Sampling Based Path Planning Using RRT*N Algorithm

Sameer Arjun S
Maryland Applied Graduate Engineering
University of Marlyand
ssarjun@umd.edu

Rashmi Kapu
Maryland Applied Graduate Engineering
University of Marlyand
rashmik@umd.edu

Abstract

*This project aims to validate the RRT*N (Normalized) algorithm which is built on the RRT* algorithm and which uses a normalization function to improve the processing time required to generate the path. By restricting the area in which the algorithm's nodes can be generated, a net improvement in the speed is observed. A further step is taken where the restriction is controlled to either be in the form of a normal probability distribution or a uniform probability distribution. Both modifications are presented with the normal distribution resulting in significant improvements in the processing times.*

1. Introduction

A sampling-based algorithm is a type of algorithm used in robotics, motion planning, and other areas of artificial intelligence that deal with generating a sequence of feasible solutions based on sampling from a continuous or discrete space. In sampling-based algorithms, the configuration space, which is the space of all possible configurations of a system, is sampled to generate a set of potential solutions.

There are several types of sampling-based algorithms, including Probabilistic Roadmap (PRM), Rapidly exploring Random Trees (RRT), and their variants such as RRT*, and Hybrid A*. RRT stands for Rapidly-exploring Random Trees, an effective sampling probabilistic algorithm that generates the shortest path with the least cost. RRT* is a modified version of RRT which iteratively looks to rewire the tree to improve the costs of the path to the goal. RRT*N (Normalized) is an extension of RRT* where the nodes are generated in a specified workspace to reduce the computation time significantly without impacting the accuracy of the shortest path generated through the RRT* algorithm.

This paper is organized as follows: A brief literature review of the previous relevant research is presented in Section II. Then Section III, the RRT* algorithm is presented initially, and then the improvements of this algorithm called the RRT*N algorithm are presented. Finally, Section IV contains the results of the work and also mentions the potential future work.

2. Literature Review

The RRT* algorithm is found to be the most adaptable single-query sampling-based path planning algorithm for dynamic environments and non-holonomic constraints [3], because of which several researchers have worked to further refine the algorithm in the wake of the work by Karaman and Frazzoli in [2]. Some of the modifications included nonholonomic dynamic restrictions to specialize the method in certain applications [7, 8], while others [4, 5, 6] targeted the issues with lengthy processing times and memory usage.

Improvements to the method that don't take into account its inherent holonomic constraint property are known as holonomic RRT* approaches. These frequently try to boost memory efficiency, processing speed, or path quality. One of the major works using this approach was a memory-efficient RRT* variant known as RRT* Fixed Node (RRT*FN), whose major goal is to restrict the number of nodes that can be present in the tree at any one time, is also described in the work by Adiyatov and Varol [5]. Once this number is reached, a new node can only be created by removing an existing one. If the new node has a lower cost than a specific existing node in the tree, the existing node is deleted. This is done both locally and globally, resulting in a large decrease in memory usage at the expense of a lower-quality path.

Non-holonomic RRT* techniques, on the other hand, tries to modify the algorithm to produce pathways that are appropriate for mechanisms with kinodynamic restrictions, such as car-like robots or fixed-wing UAVs, as these robots must carry out complicated motions in order to follow a given path. In this space, the work by Webb and Berg in [7] and [8] is a good illustration of this because they developed Adapted RRT*, a variant of RRT* that takes kinodynamic restrictions into account. Their research has shown that their algorithm can offer the best trajectories for a robot

that resembles a car in five dimensions and an aerial vehicle in ten dimensions.

3. Methodology

This section of paper discusses the concepts of RRT* algorithm and later elaborates on the improvements made in the RRT*N algorithm and its working.

3.1. RRT* Algorithm

The RRT* algorithm is built upon the pre-existing Rapidly expanding Random Trees algorithm and differs from this in the manner in which the path smoothening happens to create an optimal path by reassigning the parents to the newly formed nodes in case a path with a lower cost is found. The configuration space, Z, is the first and most important piece of information. The definition of this space, which is closely related to the size of the space in which the robot will operate, is the set of all transformations that are potentially relevant to the robot. No nodes will be formed in the area of the configuration space that is occupied by obstacles, referred to as Zobs, because it is impassable for navigation. To make sure that no branch of the tree penetrates the zone Zobs, collision checks will also be run. On the other side, the remaining configuration space components, which make up Zfree's free space, are the region where the tree can develop.

RRT* algorithm will create a structure like a tree that starts at Zinit and looks for the quickest, least expensive route leading to Zgoal. Then, the exploration or expansion phase begins by creating a random node, Zrand, within Zfree. Then, Znear, the nearest node in the tree, is sought. A segment will connect Zrand and Znear if the distance between them is discovered to be within a predetermined step size, resulting in the creation of a new branch. However, if the distance between them exceeds the step size, the planner will create a new node, Znew, that is exactly one step size distant from Znear and is located in the direction of Zrand. Then the new node formation is checked for collisions and is made sure that Znew is in the feasible region.

After connecting two nodes, a check is made to see if rewiring the connection could result in a lower-cost parent node. A circle is formed around one of the nodes, and nodes within that circle are checked for a lower cost. If a lower-cost node is found, the connection is rewired. The neighbourhood radius is determined using the following equation:

$$r = \gamma \left(\frac{\log n}{n} \right)^{1/d} \tag{1}$$

where,

1. r = neighbourhood radius
2. γ = user-defined planning constant
3. n = dimension of the space
4. d = dimension of the configuration space

The process is repeated until the goal point is reached or the maximum number of nodes is reached. A goal area is typically defined instead of a goal point, often as a square or circle centred at Zgoal. This is shown in $figure - 1$ and the pseudo-code for the RRT* algorithm is also shown in $figure - 2$.

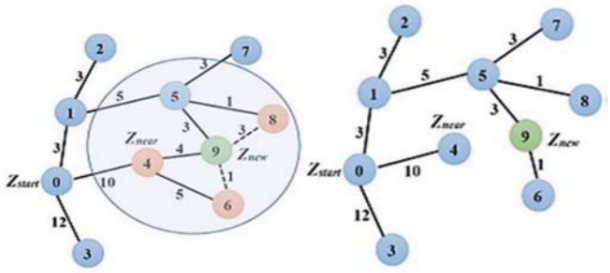

Figure 1. Rewiring process

Algorithm 1. T = (V, E) ← RRT*(z_{init})

```
T ← InitializeTree();
T ← InsertNode(Ø, z_init, T);
for i=0 to i=N do
    z_rand ← Sample(i);
    z_nearest ← Nearest(T, z_rand);
    (z_new, U_new) ← Steer (z_nearest, z_rand);
    if Obstaclefree(z_new) then
        z_near ← Near(T, z_new, |V|);
        z_min ← Chooseparent (z_near, z_nearest, z_new);
        T ← InsertNode(z_min, z_new, T);
        T ← Rewire (T, z_near, z_min, z_new);
return T
```

Figure 2. RRT* Algorithm Pseudocode

3.1.1 RRT*N

The RRT*N algorithm differs from the RRT* algorithm by changing the manner in which the random nodes are generated from a totally random process to a directed normal distribution function. The random points are generated within a standard deviation of the straight line connecting Zinit and Zgoal, represented by vector L. The standard deviation can be determined based on the distance between Zinit and Zgoal or the width of the largest obstacle when projected

normally to L. And concentrating the generated nodes towards Zgoal leads to faster processing times and slightly better path quality. Hence, the nodes are not randomly generated in the workspace but on vector L, with a normal distribution function used to determine the mean and standard deviation, which is a factor multiplied by the magnitude of the vector L. This process is explained in the pseudo-code of the RRT*N algorithm given in $figure - 3$

Algorithm 2. T = (V, E) ← RRT*N(z_{init})

```
T ← InitializeTree();
T ← InsertNode(Ø, z_init, T);
Generate L
for i=0 to i=N do
    l_rand ← Sample(i);
    z_rand ← Normal(l_rand);
    z_nearest ← Nearest(T, z_rand);
    (z_new, U_new) ← Steer (z_nearest, z_rand);
    if Obstaclefree(z_new) then
        z_near ← Near(T, z_new, |V|);
        z_min ← Chooseparent (z_near, z_nearest, z_new);
        T ← InsertNode(z_min, z_new, T);
        T ← Rewire (T, z_near, z_min, z_new);
return T
```

Figure 3. RRT*N Algorithm Pseudocode

The generated nodes are coordinates with values of (x,y) and this is fed of the normal function shown in the below equation:

$$P(c) = \frac{1}{2\pi} e^{-\frac{(c-\mu)^2}{2\sigma}}, \quad c = x, y \tag{2}$$

where,

1. P = Probability distribution
2. μ = Mean
3. σ = Standard deviation

The variable σ can be adjusted manually or automatically depending on the robot and obstacle sizes. This helps avoid local minimum issues caused by large obstacles blocking the path normal to vector L. When σ is very small, points will be generated along L, and when it's very large, RRT*N behaves like RRT* algorithm.

4. Results

The RRT*N algorithm was tested alongside the RRT* algorithm on similar obstacle spaces and also by providing the same initial and final node locations for the robot. This step was repeated for multiple iterations to validate the data obtained in terms of processing time to reach the goal and also to converge to an optimal path. This was also tested using dynamic obstacles which were introduced into the pre-existing obstacle map, to check the efficiency of this algorithm in this dynamic environment. It defers from RRT* algorithm in the way random nodes are generated. A set of n (in our case n=1000) points are generated randomly in a normalised distribution (σ is taken as 90 based on the step length and obstacle size in our case) around the line joining the start and goal nodes. A random node is picked among the generated n nodes, which is now our Zrand. The collision detection and canvas parameters are checked the same way they are, in RRT* algorithm. The obstacle map was defined and later the same start and goal coordinates were fed as inputs for both algorithms. The same scenario was tested with the condition of a circular dynamic obstacle induced as well. The results of the path planning using RRT* algorithm are presented in $figure - 4$ and the results using RRT*N algorithm are presented in $figure - 5$.

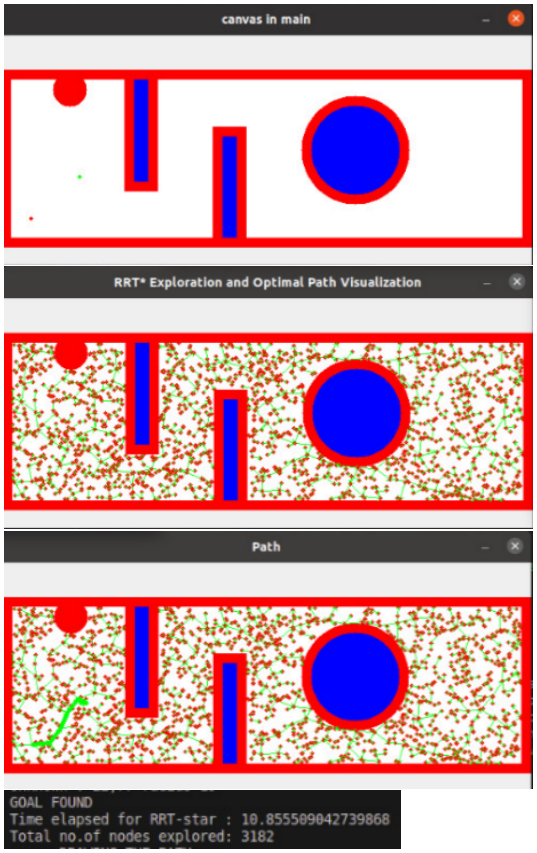

Figure 4. RRT* Algorithm test case

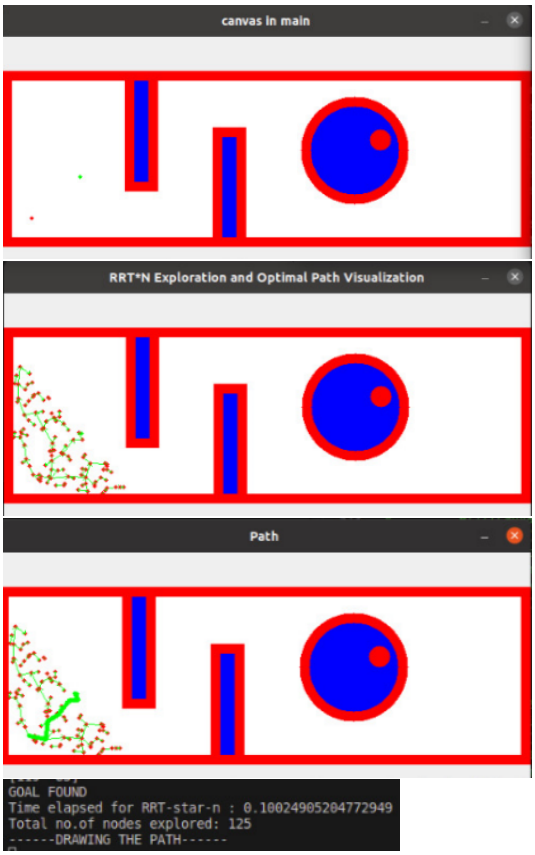

Figure 5. RRT*N Algorithm test case

The algorithms were tested in a non-dynamic environment as well with the same initial and final node locations which yielded the following results as highlighted in $figure - 6$.

The time taken to solve the paths was averaged for a finite dataset for both the algorithms and the results showed that RRT*N had an approximate 30 percent decrease in the computational time required to find the goal node compared to RRT*. However, it was observed that there was no significant improvement in the path generated by RRT*N compared to the regular RRT* algorithm. Additionally, the value of σ considered in the calculation of the normal distribution function also played an important role when there was a significant increase in the size of the obstacle, hence this is a parameter which needs to be varied based on the size of the largest obstacle in the map to get optimal results from the RRT*N algorithm.

5. Conclusion

The discussed RRT*N algorithm is able to optimally find the goal node from any given start node and also provides a much faster solution time compared to RRT* algorithm. The improvement in the generation of random nodes by directing them towards the goal node proves much better than generating them in a random approach. However, this does not improve the quality or cost of the path generated. This can be enhanced by using it along with path smoothening or other efficient techniques. Additionally, there is scope to extend this research to make the algorithm applicable to 3D space applications which is planned to be implemented in the future.

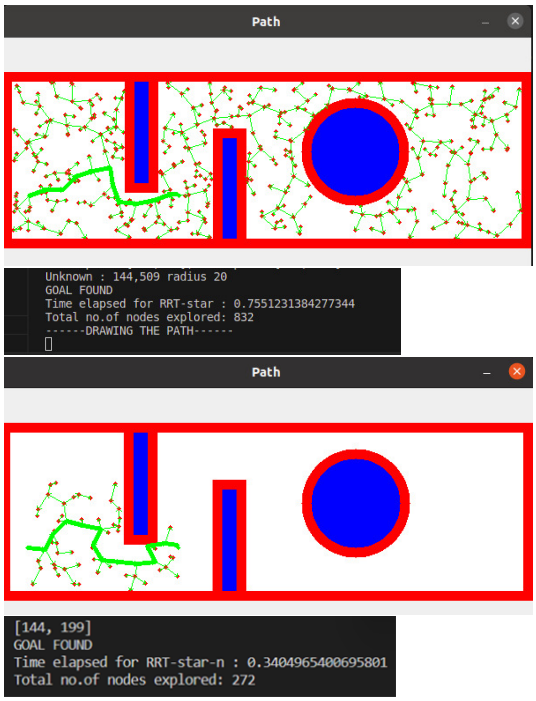

Figure 6. Non-dynamic obstacle space

6. References

1. S. M. LaValle and J. J. Kuffner, “Rapidly-exploring random trees: Progress and prospects”, Algorithmic and Computational Robotics: New Directions, pages 293-308. A K Peters, Wellesley, MA, 2001

2. S. Karaman, and E. Frazzoli, ”Sampling-based algorithms for optimal motion planning”, Int J Rob Res, vol. 30, pp. 846-894, 2011.

3. J. J. Kuffner, and S. M. Lavalle, ”RRT-connect: An efficient approach to single-query path planning”, in Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2000, pp. 1-7.

4. S. Karaman, M. Walter, A. Perez, E. Frazzoli, and S. Teller, ”Anytime motion planning using the RRT*”, presented at the IEEE International Conference on Robotics and Automation (ICRA) 2011.

5. O. Adiyatov, and H. A. Varol, "Rapidly-exploring random tree based memory efficient motion planning", presented at the IEEE International Conference of Mechatronics and Automation (ICMA), 2013.

6. 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 IEEE RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, 2014, pp. 2997- 3004.

7. D. J. Webb, and J. V. D. Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics", presented at the IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, 2013

8. D. J. Webb, and J. V. D. Berg, "Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints", arXiv:1205.5088v1, 2012.