ME5751 Lab 4 – Probabilistic Roadmap

Ryan Chao
Cal Poly Pomona
Pomona, CA
rchao@cpp.edu

Cameron R. Weigel
Cal Poly Pomona
Pomona, CA
crweigel@cpp.edu

***Abstract*— In this lab, probabilistic road mapping is performed utilizing a KD Tree graph structure, Bresenham line generation, and Dijkstra's search algorithm to construct a roadmap and generate a semi-optimized path from a series of randomly generated points. Issues faced and overcame were infinite looping, dense roadmaps which led to increased compute time, and obstacle collision even. Furthermore, with a probabilistic approach, clustering is a likely case to occur which may consume computation time. Compute times ranged from half a second up to six seconds depending on the complexity of the map. In cases where the program failed to generate a path with the set amount of initial randomly generated points, the algorithm will rerun to achieve a successful path within an iteration limit of 20 attempts. After more optimization, this algorithm would be able to avoid unnecessary motions such as having to ricochet back into the correct direction.**

Keywords*— *Probabilistic roadmap; PRM; single query; robot motion planning; Djikstra's SPT; KD Tree; Bresenham;

I. INTRODUCTION

In the last module, an A* search algorithm was utilized to determine the most optimal path from a robot's start position to a goal position whilst also avoiding obstacles. In this module, probabilistic roadmap or PRM planning is implemented to perform the same function. The differences between the two planning algorithms is that PRM generates random samples within a configuration space to help establish an optimal path. PRM planners consist of two phases: a construction phase where a roadmap is built from collision-free motions that can be made between the random samples and a query phase where edges or connections are made to obtain a path from the start and goal position. Without an optimization or query phase, the configuration space would contain connections that are invalid for motion planning. Under the conditions of randomly generating a large number of points, the planner would successfully find a path on the first iteration. However, this method would increase computation time to unacceptable levels, effectively making it useless in mobile robot motion planning. In order to balance efficiency with valid path generation, a maximum sample of 100 points is generated for a map of size 500*500. In most cases, a path is found on the first iteration, in some cases, multiple iterations are required, but in all tests, a path is found.

Figure 1. Path planning from center to random goal points.

II. BACKGROUND

Path planning is one of the vital operations required to establish effective robot navigation. It

determines whether the robot is acting in the most efficient manner. Proper application of the PRM algorithm allows for a more time-efficient path planning than that of the A* algorithm. Since the A* algorithm constructs a continuous path via a cell-to-cell approach, the majority of compute time is spent on determining whether neighboring cells are valid. To avoid or reduce this, PRM planners construct edges, or motions, between free cells that have a Manhattan distance greater than 2 i.e. not neighboring one another. This is more practical as it incorporates a discretized approach to motion planning similar to how humans navigate around obstacles to reach their destination in the virtual or real world. Instead of having to check our surroundings within a foot away before every step, we can plan out an optimal path from what is seen hundreds of feet away.

An example of a PRM planner is a single query or SQ PRM algorithm. As the name states, only a single path is generated from the start point to the goal point via randomized point generation. The edges are only established if the connection between the current and next point is not obstructed by an obstacle. Multiple query PRM creates multiple paths that resemble that of tree branches where many edges are generated with several points within a space around the current point.

Although PRM planners can effectively skip steps, they are expected to traverse in motions that can be deemed unnecessary thereby proving to not always be optimal. This is due to the arbitrary nature of PRMs, but there are methods that can be implemented to reduce such actions. This will be explored more in depth in the discussion section.

III. METHODS

The methods employed in our implementation utilize a KD-Tree structure to organize our data based on a node's nearest neighbors, and a modified Dijistra's algorithm to traverse the tree structure. In order to organize the data in a meaningful way, we make use of a graph structure implementation. This graph contains nodes and edges, nodes can be thought of as the randomly generated valid points on our map, and the edges are the connections between the nodes. The KD-Tree generates our edge connections and applies a "length" value to each edge. We ensure a point's position and connections are valid in two steps. In the first step, upon generation of the points, we check each point's location on the map and ensure it is not within an obstacle, if the point is within an obstacle, we drop the point. Second, upon generation of the nearest neighbors, we only generate edges between nodes if there is a connection between them that does not go through an obstacle. In order to accomplish this checking, we utilize the Bresenham algorithm for line generation, then we check the line's pixels for obstacle pixels. Next, utilize the edges and "length" within our Dijikstra's method to find a valid path. Due to Dijikstra's limitations, the path is not guaranteed to be optimal, but it is guaranteed to find a solution if one exists. In order to optimize run time and results, we use a random point generation algorithm that generates 100 points. In most cases, 100 points is enough to find a solution, but in certain cases, another iteration of 100 randomly picked points may be necessary. To accomplish this, we loop our point generation and path planning algorithms until a solution is found or until a max number of iterations is hit.

IV. RESULTS AND DISCUSSION

The results and discussion of the experiment will consist of testing three custom-built maps to demonstrate the complexities and considerations required of probabilistic roadmap planning. Metrics to be analyzed will be motion iterations, compute times, and path effectiveness. A hundred randomly positioned points are generated which will then be used to plan a path to the goal point.

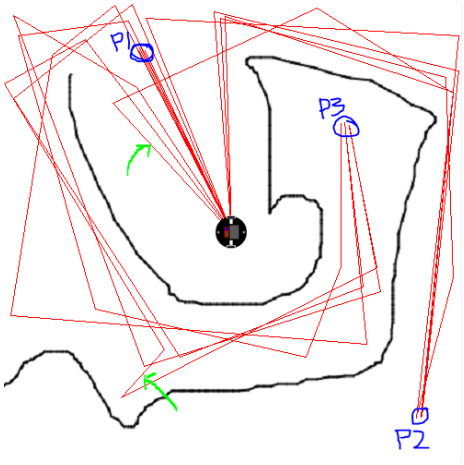

Figure 2. Test Map 1 tested with 3 different points.

In Figure 2, three points were tested five times to observe how the algorithm performs on a "medium" level difficulty map. The points in blue represent configurations sampled five times each and green arrows are pointing at unnecessary motions. Regarding point 1, it was observed to be a relatively easy goal point to reach. The average compute time was 0.485±0.070 seconds, random points used was 61±5, and average motion of 1. Time is directly proportional to the amount of points analyzed since it required 0.573 and 0.547 seconds with 66 points each whereas it required 0.434 seconds for 56 points. Since this point was well within reach of the start point, the amount of iterations and motions kept within an expected minimum of 1.

For point 2, it was observed to be more difficult to reach due to the algorithm needed to navigate around an obstacle. The average compute time was 1.108±0.317 seconds, random points used was 69±4, and an average motion of 3. This trial demonstrated time is not always directly proportional to points, since two tests required two iterations before completion with one taking 1.172 seconds with 72 random points while another required 1.502 seconds with 72 random points. Point configuration also contributes to the efficiency of the PRM algorithm incorporated. Tests that underwent 2 iterations as opposed to 1 iteration required 79.2% additional time to complete. Path-wise, the majority of the tests seem to be in agreement as shown where three of the five tests are very similar to one another.

The statistics for point 3 consist of an average compute time of 0.857±0.327 seconds, random points used was 66±4, and average motion was 5. In this trial, it can be observed how PRM planners can be inefficient as there is clearly one motion that is unnecessary in reaching the goal. However, the majority of the tests are in alignment with one another to reach the goal point. Similar to test point 2, the tests that underwent 2 iterations required 81.6% more time to complete than that of single iterations. A directly proportional relationship between iterations and compute time can also be established.

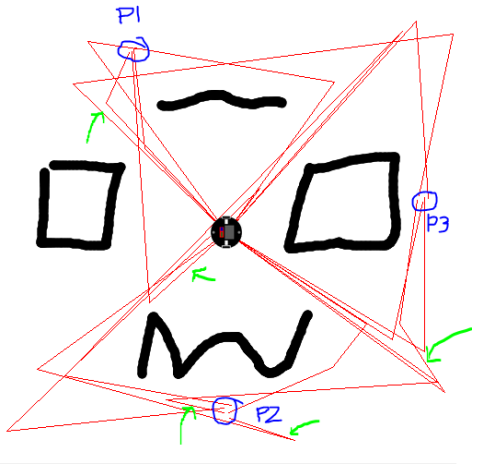

Figure 3. Test Map 2 tested with 3 different points.

In Figure 3, the algorithm is tested on a map with a difficulty that is more on the "easy" level difficulty. The majority of the paths can be navigated to within 3 motions.

For point 1, which lies in the upper left corner, all tests were completed in 1 iteration. The average compute time was 0.704±0.162 seconds, random points used was 68±7, and an average motion of 2. The shortest approach would be moving in the upward left direction, however there

are deviations where the motion moves in the bottom left direction or upper right direction which stray away from the goal point. Compute time is found to be roughly 0.7 seconds for 70 random points, 0.57 seconds for 62 random points, and 0.965 seconds for 78 random points further proving a directly proportional relationship between the two.

The tests for point 2 yielded an average compute time of 0.764±0.163 seconds, random points used was 72±6, and average motion of 3. Since the goal point is centered below an obstacle, much sharper path turns were required in comparison to the paths of point 1. An issue with PRM planners is displayed through motions that overshoot the goal point which require an additional step to reach the goal. An outlier exists here in that one test completed the path in 4 motions but only required 0.592 seconds whereas the other tests were done in 3 motions and more than 0.634 seconds each.

The tests for point 3 share similar results as that of point 2. The average compute time was 0.758±0.193 seconds, random points of 73±8, and average motion of 2. One of the tests demonstrated an outlier with a compute of 0.477 seconds in 2 motions and 60 random points. This backs up the assumption of a direct relationship between random points and time.

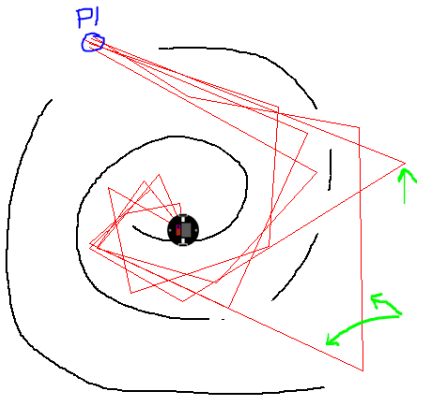

Figure 3. Test Map 3 tested with the first point.

In Figure 3, the test map holds a difficulty of "hard" for the algorithm to solve due to the presence of curved and spiral obstacles. Compared to the other two test maps, test map 3 demonstrates the importance of quality in the sample points. The average compute time was 2.203±1.182 seconds, random points of 71±7, and average motion of 5. The amount of random points is not a reliable metric to measure efficiency in maps similar to this one. However, it is observed that the quality of the randomized points in terms of where they're placed to generate a path from start point to goal point plays a crucial role. One test was completed in 0.676 seconds with 72 random points and 1 iteration whereas another was completed in 3.832 seconds with 76 random points and 5 iterations. The amount of random points are similar indicating that position matters in more complex maps.

Figure 4. Test Map 3 tested with the second point.

In Figure 4, the test is performed to observe how the algorithm performs in exiting the spiral and reaching the corner to point 2. The results are more sporadic than that of point 1. The average compute time was 2.860±2.352 seconds, random points used was 72±10, and average motion was 5. The green arrows point out unnecessary motion demonstrating one of the negative effects of using a PRM planner.

If the robot were to travel these paths, it would be ineffective usage of time. This is where certain conditions could be incorporated that prevent these points from being considered. The times recorded had 2 very good runs with 1 iteration and 2 very bad runs with 8 iterations at roughly 0.500 seconds and 5.250 seconds respectively. This data further proves that the quality of the point positioning is important in allowing for faster computation times.

From this experiment, it can be concluded that time and random points are directly proportional when it comes to simpler paths. However, when it comes to more complex maps, the quality of the points matter more than the quantity of random points generated. PRM planning has been proven not to be an exact science due to its arbitrary nature. Though it may yield paths that are similar to one another, it is ultimately random and dependent on the points supplied. It may even generate paths relatively quickly, but it may not always generate a path that is optimal to traverse.

V. CONCLUSION

The combination of Bresenham's line algorithm, KD tree search, and Dijkstra's shortest path tree has yielded a successful PRM planner. One of the main concerns before working on the algorithm was figuring out how it would deal with obstacle navigation which was quickly erased when we ran our path planner. Although it did not generate an optimal path every time, it was able to do so in a timely manner. For cases where compute time exceeded 5 seconds, it may have been better to increase the number of randomly generated points to avoid any need for reiteration.

Compute time was very dependent on the complexity of the map especially with regards to obstacles that had a more spiral pattern to it.
With our test maps, it did not exceed 6 seconds and all paths did not require more than 10 motions to be completed. Positioning of the randomly generated point was determined to play a crucial role in compute time since a poor set of points would require reiteration even though the amount of points required to complete a path would be similar between tests. This was further proven by cases where a smaller amount of points was able to solve quicker than one with more.

VI. REFERENCES

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

[2] Y Chang, California Polytechnic University, ME5751 Lecture Slides, Fall 2022

[3] H Choset, CMU CS Lecture Series, Robotic Motion Planning, Robotic Motion Planning: A* and D* Search

[4] A Patel, Red Blob Games, Implementation of A*

[5] N Swift, Easy A* Pathfinding, Medium webpage

[6] Wikipedia, A* search algorithm

[7] Geeks for Geeks, A* search algorithm