MAE 5810: Robot Perception
Final technical Report

RRT* PATH PLANNING OF PINHOLE CAMERA

December 10th, 2021

Haiyang Yu
Department of Electrical & Computer Engineering
College of Engineering
Cornell University
hy397@cornell.edu

1 Abstract

In this report, I will first introduce the task to be solved in my final project in 2.1. After that, I will introduce the RRT* algorithm and its core steps in 2.2. The results of the planning problem is shown in 2.3 and my slides. I analyse the RRT* algorithm by talking about its advantages in 2.4.1-2.4.2 and its shortages in 2.4.3-2.4.4, follows some brief proof.

2 Path Planning

2.1 problem description

In this project, The task is described as path planning for a CCTV in a museum. A laser light is attached with the PT camera. There are some expensive artworks on the bottom and we donât want to directly light up them. As a result, the path of the center of FOV should avoid these areas. As described in Figure 1, the artworks are located in the red areas. And the target of this task is to move from (0,0) to (3,3) and the units in the graph are centimeters.

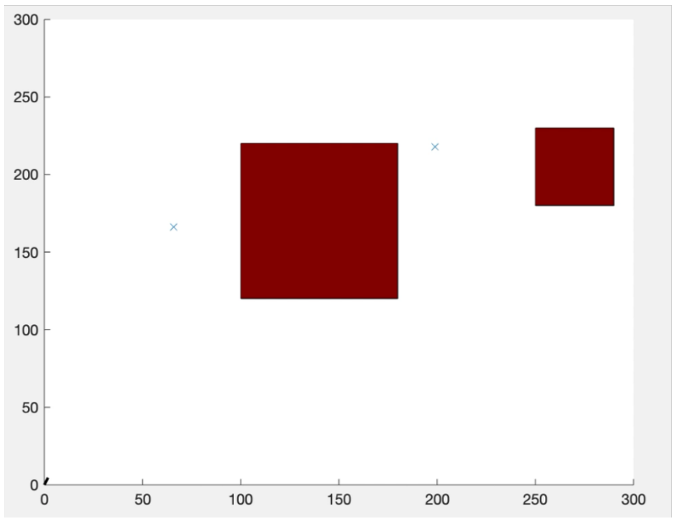


Figure 1: Planning scenario

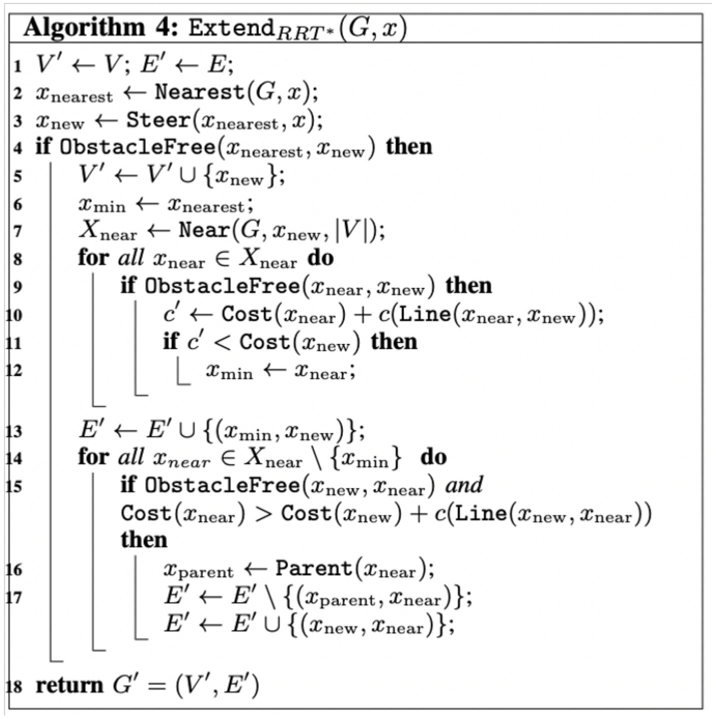
2.2 RRT* algorithm description

To solve the planning task, we implement Rapidly-Exploring Random Trees Star(RRT*) in our project. The planning steps of RRT* can be described as follows:

- Get a random point x_{rand}.

- Find the nearest point $x_{nearest}$ to x_{rand} in the tree.
- Connect $x_{nearest}$ and x_{rand}.
- Search the nodes in the tree in the circle with x_{rand} as the center and r_i as the radius.
- Find all potential parent points set $X_{potential_parents}$ to see if there is a better parent point for x_{rand}.
- Connect every points in $X_{potential_parents}$ and x_{rand}, then compute the cost of every potential path. If the cost is less than the previous path and the path between the points are collision-free, assign the parent point to the new parent point.
- If path changed, delete the previous edge in the tree and add the new edge.
- Repeat the process until meet the destination or exceed the largest iteration.

The core idea of the RRT* algorithm follow the pseudocode shown in Figure 2



```
Algorithm 4: Extend_{RRT*}(G, x)
$V' \leftarrow V$; $E' \leftarrow E$;
$x_{\text{nearest}} \leftarrow$ Nearest$(G, x)$;
$x_{\text{new}} \leftarrow$ Steer$(x_{\text{nearest}}, x)$;
if ObstacleFree$(x_{\text{nearest}}, x_{\text{new}})$ then
    $V' \leftarrow V' \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
    $X_{\text{near}} \leftarrow$ Near$(G, x_{\text{new}}, |V|)$;
    for all $x_{\text{near}} \in X_{\text{near}}$ do
        if ObstacleFree$(x_{\text{near}}, x_{\text{new}})$ then
            $c' \leftarrow$ Cost$(x_{\text{near}}) + c($Line$(x_{\text{near}}, x_{\text{new}}))$;
            if $c' <$ Cost$(x_{\text{new}})$ then
                $x_{\text{min}} \leftarrow x_{\text{near}}$;
    $E' \leftarrow E' \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    for all $x_{near} \in X_{\text{near}} \setminus \{x_{\text{min}}\}$ do
        if ObstacleFree$(x_{\text{new}}, x_{\text{near}})$ and
        Cost$(x_{\text{near}}) >$ Cost$(x_{\text{new}}) + c($Line$(x_{\text{new}}, x_{\text{near}}))$
        then
            $x_{\text{parent}} \leftarrow$ Parent$(x_{\text{near}})$;
            $E' \leftarrow E' \setminus \{(x_{\text{parent}}, x_{\text{near}})\}$;
            $E' \leftarrow E' \cup \{(x_{\text{new}}, x_{\text{near}})\}$;
return $G' = (V', E')$
```

Figure 2: RRT* pseudo code[1]

2.3 Planning result

The optimal path planned by RRT* is illustrated as the red line in Figure 3. The final result is obstacle-free. And the planning process also tries to find the path with shortest distance. The detailed planning process and tracking process are included in my slides.

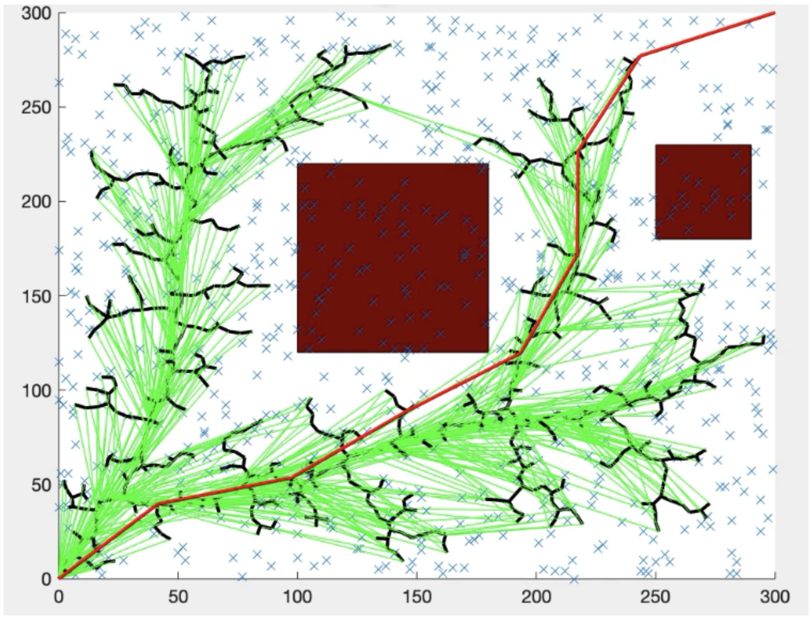


Figure 3: RRT* planning result

2.4 Analysis of RRT* algorithm

In this part, I will introduce two advantages and two disadvantages of RRT* algorithm and do some brief proof based on the work of previous scientists. Since RRT* is derivated from rapidly exploring random tree(RRT) and Rapidly-exploring Random Graph(RRG), the proof will be based on some property of RRT and RRG [2].

2.4.1 Advantage1: coverage

As for RRT algorithm and RRG algorithm, there are theorems that shows the RRT and RRG algorithms are able to produce a solution to the feasibility problem as the number of samples increase. Besides, there are theorems proved shows that if there exists a feasible collide-free path, assume the sample space is ω

$$\lim_{i \to \inf} P(V_i^{RRT} \cap X_{goal} \neq \emptyset) = 1 \tag{1}$$

$$\lim_{i \to \inf} P(V_i^{RRG} \cap X_{goal} \neq \emptyset) = 1 \tag{2}$$

Where V_i^{RRG} is the set of vertices of RRG, V_i^{RRT} is the set of vertices of RRT. That guarantee the reachable of destination after infinite iterations. In other words, the RRT and RRG algorithm are probabilistically complete.

As for RRT* algorithm, according to the steps described above, there holds

$$V_i^{RRT} = V_i^{RRT*} \tag{3}$$

$$\epsilon_i^{RRG} \subseteq \epsilon_i^{RRT*} \tag{4}$$

Where ϵ_i^{RRG} and ϵ_i^{RRT*} are the edges expanded by these algorithms.

According to theorems mentioned above, we can prove that if there is a feasible path in the sample space, there holds that

$$\lim_{i \to \inf} P(V_i^{RRT*} \cap X_{goal} \neq \emptyset) = 1 \tag{5}$$

That shows the RRT* is also probabilistic completeness and is guaranteed to find an path after finite iterations.

2.4.2 Advantage2: Fast finding sub-optimal path

As described in the algorithm steps for RRT*, the RRT* can expand randomly during the process. As a result, the algorithm will first find a feasible path and then repeat the process to increase the result. As a result, it can get a feasible but not optimal path in a short time. After that, whenever it stops, it will return a feasible path.As proven in coverage part, the later it stop, the better result it can plan.

2.4.3 Disadvantage1: not optimal path in finite iterations

Comparing to graph search based methods like Dijkstra and A* which can guarantee optimal path in the graph, RRT* has a shortage that it cannot guarantee optimal path. For RRT*, since the process repeat to randomly optimize the result, follow the similar process in 2.4.1, we can prove that when $i < \inf$, the following inequality holds:

$$P(c_i = c*) < 1 \tag{6}$$

where c is the cost of the path, c* is the minimal cost. Which means it cannot find optimal path in finite time.

2.4.4 Disadvantage2: high time consumption

The time complexity of the RRT* can be computed as the sum of time of its 4 processes. The time used for N iterations which means add N nodes to the tree is[3]:

$$\begin{aligned} T(N) =& T_{sample}(N) + T_{nearest}(N) + T_{extend}(N) + T_{add}(N) \\ =& O(N) + O(N^2 - N) + O(N) + O(N) \\ \approx& O(N^2) \end{aligned} \tag{7}$$

As a result, it may take a long time to get a path, especially for a path in a complex scenario.

3 Conclusion

In this project, I implemented the RRT* algorithm to solve the planning problem in an 2 dimension graph with some obstacles. And the ideas of the RRT* is introduced in this report. Besides, I analysed the advantages of the RRT* include the coverage property and rapid generation of sub-optimal path. I also mentioned its disadvantages on path optimization and time complexity.

References

[1] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011. [Online]. Available: https://doi.org/10.1177/0278364911406761

[2] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[3] M. Svenstrup, T. Bak, and H. J. Andersen, "Minimising computational complexity of the rrt algorithm a practical approach," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 5602–5607.