

Motion Planner for Differential Drive Robot

MEC559 Project

Joyce Chow, 112218043

Anthony Chen, 112063540

Group 10

Introduction

The RRT algorithm is a sampling based algorithm that is used to find a path in an environment from a starting point to an ending point. The environment can contain obstacles which must be avoided in the planned path. This motion planning algorithm can be implemented for differential drive mobile robots and the environment can be represented in many ways. In this project, the environment will be represented in an occupancy grid format. This means that the environment for the robot will be mapped out in a grid format, with obstacles taking up some of the grid space. The basic RRT algorithm consists of sampling random points within the open space of the environment and connecting these points together to find a valid path. The basic algorithm is good for finding a valid path, but it can be improved in many ways.

Problem Statement

The goal of this project is to further optimize the unidirectional RRT algorithm that was previously implemented. The algorithm will be optimized in three ways. First, multiple search trees will be implemented. In the previous RRT algorithm, there was only one tree that had one root expanding into the configuration space. The new algorithm will have two trees expanding bi-directionally from the starting point and the ending point. The second way to improve the RRT algorithm is to improve the sampling heuristic. Previously, nodes were randomly sampled uniformly in the configuration space. This sampling heuristic can be improved by only sampling new nodes within a certain distance from the previous node. The previous sampling heuristic could produce a node that is very far from the previous node, which increases the chance of the nodes colliding with an obstacle. This new sampling method will decrease the chance of an obstacle collision since it will be less likely for an obstacle to be in the way when the nodes are closer together. Lastly, the RRT algorithm can be improved by altering the way the tree expands

to other nodes. Previously, the RRT algorithm chose the nearest node to connect to, which does not provide the optimal path most of the time. To optimize this for the shortest path, all nodes that do not collide can be connected instead. The shortest path algorithm can then be implemented and a more optimal path will result, since all node connections are being considered instead of just the nearest ones. By combining these three implementations, the previous RRT algorithm will be further optimized.

The major variable names and definitions in the MATLAB code are detailed below:

q_0 = Initial pose

q_f = Final pose

$nodes$ = Maximum number of nodes to be plotted

x_{store} = Previous x-coordinate of node

y_{store} = Previous y-coordinate of node

$x_{current}$ = Current x-coordinate of node

$y_{current}$ = Current y-coordinate of node

$x_{storage}$ = Array to store all x-coordinates of nodes

$y_{storage}$ = Array to store all y-coordinates of nodes

s = Values of starting nodes

t = Values of ending nodes

D = Distance between nodes

G = Graph of nodes

Description of Our Algorithm

In HW3, the basic RRT algorithm is used for motion planning. However, for the project we took inspiration from the RRT* method, which greatly optimizes the path that the robot has to take to reach the end pose [1]. The optimized algorithm includes all three ways of improvement.

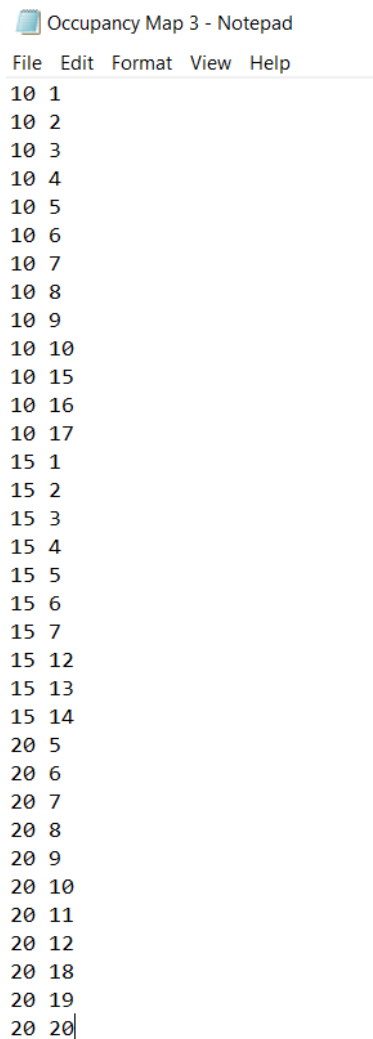
First, a bi-directional search tree is used. Using a unidirectional tree method and implementing it twice, we were able to make this algorithm possible. The first tree would consist of nodes that go from the starting pose to an approximate location towards the middle of the map environment. The second tree would go in the opposite direction, which is from the end pose to the approximate location of the middle of the map. Then, the two trees are connected and are checked for any collision. If there are no collisions in between the connection, a highlighted complete path is displayed. If there is a collision, an error message is displayed and the user is prompted to rerun the code.

The second improvement is the sampling heuristics. In Anthony's HW3, he lacks the use of a method to increment the nodes closer to goal. However, for this project, we made sure to choose our random nodes with a maximum distance in mind. By choosing a node that is within a certain radius of existing nodes, there is less of a chance of interference with an obstacle, which allows for the code to run smoother and faster. Without a method for this function, uniformly sampled nodes can be hard to connect as they are much further apart.

The third improvement is the heuristics for node expansion. In both of our HW3, we only connected the random node to the nearest node. Although this method creates a path to the end pose, it is not the most optimal path since it may be longer than necessary. To optimize this path, we connected all the nodes with each other as long as collision with an obstacle did not occur. By

doing this, the program now has many other options for path to take. Using the `shortestpath()` function on MATLAB, we were able to find which of these many paths is the shortest, which therefore optimizes the robot's path.

The occupancy grid maps were created with the help of several MATLAB functions in the Navigation Toolbox. The x and y values of the top right corner of the obstacles are typed into a text file and then read into the MATLAB code. The code then plots the environment. An example of the text file that is read is shown below:



```

File Edit Format View Help
10 1
10 2
10 3
10 4
10 5
10 6
10 7
10 8
10 9
10 10
10 15
10 16
10 17
15 1
15 2
15 3
15 4
15 5
15 6
15 7
15 12
15 13
15 14
20 5
20 6
20 7
20 8
20 9
20 10
20 11
20 12
20 18
20 19
20 20

```

Fig. 1 Text File For Occupancy Grid Map

The first column are the x values and the second column are the y values. Each set of points creates a 1x1 meter square in the grid map. By combining all these sets of points together, many obstacles can be created for the robot. Five different environments were created for this robot. It is noted, however, that the new MATLAB code only accepts the environment as an occupancy grid map and not the vertices method in Homework 3.

Pseudocode (Main Code):

1. Read occupancy grid file
2. PLOT obstacle environment
3. PLOT start and end pose
4. Set nodes = number of nodes desired
5. Set $x_store = x$ of initial pose and $y_store = y$ of initial pose
6. Implement `unidirectional_tree` function for initial pose to end pose
7. Implement `unidirectional_tree` function for end pose to initial pose
8. IF the end point from the first tree collide with the end point of the second tree
9. PRINT an error message
10. ELSE
11. Highlight the path between the points of the two trees

Pseudocodes (Unidirectional tree):

1. Initialize $q = 1$, $ii = 1$, $jj = 1$;
2. Set $x_storage = []$, $y_storage = []$, $s = []$, $t = []$
3. FOR n is less than or equal to the number of nodes
4. Obtain a random pose (this will be the current point)
5. Find the nearest point to the current point
6. FOR every value in x_store
7. Calculate the distance between the current point and the stored points
8. END
9. Find and store nearest point using distances calculated from the previous FOR loop
10. IF the distance between the current point and nearest point is less than the maximum

distance and is collision free

11. IF the nearest point does not exist in the storage variables
12. Store both the nearest point and current point into the storage variables
13. Store current index into s variable and (current index + 1) into t variable
14. Increment counter variables: ii+2 and jj+1
15. ELSE
16. Store index of nearest point into s variable
17. Store current point into storage variables
18. Store ii index into t variable
19. Increment counter variables: ii+1 and jj+1
20. END
21. PLOT current point
22. Store current point in the x_store and y_store variable
23. IF the current point is near the middle of the environment
24. Break
25. END
26. END
27. END
28. Highlight the shortest path

Results and Discussion

The previous algorithms used and their results are detailed below. Both group members' old algorithms are included to be compared to the new improved algorithm.

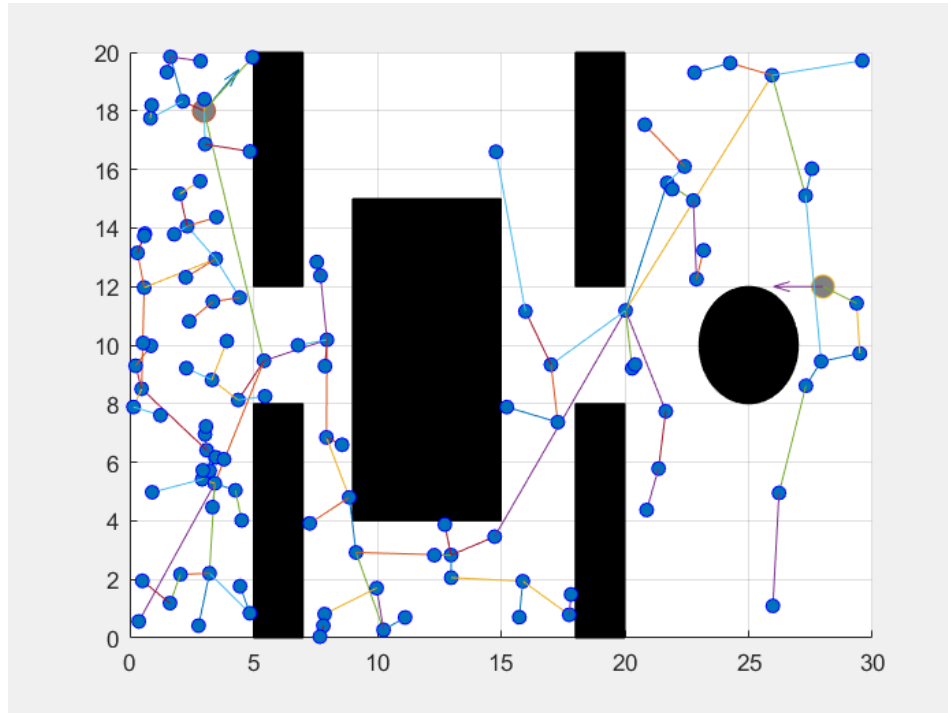


Fig. 2. Joyce's HW3 Algorithm Result

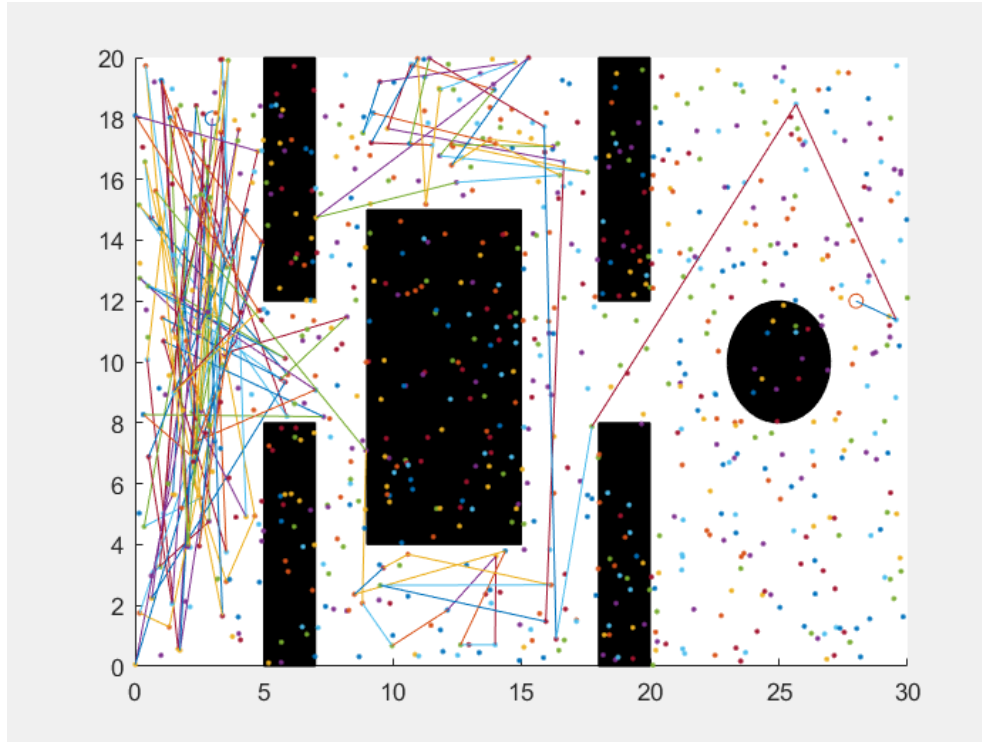


Fig. 3. Anthony's HW3 Algorithm Result

From basic observations, these previous algorithms do not provide the optimal path to travel from the starting position to the ending position nor do they highlight a path to the end pose. They are also unidirectional and nodes are randomly sampled within the configuration space. The result of the optimized algorithm for the same obstacle environment is shown below and the path is highlighted:

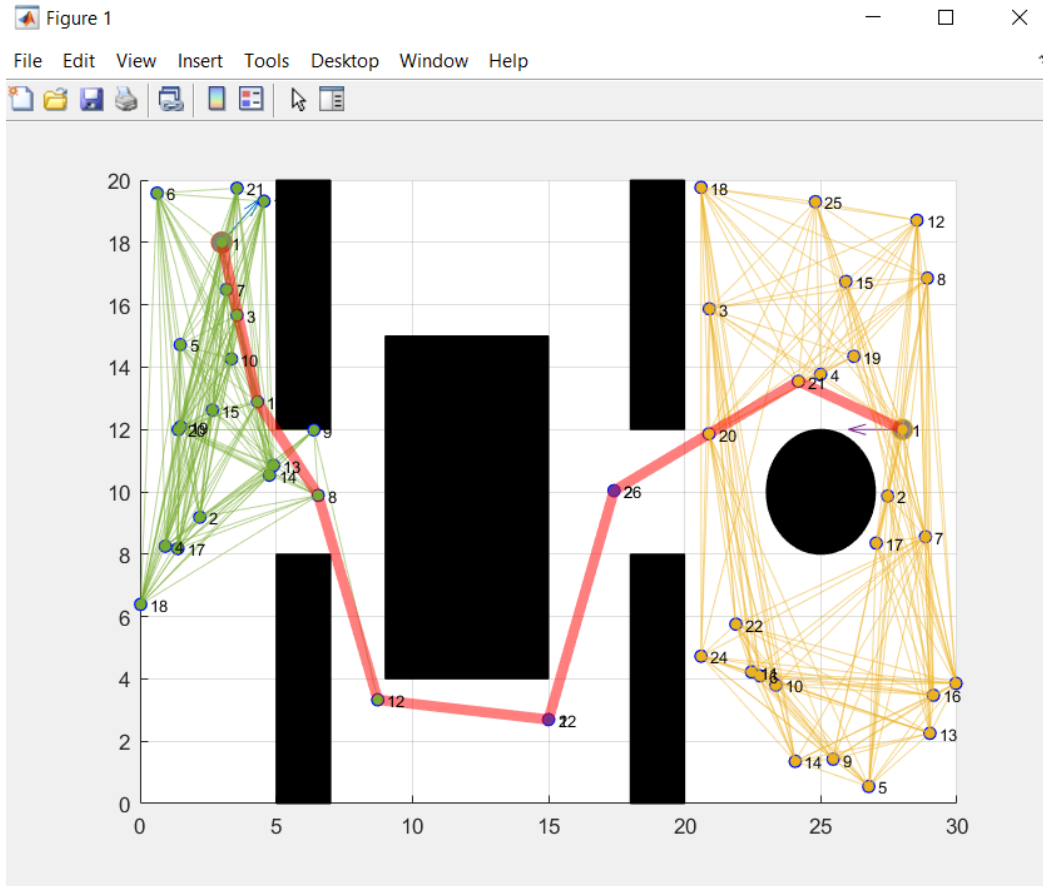


Fig. 4. Improved Algorithm's Path in HW3's Obstacle Environment

From observing the new algorithm, the planned path is greatly improved. The path is much shorter than the previous paths. The algorithm is bi-directional and samples new nodes only less than 10 meters away from the previous node. The shortest path function is used to determine the best path from the starting node to the ending node. The map above is the one from Homework 3, which was made using the vertices of the obstacles, but five more occupancy grid maps are tested with the new algorithm. The results are detailed below.

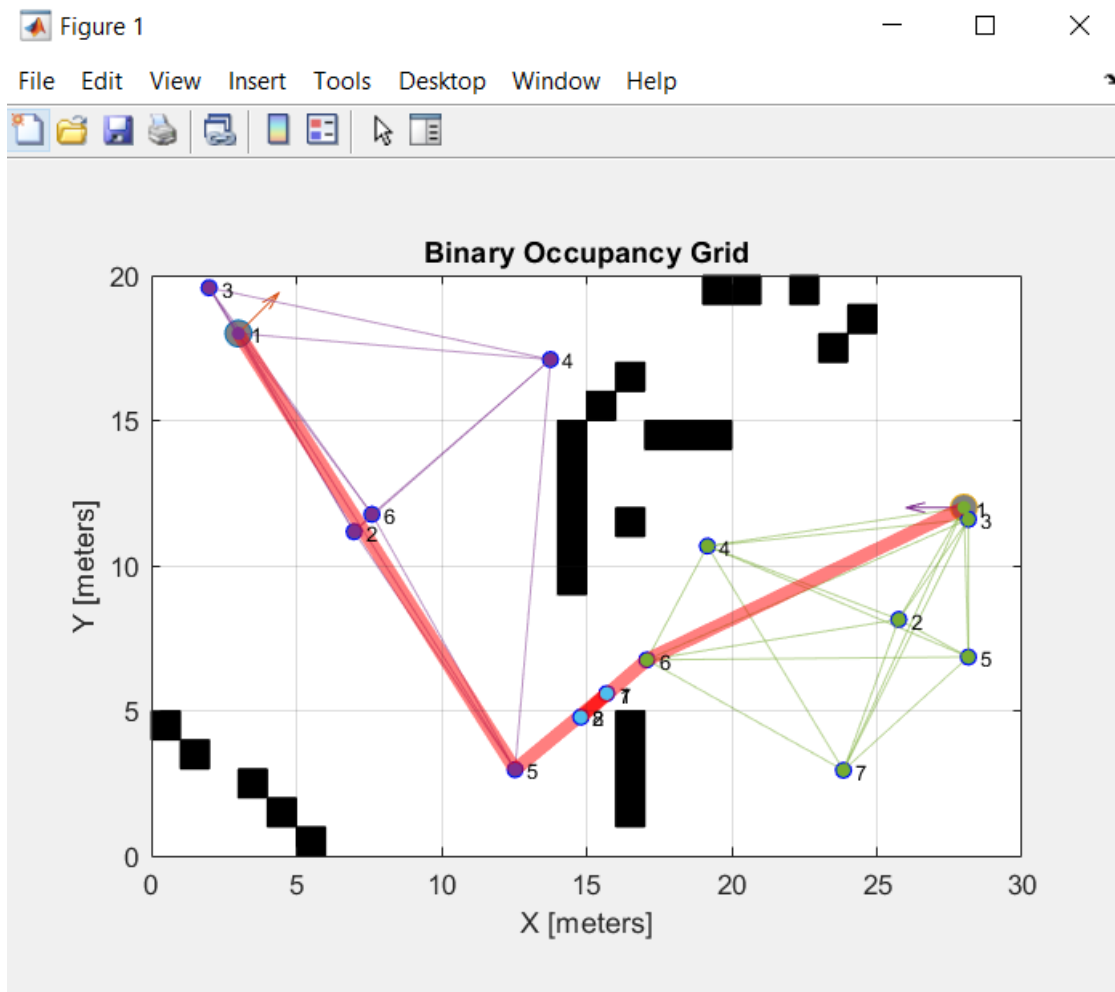
Map 1:

Fig. 5. Improved Algorithm's Path in Obstacle Environment 1

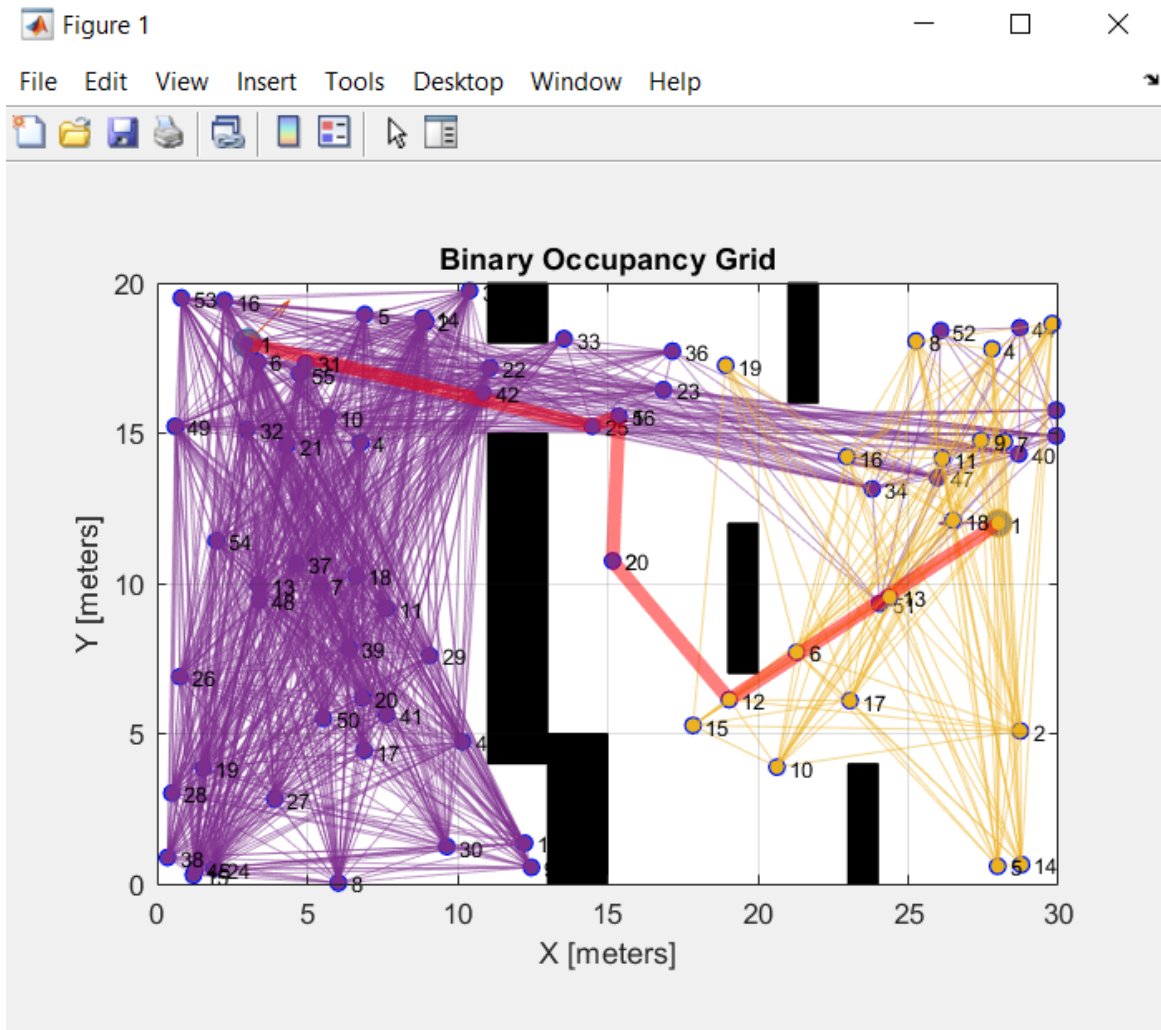
Map 2:

Fig. 6. Improved Algorithm's Path in Obstacle Environment 2

Map 3:

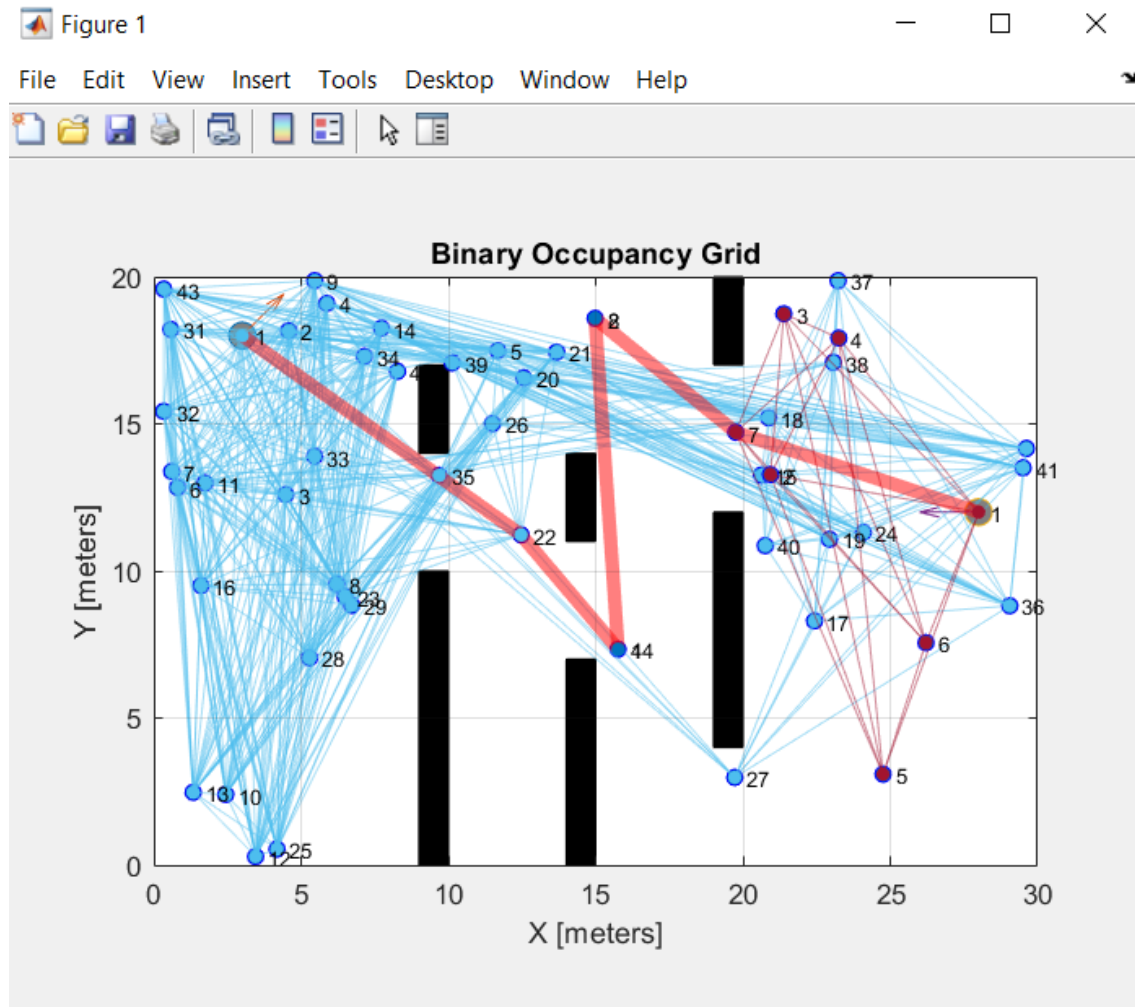
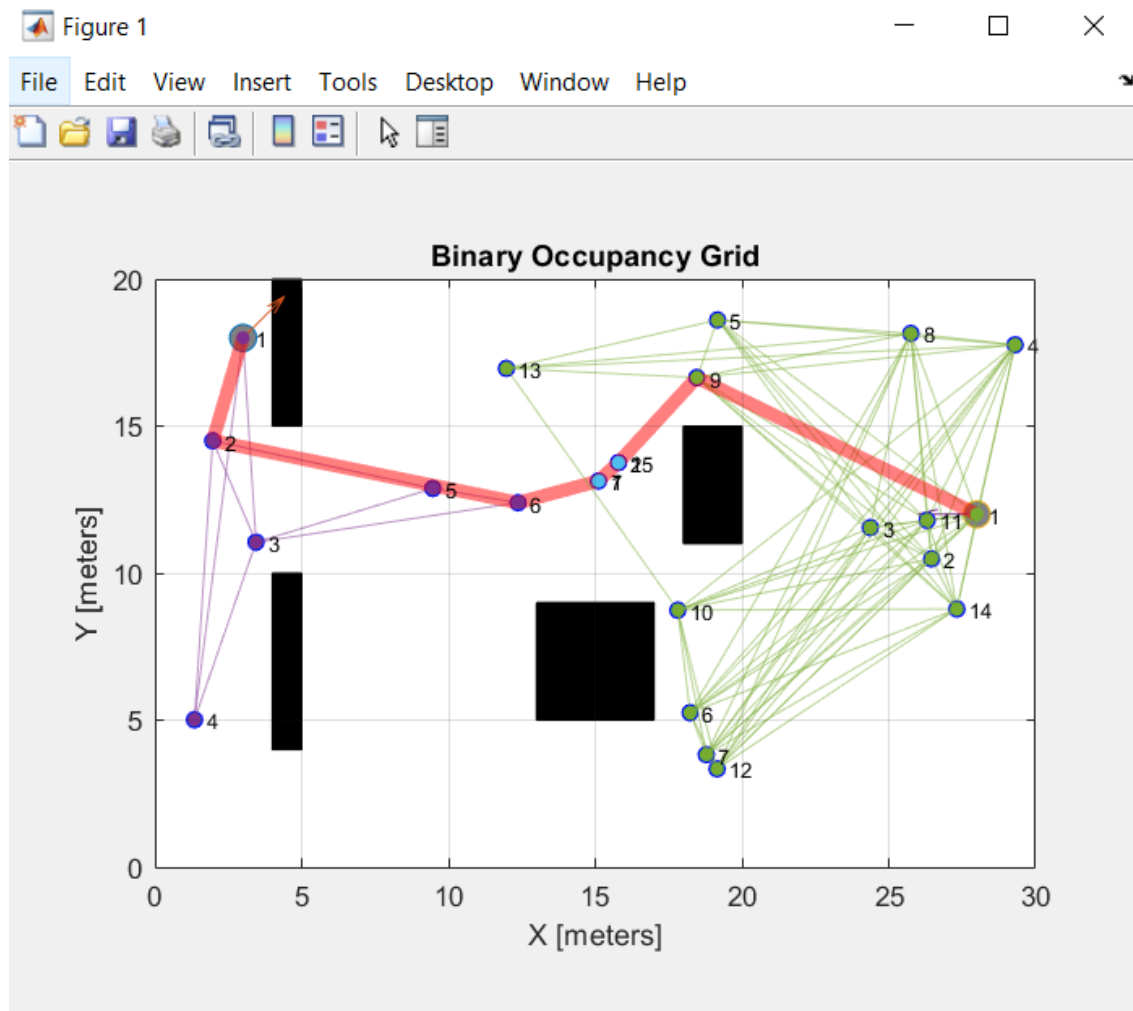


Fig. 7. Improved Algorithm's Path in Obstacle Environment 3

Map 4:*Fig. 8. Improved Algorithm's Path in Obstacle Environment 4*

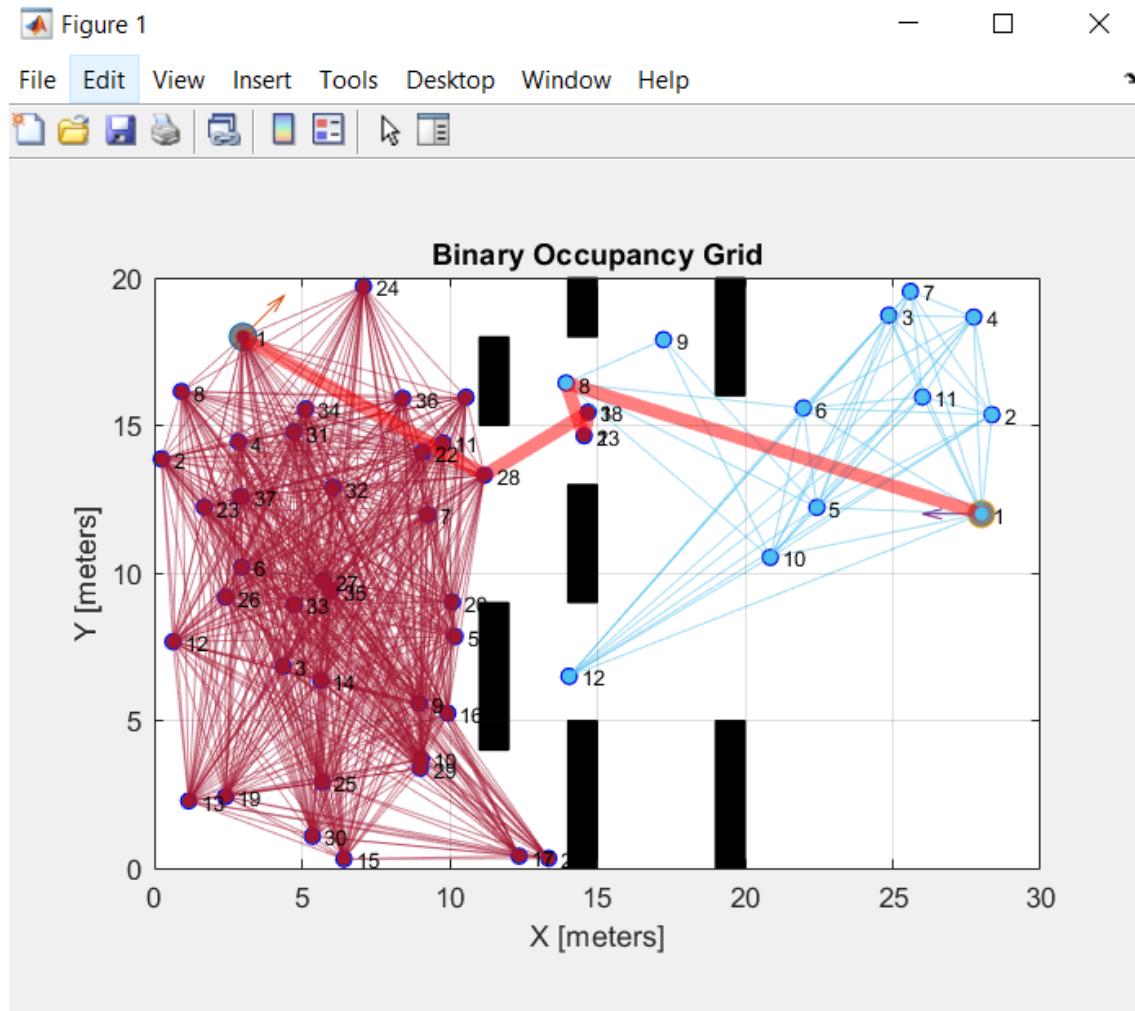
Map 5:

Fig. 9. Improved Algorithm's Path in Obstacle Environment 5

From looking at the way the new algorithm acts in these five occupancy grid maps, an optimal, but not perfect path is planned out for the robot. For each map, nodes are sampled from both the starting and ending nodes and each possible path is connected between the nodes. This happens on both sides until the midpoint of the map is reached. The nodes will then attempt to connect to each other from the starting tree to the ending tree. The shortest path is then selected and this returns an optimized path from the previous algorithm. In the event not enough samples

are produced to form a path or the path cannot be found, an error will be produced and the code will prompt the user to run the algorithm again.

Error Case When Path Cannot Be Found:

In the case of an error, the code prints a message that prompts the user to rerun the code. This may be due to unlucky random sampling or insufficient node input. By having more nodes, there is a higher chance that a path can be found.

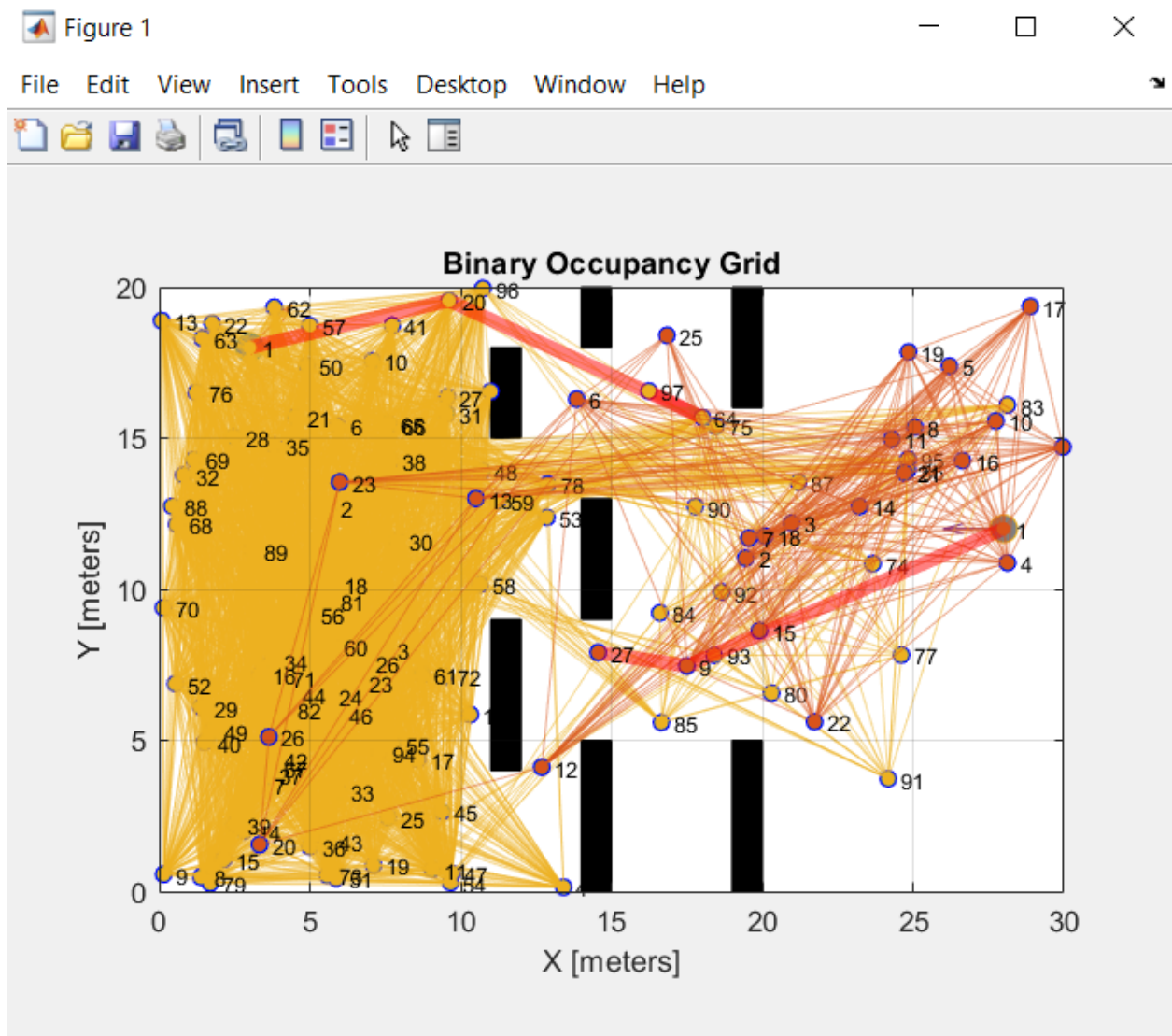
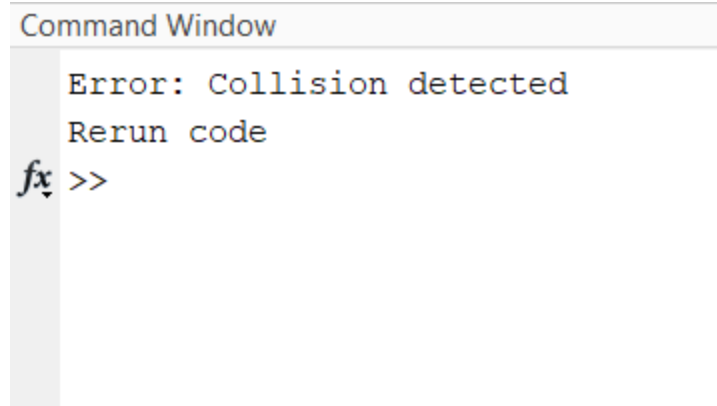


Fig. 10. Error Case When Program Cannot Find Path



```
Command Window
Error: Collision detected
Rerun code
fx >>
```

Fig. 11. Error Message When Error Case Occurs

Conclusion

Overall, the new motion planning algorithm improves on the previous one. The previous one may have produced a valid path, but certainly not an optimal one. Although the new algorithm is not perfect, the new path produced is much shorter and efficient than the previous one. By improving the algorithm by using multiple search trees, a better sampling heuristic, and an improved heuristic for the expansion of nodes, the more optimal path was found. The algorithm can even be further improved by optimizing the efficiency of the run time of the algorithm, but there was not a significant difference between the run time of the new algorithm and the old one. Although more connections between the nodes were created, the goal was to obtain a more optimal path for the robot. This goal was reached in the end and it proved to be effective throughout the five different environments.

References

- [1] MATLAB, 2020, “Path Planning with A* and RRT | Autonomous Navigation, Part 4,” Video.
<https://www.youtube.com/watch?v=QR3U1dgc5RE>
- [2] erbal, 2015, “How to check whether two lines intersect or not?” StackOverflow,
<https://stackoverflow.com/questions/27928373/how-to-check-whether-two-lines-intersect-or-not>
- [3] MATLAB, 2022, “shortestpath,” MATLAB,
<https://www.mathworks.com/help/matlab/ref/graph.shortestpath.html>
- [4] MATLAB, 2022, “setOccupancy,” MATLAB,
<https://www.mathworks.com/help/nav/ref/binaryoccupancymap.setoccupancy.html>