Rapidly Exploring Random Trees

|  |  |  |
| --- | --- | --- |
| **MENTOR** | Professor N. S. Kumar |  |
| **MENTEE** | Arya Rajiv Chaloli | PES1201700253 |
| **SUBJECT** | Design and Analysis of Algorithms | UE17CS251 |

**AIM**

Create an implementation of the Rapidly-exploring Random Trees (RRT) for path planning applications in the C Programming language and to delve further on to improving the efficiency during run-time by parallelizing the operation of generating the tree.

**DESCRIPTION**

Motion planning is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement.

Primarily it consists of a configuration space, the set of all possible configurations that the system can be in, which comprises of the free space, obstacle space and the target space.

Motion planning has several robotics applications, such as autonomy, automation, and robot design in CAD software, as well as applications in other fields, such as animating digital characters, video game, artificial intelligence, architectural design, robotic surgery, and the study of biological molecules.

**APPROACH**

Motion or path planning can be implemented using one of the following approaches.

1. Grid-based algorithms, that overlay a grid on top of configuration space, or geometric algorithms that compute the shape and connectivity of the free space, are used to solve low-dimensional path planning problems.
2. Exact motion planning for high-dimensional systems under complex constraints is computationally intractable.
3. Potential-field algorithms are efficient, but fall prey to local minima with an exception of harmonic potential fields.
4. Sampling-based algorithms avoid the problem of local minima, and solve many problems quite quickly. They are unable to determine that no path exists, but they have a probability of failure that decreases to zero as more time is spent. Sampling-based algorithms are currently considered state-of-the-art for motion planning in high-dimensional spaces, and have been applied to problems which have dozens or even hundreds of dimensions (robotic manipulators, biological molecules, animated digital characters, and legged robots).

**REFERENCES**

1. <https://en.wikipedia.org/wiki/Motion_planning>
2. Nikolaus Correll

Introduction to Autonomous Robots,

2016 Magellan Scientiﬁc

<https://github.com/correll/Introduction-to-Autonomous-Robots/releases>

1. Planning Algorithms

By [Steven M. LaValle](http://msl.cs.uiuc.edu/~lavalle/),

Copyright 2006

Cambridge University Press,

<http://planning.cs.uiuc.edu/>

Theory:

A **rapidly exploring random tree** (RRT) is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) designed to efficiently search [nonconvex](https://en.wikipedia.org/wiki/Convex_space), high-dimensional spaces by randomly building a [space-filling tree](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Steven_M._LaValle) and [James J. Kuffner Jr.](https://en.wikipedia.org/wiki/James_J._Kuffner_Jr.) [[1]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-lavalle_tr98-1) .[[2]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-lavalle_ijrr01-2) They easily handle problems with obstacles and differential constraints ([nonholonomic](https://en.wikipedia.org/wiki/Degrees_of_freedom_(engineering)) and kinodynamic) and have been widely used in [autonomous](https://en.wikipedia.org/wiki/Autonomous) [robotic](https://en.wikipedia.org/wiki/Robotics) [motion planning](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Monte_Carlo_method) method to bias search into the largest [Voronoi regions](https://en.wikipedia.org/wiki/Voronoi_diagram) of a graph in a configuration space. Some variations can even be considered [stochastic](https://en.wikipedia.org/wiki/Stochastic) [fractals](https://en.wikipedia.org/wiki/Fractal).[[3]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-3)

## Description

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](https://en.wikipedia.org/wiki/Voronoi_diagram). As the largest [Voronoi regions](https://en.wikipedia.org/wiki/Voronoi_diagram) 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.

## Variants and improvements for motion planning[[edit](https://en.wikipedia.org/w/index.php?title=Rapidly-exploring_random_tree&action=edit&section=3)]

* Parti-game directed RRTs (PDRRTs),[[5]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-5) a method that combines RRTs with the parti-game method[[6]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-6) to refine the search where it is needed (for example around obstacles) to be able to plan faster and solve more [motion planning](https://en.wikipedia.org/wiki/Motion_planning) problems than RRT
* Closed-loop rapidly-exploring random (CL-RRT),[[7]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-7) an extension of RRT that samples an input to a stable closed-loop system consisting of the vehicle and a controller

It has been shown that, under 'mild technical conditions', the cost of the best path in the RRT converges almost surely to a non-optimal value.[[8]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-incremental-8) For that reason, it is desirable to find variants of the RRT that converges to the optimum:

* Rapidly-exploring random graph (RRG) and RRT\*,[[8]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-incremental-8)[[9]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-9)[[10]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-10) a variant of RRT that converges towards an optimal solution
* RRT\*-Smart,[[11]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-11) a method for [accelerating the convergence rate](https://en.wikipedia.org/wiki/Acceleration_of_convergence) of RRT\* by using path optimization (in a similar fashion to [Theta\*](https://en.wikipedia.org/wiki/Theta*)) and intelligent sampling (by biasing sampling towards path vertices, which – after path optimization – are likely to be close to obstacles)
* A\*-RRT and A\*-RRT\*,[[12]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-12) a two-phase [motion planning](https://en.wikipedia.org/wiki/Motion_planning) method that uses a [graph search algorithm](https://en.wikipedia.org/wiki/Graph_search_algorithm) to search for an initial feasible path in a low-dimensional space (not considering the complete state space) in a first phase, avoiding hazardous areas and preferring low-risk routes, which is then used to focus the RRT\* search in the continuous high-dimensional space in a second phase
* RRT\*FN,[[13]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-13)[[14]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-14)[[15]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-15) RRT\* with a fixed number of nodes, which randomly removes a leaf node in the tree in every iteration
* RRT\*-AR,[[16]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-16) sampling-based alternate routes planning
* Informed RRT\*,[[17]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-17)[[18]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-18) improves the convergence speed of RRT\* by introducing a heuristic, similar to the way in which [A\*](https://en.wikipedia.org/wiki/A*) improves upon [Dijkstra's algorithm](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)
* Real-Time RRT\* (RT-RRT\*),[[19]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-19) a variant of RRT\* and informed RRT\* that uses an online tree rewiring strategy that allows the tree root to move with the agent without discarding previously sampled paths, in order to obtain [real-time](https://en.wikipedia.org/wiki/Real-time_computing) path-planning in a dynamic environment such as a computer game
* RRTX and RRT#,[[20]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-20)[[21]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-21) optimization of RRT\* for dynamic environments
* Theta\*-RRT,[[22]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-22) a two-phase [motion planning](https://en.wikipedia.org/wiki/Motion_planning) method similar to A\*-RRT\* that uses a hierarchical combination of [any-angle search](https://en.wikipedia.org/wiki/Any-angle_search) with RRT motion planning for fast trajectory generation in environments with complex [nonholonomic](https://en.wikipedia.org/wiki/Nonholonomic) constraints
* RRT\* FND,[[23]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-23) extension of RRT\* for dynamic environments
* RRT-GPU[[24]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-24), three-dimensional RRT implementation that utilizes hardware acceleration
* APF-RRT, [[25]](https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree#cite_note-25) A combination of RRT planner with Artificial Potential Fields method that simplify the replanning task

References:

1. Web page maintained by [Steve LaValle](http://msl.cs.uiuc.edu/~lavalle)

<http://msl.cs.uiuc.edu/rrt/>

1. Planning Algorithms

By [Steven M. LaValle](http://msl.cs.uiuc.edu/~lavalle/),

Copyright 2006

Cambridge University Press,

<http://planning.cs.uiuc.edu/>

1. Adaptive Motion Planning with Artificial Potential Fields Using a Prior Path
2. Further on RRTs by the author himself

<http://msl.cs.uiuc.edu/~lavalle/rrtpubs.html>

1. more...

Other Ref:

1. Wiki : <https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree>
2. Python implementation : <https://github.com/jpgmoreira/RRT-Path-Finder>
3. All robotics algo : <https://github.com/AtsushiSakai/PythonRobotics>
4. C++ implementation : <https://github.com/RoboJackets/rrt>