Aerial Vehicle Navigation Planning

1. PROBLEM

Traditionally, path planning algorithms solve problems in two-dimensional space. Some applications, such as autonomous aerial vehicle planning, require guidance in three-dimensions. Aerial vehicle navigation is a complex problem because it may involve standard movements, such as forward and backward, as well as navigation over and under obstacles in the environment. Additionally, these vehicles may also operate under fuel and timing constraints which makes the ability to quickly and efficiently find short paths essential. In complex worlds with a significant amount of obstructions, path planning algorithms can solve this problem by locating an optimal path between two points.

2. RELATED WORK

Rapidly-exploring Random Trees (RRTs) is an example of a Motion Planning algorithm that is based of randomization. Other examples of randomized motion planning algorithms include, but are not limited to, randomized potential fields and expansive-space planners. Randomized potential fields is a sampling based planner that uses random-walks to escape local minima, using a potential function that tries to estimate the cost from any state configuration to the goal [LaValle 2006]. The main drawback, however, was that the method involved many heuristic parameters that had to be adjusted for each problem [LaValle 2006]. Expansive-space planners attempt to explore new space by probabilistically choosing new verticies from different predetermined neighborhoods [LaValle 2006]. The main drawbacks are that the planner requires substantial problem-specific parameter tuning [LaValle 2006]. A* : is an extension of Dijkstras algorithm that tries to reduce the total number of states explored by incorporating a heuristic estimate of the cost to get to the goal from a given state [Russell and Norvig 2010]. A* is not originally for motion planning or random, but can be adapted as such for interesting comparisons.

3. APPROACH

Rapidly exploring random trees and A* search were evaluated on randomly generated environments. A three dimensional grid of nodes and connections were created to form an undirected graph as shown in Figure 1. The connections in the graph were also given a uniform cost. The grid consisted of nodes that are spaced one hundred units apart arranged in a cube with an equal length, width, and height. The algorithms were tested on environments where the dimensions of the cube varied from 600 units to 1200 units.

While planning, each algorithm must also navigate around obstructions that were created and placed in the environment. Nodes cannot exist within obstructions and connections cannot pass through an obstruction. Each obstructions coordinates as well as height and length were varied based on the size of the graph. Every

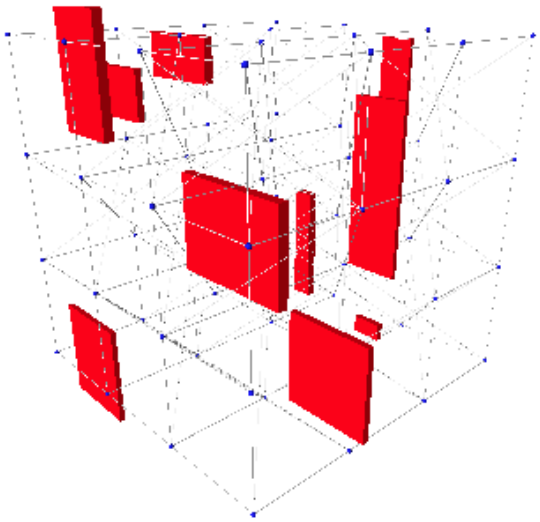

Fig. 1. A randomly generated 3D environment.

environment was tested twice with a range of 10 to 50 obstructions. In each test, the start node is set to the origin of the graph and the goal node was chosen randomly from the graph. An example run within our 3D navigation simulation can be seen in Figure 3.

An existing A* search implementation was modified to operate in three dimensions and evaluated on the random graphs. An admissible heuristic was created for A* by calculating the three-dimensional euclidean distance from the current node to the goal node.

Rapidly-exploring Random Trees (RRTs) are introduced as a planning algorithm to quickly search high-dimensional spaces that have both algebraic constraints (obstacles) and differential constraints (nonholonomics and dynamics) [LaValle 1998]. The key idea is to bias exploration toward unexplored portions of the space by sampling from all points in the state space, and incrementally pulling the search tree toward them [LaValle and Kuffner 2001].

The basic RRT algorithm is given in Figure 2. Each iteration of construction attempts to extend the RRT by adding a new vertex that is biased by a randomly-selected state x_{rand}. The $EXTEND$ function selects the nearest vertex already in the RRT x_{near}, and applies a valid motion towards the randomly selected state input, creating a new node to be added to the RRT x_{new} [LaValle and

```
BUILD_RRT(x_init)
 1  T.init(x_init);
 2  for k = 1 to K do
 3       x_rand ← RANDOM_STATE();
 4       EXTEND(T, x_rand);
 5  Return T
```

```
EXTEND(T, x)
 1  x_near ← NEAREST_NEIGHBOR(x, T);
 2  if NEW_STATE(x, x_near, x_new, u_new) then
 3       T.add_vertex(x_new);
 4       T.add_edge(x_near, x_new, u_new);
 5       if x_new = x then
 6            Return Reached;
 7       else
 8            Return Advanced;
 9  Return Trapped;
```

Fig. 2. RRT algorithm.

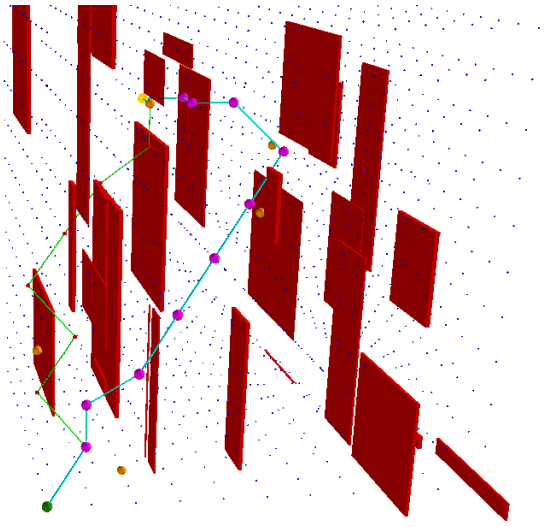

Fig. 3. Example run of the two algorithms in a randomly generated world, starting from the bottom corner navigating to mid-top. - Green: A* — Cyan: RRT

Kuffner 2001]. This biases the RRT to rapidly explore the state space.

4. EVALUATION

With this path planning simulation (Figure 3), a number of variables and results were recorded, including: the number of nodes in the graph, the number of obstructions, the running time of both algorithms, the final path length of both algorithms, and the number of nodes that were expanded by both algorithms. Our evaluation focuses on the number of expanded nodes, the final path length, and the running time as a function of the number of nodes in the graph.

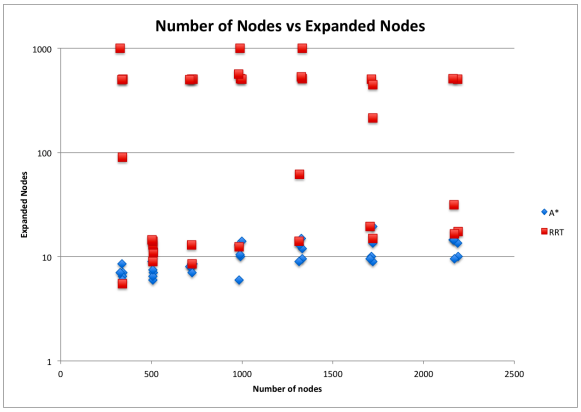

Fig. 4. Number of nodes expanded for each graph size.

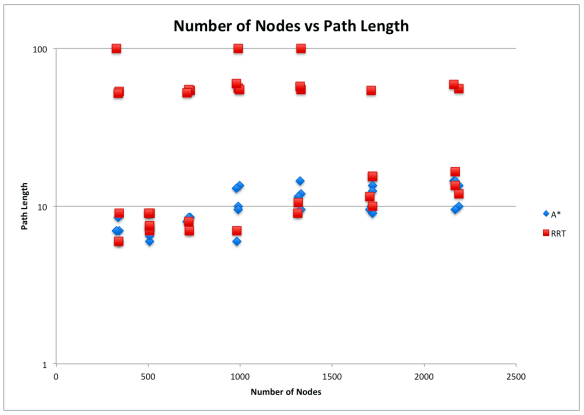

Fig. 5. Number of nodes in final path for each graph size.

4.1 Nodes Expanded

The number of nodes that each algorithm expanded as a function of graph size is shown in Figure 4. When using the term expanding, we mean that the search algorithm moved to a certain node and can begin evaluating if it should move to the neighbors of the new node. Even when the graph size reaches more than 2000 nodes, A* search expanded at most 28 nodes. From our testing, A* typically expanded only the nodes that are on the optimal path with occasional expansions of a small number of additional nodes. In some cases, RRTs expanded significantly more nodes than A*. However, in about 50% of the simulations, RRTs expanded a similar amount of nodes as A*.

4.2 Path Length

The length of the final path was also recorded, as seen in Figure 5. In most of the simulations, RRTs had a final path length that is equal to or a few units longer than the final path that A* found. RRTs never found a path that is shorter than A* search. In some cases, the final path length for RRTs was much longer than that of A*.

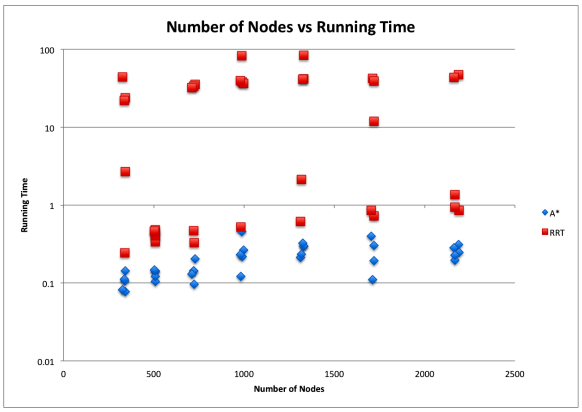

Fig. 6. Time to find the final path for each graph size.

4.3 Runtime

Finally, the running time of each simulation as a function of the number of nodes in the graph was also recorded. The results from this testing can be seen in Figure 6. In every graph, A* search had a running time that was less than the running time of the RRT searching algorithm. A* typically required a third of a second or less to compute the optimal path. In some cases, the RRT algorithm spent a similar amount of time as A* search. In most cases, the computation time for RRTs was two or three times longer than that of A*. In some simulations, RRTs required around 90 seconds to perform path planning.

5. DISCUSSION

An analysis of our results is described below. Planned additions to this project are explained in the future work section.

5.1 Analysis

The random nature of RRTs generated interesting results with respect to A* search. In some cases, RRTs found a similar path as A*, but in about half of our simulations, RRTs found a path that is significantly longer. RRTs randomly choose a point that is some distance away from currently visited nodes and the algorithm then moves towards the point. This random movement continues until the goal node is reached. While A* is continually calculating the best path, RRTs are operating in a random fashion, which can produces sub-optimal results. In fact, it has been shown that as the number of samples increases, the RRT algorithm converges to an optimal solution with probability zero [Karaman and Frazzoli 2010].

For our testing, limits were placed on the RRT algorithm to ensure that the running time is reasonable for a large number of simulations. The RRTs were limited to expanding 1000 nodes. This constraint may have influenced some of the failed attempts at finding the goal, however, the paths in these solutions would have been slightly more inefficient than the results currently show.

Results also show (Figure 6) that the runtime of RRT is on average slightly worse than A*, but still comparable. A longer runtime is expected due to the extra calculations and randomness in RRT. Sometimes, the runtime was over 100x slower than A*, probably due to a worst case ordering of sampling the state space. The majority of the time, both algorithms finished in well under 1 second.

Based on our testing, the A* algorithm would be the obvious choice for path planning of an autonomous aerial vehicle if the goal is to get from one point to another in the shortest amount of time. However, if the vehicle is used to explore an environment, RRTs could provide a useful planning algorithm. The RRT algorithm could easily be modified so that a significant area of the graph is visited which makes this algorithm ideal for exploration situations.

A visualization of each algorithm and its path was initially a fun novelty, as it was neat to see both A* and RRT choose different paths to the same goal. The 3D visualization, however, later proved to be an invaluable debugging tool. The obstacle avoidance check for RRT was initially not working correctly, allowing the tree to travel straight through walls, but viewing the tree in 3D helped catch that and the problem was then resolved. The visualization also helped us understand how each algorithm works. When testing on large graphs, it is very easy to see that the path generated by A* is much shorter than the path generated by RRTs.

5.2 Future Work

In the experiments in this paper, only holonomic aerial vehicle planning was modeled. This was mainly for making the comparison to A* easier, as we found its difficult to model non-holonomic constraints with A*, but trivial with RRTs. In future work, more advanced vehicle modeling can be simulated, including vehicles with non-holonomic and kinodynamic constraints that can be compared to other motion planning algorithms.

We would also like to expand our work by incorporating the RRT* algorithm. We would expect the RRT* algorithm to perform similarly to A* search, but would allow for the modeling of non-holonomic constraints [Karaman and Frazzoli 2010].

REFERENCES

KARAMAN, S. AND FRAZZOLI, E. 2010. Incremental sampling-based algorithms for optimal motion planning. *CoRR abs/1005.0416*.

LAVALLE, S. 1998. Rapidly-exploring random trees - a new tool for path planning. *Computer Science Dept., Iowa State Univ.*.

LAVALLE, S. 2006. *Planning Algorithms*. Cambridge University Press.

LAVALLE, S. AND KUFFNER, J. 2001. Rapidly-exploring random trees: Progress and prospects.

RUSSELL, S. J. AND NORVIG, P. 2010. *Artificial Intelligence: A Modern Approach, Third Edition*. Printince Hall.