Autonomous Robots Lab Report 2
Rapidly Exploring Random Tree

Raabid Hussain

April 10, 2016

1 Objective

The objective of this lab work was to implement the rapidly exploring random tree (RRT) path planning algorithm on MATLAB. We provided with a discretized map of the environment. Given the start position and the goal position, we had to create a tree using the RRT algorithm. This RRT tree was then was to compute a path from the start position to the goal position which later was smoothed to decrease the number of way-points or rotation movements for the robot.

2 Introduction

Motion planning is one of the key notions in autonomous robotics. It involves finding a path between the current position of the robot and the goal position, satisfying all the physical constraints of the robot and the environment. There have been different approaches to path planning algorithms which can be characterized into behavior based algorithms, grid-based search, interval based search, geometric algorithms, potential fields and sampling based algorithms. In most of these algorithms, the map of the environment has to be previously known.

This lab work constituted of implementing one of the sampling based techniques known as a rapidly exploring random tree (RRT). RRT is specifically designed to efficiently obtain a path in non-convex and high dimensional

environments. It operates by building a space-filling tree structure inside the environment. Randomly drawn samples are used to construct the tree which has a bias to growing in the direction of a given point in space. It has also been found to grow efficiently towards unsearched areas in the environment.

The RRT algorithm starts by taking into account the map of the environment, the starting position and the goal position for the robot. It then generates a random point in the empty spaces of the environment. The nearest state in the tree from the randomly generated point is found and an attempt is made to create a feasible connection between the two points. The feasibility condition is mainly comprised of the fact the connection should not pass through any obstacle in space. If the connection is deemed feasible it is added to the tree. The length of the connection is normally limited by a factor. If the randomly generated point is farther than this factor than the length of the connection is governed by this factor, otherwise the actual distance between the points is used.

In addition a bias is added when generating the random point in space. This bias replaces the random point by the position of the goal occasionally. This leads to a faster convergence to the solution. The algorithm stops when a successful connection between the start ad goal position has been made. In other words, when the goal position is one of the nodes in the tree. Since the sample is generated through a random distribution, the solution to a given problem is always different.

3 Implementation of Rapidly exploring Random Tree Algorithm

MATLAB was used to implement the RRT based path planning structure. A discretized map of the environment was provided in form of a sparse matrix. The matrix contained a 0 value to represent the free spaces and a 1 value to represent areas occupied by obstacles. In addition the start and goal position were provided too. A loop was initialized that limited the algorithm to a specific number of nodes or iterations. This was done as the algorithm does not check if there is indeed a path available or not. So, the maximum number of nodes in the tree was constrained to a reasonable number like 1000.

3.1 Generating the Tree

A matrix called 'vertices' was initialized to the start position. This matrix was used to store all the nodes of the tree. The first thing was to generate a biased random point in the free space of the environment. This was achieved by generating a random number with uniform distribution between 0 and 1. If the generated number was below the bias percentage then the sample was given the coordinates of the goal position. Otherwise, two random numbers between 0 and 1 were generated and multiplied by the size of the map to get x and y coordinates of the sample point respectively. The generated sample point was compared with the map of the environment provided to check if the point was indeed in a free space. If it was not then a new point was generated until the point was in the free space.

Once the random sample is generated, the euclidean distance between the sample and all the nodes in the tree is computed and the node with the shortest distance is termed as the nearest node. The tree will be extended from this node. If the distance between this node and the sample is less than the threshold distance then the sample point itself is taken into account. In case this is not true then, the line connecting the two points is sectioned with a length equivalent to the threshold and that point taken into account.

Next, we had to check if the path between the two points is visible or not. For this, points spaced at 3 pixels were generated between the two points. Ideally, the gap was chosen to be 1, however, this increased the runtime so in the end a gap size of 3 was seen to be reasonable. All the generated points were checked to see if there was an obstacle at their respective location on the map. If all the points were in free space then the point was added to the tree, otherwise a new sample point was generated again and the process repeated. The links or branches of the tree were stored in a matrix which contained the start node and the end node indexes for each branch. This matrix is called 'edges' in the code.

The above process is repeated all over until the new node in the tree is the goal position. At this position in time, the main loop is stopped. All the nodes in the tree are drawn on the map along with their respective

connections. A sample output of this stage is shown in figure 1.

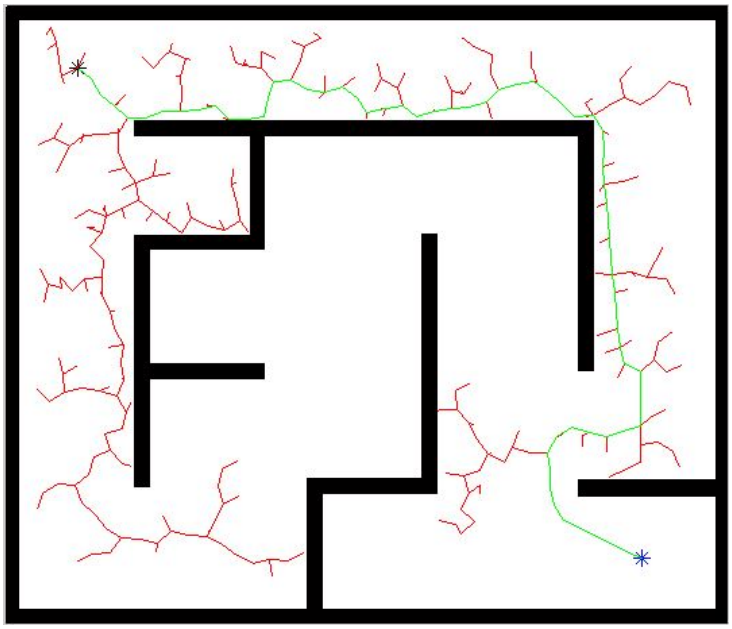

Figure 1: The tree generated by the RRT algorithm for a sample maze shaped environment. The red lines represent the tree, whereas the green lines represents the path found between the start and the goal position

The red line represent the tree whereas the green line represents the path found. The path line is found by starting from the last point in the edges matrix. The code traverse backwards by matching the starting node of the branch with the ending nodes of all the other branches. If a match is found the particular node is added to the path list. This repeated until the path matrix has the first node as the latest introduction into the path list.

3.2 Smoothing the generated Path

The path generated by the RRT algorithm is often too rough i.e. there are too many nodes. This in practical scenarios is seen to be highly repulsive as more the number of turns the robot has to make, the more the error and runtime introduced. So, a smoothing function was introduced. It takes the map, nodes in the tree, the generated path and a factor as inputs.

The function starts by creating a line between the last node and the first node in the tree. It then generates sample points on the line, each spaced by the factor distance input to the function. If all the sample points were found to be in free space then the branch was added to the 'path_smooth' list. In case any of the points was found to be on the obstacles then the connection with the next point was checked with the last point. This was repeated until a free connection was found in which case the connection was added to the smooth path list. Then all the points before the chosen points were checked as mentioned above for a free connection. This was repeated until the start point was added to the smooth path list.

A sample result of the smoothed path is shown in figure 2, where the red line represents the path generated by the RRT algorithm and green lines represent the smoothed path.

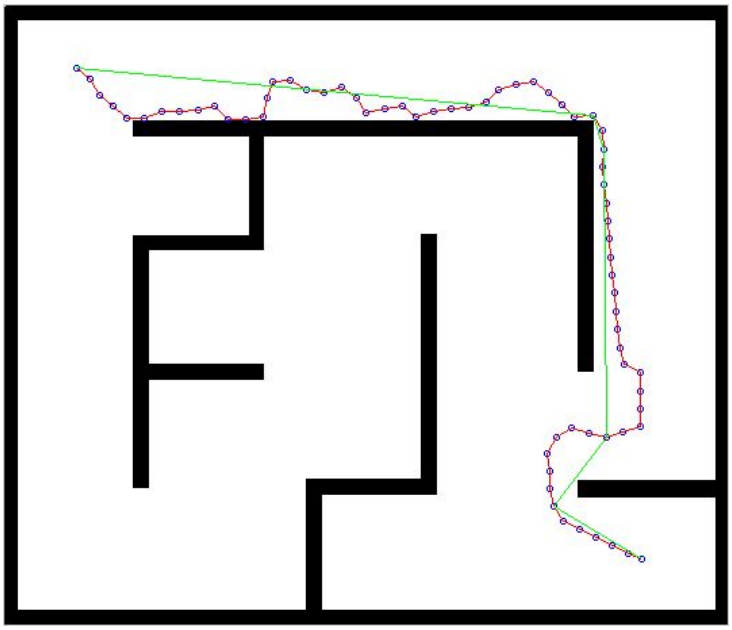

Figure 2: The result of the smooth function. The red line represents the path generated by the RRT algorithm, the blue circles represent the nodes in the path/tree and the green line represents the smoothed path.

3.3 Sample Results for Non-Concave Objects

The main advantage of the RRT algorithm is that it can deal with non-convex obstacles too. So, a sample environment was tested as shown in figure 3. Here the red lines represent the tree branches, the green line represents the path generated by the RRT algorithm and the blue line represents the smoothed path.

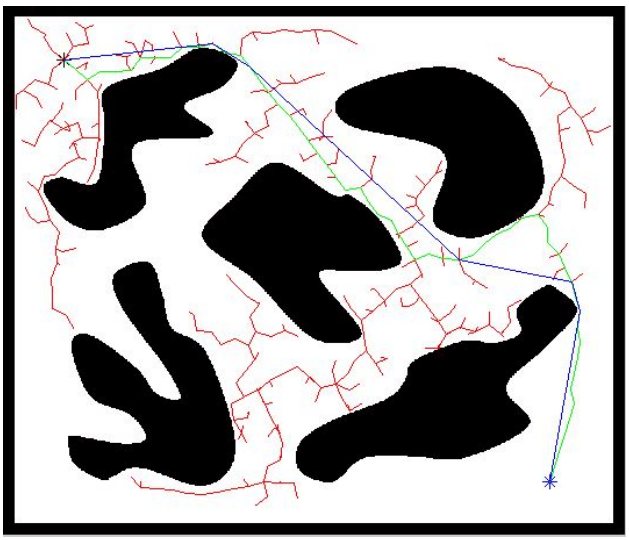

Figure 3: The tree generated by the RRT algorithm for a sample map with circular obstacles. The red lines represent the tree, the green line represents the path generated by the RRT algorithm, whereas the blue line represents the smoothed path.

4 Conclusion

In this lab assignment, a sample based path planning technique called the RRT algorithm was implemented on MATLAB. A tree was generated in the free space between the start and the goal positions. This was followed by finding the path between the start and goal positions using this tree. Since, the generated path had too many turns for the robot, the path was also smoothed at the end. The algorithm was tested on various environment. The parameters should be chosen carefully to reach a solution in appropriate time. All the tests conducted were successful.