Field-aided RRT*

Ishan Chadha
College of Computing
Georgia Institute of Technology
ichadha3@gatech.edu

Mudit Gupta
College of Computing
Georgia Institute of Technology
mgupta303@gatech.edu

I. Introduction

RRT* works well for static, fully observable environments, but when the environment is partially observable or changing, then replanning must be achieved efficiently and accurately. Our goal is to investigate improvements to RRT for partially-observable environments by incorporating ideas from other path-planning domains.

II. Background and Related Works

A. RRT and Variants

A rapidly-exploring random tree (RRT) is an algorithm that randomly samples a certain number of nodes in a nonconvex search space and then builds a tree from the start node to the goal node along these samples. In order to connect the tree at each time step, the tree is extended towards either the nearest node that does not cause a cycle in the tree or a point that lies on the path between the current node and the nearest node that is limited by some growth factor.

RRT* expands on the concept of RRT in two ways: it creates a ball of volume $V = \gamma log(n)/n$ where γ is a constant and n is the number of samples taken, and it performs a smoothing procedure to connect nodes that are not adjacent but can be connected without the tree intersecting an obstacle. These improvements allow RRT* to achieve asymptotic optimality while simultaneously creating a path that can be more easily traversed due to the removal of zigzagging edges between proximal nodes in the tree.

B. A* Graph Search and Variants

A* is an informed search algorithm that incorporates the cost of traversing a node n, g(n), with the cost left to get to the goal, h(n). The cost to get to the goal, h(n), is approximated via an admissible heuristic. Seen nodes are traversed based on a priority queue ordered by $f(n) = g(n) + h(n)$, and a path from the start to the goal is constructed.

First described by Koenig and Likhachev in the paper Incremental A* [1], Lifelong Planning A* (LPA*) improves the performance of A* in a dynamically changing environment by taking into account neighboring nodes when processing the current node being traversed. More specifically, every node n has a predecessor n' from which it is extended, and the node is considered locally consistent if g(n) equals rhs(n). The value rhs(n) is defined as g(n')+d(n',n), where d(n',n) yields the cost of getting from n' to n. Nodes are added to a priority queue for reevaluation when they are locally inconsistent and keyed by two values: first, $min\{g(n), rhs(n)\} + h(n)$, and second, $min\{g(n), rhs(n)\}$. If rhs(n) is less than g(n) (locally overconsistent, the parent n' is now more cheaply reachable), g(n) is set to rhs(n), and if rhs(n) is greater than g(n) (locally underconsistent, the node n is more costly to be reached from n' than previously determined), g(n) is set to infinity. After this, if the node is locally consistent, we pop it from the queue; otherwise, we update its key and add it back to the queue. Since updating the g-value of a node may also affect the rhs-value of the node's successors, all of the node's successors are also reevaluated, leading to a cascading effect as new obstacles are discovered. LPA* essentially allows only a few of the nodes to be expanded again every time an obstacle is encountered rather than all the nodes of the A* graph, which is more efficient.

D* Lite, also described by Koenig and Likhachev [2], is an extension of LPA* which adds a couple of significant optimizations, most notably keying the priority queue for reevaluating nodes differently. Rather than reordering the entire queue, the difference in h(n, goal) between before and after the obstacle was detected is added to every element since the change in the heuristic of every element in the reevaluation queue will be lower bounded by this value. Another optimization is that D* Lite is run from the goal node to the start node since obstacle detection happens closer to the robot rather than the goal node, so we preserve the parts of the tree that are closer to the goal node.

C. RRT^x for Partially Observable Environments

RRT^x, introduced by Otte and Frazzoli [3], uses a priority queue to update weights in a similar fashion to D* Lite, except applied to RRT* rather than A*. In contrast with other RRT* variants, to tackle partially observable environments that may dynamically change, RRT^x has been shown to be asymptotically optimal as well as performing both accurately and quickly.

D. Artificial Potential Fields

Li et al proposed PQ-RRT* in 2020 [4], which builds on RRT* by slightly expanding the search space for finding the nearest node, optimizing the rewiring procedure, and, most importantly, utilizing an artificial potential field to push selected nodes towards the goal state during tree extension. Also, a repulsive force emanates from obstacles, which further

improves the path that the tree follows. Random sampling helps the tree avoid local minima in the field.

III. Proposed Method

We propose two main changes to the existing state-of-the-art solutions for RRT* in partially observable environments. RRT^x puts all severed nodes into a priority queue based on their cost and rewires each one of them back into the tree. Our solution will instead handle obstacle discovery by first determining some set of tree connections that must be severed, performing a potential field update on existing nodes, and completing randomized rewiring of the severed portion of the tree.

A. Tree-connection Severing

Whenever an obstacle is found, all nodes that fall within the bounds of the obstacle can be deleted from the tree by storing node locations in a spatial hashmap. Every child (direct and indirect) of any node in this set should be severed from its parent and added to a free set.

B. Potential Field Update

Whenever obstacles are detected, the potential field is updated based on the new obstacles. Once the field is updated, we will apply a small force to every node in the tree. Nodes that are pushed onto each other may be merged for optimization purposes.

C. Randomized Rewiring Step

Finally, the free set of nodes should be randomly sampled and connected to the severed portion of the tree via the normal RRT* wiring process. Rewiring will terminate early once the robot's current position is added to the tree. Then, tree traversal will resume from the frontier node.

IV. Evaluation

The performance of our approach algorithm will be compared to the RRTx implementation based on the following metrics:

- Experimental runtime of RRTx and Field-aided RRT runtime of RRTx and Field-aided RRT
- Path optimality based on length of each leg of the traversed path
- Path smoothness (not formally defined yet)

We also plan to include a formal proof of the asymptotic optimality of (or lack thereof) the new algorithm.

V. Hypotheses

The field updates should help the robot more easily find paths that go around obstacles rather than repeatedly running back into the same obstacle and getting stuck. The lack of a reevaluation queue should make replanning time faster compared to RRTx while maintaining ample distance from obstacles due to the potential field update.

References

[1] S. Koenig and M. Likhachev, "Incremental a*," in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, (Cambridge, MA, USA), p. 1539–1546, MIT Press, 2001.

[2] S. Koenig and M. Likhachev, "D*lite," in *Eighteenth National Conference on Artificial Intelligence*, (USA), p. 476–483, American Association for Artificial Intelligence, 2002.

[3] M. Otte and E. Frazzoli, *RRT^X: Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles*, pp. 461–478. 04 2015.

[4] Y. Li, W. Wei, Y. Gao, D. Wang, and Z. Fan, "Pq-rrt*: An improved path planning algorithm for mobile robots," *Expert Systems with Applications*, vol. 152, p. 113425, 2020.