# Motion Planning Workshop
# Introduction to Perception and Robotics

Sourava Prasad Mishra & Amrit Kumar
Mosig
Ensimag-Ujf

April 17 2013

## 1 Introduction

The objective of this project is to implement two autonomous path finding algorithms from a source to a destination point in a discrete environment with obstacles. As demanded, we present the following two solutions detailed in the next sections :

1. Discrete numeric navigation function with gradient descent
2. A* algorithm

The implementation provides a user with results from both the above cited methods. A specific data structure is conceived to store the workspace upon which the two motion planning schemes have been built.

The general working of the presented motion planning program is as follows :

1. Firstly, an instance of the data structure is created.
2. Then the user is asked to enter various workspace parameters at run-time.
3. The workspace is initialized (with random obstacles) along with a check on the source and destination coordinates, for if they are over an obstacle.
4. The navigation function is used to compute the distance matrix using wavefront algorithm **??**. The distance matrix stores the distance of all the cells from the destination.
5. An optimal path, if exists is calculated from source to destination using the previously obtained distance matrix and printed on the standard output.
6. Similarly a path is found using A* algorithm and printed on the standard output.

## 2 Data Structure

Proper storage and programmatic representation of the workspace is an important aspect in the context of motion planning. We are using a C programming language struct **??** to encapsulate the workspace and further typed with typedef. The structure is named Workspace, typed as workspace and it contains :

– the dimensions of Workspace as integers.
– source and destination coordinates are of type integers.
– 2 two-dimensional arrays, for storing the cell-type of the workspace and the distance matrix.
The convention used to represent workspace is : 0 for empty cells and $-1$ for obstacles. The placing of obstacles in the workspace is a random process and consists of little less than 50% of all the cells in the workspace. However this can be changed if desired, in the function input() of util.c program.
– two additional arrays for storing the $(x, y)$ coordinates of the cells forming the path from source to destination.

```
struct  Workspace {
        int nbrows, nbcolumns, sx, sy, dx, dy;
        int **grid, **distance;
        int *pathx, *pathy;
};
typedef struct Workspace workspace;
```

Code 1 – Workspace struct

# 3 Navigation Function & Gradient Descent

The discrete navigation function is used to obtain the distance matrix previously discussed. Breadth first search (BFS) traversal algorithm **??** is used for the purpose. The space complexity of BFS is O(|V|+|E|), where V and E are set of vertices and edges respectively.

Once the distance matrix is built, gradient descent algorithm is applied to find the nearest cell to the current cell and is added to the path. The algorithm starts with the source cell and terminates when the current cell becomes the destination cell.

## 3.1 Calculating Distance Matrix

The iterative version of the wavefront algorithm is implemented which resembles BFS. This is done in the void computeDistance(workspace* w) function in navigation.c program. The pseudo-code is presented in **??** :

**Algorithm 1**: Compute Distance Matrix

```
   Data: workspace :=(grid, distance, source, destination)
   Result: Compute the distance matrix
 1 Queue Q := φ;
 2 Enqueue(Q, destination);
 3 distance[destination]=0;
   /* mark destination as visited                                    */
 4 grid[destination]←1;
 5 while Q is not empty do
 6 |   u ← Dequeue(Q) ;
 7 |   for v adjacent to u, free and not visited do
 8 |   |   Enqueue(Q,v);
   |   |   /* mark v as visited                                     */
 9 |   |   grid[v] ← 1;
10 |   |   distance[v] ← distance[u]+1;
```

# 4 A* Algorithm

Our final objective was to implement A* search algorithm **??**, which is a fairly straightforward popular algorithm. The usual *Manhattan distance* is used to estimate the heuristic distance.

# 5 Execution of source code

The motion planning workshop code which is submitted along with this project have been tested on ensibm server at ENSIMAG. The program, however can be executed on any standard Linux terminal by typing './motionplanning'.

A sample run :

```
 machine@machine:~$./motionplanning
 Enter the size of the grid:5
5

 Enter source coordinate(for x, y >= 0):0
2

 Enter destination coordinate(for x, y >= 0):2
2
The destination or the source is at an obstacle
The source is (0,2)
The destination is (2,2)
```

```
Displaying the workspace:
 0 -1  0  0  0
 0  0 -1  0 -1
 0 -1 -1 -1  0
-1  0 -1  0  0
 0  0  0  0 -1
```

During the execution of the program, placing of obstacles happen at random. In the above case, the destination happens to be over an obstacle and hence, the program terminates after displaying the workspace. In such a case, the user can choose to re-run with a different set of values for the workspace.

A sample program output when the source and destination are not over obstacles :

```
 machine@machine:~$./motionplanning
 Enter the size of the grid:7
8

 Enter source coordinate(for x, y >= 0):3
3

 Enter destination coordinate(for x, y >= 0):2
2
The source is (3,3)
The destination is (2,2)

Displaying the workspace:
 0  0 -1  0  0 -1 -1 -1
 0  0  0  0  0  0  0 -1
 0  0  0 -1 -1  0 -1  0
-1  0  0  0  0 -1  0  0
-1  0  0  0  0 -1  0  0
 0  0 -1  0 -1 -1  0  0
-1 -1  0  0 -1  0 -1 -1

After applying the wavefront algorithm
----------------------------------------
Displaying the distance matrix:
 4  3 -1  3  4 -1 -1 -1
 3  2  1  2  3  4  5 -1
 2  1  0 -1 -1  5 -1  0
-1  2  1  2  3 -1  0  0
-1  3  2  3  4 -1  0  0
 5  4 -1  4 -1 -1  0  0
-1 -1  6  5 -1  0 -1 -1
```

```
Displaying the path found from source to destination
Starting from -> (3, 3, [2]) -> (3, 2, [1]) -> (2, 2, [0]) -> Goal Reached !!

Result of A* algo
Printing the path now in reveser order
(2,2) -> (3,2) -> (3,3)
```

«««< HEAD The zeros except at the destination cell indicate that the cells are not reachable.

We provide here three more *interesting* test cases, which convinced us of the correctness of the implementation : ======= We provide you with few more interesting test cases, which convinced us about the efficient working of out program : »»»> 0419a0eae05e35a78ea633672d3b7397867c406

```
TEST CASE 1
-----------
machine@machine:~$./motionplanning
The source is (0,1)
The destination is (2,2)

Displaying the workspace:
 0  0  0  0  0
 0 -1 -1 -1  0
 0 -1  0  0  0
 0 -1 -1 -1  0
 0  0  0  0  0

After applying the wavefront algorithm
----------------------------------------
Displaying the distance matrix:
 8  7  6  5  4
 9 -1 -1 -1  3
 10 -1  0  1  2
 9 -1 -1 -1  3
 8  7  6  5  4

Displaying the path found from source to destination
Starting from -> (0, 1, [7]) -> (0, 2, [6]) -> (0, 3, [5]) -> (0, 4, [4]) -> (1, 4, [3])

Result of A* algo
Printing the path now in reveser order
(2,2) -> (2,3) -> (2,4) -> (1,4) -> (0,4) -> (0,3) -> (0,2) -> (0,1)

TEST CASE 2
-----------
machine@machine:~$./motionplanning
```

```
The source is (0,0)
The destination is (2,2)

Displaying the workspace:
 0 -1  0
 0  0  0
 0 -1  0
 0 -1  0

After applying the wavefront algorithm
----------------------------------------
Displaying the distance matrix:
 4 -1  2
 3  2  1
 4 -1  0
 5 -1  1

Displaying the path found from source to destination
Starting from -> (0, 0, [4]) -> (1, 0, [3]) -> (1, 1, [2]) -> (1, 2, [1]) -> (2, 2, [0])

Result of A* algo
Printing the path now in reveser order
(2,2) -> (1,2) -> (1,1) -> (1,0) -> (0,0)

TEST CASE 3
-----------
machine@machine:~$./motionplanning
The source is (0,1)
The destination is (2,2)

Displaying the workspace:
 0  0  0  0  0
 0 -1 -1 -1  0
 0 -1  0 -1  0
 0 -1 -1 -1  0
 0  0  0  0  0

After applying the wavefront algorithm
----------------------------------------
Displaying the distance matrix:
 0  0  0  0  0
 0 -1 -1 -1  0
 0 -1  0 -1  0
 0 -1 -1 -1  0
 0  0  0  0  0
```

```
The source is (0,1)
The destination is (2,2)

Displaying the workspace:
 0  0  0  0  0
 0 -1 -1 -1  0
 0 -1  0 -1  0
 0 -1 -1 -1  0
 0  0  0  0  0

Path could not be found
Result of A* algo
Search failed !!
```

«««< HEAD
=======

```
TEST CASE 3
-----------
machine@machine:~$./motionplanning
The source is (0,0)
The destination is (2,2)

Displaying the workspace:
 0 -1  0
 0  0  0
 0 -1  0
 0 -1  0

After applying the wavefront algorithm
---------------------------------------
Displaying the distance matrix:
 4 -1  2
 3  2  1
 4 -1  0
 5 -1  1

Displaying the path found from source to destination
Starting from -> (0, 0, [4]) -> (1, 0, [3]) -> (1, 1, [2]) -> (1, 2, [1]) -> (2, 2, [0])

Result of A* algo
Printing the path now in reveser order
(2,2) -> (1,2) -> (1,1) -> (1,0) -> (0,0)
```

»»»> 0419a0eae05e35a78ea633672d3b7397867c4066

# 6 References


1. Wavefront algorithm, http://www.societyofrobots.com/programming_wavefront.shtml
2. Breadth first search traversal algorithm, http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/breadthSearch.htm
3. A* algorithm http://en.wikipedia.org/wiki/A*_search_algorithm