Autonomous Robotics

Lab 1 Report: Potential Functions – Brushfire Algorithm and Wavefront planner

Mohammad Rami Koujan

M.Sc. VIBOT

University of Girona

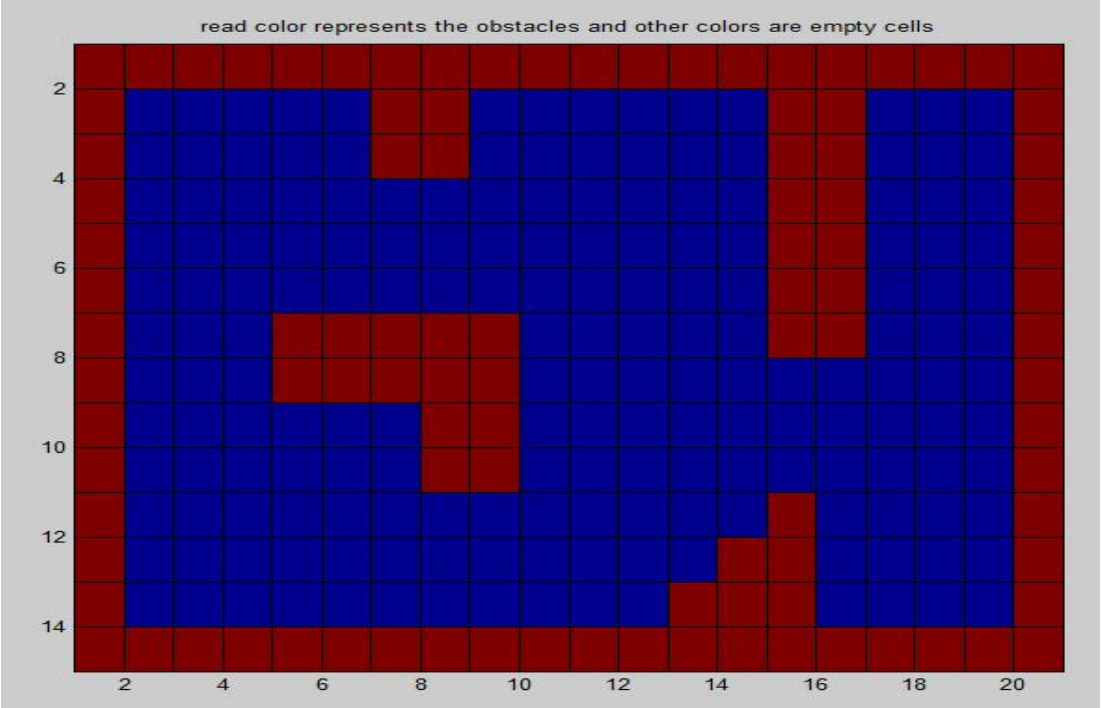
1. Introduction

This lab assignment focuses on the analysis, design and implementation of some path planning algorithms based on potential functions, specifically brushfire and wavefront. The program that is used to implement the mentioned algorithms is Matlab. In fact, brushfire algorithm aims at generating the potential of repulsive obstacles of a given map while the wavefront planner has the responsibility of detecting the optimal trajectory from a start point to the goal in a 2D environment that contains obstacles and is closed.

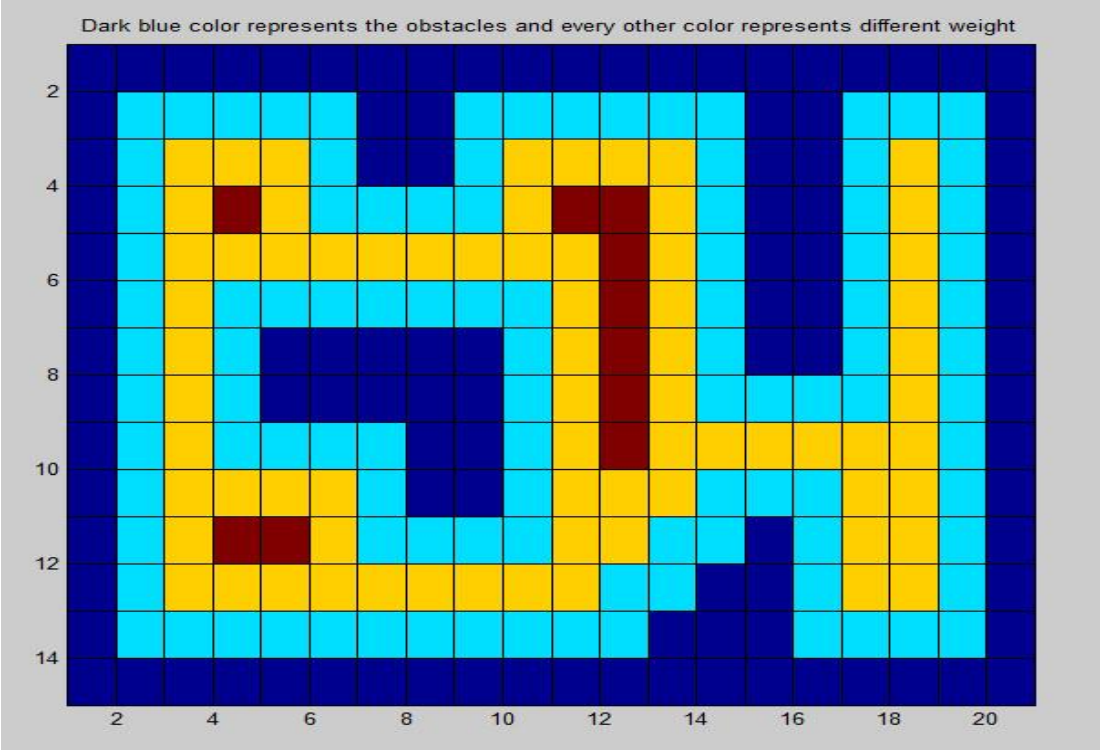
2. Brushfire algorithm

As it is mentioned previously, the purpose of this algorithm is to compute the repulsive potential of the obstacles' cells in a given map. This algorithm, indeed, is implemented as follow:

First, the programmed function searches for the obstacles' cells which are assumed to have a value 1. Then, it gets all the 8 neighbors of each obstacle and check whether any of these neighbors equal to zero, which means they are not assigned any value yet, or some other value. For the ones that have a zero value, they are given the value of the obstacle plus one. This indeed marks the end of the first iteration. The next iteration continues in the same manner but this time searching for cells with value equals to two then setting the free neighbors to three and so on until scanning all map cells. One efficient trick in the implementation is to check the number of cells that are checked so far in order to decrease the maximum value of the iterator. That is to say, instead of trying to iterate in a brute force method through each cell and repeat the previously stated procedure, the function checks the number of cells which have assigned a value and decreases the value of the iterator by the same number. Figure 1 shows an example of a map and its brushfire figure generated by the explained function. In the brushfire figure the different hues represent distinct distances from the obstacles, which are shown in blue color. It is important to mention that for bigger maps it is much more difficult to recognize the different cells just by color and it depends on the color map used. Hence, it is better to relay on the numbers rather than just take a quick look at their visual maps.



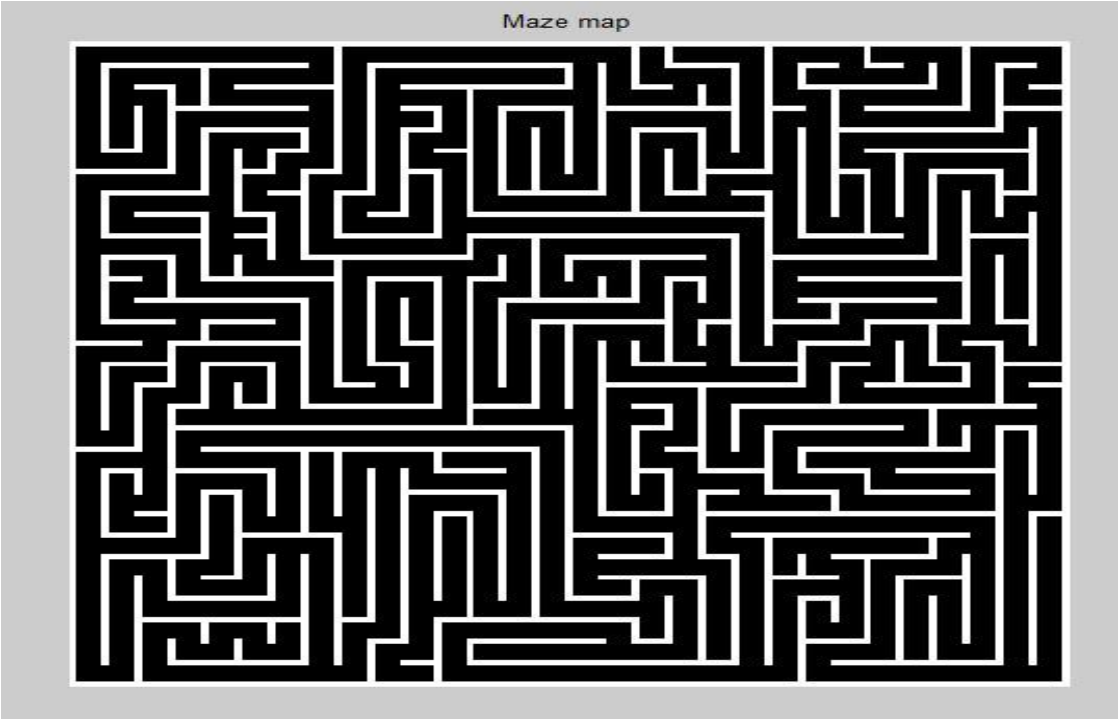
(a)



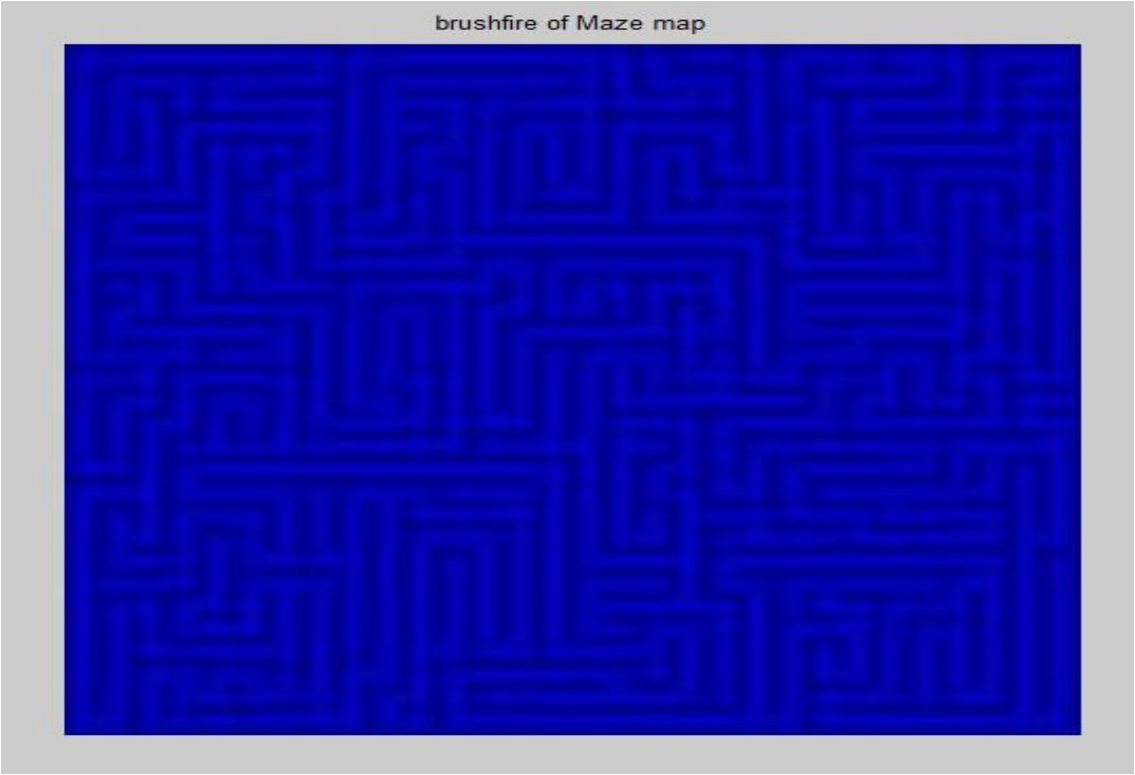
(b)

Figure 1(a,b): original and brushfire maps

The previous algorithm was also tested with three other maps, maze, mazeBig, and ObstaclesBig. The results are demonstrated in figures 2, 3, and 4 respectively.

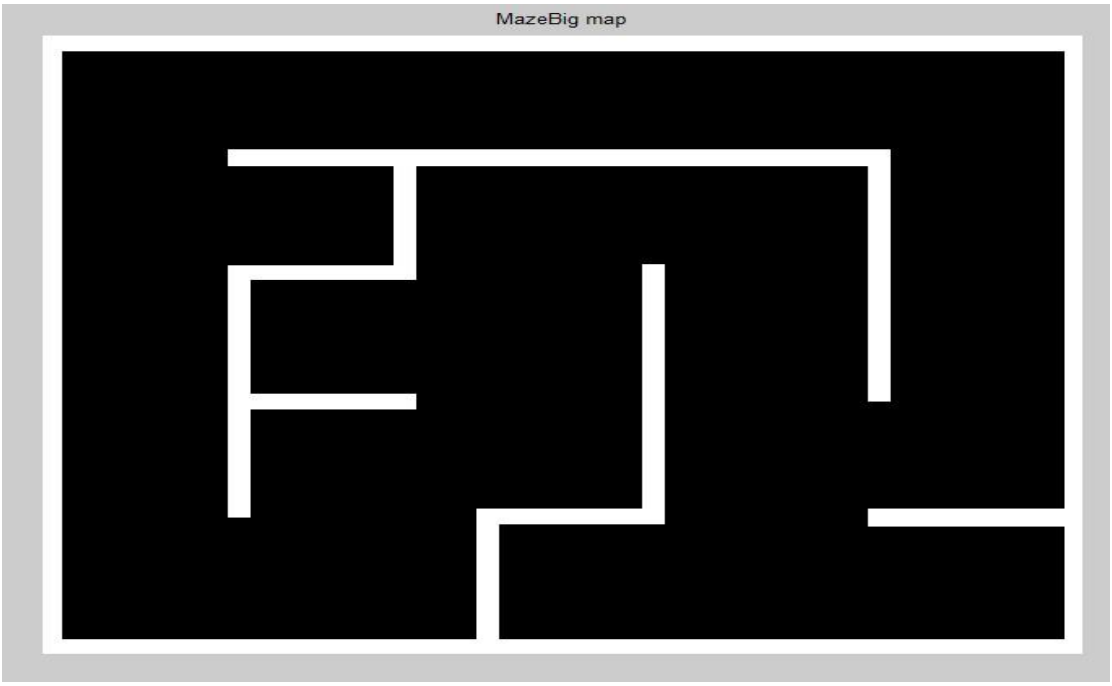


(a)

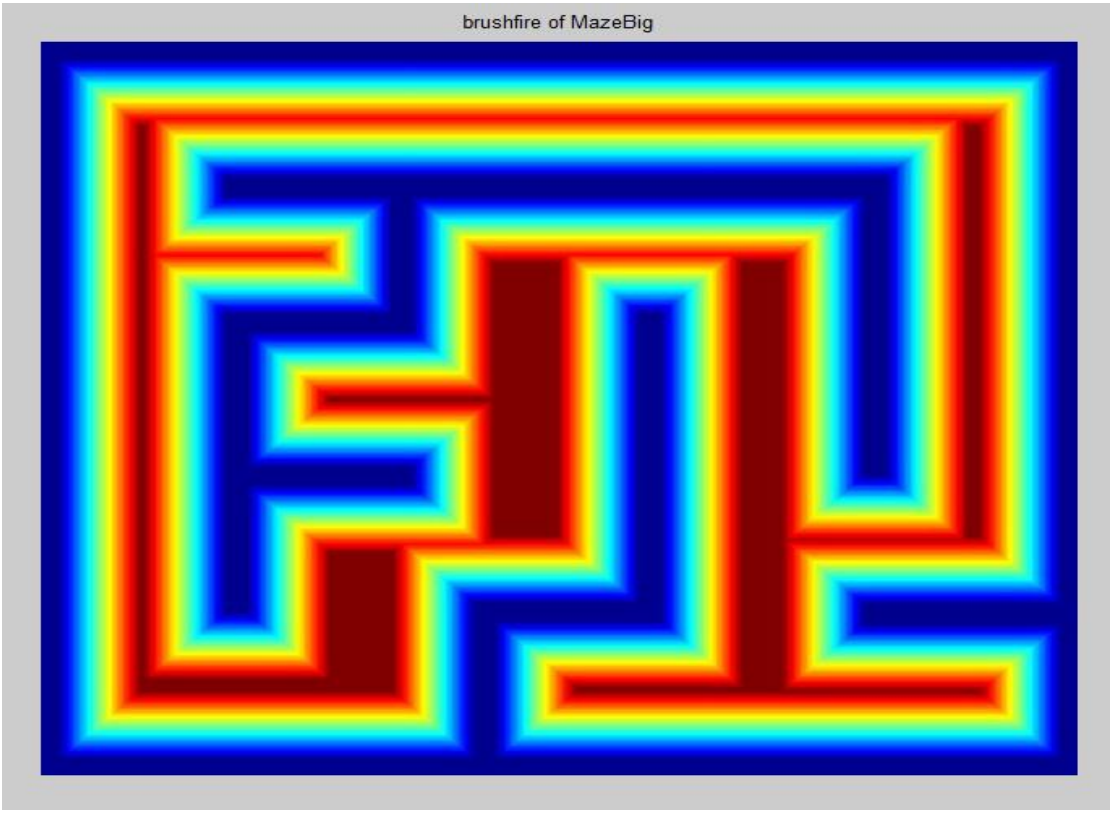


(b)

Figure 2(a,b): original and brushfire maps

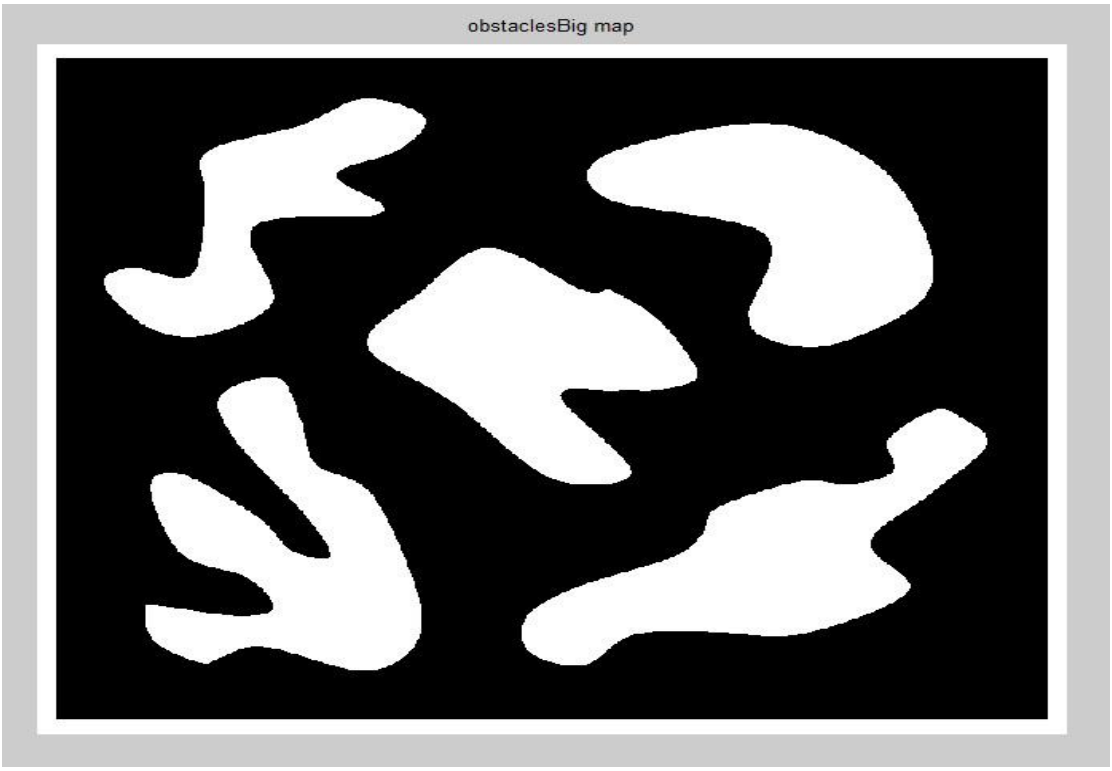


(a)

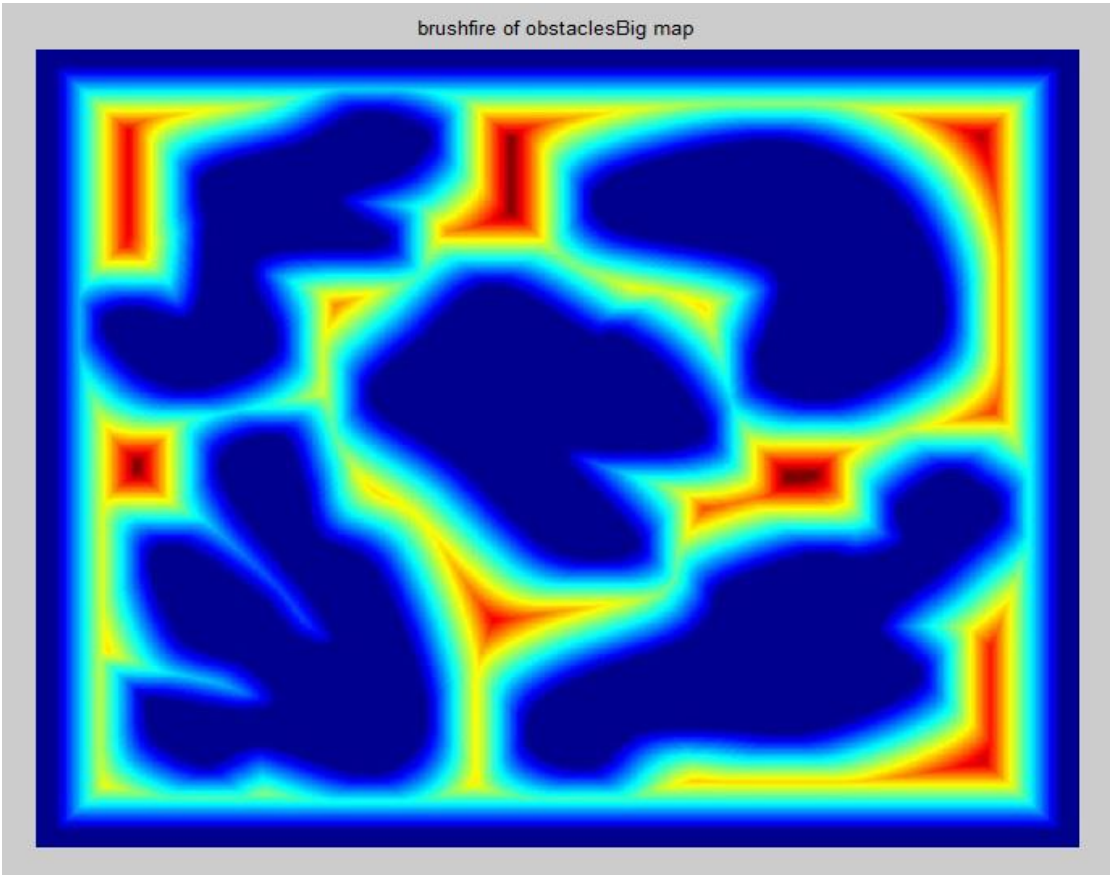


(b)

Figure 3(a,b): original and brushfire maps



(a)



(b)

Figure 4(a,b): original and brushfire maps

The following table presents the corresponding run time of brushfire algorithm for the previously shown maps.

Map's name	***Map's size***	***Time response(s)***
The given map in the assignment	14*20 cells	0.944 s
Maze	243*243 cells	2.177 s
mazeBig	803*683 cells	45.710 s
ObstaclesBig	803*683 cells	44.931 s

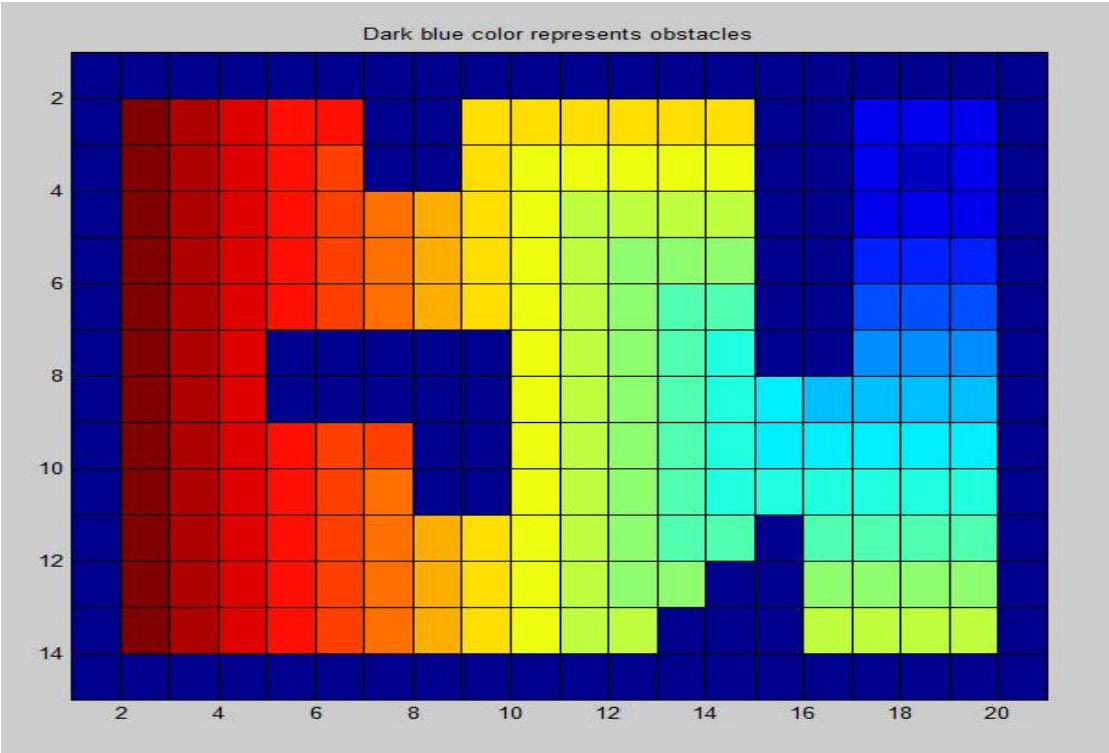
Table 1: brushfire algorithm time response

The previous table suggests that bigger maps impose higher time response.

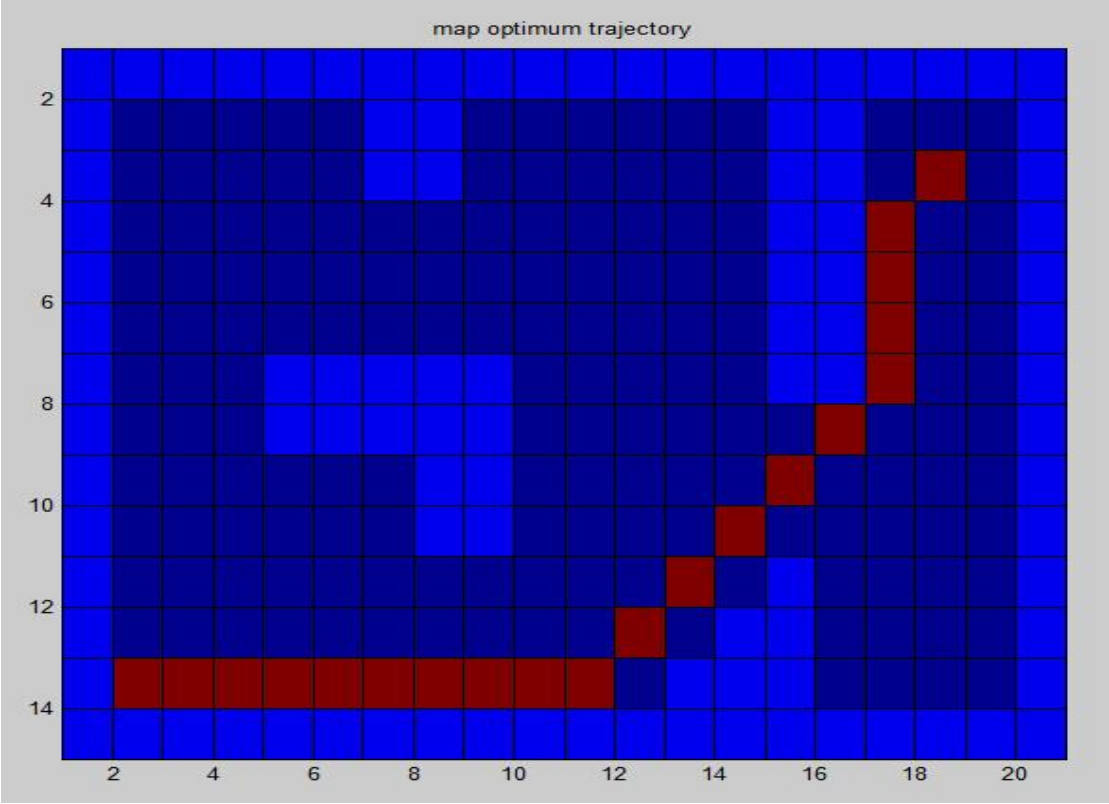
3. Wavefront planner

The aim of this algorithm is to find the optimal trajectory between a goal and a start cell after building the wavefront value map. The principle of implementation of this algorithm is initially similar to the brushfire method. Therefore, the function begins from the goal point and then obtains the 8 neighbors of this cell. If any of those neighbors have a value of zero, it is given a value of 3, the goal assumed to have a value of 2. Then, the process continues but this time looking for cells with values 3 to give their free neighbors label of 4. This methodology keeps repeating itself till it covers all the map with low value cells which are near the goal and high value cells that are far from the goal. The final map has the shape of a wave propagating away from the goal.

In the next step of this algorithm the objective is to find the optimal trajectory between a start point and a goal. In order to do so, the implemented function begins from the start point by checking its neighbors to find the one with the minimum label, prioritizing , left, right, up, down, and counter clock-wise for the diagonal elements. After that, the function follows the minimum direction, store the current cell index, and repeat the last step for the new cell till it gets to the goal. Figures 5 to 8 show the wavefront value map and the corresponding trajectory for 4 different maps, the one provided in the assignment, maze, mazeBig, and ObstaclesBig.

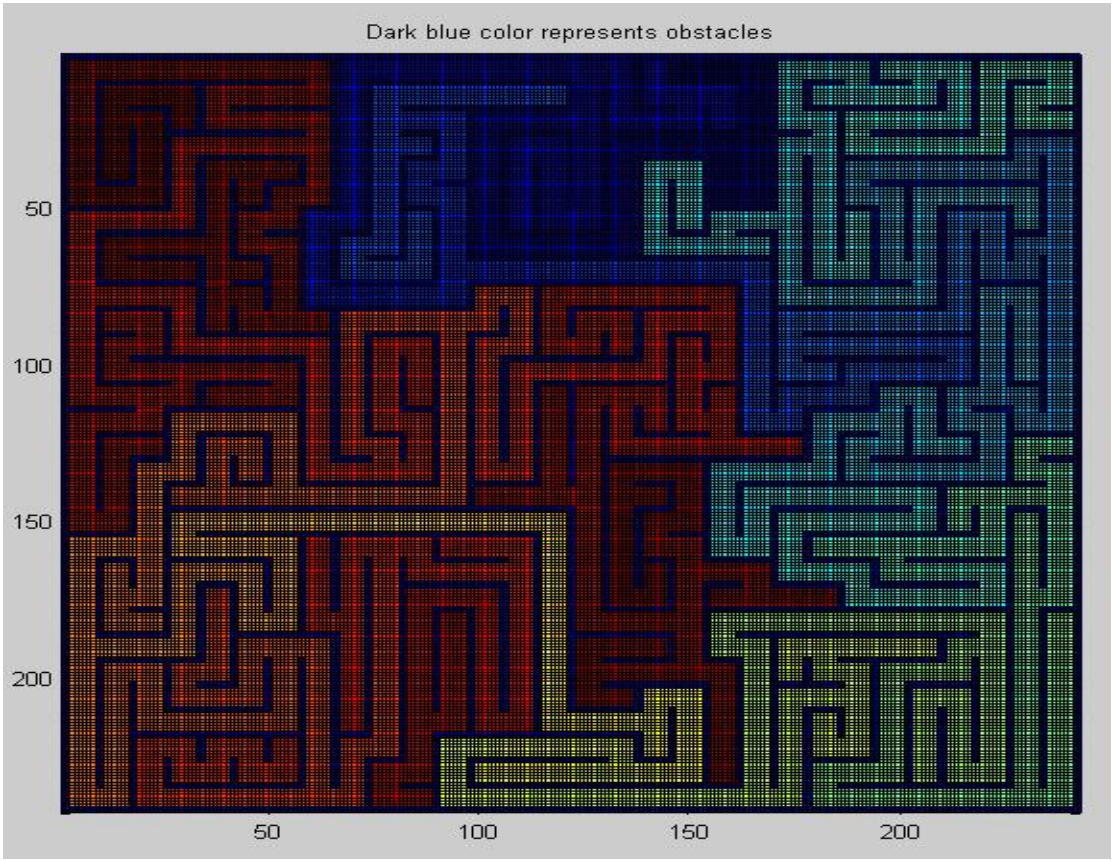


(a)

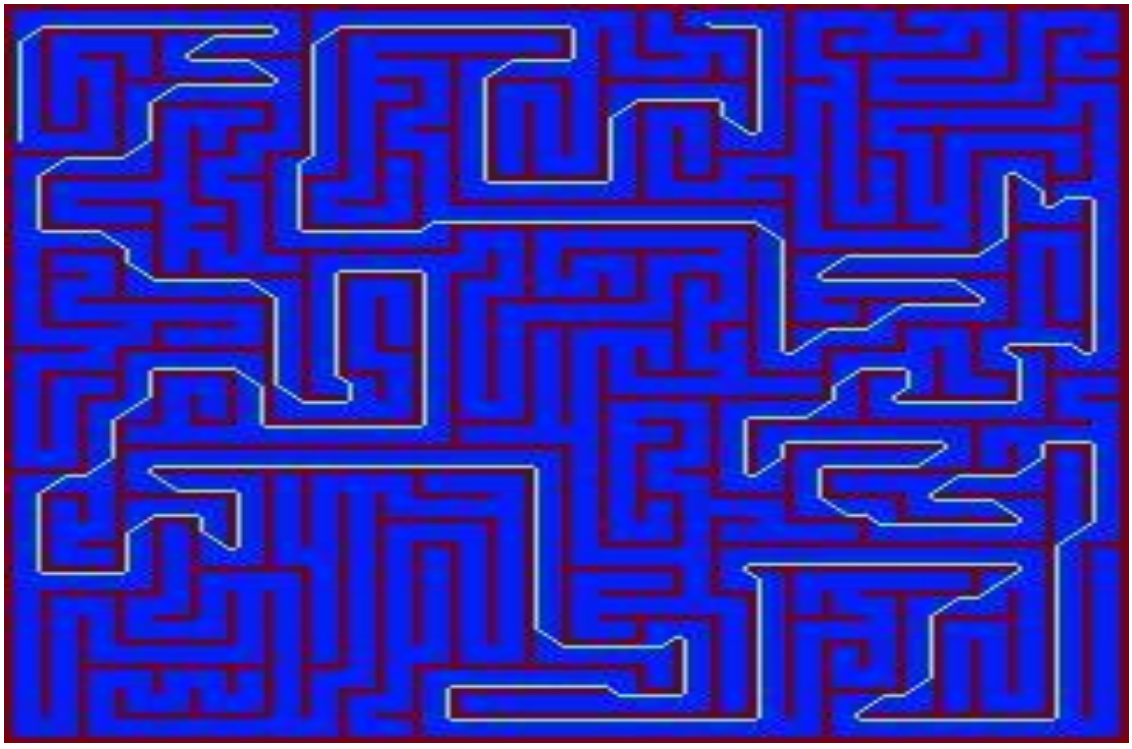


(b)

Figure 5(a,b): wavefront value and trajectory maps



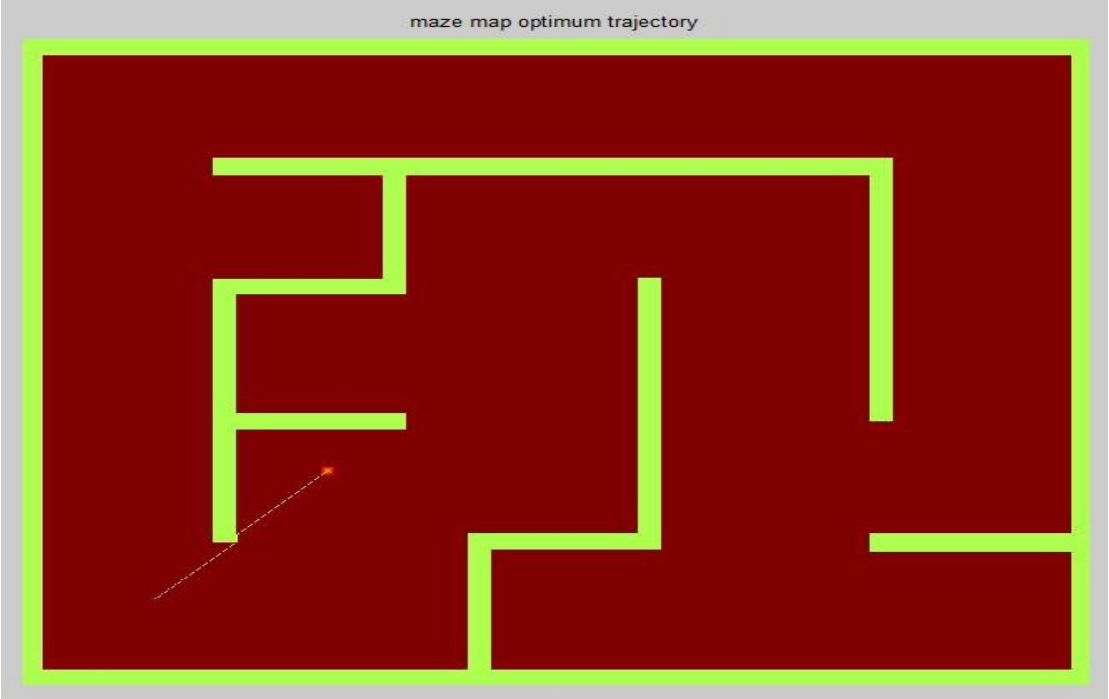
(a)



(b)

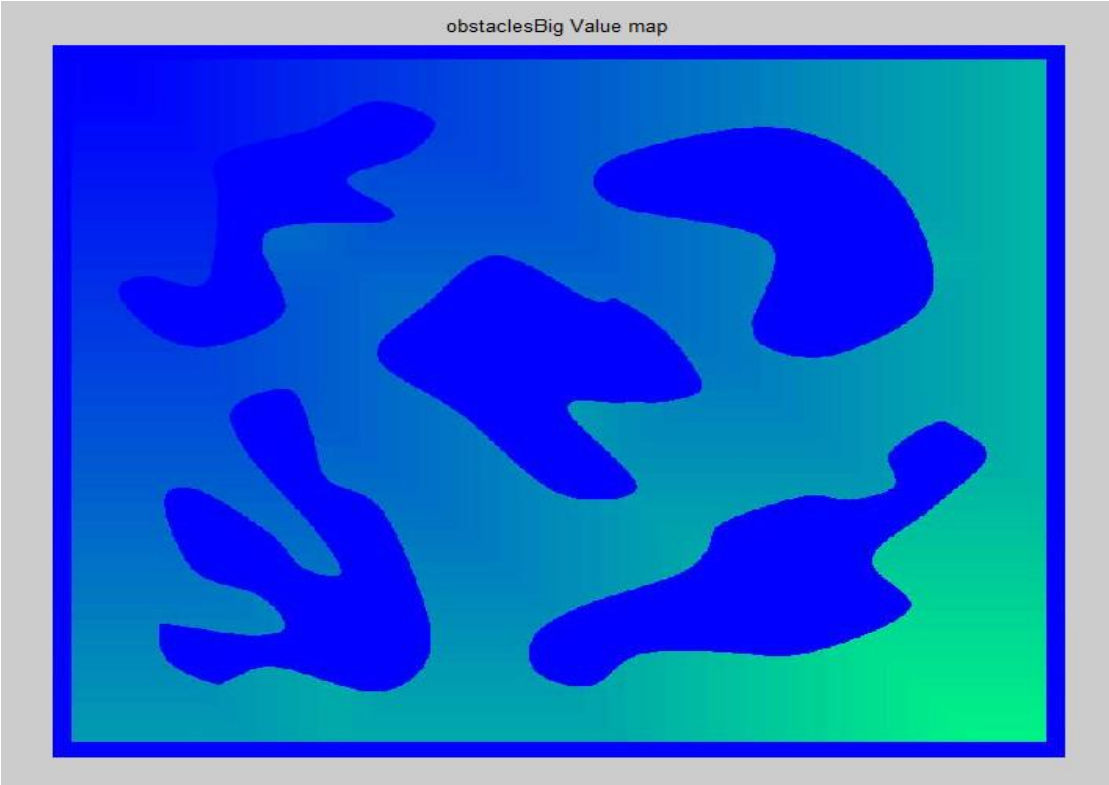
Figure 6(a,b): wavefront value and trajectory maps

(a)

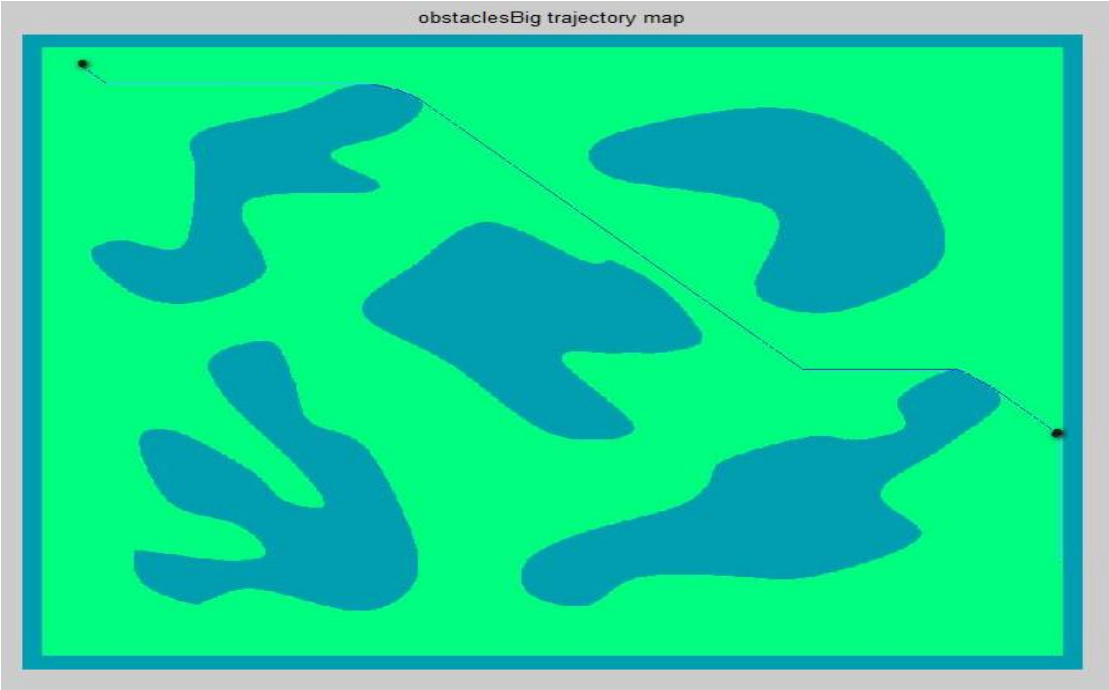


(b)

Figure 7(a,b): wavefront value and trajectory maps



(a)



(b)

Figure 8(a,b): wavefront value and trajectory maps

The following table gives information about the time response of the algorithm for the four tested maps.

Map's name	*Map's size*	*Time response(s)*
The given map in the assignment	14*20 cells	0.605 s
Maze	243*243 cells	70.932 s
mazeBig	802*687 cells	211.581 s
ObstaclesBig	803*683 cells	225.447 s

Table 1: Wavefront algorithm time response

It is evident from the table that increasing the map size implies an increase also in the time response of the algorithm. The reason for this increase is the higher number of iterations and calculations being computed during the run time. Furthermore, the time response depends on the location of both the goal and the start points as well, further distance suggests higher time response.

4. Conclusion

During this lab sessions, some path planning algorithms based on potential functions, brushfire and wavefront, are analyzed, designed, and implemented successfully in Matlab program. Approximately, half of the time spent for having the first version of both algorithms running and the second half for optimization part. Indeed, big maps need full attention during testing because any small error means running the algorithm all over again and some of the defects in the algorithm are highly possible to be detected when testing it on those maps.