Design and Implementation of Flood Fill and Pledge Algorithm for Maze Robot

Semuil Tjiharjadi
Computer Engineering Dept., Maranatha Christian University, Bandung, Indonesia
Email: semuiltj@gmail.com

***Abstract*—Maze Robot is a path finding autonomous mobile robot which can reach a certain point. One of its capabilities is moving from one point to another autonomously. Maze Robot is able to explore an unknown environment. Map the environment and seeking good path to reach a certain point. This MazeRobot is a mobile robot which moves using wheels with differential steering type. It is designed to solve a maze environment that has a size of 5 x 5 cells and it is used to implement the flood-fill algorithm and the pledge algorithm. It is using ultrasonic range finders to detect walls and opening in the maze. The robot has ability to use pledge algorithm to collect the information and learn the maze, it finds all possible routes and solve the problem using the shortest one. Result of experiments show the robot can explore the maze and map it, Robot also can find the shortest path to destination point with 80% success rate.**

***Index Terms*—flood fill algorithm, pledge algorithm, path finding, maze**

I. Introduction

One of important features of mobile robotics is autonomous navigation. It is the ability of the robot to independently move to target location without being controlled. There are many algorithms have been developed for this purpose, each of them is having their own strengths and weaknesses.

Autonomous navigation is an important feature of mobile robotics. It allows the robot to independently move from a place to target location without a tele-operator. There are several techniques and algorithms have been developed for this purpose, each of them having their own advantages and disadvantages [1-7].

As an autonomous robot, Path Finding Robot uses structured techniques and controlled implementation of autonomous navigation which is preferable in studying specific aspect of Flood Fill Algorithm and Pledge Algorithm [1]. This research discusses implementation of a small size mobile robot designed to solve a maze based on the both algorithms.

Robot maze problems are based on decision making algorithm that is very important field of robotics. Mobile robot has path finding task to solve a maze in the least time possible and using the shortest way [2]. It must navigate from a corner of a maze to the center as fast as possible [3].

The robot knows where the starting and target location, but it must look all information about the obstacles to achieve target location. The maze is composed of 25 square cells, where the size of each cell is about 18 cm x 18 cm. The cells are arranged to form a 5 rows x 5 columns maze. One cell at its corners is a starting location and the target location is at the center of the maze. Only one cell is opened for passing. Maze walls and support platform's requirements are provided in the IEEE standard.

II. Literature Review

A. Breadth First Search

Breadth First Search is a search algorithm that begins at the root node and explores all the neighboring nodes until it finds the goal. It needs large memory space. It discovers few solutions and at least one has shortest path. All nodes obtained by expanding a nearest neighbor node in First In First Out queue. Breadth First Search works poorly when the solutions have long path. It has large space complexity.

B. Depth First Search

Depth First Search is an algorithm for searching a graph or tree data structure uses Last In First Out queue. It is simple to implement, starting at the root node and goes as far as it can down in path, and then backtracks until it finds an unexplored path, and then explores the new one, until it finds the target. Depth First Search's problem it requires large computing power, for small increase in map size, runtime increases exponentially [7].

C. Heuristic Function

Heuristic function is a function that is using all mapping information to inform the search about the right direction to a goal. It maps problem state descriptor to a number which represents degree of desirability. It plays vital role in optimization problem [8].

D. Genetic Algorithm

Genetic algorithm is inspired by natural evolution to find approximate optimal solution. Advantages of Genetic algorithm are it solves problem with multiple solutions. But it needs very large input and data. Problems of Genetic algorithm are certain optimization cases cannot be solved due to poorly known fitness function. It is not able to assure constant optimization response times because of the entire population are improving [9].

Manuscript received July 10, 2018; revised April 26, 2019.

doi: 10.18178/ijmerr.8.4.632-638

E. A* Algorithm

A* is one of most popular methods for finding the shortest path in a maze area. It is developed as combination heuristic approaches like Best First Search (BFS) and formal approaches like Dijkstra's algorithm. It is an algorithm which cost associated with each node is calculated using admissible heuristic likes BFS. It follows its path with lowest known heuristic cost. Likes BFS that needs large memory requirement to store its drawback information, A* also needs the large memory too for the same reason because entire open list is to be saved [4].

F. Flood Fill Algorithm

Flood fill algorithm that also known as seed fill algorithm, is an algorithm that determines the area connected to a given node in a multi-dimensional array. This algorithm needs all information of maze and proper planning [3]. It is used widely for robot maze problem.

The Flood fill algorithm gives values to each node that represents the distance of the node from the center. It floods the labyrinth when it reaches a new cell or node. This algorithm requires continue update [11].

G. Wall Follower Algorithm

Wall follower algorithm is used left or right-hand rule. Robot detects its left or right side on the wall at the start of the maze, and then starts moving. Never lose left or right-side detection. It works for a simply connected maze.

H. Pledge Algorithm

The pledge algorithm is designed for circular obstacles and has an initial direction to move forward. The robot will run in the main direction until it finds obstacles. When the robot finds an obstacle, the robot will use a wall follower search method and will avoid obstacles by prioritizing the right or left side. It will calculate total turn and try to return to initial direction (total turn count is "0") [8].

III. Hardware Design

This research is using miniQ 2WD robot chassis as robot base construction. Fig. 1 is the chassis of the robot. It consists of a robot chassis with 122mm diameter, a couple wheels, a piece of ball caster and a couple Direct Current (DC) motors which have gearbox and also DC motor bracket.

Figure 1. 12*WD miniQ robot chassis.*

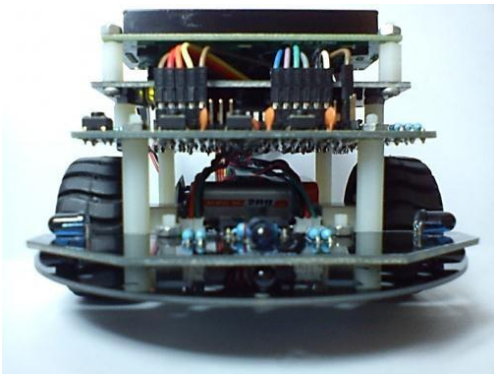

Figure 2. Mobile Robot from side view.

This maze robot also has a couple pieces rotary encoder. Rotary encoder attached to the DC motor to calculate the rotation of the wheels. It is shown in Fig. 2 [12]. Fig. 3 is shown the block diagram of design of whole hardware system and the flowchart of main program can be seen at Fig. 4.

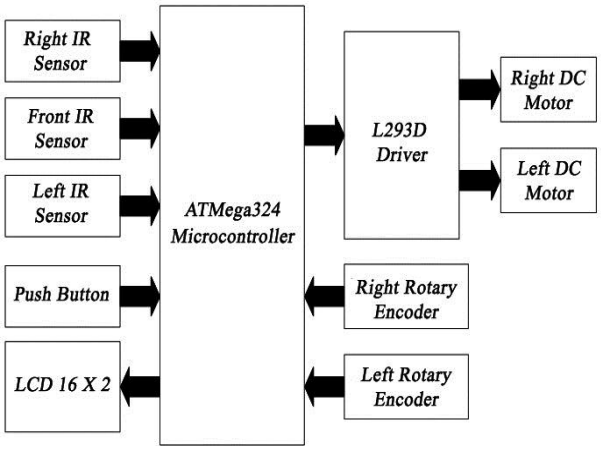

Figure 3. Maze robot's block diagram.

It has three infrared sensors to detect front, right and left position of the maze wall. This maze robot uses driver L293D to control the speed and rotation of a DC Motor [13]. It also has rotary encoder that has a job to calculate the rotation of both wheels. Push button is used to start the robot.

Robot system would drive DC motors to move the wheels. It would control the robot to move forward, turn to the left or right, and rotates reverse [14]. This maze robot has an AT Mega 324 microcontroller to respond the input signal and run the actuator based on processing algorithms [10]. All status and information are displayed on the Liquid Crystal Display (LCD) 16 x 2 at Figure 5.

The maze designed for the robot to solve is of the size of 5×5 cells as shown in Fig. 6. The actual maze constructed, as shown in Fig. 7, has a physical size of about 1.32 m^2. The maze was designed so that it will have two paths for it to be solved. One of the paths is longer than the other. The robot (Fig. 2) must decide which one of the paths is shorter and solve the maze through that path.

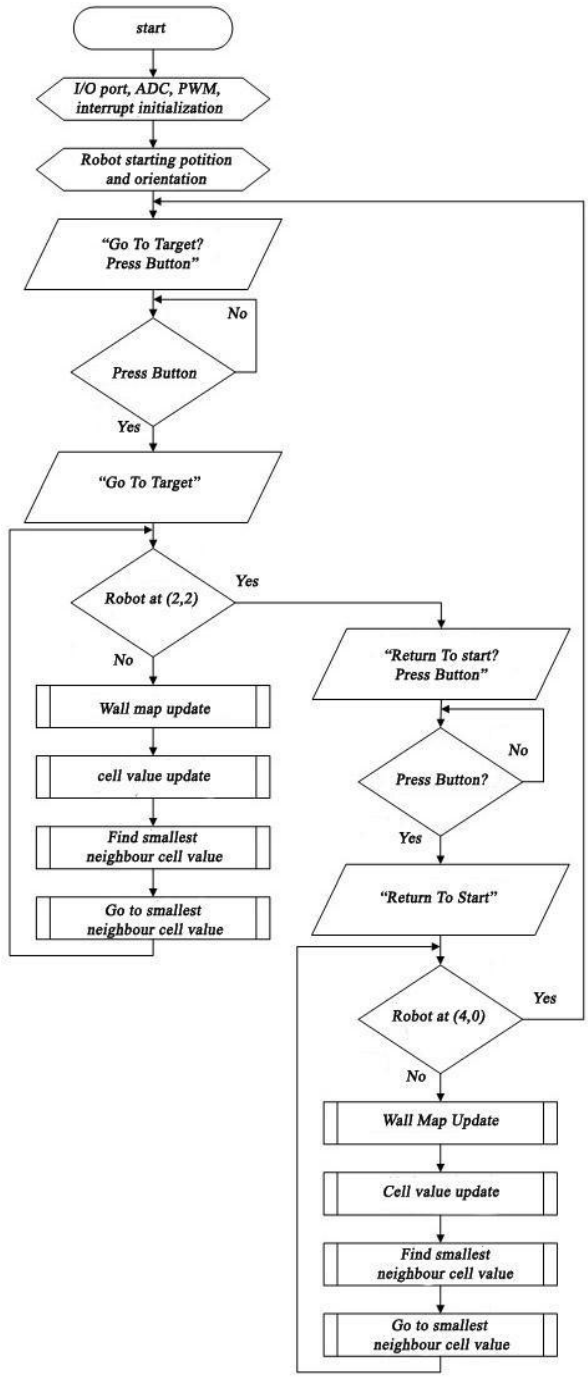

Figure 4. Flowchart of the main program.

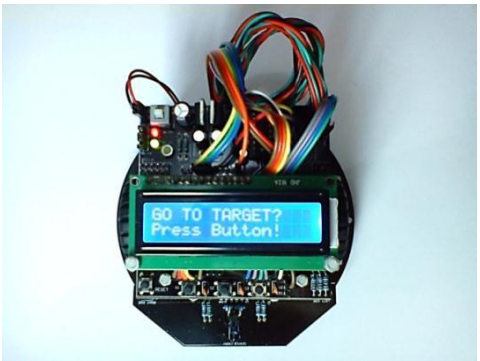

Figure 5. Mobile robot from above view.

IV. ALGORITHM

There are several algorithms that can be implemented to solve the maze cases. One of the suitable algorithms to search goal in the middle of the maze is Flood-fill algorithm. In this case, flood-fill algorithm was chosen to solve the maze due to its simplicity but efficient [3].

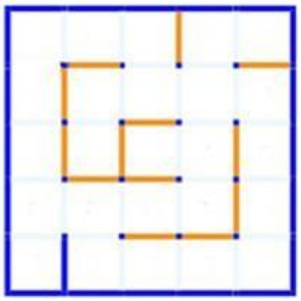

Figure 6. The layout of maze.

Figure 7. The maze arena.

Together with the flood-fill algorithm that is used to find the fastest way to reach the destination, a pledge algorithm is used to determine the priority of the direction taken when the robot finds the same priority value based on the flood-fill algorithm. The pledge algorithm will give the +1 value to the 'Turn' variable every time you turn right and -1 value every time you turn left. The goal is to achieve the goal by prioritizing the smallest possible 'Turn' variable value. So that every time the pledge algorithm finds an intersection, the turn decision that is taken is to reduce the 'Turn' variable value of the rotation. This pledge algorithm is used to help flood-fill algorithms so that they have smarter decisions [5].

Artificial Intelligence program has two-dimensional memory array to map the maze's arena which has size of 5x5. The memory array is used to store information in each cell walls of the maze and each cell value information. The robot's positions in the program are expressed by the coordinates (row, column). The movement of the robot in the array is done to position the robot as in Figure 8.

The coordinates of the line will increase 1 when the robot moves one cell to the South. On the other hand, it will be reduced by 1 when the robot moves to the North. The column will be reduced by 1 when robot moves to the West, and it will be increased by 1 when robot moves to the East. Robot has already information about the initial orientation, the initial position, the size of the maze and the location of the maze's outer walls.

The Flood fill algorithm has four main steps: the first is wall data updates, second is cell value updates, the third is the smallest neighbor cell calculation, and the last is moving to the smallest neighbor cell.

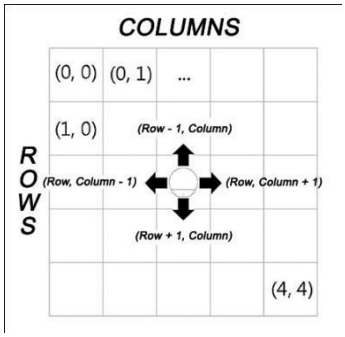

Figure 8. Robot's Array Movement

A. Wall Data Update

Robot will check its environment, any walls in its three directions: right, left and front directions. The robot will also detect the distance of any obstacle of its three directions. Anyone exceed 20 cm is updated as "wall" on its respective side. Flowchart in the Figure 9 describes the wall data update mechanism.

The maze robot also needs to know which direction it is facing so it knows where to go: north, east, west or south. Table 1 describes the relation of robot orientation and wall sensor detection. The robot has an initial orientation when it starts at the beginning and will continue to track changes in direction.

TABLE I. ROBOT ORIENTATION AND WALL DETECTION

Robot Orientation	Wall Sensor Detection		
	Right	Front	Left
South	West wall	South wall	East wall
West	North wall	West wall	South wall
North	East wall	North wall	West wall
East	South wall	East wall	North wall

B. Cell Value Update

Update cell values (refill each cell with a new value) serves to adjust the value in each cell wall position that has been updated by the robot. Values stored in a 2-dimensional array of 5x5 memory cells. Update cell values is done using the flood fill algorithm.

Updating the cell value subroutine to function by resetting the previous cell value, then giving a value of 255 in each cell, then filling in the values of these cells in stages, the initial value 0 to all cells filled in with the value. The cells that will be updated are the current level array while the neighboring cells will be entered in the next level array. After the value filling process is complete, the cells in array next level will be moved to the current level array to do the next value. The update process will be complete if the array cell next level is empty.

C. The Smallest Neighbor Cell Calculation

Subroutine determines the smallest neighbor cell that functions to find neighboring cells that have the smallest value. The search for the smallest neighbor cell is done based on priority, so that if there is more than one neighbor cell that has the smallest value, then the cell selected is a cell that has a higher priority.

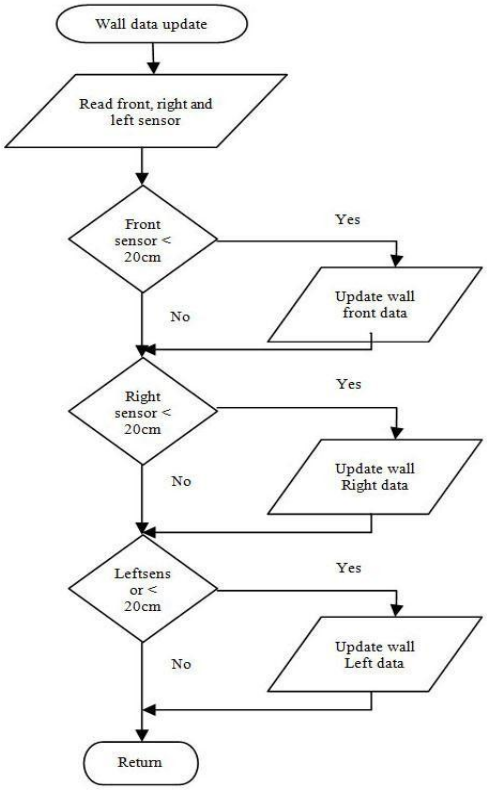

Figure 9. Flowchart for updating wall location at each cell

Prioritization is arranged based on the movement of the robot moving forward one cell has the first priority, the second priority is to move one cell to the right, while the third priority is to move one cell to the left, and the fourth or last priority is to move one cell backwards. For example, if a robot faces the East, then the East cell has the first priority, the two South has the priority cell, the cell has the third priority North and the Western cell has the fourth priority as in Figure 10. If the robot faces the East, the East cells have the first priority, South cells have a second priority, North has third priority cells, and Western cells have a fourth priority.

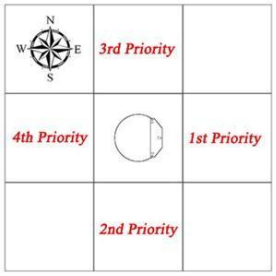

Figure 10. Priority of Neighbor cell

D. Moving to the Smallest Neighbor cell

Program subroutines move the robot to the smallest neighboring cells, after the robot finds neighboring cells. To move to a cell, the robot must know the location of the cell. Next, the robot will move to the cell by observing

orientation. For example, if the South cell is the smallest cell and the orientation of the robot is facing west, then moves to the position of the cell, the robot must turn left, then move forward as in Fig. 11. If the South cell is the smallest cell and the orientation of the robot is facing east, then moving to the position of the cell, the robot must rotate to the right, then move forward.

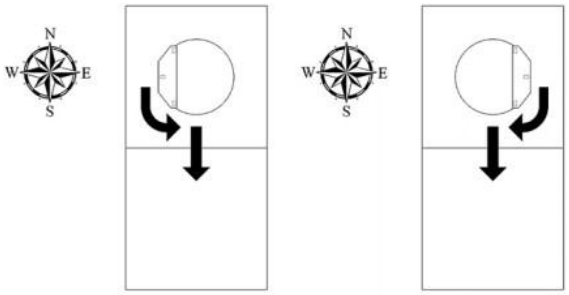

Figure 11. Moving to smallest neighbour cell.

V. Results and Discussion

In this experiment, Robot will learn to find the shortest path from the starting cell (line 4, column 0) to the destination cell (row 2, column 2) and then back again to the initial cell. The initial orientation of the robot is facing the North.

The maze simulator program aims to facilitate the observation on how the flood fill algorithm. Figure 12 is a view maze simulator program. Maze blue wall is a wall that position known to the robot. While the maze walls are colored orange wall position is not known by the robot.

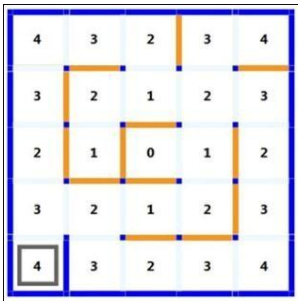

Figure 12. Simulation search path to cell (2,2), Turn = 0

First experiment, Robot will perform a search of the initial cell lines (4.0) to the destination cell (2, 2). Flood fill algorithm simulation results when a search of the cell lines (4, 0) to the cell (2, 2) are shown in Figure 12to 22.

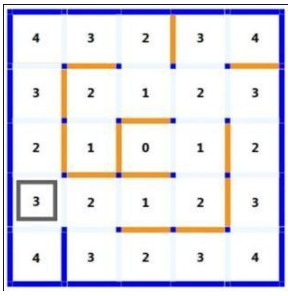

Figure 13. Simulation search path to cell (2,2), Turn = 0

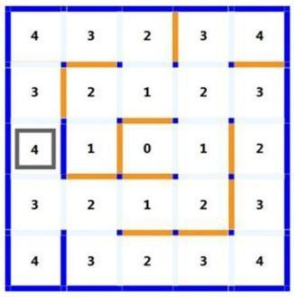

Figure 14. Simulation search path to cell (2,2), Turn = 0

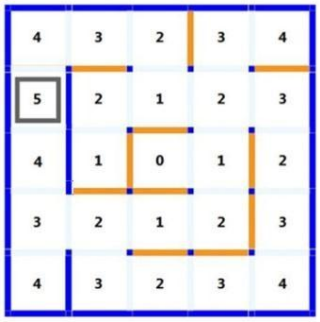

Figure 15. Simulation search path to cell (2,2), Turn = 0

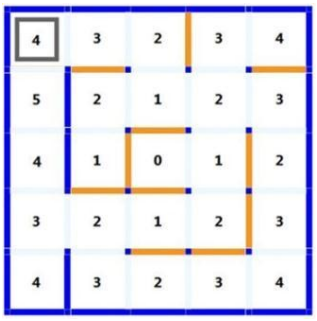

Figure 16. Simulation search path to cell (2,2), Turn = 0

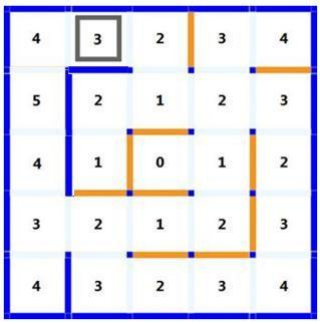

Figure 17. Simulation search path to cell (2,2), Turn = 1

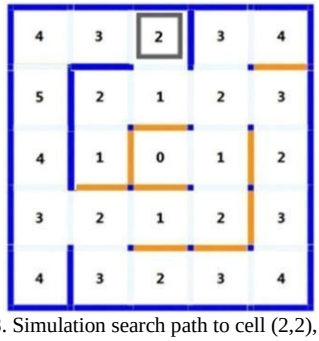

Figure 18. Simulation search path to cell (2,2), Turn = 1

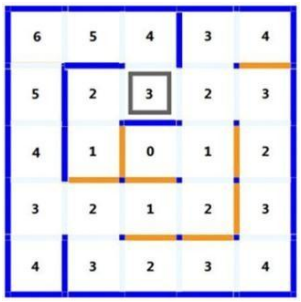

Figure 19. Simulation search path to cell (2,2), Turn = 2

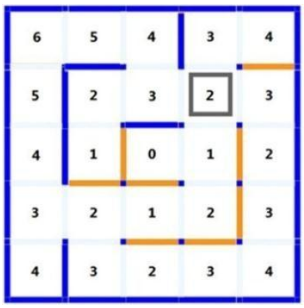

Figure 20. Simulation search path to cell (2,2)), Turn = 1

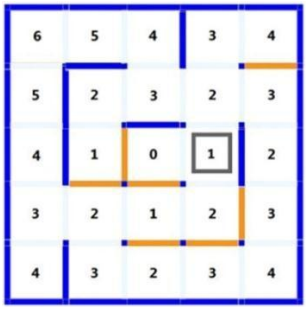

Figure 21. Simulation search path to cell (2,2)), Turn = 2

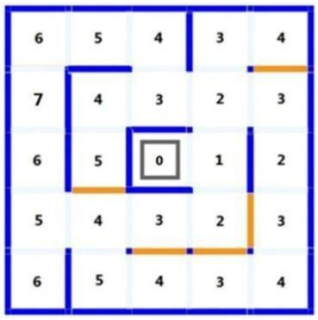

Figure 22. Simulation search path to cell (2,2)), Turn = 3

The second experiment is an attempt to find the path of the robot to the starting point of the experiment 1. The robot spins to look for the direction of the starting position. If the robot gets more than one possible initial direction, then the south direction will be set as the first direction. If there is only one choice, then the initial direction of the robot position is directed at the open wall. In this second experiment, the robot gets East as the starting direction. This robot trip can be seen in Figures 23 to 28.

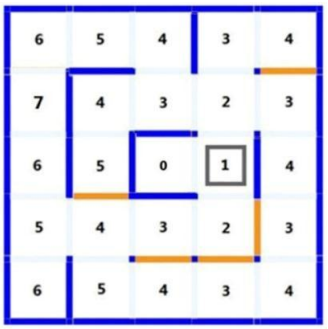

Figure 23. Simulation search path to cell (2,2)), Turn = 0

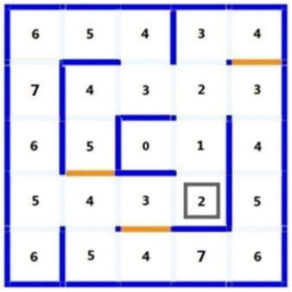

Figure 24. Simulation search path to cell (2,2)), Turn = 1

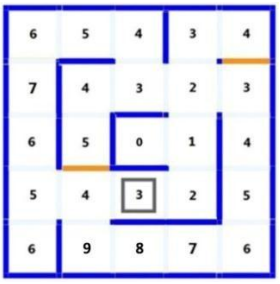

Figure 25. Simulation search path to cell (2,2)), Turn = 2

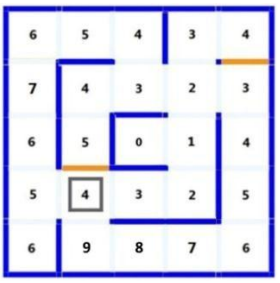

Figure 26. Simulation search path to cell (2,2), Turn = 2

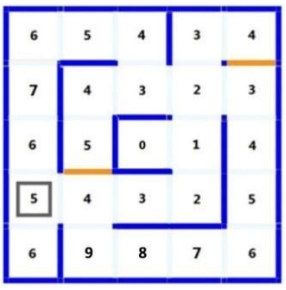

Figure 27. Simulation search path to cell (2,2), Turn = 2

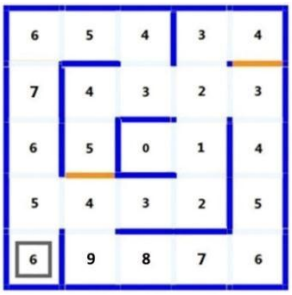

Figure 28. Simulation search path to cell (2,2), Turn = 1

After the robot updates the wall data while running a search on the first experiment and travels home in the second experiment, the robot has enough data to find the fastest path to the destination in the cell (2,2). All robot search and back home information make the robot can find the shortest path the cell (2.2) on third experiment. It can be seen in Table II.

TABLE II. ALL ROBOT EXPERIMENTS

	Routes	**Number of steps**
First run	(4,0) →(3,0) → (2,0) → (1,0) → (2,0) → (3,0) → (3,1) → (3,2) → (3,3) → (2,3) → (2,2)	10
Return home	(2,2) → (2,3) → (3,3) → (3,2) → (3,1) → (3,0) → (4,0)	6
Second run	(4,0) →(3,0) → (3,1) → (3,2) → (3,3) → (2,3) → (2,2)	6

Wall map data will be updated when the robot go to cells that have not been visited before. Flood fill algorithm will update the value of the cell based on the position of the wall that has been mapped out by the robot. Robots always perform movement to neighboring cells which have the smallest value. If there is more than one neighboring cell that has the smallest value, then the cell selection will be done on a priority basis. Go forward has first priority, turn to the right has the second priority, turn to the left has a third priority, and move backwards has a fourth priority.

The value is changed in accordance with the position of the wall that has been mapped out by the robot. Cell values represent the cell distance to the destination cell.

VI. CONCLUSION

This design and implementation of the robot is a study about the ability to equip a small mobile robot with the ability to learn how to navigate in unknown environment based on its own decisions. The flood-fill algorithm was found to be an effective tool for maze-solving of a moderate size. For the robot to make its decisions it relies on inputs from several sensors, namely the ultrasonic range sensors and wheel rotation decoders.

The robot has successfully able to map the maze in the first, return home and second runs. In its second run it reaches its target cell through the shortest route it has mapped in the previous first run and return home.

Future works may include to studying the robot's maze solving capability in a larger, more complex maze and more combination of algorithms. It is also need better object sensor, such as a wide laser range finder, for better search.

REFERENCES

[1] Bekti, Samudra Harapan, "Pencarian shortest path dinamikdenganalgoritma bellman based flood fill dan implementasinya pada robot micromouse," Institut Teknologi Bandung, 2009.

[2] Elshamarka, Ibrahim and Abu Bakar SayutiSaman, "Design and implementation of a robot for maze-solving using flood-fill algorithm," Universiti Teknologi Petronas, 2012.

[3] Tjiharjadi, S. and E. Setiawan, "Design and implementation of path finding robot using flood fill algorithm," *International Journal of Mechanical Engineering and Robotics Research*, vol. 5, no. 3, July 2016, pp. 180-185.

[4] Tjiharjadi, S., M. C. Wijaya, and E. Setiawan, "Optimization maze robot using A* and flood fill algorithm," *International Journal of Mechanical Engineering and Robotics Research*, vol. 6, no. 5, September 2017, pp. 366-372.

[5] I. Elshamarka and A. B. S. Saman, "Design and implementation of a robot for maze-solving using flood-fill algorithm," *International Journal of Computer Applications*, vol. 56, no. 5, pp. 8-13, October 2012.

[6] A. Ansari, M. A. Sayyed, K. Ratlamwala, and P. Shaikh, "An optimized hybrid approach for path finding", *International Journal in Foundations of Computer Science & Technology (IJFCST)*, vol. 5 no. 2, pp. 47-58, March 2015.

[7] K. Sharma and C. Munshi, "A comprehensive and comparative study of maze-solving techniques by implementing graph theory," *IOSR Journal of Computer Engineering*, vol. 17, no. 1, Ver. IV, pp. 24-29, 2015.

[8] R. K. Sreekanth, "Artificial intelligence algorithms," *IOSR Journal of Computer Engineering (IOSRJCE)*, vol. 6, no. 3 September-October, 2012.

[9] Cook, David., *Intermediate Robot Building*. New York: Apress. 2010.

[10] Mazidi, M. Ali, S. Niami, D. S. Niami., *The AVR Microcontroller and Embedded System*. New Jersey: Prentice Hall. 2011.

[11] B. Thomas, *Embedded Robotics*. Berlin: Springer. 2006.

[12] Rizqiawan, Arwindra, *Sekilas Rotary Encoder*. http://konversi.wordpress.com/2009/06/12/sekilas-rotary-encoder/, Juni 2014.

[13] S. Paul, *Practical Electronics for Inventors*. New York: McGraw-Hill. 2000.

[14] G. W. Lucas, *A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators*. [Online]. Available: http://rossum.sourceforge.net/papers/DiffSteer/, Juni 2014.

Semuil Tjiharjadi is currently serves as vice rector of capital human management, assets and development. He is also Lectures in Computer Engineering Department. His major research on Robotics, Computer automation, control and security. He has written several books, To Be a Great Effective Leader (Jogjakarta, Indonesia: Andi Offset, 2012), Multimedia Programming by SMIL (Jogjakarta, Indonesia: Andi Offset, 2008), Computer Business Application (Bandung, Indonesia: Informatics, 2006) and so on. The various academic bodies on which he contributed as: Head of Computer Engineering Department, Member: Senate of University, Member: APTIKOM, Member: MSDN Connection, Member: AAJI.