MOBILE ROBOTICS REPORT

LAB 2
PATH-PLANNING

Submitted by:

ARIJIT MALLICK
SIVANAND YADAV KATAMANI

Professor:
Philippe Martinet

MAPPING BASED ANALYSIS

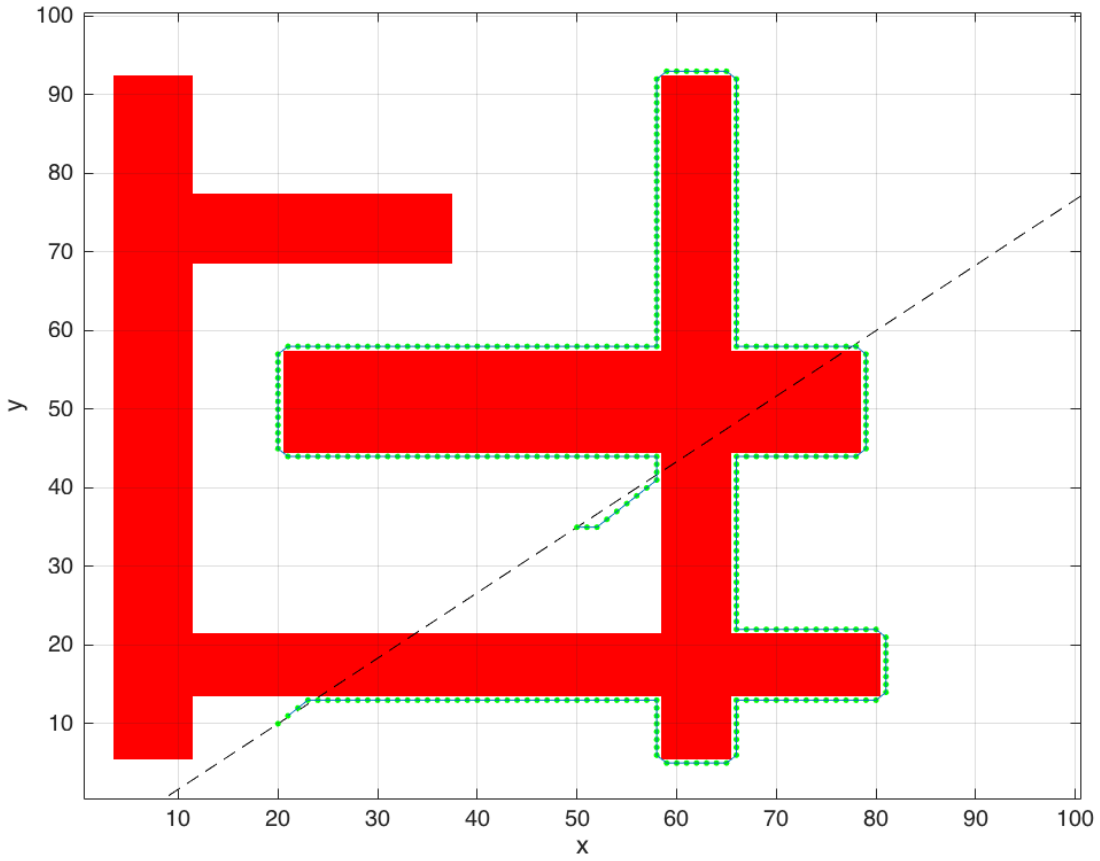
I. Bug2 algorithm

As the name suggests, the Bug2 algorithm is an Insect-inspired algorithm. In this case, it is assumed that the direction to the goal is known, and the robot can measure distance between the given two points. In short, let's assume that the dotted line from the starting point to the goal point is called the M-line. The robotic movements are based on this line in Bug2 algorithm. In short, we can say that the algorithm briefly does the following:

a. The robot heads towards the goal following the M-line.
b. If there is an obstacle, the robot will follow the M-line until a collision with the obstacle, and follows it until it finds the m-line again.
c. Once M-line is found again, it will leave the obstacle and head towards the goal again.
d. If it finds obstacle again, it will repeat the sequence 'b' and eventually 'c'.

In our lab, we have used the robotics toolbox and vision toolbox of Peter Corke in order to generate the paths and also find optimal solutions, including algorithms. We have done the following analysis :

A. <u>Pre loaded test map analysis</u>
Following graphs show the results of pre -loaded map analysis:



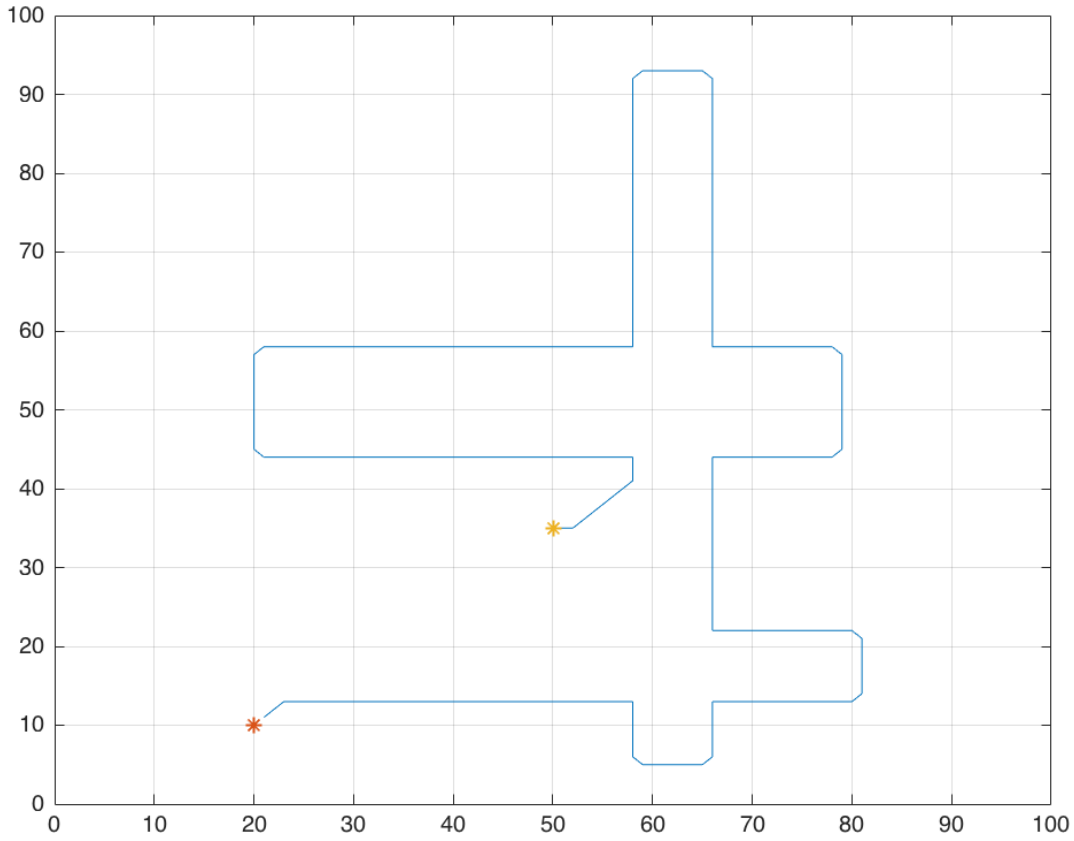
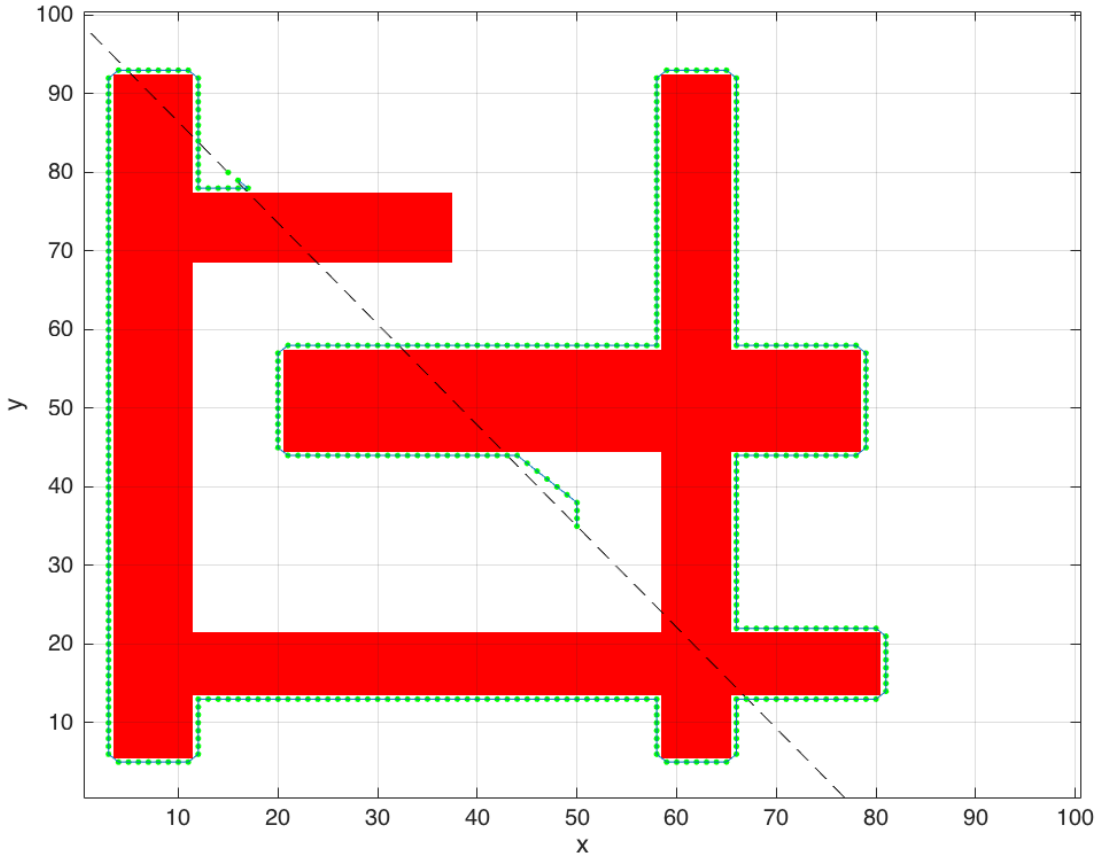


figure 1a and 1b : Bug2 algorithm being executed with start point (20,10) and goal point (50,35). Second image shows the extracted path from the main map.



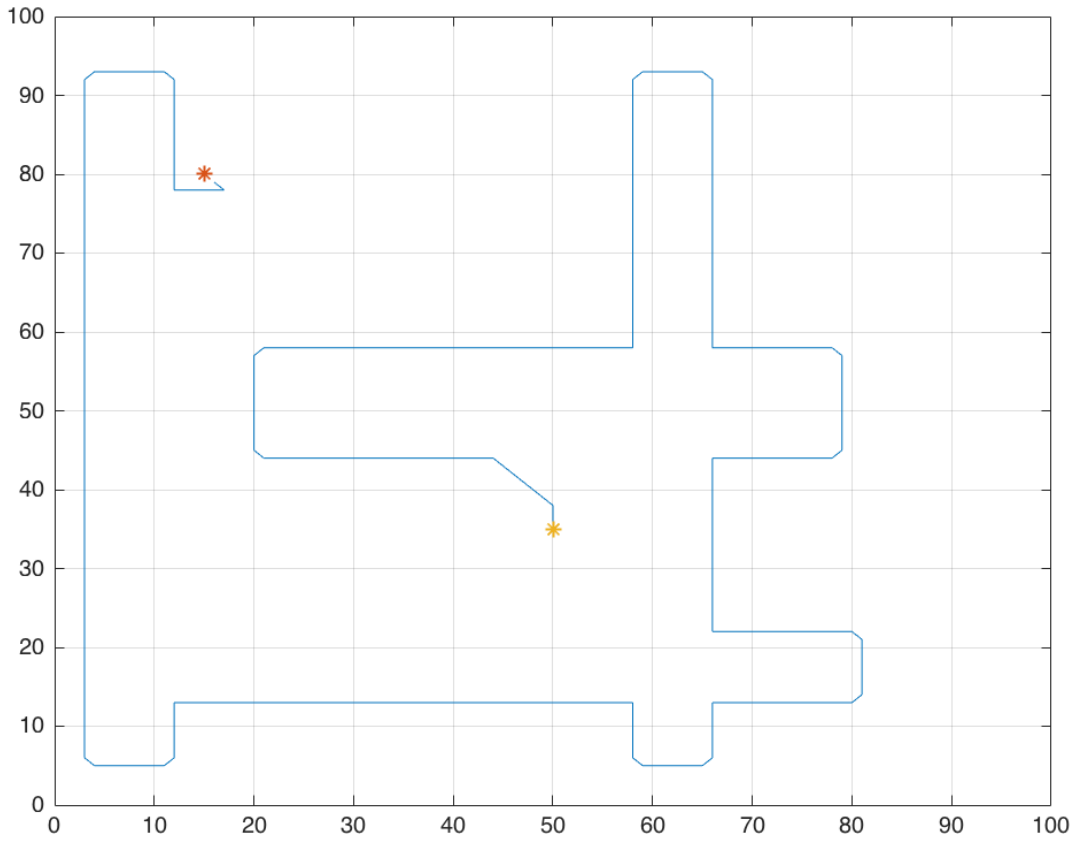


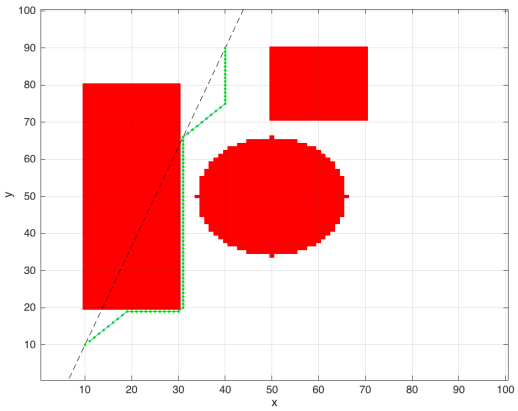
figure 1c and 1d : Bug2 algorithm being executed with start point (15,80) and goal point (50,35). Second image shows the extracted path from the main map.

Following additional Matlab code has been used to extract the real path:

```
figure(2)
plot(m(:,1),m(:,2));
hold on
plot(start(1,1),start(1,2),'*');
hold on
plot(goalx,goaly,'*');
grid on
axis([0 100 0 100]);
hold off
```

B. Custom made test map analysis

In this section, we will use our custom generated map, with the help of ‘makemap()’ command from the robotics toolbox. Following graphical representation denotes the aforesaid parametric changes and its corresponding outputs.



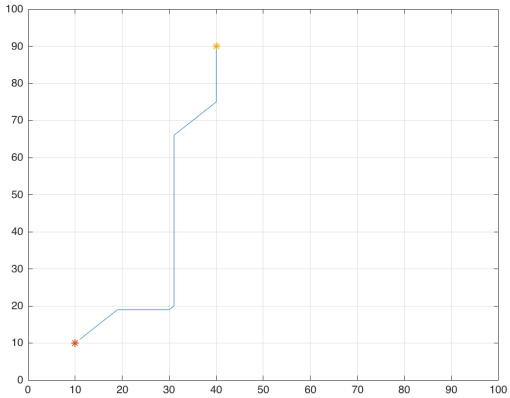


figure 1e and 1f: Running on custom made map, red point indicating start point and orange point indicating the finishing point in the graph (fig 1f)
Initial point: (10,10); goal point: (40,90)

II. BUG3 algorithm

It can be roughly observed from figure 1f that in Bug2 algorithm, the robot does not exactly follow the M-line, and rather has a tendency to deviate it. In the second case, we have proposed an addition to Bug2 algorithm in order to compensate for this effect. It will force the point (here, the robot) to return to the M-line, whenever it gets deviated. The following modifications has been proposed:

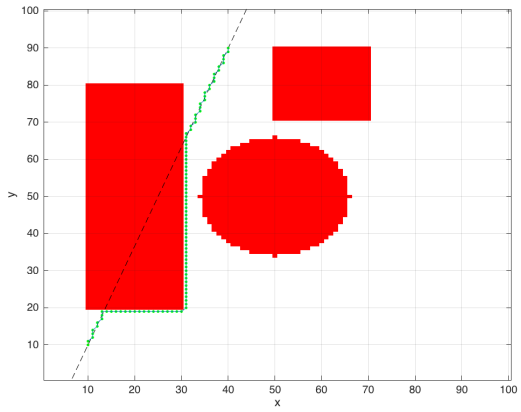
```
%% BUG3

length(3)=0;
xdir=[robot(1)+dx;robot(2)];
ydir=[robot(1);robot(2)+dy];

diag=[robot(1)+dx;robot(2)+dy];
% product gives a value for distance between mline and points

length(1)=abs( [xdir' 1]*bug.mline');
length(2)=abs( [ydir' 1]*bug.mline');
length(3)=abs( [diag' 1]*bug.mline');
% index shows the minimum distance to go,
[minm,I]=min(length);
if I==1
  dy=0;
elseif I==2
  dx=0;
  else
     dx=dx;
     dy=dy;
  end
```

The following motion plots demonstrates the effect of adding the Bug3 modifications.



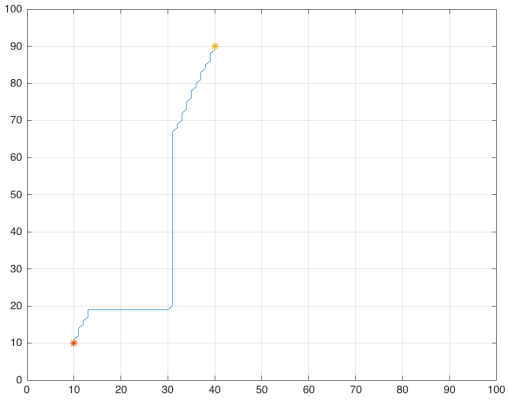
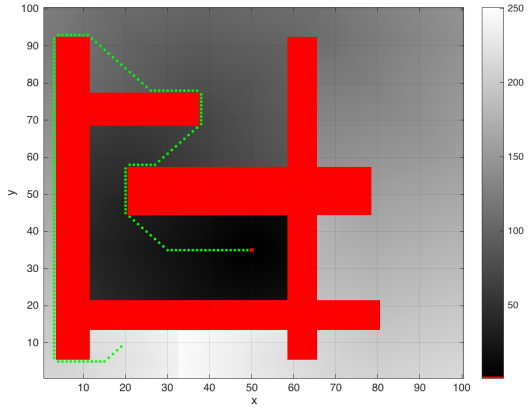


figure 1g and 1h: Aforesaid figures represent the motion change of the robot with modified Bug3 algorithm. It can clearly seen while comparing images 1f and 1h that the eventual deviation tendency from the M-line has been prevented.
Initial point: (10,10); goal point: (40,90)

DISTANCE BASED METHOD ANALYSIS

I. Distance Transform Method

As the name suggests, this kind of motion planning is based on the minimum distance computation analysis. Theoretically, it is very simple. The algorithm will calculate the minimum distance (Euclidean, preferably but Manhattan distance also present) between start point and goal point, including obstacles. We have simulated the algorithm using pre-loaded as well as custom maps. The following maps show the results.



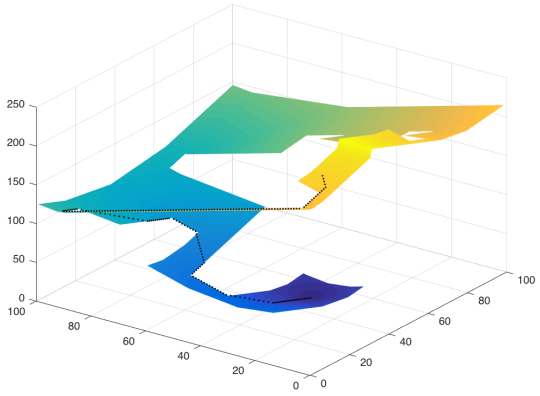


figure 2a and b: As the theory suggests, the distance points can be seen graphically. In figure 2a, the darker cell indicates that the pixels are nearer to the goal, as the corresponding intensity matching with pixel is shown in a bar graph form. In figure 2b, we can see the 3D form of this increasing and decreasing distance analysis. We can see that the point travels from a region of higher length to lower length. In this case, the path distance to the goal is denoted by the height of the graph.
Initial point: (20,10); goal point: (50,35)

In the second case , we will present a custom made map and show the behaviour of the DTM based method.

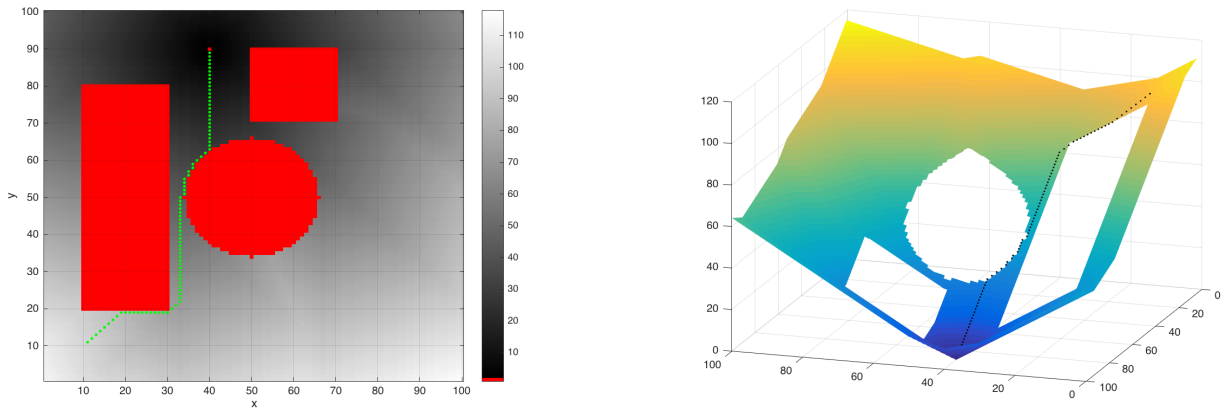
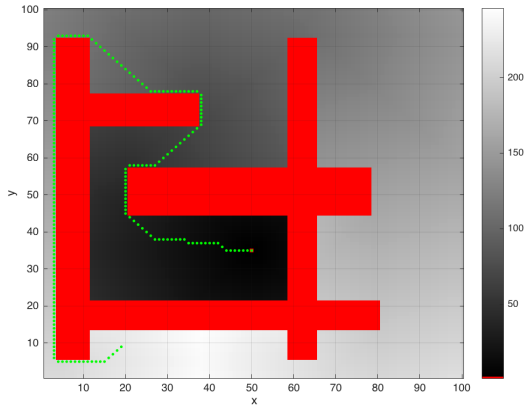


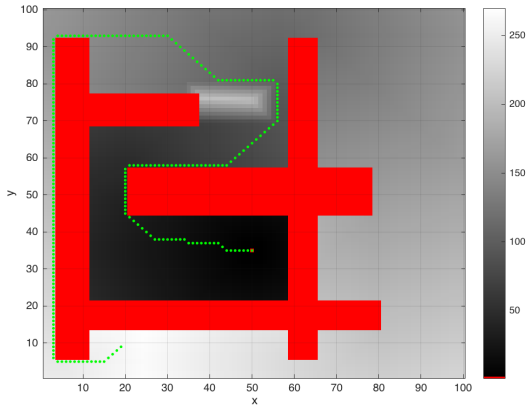
figure 2c and 2d: As expected, the robot would take the shortest path, and it tends to go to the darkest pixels region. In the next 3D plot, the robot goes to the lowest point, which is the goal.
Initial point: (10,10); goal point: (40,90)

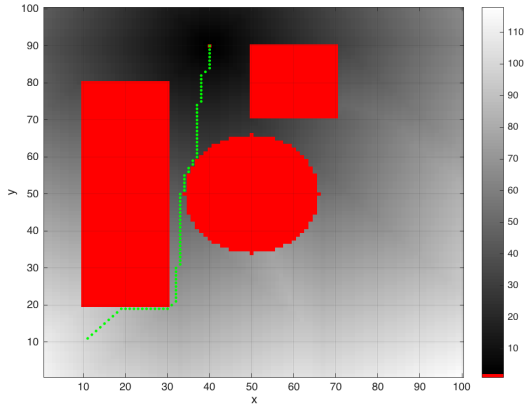
II. D* Algorithm

In this algorithm, the robot assumes that it has to navigate under unknown circumstances. The general idea is to navigate and record unknown terrains, and whenever it faces any obstacle, it will record it and re-plan its path, adding the new constraints. In case of a D* algorithm, the robot will analyse the path and set a cost to it, and eventually will take the minimum cost path. But the main advantage lies in the fact that if a new or a dynamic obstacle is faced, it will ultimately recalculate the path-cost online. There is no need for remapping from the start and assigning new cost values. In this algorithm, if a variable c is the cost to horizontal and vertical grids, the diagonal grids will have a cost of $\sqrt{2}c$. The cells representing obstacle is denoted by $c = \infty$.

Graphs in the following page shows the simulation in different terrains.







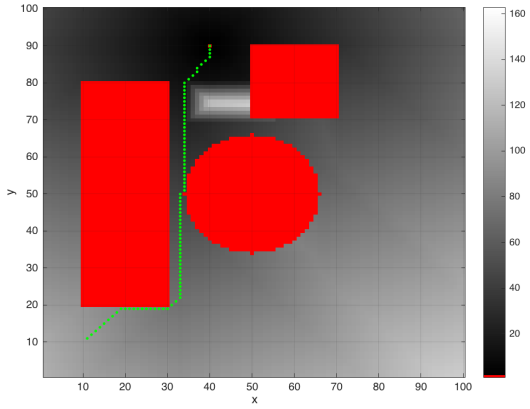
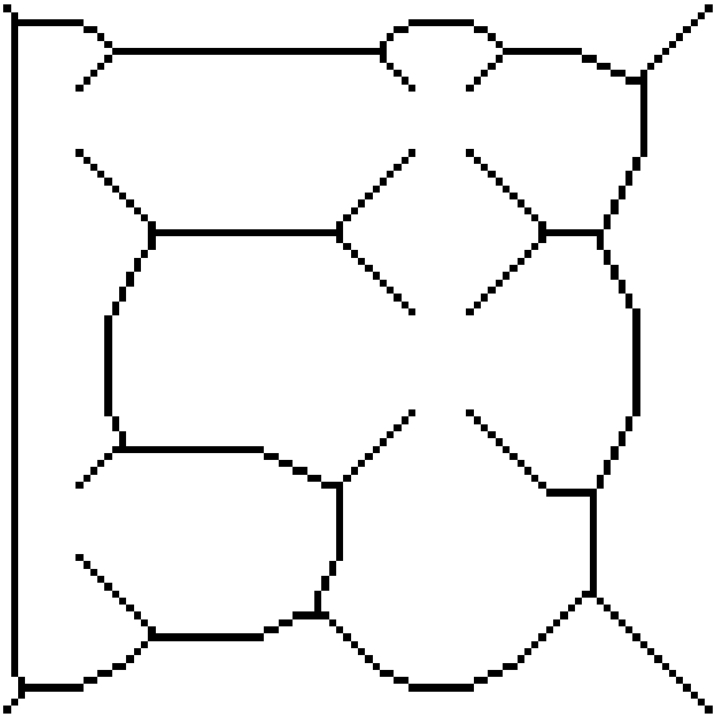
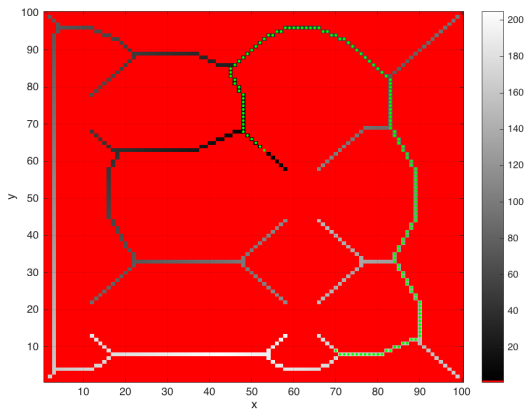


figure 2e,2f : The pre-loaded map shows the result as follows. The cost function is changed in the next image. The whiter area shows the introduction of high cost. It can be clearly seen that the robot will deviate away from the high cost path.
Initial point: (20,10); goal point: (50,35)
figure 2g,2h : The custom made map shows this property as well. The path deviates clearly a little bit away from the whiter region.
Initial point: (10,10); goal point: (40,90)

VORONOI GRAPH

Voronoi graph is a part of Roadmap Approach strategy for Robot motion planning. Normally, a generalised voronoi graph takes in the actual map as input and computes the locus of points equidistant from the closest two or more obstacle boundaries, including the workspace boundary as well. The target is to maximise the clearance between the points and obstacles. Whenever a start point is selected, it takes the path from that point and goes to the nearest network of voronoi graph lines, as will be shown in the simulations. So, in short, Moroni graph is obtained as a network of lines equidistant from the two closest object; hence generating a very safe roadmap to avoid the obstacles.





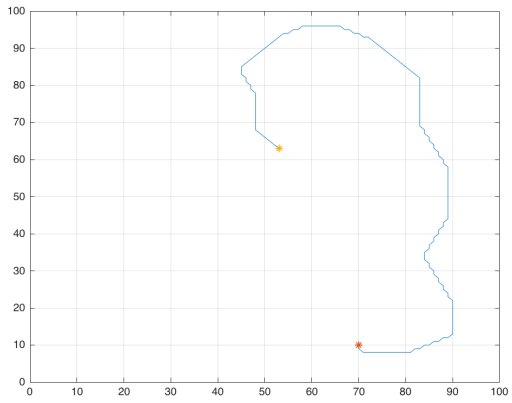


figure 3a,3b,3c: For the preloaded map, the skeleton graph is produced in step1 (fig 3a). Secondly, we get the optimum path along the lines, in Voronoi graph. Fig 3c shows the extracted path of the proposed plan from Voronoi graph.

figure 3d: The overall map with the path obtained from D* algorithm applied after obtaining the Moroni graph. Here Initial point: (70,10); goal point: (53,63).

PROBABILISTIC ROADMAP MAPPING: PRM

As the name suggests, this method is a probabilistic method which involves selecting a random array of points in the path. Normally, Distance based method and Roadmap approaches consume a considerable amount of time, which is not feasible in real life situation. Hence, PRM method is employed where a fixed number of random points are generated and eventually connected from start to goal point; obviously avoiding the obstacles in the path.
In the following section, we will demonstrate its application by simulation with the help of predefined functions in the toolbox.

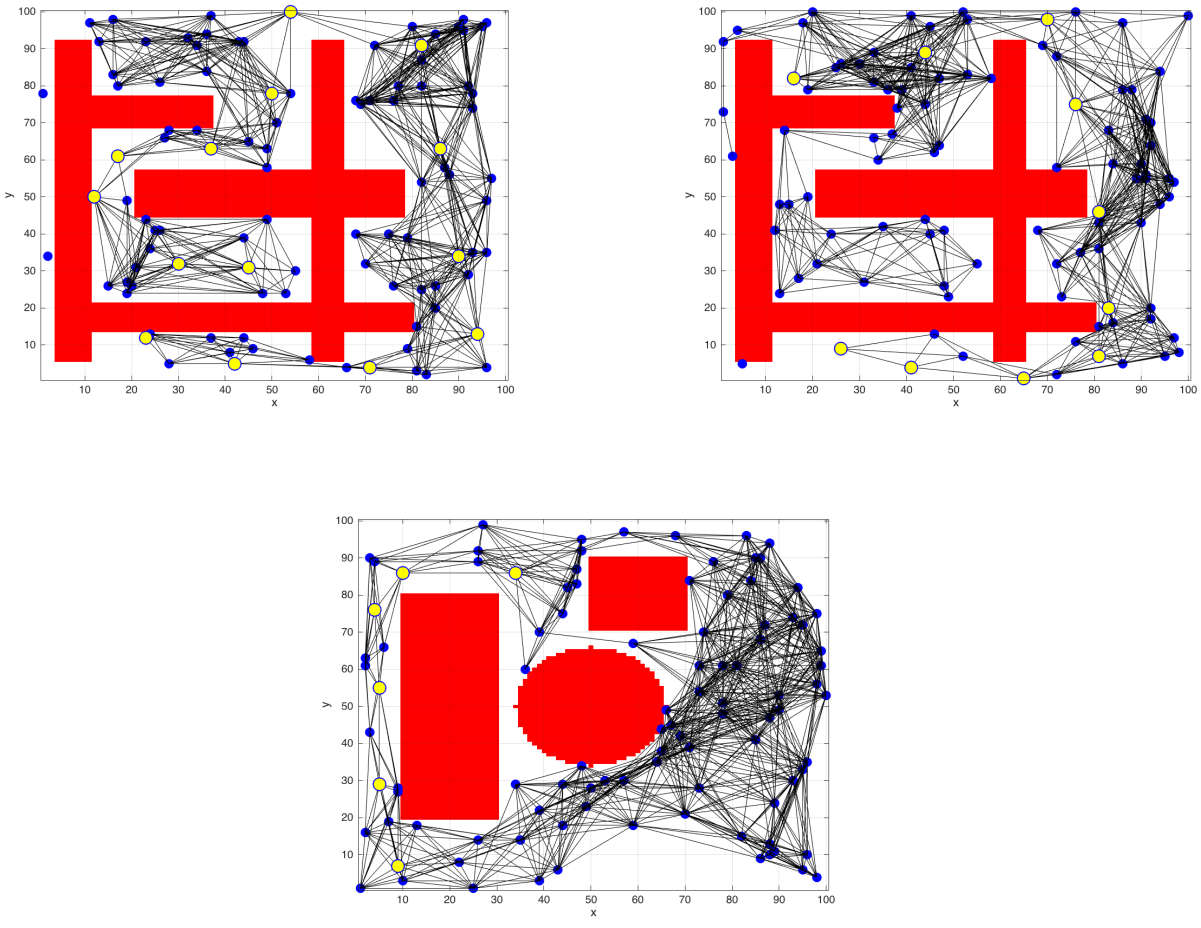


figure 4a, 4b, 4c: In this case, we have considered the robot to be as of a point object and tested it for different map and different starting and finishing points. fig a (upper left), figure b (upper right) and figure c (lower middle) has starting and beginning points as (20,10) and (50,35); (15,80) and (20,10); (10,10) and (40,90).

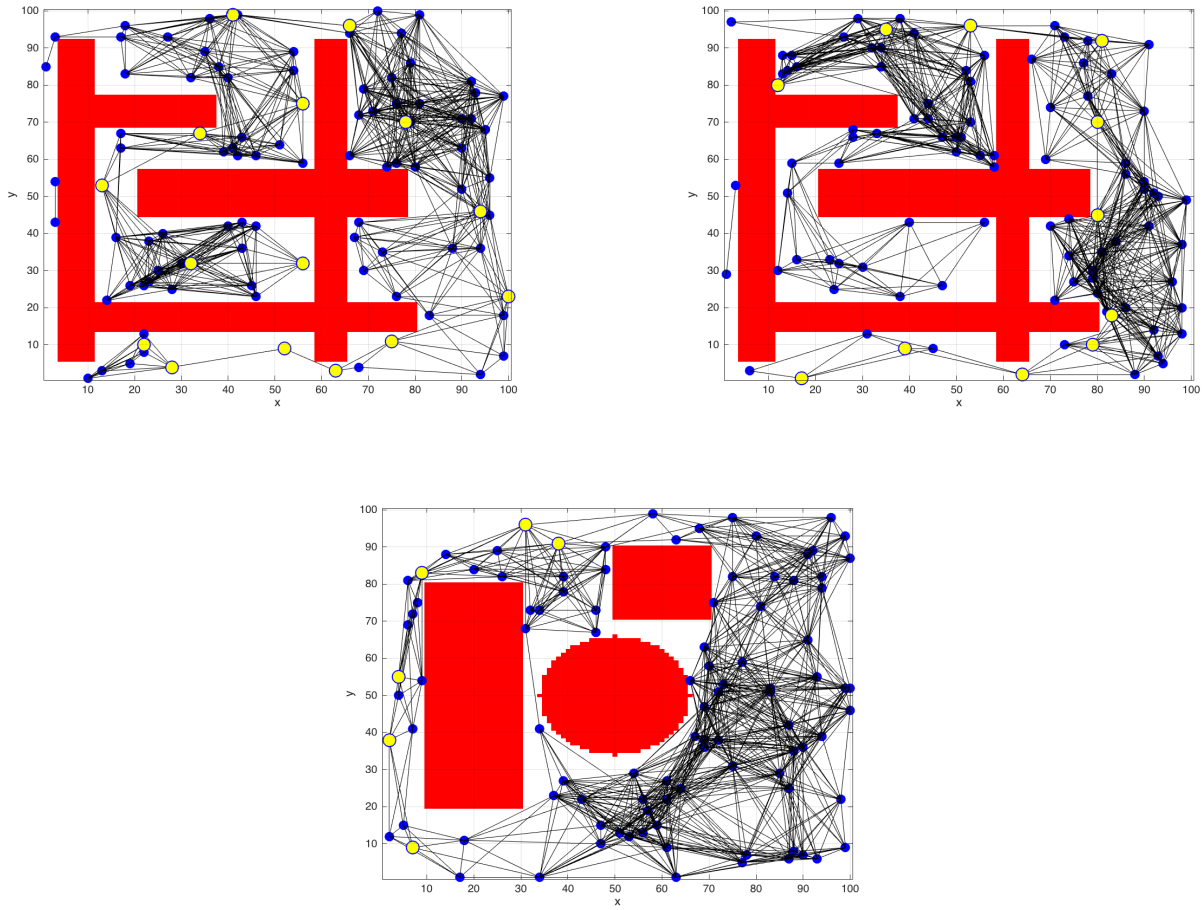


figure 4d, 4e, 4f: In this case,we have considered real life scenario assuming a constant size of the robot. Comparatively, this solution took some time in order to figure out the optimum path. In any case, this is the feasible scenario for a robot for path planning. fig a (upper left), figure b (upper right) and figure c (lower middle) has starting and beginning points as (20,10) and (50,35); (15,80) and (20,10); (10,10) and (40,90).