Combining current existing RRT path-generating methods to optimize simulated UAV flight

Jesse Yan

Project purpose

Rapidly-exploring Random Tree (RRT) path-planning algorithms are widely used in the field of drones and other unmanned aerial vehicles (UAVs). RRT is a sampling-based algorithm that incrementally builds a tree through a state space by randomly selecting points and extending the tree toward them. As noted in prior literature, “RRT is an algorithm based on decision trees and random sampling searches around a determined state space … Since RRT is a sample algorithm, there is no guarantee of finding the optimal solution to the problem” (MDPI, 2023). This inherent randomness often results in inefficient exploration and non-optimal paths, particularly in large or complex environments.

The purpose of this project was to improve upon existing RRT path-planning algorithms by combining multiple additional functionalities together in order to generate more optimized and feasible paths in 3D space within the same amount of time or less.

Specific implementation and key features

Starting with a basic Python 3 implementation of an RRT algorithm that generates paths for 3D space, the following functions were added:

1. Path “smoothing” or “shortening”: A postprocess by which the algorithm will remove any waypoints it deems unnecessary, e.g. a path segment with multiple points in a straight line will become a path with only a startpoint and an endpoint. This increases path efficiency and decreases the time needed for a UAV to travel the same distance.

2. Limiting the maximum number of nodes - pruning and goal bias: Once the number of nodes in the tree the algorithm generates exceeds a user-specified threshold, the algorithm will identify which current node is “best” by finding the node with the lowest Euclidean distance from the goalpoint and removing all other nodes. This allows path generation to maintain a constantly high level of speed, which is especially useful in large or complicated maps; while also biasing waypoints that are closer to the intended goal.

3. Dead end detection and backtracking: If the algorithm has not made any progress for a certain user-specified number of iterations, e.g. it has been stuck on the same node for too long, then it will find an earlier node and start over from that point. This function has not yet been tested in detail.

4. Bidirectional path generation: The algorithm will generate trees both from the startpoint and the goalpoint, and identify the point where the two paths meet. This greatly

increases the overall speed of the algorithm and is extremely useful especially for large and complex maps.

Sample results and test cases

Link to full code and variables: https://github.com/Pr0gram-Creat0r-1/Python-RRT

This code was simulated in Gazebo 11 (Open Source Robotics Foundation [OSRF], 2021) with a PX4 model of an Iris Quadcopter (PX4 Development Team, 2024), using ROS Noetic (Open Source Robotics Foundation [OSRF], 2020) to allow the quadcopter to carry out the directions given by the path-generating algorithm.

Table of measures of average path generation time (seconds) in various maps:

This algorithm was tested on 5 different sample maps of varying characteristics, with 3 trials for each. The variables were kept constant and the only change was the map being used for path generation.

Map 1: 50x50x50 cubic meters with many obstacles. (3 required waypoints).

Map 2: Small area with a single opening in a wall that the UAV must pass through. (1 required waypoint).

Map 3: Multiple floating slabs with space in between that the UAV must pass through to reach the top. (1 required waypoint).

Map 4: Densely placed blocks in a 50x50 square meter area that UAV must navigate through. (1 required waypoint).

Map 5: Staggered blocks forming a simple maze (see Images and Figures, Fig. 1). (1 required waypoint).

Map Number	**Trial 1 time**	**Trial 2 time**	**Trial 3 time**	**Average time**
1	0.061	0.067	0.069	0.066
2	0.005	0.008	0.007	0.007

3	0.116	0.466	0.263	0.282
4	0.096	0.108	0.088	0.097
5	0.414	0.439	0.48	0.444

Notes:

- On map 5, it was found that 2 additional required waypoints besides the final goalpoint had to be added in order for a path to generate at all.
- The data shows that there does not seem to be any correlation between map "complexity" and increases in path generation time; nor is map size as major of a factor as was previously hypothesized. Rather, the biggest delays and challenges in generation seem to occur when many long detours are necessitated, such as is the case in maps 3 and 5.
- The largest improvements in speed appear to be associated with the use of bidirectional path generation. This observation aligns with established research, which states that "the bidirectional (two-tree) variants of the RRT algorithm … have been empirically observed to show very fast convergence" (Karaman & Frazzoli, 2013). Thus, bidirectional expansion appears to be the dominant factor contributing to the consistently low path generation times observed across these test cases.

Images and figures:

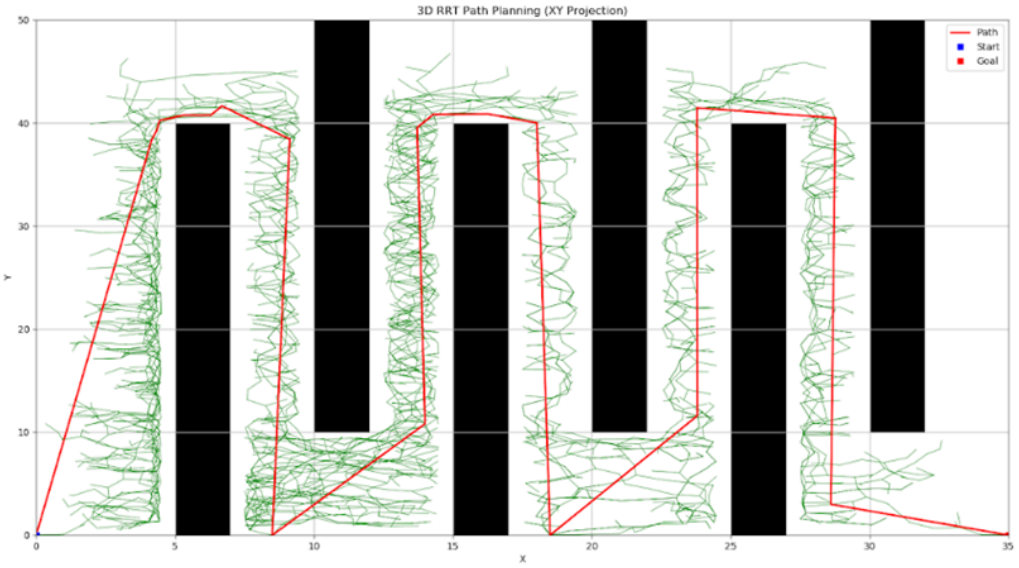

Fig. 1: Matplotlib visualization of RRT path generation on map 5.

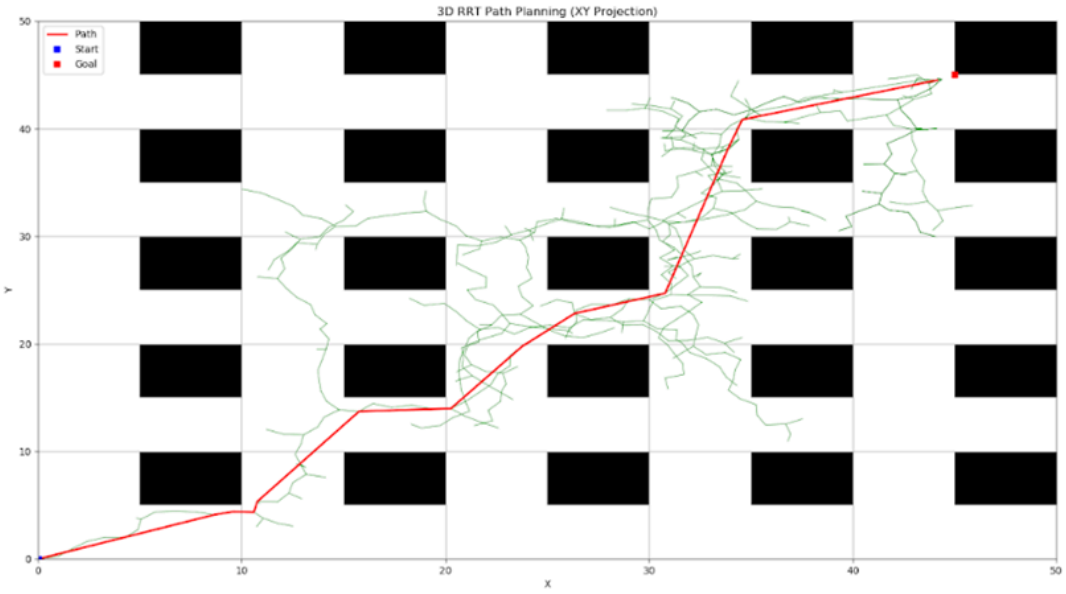

Fig. 2: Matplotlib visualization of RRT path generation on map 4. This path, which does not contain major detours, was significantly easier and faster to generate, as can be seen by the relative lack of exploration, represented by the green branches.

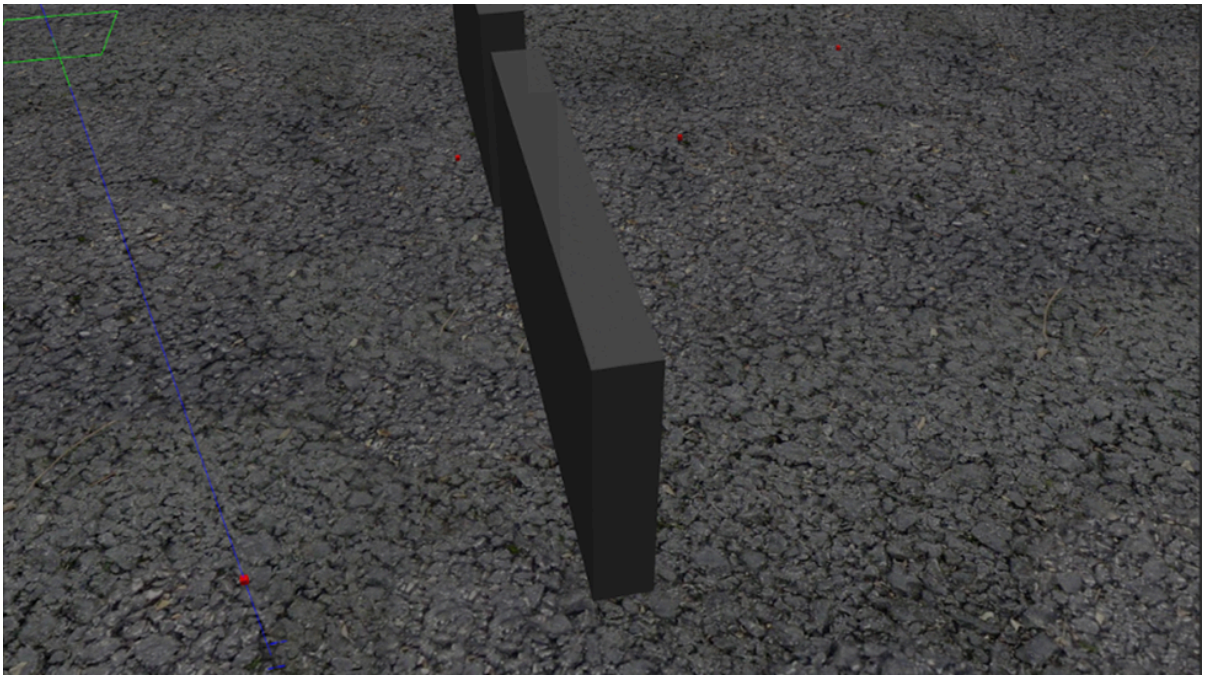

Fig. 2: 3D Gazebo rendering of RRT path in Map 2. UAV is near the bottom left and waypoints are represented by red dots.

Conclusion

This project was able to make some advancements and improvements to the basic RRT algorithm and combine them together into one program, turning it into a more optimized option that can maintain a fast computing speed in large and complex worlds and generate paths that are considerably more efficient. Future endeavors could include using some of these concepts in the asymptotically optimal RRT* path generating algorithm to offset its greatest drawback, lack of speed. More testing and development of the implemented functionalities would also be advisable, with the use of considerably larger and more complex maps, different types of UAVs, more variables being tested besides path generation time, etc. Perhaps map and obstacle generation can be handled automatically by mapping programs instead of creating worlds manually. Overall, extending the current functionalities to a wider variety of scenarios, and combining them with a more inherently optimized algorithm would yield major advancements and uses.

References

Karaman, S., & Frazzoli, E. (2013). Sampling-based algorithms for optimal motion planning. Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory.

https://dspace.mit.edu/handle/1721.1/79884

MDPI. (2023). Rapidly exploring random tree algorithm definitions and applications. Agriculture, 13(2), 354.
https://www.mdpi.com/2077-0472/13/2/354

Open Source Robotics Foundation. (2020). ROS Noetic Ninjemys.
https://wiki.ros.org/noetic

Open Source Robotics Foundation. (2021). Gazebo 11 release.
https://classic.gazebosim.org/blog/gazebo11

PX4 Development Team. (2024). PX4 autopilot [Software].
https://github.com/PX4/PX4-Autopilot