Rapidly-exploring Random Trees for UAVs in an Urban Environment

Carter Berlind[1], Adam Rozman[1], and Charles DeLorey[1]

***Abstract*— Tree-based exploration and path-planning algorithms have shown value in producing fast and efficient results. Specifically, Rapidly-exploring Random Trees (RRT) and its asymptotically optimal variant, RRT*, produce favorable results even in cluttered, obstacle-rich environments. In this paper, we apply these properties to Unmanned Aerial Vehicles (UAVs), specifically quadrotor drones and examine their performance compared to other navigation and path-planning algorithms in use today. The developed algorithm will be evaluated based on the overall length of the path found as well as a drone's ability to follow it. These algorithms have been widely adopted in recent UAV navigation, and we show how they are able to effectively navigate 3D environments.**

In this work, we present a review of recent works in the fields of path planning and drones. Next, the RRT and RRT* algorithms will be explained, then the implementation of these algorithms into an arena designed to mimic the workspace of an aerial drone in a cluttered environment. Then, we show the performance of our algorithm on a simulated drone with obstacles and evaluate the ability of the drone to safely complete the path. Finally, we discuss the significance of our work, strengths, and weaknesses, and identify avenues for future work and development.

I. Introduction

Tree-based path-finding algorithms, for a multitude of reasons, have become increasingly widely used in recent literature. There is a multitude of qualities that have made them desirable for both a basis to develop new tree-based search algorithms, as well as effective off-the-shelf path planners that can be used and implemented in a variety of different systems. Among the qualities that are desirable in these algorithms, specifically in Rapidly-Exploring Random Tree (RRT) algorithms, are speed, probabilistic completeness, and ease of re-planning. These algorithms have been altered to make them more compatible with a variety of different systems, such as incorporating additional connections as found in Rapidly-exploring Random Graphs (RRG), adding a re-wiring step as seen in RRT*, and kinodynamic expansions based on the physical properties and control of the robot as seen in Kinodynamic RRT.

In this paper, we focus specifically on the use of these algorithms with quadrotor drones. These drones have seen a great deal of use in both industry and literature due to their ability to easily move freely in large, three-dimensional environments. This capability of drones makes them ideal for applications such as surveillance, aerial photography, and a variety of robotic tasks. This extensibility and broad adoption make them ideal candidates for work exploring path-finding algorithms. To evaluate the performance of the RRT* implementation, it is compared to a base RRT algorithm using implementations from the MATLAB UAV Toolbox.

In the following sections, we will analyze the literature describing and implementing the RRT algorithm and its variations, define the steps of the algorithm and its mathematical properties, and discuss our method for implementation and evaluation when compared both to its variation as well as other planning methods.

II. Background and Prior Work

The appropriateness of path-planning algorithms depends on the planning problem such as configuration space, environment representation, and how robust the solution must be. Path planning algorithms can generally be categorized into three categories: potential-based methods, discretization-based methods, and sampling-based methods. Potential-based methods minimize the path through a space, given attractive potential functions drawing them towards the goal in conjunction with repulsive potential functions driving them away from obstacles [1]. These are realized with potential functions like control barrier functions (CBF) and control Lyapunov functions (CLF). Discretization methods first create a graph representation of the environment, perform a graph search algorithm (such as A*), and then return the solution to the original configuration space [2]. Sampling-based methods plan by sampling points in space and progressively generating maps. For this reason, sampling-based methods are well-suited to real-time path planning and generation as they do not require full understanding of the configuration space. Perhaps the most basic and unfocused version of a sampling-based method is the probabilistic roadmap [3].

The Rapidly-exploring Random Tree algorithm (RRT) family has been adopted for path planning in both academic literature and industry. This collection of algorithms is a versatile tool for a variety of different systems and environments. In recent years, RRT algorithms have been implemented for low-level planning for autonomous driving, aerial navigation, and systems with disparate dynamics and goals.

RRT is a sampling-based planning method introduced by Steven LaValle in 1998 [4]. The algorithm was designed to handle nonholonomic systems, making it applicable to many real-world systems. Sampling-based planning methods are probabilistically complete, that is to say, the algorithm will find a solution given an infinite runtime if one exists. LaValle introduced the algorithm for kinodynamic planning which accounts for the dynamics as well as kinematics of a system [5]. Because RRT does not need explicit information on obstacles in the configuration space, it can deal with dynamic or cluttered environments and high-dimensional

This work was not supported by any organization

[1]All authors are with the Department of Mechanical Engineering, Boston University, 110 Cummington Mall, MA 02215, United States

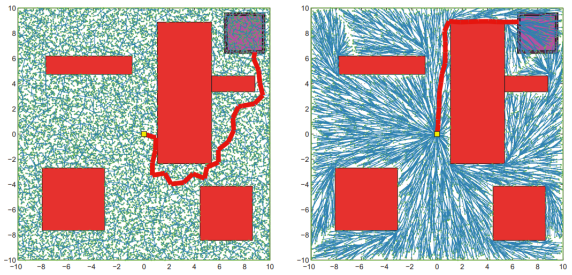


Fig. 1: Path comparison of RRT (left) and RRT* (right) [2]

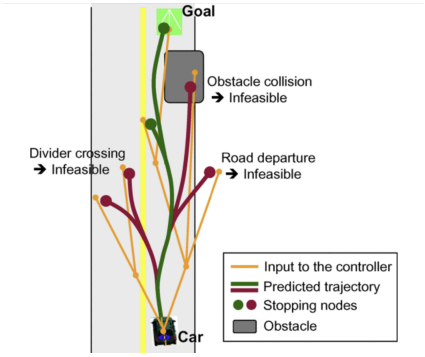


Fig. 2: RRT Expanded for Autonomous Driving [4]

complex problems. However, the solution generated by basic RRT is far from optimal. Due to its ease of implementation, computational efficiency, and overall effectiveness, the fundamentals of RRT are still relevant today and have been used to build improved methods.

In 2011, Karaman and Frazolli introduced a new variant of the algorithm called RRT* which produces an asymptotically optimal path by breaking weak branches of the tree and replacing them with more efficient ones whenever a node is added [6]. As the algorithm is asymptotically optimal, it will converge to an optimal solution given enough iterations. Figure 1 shows a comparison of the paths generated by RRT and RRT* using the same sample sequence. The vertices are the same, only the way the edges are connected differs (i.e. they are "rewired") and is far closer to the optimal path. Given the drastic improvement of RRT* over RRT, RRT* has been the focus of research to further improve the performance and convergence of the path planning algorithm. Convergence is improved and memory use is lowered by using heuristics like connectivity region, goal-biased boundary sampling, concentrated sampling (constrain new nodes to vicinity around initial path), and node rejection (reject high-cost nodes). Noreen *et al.* [7] introduced an improved version of RRT* in 2018 called RRT*-adjustable bounds (RRT*-AB). Kalisiak and van de Panne [8] developed RRT-blossom, which improved the performance of RRT on both highly-constrained as well as loosely-constrained environments. Otte and Frazzoli [9] introduced RRT^X, an asymptotically optimal path planning algorithm for dynamic environments in real time. To improve the convergence of RRT* using an ellipsoidal heuristic, Informed RRT* was developed by Gammell *et al.* [10].

A. Application to Autonomous Driving and UAV Navigation

Autonomous driving is another application to see the adoption of these algorithms for planning. While high level route planning is often done with A* and Dijkstra algorithms, there is still a need for low level planning such as changing lanes and avoiding cars on freeways. Papers such as Kuwata *et al.* [11] detail the use of RRT for such purposes. Figure 2 is an image from this paper displaying how this might be implemented for an obstruction on a highway. Their use in these systems was adopted due to the speed at which one is able to obtain a path that is not in collision with any other cars or obstacles and the ease in applying rules such as lane maintenance. In addition, for systems such as this it is relatively easy to apply kinematic constraints such as that of a Dubins vehicle [12]. These constraints make it easy for vehicles to follow any trajectory generated by the RRT algorithm.

B. The 3D Problem

Some problems arise when applying RRT algorithms to 3D spaces because of their slow convergence to optimality. An approach to mitigating this problem is to introduce biased sampling, an artificial potential field method, or a combination of both such as demonstrated by Wu *et al.* [13] and Fu *et al.* [14]. This is implemented by adjusting the random sample point by pulling it toward the direction of the tree, allowing trees to better explore different directions in 3D. Potential-guided RRT methods can be restrictive as they are best suited for obstacles that are spherical or cylindrical and would be inappropriate for modeling buildings in an urban environment. The high-level sum of these approaches can be described simply as RRT with heuristics. There have been many other modified versions of the RRT* algorithm proposed for UAV path planning [15]–[18]. Killian and Backhaus [19] provided experimental implementation using a drone's onboard sensors to generate a live occupancy map. RRT* was used to generate a path keeping a safety distance from detected obstacles. Their drone was able to successfully navigate a terrain with columns and a U-shaped obstacle very close to the optimal path. However, real vehicles will experience an amount of drift and need a controller to keep on their path. The experiment indicated that the drone's sensors were the limiting factor of this algorithm as at a high speed the drone was unable to detect the obstacle which resulted in a crash.

III. RRT, RRG, and RRT* Algorithms

The RRT algorithm is a sampling-based algorithm effective for exploring non-convex environments. The algorithm functions by building a tree of nodes starting from an initial point in the environment until the goal is reached. Starting from the origin, nodes are found by randomly generating a point within the environment which determines the direction of travel. The new node is added at a "resolution" increment in this direction from the nearest existing node if the point does not collide with any obstacles. The new node and closest node

are connected with an edge. This process repeats, generating a tree until a node is generated within a tolerance of the goal. The RRT* algorithm is an extension of RRT which applies a lowest-cost connection to the new node and a rewiring step whenever a new node is added. These steps work by considering neighboring nodes within a search radius of the new node. Each is evaluated to find the distance from the origin to the new node, and the shortest connection is made. The "rewiring" step is carried out by considering whether the cost of any neighboring node would be lowered by connecting it through the new node. If it is found to be cheaper, that connection is made, and the previous edge which the neighboring node was connected to is trimmed. With sufficient iterations, this approaches the optimal path to the goal.

RRT algorithms are probabilistically complete algorithms that returns a path if one exists. The algorithm samples the free configuration space, $\mathcal{C}_{free}$, and expands in the direction of the sampled configuration in order to build a tree that will eventually connect a starting and goal region. For this case, our free configuration space and the task space $\mathcal{X}$ are equivalent, meaning $x_{start} \in \mathcal{X}_{free}$ and $\mathcal{X}_{goal} \subset \mathcal{X}_{free}$ are additionally equivalent to their configuration space representation. We include obstacles such that $\mathcal{X}_{obs} \subset \mathcal{X}$ are excluded from $\mathcal{X}_{free}$. The construction of the RRT algorithm also ensures that edges connecting sampled points exist exclusively in $\mathcal{X}_{free}$ ensuring that paths found by the algorithm never come into contact with the obstacles.

In addition, we also define $u \in U$ as a control input. For our most simple case, we trivialize this as an expansion in the direction of the node sampled in $\mathcal{X}_{free}$ with resolution step Δx. The control, u, is applied at the nearest node that exists in tree, $\mathcal{T}$, which contains the previous nodes and edges found through expansion. Expansion begins from x_{start} which is appended as the first node in $\mathcal{T}$. Once this expansion adds a node $x_{new} \in \mathcal{X}_{goal}$ to $\mathcal{T}$, the algorithm returns a tree which now contains a path from x_{start} to $\mathcal{X}_{goal}$. The steps to generate a tree with K nodes are shown in Algorithm 1 from LaValle, 1998.

Algorithm 1 $GenerateRRT(x_{start}, K, \Delta x)$

$\mathcal{T}.init(x_{start})$
for $k \in 1, \ldots, K$ **do**
  $x_{rand} \leftarrow Sample\_Free()$
  $x_{near} \leftarrow Nearest\_Node(x_{rand}, \mathcal{T})$
  $x_{new} \leftarrow Steer(x_{near}, x_{new}, \Delta x)$
  if $ObstacleFree(x_{near}, x_{new})$ **then**
    $\mathcal{T}.add\_vertex(x_{new})$
    $\mathcal{T}.add\_edge(x_{near}, x_{new})$
  end if
end for
Return $\mathcal{T}$

To adjust the algorithm to become RRG requires additional connections to take place. Rather than connecting x_{new} to just x_{near}, we connect x_{new} to all nodes on the graph that exist within a predefined neighborhood of the new node. After the graph is expanded, paths can be found through graph search algorithms such as A*. The issue with this algorithm is the time complexity, which makes it difficult to implement in systems that take advantage of online planning. The query based time complexity, as well as the space complexity, are each $O(nlogn)$. However, this method is asymptotically optimal, meaning that with infinite queries, it will find a perfectly optimal path. This is a property not held by the base RRT algorithm. However, the base RRT algorithm has time complexity $O(n)$ meaning that it runs much faster than RRG.

RRT* yields the same time complexity of RRT with RRG's benefit of being asymptotically optimal. RRT* logic is shown in Algorithm 2. In this algorithm, we additionally define E as the set of edges contained within the tree in order to simplify the description of the rewiring algorithm.

Algorithm 2 $GenerateRRT^*(x_{start}, K, \Delta x, r_{search})$

$\mathcal{T}.init(x_{start})$
for $i \in 1, \ldots, K$ **do**
  $x_{rand} \leftarrow Sample\_Free()$
  $x_{nearest} \leftarrow Nearest\_Node(x_{rand}, \mathcal{T})$
  $x_{new} \leftarrow Steer(x_{nearest}, x_{new}, \Delta x)$
  if $ObstacleFree(x_{nearest}, x_{new})$ **then**
    $\mathcal{T}.add\_vertex(x_{new})$
    $X_{neighbors} \leftarrow Near(x_{new}, r_{search})$
    $x_{cheapest} \leftarrow x_{nearest}$
    $c_{min} \leftarrow Cost(x_{nearest}) + dist(x_{nearest}, x_{new})$
    for $x_{test} \in X_{neighborhood}$ **do**
      $c_{new} = Cost(x_{test}) + dist(x_{test}, x_{new})$
      if $c_{new} < c_{min}$ **then**
        $c_{min} \leftarrow c_{new}; x_{cheapest} \leftarrow x_{test}$
      end if
    end for
    $\mathcal{T}.add\_edge(x_{cheapest}, x_{new})$
    for $x_{test} \in X_{neighborhood}$ **do** // rewire step
      $c_{new} = Cost(x_{test}) + dist(x_{test}, x_{new})$
      if $c_{new} < Cost(x_{test})$ **then**
        $\mathcal{T}.add\_edge(x_{new}, x_{test})$
        $\mathcal{T}.remove\_edge(x_{test_{parent}}, x_{test})$
      end if
    end for
  end if
end for
Return $\mathcal{T}$

IV. Experimental Description

In this section, we will lay out our methods for experimentation and comparison of RRT algorithms applied to quadrotor drones in a simulated environment. Due to the versatility of these robots, their use in both academia and industry has been widely adopted and studied in various forms.

A. Algorithm Implementation

The RRT, RRT*, and 3D RRT* algorithms were all implemented in Python, making use of popular packages

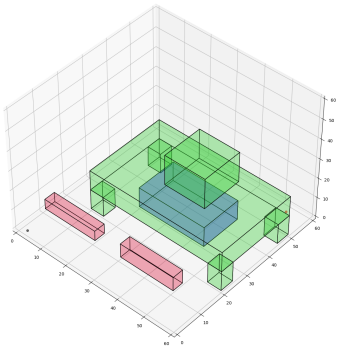
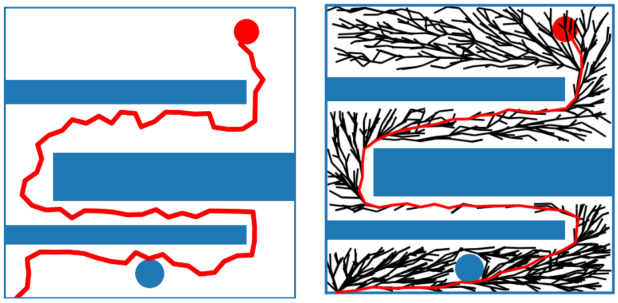


Fig. 3: 3D "table environment"



(a) (b)

Fig. 4: 2D RRT path vs RRT* path and tree

including NumPy, SciPy, Matplotlib, and NetworkX [20]. The trees are implemented as a list of (x, y) coordinates contained in a NetworkX graph object for ease of shortest path computation and visualization. Obstacles are represented as rectangular corner points, or in the case of circles as a centerpoint and radius. Use of Matplotlib enables robust geometry interrogation via object methods, as well as easy visualization because NetworkX uses Matplotlib.

To simulate the cramped and geometrically varied conditions of an urban environment, a series of rectangular and circular obstacles are placed in the space. The x_{start} and x_{goal} are selected from two opposite edges of the rectangular workspace. Any environment where there is no valid path between the start and goal is removed from the experiment set.

B. Quadrotor Integration

As stated in the previous section, quadrotor drones are increasingly popular robots in both industry and literature, making them sensible platforms for investigating effective planning algorithms. We are particularly interested in the drone's ability to follow the paths resulting from expansion, based on the total distance traveled by the drones and how it compares to the total distance of the path outputted by the algorithm, as well as the total time the drones take to reach the goal region as compared across the different algorithm.

In order to simulate the use of these robots, we use the UAV Toolbox which is found as an extension of MATLAB's Simulink software. This system allows us to provide a model which applies controls and follows trajectories with the dynamics of real quadrotors. This simulation verification allows us to test robotic trajectories without accepting the challenges of real hardware while still yielding the benefits of providing a near-lifelike interpretation of the effectiveness of these algorithms.

This system does not account for the intricacies of real-world drone-obstacle collisions. As the path plans are generated by separate offline algorithms, the simulated drone simply follows the provided path. If the drone is unable to keep following paths and enters a restricted region in the physical arena, we simply flag this as a crash in order to preserve the safety of the robot. As we aim to simulate a relatively cluttered environment, we anticipate some inability to maintain flight outside of obstacles for paths found with many redirections which informs our decision to not use physical boundaries.

C. Expansion of RRT to 3D

As previously described, RRT algorithms applied to 3D exhibit slow optimality convergence and large space requirements. This paper proposes modifying the existing 2D RRT code to sample 3D coordinates and introducing the least-cost connection and rewiring steps of RRT*. If necessary, a B-spline smoothing step can be applied for a smoother path better suited for the drone to follow.

D. Algorithm Evaluation

Statistically relevant metrics are needed to characterize the performance of these path planning algorithms. In this work, the principal metrics explored are: runtime performance, difference in path length between the quadcopter path and the RRT* path, RRT* convergence to solution (show cost minimization over many iterations). As RRT* is desirable for its convergence to an optimal solution while RRT provides the first valid solution found, we evaluate our RRT* implementation to show its cost minimization over iterations compared to RRT. Then, evaluating these same metrics after extending the RRT* implementation into 3 dimensions.

To benchmark these results and demonstrate their performance, they are compared to RRT and RRT* path planning algorithms from the MATLAB UAV Toolbox package [21]. Use of this robust, ubiquitous software enables quick comparison of performance, and grounds the work described in this paper within the domain of robot path planning research. As appropriately exhaustive tests for the implemented algorithms, a series of test environments ($n = 10$) are designed with varying obstacle composition, number of obstacles, and path length. Each algorithm is run 5 times on each environment and the results are evaluated as individual performances for self-comparison, as well as averaged for comparison to the other algorithms.

V. Results

The tables below present statistics of the algorithm implementations for a series of experiments in comparison to time taken to follow the path in the MATLAB simulated environment.

2D RRT paths			
	Path Length	**Time to find (s)**	**Time in sim (s)**
Path 1	219	12	17.39
Path 2	210	12	17.16
Path 3	231	10	19
Path 4	216	11	18.09
Path 5	207	11	18.11
Mean	216.6	11.2	17.95
Std dev	9.3	0.84	0.72

TABLE I: 2D RRT experiments

2D RRT* paths			
	Path Length	**Time to find (s)**	**Time in sim (s)**
Path 1	174	170	15.34
Path 2	175	138	14.72
Path 3	176	112	14.74
Path 4	171	185	14.7
Path 5	177	152	16.36
Mean	174.6	151.4	15.18
Std dev	2.3	28	0.71

TABLE II: 2D RRT* experiments

3D RRT paths			
	Path Length	**Time to find (s)**	**Time in sim (s)**
Path 1	108	1.57	8.15
Path 2	165	3.4	11.83
Path 3	135	0.49	10
Path 4	153	0.41	10.88
Path 5	141	2.36	10.73
Mean	140.4	1.646	10.32
Std dev	21	1.3	1.4

TABLE III: 3D RRT experiments

3D RRT* paths			
	Path Length	**Time to find (s)**	**Time in sim (s)**
Path 1	117	106	8.57
Path 2	130	41	9.65
Path 3	116	118	9.28
Path 4	127	43	9.1
Path 5	120	44	8.63
Mean	122	70.4	9.05
Std dev	6.2	38	0.45

TABLE IV: 3D RRT* experiments

Table I,II,III,IV show the path lengths, time elapsed to perform the particular algorithm, and execution time of the trajectory in the quadcoper simulation environment for 5 runs of the algorithm. Animated simulation of UAV following waypoints in the environment can be produced by following execution steps in `README.md`. Figure 5 shows plotted trajectories in the 2D and 3D environments from the four algorithms (paths in red, obstacles in blue, obstacle centers in dark blue diamonds).

VI. Discussion

Hardware implementation of control algorithms brings with it intrinsic complications that warrant creative and patient approaches. In the event hardware refuses to cooperate, computation and simulation are effective mediums to demonstrate work as simulation techniques may offer approximations of systems that very closely mimic the control and action of real-world systems in a controlled environment. In addition,

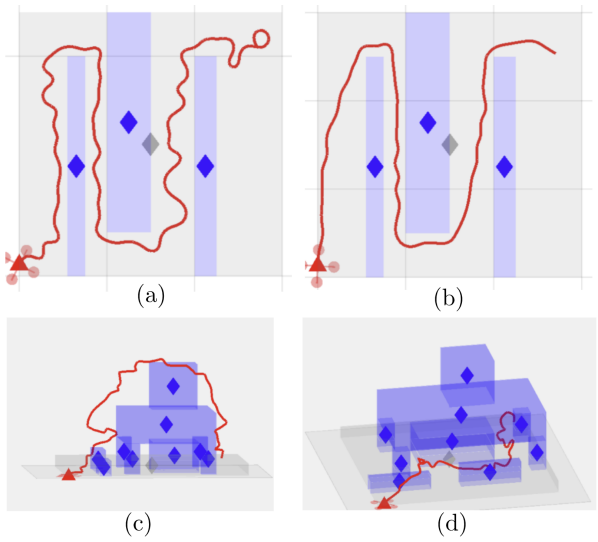


Fig. 5: Simulation results with (a) 2D RRT, (b) 2D RRT*, (c) 3D RRT, and (d) 3D RRT*

there are benefits to working in these environments such as safety and cost of materials, especially if the hardware is defective or experiences mechanical failure over the course of experimentation.

Our RRT* implementation path length is markedly lower than RRT by an average of 20%, which demonstrates the asymptotic optimality exhibited in RRT* but not in base RRTs. However, due to the increased computational cost of RRT*, the time taken to evaluate on the same number of iterations increased from RRT to RRT* by 93%. Due to the decrease in path lengths, as well as trajectories that are far easier to follow given the UAV's dynamics, the time taken by the drone in simulation to complete the trajectories generated in RRT* was far faster than ones found by RRT, a result the holds true for the 2D case and the 3D case alike.

In comparing the two and three-dimensional cases, we find that the paths which are found in two dimensions yield far more consistent results, with a marked increase in standard deviation when the extra dimension is added (Table II and Table IV). We associate this with the intrinsic difficulty of performing completely random sampling in three dimensions, as success will vary quite widely. This is also the case for the standard deviation of the 3D RRT* path times in our implementation, attributed to the greater amount of computation required for this algorithm. In Figure 5, variations in the appearance of the paths produced by the two algorithms can be observed. The RRT paths are usually much farther from optimal and can yield jagged edges which represent challenges for robots with dynamic and kinematic constraints such as UAVs, and may even generate loops within the path before reaching the target. The RRT* paths are smoother due to node rewiring and better suited to a real drone path with fewer abrupt inputs and changes in direction. As can be seen in the 3D "table environment" in Figures 5c and 5d, the RRT path takes an indirect path, farther above the obstacles when compared to the RRT* algorithm, which finds a far smoother

path through the cluttered environment obstacles.

A. Future Work

The primary direction for further development of this work is the hardware implementation of the presented algorithms. Simulations provide a great deal of information and can closely mimic the behavior of hardware. However, hardware comes with additional challenges such as potential inaccuracies in control and other factors which may impact the system's ability to follow trajectories. This being the case, it is likely we may see an even greater difference in the ability of the UAV's ability to perform each algorithm than shown in the simulation. After further work to make a controlled environment with working hardware, the use of real physical obstacles and sensor feedback to the robot would enable autonomous navigation in the real world using these path-planning algorithms.

When addressing real-world systems, additional considerations must be taken into account. These planning algorithms were run offline with the path being computed before the drone began its path. Therefore, it is not applicable to unknown or dynamic environments. As urban environments can vary greatly by the minute due to factors such as trucks stopped to unload, construction vehicles, and cranes, a strategy would be necessary to perform re-planning while the drone is in flight becomes a point of great interest in extending the work laid out in this paper. This extension may take place through the use of algorithms such as RRT^x which is designed with fast re-planning in mind. In addition, it is of great interest in real-world systems to verify the drone's ability to use a trajectory through the perception of the environment.

VII. Conclusion

In this paper, we set out to show an implementation of RRT and RRT* algorithms in environments with some clutter and show an extension into three dimensions. We performed this implementation using simulation in order to apply these algorithms in a controlled environment. We applied these algorithms multiple times in order to understand the behaviors each algorithm exhibits. By monitoring this data and our simulations we were able to better understand these tools and how they apply to UAVs in cluttered environments.

The RRT* algorithm performed remarkably well at generating near-optimal paths through the simulated environment. The smoother path generated by RRT* would additionally be much more suitable for real drone navigation in cluttered environments because of the directness of the paths and less abrupt changes in direction which could make overshoot and collision more likely, as well as increased noise from rapid changes in direction. However, it would be beneficial to increase the speed of this algorithm to be closer to the very quick RRT algorithm. This could be achieved through a strategy such as biased sampling. While traditional RRTs generate a random coordinate anywhere in the domain to determine the direction of travel, the path function could be replaced with one which samples a Gaussian distribution density curve with the mean oriented towards the goal. The standard deviation of the distribution can be adjusted to control the aggressiveness of the algorithm.

References

[1] Y. Akagi and P. Raksincharoensak, "Stochastic driver speed control behavior modeling in urban intersections using risk potential-based motion planning framework," *2015 IEEE Intelligent Vehicles Symposium (IV)*, pp. 368–373, 2015.

[2] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *Journal of Artificial Intelligence Research*, vol. 39, pp. 533–579, Sept. 2010.

[3] L. Kavraki, P. Svestka, J.-C. Latombe, and M. 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.

[4] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[5] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, 1999, pp. 473–479 vol.1.

[6] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," 2011. [Online]. Available: https://arxiv.org/abs/1105.1186

[7] I. Noreen, A. Khan, H. Ryu, N. L. Doh, and Z. Habib, "Optimal path planning in cluttered environment using rrt*-ab," *Intelligent Service Robotics*, vol. 11, pp. 41–52, 2018.

[8] M. Kalisiak and M. van de Panne, "Rrt-blossom: Rrt with a local flood-fill behavior," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006, pp. 1237–1242.

[9] M. W. Otte and E. Frazzoli, "Rrtx: Real-time motion planning/replanning for environments with unpredictable obstacles," in *Workshop on the Algorithmic Foundations of Robotics*, 2014.

[10] 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 *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004.

[11] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[12] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 7681–7687.

[13] X. Wu, L. Xu, R. Zhen, and X. Wu, "Biased sampling potentially guided intelligent bidirectional rrt algorithm for uav path planning in 3d environment," *Mathematical Problems in Engineering*, 2019.

[14] Q. Fu, X. Lan, Y. Ji, X. Sun, and F. Ren, "Heuristic rrt fusion a for 3d path planning of uav," in *2022 IEEE 6th Information Technology and Mechatronics Engineering Conference (ITOEC)*, vol. 6, 2022, pp. 1433–1443.

[15] Y. Lin and S. Saripalli, "Sampling-based path planning for uav collision avoidance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 11, pp. 3179–3192, 2017.

[16] Rodriguez, X. Tang, J.-M. Lien, and N. Amato, "An obstacle-based rapidly-exploring random tree," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 2006, pp. 895–900.

[17] J.-K. Park and T.-M. Chung, "Boundary-rrt* algorithm for drone collision avoidance and interleaved path re-planning," *J. Inf. Process. Syst.*, vol. 16, pp. 1324–1342, 2020.

[18] F. Kiani, A. Seyyedabbasi, R. Aliyev, M. U. Gulle, H. Basyildiz, and M. A. Shah, "Adapted-rrt: Novel hybrid method to solve three-dimensional path planning problem using sampling and metaheuristic-based algorithms," *Neural Comput. Appl.*, vol. 33, no. 22, p. 15569–15599, nov 2021. [Online]. Available: https://doi.org/10.1007/s00521-021-06179-0

[19] L. Killian and J. Backhaus, "Utilizing the RRT*-Algorithm for Collision Avoidance in UAV Photogrammetry Missions," *arXiv e-prints*, p. arXiv:2108.03863, Aug. 2021.

[20] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," 1 2008. [Online]. Available: https://www.osti.gov/biblio/960616

[21] I. The MathWorks, "Matlab uav toolbox r2022b, v. 1.0,," https://www.mathworks.com/products/uav.html?s_tid=AO_PR_info, [Accessed 14-Dec-2022].