A Comparative Study of RRT, RRT*, BRRT, and BRRT* Algorithms in 2D and 3D Spaces

1st Gautham Ramkumar
College of Engineering
Northeastern University
Boston, United States of America
ramkumar.g@northeastern.edu

2nd Yash Sanjay Wakde
College of Engineering
Northeastern University
Boston, United States of America
wakde.y@northeastern.edu

3rd Adithya Rajendran
Khoury College of Computer Sciences
Northeastern University
Boston, United States of America
rajendran.ad@northeastern.edu

4th Kevin Jason
College of Engineering
Northeastern University
Boston, United States of America
jason.ke@northeastern.edu

***Abstract*—In this research, four path planning algorithms, Rapidly Exploring Random Tree (RRT), RRT *, Bidirectional RRT (BRRT) and Bidirectional RRT * (BRRT *)—are thoroughly compared in two- and three-dimensional environments. The study assesses the performance, effectiveness, and optimality of these algorithms in a variety of situations, such as dynamic impediments and congested areas. Using defined benchmarks and surroundings inspired by the real world, we implement and test these algorithms in Python, C++, and ROS1. The main performance indicators that we examine are the number of iterations, the length of the path, the execution time, and the smoothness of the path. Our results illustrate the trade-offs between path optimality and computational complexity, offering insights into the advantages, disadvantages, and suitability of each approach in various robotic navigation scenarios. For roboticists and designers of autonomous systems, this study provides insightful advice.**

I. Introduction

Path planning is a critical component in robotics, enabling autonomous systems to navigate efficiently through complex and constrained environments. This project focuses on implementing and comparing four key sampling-based path planning algorithms: Rapidly Exploring Random Tree (RRT), Rapidly Exploring Random Tree Star (RRT*), Bidirectional Rapidly Exploring Random Tree (BRRT), and Bidirectional Rapidly Exploring Random Tree Star (BRRT*). These algorithms have been chosen for their foundational importance in robotics and their practical utility in solving navigation problems in both two-dimensional and three-dimensional spaces.

Recent advancements in sampling-based path planning algorithms have shown significant improvements in efficiency and optimality. The RRT algorithm, known for its rapid exploration capabilities, has been enhanced through various modifications. For instance, Li et al. (2020) proposed the PQ-RRT* algorithm, which combines a potentially guided method with Quick-RRT* to achieve faster convergence and better initial path quality[1]. Similarly, Ye et al. (2021) introduced the AtBi-RRT algorithm, incorporating a gravitational algorithm with adaptive step and gravitational coefficient adjustment, demonstrating improved search efficiency in complex obstacle regions[1].

The motivation for this project stems from the need to understand the strengths and limitations of these algorithms in various scenarios, including obstacle-dense environments and varied configurations. While RRT provides a baseline for rapid exploration, RRT* introduces path optimization by minimizing path cost. BRRT accelerates exploration through bidirectional search, and BRRT* further enhances this by combining bidirectional exploration with path optimization.

Our choice to implement BRRT* is inspired by its potential to address the limitations of RRT, RRT*, and BRRT, particularly in large or constrained search spaces where bidirectional exploration can significantly reduce computation time and improve convergence. This aligns with recent research, such as the work by Wang et al. (2021), who proposed a continuous bidirectional Quick-RRT* (CBQ-RRT*) algorithm, demonstrating improved path planning efficiency for mobile robots in complex orchard environments[1].

Furthermore, the growing demand for efficient and reliable navigation solutions in modern robotics applications necessitates a comprehensive comparison of these algorithms. Recent studies, like the one conducted by Tengesdal et al. (2024), have highlighted the importance of such comparative analyses, particularly in specific domains like maritime trajectory planning[2].

By comparing these algorithms under the same experimental conditions, we aim to evaluate their performance in terms of computational efficiency, path length, execution time, and smoothness. This project builds upon recent research trends, such as the hybrid approaches discussed by Wang et al. (2021), who combined reinforcement learning with BRRT to enhance path planning capabilities[5].

This project is based on the framework presented in the research paper Optimal Path Planning Using Bidirectional

Rapidly Exploring Random Tree Star-Dynamic Window Approach (BRRT-DWA) with Adaptive Monte Carlo Localization (AMCL) for Mobile Robots. Although we did not implement the Dynamic Window Approach (DWA) or AMCL[4], the insights from this paper guided our understanding of the capabilities of BRRT* and inspired our focus on this algorithm. Through our implementation and analysis, we aim to demonstrate the practical trade-offs between these algorithms and provide a detailed assessment of their suitability for various robotic navigation tasks. This study will contribute to the ongoing research in optimal path planning for autonomous systems, addressing the challenges highlighted in recent literature and providing insights for future developments in this rapidly evolving field.

II. Theoretical Foundations of Path-Planning Algorithms

This section discusses the theoretical foundations of four advanced path-planning algorithms implemented in this project: Rapidly Exploring Random Tree (RRT), Rapidly Exploring Random Tree Star (RRT*), Bidirectional Rapidly Exploring Random Tree (BRRT), and Bidirectional Rapidly Exploring Random Tree Star (BRRT*). Each algorithm is described in terms of its underlying principles, mathematical formulations, advantages, and limitations, with a focus on recent developments and applications.

A. Rapidly Exploring Random Tree (RRT)

The Rapidly Exploring Random Tree (RRT) algorithm represents a sampling-based path-planning methodology designed to navigate complex configuration spaces efficiently. Developed to address the challenges of high-dimensional path planning, RRT introduces a probabilistic approach to exploring configuration space through intelligent random sampling and tree expansion strategies[6].

The fundamental mathematical formulation of RRT can be described through several key equations and principles. The algorithm's core objective is to incrementally construct a tree that explores the configuration space by connecting random samples to the existing tree structure. The random sampling process follows a uniform distribution across the free configuration spaces. For each iteration, the algorithm selects the nearest existing tree node by using a distance metric, typically Euclidean distance:

$$q_{\text{near}} = \arg\min_{q_{\text{node}} \in T} \text{distance}(q_{\text{rand}}, q_{\text{node}}) \tag{1}$$

Recent advancements in RRT have focused on improving its efficiency and applicability in various domains. For instance, Li et al.[1] proposed the PQ-RRT* algorithm, which combines a potentially guided method with Quick-RRT* to achieve faster convergence and better initial path quality.

B. Rapidly Exploring Random Tree Star (RRT)*

The Rapidly Exploring Random Tree Star (RRT*) algorithm emerges as a sophisticated enhancement to the original RRT, introducing a crucial optimization mechanism that guarantees asymptotic path optimality. By implementing a sophisticated rewiring process, RRT* systematically refines the tree structure to minimize path costs progressively[6].

The algorithm's core innovation lies in its neighbor search and rewiring strategy. For a newly added node, the algorithm identifies neighboring nodes within a radius and selects a parent node that minimizes the total cost from the start node. The cost minimization can be formally expressed as:

$$q_{\text{parent}} = \arg\min_{q_{\text{parent}}} \left(C(q_{\text{start}} \to q_{\text{parent}}) + \text{distance}(q_{\text{parent}}, q_{\text{new}}) \right) \tag{2}$$

Recent studies have focused on enhancing RRT* for specific applications. For example, Ye et al.[1] introduced the AtBi-RRT algorithm, incorporating a gravitational algorithm with adaptive step and gravitational coefficient adjustment, demonstrating improved search efficiency in complex obstacle regions.

C. Bidirectional Rapidly Exploring Random Tree (BRRT)

The Bidirectional Rapidly Exploring Random Tree (BRRT) algorithm represents an innovative approach to path planning that substantially improves exploration efficiency by simultaneously growing trees from both start and goal configurations. This dual-tree strategy fundamentally transforms the path discovery process, reducing computational complexity and convergence time.

BRRT maintains two distinct trees originating from the start configuration and initialized from the goal configuration. The connection probability between the start-tree and goal-tree is governed by a distance threshold ϵ. Mathematically, the connection condition is defined as:

$$\text{connect} = \left(\min \left(\text{distance}(q_{\text{start\_node}}, q_{\text{goal\_node}}) \right) \leq \epsilon \right) \tag{3}$$

Recent research has focused on improving BRRT's performance in dynamic environments. Wang et al.[5] proposed a continuous bidirectional Quick-RRT* (CBQ-RRT*) algorithm, demonstrating improved path planning efficiency for mobile robots in complex orchard environments.

D. Bidirectional Rapidly Exploring Random Tree Star (BRRT)*

The Bidirectional Rapidly Exploring Random Tree Star (BRRT*) algorithm represents the pinnacle of path-planning techniques, synergistically combining the bidirectional exploration strategy of BRRT with the optimal path refinement process of RRT*[2]. This sophisticated approach ensures efficient and asymptotically optimal path planning across complex configuration spaces.

BRRT* extends the bidirectional paradigm by applying rewiring processes to both the start-tree and goal-tree. The mathematical formulation involves minimizing the total path cost across both trees, expressed as:

$$C(\pi) = \min\left(C(T_{\text{start}}) + C(T_{\text{goal}})\right) \tag{4}$$

Recent advancements in BRRT* have focused on its application in specific domains. For instance, Tengesdal et al.[2] highlighted the importance of comparative analyses of path-planning algorithms, particularly in specific domains like maritime trajectory planning.

These algorithms represent the cutting edge of path-planning techniques, each offering unique advantages and trade-offs in terms of efficiency, optimality, and computational complexity. Their ongoing development and application across various domains continue to push the boundaries of autonomous navigation and robotics.

III. Problem Statement

The primary objective of this project is to implement and analyze the performance of four sampling-based path planning algorithms: Rapidly Exploring Random Tree (RRT), Rapidly Exploring Random Tree Star (RRT*), Bidirectional Rapidly Exploring Random Tree (BRRT), and Bidirectional Rapidly Exploring Random Tree Star (BRRT*). The study is conducted in both two-dimensional (2D) and three-dimensional (3D) environments to assess their ability to generate collision-free paths that connect a given start position to a goal position. The focus is on understanding the trade-offs between computational efficiency, path length, and path smoothness across different algorithms and environmental configurations.

A. Mathematical Formulation

1) **Configuration Space ($\mathcal{C}$)**:
   - $\mathcal{C}$ represents the set of all possible configurations (positions and orientations) of the robot.
   - For a 2D environment, $\mathcal{C} \subseteq \mathbb{R}^2$.
   - For a 3D environment, $\mathcal{C} \subseteq \mathbb{R}^3$.
2) **Obstacle Space ($\mathcal{C}_{\text{obs}}$)**:
   $$\mathcal{C}_{\text{obs}} \subset \mathcal{C}$$
   represents the subset of configurations that result in a collision with obstacles.
3) **Free Space ($\mathcal{C}_{\text{free}}$)**:
   $$\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}$$
   is the set of all collision-free configurations.
4) **Start and Goal Configurations**:
   $$q_{\text{start}} \in \mathcal{C}_{\text{free}}, \quad q_{\text{goal}} \in \mathcal{C}_{\text{free}}$$
   where q_{start} is the initial position of the robot and q_{goal} is the desired final position.
5) **Path (π)**:
   $$\pi : [0,1] \to \mathcal{C}_{\text{free}}$$
   is a continuous function such that $\pi(0) = q_{\text{start}}$ and $\pi(1) = q_{\text{goal}}$. The path must lie entirely within $\mathcal{C}_{\text{free}}$, i.e., $\pi(s) \in \mathcal{C}_{\text{free}}, \forall s \in [0,1]$.
6) **Cost Function ($\mathcal{C}(\pi)$)**: The cost of a path π is defined as:
   $$\mathcal{C}(\pi) = \int_0^1 c(\pi(s))\, ds$$
   where $c(\pi(s))$ represents the cost rate at configuration $\pi(s)$, typically proportional to path length or traversal time.
7) **Optimization Objective**: Find a collision-free path π that minimizes the cost:
   $$\pi^* = \underset{\pi}{\operatorname{argmin}}\, \mathcal{C}(\pi)$$

B. Problem Definition

Given:

- A configuration space $\mathcal{C}$ with defined obstacles $\mathcal{C}_{\text{obs}}$ and free space $\mathcal{C}_{\text{free}}$.
- Start and goal configurations $q_{\text{start}}, q_{\text{goal}} \in \mathcal{C}_{\text{free}}$.

Find:

- A collision-free path $\pi : [0,1] \to \mathcal{C}_{\text{free}}$ such that $\pi(0) = q_{\text{start}}$ and $\pi(1) = q_{\text{goal}}$.

Objective:

- Compare and analyze the performance of RRT, RRT*, BRRT, and BRRT* in generating feasible and optimal paths.
- Evaluate performance metrics, including:
  1) Path length ($\mathcal{C}(\pi)$).
  2) Computational efficiency (execution time and number of iterations).
  3) Smoothness of the resulting path.

Constraints:

- The algorithms are tested in multiple environments, including:
  1) Three distinct 2D maps with varying obstacle densities.
  2) A 3D environment to evaluate scalability in higher-dimensional spaces.

IV. Methodology

This project focuses on implementing and comparing four sampling-based path planning algorithms: Rapidly Exploring Random Tree (RRT), Rapidly Exploring Random Tree Star (RRT*), Bidirectional Rapidly Exploring Random Tree (BRRT), and Bidirectional Rapidly Exploring Random Tree Star (BRRT*). The methodology includes a step-by-step implementation of each algorithm, highlighting their mathematical foundations, computational processes, and parameter configurations. This section ensures reproducibility by detailing the logic, equations, and parameter definitions for each algorithm.

A. Rapidly Exploring Random Tree (RRT)

The RRT algorithm incrementally builds a tree by sampling the free space and connecting new points to the existing tree.

1) **Initialization**:
   $$T = (V, E), \quad V = \{q_{\text{start}}\}, \quad E = \emptyset$$

Define the maximum number of iterations $N_{\max}$ and step size η.

2) **Sampling**: Sample a random configuration:

$$q_{\text{rand}} \in \mathcal{C}_{\text{free}}$$

3) **Nearest Node Selection**: Find the nearest node q_{near} in the tree:

$$q_{\text{near}} = \underset{q \in V}{\operatorname{argmin}} \|q_{\text{rand}} - q\|$$

4) **Steer**: Compute a new node q_{new} by moving from q_{near} towards q_{rand} by a step η:

$$q_{\text{new}} = q_{\text{near}} + \eta \cdot \frac{q_{\text{rand}} - q_{\text{near}}}{\|q_{\text{rand}} - q_{\text{near}}\|}$$

5) **Collision Check**: Ensure that the edge $(q_{\text{near}}, q_{\text{new}})$ lies in $\mathcal{C}_{\text{free}}$.

6) **Add Node and Edge**: If valid, update the tree:

$$V = V \cup \{q_{\text{new}}\}, \quad E = E \cup \{(q_{\text{near}}, q_{\text{new}})\}$$

7) **Termination**: Repeat until q_{goal} is reached (within a threshold ϵ) or $N_{\max}$ iterations are completed.

B. Rapidly Exploring Random Tree Star (RRT*)

The RRT* algorithm improves RRT by rewiring the tree to optimize path cost.

1) **Neighbor Search**: Identify nodes within a radius r of q_{new}:

$$\mathcal{N} = \{q \in V \mid \|q_{\text{new}} - q\| \leq r\}$$

2) **Choose Optimal Parent**: Select $q_{\text{parent}} \in \mathcal{N}$ that minimizes the cost:

$$q_{\text{parent}} = \underset{q \in \mathcal{N}}{\operatorname{argmin}} \mathcal{C}(q_{\text{start}}, q) + \mathcal{C}(q, q_{\text{new}})$$

3) **Rewire Neighbors**: For each $q_{\text{neighbor}} \in \mathcal{N}$, check if rewiring through q_{new} reduces cost:

$$\mathcal{C}(q_{\text{start}}, q_{\text{new}}) + \mathcal{C}(q_{\text{new}}, q_{\text{neighbor}}) < \mathcal{C}(q_{\text{start}}, q_{\text{neighbor}})$$

If true, update the tree:

$$E = (E \setminus \{(q_{\text{old}}, q_{\text{neighbor}})\}) \cup \{(q_{\text{new}}, q_{\text{neighbor}})\}$$

C. Bidirectional Rapidly Exploring Random Tree (BRRT)

BRRT grows two trees, one from q_{start} and one from q_{goal}, and attempts to connect them.

1) **Initialize Two Trees**:

$$T_{\text{start}} = (V_{\text{start}}, E_{\text{start}}), \quad T_{\text{goal}} = (V_{\text{goal}}, E_{\text{goal}})$$

$$V_{\text{start}} = \{q_{\text{start}}\}, \quad V_{\text{goal}} = \{q_{\text{goal}}\}$$

2) **Alternate Tree Expansion**: Grow T_{start} towards q_{rand} using the RRT procedure, then swap roles to grow T_{goal}.

3) **Connect Trees**: Attempt to connect the nearest nodes between T_{start} and T_{goal}. If successful, terminate.

D. Bidirectional Rapidly Exploring Random Tree Star (BRRT*)

BRRT* integrates the RRT* rewiring mechanism into BRRT.

1) **Rewire Each Tree**: Apply the RRT* rewiring process after adding q_{new} in both T_{start} and T_{goal}.

2) **Bidirectional Reconnection**: Check all neighbors in T_{start} and T_{goal} when attempting to connect the two trees, ensuring minimal path cost.

E. Parameter Definitions

- η: Step size for tree expansion.
- r: Radius for neighbor search (used in RRT* and BRRT*).
- ϵ: Connection threshold for goal or bidirectional tree termination.
- $N_{\max}$: Maximum number of iterations.

V. DETAILS OF IMPLEMENTATION

A. Setup and Implementation on 2D Environments

The research implemented four path planning algorithms—RRT, RRT*, BRRT, and BRRT*—using Python and Jupyter Notebook as the primary development environment. The experimental framework utilized three distinct 2D occupancy maps to comprehensively evaluate algorithm performance. OpenCV library was employed to facilitate visualization and node marking on the occupancy grid maps, enabling precise spatial representation and path planning analysis.

B. Performance Metrics

1) Path Length Calculation: Path length represents a critical metric for evaluating the efficiency of path planning algorithms. The metric quantifies the total geometric distance traversed from the start to the goal configuration. Mathematically, path length is computed as the cumulative Euclidean distance between consecutive path points. The mathematical formulation for path length is defined as:

$$\text{Path Length} = \sum_{i=0}^{n-2} \text{Euclidean Distance}(path[i], path[i+1])$$

Where:

- n represents the total number of points in the path
- $path[i]$ denotes the current point
- $path[i+1]$ represents the subsequent point

The Euclidean distance between two points (x_1, y_1) and (x_2, y_2) is calculated using the fundamental distance formula:

$$\text{Euclidean Distance}(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This metric provides a quantitative measure of path efficiency, with shorter distances indicating more optimal trajectories.

2) Computational Time Analysis: Computational time measurement captures the algorithmic efficiency and computational complexity of path planning strategies. The time taken is computed as the temporal difference between the algorithm's initiation and completion. The time calculation is formally expressed as:

$$\text{Time Taken} = \text{End Time} - \text{Start Time}$$

Utilizing Python's `time.time()` function, which returns the current time in seconds since the epoch, enables precise temporal measurement. This metric provides insights into the computational overhead associated with different path planning approaches.

3) Path Smoothness Evaluation: Path smoothness quantifies the geometric continuity and angular variation between consecutive path segments. The metric employs statistical analysis of angular transitions to assess trajectory quality. The angular calculation between three consecutive points a, b, and c utilizes the cosine rule:

$$\theta = \arccos\left(\frac{(b-a)\cdot(c-b)}{|b-a||c-b|}\right)$$

Where:

- a, b, and c represent sequential path points
- $(b-a)$ and $(c-b)$ define consecutive path segment vectors

Path smoothness is ultimately determined by computing the standard deviation of these calculated angles:

$$\text{Smoothness} = \text{std}(angles)$$

A lower smoothness value indicates more continuous and geometrically consistent path trajectories.

4) Iteration Count Measurement: The iteration count metric quantifies the algorithmic exploration complexity by tracking the number of sampling and tree expansion attempts. This measurement provides insight into the algorithm's convergence characteristics and computational efficiency. Mathematically, the iteration count is expressed as:

Number of Iterations = Total number of loop iterations in the algorithm

The metric is bounded by a predefined maximum iteration limit, ensuring computational feasibility and preventing infinite exploration scenarios.

C. Setup and Implementation on 3D Environments

The research undertook a sophisticated implementation of advanced path planning algorithms—Rapidly Exploring Random Tree (RRT), Rapidly Exploring Random Tree Star (RRT*), Bidirectional Rapidly Exploring Random Tree (BRRT), and Bidirectional Rapidly Exploring Random Tree Star (BRRT*)—utilizing C++ as the primary programming language within the Robot Operating System (ROS) Noetic framework. This technological ecosystem provides a robust and flexible environment for developing complex robotic navigation algorithms.

D. Development Environment and Tools

The selection of C++ and ROS-Noetic represents a strategic choice for high-performance robotic path planning research. C++ offers exceptional computational efficiency and low-level system control, while ROS-Noetic provides a comprehensive middleware framework that facilitates seamless integration of robotic perception, computation, and actuation components. The experimental methodology centered on a detailed three-dimensional (3D) mapping approach, enabling comprehensive algorithmic evaluation across spatially complex environments. Rviz, a powerful ROS visualization tool, was instrumental in facilitating node visualization, spatial representation, and real-time path planning analysis. This visualization capability allows researchers to gain profound insights into algorithmic behavior and path generation strategies.

E. Performance Metrics

1) Path Length Calculation: Path length serves as a fundamental metric for quantifying trajectory efficiency and algorithmic performance. The metric provides a precise geometric representation of the total distance traversed from the start to the goal configuration. The mathematical formulation for path length is rigorously defined as:

$$\text{Path Length} = \sum_{i=0}^{n-2} \text{Euclidean Distance}(path[i], path[i+1])$$

Where:

- n represents the total number of points in the generated path
- $path[i]$ denotes the current spatial coordinate
- $path[i+1]$ represents the subsequent spatial coordinate

The three-dimensional Euclidean distance between points (x_1, y_1, z_1) and (x_2, y_2, z_2) is calculated using the extended distance formula:

$$\text{Euclidean Distance}(p_1, p_2) = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2}$$

This metric offers a quantitative assessment of path optimality, with shorter distances indicating more efficient trajectories through the three-dimensional configuration space.

2) Computational Time Analysis: Computational time measurement provides critical insights into the algorithmic efficiency and computational complexity of the implemented path planning strategies. The temporal performance metric captures the total duration required for path generation. The time calculation is formally expressed as:

$$\text{Time Taken} = \text{End Time} - \text{Start Time}$$

Utilizing high-precision timing mechanisms within the C++ environment enables accurate temporal measurements, providing researchers with a comprehensive understanding of computational overhead associated with different path planning approaches.

F. Experimental Significance

By comprehensively analyzing path length, computational time, smoothness, and iteration count, researchers can systematically compare and evaluate the performance characteristics of different path planning algorithms. This multi-dimensional assessment enables nuanced understanding of algorithmic strengths and limitations across varied environmental configurations. The proposed evaluation methodology provides a robust framework for quantitative performance comparison, facilitating informed algorithm selection based on specific navigational requirements and computational constraints.

VI. RESULTS

The experimental methodology employed three distinct maps, each featuring three unique start and goal node configurations. This strategic approach enables a multifaceted investigation of algorithmic performance across varied spatial environments. By utilizing multiple start and goal node combinations, the research methodology mitigates potential bias and provides a more comprehensive understanding of algorithmic behavior under diverse navigational scenarios.

The path planning algorithms were implemented with a consistent set of configuration parameters, ensuring comparative consistency and methodological rigor:

1) Maximum Iterations: 2500 iterations were established as the maximum computational limit for each algorithmic execution. This parameter balances computational efficiency with comprehensive exploration of the configuration space, providing sufficient opportunity for path discovery while preventing excessive computational overhead.

2) Step Size: A step size of 10 units was uniformly applied across all implemented algorithms. The step size parameter governs the incremental expansion of the exploration tree, determining the granularity of spatial sampling and trajectory generation.

3) Goal Tolerance: A goal tolerance of 20 units was defined, establishing a permissible proximity threshold for considering a path successfully completed. This parameter introduces a practical margin of convergence, acknowledging the inherent challenges of precise point-to-point navigation in complex spatial environments.

4) Rewire Radius: For optimization-oriented algorithms RRT* and BRRT*, a rewire radius of 20 units was implemented. This parameter controls the neighborhood search radius during the path refinement process, enabling local optimization of trajectory segments while maintaining computational efficiency.

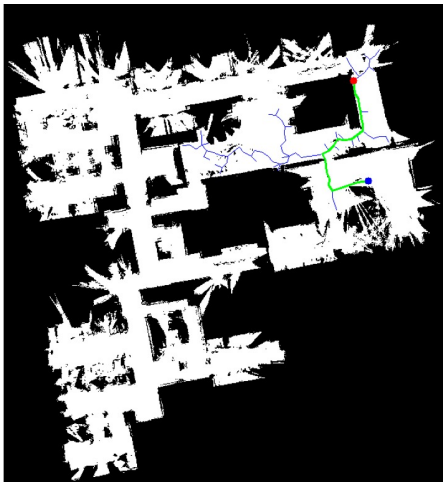

Fig. 1. Path generated by RRT in 2D.

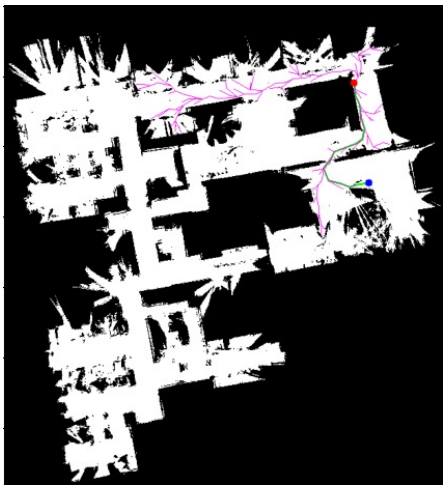

Fig. 2. Path generated by RRT* in 2D.

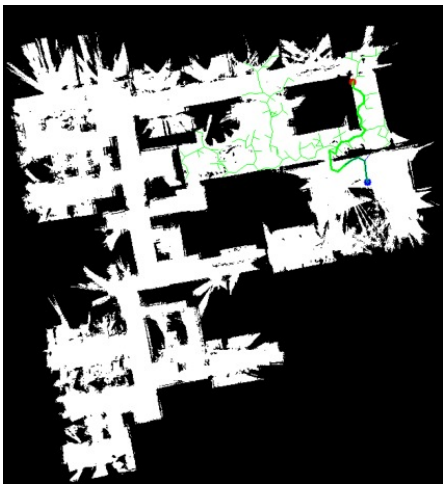

Fig. 3. Path generated by BRRT in 2D

3D Implementation Results

The results demonstrate the performance of four path planning algorithms: RRT, RRT*, BRRT, and BRRT*. Among these, RRT showed the fastest initial path generation time (0.000266901 seconds) but produced the longest and least efficient paths (47.9945), reflecting its lack of optimization. RRT* improved significantly by incorporating a rewiring step, resulting in optimized paths with a shorter length of 33.4769, albeit at a higher computational cost (0.00107536 seconds). BRRT introduced bidirectional exploration, achieving faster planning time (0.00012664 seconds) and shorter paths (33.5197) than RRT, but without guaranteed optimality. BRRT* emerged as the best algorithm, combining the fast convergence of bidirectional search with the optimization capabilities of RRT*, resulting in the shortest and most optimal paths (33.4732) with the fastest planning time (0.000088347 seconds). This balance of speed and path quality makes BRRT* ideal for real-time robotic applications in complex environments, outperforming the other algorithms in terms of both computational efficiency and navigation precision.

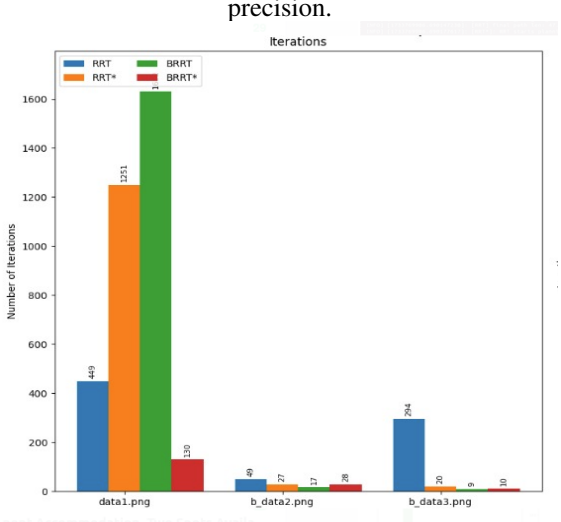

Fig 4. Iteration comparison of all four 2D algorithms

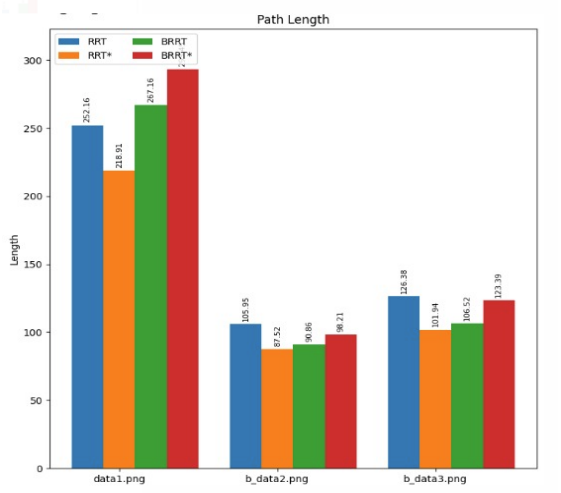

Fig 5. Length comparison of all four 2D algorithms

Fig 6. Path generated by BRRT* in 2D.

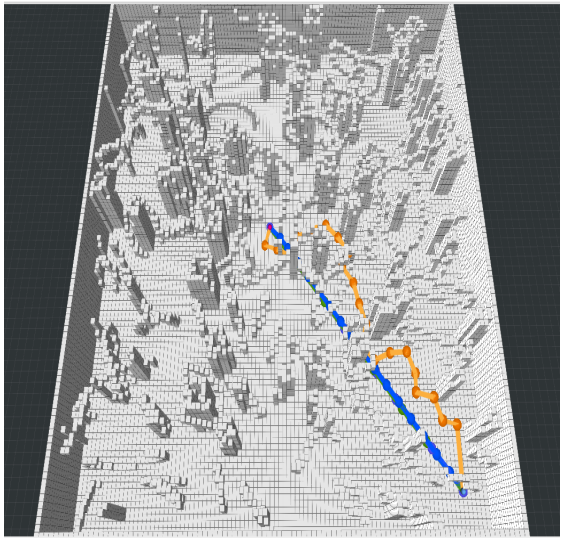

Fig 7. Path generated by all algorithms in 3D.

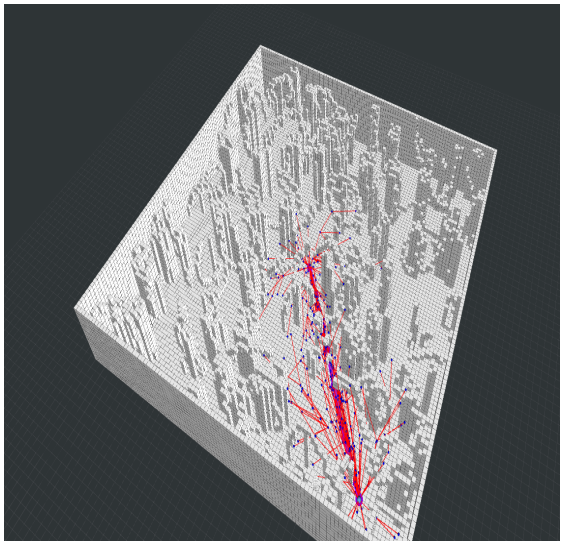

Fig 8. Path generated by BRRT* in 3D.

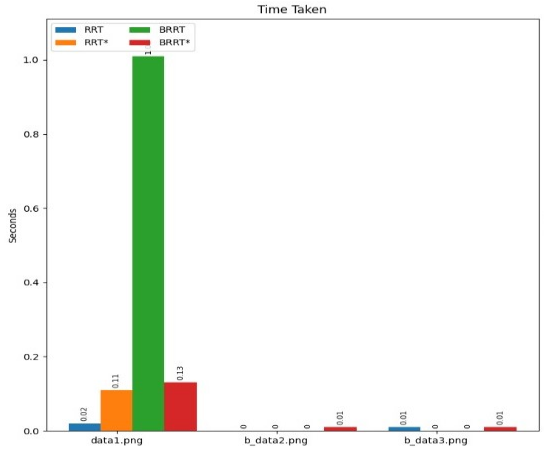

Fig 9. Time comparison of all four 2D algorithms

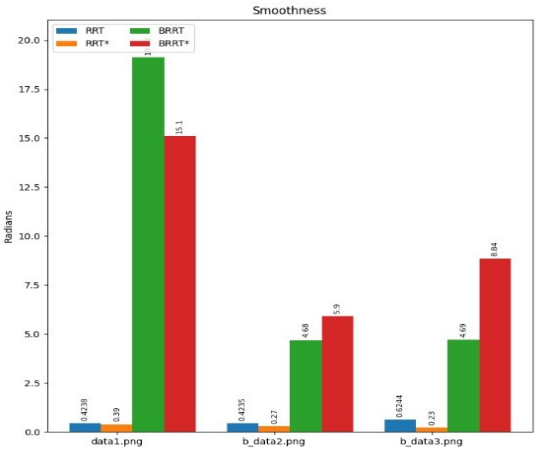

Fig 10. Smoothness comparison of all four 2D algorithms

ALGORITHM	PATH LENGTH	TIME TAKEN (Seconds)
RRT	47.8634	0.000266901
RRT*	39.9626	0.00107536
BRRT	33.5197	0.000012664
BRRT*	33.4732	0.000088347

Fig 11. Result of comparing all four algorithms in 3D

VII. CONCLUSION

The comparative analysis of 2D and 3D implementations highlights the advantages of the BRRT* algorithm over traditional path-planning algorithms like RRT, RRT*, and BRRT. BRRT* excels in efficiency, trajectory smoothness, and overall practicality, making it the optimal choice for real-world applications.

In 2D scenarios, BRRT* achieves convergence with significantly fewer iterations, especially in complex datasets. While RRT* might yield slightly shorter paths, BRRT* generates comparable-length trajectories at a much lower computational cost and produces smoother paths essential for fluid navigation. Its speed and efficiency make it ideal for real-time applications, outperforming RRT, RRT*, and BRRT, which struggle with longer paths and smoothness.

In 3D implementations, BRRT* achieves a near-optimal path length of 33.4732 units, comparable to BRRT but with smoother trajectories and faster computation times. This advantage makes it particularly suitable for time-sensitive applications in robotics and autonomous navigation. Although RRT* optimizes for path length relative to RRT, it falls short in computational efficiency. Similarly, while BRRT performs well, it cannot match BRRT* in efficiency.

In conclusion, BRRT* emerges as the optimal path-planning algorithm, balancing efficiency (iterations and computation time), effectiveness (path length), and practicality (smoothness). Its consistent performance across both 2D and 3D environments strongly supports its adoption as the superior solution for path-planning tasks, meeting the demands of real-world scenarios with precision, speed, and adaptability.

VIII. CODE AND DATA AVAILABILITY

The data and code implementation for this project are publicly available on the following GitHub repositories:

- https://gitlab.com/ramkumar.g/mobile-robotics
- https://gitlab.com/wakde.y/mobile$_robotics_project$

These repositories contain the necessary code, algorithms, and datasets used in our comparative study of RRT, RRT*, BRRT, and BRRT* algorithms in 2D and 3D spaces. Interested researchers and practitioners can access, review, and utilize these resources for further study, replication, or extension of our work.

IX. ACKNOWLEDGMENT

We extend our heartfelt thanks to our professor, Dr. David M. Rosen, for his invaluable guidance, expertise, and unwavering support throughout the course. His insightful lectures and the opportunity to apply in-class concepts through this project work have been instrumental in deepening our understanding of path planning algorithms and their practical applications.

We are deeply grateful to the College of Engineering at Northeastern University for providing us with the necessary resources, facilities, and academic environment conducive to conducting this research. The support from the college's faculty and staff has been crucial in overcoming various challenges encountered during the project.

A special thanks to our fellow students and peers who participated in our discussions and provided valuable feedback during various stages of the project. Their diverse perspectives and critical insights have undoubtedly enhanced the quality of our work and fostered a collaborative learning environment.

We would also like to acknowledge the contributions of the open-source community and researchers in the field of robotics and path planning. Their publicly available resources, libraries, and research papers have been invaluable in our implementation and analysis of the RRT, RRT*, BRRT, and BRRT* algorithms.

This project has been a collaborative effort, and its success is a testament to the support and contributions of all those mentioned above and many others who have assisted in ways both big and small. We are truly grateful for this enriching experience and the opportunity to contribute to the field of robotics and path planning.

REFERENCES

[1] Ye, L., Li, J., & Li, P. (2024). Improving path planning for mobile robots in complex orchard environments: the continuous bidirectional Quick-RRT* algorithm. *Frontiers in Plant Science*, 15, 1337638.

[2] Tengesdal, T., Pedersen, T. A., Johansen, T. A. (2024). A Comparative Study of Rapidly-exploring Random Tree Algorithms Applied to Ship Trajectory Planning and Behavior Generation. arXiv preprint arXiv:2403.01194.

[3] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *arXiv preprint arXiv:1703.08944*, 2017.

[4] W. Ayalew, A. A. Gebrewahd, and A. A. Gebrehiwot, "Optimal path planning using bidirectional rapidly exploring random tree star-dynamic window approach (BRRT*-DWA) with adaptive Monte Carlo localization (AMCL) for mobile robot," *Engineering Research Express*, vol. 6, no. 3, p. 03521, 2024.

[5] J. Wang, K. Hirota, X. Wu, Y. Dai, and Z. Jia, "Hybrid Bidirectional Rapidly Exploring Random Tree Path Planning Algorithm with Reinforcement Learning," J. Adv. Comput. Intell. Intell. Inform., Vol.25 No.1, pp. 121-129, 2021..

[6] Luo, S., Zhang, M., Zhuang, Y., Ma, C., Li, Q. (2023). A survey of path planning of industrial robots based on rapidly exploring random trees. Frontiers in Neurorobotics, 17, 1268447.