IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Bi-directional Adaptive Probabilistic Method with a Triangular Segmented Interpolation for Robot Path Planning in Complex Dynamic-Environments

SUHAIB AL-ANSARRY[1], SALAH AL-DARRAJI[1],ASMAA SHAREEF[1],DHAFER G. HONI[1,2], and FRANCESCA FALLUCCHI[3]

[1]Department of Computer Science, College of Education for Pure Sciences, University of Basrah, Basrah, Iraq
[2]University of Warith Al-Anbiyaa, Karbalaa, Iraq
[3] Department of Engineering Science, Guglielmo Marconi University, 00193 Roma, Italy

Corresponding author: Salah Al-Darraji (e-mail: aldarraji@uobasrah.edu.iq).

ABSTRACT Path planning is a fundamental aspect of mobile robots and autonomous systems. Methods of path planning are used in robotics to create a path for a robot or autonomous system to follow from a starting position to a goal one while avoiding obstacles and satisfying any additional conditions. There are many different methods to plan the path, including probabilistic methods, heuristics-based approaches, and optimization-based methods. In this paper, we propose a new path planning method called Dynamic Adaptive RRT-connect with a Triangular Segmented Interpolation. Our method improves the traditional RRT algorithms by using an Adaptive-RRT approach, where a random node is chosen as a new node to increase tree exploration. Then, we use a Bi-directional scheme to further enhance the convergence time and cost. Additionally, our method employs Triangular Segmented Interpolation (TSI) method to improve path length and smoothness. Finally, we operate this method within a dynamic environment depending on the Dynamic Window Approach (DWA). Experiments on a variety of environments have shown that our proposed method achieves better than the RRT and RRT-connect algorithms individually in terms of computation time (reduced by 90-80%), cost (reduced by 82-63%), and path length (shorten by 17-12%) besides the ability to avoid dynamic obstacles efficiently.

INDEX TERMS Autonomous System, Dynamic Obstacles, Interpolation, Probabilistic Methods, Robot Path Planning.

I. INTRODUCTION

UNMANNED Aerial Vehicles (UAVs) are increasingly being used in a variety of tasks due to their small size, low cost, ease of use, and ability to operate in hazardous environments. UAV path planning involves finding a safe, collision-free path from a starting position to a goal position while considering any potential threats and performance constraints of the UAV, and minimizing the flying time as much as possible [1]. However, traditional algorithms for UAV path planning may not always be suitable for complex environments and may not be able to meet the specific needs of the mission.

Traditional classical methods such as mathematical induction, dynamic induction, and optimal control methods may not be suitable for describing system dynamics and uncertain environments in UAV path planning due to their computational intensity and tendency to get stuck in local optima [2]. For example, the A* algorithm [3] relies heavily on the cost function, which must properly weigh various constraints. If the cost function is not designed properly, the algorithm's search space can grow exponentially. While genetic algorithms are efficient and globally robust optimization algorithms, they tend to converge slowly after reaching an optimal solution and may not be suitable for real-time trajectory planning, or the multi-threaded versions of probabilistic methods [4] which are needed for super equipment.

Random sampling algorithms, such as the probabilistic roadmap (PRM) algorithm [5], PRM* [6], and the Rapidly Exploring Random Tree (RRT) algorithm, are effective in solving UAV trajectory planning problems. The PRM algorithm generates a roadmap by setting random points in space and linking them and then uses heuristic knowledge to guide the search for the best or second-best flight path from the initial state to the target state. The RRT algorithm which is

first introduced by Steven M. LaValle in 1998 [7], relies on a randomly-selected strategy and collision-checking method to obtain a path in the mission environment and uses a tree structure to describe the path, which makes the search more efficient. However, the RRT algorithm may get stuck in a local minimum area when the distribution of threat sources in the mission environment is unclear, which is fixed by the method called RRT-connect [8], and the search may become less efficient as it increases the number of search failures. RRT* and RRT*-connect [9] [10] improved variants of traditional RRT and RRT-connect respectively, which introduced an optimized path against increased computational time and cost.

In [11] combines the artificial attractive field (AAF) approach with the (RRT) algorithm, resulting in an improved AAF-RRT method with better search efficiency and collision avoidance ability for global and local planning. Another method [12] called improved bidirectional RRT* introduces to addresses the problems of a high degree of randomness, low search efficiency, and many inflection points in traditional bidirectional RRT*. The improvements include constraining the expansion direction of the random tree by an improved artificial potential field method, biasing the random tree sampling towards the target point, and optimizing the planned path by extracting key points. An adaptation of the standard RRT algorithm for real-time path planning in congested environments has been proposed in [13], which involves adjusting the step size based on the distance from the root node, resulting in more precise short-term plans and faster generation of coarse long-term plans. In the adaptive RRT with dynamic step (DRRT) [14], the author addresses the issues of the RRT algorithm such as falling into local optimum areas and longer planning time. Additionally, the issue of dimensionality of high degree-of-freedom articulated robots handles also in an adaptive manner [15] through body selection which chooses necessary robot bodies and joints based on the complexity of path planning. Zhang et al. [16] use the traditional RRT algorithms with a fixed step length and bias probability which can lead to excessive nodes and poor performance, while in [17], they consider the relationship between environment complexity and the step size and bias probability in the RRT algorithm and adjusts the two parameters accordingly.

On the other hand, Zeng et al. [18] combine RRT and DWA algorithms to explore a sparse, relatively small-size point cloud local map, similarly in [19] authors present an innovative approach that minimizes planning time and solves sharp path problems. Moreover, Jia et al. [20] present a framework for dynamic path planning that avoids collisions with dynamic obstacles in a partially unknown environment. In addition, Dai et al. [21] propose a novel algorithm that uses a greedy approach to determine whether a new node can directly reach the target point, and Kang et al. [22] present a bidirectional interpolation method for post-processing in sampling-based robot path planning algorithms. Examples of methods that deal with the concept of dynamic path planning and path smoothness can be found in articles [23], [24]. These contributions have the potential to make autonomous robots more efficient and practical in real-world scenarios.

In this paper, we improve the performance of the RRT algorithm by introducing a directional adaptive variation. The Adaptive-RRT algorithm (A-RRT) is an improved version of the RRT algorithm that uses the random node directly as a new node to increase the speed of tree exploration. It has proven to be faster and more efficient in finding the solution compared to the original RRT algorithm. Then, we apply the concept of bi-directional search to propose the (A-RRT connect) which is improved the planning efficiency based on the combination of the adaptive two trees during the sampling process leading to decreasing the convergence time and the cost of generating nodes. Next, Triangular Segmented Interpolation (TSI) is proposed as a new manner that helps in producing a short and smooth path. Finally, we employ the Dynamic Window Approach (DWA) [25] to enable the robot to safely reach its target point and avoid temporary obstacles (dynamic obstacles) in the global path.

II. METHODOLOGY

Dynamic path planning is a challenging problem in robotics and autonomous systems, as it requires the ability to find an optimal path in real-time while considering changes in the environment. In this work, we present an improved method for dynamic path planning called (Dynamic A-RRT-connect TSI) algorithm. The proposed algorithm is a combination of the A-RRT-connect with TSI, besides the Dynamic Window Approach. The algorithm operates in a bi-directional manner, starting from both the start position and the goal position, to find a collision-free path in a dynamic environment. The proposed method (Dynamic A-RRT-connect TSI) algorithm can be broken down into the following steps:

1) Initialize the algorithm by setting the start and goal positions for the robot, as well as any other algorithm parameters such as the step size, number of iterations, and goal distance threshold.
2) Begin growing the tree from the start position. The tree is initially empty, so the first node added is the start position.
3) Sample a random point in the environment. If the point is in free space, proceed to the next step. If the point is in a collision, discard it and sample another point.
4) Using the Adaptive-RRT approach, select the closest node in the tree to the random point as the parent node, and then extend the tree towards the random point.
5) Simultaneously grow another tree from the goal position, which will allow the algorithm to quickly converge on the path between the start and goal positions.
6) Check if the distance between the new node and the goal is below a predefined threshold. If it is, a path to the goal has been found and the algorithm terminates. If not, add the new node to the tree and continue growing the trees.
7) Once the two trees meet, the algorithm terminates and the path between the start and goal positions is returned.

8) Use the Triangular Segmented Interpolation (TSI) method to enhance the resulting path.
9) Employ the Dynamic Window Approach (DWA) to calculate the attractive and repulsive forces acting on the new node. These forces will be used to guide the robot toward the goal while avoiding obstacles in real-time.
10) When an obstacle appears suddenly, step-8 is repeated to avoid the dynamic obstacle and find the next valid point on the resulting path to guide the robot around the obstacle safely.

A. ADAPTIVE-RRT (A-RRT)

Rapidly-Exploring Random Tree (RRT) is a popular path planning algorithm used in robotics and control systems. The algorithm starts with an empty tree and adds the start node to it. The algorithm then enters a loop that runs for a maximum number of iterations. In each iteration, as shown in Fig. 1, a random node is sampled from the environment and the nearest node in the tree to the random node is found. The tree is then extended towards the random node by creating a new node, with the step size (ϵ) determining the size of the extension. If the new node is in free space, it is added to the tree, and the distance between the new node and the goal is checked. If the distance between the new node and the goal is below a predefined threshold, a path to the goal has been found, and the algorithm returns the path. If the maximum number of iterations is reached and no path is found, the algorithm returns "No path found."

The RRT algorithm uses the following mathematical formulas:

1) **Distance calculation**: The distance between two nodes in the tree is calculated using a metric, such as Euclidean distance. The equation for Euclidean distance between points (x_1, y_1) and (x_2, y_2) is:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

2) **Nearest node calculation**: The nearest node in the tree to the random node is found using the distance calculation. Given a set of nodes in the tree and a random node, the nearest node can be found using the following formula:

$$nearestNode = min(d(tree[i], randNode)) \tag{2}$$

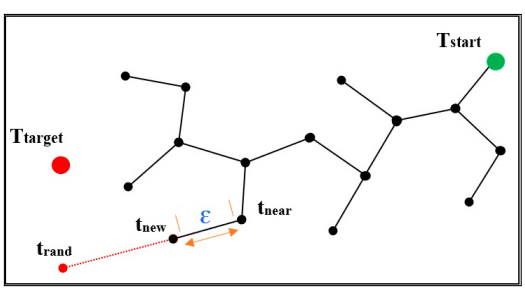


FIGURE 1. The RRT Construct [26].

for all i in $[1, n]$, where n is the number of nodes in the tree.

3) **New node calculation**: The new node is calculated by extending the tree towards the random node by step size.

The first level of the proposed work is the Adaptive-RRT (A-RRT), which is similar to the traditional RRT algorithm, with a modified step of sampling a random child node for each new node added to the tree, which leads to improve the tree expansion, besides reducing the cost and convergence time efficiently.

The proposed algorithm works iteratively to generate random nodes within the search space and connect them to a tree. The tree grows by connecting newly generated nodes to their nearest nodes within the tree while ensuring that they avoid obstacles. The algorithm continues to generate random nodes and connect them to the tree until either the goal is reached or the maximum number of iterations is exceeded. During each iteration of the algorithm, a random node is generated within the search space. If this node falls within an obstacle, the node is omitted, and a new random node is generated. If the node is valid, the nearest node in the tree to the random node is found. Next, the line between the nearest node and the random node does is checked to ensure it does not intersect with any obstacle. If the line intersects with an obstacle, the algorithm calculates the midpoint between the nearest node and the random node and moves the random node towards the midpoint to avoid the obstacle. A new node is created from the updated random node, and it is added to the tree with the nearest node as its parent node if it is a valid node, otherwise, the process of finding another midpoint continues until finding an obstacle-free line or the length of the line is too small. The algorithm continues to iterate, generating random nodes and connecting them to the tree while avoiding obstacles, until either the goal is reached, or the maximum number of iterations is exceeded. If the goal is reachable, the goal node is added to the tree, and the function returns the tree. If the maximum number of iterations is reached without finding a feasible path, the function returns None, indicating that no path was found. Algorithm 1 explains the whole process. Fig. 2 show the rapid tree exploration with low node expansion, besides the resulting path in blue color.

B. ADAPTIVE-RRT-CONNECT (A-RRT CONNECT)

The scheme of bi-directional search is a popular used for finding a feasible path between two points in a configuration space. Unlike traditional path planning algorithms that start from the start node and expand the search space toward the goal node, the bi-directional algorithm, as shown in Fig. 3, uses two search trees, one starting from the start node and the other starting from the goal node, to search for a feasible path. The two trees then grow toward each other until they meet in the middle.

The bi-directional algorithm has several advantages over traditional search algorithms. Firstly, it reduces the search

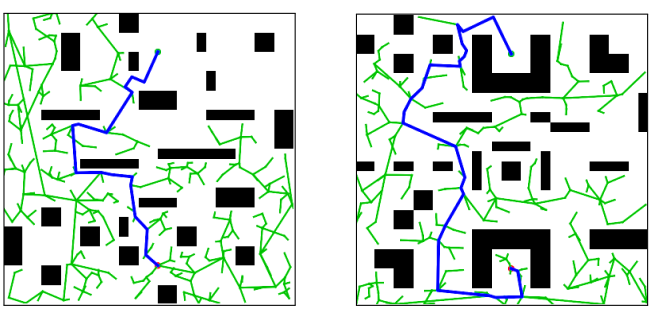
space, as the search is performed from both ends of the problem, leading to a faster search. Secondly, the algorithm reduces the probability of getting stuck in dead-end spaces, which often happens in traditional algorithms. Lastly, it increases the probability of finding an optimal path by allowing the algorithm to search for a path from both the start and goal nodes, rather than searching for a path only from the start node. One challenge with the bi-directional path planning scheme is that it requires finding a connection between the two search trees once they meet in the middle. This is typically done by connecting the nodes that are closest to each other, creating a path between the start and goal nodes. Despite this challenge, the bi-directional path planning scheme is widely used in robotics, computer vision, and other related fields. It is particularly useful in high-dimensional configuration spaces, where traditional algorithms may struggle to find feasible paths. Additionally, it is a powerful tool for finding optimal paths in complex search spaces, making it an essential part of many modern path planning algorithms.

In the second level of the proposed method, the proposed algorithm starts by initializing two trees, one for the start configuration and the other for the goal configuration. It then iterates for a maximum number of iterations provided as input or a feasible path is found. In each iteration, the algorithm samples a random configuration within the search space, checks if it is in an obstacle, and continues to the next iteration if it is. Then, finds the nearest node in both the start and goal trees to the random configuration, using a distance metric such as Euclidean distance. It then attempts to connect the nearest nodes in the two trees by checking if a straight line between them does not intersect any obstacles in the search space. If a connection is feasible, a new node is created and added to the start tree, and the same is done for the goal tree. If a newly added node in the start tree is close enough to the goal configuration (i.e., within the goal distance threshold), the algorithm attempts to connect it to the goal tree. If successful, the algorithm returns a path connecting the start and goal configurations. If the connection is not feasible, the algorithm creates a new configuration by taking the midpoint between the nearest node and the random configuration and moving towards the random configuration by a factor of 0.5. The

Algorithm 1: Adaptive-RRT (A-RRT)

Input : **start**, **goal**, **search_space**, **obstacles**, **max_iterations**, **goal_distance_threshold**, **line_length_threshold**
Output: **tree** = [T]

```
T = InitializeTree (start)
for i = 1 to max_iterations do
    random_node= SampleRandomNode (search_space)
    if not IsNodeInObstacle (random_node, obstacles) then
        nearest_node = (random_node, T)
        line = LineSegment (nearest_node, random_node)
        while IsLineIntersectObstacle (line, obstacles) and Length (line) ≥ line_length_threshold do
            random_node = CalculateMidpoint(nearest_node, random_node)
            line = LineSegment (nearest_node, random_node)
        end while
        if Length (line) ≥ line_length_threshold then
            AddNodeToTree (T, nearest_node, random_node)
            if IsGoalReachable (random_node, goal, obstacles, goal_distance_threshold) then
                AddGoalNodeToTree (random_node, goal)
                Return T
            end if
        end if
    end if
end for
```



(a) (b)

FIGURE 2. **The Adaptive-RRT Tree (a) MAP-1 (b) MAP-2.**

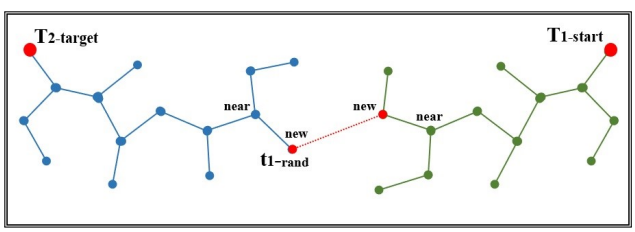
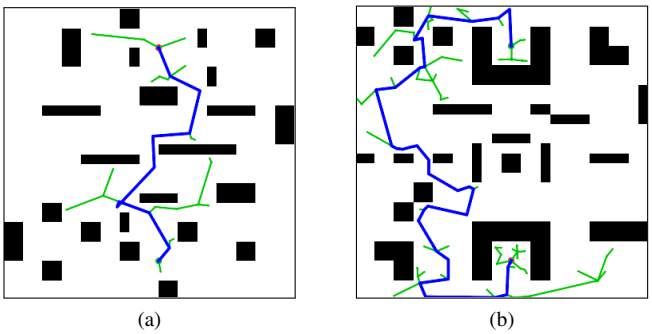


FIGURE 3. **The RRT-connect Construct.**



(a) (b)

FIGURE 4. The A-RRT connect Tree (a) MAP-1 (b) MAP-2.

algorithm then repeats the process of finding the nearest node and attempting to connect to the trees. If the algorithm reaches the maximum number of iterations without finding a path, it returns None, indicating that no path was found. Fig. 4 shows the tree expansion and resulting path in a blue color of the proposed A-RRT connect clearly. Algorithm 2 explains the proposed method (A-RRT connect).

C. A-RRT CONNECT BASED ON TRIANGULAR SEGMENTED-INTERPOLATION (A-RRT CONNECT TSI)

Triangular Midpoint Interpolation (TMI) is a method used to enhance path length and smoothness in geometric shapes. It is particularly useful when working with polygons, which are typically made up of straight lines and sharp angles. The basic idea behind triangular midpoint interpolation, as shown in Fig. 5, is to add a new vertex at the midpoint of each existing line segment in the polygon. This creates a new set of triangles that can be used to interpolate new points along the polygon's path. By adding these new vertices, the overall path length is increased and the polygon becomes smoother.

The algorithm for triangular midpoint interpolation can be broken down into several steps:

1) Create a list of all line segments in the polygon.
2) For each line segment, find its midpoint and add a new vertex at that location.
3) Create a new set of triangles using the existing vertices and the newly added vertices.
4) For each new triangle, calculate the centroid (the average position of its vertices) and add that point to the list of new vertices.
5) Repeat steps 3-4 until the desired level of smoothness is achieved.

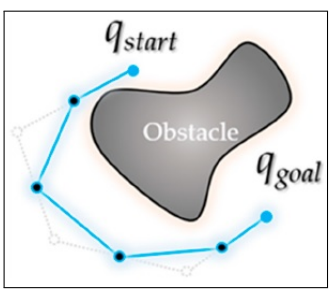


FIGURE 5. Single-Phase Triangular Midpoint Interpolation Method [22].

Algorithm 2: A-RRT Connect

```
Input  : start, goal, search_space, obstacles,
         max_iterations, goal_distance_threshold,
         line_length_threshold
Output: tree = [T]

start_tree = InitializeTree (start)
goal_tree = InitializeTree (goal)
for i = 1 to max_iterations do
    random_node1 = SampleRandomNode
     (search_space)
    random_node2= SampleRandomNode
     (search_space)
    if not IsNodeInObstacle (random_node,
     obstacles) then
        nearest_node1 = (random_node1, T)
        nearest_node2 = (random_node2, T)
        line1 = LineSeg (nearest_node1,
         random_node1)
        line2 = LineSeg (nearest_node2,
         random_node2)
        while IsLineIntersectObstacle
         (line1, obstacles) and Length (line1) ≥
         line_length_threshold do
            random_node1 =
             CalculateMidpoint(nearest_node1,
             random_node1)
            line1 = LineSeg (nearest_node1,
             random_node1)
        end while
        if Length (line1)>=line_length_threshold
         then
            AddNodeToTree (start_tree,
             nearest_node1, random_node1)
            if IsGoalReachable (random_node1,
             goal, obstacles,
             goal_distance_threshold) then
                AddGoalNodeToTree
                 (random_node1, goal)
                Return T
            end if
        end if
    end if
end for
```

Once the new vertices have been added and the new triangles have been created, the path can be interpolated using a variety of techniques. One common approach is to use linear interpolation between adjacent vertices, which produces a smooth path that follows the polygon's original shape. In this

work, as shown in Fig. 6, we modify the method of TMI to Triangular Segment-endpoint Interpolation (TSI) which is helped to enhance the path length besides the time and cost. In algorithm 3, we start with a set of points that represent the vertices of the path want to plan. Then divide the path into segments by connecting every two consecutive vertices. This will create a set of line segments that form the path, for each set of three consecutive vertices, do the following:

- Divide the first line segment into smaller segments using segment endpoint interpolation.
- Divide the second line segment into smaller segments using segment endpoint interpolation.
- Connect the new vertices with straight lines, creating a new set of line segments that form the smoothed path.

Next, for each pair of adjacent segments in the first and second lines, do the following:

- Create a set of parallel lines for the pair of segments.
- For each pair of parallel lines, find the valid connection between the endpoints that do not intersect with any obstacles.
- Connect the endpoints of the segments with the valid connection, forming a path that avoids obstacles.

Finally, connect the endpoints of all the connected segments to form a complete path that avoids obstacles and solves the issue of a sharp path. Algorithm 3 describes a method for planning a path between a set of vertices while avoiding obstacles.

D. DYNAMIC A-RRT-CONNECT TSI (DA-RRT-CONNECT TSI)

In the proposed work, Dynamic Window Approach (DWA) is used as a local planner to find the path between two points while avoiding the dynamic obstacles as the robot moves in real-time depending on the Lidar scan, which makes it well-suited for applications where dynamic obstacles are expected to appear frequently and unpredictably. The DWA Planner algorithm generates a set of possible trajectories based on the dynamic window concept, which defines a range of achievable velocities and angular velocities for the robot. The algorithm evaluates each trajectory by calculating a cost function that considers several factors such as distance to the destination, proximity to obstacles, and smoothness of the trajectory. After evaluating all possible trajectories, the DWA Planner algorithm selects the optimal path for the robot to follow with the lowest cost. Overall, the DWA Planner algorithm provides an effective and efficient approach to path planning for mobile robots and autonomous vehicles, enabling them to navigate through complex and dynamic environments.

In this research, we modify the planner as shown in Fig. 7 to start the path-finding process from the current stop position and update the list of obstacles in the environment using the UpdateObstacles algorithm before each iteration. This will allow the algorithm to adapt to changes in the environment and find a path around dynamic obstacles.

The function of dynamic obstacle avoidance is implemented in the following part of the algorithm:

Algorithm 3: Triangular Segment Interpolated (TSI)

```
Input  : initial_path, seg_size
Output: smoothed_path
path = GetSegPath (initial_path)
changed = True
while changed do
    changed = False
    smoothed_path = []
    i = 0
    p1 = path[i]
    smoothed_path.append(p1)
    while i < Len (path)-2 do
        p2 = path[i+1]
        p3 = path[i+2]
        newline = LineSeg (p1, p3)
        if not IntersectObs (newline) then
            p1 = p3
            i = i + 2
            smoothed_path.append(p3)
            changed = True
        else
            line1 = LineSeg (p1,p2)
            line2 = LineSeg (p2,p3)
            segments1 = Int (Len (line1) / seg_size)
            segments2 = Int (Len (line2) / seg_size)
            if segments1==0 or segments2==0 then
                smoothed_path.append(p2)
                p1 = p2
                i = i + 1
            else
                newp1 = p1 + seg_size
                newp3 = p3 – seg_size
                newline = LineSeg (newp1, newp3)
                while IntersectObs (newline) and
                 Dist (newp1,p2) ≥ seg_size and
                 Dist (newp3,p2) ≥ seg_size do
                    newp1 = p1 + seg_size
                    newp3 = p3 – seg_size
                    newline=LineSeg
                     (newp1,newp3)
                end while
                if Dist (newp1,p2) ≥ seg_size and
                 Dist (newp3,p2) ≥ seg_size then
                    smoothed_path.append(newp1)
                    smoothed_path.append(newp3)
                    p1 = newp3
                    i = i + 1
                    changed = True
                else
                    smoothed_path.append(p2)
                    p1 = p2
                    i = i + 1
                end if
            end if
        end if
    end while
end while
```

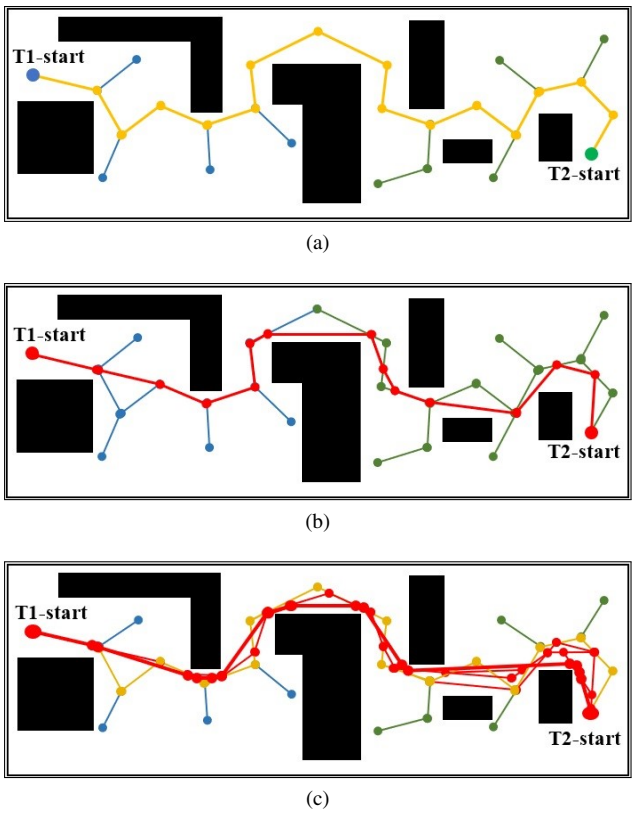


FIGURE 6. The Mechanism of TSI (a) Initial Path (b) Single-Phase Path (c) Multi-Phase Path.

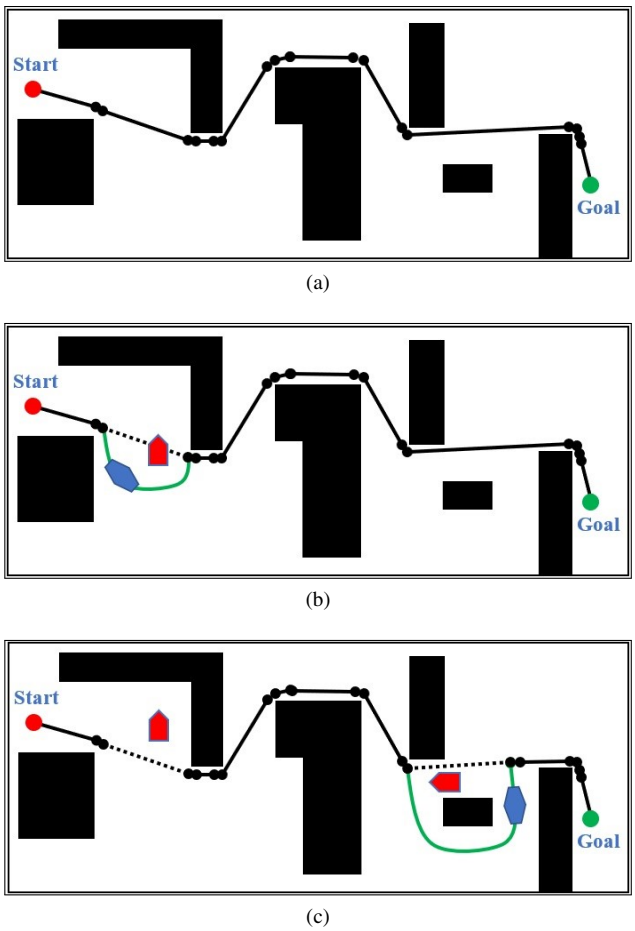


FIGURE 7. Dynamic Path Planning (a) Optimal Global Path (b-c) Local planning within Dynamic Map.

```
UpdateObstacles(obstacles)
{    ...
     DWALocalPlanner = CalculateOccupiedSpaces(newNode,
obstacles)
     ...
}
```

The UpdateObstacles function updates the list of obstacles in the environment, which is used by the CalculateOccupiedSpaces function to check the obstacle area for each new node. This function is used to steer the new nodes away from obstacles, allowing the algorithm to adapt to changes in the environment and find a path around obstacles. To re-plan the path from the current position to the next valid position, we modify the planner as follows:

1) Store the current position of the robot in a variable currentPosition.
2) After updating the list of obstacles, find the nearest node in the trees to the currentPosition using the FindNearestNode function.
3) Start the path-finding process from the nearest node in the trees (global path), instead of the start and goal nodes, by updating the start and goal variables.
4) Repeat the path-finding process until a valid path is found, or until a maximum number of iterations is reached.

The algorithm of (A-RRT-connect TSI) is used as a global planner to generate an optimized path between the start and goal points, then handled the dynamic obstacles using a local planner of DWA to re-plan the path from the current stop position to the next valid position by finding the nearest node on the global path from the current position, then extend the subnew path locally. Dynamic A-RRT-connect TSI introduces a high-level overview of how one could use the DWA planner to re-plan the path in response to dynamic obstacles. Here is a more detailed explanation of the steps basing on the proposed method through the flowchart in Fig. 8.

The proposed algorithm has several advantages compared to other path planning methods. It is able to handle dynamic environments with moving obstacles, and it can generate smooth, collision-free paths even in cluttered environments.

III. EXPERIMENTS AND RESULTS

The experiments were conducted on an Intel Core i7 laptop with 8GB of RAM and an NVIDIA GeForce GTX graphics card, running Ubuntu 20.04. The static experiments were conducted using three different complexity maps, each with varying levels of obstacles and challenges for the path planning algorithm. The maps for the static experiments were

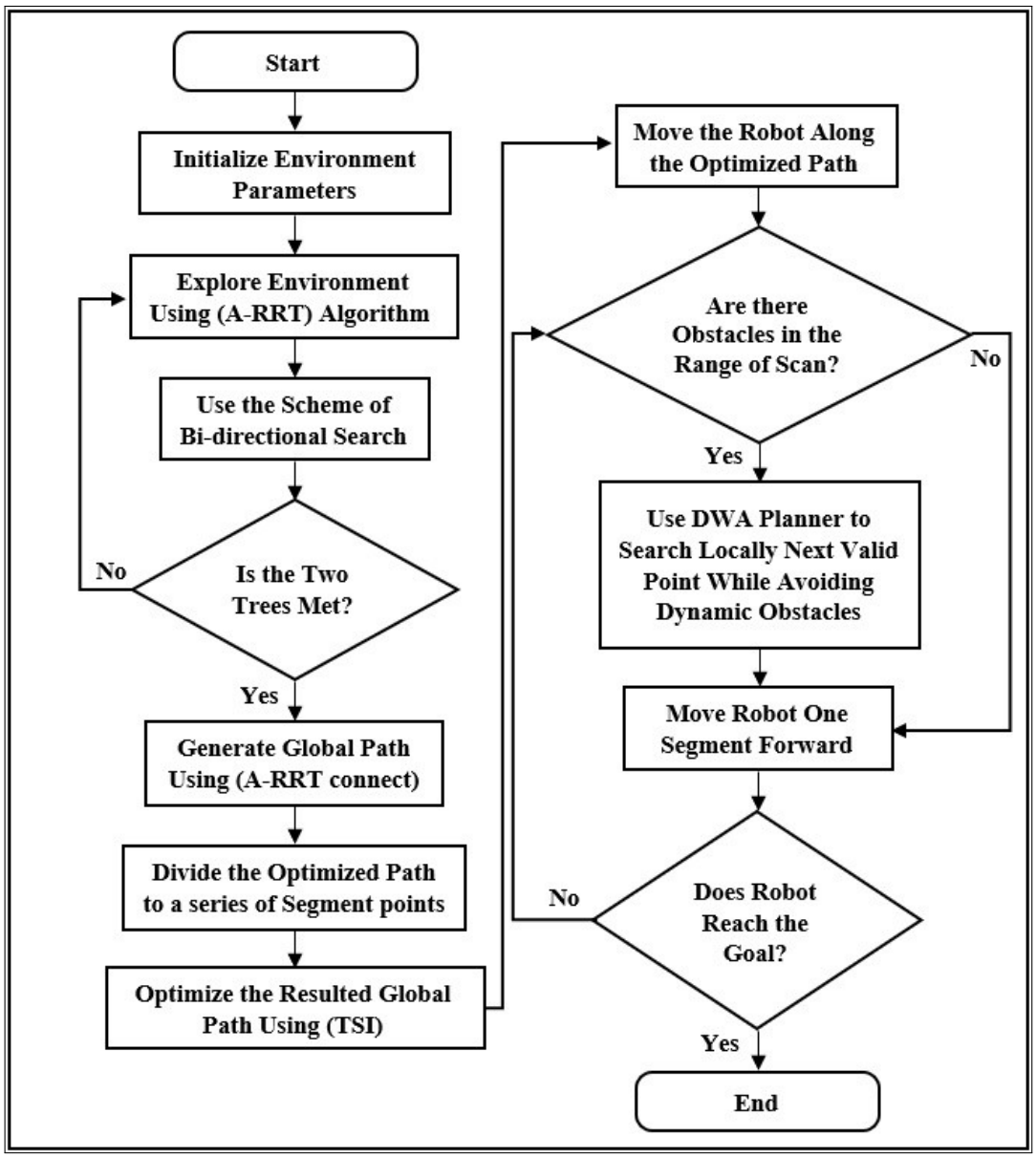


FIGURE 8. Flowchart of the Proposed Method steps.

loaded into the Rviz environment within the Robotic Operating System (ROS) framework, and the path planning algorithm was run for each map. The results were recorded and analyzed to determine the performance of the algorithm in static environments. For the dynamic experiment, the Gazebo simulation was used to introduce changes in the environment, such as moving obstacles, while the algorithm was running. The algorithm's ability to find a path while considering these dynamic changes was evaluated and compared to the results from the static experiments. The robot used in the experiment was equipped with sensors for obstacle detection and mapping.

A. EXPERIMENT-1 (STATIC ENVIRONMENTS)

The comparisons of this experiment have been implemented within a non-holonomic indoor environment of size (1500*1500 cm) that has local minima regions and a crowd of static obstacles for (1000 runs). The experiment among different methods was conducted based on the (time, cost, path length, and standard deviation) with a bias of (0.25). The start and goal points are acted by red circles, and the trees by blue and green lines. Finally, the orange line represents the initial path, while the red line is the optimum one.

- First comparison, as displayed in Fig. 9, shows the tree exploration of (RRT and A-RRT), and how the nodes spread extensively in the first map compared to the spread of the A-RRT method for the same map, which re-

TABLE 1. Experimental Results Comparison between (RRT and A-RRT)

Static Map	Method	Time (sec.)	Time Std. Dev.	Cost (node)
MAP-1	RRT	0.424	0.368	841
	RRT	0.042	0.041	148
MAP-2	A-RRT	3.790	1.874	4534
	A-RRT	0.662	0.389	2645

TABLE 2. Experimental Results Comparison between (RRT-Connect and A-RRT-connect)

Static Map	Method	Time (sec.)	Time Std. Dev.	Cost (node)
MAP-1	RRT-Connect	0.210	0.211	410
	A-RRT-Connect	0.042	0.041	148
MAP-2	RRT-Connect	1.233	0.509	3172
	A-RRT-Connect	0.580	0.472	823

duces the cost consumed in the process of searching the goal. The same approaches were followed in the second map, where RRT showed a wider spread compared to the first map, due to the high map complexity as well as the presence of local minimum regions where the goal point is there. The results listed in Table 1 illustrated that the A-RRT convergence time that is needed to find the goal has been improved in both environments.

- In the second comparison, as illustrated in Table 2, we tested (RRT-connect and A-RRT-connect) techniques in the same aforementioned environments. Fig. 10 illustrates the directional tree exploration of each method and highlights how extensively the nodes are distributed in both maps when compared to the spread of the proposed method for the same map. This reduced the cost incurred during the search for the goal. The same strategies were also applied in the second map, where the RRT-connect displayed a more spread than in the first map, due to the local minimum regions. Consequently, the proposed method's convergence time to reach the goal was enhanced in both environments.
- Third comparison, as illustrated in Table 3, focuses on the path length obtained through the previously described search operations. The cost of generating the initial path (orange line) in both RRT and RRT-connect is substantial when compared to the cost of generating the optimized path (red line) using the proposed A-RRT connect. Furthermore, the time required to create the final optimized path is comparable to that of RRT and RRT-connect. Fig. 11 displays the path length of each method and demonstrates how the optimized path (red path) is shorter and smoother than other paths.

B. EXPERIMENT-2 (DYNAMIC ENVIRONMENT)

This experiment, as illustrated in Table 4, has been tested on a dynamic map of size (500*500 cm), and the results of (100 runs) have been evaluated depending on the t-test measure. The time and cost values had a narrow range between the highest and lowest values. Furthermore, the value of the standard deviation was small, and the average of the absolute value from the median was convincing also.

TABLE 3. Experimental Results Comparison of (RRT, RRT-Connect, and A-RRT-connect TSI)

Static Map	Method	Path Length (cm)	Std. Dev.
MAP-1	RRT	1698	345.39
	RRT-Connect	1596	269.02
	A-RRT-Connect TSI	1398	235.84
MAP-2	RRT	2772	471.31
	RRT-Connect	2360	331.72
	A-RRT-Connect TSI	2257	257.39

TABLE 4. T-test Comparison based on Robot Velocity (13 cm/sec) and Decision Time

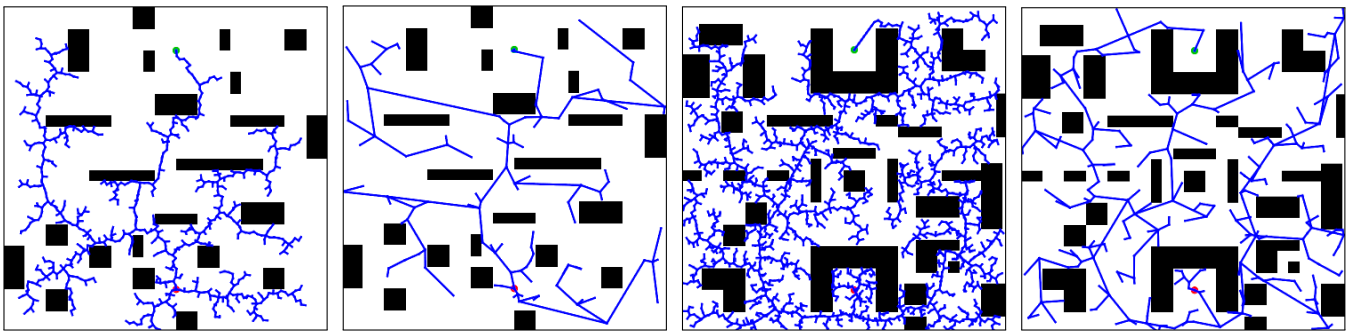
DA-RRT-connect TSI	Time (sec.)	Cost (node)	Path Length (cm)
Mean	59.153	164	691
Median	56.153	112	652
Standard Deviation	12.251	151	66
Highest Value	70.076	951	833
Lowest Value	54.307	93	628
Average Absolute Deviation from Median	23.397	47	272

In this part of the experiment, we replicated the situation of local planning in a dynamic environment depicted in Fig. 12. The paths generated, particularly in real-time, demonstrate the adaptable nature of this method in the presence of unforeseen obstacles across different scenarios. The simulation was implemented utilizing the ROS framework and TURTLEBOT3 model (burger). In the Gazebo view, static obstacles are shown as grey blocks, dynamic obstacles as grey cylinders, and the Lidar range is displayed as blue. In the Rviz view, the global path is depicted as a red line, while the local path is represented by a green one. The start and goal positions are indicated by red and green circles, respectively.

IV. CONCLUSION

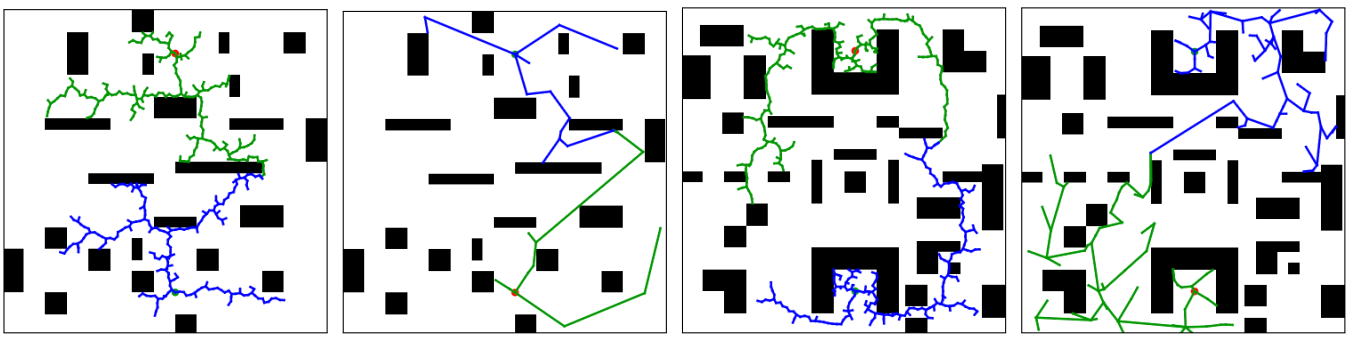
In this paper, we proposed a new dynamic hybrid path planning method that combines the bi-directional Adaptive RRT algorithm with the Triangular Segmented Interpolation (TSI) and the Dynamic Window Approach (DWA). We enhanced the performance of the method by extending it to work adaptively in both directions, from the initial state to the goal point and likewise from the final state to the start point. Then, our method leverages the fast tree exploration of the Adaptive-RRT connect algorithm and the obstacle-avoiding capabilities of the DWA algorithm to find a collision-free path to the goal promptly. The results of our simulations show that the proposed method is able to find a safe and efficient path promptly, even in dynamic environments with varying levels of noise and obstacles. The method is especially useful in scenarios where the environment is changing constantly and re-planning is needed. Experiments conducted in a variety of environments show that our proposed method consistently outperforms computation time, cost, and path optimality.

In future work, there are several directions in which the proposed method can be further improved. One possibility is to incorporate additional information about the environment, such as the curvature of the obstacles or the terrain, to guide



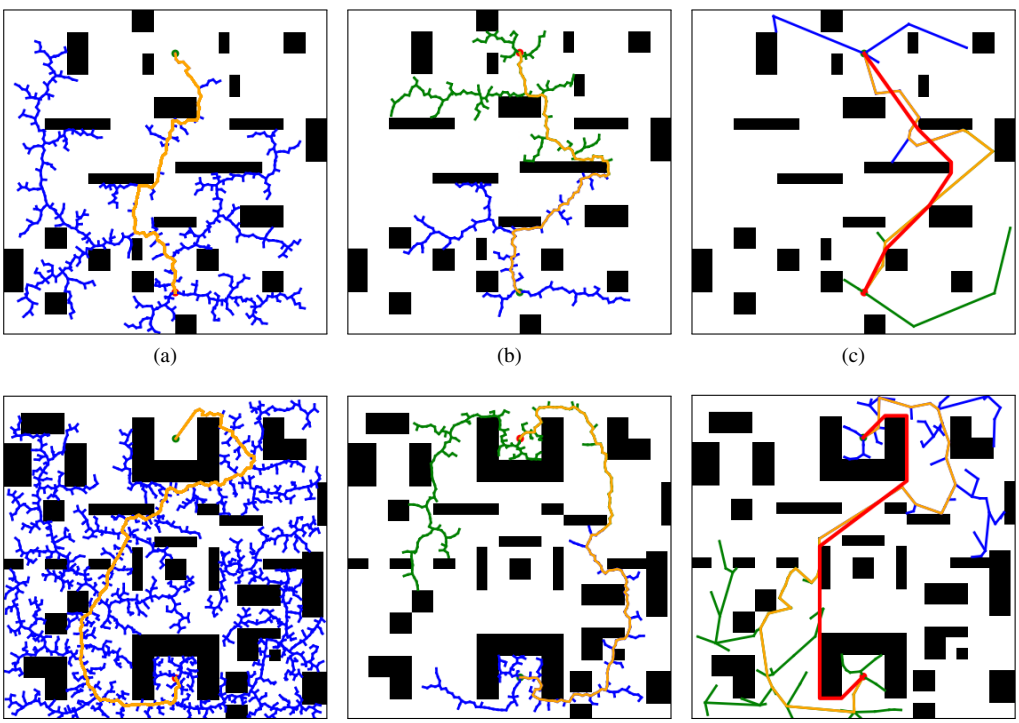
(a) (b) (c) (d)

FIGURE 9. **Tree Exploration of Each Method in (a-b) MAP-1 (c-d) MAP-2.**



(a) (b) (c) (d)

FIGURE 10. **Tree Exploration of Each Method in (a-b) MAP-1 (c-d) MAP-2.**



(a) (b) (c)

(d) (e) (f)

FIGURE 11. **Path length for Each Method in (a-c) MAP-1 (d-f) MAP-2.**

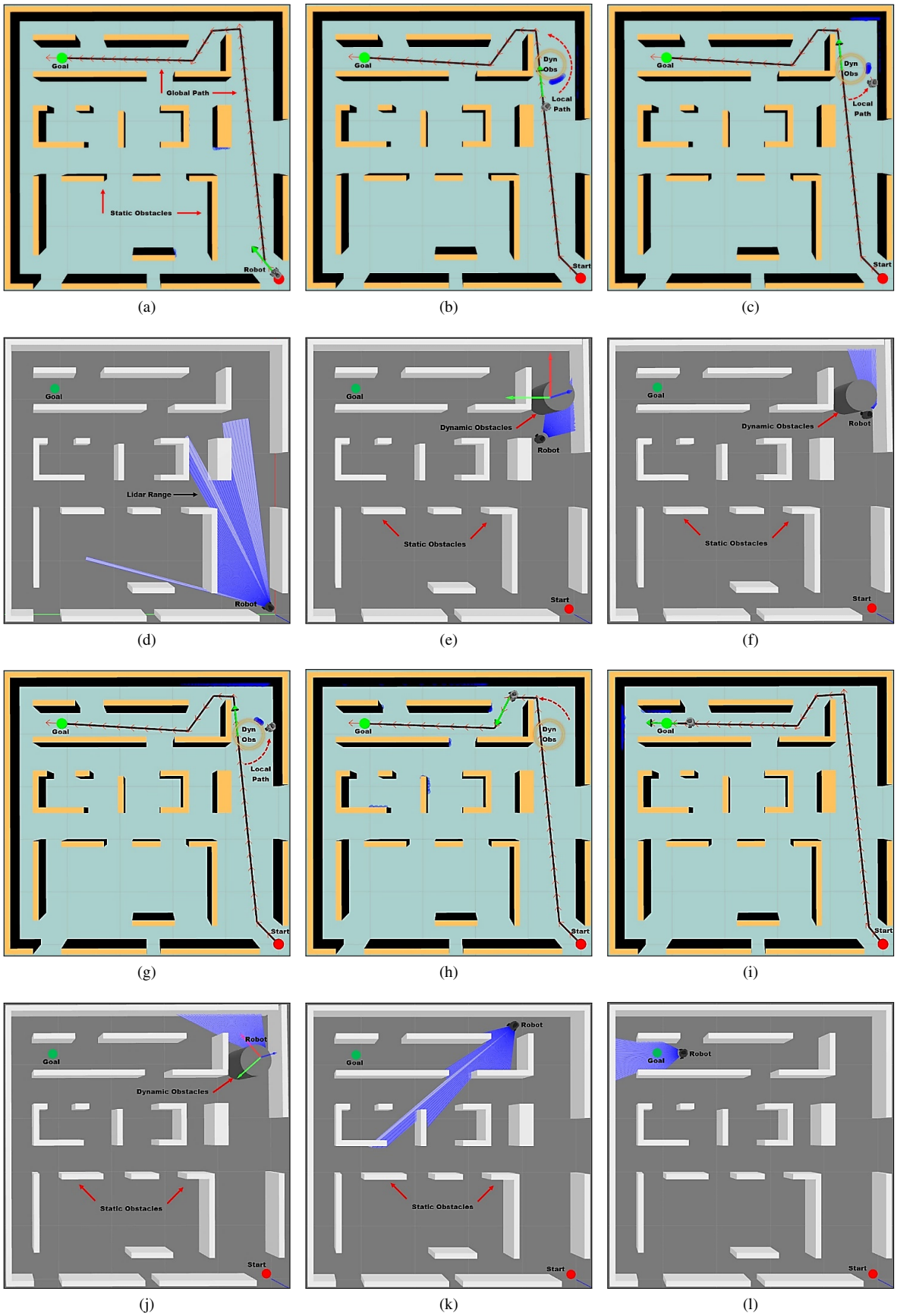


FIGURE 12. Screen Snapshots of Path Planning Simulation in a Dynamic Map (a,b,c,g,h,i) Rviz View (d,e,f,j,k,l) Gazebo View.

the tree expansion and path smoothing. Another possibility is to use machine learning techniques, such as reinforcement learning or imitation learning, to adapt the parameters of the method to the specific environment and improve its performance.

REFERENCES

[1] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[2] T. T. Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," *Robotics and Autonomous Systems*, vol. 86, pp. 13–28, 2016.

[3] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[4] S. Al-Ansarry and S. Al-Darraji, "Mt hybrid rrt-a* regression-based: An enhanced path planning method for an autonomous mobile robots," *Journal of Basrah Researches ((Sciences))*, vol. 47, no. 1, 2021.

[5] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[6] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[7] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[8] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.

[9] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, no. 2, 2010.

[10] S. Klemm, J. Oberländer, A. Hermann, A. Roennau, T. Schamm, J. M. Zollner, and R. Dillmann, "Rrt-connect: Faster, asymptotically optimal motion planning," in *2015 IEEE international conference on robotics and biomimetics (ROBIO)*. IEEE, 2015, pp. 1670–1677.

[11] L. Hong, C. Song, P. Yang, and W. Cui, "Two-layer path planner for auvs based on the improved aaf-rrt algorithm," *Journal of Marine Science and Application*, vol. 21, no. 1, pp. 102–115, 2022.

[12] P. Xin, X. Wang, X. Liu, Y. Wang, Z. Zhai, and X. Ma, "Improved bidirectional rrt* algorithm for robot path planning," *Sensors*, vol. 23, no. 2, p. 1041, 2023.

[13] M. McCourt, C. T. Ton, S. S. Mehta, and J. W. Curtis, "Adaptive step-length rrt algorithm for improved coverage," in *AIAA Guidance, Navigation, and Control Conference*, 2016, p. 0638.

[14] N. Lin and Y.-l. Zhang, "An adaptive rrt based on dynamic step for uavs route planning," in *2014 IEEE 5th International Conference on Software Engineering and Service Science*. IEEE, 2014, pp. 1111–1114.

[15] D.-H. Kim, Y.-S. Choi, T. Park, J. Y. Lee, and C.-S. Han, "Efficient path planning for high-dof articulated robots with adaptive dimensionality," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2355–2360.

[16] S. Zhang, P. Jiexin, S. Yanna, and L. Sun, "Smooth path planning for mobile robot based on adaptive rapidly-exploring random tree," in *2018 IEEE International Conference on Information and Automation (ICIA)*. IEEE, 2018, pp. 591–596.

[17] S. Zhang, J. Pu, Y. Si, and L. Sun, "Path planning for mobile robot using improved adaptive rapidly-exploring random tree," in *2019 International Conference on Control, Automation and Information Sciences (ICCAIS)*. IEEE, 2019, pp. 1–6.

[18] T. Zeng and B. Si, "Mobile robot exploration based on rapidly-exploring random trees and dynamic window approach," in *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2019, pp. 51–57.

[19] J.-G. Kang, Y.-S. Choi, and J.-W. Jung, "A method of enhancing rapidly-exploring random tree robot path planning using midpoint interpolation," *Applied Sciences*, vol. 11, no. 18, p. 8483, 2021.

[20] T. Jia, S. Han, P. Wang, W. Zhang, and Y. Chang, "Dynamic obstacle avoidance path planning for uav," in *2020 3rd International Conference on Unmanned Systems (ICUS)*. IEEE, 2020, pp. 814–818.

[21] J. Dai, D. Li, J. Zhao, and Y. Li, "Autonomous navigation of robots based on the improved informed-rrt algorithm and dwa," *Journal of Robotics*, vol. 2022, 2022.

[22] T.-W. Kang, J.-G. Kang, and J.-W. Jung, "A bidirectional interpolation method for post-processing in sampling-based robot path planning," *Sensors*, vol. 21, no. 21, p. 7425, 2021.

[23] A. Shareef and S. Al-Darraji, "Dynamic multi-threaded path planning based on grasshopper optimization algorithm," in *2022 Iraqi International Conference on Communication and Information Technologies (IICCIT)*. IEEE, 2022, pp. 159–164.

[24] ——, "Grasshopper optimization algorithm based path planning for autonomous mobile robot," *Bulletin of Electrical Engineering and Informatics*, vol. 11, no. 6, pp. 3551–3561, 2022.

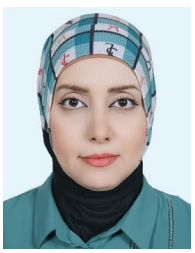
[25] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[26] S. Al-Ansarry and S. Al-Darraji, "Hybrid rrt-a*: An improved path planning method for an autonomous mobile robots." *Iraqi Journal for Electrical & Electronic Engineering*, vol. 17, no. 1, 2021.

SUHAIB AL-ANSARRY received the bachelor degree in Computer Science, Master's degree in Artificial Intelligence from the University of Basrah, in 2007 and 2021 respectively. He is currently a Lecturer with the University of Basrah-Iraq. His research interests include Robotic Navigation, Motion Control, and Path Planning.

SALAH AL-DARRAJI is an Assistant Professor at the Department of Computer Science, College of Education for Pure Sciences, University of Basrah. He studied Computer Science at the University of Basrah (Iraq). He got a Master degree in Computer Science from the same university. He was awarded scholarship by the MoHESR (Iraq) - DAAD (Germany) cooperation program to complete his PhD in the Robotic Research Lab at the University of Kaiserslautern. During his PhD, the focus of his research is on the nonverbal communication with humanoid robot. He completed his PhD in the year 2016.

His research interests includes artificial intelligence, machine learning, deep learning, computer vision, robotics, and humanoid robots. He can be contacted via aldarraji@uobasrah.edu.iq.



ASMAA SHAREEF received her BSc in computer science from the University of Basrah, Iraq, in 2006. She received her MSc degree from the same department in Artificial Intelligence and robotics. She is currently a Lecturer with the University of Basrah-Iraq.

Her research interests include machine learning, deep learning, Artificial intelligence, and Robotics. She can be contacted at the emails: asmaa.shareef@uobasrah.edu.iq and asmaa.sharef@gmail.com.

DHAFER G. HONI received the Bachelor's in 2013 and Master's degree in 2016 from University of Thi-Qar , iraq. His research interest in Artificial intelligence, Machine Learning, Deep Learning, Neural Networks , Computer Vision & Medical image processing. He published many articles that indexd in Scientific Citation Index (SCI)and Scopus. Now He work as a rapporteur in Computer Sciences Department College of education for pure science- basrah University., iraq.

FRANCESCA FALLUCCHI is a researcher of University of Rome GUGLIELMO MARCONI. From 01/07/2017 to 31/12/2017 she is an information scientist in the Digital Information and Research Infrastructures (DIRI) at the Georg Eckert Institute where she works in the project "WorldViews".

In 2015, she was an employee at the company 4IT Solution S.R.L. and she took care of the realization of systems for the management of big data from structured and unstructured open data. From 2012 to 2014 she was an employer-coordinated freelance worker at AgID (Agenzia per l'italia digitale) where she has carried out research and development activities in Open Data with semantic techniques. Her main contribution is the development of methods for the knowledge discovery from structured and unstructured PA data. In particular, she has supported the development of the PA search engine and she has contributed to build and to populate the PA knowledge base and to discovery e-services in the public sector. She has got the PhD in Computer Science and Engineering at the University of Rome Tor Vergata, with the thesis "Exploiting Transitivity in Probabilistic Models for Ontology Learning". She participated in the following research projects: italia.gov.it, D4Science and Diligent. She was author of many publications in conferences and journals in the Semantic Web, NLP, machine learning, Digital Humanities and related research fields.