Maze Solving Algorithms for Micro Mouse

Swati Mishra
swati.mishra.07@gmail.com
Inderprastha Engineering College, Ghaziabad
Pankaj Bande
pankajb@isquareit.ac.in
International Institute of Information Technology,Pune

Abstract

The problem of micro-mouse is 30 years old but its importance in the field of robotics is unparalleled, as it requires a complete analysis & proper planning to be solved. This paper covers one of the most important areas of robot, "Decision making Algorithm" or in lay-man's language, "Robot Intelligence". For starting in the field of micro-mouse it is very difficult to begin with highly sophisticated algorithms. This paper begins with very basic wall follower logic to solve the maze. And gradually improves the algorithm to accurately solve the maze in shortest time with some more intelligence. The Algorithm is developed up to some sophisticated level as Flood-Fill algorithm. The paper would help all the beginners in this fascinating field, as they proceed towards development of the "brain of the system", particularly for robots concerned with path planning and navigation.

Keywords: Mobile Robot Navigation, Algorithms, Micromouse, Flood fill, Djikstra

1. Introduction

Autonomous agents are mobile versatile machines capable of interacting with an environment and executing a variety of tasks in unpredictable conditions. Autonomy means capability of navigating the environment; navigation, in turn, necessarily relies on a topological and metric description of the environment [6].

One of the major components for the creation of autonomous robot is the ability of the robot to "plan its path" and in general the ability to "plan its motion". In a limited or carefully engineered environment it is possible to program the robot for all possible combinations of motions in order to accomplish specific task [2]. The problem of path planning is not confined to the field of robotics but its applications exist in various genres. For example, molecule folding, assembly/disassembly problems and computer animations are areas where comparable problems arise [7].

A robot is a mechanical device, which performs automated physical tasks, either according to direct human supervision, a pre-defined program, or a set of General guidelines using artificial intelligence techniques. There is a paradigm shift in engineering education from conventional classroom teaching to hands on projects, robotic projects are very useful for students who have to deal with an open ended problem and this way their creativity is stimulated. As project incorporates wide range of engineering fields, they can convert variety of theoretical study into practice. Furthermore, they learn teamwork.

To stimulate this learning process in upcoming engineering, a wide range of robotic competitions is conducted world wide, and the most sought after among them is MICROMOUSE. It is one of the most efficient ways to inculcate the true "engineering values" in students. As mentioned above, the change in the mode of education is under revolution, as a result of which not much students are exposed to the field of robotics.

Traditional maze solving algorithms, while appropriate for purely software solutions, break down when faced with real world constraints. When designing software for a maze-solving robot, it is important to consider many factors not traditionally addressed in maze solving algorithms. Our information about the maze changes as we explore, so we must make certain assumptions about unseen portions of the maze at certain steps in most algorithms.

2. The Wall Follower Logic

2.1. Maze Interpretation

The maze in the Micromouse competition consists of multiples of an 18cmx18cm square. It consists of 16x16 unit squares. It has a center, which is the destination cell for the Micromouse. The robot has to search the entire maze and find the path that it can travel in the shortest possible time. This maze is a standard used in all the competitions held worldwide by the IEEE and other standard Institutions. So we have maintained these standards for use in our experimentation. Here we have

978-0-7695-3493-0/08 $25.00 © 2008 IEEE
DOI 10.1109/SITIS.2008.104

taken standard mazes for better comparison of efficiency of any of the logics described. The maze is interpreted in the form of a two dimensional array, with each cell represented by coordinates**.**

Rows and columns distinguish the array, rows denoted by 'R' and columns denoted by 'C' hence, the increment and decrement in R and C will signify different cells. Now, the movement of the Micromouse on the maze is purely cell wise MAPPING.

When we interpret the maze mathematically, we develop the approach towards solving a higher end algorithm that is more complicated.

For this mathematical approach, we have assumed our cells to be of dimensions 9cmx9cm.So virtually our maze is of 32x32 unit squares. Henceforth, each REAL square has been divided into two VIRTUAL squares. This solves our problem a bit!

2.2. The basic algorithm

Here, we are developing the LEFT WALL FOLLOWER LOGIC. This works on the rule of following the left wall continuously until it leads to the centre. The RIGHT WALL FOLOWER is similar; the only difference lies in the wall being followed.

The Micromouse senses the wall on the left, and follows it to wherever it leads until the centre is reached. This simple logic used for solving the maze is demonstrated by the following algorithm.

Step 1: Sense the left wall.

Step 2: If left wall present then flagl=1, if not then flagl=0.

Step 3: If flagl=1 then step 4 else turn left by 90 degrees.

Step 4: Sense the front wall.

Step 5: If front wall present then flagf=1, if not then flagf=0.

Step 6: if flagf=0, move straight else turn right by 90 degrees.

Step 7: return to step1.

2.3. Problems encountered

The biggest of its loopholes is its inefficiency to stop the execution by itself. Another of its drawbacks is the **"absence of intelligence"** in the device. It does not have the ability to detect its position and direction, and determine whether or not, during is course of path finding, it has reached the centre or not.

To overcome the above problems, the micro-mouse maze solving algorithm is modified mathematically as follows.

2.4. A mathematical approach

Here, we assume an array of 32 rows and 32 columns & the starting cell indicating 0^{th} row & 0^{th} column. If the array is denoted by M, then each cell is represented by M[R][C]. We assume the present position of array as M2 [R2][C2] , the previous position as M1[R1][C1], and the next position as M3[R3][C3].

If R2-R1>0, then it is currently moving straight, upwards through the maze array.

If R2-R1<0, then it is currently moving straight, downwards through the maze array.

If C2-C1>0, then it is currently moving rightwards, through the maze array.

If C2-C1<0, then it is currently moving leftwards, through the maze array.

I. FOR STRAIGHT MOVEMENT,
  Upwards: R3=R2+1, C3=C2
  Downwards: R3=R2-R1, C3=C2
  Rightwards: R3=R2, C3=C2+1
  Leftwards: R3=R2, C3=C2-1

II. FOR 90 DEG RIGHT TURN,
  Upwards: R3=R2, C3=C2+1
  Downwards: R3=R2, C3=C2-1
  Rightwards: R3=R2-1, C3=C2
  Leftwards: R3=R2+1, C3=C2

III. FOR 90DEG LEFT TURN,
  Upwards: R3=R2, C3=C2-1
  Downwards: R3=R2, C3=C2+1
  Rightwards: R3=R2+1, C3=C2
  Leftwards: R3=R2-1, C3=C2.

IV. FOR CHANGING THE POSITION AFTER THE ROBOT MOVES AHEAD,
  R1=R2, C1=C2, R2=R3, C2=C3

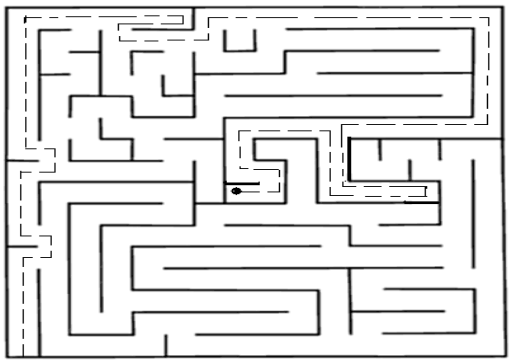

Figure1. Left wall follower: solvable maze

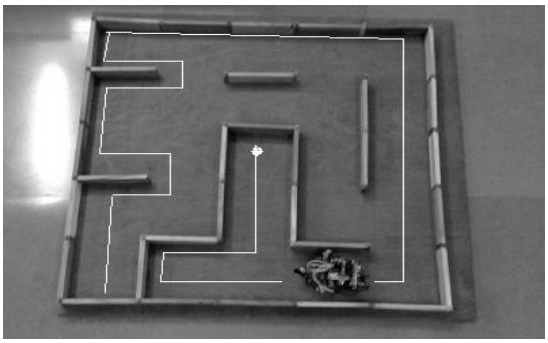

Image 1. Left wall follower sample maze solved
White line shows the path followed by robot.
The time taken by the robot can be calculated by assuming the cell to cell movement time to be 2.5 sec. and the turning time to be 1 sec. then the total time taken by the robot is (140×2.5) + (27×1) = 404sec. for reaching the centre of this maze. The above wall follower logic has been implemented on a sample realistic maze of 5x5 units.(as defined by the IEEE standards)
There are however, limitations with this algorithm, which is, that it can solve mazes of a particular style only.

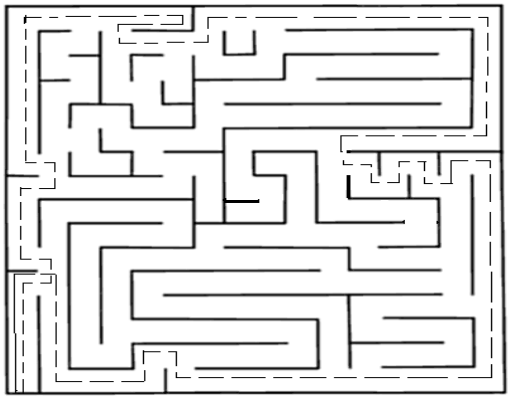

Figure2. Left wall follower: unsolvable maze
The above maze is not being solved using the left wall follower logic. This is one of the drawbacks of this logic.
The algorithm is not efficient enough to solve the mazes of high complexities and the ones which have multiple paths leading to the centre.
The following practical demonstrations prove its failure

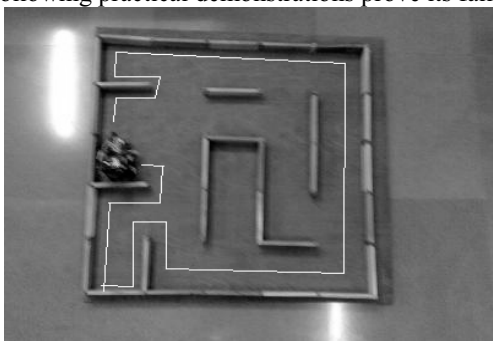

Image 2. Left Wall Follower micro-mouse in infinite loop:

3. Djikstra's algorithm

So we observe that the design of an autonomous navigation system with multiple tasks to be accomplished in unknown environments represents a complex undertaking. With the simultaneous purposes of capturing targets and avoiding obstacles, the challenge may become still more intricate if the configuration of obstacles and targets creates local minima, like concave shapes and mazes between the robot and the target. Pure reactive navigation systems are not able to deal properly with such hampering scenarios, requiring additional cognitive apparatus. Concepts from immune network theory are then employed to convert an earlier reactive robot controller, based on learning classifier systems, into a connectionist device. Starting from no *a priori* knowledge, both the classifiers and their connections are evolved during the robot navigation. [1]
As a conclusion from the above fact we now move a step ahead and implement the Djikstra's Shortest Path algorithm for solving our problem. The algorithm deals with finding the shortest path from a directed graph of a given set of nodes.
The approach adopts a strategy of multi-behavior coordination, in which a novel path-searching behavior is developed for determining the shortest path [11].

3.1. Maze Solving

The input of the algorithm consists of a weighted directed graph G and a source vertex s in G. We will denote V the set of all vertices in the graph G. Each edge of the graph is an ordered pair of vertices (u, v) representing a connection from vertex u to vertex v. The set of all edges is denoted E. Weights of edges are given by a weight function w: E → [0, ∞); therefore w(u,v) is the cost of moving directly from vertex u to vertex v. The cost of an edge can be thought of as (a generalization of) the distance between those two vertices. The cost of a path between two vertices is the sum of costs of the edges in that path. For a given pair of vertices s and t in V, the algorithm finds the path from s to t with lowest cost (i.e. the shortest path).So now this fundamental nature of the algorithm can be used to find the graph. The stepwise functioning of the algorithm has been described as below.

STEP 1: Start "ready set" with start node
Set start distance to 0, dist[*s*] =0;
others to infinite: dist[*i*]= (for *i s*);
Set Ready = { }.
STEP 2: Select node with shortest distance from the starting point that is not in Ready set
Ready = Ready + {*n*}.

STEP 3: Compute distances to all of its neighbors
For each neighbor node *m* of *n*
Check if dist[*n*] +edge (*n*, *m*) < dist[*m*]
If yes then dist[*m*] = dist[*n*] +edge (*n*, *m*);
STEP 4: Store path predecessors.
pre[*m*] = *n*;
STEP 5: Add current node to "ready set".
STEP 6: Check if any node is left, if yes goto step 2
STEP 7: end.

Now the problem arises of how to generate the directed graph G of the nodes we have talked about so far. So for this we need to get the Micromouse traverse the whole maze and generate the nodes! So the traversal function is defined as follows.
Traverse ( ):
STEP 1: Move straight
STEP 2: Check if any wall is present in front. If yes, then goto step 3 else goto step 1.
STEP 3: check if the current location is present in V, if no then add the location in the set V, Calculate the distance between the previous and the present location. Store the value in the set E, else take 180 degree turn and traverse to the previous entry of V.
STEP 4: Check if wall is present on right. If present, take a 90 degree turn left if not then take a 90 degree turn right.
STEP 5: Goto step 1
Repeat steps 1 to 5 till the entire maze is traversed.
Calculation of distance between two consecutive nodes is done by adding a counter circuit at the base of the chasis near the wheels so as to count the number of cells between the destination and the source location. This number would denote the weight of the edge E.
Each location is represented by a cell of a 16x16 two dimensional array, cell being represented by [R, C], R represents row, and C represents column. So the vertex set V consists of a pair of variables [R, C] representing a single node.

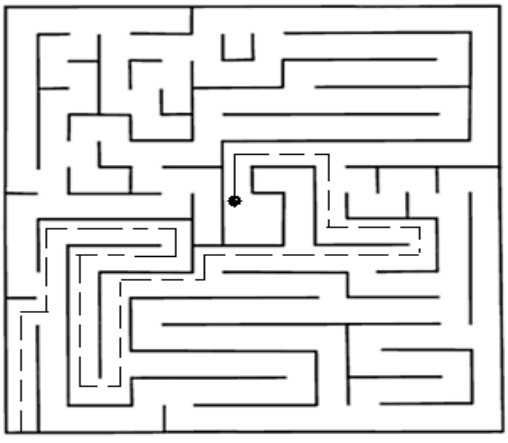

Figure 3. Maze as solved by Djikstra's algorithm

After generating the graph the final algorithm is applied on the graph and the shortest path reachable to the destination node which is the centre is obtained.

The time taken by the robot can be calculated. If we assume the cell to cell movement time to be 5 sec. and the turning time to be 1 sec. then the total time taken by the robot is (50*5) +(16*1)= 266 sec. for reaching the center of this maze.
Now we see that it effectively calculates the shortest path but to find that path it has to travel the entire maze. This traversal is to generate the directed graph G. Though the time taken to reach the centre is less but it takes more time to traverse the maze. So if we calculate the total time taken, then it would be more in this case.

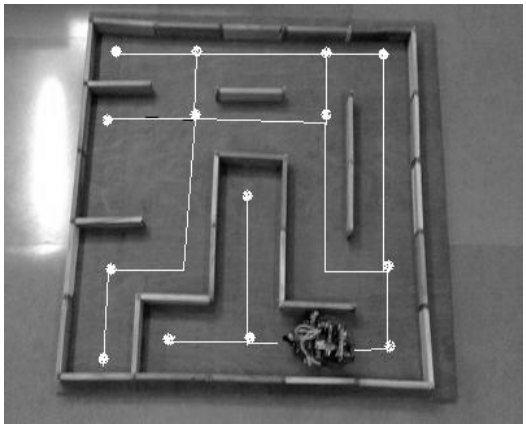

Image 3. Maze being solved by Djikstra's algorithm. White line shows the path traversed by the device and the white points show the nodes as perceived by the algorithm.

3.2 Drawbacks of the algorithm

There are, however, problems in using this algorithm, the major one being that the whole maze has to be traversed. For identifying the nodes, it is important to travel all the parts of the maze, irrespective of whether that portion of the maze contains the shortest path or not. Now, this is time consuming and also a lot of energy is wasted in the traversal. This solution also requires a lot of time for finding the shortest path as after the generation of the connected graph G, it has to check for all the connections that lead to the centre. This increases the execution time of the algorithm. Even for smaller mazes it will have to travel all the blocks before starting to find the shortest path. This problem would be solved if we can design a way where both the maze interpretation and path finding are done simultaneously.
The other problem is that for counting the number of cells to generate the edge set E, an additional hardware is involved which includes a counter circuit that counts the

rotation of the wheels and hence the distance between two consecutive locations. This adds to the complexity of the design and increases the probability of error in input data from the external environment.
To avoid such complexities, we use yet another solution to solve our problem which has been described below.

4. Flood Fill algorithm

The speed of robot to find its path, affected by the applied algorithm, acts the main part in the present project. The flood-fill algorithm involves assigning values to each of the cells in the maze where these values represent the distance from any cell on the maze to the destination cell. The destination cell, therefore, is assigned a value of 0. If the mouse is standing in a cell with a value of 1**,** it is 1 cell away from the goal. If the mouse is standing in a cell with a value of 3, it is 3 cells away from the goal. Assuming the robot cannot move diagonally [3].
The maze is represented as a 16x16 array in the memory. The centre is given the value (0, 0).all cells in its immediate vicinity are assigned 1, the cells next to it as 2, and so on. The array is divided into 4 symmetrical regions and then the assignment is done.
Upper left quarter, loop decrements the column, increments the row: R= R+j, C=C-i, i, j vary from 0 to 8.
Upper right quarter, loop increments the column, increments the row: R=R+j, C=C+i, i, j vary from 0 to 8.
Lower left quarter, loop decrements the column, decrements the row: R=R-j, C=C-i, i, j vary from 0 to 8.
Lower right quarter, loop increments the column, decrements the row: R=R-j, C=C+i, i, j vary from 0 to 8.

decr= 0, incr= 1 decc= 1, incc=0	decr=0, incr= 1 decc= 0, incc= 1
decr= 1, incr= 0 decc= 1, incc=0	decr= 1, incr= 0 decc= 0, incc=1

Figure 4. Values of the 4 variables in the 4 quadrants

So it is combined into a MATHEMATICAL EQUATION as follows:

Row increment and decrement: R-(i*decr) + (i*incr)
Column increment and decrement: C-(j*decc) + (j*incc)

Now when the assignment is done, the entire equation is as follows: (where the variables i, j vary in the loop from 0 to 8)

```
MAZE[R-(i*decr) + (i*incr)][C-(j*decc)+j*incc)]
= i+j;
```

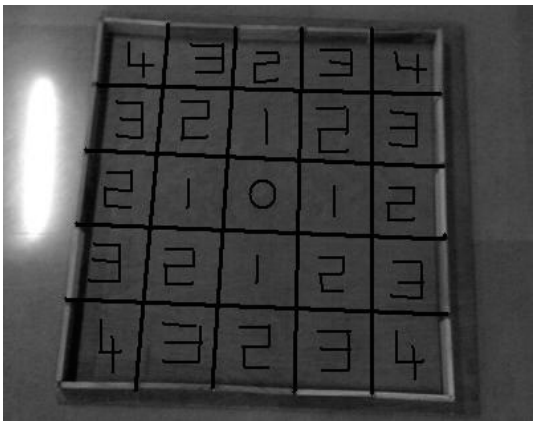

Image 4. An actual maze as depicted in the memory of the micro mouse.

Formation of array temp [4] for each cell:
Each cell is interpreted as an array cell of a 2-d array and is represented with a value, R and C, which represents a row and column, respectively. The values, therefore, of the neighboring cells are as shown in the diagram. The values of the cell arrays are as according to the index values assigned to a 16x16 array in the computer memory. Initially the value of the first cell is assigned as, R=1, C=1.

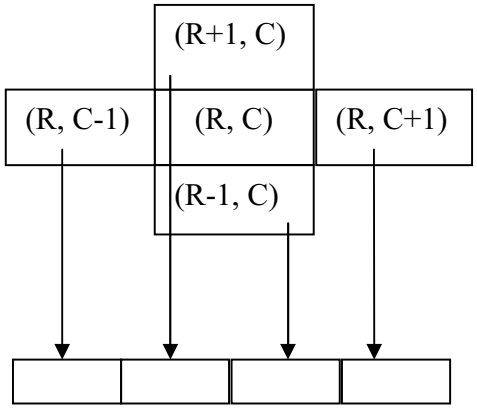

Figure 5. Storing elements in array; temp [4]

The values of the neighbors are stored in the above array. After the values are stored in the array, they are sorted using any kind of sorting. We have used here selection sort.

After this, the array is ready for further processing, which includes the deciding of which path to be taken and which values in the map to be changed.

The maze after being flooded is then traversed and the map of the maze is updated after every traversal. Every time a new cell is traversed, it creates the array described above and decides the lowest value nearby that can be

traversed. The path followed is always from a higher value to a lower value.

Main():
START: form array temp[4] for Maze[R][C].
STEP 1: From the array, select the ith element, (i=1 initially)
STEP 2: If the value temp [i]<= Maze [R][C] , then go to step 4. If temp [i]>=Maze[R][C], call check(R,C).
Step 3: if the value temp [1]=256, turn 180 deg, i=i+1,goto step 1
STEP 4: locate the cell of the value temp [i].
STEP 5: check if the wall is present in the way, if yes then, i++, go to step1
STEP 6: check if Maze[R1][C1]=temp[i+1], if yes, then use call Locate(R, C, temp(i+1)) algorithm defined below, to locate its cell.
STEP 7: store the result in (R2, C2)
STEP 8: call the function direction of move (R1, R2, C1, C2)
STEP 9: move to Maze (R', C')
STEP 10: update value, R=R', C=C'
STEP 11: check if Maze[R][C]=0, if yes, call return to start ( ), else go to START.
STEP 12: call follow ( ).

Decide which cell is preferable to move by using the following algorithm: (direction of move (R1, R2,C1,C2))
STEP 1: check which cell is obstructed by a wall
STEP 2: move in direction of no wall, if wall is present before all selected cells, then i=i+2, go to
STEP 3: give the priority to forward straight movement.
STEP 4: if the wall is in front, move towards right or left, priority can be given to any direction.
STEP 5: if it is dead end, then move in backward direction turn 180deg.
STEP 6: return the decided value in (R', C').

Location of the value in array, which is first dealt with, is found by following routine:
STEP 1: Initialize: flag1=0, flag2=0, flag3=0, flag4=0
STEP 2: Locate(R, C, temp [i])
{check if temp [i]==Maze [R+1][C], if yes
flag1=1; R1=R+1, C1=C;
check if temp[i]==Maze[R-1][C], if yes
flag2=1; R1=R-1, C1=C;
check if temp[i]==Maze[R][C+1], if yes
flag3=1; R1=R, C1=C+1
check if temp[i]==Maze[R][C-1], if yes
flag4=1; R1=R, C1=C-1 }
STEP 3: Return (R1, C1)
The assignment of flags deals with the problem that arises when more than one cell has the same value.
Check (R, C, i)
{Step 1: Maze[R][C]=temp[i] + 1
Step 2: update()
Step 3: return to step 2 of main( )}

There is an updating based on the fact that the value of the cell near the center is always less than the value of the cell away from the center. So,

For 1<R<8, and 1<C<8, Maze[R+1][C]<Maze[R][C]
Maze[R][C+1]<Maze[R][C]

For 9<R<16, and 1<C<8, Maze[R+1][C]>Maze[R][C]
Maze[R][C+1]<Maze[R][C]

For 1<R<8, and 9<C<16,
Maze[R+1][C]<Maze[R][C]
Maze[R][C+1]>Maze[R][C]

For 9<R<16, and 9<C<16,
Maze[R+1][C]>Maze[R][C] Maze[R][C+1]>Maze[R][C]

So the equation which determines the update ( ) function is as follows:

If Maze [R-(i*decr) + (i*incr)][C-(j*decc)+j*incc)] >= Maze [R-(i*decr) + (i*incr)][C-((j+1)* decc)+((j+1)*incc)], then

Maze [R-(i*decr) + (i*incr)][C-(j*decc)+j*incc)]+1 = Maze [R-(i*decr) + (i*incr)][C-((j+1)* decc)+((j+1)*incc)], and

If Maze [R-(i*decr) + (i*incr)][C-(j*decc)+j*incc)] >= Maze [R-((i+1)*decr) + ((i+1)*incr)][C-(j* decc)+((j)*incc)], then

Maze [R-(i*decr) + (i*incr)][C-(j*decc)+j*incc)]+1 = Maze [R-((i+1)*decr) + ((i+1)*incr)][C-(j* decc)+(j*incc)]
Where i, j are variables varying from 0 to 8, in loop.

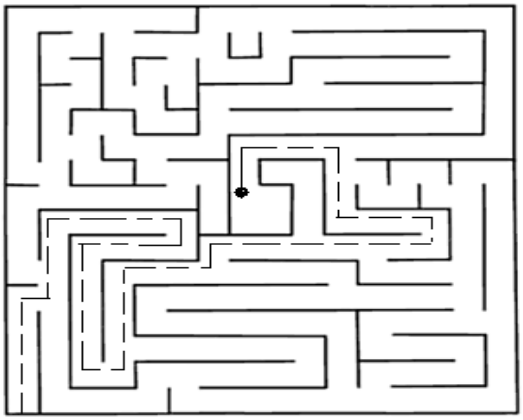

Figure 6. Maze solved through flood fill algorithm:
The above maze is solved using the flood fill logic. The

time taken by the robot can be calculated. If we assume the cell to cell movement time to be 5 sec. and the turning time to be 1 sec. then the total time taken by the robot is (50*5) +(16*1)= 266 sec. for reaching the center .

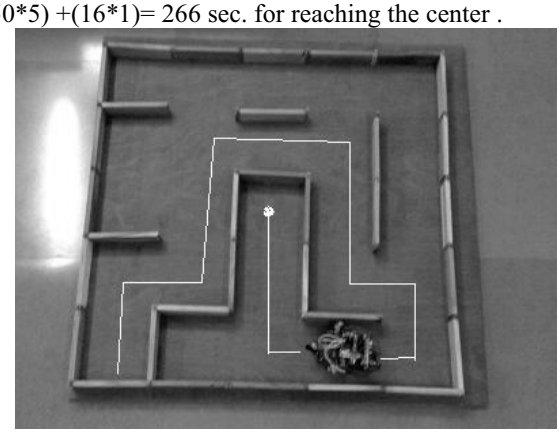

Image 5: Sample maze solved through flood fill algorithm:
White line shows the path followed by robot.

5. Result:

Motion planning is a key requirement demanded of autonomous robots. Given a task to fulfill, the robot has to plan its actions including collision-free movement of actuators or the whole robotic platform
A comparative study on the path length & time taken performance of our robot with regards to different algorithms is also done. Both simulation and real tests are performed.
The comparison of different algorithms is as follows:
Table1: Comparison between various algorithms.

	Left wall Follower	Right wall Follower	Djikstra's	Flood fill
Cell to cell movements	70	62	50	50
turns	27	25	16	16
Time taken (in sec.)	307	283	266 + extra time for graph generation	266

The problem encountered in wall follower algorithm either left wall or right wall follower is solved by Djikstra's algorithm. And this algorithm is once again refined to Flood fill. The right/left wall follower logics are restricted to a limited kind of the mazes only, while the Djikstra's algorithm can solve practically any kind of maze. But it requires a lot of time for maze interpretation and mapping which reduces its efficiency. This problem is once again reduced in the Flood fill algorithm where the maze interpretation or we can say map generation is done along with the maze solving. Both the tasks performed together improve the efficiency of the micromouse. An elaborate analysis of the above algorithms gives us a basis of how to proceed in path planning, of intelligent devices capable of navigation.
Also we have used a standard IEEE maze for experimentation. If we change the dimensions, the time of traversal may change but the algorithm will not. In case of non-uniform mazes, an extra hardware of a sensor would be required to detect whether a cell has been traversed or not. This would add to the complexity of the program. The future work of this paper gives an emphasis on this problem.

6. Conclusion:

Hence we conclude that, if we don't have any time and hardware constraint we can effectively use the Djikstra's algorithm, but if both are the constraints then Flood fill would be superior to others. Further, if we do not wish to have any complex calculation to embed in the system, that means, if we have a memory constraint as well as the maze to be solved is pretty easy, we can stick to the left\right wall follower. But for this we need to have a previous knowledge of the maze, whether it is right-walled or left-walled. Thus, the flood fill is by far the most effective of all, with fewer or almost no drawbacks apart from complex software which is difficult to code.
The speed of robot to find its path, affected by the applied algorithm, acts the main part in the present projects that are concerned with robot navigation. While there is no limitation to improve the algorithms, there are some restrictions on developing robot's mechanic or electronic. Developing algorithm is usually cheaper than the other parts.

7. Acknowledge:

The team is grateful to Prof. Rabinder Henry of International Institute of Information Technology for some useful discussions. We are also grateful to Prof. R.K Bassi, Director(academics)Inderprastha Engineering College and Prof A.K Giri, HOD(Electronics deptt.) Inderprastha Engineering College, for their kind support. The team is also thankful to friends for their useful cooperation.

8. References:

[1] Renato Reder Cazangi, Associate, *Member, IEEE*, and Fernando J. Von Zuben, *Member, IEEE* **"**Immune Learning Classifier Networks: Evolving Nodes and Connections" , IEEE Congress on Evolutionary Computation ,Canada, 2006

[2] Dimitris C. Dracopoulos;"Robot Path Planning for Maze Navigation", 1998.

[3] Babak Hosseini Kazerouni, Mona Behnam Moradi and Pooya Hosseini Kazerouni;"Variable Priorities in Maze-Sloving Algorithms for Robot's Movement", 2003.

[4] Sung-Hee Lee, Junggon Kim, F.C.Park, Munsang Km, and James E.Bobrow;"Newton-Type Algorithms for Dynamics-Based Robot Movement Optimization", Digital Object Identifier, 2004.

[5] Horst-michael gross, Alexander Koenig; "Robust Omniview-basad Probilistic Self-loalization for Mobile Robots in Large Maze-like Environments", proceedings of the 17th International Conference on Pattern Recognition, ICPR-2004.

[6] Shinichiro Yano, Manabu Noda, Hisahiro Itani, Masayuki Natsume, Haruhiko Itoh and Hajime Hattori, Tadashi Odashima, Kazuo Kaya, Shinya Kataoka and Hideo Yuasa, Xiangjun Li, Mitsuhiro Ando, Wateru Nogimori and Takahiro Yamada; "A Study of Maze Searching With Multiple Robots System", 7th International Symposium on MicroMachine and Human Science, 1996.

[7] Frank Lingelbach; "Path Planning using Probabilistic Cell Decomposition", International Conference on Robotics & Automation, 2004.

[8] Javier Antich and Alberto Ortiz; "Extending the potential Fields Approach to Avoid Trapping Situations", CICYT-DPI-2001.

[9] Gorden Mc Comb, Myke Predko,"Robot Builder's Bonanza", Mc-Graw Hill, 2006.

[10] Thomas Braunl; "Embedded Robotics mobile robot design and applications with Embedded systems", Springer 2006.

[11]Meng Wang, James N.K. Liu, "Fuzzy Logic Based Robot Path Planning in unknown Environment, the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 2005

[12] Roland Buchi, Gilles Caprari, Vladimir Vuscovic, Roland Siegwart; "A Remote Controlled Mobile Mini Robot", 7th International Symposium on MicroMachine and Human Science, 1996.