Analysis of a Robot Model with Probabilistic Roadmap Path Planning Control

Evelyn Ware[1], Emilia Psacharopoulos[1]
April 25, 2022

[1]Department of Mathematics, University of Michigan, 530 Church St, Ann Arbor MI 48109

Abstract. We study an autonomous Path Planning robot model in Simulink. We investigate the path planning algorithm in detail and implement our version of the plant. Lastly, we suggest further applications for this model.

1. Introduction. Autonomous robotics has an enormous potential as a developing field. Among many positive contributions, automating jobs that are too dangerous for humans will undoubtedly create safer conditions for the workforce. In order to design an autonomous system, one needs to heavily consider the path planning, system dynamics, and control theory of the particular problem. Our research paper addresses a particular problem regarding the efficiency of an autonomous bicycle robotic model in Simulink. We explore the existing system's control system and vehicle dynamics, and extend our study to develop our own model of the vehicle dynamics in the system and determine the robustness of the path planning algorithm.

2. Background. The original model is primarily composed of three main components: Planning, Control, and the Plant Model [1]. This is the general overview of the architecture in Simulink:

1. The Planer module takes in the Start Location, Goal Location, and Map as parameters. These parameters are initialized in MATLAB by the user. The Planner then uses a Probabilistic Roadmap as its path planning algorithm, and outputs an array of node coordinate directions for the robot to follow.
2. The Control module inputs those coordinates and calculates the necessary linear and angular velocities of the robot to follow that path given the current location of the robot on that path. Additionally, the Controller checks if the goal has been reached within a threshold to then stop the simulation.
3. The Plant module feeds the linear and angular velocity outputs from the Control module through a system of differential equations. These differential equations model a simple bicycle without dynamics in the tires. The kinematic equations give the robot's current (x,y) coordinates, as well as the heading angle.
4. The system outputs those current coordinates to the map, then feeds the current coordinates back into the Controller. Steps 2-3 repeat until the threshold has been reached, or the user manually stops the simulation after enough time has elapsed to determine that the robot will not reach the end goal.

3. Robot Control. The Controller module considers the current position and orientation of the robot from the Plant module then computes the necessary linear and angular velocities, or linear

velocity and heading angle, to follow the given path from the Planner module. The robot moves at a single value of linear velocity at all times during a given simulation. The linear velocity can either be set up to a maximum of 1.0 m/s or as a default of 0.1 m/s. Additionally, the maximum angular velocity is set at 1 rad/s. For simplicity, we never changed the linear velocity scalar value nor the maximum angular velocity scalar value from its default in any of our experimental simulations.

The “LookaheadDistance” parameter is one of the most important parameters in the entire system. It chooses which point the robot should follow given the list of coordinates from the Planner module. For instance, a higher “LookaheadDistance” would result in the robot following a path with less intermediate turns to different edges on the graph, given there are no intermediate obstacles in the way. Also, a smaller “LookaheadDistance” would induce oscillations and unstable behavior. It is clear that varying this parameter drastically changes the path that the robot takes. Since we are only focusing on the path planning algorithm, we again left this parameter at its default value of 0.5 m across all simulations.

Lastly, The Controller module also contains the base case implementation to break out of the recursive algorithm. The “Check Distance to Goal” controller block implements the following logic:

$$if(\sqrt{(current\ coordinates\ -\ goal\ coordinates)^2}\ <\ threshold),\ then\ stop\ simulation$$

The Simulation sets the default threshold value to 0.25. For consistent results, we decided to leave the threshold as this default value in all experimental simulations [2].

4. Original Plant Model. This plant model implemented the kinematic equations governing a simple bicycle model. This model is “simple” because there are no dynamics from the tires, and therefore can be thought of as being a frictionless system with no energy loss on rails. The kinematic equations are a system of three differential equations to get the current (x,y) coordinates and the heading angle of the robot [3].

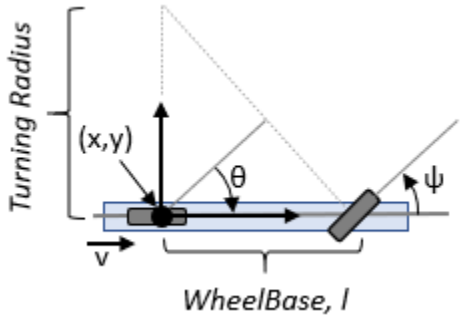


Figure 1: A diagram showing the relevant variables to the robot vehicle system [3].

The model specifications do not include any detail on these governing equations, so we decided to implement our own system of differential equations and compare the behavior of our plant model to the behavior of the robot's original plant model. We are assuming that this model is a continuous time system, and therefore used continuous integration to manipulate the derivative signals. The steering angle of the bicycle is limited to +/- pi/3 to mimic the steering motion of a real bicycle. Figures 2 and 3 describe our system.

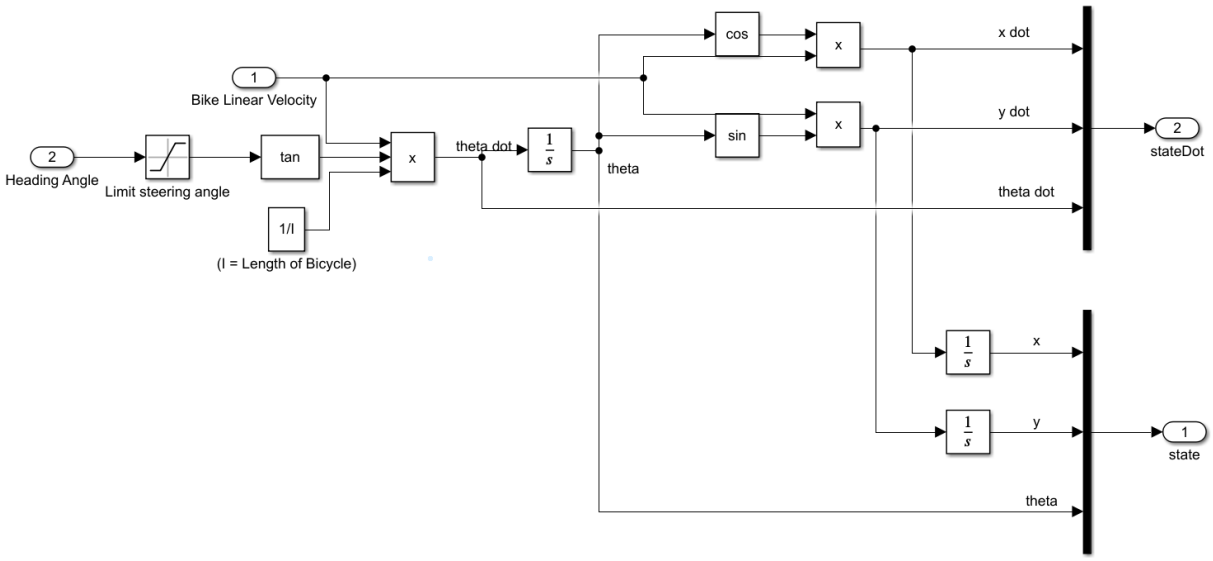


Figure 2: Our plant model in Simulink. Note: we had the Controller block output the heading angle instead of the angular velocity in this implementation.

$$\dot{x} = vcos(\Theta)$$
$$\dot{y} = vsin(\Theta)$$
$$\dot{\Theta} = \omega = \frac{v}{R} = \frac{vtan(\psi)}{I}$$

Figure 3: Our system of differential equations to model the vehicle dynamics.

We tested our plant model versus the original plant model using the same starting coordinates, ending coordinates, map, and path planning settings (using the MATLAB command *rng* to produce the same results).

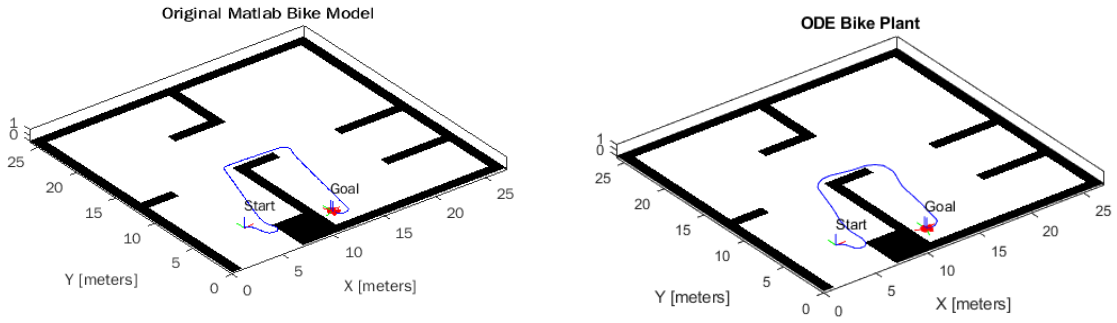


Figure 4: Comparison of bicycle path through maze for MATLAB model and our ODE model.

When comparing our model with the original model, we are able to confirm that our model has similar performance when given the same start and goal locations. Both models have a slight overshoot at the beginning of the path before turning and moving around the obstacle towards the goal. The path of the robot appears to be smoother in our implementation which seems more realistic for a model of a bicycle. This is likely because of the continuous time integrator blocks in our implementation.

5. Path Planning Algorithm. The path planning algorithm follows a Probabilistic Roadmap (PRM) implementation [4]. Ideally, PRM works much better in higher dimensional environments where producing dense graphs and storing those edges in an adjacency matrix is much more computationally expensive [5]. Our robotic system can only move on its 'wheels' in the x-y direction, and therefore is a 2-degree-of-freedom system, so the savings will not be as noticeable in space and time as it would be in a higher dimensional system. PRM trades slight amounts of efficiency in regards to the path taken to potentially lose enormous quantities in runtime complexity and memory. We will walk through the PRM implementation to explain this significance then test the robustness of the implementation.

5.1 Probabilistic Roadmap Implementation.

1. The user loads their desired map into the function `binaryOccupancyMap`.
2. Then, the user uses the function `mobileRobotPRM` on the generated `binaryOccupancyMap` with a specified number of nodes, `numNodes`, as an input parameter. This function distributes the nodes in random valid positions across the map with a uniform random sampling, then connects an edge between any node within a Euclidean distance of length `ConnectionDistance` to another node to generate the PRM undirected graph for the given parameters. `ConnectionDistance` is a parameter designated by the user. We examine the impacts of this parameter in our experimental trials below, but namely in combination with `numNodes`, `ConnectionDistance` will either implement a sparse or dense graph. Clearly, the savings in runtime and memory can be directly linked to the efficient combination of these two parameters. For example, if the user sets the `ConnectionDistance` to be larger than the span of the map itself, they are essentially creating a fully-connected graph which will have a noticeably

negative impact on the runtime and memory given there are many `numNodes`, but the robot may take a more efficient path. However, if the user sets `ConnectionDistance` to be too small, there may be a case without any edges between vertices, and then the path planning algorithm would not produce any valid paths to the goal location [6].

Algorithm 6 Roadmap Construction Algorithm

Input:

n : number of nodes to put in the roadmap

k : number of closest neighbors to examine for each configuration

Output:

A roadmap $G = (V, E)$

```
1:  V ← ∅
2:  E ← ∅
3:  while |V| < n do
4:    repeat
5:      q ← a random configuration in 𝒬
6:    until q is collision-free
7:    V ← V ∪ {q}
8:  end while
9:  for all q ∈ V do
10:   N_q ← the k closest neighbors of q chosen from V according to dist
11:   for all q′ ∈ N_q do
12:     if (q, q′) ∉ E and Δ(q, q′) ≠ NIL then
13:       E ← E ∪ {(q, q′)}
14:     end if
15:   end for
16: end for
```

Figure 5: Pseudocode implementation of the mobileRobotPRM algorithm. Δ is a boolean function that returns whether the proposed edge crosses a border, and is therefore not allowed. The algorithm above uses k nearest neighbors to choose its edge connections, but our implementation instead connects all neighbors within Euclidean ConnectionDistance. This implementation ensures that there are no repeated edges in the graph, and all edges do not cross a border [7].

3. Lastly, the user calls the algorithm `findpath` on the generated Probabilistic Roadmap. `findpath` first adds the combinations of all possible paths starting at the initial coordinate and ending at the goal coordinate (lines 1 through 18 in Figure 6). Starting with the initial coordinate of the robot, the algorithm queries the nodes in the graph to find the nearest neighbor to the initial location of the robot, and checks that it can create an unobstructed path until a path from the initial coordinate to a node in the graph is found. The same process is implemented to connect the goal location to the graph. This connects the PRM graph created in the map to the specific start and end goals of the robot [8]. Then, the algorithm implements Dijkstra's algorithm (line 20 in Figure 6) which is the fastest algorithm to find the shortest path between a starting node to any other node on a weighted graph. In our case, the weights of the connections between nodes are the Euclidean distances between those nodes. Essentially, Dijkstra's Algorithm minimizes the Euclidean distance that the robot has to traverse between nodes to reach the goal location [9].

Algorithm 7 Solve Query Algorithm

Input:

q_{init}: the initial configuration
q_{goal}: the goal configuration
k: the number of closest neighbors to examine for each configuration
$G = (V, E)$: the roadmap computed by algorithm 6

Output:

A path from q_{init} to q_{goal} or failure

1: $N_{q_{\text{init}}} \leftarrow$ the k closest neighbors of q_{init} from V according to *dist*
2: $N_{q_{\text{goal}}} \leftarrow$ the k closest neighbors of q_{goal} from V according to *dist*
3: $V \leftarrow \{q_{\text{init}}\} \cup \{q_{\text{goal}}\} \cup V$
4: set q' to be the closest neighbor of q_{init} in $N_{q_{\text{init}}}$
5: **repeat**
6: **if** $\Delta(q_{\text{init}}, q') \neq$ `NIL` **then**
7: $E \leftarrow (q_{\text{init}}, q') \cup E$
8: **else**
9: set q' to be the next closest neighbor of q_{init} in $N_{q_{\text{init}}}$
10: **end if**
11: **until** a connection was succesful or the set $N_{q_{\text{init}}}$ is empty
12: set q' to be the closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
13: **repeat**
14: **if** $\Delta(q_{\text{goal}}, q') \neq$ `NIL` **then**
15: $E \leftarrow (q_{\text{goal}}, q') \cup E$
16: **else**
17: set q' to be the next closest neighbor of q_{goal} in $N_{q_{\text{goal}}}$
18: **end if**
19: **until** a connection was succesful or the set $N_{q_{\text{goal}}}$ is empty
20: $P \leftarrow$ shortest path(q_{init}, q_{goal}, G)
21: **if** P is not empty **then**
22: **return** P
23: **else**
24: **return** `failure`
25: **end if**

Figure 6: Dijkstra's Algorithm in the `findpath` function is implemented on line 20 [7].

5.2 Probabilistic Roadmap Robustness Analysis.

To test the robustness of the PRM algorithm, we vary the number of nodes generated in the map and the maximum connection distance between the nodes. We compare results between two different target maps, one simple map and one complicated map.

For a simple map with a large number of nodes and long connection distance, the PRM algorithm has a high probability of finding a path between the start location and the goal location. As shown in Figure 7, when the number of nodes is 100 and the maximum distance between nodes is set to be longer than the size of the map, the connections formed between nodes are very dense across the entire map. The algorithm is able to find multiple paths to the desired location and chooses the shortest path. This is expected from the PRM algorithm, because it is probabilistically complete. As the number of nodes is increased, the probability of failing to find a path decays exponentially to zero [10].

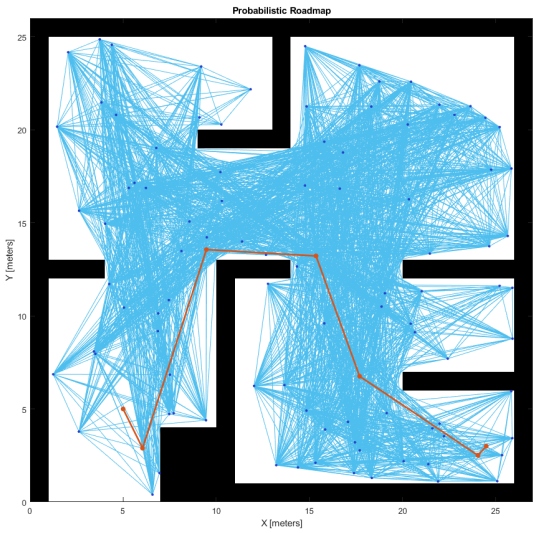
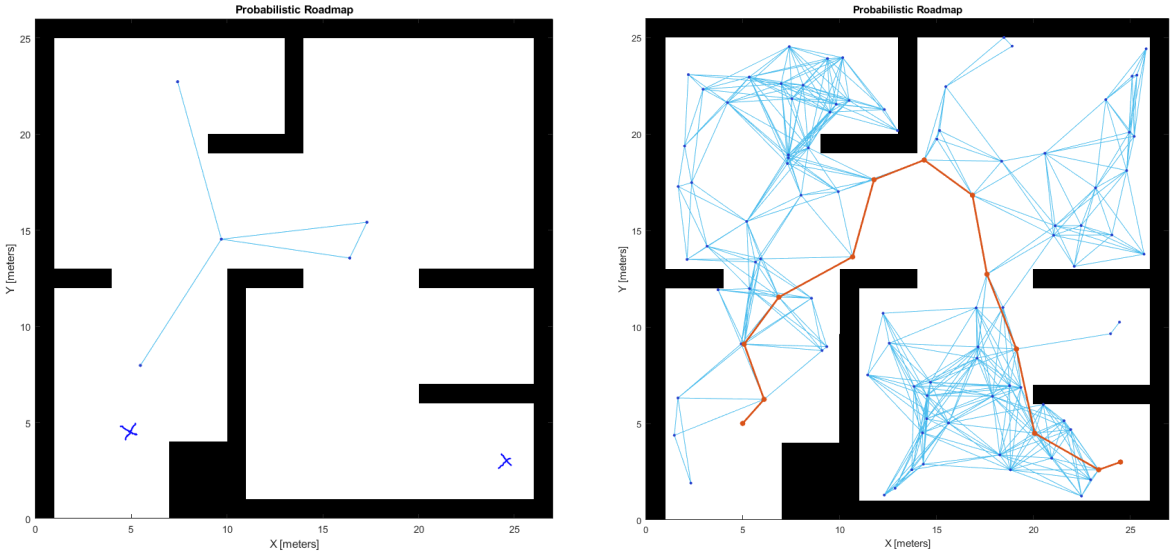


Figure 7: Probabilistic roadmap showing node connections and optimal path.

However, if we decrease the number of nodes and the allowed connection distance between the nodes, the algorithm is not always able to find a path, as shown in Figure 8. Even when a path is found, it is unlikely to be optimal because the algorithm is forced to find a path through nodes placed sparsely in the map as shown in figure 9.



Figures 8-9: Probabilistic roadmap with few nodes (no path found) (left) and less efficient path found (right).

The average, maximum and minimum path distance are shown in Table 1 for different node and distance settings in the simple map. For a high number of nodes and large maximum connection distance, the algorithm finds a path in 100% of trials and has the shortest average path length, maximum/minimum path length, and standard deviation of path length. As the number of nodes and connection distance is decreased, the algorithm is less likely to find a path

to the desired location. The average path length also increases because the path found by the algorithm may have to take a more roundabout route through the available nodes to reach the desired location. The standard deviation of path length is also higher, showing that there is also more variability in the path resulting from the algorithm when we use less nodes or a shorter connection distance.

Table 1: Simple map PRM testing with varied number of nodes and connection distance.

	Test 1	**Test 2**	**Test 3**	**Test 4**
Number of Nodes	100	100	5	5
Connection Distance	40	5	40	10
% of trials that found a path (out of 10000)	100	81.59	13.07	1.18
Average Path Length	33.39	36.83	49.38	39.94
Max Path Length	44.41	63.34	82.33	51.30
Min Path Length	28.70	29.06	32.05	31.19
SD Path Length	1.85	3.70	9.16	4.40

Increasing the complexity of the map (number of obstacles) and decreasing the amount of obstacle free area decreases the probability of the PRM algorithm successfully finding a path. If the goal location is in a room that only has a small entrance clear of obstacles, it is less likely that the random placement of nodes will fall in a way that allows the PRM algorithm to find a path into the room. However, if the number of nodes is large enough, the PRM algorithm is still very effective at finding a path. To demonstrate this, we design our own map as shown in Figure 10. This map has multiple obstacles covering 50.24% of the map area (vs. 20.66% coverage in the simple map).

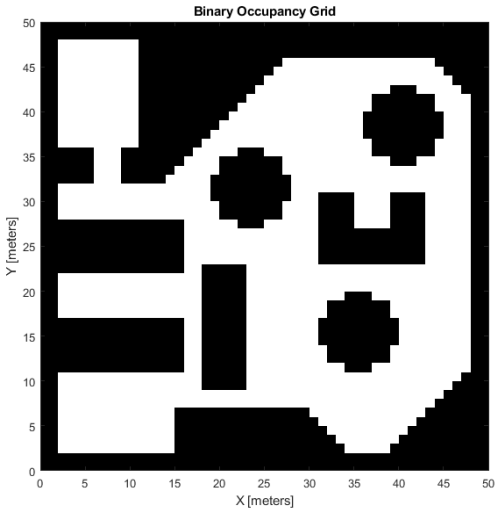
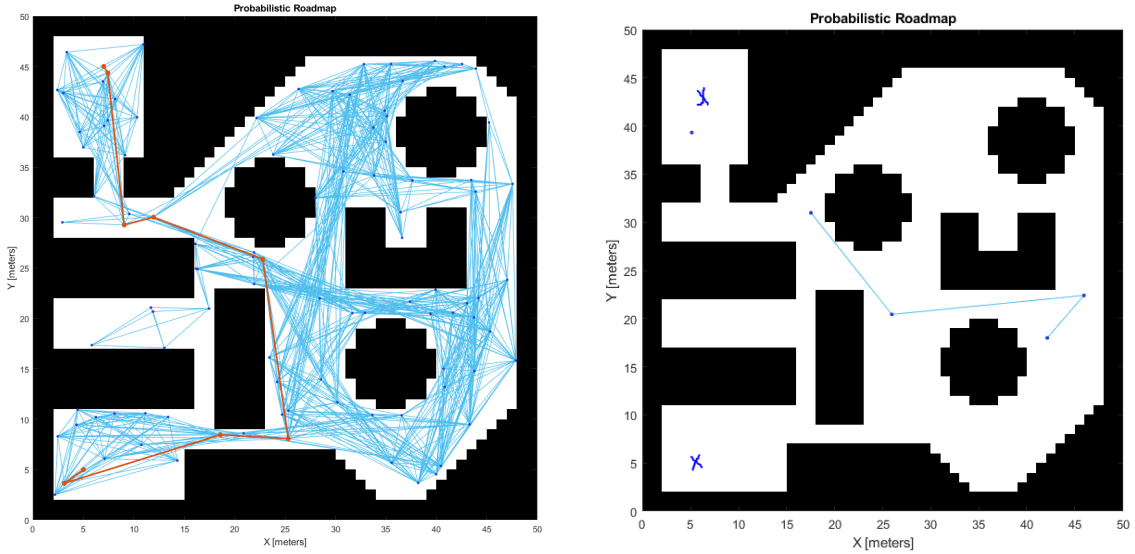


Figure 10: Our complex map design for PRM algorithm testing with 50.24% coverage.

In this case, when many nodes (100 nodes) are placed on the map the algorithm is able to find a path to the desired location in 86% of the trials. When the number of nodes is small (5 nodes), the PRM algorithm is almost never able to find a path. Compared to the simple map, the PRM algorithm is much less likely to find a path in the complex map with a small number of nodes because of the multiple obstacles in the map. Interestingly, the standard deviations between different tests of the complex map are grouped more closely together than in the simple map tests. The difference is especially large between Test 1 and Test 3. Intuitively this makes sense because in the simple map there are many possible paths to reach the desired location, so even with a small number of nodes, when the connection distance is large enough, a path can be found (even though it might be long and sub-optimal). In the complex map the obstacles make it harder for different paths to be formed since there is less unobstructed space for connections.



Figures 11-12: Complex map with many nodes and shortest path (left) and few nodes with no path found (right)

Table 2: Complex map PRM testing with varied number of nodes and connection distance.

	Test 1	**Test 2**	**Test 3**	**Test 4**
Number of Nodes	100	100	5	5
Connection Distance	40	10	40	10
% of trials that found a path (out of 10000)	86.22	43.49	0.02	0
Average Path Length	66.67	69.24	64.76	N/A
Max Path Length	110.43	138.26	74.10	N/A
Min Path Length	52.58	52.43	55.42	N/A
SD Path Length	8.37	10.77	13.21	N/A

The final map we examine is a version of the complex map with 73.4% of its area occupied by obstacles as shown in Figure 13.

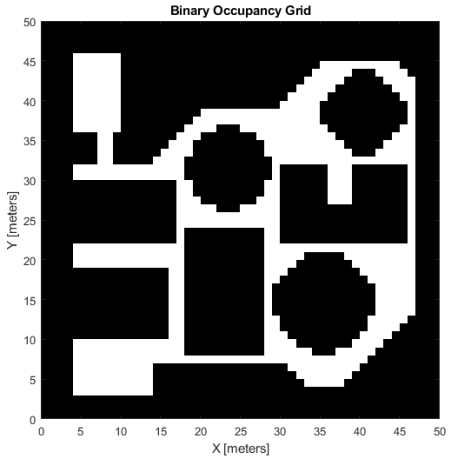


Figure 13: Our complex map with 73.4% area occupied.

In this map, even with a large number of nodes and long maximum connection distance, a path is only successfully found in about 50% of trials. When the number of nodes is small (5), no path can be found in 10,000 trials. This suggests that when there are a large number of obstacles in the map, it is more effective to have a large number of nodes than to allow for long connections between nodes.

Table 3: Highest complexity map PRM testing with varied number of nodes and connection distance.

	Test 1	**Test 2**	**Test 3**	**Test 4**
Number of Nodes	100	100	5	5
Connection Distance	40	10	40	10
% of trials that found a path (out of 10000)	52.16	26.15	0	0
Average Path Length	67.50	65.35	N/A	N/A
Max Path Length	143.94	152.37	N/A	N/A
Min Path Length	50.65	50.74	N/A	N/A
SD Path Length	18.34	17.04	N/A	N/A

Next we examine the minimum number of nodes needed to successfully find a path in each of our 3 maps. The initial number of nodes in the PRM algorithm is set to 1 and increased by 1 until a path is found. This process is repeated over 10,000 trials to find the average minimum number of nodes required for each map, as shown in Table 4. The simplest map requires the fewest number of nodes, more than 3 times less than the medium complexity map and 6 times less than the highest complexity map.

Table 4: Minimum number of nodes required to find a path for different map complexities.

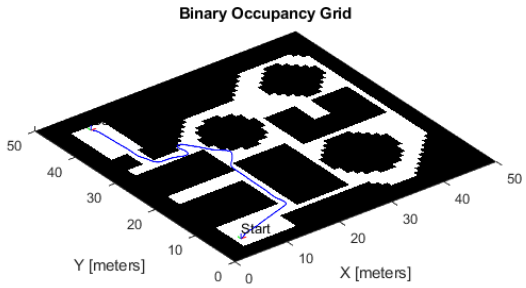
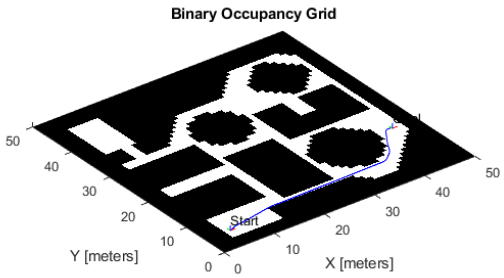
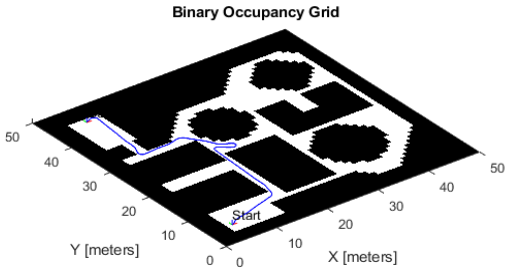
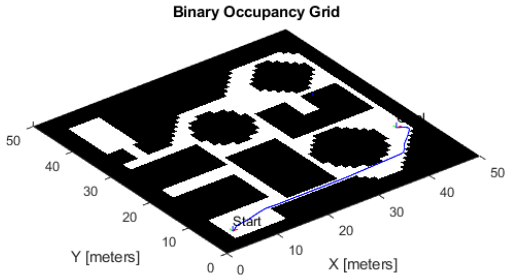
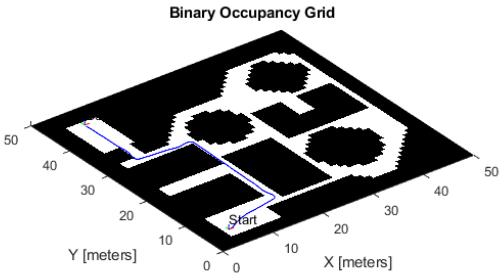
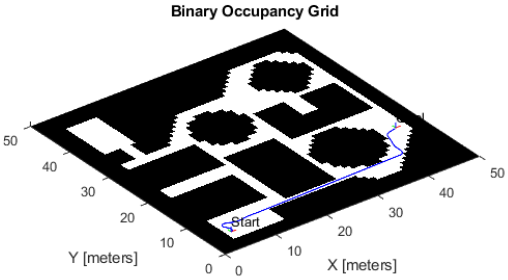
	20% Obstacles	50% Obstacles	73% Obstacles
Minimum # of Nodes before path is formed	7.07	25.71	47.26

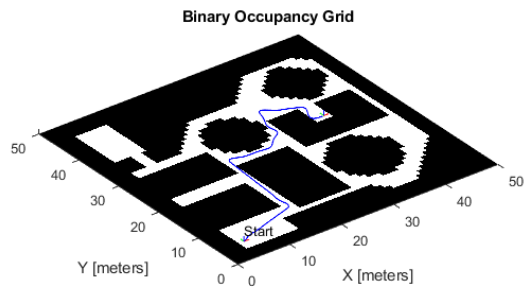
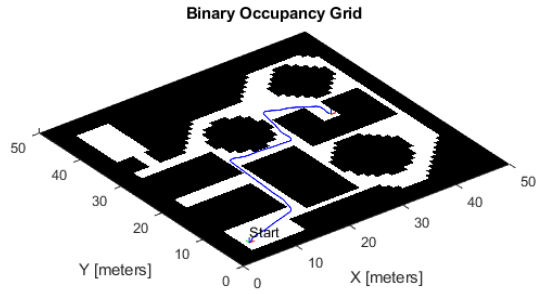
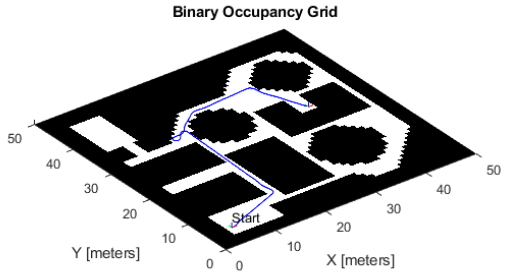
6. Results.

We use the PRM algorithm to find a path for a simple bicycle robot to move from a start location to a desired goal location in the most complex map. The model is implemented in Simulink. First a path is found using the PRM algorithm. Next a simple Pure Pursuit controller generates the steering angle and velocity commands for controlling the bicycle based on the map found by the PRM algorithm. Finally the ODE model of the bicycle robot simulates the motion of a bicycle through the map. We vary the number of nodes used in the PRM algorithm and observe the effect on the bicycle model.

We test the model with 300, 100, and 10 nodes and vary the goal location. The PRM algorithm is run until a valid path is found. Results are displayed in Table 5.

Table 5: Bicycle robot path comparisons for varied number of connection nodes.

N = 300	 Distance = 62.57	 Distance = 44.00
N = 100	 Distance = 66.95	 Distance = 46.06
N = 10	 Distance = 58.41	 Distance = 47.74

N = 300	 Distance = 61.57
N = 100	 Distance = 60.82
N = 10	 Distance = 74.0

The PRMs with higher number of nodes have higher computational complexity to generate the node connection map, however they are able to find a valid path on the first try. For the 10 node PRM, multiple node placements are required before a valid path is found. For a system that requires high reliability, increasing the number of nodes may be worth the increase in computational complexity because a path is much more likely to be found. From our experimental trials, it can also be observed that increasing the number of nodes from 100 to 300 does not produce a significant increase in performance in terms of finding an optimal path. This suggests that there is an optimal number of nodes for a specific map and increasing the number of nodes further will not improve performance and will only increase complexity.

We expected that the higher number of nodes would always yield the best path to the target location, however our experimental results show that this is not always the case. For the first target located in the upper left corner of the map, the PRM with the least number of nodes produces the shortest path. Both paths produced by PRMs with 100 and 300 nodes contain portions around (x, y) = (10, 30) where the path seems to wander around before moving to the goal location. This is likely due to the PRM with 10 nodes having to go through multiple iterations of node placements before finding one that will allow connections around the obstacles, and since the nodes are scarce, the connections are more direct. The PRMs produced with many nodes are able to immediately find paths from the initial node placement since the placement is more dense, but the paths require traveling through more nodes which causes the wandering behavior observed in the first location trial.

For the second goal location placed in the bottom right corner of the map, the paths resulting from the three PRM node cases are similar in appearance and length. The shortest path is found by the PRM with 300 nodes, but only by a small margin. To get to this goal location, the robot does not have to navigate around many obstacles so it seems reasonable that all PRMs produce a similar path.

For the final goal location placed in the upper right area of the map, the paths produced by the PRMs with 100 and 300 nodes are significantly shorter than the path produced by the PRM with 10 nodes. Both the PRMS with 100 and 300 nodes take a path through the obstacles to reach the goal location, which is not possible for the PRM with 10 nodes. The path produced by this PRM has longer regions where the robot travels in a straight line and the path wraps around the perimeter rather than moving through the obstacles. Since there are fewer nodes, the PRM is forced to produce a path that has longer connection distances resulting in a less direct path. This target location shows the benefit of increasing the number of nodes in the PRM. The algorithm is much more effective at producing a path through a field of obstacles when the number of connection nodes is high, which is vital for complex real-world applications.

7 Conclusions.

In this paper we explore the use of the PRM algorithm for robot path planning. The PRM algorithm generates a graph of randomly placed nodes across a specified map. It adds edges to the graph between every node pair that is within a specified Euclidean distance of each other. It then searches through all possible combinations of nodes to find the most efficient path from a specified start location to a goal location using Dijkstra's Algorithm. We test the PRM algorithm's robustness by studying the application to three different maps with varying levels of complexity. We compare the effect of decreasing the number of connection nodes and maximum length of connection distance used in the PRM algorithm. Across all maps, having a larger number of nodes provides a higher likelihood of successfully finding a path than increasing the connection distance between nodes.

We apply the PRM algorithm with an autonomous Simulink bicycle robot control model to study the efficiency of its path taken to the goal location. By varying the number of nodes used by the PRM algorithm, we can change the path that the bicycle takes. We examine multiple goal

locations on various customized maps to compare the effect of the number of connection nodes on the bicycle's path. We find that while a high number of nodes in the PRM results in a high complexity graph, it is more effective at consistently finding a path to the goal location. Additionally, when the optimal path to the goal consists of mostly straight movement, the resulting path from PRMs with a large number of nodes vs a small number of nodes is not significantly different. However, if the optimal path requires moving around multiple obstacles, the PRM with a large number of nodes is much more effective at finding the shortest path.

Future work could add dynamics to the bicycle control Simulink model by exploring a more complicated robot which includes friction from the tires. This would add an additional challenge for the controller, especially if we randomized the properties of the map's surface on a given trial. Additionally, in this work we only explored the PRM algorithm, however other path planning algorithms could be implemented such as Rapidly-exploring Random Trees (RRT) to compare performance. Rather than starting with a specified number of nodes on a map, the RRT algorithm incrementally builds up a network until a solution is reached. This algorithm has the benefit of not needing to have prior knowledge of the map [10].

References

[1] "Plan Path for a Bicycle Robot in Simulink." *Plan Path for a Bicycle Robot in Simulink - MATLAB & Simulink*, MATLAB, 2019, https://www.mathworks.com/help/robotics/ug/plan-path-for-a-bicycle-robot-in-simulink.html.
[2] "ControllerPurePursuit." *Create Controller to Follow Set of Waypoints - MATLAB*, MATLAB, 2019, https://www.mathworks.com/help/robotics/ref/controllerpurepursuit-system-object.html.
[3] "Bicycle Kinematic Model." *Compute Car-like Vehicle Motion Using Bicycle Kinematic Model - Simulink*, Simulink, 2019, https://www.mathworks.com/help/robotics/ref/bicyclekinematicmodel.html.
[4] "Probabilistic Roadmaps (PRM)." *Probabilistic Roadmaps (PRM) - MATLAB & Simulink*, https://www.mathworks.com/help/robotics/ug/probabilistic-roadmaps-prm.html.
[5] L. E. Kavraki, P. Svestka, J. -. Latombe and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," in *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566-580, Aug. 1996, doi: 10.1109/70.508439.
[6] "MobileRobotPRM." *Create Probabilistic Roadmap Path Planner - MATLAB*, 2019, https://www.mathworks.com/help/robotics/ref/mobilerobotprm.html.
[7] Allen, Peter K. *Probabilistic Roadmap Path Planning*. 2015, http://www.cs.columbia.edu/~allen/F15/NOTES/Probabilisticpath.pdf.
[8] "Findpath." *Find Path between Start and Goal Points on Roadmap - MATLAB*, MATLAB, 2019, https://www.mathworks.com/help/robotics/ref/mobilerobotprm.findpath.html.
[9] "Dijsktra's Algorithm." *GeeksforGeeks*, 22 Feb. 2022, https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/.
[10] Karaman, Sertac, and Emilio Frazzoli. "Sampling-Based Algorithms for Optimal Motion Planning." *The International Journal of Robotics Research*, 22 June 2011, https://journals-sagepub-com.proxy.lib.umich.edu/doi/abs/10.1177/0278364911406761?casa_token=s-a8lSIVht0AAAAA:JPX6zNxzhIfA780DbanmDHrKmLHLGnXPYf9My6ZsTJC972cbNLnoSvOdRnJ8wzlQA2cvWQgm3qda.