Module 4 Probabilistic Roadmap (PRM)

Johanna Martinez-Felix
Mechanical Engineering Department
California Politechnic University,
Pomona
Pomona United States
johannafelix@cpp.edu

***Abstract*—The mission of this module is to plan a path to a goal position from the current position point. The methos of implementation requested is by single query probabilistic roadmap. This method selects a random node to expand to give various possible paths. When a path is found near the range of the goal node the random selection of nodes stops and the path that was found is reconstructed from goal node to the start node.**

***Keywords*—path, single query, probabilistic roadmap, range, random**

I. Introduction

Single query probabilistic road mapping (PRM) offers a quick way to find a path form start node to goal node without hitting any obstacles. It does this by selecting random nodes and checking if the node is in free space or if its an obstacle node. Then it is connected to some random neighbor. The algorithm continues to search until and edge node is within region of the goal node or until the limited number of iterations allowed has been reached. Figure 1 shows a probabilistic road map example; node that all the nodes are connected in this example, the distance from node to node varies and obstacles were avoided. When a path is found from the start node to a node in the goal range the path is reconstructed moving backwards from end point to start point.

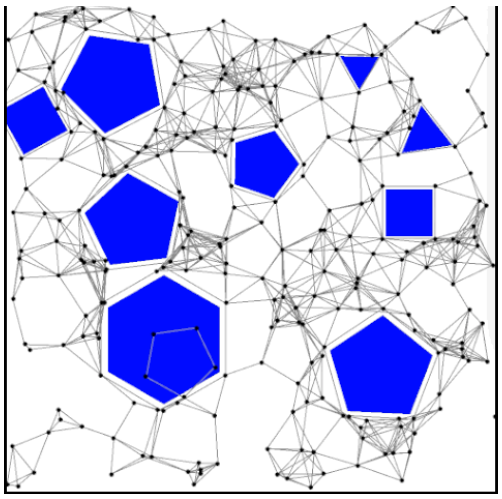

Figure 1 generated probabilistic road map

There are different methods to improve finding a path from PRM. One could implement a rapidly exploring dense tree RDT. An RDT is a dense topology tree that thru bias wants to search the unsearched areas of a graph. Under this method of random probability search a KD(k-Dimensional) tree, a type of binary search tree, can be used to break up an edge to get to a given node sooner. The tree can also be implemented and manipulated in a way that allows the tree to grow towards the goal node.

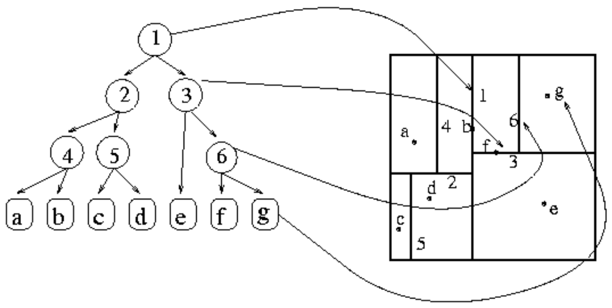

Figure 2 example of KD tree for visualization

Using the KD tree method, a bi-directional growth can also be implemented. Bi-directional grown means that path growth begins from both the start and goal node as shown in Figure 3. In this search methods algorithm, the stop point of the bi-directional growth tree is when an edge node stemming from the start node and a edge node stemming from the goal node are equal to each other. However, in this method an addition al step would be implemented so that the two trees coming from the start and goal are balanced and grow cohesively. The additional step is in the for of a "SWAP"; not swapping would result in continuing tree growth.

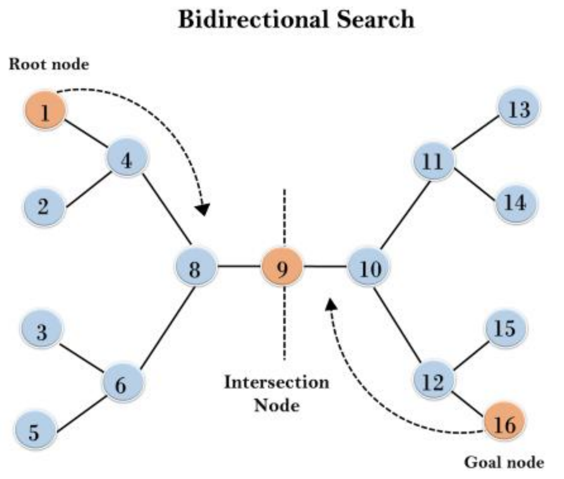

Figure 3 Bi-directional tree search for visualization

II. Method Description and Results

For this assigned module only the single query probabilistic map will be implemented as instructed on the assignment page. The first task is to put the start node in the road map with the nodes and edges. Here, the road map is an appendix list. The second task is to select a random node to expand(c).

As we know, there is still a gap in computer science when it comes to achieving real randomness. There are several ways to generate a “random” number. In this module we will use pythons random import library. This will help us achieve the second task. The randomly selected nodes to expand become stored in a list.

The third task is to generate a new random node (c’) from the randomly selected node we chose to expand previously (c). The random selection is similar to the

The fourth task is to check if we have hit an obstacle. If it is found that no obstacle has been hit the node edge can go into a list of viable paths to take in free space. If an obstacle is hit it does not get recorded in the viable paths list. Results for three maps running different number of iterations for this steps method description can be seen in Figures 4 thru 9.

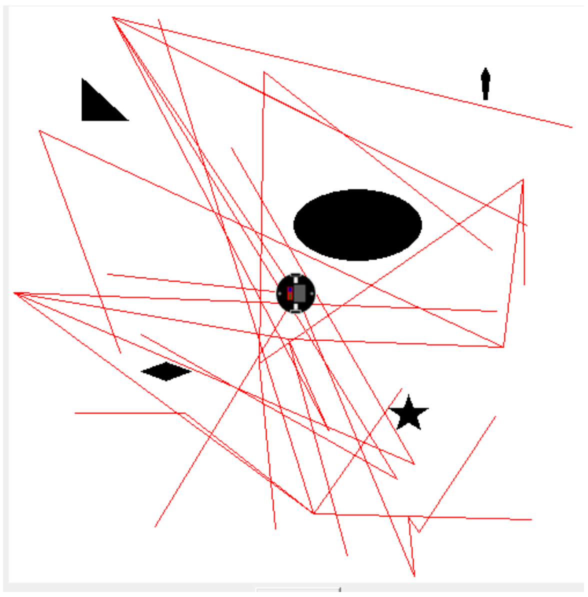
Figure 4. Map 1, 50 Iterations

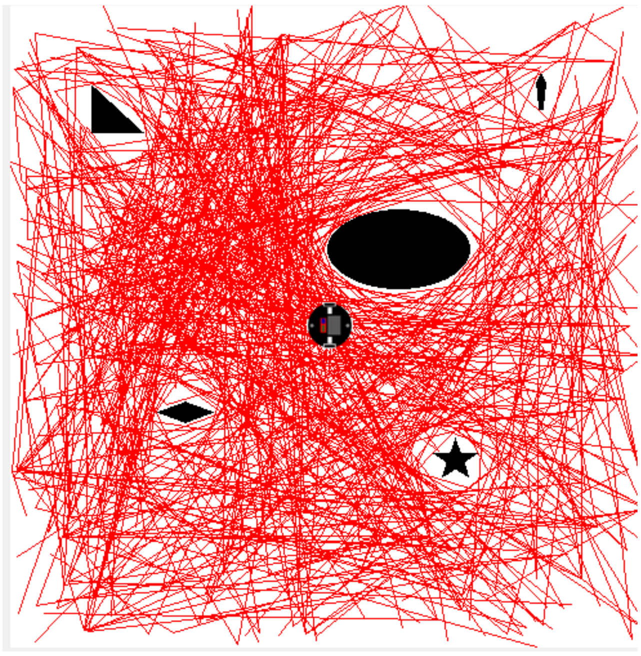
Figure 5. Map 1, 1000 iterations

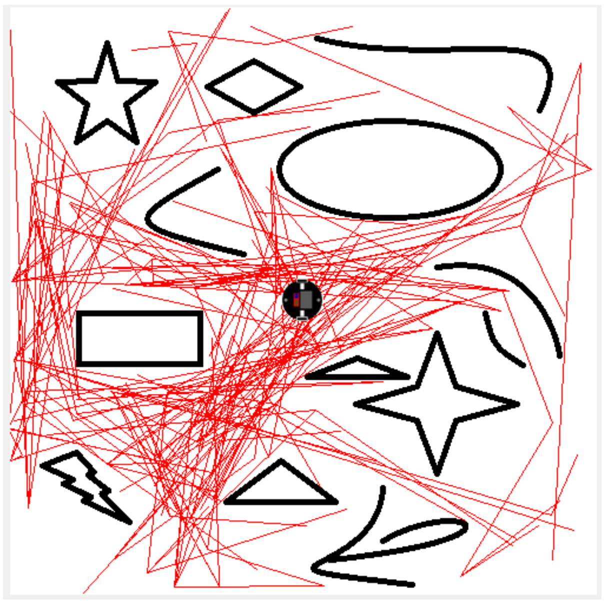
Figure 6. Map 2, 1000 iterations

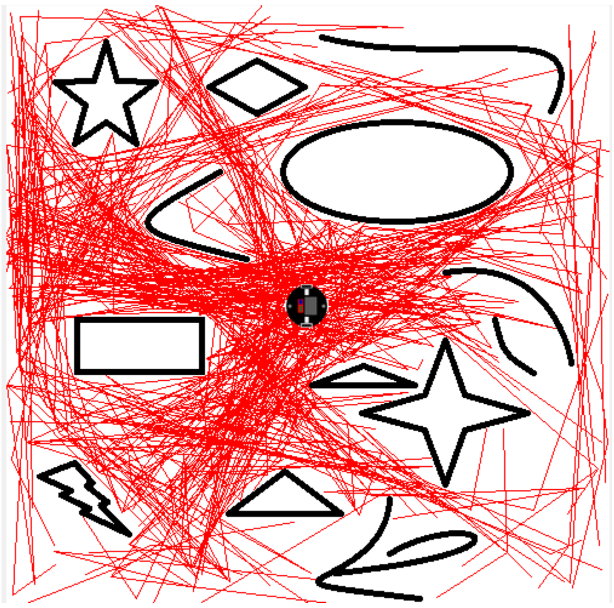
Figure7. Map 2, 3000 iterations. More nodes NOT added than added

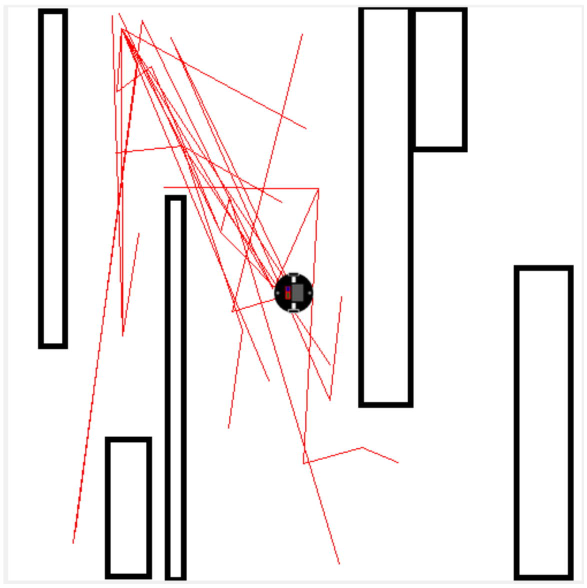
Figure 8. Map 3, 100 iterations

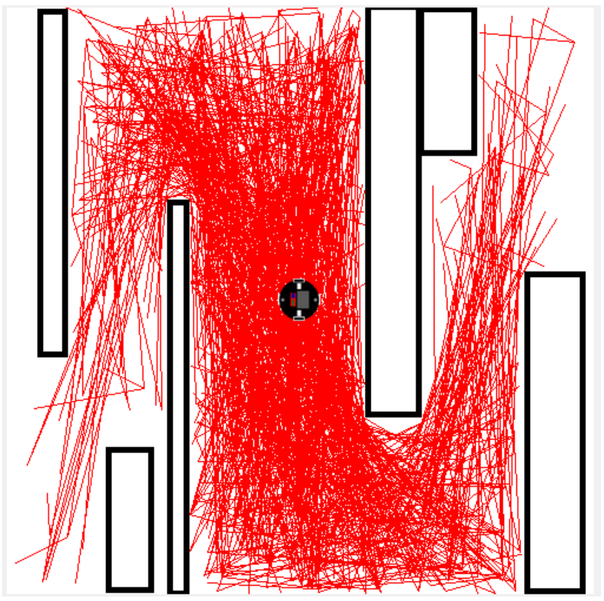
Figure 9. Map 3, 3000 iterations

The fifth task is to stop searching or expanding nodes if we have found a node that is with in region of the goal node. Because the nodes that are being searched are random it can be difficult to find the exact goal node; the algorithm can go on forever without getting the exact goal, thus we redefine hitting the goal as being withing a certain region of the goal. This will also allow a solution to be found even faster. The number of random nodes searched is also limited so in the case that there is no possible path to the goal the algorithm has a stopping point.

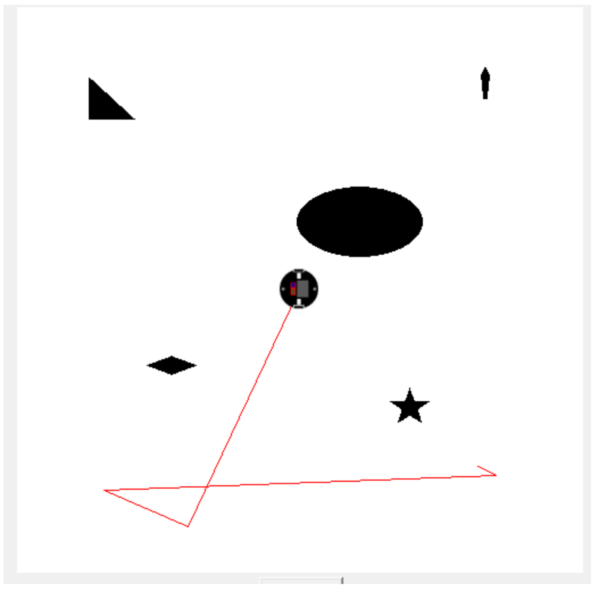
Figure 10. Map 1, 473 Iterations

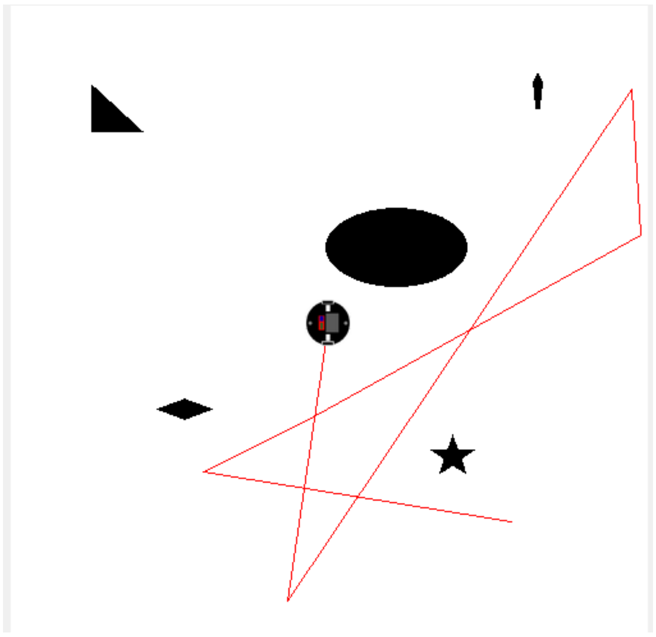
Figure 11. Map 1, 1397 Iterations

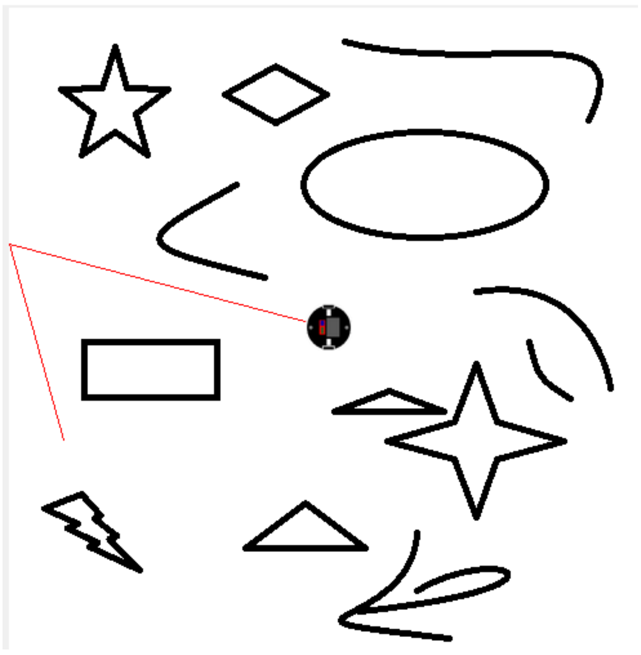

Figure 12. Map 2, 731 iterations

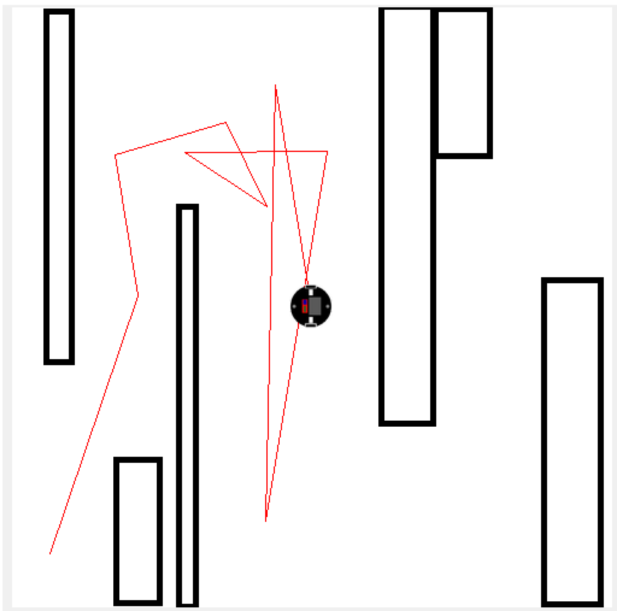

Figure13 . Map 3, 31766 Iterations

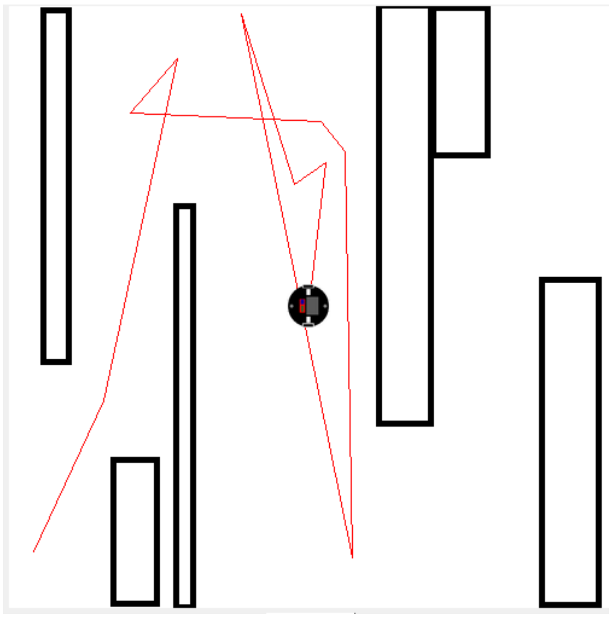

Figure 14. Map 3, 146564 Iterations

Figures 10 thru 14 are the result of implementing the fifth task on the three maps.

Further refinement is having a node in the end game region also prompt a child to grandparent search that will allow the path to be untangled as seen in the previous task(Figure 15). As well search that will find the shortest path on the roads from goal to start. When a viable path is found the path is returned and can be mapped by reconstructing the path. Results of these activities in the algorithm can be seen In Figures 15 thru 21.

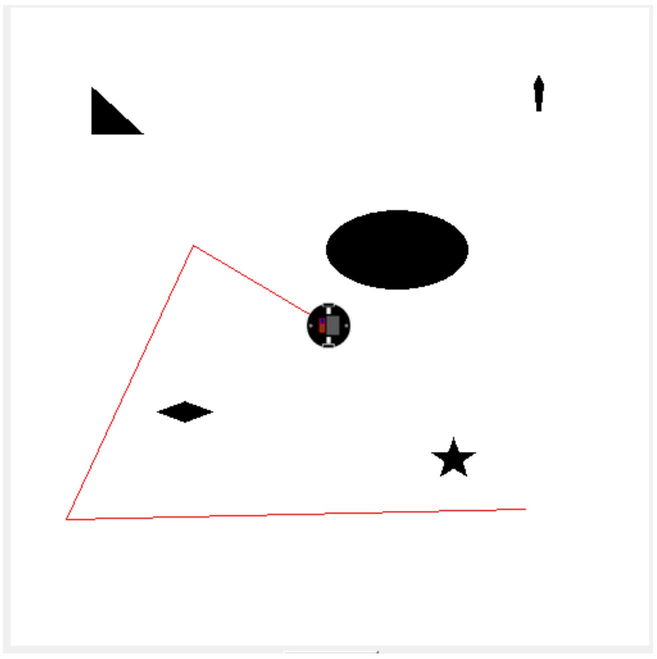

Figure 15. Map 1, 2679 Iterations, shows no crossing within the path from start to goal. Depicts child to grand parent connection.

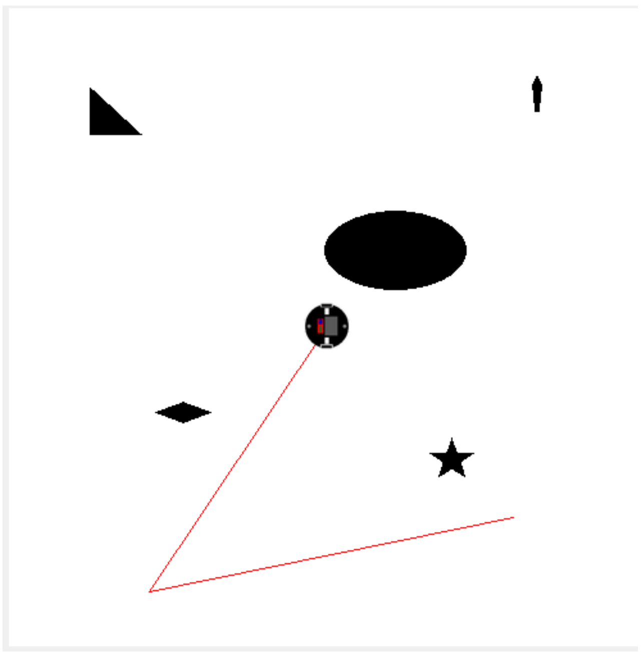
Figure 16. Map 1, 837 Iterations

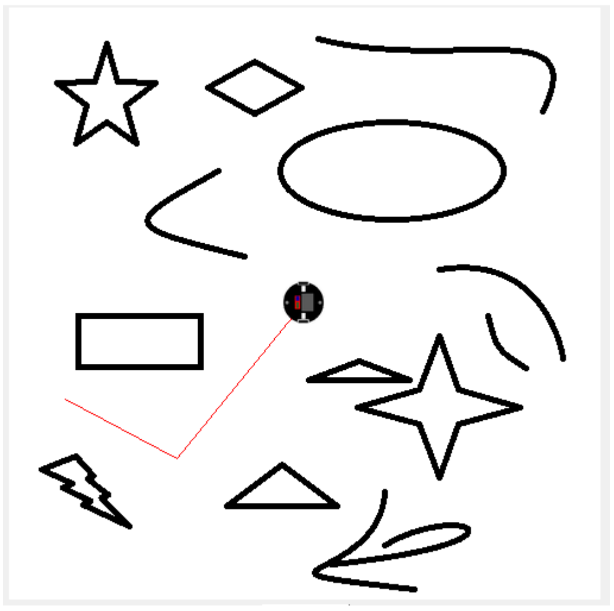
Figure 18. Map 2, 1050 iterations

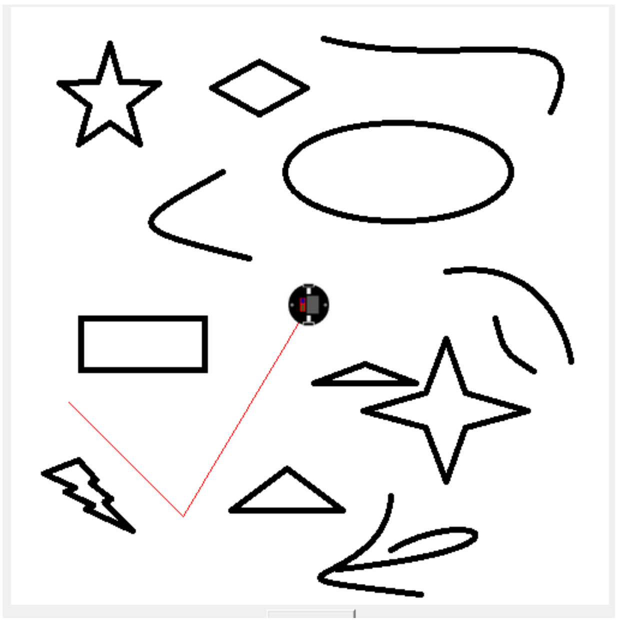
Figure 17. Map 2, 2773 Iterations

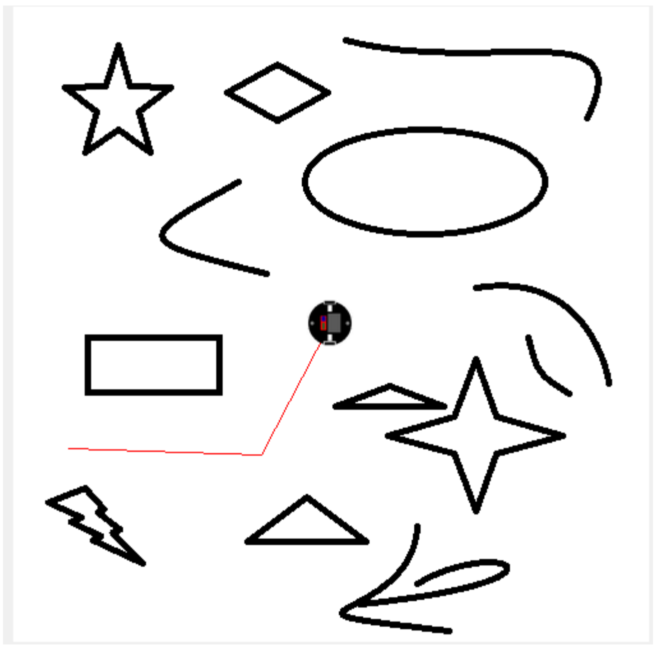
Figure 19. Map 2, 43 Iterations

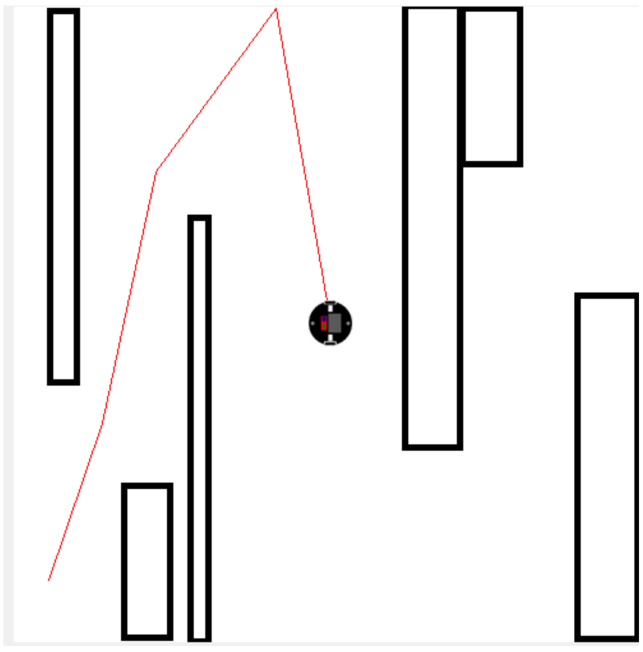
Figure 20. Map 3 , Iterations 162560

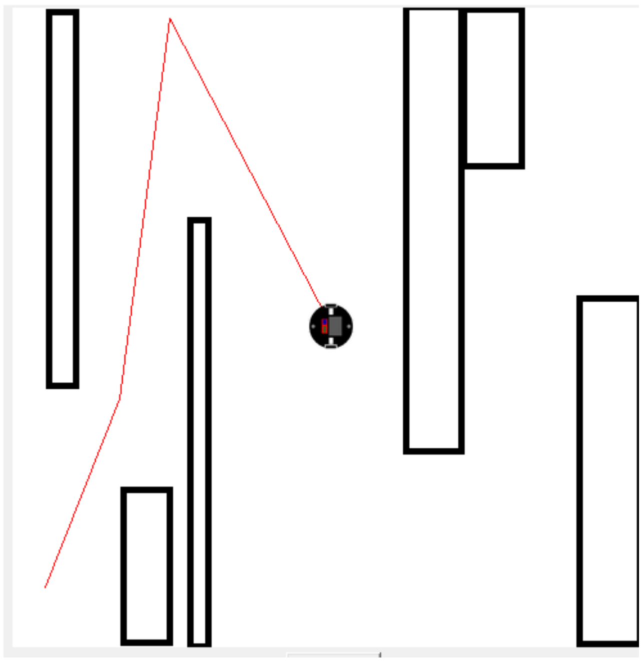
Figure 21. Map 3, 99589 Iterations.

III. Analysis and Conclusion

Thru the results in figures 4 thru 9 we note that random generation of paths has an easier time on maps with more open space. When a map has obstacles more cluttered together it is more difficult to generate a random path to maneuver thru the obstacle. Figure 22 depict a map that more clearly shows the low probability in finding a path in an area that would be considered hard to reach from the start point.

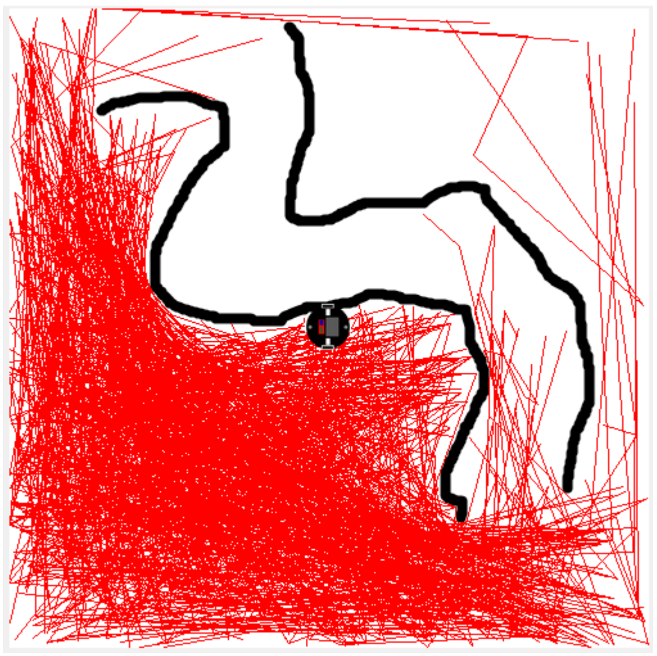
Figure 22. Experiment map, 3000 Iterations showing less probability of finding paths in hard to reach areas

We note that this also affects the number of iterations is takes to find a viable path. Map 1 had significantly lower number of iterations generated than map 3. It is also possible that this low probability to find paths thru map areas with a dense number of obstacles results in no path being found within an acceptable number of iterations allowed. As mentioned previously, we limit the number iterations allowed so that the algorithm does not go on forever. This issue was witness with map number 2 when trying to search for a path on the lower right region of the map. It is possible to have a path in that region as witnessed in the A* search algorithm module 3. However, the probability of finding a path in the lower right region of map two is so low that the search can go on for an unhealth amount of time; the iterations needed to be limited and a path was never found for a point in that region.

In general, the more iteration we have, the more viable paths we find, the better chance there is at finding a decent path. Finding a decent path would be finding a more direct path. However, to find a path, any path, the quickest it was found that the found path often intersected itself as shown in figure 10, 11, 13 and 14. The solution to connect child to grandparent adds to the time it takes to find a solution.

It was also found that the number of iterations were independent of the path found. For example, looking at figure 17 and 18 for Map 2, the solution gave a similar paths, however, the number of iterations were different. As in we can confirm that the path was "randomly" found. The number of iterations also correlates to the directness of the path from start to goal. In figure 19, also for map 2, we see that it has the most direct path to the goal point. Comparing this figure to 17 and 18 we can conclude that more direct path randomly found are related to a lower number of iterations searched. This is seen again when comparing figure 10 to figure 11 and figure 15 to figure 16. This is because as analyzed previously these figures provide more open space from the start node to the goal node. So, the probability of finding a path is higher and faster; more options can get generated within a given iteration frame.

ACKNOWLEDGMENT *(Heading 5)*

Thanks professor for helping understand and implement probabilistic path finding and generation of a probabilistic road map.

REFERENCES

[1] R. Siegwart, I. Nourbakhsh, D. Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd ed., The MIT Press : Cambridge, 2011. pp 57 – 98.

[2] Y. Chang, Robotics Motion Planning , Lecture Note Set #17-18. CPP: ME5751, 2022.

[3] https://en.wikipedia.org/wiki/Probabilistic_roadmap

[4] https://stackoverflow.com/questions/40205223/priority-queue-with-tuples-and-dicts

[5] https://www.geeksforgeeks.org/python-dictionary/

[6] https://www.w3schools.com/python/python_dictionaries.asp

[7] https://www.geeksforgeeks.org/priority-queue-set-1-introduction/

[8] http://groups.csail.mit.edu/graphics/classes/6.838/S98/meetings/m13/kd.html

[9] https://www.javatpoint.com/ai-uninformed-search-algorithms