UNIVERSITAT DE GIRONA

AUTONOMOUS ROBOTS

LABORATORY REPORT

Rapidly Exploring Random Trees

Author
Mohit VAISHNAV

Supervisor
Dr. Perez Marc CARRERAS
Eduard Vidal GARCIA

Submission Date 25, March

1 Introduction

RRT [1] has been made to efficiently search *nonconvex* spaces building space filled tree. It is biased to grow towards unsearched areas of the problem using a tree made from samples drawn randomly.It is a technique to generate open loop trajectories for non linear system. RRT is considered as *Monte Carlo* method to search into larges *Voroni regions* of a graph. It is suited for path planning problems involving differential contrains and obstacles. RRT alone is insufficient to solve planning problems but acts as a important component to be incorporated in different planning algorithms.

2 Algorithm

Pseudo code of the algorithm is shown below [2]. *RRT* rooted at a configuration space (C) q_{init} having K vertices are constructed as follows: Choose a random configuration q_{rand} in C after which a vertex x_{near} is selected closest to q_{rand}. Next, new configuration q_{new} is selected by moving incremental distance Δ q in the direction of q_{rand} (motion in any direction is possible). Where differential constraints exists, inputs are applied to corresponding control system and new configuration is obtained by numerical integration. Finally, a new vertex, q_{new} is added and a new edge is added from q_{near} to q_{new}

```
Algorithm BuildRRT
  Input: Initial configuration q_init, number of vertices in RRT K, incremental distance Δq)
  Output: RRT graph G

  G.init(q_init)
  for k = 1 to K
    q_rand ← RAND_CONF()
    q_near ← NEAREST_VERTEX(q_rand, G)
    q_new ← NEW_CONF(q_near, q_rand, Δq)
    G.add_vertex(q_new)
    G.add_edge(q_near, q_new)
  return G
```

3 Implementation

We were given initially the *map* on which it has to be tested and presented the results in which 0 represents the free spaces and 1 represents obstacles.

Additionally start and the goal positions were also provided. To put a limit on number of iteration possible it was limited to 1000 in count.

3.1 To generate a tree

Matrix called *vertices* was used to store the nodes of a tree. Using a random function which generate a value between 0 and 1, a point is generated in a free space. If it turns out to be less than bias percentage p then the samples takes the value of *goal* position otherwise it is converted into a location inside the map which could either be in free space or obstacle. So this process is repeated until it lied in the free space.

Once there is a point, euclidean distance is computed between sample and all the nodes to get the shortest ones from it and tree is extended from here. If this distance is found to be less than *delta_q* then same point is taken into account otherwise the line connecting two points with length equal to that of *threshold*. After getting the path we check for the visibility of it for which three points are generated in the vicinity around the point (ideally should have been 1). If there were no obstacle in that vicinity, that point is declared as free otherwise the whole process is repeated till a free points is generated. A variable *edges* has the information like start node, end node, branches of a tree in a sorted order.

We repeat the same process till the new node (q_{new}) is equal to goal node (q_{goal}). In the output figure, path in *green* colour represents the destined path between starting and goal position whereas the three structure is represented by *red* color.

3.2 Smoothing the path

Because of the random process , path generated between nodes are very distributed and it has to be made smooth to decrease the number of turns and make it more conducive path to traverse.

Function starts with forming a line between first and last node in a tree and generates sample around the line. It they all are found to be in a *free* space then this branch is added to the *path_smooth* list. If it encounters any obstacle, connection with the last point and next point is checked. This process is repeated till the time we get free connection and hence a *smooth path*.

4 Output

Fig 1 shows the two type of obstacle maps used for our results purpose. After implementing the algorithms we obtained the result which are displayed in the form of Figure 2, 3. To show the behaviour of random function in *MATLab* where it forms the different path as we can see in the above mentioned figures.

(a) Map 1 (b) Map 2

Figure 1: Obstacles Map

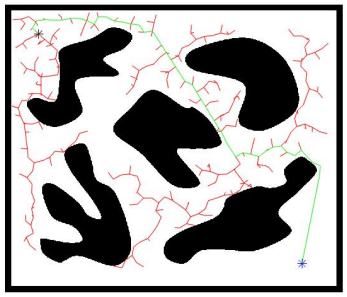
5 Conclusion

Although construction method is simple there is no easy task to find the method which yields desirable behaviour. For eg, a tree is generated by incrementally selecting a vertex at random and input at random, then applying the input to generate a new vertex. It also suffers from bias towards places already visited.

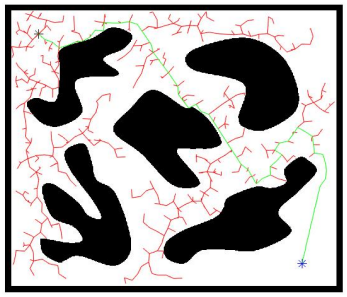
References

[1] UIUC. About rrt. http://msl.cs.uiuc.edu/rrt/about.html.

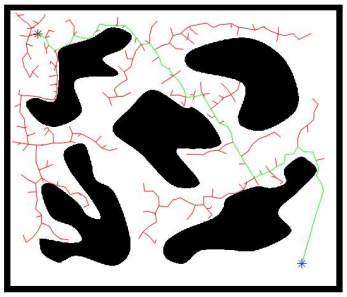
[2] Wiki. Algorithm. https://en.wikipedia.org/wiki/Rapidly-exploring_random_tree. Accessed March 20, 2018.



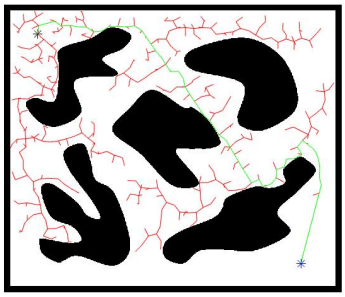
(a) Map 1: Trial 1



(b) Map 1: Trial 2

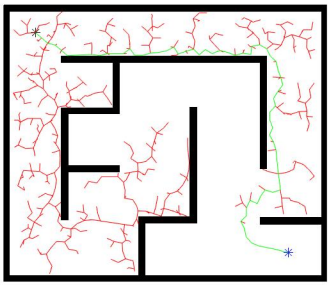


(c) Map 1: Trial 3

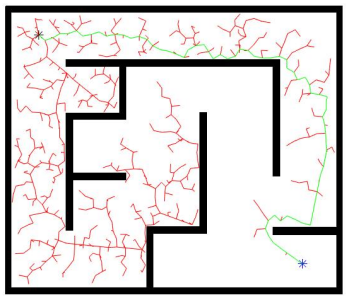


(d) Map 1: Trial 4

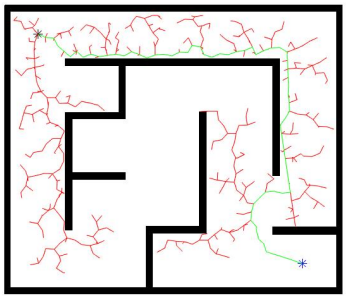
Figure 2: Map 1 with different trials and different path



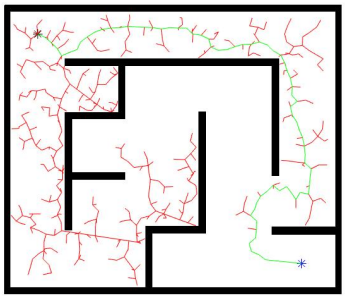
(a) Map 2: Trial 1



(b) Map 2: Trial 2



(c) Map 2: Trial 3



(d) Map 2: Trial 4

Figure 3: Map 2 with different trials and different path