ENPM661 Final Report
Fast RRT based Optimal Path finding Algorithm
Group 43

Vijay Chevireddi*, Chandhan Saai Katuri†, Abraruddin Syed‡
Univeirsity of Maryland
College Park, Maryland

***Abstract*—In this project, we used Fast-RRT, a pathfinding algorithm, for motion planning in robotic navigation. Fast-RRT, which enhances the standard Rapidly Exploring Random Trees (RRT) method, is designed to efficiently compute near-optimal paths, particularly in challenging environments. Comprising two primary components, Fast Sampling Strategy and Random Steer, this algorithm first rapidly identifies an initial feasible path and then optimizes it by integrating multiple paths to achieve near-optimality. Our implementation of Fast-RRT in the Gazebo simulation environment, coupled with ROS for a TurtleBot, demonstrates significant improvements over traditional RRT and RRT* algorithms. Specifically, Fast-RRT incorporates a Fast-Sampling technique that selectively samples within unexplored areas of the environment, enhancing both the speed and stability of the search. Furthermore, the introduction of a Random Steering expansion strategy effectively addresses the limitations commonly encountered in narrow passages. Experimental results have shown that Fast-RRT found path-finding solutions up to 20 times faster than RRT*, indicating immense potential in realizing advanced robotic navigation tasks. The report gives a detailed discussion on implementation, challenges experienced during implementation, and the comparative advantages of using Fast-RRT in a simulated robotics scenario.**

I. Introduction

Graph-based methods, exemplified by Dijkstra's algorithm and the A* algorithm, are noted for discretizing the state space into a graph to find optimal paths but struggle with large-scale applications. Sampling-based methods, such as PRM and RRT, are highlighted for their effectiveness in larger spaces by building graphs or trees through random state space sampling.

In this context, the Fast-RRT algorithm is introduced as an enhancement of the traditional RRT method. It is described as significantly improving upon the original by integrating two novel approaches: Fast Sampling and Random Steering. These innovations are credited with addressing the high variability in search times and suboptimal performance in restricted areas that are common in traditional RRT implementations.

The Fast-RRT algorithm is characterized by its two-phase process: an initial rapid pathfinding phase using Improved RRT, and a subsequent optimization phase called Fast-Optimal, which merges and refines paths to approach optimality more quickly. This methodological innovation is claimed to accelerate the search for near-optimal paths by up to twenty times compared to the RRT* algorithm.

Through this approach, Fast-RRT is portrayed as offering significant advancements in the field of motion planning, suggesting its potential applicability in a range of practical scenarios.

A. Rapidly-exploring Random Tree (RRT)

The RRT algorithm operates by incrementally building a tree rooted at the initial state and expanding towards randomly sampled points within the state space. Below is an illustration of the RRT process:

The steps involved in the RRT algorithm are outlined as follows:

1) **Initialization**: Start with a tree T containing only the initial state x_{init}.
2) **Sampling**: Randomly sample a state x_{rand} from the free state space X_{free}.
3) **Nearest Neighbor**: Find x_{near}, the closest node in T to x_{rand}.
4) **Steering**: Generate a new state x_{new} by applying a steering function from x_{near} to x_{rand}.
5) **Collision Checking**: Check if the path from x_{near} to x_{new} is free of obstacles.
6) **Node Addition**: If the path is clear, add x_{new} to T and connect x_{near} to x_{new} with an edge.
7) **Goal Check**: If x_{new} is within the target region X_{goal}, return the path from x_{init} to x_{new}.

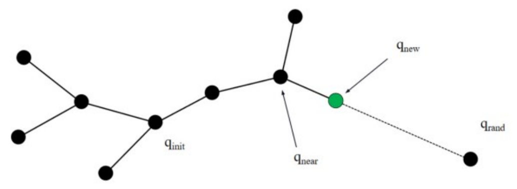

Fig. 1. Illustration of the RRT algorithm in action.

B. Rapidly-exploring Random Tree Star (RRT)*

RRT* enhances RRT by introducing a rewire step to optimize the path efficiency. The additional steps in RRT* include:

1) **Optimal Connection**: After adding x_{new}, consider if any node within a predefined radius r could benefit from having x_{new} as its parent, thereby obtaining a more optimal path cost.

2) **Rewiring**: Adjust the parent of each neighboring node if connecting it to x_{new} provides a shorter path than its current parent.
3) **Path Optimization**: As the number of samples increases, continuously evaluate and adjust the connections to ensure optimal paths from x_{init}.

C. Limitations and Practical Applications

While RRT efficiently finds feasible paths, it does not guarantee path optimality. The outcome heavily depends on the randomness of the sampling process. Conversely, RRT* aims to optimize the path over time but at the cost of increased computational effort, which may not be suitable for time-sensitive applications.

Both algorithms have been adapted to meet specific needs and constraints in various environments, demonstrating their versatility and robustness in tackling complex navigation tasks in dynamic settings.

II. Literature Review

In their 2021 article, Wu et al. (2021), introduce Fast-RRT, a novel algorithm designed to address the inefficiencies of the Rapidly-exploring Random Trees (RRT) in motion planning. Traditional RRT algorithms, while effective in finding feasible paths quickly, often suffer from significant variances in search times and perform poorly in environments with narrow passages. The proposed Fast-RRT algorithm enhances the conventional RRT by integrating an Improved RRT module for faster initial path finding and a Fast-Optimal module for combining paths to achieve near-optimal solutions. Notably, the Fast-RRT outperforms traditional RRT and RRT* in terms of speed by avoiding redundant samplings and optimizing path fusion, making it up to 20 times faster than the RRT* algorithm in obtaining near-optimal paths, thus showing significant potential for practical applications in areas like autonomous vehicle navigation and robotics

III. Methodology

The combination of fast sampling and random steering is what makes fast-RRT fast.

A. Fast Sampling

The Rapidly-exploring Random Tree (RRT) algorithm is a foundational tool in path planning that involves random state sampling and tree expansion strategies. The basic process of the RRT involves:

1) Randomly sampling a state from the entire state space, termed x_{rand}.
2) Steering from the nearest node in the existing tree (x_{near}) towards x_{rand}, resulting in a new node x_{new}.
3) Checking if the new node x_{new} is within a predefined threshold r of a target point, and if so, considering the target as reached. If not, the search continues.
4) Defining an explored area centered at each node with radius r. This area is considered explored, and ideally, new nodes should extend into unexplored regions.

Despite its effectiveness, RRT faces challenges such as invalid expansions where the new node's explored area overlaps significantly with previously explored regions, thus contributing little to new explorations and leading to inefficiencies.

To address these issues, a modified algorithm called **Fast-Sampling** was used, which enhances the efficiency of RRT by:

- **Avoiding previously sampled areas:** Fast-Sampling refuses to sample within the bounds of areas already explored, effectively reducing the number of redundant expansions.
- **Focusing on unexplored spaces:** By guiding the tree's expansion towards genuinely unexplored territories, Fast-Sampling significantly speeds up the exploration process and reduces the computational load and memory usage.

Figures illustrating the principles of standard RRT and Fast-Sampling are shown below. Figure 2 demonstrates the issue of overlapping explored areas in traditional RRT, while Figure 3 highlights the effectiveness of the Fast-Sampling approach.

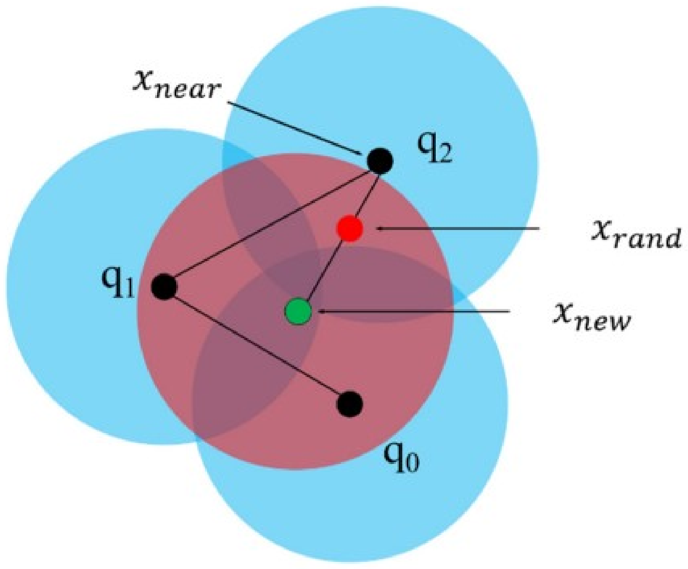

Fig. 2. Invalid expansions in traditional RRT where new nodes overlap with previously explored areas.

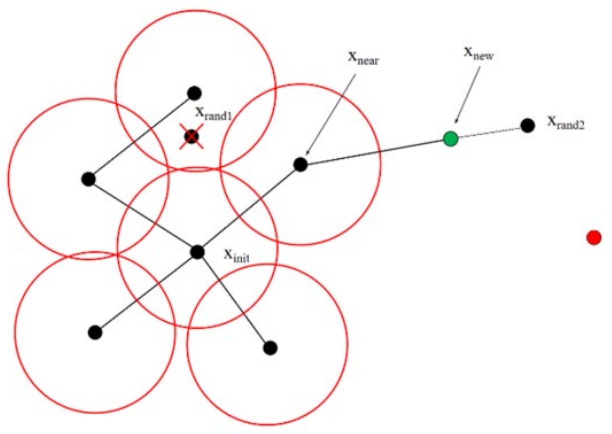

Fig. 3. Illustration of Fast-Sampling, avoiding redundant expansions by focusing on unexplored areas.

B. Random Steering

The standard RRT algorithm can struggle in environments characterized by narrow passages due to frequent failed expansions where paths extend into obstacles, leading to inefficient space exploration.

To overcome these challenges, a modification named "Fast-Expand" has been proposed. This strategy alters the expansion mechanism of the algorithm to enhance its ability to navigate through constrained spaces. Originally, the algorithm expands by generating a random state, x_{rand}, and attempting to connect this state with the closest node in the tree, x_{near}. However, if this expansion directly leads to a collision with an obstacle, the attempt fails.

The "Fast-Expand" modification introduces a fallback mechanism: if an expansion toward a randomly generated state, x_{rand1}, fails due to an obstacle, the algorithm does not simply terminate at that failure. Instead, it generates a new direction for expansion, denoted by θ. This new direction is utilized to attempt another expansion from x_{near} to a new state, x_{new1}. If this path is obstacle-free, it is added to the tree. Otherwise, the algorithm continues to sample another random state, thereby adapting until it successfully navigates through the narrow area.

This method, Fast-Expand, enhances the algorithm's efficiency by reducing the number of futile expansions in environments where navigation through narrow passages is crucial. This makes the Fast-Expand adaptation particularly adept for use in complex terrains where obstacles frequently flank viable paths, optimizing the RRT algorithm for more challenging environments.

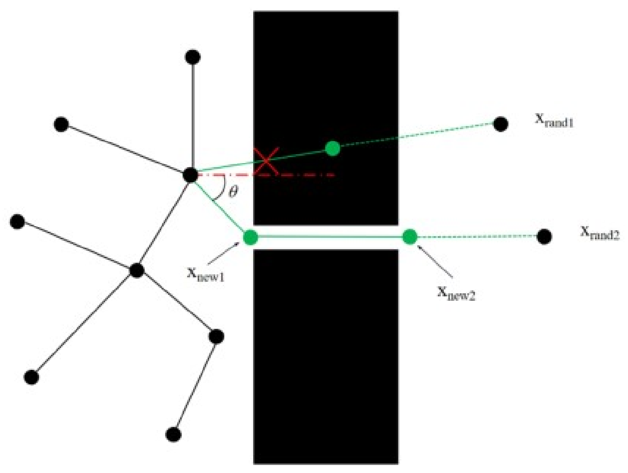

Fig. 4. Random Steering

IV. Pseudo Code

Algorithm 1 Main Function

```
1: procedure MAIN
2:     Initialize obstacle_map, start, goal, step_size, and obstacles
3:     Create obstacles on the obstacle_map
4:     $path \leftarrow$ fast_rrt($start, goal, obstacle\_map, step\_size$)
5:     if path exists then
6:         Draw the path on the obstacle_map
7:     end if
8:     Display the obstacle_map with the path
9:     Wait for user input and close the window
10: end procedure
```

The algorithm outlined in Algorithm 1 presents a main function for path planning using the Rapidly-exploring Random Tree (RRT) algorithm. In this algorithm, the initialization of variables such as the obstacle map, start and goal positions, step size, and obstacles is crucial for setting up the planning environment. Subsequently, obstacles are created on the obstacle map, and the `fast_rrt` function is called to compute a path from the start to the goal.

The `fast_rrt` function implements the core RRT algorithm, which involves randomly sampling points in the configuration space, expanding the tree towards feasible configurations, and efficiently searching for a path. This algorithm is fundamental in motion planning for robotics and autonomous systems, enabling them to navigate complex environments by generating feasible trajectories while avoiding obstacles.

Extensions and optimizations of the RRT algorithm, such as RRT* or RRT-Connect, can further enhance path planning performance by addressing issues like exploration efficiency and path quality. These improvements are particularly crucial in real-world applications where real-time planning and obstacle avoidance are essential for safe and efficient operation.

The algorithm presented in Algorithm 2 depicts the Fast RRT Function, which serves as a key component in path

Algorithm 2 Fast RRT Function

```
function FAST_RRT($start, goal, obstacle\_map, step\_size$)
    $path \leftarrow$ improved_rrt($start, goal, obstacle\_map, step\_size$)
    if path exists then
        return path
    else
        return None
    end if
end function
```

planning using the Rapidly-exploring Random Tree (RRT) algorithm. This function takes input parameters including the start and goal positions, obstacle map, and step size, and utilizes an improved version of the RRT algorithm, denoted as `improved_rrt`, to compute a path from the start to the goal.

The `fast_rrt` function encapsulates the core logic of the RRT algorithm, leveraging the efficiency and effectiveness of RRT-based methods in generating feasible paths in complex environments. By calling the `improved_rrt` function and checking for the existence of a path, the `fast_rrt` function provides a streamlined approach to path planning, returning a valid path if one is found, or `None` otherwise.

The Improved RRT Function, as depicted in Algorithm 3, plays a crucial role in path planning algorithms by enhancing the efficiency and effectiveness of the Rapidly-exploring Random Tree (RRT) method. This function takes input parameters including the start and goal positions, obstacle map, step size, and maximum iterations (`max_iter`) to control the exploration process.

The algorithm starts by initializing nodes with the start node and an explored set to keep track of visited configurations. It then iteratively explores the configuration space by sampling random points using the `fast_sample` function, finding the nearest node to each random point, and steering towards new configurations using the `steer` function. If the new configuration is valid and has not been explored, a new node is created and added to the tree.

The `improved_rrt` function incorporates various techniques such as random steering (`random_steer`) when direct steering fails, marking explored points to avoid redundant exploration, and tracing the path back from the goal configuration once reached. These strategies contribute to efficient and reliable path generation in complex environments with obstacles.

The Fast Sample Function described in Algorithm 4 plays a crucial role in the Rapidly-exploring Random Tree (RRT) algorithm by efficiently generating random points within the boundaries of the obstacle map. This function takes input parameters including the obstacle map, nodes in the RRT, and a threshold value to control random sampling.

The algorithm operates in a loop, continuously generating random points within the map boundaries. It checks if each generated point is valid and not within the explored area of any existing node in the RRT. If a valid point meeting these criteria is found, it is returned by the function.

Algorithm 3 Improved RRT Function

```
function IMPROVED_RRT($start$, $goal$, $obstacle\_map$, $step\_size$, $max\_iter$)
    Initialize nodes with the start node
    Initialize explored set
    for each iteration up to max_iter do
        $random\_point \leftarrow$ fast_sample($obstacle\_map, nodes, step\_size$)
        $nearest\_node \leftarrow$ find_nearest_node($nodes, random\_point$)
        $new\_point \leftarrow$ steer($random\_point, nearest\_node,$
            $step\_size, obstacle\_map$)
        if $new\_point$ is None then
            $new\_point \leftarrow$ random_steer($nearest\_node,$
                $step\_size, obstacle\_map$)
        end if
        if $new\_point$ is valid and not explored then
            Create new_node with $new\_point$ and $nearest\_node$ as parent
            Add new_node to nodes
            Mark $new\_point$ as explored
            if $new\_point$ reaches the goal then
                return trace_path(new_node)
            end if
            Draw the tree edge from $nearest\_node$ to $new\_node$ on the obstacle_map
            Draw the $new\_node$ on the obstacle_map
        end if
    end for
    return None
end function
```

Algorithm 4 Fast Sample Function

```
function FAST_SAMPLE($obstacle\_map, nodes, threshold$)
    while True do
        Generate a random point within the_map boundaries
        if the point is valid and not in the explored area of any node then
            return the point
        end if
    end while
end function
```

The Random Steer Function depicted in Algorithm 5 is a crucial component of path planning algorithms, particularly in the context of the Rapidly-exploring Random Tree (RRT) algorithm. This function is responsible for generating a random steering angle and calculating a new point based on the nearest node, step size, and the generated angle.

The algorithm starts by generating a random angle θ, which determines the direction of the steering from the nearest node. It then calculates a new point new_point using the nearest node, step size, and the generated angle θ. If the new point is

Algorithm 5 Random Steer Function

```
1: function RANDOM_STEER($nearest\_node, step\_size, obstacle\_map$)
2:     Generate a random angle $\theta$
3:     Calculate $new\_point$ using $nearest\_node$, $step\_size$, and $\theta$
4:     if $new\_point$ is valid then
5:         return $new\_point$
6:     else
7:         return None
8:     end if
9: end function
```

valid and does not collide with obstacles, it is returned by the function; otherwise, it returns `None`.

Algorithm 6 Trace Path Function

```
1: function TRACE_PATH($node$)
2:     Initialize an empty path
3:     while node exists do
4:         Add node's point to the path
5:         Set node to its parent
6:     end while
7:     Reverse the path and return it
8: end function
```

The Trace Path Function presented in Algorithm 6 is a fundamental component in path planning algorithms, specifically in the context of tree-based methods like the Rapidly-exploring Random Tree (RRT) algorithm. This function is responsible for tracing the path from a given node back to the start node or root of the tree.

The algorithm starts by initializing an empty path. It then iterates through the nodes starting from the given node and traverses upward towards the root of the tree by following each node's parent pointers. During this traversal, it adds each node's point to the path until it reaches the root node.

After completing the traversal, the algorithm reverses the path to ensure that it starts from the start node and ends at the goal node. Finally, the reversed path is returned by the function.

TABLE I
COMPARISONS FOR MAP-1

Step Size- 30	Normal - 0.36s Fast - 0.12s
Step Size- 20	Normal - 0.55s Fast - 0.27s
Step Size- 10	Normal - 24.82s Fast - 13.81s

V. RESULTS AND COMPARISIONS

A. MAP-1

Below are some visualized comparisions between Fast-RRT and Normal-RRT

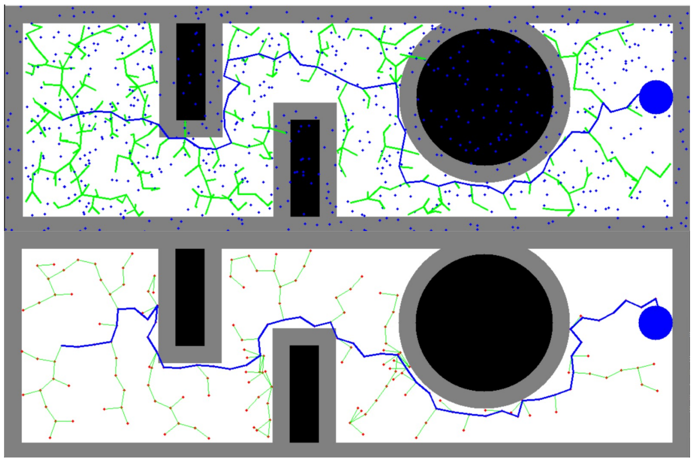

Fig. 5. Top Image: Normal RRT Bottom Image: Fast RRT.

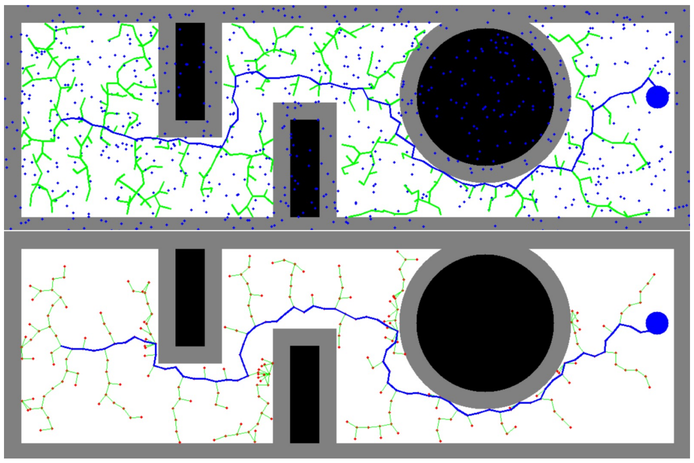

Fig. 6. Top Image: Normal RRT Bottom Image: Fast RRT.

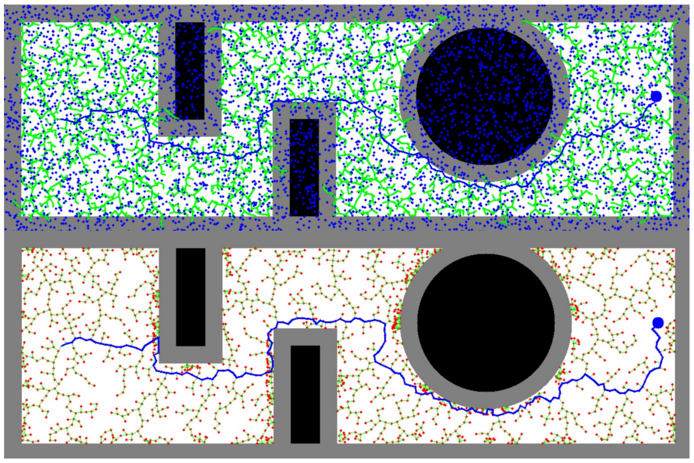

Fig. 7. Top Image: Normal RRT Bottom Image: Fast RRT.

TABLE II
COMPARISONS FOR MAP-2

Step Size- 30	Normal - 1.48s Fast - 0.57s
Step Size- 20	Normal - 1.55s Fast - 0.67s
Step Size- 10	Normal - 29.15s Fast - 5.02s

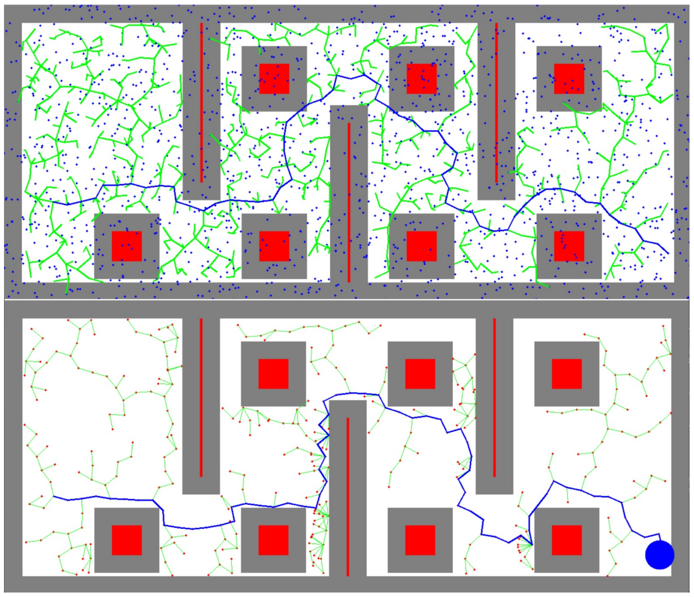

Fig. 8. Top Image: Normal RRT Bottom Image: Fast RRT.

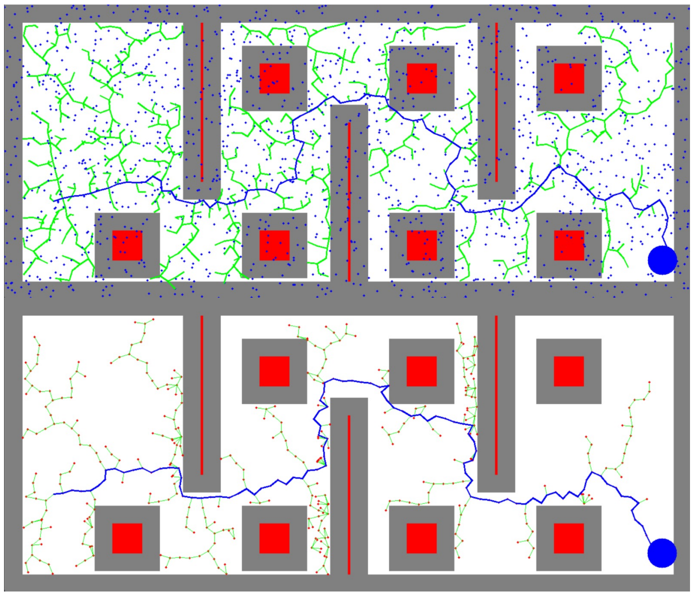

Fig. 9. Top Image: Normal RRT Bottom Image: Fast RRT.

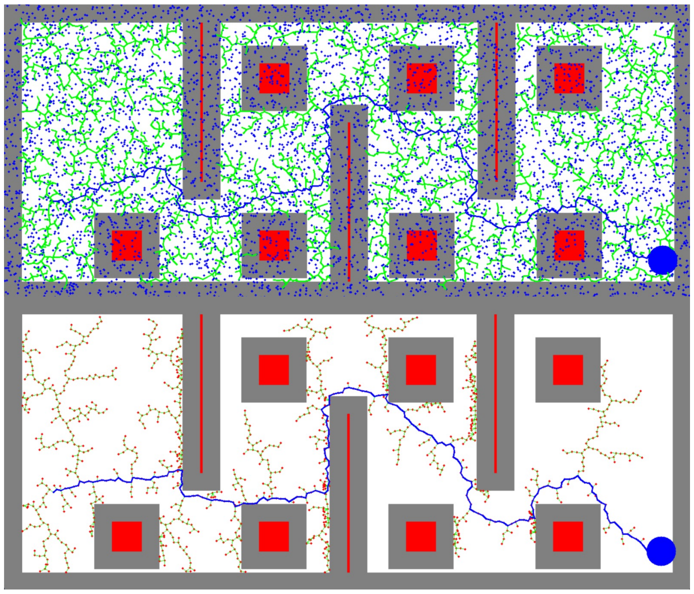

Fig. 10. Top Image: Normal RRT Bottom Image: Fast RRT.

VI. Conclusions

The Fast-RRT algorithm introduced offers significant improvements over the standard RRT algorithm for path planning in complex environments. The key conclusions are:

1) The Fast-Sampling strategy reduces redundant and invalid expansions, leading to faster convergence.
2) The Random-Steering mechanism enhances the algorithm's ability to handle narrow passages.
3) The algorithm demonstrates faster convergence and improved computational efficiency compared to the standard RRT.
4) The code provides a foundation for incorporating path optimization techniques to obtain near-optimal paths.
5) The visualization features aid in debugging and analysis of the algorithm's behavior.
6) The Fast-RRT algorithm is suitable for various robotics and autonomous systems applications.
7) Future enhancements, such as advanced optimization techniques and adaptive step sizes, could further improve performance and path quality.

In summary, the Fast-RRT algorithm offers a viable method for effective and efficient path planning, providing enhanced exploration capabilities and faster convergence in difficult situations. The given code can be used as a foundation for future study and advancement of sampling-based path-planning algorithms.

VII. References

[1] Z. Wu, Z. Meng, W. Zhao, and Z. Wu, "Fast-RRT: A RRT-based optimal path finding method," Applied Sciences (Switzerland), vol. 11, no. 24, Dec. 2021, doi: 10.3390/app112411777.

[2]Q. W. Zhang, L. X. Li, L. M. Zheng, and B. B. Li, "An Improved Path Planning Algorithm Based on RRT," in Proceedings - 2022 11th International Conference of Information and Communication Technology, ICTech 2022, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 149–152. doi: 10.1109/ICTech55460.2022.00037.