Solving a Reconfigurable Maze using Hybrid Wall Follower Algorithm

Abu Bakar Sayuti Saman
Department of Electrical and Electronic Enginnering
Universiti Teknologi PETRONAS
Seri Iskandar 31750 Malaysia

Issa Abdramane
Department of Electrical and Electronic Enginnering
Universiti Teknologi PETRONAS
Seri Iskandar 31750 Malaysia

ABSTRACT
A key feature of an autonomous vehicle is the ability to get to a target location while traversing through a previously unknown environment. Mapping the environment will allow the vehicle to find an optimum path. This paper explores this issue by programming a mobile robot to find the shortest route in a reconfigurable maze. A wall follower algorithm with combined left-hand and right-hand rules is implemented upon several different maze configurations. It is found that the hybrid algorithm has improved the maze solving capabilities of the maze robot significantly.

General Terms
Autonomous, navigation, robotics, localization, mapping.

Keywords
Reconfigurable maze, path optimization, micromouse, wall follower.

1. INTRODUCTION
An autonomous vehicle is basically a mobile robot that does not require external assistance in order to drive itself from one location to another. It must have a localization system to know its location relative to the surrounding. It uses a mapping system to gather important information such as obstacles, terrain and path from the environment it is in.

On a terrestrial open space, data from the surrounding terrain is detected using many types of sensors such as ultrasonic sensors, laser range finders and cameras [1,2]. The robot also can rely on global reference point such as GPS coordinates. In some other cases such as autonomous in a building, the system needs to be able to map its surrounding by getting information from its environment. In confined spaces, such as in caves [3], underground tunnels, or under the rubbles of a collapsed building, the robot can only rely on limited sensors. This is because the overall size of the robot must be relatively small and light so that it is more maneuverable. It has to depend on simple and small and low power sensors such as ultrasonic sensors and infrared sensors.

In developing mapping algorithm for an autonomous robot in a confined space described above, a maze can be used to simulate the environment. Many algorithms for maze navigation and maze solving have been developed and continue to be improved over the years.

A famous competition where the algorithms are put to test is called Micromouse, an international event which is very popular in the United Kingdom, Japan, India and South Korea. The main idea of the event is to provide maze robots with a competitive arena in finding a target point through the shortest path possible and with the least amount of time. Initially, a robot will navigate the maze to find the target point. Once the target point is located, the robot will identify the shortest path. In the second round, the robot should be able to navigate the maze through the shortest path and shortest time towards its goal [4,5].

2. MAZE-SOLVING ALGORITHMS
Some of classic maze solving algorithms usually employed are random mouse, wall follower and flood fill algorithms. The wall follower algorithm is commonly used when the position of the target point is unknown. The target is usually identified with a unique marking. On the other hand, the flood fill algorithm is commonly used when the position of the target point is geometrically known but the robot needs to find the shortest route. The following sections describe the flood fill and wall follower algorithms [6-9].

2.1 Flood Fill Algorithm
The flood fill algorithm is by far the most famous and efficient algorithm to solve all types of maze but commonly with a preset target point. The maze is made up of cells (x,y) that are represented by a two dimensional array. The cell that contains the target point is called the target cell, located at (0,0). Initially, the algorithm assigns to each cell a value that represents the distance between the cell and the target cell. Relative to the target cell, its immediate neighbouring cells have the distance values of 1. The cells next to them will contain values 2; and the next cells, values of 3; and so on [9,10].

The flood fill algorithm gets the current information of the cell that the robot resides in and predicts its distance from the end cell. While moving towards the goal, it updates all walls encountered and makes the correct turn if it has to. Based on the assumption of the goal point, the robot should be able to make the correct turn and avoid taking unnecessary routes algorithm. Figure 3 shows a flow chart abstracting a typical flood-fill algorithm [5,11].

2.2 Wall Follower Algorithm
In wall follower algorithm, the robot will keep an eye at the right or left wall and navigate throughout the maze until it finds the target point. This algorithm is proven to be very efficient for mazes that are wall-linked to the target point [4, 6]. This makes it very suitable for mazes where the target point is located at the periphery, a situation where the robot almost appears like trying to escape the maze.

There are two types of wall follower algorithm: left-hand rule and right-hand rule. The two algorithms work the same way except turning priority will be either to the left or to the right depending on the type of rule used [5,6].

2.2.1 Left-hand Rule

The left-hand rule works in such a way that the robot focuses more on its left-side and front-side while it has options for turns. The robot will turn right only if there are no other possibilities while it always turns to the left if there is an option, as illustrated by the flowchart in Figure 1 [12].

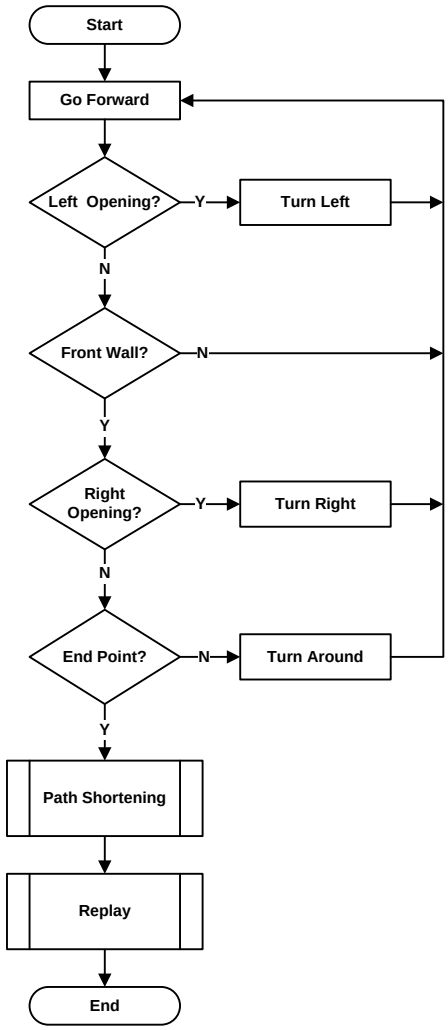

Fig 1: Flow-chart for wall follower algorithm using the left-hand rule navigation

A pseudo-code for wall follower algorithm with the left-hand rule is as follows:

```
# Wall follower, Left-hand Rule
While Not target_point
If left is open Then
    turn_left
Else If front is open Then
    go_forward
Else If right is open Then
    turn_right
Else
    turn_around
Loop
```

When applied to a maze, the algorithm will not necessarily find the shortest path to the destination cell. The maze shown in Figure 2 is more suitable for the right-hand rule method. When the left-hand rule is applied, the algorithm does not find the most optimum path the goal. In contrast, when right-hand rule is applied, a shorter path is found, as shown in Figure 3.

For the same type of maze, the performance of using left-hand and right-hand rule will differ in the performance of the navigation.

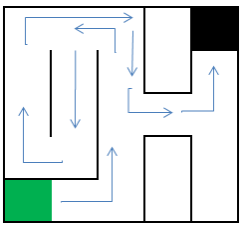

Fig 2: Left-hand rule implementation not the most efficient in this maze

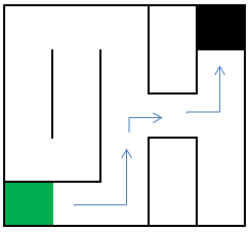

Fig 3: Right-hand rule implementation produces a better result but still not the shortest path

The algorithm for right-hand rule only slightly differs from the left-hand rule, as highlighted in the following pseudo-code:

```
# Wall follower, Right-hand Rule
While Not target_point
If right is open Then
    turn_right
Else If front is open Then
    go_forward
Else If left is open Then
    turn_left
Else
    Turn_around
Loop
```

2.2.2 Failure of Wall Follower Algorithm

A major drawback of this algorithm is that it can be used only on simple mazes where the target point is wall-linked. Prior knowledge whether the maze is left-walled or right-walled is indispensable or otherwise the robot will keep looping through the maze forever. All these factors made the wall follower algorithm not really suitable for maze solving competitions because of its lack of intelligence for the robot. Figure 4 illustrates an example of a failure of wall follower algorithm with left-hand rule when implemented in a maze where the destination is located in the middle.

3. IMPLEMENTATION OF WALL FOLLOWER ALGORITHM IN A RECONFIGURABLE MAZE

This paper describes the implementation of a wall follower algorithm on a variety of mazes. The base maze is fixed but the layout of the wall is reconfigurable. It is a continuation of previous related work whereby a fixed maze was used for an implementation of the flood fill algorithm [9]. This work is part of an on-going work in developing a highly intelligent autonomous navigation system for small mobile robot in a confined space.

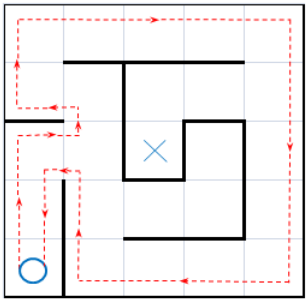

Fig 4: Failure of Left-hand Rule Wall Follower Algorithm

In the fixed maze, the target cell was fixed in the middle. The algorithm was used to find the shortest path to the goal and this path was saved in its memory so that it can always get back to the target cell using the shortest path.

In this work, the walls of the maze are detachable, thus the maze can be configured in many different configurations. However, the maze is still made up of cells of the same size. The target point is not fixed to a specific cell. This explores the usability of wall follower algorithm in finding a path that solve an unknown maze with an unknown target location and later optimizing the path.

To navigate different type of mazes, both left-hand and right-hand navigation rules of the wall-follower algorithm are adopted. In this case, the navigation rule is selected based on the first side opening encountered. This combination of both rules maximizes the capability of the robot to be able to navigate more complex type of mazes and also in some cases, will help to avoid unnecessary long navigations. Figure 5 illustrates the combined navigation rule used in the wall follower algorithm implementation.

4. RESULTS AND ANALYSIS

The use of the wall-follower algorithm with the combined navigation rules can greatly improve the navigation time of the vehicle. An experiment was conducted using Configuration A which is illustrated in Figure 6. The use of the hybrid algorithm is based on which side an opening is encountered first. If the left opening is encountered first, the left-hand rule is selected. Whereas if the right opening is encountered first, then the right-hand rule algorithm is selected.

From the figure, one can see that the choice of the right hand rule algorithm is optimal. The two different line and arrow types indicate the choice taken by the vehicle based on the navigation rules used. The dashed lines show the path traversed using the left-hand rule only. The solid lines indicate the choice made by the hybrid algorithm to use the right-hand rule base on the location of first opening encountered on the right side of the robot. It clearly shows that the use of the hybrid algorithm provides more intelligence to the robot in navigating the maze.

When traversing the maze to find the target cell, the path is memorized by recording every move that have been executed. The moves are denoted by `F` (for moving forward), `L` (turning left), `R` (turning right) and `B` (turning back or making a 180^0 turn). Figure 7 illustrates all the moves recorded when navigating a maze of a simpler configuration, called Configuration B.

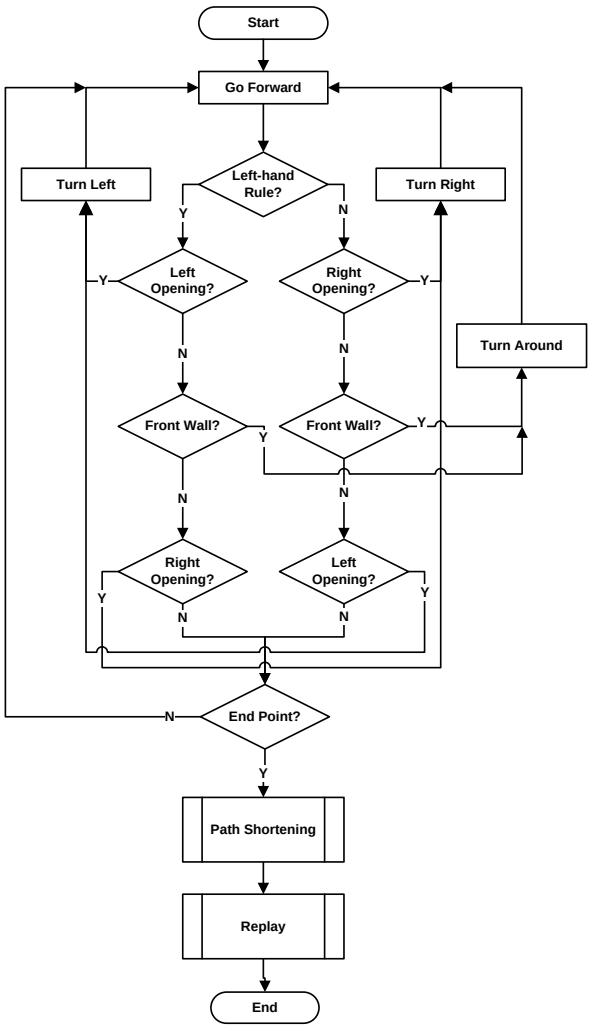

Fig 5: Flowchart of Hybrid Wall Follower Algorithm

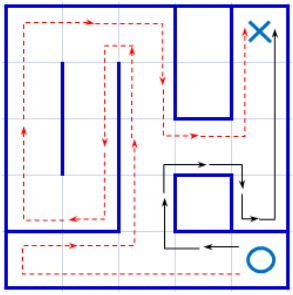

Fig 6: Optimum Navigation by Choosing the Right-hand Rule of the Wall Follower Algorithm in Configuration A

After the target cell is reached, the robot can be placed at its starting cell again and it will autonomously navigate to the target cell using the path that it has recorded. In this case the path is already optimum, thus no further processing was done.

4.1 Path Optimization

By choosing the correct navigation rule to follow, the path taken may be shorter but not necessarily the most optimum in terms of moves taken. The path may contain redundant moves. A move is considered redundant when it took the robot back to where it was before.

After the target cell has been found during the initial navigation, the whole path is analyzed so that it can be optimized. The algorithm goes through the list of moves in the path, identifies any sequence that contains a redundant move and replaces the sequence with an optimized i.e. shorter one.

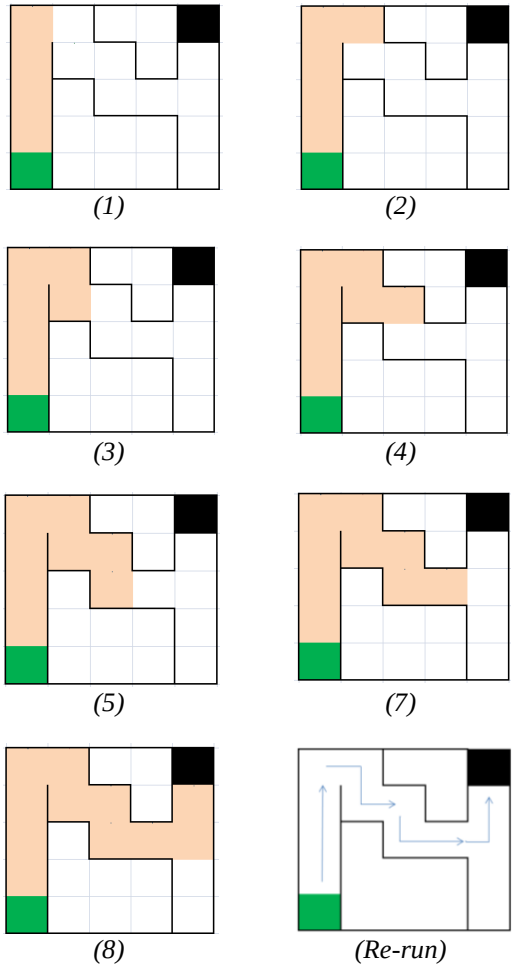

Fig 7: Navigation Through Configuration B

Primarily, the `B` move is always redundant. For example, consider a sequence `FBL` where the robot turns left, moves forward, makes a 180^0 turn and then turns left. This sequence is equivalent to `FR` where the robot only needs to move forward and then turn right.

Other similar sequences are analyzed this way. Some of the sequences that have been identified to contain redundancy and can be optimized are listed in Table 1 together with their shorter equivalents.

The path optimization is done through a path shortening algorithm described with the following pseudo-code.

```
for i = 1 to M
    if m[i] = B then
        if m[i-1] = L then
            n[j] = R
            j = j+1
        else if m[i-1] = L and m[i+1] = R then
            n[j] = B
            j = j+1
        else if m[i-1] = L and m[i+1] = F then
            n[j] = R
            j = j+1
        else if m[i-1] = R and m[i+1] = L then
            n[j] = B
            j = j+1
        else if m[i-1] = F and m[i+1] = L then
            n[j] = R
            j = j+1
        else if m[i-1] = F and m[i+1] = F then
            n[j] = B
            j = j+1
        else if m[i-1] = L and m[i+1] = L then
            n[j] = F
            j = j+1
    else if m[i-1] != B and m[i+1] != B then
            n[j] = B
            j = j+1
next i
```

Table 1. List of Move Sequences and Their Optimized Equivalents

Original Sequence	Shortened Sequence
LB	LR
LBR	LB
LBF	LR
RBL	RB
FBL	FR
FBF	FB
LBL	LF

The path shortening algorithm is based on the data in Table 1. In the pseudo-code: `M` is total number of moves in current list, `m[i]` is current move, `m[i-1]` is previous move, `m[i+1]` is next move, `n[j]` is current move in the new list of moves, `i` holds the counter for the current move being examined and `j` holds the counter for the current move in the new list.

The implementation of the path shortening algorithm is illustrated in Figure 8. In this implementation, the algorithm is applied onto the list of moves (path) several times until all the necessary `B` moves are removed.

When the robot is placed back at the starting point, it autonomously navigates towards the destination cell using the optimized path information.

5. CONCLUSION

A wall follower algorithm with selectable left-hand or right-hand navigation rule can provide more flexibility and intelligence for maze navigation. The selection of the navigation rule is done dynamically based the scenario encountered by the vehicle. With the addition of path optimization done through the path shortening algorithm, the combined system can provide practical improvement to autonomous vehicles.

While the algorithm improves the navigational capability of the robot, it is however unable to cope with certain maze configurations. A set of walls that form an island, will force the robot to encircle it indefinitely. An improvement being worked on is the detection of the infinite looping and the ability to switch to another navigation algorithm to exit from it.

Future works may also include other types of small sensors to improve the ability of the robot to detect its surrounding more

accurately. The processing power of the robot can also be improved by using faster and more powerful microcontrollers.

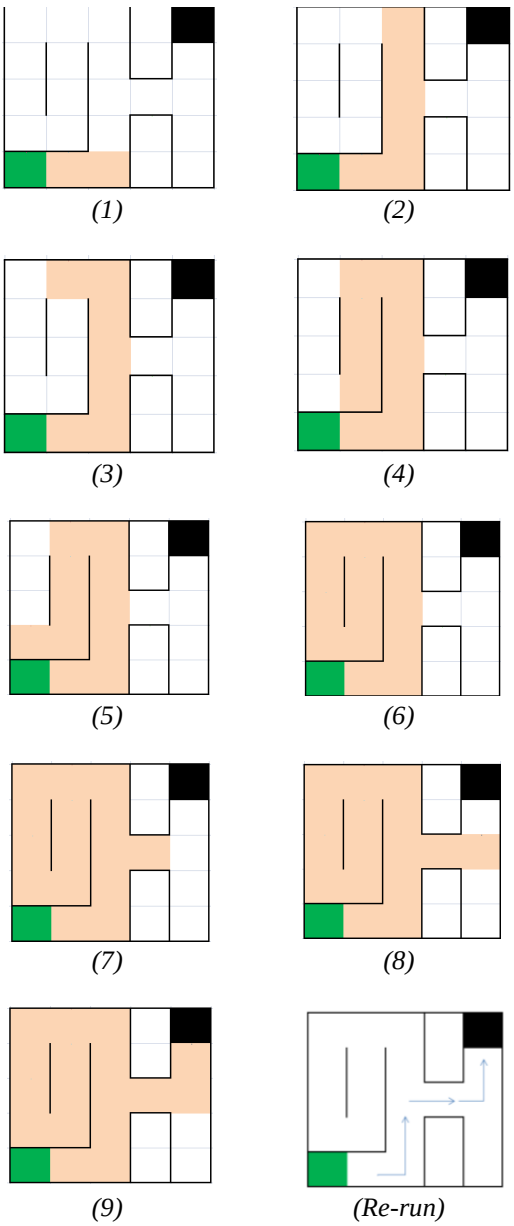

Fig 8: Navigation through Configuration C

6. REFERENCES

[1] Verner, I.M. and D.J. Ahlgren, Robot contest as a laboratory for experiential engineering education. J. Educ. Resour. Comput., 2004. 4(2): p. 2.

[2] Achmad, B. and M.N. Karsiti. Visual-based fuzzy navigation system for mobile robot: Wall and corridor follower. in Intelligent and Advanced Systems, 2007. ICIAS 2007. International Conference on. 2007. Kuala Lumpur.

[3] Vignesh, S., et al., Cave Exploration of Mobile Robots using Soft Computing Algorithms. International Journal of Computer Applications 71(22):14-18, June 2013. Published by Foundation of Computer Science, New York, USA.

[4] Mishra, S. and P. Bande. Maze Solving Algorithms for Micro Mouse. in Signal Image Technology and Internet Based Systems, 2008. SITIS '08. IEEE International Conference on. 2008.

[5] Cai, J., et al., An Algorithm of Micromouse Maze Solving, in Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology. 2010, IEEE Computer Society. p. 1995-2000.

[6] Jianping, C., et al. A micromouse maze sovling simulator. in Future Computer and Communication (ICFCC), 2010 2nd International Conference on. 2010.

[7] Adil, M.J.S. A Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory. 2010.

[8] Sharma, M. and K. Robeonics. Algorithms for Micro-mouse. in Future Computer and Communication, 2009. ICFCC 2009. International Conference on. 2009.

[9] Elshamarka, I. and A.B.S. Saman, Article: Design and Implementation of a Robot for Maze-Solving using Flood-Fill Algorithm. International Journal of Computer Applications, 2012. 56(5): p. 6.

[10] Cai, Z., L. Ye, and A. Yang. FloodFill Maze Solving with Expected Toll of Penetrating Unknown Walls. in 2012 IEEE 14th International Conference on High Performance Computing and Communications. 2012.

[11] Willardson, D.M., Analysis of Micromouse Maze Solving Algorithm, in Learning from Data. 2001, Portland State University.

[12] Babula, M. Simulated maze solving algorithms through unknown mazes. in Proceedings of XVIIIth Concurrency, Specification and Programming (CS&P) Workshop. 2009.