Real-Time Path Planning Algorithm Based On RRT*

Jeffin Johny Kachappilly
A James Clark School of Engineering
University of Maryland
College Park, MD, USA
jeffinjk@umd.edu

Pradip Kathiriya
A James Clark School of Engineering
University of Maryland
College Park, MD, USA
pradip@umd.edu

***Abstract*—This project deals with the real-time implementation of a path-planning algorithm, which utilizes the Rapidly Exploring Random Tree (RRT) algorithm and its variants. The algorithm is capable of handling multiple-goal query scenarios and the agent (robot) determines and executes the path to these goal locations in real-time. The said architecture employs an online tree rewiring strategy, that allows the tree root to move along with the agent position without abandoning the previously sampled paths. If the tree does not get generated within a specified time constraint, the robot executes the path with the available tree structure.**

The algorithm is visualized in a 2D environment, which comprises just static obstacles. The agent navigates through this maze avoiding obstacles within a predefined radius and reaches the first goal position. And the algorithm is implemented again for subsequent goal and obstacle locations.

Keywords—Rapidly Exploring Random Tree, Predefined radius, online tree rewiring.

I. INTRODUCTION

Path planning, also widely known as motion planning is a computational problem for estimating a valid sequence of configurations that moves the object from the source to the destination while avoiding obstacles in the process.

In recent years, robotic motion planning problem has made significant strides in the development of novel path planning algorithms. Even though robots may possess various characteristics such as sensing, actuation, size, workspace, application, etc., the problem of navigating through a complex environment is embedded and essential in almost all robotics applications. Moreover, this problem is not only limited to robotics but also is relevant to other disciplines such as verification, computational biology, and computer animation.

A path-planning algorithm to address this problem is said to be complete if it terminates in finite time and returns a valid solution if one exists, and failure otherwise. But, one should keep in mind that not all algorithms are favored in terms of a computational point of view.

Many of the previous approaches for path-planning in real-time include heuristic methods, potential field methods, and sampling methods such as rapidly exploring random trees(RRTs).

Take for instance A*, which was tried to be implemented in real-time for gaming industries. One major practical drawback was its space complexity, as it stores all generated nodes in memory. Thus, in practical travel-routing systems, it is generally outperformed by algorithms that can pre-process the graph to attain better performance, as well as memory-bounded approaches.

Although the response time of the potential field algorithms is fast and implemented in real-time, however, occasional trapping into local minima is a major concern.

RRTs are efficient for single-query tasks in a continuous environment, i.e. when the goal point is fixed. The different iterations of RRTs, either regrow the tree from scratch or prune away the infeasible branches. The algorithm implemented in this project stores the whole tree structure from the first iteration and deploys rewiring of certain nodes for subsequent goal locations. By choosing this approach, it is efficiently able to find multiple goal locations in real-time.

In the following section, a brief overview of the relevant literature on real-time path planning algorithms is presented. Section 3 deals with the real-time problem statement and covers the notation used in pseudocode and also provides a glimpse of the 2D environment. A detailed description of the implemented RT-RRT* method is done in section 4. Sections 5 and 6, showcase the simulation results and the conclusion learnt from this algorithm, respectively.

II. LITERATURE REVIEW

In the upcoming subsections, a detailed study on the currently used approaches on real-time path planning are conducted.

A. RRTs and their variants :

Probabilistic sampling-based approaches have recently become very popular. In particular, Rapidly-exploring Random Trees (RRTs) have been shown to be effective for solving single-shot path planning problems in complex configuration environments. By combining random sampling of the configuration space with biased sampling around the goal configuration, RRTs efficiently provide solutions to problems involving vast, high-dimensional configuration spaces that would be intractable using deterministic approaches. As a result, they have been widely used in robotics.

However, RRTs are not asymptotically optimal and only provide a valid and not optimal path to the goal as rewiring of nodes is not performed. RRT* on the other hand, provides an optimal path as rewiring between nodes is allowed. RRT* is asymptotically stable and its only drawback is that it rewires the entire tree structure, which is not feasible in a large environment. Focused sampling-based methods became possible with the emergence of Informed RRT*. In informed RRT*, the start and goal positions are considered focal points of the ellipsoid. This is especially useful for achieving a greater convergence rate in larger environments as the rewiring is limited to this ellipsoid. So the algorithm implemented in this paper, combines all the advantages of various RRT variants and executes in real-time.

B. *Other tree-based path planning approaches:*

The Closed-loop RRT (CL-RRT) algorithm, extends the RRT by making use of a low-level controller and planning over the closed-loop dynamics. The tree of the algorithm is just generated for a small region. This makes the search time to reduce significantly but does not provide an optimal path to the goal. CL-RRT prunes the infeasible branches when the agent moves on the tree and ensures that the agent will not deviate from the planned path. What makes RT-RRT*, different from CL-RRT is the rewiring of branches which makes the path effective as well. This makes the algorithm implemented here, cope up with multi-query tasks and traverse large worlds more efficiently.

C. *Potential based path planning* :

Potential fields are a well-studied technique that works by utilizing only the local information to guide the robot's movement. This approach is therefore quite quick, and makes it appropriate for real-time obstacle avoidance as well as path planning in relatively simple spaces.

The main idea is to consider the robot's configuration as being a particle in some energy potential landscape. Due to "gravity" the particle will feel some virtual forces equal to the negative of the gradient of this landscape. If the landscape is constructed to have a global minimum at the goal, then by following the gradient the particle will, hopefully, arrive at the goal.

This method works well when the robot simply needs to move slightly away from a straight-line path to avoid obstacles, provided that obstacles are relatively convex and spatially distant. Its main advantages are :

1) speed of computation, and

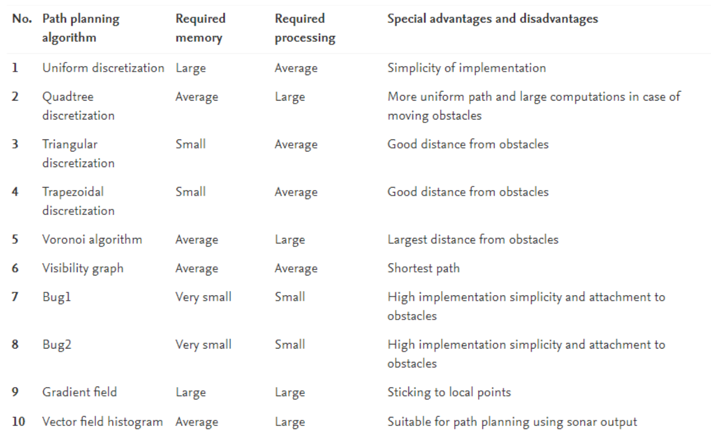
2) only local information about obstacles is needed.

However, like other local methods it is prone to local minima caused either by concave obstacles, or narrow passages where the repulsive forces from either side of the passage cancel out the attractive force.

D. *Graph-based path planning* :

The A* Algorithm is a popular graph traversal path planning algorithm. A* operates similarly to Dijkstra's algorithm with the added advantage that it guides its search towards the most promising states by employing a heuristic function, potentially saving a significant amount of computation time [3].A* is the most widely used for approaching a near-optimal solution with the available data-set/node. The graph-based algorithm comes with the added disadvantage of processing the path generated even after the exploration of the environment. This approach is not suitable when there are multiple goal positions as the entire graph has to be searched again for new goal position.[2]

TABLE I. Different algorithms and their features



No.	Path planning algorithm	Required memory	Required processing	Special advantages and disadvantages
1	Uniform discretization	Large	Average	Simplicity of implementation
2	Quadtree discretization	Average	Large	More uniform path and large computations in case of moving obstacles
3	Triangular discretization	Small	Average	Good distance from obstacles
4	Trapezoidal discretization	Small	Average	Good distance from obstacles
5	Voronoi algorithm	Average	Large	Largest distance from obstacles
6	Visibility graph	Average	Average	Shortest path
7	Bug1	Very small	Small	High implementation simplicity and attachment to obstacles
8	Bug2	Very small	Small	High implementation simplicity and attachment to obstacles
9	Gradient field	Large	Large	Sticking to local points
10	Vector field histogram	Average	Large	Suitable for path planning using sonar output

III. NOTATIONS AND ENVIRONMENT DEVELOPED

Let χ denotes the workspace in which the algorithm works, where $\chi \subseteq R^2$. χ_{free} and χ_{obs}, denotes the free space and obstacle space, respectively. The tree is denoted by τ and each node inside it is represented by $x_i \in \chi_{free}$. 'x' also denotes any position in the environment, other than the nodes, for instance, $x_{goal} \in \chi_{free}$. represents the multiple goal locations. x_0 denotes the current tree root and x_a represents the agent position. Q_r and Q_s are queues initialized for rewiring algorithm. In the path planner, a limit for planned nodes are set k , which limits the planning of a path ahead of x_0.

The formulation of real-time path planning in dynamic scenario as follows:

An optimal path from agent location to goal point, is determined. For estimating minimal path, cost-to-reach is calculated which is the Euclidean length of path from x_0 to x_i . To implement the algorithm in real-time, a limit is set for time within which Tree expansion and rewiring occurs. So if a path to goal is not generated within the time constraint, the robot moves towards the goal position based on the Euclidean distance between start and goal positions.

As the entire tree structure is stored for various iterations for multiple goal query instances, the main challenge is how to rewire the tree efficiently to find optimal path towards each goal location.

The figure below represents the map implemented in 2D environment. The linear black lines signify 'the wall obstacles'. The red hexagon denotes the 'initial goal position'. The blue cross denotes the 'current robot location'.

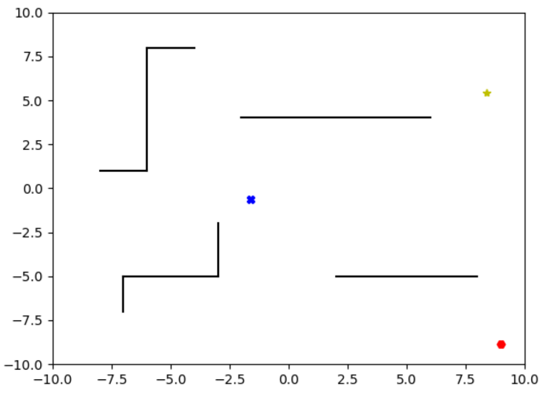


Fig. 1. Map created using matplotlib

IV. Methodology

This section deals with the detailed explanation of the methodology followed. Algorithm 1, represents the overall structure of the Real-Time Rapidly Exploring Random Tree Star algorithm. The main loop contains, other sub algorithms which are for Tree Expansion and rewiring(Algorithm 2) and for path planning(Algorithm 6). Initially the tree is initialized with x_a as root (line2). Line 4 represents the updation of the goal points and position of the agent that is used later in the Path Planning and the Tree Expansion- Rewiring algorithm. Expansion and Rewiring of tree, for a specific time constraint (lines 5-6). Path planning from tree root (x_0) to (x_k), where k is the user-defined limit for planning a path (line 7). After completion of path planning, the agent is transferred to the next immediate node after x_0 in the planned path x_1 (lines 8-9). Thereby, enabling the agent to move towards goal position for the planned path.

Algorithm 1 RT-RRT*: Our Real-Time Path Planning

```
1: Input: $\mathbf{x}_a, \mathcal{X}_{obs}, \mathbf{x}_{goal}$
2: Initialize $\mathcal{T}$ with $\mathbf{x}_a, \mathcal{Q}_r, \mathcal{Q}_s$
3: loop
4:     Update $\mathbf{x}_{goal}, \mathbf{x}_a, \mathcal{X}_{free}$ and $\mathcal{X}_{obs}$
5:     while time is left for Expansion and Rewiring do
6:         Expand and Rewire $\mathcal{T}$ using Algorithm 2
7:     Plan $(\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_k)$ to the goal using Algorithm 6
8:     if $\mathbf{x}_a$ is close to $\mathbf{x}_0$ then
9:         $\mathbf{x}_0 \leftarrow \mathbf{x}_1$
10:    Move the agent toward $\mathbf{x}_0$ for a limited time
11: end loop
```

IV.I Tree expansion and rewiring:

Random sampling of nodes (x_{rand}), using modified Informed RRT* technique (line 2). By retaining τ between iterations, the tree grows too large (but with a limited number of the nodes) to be handled wholly in real-time path planning. So only a subset of nodes (χ_{SI}) are focused for a given instance (line 3). 'FindNodesNear' in Algorithm 2, same as RRT*, returns the nodes in χ_{SI} that are at most at a calculated distance from x_{rand} (line 5). The condition where tree has stopped adding more nodes (line 6). The nodes are added until it covers the tree (line 7). The rewiring of the sampled node are shown in (lines 8,10,11). Where $Q_r$is the queue, which is initialized for rewiring of random nodes.

Algorithm 2 Tree Expansion-and-Rewiring

```
1: Input: $\mathcal{T}, \mathcal{Q}_r, \mathcal{Q}_s, k_{max}, r_s$
2: Sample $\mathbf{x}_{rand}$ using (1)
3: $\mathbf{x}_{closest} = \arg\min_{\mathbf{x} \in \mathcal{X}_{SI}} \mathtt{dist}(\mathbf{x}, \mathbf{x}_{rand})$
4: if $\mathtt{line}(\mathbf{x}_{closest}, \mathbf{x}_{rand}) \subset \mathcal{X}_{free}$ then
5:     $\mathcal{X}_{near} = \mathtt{FindNodesNear}(\mathbf{x}_{rand}, \mathcal{X}_{SI})$
6:     if $|\mathcal{X}_{near}| < k_{max}$ or $|\mathbf{x}_{closest} - \mathbf{x}_{rand}| > r_s$ then
7:         AddNodeToTree($\mathcal{T}, \mathbf{x}_{rand}, \mathbf{x}_{closest}, \mathcal{X}_{near}$)
8:         Push $\mathbf{x}_{rand}$ to the first of $\mathcal{Q}_r$
9:     else
10:        Push $\mathbf{x}_{closest}$ to the first of $\mathcal{Q}_r$
11:    RewireRandomNode($\mathcal{Q}_r, \mathcal{T}$)
12: RewireFromRoot($\mathcal{Q}_s, \mathcal{T}$)
```

IV.II Random Sampling:

The sampling used in Algorithm 2, which is implemented for the project is as follows. A random number is generated between 0 and 100. A condition is set for random generation, say 5. If the generated number is greater than 5, then uniform exploration is implemented in the map. But say, if the number is less than 5, the goal node would be assigned as the random. This, implementation of random samples makes sure that the robot moves towards the goal position for around 5percent of the time.

IV.III Adding Node to the tree:

The same process of addition of nodes in RRT* is followed. Algorithm 3 finds the parent with the minimum cost-to-reach (lines 2-6). The computation of c_i is related to the length of the path from x_0 to x_i. Thus, cost(x_i) needs to compute c_i whenever c_j for any intermediate node in the path to x_i is changed. N_τ and E_τ, represents the nodes and edges(which is a connection between 2 nodes) in a tree. Line 5 checks whether the cost is minimum and also the line joining between new node and existing node is Obstacle free. Only then the node is added as shown in line 6.

Algorithm 3 Add Node To Tree

```
1: Input: $\mathcal{T}, \mathbf{x}_{new}, \mathbf{x}_{closest}, \mathcal{X}_{near}$
2: $\mathbf{x}_{min} = \mathbf{x}_{closest}, c_{min} = \mathtt{cost}(\mathbf{x}_{closest}) + \mathtt{dist}(\mathbf{x}_{closest}, \mathbf{x}_{new})$
3: for $\mathbf{x}_{near} \in \mathcal{X}_{near}$ do
4:     $c_{new} = \mathtt{cost}(\mathbf{x}_{near}) + \mathtt{dist}(\mathbf{x}_{near}, \mathbf{x}_{new})$
5:     if $c_{new} < c_{min}$ and $\mathtt{line}(\mathbf{x}_{near}, \mathbf{x}_{new}) \in \mathcal{X}_{free}$ then
6:         $c_{min} = c_{new}, \mathbf{x}_{min} = \mathbf{x}_{near}$
7: $V_\mathcal{T} \leftarrow V_\mathcal{T} \cup \{\mathbf{x}_{new}\}, E_\mathcal{T} \leftarrow E_\mathcal{T} \cup \{\mathbf{x}_{min}, \mathbf{x}_{new}\}$
```

IV.IV Rewiring the tree:

Rewiring is done when a node gets a lower cost-to-reach value by passing from another node instead of its parent (same as in RRT*). The only additional implementation as opposed to RRT* is the rewiring of already added nodes due to change in goal positions.

Algorithm 4 focuses in the rewiring of random nodes. (Lines 3-7) focuses on the rewiring of random neighbour nodes. Line 8 shows the addition of x_{near} to Q_r, as the nodes around near node has the potential to be rewired. Rewiring continues until the condition in line 9 is satisfied. If time runs out and there are still nodes remaining in Q_r, the rewiring is continued in the next iteration.

This is done using both focused and uniform sampling, but in patches instead of just one node.

Algorithm 4 Rewire Random Nodes

```
1: Input: $\mathcal{Q}_r$, $\mathcal{T}$
2: repeat
3:     $\mathbf{x}_r$=PopFirst($\mathcal{Q}_r$), $\mathcal{X}_{near}$=FindNodesNear($\mathbf{x}_r$, $\mathcal{X}_{SI}$)
4:     for $\mathbf{x}_{near} \in \mathcal{X}_{near}$ do
5:         $c_{old}$=cost($\mathbf{x}_{near}$), $c_{new}$=cost($\mathbf{x}_r$)+dist($\mathbf{x}_r$, $\mathbf{x}_{near}$)
6:         if $c_{new} < c_{old}$ and line($\mathbf{x}_r$, $\mathbf{x}_{near}$) $\in \mathcal{X}_{free}$ then
7:             $E_{\mathcal{T}} \leftarrow (E_{\mathcal{T}} \setminus \{$Parent($\mathbf{x}_{near}$), $\mathbf{x}_{near}\}) \cup \{\mathbf{x}_r, \mathbf{x}_{near}\}$
8:             Push $\mathbf{x}_{near}$ to the end of $\mathcal{Q}_r$
9: until Time is up or $\mathcal{Q}_r$ is empty.
```

IV.V Path Planning pseudocode:

Two planning instances are implemented:

1) When tree reaches goal (lines 2-4)

2) Tree has not reached x_{goal} (lines 6-10).

For the first instance, the nodes are rewired for optimal path. For the second way, a cost function is assigned to get close to goal position. To prevent trapping in local minima and enable planning to visit other branches, a k-step path is planned (lines 6,7) and block already visited nodes (line 10). When path planning reaches a node and cannot traverse any further (line 8), that particular path is blocked (lines 9,10). Then the best path which is closest to goal position is updated (line11). However, the agent follows the best path if it leads to some place closer than the current place (line 12).

Algorithm 6 Plan a Path for k Steps

```
1: Input: $\mathcal{T}$, $\mathbf{x}_{goal}$
2: if Tree has reached $\mathbf{x}_{goal}$ then
3:     Update path from $\mathbf{x}_{goal}$ to $\mathbf{x}_0$ if the path is rewired
4:     $(\mathbf{x}_0, ..., \mathbf{x}_k) \leftarrow (\mathbf{x}_0, ..., \mathbf{x}_{goal})$
5: else
6:     for $\mathbf{x}_i \in (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_k)$ do
7:         $\mathbf{x}_i$=child of $\mathbf{x}_{i-1}$ with minimum $f_c$=cost($\mathbf{x}_c$)+H($\mathbf{x}_c$)
8:         if $\mathbf{x}_i$ is leaf or its children are blocked then
9:             $(\mathbf{x}_0^{'}, ..., \mathbf{x}_k^{'}) \leftarrow (\mathbf{x}_0, ..., \mathbf{x}_i)$
10:            Block $\mathbf{x}_i$ and Break;
11:    Update best path with $(\mathbf{x}_0^{'}, ..., \mathbf{x}_k^{'})$ if necessary
12:    $(\mathbf{x}_0, ..., \mathbf{x}_k) \leftarrow$ choose to stay in $\mathbf{x}_0$ or follow best path
13: return $(\mathbf{x}_0, ..., \mathbf{x}_k)$
```

V. Simulations and Results

The simulation was implemented using matplotlib library. The 2D map has dimensions of 20X20. The robot is chosen as a point robot for the ease of simplicity, with no holonomic constraints. The first case, represents the branching of nodes for the initial robot and goal position. The second figure, shows the updated robot position which moves along the tree structure where the tree was determined using the first goal position, and also shows the new goal position as well. In the final figure, a new goal and robot position is shown. The nodes were rewired in between to find optimal path.

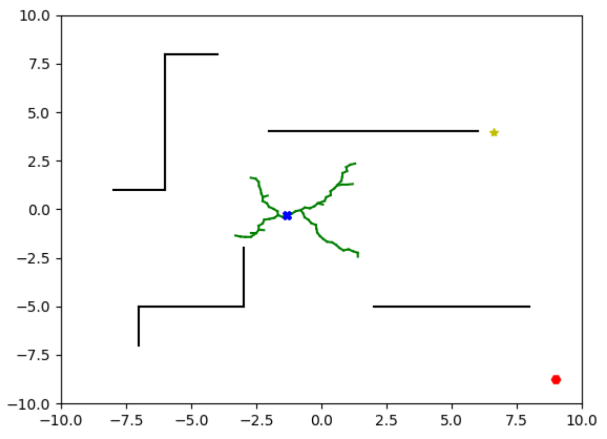


Fig. 2. Initial robot and goal position

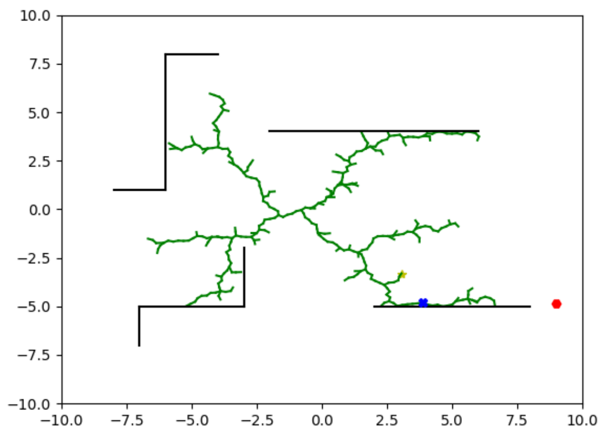


Fig. 3. Updated goal and robot positions

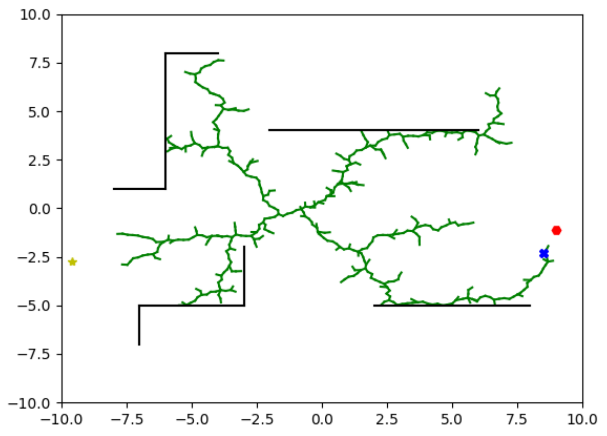


Fig. 4. Agent is near the new goal position

VI. Conclusion

In this paper, the real-time implementation of RRT* was completed. The real-time capability was achieved by interleaving the path planning with the tree expansion and rewiring. Furthermore, the storing of whole tree structure in between iterations, made certain that the algorithm is quick. The tree continues to grow until it covers the environment. Rewiring of the random nodes inside the tree was implemented.

The simulation results, shows that the tree retention and rewiring made possible to determine optimal paths for multiple-goal queries. The major drawback, this algorithm has is the large usage of memory for storing the entire tree for each iteration.

References

[1] Kourosh Naderi, Joose Rajamäki, and Perttu Hämäläinen. "RT-RRT*: A real-time path planning algorithm based on RRT". In: Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games. ACM. 2015, pp. 113–118.

[2] Shahram Azadi, Reza Kazemi and Hamidreza Rezaei Nedamani. "Vehicle Dynamics and Control".2021

[3] Karthik Karur, Nithin Sharma, Chinmay Dharmatti and Joshua E.Siegal. "A survey of Path planning Algorithms for Mobile Robots. 2021.

[4] Karaman. S., and Frazolli,.E "Sampling-based Algorithms for Optimal Motion Planning". 2011.

[5] Bruce.J and Veloso.M. "Real-time randomized path planning for robot navigation".2002

[6] Hatib, O. "Real-time obstacle avoidance for manipulators and mobile robots." The International Journal of Robotics Research. 1986