Lab 4 - A* Algorithm

Songyou Peng
psy920710@gmail.com

I. INTRODUCTION

As we all know, in most of the robotics scenarios, the map can be modelled as a connected graph and the path planning problem is able to be simplified to a graph searching problem. Thus, a graph search algorithm A* method has been implemented in this lab.

Throughout the lab, we will firstly discuss the details of our work and illustrate the results in different scenarios. And then the way that we calculate the path will also be introduced. Problems that we faced will be put in the end.

II. IMPLEMENTATION

A. A* algorithm

The secret to A* algorithm success in shortest path finding for robotics is that the method combines the idea of Dijkstra's algorithm [1] (prefer vertices that are close to the starting point) with Greedy Best-First-Search [2] (prefer vertices that are close to the goal) [3]. Assuming that we have n vertices in the map, $g(n)$ represents the real distance from the starting vertex to all the others, and $h(n)$ represents the heuristic distance from goal point to the all the others. Hence, in each iteration, the method always chooses the vertex i that has the lowest cost $f(i) = g(i) + h(i)$. This is the core idea of the A* method and the implementation details will be brought up in the following.

First of all, as the preparation, heuristic distances for all the vertices and a matrix storing the exact distance between two connected vertices are calculated in advance. Beginning from the starting point, the costs f for all the connected vertices to the starting point are obtained and inserted into the open list O together with the indexes of them and their "parent" - starting point. Secondly, the vertex having lowest f is picked up from O as the next "parent" point. Finally, The index of this vertex and its parent are embedded into the close list C, which will be applied to acquire the shortest path.

Once having a new "parent", the same procedure that checking all the connected vertices is implemented. The only difference from the first step is that we need to verify the vertices. The O list will not be updated when these two cases happen.

- The connected vertex is already on the close list C.
- The distance between the vertex and the starting point passing through the current "parent " is not shorter than other "parent".

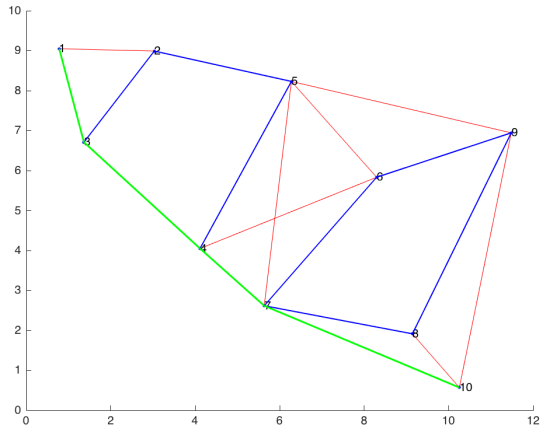
For the first one, it is reasonable not to visit the vertex in the C again. For the second, we already know following this route will not provide the shortest path. If the vertices don't meet these two conditions, we can insert them into O and pop out the vertex with smallest f and set it as the next "parent". If we find out the vertex with the shortest distance in C is the goal point, A* algorithm finishes, and the best path has been found.

B. Calculation of shortest path

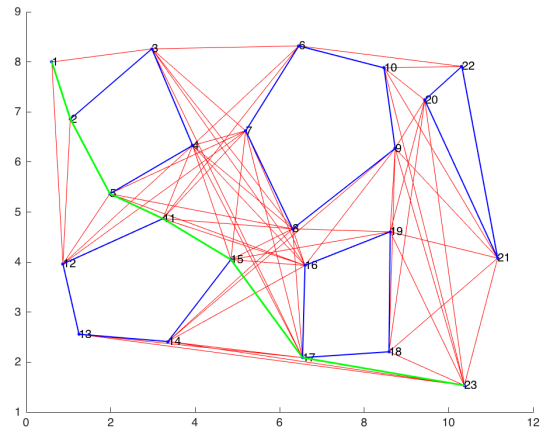
In order to obtain the shortest path for a certain scenario, we back propagate from the goal point to the starting point. Every row of the C list is in the format like [Vertex, Parent]. The last vertex is the goal point, of which we find the parent. And then we check where the parent of the last vertex, so on and so forth, until the starting point becomes the last parent. The best trajectory is acquired. Some results are shown in Fig. 2.

C. Faced problem

The first version of our implementation worked fine for the two maps with convex polygons in Fig. 2. However, when we tried another scenario with non-convex objects inside, some problems happened, which made us realise in fact there existed some problems in our code. In Fig. 2(a), there

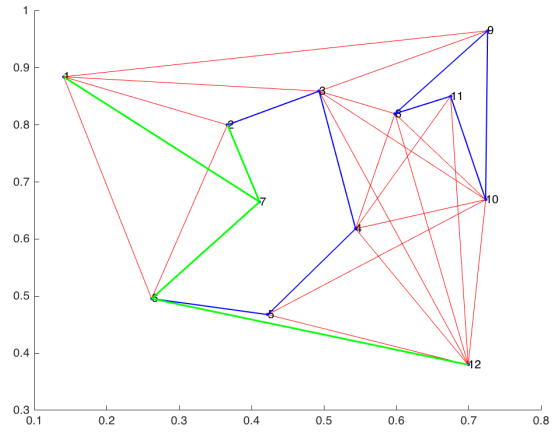


(a) Simple convex obstacle

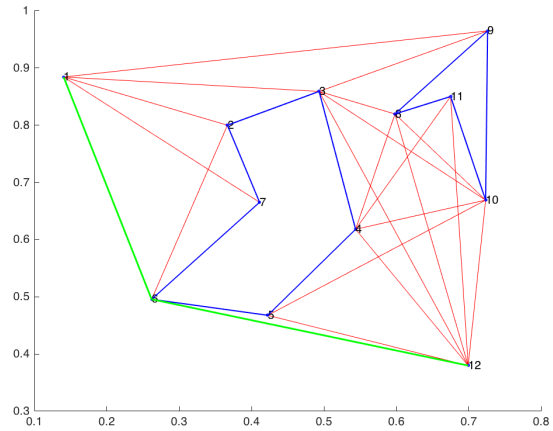


(b) Complex convex obstacle

Fig. 1. Illustrations for the shortest path using A* algorithm in convex obstacle scenerios. Green lines are the shortest path.



(a) Bad result



(b) Good result

Fig. 2. Illustration the optimal path before and after solving faced problems.

is no trouble for us to notice some unexpected paths are presented. After checking our code, we found out it was actually lucky to get the correct the result for the first two cases. There were two main problems due to not fully understanding the algorithm at that time.

One problem is we set the last visiting vertex directly as the parent without checking if this lead to a longer path or not. The other one is, instead of checking whether the vertex is in the C list, we checked if the vertex was in the O list. This brought up the problem that once a vertex inserted into O list, the vertex distance f could never be updated so the distance may not be the shortest. After solving these two problems, the result is shown in Fig. 2(b).

III. Conclusions

In this lab, the famous A* algorithm is implemented for robotics path planning and tested in different scenarios. the algorithm can provide an approximated best result very fast.

References

[1] Dijkstra, Edsger W. "A note on two problems in connexion with graphs." Numerische mathematik 1, no. 1 (1959): 269-271.

[2] Bonet, Blai, and Hctor Geffner. "Planning as heuristic search." Artificial Intelligence 129, no. 1 (2001): 5-33.

[3] Patel, Amit. "Introduction to A* From Amit's Thoughts on Pathfinding." Introduction to A*. Accessed May 07, 2016. http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.