Multiple Mobile Robots Motion Planning with RRT

Zhiang Chen
Case Western Reserve University
Cleveland, Ohio
zsxc251@case.edu

***Abstract*—Rapidly-exploring random trees (RRT) is suitable to find a single-query solution in high-dimensional space. This project utilizes RRT to do motion planning for multiple mobile robots. Three basic concepts are introduced to adapt RRT for high-dimensional space. Two sampling strategies are presented and compared. Probabilistic bias sampling performs well for simple map, while uniform sampling is more suitable to complex map. RRT does well when implementing in simple map. As the map's complexity increases, the number of the robots significantly affects the convergence time. And RRT performers worse with large configuration space and complex map.**

Keywords—RRT, Multiple Robots, Motion Planning.

I. INTRODUCTION

Some of the most significant challenges confronting autonomous robotics lie in the area of automatic motion planning. Simple planners like Bug and Bug-like algorithms [1] are straightforward to implement but lose generalization to apply in high-dimensional configuration space. One of hardest challenges for robots with high-dimensional[1] configuration space are representing obstacles and boundaries in configuration space. Algorithms such as navigation potential functions [2], generalized Voronoi diagram (GVD) confront this same difficulty.

Sampling-based planners employ a variety of strategies for generating samples and for connecting the samples with paths to obtain solutions to path-planning problems [3]. Probabilistic RoadMap planner (PRM) [4] exploits that it is cheap to check if a single robot configuration is valid or not in configurations space. PRM as a multiple-query planner creates a roadmap that captures the connectivity of collision-free configuration space (Q_{free}) and then answers multiple user-defined queries very fast. In many planning instances, the answer to a single-query is of interest and single-query planners attempt to solve a query as fast as possible and do not focus on the exploration of the entire Q_{free} . RRT as a single-query planning algorithm that efficiently covers the space between the initial configuration (q_{init}) and goal configuration (q_{goal}).

[1] High-dimensional space specifies the space with more than two-dimension.
[2] The data structure and algorithm in this report use C plus plus pseudocode.
[3] When configuration space is not specified with high-dimension, it can be high-dimensional or two-dimensional space.

When multiple robots working in a space, they should have their own valid paths in Q_{free} and avoid crashing others. Two solutions come based on whether the communication among the robots is required when they are on the way of other's path. First, motion planners independently find a valid path for each single robot, and a predefined protocol is required to deal with the cases when the robots are blocked. The advantage of this solution is the motion planners only need to deal with two- or three-dimensional configuration. The shortage is that an additional protocol is required and it is ad hoc.

The other solution is that the motion planner should consider the collision among robots when finding valid path. In this scenario, the dimensionaliy of manifolds increases proportionally to the number of the robots. In this project, I implemented RRT to give a generalized solution to this problem.

II. TREE CONSTRUCTION

The basic concepts (or data structures) and their validation method are firstly introduced. Then I will present how to build trees and extend trees.

A. Basic Concepts[2]

1) Vertex

Assuming the mobile robots have two-DOF. And the pose of the robot is represented by position (x,y). The reasons why the orientation is ignored are that considering the orientation increases the dimensionality of the manifolds, and it is unnecessary to care about the orientation of the mobile robot except when there exists narrow passage in the map. Vertex represents one point in configuration space[3]. It consists the positions of all robots. Table 1 shows the structure of Vertex.

```
Vertex{
  number; // the number of robots
  parent_index; // its parent's index
  positions; // the positions of all robots
}
```

Table 1

When considering n mobile robots, the dimensionality of the Vertex is $2n$.

2) Edge

Edge consists two Vertex, one start Vertex and one end Vertex:

```
Edge{
   Vertex start;
   Vertex end;
}
```

Table 2

3) Tree

Tree consists of a vector of Vertex:

```
Tree{
   vector<Vertex> vertexes;
}
```

Table 3

It's unnecessary to add Edge into Tree since each Vertex in Tree contains the index (in the vector) of its parent. These indices can replace the Edge to describe the Tree.

B. Checking Collision

As mentioned in introduction, the motion planner need to check two types of collision, the collision from the obstacles in the map and the one from other robots (self-collision when considering the robots as one system). Since the pose of the robot is represented by the its position, the shape of differential-steer and carlike mobile robot can be simplified as a circle; the shape of omnidirectional robot is fixed. In this project circular shape is considered and the its center represents the position of the robot. Sampled Vertex from configuration space first needs to be converted into workspace, and then be checked whether it is collided with the obstacles in the map. That means every position in the Vertex is required to be checked. In practice, the map consists of binary state cells, occupied and unoccupied cells. The resolution of the map depends on the size of the cell. The higher resolution map, the smaller cell, increases the computational complexity. And the size of the map also affects the convergence rate, which will be presented later.

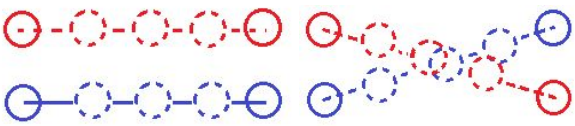

Figure 1

Figure 1. The Edge (consisting of two paths) on the left picture is self-collision free. The Edge on the right is in self-collision.

Edge consisting of start Vertex and end Vertex can also represents the robots' motion from time t_s to t_e, which means all robots move from the positions in start Vertex at t_s to the positions in end Vertex at t_e. The end Vertex is the parent of the start Vertex from next Edge until the trees are merged. If Edge is considered as path $[0,1] \to P$, the self-collision checking is to check the collision among the robots along the Edge in every step_size in figure 1. The step_size depends on the size of the robot.

Build RRT Algorithm

Input:

V : the Vertex where tree is rooted

n : the number of attempts to expand the tree

Output:

T : A tree that is rooted at V_{init} and has $\le n+1$ Vertex

```
1. V.parent = -1
2. vertexes.push_back(V)
3. for i = 1 to n do
4.    V_rand = randomly_generate_from_Q_free()
5.    extend(T, V_rand)
6. end for
7. Return T
```

Table 4

Extend RRT Algorithm

Input:

T : the tree needed to be extended

V : the Vertex attempting to add to the tree

Output:

T_{new} : A tree that is rooted at V_{init} and has $\le n+1$ Vertex

```
1. V_near = find_closeset_Vertex(T)
2. Edge.start = V_near
3. Edge.end = V
4. If(Edge_is_collision_free)
5.    V.parent = getIndex(V_near)
6.    T.vertexes.push_back(V)
7.    T_new = T
8.    Return T_new
9. else
10.   V_new = process_V_near_by_step_
      size_along_Edge_between_V_near_and_V(V_near,V)
11.   If(Edge_V_new_V == collision_free)
12.      T.vertexes.push_back(V_new)
13.      T_new = T
14.      Return T_new
15.   else
16.      return T
17.   end if
16. end if
```

Table 5

C. Building Trees

Two Tree, T_{init} and T_{goal} are rooted at two Vertex, V_{init} and V_{goal}. When building tree, n sampled Vertex attempt to expand the tree in table 4.

D. *Extending Trees*

If randomly sampled V_{rand} is validated, it attempts to add to the tree in table 5. In practice, the pointer to the tree as input is preferable. The tree would try to connect the random Vertex first, if failed then it would attempt to connect one Vertex along the Edge.

E. *Merging Trees*

After each valid extension, the newly added Vertex attempts to connect the closest Vertex in the other.

F. *Getting Path*

Since each Vertex in the tree contains the index of its parent, and every Vertex except V_{init} and V_{goal} has and only has one parent. If V_{node} is the Vertex connecting two trees, one path can be obtained by searching the parent of V_{node}, parent of the parent, and keeping searching to meet the root in one tree. Two paths from two tress then can be integrated into one path.

III. Sampling Vertex

Currently, there exist two sampling strategies. One is to select a Vertex uniformly at random from configuration space, the other is to build a sampling function that alternates between uniform samples and samples biased toward regions that contain the initial or goal Vertex.

According to the results from my project, when the map is simple and contains no narrow passage, the latter one is more efficient than the former. However, when the map is relatively complex, they have almost the same performance. And when the map is really complex, the former one even converges a little faster than the latter. This can be easily explained that adding the bias makes the algorithm greedy to connect the tree to the other tree's root, while uniform sampling tends to exploring the Q_{free}. Thus the one with probabilistic bias is faster when the map is simple. As the map becomes complex, the number of local minimum increases. The greedy algorithm is more likely to extend the tree toward local minima and thus spend much time on recovering from local minima.

IV. Results

RRT as heuristic algorithm trades off completeness for practical efficiency. The rate of convergence is opportunistic but we can still have a glance of the influence of some factors from the results of the project. The initial positions were swapped with the goal positions recursively. For example, robot0's initial position is set as robot1's goal, and robot1's initial position is set as robot2's goal. The paths were not postprocessed here.

A. *Simple Map*

When implementing on simple map, the number of the robots has little influence on the rate of convergence in figure 2. And the algorithm can be converged very fast.

B. *Complex Map*

In this scenario, as the number of the robots increases, the convergence rate decreases rapidly in figure 3. RRT is still able to deal with 5 robots though the convergence time is more than 300 seconds.

C. *Large Configuration Space*

The map is largely expanded in figure 4. RRT is able to do 3 and sometimes even 4 robots, but it took more than two to get converged for 5 robots.

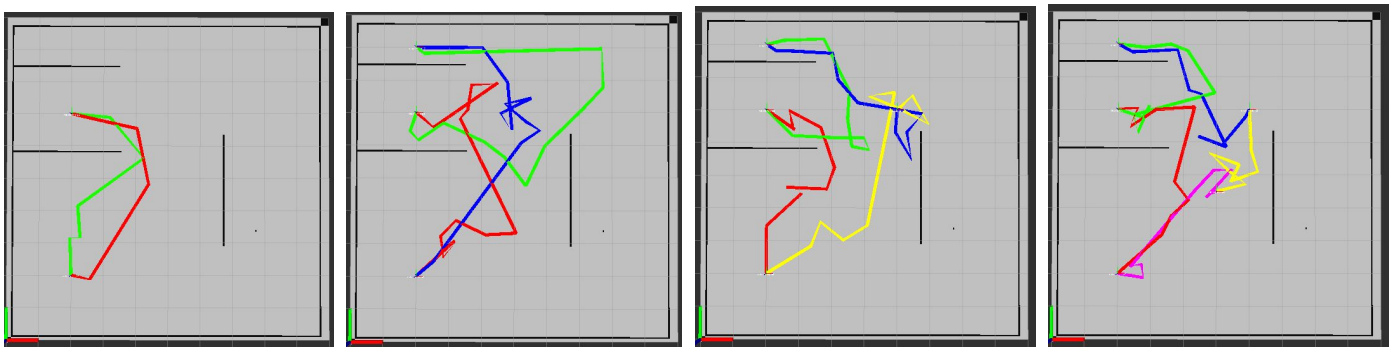

Figure 2

Figure2. From left to right, the number of the robots varies from 2 to 5. And the convergence time are 0.0050, 0.0305, 0.0309, 0.0350 seconds.

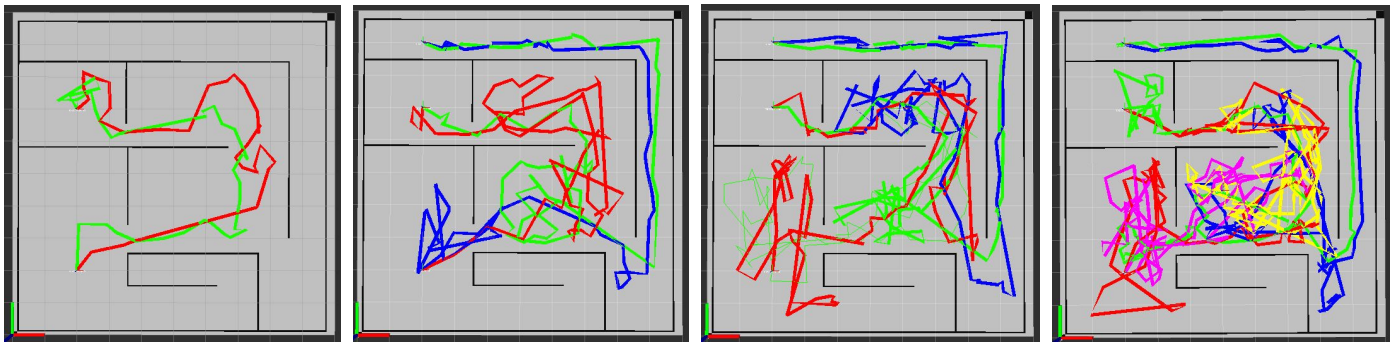

Figure 3

Figure 3. From left to right, the number of the robots varies from 2 to 5. And the convergence time are 0.2115, 2.6954, 23.944, 326.27 seconds.

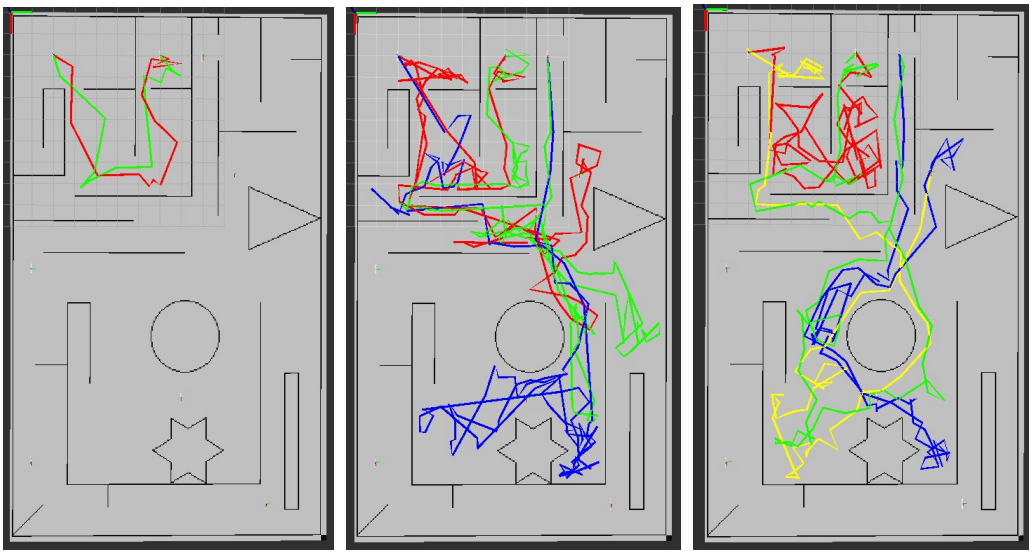

Figure 4

Figure 4. From left to right, the number of the robots varies from 2 to 4. And the convergence time are 0.8329, 134.97, 62.29 seconds. This map is expanded based on the map in figure 3. They have almost the same complexity, but RRT did better with smaller configuration space.

ACKNOWLEDGMENT

I thank M. Cenk Cavusoglu for his insight and expertise that greatly assisted the project.

REFERENCES

[1] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica, 2:403-430, 1987.

[2] D. E. Koditschek and E. Rimon. Robot navigation functions on manifolds with boundary. Advances in Applied Mathematics, 11:412-422, 1990.

[3] H. Choset, K. Lynch, S. Hutchinson, "Principle of Robot Motion: Theory, Algorithms and Implementations", MIT Press, 2005.

[4] L.K. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration space. IEEE Transactions on Robotics and Automation, 12(4):566-580, June 1996.

Appendix:
Github repository: https://github.com/ZhiangChen/multi_motion_planning