Analytical Study of RRT Algorithm Enhancements Visualized Using ROS Gazebo

Pulkit Mehta
UID: 117551693
University of Maryland
College Park, MD, USA

Anirudh Krishnan Komaralingam
UID: 117446432
University of Maryland
College Park, MD, USA

I. Introduction

Autonomous capabilities of a robot are highly dependent on ability of the robot to plan its motion. This holds for mobile robots as well as for manipulators. Among the various methods of path planning, Rapidly-Exploring Random Tree (RRT) based methods are popular and ubiquitous because of their advantages over traditional search-based planning methods, namely space and time complexity. As these methods are increasingly used, several new iterations of improvements to the RRT algorithm are developed. In many cases, these enhancements are suitable to be used outside of their initial purpose. This report studies the performance of a collection of RRT algorithm enhancements.

The efficacy of the various methods is tested in two ways. To visualize the specific technical difference of each method, a point robot in a 2D environment with obstacles is established. A point robot is made to plan its path and move from a prescribed start point to a goal point. Then ROS and Gazebo are used to simulate a mobile robot in an identical environment with the same obstacles. The robot follows the path generated by each algorithm.

II. Motivation

The original RRT algorithm from LaValle [3] samples points in the configuration space at random. The resulting path can be highly sub-optimal in a complex environment. Biasing the search to the goal in any proportion opens the possibility of being trapped in a local minimal feature or obstacle.

To address the quality of the path is jagged by nature of the search, path smoothing steps are employed after the initial path to the goal is located. RRT* is a well known example of this, where new nodes are placed to be locally optimal and the tree is frequently rewired to reflect the existence of new nodes. The path produced this way tends to be the optimal path as the number of nodes tends to infinity, although in reality far fewer nodes are needed to produce a usable solution. This path optimization comes at the cost of overhead and RRT* paths are slower to generate than RRT ones.

RRT-Connect from Kuffner and LaValle give us a faster implementation of RRT which involves growing an RRT from both start and goal point and having the trees intersect. This does not address the quality of the path generated but generally improves performance in complex environments and is faster to converge.

A recent enhancement build on RRT-Connect comes from Chen, Zhao, and Xu [1] in the form of Improved RRT-Connect (IRRT-Connect). This methods developed with an environment of changing obstacles in mind grows a new RRT from a point in between the ones generated at the start and goal point. This aims to bridge broken paths in the event of an obstacle cutting off the initial solution.

III. RRT

The Rapidly exploring Random Trees(RRT) algorithm is a sampling-based path planning algorithm, which is popular due to its simplicity and fast search capability, and its adeptness at solving path planning problems in multi-dimensional spaces and complex environments. RRT constructs a path search tree by incrementally generating a random series of sample points in Configuration Space with the starting point as the root node. Fig 1 shows the path planning process of the RRT algorithm. The algorithm generates random sampling points in space, iterates through all the nodes in the search tree and finds the nearest node to the sampling point as the expansion node of the tree. Then determine whether there is an obstacle in the line, if there is, then discard the sample point and regenerate the sample point; if not, generate new nodes by connecting in the direction of random point with the minimum step E as the length, and add the newly generated nodes to the random tree. The above steps are repeated until the end point is reached or the set maximum number of iterations is reached. Finally, a path from the starting point to the end point is obtained by retracing all nodes of the random tree from the end point. Since the nodes are sampled randomly, the paths obtained by this algorithm are also not globally optimal. It is the stochastic nature of the RRT path planning algorithm that causes the algorithm to have probabilistic completeness in complex environments, where the probability of the algorithm finding a feasible path converges to one given a sufficiently large number of planning iterations.

Algorithm 1: RRT.

$T.V \leftarrow x_{\text{init}}$;
while $NotReachStop$ **do**
  $x_{\text{rand}} \leftarrow Sample\_free$;
  $x_{\text{nearest}} \leftarrow NearestNode(x_{\text{rand}}, T.V)$;
  $x_{\text{new}} \leftarrow Steer(x_{\text{nearest}}, x_{\text{rand}})$;
  if $Collision\_check(x_{\text{nearest}}, x_{\text{new}}) == True$ **then**
    $T.V \leftarrow T.V \bigcup x_{\text{new}}$;
  end if
end while

Fig. 1. RRT Algorithm

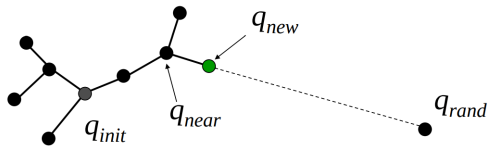

Fig. 2. RRT growth strategy

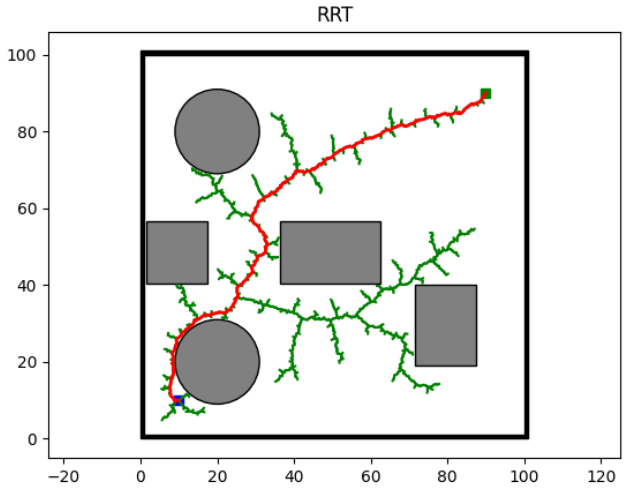

Fig. 3. Path found using RRT

IV. RRT Enhancements

A. RRT*

The RRT approach is described in Fig 4. The RRT is actually a variant of the method RRT, but the main difference from RRT is that RRT* will go on with the procedure of Rewire after new node is generated and added into the tree structure. In the view of optimization, RRT can only guarantee establishing a plan rapidly but not to optimize it. In contrast, RRT*, inheriting the merits of RRT and A*, enables the local plan to converge to global optimal through the procedure of Rewire. This Rewire procedure is described in Fig 5. The RRT* algorithm takes into account the cost of each node to the starting point, the lower the cost, the more likely it is to be selected as the path node. The RRT* algorithm finds the initial path in the same way as RRT, the difference being that the former does not end once it has found the initial path, but continues to generate sample points and continually updates the initial path. As the number of sample points increases, the initial path moves closer to the optimal path. Compared to the RRT and RRT-Connect algorithms, the RRT* algorithm improves the quality of the path but requires a large number of iterations and a slower convergence rate, so the planning time increases substantially.

Algorithm 2: RRT*.

$T.V \leftarrow x_{\text{init}}$;
while $NotReachStop$ **do**
  $Cost_{\text{min}} \leftarrow +\infty$;
  $x_{\text{rand}} \leftarrow Sample\_free$;
  $x_{\text{nearest}} \leftarrow Nearest(x_{\text{rand}}, T.V)$;
  $x_{\text{temp}} \leftarrow Steer(x_{\text{nearest}}, x_{\text{rand}})$;
  if $Collision\_check(x_{\text{nearest}}, x_{\text{new}}) == True$ **then**
    $x_{\text{new}} = x_{\text{temp}}$;
  else
    Continue;
  end if
  $X_{\text{near}} \leftarrow K-NearsestSets(x_{\text{new}}, T.V)$;
  $Cost_{\text{min}} \leftarrow CostFromInit(x_{\text{new}})$;
  for $x_{\text{near}} \in X_{\text{near}}$ **do**
    $x_{\text{parent}} \leftarrow Rewire(x_{\text{new}}, x_{\text{near}})$;
  end for
  $T.V.add(x_{\text{new}})$;
  $T.E.add(x_{\text{parent}}, x_{\text{new}})$;
  for $x_{\text{near}} \in X_{\text{near}}$ **do**
    $T \leftarrow Rewire(x_{\text{near}}, x_{\text{new}})$;
  end for
end while

Fig. 4. RRT* Algorithm

Algorithm 3: $Rewire(x_1, x_2)$.

if $Condition\_check(x_1, x_2) == True$ **then**
  if $Cost(x_1, x_2) + CostFromInit(x_1) < Cost_{\text{min}}$ **then**
    $Cost_{\text{min}} \leftarrow (Cost(x_1, x_2) + CostFromInit(x_1))$;
    $x_{\text{parent}} \leftarrow x_2$;
  end if
end if

Fig. 5. Rewiring

B. RRT*-Smart

Initially, RRT*-Smart randomly searches the state space as RRT* does. Similarly, the first path is found just like the RRT* would try to find a path by random sampling in the configuration space. Once this first path is found, it then optimizes it by interconnecting the directly visible nodes. This optimized path yields biasing points for intelligent sampling. At these biasing points, sampling takes place at regular intervals, which are governed by a constant b that in turn depends upon the biasing ratio. This process is continued, as the algorithm progresses and the path is optimized continuously. Whenever, a shorter path is found, the biasing shifts towards the new path.

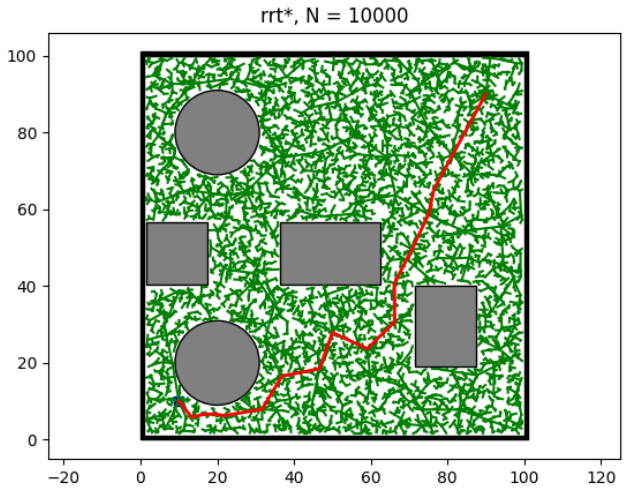

Fig. 6. Path found using RRT*

```
1 T ← InitializeTree();
2 T ← InsertNode(Ø, zinit, T);
3 for i=0 to i=N do
4   if i=n+b, n+2b, n+3b.... then
5     zrand ← Sample(i, zbeacons);
6   else
7     zrand ← Sample(i);
8     znearest ← Nearest(T, zrand);
9   (xnew, unew, Tnew) ← Steer (znearest, zrand);
10  if Obstaclefree(xnew) then
11    Znear ← Near(T, znew, |V|);
12    zmin ← Chooseparent (Znear, znearest, znew, xnew);
13    T ← InsertNode(zmin, znew, T);
14    T ← Rewire (T, Znear, zmin, znew);
15    if InitialPathFound then
16      n ← i;
17    (T, directcost) ← PathOptimization(T, zinit, zgoal);
18    if (directcostnew < directcostold)
19      zbeacons ← PathOptimization(T, zinit, zgoal);
20 return T
```

Fig. 7. RRT*-Smart Algorithm

C. Path Optimization

Once RRT* gives an initial path, the nodes in the path that are visible to each other are directly connected. An iterative process starts from goal moves towards initial checking for direct connections with successive parents of each node until the collision free condition fails. By the end of this process, no more directly connectable nodes are present. Hence the path is optimized based on the concept of the Triangular Inequality as illustrated in Fig. 8. According to the Triangular Inequality, c is always less than the sum of a and b, and hence always gives a shorter path.

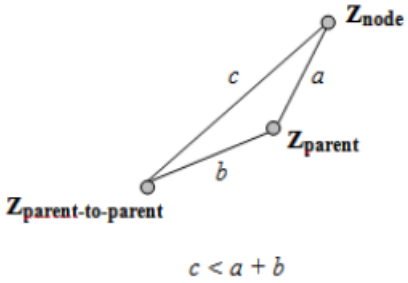

Fig. 8. Triangular Inequality

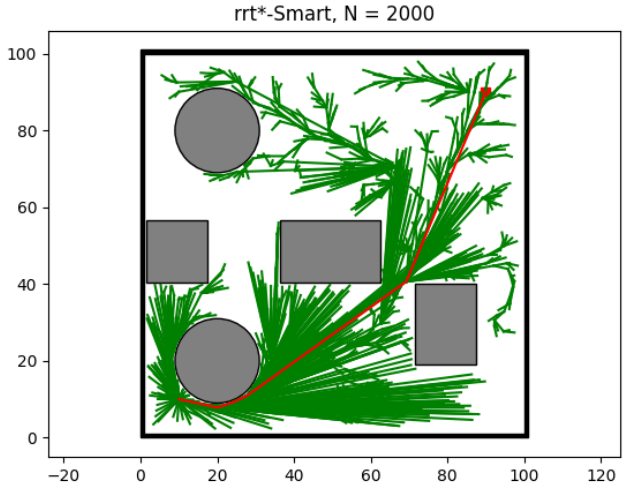

Fig. 9. Path found using RRT*-Smart

D. RRT-Connect

Since the RRT algorithm generates sampled nodes with the same probability throughout the configuration space and does not consider expansion in the direction of the target point, spanning tree exploration is somewhat blind. To solve this problem, a bidirectional RRT algorithm, RRT-Connect with the idea of greedy extensions, was proposed. Compared to the RRT algorithm, RRT-Connect makes two significant improvements: (1) It generates a random tree from the start and end state points, and the planning is completed when the two trees intersect, which greatly improves the search speed of the algorithm. (2) A node-greedy expansion strategy is used, where at each iteration of node generation, the algorithm tries to use the nearest node of another tree as the expansion direction of this tree, making the two trees intersect quickly. Also, the algorithm will continue to expand a node in that direction if it does not encounter an obstacle when expanding in that direction. If an obstacle is encountered, the algorithm will use a swap function that allows another random tree to be expanded, which largely avoids the algorithm falling into a local optimum dilemma.

By using several of these strategies, RRT-Connect search

speed and search efficiency have been dramatically improved. Fig. 10 shows the RRT-Connect algorithm extension process. The algorithm is initially two trees with the start and end points as root nodes, and is extended greedily, effectively reducing the extension of the random tree to unknown regions. However, RRT-Connect is similar to the RRT algorithm in that although a feasible path can be obtained after several iterations, there is no guarantee that the resulting path is optimal or sub-optimal.

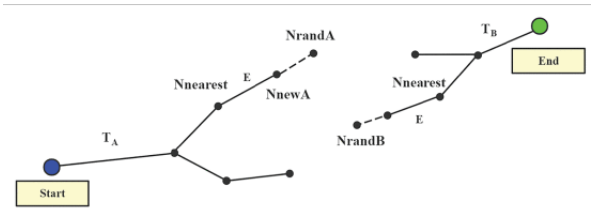

Fig. 10. RRT-Connect Extension.

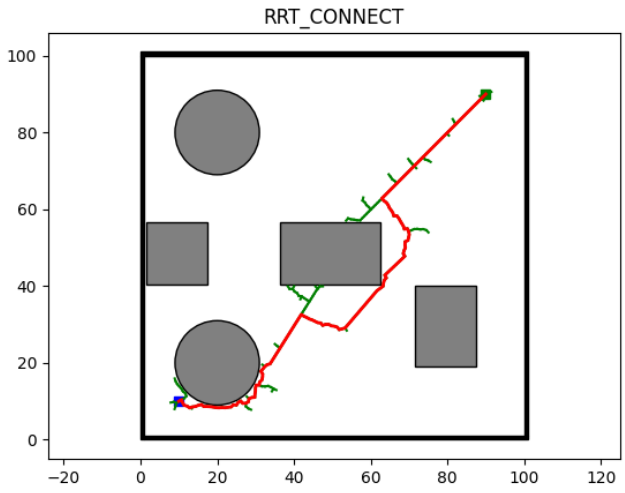

Fig. 11. Path found using RRT-Connect.

E. Improved RRT-Connect

IRRT-Connect builds on RRT-Connect with the following modifications: (1) In order to speed up the exploration of spanning trees, this paper generates a simple and efficient third node in the configuration space based on the idea of dichotomous points, allowing the algorithm to be extended with four trees. (2) In order to solve the blind search feature of the traditional RRT-Connect algorithm, by increasing the guiding force, the spanning tree is biased towards the direction of the target point every time when expanding new nodes, which speeds up the search efficiency of the algorithm.
This technique has proven to reduce the number of required iterations, reduce the path finding time and results in a shorter path compared to standard RRT-Connect.

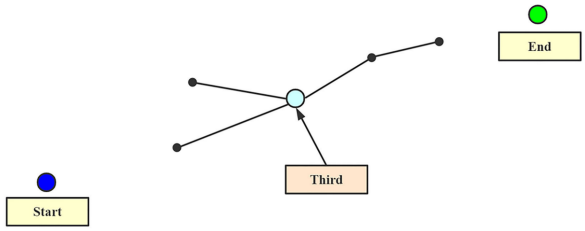

Fig. 12. IRRT-Connect Strategy

V. Simulation

The focus of this work is on the analysis of different sampling based algorithms, mainly variations of RRT rather than the modelling aspect. The generated path coordinates obtained by different methods are sequentially fed to move the turtlebot in the given world. An occupancy grid map is created by collecting the laser scanned data of the gazebo world by moving the turtlebot. For the sake of simplicity, movebase is used as a local planner to efficiently move the turtlebot to the desired point coordinates generated from the RRT algorithms. The figures below show the gazebo world and the turtlebot navigation in RViz.

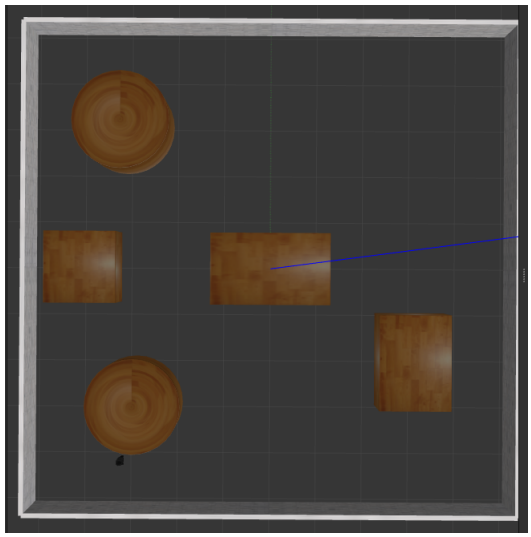

Fig. 13. Gazebo world with Turtlebot.

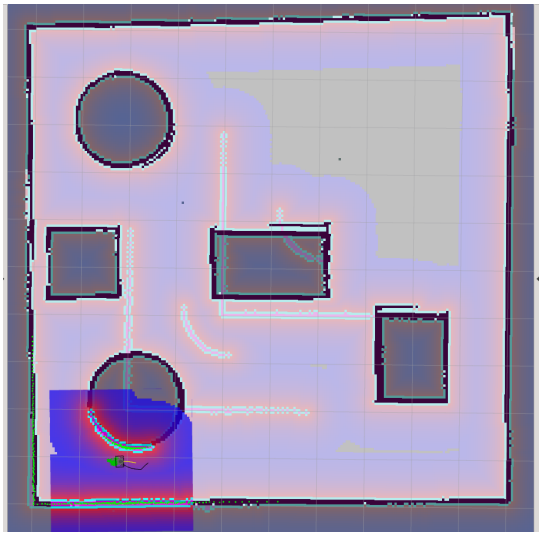

Fig. 14. Turtle navigating to point coordinates found using RRT-Connect in RViz.

REFERENCES

[1] J. Chen, Y. Zhao and X. Xu, "Improved RRT-Connect Based Path Planning Algorithm for Mobile Robots," in IEEE Access, vol. 9, pp. 145988-145999, 2021, doi: 10.1109/ACCESS.2021.3123622.

[2] L. Chen, Y. Shan, W. Tian, B. Li and D. Cao, "A Fast and Efficient Double-Tree RRT-Like Sampling-Based Planner Applying on Mobile Robotic Systems," in IEEE/ASME Transactions on Mechatronics, vol. 23, no. 6, pp. 2568-2578, Dec. 2018, doi: 10.1109/TMECH.2018.2821767.

[3] Steven M. LaValle, Planning Algorithms. 2006. Cambridge University Press.