International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

# Optimization and Simulation of A Navigation Robot in Mazes

Joel Yew-Hao Hii, Jer-Vui Lee, Yea-Dat Chuah
Department of Mechatronics & BioMedical Engineering
Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman, Kajang, Malaysia

***Abstract* — Path planning has been relatively well known in the fields of robotics, especially on autonomous navigation robot. It has the capability to venture from one place to another by using the optimal path acquired from path planning method. Not only that, path planning has also been implemented for maze solving. Throughout the years, many scientists had been formulating all sorts of path planning algorithm, whereby they are implemented on a navigation robot for experimental testing in real life mazes. Therefore, this study will focus on three selected path planning algorithms (A*, Breadth-First Search and Left-Hand Rule), where a comparison test will be conducted to acquire the most appropriate one among the rest. The selected one will then be added with enhanced feature, where it will be simulated in a 2D graphical rendering surface. After this, the path planning algorithm will then be implemented in a physical constructed robot, where it will be tested again in a physical maze. The findings and results will be discussed accordingly.**

***Keywords—Path planning; A* algorithm; Breadth First Search algorithm; Left-Hand Rule algorithm; mazes; navigation robot***

## I. INTRODUCTION

In recent years, navigation robot has been widely used in performing tasks such as rescue operation, disaster relief and space exploration [1]. One of the noteworthy features that can be considered is their path planning capability. Path navigation is a process where the route or path that is planned must be correct to ensure that the navigation robot is able to move safely and freely without getting lost or colliding with other objects [2]. It has been relatively well known in the fields of robotics where it plays an essential role in the navigation of autonomous robot [3]. Although path planning has coexisted alongside autonomous robots, it is still unable to obtain a universal solution [4]. In recent years, various kind of path planning algorithm had been developed. Each algorithm has their own specific aspect, where some requires the information of the maze while others do not. Hance, three selected path planning algorithm techniques, namely A-Star (A*) algorithm, Breadth First Search (BFS) algorithm and Left-Hand Rule (LHR) algorithm are selected for comparison and examination, where the most appropriate one will be chosen and implemented with enhanced feature. Moreover, the path planning algorithm will also be tested in a physical navigation robot and maze.

## II. PATH PLANNING ALGORITHGMS

### *A. A-Star (A*) Algorithm*

A* algorithm is known as the best path planning algorithm, where it can be applied on any topological configuration space [5]. The algorithm is a combination of both exhaustive search and greedy search [6]. Exhaustive search guarantees an optimal path but requires long computational time. Greedy search excludes the optimal path by obtaining the overall search with less computational time [7]. Both exhaustive search and greedy search are represented as the movement cost and heuristic cost of the distance.

Furthermore, A* relies of two lists, namely an open-list and a closed-list [8]. The open-list contains a list of nodes that can be travel to, while the closed-list contains all the nodes that had already been travelled. The algorithm will check all adjacent nodes from its current location and add them together into the open-list.

The current location will be added with the adjacent nodes that are included into the movement cost. At the same time, the heuristic cost will be computed for those adjacent nodes that have not yet been calculated [9]. The equation for movement cost can be seen:

$$g_n = g_c + \sqrt{(r_n - r_c)^2 + (c_n - c_c)^2} \qquad (2.1)$$

Where $g_n$ and $g_c$ is the movement cost for both adjacent node and current node, $r_n$ and $c_n$ is the row and column coordinate for adjacent node, while $r_c$ and $c_c$ is the row and column coordinate for current node. The lower movement cost on adjacent node will be prioritize [9]. The equation for heuristic cost can be seen:

$$h_n = \sqrt{(r_f - r_n)^2 + (c_f - c_n)^2} \qquad (2.2)$$

Where $h_n$ represents the calculated heuristic cost, $r_f$ and $c_f$ represents the row and column coordinate of the desired ending location, while $r_n$ and $c_n$ represents the row and column coordinate of the adjacent node.

The summation of both movement cost and heuristic cost will be evaluated as the fitness cost. It will be listed in the open-list, while the node with the lowest overall cost will become the new current node [10]. The equation of the fitness cost can be seen:

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

$$f_n = g_n + h_n \quad (2.3)$$

Where $f_n$ represents the fitness cost, $g_n$ represents the movement cost and $h_n$ represents the heuristic cost. The whole process is repeated until the desired ending location is reached.

### *B. Breadth First Search (BFS) Algorithm*

Breadth First Search algorithm is an algorithm that is used for traversing either a tree or graph data structure [11]. The algorithm works efficiently by visiting and marking all the key cells in an accurate breadthwise fashion in the graph. It expands the cell into different levels respectively while utilizing the First-In, First-Out (FIFO) manner to visit the nodes [12]. The ordering is based on the enumeration of cells in a graph where the possible output is applicable. Hence, using Graph, G = (V, E) and a source cell υ, the BFS will traverse the edges of G to find all the reachable cells from υ. At the same time, the algorithm will also compute the shortest distance to any reachable cell. Any path between two points in a BFS tree will correspond accordingly from the root υ to any other node *s*.

In maze solving, BFS is used to find the goal cell from the given source cell. Starting from the source cell, the overall layer of unvisited cell will be visited and added into the visited set [13]. As mentioned, BFS algorithm will label each cell from the start to all neighbouring vertex while marking the start cell as 'zero' [14]. The algorithm will constantly update the cell that are immediate neighbours from the start cell. The search continues until the goal is found.

### *C. Left-Hand Rule (LHR) Algorithm*

Left-Hand Rule algorithm is recognized as the most common wall follower algorithm [15]. This algorithm operates by taking precedence to the left-hand side in the maze [16]. Therefore, when the algorithm is placed into the maze, it will run accordingly by sticking to the left-hand side at all times. As the algorithm reaches an intersection, the left side will take priority first. If the left side is open, the algorithm will turn left then move forward. Otherwise, it will check to see whether it can move forward or not. If there is no forward option, the final resort is by turning right [17].

## III. METHODOLOGY

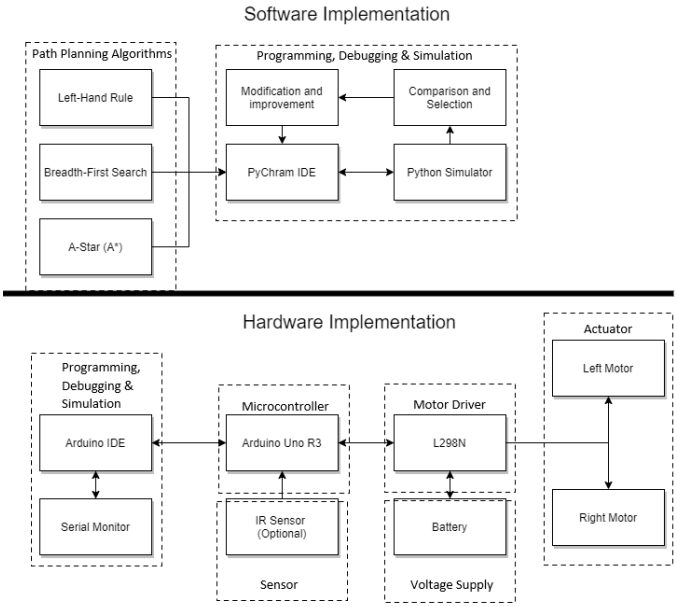



Fig. 1: Project Overview

The overview of the project is categorized into two parts, namely hardware and software. The software part will cover mainly on the comparison, selection, feature added and implementation of the path planning algorithm into the navigational robot. First and foremost, the three selected path planning algorithms will be tested and evaluated using three different maze size, namely 5 x 5, 10 x 10 and 15 x 15 in python software platform. This is to verify the most suitable path planning algorithm in terms of time and number of paths taken. After that, the selected one will be added with new feature and tested on the python software. Moreover, the path planning algorithm code will be implemented into the physical robot through Arduino software.

On the other hand, the hardware side will cover mainly on the construction and development of the physical robot and maze. This includes the hardware purchase and gathering, the connection of various components and material.

## IV. RESULTS AND DISCUSSIONS

### *A. Comparison Test*

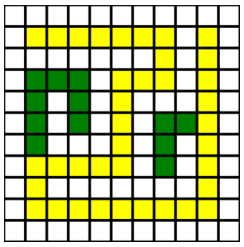


Fig. 2: Result for 5 x 5 maze

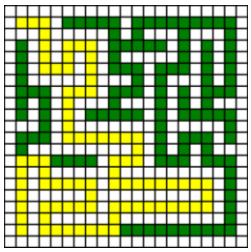


Fig. 3: Result for 10 x 10 maze

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

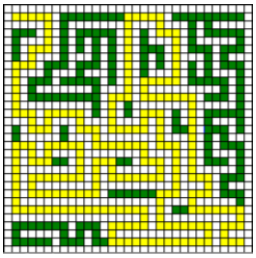


Fig. 4: Result for 15 x 15 maze

The path planning algorithm was tested in the python program, where the results can be seen in Figure 2, Figure 3 and Figure 4. The white block represents the wall of the maze while the block in black colour represents the empty path in the maze. Moreover, the green colour blocks represent the searched paths and the yellow blocks represent the optimal paths. Two criteria were used to evaluate the performance, namely average time and path taken. The time will cover the beginning where the algorithm starts to solve the maze until the optimal path is attained. Nonetheless, the path is taken for each step the algorithm takes during the process. The result on each path planning algorithm can be seen in TABLE I (5 x5), TABLE II (10 x 10) and TABLE III (15 x 15).

TABLE I. COMPARISON TEST FOR PATH PLANNING ALGORITHM IN A 5 X 5 MAZE

| Parameter | A* | BFS | LHR |
|---|---|---|---|
| 1st Time Taken (s) | 0.848896 | 1.444130 | 1.572429 |
| 2nd Time Taken (s) | 0.844153 | 1.352412 | 1.550257 |
| 3rd Time Taken (s) | 0.845855 | 1.397986 | 1.554386 |
| **Average Time Taken (s)** | **0.846301 (≈ 0.85)** | **1.398176 (≈ 1.40)** | **1.559024 (≈ 1.56)** |
| **Total Path Length (cells)** | **36** | **36** | **42** |

TABLE II. COMPARISON TEST FOR PATH PLANNING ALGORITHM IN A 10 X 10 MAZE

| Parameter | A* | BFS | LHR |
|---|---|---|---|
| 1st Time Taken (s) | 1.943768 | 5.058832 | 9.947831 |
| 2nd Time Taken (s) | 1.893884 | 5.088562 | 9.779287 |
| 3rd Time Taken (s) | 1.908919 | 5.009050 | 9.911231 |
| **Average Time Taken (s)** | **1.915523 (≈ 1.92)** | **5.052148 (≈ 5.05)** | **9.879450 (≈ 9.88)** |
| **Total Path Length (cells)** | **82** | **82** | **274** |

TABLE III. COMPARISON TEST FOR PATH PLANNING ALGORITHM IN A 15 X 15 MAZE

| Parameter | A* | BFS | LHR |
|---|---|---|---|
| 1st Time Taken (s) | 6.164448 | 13.469903 | 19.645816 |
| 2nd Time Taken (s) | 6.042483 | 13.428365 | 19.616728 |
| 3rd Time Taken (s) | 6.045526 | 13.535261 | 19.618392 |
| **Average Time Taken (s)** | **6.084152 (≈ 6.08)** | **13.477843 (≈ 13.48)** | **19.626977 (≈ 19.63)** |
| **Total Path Length (cells)** | **248** | **248** | **539** |

From the comparison test, the average time for LHR algorithm was the longest when compared to BFS and A* algorithm, requiring an average time of 1.56 seconds, 9.88 seconds and 19.63 seconds to solve the 5 x 5, 10 x 10 and 15 x 15 maze respectively. Besides that, the total path length it took was 42 cells, 274 cells and 539 cells. This is because the algorithm did not require any assumptions whereby it moves along the maze by staying on the left hand side. This will cause it to make numerous U-turns in the event of a bigger maze, thereby resulting in longer time needed to complete on a bigger maze.

Next, the average time taken using BFS algorithm was lesser compared to LHR algorithm, roughly using an average time of 1.40 seconds, 5.05 seconds and 13.48 seconds to solve the 5 x 5, 10 x 10 and 15 x 15 maze respectively. The total path length it took was 36 cells, 82 cells and 248 cells. It was able to do so because it had an initial understanding on the given maze and it does not require a person to enter and solve it, thereby using a shorter time to solve the maze respectively.

A* algorithm was able to obtained the shortest time in solving the maze as compared to LHR and BFS algorithm, requiring an average time of 0.85 seconds, 1.92 seconds and 6.08 seconds to solve the 5 by 5, 10 by 10 and 15 by 15 maze respectively. Just like BFS algorithm, A* algorithm had also used 36 cells, 82 cells and 248 cells to finish the given maze. The reason behind this was that the algorithm consists of both movement cost and heuristic cost. Unlike BFS algorithm, A* was able to find the exit faster without considering every path in the maze, thereby using shorter time to solved the maze. The summary of the comparison is listed in TABLE IV.

TABLE IV. SUMMARY OF COMPARISON BETWEEN THREE SELECTED PATH PLANNING ALGORITHMS

| Parameter | A* | BFS | LHR |
|---|---|---|---|
| Average Time Taken (s) | Shortest | Moderate | Longest |
| Total Path Length | Lowest | Lowest | Highest |

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

Based on the comparison made on each path planning algorithm on different size of maze, the most appropriate one to be chosen was A* algorithm. LHR algorithm had used the longest time and highest path taken whereas BFS algorithm had used moderate time and lowest path taken. On the other hand, A* algorithm was the only one that had used the shortest amount of time and path taken, thereby chosen as the appropriate one among the three.

### *B. Feature Added to the Path Planning Algorithm*

The A* algorithm was added with a new feature where 2D pygame capability was implemented. The implementation has provided user with more freedom and flexibility in designing the maze size instead of initializing it in the python code. The flowchart can be viewed in Figure 5:

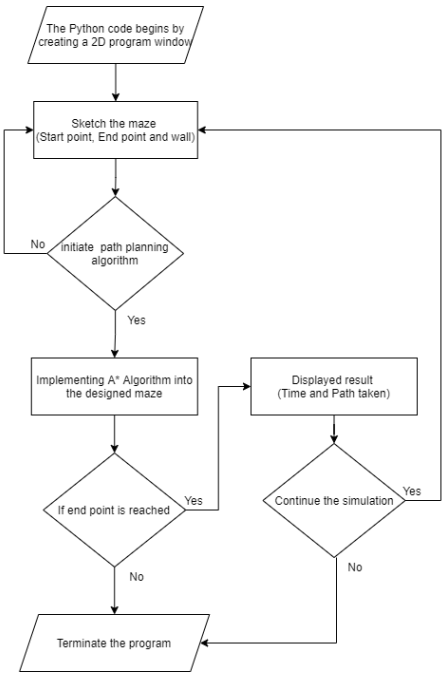



Fig. 5: Flowchart regarding to the added feature in A* algorithm.

A 2D program window is generated once the python code runs. At this moment, the user can freely design the maze based on their own discretion. The user can position the starting point, ending point and the wall of the maze. Besides that, the user can also remove the blocks if needed. The user's design must be able to ensure that the path planning algorithm is able to solve it, else the whole program will generate an error. Figure 6 shows the sample result of the added feature.

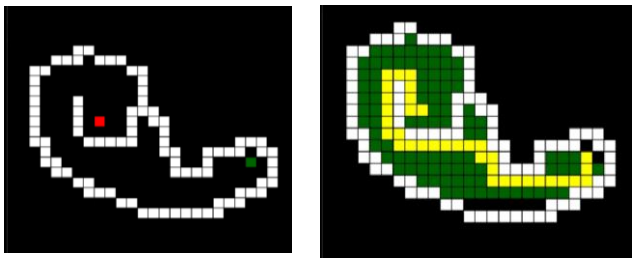


Fig. 6: Simulated results of the feature added to the path planning algorithm

### *C. Physical Maze and Navigation Robot*

Fig. 7: Physical Maze

A 3 by 3 physical maze is constructed as shown in Figure 7. The maze is attached using simple materials such as binder clips and carboards.

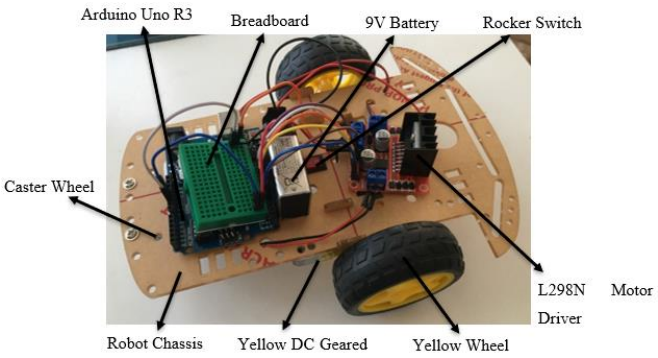



Fig. 8: Physical Navigation Robot

The physical navigation robot is constructed mainly to examine the path planning algorithm on a physical maze. The necessary components are listed alongside the constructed navigation robot in Figure 8.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

*D. Simulation*

```
2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2
-----------------------------------
2 | 1 | 1 | 1 | 2 | 0 | 2 | 2 | 2 | 2
-----------------------------------
2 | 2 | 2 | 1 | 0 | 0 | 2 | 2 | 2 | 2
-----------------------------------
2 | 0 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2
-----------------------------------
2 | 0 | 0 | 1 | 1 | 1 | 2 | 2 | 2 | 2
-----------------------------------
2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2
-----------------------------------
2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2
-----------------------------------
2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2
-----------------------------------
2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2
-----------------------------------
2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Goal Reached
.r r d d d r r
```

Fig. 9: Simulation Result in Arduino IDE Serial Monitor

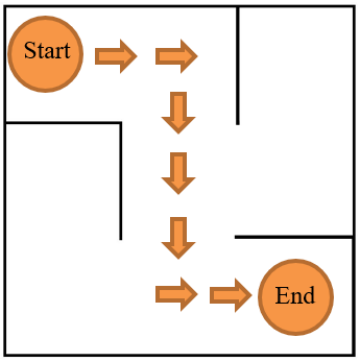
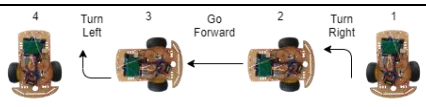
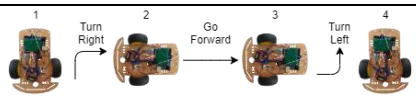
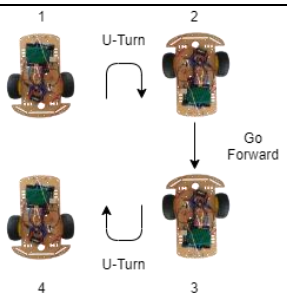



Fig. 10: The movement generated from Arduino to Navigation Robot

The feature added A* algorithm python code was written into the Arduino code. The information regarding the 3 by 3 physical maze is coded into the program as well for simulation purposes. The generated maze in Arduino can be seen in Figure 9. The information within the maze is initialized respectively, where 0 will represent the path, 1 will represent the start position, while 2 represents the wall and 3 represents the goal position. Once the goal is reached, letters will be displayed in the bottom side of the serial monitor with the direction of movement needed for the navigation robot. Figure 10 depicts the intended movement on the navigation robot to solve the maze. The respective movement for each direction can be elaborated in TABLE V.

TABLE V. RESPECTIVE DIRECTION AND MOVEMENT FOR THE NAVIGATION ROBOT

| Letter (Direction) | Movement of navigation robot |
|---|---|
| l (Left) |  |
| r (Right) |  |
| u (Up) |  |
| d (Down) |  |







Once the letters are displayed, the navigation robot will move accordingly to the given letter to solve the maze. Figure 11 shows the initial and final position of the navigation robot.

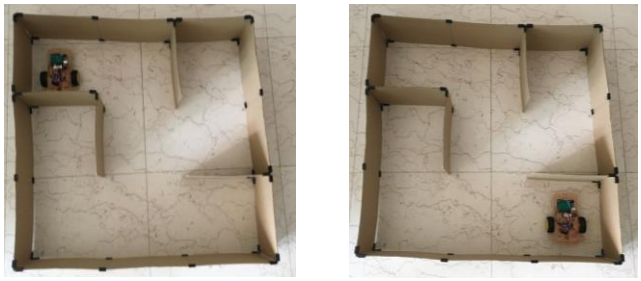


Fig. 11: Initial and Final Position of the Navigation Robot

## V. CONCLUSION

In a nutshell, the objectives stated in the report were achieved. Three path planning algorithms had undergone comparison test successfully using the python simulation program in PyCharm. Next, a new feature was added and implemented into the path planning algorithm where a free space with designing capability can be done through the incorporation of pygame library. The feature was added and tested successfully on a 2D window program simulation. Furthermore, the path planning algorithm was implemented into the physical navigation robot. The code was written in Arduino language whereby the grid was used for representing the maze virtually. The experiment was done successfully whereby the navigation robot was able to finish the given maze.

## ACKNOWLEDGMENT

I would like to express my gratitude and sincere thanks to my research supervisors, Dr. Lee Jer Vui and Ir. Dr. Chuah Yea Dat for their invaluable advices, guidances and enormous patience thoughout the development of the research.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 10 Issue 05, May-2021

## REFERENCES

[1] Gul, F., Rahiman, W., & Nazli Alhady, S. S. (2019). A comprehensive study for robot navigation techniques. *Cogent Engineering*, *6*(1).

[2] Sariff, N., & Buniyamin, N. (2006). An overview of autonomous mobile robot path planning algorithms. *SCOReD 2006 - Proceedings of 2006 4th Student Conference on Research and Development "Towards Enhancing Research Excellence in the Region,"* June, 183–188.

[3] Rahman, M. (2017). *Autonomous maze solving robot*.

[4] Aqel, M. O. A., Issa, A., Khdair, M., Elhabbash, M., Abubaker, M., & Massoud, M. (2017). Intelligent maze solving robot based on image processing and graph theory algorithms. *Proceedings - 2017 International Conference on Promising Electronic Technologies, ICPET 2017*, *October*, 48–53.

[5] Cui, S.-G., & Wang, H., & & Yang, L. . (2012). *A simulation study of A-star algorithm for robot path planning*. 506–510.

[6] Tjiharjadi, S., Wijaya, M. C., & Setiawan, E. (2017). Optimization maze robot using A* and flood fill algorithm. *International Journal of Mechanical Engineering and Robotics Research*, *6*(5), 366–372.

[7] Khantanapoka, K., & Chinnasarn, K. (2009). Pathfinding of 2D & 3D game real-time strategy with Depth Direction A* algorithm for multi-layer. *2009 8th International Symposium on Natural Language Processing, SNLP '09*, 184–188.

[8] Warren, C. W. (1993). *Fast Path Planning Using Modified A * Method*. 662–667.

[9] Gilbert, G., & Rivera, A. (2012). *Path planning for general mazes*. MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY.

[10] Ė, F. D., Babinec, A., Kajan, M., Be, P., & Florek, M. (2014). *Path planning with modified A star algorithm for a mobile robot*. *96*, 59–69.

[11] Babula, M. (2009). Simulated Maze Solving Algorithms through Unknown Mazes. *Proceedings of the CS&P 2009, Concurrency, Specification & Programming*, *September*, 13–22.

[12] Kumar, N., & Kaur, S. (2019). A Review of Various Maze Solving Algorithms Based on Graph Theory. *International Journal for Scientific Research & Development*, *6*(12), 2–6.

[13] Sadik, A. M. J., Dhali, M. A., Farid, H. M. A. B., Rashid, T. U., & Syeed, A. (2010). A comprehensive and comparative study of maze-solving techniques by implementing graph theory. *Proceedings - International Conference on Artificial Intelligence and Computational Intelligence, AICI 2010*, *1*(1), 52–56.

[14] Sharma, K., & Munshi, C. (2015). A Comprehensive and Comparative Study Of Maze-Solving Techniques by Implementing Graph Theory. *IOSR Journal of Computer Engineering (IOSR-JCE)*, *17*(1), 24–29.

[15] Hamada, K. (2013). A picturesque maze generation algorithm with any given endpoints. *Journal of Information Processing*, *21*(3), 393–397.

[16] Hualong, J., Hongqi, W., & Yonghong, T. (2011). Design and realization of a maze robot. *2011 International Conference on Consumer Electronics, Communications and Networks, CECNet 2011 - Proceedings*, 201–204.

[17] Cai, J., Wan, X., Huo, M., & Wu, J. (2010). An algorithm of micromouse maze solving. *Proceedings - 10th IEEE International Conference on Computer and Information Technology, CIT-2010, 7th IEEE International Conference on Embedded Software and Systems, ICESS-2010, ScalCom-2010, Cit*, 1995–2000.