Autonomous Robots
Potential functions report

Eduardo Enrique Ochoa

March 1, 2018

1 Introduction

Potential functions are functions that are used in the robotics field to move a robot form a high potential point to a low potential one. Usually this high potential point is set to be the start position of or robot and with this functions this is able to move until it reaches the goal point, which is the point of lowest potential. This type of functions are programmed to get the robot to move in an environment without crashing into obstacles and reaching the goal in an optimal way.

This lab-work consisted in the implementation of two well known algorithms that use this kind of approach, those are the *Brushfire algorithm* and the *Wavefront planner*. Using MATLAB this two algorithms have been implemented and tested with different types of environment and a moving trajectory for a start position to the goal is obtained and displayed.

2 Methodology

The lab-work consisted on implementing two path planning algorithms well known in the robotics field, the **Brushfire algorithm** and **Wavefront planner**. These two algorithms are based on potential functions and consist on computing a potential value for each possible position of the robot on the environment. With this our robot can find a path towards a desired goal which is in the direction of the lower potential values and can avoid obstacles by assigning a repulsive potential to them. As a training set we made use of a environment with a size of 14x20, that is 14 rows and 20 columns and it is represented in figure 1.

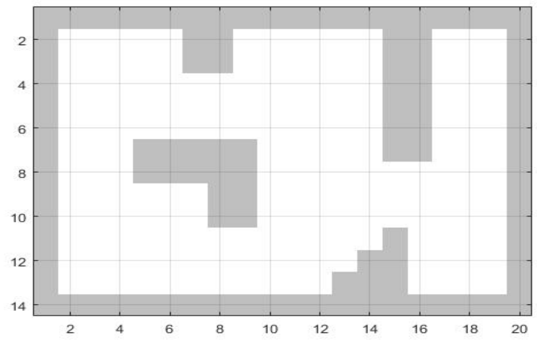


Figure 1: Training environment for algorithm implementation

2.1 Brushfire algorithm

The Brushfire algorithm starts with a known and initialized environment in which all possible position in which exists an obstacle are set to 1 and the free space is set to 0. Then through each iteration of the algorithm we assign to each cell a value that represents the minimum distance of that cell to an obstacle. In this evaluation it is important to define the connectivity that will be used on our environment and that can generate different kind of solutions.

A brushfire algorithm implementation has been realized in MATLAB and is used by calling a function [**value_map**] = **brushfire(map)** that takes one array as argument *map* which represent the original environment and returns a *value_map* array which contains the modified environment.The function takes into account an 8-point connectivity for each cell and iterates through all the positions until there is no 0 cell values in the map.
Two of the problem faced during the implementation, due to not knowing all the functions in MATLAB, was to localize each cell with value equal to zero and then compared the connectivity to know what value had to be assign to the cell. This issue was solved looking through the MATLAB documentation and, after obtaining each cell coordinate (x, y) with value equal to zero, the first iteration of the algorithm will just assign values to the cells that are connected to a one or more cells with 1 value. In which case it will assign 2 to that cell.

In the same way the function recursively goes through each zero position that has been left on the array and with each iterations assign values to the ones that are connected to the last ones that were assigned and that are closer to the obstacles. After all the processing it returns the mapped environment. This can be seen in Figure 2 in which we can observe the original map and besides it a gradient map with the obstacles being represented with the yellow color.

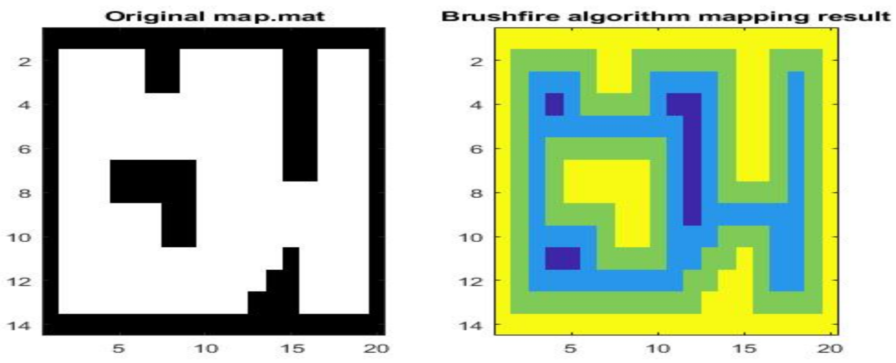


Figure 2: Original map and output of the brushfire algorithm implementation

2.2 Wavefront planner

The Wavefront planner algorithm is based on the brushfire algorithm and starts with the same initialized environment but the variant of the algorithm is that it places a goal position in a cell of the known environment and generates a potential value for each cell that represents the distance to the goal. The implemented solution to this algorithm calls a MATLAB function **wavefront** which takes as arguments a environment array *map*, and the x and y position of the start and goal of our robot, defined as *start_row, start_col, goal_row, goal_column.* The function uses a 8-point connectivity to search in the neighborhood of each cell.

In this algorithm throughout the assigning step of each cell a queue is created to traverse each position and store those neighbors that have not been initialized. The queue starts with one cell that corresponds to the goal and looks for the neighbors, then each neighbor is set and stored in the queue for visiting later. In this step a simultaneous check is done to see if the cell has not been assigned before or that it is not an obstacle, in which case they are not stored in the queue. After the processing we got an image as the one shown in figure 3.

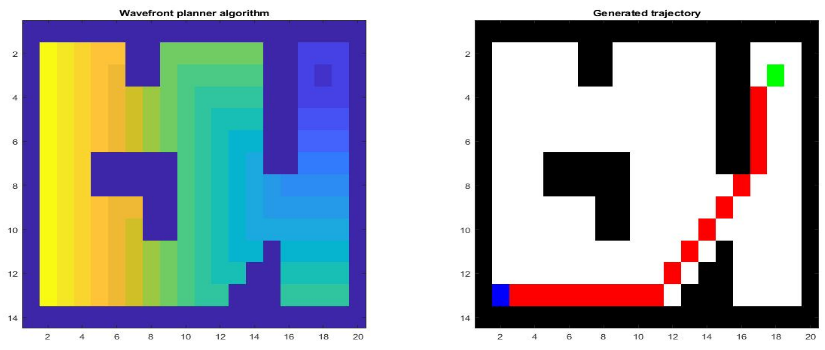
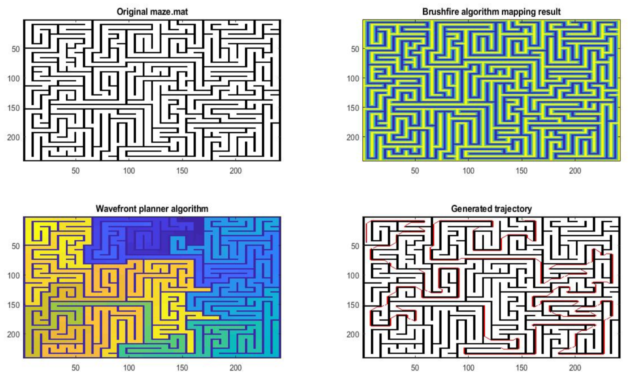


Figure 3: Wavefront planner algorithm output and map trajectory computation

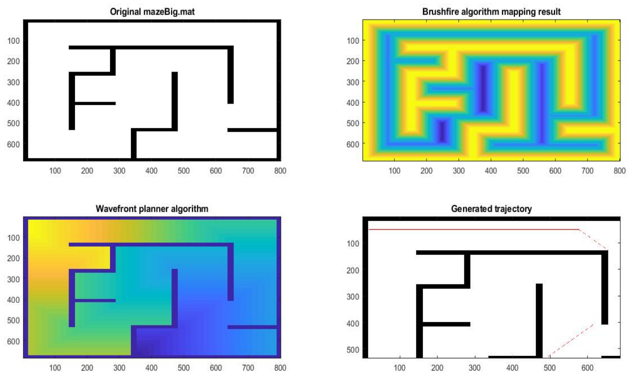
The function returns a value_map with the gradient distant of each cell to the goal and returns too the optimal trajectory to it. This trajectory is computed by using the start position coordinates given as parameters and checking the current pixel with each neighbor and computing a cost evaluation. If the neighbor gives a cost reduction to the value of the actual pixel then this is added to trajectory array and in the next iteration is evaluated. This is done until goal is reached. In figure 3 it is possible to see the solution for our tranning environment.

3 Results

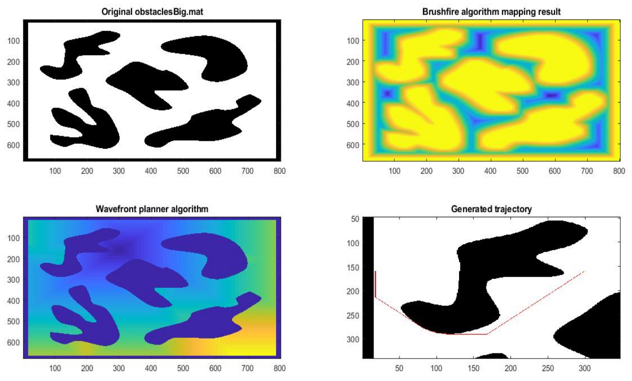
Both algorithms have been tested with different sets of initialized environments. This three test have been done with the files **maze.mat**, **mazeBig.mat** and **obstaclesBig.mat** and the results can be appreciated in Figure 4. For figure 4a we used the parameters **wavefront(maze,45,4,5,150)** that defines the goal at position $(5, 150)$ and the starting point in point $(45, 4)$.



(a) maze.mat environment



(b) mazeBig.mat environment



(c) obstaclesBig.mat

Figure 4: Brushfire algorithm and Wavefront planner functions applied to test environments

We did the same for 4b and 4c using its corresponding environments and goal positions of $(671, 353)$ and $(300, 750)$ respectively. And for starting points we use $(50, 17)$ and $(161, 18)$ respectively. Due to the size of this two last environment a zoom of the trajectory was captured in the figures to observe portion of the path generated.

4 Conclusions

Algorithms based on potential functions are useful for finding trajectories in known environments, this functions have proved through the results that we can placed a robot in a configuration space and then by looking for the direction in which the gradient becomes less we can reach a certain position and at the same time avoid collision of our robot with the obstacles. It is yet not possible to compare the computational time between other methods as we have not seen this but we can be sure that if a method reduce the computation of cells and look for paths with less cells compared then this time can be reduced have a solution with less distance.
By now the two implemented algorithms has the advantage of being easy to implement and introduce ourselves to some methods of the path planning issue when we have to navigate with a robot in a room.