedral obstacles in various positions with different dimensions.

```
%Obstacle 1:

    for m=35:55
        for n=25:45
            map(m,n)=1;
        end
    end

     %Obstacle 2:
```

```
for m=60:65
    for n=60:75
        map(m,n)=1;
    end
end

 %Obstacle 3:

for m=5:20
    for n=10:12
        map(m,n)=1;
    end
end

%Obstacle 4:

for m=20:35
    for n=80:90
        map(m,n)=1;
    end
end

    %Obstacle 5:

for m=65:85
    for n=15:40
        map(m,n)=1;
    end
end
```

And finally, we have to define all the borders of the plane, so the tree does not expand further that.

```
for m=1
     for n=1:100
         map(m,n)=1;
     end
 end

   for m=100
     for n=1:100
         map(m,n)=1;
     end
   end

 for m=1:100
     for n=1
         map(m,n)=1;
     end
 end

  for m=1:100
     for n=100
         map(m,n)=1;
```

```
        end
    end
```

Initializing the Map and the Point Robot:

Second step was to define the position of the starting point and ending point of the robot. I just chose two random positions in the 2D plane for each point.

```
Start_Point = [17,19]; %Position of the starting point
 End_Point=[88,90]; %Position of the end point
 map=plainmap(Start_Point,End_Point); %Map creation function
 imagesc(map) %Display image with scaled colors
```

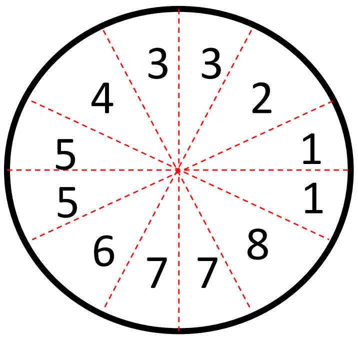
Moving Towards Next Point Algorithm:

In this step, we have to define a function which helps us to expand the tree and move toward the selected random point from the initial point by creating a new point between. The algorithm behind this function is to get the information between the near point and the random generated point, calculate the angle between those two points and then decides the moving direction of the near point.

In order to calculate the angle between the near node and the random point:

```
Euc_Dist=1;
  Xdiff=Next(1)-Current(1);
  Ydiff=Next(2)-Current(2);
  Theta=atan2(Ydiff,Xdiff);
    if Theta < 0
       Theta = Theta + 2*pi;
    End
```

In order to specify the direction of the moving robot, I assumed that if we have a full circle, we can divide it to 12 parts and each part is $\pi/6$ degrees. Based on this division, and the angle between those two points, I specified that how should the near point move towards the random point.



In the circle shown above, the same numbers in each part of the circle, are showing the same moving direction. For example, if the angle between the two points are in area 3, the “near point” will be moving towards “North” and the “new point” will be created there.

```
%Move to the East
if(Theta<=pi/6 && Theta>=0)
    Motion(1)=1*Euc_Dist;
    Motion(2)=0;
end
%Move to the North East
if(Theta<pi/3 && Theta>pi/6)
    Motion(1)=1*Euc_Dist;
    Motion(2)=1*Euc_Dist;
end
%Move to the North
if(Theta<=2*pi/3 && Theta>=pi/3)
    Motion(1)=0;
    Motion(2)=1*Euc_Dist;
end
%Move to the North West
if(Theta<5*pi/6 && Theta>2*pi/3)
    Motion(1)=-1*Euc_Dist;
    Motion(2)=1*Euc_Dist;
end
%Move to the West
if(Theta<=7*pi/6 && Theta>=5*pi/6)
    Motion(1)=-1*Euc_Dist;
    Motion(2)=0;
end
%Move to the South West
if(Theta<4*pi/3 && Theta>7*pi/6)
    Motion(1)=-1*Euc_Dist;
    Motion(2)=-1*Euc_Dist;
end
%Move to the South
if(Theta<=5*pi/3 && Theta>=4*pi/3)
    Motion(1)=0;
    Motion(2)=-1*Euc_Dist;
end
%Move to the South East
if(Theta<11*pi/6 && Theta>5*pi/3)
    Motion(1)=1*Euc_Dist;
    Motion(2)=-1*Euc_Dist;
end
if(Theta<=2*pi && Theta>=11*pi/6)
    Motion(1)=1*Euc_Dist;
    Motion(2)=0;
end
```

RRT Algorithm:

Certainly, the most important part was to complete the RRT algorithm. For the sake of simplicity, I explain the whole process in 6 steps briefly for this project.

1- **Generate a random point using the defined function.**

```
      Random_Point=[randi([1,100]), randi([1,100])];
```

2- **Check the closest vertex from the existing tree.**

```
for j=1:idx
Neighbor(j)=pdist([NeighborX(j),NeighborY(j);Random_Point(1),Random_Poi
nt(2)]);
end
[Distance, NN]=min(Neighbor); %Find the minimum distance and its
relative index
```

3- **Create the new node towards the random point from the closest vertex and its relative edge.**

```
Start_Point=Near_Point+Direction(Near_Point,Random_Point);
```

4- **Check if this node is within the range and it does not collide with any obstacle on the map. (Is it occupied or not)**

```
if map(Start_Point(1),Start_Point(2)) ~=1
                Occ=-1;
                for k=1:idx
                    if Start_Point(1)==NeighborX(k) && Start_Point(2)
== NeighborY(k)
                        Occ=1;
                        break
                    end
                end
```

5- **Add the new added node and its relative edge to the existing lists**.

```
if Occ==-1
          idx=idx+1;
          Graph_RRT = addnode(Graph_RRT,1);
          Graph_RRT = addedge(Graph_RRT,NN,idx);
          NeighborX(idx)=Start_Point(1);
          NeighborY(idx)=Start_Point(2);
        End
```

6- **Repeat this process till we reach to the End point**.

```
while (any(Start_Point ~= End_Point))
```

Finding the Shortest Path:

There is a command in MATLAB which gives you the shortest path between two single nodes. I have not written the shortest path finding algorithm here. I just used the MATLAB command which simply gives us the shortest path between two nodes in a graph.

```
Short_Path = shortestpath(Graph_RRT,1,idx
highlight(Plot_RRT,Short_Path,'NodeColor','b','EdgeColor','r');
```

Challenges and Achievements:

In this part, I would like to explain all the challenges that I have faced during this project. The most challenging part was to come up with an algorithm for the control input of the RRT algorithm (which is defined as moving function or direction function in my project/report). As it was mentioned in the class, the control input algorithm is one the most time-consuming parts of the whole motion planning algorithm and it is absolutely important to have an efficient algorithm. I am totally aware of that my algorithm is not efficient but the challenges that I had in this part, cause lots of problems for me to reach to the “end point” from the “start point”. Before finding out the final algorithm, usually the starting point could not reach to the end point at all. The main reason behind this was choosing wrong directions to move for the control input of the RRT algorithm.

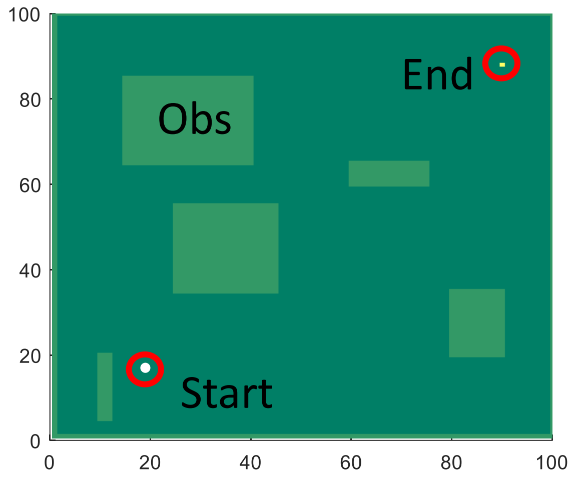
Second challenge here was to choose the optimum value for the Euclidean distance. Larger values for Euclidean distance may results to faster motion planning algorithm but it usually causes an error because the new created points exceed array bounds. So basically, the Euclidean distance should be an integer value because we have a grid map and the minimum distance the robot can move is one.

Third challenge here was to check whether the new created point is colliding with any obstacle or not. In this section, which is a part of the RRT algorithm, first I could not figure out a proper algorithm and the new created nodes could not detect the obstacles and they were passing the obstacles on top of them. I finally just put the algorithm to check whether the node is occupied or not.

In this project, I originally planned to have a point robot in a 2D space with couple of polyhedral obstacles which was **completely achieved**. I planned to have holonomic motion model in which its controllable degrees of freedom is equal to the total degrees of freedom which was **achieved**. I planned to implement the Rapidly exploring Random Tree (RRT) algorithm in a 2D map which was completely **achieved**. And finally, I really liked to add a shortest path finder algorithm to this project to find the shortest path between the start and end point and this was also **achieved** although actually the algorithm was not written by myself. I was planning to add the non-holonomic motion model to the project which is **not achieved** yet due to the lack of time. I also wanted to implement kd-trees algorithm to increase the efficiency of the RRT algorithm, which is super interesting to me, but I couldn’t actually implement it. Hopefully I can make it before the presentation session. Another part that I was trying to work on it and I am still working on it, is to implement the bidirectional RRT which is more efficient that the classic RRT. This is also not achieved yet but I am hoping I can finish it before the meeting. The most important part that I am currently missing is the ROS visualization part which unfortunately I am still working on it and I do my best to present it at the meeting. This is my first time using ROS and it is very challenging for me to connect MATLAB and ROS together.

Examples:

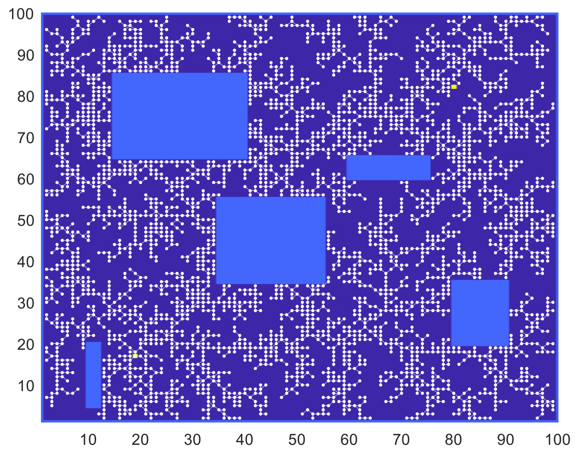
Here I would like to show couple of interesting examples. First let us see the default map with obstacles:



Now we can try different Euclidean distances and compare them together by the number of iterations it takes to reach to the end point:

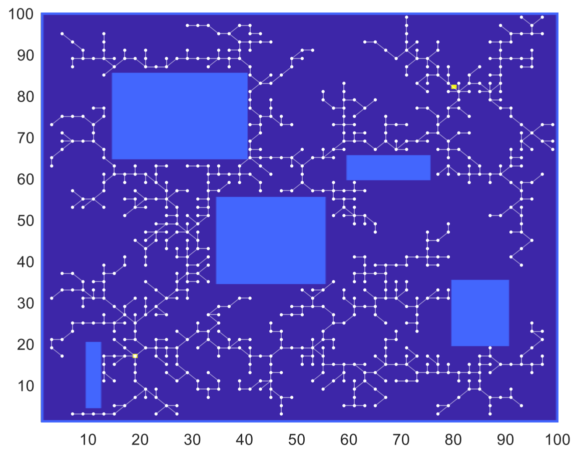
Euclidean Distance= 1 ,

# Iterations = 4319

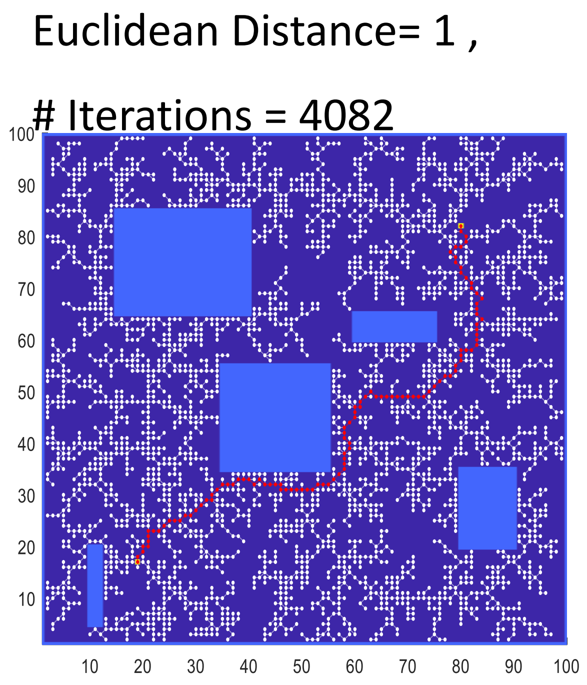


Euclidean Distance= 2 ,

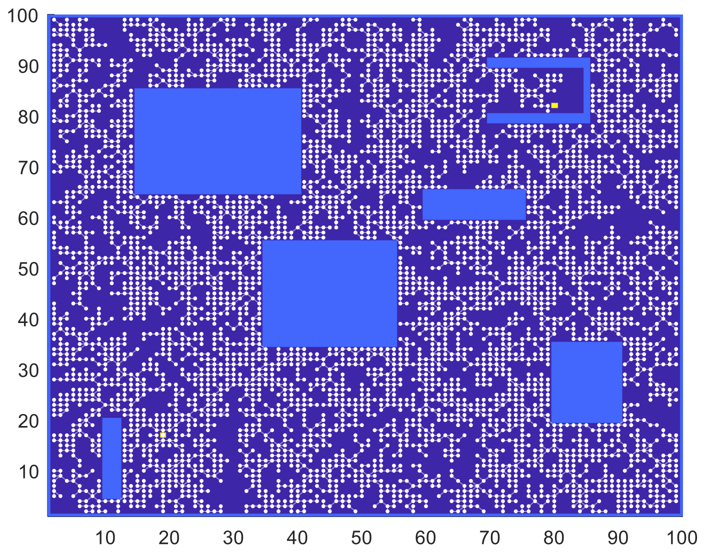
# Iterations = 955



Next, I would like to show an example where after we reach to then end point, by using the predefined command in MATLAB, we can see the shortest path between two points. The following example is done in a 2D map, with obstacles and the Euclidean distance of 1:



And one final example that the proposed algorithm could not solve it and reach to the end point is the following example:



In this example, I put the end point a box which was open from one side and interestingly, the RRT algorithm could not reach to the end point in a timely manner (My system could not run the simulation more, so I stopped it at this stage). I am still working on this example as well to find out the reason behind this.

References:

1- Course Lectures (Lecture 20 & 21)
2- http://lavalle.pl/rrt/projects.html
3- https://github.com/olzhas/rrt_toolbox
4- https://github.com/shrestha-pranav/RRT
5- https://github.com/EwingKang/Dubins-RRT-for-MATLAB
6- https://github.com/cazevedo/rrt
7- https://github.com/adnanmunawar/matlab-rrt-variants
8- https://www.cs.cmu.edu/~motionplanning/lecture/lec20.pdf
9- https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree
10- https://www.mathworks.com/products/ros.html