University of Girona

Autonomous Robots

Lab 1

Potential Functions

Author:
Di MENG

Supervisor:
Dr. Marc Carreras

March 5, 2017

1 Introduction

The goal of this lab is to grab the knowledge of Potential Functions which are Brushfire algorithm and Wavefront algorithm and program these two algorithm in Matlab to realize a simple path planning.

The case is that there is a map with obstacles which show value 1 in the image and the values of available space in the map show 0. Based on the given map, we have to generate the potential of repulsive obstacles and find an optimal trajectory to the goal point in the 2D map.

This report is mainly divided into three parts which are the ways to realize Brushfire Algorithm and Wavefront Algorithm and Results Evaluation.

2 Brushfire Algorithm

This algorithm is mainly to generate the potential of repulsive obstacles based on the given map. The obstacles are assigned with value 1 which have a low energy and every cell in the map holds the energy with respect to the distance to obstacles.

Basically, my code does the following steps to generate the value map:

1. Go through all the pixels in the map and find the pixels with value 1 which are obstacles.
2. Assign the pixels next to the obstacles(8 connectivity) with the value plus 1.
3. Assign the pixels of value 2 with value plus 1.
4. Iterate the assignment with value plus 1 until there's no pixels with value 0 in the map.

In the end, we get a matrix full of values equal to and greater than 1. The result is a distance map where each cell holds the minimum distance to an obstacle.

3 Wavefront Algorithm

This algorithm is basically divided into two parts. Firstly apply the Brushfire algorithm starting from the goal, so that the goal holds the lowest energy and the start point holds the highest energy. The configuration is built for robot moving to the lower energy. There are many possible paths for robot to reach the goal from the start point. What we have to do about this algorithm is to find a shortest trajectory along to the gradient.

The code follows the next steps:

1. Generate a value map using the Brushfire algorithm starting from the goal position.
2. Put the current position to the start point.
3. Find the minimum value in the 8 neighbors of current pixel which actually is the value minus 1, and consider this position as current position and put it into the trajectory vector.
4. Continue the step 2 until the current position reach the goal.
5. Finally we get a set of coordinates saved in a vector which is the optical trajectory.

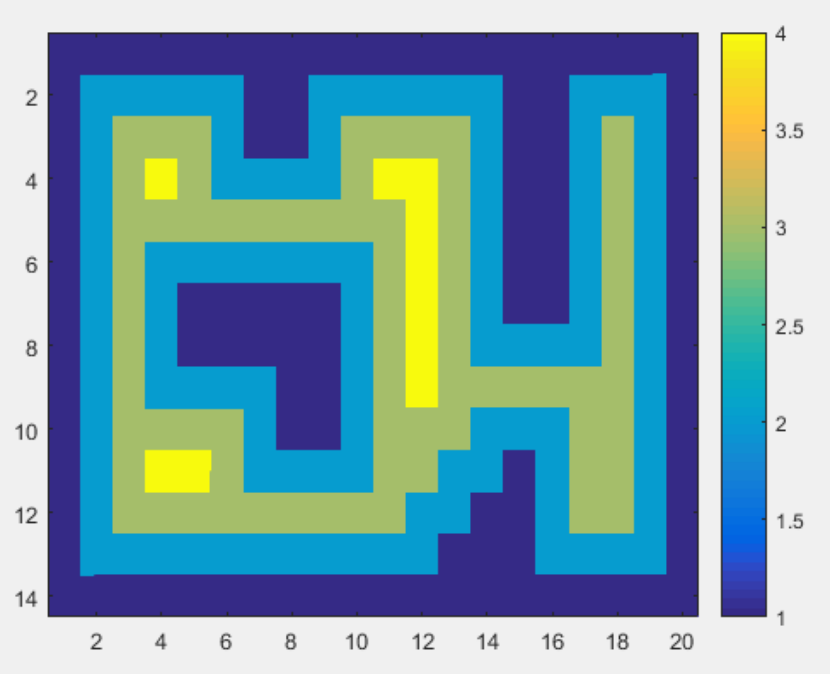
About the problem that wavefront planner does not distinguish the up, down, left, right, diagonal movements and sometimes choose the diagonal one which would cause longer path, my solution is that considering the 4 connectivity firstly when choosing the next step of trajectory. If it dose not work, then consider the 8 connectivity.

From the results we can see that there's no jagged path.

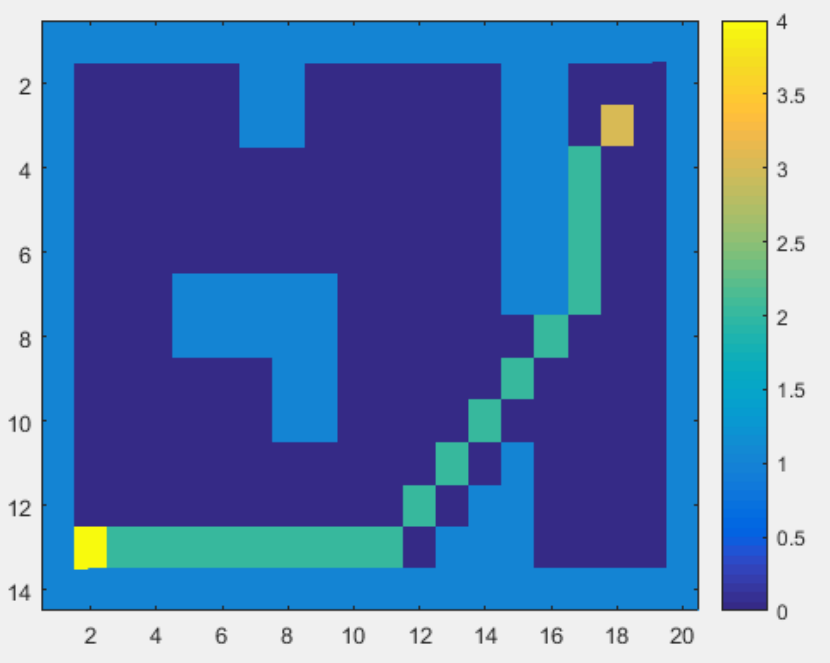
4 Results Evaluation

There is a script "demo.m" for evaluating the functions "brushfire.m" and "wavefront.m". Inside the "demo.m", users can input the map and visualize the trajectory.

4.1 Simple map



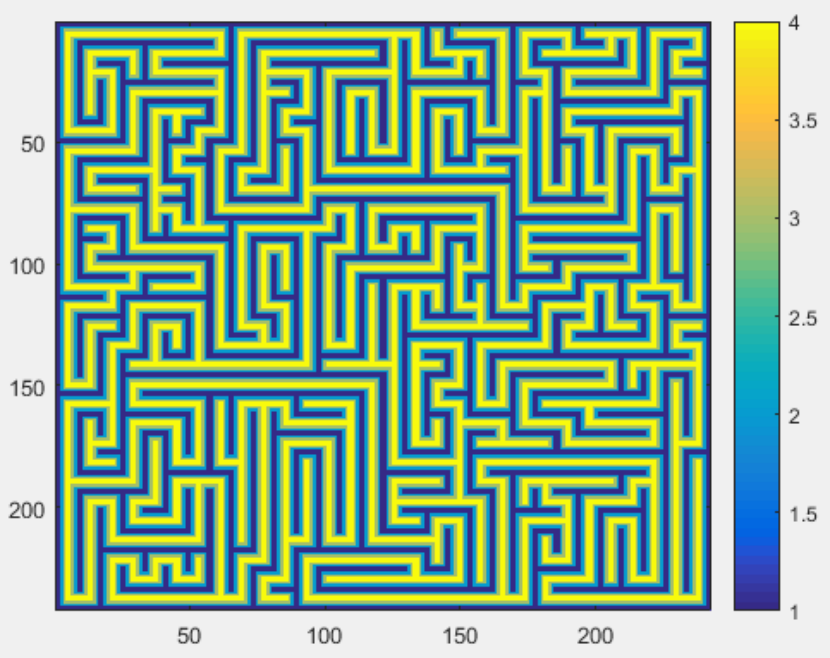
(a) Value map



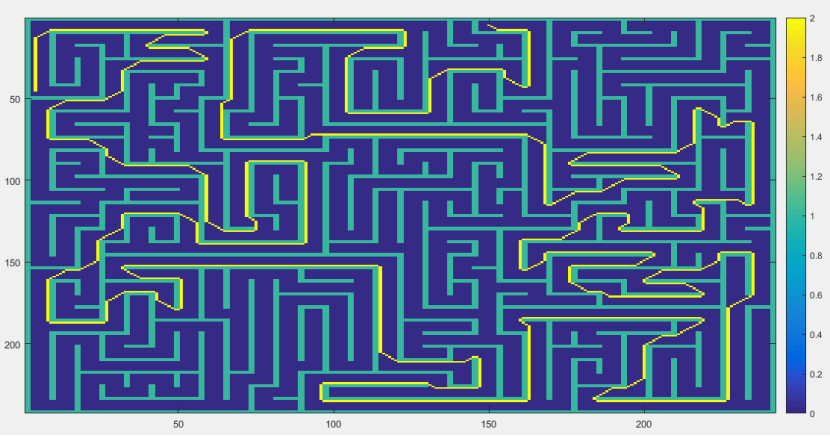
(b) Trajectory

Figure 1: Path planning of simple map

4.2 Maze



(a) Value map



(b) Trajectory

Figure 2: Path planning of maze

4.3 Evaluation

Time(s)	**Brushfire**	**Wavefront**
Map	0.013256	0.038180
Maze	0.945146	630.651559

As we can see, the results look proper. The trajectory goes to the goal following the gradient which is the shortest path. And the path is straight forward without jagged waste.

4.4 Problems and Solution

The problem personally faced about this lab is that when running the Wavefront algorithm, it is really time consuming.

It is because every time I search the pixels which are assigned specific values, my code goes through all the pixels of the map which is not necessary at all.

The solution is that the pixels whose neighbors to be explored should be put in a queue and this queue will be updated all the time. The value map would be generated until the queue went empty.

The mistake I've done is that I mainly focus on realizing the algorithm but not thinking in a efficient way. I will probably recode the algorithm in a more efficient way after the submission and send it by email.