IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Journal Européen des Systèmes Automatisés
Vol. 59, No. 2, February, 2026, pp. 395-402
Journal homepage: http://iieta.org/journals/jesa

Mobile Robot Path Planning Based on Voronoi Diagram and Bidirectional Rapidly-Exploring Algorithm in Narrow Environments

Suha A. Khudhair*, Wesam M. Jasim

College of Computer Science and Information Technology, University of Anbar, Ramadi 31001, Iraq

Corresponding Author Email: suh23c1007@uoanbar.edu.iq

https://doi.org/10.18280/jesa.590209

Received: 16 September 2025
Revised: 13 January 2026
Accepted: 1 February 2026
Available online: 28 February 2026

Keywords:
robot path planning, Bidirectional Rapidly-exploring algorithm, Voronoi diagram, narrow environment

ABSTRACT

Path planning for mobile robots and their integration into narrow environments is a challenging process. That typically requires a balance among the most accurate path, safety, and computational efficiency. This paper presents a hybrid approach for planning the path of mobile robots in narrow complex environments. This approach is based on integrating the Voronoi diagram (VD) with the Bidirectional Rapidly-exploring algorithm (Bi-RRT). The main purpose of the integration process is to achieve a balance between computational efficiency on the one hand and operational security (avoiding obstacles) on the other hand. Voronoi helps Bi-RRT to get the path better by dividing the area of the environment to peace. The experimental results show that the Voronoi-guided sampling helps construct routes that are naturally farther away from obstacles that get in the way. The main contribution in this paper is combining Bi-RRT with VD which makes path planning faster, shorter, more reliable, and of better quality. It shows that the combination method improves the desired results with short and safe path from the starting point to the end point in narrow environment.

1. INTRODUCTION

Recently, mobile robots are being used in a lot of various fields, such as entertainment, medicine, mining, rescue, education, the military, aerospace, and even farming, to name a few [1]. Path planning is a key area of research in robotics that looks for a way for a robot to get from one place to another without running into obstacles or wasting time [2]. Path planning can be classified into dynamic and static path planning based on the environment. The environment that includes moving obstacles is the dynamic environment; while the static environment objects does not move [3]. There are two elements in robot path planning: global path planning and local path planning [4]. In global path planning, the mobile robot understands where the barriers are, what kind of environment it is in, and where it wants to go [5]. Local path planning, in which mobile robots that have no prior knowledge of their environment and rely on a local sensor to collect data and then construct a new path in response [6]. In touch with it, the smoothness, length, safety, and computation time of the path design are some of the issues that are looked at [7]. Researchers are still having a hard time with path design, especially when it comes to creating paths in messy places [8]. To solve the difficulty of planning the path of a mobile robot, researchers have looked into roadmap path planning approaches as probabilistic roadmap (PRM), visibility graph (VG), rapidly exploring random trees (RRT), and Voronoi diagram (VD) [9]. The VD is a useful tool for getting through tight spaces and mazes. This strategy is the safest for the planner since it makes sure that the robot stays as far away from any impediments as possible. With the use of VD, certain algorithms have already gotten close to real-time efficiency in planning the paths of mobile robots [10]. The widely used RRT technique is a type of sampling-based algorithm that may avoid complicated space constructions by using a collision check module. This makes it a good choice for addressing planning problems with many dimensions or constraints. But when the RRT algorithm has to deal with complicated situations like mazes, tight paths, and concave traps, it usually doesn't work as well [11]. The Bi-RRT method is a more advanced way to design paths that improves from the RRT algorithm. It is an algorithm for searching a random tree in both directions. The Bi-RRT method starts searches from both the start and goal positions at the same time, which is different from the typical RRT algorithm [12].

The main contributions of this paper are summarized as follows:

1. Introducing the VD to divide the area of research improves the work of Bi-RRT to find the safest and shortest path.
2. The use of VD leads to enhancing the proposed path planning algorithm to move safely in a narrow environment.

The rest of this paper is organized as follows: Section 2 illustrates some of the previous works related to this paper's topic. The methodology phases, such as VD, RRT, Bi-RRT, and the combinations, are demonstrated in Section 3. The obtained results are discussed and compared with the existing works in Section 4 to show the effectiveness of the proposed combination. Section 5 presents the conclusions.

2. RELATED WORK

The researchers faced the path planning challenge as the most interesting topic. The basic purpose of the robot problem is to give a mobile robot a safe place to be. The path must also be the best one. In the literature, there have been different studies written about the mobile robot path planning problems. The authors in a recent work [10] propose a new way for tackling the narrow passage problem in both 2D and 3D environments using superquadrics (SQ) representation and VD together. This approach uses the SQ formulation to make obstacles bigger, which makes it impossible to go through them, and the Voronoi hyperplane to make sure the path is as clear as possible. The outcome was that the robot had more possible courses (i.e., paths that were less likely to cause collisions), bigger safety margins from obstacles, orientation guidance (in line with the direction of the limited corridor), and quicker calculation. Authors in a previous study [13] uses the VD to avoid obstacles and ant colony optimization to determine the best path. By focusing on avoiding obstacles to ensure safety while maximizing the shortest path, the combination of both strategies offers a useful addition to robotic path planning. Experimental results demonstrate that the hybrid method works well for the desired goal. Based on the role of sample points in RRT. First, based on how the random tree is growing toward the present sampling point, a greedy sampling space reduction method was suggested in this study [14]. This technique changes the sample space dynamically to cut down on the random tree's unnecessary growth in space. Second, a novel way to judge narrow passages was suggested based on the area around the sample location. It cuts down on planning time and memory needs and makes pathways that are more realistic and closer to the best ones. It works better than baseline RRT and RRV. A generic autonomous path planning method called Node Control (NC-RRT) that was based on the structure of the RRT algorithm was suggested in this study [15]. In which, First, a way to progressively change the sampling area is suggested to help with exploration, which speed up the search. The node control technique is also included to limit the extended nodes of the tree. This helps to cut down on the number of incorrect nodes and get the boundary nodes (or near-border nodes). The result was shorter planning periods and improved success rates in tight or congested spaces. fewer pointless excursions more dependable pathways that don't hit the border or obstacles. A heuristic path planning algorithm based on a Generalized Voronoi Diagram (GVD) that creates a heuristic path, helps the RRTs' sampling process, and makes the motion planning of RRTs more efficient was presented in this study [2]. As a result, the final paths tend to have more clearance and be smoother because they were first aligned with the GVD.A new algorithm called complex environments rapidly-exploring random tree (CERRT) was proposed in this study [11] for the robot path planning problem. The RRT algorithm has problems with path planning for mobile robots in complex environments because it is not sensitive to the environment, is not very efficient, and produces poor path quality. It solves these problems by combining proactive extension, environmental awareness, and bidirectional optimization, making it very useful for real-world robot navigation in tight or obstacle-dense environments. The results show that CERRT works better than RRT and RRV. The performance metrics for RRV and RRT improved, with shorter planning times, fewer sampled nodes, greater success rate, shorter and smoother final pathways. By adding random sampling, the proposed model in this study [16] improves the RRT algorithm's ability to create routes for self-driving cars. First, it was suggested to utilize a variable steering angle approach to reduce the number of search nodes in the random tree because the traditional RRT method is random. Second, a collision detection approach that uses Kalman filtering was suggested to filter out collision points and get rid of extra nodes. This makes the path planning more accurate by lowering the number of valid points on the planned path. Lastly, Bezier curves are used to smooth out the route and adjust how curved it is to make sure it stays smooth. The method is both quick and accurate since it uses steering angle control, Kalman filtering, and Bezier smoothing. This makes it a great choice for planning autonomous driving paths in complicated settings. An enhanced Probabilistic Road Map PRM was proposed in this study [17] to fix the problems that PRM has when it comes to dealing with tight passages. By improving the typical PRM sampling approach, the number of sample sites in a tight passage was enhanced. It helps the algorithm adjust better when it comes across a narrow route. A previous study [18] presents the "Herarchical Annotated Skeleton-guided RRT" (HAS-RRT) as novel way to design paths that uses RRT and topological skeletons (workspace graphs) to enhance the search, especially in huge and difficult regions. HAS-RRT speeds up motion planning by a lot without lowering the quality of the paths. It works better than RRT versions because it cuts down on runtime by up to 91%, makes the system more reliable (higher success rates), keeps the path quality good, and is adaptable to places where guidance is weak or missing.

Moreover, a new approach called the adjustable probability and sampling area and the Dijkstra optimization-based RRT algorithm (APSD-RRT) was suggested in this study [19]. This method has two parts: an APS-RRT planner and an optimizer that deal with problems with the RRT algorithm, such how slowly it plans. High unpredictability and bad path quality after a number of tests show that this strategy can do a lot better when it comes to balancing processing cost and performance. A simple approach was presented in a prior work [20] to improve bidirectional heuristic search such that it can find rapid and limited sub-optimal solutions. Bidirectional heuristic search has to find a balance between making sure that borders cross and keeping planning efficiency high. This method use of the enormous free space in high Degrees-of-Freedom (DoF) motion planning to link frontiers using an extend operator that is common in sampling-based planners. It gets much quicker search results than previous search-based methods in simulated tests. It also makes progress in closing the performance gap between heuristic-based and sampling-based planners in high DoF motion planning.

3. METHODOLOGY

3.1 Voronoi diagram construction

In robot path planning, VD divides the robot's workspace into numerous areas, and the boundaries of these areas make up a Road Map RM. Each Voronoi area is closer to the node that made it than to any other node that made it. The generating obstacles are closer to any node in any area than to any other obstacle. The neighbor rule is used to find edges that are the same distance from the nodes. It has a configuration space S with nodes S={s1,s2,s3,...,si}, a distance function d, and *N*

indicating a set of nodes in *S*, Voronoi region is made up of a group of nodes Ni in S that are closer to one another than they are to other nodes Nj where Ni ≠ Nj. [21]. So it may use Eq. (1) to find the VD.

$$Vd = \{s \epsilon S \mid d(x,\ Ni) \leq\ dx, Nj \forall i \neq j\} \quad (1)$$

In our method, VD construction starts with point sampling. This is a sampling method where points are evenly spaced out with a fixed minimum separation, distance (radius) that was set to 5 by default. This means that the minimum distance between any two sampled points is 5. This parameter determines the density and uniformity of the resultant distribution.

So the goal of point generation is to distribute points in a selected environment in a way that ensures appropriate spacing between them (positional disks). This mechanism is achieved by generating random points ≥ 1875 with the condition that no two points are closer than the specified radius, as shown in Figure 1.

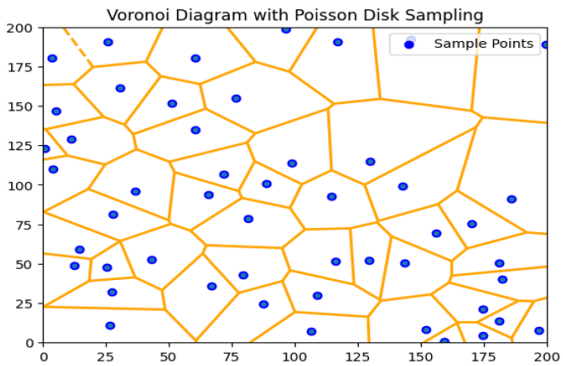

Figure 1. Voronoi diagram (VD)

Then add boundary points ,which are four points that are very far away from each other, and add them to the set that was made. This is a technical step, in which *scipy.spatial.Voronoi function* has been taken. The Voronoi function takes care of the edges of the diagram and stops regions from going on forever**.** The *scipy.spatial.Voronoi function* is used to make the VD from these locations. This function divides the space into areas, each of which contains all the points that are closer to one input point than to any other. We get the vertices nodes and edges of the VD that we made. After that, the edges are processed again to find the ones that are inside the environment's borders and to figure out which of the initial input points are neighbors with each other since their Voronoi cells share a boundary.

3.2 Obstacle representation and collision verification

The function create_obstacles_from_list. The function takes four numbers for each obstacle: (x), (y), (w), and (h). The numbers (x) and (y) denote where the obstruction is in two-dimensional space. The number (w) shows how far along the (x)-axis the distance is, while the number (h) shows how far along the (y)-axis the distance is. You can use these numbers to construct a rectangle that displays the barrier in a geometric style. The first point, (x, y), marks the lower-left corner. The second point, (x + w, y), marks the lower-right corner after adding the width. The third point, (x + w, y + h), marks the upper-right corner after adding both width and height. The fourth point, (x, y + h), marks the upper-left corner after adding the height only. These four points constitute a closed rectangle that represents the blockage in space. After changing all the elements in the list this way, the resulting rectangular shapes are stored together, and a combined spatial form is also made to show them as one blocked area when they touch or overlap. This offers us both a discrete geometric representation for each barrier and a combined one that we can use to check for collisions and figure out the best way to get around in the environment. Using the (in_collision) and (edge_in_collision) functions to check for collisions ensures that points or edges don't cross any obstacles.

3.3 Standard rapidly exploring random trees

The STANDERD-RRT method is what RRT-VD is based on. STANDERD-RRT keeps an underlying tree data structure. First, Standerd-RRT constructs a tree that starts at xinit. In each new iteration, xrand is found by randomly sampling in the configuration space X, going through the current nodes in the tree, choosing the node xnear that is closest to xrand, and making xnew using the steering function with a given expansion size. If the edge {xnear, xnew} is not in Xobs, xnew is added to the tree as a child of xnear, and the edge {xnear, xnew} is noted. When xnew is in the range of Xgoal the planning is over and the tree is sent back. The return fails if the time-out or the number of iterations is too high. The principle of RRT algorithm is illustrated in Figure 2. RRT-Biased is a good way to make RRT better. The Xgoal is set as the sample point based on a particular probability. This makes the tree grow quicker toward the goal region.

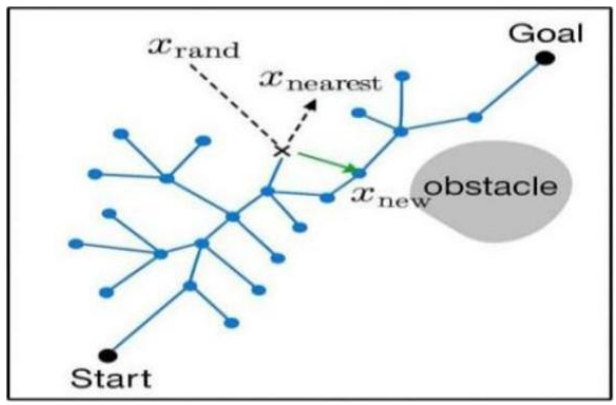

Figure 2. Principle of rapidly exploring random trees (RRT) algorithm

3.4 Rapidly exploring random trees-Voronoi diagram

The first function in this approach is to get the Voronoi_graph that was made and the union of the obstacle polygons as input. It goes through all the edges in the VD and sees whether any of them cross any of the obstacles. A free edge is one that does not touch any obstacles. It is added to the free edge list. This function does a good job of finding the areas of the VD that are in free space and may be utilized to design a path. The second function (sample-from-voronoi-edges) picks a random sample point from one of the free edges that the first function found. It initially looks to see whether there are any free edges. If not, it returns nothing. Eqs. (2) and (3) below are used to identify a sample or point on the free edge (p1, p2):

$$x = x1 + t*(x2 - x1) \quad (2)$$

$$y=y1+t*(y2-y1),\ t\epsilon[0,1] \quad (3)$$

$$Sample = (x, y) \quad (4)$$

RRT-VD algorithm begins by initializing a search tree that initially comprises only the start node, along with a reference structure that ties each node to its parent, enabling reconstruction of the path once the objective is achieved. The program then conducts an iterative loop that forms the core of incremental tree growth. During each iteration, a spatial sample is collected by choosing a point along a Voronoi edge that does not collide. It has a "goal bias" feature, which means that given a certain chance (goal bias), each iteration will generate a random number between 0 and 1. If the random number is less than the goal base, it will return the goal position as sample. This helps guide the search toward the target. If the target is not sampled, it tries to choose a random edge from the free edge list and a random location along that edge. It then checks to see if this location is hitting obstacles. The sample point is returned if it does not hit obstacles. The function attempts this sampling procedure up to tries number identifying the nearest node in the current RRT to this sample, aiming to expand the RRT towards the sample using the steer function. This function tries to "steer" from the nearest to the sample by utilizing the vector equation to find the vector between the two nodes using Eq. (5).

$$\overrightarrow{v} = Xsample - Xnearest \quad (5)$$

$$dist\ i = \sqrt{(Xi - Xs)^2 + (Yi - Ys)^2}$$
$$d = \| Xsample - Xnearest \| \quad (6)$$

$$direaction = \frac{Xsample - Xnearest}{\| Xsample - Xnearest \|} \quad (7)$$

$$\text{Xnew= Xnearest+} \frac{\text{Xsample -Xnearest}}{\|\text{Xsample -Xnearest}\|} * \eta \quad (8)$$

Calculating the distance between the nearest node and the sample could be using Eclidiance Eq. (6). If the distance to the sample is smaller than the step size (η), it just gives back the sample. Otherwise calculate the direction using Eq. (7) and then calculate new node using Eq. (8).

This is a key part of RRT that grows the tree toward the sampled point, checks for collisions along the new edge, and adds the new node to the RRT if it is legitimate. It would also see if the new node is close enough to the target to be considered the path discovered.

In this paper, the RRT algorithms that use VD was employed to achieve biased sampling in our method RRT-VD. They don't sample the full environment at random; instead, they just sample the edges of the VD that are in free space. This might make exploring and identifying paths faster and more efficient, especially when obstacles are convoluted. Algorithm 1 shows the RRT-VD steps.

Algorithm 1. Rapidly exploring random trees-Voronoi diagram

Algorithm RRT_Voronoi_Search(start, goal, obstacles_union, free_edges,width,height,max_iters,step_size,goal_threshold ,goal_bias)

Input : start, goal , obstacles_union ,free_edges ,width, height, max_iters ,step_size, goal_threshold , goal_bias
Output: rrt_nodes ,parents

1: rrt_nodes ← [start]
2: parents ← dictionary
3: parents[0] ← None
4: For i from 1 to max_iters do
5:sample←SampleFromVoronoiEdges(free_edges, obstacles_union, goal_bias = goal_bias, goal = goal)
6: If sample is None then
7: Continue to next I
8: End If
9: nearest_idx ← NearestNode(rrt_nodes, sample)
10: nearest_pt ← rrt_nodes[nearest_idx]
11: new_pt ← Steer(nearest_pt, sample, step_size)
12: If NOT EdgeInCollision(nearest_pt, new_pt, obstacles_union) then
13: Append new_pt to rrt_nodes
14: new_idx ← length(rrt_nodes) - 1
15: parents[new_idx] ← nearest_idx
16: If Distance(new_pt, goal) < goal_threshold then
17: Append goal to rrt_nodes
18: goal_idx ← length(rrt_nodes) - 1
19: parents[goal_idx] ← new_idx
20: Return rrt_nodes, parents
21: End If
22: End If
23: End For
24: Return rrt_nodes, parents
25: End Algorithm

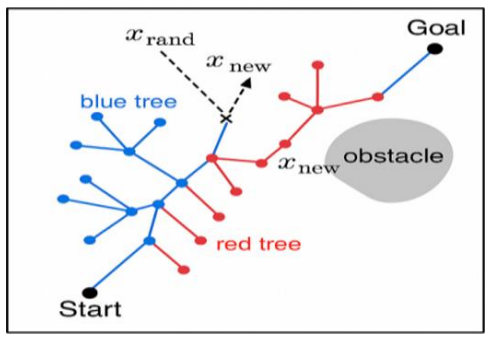

Figure 3. Principle of Bidirectional rapidly-exploring (Bi-RRT) algorithm

3.5 Bidirectional Rapidly-exploring algorithm

The Bi-RRT method is a more advanced way to design a path that comes from the RRT algorithm. It is an algorithm for searching a random tree in both directions. The Bi-RRT method starts searches from both the start and goal positions at the same time, creating two separate random trees. This is different from the typical RRT algorithm. These trees grow at the same time in the sample space, using random sampling to explore it. The algorithm will keep improving the path until it reaches the maximum number of iterations. As the trees grow, they may cross and link. This restriction on iterations stops the algorithm from running forever and makes sure that a workable route is found in a reasonable amount of time. The Bi-RRT algorithm's ability to search in two directions at once makes it much better at discovering paths since it uses the search space more effectively. The principle of Bi-RRT

algorithm is demonstrated in Figure 3.

3.6 Bidirectional Rapidly-exploring algorithm-Voronoi diagram

The proposed Bi-RRT-VD technique looks at the surroundings and finds barriers and open space. It then creates a VD, where each edge shows the locus of points that are the same distance from obstacles. These edges are in open, safe areas. They help the tree grow by showing it how to do so. Bi-RRT makes two trees, tree A as a root at the beginning and B as a root at the end. Each tree will grow on its own, trying to meet in the center. The two trees will also choose a sample from the free edge of the VD that doesn't have any impediments in the way (RRT-VD). This makes it more likely that spots distant from barriers will be chosen, which encourages people to explore vast, secure pathways. From the point Xrandm that was sampled. In Tree A, find the node xnear that is closest. Make a new setup Xnew is the result of advancing from Xnear to Xrand in modest steps. Tree A now has Xnew. Tree B tries to reach out to the new node. Xnew If it works, a route will be made between the two trees. Now the roles of the trees are switched: tree B will grow and tree A will try to link. This process goes on until a path is identified. Once the two trees are connected, the path from the start, the connecting point, and the objective may be rebuilt by following the branches of both trees. Algorithm 2 shows the proposed Bi-RRT-VD steps.

Algorithm 2. Bidirectional Rapidly exploring trees algorithm-Voronoi diagram

```
Algorithm BI_RRT_VORONOI_SEARCH(start, goal, obstacles_union, free_edge, width, height, max_iters, step_size, goal_threshold, goal_bias)
FUNCTION SAMPLE():
return SAMPLE_FROM_VORONOI_EDGES(free_edges, obstacles_union, goal_bias, goal)
TREE_A_NODES ← [start]
TREE_A_PARENTS ← map()
TREE_A_PARENTS[0] ← NIL
TREE_B_NODES ← [goal]
TREE_B_PARENTS ← map()
TREE_B_PARENTS[0] ← NIL
FUNCTION EXTEND(tree_nodes, tree_parents, sample_pt):
idx_nearest ← NEAREST_NODE(tree_nodes, sample_pt)
nearest_pt ← tree_nodes[idx_nearest]
new_pt ← STEER(nearest_pt, sample_pt, step_size)
IF EDGE_IN_COLLISION(nearest_pt, new_pt, obstacles_union) = FALSE THEN
APPEND new_pt TO tree_nodes
new_idx ← LENGTH(tree_nodes) - 1
tree_parents[new_idx] ← idx_nearest
RETURN (new_pt, new_idx, TRUE)
ELSE
RETURN (NULL, NULL, FALSE)
END FUNCTION
FUNCTION CONNECT(tree_nodes, tree_parents, target_pt):
LOOP:
idx_nearest ← NEAREST_NODE(tree_nodes, target_pt)
nearest_pt ← tree_nodes[idx_nearest]
dist ← DISTANCE(nearest_pt, target_pt)
IF dist < step_size THEN
new_pt ← target_pt
ELSE
new_pt ← STEER(nearest_pt, target_pt, step_size)
END IF
IF EDGE_IN_COLLISION(nearest_pt, new_pt, obstacles_union) = FALSE THEN
APPEND new_pt TO tree_nodes
new_idx ← LENGTH(tree_nodes) - 1
tree_parents[new_idx] ← idx_nearest
IF DISTANCE(new_pt, target_pt) < 1e-5 THEN
RETURN (TRUE, new_idx)
END IF
IF dist < step_size THEN
RETURN (TRUE, new_idx)
END IF
ELSE
RETURN (FALSE, idx_nearest)
END IF
END LOOP
END FUNCTION
FOR i FROM 0 TO max_iters - 1 DO
sample_pt ← SAMPLE()
IF sample_pt = NULL THEN
CONTINUE
END IF
(new_ptA, idxA, successA) ← EXTEND(TREE_A_NODES, TREE_A_PARENTS, sample_pt)
IF successA = FALSE THEN
CONTINUE
END IF (successB, new_idxB) ← CONNECT(TREE_B_NODES, TREE_B_PARENTS, new_ptA)
IF successB = TRUE THEN
RETURN (TREE_A_NODES, TREE_A_PARENTS,
TREE_B_NODES, TREE_B_PARENTS, idxA, new_idxB)
END IF
TEMP_NODES ← TREE_A_NODES
TEMP_PARENTS ← TREE_A_PARENTS
TREE_A_NODES ← TREE_B_NODES
TREE_A_PARENTS ← TREE_B_PARENTS
TREE_B_NODES ← TEMP_NODES
TREE_B_PARENTS ← TEMP_PARENTS
END FOR
RETURN NULL
END ALGORITHM
```

4. RESULTS

Our proposed methods were tested using single robot and static obstacles, where environment is a 2D plane 500 × 500 grid map with start point and goal point. The algorithms executed with python 3.12 program language. To evaluate the performance of the proposed methods and to know which of these methods is the best, these methods were compared with

each other. Two environments (map 1, map2) are used to make the comparison of the methods. Every method was run ten times on the two environments. The results of all methods are the average obtained over ten runs on each environment. The result represents the value for criteria (path length, path smoothness, execution time, min clearance) of each method. The sum angle of path represents the smoothness objective.For two consecutive segments $p1\ p2$ and $p2\ p3$, $pi = (xi\ , yi)$, $i = 1 - 3$, the deviation angle is calculated with Eq. (9).

$$Smoothness = \sum_{i=0}^{n-2} Angle(pi, pi+1, p+2) \quad (9)$$

In map 1, standard RRT has the highest smoothness average of (43.7872), path length of (81.5) and execution time of (0.4693). While it improves when the standard RRT was combined with VD to be (40.1547), path length (79.9) and execution time (0.1150). Bi-RRT algorithm recorded highest path length average (83.0000) with less min clearance (0.1093), and (0.1369) execution time. While Bi-RRT-VD algorithm was the best in path length (71.5), execution time (0.0234) and min clearance (0.1990). Four methods were tested in a Narrow Environment with Specific Obstacles as shown in Figure 4.

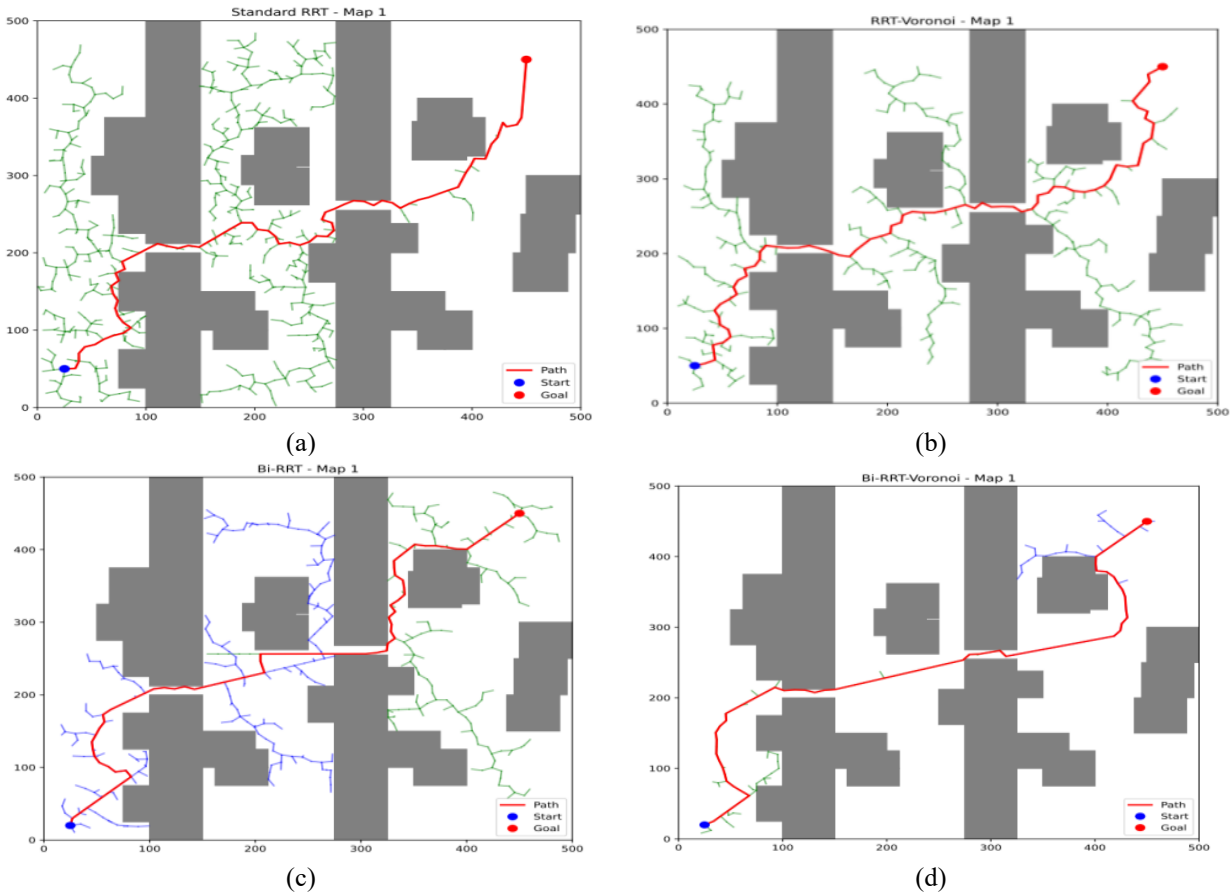

Figure 4. Map 1: (a) Standard RRT, (b) Voronoi-RRT, (c) Bi-RRT, (d) Voronoi-Bi-RRT
Note: RRT: rapidly exploring random trees; Bi-RRT: Bidirectional Rapidly-exploring algorithm

The results of our methods (RRT, RRT-VD, Bi-RRT, Bi-RRT-VD) were compared with algorithms in ref. [19] (RRT*, APSD-RRT) and demonstrated in Table 1.

In map 2, the standard RRT recorded high smoothness (42.4487) and it was enhanced when it was combined with VD. While the execution time and the path length are (0.0807) and (74), respectively. Moreover, Bi-RRT with Voronoi was the best in terms of execution time and path length among the other algorithms, where the execution time is (0.0054) and the path length is (70). It also shows a significant improvement in min clearance, which (0.1234) using Bi-RRT and become (0.1901) using Bi-RRT with Voronoi. The smoothness was better in Bi-RRT than in Bi-RRT with Voronoi. Four Methods were tested in a narrow environment with specific obstacles is shown in Figure 5.

The results of all methods on the second environment are listed in Table 2, with comparison with methods (CSA-RRT, NC-RRT) in this study [13].

Table 1. The comparison among the four algorithms in map 1 with reference to the study [19]

Algorithm	Average Time	Average Path Planning	SD Path Planning	Average Smoothness	Average Min Clearance
RRT	0.4693	81.5	3.107	43.7872	0.1915
RRT-VD	0.1150	79.9	2.885	40.1547	0.1269
Bi-RRT	0.1369	83	4.395	29.5600	0.1093
Bi-RRT-VD	0.0231	71.5	1.506	28.6133	0.1990

RRT*	5.51	693.199
APSD-RRT	0.22	674.35

Note: RRT: rapidly exploring random trees; Bi-RRT: Bidirectional Rapidly-exploring algorithm; VD: Voronoi diagram

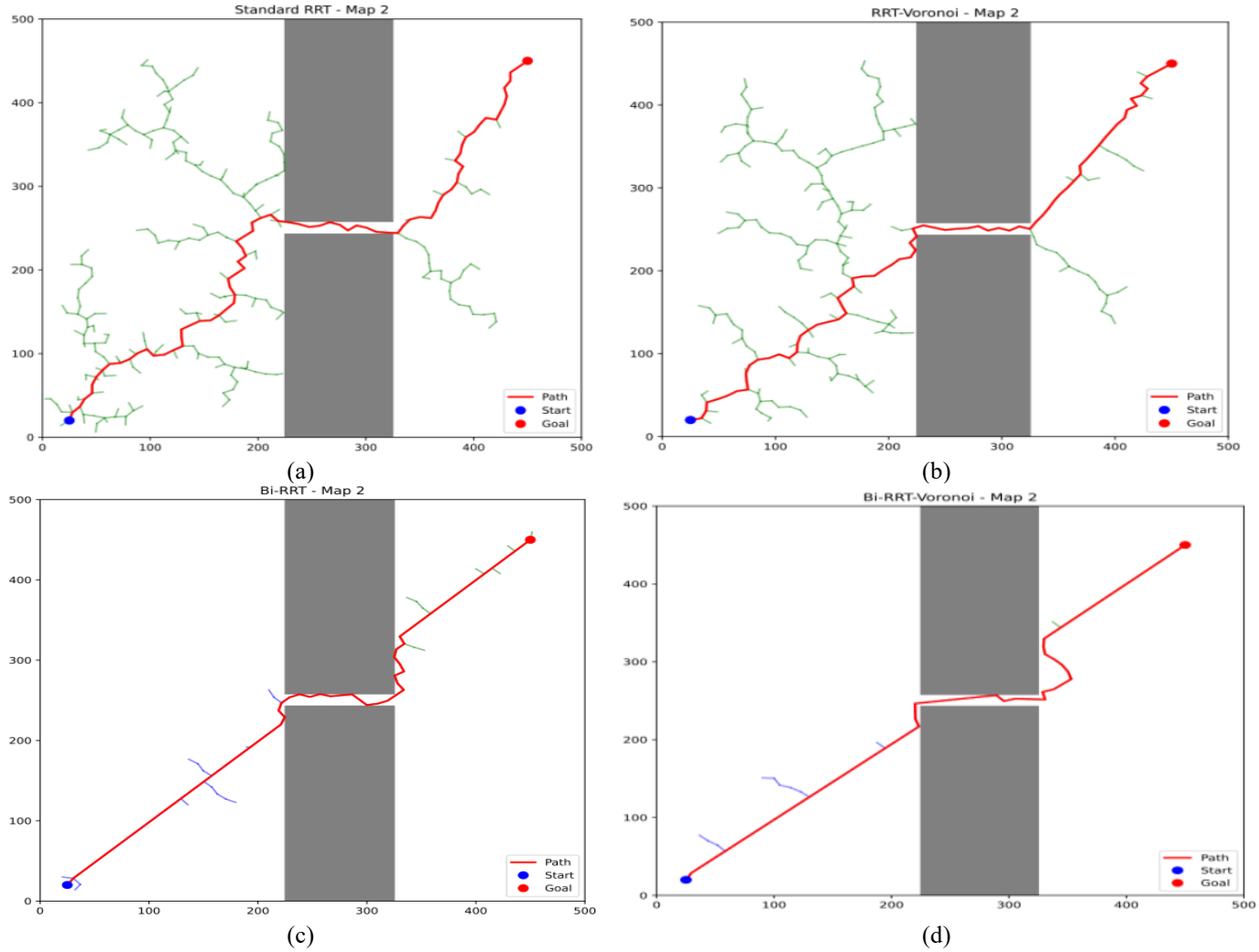

Figure 5. Map 2: (a) Standard RRT, (b) RRT-Voronoi, (c) Bi-RRT, (d) Bi-RRT-Voronoi

Note: RRT: rapidly exploring random trees; Bi-RRT: Bidirectional Rapidly-exploring algorithm

Table 2. The comparison among the four algorithms in map 2 with reference to the study [15]

Algorithm	Average Time	Average Path Planning	SD Path Planning	Average Smoothness	Average Min Clearance
RRT	0.1871	75	2.108	42.4487	0.2157
RRT-VD	0.0807	74	2.403	36.7925	0.1962
Bi-RRT	0.0253	74.6	4.818	16.7202	0.1234
Bi-RRT-VD	0.0054	70	1.414	17.9357	0.1901
CSA-RRT	0.106	868.609			
NC-RRT	0.117	942.615			

Note: RRT: rapidly exploring random trees; Bi-RRT: Bidirectional Rapidly-exploring algorithm; VD: Voronoi diagram

5. CONCLUSIONS

In this paper, we proposed a combination of two algorithms: RRT and Bi-RRT with Voronoi. This combination is improved the motion planning in cluttered and narrow environments. The use of Voronoi sampling makes it more likely that sample sites will be found in big open areas and tight hallways. This makes path planning faster and more accurate. Bi-RRT with Voronoi combines two strong path planning strategies: Bi-RRT, which creates two trees, one from the start point and the other from the target point. These trees grow toward each other and try to meet in the middle to form a complete path. VD which is a network of lines that represent the equidistant paths between obstacles. It helps in selecting samples from open and safe areas, far from obstacles, and improving the planner's efficiency and safety. The result showed that guided by Voronoi edges, the trees explore more efficiently and meet faster, improving convergence time especially in environments with narrow passages or many obstacles. This combination strategy increases the likelihood of finding a valid path in terms of safe, short, smooth, and less execution time. The current work is limited by its reliance on

a static, two-dimensional environment, which restricts its applicability to more complex, dynamic or three-dimensional settings. The performance of the VD may also degrade in extremely narrow passages, and the method lacks real-time replanning capabilities, with computational cost increasing in densely obstructed maps. Future extensions may focus on enabling operation in dynamic and 3D environments, incorporating adaptive Voronoi updates during execution, adding real-time replanning mechanisms, improving path quality, and validating the approach on a physical mobile robot.

REFERENCES

[1] Abed, B.M., Jasim, W.M. (2023). Hybrid approach for multi-objective optimization path planning with moving target. Indonesian Journal Electrical Engineering and Computer Science, 29(1): 348-357. https://doi.org/10.11591/ijeecs.v29.i1.pp348-357

[2] Chi, W., Ding, Z., Wang, J., Chen, G., Sun, L. (2021). A generalized Voronoi diagram-based efficient heuristic path planning method for RRTs in mobile robots. IEEE Transactions on Industrial Electronics, 69(5): 4926-4937. https://doi.org/10.1109/TIE.2021.3078390

[3] Amar, L.B., Jasim, W.M. (2021). Hybrid metaheuristic approach for robot path planning in dynamic environment. Bulletin of Electrical Engineering and Informatics, 10(4): 2152-2162. https://doi.org/10.11591/eei.v10i4.2836

[4] Adzhar, N., Yusof, Y., Ahmad, M.A. (2020). A review on autonomous mobile robot path planning algorithms. Advances in Science, Technology and Engineering Systems Journal, 5(3): 236-240. https://doi.org/10.25046/aj050330

[5] Patle, B.K., Pandey, A., Parhi, D.R.K., Jagadeesh, A.J.D.T. (2019). A review: On path planning strategies for navigation of mobile robot. Defence Technology, 15(4): 582-606. https://doi.org/10.1016/j.dt.2019.04.011

[6] Abed, B.M., Jasim, W.M. (2022). Multi-objective optimization path planning with moving target. International Journal of Artificial Intelligence, 11(3): 1184-1196. https://doi.org/10.11591/ijai.v11.i3.pp1184-1196

[7] Liu, L., Wang, X., Yang, X., Liu, H., Li, J., Wang, P. (2023). Path planning techniques for mobile robots: Review and prospect. Expert Systems with Applications, 227: 120254. https://doi.org/10.1016/j.eswa.2023.120254

[8] Jiang, L., Wang, S., Meng, J., Zhang, X., Li, G., Xie, Y. (2019). A fast path planning method for mobile robot based on Voronoi diagram and improved d* algorithm. In 2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Hong Kong, China, pp. 784-789. https://doi.org/10.1109/AIM.2019.8868362

[9] Ayawli, B.B.K., Appiah, A.Y., Nti, I.K., Kyeremeh, F., Ayawli, E.I. (2021). Path planning for mobile robots using Morphological Dilation Voronoi diagram Roadmap algorithm. Scientific African, 12: e00745. https://doi.org/10.1016/j.sciaf.2021.e00745

[10] Yang, L., Iyer, G., Lou, B., Turlapati, S.H., Lv, C., Campolo, D. (2025). Path planning in complex environments with superquadrics and Voronoi-based orientation. In 2025 IEEE International Conference on Robotics and Biomimetics (ROBIO), Chengdu, China, pp. 1590-1597. https://doi.org/10.1109/ROBIO66223.2025.11376314

[11] Hao, K., Yang, Y., Li, Z., Liu, Y., Zhao, X. (2023). CERRT: A mobile robot path planning algorithm based on RRT in complex environments. Applied Sciences, 13(17): 9666. https://doi.org/10.3390/app13179666

[12] Li, N., Han, S.I. (2025). Adaptive Bi-directional RRT algorithm for three-dimensional path planning of unmanned aerial vehicles in complex environments. IEEE Access, 13: 23748-23767. https://doi.org/10.1109/ACCESS.2025.3537697

[13] Tuncer, A. (2023). Path planning of autonomous mobile robots based on Voronoi diagram and ant colony optimization. Journal of Innovative Engineering and Natural Science, 4(1): 138-146. https://doi.org/10.61112/jiens.1365282

[14] Chai, Q., Wang, Y. (2022). RJ-RRT: Improved RRT for path planning in narrow passages. Applied Sciences, 12(23): 12033. https://doi.org/10.3390/app122312033

[15] Wang, X., Luo, X., Han, B., Chen, Y., Liang, G., Zheng, K. (2020). Collision-free path planning method for robots based on an improved rapidly-exploring random tree algorithm. Applied Sciences, 10(4): 1381. https://doi.org/10.3390/app10041381

[16] Zhu, W., Qiu, G. (2025). Path planning of intelligent mobile robots with an improved RRT algorithm. Applied Sciences, 15(6): 3370. https://doi.org/10.3390/app15063370

[17] Cao, K., Cheng, Q., Gao, S., Chen, Y., Chen, C. (2019). Improved PRM for path planning in narrow passages. In 2019 IEEE International Conference on Mechatronics and Automation (ICMA), Tianjin, China, pp. 45-50. https://doi.org/10.1109/ICMA.2019.8816425

[18] Uwacu, D., Yammanuru, A., Nallamotu, K., Chalasani, V., Morales, M., Amato, N.M. (2025). HAS-RRT: RRT-based motion planning using topological guidance. IEEE Robotics and Automation Letters, 10(6): 6223-6230. https://doi.org/10.1109/LRA.2025.3560878

[19] Li, X., Tong, Y. (2023). Path planning of a mobile robot based on the improved RRT algorithm. Applied Sciences, 14(1): 25. https://doi.org/10.3390/app14010025

[20] Cheng, A., Saxena, D.M., Likhachev, M. (2019). Bidirectional heuristic search for motion planning with an extend operator. In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, pp. 7425-7430. https://doi.org/10.1109/IROS40897.2019.8967796

[21] Ayawli, B.B.K., Dawson, J.K., Badu, E., Ayawli, I.E.B., Lamusah, D. (2025). Comparative analysis of popular mobile robot roadmap path-planning methods. Robotica, 43(4): 1548-1571. https://doi.org/10.1017/S0263574725000281