RRT*N: An Improved Rapidly-Exploring Random Tree Approach for Reduced Processing Times

Hussein Mohammed, Mohammad Jaradat, and Lotfi Romdhane
Mechatronics Graduate Program, College of Engineering
American University of Sharjah
Sharjah, United Arab Emirates

In this paper, we aim to improve on the processing time taken to run the Rapidly-Exploring Random Tree Star (RRT*) algorithm with no regard for the path length. By restricting the area or volume in which the algorithm's nodes can be generated, a net improvement of the speed is observer. A further step is taken where the restriction is controller 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.

Keywords—RRT; RRT*; path planning; optimal path; robot; navigation

I. INTRODUCTION

In the field of navigation and path planning, the aim is to find an obstacle free path between the starting point and the goal point with the least possible cost, which corresponds to the shortest, smoothest, and the most dynamically possible path. Recent research has been mostly taking up RRT*, which is a single query sampling-based path planning approach introduced by Karaman and Frazzoli [1], mainly due to its probabilistic completeness, flexibility, and lower cost when operated in higher dimensions as opposed to grid based algorithms [1]. Their work on the RRT* yielded a substantial improvement compared to the work done by LaValle on the RRT in [2]. Indeed, the RRT* algorithm can rewire itself resulting in paths that are much closer to the theoretical optimum paths. The aim of this paper is to present a further improvement on the RRT* algorithm in terms of computational time with a slight improvement on the path quality.

The organization of this paper is as follows: a concise literature review of the relevant research is presented in Section II. Then a discussion of the RRT* methodology is presented in Section III along with a detailed description of the contribution of this paper. Section IV contains the obtained results along with a brief discussion. Finally, Section V contains our conclusion along with future recommendations.

II. LITERATURE REVIEW

Among the single-query sampling based path planning algorithms, RRT* is found to be the most flexible for dynamic environments and non-holonomic constraints [3] and as a result, following the work done by Karaman and Frazzoli in [1], several researchers worked to further improve the algorithm some of which are cited in this paper. Some of the improvements were targeting the long processing times and memory consumption issues [4, 5, 6] while other modifications incorporated non-holonomic dynamic constraints to specialize the algorithm in certain applications [7-10, 14].

Holonomic RRT* approaches are those where improvements aim to add upon the algorithm itself without addressing its default holonomic constraint property. These typically aim for improvements on the processing times, path quality, or memory efficiency.

For instance, in [4] Karaman et al. proposed a modification of the RRT*, called Anytime RRT*, which is an online incarnation of RRT* where the algorithm is limited to run for a predefined amount of time. The goal of this approach is to deal with the large computation time usually faced with RRT*. Anytime RRT* would provide a quick yet relatively unoptimized initial path which would be optimized for the remainder of the duration.

Also, in the work done by Adiyatov and Varol [5] a memory efficient version of RRT* is introduced called RRT* Fixed Node (RRT*FN) where the main idea is to limit the number of nodes that can be present in the tree at a given time. This means that once this limit is reached, a new node can only be added by deleting a previous node. The deletion is done if the new node offers better cost than a certain existing node in the tree. This is done on both the local and global scales resulting in significant reduction in memory consumption at the cost of reduced path quality.

Another version of RRT* was presented by Gammell et al. in [6] called Informed RRT*. Like the name suggests, this version of RRT* has extra information known to it, specifically the path generated from each prior iteration. This means that the regular RRT* algorithm runs first to generate a path; this path is then utilized to form an ellipsoid which determines the space in which the next trial of the algorithm will run. This keeps running resulting in consecutively smaller and smaller ellipsoids eventually resulting in a highly optimized path at the cost of higher computational time.

Meanwhile, non-holonomic RRT* approaches are those that aim to adapt the algorithm to provide paths that are suitable for mechanism with kinodynamic constraints such as car-like robots or fixed-wing UAVs since such robots need to perform complex motions to follow a certain path.

An example of this is the work done by Webb and Berg in [7] and [8] where they came up with a version of RRT* that allows for kinodynamic constraints called Adapted RRT*. Their work showed that their algorithm can provide optimum trajectories for a 5-dimensional state space car-like robot and a 10-dimensional state space aerial vehicle.

Also, work done by Lee et al. in [9] demonstrates their spline-based version of RRT* which they called SRRT* that allowed for the algorithm to be extended to fixed-wing aerial vehicles flying along 3D trajectories. Similar work was done by Alejo et al. in [10] where they present RRT*i, a version of RRT* that is optimized for UAVs where an initial path is generated like the working of RRT* but then the path is refined using Gaussian distributions and node rejection to allow for smoother more viable trajectories.

III. MEHTODOLOGY

This section will discuss the concepts of RRT* and its operation. It is necessary to demonstrate how the original algorithm operates to properly compare it to the optimized version being introduced in this paper.

A. Initialization

The first and most critical piece of information is the configuration space, Z. This space is defined as the set of all possible transformation that are applicable to the robot; this is closely related to the dimension of the space in which the robot will operate [11]. The portion of the configuration space occupied by obstacles, denoted Zobs, is inaccessible for navigation hence no nodes will be generated in that region. Moreover, collision checks will be performed to make sure that no branch of the tree enters the region Zobs. On the other hand, the remaining elements in the configuration space, which constitute the free space, Zfree, is the area in which the tree can grow. The starting point, Zinit, and the goal point, Zgoal, which are in the free space are then defined.

B. Execution

RRT* will construct a tree-like structure that originates from Zinit and attempts to find the shortest, least cost path leading to Zgoal. The exploration or expansion process then proceeds starting with the generation of a random node, Zrand, that is within Zfree. The nearest node in the tree, Znear, is then searched for. If the distance between Zrand and Znear is found to be within a predefined step size, a segment will connect them, hence creating a new branch. If, however, the distance between them is larger than the step size, the planner will generate a new node, Znew, that is in the direction of Zrand which is exactly one step size away from Znear. At this point, a collision check is done to make sure that Znew is feasible, and if so, the two will be connected. Fig. 1. shows this process:

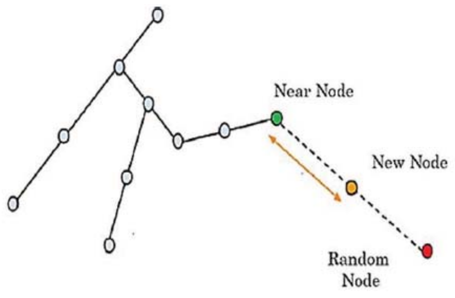
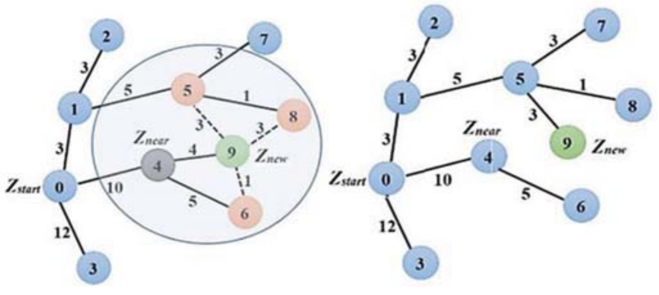


Fig. 1. RRT* algorithm expansion process [3]

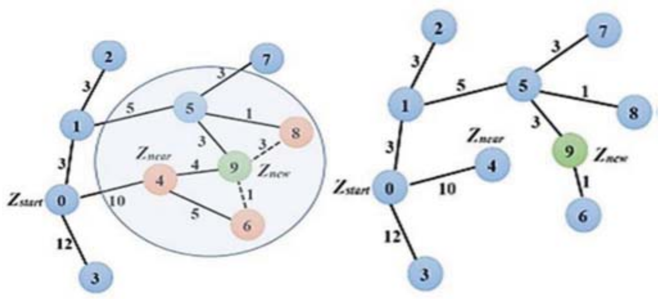
Once two nodes are connected, a second check is made to see whether it is possible to rewire the most recent connection such that Znew would have a lower cost parent node. This is done by defining a parameter called neighborhood radius which basically forms a circle with its center being Znew where all the nodes within this circle are checked whether they would be lower-cost parents for Znew. If a node is found to be of lower cost, called Zmin, than the current parent node, the last connection will be redone such that Znew is now connected to Zmin rather than Znear. The neighborhood radius is usually found using the following relation:

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

Where n is the dimension of the space, d is the dimension of the configuration space (both are the same in most cases) and γ is a user-defined planning constant. The neighborhood radius is typically two to three times the step size to allow for effective rewiring of the tree branches. In all tests demonstrated in this paper, the neighborhood radius is 2.5 times the step size. The process described above can be seen in Fig. 2.



(a) Find near neighbours (b) Select best parent



(c) Check for cost rewiring (d) Rewired tree

Fig. 2. RRT* neighbor node search and tree rewiring [12]

This process is repeated until either the goal point is reached, or the specified number of nodes is used up. It is extremely rare for the tree to reach the goal point hence usually a goal area is defined instead of a goal point, which could either be a square or a circle whose center is Zgoal. In this paper, the goal area is a circle with a center Zgoal and a radius "Vision" as a reference to a robot's range of vision. A pseudocode is shown in Table. I to summarize the working of RRT*.

TABLE I. RRT* ALGORITHM PSEUDOCODE [13]

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

```
1 T ← InitializeTree();
2 T ← InsertNode(Ø, $z_{init}$, $T$);
3 for $i$=0 to $i$=$N$ do
4     $z_{rand}$ ← Sample($i$);
5     $z_{nearest}$ ← Nearest($T$, $z_{rand}$);
6     ($z_{new}$, $U_{new}$) ← Steer ($z_{nearest}$, $z_{rand}$);
7     if Obstaclefree($z_{new}$) then
8         $z_{near}$ ← Near($T$, $z_{new}$, $|V|$);
9         $z_{min}$ ← Chooseparent ($z_{near}$, $z_{nearest}$, $z_{new}$);
10        $T$ ← InsertNode($z_{min}$, $z_{new}$, $T$);
11        $T$ ← Rewire ($T$, $z_{near}$, $z_{min}$, $z_{new}$);
12 return T
```

C. Proposed Approach

While working with RRT*, it is apparent that is significantly improved when compared to RRT or other sampling-based planning algorithms, mainly due to its optimizations on path quality and processing times that are greatly attributed to its probabilistic completeness [1]. Despite this, it remains clear that RRT* can still be significantly improved for scenario specific performance.

In this case, it is desired to reduce the processing time by influencing the way the nodes are generated, specifically by changing it from completely random to a directed normal distribution function. This means that points are only generated within a specified standard deviation of the straight line connecting Zinit and Zgoal, donated by vector L. The standard deviation parameter could either be set based on the value of the distance between Zinit and Zgoal or based on the width of the largest obstacle in the way when projected normal to L.

Doing so will concentrate the nodes generated such that there is a much higher tendency that the branches grow towards Zgoal. This results in significant decreases in the processing times with slight improvements on path length and quality.

So, the major modification is that nodes are not generated at random in the workspace itself but rather on the vector L and then are fed into a normal distribution function that takes it as the mean value in the bell curve and the standard deviation is defined as a factor multiplied by the magnitude of the vector L. The modified algorithm, RRT* Normal, referred to as RRT*N from now onwards is shown in Table II.

TABLE II. RRT*N ALGORITHM PSEUDOCODE

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

```
1 T ← InitializeTree();
2 T ← InsertNode(Ø, $z_{init}$, $T$);
3 Generate L
4 for $i$=0 to $i$=$N$ do
5     $l_{rand}$ ← Sample($i$);
5     $z_{rand}$ ← Normal($l_{rand}$);
6     $z_{nearest}$ ← Nearest($T$, $z_{rand}$);
7     ($z_{new}$, $U_{new}$) ← Steer ($z_{nearest}$, $z_{rand}$);
8     if Obstaclefree($z_{new}$) then
9         $z_{near}$ ← Near($T$, $z_{new}$, $|V|$);
10        $z_{min}$ ← Chooseparent ($z_{near}$, $z_{nearest}$, $z_{new}$);
11        $T$ ← InsertNode($z_{min}$, $z_{new}$, $T$);
12        $T$ ← Rewire ($T$, $z_{near}$, $z_{min}$, $z_{new}$);
13 return T
```

The algorithm is mostly the same apart from how the nodes are generated and added to the tree. As mentioned, the sample generated is now a point along the vector L. This is then fed to the normal function shown in equation (2).

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

As mentioned, the node returned by "Sample", Lrand, has its x and y components fed to the "Normal" function which calculates the probability distribution and then decides the location of the x and y components of Zrand respectively. Also, σ is the desired standard deviation and μ is the mean; the mean is taken as zero in all our trials. Fig. 3 shows the probability distribution for the placement of Zrand for a standard deviation value of 1.

The variable σ can be varied either manually or automatically through modifying the algorithm. Its modification is influenced by the size of the robot and/or the size of the largest obstacle along L. By doing so we easily avoid local minimum issues where the tree would get stuck in front of an obstacle due to its large size blocking the path normal to vector L. When σ is very small, all the points will be generated along L itself and similarly when it is very large, RRT*N will show behavior identical to that of RRT*.

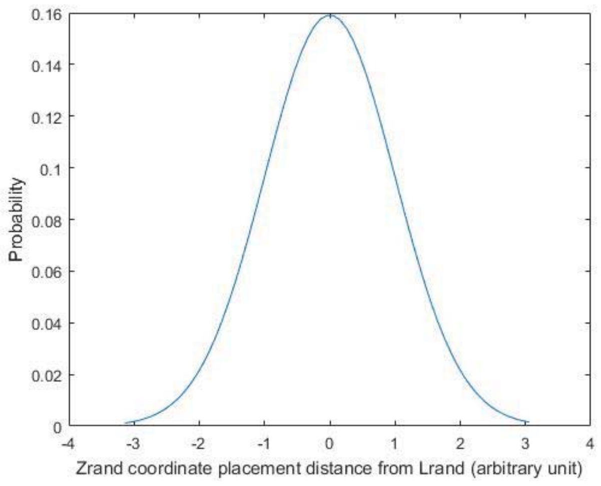


Fig. 3. Zrand placement probability distribution

Another parallel modification made was to apply the same concept but with a uniform node distribution within a predefined maximum distance from L, this results in what appears like a car lane. What this means is that the nodes are generated the same way as they are in RRT* but are strictly limited by a maximum normal distance away from L. It was seen to be inferior to its normal distribution counterpart in our specific scenario because this approach fails quickly when tested in extreme situations unless the width of the "lane" is significantly increased in which case it does not offer much improvement over RRT* but is thought to be relevant to the work done by [14]. Fig. 4, Fig. 5, and Fig. 6 demonstrate how a typical 2500 node trial looks like in 2D using RRT*, RRT*N, and RRT* Uniform respectively with all having the same parameters. It is to be noted that the algorithm will stop once it reaches within Vision of Zgoal otherwise it continues until all nodes are used up.

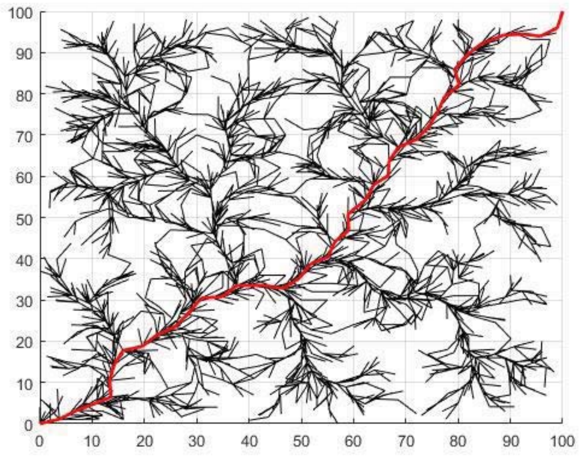


Fig. 4. RRT* typical trial in environment with no obstacles

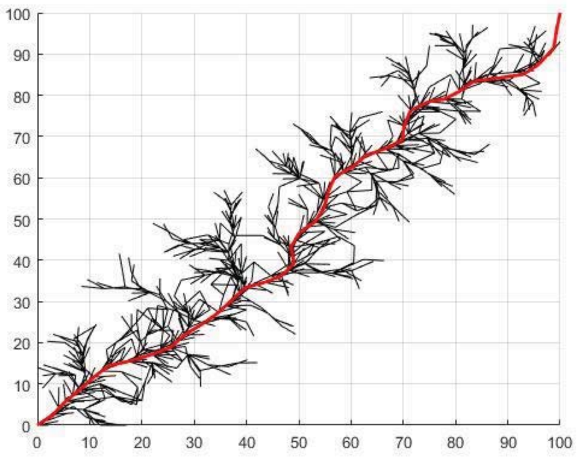


Fig. 5. RRT*N typical trial in environment with no obstacles

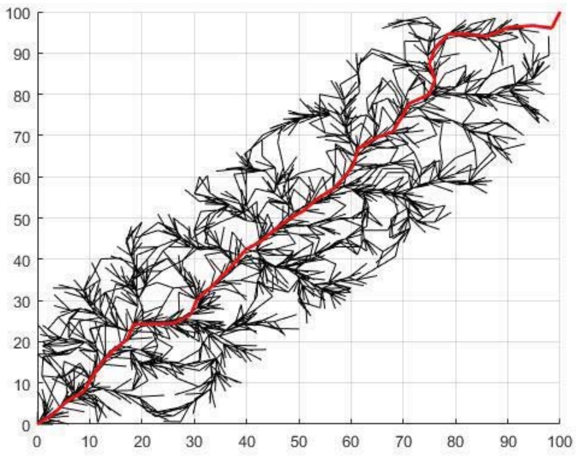


Fig. 6. RRT* Uniform typical trial in environment with no obstacles

The figures above are simply to demonstrate how an experiment with each algorithm looks like, more about the specifics will be discussed next. The step size used is 2 units while the neighborhood radius used is 5 units. The Vision used is 5 units. In the case of RRT*N, 4 standard deviations of the nodes (±2σ) are generated within 15% of the magnitude of L while for RRT* Uniform all the points are within 15% of the magnitude of L. Again, RRT* Uniform is not the focus of this paper and hence will not be analyzed further.

Next, the specifics of how RRT*N compares to RRT* will be discussed. For the comparison, a specific environment with fixed obstacles will be used. The environment used is taken from [1] to replicate the scenario used and is shown in Fig. 7. along with the optimum path obtained using visibility graph method.

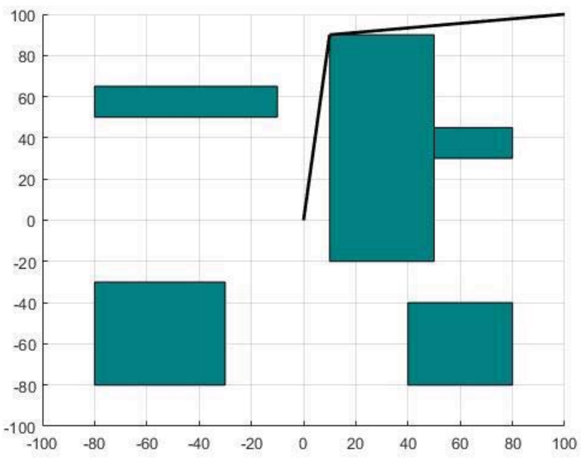


Fig. 7. Shortest possible path in environment with obstacles

For reference, the length of the path shown is 181.108 units as this will be compared to the mean path length obtained by both algorithms. RRT* and RRT*N were each executed 250 times with a node limit of 5000 in each trial. All the parameters are the same as previously however for RRT*N, 4 standard deviations of the nodes (±2σ) are now generated within 45% of the magnitude of L to account for the obstacles. One new parameter will be seen, called success rated, which means the percentage of the trials where the tree was able to generate a path within Vision distance of the goal, basically reaching the goal area.

IV. Results and Discussion

A typical run for each algorithm in the scenario presented in Fig. 7 is shown in Fig. 8 and Fig. 9

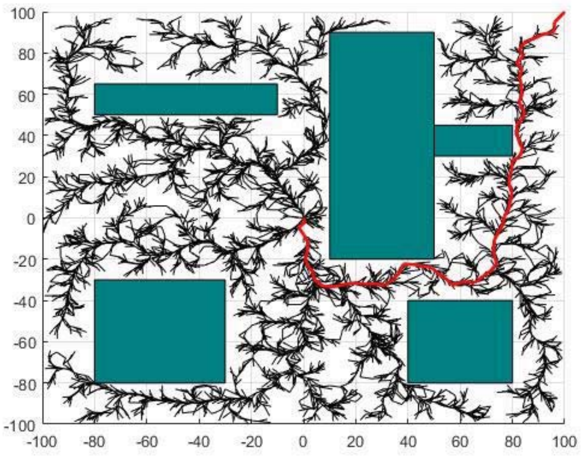


Fig. 8. RRT* typical trial in environment with obstacles

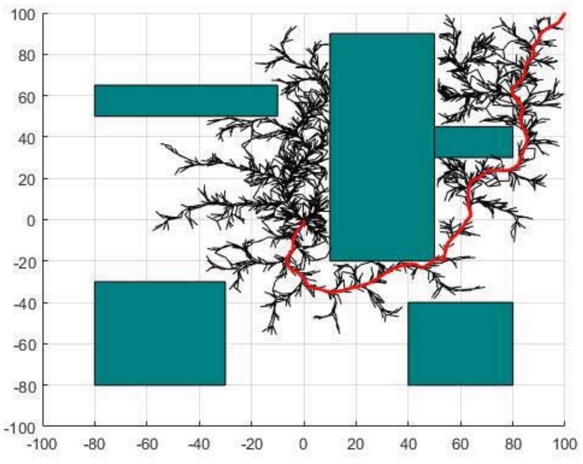


Fig. 9. RRT*N typical trial in environment with obstacles

It is immediately apparent that RRT*N is more likely to reach Zgoal quicker due to its probability distribution having a higher tendency of driving its branches towards the goal region. This also contributes to it not favoring exploring distant regions in the map which typically is time consuming and reflects negatively on the path quality. However, in both cases, the path was not along the direction of the shortest path mainly due to the relatively small number of nodes used in those trials which goes to show that RRT*N does not contribute much towards the quality or the cost of the path as it does with the computational time. The data collected from the 250 loops for each algorithm regarding CPU times and path lengths are shown in Fig. 10 and Fig. 11.

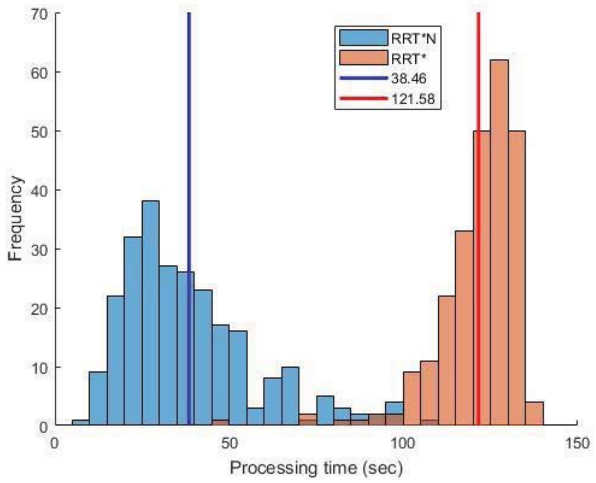


Fig. 10. RRT*N and RRT* algorithms processing times histogram

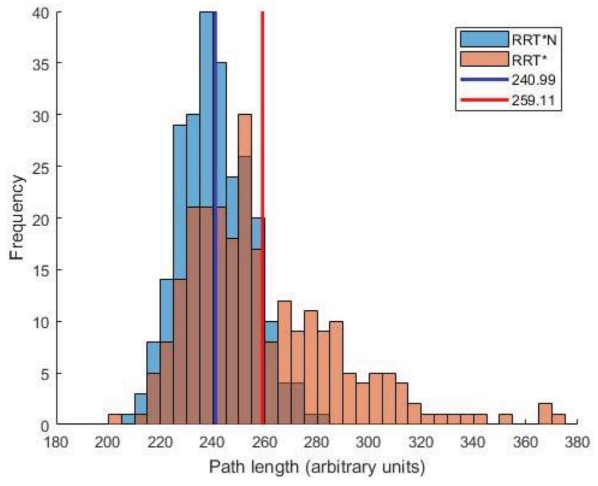


Fig. 11. RRT*N and RRT* algorithms path lengths histogram

Table III summarizes the results of all 250 trials listing all parameters and criteria.

TABLE III. RRT* AND RRT*N COMPARISON

Parameter	**RRT***	**RRT*N**
Average time (s)	121.575	38.456
Time standard deviation (s)	11.416	19.176
Average length (arbitrary)	259.105	240.992
Length standard deviation (arbitrary)	29.406	13.322
Success rate	7.2%	94.0%

As seen from the table, RRT*N significantly improves on the CPU time on average (31.6% of RRT* time) with trade off in standard deviation (67.8% increase). Also, path length in RRT*N is improved as it is 33.1% longer than the optimum path of length 181.108 discussed previously while RRT* is 43.1% while having a significantly lower standard deviation (45.3% of RRT*). Another huge indicator is the success rate, with RRT*N having a success rate of 94% (235 out of 250 trials) and RRT* having a 7.2% success rate (18 out of 250 trials) in this scenario.

V. CONCLUSION

RRT* is a powerful path planning algorithm but there is a vast amount of improvements to be done as shown in this paper. The presented results showed that the RRT*N is roughly 3 times faster than the regular RRT* in terms of processing time. This work is currently being extended to 3D and will soon be tested using a Kobuki to provide further analysis. The final step in this research would be extending it to account for dynamic obstacles as well.

REFERENCES

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

[2] 2S. 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.

[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.

[9] Lee, H. Song, and D. H. Shim, "Optimal path planning based on spline-RRT* for fixed-wing UAVs operating in three-dimensional environments", presented at the 14th International Conference on Control, Automation and Systems (ICCAS 2014), Korea, 2014.

[10] Alejo, J. A. Cobano, G. Heredia, J. R. Martínez-De Dios, and A. Ollero, "Efficient trajectory planning for wsn data collection with multiple UAVs", in Cooperative robots and sensor networks. vol. 604, ed: Springer International Publishing, 2015, pp. 53-75.

[11] S. M. Lavalle, Planning algorithms: Cambridge University Press, 2006.

[12] Noreen, A. Khan, and Z. Habib, "A comparison of RRT, RRT* and RRT*-smart path planning algorithms", IJCSNS, vol. 16, pp. 20-27, 2016.

[13] Noreen, Iram & Khan, Amna & Habib, Zulfiqar. (2016). Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions. International Journal of Advanced Computer Science and Applications. 7. 10.14569/IJACSA.2016.071114.

[14] X. Lan, and S. Di Cairano, "Continuous curvature path planning for autonomous vehicle maneuvers using RRT*", presented at the European Control Conference (ECC), 2015.