CS8803 ACRL HW3: Car in Maze

Tianrong Chen*,Yunzhi Lin†, Yanjie Guo‡
Georgia Institute of Technology, Atlanta, GA, 30313, USA

I. Introduction

In this homework, We applied two kinds of methods to solve this problem. The first one is RRT. We modified the vanilla RRT slightly to make sure each new node is valid for actions, which means, the node is reachable. However, this method will cause a serious problem: when the car is stuck in a narrow lane which is a dead end lane, the valid sampling node is sparse since the sampling node is required to be reachable. As a result, the car will be trapped in an endless loop. The second method we applied is MDP Value-iteration. In this method, we can iterate on the Q-Value function for each discretized state in the Maze, and find the optimal action to minimize the moves as our cost. With a suitable cost function, we can stay in the middle of a road as much as possible. This makes sure the car robot have a safest way.

II. Methodology

A. Rapidly-exploring Random Trees

Rapidly-exploring Random Trees, as its name suggests, is a path planning algorithm aimed at searching over a high-dimensional space to find a feasible way by building a random tree. With randomly picked samples, the algorithm calculates the corresponding nodes according to some specific metrics. We propose this method because of its outstanding characteristics: rapidity.

To absorb the advantage of this method, we proposed an online RRT with valid control checking method to solve this problem. Our method is as follows: first, we assume all the roadblocks are open, and then we apply RRT to plan a feasible path which can lead the car robot to reach the goal. Then, after we apply the control sequence, if we encounter a roadblock, we will update the observed map. Then we will re-plan the path to manage to reach the goal again. For each sampled node, we not only evaluate whether it is in the obstacle as the traditional RRT[1] does, we also check whether this node is reachable to make sure that as long as a node is added into the tree, the node is achievable.

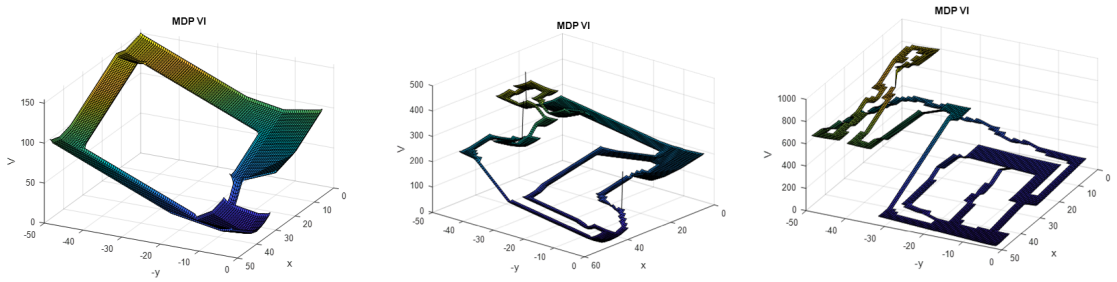
B. MDP value-iteration

The intuition of this method comes from the difficulties of finding a valid control to get desired RRT leaves. In detail, it means that either we can have a relatively good trajectory, but we cannot get there without collision; or the feasible RRT is confined in some region that is not hard to reach. The reason for this situation is that, now we consider the car as a rectangle instead of a node which is the classic default configuration for RRT setting. This configuration is easy to meet with when the road is wide, and the feasible region is large. However, in this problem, if the roadblocks are closed, in extreme conditions, the car need to go along a narrow, long and rugged road which will lead to a crash easily. So we have to find a safer way to generate better policy. The key problem in the former method is: the sampled nodes are not so safe to achieve and to leave. So we need a method to stay in the middle of a road. To reach this goal, we are going to use value iteration based on MDP. First, we assume the cost of the goal state is 0. And we add the extra cost when the state is closed to wall, specifically, if the wall existed in any four corners of the current state, or in four sides,the cost of the state will be added as follows:

$$
\begin{aligned}
Cost_{state}(s) &= \sum_{c \in corner} \mathbb{I}(Wall \in c) + \sqrt{2} \sum_{s \in sides} \mathbb{I}(Wall \in s) \\
Cost_{action}(a) &= \sqrt{2}\mathbb{I}(\theta = \pi/4 \times k) + \mathbb{I}(\theta = \pi/2 \times k) \\
k &\in \mathbb{Z}^+ \\
Q(s,a) &= Cost_{state}(s) + Cost_{action}(a) + \min_{a'} Q(s', a')
\end{aligned}
$$

*Master Student, Graduate research assistant, Electrical and Computer Engineering.
†Master Student, Graduate research assistant, Electrical and Computer Engineering.
‡PhD Student, Graduate teaching assistant, Daniel Guggenheim School of Aerospace Engineering.

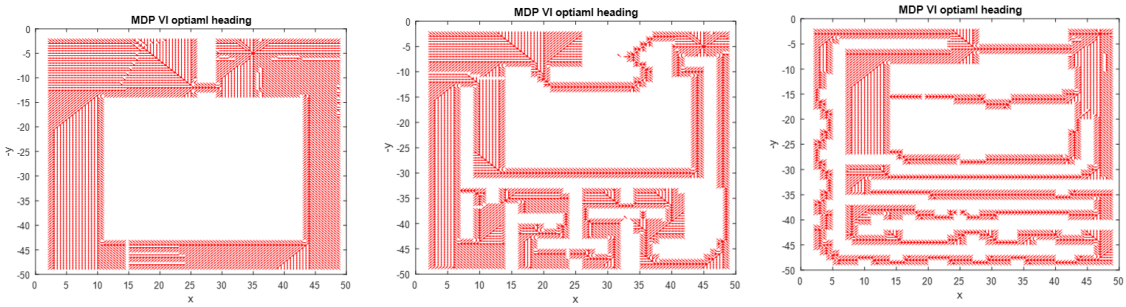
Through this function, we can plot the value of each valid state in the maze as shown in fig. 1:



(a) Map 1 (b) Map 2 (c) Map 3

Fig. 1. Value functions for the three maps

Where the x, y represent for the x and y coordinates of the states in the map, and z axis represents for the total cost to go. Our policy can then be generated by finding the minimum-cost action in each state, which is shown in fig. 2:



(a) Map 1 (b) Map 2 (c) Map 3

Fig. 2. Action maps for the three maps

Each arrow represents for the policy at the current state. Meanwhile, this policy keeps the car in the middle of the road, which is also what we desire.

To evaluate the policy map, we can find the policy only have eight directions. In order to make this problem more convenient, we only design eight controls to satisfy these eight actions. Concerning the generalization situation, we design a specific control method to satisfy any directions. However, we only use the subset of the control method.

III. Maneuver

A. Spinning around

In the human point of view, when we are stuck in a narrow dead-end road, it is a good idea to turn around in an approximated way. The solution of this problem is moving ahead with maximum steering, then moving backward a little bit, then keeping doing this. This method can let us adjust the direction of the car to any direction we want without shifting the current x,y state. It has significant meaning because we can turn around in a narrow road in this way. Otherwise, it is too hard for a car-like robot to turn around. Because under the limitation of the dynamic function of the car robot, it can only reverse in a straight line. So we go on to think of the maximum steering angle, we find that by setting u as -2, the car will reverse straightly 0.05 pixel. By setting u from -1 to 1, the car can turn left or right up to 0.05 rad respectively. Moreover, we note that in one step, by different u setting in [-1,1], the car will always step forward by

around 0.15 pixel. So if we want to spin around, we can do a series of action $\{-2, -2, -2, \pm 1\}$. It allow us to adjust the direction of the car by 0.05 rad without shifting the current x,y state. By keeping doing this series of actions, we can spin around in any degree.

B. Steering

If we only need to go to some place in a small neighborhood, we can assume that there is no obstacles between the two points. Given the ability to spin around, a simple control is to adjust the heading to point to the new position, and then go straight on. For most cases, this control will work, though it is not efficient. In some extreme cases, we need some heuristic backup adjustments. If the original control fails, try the following adjustments until it works or no adjustments are available: execute it without the last step; spin in the opposite direction; turn to the opposite direction and going back; go straight for 4 steps; back 1 step; turn right for 1 step. These adjustment can solve most extreme cases but we will see the car can get stuck in some place anyway.

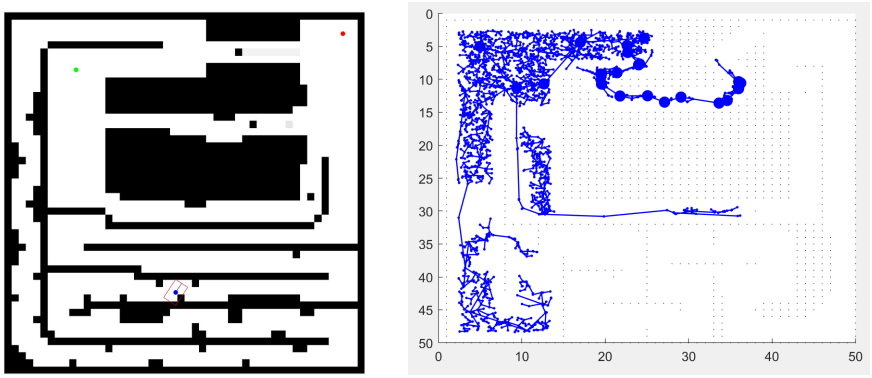
IV. Analysis

A. RRT

For the first map, the RRT method works well as we expected, because of the large feasible region the map.The car can always find the sub-optimal path to reach the goal. Meanwhile, all the path is reachable according to the nature of our modified RRT. For the second map, the RRT method can find a reachable path, and a feasible control to reach the goal while all roadblock are open. However, When all the roadblocks are closed, the car cannot find a feasible solution for this problem as expected. Since RRT is sub-optimal path, the car cannot reach the leaf of the graph even though the graph shows that the path is find. Another problem of RRT method we used in this problem is that: as we known, RRT works terribly in a maze because RRT will generate large amount of redundant samples in the maze environment. Meanwhile, our method only cares about the feasible samples, combined with these two properties, it is almost impossible to find a tree to reach the goal state.

B. MDP-Iteration

For the first and second maps, the MDP-iteration method has robust performance in the following two senses: the car will find the shortest path in an open space, and also stay in the middle in a narrow lane. The balance between fast and safety is achieved by using a suitable cost function.



(a) MDP failure case for map 3

(b) RRT failure case of map 2

Fig. 3. Failure cases of the algorithms

V. result

A. RRT method result

Considering the page limitation, we will not show many details of our RRT method since it could not solve most of cases in Map2 & Map3. Here we just give out a fail case of our RRT algorithm as shown in fig. 3 (a):

Just as the above fig. 3 shows, it is very hard for our proposed algorithm to find the downward path since it is too narrow. In most of the cases, our car is just trapped in a narrow region and could not find the way to the goal.

B. MDP method result

For MDP-method, we do not rely on sampling so the result will be consistent. We run through all three maps and run all the possible situations in the map to check the performance of our method. The following tables are the results. The title shows which map we are using, the index represents for the individual situation in one map (eg. index 1 represent for the map 1 in which the roadblock 1 is close, index 2 represent for the map 1 in which the roadblock 1 is open). The step means how much step we need to take in order to reach the goal.(remind: there is a constrain of the step in this problem, which is 100000). According to the following table, we can find that, as long as we manage to reach the goal, the step will not exceed the constrain. Meanwhile, the success rate of three map is: 100%, 100%, 94.7%.

Index	1	2
Step	1907	777

Table 1 Map 1

Index	1	2	3	4	5	6	7	8	9	10
Step	8641	1374	1374	1374	1374	1374	1374	1374	1374	1374
Index	11	12	13	14	15	16	17	18	19	20
Step	4128	1374	1374	1374	1374	8641	1374	1374	1374	1374

Table 2 Map 2

Index	1	2	3	4	5	6	7	8	9	10
Step	2786	2786	2706	654	654	2800	654	654	654	654
Index	11	12	13	14	15	16	17	18	19	20
Step	654	654	Fail	654	654	654	654	2800	654	Fail
Index	21	22	23	24	25	26	27	28	29	30
Step	654	654	2774	654	2706	654	654	654	654	654
Index	31	32	33	34	35	36	37	38	39	40
Step	2786	654	2774	654	654	654	2680	654	Fail	2774

Table 3 Map 3

VI. Conclusion and Future Work

To sum up, we tried two method to solve this problem. The essential difference between these two is, one of methods is random serach based, and another one is policy based. RRT only success for the first one, but DMP-VI can success in all three maps except some extreme case. In terms of future work, we will tune the parameter in the calculation of cost which will lead to more general policy. Also, we will also do research on the RRT based method to improve the efficiency of sampling, to make this algorithm also be suitble in maze environemnt.

References

[1] LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning." (1998).

[2] Hsu, David, et al. "The bridge test for sampling narrow passages with probabilistic roadmap planners." 2003 IEEE international conference on robotics and automation (cat. no. 03CH37422). Vol. 3. IEEE, 2003.