Anytime Motion Planning using RRT*

Vasista Ayyagari
Robotics Engineering
A. James Clark School of Engineering
University of Maryland
College Park, USA.
vasista1997@gmail.com

Naman Gupta
Robotics Engineering
A. James Clark School of Engineering
University of Maryland
College Park, USA.
ngupta98@terpmail.umd.edu

Moumita Paul
Robotics Engineering
A. James Clark School of Engineering
University of Maryland
College Park, USA.
moumita@umd.edu

***Abstract*—The RRT algorithm is an incremental sampling based algorithm that efficiently computes motion plans but it converges far from the optimal solution. RRT* on the other hand, computes feasible solution quickly but also converges at an optimal solution. In practical scenario, anytime algorithms are favored that keeps on improving the solution towards an optimal solution. This report is an implementation of a conference paper titled *'Anytime Motion Planning using RRT*'* where it presents two key extensions of RRT algorithm. Committed trajectories and Branch-and-Bound Tree adaptations, which together enables the algorithm to make more efficient use of computation time online, resulting in an anytime algorithm for real-time implementation. We implemented the method using ROS Gazebo as our simulation environment and compared the operation of the RRT and RRT using Python and OpenCV. We demonstrated the experimental results using Turtlebot 3 for displaying the path generation.**

I. Introduction

The Motion Planning problem in Robotics attempts to solve for the best possible path for a given set of environmental condition. Multiple algorithms solve and optimise different parameters that results in different path across various spaces and dimensions [1]. There are two sets of conditions that Path Planning tries to achieve while generating paths. These are collision avoidance and minimize the total path length or time taken to traverse the path [2]. Algorithms like Breadth First Search, A* and Dijkstra are some of the popular algorithms that achieve these conditions. However, a major drawback of the algorithms is that they do not scale to dimension as their run time grows exponentially [3].

Rapidly-Growing Random Trees(RRT) is an another popular randomized algorithm that provides feasible paths albeit not optimal in acceptable run times and scales well to dimensions. RRT efficiently searches non-convex, high-dimensional spaces by randomly building a space-filling tree. The tree grows incrementally by randomly drawing samples from the search space . But the Growth factor can limit the length of the connection between the tree and a new state.Therefore, the RRT algorithm gives sub-optimal paths [4].

The RRT* is a improved version of the RRT that provides asymptotically optimal paths in terms of euclidean distances but runs slightly slower than the former. It achieves better paths by reconnecting the nodes to the node with the least cost within a neighbourhood [5].

These algorithms can also handle differential constraints. However, the application of these algorithms is limited as a prepossessing step rather than a real-time applications [6]. The existing RRT* algorithms can be modified or extended to achieve real-time motion planning aka. anytime-planning. In this project, we propose the implementation of ”Anytime Motion Planning using the RRT*” by Karaman S, et al. This paper addresses the challenge of anytime planning using an extension of RRT* on differential constrained robots in a static environment.

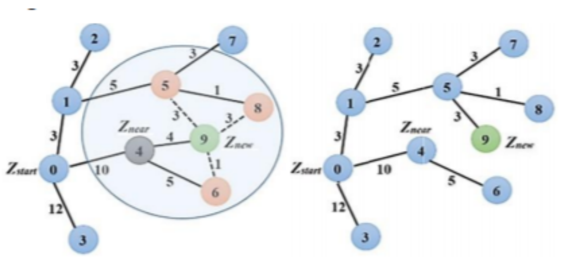
A version of RRT* is proposed [7] that consists of a prepossessing method that creates a quick and feasible path using RRT* and an online method that optimises the path and potentially change the initially proposed path on the fly. The pre-possessing step is the standard RRT* algorithm while the paper proposes two extensions for the online method namely the ”Committed Trajectory” and ”Branch-and-Bound Technique”. The Committed Trajectory technique commits to an initial fraction of the generated path and improves the rest of the path by generating more node for the RRT* algorithm, this creates a path with lower costs. The Branch-and-Bound Technique periodically deletes nodes from the tree whose cost is greater than the pre-generated path to improve search efficiency. Finally, the paper discusses the dynamics of the Dubins Curve steering functions. The results show improved paths that have been generated the robots' control phase and hence stride towards real-time implementations.

II. Method

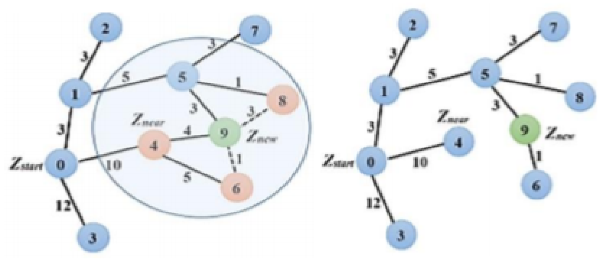
A. Based on RRT* Algorithm

This subsection introduces important path planning concepts related to RRT* in order to provide a better understanding of this study. It is essential to introduce the basic operations of RRT* prior to describe its variant approaches.
The process of selecting least cost parent and rewiring tree are two most promising features of RRT* and contribute to asymptotic optimal property of RRT*. Though best parent selection and rewiring of tree improve the path quality, however, these features have an efficiency trade-off with path quality

and make convergence slow as number of nodes in the tree increase. When z_g is found, a path connecting z_i and z_g is established. This path is improved as planner continues until a predefined number of iterations are executed or given time expires.



(a) Find near neighbors (b) Select best parent



(c) Check cost for rewiring (d) Rewired tree

B. Overview

Anytime RRT* is an extended version of the RRT* algorithm proposed by Sertac Karaman et al. There are two phases. An offline phase that runs the normal RRT* to obtain a feasible path (need not be optimal) in real time on a static environment. Once a path is obtained, the robot begins to follow the path in the control phase. During this control phase, two extensions namely Committed Trajectory and Branch-and-Bound technique is iterative implemented which asymptotically improves the computed path in terms of path length [7].

C. Committed Trajectory

A fraction of the initial trajectory is fixed while the rest is flexible (is subject to change) and the anytime RRT* algorithms creates more tree nodes which will be attached to the flexible fraction of the path. In this way the flexible fraction asymptotically becomes optimal. This planning occurs in real time

D. Branch-and-Bound Technique

The cost associated with the initial path is computed. Any node with its sub-tree with a cost higher than this cost is deleted. Any new nodes created in the committed trajectory with cost higher than the initial part is discarded. This technique improves computational efficiency.

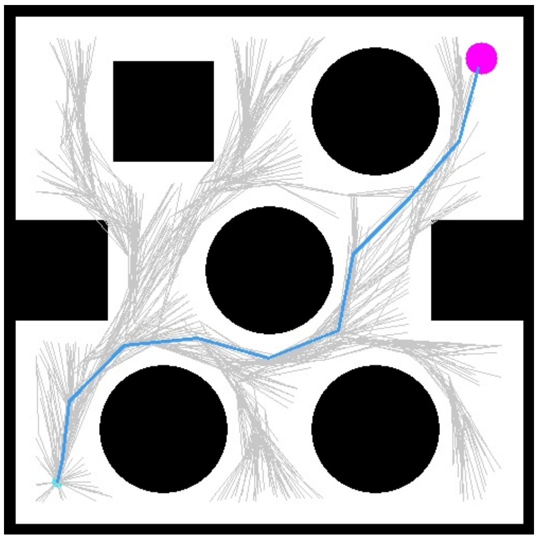


Fig. 1: Path created from random sampling

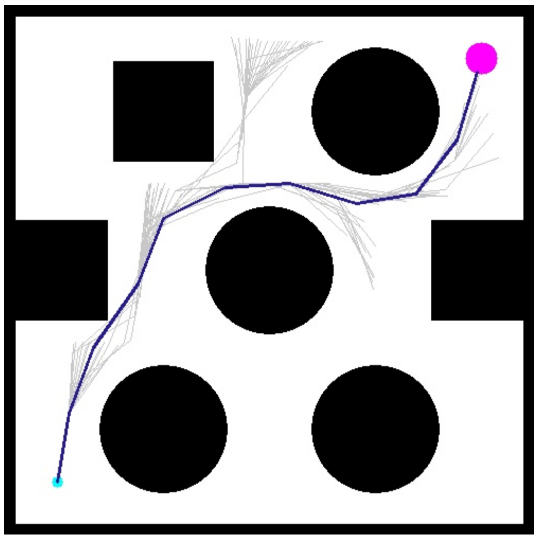


Fig. 2: Path created from sampling from normal distribution with mean centred at the goal node

First, the environment is being initialized which includes start, goal and obstacle space. Then, the online planning algorithm starts an initial planning phase in which RRT* runs until robot starts moving towards its goal. Here, the time is domain dependent.

Once the RRT* planning is completed it goes to iterative planning, in which optimal path is found by committed trajectory planning and branch and bound technique. If an optimal path is found for the robot, then the robot executes the path and if it does not, then it again runs RRT* from the initial phase.This loops continues until robot reaches to it's final goal.

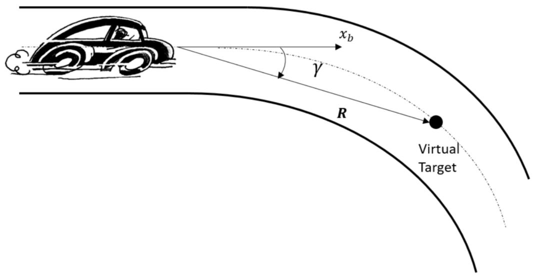


Fig. 3: Pure Pursuit Guidance Controller

III. Controller

The robot needs to follow the pre-defined path that is provided by the RRT* algorithm when traversing in a world and to avoid any drifting, we implement autonomous steering control. PID controller is employed to control the trajectory of the robot using the concept of cross track error.

A. Pure Pursuit Guidance Controller

A simple PID controller can be used to control a turtlebot. A constant linear velocity is set while the controller varies the value of the angular velocity. In this regard, the error function is the difference between the current yaw and direction towards to next way-point from the current position [8].

The equation is written as

$$\gamma = a\tan 2(\frac{y_g - y}{x_g - x}) \tag{1}$$

Here, γ is steering angle, x_g, y_g are goal coordinates and x, y are current robot coordinates.

B. Updating Way-points

To traverse a path from start point to goal point, the anytime RRT* does iterative planning and keeps on updating the way-points that is the temporary goal point for the robot. When the robot reaches a temporary goal point, the way-point needs to updated and for this, the distance between the initial way-point and the robot is used. As the distance is completely travelled, that means the percentage is greater than 1, the initial way-point is now the previous temporary goal point and the temporary goal point is now updated as the new way-point. The distance between the robot and initial way-point is calculated as

$$d = \frac{R_x \Delta x + R_y \Delta y}{\Delta x \Delta x + \Delta y \Delta y} \tag{2}$$

If $d > 1$, we update the way-points. Here, $R_x = x - x_1$, $R_y = y - y_1$, $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$, where x_1, x_2, y_1 and y_2 are way-points of initial position and final position respectively and x & y are current position coordinates of the robot.

IV. Results and Simulation

We are using ROS [9] Gazebo for simulation and Turtlebot 3 as our robot using Python-OpenCV [10] . Figure 4 is the world we are using for simulation and figure 5 is Turtlebot 3: Burger model that is our robot. We choose Turtlebot 3 because it supports both ROS Melodic as well as ROS Kinetic.

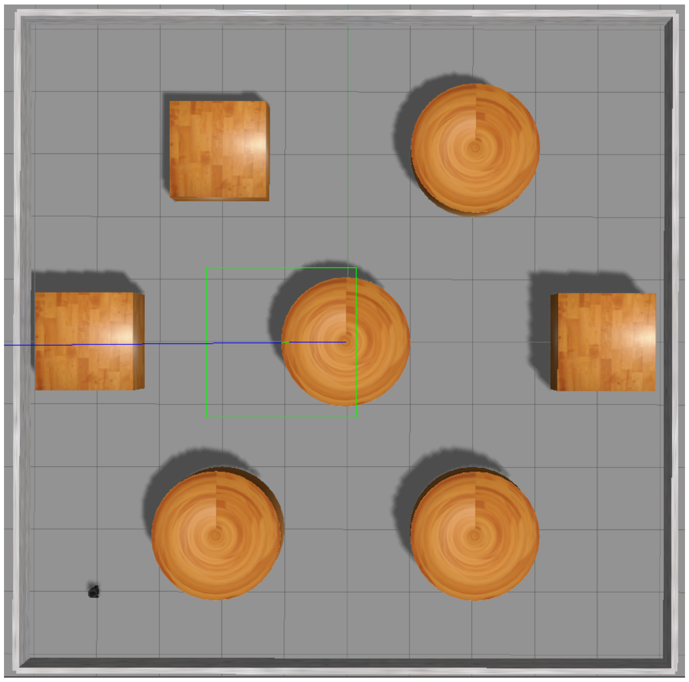
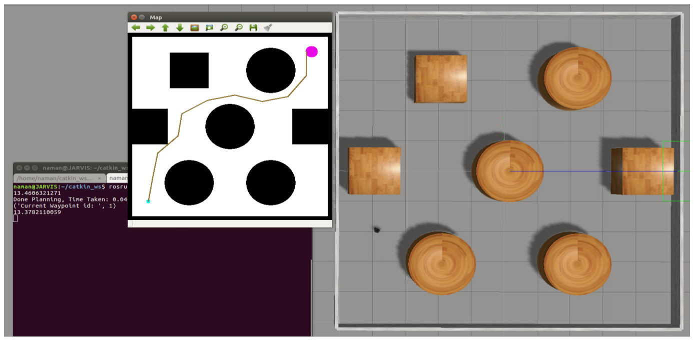


Fig. 4: Gazebo World

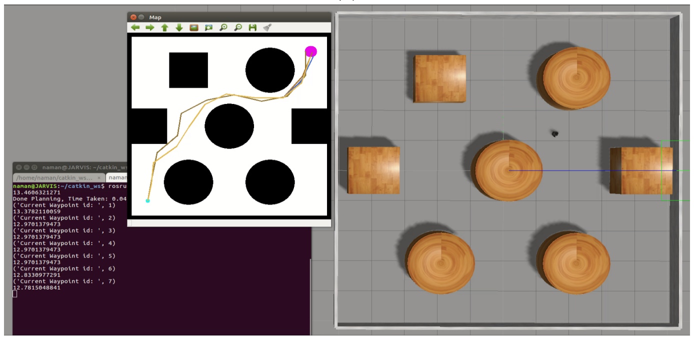
Fig. 5: Turtlebot 3: Burger

In figure 7, the simulation of the anytime algorithm is done. As the robot moves from start point to goal point, it initially follows the committed trajectory. After the robot receives way-points for committed trajectory, the robot follows the path and meanwhile, RRT* plans a new path keeping final way-point of the trajectory as a new start point. Figure 7(b) shows the newly generated path for the next committed



```
naman@JARVIS:~/catkin_ws$ rosru
13.4606321271
Done Planning, Time Taken: 0.04
('Current Waypoint id: ', 1)
13.3782110059
```

(a)



```
naman@JARVIS:~/catkin_ws$ rosru
13.4606321271
Done Planning, Time Taken: 0.04
('Current Waypoint id: ', 1)
13.3782110059
('Current Waypoint id: ', 2)
12.9701379473
('Current Waypoint id: ', 3)
12.9701379473
('Current Waypoint id: ', 4)
12.9701379473
('Current Waypoint id: ', 5)
12.9701379473
('Current Waypoint id: ', 6)
12.8330977291
('Current Waypoint id: ', 7)
12.7815048841
```

(b)

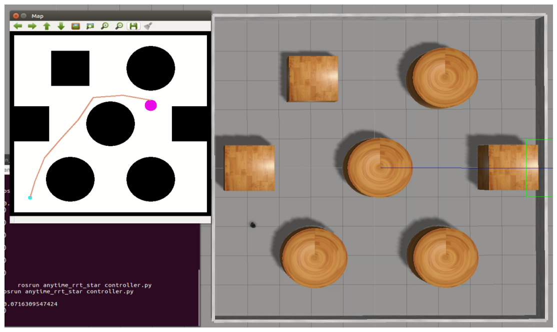
Fig. 6: Implementation of anytime algorithm. (a) shows the initial path generated by the algorithm and robot moving towards the committed trajectory. (b) shows the path rerouted as the robot completes the committed trajectory.

trajectory and the robot follows the new path.

Figure 8 describes the reduction of cost as the way-points are increased. The cost reduces from 13.86 meters to 12.77 meters in 11 way-points. The planning time also varies during every run, in this result the initial RRT* path is planned in 0.3179 seconds.The red curve shows the path length/cost from start point to goal point and blue curve shows the actual path cost reducing approximately by 1 meter.

V. Conclusion

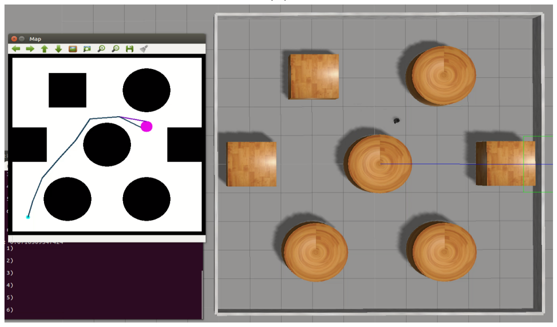
The paper is successfully implemented in simulation and apart from this, we observed a variety of path generation in RRT* using random function. In figure-1 and 2 when we compare both the paths generated, the number of nodes explored using a normal distribution random function is less and has planning time on an average of 0.04 seconds as compared to random function which has planning time of 0.23 seconds. However, the normal distribution random function can create a local minima that can cause the algorithm to fall into an infinite loop. The random distribution function can create trees that have more branches which reach a larger portion of the free space and hence will not create spurious local minima. The anytime algorithm is designed for real world applications that reduces time for planning and allocates more time for task completion. The online planner (committed trajectory and branch and bound technique) it runs parallel to the controller hence, it is trivial that it is a real time planner. In addition, the offline planner finishes planning on an average of 0.1 seconds, therefore, the overall algorithm that we implemented is a real time path planner.



```
 rosrun anytime_rrt_star controller.py
osrun anytime_rrt_star controller.py

.0716309547424
```

(a)



(b)

Fig. 7: Implementation of anytime algorithm for another run.

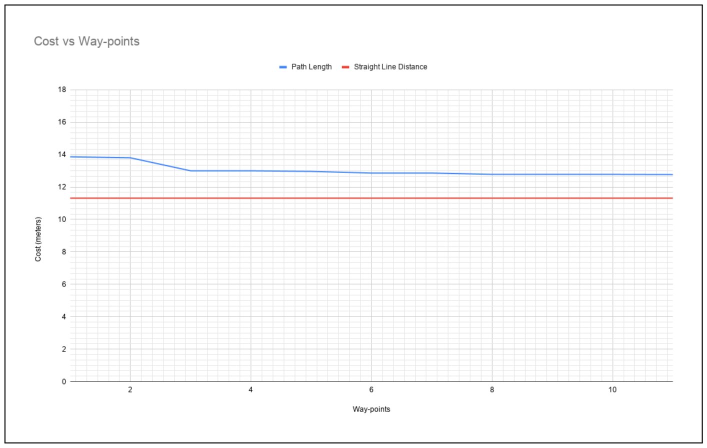


Fig. 8: Cost vs Way-points: As the robot moves along the committed trajectory, it concurrently finds paths with better cost. As a result, while the robots clears more way-points, the path is updated to a better path. Hence we see a total reduction in the total path length

References

[1] M. B. K. A. Wolfram Burgard, Cyrill Stachniss, “Robot motion planning,” *UNI FREIBURG*, 2011.

[2] A. Yershova and S. M. LaValle, “Improving motion-planning algorithms by efficient nearest-neighbor searching,” *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 151–157, 2007.

[3] B. Roy, “A-star search algorithm,” *Towards Data Science*, 2019. https://towardsdatascience.com/a-star-a-search-algorithm-eb495fb156bb.

[4] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[5] S. Karaman and E. Frazzoli, “Incremental sampling-based algorithms for optimal motion planning,” *Robotics Science and Systems VI*, vol. 104, no. 2, 2010.

[6] I. Noreen, A. Khan, and Z. Habib, “A comparison of rrt, rrt* and rrt*-smart path planning algorithms,” *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 16, no. 10, p. 20, 2016.

[7] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, “Anytime motion planning using the rrt,” in *2011 IEEE International Conference on Robotics and Automation*, pp. 1478–1483, IEEE, 2011.

[8] *Pure Pursuit Guidance for Car-Like Ground Vehicle Trajectory Tracking*, vol. Volume 2: Mechatronics; Estimation and Identification; Uncertain Systems and Robustness; Path Planning and Motion Control; Tracking Control Systems; Multi-Agent and Networked Systems; Manufacturing; Intelligent Transportation and Vehicles; Sensors and Actuators; Diagnostics and Detection; Unmanned, Ground and Surface Robotics; Motion and Vibration Control Applications of *Dynamic Systems and Control Conference*, 10 2017. V002T21A015.

[9] Stanford Artificial Intelligence Laboratory et al., “Robotic operating system.”

[10] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*.