Comparative Analysis of Path Planning Algorithms through RRT and RRT*

Submitted by:

Aratrik Paul	16BCE0519
Shikhar Singh	16BCE2316

Under the guidance of:

Prof. Narayanamoorthy M

Robotics and It’s Applications, CSE 3011

School of Computer Science and Engineering

VIT, Vellore

Acknowledgement

This project would not have been possible without the guidance and support of our faculty, Prof. Narayanamoorthy M. We would also like to thank the authors of the paper for their innovative work that enabled us to think beyond the classroom's scope.

We would also like to thank the University Management for giving us an opportunity to carry out our academic activities in the University.

Aratrik Paul
Shikhar Singh

Abstract

Path-planning is an important primitive for autonomous mobile robots that lets robots find the shortest or otherwise optimal path between two points. Otherwise optimal paths could be paths that minimize the amount of turning, the amount of braking or whatever a specific application requires. Algorithms to find a shortest path are important not only in robotics, but also in network routing, video games and gene sequencing. This report deals with the comparative study of two such algorithms that are used in path planning namely RRT and RRT* through MATLAB simulations.

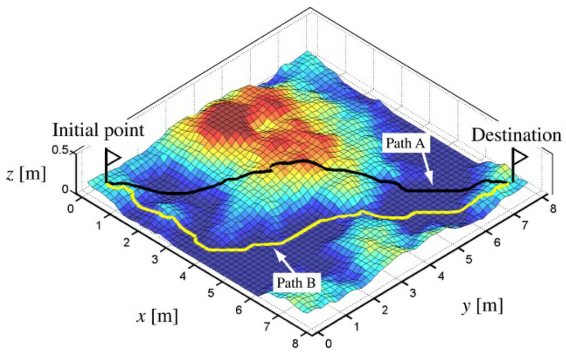
Introduction

Path Planning for mobile robots has many useful applications in autonomous cars Unmanned Aerial Vehicles (UAVs), industrial forklifts, surveillance operations and planetary space missions. Path planning algorithms aim to find a collision free path from an initial state to a goal state with optimal or near optimal path cost. Sampling Based Planning (SBP) algorithms have been extensively used for path planning of mobile robots in recent years. SBP algorithms are known to provide quick solutions for

complex and high dimensional problems using randomized sampling in search space. However, SBPs are probabilistic complete, i.e., a solution will be provided, if one exists, given infinite runtime. Lavelle proposed Rapidly exploring Random Tree (RRT), which is a well-known SBP algorithm. RRT supports dynamic environment and non-holonomic constraints for car like robots very well. However, path generated by RRT was not optimal. Karaman and Frazzoli proposed a variation of RRT called RRT* with proven asymptotically optimal property. RRT* improved path quality by introducing major features of tree rewiring and best neighbour search. However, it obtained asymptotic optimality at the cost of execution time and slower path convergence rate. Nasir et al. presented RRT* Smart [10] with main focus on improving convergence rate of RRT*. Two major features introduced by RRT*-Smart called intelligent sampling and path optimization improved path cost and convergence rate. RRT and RRT* have numerous successful applications in robotics. Significant body of research has addressed the problem of optimal path planning using RRT* in recent years. These methodologies have gained tremendous success in solving single -query high dimensional complex problems.

This project presents a simulation based experimental comparison of the aforementioned algorithms in an environment cluttered with obstacles. Effect of different parameters on the performance of these approaches is discussed. Further, limitations and future directions for improvement are also suggested.

Literature Survey



RRT, RRT*, and RRT*-Smart operate in a configuration space, which is set of all possible transformations that could be applied to the robot [4, 11]. Let the given state

space be denoted by a set Z $\mathbf{R}^n$, n $\mathbf{N}$ where n represents the dimension of the given search space. The area of the search space which is occupied by obstacles is represented by Z_{obs} Z and region free from obstacles is represented by Z free Z / Z_{obs} .z_{goal} Z free represents goal and z_{init} Z_{free} represents starting point. z_{init} and z_{goal} are provided to planner as inputs. The problem is to find a collision free path between initial z_{init} and goal z_{goal} states in Z free , in least possible time t R , with Table 1: RRT Algorithm.

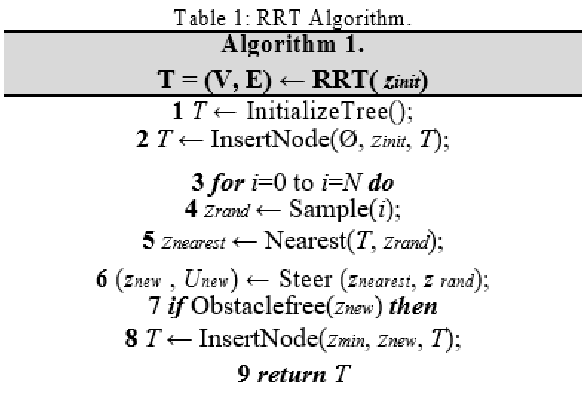


Table 1: RRT Algorithm.

```
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 T ← InsertNode($z_{min}$, $z_{new}$, T);
9 return T
```

RRT

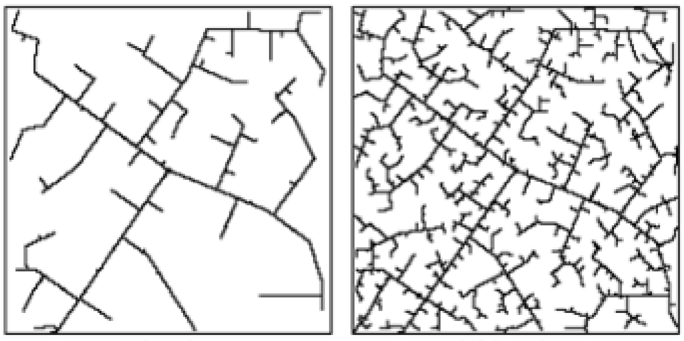
A rapidly exploring random tree (RRT) is an algorithm designed to efficiently search nonconvex, high-dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem. RRTs were developed by Steven M. LaValle and James J. Kuffner Jr. They easily handle problems with obstacles and differential constraints (nonholonomic and kinodynamic) and have been widely used in autonomous robotic motion planning.

RRTs can be viewed as a technique to generate open-loop trajectories for nonlinear systems with state constraints. An RRT can also be considered as a Monte-Carlo method to bias search into the largest Voronoi regions of a graph in a configuration space. Some variations can even be considered stochastic fractals.

An RRT grows a tree rooted at the starting configuration by using random samples from the search space. As each sample is drawn, a connection is attempted between it and the nearest state in the tree. If the connection is feasible (passes entirely through free space and obeys any constraints), this results in the addition of the new state to the tree. With uniform sampling of the search space, the probability of expanding an existing state is proportional to the size of its Voronoi region. As the largest Voronoi regions belong to the states on the frontier of the search, this means that the tree preferentially expands towards large unsearched areas.

The length of the connection between the tree and a new state is frequently limited by a growth factor. If the random sample is further from its nearest state in the tree than this limit allows, a new state at the maximum distance from the tree along the line to the random sample is used instead of the random sample itself. The random samples can then be viewed as controlling the direction of the tree growth while the growth factor determines its rate. This maintains the space-filling bias of the RRT while limiting the size of the incremental growth.

RRT growth can be biased by increasing the probability of sampling states from a specific area. Most practical implementations of RRTs make use of this to guide the search towards the planning problem goals. This is accomplished by introducing a small probability of sampling the goal to the state sampling procedure. The higher this probability, the more greedily the tree grows towards the goal.


45 iterations 390 iterations

RRT*

RRT* inherits all the properties of RRT and works similar to RRT. However, it introduced two promising features called near neighbour search and rewiring tree operations. Near neighbour operations finds the best parent node for the new node before its insertion in tree. This process is performed within the area of a ball of radius defined by

$$k = \gamma \left(\frac{\log(n)}{n} \right)^{\frac{1}{d}}$$

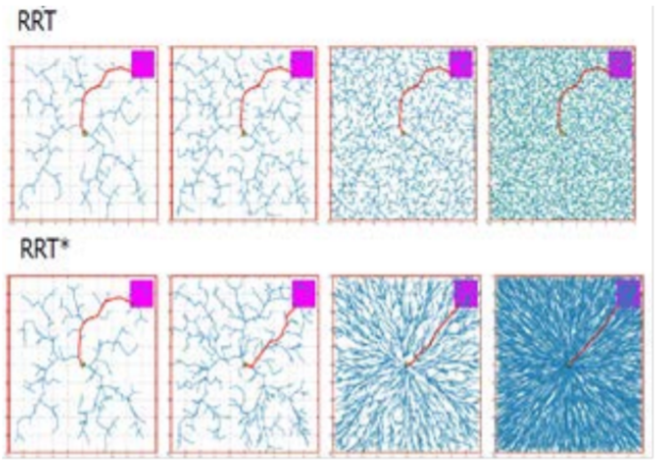
where d is the search space dimension and γ is the planning constant based on environment. Rewiring operation rebuilds the tree within this radius of area k to maintain the tree with minimal cost between tree connections. Space exploration and improvement of path quality is shown in Figure. As the number of iterations increase, RRT* improves its path cost gradually due to its asymptotic quality, whereas RRT does not improves its jaggy and suboptimal path.

Due to increased efficiency to get less jagged and shorter path, features of rewiring and neighbor search are being adapted in recent revisions of RRT*. However, these operations have an efficiency trade-off. Though, it improved path cost but on the other hand it also slowed down convergence rate of RRT*. The details of the two new features introduced in RRT* are as follows:-

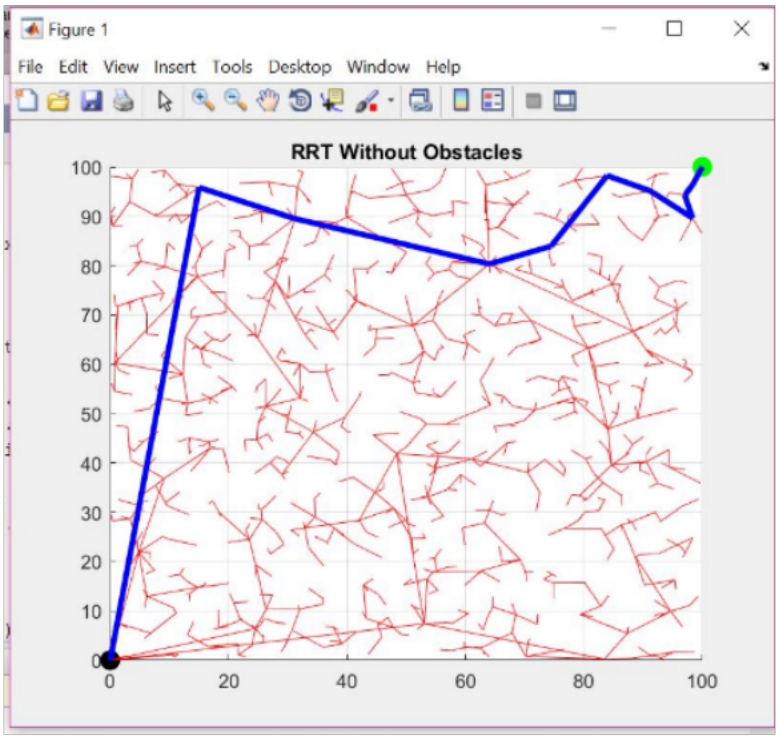
Rewire: This function checks if the cost to the nodes in z_{near} is less through z_{new} as compared to their older costs, then its parent is changed to z_{new}.

ChooseParent: This function selects the best parent z_{new} from the nearby nodes.

Simulation Results



Output



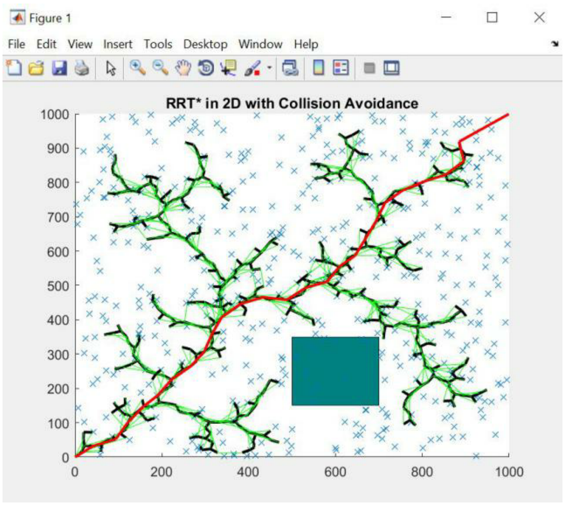
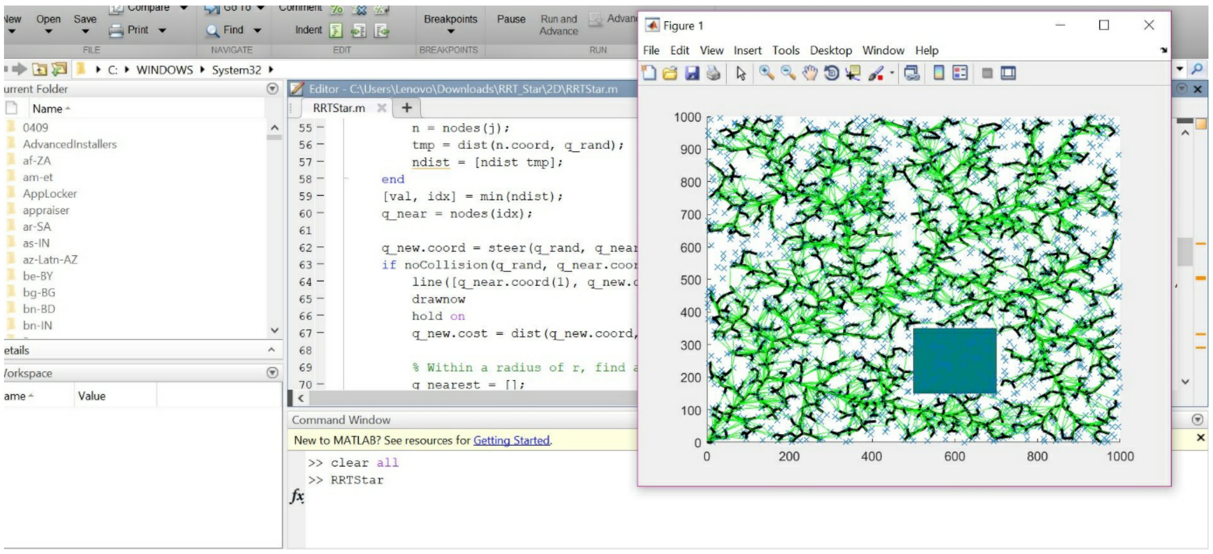


Figure 1
File Edit View Insert Tools Desktop Window Help
RRT* in 2D with Collision Avoidance



Editor - C:\Users\Lenovo\Downloads\RRT_Star\2D\RRTStar.m

```
            n = nodes(j);
            tmp = dist(n.coord, q_rand);
            ndist = [ndist tmp];
        end
        [val, idx] = min(ndist);
        q_near = nodes(idx);

        q_new.coord = steer(q_rand, q_near
        if noCollision(q_rand, q_near.coor
            line([q_near.coord(1), q_new.c
            drawnow
            hold on
            q_new.cost = dist(q_new.coord,

            % Within a radius of r, find a
            q_nearest = [];
```

Command Window

New to MATLAB? See resources for Getting Started.

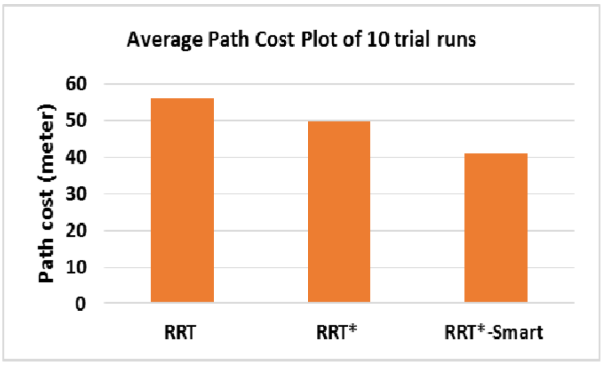
```
>> clear all
>> RRTStar
```

Results and Inferences

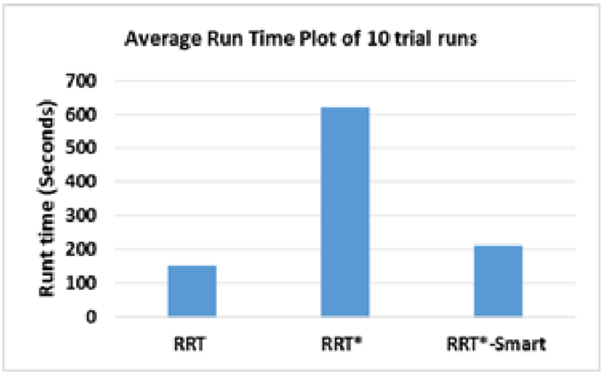
Results show that RRT* and RRT*-Smart improve initial path as compared to RRT remarkably. However, they both take more execution time than basic RRT. This is due to the fact that RRT* and RRT*-Smart use two additional operations than RRT i.e., rewiring tree and best neighbour search. These two features improve the path cost to generate less jaggy and shorter path. On the other hand, these features also slow down the convergence rate and increase computational time. Further, RRT*-Smart evidently takes less time than RRT*. Its optimization operation and intelligent sampling operation converge it quickly to shorter path in a smaller number of iterations than RRT*.

Analysis of results also showed that RRT*-Smart converges quickly than both other approaches. However, it also suffers from large number of nodes like RRT and RRT*. Because switching to intelligent sampling in RRT*-Smart also generates denser tree in identified beacons area. In all these approaches, tree gets populated with such nodes which do not contribute in the final solution. They merely increase the tree density and memory requirements consequently effecting computational time. Developing operations to limit tree nodes to maximum useful nodes could increase the efficiency of these approaches.

Cutting-edge research has evidenced that RRT and RRT* based approaches are successful for high dimensional complex problems. The analysis of these approaches have shown that they confront issues of optimality and large number of nodes in search space. They improve path quality at the cost of computational efficiency. This performance trade-off could lead to more difficulty while dealing with high dimensions, high degrees of freedom and non-holonomic constraints. We plan to grow search tree more intelligently with improved computational efficiency for high dimensional complex problem using wheeled mobile robots. Future work in this direction is a thriving area of research.



(a) Path cost analysis for obstacle cluttered



(b) Run time analysis for obstacle cluttered

References

1. A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms Iram Noreen1 , Amna Khan2 , Zulfiqar Habib3 Department of Computer Science, COMSATS Institute of Information Technology, Lahore.
2. Informed Sampling for Asymptotically Optimal Path Planning -Jonathan D. Gammell, Member,IEEE, Timothy D. Barfoot, Senior Member, IEEE,and Siddhartha S. Srinivasa, Fellow, IEEE