Evaluation of Modified Flood Fill Algorithm for Shortest Path Navigation in Robotics

[1]Anjali G. Chauhan, [2]Haresh A. Suthar
[1,2]Dept. of EC, PIET, Baroda, Gujrat, India

Abstract
This paper is to solve the problem of maze by using modified flood fill algorithm. It is need to be designed negotiate a path to the center of a maze in the optimal amount of time. It requires to a complete analysis and proper planning to be solved. This paper gradually improves the algorithm to accurately solve the maze in shortest time with some more intelligence. This modified flood fill Algorithm is reduce the complexity of understanding the algorithm and can reduce the memory usage and time taken for processing at each step. This algorithm is just straight forward and a robot programmer can easily implement on any maze solving robot. Purpose of this paper is to investigate the Modified Flood Fill algorithm, and evaluate the system using Modified Flood Fill algorithm.

Keywords
Algorithm, Shortest Path, Maze, Maze Solving Robot, Microcontroller

I. Introduction

The Micro mouse encompasses a vast range of engineering fields which can be divided into two categories: hardware and software. With the technology available for robotics today, the Micro mouse competition has become increasingly more complex since its inception several decades ago. Advanced microcontrollers and micro processors have transferred much of the complex logic that used to be implemented in hardware to software. The hardware is responsible for perceiving the surrounding environment and moving about the maze. The software, on the other hand, is responsible for navigating the maze, interpreting the environment, and sending control signals to the different subcomponent. The hardware has been further subdivided into component. The different hardware components are: power, sensors, control. The power system consists of the battery pack and voltage regulation scheme in the circuit. The sensors are the means through which the Micro mouse detects walls and traverses the maze with proper alignment in the center of a pathway. The focus of the software has been on the selection and development of algorithms for solving mazes since finding the center of the maze is the primary goal of the Micro mouse. The algorithms that we analyzed in detail are Depth First Search, Wall follower and Flood-Fill. Wall-Following is a trivial algorithm that is usually unsuccessful if implemented for IEEE Micro mouse mazes, while Depth-First Search is an intuitive algorithm that also proves ineffective due to wasted time searching the entire maze. As such, the Flood-Fill algorithm and its many variations result in the best searching techniques.

II. Flood Fill Algorithm

Flood-Fill is better suited to the Micro mouse than the two algorithms discussed above. The flood-fill algorithm is a good way of finding the shortest path from the start cell to the destination cell. This 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 robot is standing in a cell with a value of 1, it is 1 cell away from the goal. If it is standing in a cell with a value of 3, it is 3 cells away from the goal. When it comes time to make a move, the robot must examine all adjacent cells which are not separated by walls and choose the one with the lowest distance value. If there are more than one neighboring cell with the same value the robot will try to choose the one in which it makes the least amount of turn.

By comparing these three algorithms we have seen that flood fill algorithm is best for finding shortest path as fast as possible. Wall follower is very simple algorithm and it is no guarantee to find shortest path. Depth first search algorithm is find shortest path but it is not able to find fastest path. So flood fill algorithm is best for finding shortest path [3].

III. Rules of Flood Fill Algoritm

Almost each block of the maze will be visited. Only the blocks which are closed and the block on which the only one available path comes only through the destination position will not be visited. Each block will be given the numbering in increasing order counted as the number of steps. The numbering of current route and found shortest path will be kept in memory. The robot will move forward if there is no option available to move to any other side i.e. the only one path is available. When there are options available on some place i.e. move left, move right or move straight means more than one routes available, then that place will be assigned a check point with the number of checkpoint i.e. c1, c2 etc. The checkpoint will also be kept in memory along with numbering. From the checkpoint all available paths will be visited one by one considered as routes of the tree. The series of priority can be LCR, LRC and RCL etc. In my case it is taken Left-Centre-Right. If there is a dead end on the one route, then the robot will move back to checkpoint reversing the route till check point of that route blocking the route till checkpoint and will move to next route of new checkpoint will be completed and then it will back to the previous one. There will also be considered as dead end when any previous incomplete checkpoint comes in the way and when it reaches to other side the marked dead end will be considered dead-end from other side. When the destination is found then the number of that block and all the checkpoints of that path are noted. That gives the total no of steps taken to reach the destination using that path. On the destination if all checkpoints are completed then it moves forward otherwise it moves back to complete the routes. If then another path to destination is found to reach the destination then the total number steps will be compared with previous completed path. The shortest on will be kept and larger one will be set to waiting state. If the destination is found using one path and another path comes on the way of that already found destination path then the current number of that path will be compared with the found number [1].

IV. System Design

This system contains two programs.

- User Interface Program (Visual Basic program installed in Computer)
- Flood Fill Algorithm (Embedded C program installed in

Microcontroller)

User Interface program is using for interface the user with microcontroller board. And it also gives attraction to user. Working of this program is to get the inputs from user and pass to the Flood Fill Algorithm which is installed in microcontroller. It also gets the result from Flood Fill Algorithm and gives it to the user. It is done in VISUAL BASIC coding.

Working of Flood Fill Algorithm is getting the maze, source and destination location from user interface program. And find out the shortest path between source and destination, finally it will give shortest path (result) to user interface program or user.

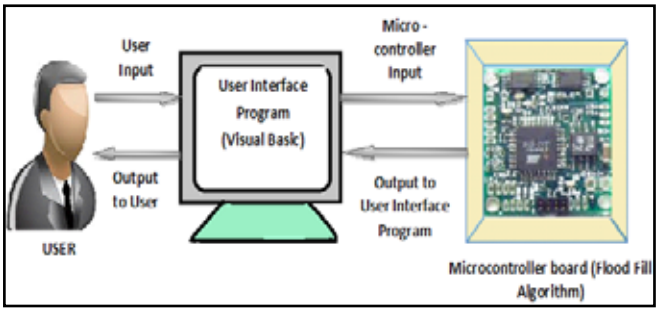

Fig. 1 General Block Diagram

V. Flowchart of Flood Fill Algorithm

The primary purpose of the software is to maintain control over the hardware at all times and determine where to move by solving the maze. Controlling the hardware consist of reading the sensors, setting the motor speed, and communicating with any external peripherals.

Since each of the motors speeds are controlled independently, alignment corrections can be made by increasing or decreasing the speed of a single motor.

In addition to controlling the hardware, the software must also keep track of the current position within the maze and determine where to move based on the selected algorithm.

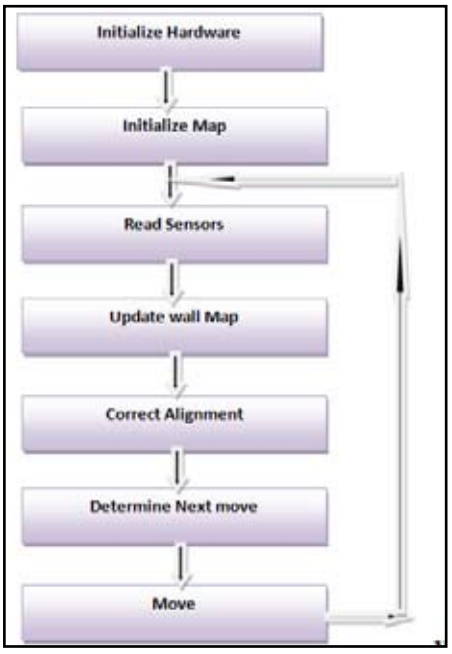

Fig. 2: Flow Chart of Flood Fill Algorithm

VI. System Flowchart

The input data is given through the PC. For that Visual Basic Graphical interface is used. When data is given through the PC USB to UART converter IC pl2303 is used. Then data is loaded in AVR At Mega16.There is one LCD display that gives the output result. The power supply unit that supply the power to AVR microcontroller.

Input is given by the port and algorithm will give output. Keyboard is used for input. This controller is connected to PC. When we give the input than algorithm will give the output. That algorithm is in our PC. AVR At mega 16 isexecute one million instruction per second So it will reduce the speed.

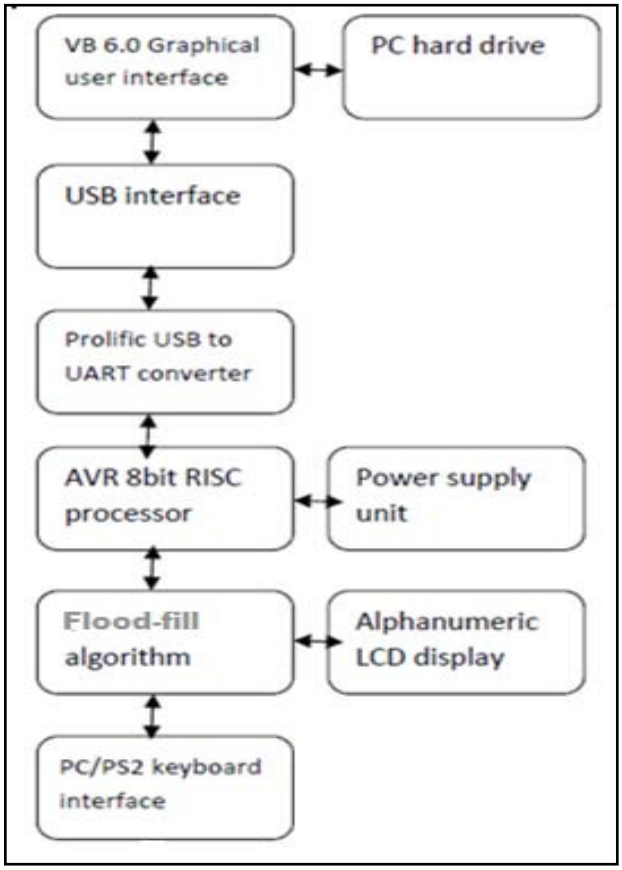

Fig. 3: System Flowchart

VII. Modified Flood Fill Algorithm

In modified flood-fill algorithm I have modified the some features. It is similar to the regular flood-fill algorithm only that is faster because, instead of flooding the entire maze, by updating only those values which need to be updated, the robot can make its next move much quicker. In order to implement this algorithm one rule must be followed every time the robot arrives into a new cell: If a cell is not the destination cell, its value should be one plus the minimum value of its open neighbors. After the distance values were updated, the robot must find the neighboring cell with the lowest distance value and it can once again follow the distance values in descending order. As the cells are mapped with the numbers, at each cell the robot is expected to take three decisions.

It finds fastest path (as well as shortest path). It includes effect of no. of turns. Each cell works on four bits and another four bit we can use for our own purpose. So it reduces the memory.

- Move to cell which it has gone to least
- Move to the cell that has minimum cell value
- If possible the robot must try to go straight.

A. Mapped System in Memory

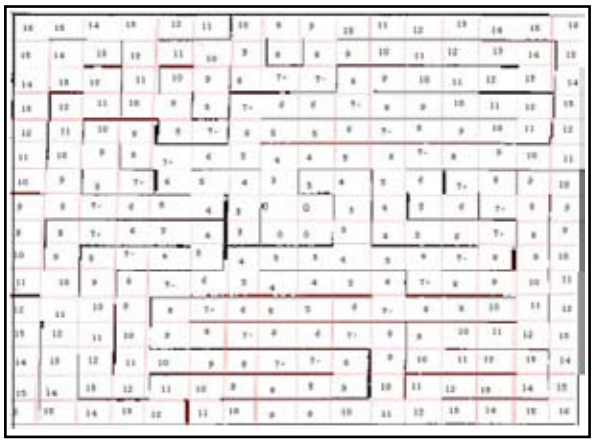

Fig. 4: Mapped Systems

The maze solving algorithm implemented in the system is self developed with improvements from the basic form of bellman flooding algorithm. The algorithm requires around 256 X 3 bytes of memory. The selected microcontroller for implementation had only 256 Kbytes of memory, thus a major memory crisis was to be tackled on the software basis. After appropriate analysis the problem statement was simplified to three rules which if followed would direct the robots to the centre of the maze [5].

VIII. Software Module

- Embedded C
- AVR studio 4.12 Debugging Platform
- Win AVR Simulator
- EAGLE (layout editor for Designing PCB layouts)
- Visual Basic 6

A. Optimization in Flood Fill Algorithm (Software)

This software will be implemented in Embedded C. In this software algorithm there are three important functions.

- Control Function.
- Simulator.
- Logical function.

Input (maze, source, destination)

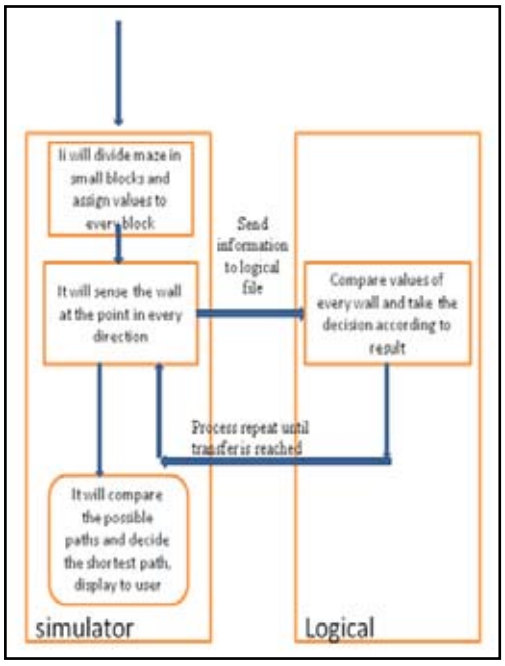

Fig. 5: Software Flow Chart

Control function contains user interface algorithm. In the control function there is visual basic based software which can be installing in host computer. VB (visual basic) software will get the inputs and commands from user and transfer to the embedded c software. And VB software also gets the input from embedded software (microcontroller). And display to the user. Embedded c software will be installed in microcontroller.

Embedded c software has two functions. One is simulator and another is logical file. Simulator will take the input data from the user interface algorithm or user. Input data contains input maze, source and destination. And then simulator divide the maze into small blocks and assign value to each block. It will assign every block in maze and this process will repeat to last block of maze. At last it will create one binary file which contains values of every block and then it will pass this information to the logical file. At the opposite side logical file will compare values of every block and take the decision according to the result. This process will repeat until destination position is achieved. It will give result back to simulator file. Simulator file will compare different paths and decide the shortest path. It will give shortest path to user interface algorithm.

IX. Simulation Result

A. User window

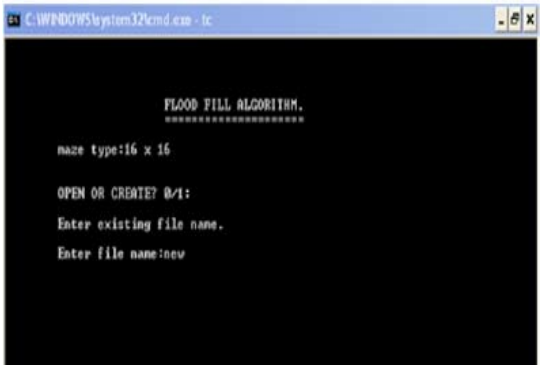

Fig. 6: User Window

When the program is run then this window will see. In this window maze type is 16×16.There are two option 0 and 1.0 is for exiting file and 1 is for new file.

B. Input Maze

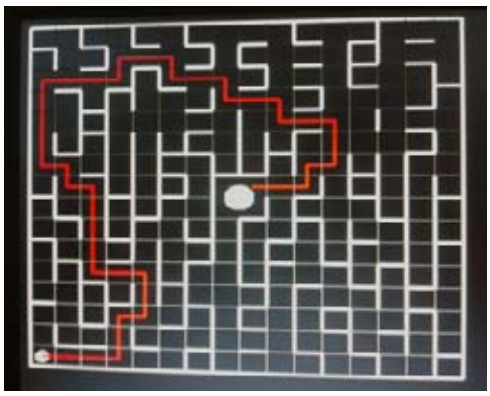

Fig. 7: Input Maze

Here is a maze that I have created. Here red line is shown shortest path.

B. Path Matrix (Initial state)

Fig. 8: Path Matrix (Input State)

This matrix is for initial stage. This matrix will change when robot move for one path to another.
Second Stage

C. MAZE (Stage 2)

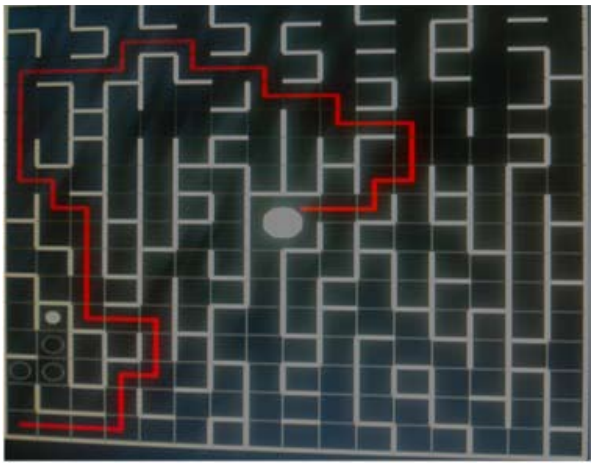

Fig. 9: Maze (Stage 2)

Here robot is moved three step but no any wall is detected so its path metrics is same.

D. Wall Matrix

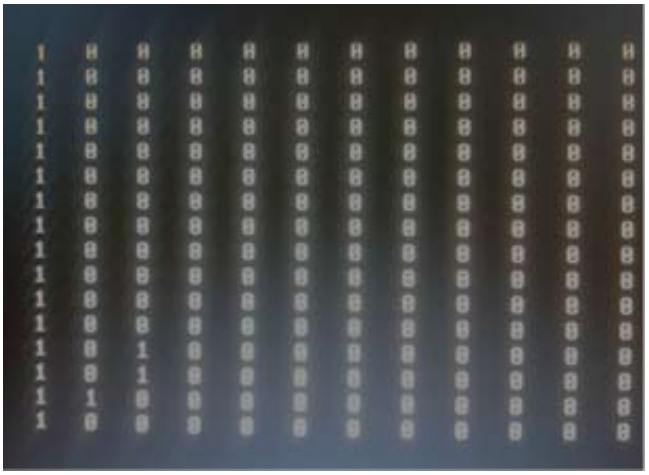

Fig. 10: Wall Matrix

E. Path Matrix

Fig. 11: Path Matrix

X. Conclusion

The Flood fill algorithm has such serious limitation so it can find shortest path but it can not find fastest path. So its need to improve the hardware and the complexity of the algorithm. So modified some feature in flood fill algorithm. This modified flood fill algorithm is the most important part of the project. This modified flood fill algorithm presented here is better then the flood fill algorithm in all cases but it requires the amount of processing and memory and the complex calculations. In this method the shortest path as well as fastest path is guaranteed. .

XI. Future Work

In next future work, the algorithm can be optimized into the controller so that it will find the short and fastest path by detecting wall and other obstacles in its way. Visual basic can be used for providing Graphical User Interface (GUI).
The next step in the implementation of this research is to build a small robot that is able to use this algorithm in a real world application. There is a still have a considerable amount of work remaining in the implementation. We have to do the PC board layout and construction. We must also code for the maze solving algorithm into the microcontroller. Associated with all of these tasks is an extensive amount of testing, for which we need to obtain or construct at least part of a life size maze.

References

[1] IMan preet Singh, Gianetan Singh,"A New Shortest Path Finding Algorithm For A Maze Solving Robot With Simulator", Vol. 2, No. 2, pp. 445-449, 2011.
[2] R. Clark, A. El-Osery, K. Wedeward, S. Bruder,"Proc. International Test and Evaluation Association Workshop on Modeling and Simulation", by Intelligent Systems and Robotics Group (ISRG) New Mexico Tech Socorro, NM, December 2004.
[3] Babak Hosseini Kazerouni, Mona Behnam Moradi, Pooya Hosseini Kazerouni,"Variable Priorities in Maze-Solving Algorithms for Robot's Movement", by Students of Electrical and Computer Engineering Department, Shahid Beheshti University.
[4] Tondra De,"A Detailed Design and Analysis of Micro mouse", University of Nevada, 2002.

[5] David Walden,"The Bellman-Ford Algorithm and Distributed Bellman-Ford", Drafted in April and May, 2003.
[6] Mongkol Ekpanyapong Thaisiri Wa terwai, Sung Kyu Lim,"Statistical Bellman-Ford Algorithm With An Application to Retiming", by School of Electrical and Computer Engineering Georgia Institute of Technology.
[7] Dan Puiu, Caius Suliman, Florin Moldoveanu Mihai Cernat Stylianos Papazis,"Microcontroller Based Autonomous Maze Solver Robot", by The 4th International Conference on Interdisciplinary in Education ICIE'09, May 21-22, 2009, Vilnius, Lithuania.
[8] [Online] Available: http://www.micromouseinfo.com/index.html