
FINAL YEAR PROJECT 2

FINAL REPORT

# DEVELOPMENT OF ALGORITHM FOR IMPROVED MAZE NAVIGATION

BY

FAIZ AYDIL BIN AHMAD

14682

ELECTRICAL AND ELECTRONICS ENGINEERING

SV: MR. ABU BAKAR SAYUTI B. M. SAMAN

ABSRACT

Autonomous navigation is a crucial technology that helps a mobile robot to move independently and navigating through unknown areas that are impossible for human to venture due to limitation of physical abilities or even danger that may threaten life. By solving a maze, the algorithms and behaviour of the robot can be studied and improved. This paper describes the development of algorithm for improved maze navigation and it is a continuation on a previous project. Detection of walls and opening in the maze were accomplished using ultrasonic range-finders. The robot will be able to learn the maze, find all possible routes and solve it using the shortest one.

TABLE OF CONTENT

**LIST OF FIGURES:**



**LIST OF TABLES:**

# CHAPTER 1: INTRODUCTION

## BACKGROUND

Robotic field has caught a big attention to our world as it conveniently increases efficiency of especially monotonous work. This field does include the industrial robots and mobile robots that is used for various purposes. As the fields become open for everyone, a lot of competition such as MicroMouse competition started to build up as to making the best robot. In the MicroMouse competition, it normally uses 256 square units of maze. Micromice or the robot will compete to solve the maze without the needs of any manual assistance. In order to do so, suitable algorithm should be use to solve the maze in the least time possible.

## PROBLEM STATEMENT

The problem faced is that whether the algorithm will be using wall-following, flood-filling, A* search algorithm and many more method possible or whether to combine two or more algorithm in a robot. There are pro and cons for every each of the algorithm that may varies the results in different efficiency and effectiveness in solving the maze. Other consideration is that how much time will it take to find the destination as the robot itself will take time to process it's surrounding depending on the algorithm being used by the robot.

## OBJECTIVES AND SCOPE OF STUDY

This study is aiming on producing a mobile robot that can solve and navigate through a maze without making any collisions with the walls within the maze itself. It will be using PIC microcontroller as the central processing unit for the whole robot itself. Note that this project is a continuation to a previous project that focuses on algorithm development.

The objectives of the study is mainly:

i. To build a mobile robot capable in navigating and solving a maze.

ii. To study the best algorithm of navigating and solving the maze.

iii. To implement the maze-solving algorithm on a real maze robot.

The scope of study for this project is:

I. Software: Studying the algorithm used in maze solving robot and developing a hybrid algorithm that will perform better in the real environment

II. Hardware: Further study in terms of fabricating the robot model which includes tuning and synchronizing each component to ensure error free

## CHAPTER 2: LITERATURE REVIEW & THEORY

The most crucial thing to be decided in this project is the algorithm that it will be using to navigate through and solve the maze. The hardware part that is the sensor, PIC microcontroller, motors and others can easily be determined after the algorithm is decided which is of course, following the necessities of the algorithm in terms of those hardware requirement.

In general, the problem faced by any developer for this project is that the real mobile robot in real environments does not work as accurate as the simulation of the mobile robot navigating through the maze. This is due to many factor which are mostly related to the hardware configuration of the robot itself. Sometimes errors from the sensors and, errors from the motors and the wheels such as slipping and other kinds of hardware intrusion affects a lot in the mission of navigating through the maze [2]. In order to reduce these kind of errors, the motor need to be calibrated before being used in the robot itself. The sensors must be damage-free so that it will reduce the probability of giving faulty inputs.

In determining the algorithms, the most efficient and effective method must be chosen. As each of the method proposed has their own pros and cons, finding the method with least disadvantages may help in making this project a successful project. As the wall-following method may took time in determining its route, it is the convenient way of finding the destination set in the maze. The flood-fill technique also take quite amount of time as it will have to undergo four steps before it moves for each cell it goes through that is update walls, flood maze, turn determination and moving to the next cell [1]. The A* search algorithm uses nodes as it paths in determining the shortest route to the destination. However, this method are mostly not suitable for maze navigation as it took time to determining every each path for every each cells in the maze.

## CHAPTER 3: METHODOLOGY/PROJECT WORK

Research Methodology:

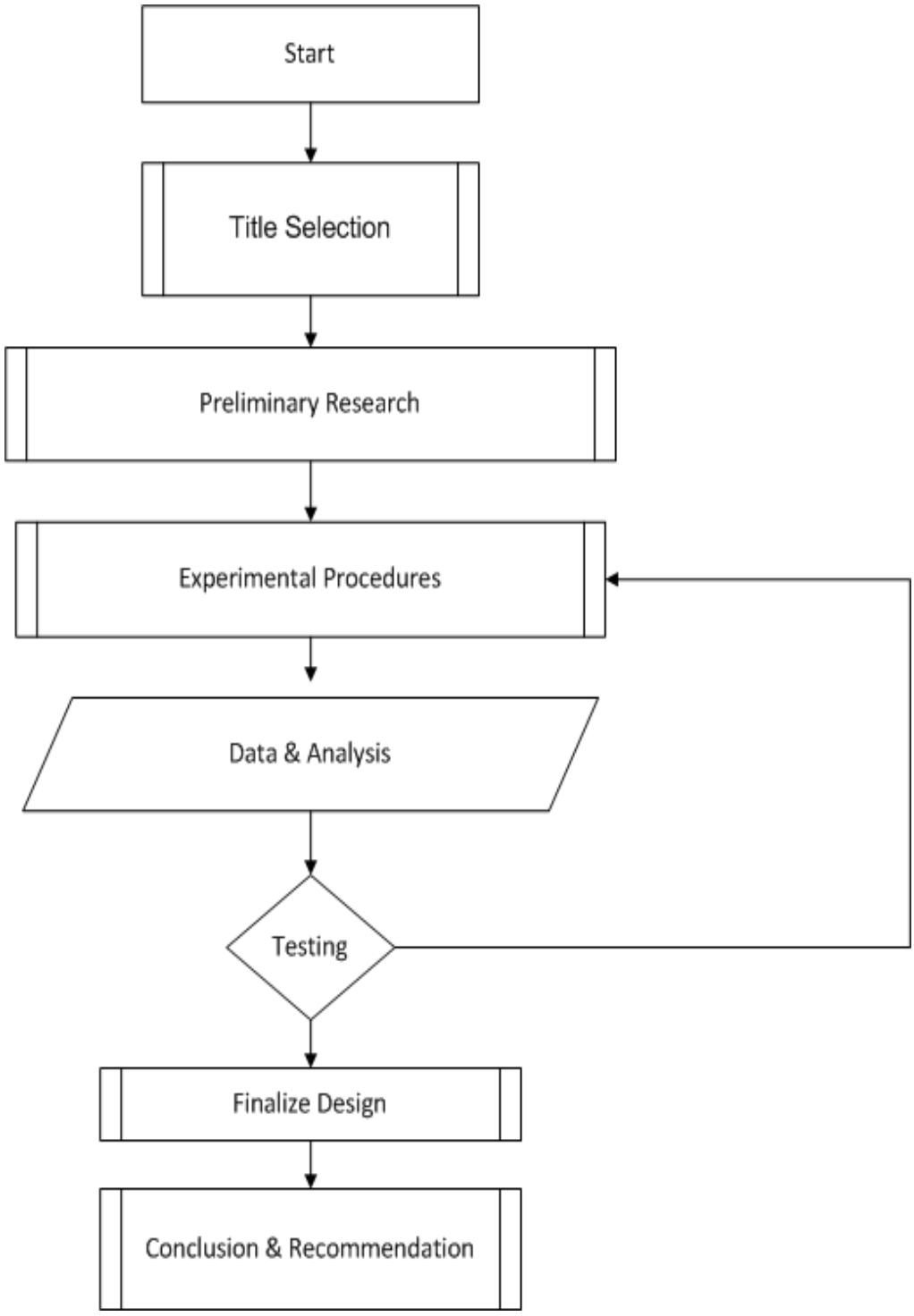


*Figure 1: Methodology of the project*

Project Activities:

Preliminary Research

In the progress of completing the prototype for the maze robot after purchasing all equipment that are required for the project. All calibration is executed including sensor calibration, motor calibration and most importantly the development of the software code including the improved flood-fill algorithm for the maze solving robot.

Experimental Procedure

Tuning and calibration of the motor, sensor along with the coding are conducted to ensure synchronization of all the equipment needed for the robot. Basic algorithm are used to test the robot to ensure it worked well. Then only after being ensured of its functionality, the improved flood-fill algorithm is included in the codes.

Data and Analysis

Collect results from the simulation of RobotBasic and Arduino Compiler. The simulation result is analyses on the range of effectiveness of the ultrasonic sensor and also how the robot movement as it go through the maze based on the coding. Then real testing is executed to compare simulation and the prototype.

Testing and Finalized Design

All of the tested sensors are codes are being improved for maximum effectiveness of wall detection and maze solving. Results from the simulation and measurement will be compared. Robot are then being tested of its capability of detecting obstacles in the maze and avoided them.

Conclusion and Recommendation

The overall design parameters and result will be described and highlighted according to the results obtained from the testing.

- PROJECT FLOW/GANTT CHARTT

| Weeks | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Research work | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Purchasing | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Fabrication | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Hardware test | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Algorithm Navigation | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Algorithm Simulation | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

*Table 1: Project Flow/Gantt Chart*

# CHAPTER 4: RESULTS AND DISCUSSION

This chapter will explain about the findings and discussion of the project itself including the testing of the sensors used in the maze robot solver.

## Microcontroller programming

There are many options of microcontroller that can be used in this project. Although, we will only be comparing the most used microcontroller that is Arduino and PIC. The best microcontroller must be choose so that it meet the programming code that may use quite a lot of memory and RAM. In comparison, PIC is only a chip whereas Arduino is a platform itself. So in terms of saving time and work, Arduino provides a very convenient function as the platform itself has all what it needs in doing the project whereas PIC might need a PCB board and any other component to make it as complete as a platform like Arduino. However, PIC is much cheaper and in terms of data storage and RAM, some of the PIC does exceed the specification of the Arduino. Also, PIC might be the best option if the mobile robot platform itself uses PIC as its main microcontroller where the platform will be equipped with PIC itself. It fully depends on what mobile robot platform that will be used in this project. But as the platform of mobile robot does not include the microcontroller, it is decided that Arduino Uno will be used as it is the best option for this condition.


*Figure 2: Arduino Uno board*

Sensors

The suitable sensor that can be used in this project is ultrasonic. This due some of the ultrasonic sensor's advantages which are;

i. Able to measures and detects distances to moving objects.

ii. Impervious to target materials, surface and colour.

iii. Solid-state units have virtually unlimited, maintenance-free lifespan.

iv. Detects small objects over long operating distances.

v. Resistant to external disturbances such as vibration, infrared radiation, ambient noise and EMI radiation.

vi. Ultrasonic sensors are not affected by dust, dirt or high-moisture environments.

vii. Discrete distances to moving objects can be detected and measured.

viii. Less affected by target materials and surfaces, and not affected by colour. Solid-state units have virtually unlimited, maintenance free life. Can detect small objects over long operating distances.

Flood-fill algorithm

This algorithm will assign values to every each of the cell inside the maze whereas the values will represent the distance from any cell on a maze to the destination [4]. It is considered as the best algorithm in maze solving. It involves in assigning values to each of cells of maze where the values represent the distance from any cell on a maze to destination cell. The algorithm contain of four parts:

- Update walls
- Flood maze
- Turn determination
- Move to next cell

This will be the main algorithm that will be combine with other feasible algorithm to make the maze solving algorithm much more efficient by saving time in deciding.

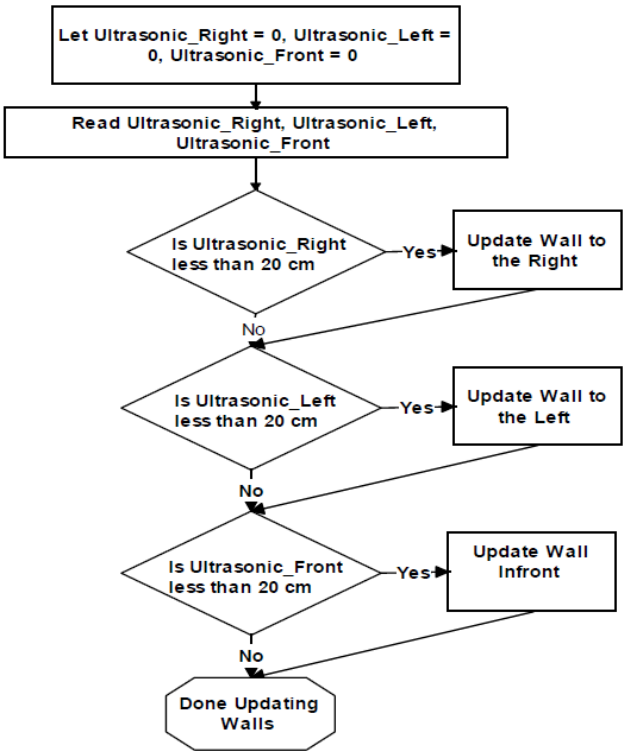


*Figure 3: Suggested flow chart for the flood-fill algorithm*

Mobile robot platform

*Figure 4: Mobile robot platform*

The image above shows the platform that will be used in this project. We decided to choose this platform as it is much more suitable in moving and navigating through narrow maze as it is much more compact in terms of the design. Also, considering the time factor, buying a ready-made platform instead of building itself helps reduce the time spent and more time can be used to develop the algorithm as it has the bigger priority.

Software

As the platform itself uses Arduino Mega as the microcontroller, the software that will be used is Arduino C Compiler itself for the programming part. For simulation of the robot, RobotBasic will be used to determine the effectiveness of the algorithm used for this project.

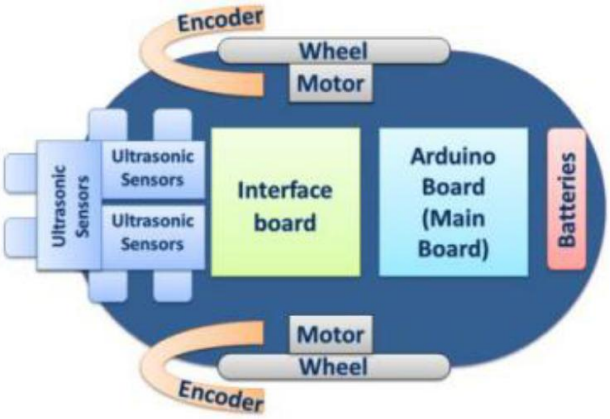


*Figure 5: Design of the robot*

The above figure shows the design of the robot. The ultrasonic sensor is placed in front of the robot for wall detection. The interface board will connect the ultrasonic sensors and the encoder to the Arduino board. For the movement of the robot, when turning left or right, one of the wheel will stop moving and the other will move according to the desired location. For the robot to map the whole maze, the circumference of the wheel will be noted in order to plan its movement using the algorithm.

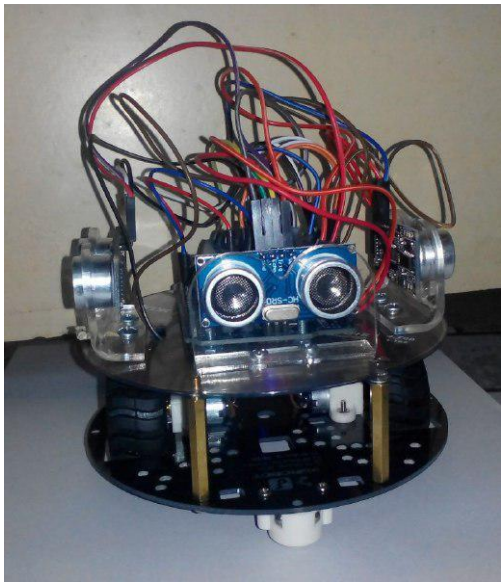

*Figure 6: Actual robot*

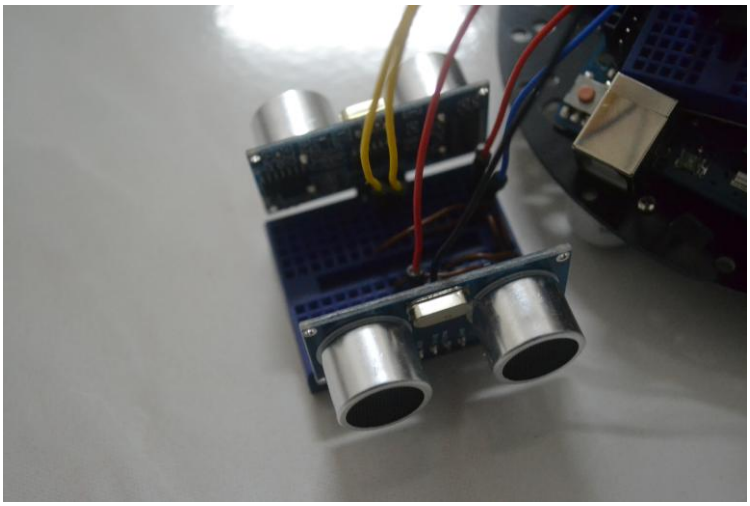

*Figure 7: Ultrasonic sensor used*

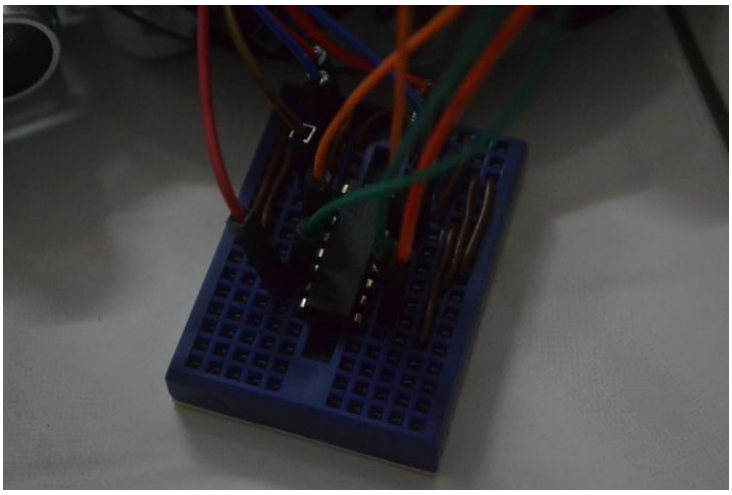

*Figure 8: L293D, motor driver used*

*Figure 9: Power source for the robot; 9V battery*

**Description for the robot development:**

Three ultrasonic sensor is used for the robot; front, left and right. The position of the left and right sensors will be placed 90 degrees as it has the function of detecting wall on each side and ensure it does not run to near or even crash into the wall. The front sensor will be used to ensure the robot to detect any obstacles in front of it and at the same time as an input for the algorithm when approaching a junction to choose the path that will lead it to the end point. In order to ensure that the sensors picks up the input that we need, a certain amount of delay time should be given in the codes so that the results will be more accurate and the robot can move smoothly. However, if the delay time is too long, it might cause the robot to be slow in deciding its action thus increase the amount of time for the robot to solve the maze. Basic code of algorithm is already applied to the robot for testing purposes. Only after being calibrated and synchronized with the sensors and motors the improved flood-fill algorithm will be included inside the coding. Problem faced so far is that the supply power from the Arduino board are not enough to make the motor move. Although the code works but because of the weak power supply, the robot can barely move. In order to solve the problem, a direct supply from the 9V battery will be drawn to the L293D motor driver so that it will have enough power to operate. Other problem faced is that the speed of the motor is not balanced so it might be able not move perfectly straight. To solve this problem, tuning is needed to make sure the motor does not have any zero error and thus enables it to move straight.

.

**Test run results:**

**Maze design 1:**

The robot is then tested to two design of maze to determine how well the robot solve the maze. As the maze is a reconfigurable maze, it makes the building of the maze much easier and not time consuming. The reconfigurable maze can then be used for the continuation or other project that requires such equipment.

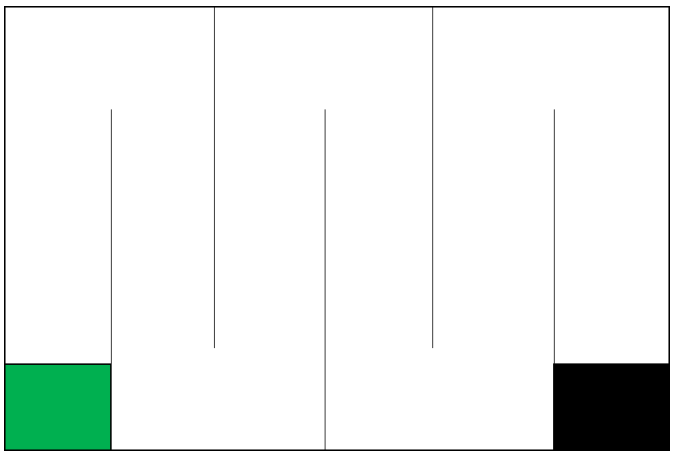

*Figure 10: Simple maze for testing*

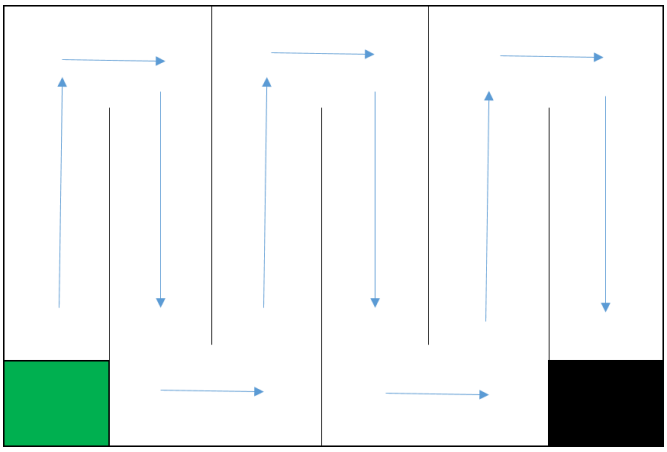

*Figure 11: Testing result from the robot*

The figure above shows the result of the testing for the simple maze. No major problem occurred while the robot is navigating through the maze. Although, sometimes the robot take some delay in turning to the correct path due to unsynchronized coding with real test. After being tested again with making the delay a bit shorter, the robot run smoothly through the maze.

**Maze design 2:**

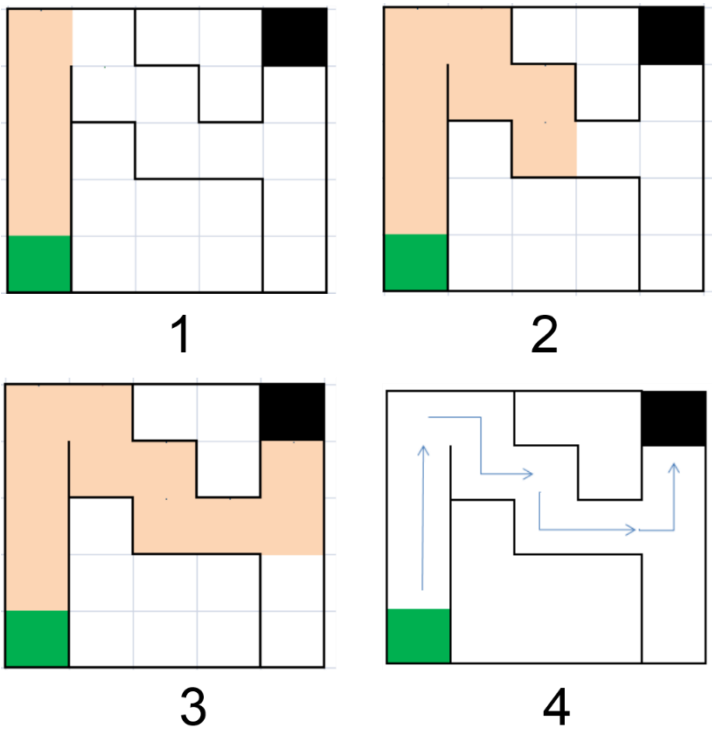


*Figure 12: Result for test run with another maze design*

The green spot indicates as the starting point for the robot while the black spot indicates the end point. While the peach colour shows the path taken by the robot. The result from the test run shows that the robot are able to find the end point of the maze. Although, the maze is quite simple and not challenging enough to really know how well the algorithm works.

## CHAPTER 5: CONCLUSION & RECOMMENDATION

As a conclusion, the maze-solving robot is an algorithm development project. It requires knowledge on both hardware and software in Arduino board, DC motor, sensors as well as motor driver circuit. The design of the robot does affect how well the robot can perform on a real maze. Also, the tuning of both sensor and motor are very important as it determines how smooth the robot can explore inside the maze. Therefore, the motor should be made zero-error free and the ultrasonic should be tested and then the codes should be adjusted according to the range of detection of the sensor. Also, the choosing of power source for the robot does determine how the motor will efficiently move without being disturbed by lack of power or anything related to the power source problem. If the budget for the project is high, investing on a high quality ultrasonic sensor will highly increase the efficiency of the obstacles detection that may help improves the time for the robot to solve the maze. In order to put this project to a whole new level, not only the improvement of navigation algorithm need to be developed to enhance the technology in navigation robot and push the robot to more extreme condition, but also the improvement on used sensor in project such as GPS tracking and image processing does make the robot much more reliable in any terrains it will be exploring.

## REFERENCES


[1] H. Dang, J. Song and Q. Guo, "An Efficient Algorithm for Robot Maze-Solving," School of Electric and Information Engineering, Shaanxi University of Science and Technology, China, 2010.

[2] B. H. Kazerouni, M. B. Moradi and P. H. Kazerouni, "Variable Priorities in Maze-Solving Algorithms for Robot's Movement," Students of Electrical and Computer Engineering, Shahid Behesti University, 2003.

[3] N. Yilmaz and S. Sagiroglu, "Web-Based Maze Robot Learning Using Fuzzy Motion Control System," Faculty of Engineering-Architecture, Selcuk University, Turkiye, 2007.

[4] Wikipedia. (2014, August 16). *Flood-fill* [Online]. Available: http://en.wikipedia.org/wiki/Flood_fill

[5] J. Dixon, O. Henlich. (2014, October 16). Maze Robot Navigation [Online]. Available: http://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol4/jmd/

## APPENDICES

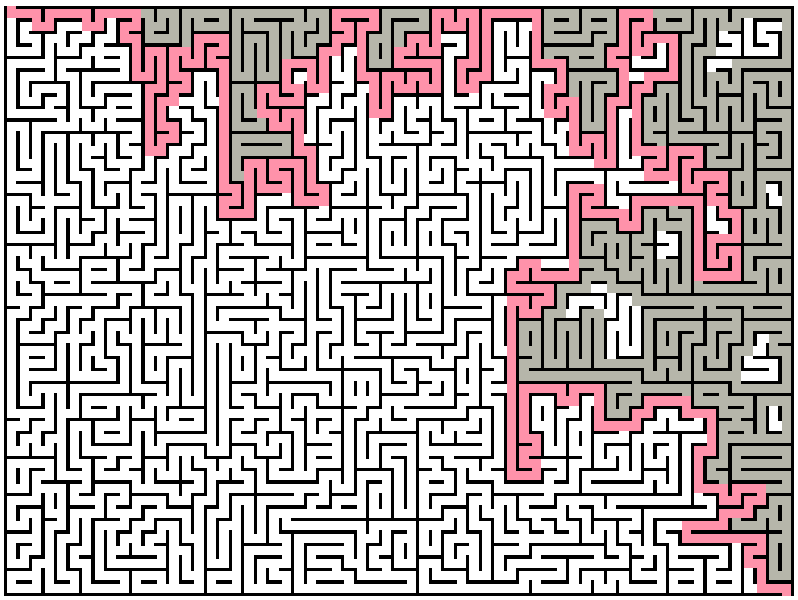

*Figure 11: Wall Follower Maze Solution*

**Coding**

```
#include <NewPing.h>      // Ping Library
// PORTS - INPUT
#define SONAR_LEFT_ECHO 7
#define SONAR_RIGHT_ECHO 13
#define SONAR_FRONT_ECHO 17
#define SONAR_BACK_ECHO A5
// PORTS - OUTPUT
#define MOTOR_SPD_L 10  // Left Motor PWM out
#define MOTOR_SPD_R 11  // Right Motor PWM out
#define MOTOR_DIR_L 8  // Left Motor Direction
#define MOTOR_DIR_R 9  // Right Motor Direction
#define SONAR_LEFT_TRIG 6
#define SONAR_RIGHT_TRIG 12
#define SONAR_FRONT_TRIG 16
#define SONAR_BACK_TRIG A4
// VARIABLES - INPUT
int sonarLeftVAL = 0;
int sonarRightVAL = 0;
int sonarFrontVAL = 0;
int sonarBackVAL = 0;
// VARIABLES - OUTPUT

  int motorSPD_L = 0;
  int motorSPD_R = 0;
  // VARIABLES - SYSTEM
  int newMotorSPD_L = 0;
  int newMotorSPD_R = 0;
  int maxPing = 100; // Limit sensors to reading 100cm
  NewPing SONARLeft(SONAR_LEFT_TRIG, SONAR_LEFT_ECHO, maxPing); // TRIG, ECHO,
MaxDistance
  NewPing SONARRight(SONAR_RIGHT_TRIG, SONAR_RIGHT_ECHO, maxPing);
```

```
NewPing SONARFront(SONAR_FRONT_TRIG, SONAR_FRONT_ECHO, maxPing);
NewPing SONARBack(SONAR_BACK_TRIG, SONAR_BACK_ECHO, maxPing);
int basicVelocity = 0;  // How much the CI feels like it should be moving forward
int urgTurn_L = 0;   // urge to turn to the Lefteft
int urgTurn_R = 0;   // urge to turn to the Right
int urgMotor_L = 0;   // urge to move Left motor forward
int urgMotor_R = 0;   // urge to move Right motor forward
int urgFatigue = 0; // Determines motor acceleration rate
// ---------- SYSTEM SETUP ----------
void setup() {
  // SETUP I/O
  pinMode(MOTOR_DIR_L, OUTPUT);
  pinMode(MOTOR_DIR_R, OUTPUT);
  //pinMode(SONAR_BACK_TRIG, OUTPUT);
    Serial.begin(9600);
  ResetSystem();
}
// // ---------- MAIN LOOP ----------
void loop() {
  ReadSensors();
  AvoidWalls();  // Determine actuator speeds based on sensor data
```

```
 SetMotors();   // Convert L&R speeds to motor control data
 SerialDebug(); // OUTPUT SERIAL DEBUGGING INFO
}
void ResetSystem() { // ----------- STOP -----------
 digitalWrite(MOTOR_DIR_L, LOW);     // MOTOR FORWARD DIRECTION
 digitalWrite(MOTOR_DIR_R, LOW);   // MOTOR FORWARD DIRECTION
 analogWrite(MOTOR_SPD_L, 0);       // MOTOR SPEED ZERO
 analogWrite(MOTOR_SPD_R, 0);       // MOTOR SPEED ZERO
 delay(10);


 }
 // ------------- READ SENSOR VALUES AND STORE -------------
 void ReadSensors() {
  // ----- SONAR -----
  delay(5); // delay to avoid echos between sensors
  sonarLeftVAL = SONARLeft.ping_cm();  // Get distance in cm from sonar device
  if (SONARLeft.check_timer() == 0) sonarLeftVAL = maxPing;  // set measurment to max instead of 0 if
distance is large.
  delay(5); // delay to avoid echos between sensors
  sonarFrontVAL = SONARFront.ping_cm();
  if (SONARFront.check_timer() == 0) sonarFrontVAL = maxPing;
  delay(5); // delay to avoid echos between sensors
  sonarRightVAL = SONARRight.ping_cm();
  if (SONARRight.check_timer() == 0) sonarRightVAL = maxPing;
  delay(5); // delay to avoid echos between sensors
  sonarBackVAL = SONARBack.ping_cm();
  if (SONARBack.check_timer() == 0) sonarBackVAL = maxPing;
 }
 // ------------- PROCESS AND INTERPRET SENSOR DATA -------------
 void AvoidWalls() {
  // GET/SET URGE VALUES
```

```
basicVelocity = 150; // Set above 0 to make it continuously move forward
urgMotor_L = 0;
urgMotor_R = 0;
urgFatigue = 0;
// AVOID WALLS AT SIDE
urgTurn_L += maxPing*maxPing - ((maxPing-sonarRightVAL) * (maxPing-sonarRightVAL));
//inverse proportional to square of rightval
urgTurn_R += maxPing*maxPing - ((maxPing-sonarLeftVAL) * (maxPing-sonarLeftVAL));
urgMotor_L -= 0.1*(maxPing*maxPing - ((maxPing-sonarRightVAL) * (maxPing-sonarRightVAL)));
urgMotor_R -= 0.1*(maxPing*maxPing - ((maxPing-sonarLeftVAL) * (maxPing-sonarLeftVAL)));



 // AVOID OBJECTS IN FRONT
 urgMotor_L += maxPing*maxPing - 0.5*((maxPing-sonarFrontVAL) * (maxPing-sonarFrontVAL)) -
((maxPing-sonarFrontVAL) * (maxPing-sonarFrontVAL));
 urgMotor_R += maxPing*maxPing - 0.5*((maxPing-sonarFrontVAL) * (maxPing-sonarFrontVAL)) -
((maxPing-sonarFrontVAL) * (maxPing-sonarFrontVAL));
  // SCALE URGES TO PWM output values (255)
 urgTurn_L = 255 - map(urgTurn_L, 0, 1.8*maxPing*maxPing, -255, 255);  // Scale to within PWM
output limits
 urgTurn_R = 255 - map(urgTurn_R, 0, 1.8*maxPing*maxPing, -255, 255);
 urgMotor_L = map(urgMotor_L, 0, 1.8*maxPing*maxPing, -255, 255);  // Scale to within PWM output
limits
 urgMotor_R = map(urgMotor_R, 0, 1.8*maxPing*maxPing, -255, 255);
// SET MOTOR SPEED
 newMotorSPD_L = basicVelocity + urgMotor_L + (urgTurn_R/4) - (urgTurn_L/2) + 60;
 newMotorSPD_R = basicVelocity + urgMotor_R + (urgTurn_L/4) - (urgTurn_R/2) + 60;
  // Clip to 255/-255 (negative value means reverse direction)
 if (newMotorSPD_L > 255) newMotorSPD_L = 255;
 if (newMotorSPD_L < -255) newMotorSPD_L = -255;
 if (newMotorSPD_R > 255) newMotorSPD_R = 255;
 if (newMotorSPD_R < -255) newMotorSPD_R = -255;
}
// ---------- SET MOTOR DIRECTION AND PWM OUTPUTS ----------
```

```
void SetMotors() { // Convert calculated speed changes to actuator control data
 int lrgSpdDelta = 0; //Used to store largest change in speed
 // SET MOTOR DIRECTION
 if (newMotorSPD_L < 0) {
  digitalWrite(MOTOR_DIR_L, LOW); // MOTOR L REVERSE
 } else {
  digitalWrite(MOTOR_DIR_L, HIGH); // MOTOR L FWD
 }
 if (newMotorSPD_R < 0) {
```

```
      digitalWrite(MOTOR_DIR_R, LOW); // MOTOR L REVERSE
     } else {
      digitalWrite(MOTOR_DIR_R, HIGH); // MOTOR R FWD
     }
     if (urgFatigue > 0) { // If using acceleration
      // CALCULATE SPEED CHANGES
      int dltaSpd_L = newMotorSPD_L - motorSPD_L;
      int dltaSpd_R = newMotorSPD_R - motorSPD_R;
      // Find largest speed difference
      if (abs(dltaSpd_L) >= abs(dltaSpd_L)) {
       lrgSpdDelta = abs(dltaSpd_L);
      } else {
       lrgSpdDelta = abs(dltaSpd_R);
      }
      // ACCELERATE MOTORS
      for (int i=0; i<lrgSpdDelta; i++) {
       if (newMotorSPD_L < motorSPD_L) motorSPD_L--;
       if (newMotorSPD_L > motorSPD_L) motorSPD_L++;
       if (newMotorSPD_R < motorSPD_R) motorSPD_R--;
       if (newMotorSPD_R > motorSPD_R) motorSPD_R++;
       analogWrite(MOTOR_SPD_L, abs(motorSPD_L));
  analogWrite(MOTOR_SPD_R, abs(motorSPD_R));
  delay(urgFatigue);// Determines Acceleration
  }
 } else {
  motorSPD_L = newMotorSPD_L;
  motorSPD_R = newMotorSPD_R;
  analogWrite(MOTOR_SPD_L, abs(motorSPD_L));
  analogWrite(MOTOR_SPD_R, abs(motorSPD_R));
 }
```

```
}// END SetMotors
// ---------- SEND DEBUG INFO ON SERIAL PORT ----------
void SerialDebug() {
  Serial.print("Sonar LEFT = ");
  Serial.println(sonarLeftVAL);
}
```