ME 5751 - Module 2: Brushfire Algorithm

Johanna Martinez-Felix
California Polytechnic University
Mechanical Engineering Dep.
Pomona, CA
johannafelix@cpp.edu

***Abstract*— In this module, cost maps are drawn/generated using the windows paint application. The maps are saved in the maps folder and loaded by name in the python script to be displayed in the GUI. Then the brushfire algorithm is applied through breadth first search. The purpose of applying the brushfire algorithm is to count the distance from an obstacle to free space in order to eventually achieve motion planning.**

Keywords*— *Cost map, brushfire algorithm, breadth-first search, obstacle, free space, motion planning

I. GRID-BASED REPRESENTATION AND PLANNING METHODS

This exercise uses numerical values that represent the "cost" of navigation for the robot. This is also represented with color shades from black to white. The color black is considered a hard obstacle and is given number value "0". The color white is given value "255" and represents free space. In between those two values we have shades of gray represented. The cumulation of these map characteristics help form our definition for the grid-based representation and helps apply the brushfire algorithm to determine the max distance from obstacles to free space.

The planning methods to turn a grid or the map of pixels we have into a graph, we must decide whether the centers of the grid cells are 4-connected or 8-connected. Four connected grid cells mean that an edge exists between two free grid cells if they are directly left, right, up, or down of each other. Eight connected grid cells means that neighboring free grid cells along a diagonal are also connected. The center of each free grid cell is considered to be a node of the graph, and edges are between the 4- or 8-connected free grid cells.

II. BRUSH FIRE PROCEDURE TAKEN

For the application brushfire the 4-connect method was taken. The test map is the grid on an x-y axis and implemented as shown in figure 1.

```
direc=[[0,-1],[1,0],[0,1],[-1,0]]
```

Figure 1 Four connect method, described the neighbors to the left, right top and bottom of the zero-value pixel.

When a zero-value pixel is found it gets stored in to queue. Then from the location of the zero-value pixel it begins to search for other obstacles; it looks at the neighbors. When the algorithm has completed going through the surrounding pixels of the same distance it pops queue to move on the next set of neighbors, a larger distance from the obstacle. When doing so the search leaves behind a measured value of distance to the initial obstacle. Figure 2 shows the implementation of queue.

```
q = []
for i  in range(len(self.map)):
    for j in range(len(self.map[0])):
        if self.map[i][j] == .0:
            q.append((i,j))

q.append([-1,-1])

rows=len(self.map)
cols = len(self.map[0])

if len(q) == 1 or len(q) == rows*cols+1:
    return

dist=1
while(len(q)>0):
    [i,j]=q.pop(0)
    if(i==-1):
        if(len(q)>0):
            q.append([-1,-1])
            dist+=1
```

Figure 2 Queue is represented as "q" it enables the growth effect of the search from pixel to pixel from the obstacle.

III. MODULE EXPERIMENTATION

Three 500x500 pixel maps were self-created to implement brush-fire algorithm.

Map one is displayed as figure 3. It's a simple obstacle course with different filled in shapes randomly placed. Its worth testing due to the randomness of the locations and the variety of obstacles both in size and number in existence.

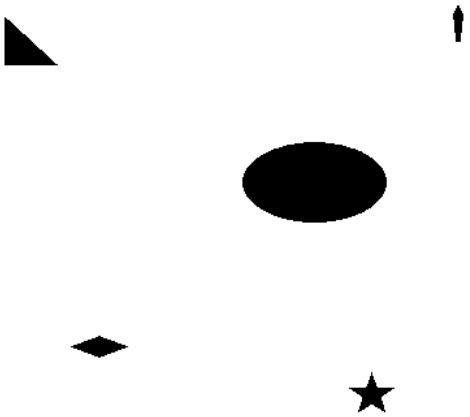

Figure 3 Test map 1.

Test map 2 shown in Figure 4. This is meant to be a good test map because it is to act as a path, to test how well the distance measurement stays centered in between two barriers.

Figure 4 Test map 2

Test map 3 is shown in Figure 5. This map was created for the purpose of observing the distance measurement of the inside of hollow shapes.

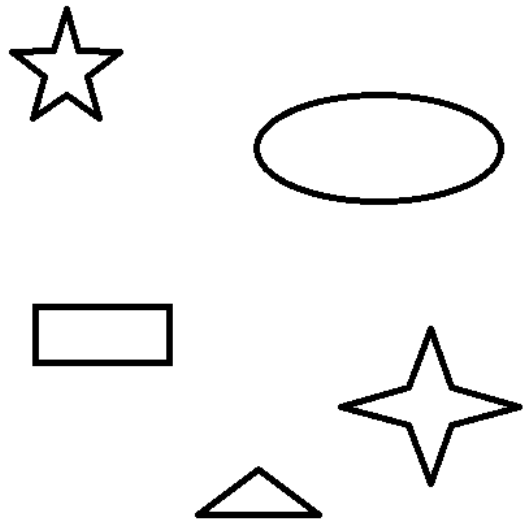

Figure 5 Test map 3

IV. Distance Map

The distance map was generated in the "compute_costmap" function after the distance of obstacles in the maps were found. (Figure 6)

```
self.maxDistance()
np.savetxt("Log/distmap.txt",self.distmap)
```

Figure 6 generating distance map

The potential function for distance used is shown below in figure 7.

```
for k in direc:

        if(i+k[0]<0 or i+k[0]==rows) or
(j+k[1]<0 or j+k[1]==cols) or
self.distmap[i+k[0]][j+k[1]]!=255.0:

        continue
        else:

        self.distmap[i+k[0]][j+k[1]]=dist

            q.append([i+k[0],j+k[1]])
```

Figure 7 Function to find distance.

If done correctly the numbers in the distance map log generated should change to show a range from 255 to 0; from free space to obstacle.

REFERENCES

[1] R. Siegwart, I. Nourbakhsh, D. Scaramuzza, Introduction to Autonomous Mobile Robots, 2nd ed., The MIT Press : Cambridge, 2011. pp 57 – 98.

[2] Y. Chang, Robotics Motion Planning , Lecture Note Set #9. CPP: ME5751, 2022.

.