Path planning for mobile robots with improved RRT Connect algorithm

*Authors: Hariharasudan Muralidaran (120172656), Harshavarthan Varatharajan (*120258688)

Abstract—This report describes the implementation and comparison of path planning algorithms for mobile robots, based on the paper "Improved RRT-Connect Based Path Planning Algorithm for Mobile Robots." The algorithms used include Rapidly Exploring Random Tree (RRT), RRT, RRT-Connect, and an improved RRT-Connect. The improved algorithm uses a third midpoint to bias the flow of the spanning tree, allowing it to converge more efficiently at a common target. By incorporating these improvements, the improved RRT-Connect algorithm achieves significant performance gains in terms of search time and path length. This report assesses each algorithm's execution performance in various environments, providing detailed insights into the effectiveness and efficiency of the improvements. The results confirm that the improved RRT-Connect consistently outperforms traditional RRT algorithms, making it an appealing approach for mobile robot path planning.*

Keywords—RRT, converge, tree, path planning.

I. INTRODUCTION

Mobile robots are increasingly being used in a variety of industries, including healthcare and agriculture, and they rely heavily on effective path planning for safe and efficient navigation. Path planning seeks to find an optimal or near-optimal route from a starting point to a destination while avoiding obstacles. Several algorithms, including Dijkstra's, A*, and heuristic-based methods and search-based approaches, have attempted to solve this challenge, but they often fail in high-dimensional or obstacle-dense environments. Sampling-based algorithms, such as Rapidly Exploring Random Tree (RRT), provide probabilistic completeness and versatility. RRT-Connect improves search performance by using bidirectional expansion and a greedy approach to reduce search time.

However, traditional RRT methods are still prone to blind exploration and suboptimal paths. In this report, we present an improved RRT-Connect algorithm that incorporates a third midpoint between the starting and target locations, resulting in a quadruple tree structure that reduces search time. Furthermore, by biassing the tree's growth towards the goal, the improved algorithm accelerates convergence while decreasing the tendency for unproductive exploration. The report compares the performance of RRT, RRT*, RRT-Connect, and improved RRT-Connect algorithms in various environments, demonstrating how these improvements result in significant increases in efficiency and reliability, making it the best approach for path planning for mobile robot.

II. LITERATURE REVIEW

Mobile robot autonomy relies on efficient path planning algorithms to ensure safe travel through dynamic and complex environments. Sariff and Buniyamin (2006) present an in-depth study of mobile robot path planning algorithms, emphasising the importance of balancing search speed and path optimality in a variety of environmental conditions. Classical algorithms, such as Dijkstra's and A*, helped shape deterministic path planning, but they struggled with high-dimensional spaces and dynamic obstacles. Heuristic approaches, such as the artificial potential field (APF) method and bio-inspired ant colony optimisation, increased search efficiency but had limitations in scalability and optimality.

To address these limitations, sampling-based path planning algorithms were developed. Elbanhawi and Simic (2014) evaluated these methods, highlighting their probabilistic completeness and ability to handle complex, high-dimensional spaces. The Rapidly-Exploring Random Tree (RRT) algorithms, first introduced by LaValle, used random sampling to rapidly explore configuration spaces. Kuffner and LaValle (2000) improved search speed further with their RRT-Connect algorithm, which used bidirectional expansion and greedy strategies. However, both approaches struggled with suboptimal path generation caused by random exploration. Karaman and Frazzoli (2011) proposed RRT* to achieve asymptotically optimal paths by refining tree structure through continuous rewiring. Gammell et al. (2014) proposed Informed RRT*, which improved sampling by concentrating on admissible heuristics within an ellipsoidal region.

Further advancements aimed to improve these sampling-based algorithms. Kang et al. (2016) proposed a goal-oriented sampling strategy that effectively directed tree growth towards a specific goal, reducing blind exploration and increasing search efficiency. Chen and Yu (2021) created an improved RRT algorithm for UAVs that strategically expands tree structures to avoid unnecessary exploration and reduce computation time. based on these developments, the improved RRT-Connect algorithm described in this report uses a third midpoint between the starting and target points to form a quadruple tree structure, which is combined with goal biassing to accelerate convergence. This combination reduces blind spots and ensures faster, more optimal pathfinding than traditional approaches.

III. METHODS OF RELATED ALGORITHMS

A. Rapidly exploring Random Tree (RRT)

1) WORKING EXPLANATION

High-dimensional spaces can be effectively searched using the sampling-based path planning algorithm known as Rapidly-Exploring Random Tree (RRT). It works especially well for issues where explicit graph construction is

impractical due to a large or non-convex search space. Beginning at a predetermined root node, the algorithm gradually constructs a search tree. Within the configuration space, random points are sampled, and the closest node on the current tree is found. The tree then grows in the direction of the sampled point from this closest node, usually with a fixed step size to avoid big jumps. The new node is added to the tree if there are no obstacles on the path leading to the sampled point.

This process continues until the search tree reaches the target node or a predetermined number of iterations are completed. If a new node gets close enough to the target point, the search is considered complete. It easy to backtrack the path from the goal node because we make use of a tree structure and in a tree structure each child has one patent and so on. The flow of the working explanation can be visualized in Figure 1. However, while the probabilistic nature of RRT ensures that a viable path is found if one exists, the paths are frequently suboptimal due to the randomness of exploration, resulting in unnecessary turns and increased path length. Despite these drawbacks, RRT is still widely used due to its efficiency in exploring large search spaces and quickly determining viable paths, making it especially useful in the mobile robotics and autonomous navigation.

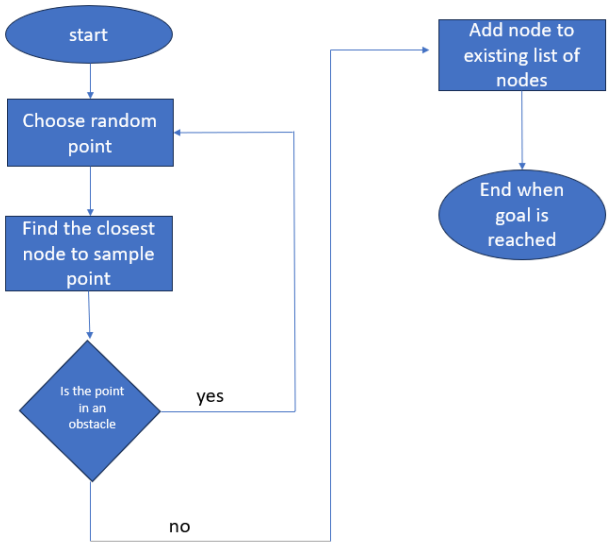

Figure 1. Flowchart for RRT

2) PSEUDO CODE

Algorithm Rapidly-exploring Random Tree (RRT)

```
procedure RRT(start, goal, area, parameters)
    Initialize tree with start node
    while not goal reached and within iteration limits
        rand_point <- select random point from area
        nearest_node <- find nearest node in tree to rand_point
        new_node <- steer towards rand_point from nearest_node, limited by step_size
        if new_node is in a free space
            add new_node to tree, connecting it to nearest_node
        if distance from new_node to goal is within tolerance
            optionally, attempt to connect directly to goal
            if connection successful
                mark goal as reached
    if goal is reached
        path <- construct path from start to goal through tree
        return path
    return failure
end procedure
```

Key Operations:
- select random point: Chooses a point randomly within the defined exploration area.
- find nearest node: Identifies the closest node in the tree to a given point using a distance metric.
- steer: Creates a new node by moving from an existing node towards a target point but only up to a specified distance (step_size).
- construct path: Traces back from the goal to the start through parent nodes in the tree.

3) DRAWBACKS

Although the Rapidly Exploring Random Tree (RRT) algorithm is useful for identifying feasible paths, it has some drawbacks. Because of its random sampling, it frequently generates longer-than-necessary and suboptimal paths. Without heuristic guidance, exploration occurs blindly, causing the tree to grow in unproductive directions. Goal biassing remains less effective in complex configurations like narrow corridors. Nearest-node searches become computationally expensive as the search tree grows, slowing down the algorithm for larger trees, particularly those in high-dimensional spaces. Furthermore, RRT prioritises feasibility over path optimisation, which means that once a viable path is identified, it does not automatically improve. These issues demonstrate how RRT* enhancements are required to ensure efficiency and optimality.

B. RRT-star

1) WORKING EXPLANATION

RRT* (Rapidly-Exploring Random Tree Star) is a significant improvement over the original RRT algorithm, which aims for asymptotically optimal paths through continuous tree rewiring. While the standard RRT algorithm finds feasible paths using random sampling, the resulting paths tend to be suboptimal because it prioritises goal achievement over path quality. RRT*, on the other hand, seeks to reduce the path cost by continually enhancing the solution.

RRT*, like RRT, begins by sampling random points within the configuration space and extending the nearest existing node to the sampled point with a fixed step size. However, before adding this new node to the tree, the algorithm determines the best connection by weighing the costs of all nodes within a given radius. The algorithm selects the parent node with the lowest path cost to the new node, ensuring that each node added is connected via the path with the least cost. This decision is made using a cost function that computes the total cost from the starting point to each potential parent.

To reduce total path cost, a local rewiring step is performed after the new node is added to the tree to determine whether nearby nodes can be reconnected to the newly added node. The tree structure is rewired to update neighbouring nodes that are more cost effective to reach via the new node (Figure 2). As more nodes are added, this local optimisation ensures that the tree is gradually improved, lowering the total path length and cost.

With enough iterations, the RRT* algorithm can find paths that asymptotically approach the optimal solution. This results in paths that not only reach the goal but are also much

higher quality than the original RRT, making RRT* an excellent choice for path planning in challenging environments.

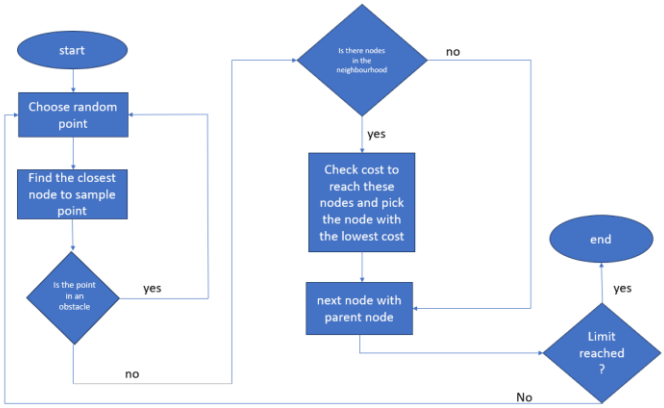

*Figure 2. Flowchart for RRT**

Initially, both RRT and RRT* sample random points to explore the configuration space. RRT* rewires nodes to improve path quality by connecting them via the node with the lowest cumulative cost. After several iterations, the RRT* path narrows to a much shorter, optimal path, whereas the RRT path remains more winding.

2) PSEUDO CODE

Algorithm RRT-Star
procedure RRT-Star(start, goal, area, parameters)
Initialize tree with start node
Set start node cost to zero
while not goal reached and within iteration limits
rand_point <- select random point from area
nearest_node <- find nearest node in tree to rand_point
new_node <- steer towards rand_point from nearest_node
if new_node is in a free space
add new_node to tree
connect new_node to tree optimally
if new_node is near goal
attempt to connect directly to goal
if goal is connected
path <- construct path from start to goal
return path
return failure
end procedure

Key Operations:
- select random point: Chooses a point randomly within the defined area.
- find nearest node: Identifies the closest node in the tree to a given point.
- steer: Moves from one node towards another but only up to a set distance.
- connect optimally: Rewires the tree around the new node if doing so reduces path costs.
- construct path: Traces back from the goal to the start through parent nodes.

3) DRAWBACKS

Although useful for determining asymptotically optimal paths, the RRT* algorithm has a few drawbacks. Compared to basic RRT, its continuous rewiring process requires more processing power, especially in high-dimensional spaces, resulting in longer computation times and increased memory usage. These computational demands are exacerbated by the fact that reaching optimality requires numerous iterations.
The rewiring step adds complexity because, as the tree grows, it becomes more difficult to locate and update neighbouring nodes. The radius chosen also influences this step; a small radius limits optimisation, whereas a large radius requires more computation.

Because RRT* depends on static structures, it might be hard to adjust to dynamic environments where obstacles are constantly changing. Even with these drawbacks, RRT* is still thought to be very good at identifying the best courses of action in challenging situations when sufficient computing power is available.

C. RRT-CONNECT

1) WORKING EXPLANATION

An extension of the Rapidly-Exploring Random Tree (RRT), the RRT-Connect algorithm uses a bidirectional search strategy to improve pathfinding efficiency. It grows two trees simultaneously, one from the starting position and one from the goal, as opposed to growing one tree from the start point. Starting from their respective start nodes, each tree progressively spreads towards random sample points in the configuration space. The algorithm switches between the start and goal trees during each iteration. The closest node to a randomly selected point in the current tree is found, and a fixed step size is then used to move towards this point. The new node is added to the tree if the path is clear of obstructions.

After a tree is extended, the algorithm tries to connect it to the other tree using a greedy connection strategy which is explained in Figure 3. It finds the nearest node in the other tree and contacts the node that just joined. This bidirectional growth and greedy connection minimise the search space and allow the trees to meet quickly by rapidly decreasing the gap between them. After linking, the algorithm iterates through both trees to generate a complete path from start to goal. By bringing the two trees closer together, this technique minimises unnecessary exploration and significantly expedites pathfinding.

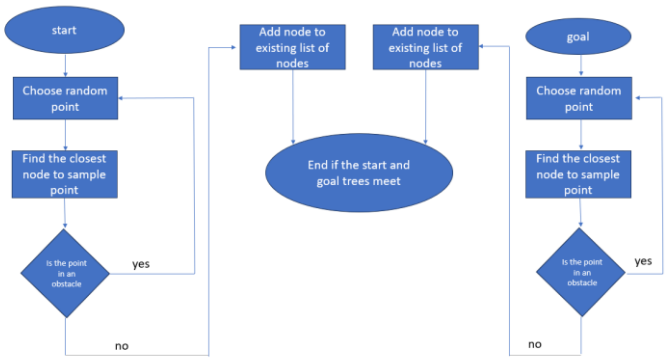

Figure 3. Flowchart for RRT-Connect

Both RRT and RRT connect essentially do the same thing but in the case of RRT connect the algorithm grows the tree from both the start and goal so the number of nodes

explored will be less and the time taken for the process will also be reduced drastically

2) PSEUDO CODE

Algorithm RRT-Connect

```
procedure RRTC(start, goal, area, parameters)
    Initialize trees with start and goal as roots
    visited_start, visited_goal ← {start}, {goal}
    parents_start, parents_goal ← {start: None}, {goal: None}
    while not (meeting_condition or iteration_limit_reached)
        rand_point_start, rand_point_goal ← generate_random_points(area)
        nearest_node_start ← find_nearest(visited_start, rand_point_start)
        nearest_node_goal ← find_nearest(visited_goal, rand_point_goal)
        new_node_start ← move_towards(nearest_node_start, rand_point_start, step_size)
        new_node_goal ← move_towards(nearest_node_goal, rand_point_goal, step_size)
        if is_free(new_node_start) and new_node_start not in visited_start
            visited_start.add(new_node_start)
            parents_start[new_node_start] ← nearest_node_start
        if is_free(new_node_goal) and new_node_goal not in visited_goal
            visited_goal.add(new_node_goal)
            parents_goal[new_node_goal] ← nearest_node_goal
        if nodes_meet(new_node_start, visited_goal) or nodes_meet(new_node_goal, visited_start)
            connection_node ← find_meeting_node(new_node_start, new_node_goal)
            break
    if connection_node is found
        path_from_start ← backtrack(parents_start, connection_node)
        path_from_goal ← backtrack(parents_goal, connection_node)
        full_path ← combine_paths(path_from_start, path_from_goal)
        return full_path
    return failure
end procedure
```

Procedures:
- generate_random_points(area): Generates random points within the defined area.
- find_nearest(visited, point): Finds the node closest to point in the visited set.
- move_towards(nearest_node, rand_pt, step_size): Steers from nearest_node towards rand_pt, limited by step_size.
- is_free(node): Checks if the node position is free of obstacles.
- nodes_meet(node, visited_other): Checks if node is close enough to any nodes in visited_other.
- find_meeting_node(node_start, node_goal): Identifies the connection node where paths from start and goal meet.
- backtrack(parents, node): Constructs the path by tracing back from node using parent links.
- combine_paths(start_path, goal_path): Combines paths from start and goal trees at meeting point.

3) DRAWBACKS

Although the RRT-Connect algorithm is a significant improvement over the original RRT, it still has some flaws. It can join two trees faster, but because it prioritises connection over optimisation, it frequently produces suboptimal paths. When the trees grow larger, particularly in high-dimensional spaces, the bidirectional growth of the algorithm necessitates frequent nearest-node searches, resulting in high computational loads. Furthermore, it performs poorly in cluttered environments, where obstacles that require multiple attempts can cause the convergence to slow down significantly. The algorithm is also prone to goal biassing, which can lead to overly aggressive expansion and insufficient configuration space exploration. Furthermore, the algorithm does not automatically optimise the path following the connection, resulting in possible but not ideal paths that have long or sharp turn. Additionally, RRT-Connect finds it challenging to adjust to dynamic environments due to the static nature of the tree structure.

However, RRT-Connect's effective bidirectional search technique makes it helpful in many of situations.

IV. IMPROVED RRT-CONNECT

A. WORKING EXPLANATION

The RRT-Connect technique is further developed by the Improved RRT-Connect algorithm, which makes improvements to speed up convergence and produce more effective paths. The search space was effectively divided into four regions by the addition of a third midpoint node, which was placed strategically between the start and goal points. One of the two main trees starts at the goal and grows towards the midpoint, while the other grows in the opposite direction, starting at the starting point and expanding towards the midpoint. From this midpoint, two additional secondary trees grow outward, one towards the goal and the other towards the starting point. All four trees cooperate to effectively connect the starting and goal points. By using a midpoint, we simplified the search space, creating a reduced area for each tree to explore, which makes convergences faster.

In order to prevent blind exploration and speed up convergence, we goal-biased all four trees. To ensure intended expansion, every tree—primary or secondary—is guided towards its goal through biased sampling. The secondary tree employs strategic growth to establish a direct connection between the target points and the primary trees, while the primary trees prioritise reaching the midpoint as soon as possible. The possibility that all four trees will efficiently meet is increased while exploration time is decreased by this bidirectional goal biasing. The algorithm traces back the entire path, from the starting point to the midpoint and ultimately to the destination, once the primary trees have connected via the secondary trees. This approach ensures that the trees converge rapidly, producing faster and more optimized paths by leveraging the midpoint structure and directed tree growth.

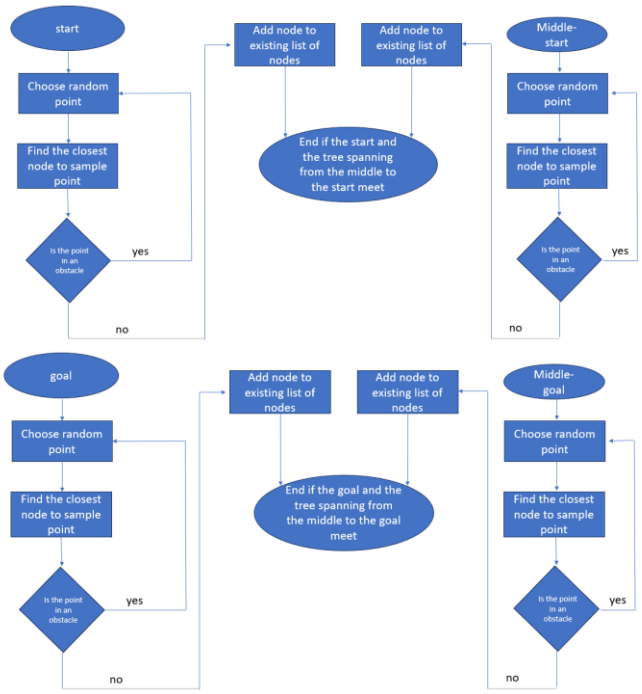

Figure 4. Flowchart for Improved RRT-Connect

As it can be seen in Figure 4. Although RRT-Connect improves things by growing two trees in both directions from the start and goal points, there is still a chance that the trees won't connect well, which will make the search take longer. Alternatively, our Improved RRT-Connect creates four regions in the space by adding a third node that serves as a midpoint. This allows two primary trees to strategically connect from the start and goal, and two secondary trees to strategically connect from the midpoint outward. Because the trees tend to grow directly towards its target, this effectively lowers blind exploration.

B. PSEUDO CODE

Algorithm Imptoved RRT Connect

Procedure IRRTC(start, goal, no_go_points, area_bounds)
mid ← calculate_midpoint(start, goal, no_go_points)
Initialize trees starting at start, mid, goal, and back from mid
visited_from_start, visited_to_mid ← {start}, {mid}
visited_from_goal, visited_to_mid_back ← {goal}, {mid}
parents_from_start, parents_to_mid ← {start: None}, {mid: None}
parents_from_goal, parents_to_mid_back ← {goal: None}, {mid: None}
while trees have not connected
Generate new points for each tree within respective bounds
Expand trees towards their new points using steer_towards
Check each tree node for free path and add to visited if free
Attempt to connect trees if nodes are close enough
if any two trees connect
connection_found ← True
break
if connection_found
Construct paths from connected nodes
Combine paths to form full route from start to goal
return combined path
return failure
end procedure

Procedures:
- calculate_midpoint(start, goal, no_go_points): Calculates a midpoint on the direct line between start and goal, adjusting for obstacles.
- steer_towards(node, target, step_size): Moves from node towards target constrained by step_size.
- is_free_of_obstacles(node, no_go_points): Checks if the path from node to target is free of obstacles in no_go_points.
- attempt_connection(node_a, node_b): Checks if two nodes from different trees can connect without obstacles.
- construct_path(parent_map, start_node, end_node): Constructs the path from start_node to end_node by backtracking through the parent_map.

C. ADVANTAGES

Compared to other RRT algorithms, the Improved RRT-Connect algorithm has a lot of benefits, such as enhanced bidirectional growth and the use of a middle node. The algorithm separates the search space into four regions by adding a third midpoint node between the start and goal. This allows two primary trees (from the start and goal) and two secondary trees (from the midpoint) to grow effectively. The algorithm guides each tree to its destination by means of goal biassing, producing paths that are more optimised and direct with fewer needless detours.

Three random sampling points, one for each tree, are used to ensure a well-balanced exploration-exploitation method. When compared with RRT and RRT-Connect, this targeted exploration decreases redundant searches and accelerates convergence. Moreover, in cluttered environments where traditional algorithms might fail, goal biassing increases pathfinding efficiency while lowering computational load. Consequently, with fewer iterations, the convergence rate increases.

V. RESULTS

The results of our experiments comparing the RRT, RRT*, RRT-Connect, and Improved RRT-Connect path-planning algorithms' performances are shown in this section. Every algorithm was tested on the identical 2D map, beginning, and ending at the same locations, to ensure uniformity. Every trial was carried out on the same computer to guarantee uniform testing conditions. To directly compare the algorithms' effectiveness in resolving the navigation problem, the main metric assessed was the amount of time it took for each algorithm to identify a workable path from the start to the goal point. The comparative analysis showcases the practical benefits of the Improved RRT-Connect algorithm concerning pathfinding efficiency and convergence speed.

Figure 5. Map Enviorment

The map we used is 600 x 200 pixels in area in Figure 5. The green areas indicate the obstacles and the white area is the free space where we choose two points for start and end. We have sampled 10 different pair of start and goal points and have done numerical comparison between the different algorithms discussed above. From the observations made it has been inferred that IRRTC has the fastest and stable results next to RRTC Figure. It can also be observed that even though RRT* is more stable than the general RRT, it takes 1 second more than the regular RRT. The effectiveness and superiority each search tree has on the other can be visually seem from the Figure 6.

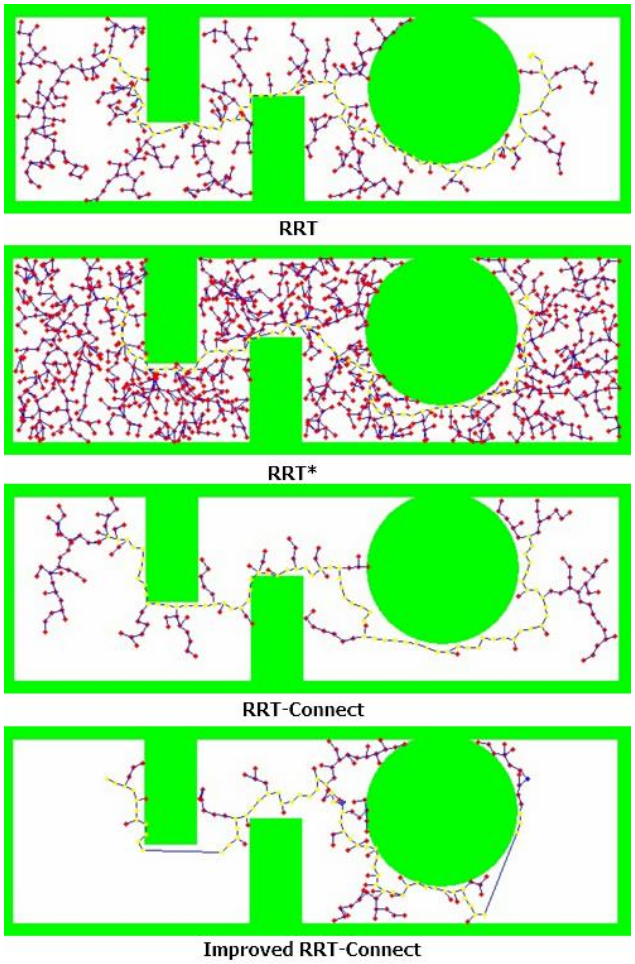

Figure 6. Search trees with same start goal

The difference in the effectiveness can be inferred from the above figure. It can also be noted that the use if third point created by the midpoint of the distance between the start and goal node can be seen in the Improved RRT-Connect tree with a blue dot. To further get a clear understanding on the effectiveness of the different algorithms, we conducted numerous tests of the same map and created and numerical visualization of the robustness and performance as seen in Table 1.

Table 1. Numerical Comparison of Algorithms

start and goal	RRT	RRT*	RRT-C	I RRT C
(20, 20), (580, 50)	0.835	1.4072	0.3612	0.1886
(100,100),(300,50)	1.085	1.358	0.1015	0.0368
(190,100), (300, 50)	0.3089	1.36	0.0597	0.0059
(190,100), (580, 50)	1.747	1.4426	0.398	0.0419
(100,100), (580, 50)	1.026	1.4689	0.3176	0.0663
(100,50), (500, 50)	0.5202	1.5166	0.345	0.07274
(100,50), (400, 180)	0.9223	1.6891	0.4565	0.0228
(100,180), (400, 180)	1.1798	1.4452	0.4367	0.0551
(100,30), (580, 170)	2.4742	1.6984	0.5902	0.0595
(300,180), (580, 180)	0.6252	1.3792	0.1072	0.0206

The IRRT-C has the shortest path planning time of all the other algotithms. The above table can be further visualized in a time vs performance graph in Figure 7.

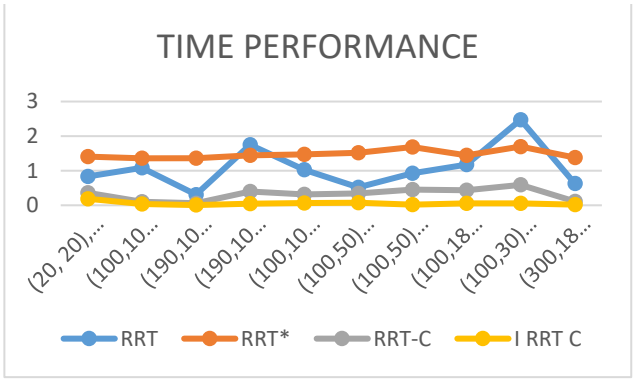

Figure 7. Time performance graph

The results depicted in the accompanying graph and tabulated data provide a comprehensive comparative analysis of the performance metrics for four path planning algorithms: Rapidly-exploring Random Tree (RRT), RRT*, RRT Connect (RRT-C), and Improved RRT Connect (I RRT C). This analysis reveals significant differences in their stability and efficiency across various test scenarios.

From the data, we observe specific performance outcomes for each test case. For instance, in the trajectory from (100, 50) to (500, 50), RRT required 0.5202 seconds to complete the path, while Improved RRT Connect dramatically reduced this time to only 0.07274 seconds. This stark reduction underscores the optimization in path planning that Improved RRT Connect achieves over its predecessors, highlighting its effectiveness in scenarios requiring rapid and complex maneuvers. Despite RRT* often showing longer computation times—as illustrated by the orange line—it is optimized to find more qualitatively optimal paths, rather than prioritizing speed, making it suitable for applications where the quality of the path is critical.

The graph further enhances our understanding by showing the relative performance stability of each algorithm. The blue line, representing RRT, exhibits variable time performance, suggesting its efficiency might wane in more complex environments. In stark contrast, the yellow line of Improved RRT Connect not only shows the lowest computation times

but also maintains a stable and consistent performance across all scenarios. This stability is crucial, indicating the algorithm's reliability in consistently delivering efficient path planning solutions regardless of environmental complexity.

RRT-C, depicted in grey, strikes a balance with better performance times than RRT and RRT* but not as swift as Improved RRT Connect. It also displays more consistent performance compared to RRT*, positioning it as a reliable option for moderately complex environments. Collectively, the graph highlights not just the raw performance differences but also the practical implications of selecting an algorithm based on the specific needs of path quality, execution speed, and adaptability to environmental complexities in real-world applications such as autonomous driving or robotics navigating unpredictable terrains. This analysis serves as a decisive guide for choosing the right algorithm tailored to the specific requirements of dynamic environments where timely and reliable decision-making is paramount.

VI. CONCLUSION

In this study, we conducted a detailed evaluation of four different path planning algorithms: Rapidly-exploring Random Tree (RRT), RRT*, RRT Connect (RRT-C), and Improved RRT Connect (I RRT C), across a series of specified start and goal coordinates in a simulated environment. The primary objective was to quantify and compare the computational efficiency and reliability of each algorithm under various conditions. Our findings indicate that while each algorithm has its unique strengths, Improved RRT Connect consistently provided the fastest and most stable performance across the diverse scenarios tested.

The graphical analysis revealed that the RRT algorithm, though quick in simpler scenarios, often struggled with consistency in more complex environments. This highlights the limitations of basic RRT in adapting to diverse and unpredictable terrain, a common challenge in real-world applications. In contrast, RRT*, represented by its longer computational times, excelled in generating more optimal paths. This characteristic makes RRT* particularly valuable in applications where the quality of the path is more critical than the speed of computation, such as in high-stakes environments where path optimality can prevent operational hazards.

Improved RRT Connect demonstrated remarkable efficiency and robustness, outperforming other algorithms in almost all test cases. This superior performance can be attributed to its enhanced algorithmic structure, which optimizes both the search space and the path connectivity process more effectively than its predecessors. The consistent low computation times and high reliability in path stability make it an ideal choice for dynamic and complex environments where quick and dependable path planning is crucial, such as in autonomous vehicle navigation and robotic surgery.

Looking forward, the integration of machine learning techniques with these path planning algorithms presents a promising avenue for research. Machine learning could potentially enable algorithms to learn from past computations and environmental interactions, thereby improving their predictive capabilities and operational efficiency. This approach could refine the algorithms' ability to handle even more complex and dynamically changing environments, further enhancing their applicability in advanced robotics and autonomous systems.

Conclusively, our study not only underscores the importance of selecting the appropriate path planning algorithm based on specific operational demands and environmental complexities but also points to the potential enhancements through technological advancements. As the field of robotics continues to evolve, the ongoing refinement and development of path planning algorithms will be crucial in achieving higher levels of autonomy and efficiency in robotic systems. This will undoubtedly have a transformative impact on various sectors, including transportation, logistics, healthcare, and more, propelling us towards a more automated future.

REFERENCES

[1] Sariff, N. and Buniyamin, N., 2006, June. An overview of autonomous mobile robot path planning algorithms. In *2006 4th student conference on research and development* (pp. 183-188). IEEE.

[2] Karaman, S. and Frazzoli, E., 2011. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, *30*(7), pp.846-894.

[3] Kang, G., Kim, Y.B., You, W.S., Lee, Y.H., Oh, H.S., Moon, H. and Choi, H.R., 2016, July. Sampling-based path planning with goal oriented sampling. In *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)* (pp. 1285-1290). IEEE.

[4] Elbanhawi, M. and Simic, M., 2014. Sampling-based robot motion planning: A review. *Ieee access*, *2*, pp.56-77.

[5] Gammell, J.D., Srinivasa, S.S. and Barfoot, T.D., 2014, September. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ international conference on intelligent robots and systems* (pp. 2997-3004). IEEE.

[6] Chen, J. and Yu, J., 2021, March. An improved path planning algorithm for UAV based on RRT. In *2021 4th international conference on advanced electronic materials, computers and software engineering (AEMCSE)* (pp. 895-898). IEEE.