Implementation and Testing of Quick – RRT* Path Planning on TurtleBot3

Sai Jagadeesh Muralikrishnan
Robotics Graduate Student
University of Maryland
College Park, Maryland
jagkrish@umd.edu

Varun Lakshmanan
Robotics Graduate Student
University of Maryland
College Park, Maryland
varunl11@umd.edu

Nitish Ravishankar Raveendran
Robotics Graduate Student
University of Maryland
College Park, Maryland
rrnitish@umd.edu

***Abstract*—In this experimental study, we present the implementation and evaluation of the Quick-RRT* (Q-RRT*) algorithm on a Turtlebot3 waffle robot within a custom simulation environment in Gazebo. Q-RRT*, which is a derivative of the standard RRT* algorithm, that aims to optimize robotic navigation through enhanced initial path solutions and faster convergence rate towards optimal paths. Our implementation goes beyond existing methods by including an improved parent selection algorithm that takes into account both direct lineage and ancestral vertices, resulting in a faster and more efficient path finding procedure. We conducted thorough performance comparisons between Q-RRT*, RRT*, and Informed RRT* in a Gazebo ROS2 simulation, demonstrating Q-RRT*'s higher efficiency and dependability. The results demonstrates that Q-RRT* significantly outperforms its predecessors in both computational overhead and path optimality, asserting its potential as a formidable tool in advanced robotic navigation systems.**

***Index Terms*—Quick-RRT*, Robotics Navigation,Sampling-based Motion Planning, Algorithm Performance Evaluation**

I. Introduction

Path planning is an essential aspect of robotics that allows autonomous robots to explore complex situations efficiently. Among the several algorithms created for this purpose, Rapidly Exploring Random Tree (RRT) and its optimal form, RRT*, have gained popularity. However, these algorithms frequently require improvements to meet real-world requirements such as faster convergence and optimum pathfinding. This paper introduces Q-RRT*, which expands the set of possible parent vertices for each new node, extending not only to those within a proximity hypersphere centered on the new node, as in traditional RRT*, but also including an ancestry up to a user-defined depth. This enhancement significantly improves the quality of initial solutions and the rate of convergence to the optimal path, making Q-RRT* essential for autonomous systems navigating environments riddled with obstacles. By conducting extensive simulations within a ROS2 environment in Gazebo, this experimental study illustrates the comparative performance of Q-RRT* against traditional RRT* and Informed RRT* algorithms, establishing its superiority in both initial solution quality and convergence speed.

II. Literature Review

The literature on path planning for autonomous systems is substantial, with a primary focus on the evolution of the Rapidly Exploring Random Tree (RRT) and its variants, which are useful in complex and dynamic situations. The basic RRT algorithm, established by LaValle in 1998, and its optimized equivalent, RRT*, introduced by Karaman and Frazzoli in 2011, are fundamental to this subject. RRT* refines the RRT technique by ensuring asymptotic optimality, laying the groundwork for future advancements.

Following versions, such as Informed-RRT* and RRT*-Smart, use sophisticated sampling procedures and path optimization approaches to increase the efficiency and speed of convergence to the ideal path, as emphasized by Gammell et al. (2014) and Nasir et al. 2013. These methods solve RRT*'s computational inefficiencies by focusing the search on favorable portions of the configuration space, therefore drastically lowering exploration overhead.

The proposed Quick-RRT* method builds upon these foundations by enhancing the parent selection process within the RRT* framework. Quick-RRT* aims to deliver faster convergence and a higher quality initial solution path by expanding the collection of likely parent nodes beyond a predetermined radius and considering their ancestry. This method not only preserves RRT*'s probabilistic completeness and asymptotic optimality, but it also introduces a new dimension in the form of ancestral lineage consideration, which has gotten little attention in the existing literature.

III. Background

This part formalizes the motion planning problem and goes deeper into the RRT* method, which is the foundation of the proposed Quick-RRT* algorithm. We also investigate Informed-RRT* to provide a clear understanding for the comparison analysis reported in Section 4. Our implementations of these principles use the Quick-RRT* algorithm on a Turtlebot3 waffle robot, with the goal of navigating efficiently in a world and performing consistently under a variety of different optimal parameters used to simulate in ROS2 within the Gazebo.

A. Problem Definition

Let $\mathcal{X} = (0,1)^d$ represent the configuration space, $\mathcal{X}_{\text{obs}}$ the obstacle region, and $\mathcal{X}_{\text{free}} = \text{cl}(\mathcal{X} \setminus \mathcal{X}_{\text{obs}})$ the closure of the free space. The initial state $\mathcal{X}_{\text{init}} \in \mathcal{X}_{\text{free}}$ and the goal area $\mathcal{X}_{\text{goal}} \subset$

$\mathcal{X}_{\text{free}}$ define the path planning problem. A path $\sigma : [0, 1] \to \mathcal{X}$ of bounded variation is deemed feasible if it is collision-free for every $t \in [0, 1]$, $\sigma(t) \in \mathcal{X}_{\text{free}}$.

Definition 1: Feasible Path Planning. A feasible path planning problem requires finding a collision-free path σ, where $\sigma(0) = \mathcal{X}_{\text{init}}$ and $\sigma(1) \in \mathcal{X}_{\text{goal}}$.

Definition 2: Optimal Path Planning. Optimal path planning seeks to find a path σ^* that minimizes the cost $c(\sigma^*)$, defined as $c(\sigma^*) = \min\{c(\sigma) : \sigma \text{ is feasible for } (\mathcal{X}_{\text{free}}, \mathcal{X}_{\text{init}}, \mathcal{X}_{\text{goal}})\}$, where $c(\sigma)$ is the cost to reach $\mathcal{X}_{\text{goal}}$ along σ.

B. *Implementation*

Our implementation using the Quick-RRT* algorithm, modified for better performance in a simulated environment created in Gazebo using the ROS2. The Turtlebot3 robot navigates in a custom-designed map, demonstrating the practical application and reliability of the algorithm. We have conducted comprehensive performance comparisons between RRT*, Informed RRT*, and Informed Q-RRT* to validate our approach and substantiate its efficiency.

C. *Overview to the Q-RRT**

Quick-RRT* is a more advanced variant of the Rapidly Exploring Random Tree (RRT*) algorithm, designed to improve initial solution quality and accelerate convergence to an optimal path in motion planning problems. The upcoming methodology section describes the changes made to Q-RRT* that improve its performance compared to regular RRT*.

IV. Methodology

A. *RRT**

For this initial stage, we are comparing the RRT* with other successor algorithm to the improvement at every stage and also this helps to understand how the Q-RRT* is evolved from the RRT* algorithm. The Algorithm 1 explains the RRT* process to find an optimal path.

Algorithm 1 RRT* Algorithm

$T \leftarrow (V, E)$
for $i = 0$ to N **do**
  $x_{\text{new}} \leftarrow \text{Sample}(i)$
  $x_{\text{nearest}} \leftarrow \text{Nearest}(T, x_{\text{new}})$
  $\sigma \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{new}})$
  if CollisionFree(σ) **then**
    $X_{\text{near}} \leftarrow \text{Near}(T, x_{\text{new}})$
    $(x_{\text{parent}}, \sigma_{\text{parent}}) \leftarrow \text{ChooseParent}(X_{\text{near}}, x_{\text{nearest}}, x_{\text{new}}, \sigma)$
    $T \leftarrow \text{Connect}(T, x_{\text{parent}}, x_{\text{new}}, \sigma_{\text{parent}})$
    $T \leftarrow \text{Rewire-RRT*}(T, x_{\text{new}}, X_{\text{near}})$
  end if
end for
return T

1) Detailed Explanation: The RRT* algorithm extends the capabilities of the basic RRT algorithm by incorporating path optimization, which minimizes the total cost from the root to all other nodes. Here is a step-by-step explanation:

1) **Initialization:** The algorithm initializes the search tree T using the vertices found V and edges E.
2) **Sampling:** In every iteration, a new sample x_{new} is found to explore the configuration space further .
3) **Nearest Vertex:** The nearest vertex x_{nearest} in T to x_{new} is identified.
4) **Steering:** A path σ is generated from x_{nearest} to x_{new}, attempting to connect directly to the new sample.
5) **Collision Checking:** The algorithm checks if the path σ is free of obstacles along the way it finds the vertices.
6) **Near Vertices:** All vertices in the proximity of x_{new} are collected into the X_{near}.
7) **Choosing Parent:** Among the vertices in X_{near}, the algorithm selects x_{parent} that results in the lowest cost path to x_{new}.
8) **Connection:** The tree T is updated by connecting x_{new} to x_{parent}.
9) **Rewiring:** The tree is potentially rewired again and again to ensure that all vertices in X_{near} have an optimal path to the tree root.
10) **Iteration and Return:** These steps are repeated for a set number of iterations N. The algorithm returns the updated tree T, finally optimized paths have been found.

Algorithm 2 ChooseParent Function

$x_{\text{min}} \leftarrow x_{\text{nearest}}$
$\sigma_{\text{min}} \leftarrow \sigma_{\text{nearest}}$
$c_{\text{min}} \leftarrow \text{Cost}(x_{\text{min}}) + \text{Cost}(\sigma_{\text{min}})$
for all $x_{\text{near}} \in X_{\text{near}}$ **do**
  $(x, \sigma) \leftarrow \text{Steer}(x_{\text{near}}, x_{\text{new}})$
  $c \leftarrow \text{Cost}(x_{\text{near}}) + \text{Cost}(\sigma)$
  if $c < c_{\text{min}}$ **then**
    if CollisionFree(σ) **then**
      $x_{\text{min}} \leftarrow x_{\text{near}}$
      $\sigma_{\text{min}} \leftarrow \sigma$
      $c_{\text{min}} \leftarrow c$
    end if
  end if
end for
return $(x_{\text{min}}, \sigma_{\text{min}})$

The ***ChooseParent*** function in algorithm 2 tries to evaluate all potential connections within a neighborhood to determine the best parent for a new vertex. It selects paths that are not only lower in cost but also free of collisions, assuring the path's efficiency.

The "Rewire-RRT*" in algorithm 3 refines in iteratiosn the path planning tree by connecting x_{new} to each nearby node x_{near} if the path is less cost and collision-free. This enhances the tree by reducing path costs. However, unlike Q-RRT*, which may use a more advanced selection and strategy to cover larger areas of the search space more efficiently, while

TABLE I
SUMMARY OF THE COMPUTATIONAL COMPLEXITY OF VARIOUS PATH PLANNING ALGORITHMS

Algorithm	Probabilistic Completeness	Asymptotic Optimality	Sampling Strategy	Time Complexity	Space Complexity
RRT	Yes	No	Uniform	$O(n \log n)$	$O(n)$
RRT*	Yes	Yes	Uniform	$O(n \log n)$	$O(n)$
Informed RRT*	Yes	Yes	Informed	$O(n \log n)$	$O(n)$
Q-RRT*	Yes	Yes	Uniform	$O(n \log n)$	$O(n)$
Informed Q-RRT*	Yes	Yes	Informed	$O(n \log n)$	$O(n)$

Algorithm 3 Rewire-RRT*

for all $x_{\text{near}} \in X_{\text{near}}$ **do**
  $\sigma \leftarrow \text{Steer}(x_{\text{new}}, x_{\text{near}})$
  if $\text{Cost}(x_{\text{new}}) + \text{Cost}(\sigma) < \text{Cost}(x_{\text{near}})$ **then**
    if $\text{CollisionFree}(\sigma)$ **then**
      $T \leftarrow \text{Reconnect}(T, x_{\text{new}}, x_{\text{near}}, \sigma)$
    end if
  end if
end for
return T

Rewire-RRT* focuses only on local optimization and missing faster convergence on a global scale.

B. *Informed-RRT**

Informed-RRT* represents a significant advancement over the traditional RRT* by integrating a node rejection strategy that optimizes the search process. Initially operating under the same principles as RRT*, Informed-RRT* introduces a critical enhancement once a solution path is identified. This algorithm employs a hyper-ellipsoid defined by the initial state x_{init} and the goal state x_{goal}, as illustrated in **Fig.1**.

The key to its efficiency lies in focusing the sampling process within this hyper-ellipsoid, significantly reducing the search space. This region, determined by the equation:

$$\|x - x_{\text{init}}\| + \|x_{\text{goal}} - x\| > c_{\text{best}},$$

ensures that only states that could lead to a shorter path than the current best c_{best}. Here, c_{best} is the cost of the best solution found so far, and $\|x, y\|$ denotes an allowable heuristic estimatation of the cost from x to y.

Informed-RRT* outperforms ordinary RRT* due to a methodological change in sampling. Informed-RRT* selects samples within the hyper-ellipsoid rather than the entire configuration space. This targeted method not only improves search efficiency, but it also increases the possibility of detecting paths through narrow passages, which RRT* may fail to do due to random sample dispersion.

By dynamically changing the sampling area after each improvement in c_{best}, Informed-RRT* reduces the area needed to search, boosting the convergence rate towards the optimal path. This feature is very useful in complex contexts where navigating across tight places is required.

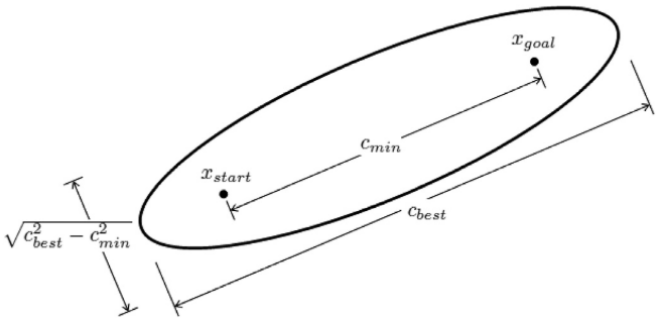

Fig. 1. Hyper-ellipsoid space definition for informed RRT*

C. *Quick-RRT**

Q-RRT* is an algorithm concentrates on the set of possible parent vertices by taking into account not only vertices within a given radius, but also their ancestry up to a user-defined depth. This method encourages the formation of a tree structure that converges more quickly to an optimal solution by making better use of a larger search space.

This section delineates the implementation of the Quick-RRT* (Q-RRT*) algorithm used in our experiments and implementation. Q-RRT* is designed to enhance the performance of the standard RRT* by dynamically adjusting its exploration space and rewiring process.

Algorithm 4 Q-RRT* Algorithm

$T \leftarrow (V, E)$
for $i = 0$ to N **do**
  $x_{\text{new}} \leftarrow \text{Sample}(i)$
  $x_{\text{nearest}} \leftarrow \text{Nearest}(T, x_{\text{new}})$
  $\sigma \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{new}})$
  if $\text{CollisionFree}(\sigma)$ **then**
    $X_{\text{near}} \leftarrow \text{Near}(T, x_{\text{new}})$
    $x_{\text{parent}}, \sigma_{\text{parent}} \leftarrow \text{Ancestry}(T, x_{\text{near}})$
    $(x_{\text{parent}}, \sigma_{\text{parent}}) \leftarrow \text{ChooseParent}(X_{\text{near}} \cup x_{\text{parent}}, x_{\text{nearest}}, x_{\text{new}}, \sigma)$
    $T \leftarrow \text{Connect}(T, x_{\text{parent}}, x_{\text{new}}, \sigma_{\text{parent}})$
    $T \leftarrow \text{Rewire-Q-RRT*}(T, x_{\text{new}}, X_{\text{near}})$
  end if
end for
return T

The Q-RRT* (Quick Rapidly-exploring Random Tree Star) algorithm is an extension of the RRT* algorithm, optimized to achieve faster convergence by enhancing the selection process of parent vertices and improving path optimization dynamically. The pseudocode provided in Algorithm 1 outlines the key steps.

1) Initialization: The procedure starts by populating the tree T with a collection of vertices V and edges E. The initialization step establishes the underlying structure that the algorithm will gradually develop upon:

```
T <- (V, E)
```

2) Main Loop: The algorithm's main execution takes place in a loop that runs until a termination condition is reached, which is commonly described by the number of iterations N. Each loop does numerous key operations:

```
for i = 0 to N do
```

3) Sampling: Each cycle generates a new sample point, x_{new}. This point is randomly picked from the configuration space, which is required for exploring the reachable space.

```
x_new <- Sample(i)
```

4) Nearest Vertex Selection: The algorithm then determines the closest vertex x_{nearest} from the tree T to the freshly sampled point x_{new}. This stage is critical for attaching the new sample to the tree in a way that potentially expands the tree into uncharted areas:

```
x_nearest <- Nearest(T, x_new)
```

5) Steering: A steering function is utilized to generate a new candidate point x_{new} from x_{nearest} to x_{rand}. This step guarantees that the transition from x_{nearest} to x_{new} follows the system's dynamics and constraints:

```
sigma <- Steer(x_nearest, x_new)
```

6) Collision Checking: Before accepting the new point, the algorithm searches for collisions along the path from x_{nearest} to x_{new}. If the road is free of barriers, the following steps are performed:

```
if CollisionFree(sigma) then
```

7) Near Vertices: The set of vertices X_{near} close to the new point x_{new} within a predetermined radius is discovered. This set is used to select the ideal parent and in the rewiring step:

```
X_near <- Near(T, x_new)
```

8) Parent Selection and Rewiring: The best parent for the new point is chosen from among the nearby vertices, taking into account their cost metrics and ability to create a lower-cost path. After linking the new point to its parent, the tree may be rewired to make the paths within the tree as short as feasible.

$x_{\text{parent}}, \sigma_{\text{parent}} \leftarrow \text{CHOOSEPARENT}(X_{\text{near}} \cup x_{\text{parent}}, x_{\text{nearest}}, x_{\text{new}})$

$T \leftarrow \text{Connect}(T, x_{\text{parent}}, x_{\text{new}}, \sigma_{\text{parent}})$

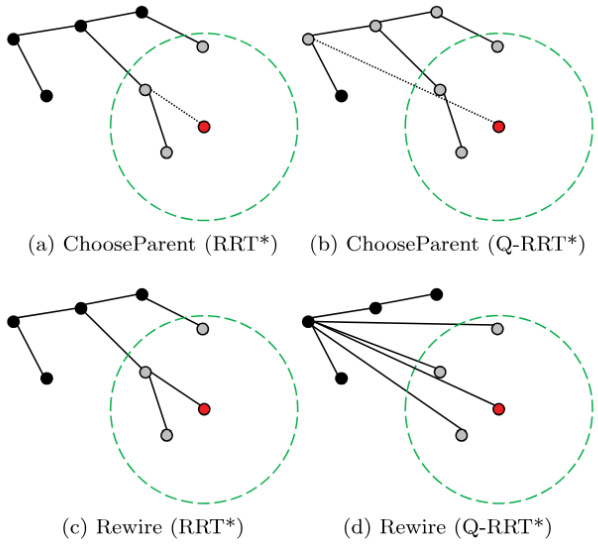

Fig. 2. Tree construction of (a) RRT* and (b) Q-RRT* and rewiring of the RRT*(c) and Q-RRT*(d). (red dot: x_{new}, dashed green circle: Near(x_{new}), grey dot: target vertex of iteration).

$T \leftarrow \text{Rewire-Q-RRT*}(T, x_{\text{new}}, X_{\text{near}})$

9) Returning the Result: Finally, the algorithm returns the tree T, which includes the path from the initial point to the goal, if such a path has been successfully found:

```
return T;
```

D. *Rewire - Q-RRT**

In the rewiring process the algorithm of Q-RRT* considers the ancestry also in the rewire procedure, while the nearby vertices are checked only if a path through either x_{new} else the parent of the x_{new} that gives the path with significantly less cost. The advantage of the tendency for the RRT* to have the common parents is increased and this helps Q-RRT* give out best initial solutions and improved solutions. The Rewire process of Q-RRT* is explained and delineated abouts its process in Algorithm 5.

Algorithm 5 Rewire-Q-RRT*

for all $x_{\text{near}} \in X_{\text{near}}$ **do**
  for all $x_{\text{from}} \in \{x_{\text{new}}\} \cup \text{ancestor}(T, x_{\text{new}})$ **do**
    $\sigma \leftarrow \text{Steer}(x_{\text{from}}, x_{\text{near}})$
    if $\text{Cost}(x_{\text{from}}) + \text{Cost}(\sigma) < \text{Cost}(x_{\text{near}})$ **then**
      if $\text{CollisionFree}(\sigma)$ **then**
        $T \leftarrow \text{Reconnect}(T, x_{\text{from}}, x_{\text{near}}, \sigma)$
      end if
    end if
  end for
end for
return T

1) Algorithm Steps:

1) **For each x_{near} in X_{near}**: This outer loop iterates over each node in close proximity of the newly inserted node x_{new}, examining it for potential reconnection.
2) **Consider each x_{from}**: For every x_{near}, The algorithm evaluates the direct connection from x_{new} as well as all ancestor nodes of x_{new}. This expands the evaluation area to include paths that may not have been optimal previously but may become preferable due to the addition of x_{new}.
3) **Steering and Cost Evaluation**:
   - **Steer**: A path σ is proposed from x_{from} to x_{near}, calculated to maintain feasibility within the environmental constraints.
   - **Cost Check**: The combined cost of moving from x_{from} through σ to x_{near} is compared with the existing cost to reach x_{near}. If the new path offers a reduction in cost, it proceeds to the next check.
4) **Collision Check**:The demonstrated path σ is tested for safety to ensure it does not cross with any obstructions. Safety is critical to ensuring the practicality of the new path.
5) **Reconnect**: If the new path is both cost-effective and collision-free, the tree (T) is modified. The parent of x_{near} is changed to x_{from}, and the associated path is modified to σ, essentially "rewiring" the tree to incorporate this newly optimized path.
6) **Return the Optimized Tree**: After all possible rewirings are assessed, the algorithm returns the optimized tree T, which ideally features reduced overall path costs and improved navigational efficiency.

V. Numerical Results

A. Experimental Setup

Experiments were conducted to evaluate and compare the performance of five path planning algorithms: RRT, RRT*, Informed RRT*, Q-RRT* and Informed Q-RRT*. The evaluation criteria based on the efficiency of path planning, the complexity of the paths generated in the map, and the computational resources utilized during each simulation. Each algorithm was tested in both a 2D simulated environment and a complex 3D environment using the Gazebo simulator. The Gazebo simulator ran in the system enabled with Ubuntu 22.04 paired with ROS 2 Humble along with the environment used for the project -3 (phase 2).

The tests included repeated passes through maps drawn in Open CV and also the gazebo in which the robot had to navigate from a starting point to a destination while avoiding obstacles. The performance metrics were path length, calculation time, and convergence rate.

B. 2-D Visualization

The 2D simulations were conducted in a environment with preset obstacles using Open CV to assess the algorithms' ability to find a feasible path efficiently. The environment size was set to a standard 500x500 grid with similar obstacles as used in the implementation paper.

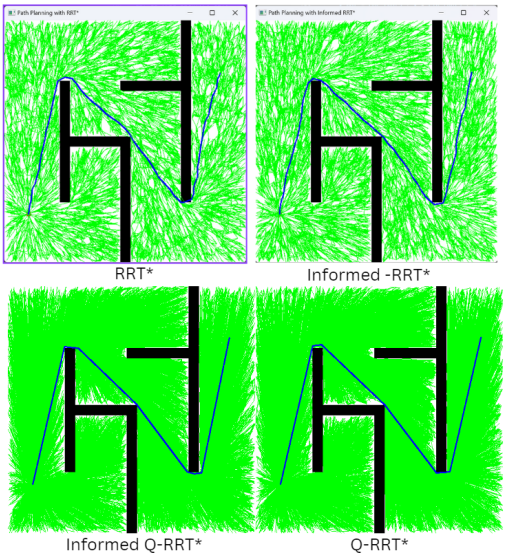

Fig. 3. 2-D visualization results at 10000 iterations

1) Methods Used For Comparison: Each algorithm was scripted to navigate from a start to a goal location inside the Open CV map grid, while the simulations running for six different iteration counts: 10,000, 5,000, 4,000, 3000 and 2000 iterations, to test performance under different computation loads.

The primary metrics for evaluation were:

- **Time to Converge**: The time took for the algorithm to find a optimal path.
- **Cost to Converge**: The cost took by the algorithm to reach the first valid path.
- **Path Cost**: The total length or cost of the path found (lower is better here).
- **Total Time Taken**: The total time took from start to the end of the simulation.

C. Results and Analysis

Table I: 10,000 Iterations At the maximum testing value of 10,000 iterations, all algorithms successfully found paths. But, Q-RRT* and Informed Q-RRT* showed more optimal path solutions in terms of lower path costs and total time taken compared to RRT and RRT*. This suggests that the depth parameter and informed sampling strategies contribute significantly to finding more efficient paths. the result can be observed in **Fig. 3**.

Table II: 5,000 Iterations When the iteration count was reduced to 5,000, a similar trend was observed. Both Q-RRT* and Informed Q-RRT* outperformed the traditional RRT and RRT* in terms of lower costs and quicker convergence times, highlighting the efficiency of these algorithms in faster computations with fewer resources.the result can be observed in **Fig. 4**.

TABLE II
COMPARISON OF ALGORITHMS PERFORMANCE AT 10,000 ITERATIONS

Algorithm	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
RRT-STAR	5.503	1193.122	962.495	228.738
Informed RRT-STAR	6.111	1182.449	959.564	255.329
Q-RRT-STAR (Depth 3)	9.382	969.815	952.368	529.101
Informed Q-RRT-STAR (Depth 3)	7.718	1019.717	959.158	241.885

TABLE III
COMPARISON OF ALGORITHMS PERFORMANCE AT 5,000 ITERATIONS

Algorithm	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
RRT-STAR	7.622	1180.321	1002.879	43.578
Informed RRT-STAR	7.222	1155.783	996.868	57.459
Q-RRT-STAR (Depth 3)	11.533	1065.766	987.106	48.025
Informed Q-RRT-STAR (Depth 3)	6.874	1013.432	965.923	58.374

TABLE IV
COMPARISON OF ALGORITHMS PERFORMANCE AT 4,000 ITERATIONS

Algorithm	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
RRT-STAR	14.472	1096.409	1030.099	48.158
Informed RRT-STAR	8.828	1108.356	1024.370	32.308
Q-RRT-STAR (Depth 3)	10.495	1001.557	973.546	27.981
Informed Q-RRT-STAR (Depth 3)	2.744	1012.518	993.856	41.809

TABLE V
COMPARISON OF ALGORITHMS PERFORMANCE AT 3,000 ITERATIONS

Algorithm	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
RRT-STAR	6.887	1164.122	1082.880	15.674
Informed RRT-STAR	3.652	1122.174	1046.243	20.181
Q-RRT-STAR (Depth 3)	8.860	997.108	993.913	16.480
Informed Q-RRT-STAR (Depth 3)	8.676	1045.813	996.000	16.584

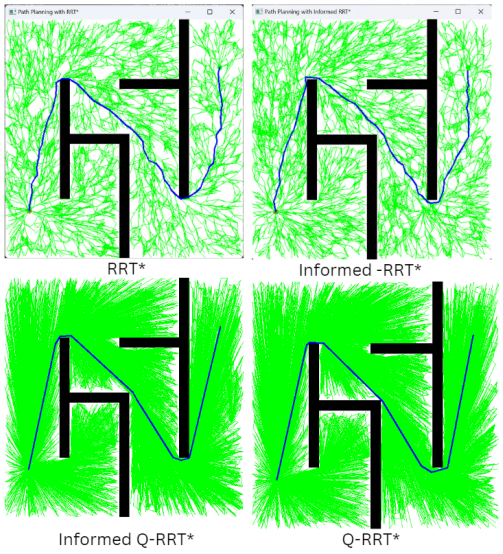

Fig. 4. 2-D visualization results at 5000 iterations

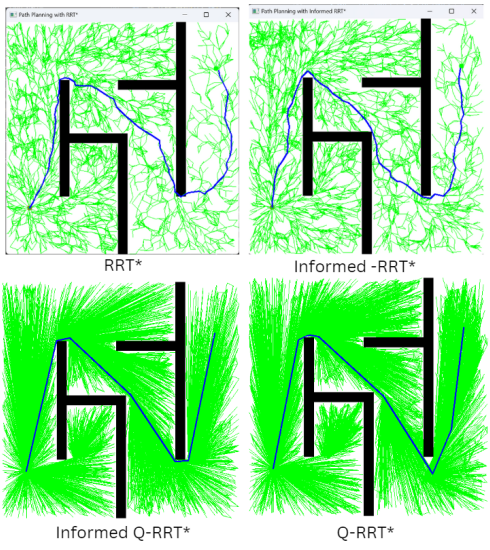

Fig. 5. 2-D visualization results at 4000 iterations

TABLE VI
COMPARISON OF ALGORITHMS PERFORMANCE AT 2,000 ITERATIONS

Algorithm	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
RRT-STAR	No Path	Inf	Inf	5.585
Informed RRT-STAR	5.876	1198.869	1192.597	7.331
Q-RRT-STAR (Depth 3)	9.82	1060.567	1052.452	6.236
Informed Q-RRT-STAR (Depth 3)	6.324	1013.364	1013.364	7.084

TABLE VII
Q-RRT-STAR PERFORMANCE ACROSS DIFFERENT DEPTHS

*10,000 Iterations

Depth	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
1	12.492	1010.491	990.357	502.638
2	9.382	969.815	952.368	529.101
3	13.071	1011.450	950.560	217.690
4	13.887	1033.517	958.616	215.451

*5,000 Iterations

Depth	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
1	10.469	1057.938	1042.116	43.959
2	11.553	1065.766	987.106	48.025
3	9.377	1050.074	966.641	48.016
4	11.875	993.451	964.424	41.337

*4,000 Iterations

Depth	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
1	11.092	973.914	976.708	26.806
2	10.495	1001.557	973.546	27.981
3	8.250	1013.001	997.446	33.752
4	12.728	1004.675	969.014	29.670

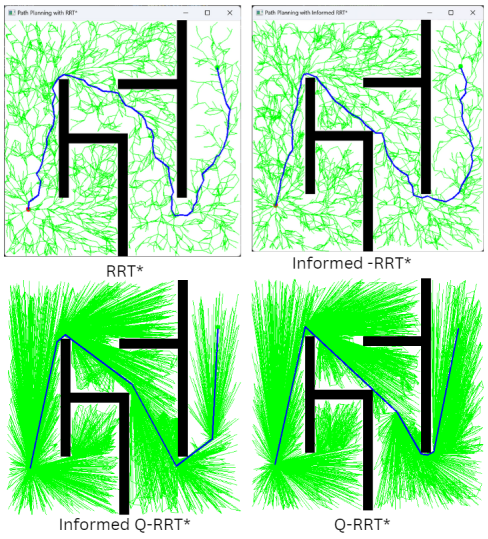

Fig. 6. 2-D visualization results at 3000 iterations

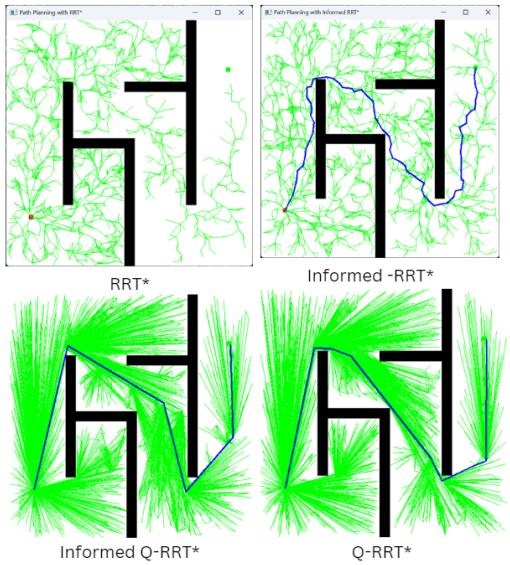

Fig. 7. 2-D visualization results at 2000 iterations

Table III: 4,000 Iterations At the iteration count of 4,000 the result in the **Fig. 5** shows the performance of Q-RRT* and especially Informed Q-RRT* became even more decreased. Informed Q-RRT* showed the most significant improvement in path optimization and computational efficiency.

Table IV: 3,000 Iterations At the iteration count of 3000, the result as shown in the **Fig. 6**, the algorithms does not show a big difference from the performance acquired from the 4000 iterations and still we could infer that Q-RRT* and informed Q-RRT* out perform the predecessors.

Table V: 2,000 Iterations Now, at this stage with decreasing the iterations to 2000, the path finding capability decreased and RRT* failed to find the path and informed RRT* does

TABLE VIII
INFORMED Q-RRT-STAR PERFORMANCE ACROSS DIFFERENT DEPTHS

*10,000 Iterations

Depth	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
1	3.977	1152.240	969.607	248.006
2	3.720	1049.576	950.699	259.390
3	7.718	1019.717	959.158	241.885
4	4.404	1037.653	948.803	263.830

*5,000 Iterations

Depth	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
1	3.716	1058.246	969.080	58.084
2	6.709	1010.411	965.136	57.119
3	6.874	1013.432	965.923	58.374
4	3.852	1000.351	957.458	60.954

*4,000 Iterations

Depth	Time to Converge (s)	Cost to Converge	Path Cost	Total Time Taken (s)
1	3.823	987.707	977.014	38.523
2	5.009	1030.480	985.653	35.909
3	2.744	1012.518	993.856	41.809
4	4.885	1033.523	975.561	38.695

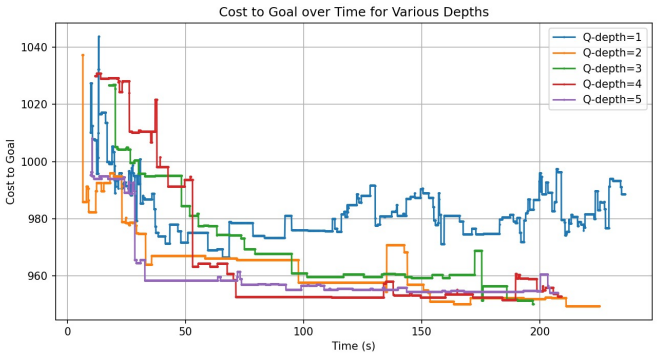

Fig. 8. Q-RRT* performance comparisons with variable depths 1-5 at 10,000 iterations

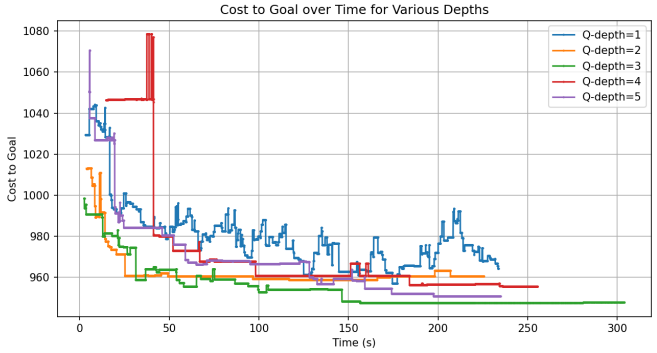

Fig. 9. Informed Q-RRT* performance comparisons with variable depths 1-5 at 10,000 iterations

not show optimal path. But Q-RRT* and informed Q-RRT* performs better even at the lower iterations and shows less cost than informed RRT*. The results can be observed from the **Fig. 7**.

D. Video Results of the 2-D implementation

The finest possible optimization of the path finding by all the algorithms compared so far has been visualized and attached here in the form of link to redirect to the video https://drive.google.com/file/d/1oeMSC2dic9kGpPTEV-Chv5lB1lhhDZjj/view?usp=sharing

E. Analysis of Q-RRT and Informed Q-RRT* with Various Depths*

From the above observations we can infer that Q-RRT* and Informed Q-RRT* outperforms the predecessors and also searches optimal path and uses less computational paths comparably. It previously used the **predefined depth as value 3**, but we used depths range from 1 to 5 in order to observe the finest optimization that could be achieved by the Q-RRT* and Informed Q-RRT* and the observed results of the Q-RRT* have been shown in in the **Fig. 8**. Also in **Fig. 9**, the performance of the informed Q-RRT* with the variable depths has been observed and it **takes less time** at maximum depth when compared to Q-RRT*. In the both cases, we can clearly see as the depth to find the neighbour parent increases, the Time to Converge is less and finds best initial solution and further rewires to find the best optimal path possible by both the algorithm.

F. Gazebo Visualization

1) Methodology: The movement logic in the script that is included is designed to drive a robot down a path chosen by the Quick RRT* algorithm, with closed-loop control provided by PID controllers. This is accomplished by a sequence of coordinated steps in the 'CloseLoopControllerNode' class used in the script, which manages the robot's motion control.

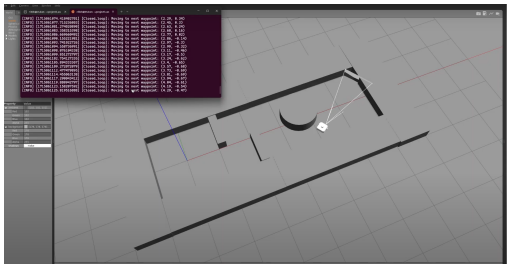

Fig. 10. Gazebo simulation visualization in defined world

Once the path planner algorithm finishes execution providing the way-points after which the way-points are fed to the 'odom' topic and 'cmd vel' topic is published for the TurtleBot to move to its first way-point and once the robot is close to the current way-point which is determined using a predefined threshold of 0.1 the index of the path way-points is incremented to target the next waypoint in the path and this process is followed for all other way-points. If all way-points are covered, the node shows "Goal Reached" in the log and stops the robot.

2) Video Results of Gazebo Visualization: The Gazebo simulations demonstrated in the world shown in the **Fig. 10** for all the five algorithms and as observed earlier it is evident that RRT* takes maximum time to find the optimal path and Informed - RRT* takes comparably less time than RRT*, while the Q-RRT* and Informed Q-RRT* shows best performance and when we compare with the Q-RRT*, Informed Q-RRT* slightly performs better when it comes to to the faster convergence and navigate along the best optimal path.

The video results of the gazebo visualization has been attached below for the reference. https://drive.google.com/file/d/1Y2LveUvn7KraKCprEd8V5AkdQiZWvRAb/view?usp=sharing

G. *Discussion*

The experiments confirm that while RRT* are effective for quick path planning in simpler environments, Informed RRT* and especially Informed Q-RRT* provide superior performance in complex environments and achieve even faster convergence rate with efficient cost. The addition of informed sampling in Informed RRT* and the parent selection optimization in Informed Q-RRT* contribute significantly to their enhanced efficiency and path quality.

VI. Future Work

A. *Multi-Robot Coordination*

Our future innovation plans will allow Quick-RRT* to be used in swarm robotics and cooperative tasks in which several agents must navigate and operate efficiently in the same space. Multi-robot coordination will take advantage of the Quick-RRT* algorithm's inherent strengths, utilizing its efficient path planning capabilities across multiple autonomous agents to achieve harmonized and synchronized operations, thereby increasing overall task efficiency and effectiveness in complex scenarios.

VII. Conclusion

This project has successfully met several key objectives, significantly advancing the capabilities of path planning algorithms used in robotic navigation. Below, we detail the main achievements:

A. *Advanced Path Planning*

The implementation of the Quick-RRT* algorithm on the Turtlebot3 platform has resulted in a significant increase in the efficiency and efficacy of path planning. Quick-RRT* is designed to optimize the exploration of the search space, allowing it to find optimal paths faster than its predecessors. This solution has proven particularly useful in complicated situations, where the capacity to develop viable pathways quickly can have a substantial impact on the overall performance of autonomous robotic systems.

B. *Algorithm Enhancements*

Key enhancements to the algorithm include the introduction of ancestral connections and dynamic rewiring, which have been instrumental in improving the convergence rate to optimal paths. These features allow Quick-RRT* to better navigate the search space:

Ancestral Connections: Quick RRT* proved to perform well using the ancestral method which find ths parents of the neighbours to rewire.**Dynamic rewiring:** This technique constantly updates the links in the search tree as new information becomes available or the environment changes.

These enhancements have resulted in a more robust and adaptable path planning system.

References

[1] In-Bae Jeong, Seung-Jae Lee, Jong-Hwan Kim, "Quick-RRT*: Triangular inequality-based implementation of RRT* with improved initial solution and convergence rate," Expert Systems with Applications, vol. 123, 2019, pp. 82-90, ISSN 0957-4174.

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

[3] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," International Journal of Robotics Research, vol. 30, no. 7, pp. 846-894, 2011.

[4] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2014, pp. 2997-3004.

[5] D. Nasir, F. Adil, T. Dang, K. Moore, and H. W. Wah, "RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution," in Proc. IEEE International Conference on Mechatronics and Automation, 2013, pp. 1651-1656.

[6] A. Shkolnik and R. Tedrake, "Path planning in 1000+ dimensions using a task-space Voronoi bias," in Proc. IEEE International Conference on Robotics and Automation (ICRA), 2009, pp. 2061-2067.

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

[8] O. Salzman and D. Halperin, "Asymptotically near-optimal RRT for fast, high-quality motion planning," IEEE Transactions on Robotics, vol. 32, no. 3, pp. 473-483, June 2016.