Using rapidly-exploring random tree-based algorithms to find smooth and optimal trajectories

B MATEBESE[1], MK BANDA[2] AND S UTETE[1]
[1]CSIR Modelling and Digital Science, PO Box 395, Pretoria, South Africa, 0001
[2]Department of Applied Mathematics, Stellenbosch University, Private Bag X1, Matieland, South Africa, 7602
Email: bmatebese.co.za – www.csir.co.za

ABSTRACT

Sampling-based methods such as Rapidly-exploring Random Tree (RRT) have been successfully used in solving motion planning problems in high-dimensional and complex environments. The RRT algorithm is the most popular and has the ability to find a feasible solution faster than other algorithms. The drawback of using RRT is that, as the number of samples increases, the probability that the algorithm converges to a sub-optimal solution increases. Furthermore, the path generated by this algorithm is not smooth (tree form). The RRT-based methods will be discussed and simulations are given to evaluate the performance of the methods.

1. INTRODUCTION

Over the past decade, significant progress has been made in motion planning, especially in the field of robotics. The motion planning problem is the fundamental problem of finding a path that will take a robot from a given initial state to a goal state without colliding with obstacles. Sampling-based algorithms such as Probabilistic Roadmaps (PRM) and RRT were introduced to solve motion planning problems in high-dimensional environments.

RRT is the most popular sampling-based method and was developed by Lavalle [1]. In this method a tree is grown incrementally from the initial state to a goal state, a feasible path is found by adding a new edge or vertex in each iteration, while avoiding obstacles. Hence, it has an advantage of finding a feasible path relatively quickly in high-dimensional and complex environments.

However, as the number of samples increase, the probability that the RRT algorithm converges to an optimal solution increases. To address this issue, the RRT* algorithm was introduced by Karaman and Frazolli [2]. The RRT* algorithm preserves the asymptotic-optimal properties, i.e., it almost converges to an optimal solution.

The other key limitation of RRT-based methods is that it produces tree-like (non-smooth) solutions. This can be a challenging task for a robot, and may cause uncertainties. These issues have drawn research interest to improve RRT-based methods to provide high quality solutions that are not only optimal, but also preserve smoothness. By doing so, these methods should have the ability to produce low cost solutions and reduce uncertainty.

2. ALGORITHMS

In this section the RRT and RRT* algorithms are discussed. The pseudo-codes for the algorithms used for the simulations are given.

2.1 RRT Algorithm

The algorithm initialises a tree with initial state as a starting position (Line 1). The random node is added by connecting it to the nearest node that is already in the tree (Line 3-12).

Algorithm 1 RRT Algorithm

1: Initialize the tree T with $x_{initial}$
2: **while** $i \leq N$ **do**
3: select a random node: x_{rand}
4: select the nearest node of x_{rand} : x_{near}
5: select input from x_{near} to the direction of x_{rand} : u
6: generate a new node by using u at x_{near} for Δt: x_{new}
7: add x_{new} to the T
8: add the new edge (x_{near}, x_{new}, u) to T
9: **if** $GoalFound$ == true **then**
10: exit: $i = N$
11: **end if**
12: **end while**
13: **return** T

These steps are repeated until the goal state is reached, and a path is found from initial position to goal position.

2.2 RRT* Algorithm

The RRT* algorithm essentially behaves the same as the RRT, except that RRT* considers all nodes in a neighbourhood of the random node and choose one with minimum cost distance (Line 4-11).

Algorithm 2 RRT* Algorithm

1: Initialize the tree T with $x_{initial}$
2: **while** $i \leq N$ **do**
3: select a random node: x_{rand}
4: select a set of nearest node of x_{rand} : X_{near}
5: **if** $CollisionFree(X_{near}, x_{rand})$ **then**
6: choose a parent node from X_{near} with minimum cost distance :x_{min}
7: **for** $x_{near} \in X_{near}$ **do**
8: **if** $CollisionFree(x_{min}, x_{rand})$ **then**
9: add x_{rand} to the tree and connect it to x_{min}
10: **end if**
11: **end for**
12: **for** $x_{near} \in X_{near}/\{x_{min}\}$ **do**
13: **if** $CollisionFree(x_{near}, x_{rand})$ **then**
14: connect x_{rand} to each node in $x_{nearest}$
15: select the node with minimum cost distance than its current parent: x_{near}
16: rewire x_{rand} and x_{near}
17: **end if**
18: **end for**
19: **end if**
20: **end while**
21: **return** T

Furthermore, adjusting the length of new connections (Line 12-18), assures that RRT* finds a more optimal path than RRT simulations.

In this section the RRT and RRT* algorithms are applied to a point problem in 2-dimensional space. Due to robots' geometric and complex shapes, solving a motion planning problem can be challenging. To simplify this problem, a robot is reduced to a point. The task is to find a path from initial point to goal point while avoiding obstacles. When a goal is found, the algorithm stops, and the best path is found by backtracking through the parent nodes from the last node to the initial node.

The best paths for RRT and RRT* are shown in **Figures 1** and **2** below. Note that in the simulations the tree expands to the corners. This agrees with the literature that the algorithms are indeed biased towards unexplored regions.

Two step sizes were used for the simulations. A step size determines the distribution of nodes during the planning process, and hence, determines the overall performance of an algorithm.

In **Figure 1** a step size of 0.02 was used, and it can be seen that the explored path is denser and has short average distance.

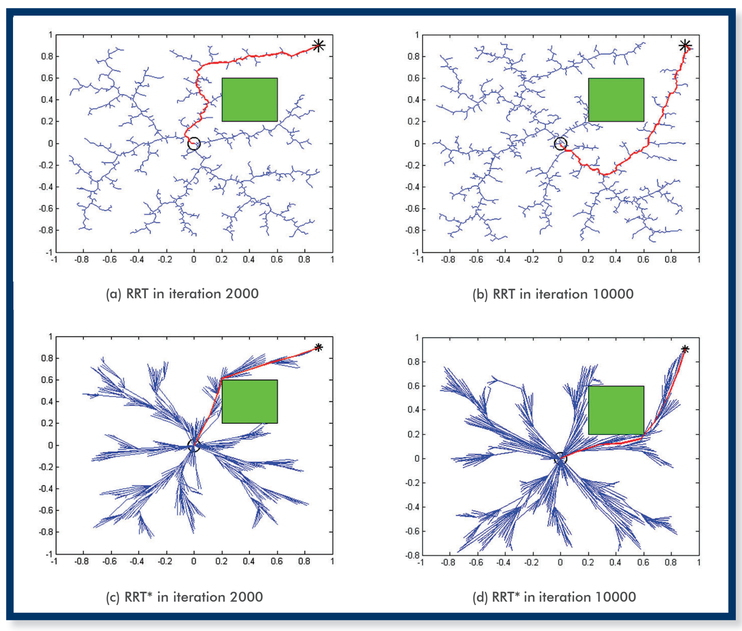


Figure 1: The tree generated by the RRT algorithm is shown in (a) and (b) and the RRT* algorithm in (c) and (d). The initial state is represented by a black circle; goal region (star), the green region represents an obstacle and the best path is highlighted in red.

A bigger step size of 0.06, as seen in **Figure 2**, produces nodes that more closely follow the sampling distribution, since the reach of the layer path segment is more likely to connect to any sample.

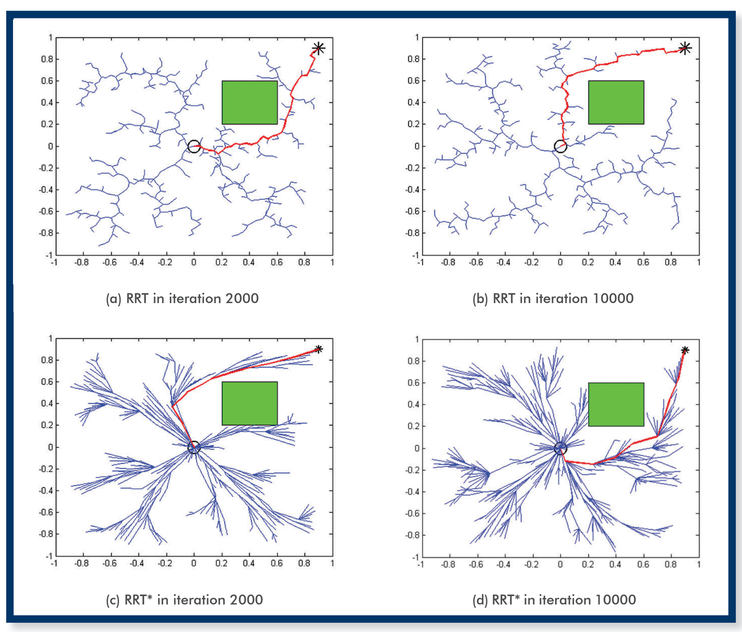
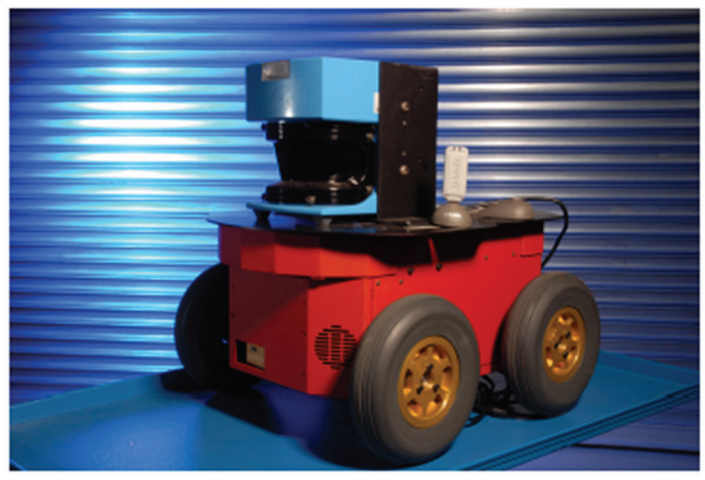


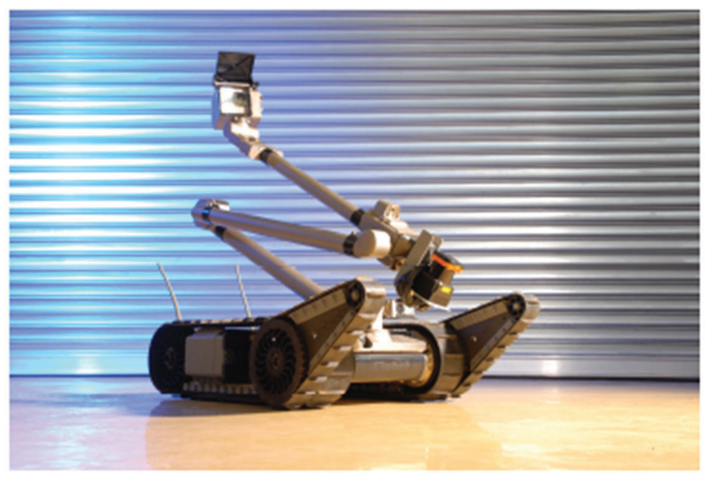
Figure 2: The tree generated by RRT and RRT* algorithms in different stages given a bigger step size.

Also note that the paths generated by these algorithms are not smooth. This induces jerky movements in the solution animation. For a robot to follow such awkward paths is challenging and can cause slippage, inaccuracy, and instability on the robot.

Sampling-based methods are used on autonomous mobile robots to find solutions that are not only optimal, but also preserve some degree of smoothness.



(a) Pioneer



(b) Packbot510*i* robot with a manipulator

4. CONCLUSION

The sampling-based method, RRT, was presented. The challenge with the RRT algorithm is that it produces sub-optimal solutions. An extension to this algorithm, RRT*, was also presented. Simulations for these algorithms were given.

Through these simulations it was shown that using a small step size is ideal, because it gives short average distances and is less likely to stretch over an obstacle. It was also shown that the resulting paths from the algorithms are non-smooth and therefore a smoothing technique is required.

For future work, the RRT and RRT* algorithms will be extended to find optimal and smooth paths. The aim is to then implement these algorithms on mobile platforms, i.e., a Pioneer and Packbot510i with a manipulator.

5. ACKNOWLEDGMENT

This work was supported by the CSIR, Mobile Intelligence Autonomous Systems (MIAS) research area.

6. REFERENCES

1. LaValle, S.M. 1998. Rapidly-exploring random trees: A new tool for path planning. Tech. Rep.
2. Karaman, S. and Frazzoli, E. 2010. Incremental sampling-based algorithms for optimal motion planning. Robotics: Sciences and Systems (RSS).

K-10032 [www.kashan.co.za]