Path planning in robotics: the Wavefront algorithm

Autonomous robots
Master in Artifitial Intelligence

Pedro Frau

1. Introduction

For robots, moving in the real world can be a really tough task to perform. First of all they do not always know where their objective is, and secondly they do not know which is the better way to avoid an obstacle and find the optimal path to their goal. Along this document we will be developing the wavefront algorithm for path planning. The idea is to implement a path planner that, given a map of a scenario with obstacles, a known initial state and a known goal position, is capable of finding the optimal path from the source to the goal so that the robot can move avoiding the obstacles.

2. Overview and code explanation

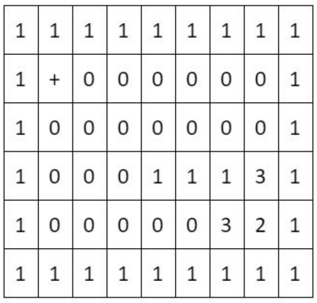
Now, with respect to the algorithm, wavefront implements a way of treating paths on a grid depending on the distance costs to the goal. In other words, each cell of the grid has a cost depending on its minimum distance to the goal and wavefront tries to find the "cheapest" way from the source point on that grid.

Given a map of the scenario in which the robot will be moving and being aware of all the obstacles and the goal position, wavefront creates a "wave" of costs iteratively from the goal to the source. Graphically, we would represent the world as a two-dimensional matrix where a 0 represents an empty space, a 1 represents an obstacle and a 2 represents the goal.

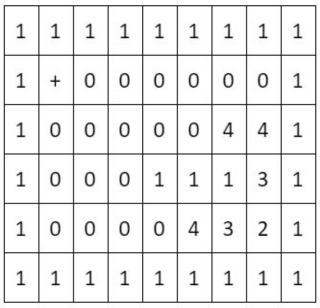
This would be the representation of world 1 that we used for the assignment (we will represent the source with the "+" sign):

1	1	1	1	1	1	1	1	1
1	+	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1
1	0	0	0	1	1	1	0	1
1	0	0	0	0	0	0	2	1
1	1	1	1	1	1	1	1	1

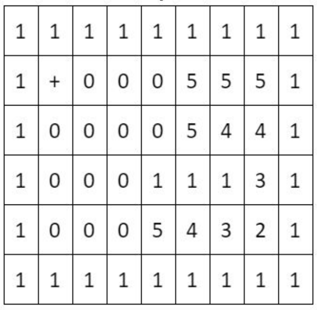
This means that our source is in point (2,2), and our goal is in point (5,8). Then iteratively, we would fill the goal adjacent cells with their cost i.e. current cell + 1:



1	1	1	1	1	1	1	1	1
1	+	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	1
1	0	0	0	1	1	1	3	1
1	0	0	0	0	0	3	2	1
1	1	1	1	1	1	1	1	1



1	1	1	1	1	1	1	1	1
1	+	0	0	0	0	0	0	1
1	0	0	0	0	0	4	4	1
1	0	0	0	1	1	1	3	1
1	0	0	0	0	4	3	2	1
1	1	1	1	1	1	1	1	1



1	1	1	1	1	1	1	1	1
1	+	0	0	0	5	5	5	1
1	0	0	0	0	5	4	4	1
1	0	0	0	1	1	1	3	1
1	0	0	0	5	4	3	2	1
1	1	1	1	1	1	1	1	1

And the final result would be the next one:

1	1	1	1	1	1	1	1	1
1	9	8	7	6	5	5	5	1
1	8	8	7	6	5	4	4	1
1	8	7	7	1	1	1	3	1
1	8	7	6	5	4	3	2	1
1	1	1	1	1	1	1	1	1

Then, starting from the source, we would check the cost of each cell and selecting the "cheapest" one as the current position at each iteration till we reach the goal. So the robot would be able to move up/down, right/left and in diagonal upon the matrix.

Now regarding the code, first of all let us do some initializations (the code has been written in matlab):

```
%%%%%%Initializations%%%%%
[matrix, sourcepos, goalpos] = matrix4 %Select the world
currentpos = sourcepos;
cost = zeros(size(matrix));%We initialize a cost matrix
%We store the position of goal while we study its adjacent cells
m1 = [goalpos(1) goalpos(2)];
cost(goalpos(1), goalpos(2)) = 2;%We give the goal a cost of 2
% We initialize a matrix of immediate adjacents
adjacent = [ 1 0; 0 1; -1 0; 0 -1;1 1;-1 -1;1 -1;-1 1];
display = true;
```

As we can see above, we start by storing the world information retrieved from the chosen world. Then we initialize the current position variable that will be changing along the execution and a cost matrix that we initialize with zeros. Finally, we store the position whose adjacent cells will be studied, we set the value of goal cell to 2 and we create an array of adjacent positions.

Then we can start with the wavefront matrix filling:

```
%%%%%Start of the wavefront algorithm%%%%%
while size(m1,1) ~= 0
  % For each cell adjacent to the current one check its particularities
  for k=1:size(adjacent,1)
    % Calculate index for current adjacent cell:
    adj = m1(1,:)+adjacent(k,:);
    % Check the existance of the adjacent cell in the map
    if min(adj) < 1
      continue
    end
    if adj(1) > size(cost,1)
      continue
    end
    if adj(2) > size(cost,2)
      continue
    end
    % Check if the adjacent cell is an obstacle
    if matrix(adj(1), adj(2)) == 1
      continue
    end
    % Check if the adjacent cell hasn't been already visited
    if cost(adj(1), adj(2)) ~= 0
      continue
    end
    % Set the cost and add the adjacent to the m1 set
    cost(adj(1), adj(2)) = cost(m1(1,1), m1(1,2)) + 1;
    m1(size(m1,1)+1,:) = adj;
  end
  m1 = m1(2:end,:);
end
```

Here, we check for each adjacent cell its existence in the map to avoid out of bounds exceptions and we check the obstacles position. Finally, we check if the cell has been already visited or not and set the cost to the cell. This part of the code has been extracted from https://gist.github.com/ipeet/1740750.

Finally, we start at the source position and we update robot's path checking that each new waypoint has a lower cost than the previous one and we update the cost of the cell with a random variable which will mark all the way. Here we can see a part of the code:

```
while goal>2
    %Check every cost of adjacent cells and mark the cell as visited if it
    %belongs to the final path
    if matrix(currentpos(1),currentpos(2))>matrix(currentpos(1),currentpos(2)+1)...
          && matrix(currentpos(1),currentpos(2)+1)>1
       currentpos = [currentpos(1), currentpos(2)+1];
       goal = matrix(currentpos(1),currentpos(2))
       matrix(currentpos(1),currentpos(2)-1) = 40;
```

```
...
```

3. Results

Now that the algorithm has been explained, let's see how are the results we get. We have 4 different worlds called matrix1, matrix2, matrix3 and matrix4. To use them we just need to call them in the first line of the wavefront.m. Then we just need to run that file and we will get an image showing the world with its obstacles and the goal. If we press any key, we will get the following one showing the world with all the found costs. Finally, if we press again any key, we will see the found path from the source to the goal.

World 1:

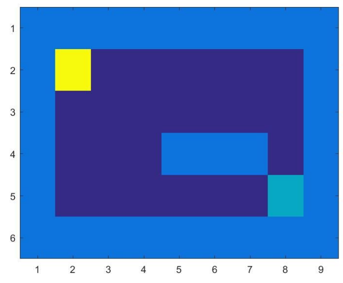


Image 1: Map of matrix 1. At the top left corner, the source. At the bottom right corner, the goal.

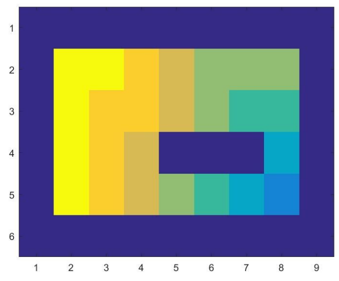


Image 2: Ascending costs in matrix 1 from goal to source.

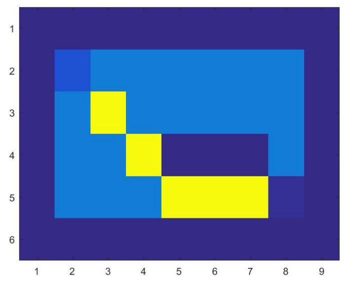


Image 3: Found path in matrix 1.

World 2:

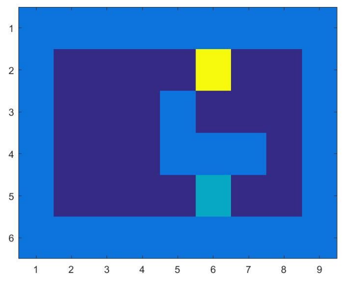


Image 4: Map of matrix 2. At the top, the source. At the bottom, the goal.

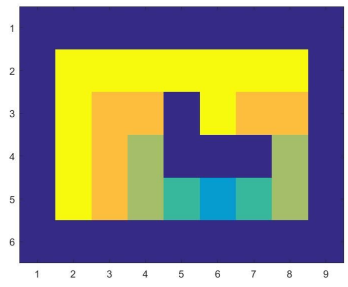


Image 5: Ascending costs in matrix 2 from goal to source.

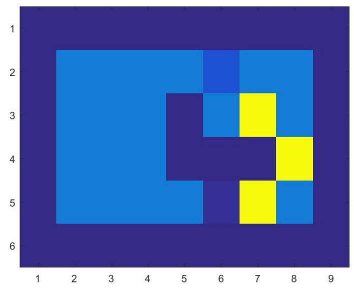


Image6: Found path in matrix 2

World 3:

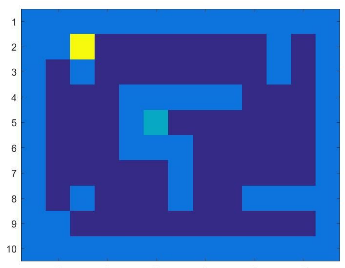


Image 7: Map of matrix 3. At the top left corner, the source. At the center, the goal.

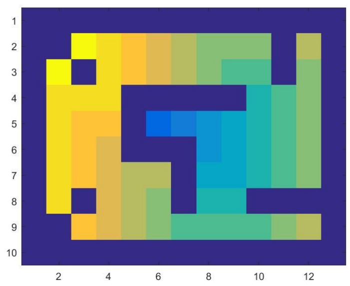


Image 8: Ascending costs in matrix 3 from goal to source.

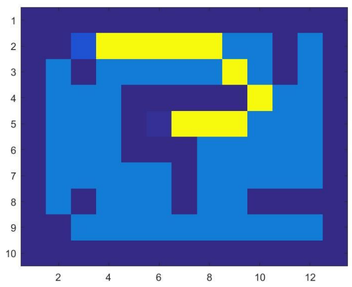


Image 9: Found path in matrix 3.

World 4:

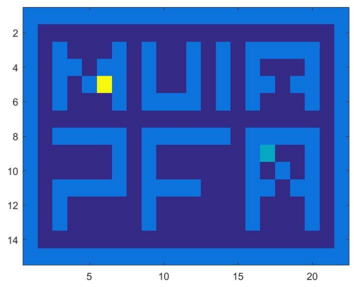


Image 10: Map of matrix 4. Under the "M" of "MUIA", the source. On the "A" of "PFA", the goal.

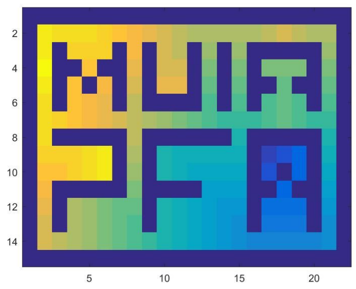


Image 11: Ascending costs in matrix 4 from goal to source.

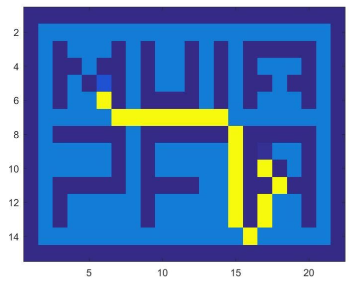


Image 12: Found path in matrix 4.

The same information can be found in the matrix format while executing the algorithm on matlab.

4. Conclusions

Wavefront algorithm is just one of many algorithms used for path planning. The main characteristic of the wavefront algorithm that may be interesting to us is its simplicity. It is easy to understand how this method works and its behaviour is easy to implement. In addition, it produces very good results in terms of optimization of the path cost.
In my opinion, it has just a few problems. For example characterizing the obstacles instead of just representing them with ones may result in a failure of the algorithm. For example in the case in which we give a coloured aerial photo of a room, the algorithm would not be able to differentiate all the object colours from the floor and empty spaces colours.
Plus, this algorithm does not involve any robot's sensors interaction which may help the robot and correct any possible calculus error the algorithm might return.

Finally, the wavefront does not take into account distances from the robot to the obstacles and robot dimensions. This means that the same path might be good for a tiny robot but might result in a crash for a big one.