Lab 1 - Brushfire algorithm & Wavefront planner

Songyou Peng
psy920710@gmail.com

I. INTRODUCTION

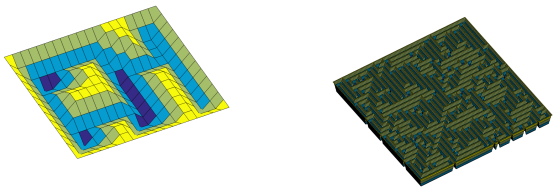
This report will briefly discuss the Brushfire algorithm and Wavefront planner algorithm as well as the work which has been done during the implementation. We will also talk about several problems faced and how we solved.

II. IMPLEMENTATION

A. Brushfire Algorithm

We first visit all the 8 neighbors of obstacles and give a new label. And then we start from all the new points and visit their neighborhood, giving another new labels as well. Finally, the algorithm stops when all the points are assigned labels.

In order to correctly show the repulsive potential that brushfire algorithm has created, we used Matlab function `surf` to represent the value_map. The smallest 14×20 map is shown in the Fig. 1(a). The yellow parts in the 3D representation image are the obstacles and the darker blue, the higher label values has. Similarly, we also plot the map from `maze.mat` (Fig. 1(b)).



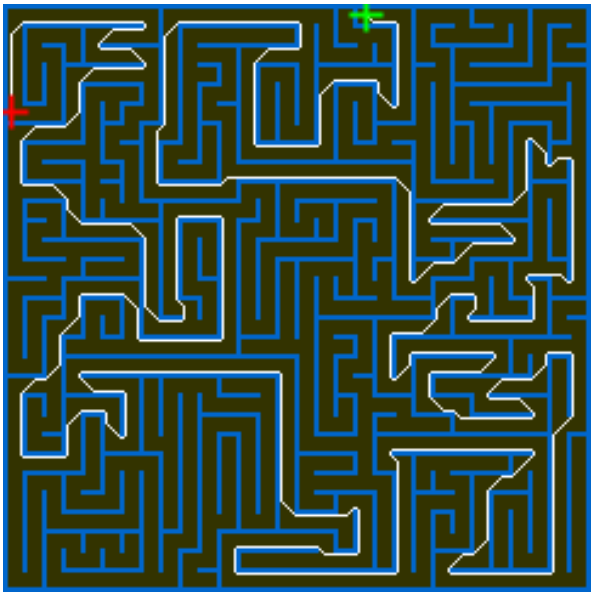
(a) 14×20 Simple maze (b) `maze.mat`

Fig. 1.

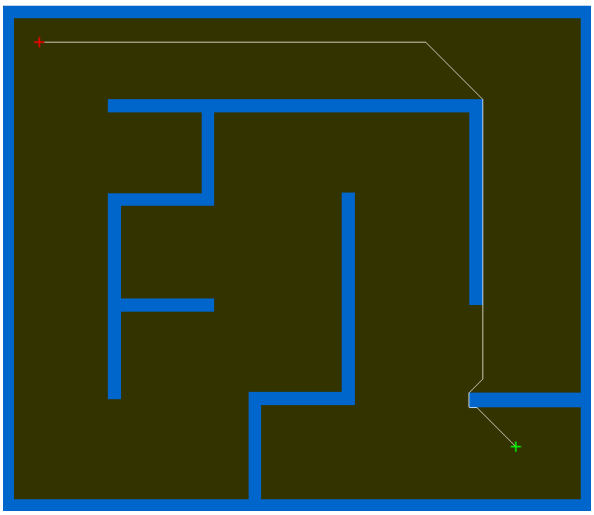
B. Wavefront Planner

Wavefront planner method is quite similar to a famous image segmentation method: region growing. Both algorithms start from a certain point, goal point in our case, and then visit the neighborhood and assign new labels to them, and then grow again from the points with new labels.

After getting the `value_map` from wavefront planner method, we are able to find the best trajectory between a given starting point and the goal point. Our method is simply checking the neighborhood of the starting point. If the value of the neighbor has the smaller, we add it to the trajectory array and use it as the next growing point and search for its neighborhood. The results are shown as follow:



(a) `maze.mat`



(b) `mazeBig.mat`

Fig. 2. Trajectories for the given map. Red cross is the starting point, while green is the goal point.

At first the wavefront algorithm didn't work for `mazeBig` map and we could not find the solution. Finally, we found out that it blamed for our ending condition. For the brushfire algorithm, the algorithm stops when all values in the `value_map` are not zero. Actually, in `mazeBig`, the value of $(1,1)$ is 0, which is supposed to be 1 in the obstacle. This is OK for brushfire algorithm as the growing process starts from the obstacles, so this 0 in $(1,1)$ can be assigned a label. However, because the wavefront algorithm starts

growing from the free area, that 0 will never be visited and the algorithm can never stop if we use all non zero as our ending condition.

In order to solve this problem, every time a pixel is visited, we put it into a waiting queue. During the process of the algorithm, all the points in the free area will be pushed to the rear of the queue. Only when all the points in the queue have been popped out, the algorithm finishes. With this ending condition, the algorithm works even if that isolated 0 point will not be visited.

Another problem we faced was the speed of the algorithm performed not well. So instead of using dynamic array to store the coordinate of each point every time, we initial a static array and use pointers to move. The left pointer is in charge of the current visiting point while the right pointer will move one step as a new point has been pushed into the queue. Therefore, when the left pointer is in the same place with the right pointer, the algorithm process stops.

One last problem we faced was about trajectory planning. In the beginning, we considered using the neighbor with shortest Euclidean distance to the goal as the next step. This method may get stuck because the point with the shortest distance may lead to the obstacles. At first, we simply changed the criteria to use the first neighbor who has a lower label value as next step. The trajectory acquired from this scheme was good but not the shortest (Fig. 3). At the same time, we noticed the robot using this algorithm may move alongside the obstacles, which may be dangerous for robots

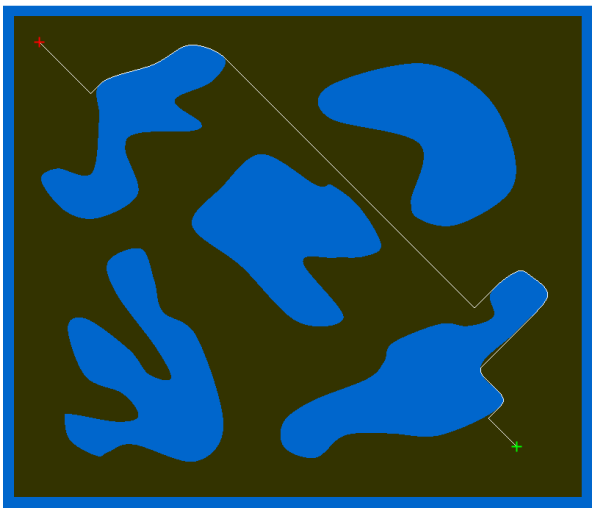


Fig. 3. First version trajectory for `obstaclesBig.mat`

Considering the problem mentioned above, finally, we gave vertical and horizontal directions the higher priority, the results of both schemes are shown in Fig. 4.

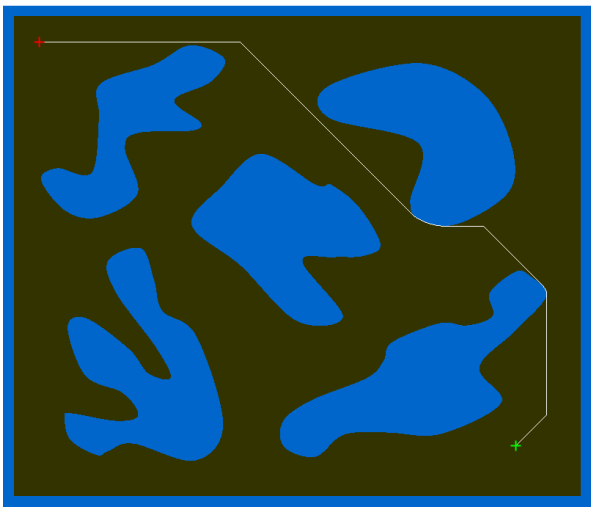


Fig. 4. Improved version trajectory for `obstaclesBig.mat`

III. Conclusions

In this lab, we implement Brushfire algorithm as well as wavefront planner, and apply both of them in several maps with different size and complexity. It turns out that both of the algorithms are really time saving and can deal with different situations.