

UNIVERSITAT DE GIRONA

AUTONOMOUS ROBOTS

LABORATORY REPORT

---

# Potential Functions

---

*Author*  
Mohit VAISHNAV

*Supervisor*  
Dr. Perez MARC  
CARRERAS

March 5, 2018



## 1 Introduction

Autonomous robots have to develop the capacity of navigation and avoid any kind of obstacles. In some case robot has the complete knowledge of environment while in few others just the idea about the goal. Hence path planning is required to take the robot from one position to a particular location. One such algorithm is known as *PotentialFieldMethod*.

This lab was meant to understand techniques of path planning based on potential functions. Herein we have implemented the algorithm namely *Bushfire* and *Wavefrontplanner*.

## 2 Environment

According to the lab sheet given, we have to design the functional implementation of these two algorithms and test them on different environmental setup with varied size of matrices already provided. All the implementation has been carried out in *MATLab* setup.

First section of the implementation concerns with the *Bushfire* algorithm which works on the repulsive potential method. Output of this function returns the *ValueMap* which are labelled as the potentials gradient across the matrices circumferencing the given matrix following with we can know of the path with the minimal value.

Moving on to the next section, we are required to implement *Wavefront* planner algorithm to compute the optimal path towards the goal.

## 3 Implementation

### 3.1 Bushfire

After struggling to find the optimum algorithm which could run fast enough to run on any kind of environment. I have created an iterating loop having the range of 1000 (assuming there could be at most 1000 last value loop).

The loop is starting from an initial position (1, 1) and connectivity of 8 point. To check the condition of infinite loop, I have kept a flag value with the number as the number of 0 available in the matrix. Once the iterator checks the value being equal to flag value (*Tempcount* in the code) it comes out of loop using *break* command.

### 3.2 Wavefront

Implementing this program was a tricky one because I had the starting position from which potential gradient has to be developed. During implementing I have designed two type of approach; while one was using just same as the above *brushfire* way where I divided the matrix in 4 different parts where a wave kind of gradient values were allotted to each pixel value.

In the second approach which is dynamic one, here the list is maintained, checking each individual value and assigning a flag value in that logical matrix, so that it is not repeated again. During the process I am also saving the path which is to be traversed to reach to goal point.

### 3.3 Results

I have substantially optimized the code so that it could run without any ample lag. Below mentioned Table shows the run time in seconds for all the size of maze for different algorithms.

### 3.4 Output

Run Time for Different Size Matrices (in sec)				
	Small	Maze	BigMaze	ObstacleBig
<b>Brushfire</b>	0.005	1.282	21.45	21.42
<b>Wavefront</b>	0.043	0.700	-	-

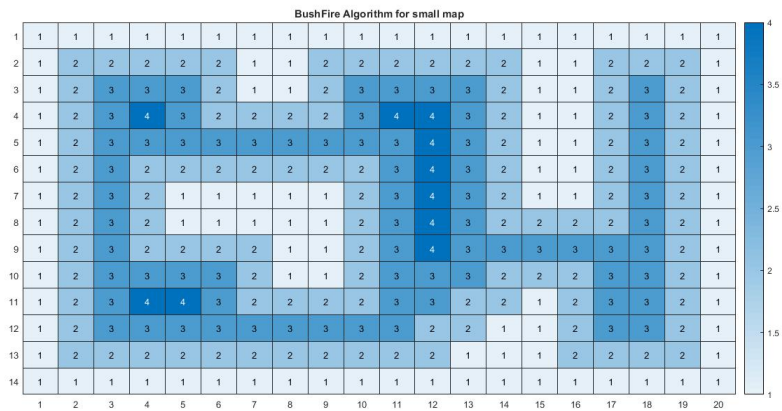


Figure 1: Small Matrix BrushFire and Path Planning

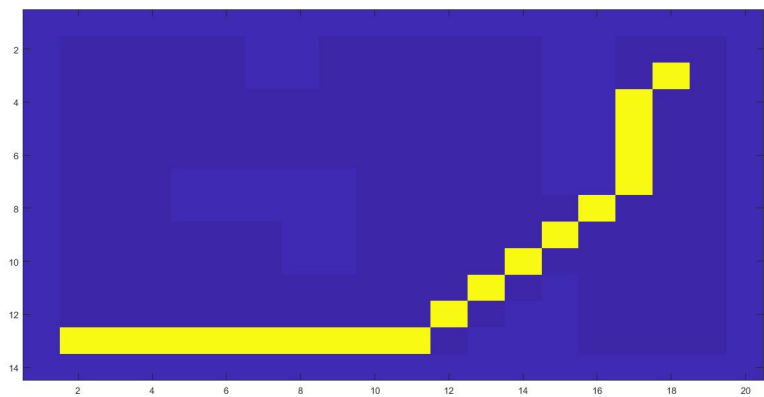


Figure 2: Path Planning for Small Matrix

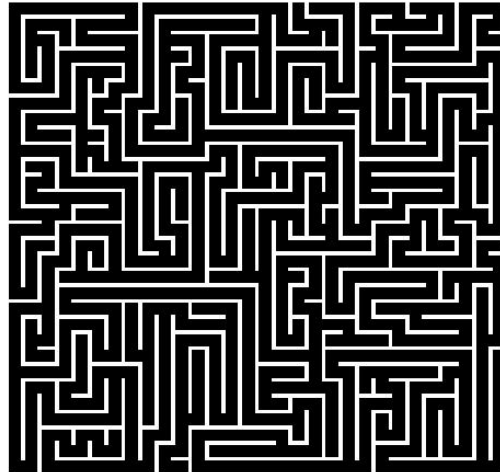


Figure 3: Maze Statement



Figure 4: Path given the initial and final position



Figure 5: Big Maze Matrix

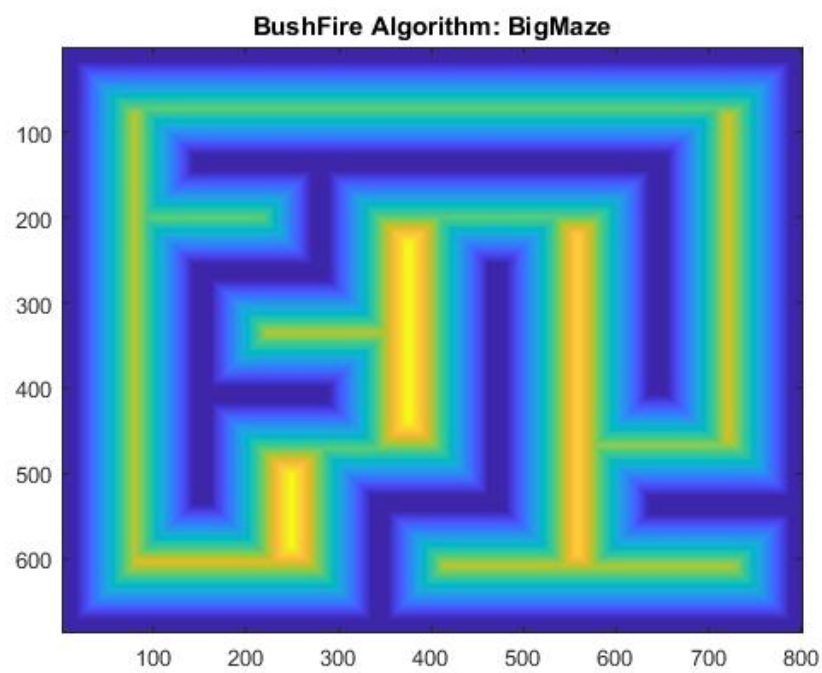


Figure 6: BrushFire display for the Big Matrix



Figure 7: Big Obstacle Image



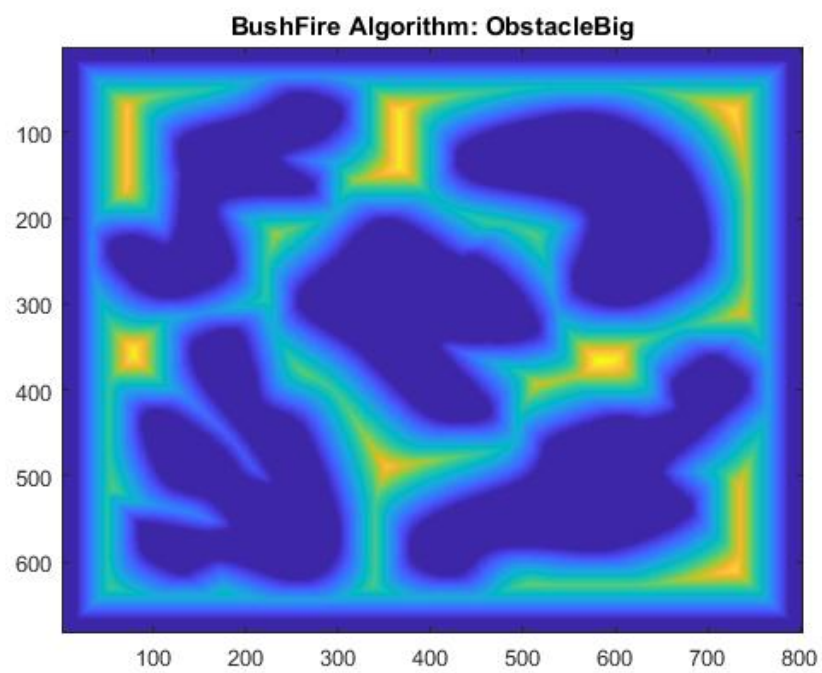


Figure 8: Big Obstacle BrushFire output