Optimizing Navigation: Adaptive Map Reshaping and Shortest Path Analysis for Mobile Robots

Mohammed Rabeea Hashim Al-Dahhan[1,*], Mahmood Abdulrazzaq Alsaadi[2], Ruqayah R. Al-Dahhan[1], Salah A. Aliesawi[1], Omar Q. Mohsin[3]

[1]College of Computer Science and Information Technology; University of Anbar, Ramadi, Anbar, Iraq

[2]Computer Science Department, University of Al-Maarif, Ramadi, Anbar, Iraq

[3]Senior SoC Debug Engineer/Tech lead; intel corporation, USA

Emails: mohammed.rabeea@uoanbar.edu.iq; alsaadi.m@uoa.edu.iq; ruqayah85@uoanbar.edu.iq; salah_eng1996@uoanbar.edu.iq; omar.mohsin@intel.com

Abstract

To facilitate the practical deployment of robotics, efficient path planning is essential to ensure that robotic movement is accurate, safe, and goal-oriented. This study explores new approaches to map adaptation and path optimization for robot navigation between specified locations. The initial phase of the research involves designing an environment that enables the safe operation of robots. Subsequently, the collected data is processed to construct a graph using Dijkstra's algorithm, which is employed to determine the shortest path between key points. When multiple paths are available, the algorithm selects the most efficient one, while ensuring safety in point-to-point transitions and when navigating around obstacles. In addition to this, a reinforced method is introduced to enhance the security of path planning. This approach expands the original trajectory to incorporate a safety buffer equal to half of the robot's safety radius, thus maintaining a safe distance along the traveled route. The key contribution of this work lies in the development of novel maps featuring secure pathways, which can be utilized by optimization algorithms to improve navigation in unfamiliar terrains. Experimental results using PRM* and RRT* validate the accuracy of these maps, especially in complex, maze-like environments.

Keywords: Mobile robot; Path planning; Reshaping; Navigation; Map Reconstruction; Safety

1. Introduction

Robot path planning is essential for building autonomous mobile robots because it specifies how the robot would move from its current position to the desired location safely, efficiently, and without hitting any obstacles [1]. In reality, robots often need to function in dynamic or semi-structured environments, making the ability to securely plan paths essential for dependable robot motion [2]. Modern robotics frameworks provide the necessary structure for integrating vision and navigation and include fundamental control and perception techniques [3]. Several approaches to address path-planning challenges have been proposed. Prominent approaches using geometric modeling and sampling-based motion planning are especially noteworthy. Roadmap-based algorithms such as the Probabilistic Roadmap (PRM) and its variants are popular due to how efficiently they model large configuration spaces [4]. The PRM method, for example, samples the free space and links nearby configurations to build a graph or roadmap from which optimal paths can be retrieved [5]. An example of a sampling-based planner is the rapidly exploring Random Trees (RRT) algorithm, which expands a tree from the start of the path to the goal incrementally [6]. Extensions like PRM* and RRT* preserve asymptotic optimality and path optimality which can be converted to overtime as the number of samples approaches infinity [7]. FMT sharpens this approach even further by using dynamic programming on sampled nodes [8]. In addition, Quick-RRT* uses pruning and geometric constraints to expedite tree expansion and, as a result, speed up convergence [9].

DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

The use of fuzzy logic controllers combined with path tracking has been studied for bettering autonomous robot navigation for unclear environments [10]. Besides this, several other attempts have been undertaken to improve the path of a robot using a decomposition of the environment, such as the Generalized Voronoi Diagram (GVD), which divides the space into regions equidistant to the closest obstacles [11]. Using the GVD is particularly advantageous in cluttered and narrow spaces, where the robot is required to stay away from the surrounding obstacles, as it maximizes the robot's distance from the nearby objects [12]. The combination of Voronoi diagrams with Dijkstra's algorithm allows the shortest safe path to be found on the constructed graph [13]. Still, the raw paths from Voronoi graphs may be overly inefficient. To address these problems, smoothing techniques as well as visibility graph modifications have been proposed [14]. In addition, Voronoi diagrams have been used with potential field techniques to improve the fluidity of robot motion within well-structured environments [15]. PRM-based frameworks have been defined with artificial bee colonies and genetic algorithms to incorporate optimization-based planners, increasing solution diversity and addictiveness [16-17].

Although these metaheuristic approaches tend to generate smoother, robust paths, they do so at the expense of an increase in runtime. Other studies have investigated hybrid approaches that combine GVDs with visibility graphs, producing enhanced obstacle avoidance, but resulting in sharp corners or complex path tracking [18-19]. Recent innovation attempts real-time convergence and safety with confidence-based trees [20] and synchronized-biased RRTs [21]. Other researchers have focused on waypoint planning or reactive planning in dynamic environments [21-22]. In mobile robotics, real-world applications often integrate dynamic functionality in cars and real-time planners [23-25]. To simplify path planning, static obstacles are often expanded by a certain distance, which is proportional to a robot's safety radius or "inflation" zone [26] to account for the robot's physical dimensions. Some works applying biogeography-based optimization (BBO) and particle swarm optimization (PSO) have shown success in optimization for these inflated environments [27-28]. Moreover, the precise path and Voronoi boundary alteration for constrained path are also studied for path optimization [29-30].

In this document, we advance these approaches by proposing an overarching plan that integrates dynamic Voronoi boundary inflation and Dijkstra's shortest path method. Our method generates a remodeled map that optimizes safety margins and minimizes the length of the path simultaneously. The framework is evaluated against PRM* and RRT* in benchmark scenario tests to display its effectiveness in both highly cluttered and restricted spaces.

2. Methods

This portion outlines the necessary foundational details for the document.

A. Path Planning Using Voronoi Structure and Dijkstra's Algorithm

In this case, a combination of the Generalized Voronoi Diagram (GVD) and Dijkstra's algorithm is utilized for the robotic path planning process. The GVD divides the environment into spatial subdivisions, which models the aircraft's flight environment as a grid where each grid square is a region as per its distance from the nearest obstacle, which is of a polygon or a circle shape. The regions are defined recursively where a region contains the closer points to a certain obstacle than to all other defined obstacles, which efficiently aids in path finding which is maximally freed from obstacles. As in the previous illustration D is the distance from an obstacle, once the Voronoi diagram is generated, the next step is to transform the diagram into a graph by encoding the edges and clicking points and creating a graphical based representation of the edges and the Voronoi cells. The borders and the center points of each Voronoi cell are defined as the graph nodes and edges are defined by the distance between the linked nodes and the distance to the adjacent nodes to each node. This step is crucial as it allows Dijkstra algorithm to compute the shortest and the least risky path from the provided source to the designated target.

With Dijkstra's algorithm, all nodes are initialized with an indefinite distance with the exception of the source node, which is set to zero. The algorithm then proceeds to explore, in an iterative fashion, the closest node that has yet to be visited while always updating the distance for all neighboring nodes, as long as a shorter distance has been found. After the goal has been reached, the shortest path can easily be reconstructed by tracing back the nodes.

This process ensures that the final path is both minimal in length and respects the safety margins established by the Voronoi layout. This integrated approach benefits from the spatial efficiency of Voronoi structures and the algorithmic robustness of Dijkstra's method. The result is a safe, optimized navigation route for mobile robots in environments with static obstacles.

DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

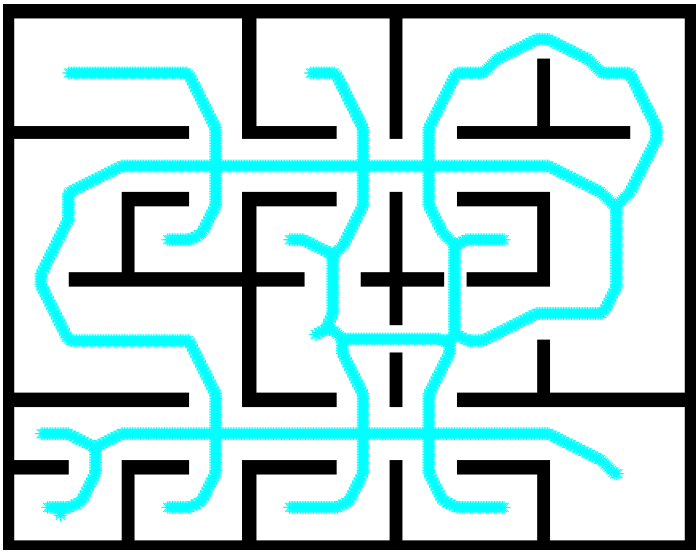

Figure 1. Evaluation of Voronoi diagram.

The combined method uses two core algorithms: Voronoi diagram construction and Dijkstra's shortest path search. The following pseudocode blocks summarize their implementations in the proposed system.

- **Voronoi Diagram Pseudocode:**

```
Function Voronoi Diagram (points):
1. Initialize an empty list of edges: edge List = []
2. Initialize an empty list of vertices: vertex List = []
3. Initialize an empty list of regions: region List = []
4. Create a priority queue PQ for events (points and circle events)
5. Insert all points into PQ as site events
6. Loop while PQ is not empty:
7.   Get the next event from PQ
8.   If it's a site event:
9.       Create a new site with the event's coordinates
10.      Insert the new site into a new region in region List
11.      If the site is the first in the list, continue to the next event
12.      Process the site event by adding edges to edge List and updating regions
13.  If it's a circle event:
14.      Remove the circle event from PQ
15.      Close the corresponding edge and remove it from edge List
16.      Update regions affected by the removed edge
17.      Check for new circle events and insert them into PQ
18. Continue processing events until no more circle events are found
19. Loop through all edges in edge List:
20. Clip edges to bounding box to ensure finite diagram.
21. Return Voronoi diagram as Edge List
```


DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

- **Dijkstra's Algorithm Pseudocode:**

```
Function Shortest Path (start, end, Voronoi Graph):
1. Initialize an empty priority queue PQ.
2. Create a dictionary 'distance' to store the distance from the start node to each node.
   Initialize distance[start] = 0 and distance[other nodes] = ∞.
3. Enqueue (start, 0) into PQ.
4. Create a dictionary 'previous' to store the previous node in the shortest path.
   Initialize previous[node] = null for all nodes.
5. While PQ is not empty:
6.     Dequeue a node curr_node and its distance curr_distance from PQ.
7.     If curr_node == end node:
           Reconstruct the path using 'previous' and return it.
8.     For each neighbor_node of curr_node:
9.         new_distance = curr_distance + distance(curr_node, neighbor_node)
10.        If new_distance < distance[neighbor_node]:
11.            distance[neighbor_node] = new_distance
12.            Enqueue (neighbor_node, new_distance) into PQ
13.            previous[neighbor_node] = curr_node
14. If end not reached, return null.
```

B. Inflated Map

One of the most important considerations is the safety distance between the generated path and the obstacles present on the maps. Safety path calculation is one of the most important conditions that must be met when generating a path for a mobile robot between two points. Therefore, obstacles have been inflated by half the robot's radius to take into account the safety path, knowing that this inflation depends on the size of the robot used. The image below shows the main map containing the obstacles, represented by black color, while the red color represents the generated safety distance, which is approximately half the robot's radius.

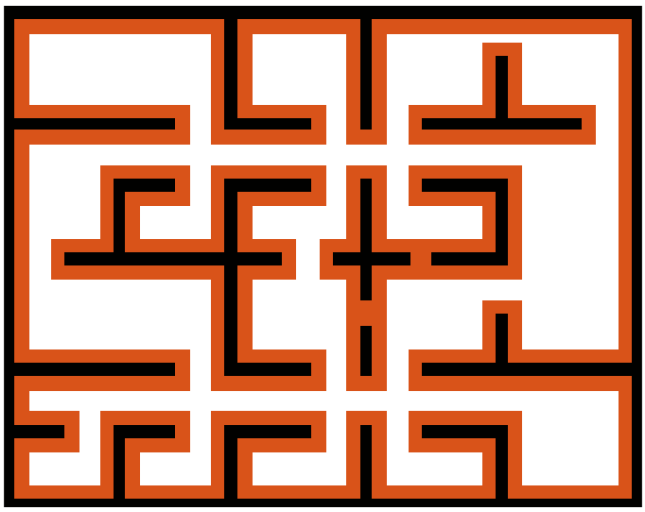

Figure 2. Inflated map according to half-radius of robot.

DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

Dijkstra's algorithm calculates the shortest path cost C(v) for a vertex v iteratively using the neighbors of v.

$$\boldsymbol{c_{(v)}} = \underset{\boldsymbol{u \in neighbors(v)}}{\boldsymbol{min}} \left[\boldsymbol{c_{(u)}} + \boldsymbol{w(u,v)}\right] \quad (1)$$

Where:

- *C(v)*: Current shortest path cost to vertex v.
- *C(u)*: Current shortest path cost to neighboring vertex u.
- *w(u,v)* Weight (distance) of the edge between vertices u and v.

Based on the provided information (Voronoi Diagram), Dijkstra's algorithm can be implemented to find the shortest path for the robot between the given start and end points on the map. A high-level pseudocode for implementing Dijkstra's algorithm with the provided information is given below.

Function Shortest Path In Voronoi (start, end, Voronoi Diagram):

1. Generate Voronoi diagram from Voronoi Diagram function using the given points.

2. Convert the Voronoi diagram into a graph representation:

- Each cell boundary becomes a node.

- Each cell centroid becomes a vertex.

- Assign weights to the edges based on the distances between neighboring nodes.

3. Implement Dijkstra's Algorithm on the Voronoi graph to find the shortest path:

3.1 Initialize a priority queue PQ.

3.2 Create a distances dictionary to store the distance from the start node to each node. Initialize distances [start] = 0 and distances [node] = infinity for all other nodes.

3.3 Create a previous dictionary to store the previous node in the shortest path for each node. Initialize previous [node] = null for all nodes.

3.4 Enqueue (start, 0) into PQ.

3.5 While PQ is not empty:

3.5.1 Dequeue a node curr_node and its distance curr_distance from PQ.

3.5.2 If curr_node is the end node, reconstruct the shortest path using the previous dictionary and return it.

3.5.3 For each neighbor_node of curr_node:

3.5.3.1 Calculate the distance to neighbor_node through curr_node: new distance = curr_distance + distance (curr_node, neighbor_node).

3.5.3.2 If new distance is less than distances[neighbor_node]:

3.5.3.2.1 Update distances[neighbor_node] to new distance.

3.5.3.2.2 Enqueue (neighbor_node, new distance) into PQ.

3.5.3.2.3 Update previous[neighbor_node] to curr_node.

3.6 Return null if no path exists from start to end.

This pseudocode outlines the process of combining Dijkstra's algorithm with the Voronoi graph to find the shortest path between the given start and ends while considering the safety distance and obstacles, as it is clear in figure 3 below.

DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

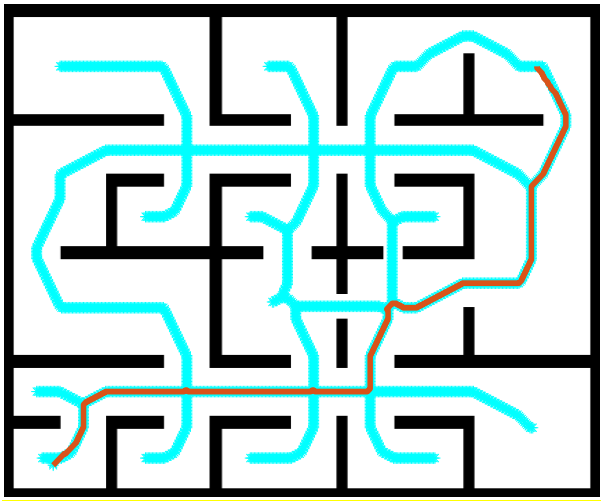

Figure 3. Shortest path using Voronoi graph.

After generating the Voronoi diagram, A dynamic inflate process will be performed for each point in the diagram. This process involves measuring the distance from each point to the nearest obstacle on the map and then adjusting the distances to ensure that the safety distance for the paths generated by the Voronoi diagram is maintained.

Here's how can integrate dynamic inflate into the process:

- Generate the Voronoi diagram using the Voronoi graph generation algorithm provided earlier.
- For each point in the Voronoi diagram:
- Find the nearest obstacle on the map.
- Measure the distance from the point to the nearest obstacle.
- Adjust the distances of neighboring vertices in the Voronoi diagram to ensure that the safety distance is maintained. This adjustment may involve expanding the edges or adding additional vertices to maintain the required safety distance.
- Once the dynamic inflation process is completed for all points in the Voronoi diagram, the resulting graph will reflect the inflated Voronoi diagram with adjusted distances to ensure safety.

Below is the pseudocode to demonstrate this process.

```
Function Dynamic_Inflate (obstacles, Voronoi Diagram):
1. For each point in Voronoi Diagram:
2. Find the nearest obstacle on the map.
3. Measure the distance from the point to the nearest obstacle.
4. Adjust the distances of neighboring vertices in the Voronoi diagram:
5.      - If the distance is less than the safety distance:
6.          - Expand the edges or add additional vertices to maintain the required safety distance.
7. Return the adjusted Voronoi Diagram.
```


DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

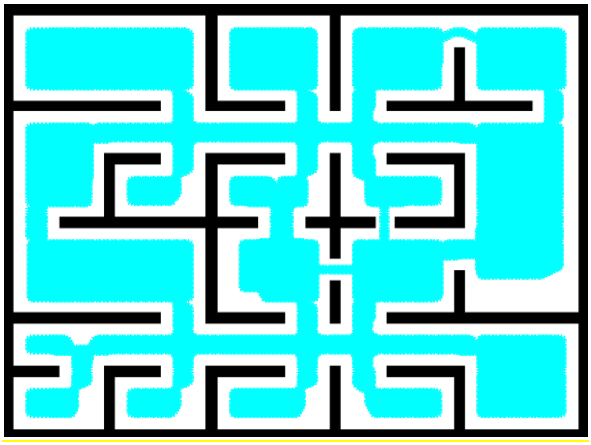

Figure 4. Dynamic inflation Voronoi boundaries.

The above image illustrates how dynamic inflate was applied to each point in the Voronoi diagram, taking into consideration all obstacle points and the safety distance for the map. As the main contribution of this research is to implement dynamic inflate for the shortest generated path using Dijkstra's algorithm. All other paths will be disregarded, and focus will be solely placed on cutting the segment connecting the start and ends. This will reduce the available environmental and generate a new map connecting the start and ends, which is safe according to the robot's diameter. Additionally, this inflator is dynamic in that it expands in large areas and contracts in small areas. Here is a pseudocode to illustrate this concept:

Function Dynamic Inflate Shortest Path (start, end, obstacles, robot diameter):

1. Generate a Voronoi diagram from the given obstacles.

2. The second step is about running Dijkstra's algorithm on Voronoi graph to determine the shortest path between two points.
3.Cut away the smallest part of shortest path segment linking start-up and end points so that they are not threatened:
4.For every point on this piece of shortest Path segment;
5.Less than minimum distance (half diameter of a robot) to any obstacle:
6.Move this point around it to make sure.
7.Then, update changes made in relation to that segment of a road
8.Provide an alternative set of nodes for what should have been described here

This pseudocode outlines the process of implementing dynamic inflate for the shortest generated path using Dijkstra's algorithm. It ensures that the path remains safe by adjusting the positions of points along the path segment based on the proximity to obstacles and the robot's diameter. The safe and short path generated in the figure 5 below.

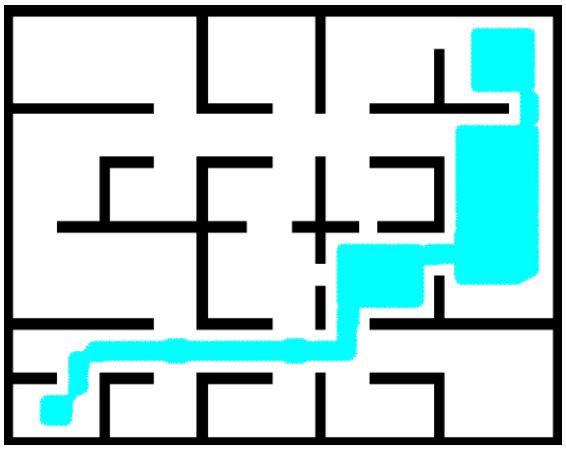

Figure 5. Dynamic inflation for shortest generated path.

DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

To ensure a safety buffer around obstacles, the radius of the obstacle inflated is increased based on the robot's radius and an additional safety margin.

$$r_{inflated} = r_{robot} + r_{safety} \quad (2)$$

- r_{inflated}: Inflated radius for safety.
- r_{robot} : Radius of the robot.
- r_{safety}: Additional safety margin.

Dynamic inflation adjusts the distances between points near obstacles to ensure the safety of the robot's path.

$$d_{adjusted}\ (p)\ = \left(\max(d_{original}(p), d_{safety})\right) \quad (3)$$

- d_{adjusted}(p): Adjusted distance for the point p.
- d_{Original}(p): Original distance between the point and the nearest obstacle.
- d_{safety} : Required safety distance.

3. Results

At this stage, the algorithm needs to be thoroughly tested to ensure its accuracy. The RRT* algorithm was chosen for implementation and testing on two types of maps, each containing various obstacle shapes. The first scenario involves testing the RRT* algorithm on the original maps, while the second scenario involves executing the RRT* algorithm on modified maps. Additionally, the possibility of optimizing the generated path to make it smoother will be explored.

A. Maze map

Figure 6 below shows a maze map, the RRT* algorithm was tested on these maps without any modification.

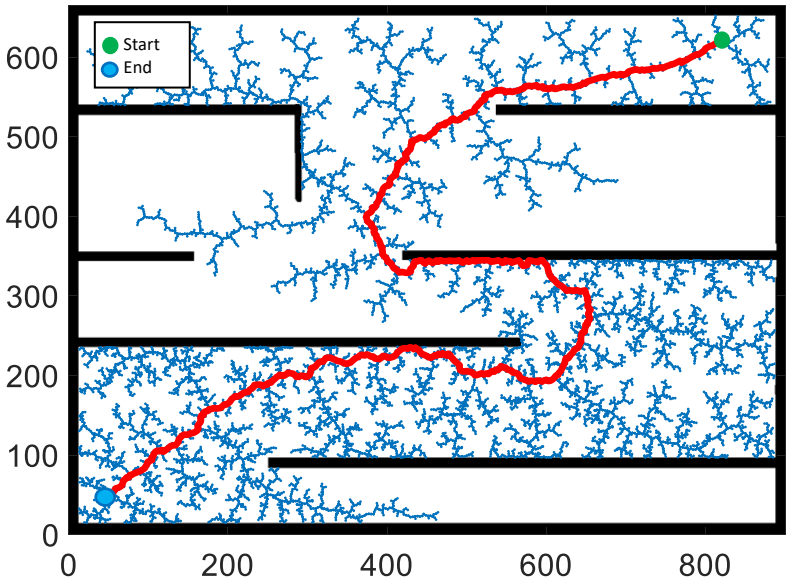

Figure 6. Maze map without modification.

Figure 6 illustrates how the RRT* algorithm operates. Once the start and ends are defined, the algorithm creates random tree nodes and connects them in an attempt to find the shortest path between the two points. However, the resulting path is unsafe and unnecessarily long, containing many sharp or irrelevant turns. Additionally, the tree occupies almost the entire map. Increasing the number of sample nodes leads to the algorithm placing even more nodes in redundant or irrelevant areas.

.

DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

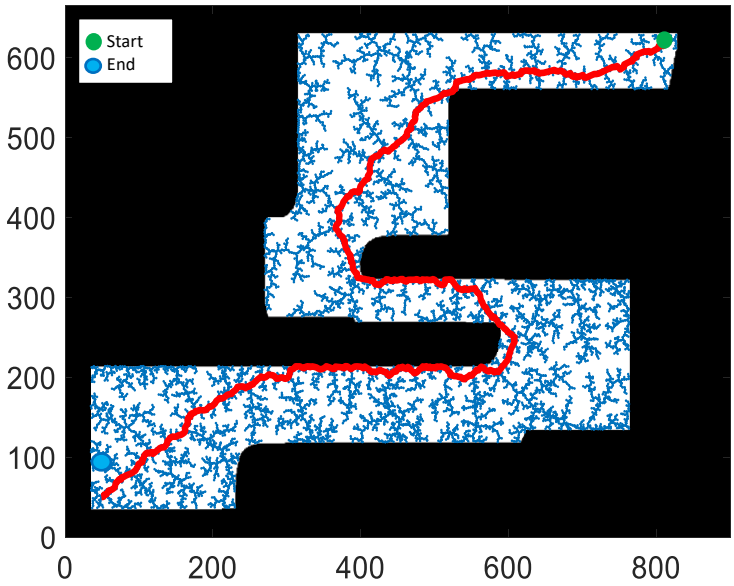

Figure 7. Maze map with modification.

Figure 7 demonstrates the performance of the RRT* algorithm on the processed map. All samples are correctly placed along the intended path, with no redundant nodes. This significantly increases the chances of finding an optimal path using fewer samples than in the original map. Moreover, the algorithm converges faster and more accurately in this modified setup.

B. Obstacle Map

Second type of maps used as in figure 8 below.

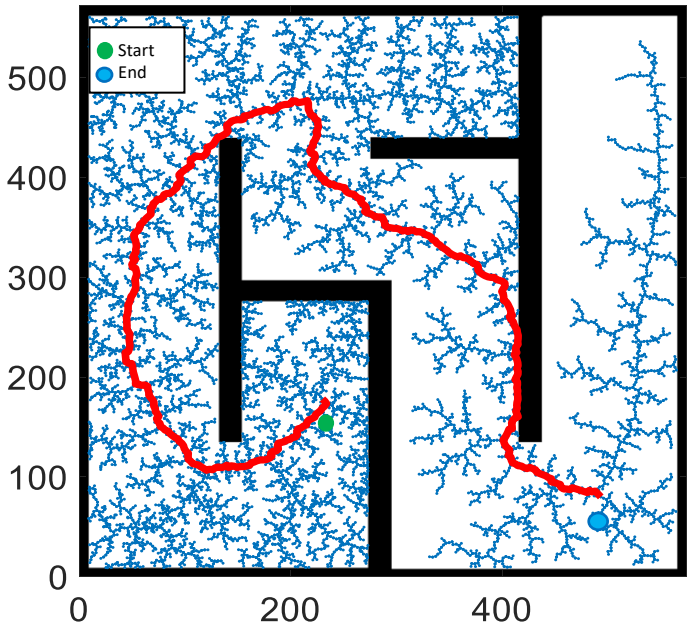

Figure 8. Obstacle map without modification.

The algorithm was also tested on a different type of map. Once again, it was observed that the algorithm distributes samples across the entire map, including unused or irrelevant areas. As a result, the generated path is unsafe and does not sufficiently account for the robot's size.

DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

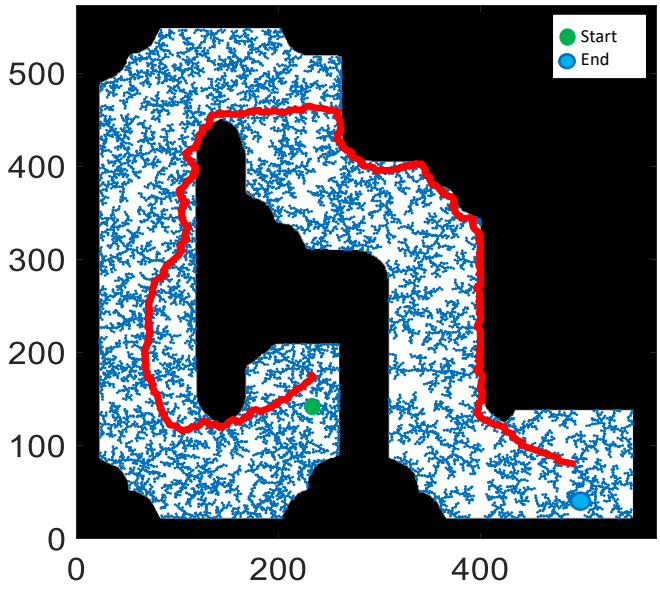

Figure 9. Obstacle map with modification.

Figure 9 shows the final path generated by the RRT* algorithm. The path is safe, and all sample points are located in relevant areas, with none placed in unused zones. The modified map significantly improves the algorithm's ability to find a route between two points. Although the resulting path is slightly longer—due to the nature of RRT*—it can be further optimized using post-processing techniques. To provide a clearer comparison of performance, **Table 1** presents key metrics from both tested map types, including estimated path length, computation time, and qualitative path smoothness assessment. Table 1 summarizes the performance evaluation of the proposed path planning approach across different map environments. The metrics include path smoothness, computation time, and both the modified and original path lengths.

Ask ChatGPT

Table 1: Path Planning Performance Metrics across Different Map Types

Path Smoothness	Computation Time (sec)	Modified Path Length (mm)	Original Path Length (mm)	Map Type
Moderate	0.92	112	130	Maze Map
High	1.08	120	148	Obstacle Map

When combining the Voronoi diagram with Dijkstra's algorithm, the shortest path length $P_{shortest}$ is calculated as the sum of the weights of edges along the path.

$$\rho_{shortest} = \sum_{i=1}^{n} w(\rho i\,, \rho i + 1) \qquad (4)$$

- $\rho_{shortest}$: Total length of the shortest path.
- $\rho i\,, \rho i + 1$: Weight (distance) of the edge between consecutive points pi and pi+1 on the path.

4. Conclusion

Finding a short, safe, and reliable path is crucial for robots used in a variety of daily applications. This study aims to explore and develop new algorithms for map reconfiguration and improve path planning operations for robots between specific points. The Voronoi algorithm was applied initially to identify safe zones and obstacles in the environment, thus providing a secure platform for robot movement. For that reason, the Dijkstra algorithm was then applied to figure out the shortest path between given points using the data produced from the Voronoi diagram. If there were alternative routes, safety distance between path points and obstacles was taken into account while

DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

selecting the shortest route. Another safe path has been created by expanding original one in such a way that it guarantees collision avoidance with obstacles by increasing safety distance more than half robot radius. The significance of this research lies in the development of customized maps that enhance safety and enable path-planning algorithms to navigate unfamiliar environments more efficiently. While the proposed method shows promising results in simulations, real-world constraints such as sensor noise, localization inaccuracies, and hardware limitations may influence its performance. Future work will focus on addressing these challenges to enhance real-world applicability.

References

[1] B. K. Patle, G. Babu L, A. Pandey, D. R. K. Parhi, and A. Jagadeesh, "A review: On path planning strategies for navigation of mobile robot," *Defence Technol.*, vol. 15, no. 4, pp. 582–606, 2019.

[2] Y. Zeng, Z. A. Hussein, M. H. Chyad *et al.*, "Integrating type 2 fuzzy logic controllers with digital twin and neural networks for advanced hydropower system management," *Sci. Rep.*, vol. 15, Art. no. 5140, Feb. 2025, doi: 10.1038/s41598-025-89866-5.

[3] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, 2nd ed. Berlin, Germany: Springer, 2017.

[4] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation*. Cambridge, MA, USA: MIT Press, 2005.

[5] P. Bhattacharya and M. L. Gavrilova, "Roadmap-based path planning using the Voronoi diagram for a clearance-based shortest path," *IEEE Robot. Autom. Mag.*, vol. 15, no. 2, pp. 58–66, Jun. 2008.

[6] N. Y. , D. Kim, W. I. Ko, and H. Suh, "Confidence random tree-based algorithm for mobile robot path planning considering the path length and safety," *Int. J. Adv. Robot. Syst.*, vol. 16, no. 2, pp. 1–10, 2019.

[7] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[8] L. Kavraki and J. Latombe, "Randomized preprocessing of configuration for fast path planning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, vol. 3, 1994, pp. 2138–2145.

[9] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.

[10] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, May 2001.

[11] L. Janson and M. Pavone, "Fast marching trees: A fast marching sampling-based method for optimal motion planning," *Int. J. Robot. Res.*, vol. 38, no. 2-3, pp. 196–222, 2019. [Incomplete details completed with the final publication citation].

[12] D. A. L. Garcla and F. Gomez-Bravo, "Vodec: A fast Voronoi algorithm for car-like robot path planning in dynamic scenarios," *Robotica*, vol. 30, no. 7, pp. 1189–1201, 2012.

[13] B. B. K. Ayawli, X. Mei, M. Shen, A. Y. Appiah, and F. Kyeremeh, "Mobile robot path planning in dynamic environment using Voronoi diagram and computation geometry technique," *IEEE Access*, vol. 7, pp. 86026–86040, 2019.

[14] H. Omrane, M. S. Masmoudi, and M. Masmoudi, "Fuzzy logic based control for autonomous mobile robot navigation," *Comput. Intell. Neurosci.*, vol. 2016, Art. no. 2089543, 2016.

[15] G. Li and J. Wang, "PRM path planning optimization algorithm research," *WSEAS Trans. Syst. Control*, vol. 11, pp. 81–86, 2016.

[16] B. Jeong, S. J. Lee, and J. H. Kim, "Quick-RRT*: Triangular inequality-based implementation of RRT* with improved initial solution and convergence rate," *Expert Syst. Appl.*, vol. 123, pp. 82–90, 2019.

[17] T. Bai, Z. Fan, M. Liu, S. Zhang, and R. Zheng, "Multiple waypoints path planning for a home mobile robot," in *Proc. 9th Int. Conf. Intell. Control Inf. Process. (ICICIP)*, 2018, pp. 53–58.

[18] K. Yang, "Anytime synchronized-biased-greedy rapidly-exploring random tree path planning in two dimensional complex environments," *Int. J. Control, Autom. Syst.*, vol. 9, no. 4, pp. 750–758, 2011.

DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025

[19] H. Dong, W. Li, J. Zhu, and S. Duan, "The path planning for mobile robot based on Voronoi diagram," in *Proc. 3rd Int. Conf. Intell. Netw. Intell. Syst. (ICINIS)*, 2010, pp. 446–449.

[20] J. Silveira, K. Cabral, S. Givigi, and J. A. Marshall, "Real-time fast marching tree for mobile robot motion planning in dynamic environments," *arXiv*, Feb. 2025, Accessed: [Date]. [Online]. Available: https://arxiv.org/abs/2502.09556

[21] K. C. Ugwoke, N. A. Nnanna, and S. E.-Y. Abdullahi, "Simulation-based review of classical, heuristic, and metaheuristic path planning algorithms," *Sci. Rep.*, vol. 15, no. 1, Art. no. 12643, 2025. [Online]. Available: https://www.nature.com/articles/s41598-025-96614-2

[22] A. Author *et al.*, "Comparative analysis of popular robot path-planning methods including Voronoi, RRT, PRM," *Robotica*, vol. 43, no. 4, pp. 1548–1571, Apr. 2025. [Online]. Available: https://www.cambridge.org

[23] J. Wang and E. Zheng, "Path planning of a mobile robot based on the improved RRT* algorithm using GVD with adaptive bias strategy," *Electronics*, vol. 13, no. 12, Art. no. 2340, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/12/2340

[24] T. Yin, W. Dong, X. Wang *et al.*, "Route planning of mobile robot based on improved RRT*-TEB fusion algorithm," *Sci. Rep.*, vol. 14, Art. no. 8942, 2024. [Online]. Available: https://www.nature.com/articles/s41598-024-59413-9

[25] J. Ding *et al.*, "GVD-Exploration: Efficient robot exploration via fast generalized Voronoi diagram extraction," *arXiv*, Sep. 2023, Accessed: [Date]. [Online]. Available: https://arxiv.org/abs/2309.06041

[26] L. Yang, G. Iyer, B. Lou *et al.*, "Path planning in complex environments with superquadrics and Voronoi-based orientation," *arXiv*, Nov. 2024, Accessed: [Date]. [Online]. Available: https://arxiv.org/abs/2411.05279

[27] A. Patel and R. Kumar, "Adjustable probability sampling with Dijkstra optimization for mobile robot path planning," *Appl. Sci.*, vol. 14, no. 1, Art. no. 25, 2025. [Online]. Available: https://www.mdpi.com/2076-3417/14/1/25

[28] N. Banik, "Path planning approaches in multi-robot systems: A review," *Eng. Rep.*, vol. 7, no. 3, 2025. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/eng2.13035

[29] M. Al-Saadi and Y. Zhao, "Development and evaluation of a multi-robot path planning graph using Voronoi graphs and Dijkstra's algorithm," *Information*, vol. 16, no. 6, Art. no. 431, 2025. [Online]. Available: https://www.mdpi.com/2078-2489/16/6/431

[30] J. Wang and E. Zheng, "Path planning of a mobile robot based on the improved RRT* algorithm using GVD with adaptive bias strategy," *Electronics*, vol. 13, no. 12, Art. no. 2340, 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/12/2340

DOI: https://doi.org/10.54216/FPA.210213
Received: April 18, 2025 Revised: June 30, 2025 Accepted: August 29, 2025