ROBOT IN A RECONFIGURABLE MAZE

By

Issa Abdramane

FINAL PROJECT REPORT

Submitted to the Department of Electrical & Electronic Engineering
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronic Engineering)

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

ROBOT IN A RECONFIGURABLE MAZE

by

Issa Abdramane

A project dissertation submitted to the
Department of Electrical & Electronic Engineering
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronic Engineering)

Approved:

_____

Mr. Abu Bakar Sayuti
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

June 2013

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

_____

Issa Abdramane

ABSTRACT

Autonomous vehicles have existed since several decades and they continue to evolve promptly. Their usage in different domains made them an interesting area for academic researchers as well as governments' projects. Issues that are still holding back the development of autonomous vehicles are the accurate mapping and localization of the surrounding that enable these vehicles to perform independently in a precise manner.

Using either the left-wall following or right-wall following algorithm alone will sometimes result in the robot being stuck in a loop and failed to solve the maze. This report describes a hybrid method where one of the two algorithms is selected based on the first opening of a reconfigurable maze. It has been demonstrated that by combining the two algorithms, unless the maze was purposely configured containing a loop, the rate of success is more than 90 percent.

ACKNOWLEDGEMENTS

First of all, I direct all my gratitude to God the Almighty for guiding me through all my steps in everything I am involved in.

I would like to express my gratitude to Mr. Abu Bakar Sayuti for all his patients and passion while supervising me throughout the final year project timeline. All the guidance and corrections provided were invaluable. All the discussion we had, were very motivating and fruitful. You have made the learning very easy and enjoyable.

I would like to extend my greatest gratitude and appreciation to my family for the constant motivation and support whenever life has given me hard time.

Last but least, I would like to thank everybody who helped me directly or indirectly in completing my project. I am grateful to have all your supports.

TABLE OF CONTENTS

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER 1: INTRODUCTION 1

1. 1. Background studies 1

1. 2. Problem statement 2

1. 3. Scope of study and objectives 2

CHAPTER 2: LITERATURE REVIEW 3

2. 1. The Micromouse competition 3

2. 2. The left-walled or right-walled algorithm 3

2. 3. The depth-first search 4

2. 4. The flood-fill algorithm 5

CHAPTER 3: METHODOLOGY 7

3. 1. Research methodology 7

3. 2. Project activities 7

3. 3. Project Gantt chart 8

CHAPTER 4: PROJECT COMPONENTS 9

4. 1. Description of the robot 9

4. 2. Arduino UNO 10

4. 3. Ping – Ultrasonic Distance Sensor 11

4. 4. IR Line Tracking Sensor (Single Bit) 12

4. 5. Reconfigurable maze for testing 13

CHAPTER 5: ISSUES AND CORRECTION 14

5. 1. HC-SR04 Ping Sensors issues 14

5. 2. Solution provided for the Ping sensors issues 15

CHAPTER 6: IMPLEMENTATION AND DISCUSSION 16

6. 1. Wall-Following Hybrid Algorithm 16

6. 2. Techniques adopted for the algorithm to work 18

6. 3. Mazes navigation 20

6. 4. Infinite loop cases 23

CHAPTER 7: IMPLEMENTATION OF HYBRID ALGORITHM 23

7. 1. Left-hand and right-hand rule 23

7. 2. Use of hybrid wall-follower algorithm 24

7. 3. Infinite loop cases 25

CONCLUSION 26

REFERENCES 27

APPENDIX 28

LIST OF FIGURES

Figure 01:	Failure of wall-follower algorithm	4
Figure 02:	Flow chart of a typical flood-fill algorithm	5
Figure 03:	Example of a robot caught in an infinite cycle	6
Figure 04:	a) Project Gantt chart – first semester	8
Figure 04:	b) Project Gantt chart – second semester	8
Figure 05:	Robot circuit diagram	9
Figure 06:	Arduino UNO R3	10
Figure 07:	Ping - Ultrasonic Distance Sensor	11
Figure 08:	IR Line Tracking Sensor (Single Bit)	12
Figure 09:	Example of types of mazes to used	13
Figure 10:	Reading from sensors	14
Figure 11:	Readings with different delays as a measure of correction	15
Figure 12:	Hybrid Wall-Follower algorithm	16
Figure 13:	The Left Hand Rule flowchart	17
Figure 14:	Navigation process in a simple maze	21
Figure 15:	Navigation in a more complex maze	22
Figure 16:	Right-Hand rule navigation	23
Figure 17:	Left-Hand rule navigation	23
Figure 18:	Hybrid wall-following algorithm	24

CHAPTER 1 INTRODUCTION

1. 1. Background Studies

Autonomous vehicles are basically vehicles that don't need external assistance in order to drive themselves from one location to another. These autonomous vehicles have gone through lots of stages of improvement and usage in different areas. Nowadays we count complete autonomous aerial vehicles used in the military domain as well as the complete driverless Google car that is about to be released for public.

Focusing on the terrestrial autonomous vehicles, they use the mapping and localization system to know their actual position. The surrounding is read by advance technology such as accurate ping sensors, line detectors, moving object detectors just to name few. Coming to the academic world, small robots are being developed to study the nature of mapping and localization for research purpose.

To make the research more fun and enjoyable, the micromouse event is invented to grasp more participation from educational institutions all the over the world. Initiated during the 1970s, the micromouse quickly got a reputation among higher learning institutions all over the world. It involves a completely autonomous robot to solve 16x16 mazes or 32x32 mazes depending on the rules and level of the competition.

The challenges in making these autonomous vehicles reside on the physical stability of the robots and the technique or algorithm that is required for the vehicles to be able to map its surroundings and move autonomously and accurately.

Taking the example of maze solving robots, there are numerous algorithms developed to solve different type of mazes. Most famous ones are the flood-fill algorithm and wall-follower algorithm. These approaches have their own strengths and weaknesses depending on the environment they are used in.

This project will be based on designing a small two-wheeled robot and development of an algorithm that is best suited for total reconfigurable maze navigation.

1. 2. Problem statement

Nowadays, there are lots of different types of robots that navigate different type of mazes. The approach to navigate mazes is different from one type to another, therefore knowing the rules in advance is more important before deciding which algorithm to use. Usually when robots are put in a situation where the maze configuration differs from what is predicted; these robots will lose their way out. Robot in a reconfigurable maze is aiming to provide a robot that navigate any kind of maze without prior knowing its type and still can find its destination point.

1. 3. Objectives and scope of study

The scope of this project will involve the design and fabrication of a small two-wheeled robot and development of an algorithm that will guide the robot to navigate autonomously. The fabrication part will involve a qualitative study of electronics components such as resistors, capacitors and alike to ensure that the robot life-span is prolonged with high performance.

The project will also require a good knowledge of programming skills to modify the existing algorithm or develop a new algorithm to better suit our need to achieve a robot that can navigate a reconfigurable maze.

The aim of this project is to design and build a robot that will be able to navigate a total reconfigurable maze. Therefore the objectives are:

- To reconstruct or design the robot to be more stable to suit the project needs;
- Produce a robot that is fast and accurate while navigating a maze
- Produce a robot that can navigate in a straight line between the walls of a maze
- Modify existing or develop a maze-solving algorithm to make the robot more intelligent in finding the goal point in any reconfigurable maze.

CHAPTER 2 LITERATURE REVIEW

Maze-solving robots are subjected very much to which algorithm being used for maze navigation. There are quite a number of different algorithms being developed and refined to solve mazes. Through this literature review we will do a comparative study of different algorithms and highlight their pros and cons [1].

2. 1. The Micromouse competition

Micromouse event has debuted since the early 1970s. It is 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 autonomous robots that are able to navigate mazes by themselves and find the goal through the shortest path possible and also with the least amount of time possible. At first, the robots will navigate the maze to find the goal point. Once the goal is located, the robot will identify the shortest path and at the second round, the robots should be able to navigate the maze through the shortest path and shortest time.

There are some rules for the mazes. At first, the mazes were 16x16. And depending on the rules of the competition, the mazes might have loop-hole, left-walled or right-walled structure and the list goes on. The same goes to the size of the robots and their weights.

Recently in Japan, they initiated a new challenging micromouse event whereby the sizes of robots are halved and the maze is upgraded to 32x32 while conserving the same size as 16x16 mazes. This has brought sets of new challenges for the competitors for designing the robot and the algorithms that they will use to solve such a complex and big mazes.

2. 2. The left-walled or right-walled algorithm

With this algorithm, the robot will keep an eye at the right or left wall and navigate throughout the maze till it finds the goal [1]. This algorithm is proven to be very efficient for mazes that are wall-linked to the goal point. The approach is very simple, after eventually finding the end point; the robot will compute the shortest distance for the second round.

The major drawbacks for this algorithm are that it can be used only on small and simple

mazes. Furthermore the goal has to be wall-linked and the 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 01 [2] below shows an example of a failure of wall follower algorithm in a practical way.

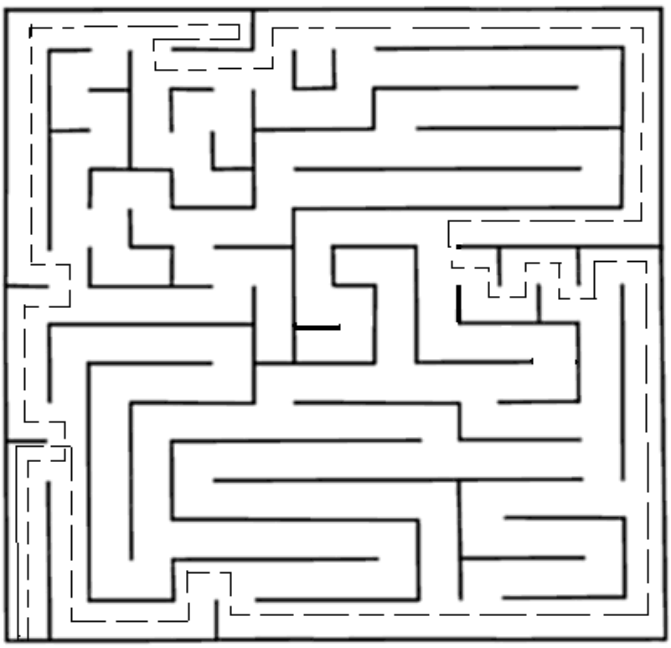

Figure 1: failure of wall-follower algorithm

2. 4. The flood-fill algorithm

The flood-fill algorithm is by far the most famous and efficient algorithm to solve all type of mazes. This algorithm assign values to each cell in the maze and these values assigned to the cells represent the actual distance between the cell and the goal point. These cells are represented by two dimensional arrays whereby the destination cell is represented by the array (0, 0). Any immediate neighboring cell will have the values of 1 in their arrays and so on. [3]

The flood-fill algorithm gets the current information of the cell that the robot resides in and predicts how far the goal point is. Going towards to 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 like the wall-follower algorithm. Figure 3 [4] shows a flow chart used in a typical flood-fill algorithm.

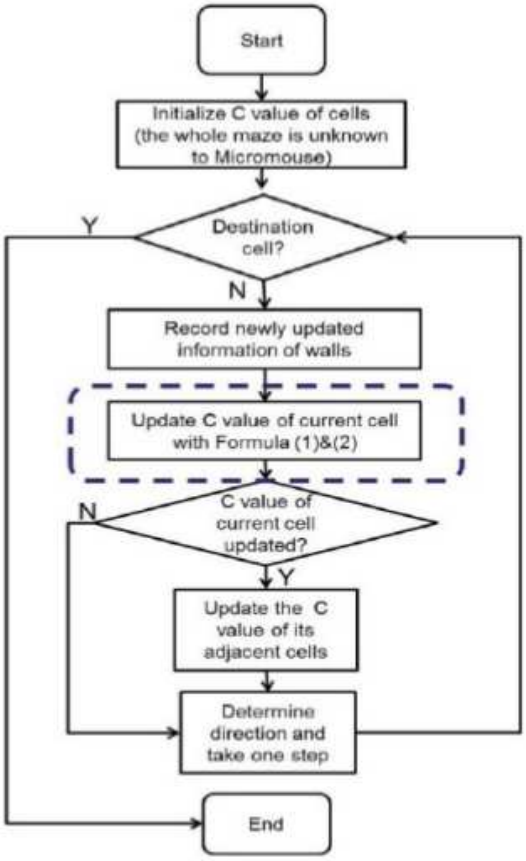

Figure 2: Flow chart of a typical flood-fill algorithm

2. 3. The depth-first search

Invented by a French Mathematician named Charles Pierre Tremaux, the depth-first search is intended to solve mazes back in the 19th century. This algorithm uses the logic of going from the initial point considered as root and going deeper into each branch until no node-child is found [1]. If the goal is along the way, the search will stop once the goal is found. Else the robot will go back and push to the stack the path navigated and look for another branch [1][3].

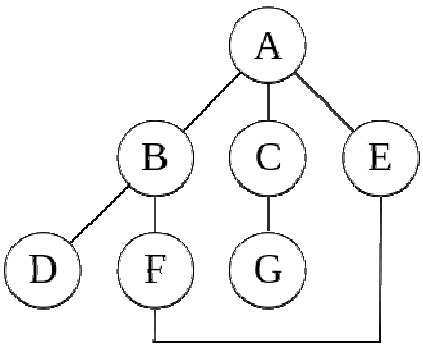

Figure 3: Example of a robot caught in an infinite cycle

This algorithm shown above is efficient for maze solving but the major drawback is that there is a possibility that the robot will navigate the entire maze before finding its goal. And this is not encouraged especially when time and distance are key factors. Another inconvenient of this algorithm is that it cannot find its way out in case it bumped into a loop-hole.

We could have noticed that all the mentioned above algorithms are very efficient in their own ways. For instance, the most famous algorithm used in micromouse competition is the flood-fill algorithm due to the nature of the maze used in the competition. In our case, the aim is to navigate fully reconfigurable mazes; therefore one of the best approaches would be the use of hybrid algorithm whereby we can combine two algorithms.

CHAPTER 3 METHODOLOGY

3. 1. Research methodology

Before diving into the project, a deep research about mapping and localization for small robot will be carried out. A research on what type of microcontroller board to be used and what kind of external input sensors to be used will be carried out as well. A wise choice of electronic components will be made based on studies and information gathered. This information will be found from related books, relevant published papers and trusted sources from the internet as well.

3. 2. Project activities

i. Gather information about small two-wheeled robots. Do research and explore the making of robots.

ii. Identify the best parts/components that can make the desired robot for the project. Do qualitative research about sensors, motors, chassis, etc... that would be used.

iii. Purchase all the necessary equipments/components that were identified previously.

iv. Assemble the robot accordingly and test its performance. Tweak if necessary to get the desired performance.

v. Make a research and gather information about the best algorithm that is suited for this kind of project.

vi. Modify the algorithm selected to meet the requirement of the project.

vii. Test the algorithm on the robot and make necessary changes until the robot perform as it is intended to do so.

3. 3. Project Gantt chart

To do / weeks	01	02	03	04	05	06	07	08	09	10	11	12	13	14
Title selection and FYP Supervisor	x	x	x											
Extended proposal		x	x	x	x	x								
Viva: proposal defence and progress evaluation								x	x					
Project progress							x	x	x	x	x	x	x	
Draft report													x	
Final report														x

Figure 04: a) Project Gantt chart – first semester

To Do / Weeks	01	02	03	04	05	06	07	08	09	10	11	12	13	14
Review of project objective	x	x												
Progress - Algorithm development		x	x	x	x	x	x	x	x	x	x	x		
Testing and debugging					x	x	x	x	x	x	x	x	x	
Progress Report								x						
Sedex presentation											x			
Report draft and technical paper													x	
Final presentation														x

Figure 04: b) Project Gantt chart – second semester

CHAPTER 4 PROJECT COMPONENTS

4. 1. Description of the robot

The first phase of this project is to have a working a running robot. The main components used to build the robot are:

- Chassis
- Two (2) Cytron C36R motors with wheels
- One (1) mini breadboard
- One (1) Arduino UNO microcontroller board
- LD293 motor driver chips
- Three (3) Ping – Distance finder – sensors
- Wires, nuts and stands
- Batteries and battery holders

Figure below shows the connections between all above mentioned elements to achieve a working robot.

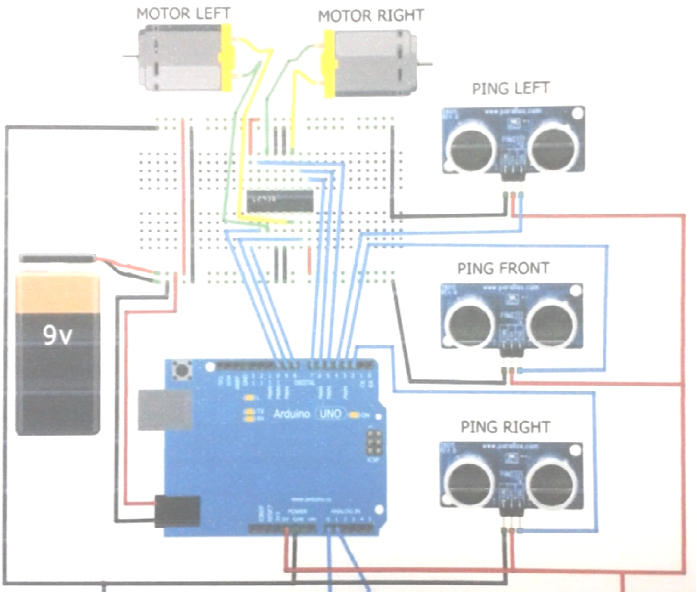

Figure 05: Circuit diagram

4. 2. Arduino UNO

Arduino is among the first open-source electronics board available. It is relatively easy to use and very flexible in its hardware design and software usage. There are more than 10 Arduino boards available and all are open-source. Arduino UNO is one of them.

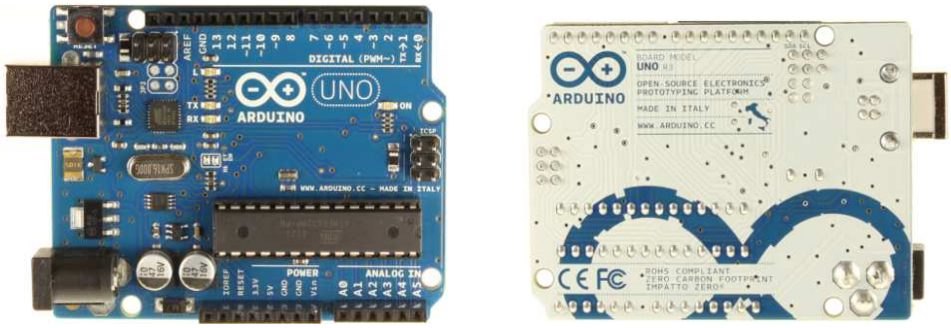

Figure 06: a) Arduino UNO R3 front *b) Arduino UNO R3 back*

Arduino UNO has the following components and running conditions:

- Microcontroller ATmega328
- Operating Voltage 5V
- Input Voltage (recommended) 7-12V
- Input Voltage (limits) 6-20V
- Digital I/O Pins 14 (of which 6 provide PWM output)
- Analog Input Pins 6
- DC Current per I/O Pin 40 mA
- DC Current for 3.3V Pin 50 mA
- Flash Memory 32 KB (ATmega328) - 0.5 KB used by bootloader
- SRAM 2 KB (ATmega328)
- EEPROM 1 KB (ATmega328)
- Clock Speed 16 MHz

Arduino syntaxes is based on C/C++ programming language, however the programming structure is much different. Basically, all Arduino programs should have two parts: the `void setup()` and the `void loop()`.

`void setup()` – all input pins and output pins are defined in this function using the Built-In-Functions

`void loop()` – all instructions are written in this function. This block will loop continuously until the board is turned off.

4. 3. Ping - Ultrasonic Distance Sensor

Ping - Ultrasonic Distance Sensor measures the distance using Sonar. An ultrasonic (unaudible) sound pulse is transmitted from the device to the object target. Based on the time taken by the emitted sound between emission and reception, the diatance between the sensor and the object is calculated. Knowing that the sound travels at the speed of light and the time taken for the sound to reach the object and bounce back, the distance is calculated using the simple relation: **d = v * t**

The Ping sensor is a low cost device which is much used in applications where accuracy is not the main concern.

Figure 07: Ping - Ultrasonic Distance Sensor

Key Specifications:

Power requirements:	+5 VDC
Communication:	Positive TTL pulse
Dimensions:	0.81 x 1.8 x 0.6 in (22 x 46 x 16 mm)
Operating temp range:	+32 to +158 °F (0 to +70 °C)

Features:

Distance measurements:	within a 2 cm to 3 m range
Communication:	Simple pulse in/pulse out
Indication:	Burst indicator LED shows measurement in progress
Power consumption:	20 mA
Narrow acceptance angle:	about 15^0
Connection:	3-pin header

4. 4. IR Line Tracking Sensor (Single Bit)

The IR Line tracking sensor is a sensor that can differentiate between white and black color. It outputs via TTL signal. It is a very simple to use and very efficient. Furthermore, it comes with a variable resistor that enables the tuning of the threshold voltage of white and black color. Combining three of this sensor will help us ensuring that the robot will be moving in a straight line. These can also used to avoid collision because since they can operate within a range of 1.5cm. they perform very nicely when used in no-contact switch scenario if coupled with relays.

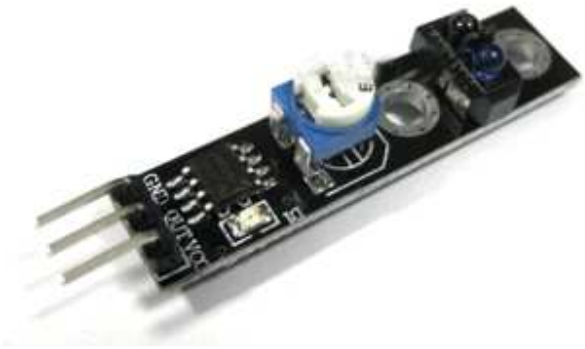

Figure 08: IR Line Tracking Sensor (Single Bit)

Features:

Detection distance:	1.5cm (tested with white paper)
Power supply:	3.3 to 5VDC
Operating current:	18 to 20mA at 5V
Operating temperature range:	0°C ~ + 50°C
Output interface:	3 wires
Output:	TTL (Black = Logic HIGH, White = Logic LOW)

The surrounding information are read by the sensors and fed to the microcontroller. This information is treated and actions are taken based on the readings. For instance, if the front Ping sensor read a distance of less of 10 cm, the robot will stop to avoid colliding with the wall. Then other readings from the adjacent sensors will determine whether the robot will take a left turn, right turn or moving backwards.

4. 5. Reconfigurable Maze for testing

To complete the project, the robot should be tested in a reconfigurable maze. A big reconfigurable maze is built for this purpose. The maze is made based on wood with closed edges. It hosts a 16x16 grid which can make a very complex maze if needed.

The structure if the maze is in such a way that it is totally reconfigurable. One can alter the interior easily to achieve any type of maze possible for 16x16 grids. The platform is equipped with grid lines that help for straight movement and perfect turning without the use of any encoder.

To conduct the experiments, a set of different mazes are needed. The first maze should be a simple maze with fewer obstacles and a goal point. The second maze should be a bit complex compared to the first maze and the third maze should have high complexity such as infinite loop and more complex structure.

Figures below show some example of mazes that can be used to conduct the experiment.

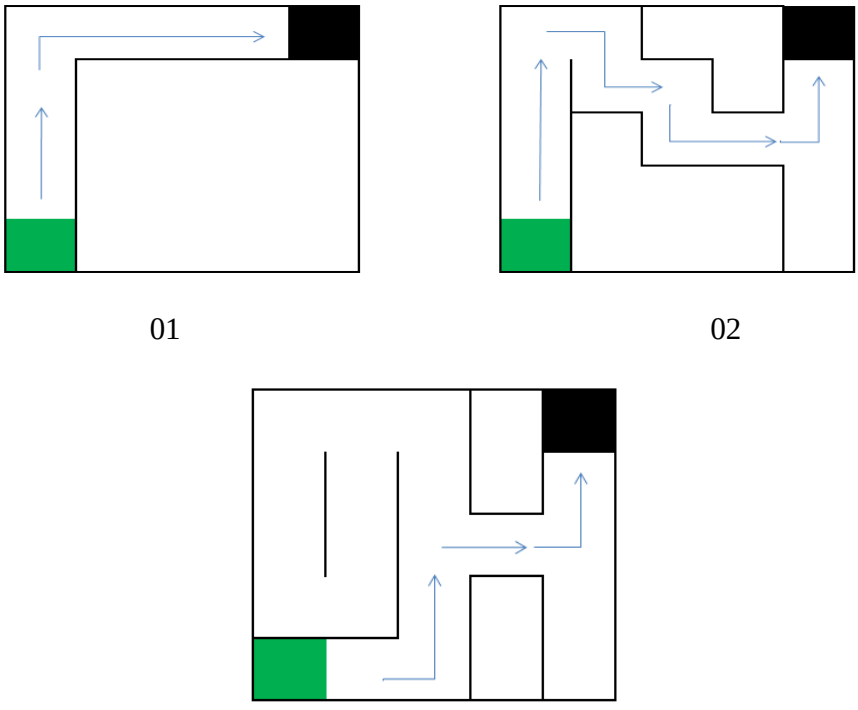

Figure 09: Example of types of mazes to used

CHAPTER 5 ISSUES AND CORRECTION

5. 1. HC-SR04 Ping Sensors issues

The HC-SR04 Ping sensor is a low cost ping sensor that has about the same performance as the Ping Ultrasonic from Parallax which is a bit expensive. The cheapness of this sensor comes along with performance price to pay.

The issue encountered with these sensors is quite unique in its genre. For instance, the accuracy of one sensor is perfect from 2cm up to 200cm with a very small range of error. However, when reading simultaneously from 2 sensors, the second reading from the second sensor is not as accurate as it's expected it to be.

Doing this with all three (3) sensors, the third reading actually fluctuates. This behavior is a bit strange since each sensor is connected to its own ping with its own reading mechanism.

The figure below shows an example of reading obtained from front ping alone followed by reading from front ping and left ping simultaneously, and then all three pings are read together simultaneously.

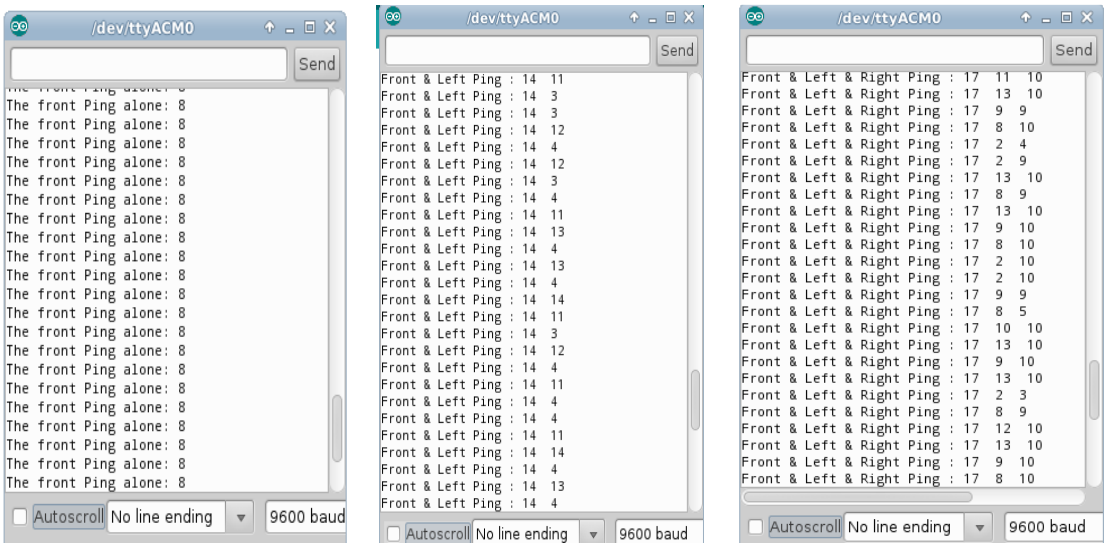

Figure 10: a) Reading of one sensor alone b) Readings of two sensors simultaneously c) Reading of three sensors simultaneously

5. 2. Solution provided for the Ping sensors issues.

To cater the above mentioned issues encountered by the ping sensors, a series of trial and errors method were conducted. It is found that playing with the delays between the reading has an impact on the reading itself. There is no delay property is mentioned in the datasheet of the sensor though.

Readings were taken for 1ms delay, 3ms delay and 5ms delay between the sensors reading. It is noticed that a delay of 1ms does not help much and reading are inaccurate. And improvement is shown with a 2ms delay between the reading but with 5ms delay, the reading were quite consistent and satisfactory. Below are the figures showing all three reading with three different delays.

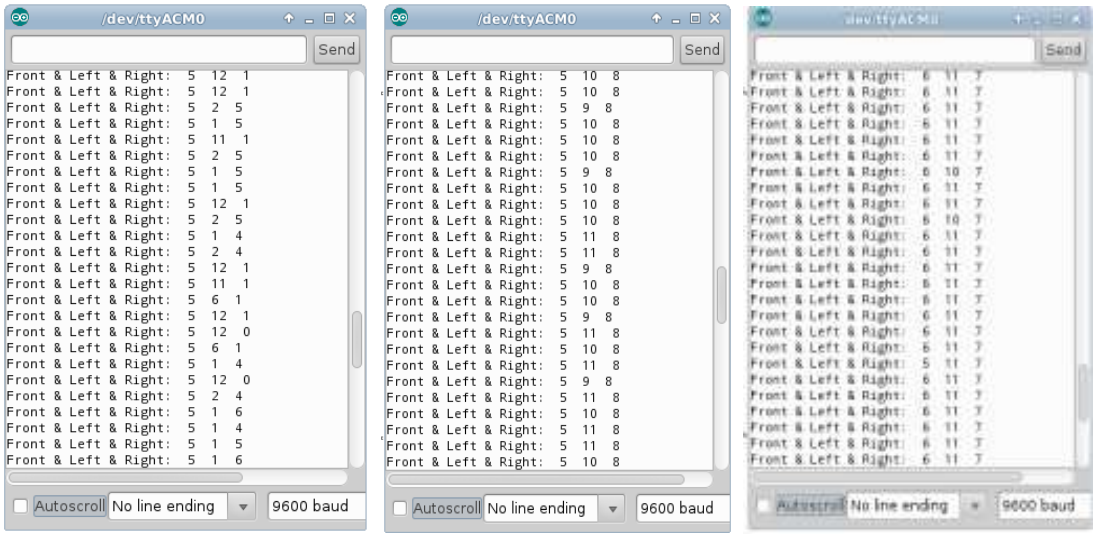

Figure 11: a) Readings with 1ms delay b) Readings with 2ms delay c) Readings with 5ms delay

CHAPTER 6 IMPLEMENTATION AND DISCUSSION

6. 1. Wall-following Hybrid Algorithm

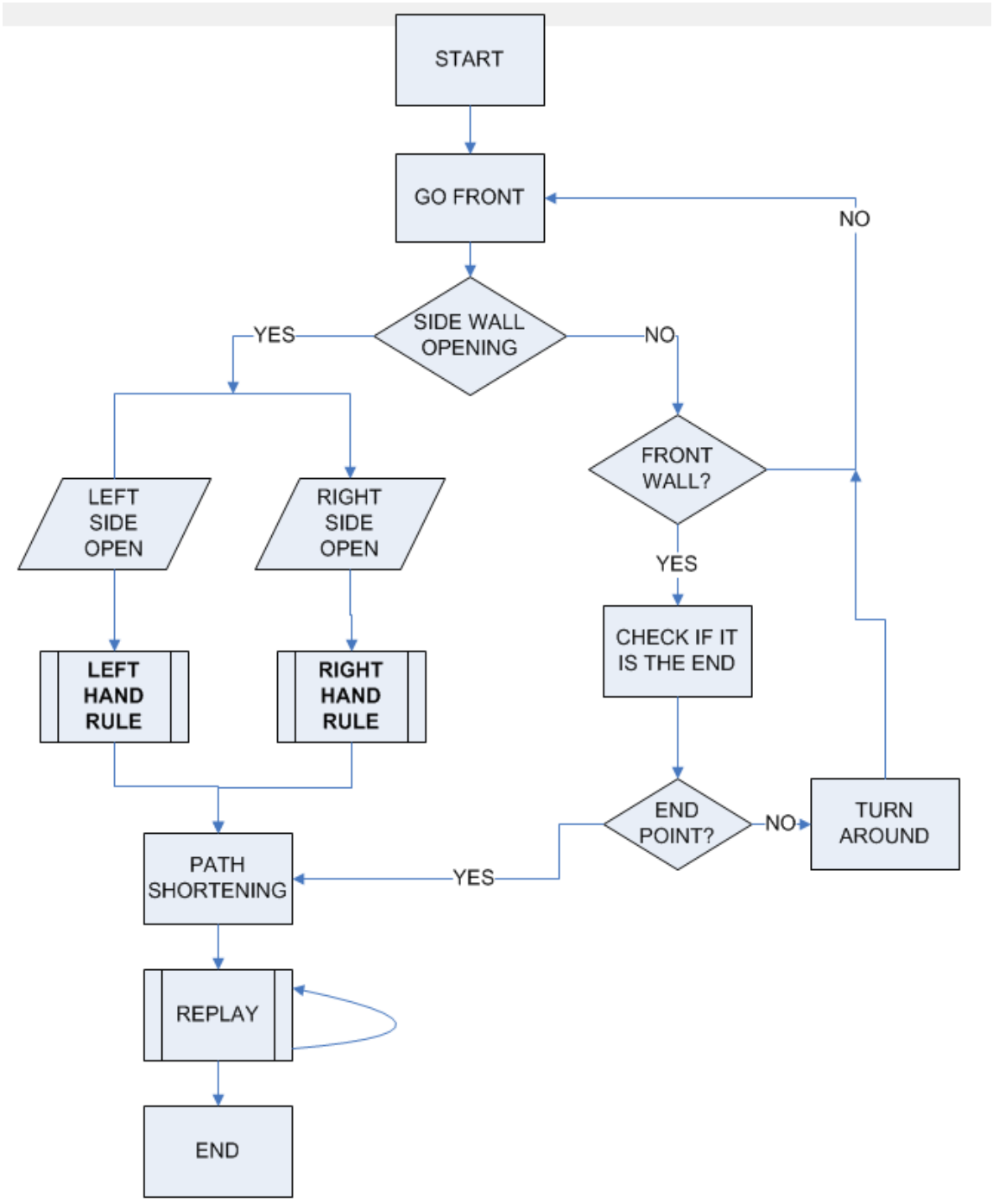

Figure 12: Hybrid Wall-Follower algorithm

To navigate different type of mazes, the wall-follower algorithm is adopted. In this case, the left-hand rule and the right-hand rule are applied based on the first side opening encountered. This combination of left-hand rule and right-hand rule is used to maximize the capability of the robot to be able to use navigate much complex type of mazes and also in some cases, it will help to avoid unnecessary long navigations.

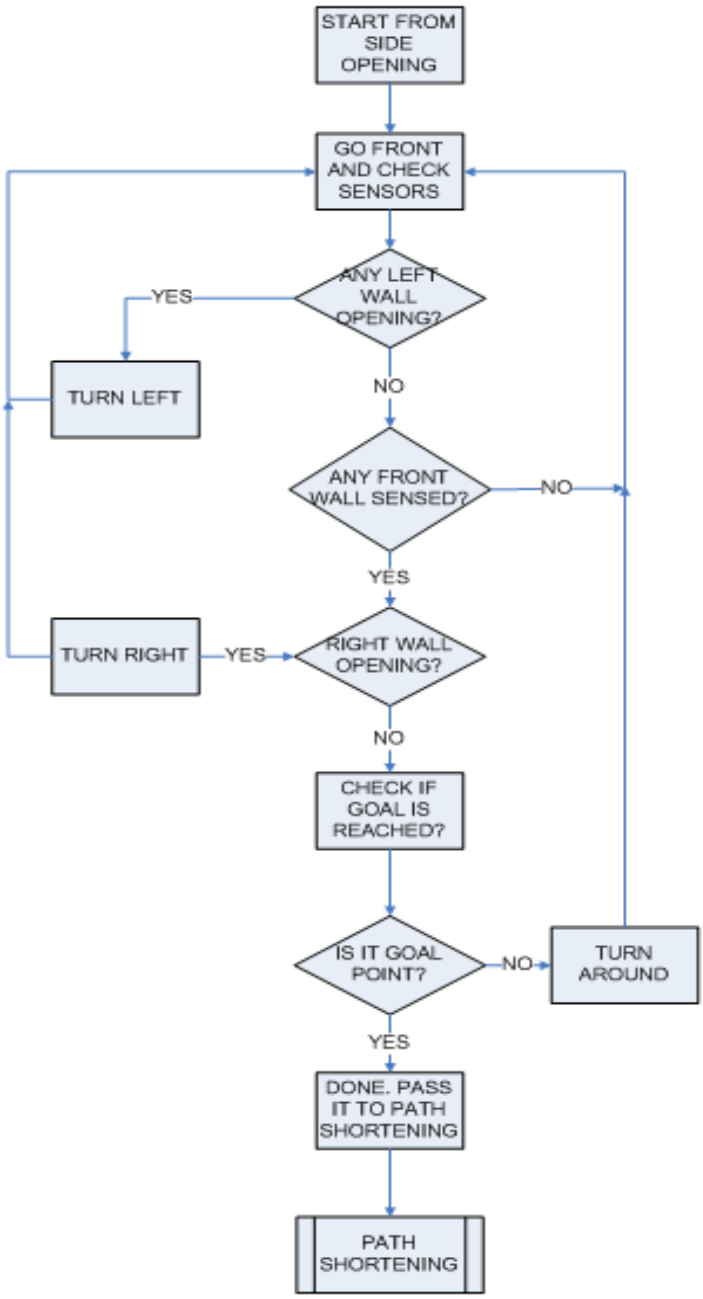

Figure 13: The Left Hand Rule flowchart

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. Here is a pseudo-code on how the left-hand rule works:

```
If front obstacle > 10cm
        If left obstacle > 10cm
                Turn left
        If left obstacle < 10cm
                Go straight
If front obstacle < 10cm
        If left obstacle > 10cm and right obstacle > 10cm
                Turn left
        If left obstacle > 10cm and Right obstacle < 10cm
                Turn left
        If Left Obstacle < 10cm and Right Obstacle > 10cm
                Turn right
        If Left Obstacle < 10cm and Right Obstacle < 10cm
                If this is End Point
                        Done
                If this is NOT End Point
                        Turn Around
```

With this algorithm, the robot will find the end point of any wall follower maze provided that the maze does not have an infinite loop. This algorithm is very efficient for simpler maze with no fix target unlike the flood-fill algorithm where the target is predetermined fix point.

6. 2. Techniques used to achieve navigation.

The robot has three Ping Distance finder sensors along with three IR Single bit sensors. The three Ping sensors are used to detect walls and the IR sensors are used for straight movement, the turns and the ending point. Below are the details about how these sensors are used to achieve accurate wall detection and perfect movements.

a- Straight movement

Controlling two motors without encoder is very much difficult especially if the aim is

to keep the robot moving in a straight line. In this case, three IR sensors are used to help the robot achieve a perfect straight movement. Below is a pseudo-code that explains how the three sensors are used to keep the robot moving straight.

```
While moving straight:
       Move forward for a short period of time
       Check the IR sensors
              If middle IR sensor alone sensing black line
                 Go forward
              If left IR sensor sensing black line
                     Decrease the speed of left motor
              If right IR sensor sensing black line
                     Decrease the speed of right motor
              If all IR sensing black line
                     Ignore this → it is an intersection
```

b- Turning left or turning right

Turning right or left is also a bit of challenge since no encoder is used for this purpose. The same three IR are again used for different purpose here.

For right turning, when the robot detects that there are obstacle in front and left side but the right side is free, the robot will take a right turn. The robot will pass the intersection before it stops. So to turn right, the technique below is applied:

```
If front and left have walls and right side is empty:
       While left & right IR are not sensing black line
              Go back // go back till the intersection line
       While left IR is sensing black line
              Stop right motor and run left motor
       While left IR is NOT sensing black line
              Stop right motor and run left motor
```

For left turning, two ways are applied. One way is for absolute left turning and another way is for optional left turning. For absolute left turning, it is similar to right turning except that we use right IR sensor.

For optional left turning, the procedure is a bit different. This case happens when there is no front obstacle but there is an option to turn left. Below is how it is done:

```
If NO front Obstacle and Left side is free
       While left & right IR are NOT on black line
       // left ping detects opening before the intersection
       Go forward
       While left IR is sensing black line
              Stop right motor and run left motor
       While left IR is NOT sensing black line
              Stop right motor and run left motor
```

c- End point and turning around

The three IR sensors are again used together with the ping sensors to detect if the robot has reached the end point or not. The pseudo-code below determines whether the robot has reached the end point or it has to take a turn around.

```
If all Pings detect wall
   Stop the robot for a short while - Go a bit to the front
          If all IR detect black line → DONE
          Else → TURN AROUND
                 While left & right IR not sensing black line
                        Go back // till the robot be on the intersection
                 While left IR is sensing black line
                        Run backward right motor and run forward left motor
                 While left IR is NOT sensing black line
                        Run backward right motor and run forward left motor
                 While left IR is sensing black line
                        Run backward right motor and run forward left motor
                 While left IR is NOT sensing black line
                        Run backward right motor and run forward left motor
```

6. 3. Maze navigation

Figures below show how the robot navigates the maze. The starting point is marked with the green color and the end point is marked by the black color. In this maze, the robot is guided by obstacles. There are no extra choices present for the robot. The robot navigates in a very easy way and there is only one path possible from the starting point to the ending point.

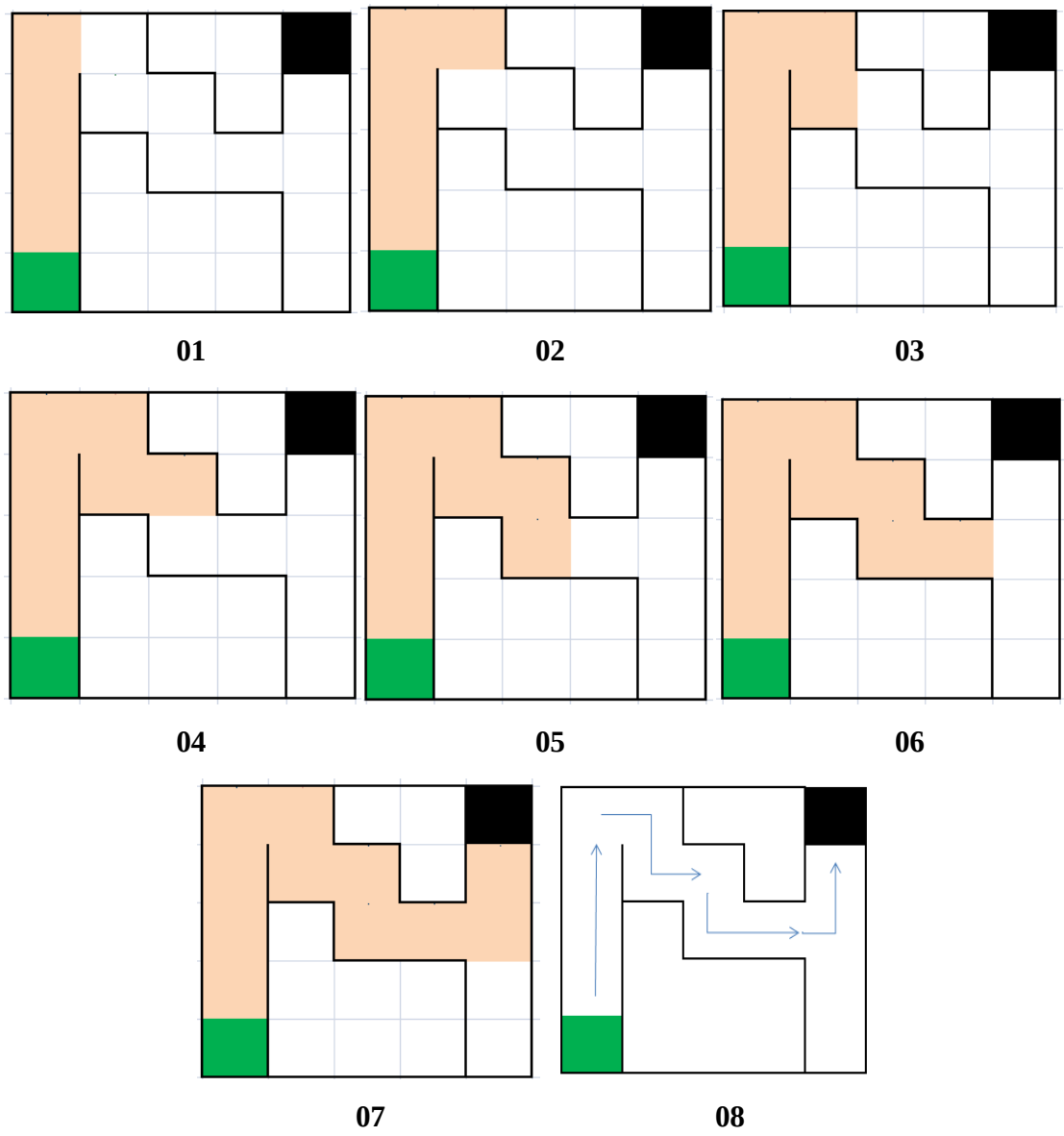

Figure 14: Navigation process in a simple maze

Here the robot will navigate a bit complex maze. As it can be noticed from the below pictures, the robot can find the end point but the path taken is not really optimized. The robot will navigate almost the entire maze before coming to the ending point. Once the end point is reached, the robot will optimize the path for next round by going through the shortest path possible.

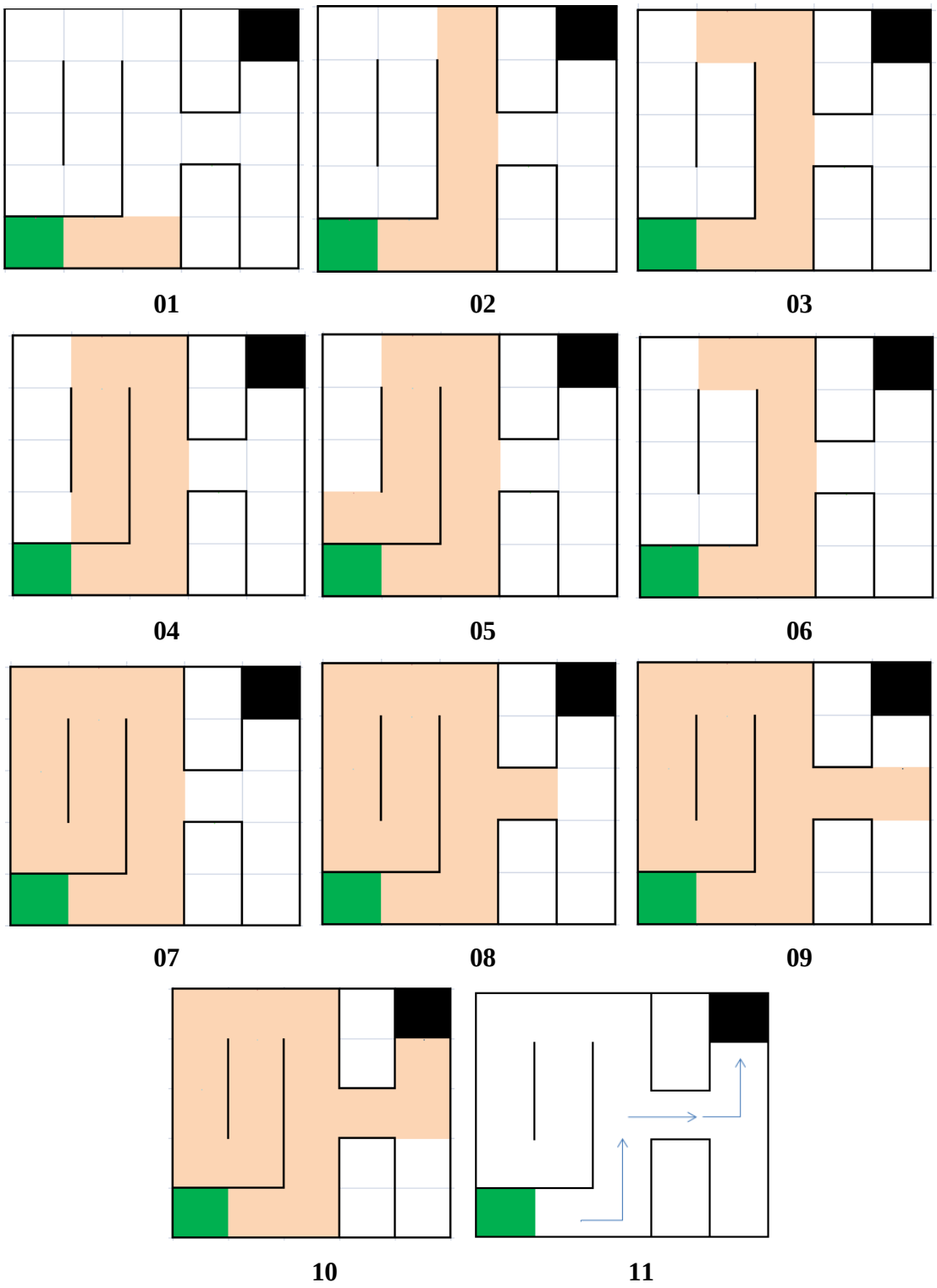

Figure 15: Navigation in a more complex maze

CHAPTER 7 IMPLEMENTATION OF HYBRID ALGORITHM

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

7. 1. Left-hand and right-hand rule

The figure 16 below shows a maze navigated using the right-hand rule algorithm. The navigation is very straightforward and the goal is achieved through the shortest path at the first time.

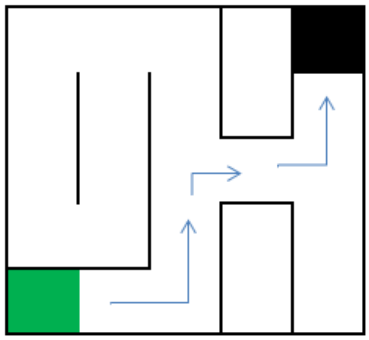

Figure 16: Right-Hand rule navigation

The figure 17 below shows the same type of maze but navigated by the use of left-hand rule algorithm. We can clearly notice that the navigation takes longer compared to the same navigation when used the right-hand rule navigation. However the shortest path computed will lead to the same shortest path.

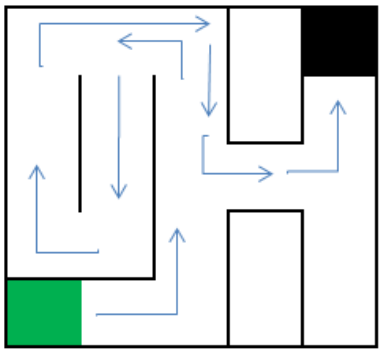

Figure 17: Left-Hand rule navigation

7. 2. Use of hybrid wall-follower algorithm

The use of hybrid wall-follower algorithm can improve greatly the navigation time of the vehicle. An example is shown below in the figure 18. The use of the hybrid is based on which side turn is encountered first. If the left turn is encountered first, the left-hand rule is selected. Whereas if the right turn is encountered first, then the right-hand rule algorithm is selected. From the picture, we can notice that the choice of the right hand rule algorithm is optimal.

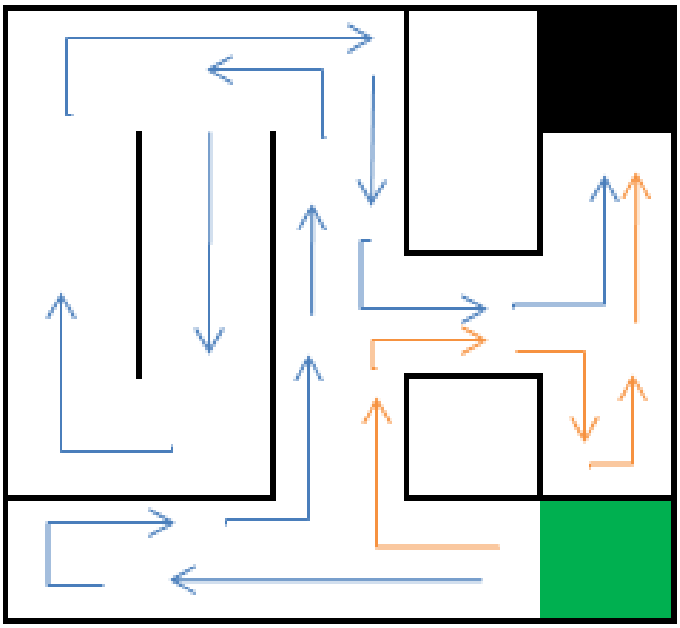

Figure 18: Hybrid wall-following algorithm

The red line indicates the choice taken by the vehicle based on the hybrid system. The blue line-path shown above is the path indicates the use of a single left-hand rule.

It is clearly shown that the use of the hybrid system has brought more intelligence to the vehicle in navigating the maze. Further implementation will the use of the same system in the scenario where the vehicle fall in an infinite loop while navigating a maze.

7. 3. Infinite loop case

In the scenario where the robot falls under an infinite loop while navigating the maze, one algorithm will not be enough for it to find its way out. The robot will keep on looping infinitely. Thus infinite loop is among the unwanted situations that need to be taken care of. In this project, an attempt to combine two different algorithms all working together will be carried out.

The challenge resides in the detection of the infinite loop. Since the size of the maze and its structure are not known by the vehicle, sensing an infinite loop will not be easy. One of the approaches might be the detection of the same pattern for a certain number of times while navigating the maze to conclude that the robot is in an infinite loop.

However, if the maze is structured in such a way that the navigation patterns are the same, the vehicle will not be able to make the right judgment call.

CONCLUSION

Throughout the project, an autonomous vehicle is built. The hardware limitations of the vehicle such as the use of low quality sensors made the learning curve very interesting. The vehicle uses IR Single bit sensors for perfect straight and turning and it also uses ultra-sonic range finder sensors for obstacle detection.

A reconfigurable maze for testing purpose has been made for the vehicle to navigate. The maze is fully reconfigurable, and helps to program the robot dynamically to suit all type of configuration possible.

The autonomous vehicle is equipped with a hybrid algorithm – left-hand wall and right hand wall – for perfect maze navigation. The robot switch from one algorithm to another depending on the situation encountered. For now, the decision is made based on the first turn encountered meaning that if first turn from start if on the left, then the left-hand algorithm will be used.

Future implementation will be on the detection of infinite loop while the vehicle is navigating the maze and the ability to switch from one algorithm to another upon sensing the infinite loop.

REFERENCES

[1] Adil M. J. Sadik, Maruf A. Dhali, Hasib M. A. B. Farid, Tafhim U Rashid, A. Syeed A "*Comprehensive and Comparative Study of Maze-Solving Techniques by Implementing Graph Theory*," 2010 International Conference on Artificial Intelligence and Computational Intelligence. Pp 52-56.

[2] Swati M., Pankaj B., "*Maze Solving Algorithms for Micro Mouse*", 2008 IEEE International Conference on Signal Image Technology and Internet Based Systems, pp 86-93.

[3] Kaizen R., "*Algorithms for Micro-mouse Manoj Sharma*", 2009 International Conference on Future Computer and Communication, pp 581-585.

[4] Zhuang Cai, Lu Ye, Ang Yang, "*FloodFill Maze Solving with Expected Toll of Penetrating Unknown Walls*", 2012 IEEE 14th International Conference on High Performance Computing and Communications, pp 1428 – 1433

[5] Patrickmccb. (2011, July 03). Maze solving robot. Retrieved from http://www.instructables.com/id/Maze-Solving-Robot/

[6] http://www.societyofrobots.com/member_tutorials/book/export/html/94

[7] Archive for the 'maze solving robot using arduino' category. (2010, October 10). Retrieved from http://satkum.wordpress.com/category/maze-solving-robot-using-arduino/

[8] Ibrahim M. E. Elshamarka, "Maze Robot: Design and implementation of autonomous vehicle navigation" May 2012, Final Year Project, Universiti Teknologi PETRONAS

APPENDIX

Code Sources for the Hybrid Wall-Follower Algorithm

```
#include <NewPing.h>

const int centerSensor = A3;  // center
const int rightSensor = A4;   // right
const int leftSensor = A5;     // left 46 for white space and 1016 for
black

const int leftMotor1 = 9;     // THE MOTORS
const int leftMotor2 = 13;
const int rightMotor1= 11;
const int rightMotor2 = 10;

const int frontPingTrig = 5;  // front sensor trigger
const int frontPingEcho = 2;  // front sensor echo

const int leftPingTrig = 7;    // left sensor trigger
const int leftPingEcho = 4;    // left sensor echo

const int rightPingTrig = 12; // right ping trigger
const int rightPingEcho = 8;  // right ping echo

#define MAX_DISTANCE 200
#define num 3

long ping_cm[num];
int cm[num];
int cm0; // front ping
int cm1;  // right ping
int cm2;  // left ping

NewPing sonar[num] = {
  NewPing(frontPingTrig, frontPingEcho, MAX_DISTANCE),
  NewPing(rightPingTrig, rightPingEcho, MAX_DISTANCE),
  NewPing(leftPingTrig, leftPingEcho, MAX_DISTANCE),
```

```
};

//DECLARATION OF VARIABLES
int centerReading;
int leftReading;
int rightReading;
int leapTime = 100;
int stop_time = 0;

int frontPing;
int leftPing;
int rightPing;

int replaystage = 0;

#define led A0

char path[30] = {};
char route[30] = {};
int pathLength;
int routeLength;
int readLength;
int check = 0;

int i, c;
int ii = 5;
int choice = 0;
int frontObstacle = 8;
int sideObstacle  = 10;


//************** the set up loop **************//
void setup ()
{
  Serial.begin(9600)  ;

  pinMode(centerSensor, INPUT);
  pinMode(leftSensor, INPUT);
```

```
  pinMode(rightSensor, INPUT);

  pinMode(leftMotor1, OUTPUT);
  pinMode(leftMotor2, OUTPUT);
  pinMode(rightMotor1, OUTPUT);
  pinMode(rightMotor2, OUTPUT);

  digitalWrite(led, LOW);
  delay(1000);
}


//***************** the main void loop **************//

void loop()
{

  frontPing = sonar[0].ping_cm();  delay(ii);
  leftPing = sonar[2].ping_cm();   delay(ii);
  rightPing = sonar[1].ping_cm();  delay(ii);

  if(choice == 0)
  {
      straight();

      frontPing = sonar[0].ping_cm();  delay(ii);
      leftPing = sonar[2].ping_cm();   delay(ii);
      rightPing = sonar[1].ping_cm();  delay(ii);

      // no side openings so we check if we are at the goal or a turn
around
      if(frontPing  <  frontObstacle  &&  leftPing  <  sideObstacle  &&
rightPing < sideObstacle)
      {
          // check for done
          front(200);
          stopall(50);
          if(analogRead(leftSensor) > 200 &&
```

```
            analogRead(centerSensor) > 200 &&
            analogRead(rightSensor) > 200)
          {
                  done();
          }
          else
          {
            stopall(50);
            turnAround();
          }
      }
      // there is a right side opening so we choose the right hand wall
      else if(leftPing < sideObstacle && rightPing > sideObstacle)
        {
          // choose the right hand wall algorithm
          choice = 1;
      }
      // there is a left side opening so we choose the left hand wall
      else if(leftPing > sideObstacle && (rightPing < sideObstacle ||
rightPing > sideObstacle))
        {
          // choose the left hand wall algorithm
          choice = 2;
      }
  }

  // Execution of the choice
  if(choice == 1)
  {
    rightHandWall();
  }

  if(choice == 2)
  {
    leftHandWall();
  }

} // end of the void loop()
```

```
//*********** all the functions **********************//

void leftHandWall() {

 // done or turn around
 if(frontPing < frontObstacle && leftPing < sideObstacle && rightPing <
sideObstacle){

  front(200);
  stopall(50);
  if(analogRead(leftSensor) > 200 &&
     analogRead(centerSensor) > 200 &&
     analogRead(rightSensor) > 200){
        done();
  }else{
    stopall(50);
    turnAround();
  }
 }

 // turning right
 else  if(frontPing  <  frontObstacle  &&  leftPing  <  sideObstacle   &&
rightPing > sideObstacle ){

   stopall(50);
   turnRight();
   stopall(100);
 }

 // turn left --> 1
 else  if(frontPing  <  frontObstacle  &&  leftPing  >  sideObstacle   &&
rightPing < sideObstacle ){ // #1

   //Serial.println("Stop #1");
   stopall(100);
```

```
   turnLeft();
   stopall(100);
 }

 // turn left --> 2
 else if(frontPing < frontObstacle && leftPing > sideObstacle   &&
rightPing > sideObstacle ){ // #2

   //Serial.println("Stop #2");
   stopall(100);
   turnLeft();
   stopall(100);
 }

 // go front --> 1
 else if(frontPing > frontObstacle && leftPing < sideObstacle   &&
rightPing < sideObstacle )
 {
   //stopall(50);
   straight();
 }

 // do front --> 2
 else if(frontPing > frontObstacle && leftPing < sideObstacle   &&
rightPing > sideObstacle )
 {
   //stopall(50);
   straight();

   if(analogRead(leftSensor) > 200 && analogRead(rightSensor) > 200)
   {
    //path[pathLength]='S';
    //Serial.println("s");
    //pathLength++;
    delay(100);
   }
 }
```

```
 // turn left critical --> 1
 else if(frontPing > frontObstacle && leftPing > sideObstacle   &&
rightPing < sideObstacle ) // #3
 {
  //Serial.println("Stop #3");
  straight();

  if(analogRead(leftSensor) > 200 && analogRead(rightSensor) > 200)
  {
    //Serial.println("crossed the critical #3");
    delay(150);
    stopall(200);
    turnLeft();
  }
 }

 // turn left critical --> 2
 else if(frontPing > frontObstacle && leftPing > frontObstacle &&
rightPing > sideObstacle)  // #4
 {
   //Serial.println("Stop #4");
   straight();

   if(analogRead(leftSensor) > 200 && analogRead(rightSensor) > 200)
   {
    //Serial.println("crossed the critical #4");
    delay(150);
    stopall(200);
    turnLeft();
   }
 }
}


// RIGHT HAND RULE

void rightHandWall() {
```

```
      // done or turn around
      if(frontPing < frontObstacle &&
            leftPing < sideObstacle &&
            rightPing < sideObstacle){

            front(200);
            stopall(50);
            if(analogRead(leftSensor) > 200 &&
                  analogRead(centerSensor) > 200 &&
                  analogRead(rightSensor) > 200){
                        done();
            }else{
                  stopall(50);
                  turnAround();
            }
      }

      // turning left
      else if(frontPing < frontObstacle && rightPing < sideObstacle  &&
leftPing > sideObstacle ){

            stopall(50);
            turnRight();
            stopall(100);
      }

      // turn right --> 1
      else if(frontPing < frontObstacle && rightPing > sideObstacle  &&
leftPing < sideObstacle ){ // #1

            //Serial.println("Stop #1");
            stopall(100);
            turnLeft();
            stopall(100);
      }

      // turn right --> 2
```

```
      else if(frontPing < frontObstacle && rightPing > sideObstacle  &&
leftPing > sideObstacle ){ // #2

            //Serial.println("Stop #2");
            stopall(100);
            turnLeft();
            stopall(100);
      }

      // go front --> 1
      else if(frontPing > frontObstacle && rightPing < sideObstacle  &&
leftPing < sideObstacle )
      {
            //stopall(50);
            straight();
      }

      // do front --> 2
      else if(frontPing > frontObstacle && rightPing < sideObstacle  &&
leftPing > sideObstacle )
      {
            //stopall(50);
            straight();

            if(analogRead(leftSensor) > 200 && analogRead(rightSensor)
> 200)
            {
                  //path[pathLength]='S';
                  //Serial.println("s");
                  //pathLength++;
                  delay(100);
            }
      }

      // turn right critical --> 1
      else if(frontPing > frontObstacle && rightPing > sideObstacle  &&
leftPing < sideObstacle ) // #3
      {
```

```
            //Serial.println("Stop #3");
            straight();

            if(analogRead(leftSensor) > 200 && analogRead(rightSensor)
> 200)
            {
                  //Serial.println("crossed the critical #3");
                  delay(150);
                  stopall(200);
                  turnLeft();
            }
      }

      // turn right critical --> 2
      else if(frontPing > frontObstacle && rightPing > frontObstacle &&
leftPing > sideObstacle)  // #4
      {
            //Serial.println("Stop #4");
            straight();

            if(analogRead(leftSensor) > 200 && analogRead(rightSensor)
> 200)
            {
                  //Serial.println("crossed the critical #4");
                  delay(150);
                  stopall(200);
                  turnLeft();
            }
      }
}

//------------- DONE -- END-MOTION -- PATH-SHORTENING --  REPLAY ------
--------//

// DONE FUNCTION
void done(){

  digitalWrite(leftMotor1, LOW);
```

```
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, LOW);

  replaystage=1;
  path[pathLength]='D';
  //Serial.println("d");
  pathLength++;
  //printPath();

  // --> shortening stuffs
  c = pathLength;
  while(check == 0){
    Serial.println("trapped");
    shortening(); //delay(300);
  }

  while(analogRead(leftSensor) > 200 &&
        analogRead(centerSensor) > 200 &&
        analogRead(rightSensor) > 200){

    digitalWrite(led, HIGH);
    delay(500);
    digitalWrite(led, LOW);
    delay(500);
  }
  delay(500);
  replay();
}


// END OF MOTION
void endMotion(){
    digitalWrite(led, LOW);
    delay(500);
    digitalWrite(led, HIGH);
    delay(200);
      digitalWrite(led, LOW);
```

```
    delay(200);
    digitalWrite(led, HIGH);
    delay(500);
  endMotion();
}

// PATH SHORTENING
void shortening(){

  check = 1;
  Serial.print("Before shortening c = ");
  Serial.println(c);

  Serial.print("Incoming array = ");
  for(i=0; i<c; i++){Serial.print(path[i]);}
  Serial.println(" ");Serial.println(" ");

  for(i=0; i < c; i++){

      if(path[i] == 'B'){  // check for B

        if(path[i-1] == 'L' && path[i+1] == 'G'){  // R condition
          route[routeLength] = 'R';
          routeLength++;

        }else if(path[i-1] == 'L' && path[i+1] == 'R'){
          route[routeLength] = 'B';
          routeLength++;

        }else if(path[i-1] == 'L' && path[i+1] == 'S'){
          route[routeLength] = 'R';
          routeLength++;

        }else if(path[i-1] == 'R' && path[i+1] == 'L'){
          route[routeLength] = 'B';
          routeLength++;

         }else if(path[i-1] == 'S' && path[i+1] == 'L'){
```

```
          route[routeLength] = 'R';
          routeLength++;

        }else if(path[i-1] == 'S' && path[i+1] == 'S'){
          route[routeLength] = 'B';
          routeLength++;

        }else if(path[i-1] == 'L' && path[i+1] == 'L'){
          route[routeLength] = 'S';
          routeLength++;
        }

      }else {    // if the current is not a B, check if previous or
after is B
        if(path[i-1] == 'B' || path[i+1] == 'B'){
          // do nothing

        }else{
          // record it
          route[routeLength] = path[i];
          routeLength++;
        }
      }
  }// end of for loop

    Serial.println("After shortening");

    c = routeLength;
    Serial.print("Length of route array is ");
    Serial.println(c);

    Serial.print("The route path = ");
    for(i=0; i<c; i++){Serial.print(route[i]);}
    Serial.println(" ");

    for(i=0; i<c; i++){
      if(route[i] == 'B'){
        check = 0;
```

```
        Serial.println("There is a B");
      }
    }
    Serial.println(" ");

    if(check == 0){
      //path[c] = {};
      for(i=0; i<c; i++){
        path[i] = route[i];
       // Serial.println(path[i]);
      }
      // clear route
      //route[c] = {};
      routeLength = 0;
    }
    Serial.println(" ");
}


// REPLAY FUNCTION
void replay(){

//  Serial.print("Index is "); Serial.print(readLength);
//  Serial.print(" the letter is: ");
//  Serial.println(route[readLength]);

  delay(10);
  frontPing = sonar[0].ping_cm(); delay(10);
  rightPing = sonar[1].ping_cm(); delay(10);
  leftPing = sonar[2].ping_cm(); delay(10);

//  Serial.println(leftPing);
//  Serial.println(rightPing);

  if(frontPing > frontObstacle && leftPing < sideObstacle)
  {
    straight();
  }
```

```
  else
  {
    // stop for a moment to check for next move
    stopall(10);

    if(route[readLength]=='D')
    {
        straight();

        if(analogRead(leftSensor) > 200 && analogRead(rightSensor) >
200)
        {
          //Serial.println("crossed the critical #3");
          delay(150);
          stopall(100);
          digitalWrite(leftMotor1, HIGH);
          digitalWrite(leftMotor2, LOW);
          digitalWrite(rightMotor1, HIGH);
          digitalWrite(rightMotor2, LOW);
          delay(100);

          digitalWrite(leftMotor1, LOW);
          digitalWrite(leftMotor2, LOW);
          digitalWrite(rightMotor1, LOW);
          digitalWrite(rightMotor2, LOW);

          if(analogRead(leftSensor) > 200 && analogRead(rightSensor) >
200){
            done(); //endMotion();
          }
        }
    }

    // left turning
    else if(route[readLength] == 'L' &&
                  leftPing > sideObstacle &&
                  (rightPing > sideObstacle ||
```

```
              rightPing < sideObstacle))
{
  straight();

  if(analogRead(leftSensor) > 200 && analogRead(rightSensor) > 200)
  {
    //Serial.println("crossed the critical #3");
    delay(150);
    stopall(100);
    turnLeft();
    readLength++;
  }
}
// Right turning
else if(route[readLength]=='R')
{
  straight();

  if(analogRead(leftSensor) > 200 && analogRead(rightSensor) > 200)
  {
    //Serial.println("crossed the critical #3");
    delay(150);
    stopall(100);
    turnRight();
    readLength++;
  }
}
// go straight
else if(route[readLength]=='S')
{
  digitalWrite(leftMotor1, HIGH);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, HIGH);
  digitalWrite(rightMotor2, LOW);
  delay(leapTime);
  straight();
  readLength++;
}
```

```
    // nothing, keep moving
    else{
      stopall(100);
      straight();
      delay(100);
      stopall(200);
    }
  }
  replay();
}

//---------------- MOVEMENT AND TURNING FUNCTIONS --------------//

// GOING STRAIGHT
void straight() {

  // go straight
  digitalWrite(leftMotor1, HIGH); //the right wheel
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, HIGH);
  digitalWrite(rightMotor2, LOW);
  delay(2);

  // turn a bit to the left
  if(analogRead(leftSensor) > 200 && analogRead(rightSensor) < 200){
    digitalWrite(leftMotor1, HIGH);
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, LOW);
    digitalWrite(rightMotor2, LOW);
    delay(2);
  }

  // or turn s bit to the right
  else  if(analogRead(rightSensor)  >  200  &&  analogRead(leftSensor)  <
200) {
    digitalWrite(leftMotor1, LOW);
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, HIGH);
```

```
    digitalWrite(rightMotor2, LOW);
    delay(1);
  }

  // or keep going straight
  else {
    digitalWrite(leftMotor1, HIGH); //the right wheel
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, HIGH);
    digitalWrite(rightMotor2, LOW);
    delay(1);
  }
}


// TURNING TO THE LEFT FUNCTION
void turnLeft() {

  digitalWrite(leftMotor1, LOW);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, LOW);
  delay(100);

  // go back till all sensors are on the black line
  while(analogRead(rightSensor) < 200 || analogRead(leftSensor) < 200){
    digitalWrite(leftMotor1, LOW);
    digitalWrite(leftMotor2, HIGH);
    digitalWrite(rightMotor1, LOW);
    digitalWrite(rightMotor2, HIGH);
    delay(3);
    digitalWrite(leftMotor1, LOW);
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, LOW);
    digitalWrite(rightMotor2, LOW);
    delay(1);
  }
```

```
// stop for a delay
digitalWrite(leftMotor1, LOW);
digitalWrite(leftMotor2, LOW);
digitalWrite(rightMotor1, LOW);
digitalWrite(rightMotor2, LOW);
delay(100);

// turn till right IR sensor is out of the black line
while(analogRead(rightSensor) > 200 ){
  digitalWrite(leftMotor1, HIGH);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, LOW);
  delay(200);
  digitalWrite(leftMotor1, LOW);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, LOW);
  delay(1);
}

// turn till right IR sensor comes to the black line
while(analogRead(rightSensor) < 200) {
  digitalWrite(leftMotor1, HIGH);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, LOW);
  delay(2);
  digitalWrite(leftMotor1, LOW);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, LOW);
  delay(1);
}

if(replaystage==0){
  path[pathLength]='L';
  //Serial.println("l");
```

```
    pathLength++;
  }
}


// TURNING TO THE RIGHT FUNCTION
void turnRight() {

  digitalWrite(leftMotor1, LOW);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, LOW);
  delay(100);

  // go back till all sensors are on the black line
  while(analogRead(rightSensor) < 200 || analogRead(leftSensor) < 200){
    digitalWrite(leftMotor1, LOW);
    digitalWrite(leftMotor2, HIGH);
    digitalWrite(rightMotor1, LOW);
    digitalWrite(rightMotor2, HIGH);
    delay(3);
    digitalWrite(leftMotor1, LOW);
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, LOW);
    digitalWrite(rightMotor2, LOW);
    delay(1);
  }

  // stop for a delay
  digitalWrite(leftMotor1, LOW);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, LOW);
  delay(100);

  // turn till the left sensor is out of the black line
  while(analogRead(leftSensor) > 200 ){
    digitalWrite(leftMotor1, LOW);
```

```
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, HIGH);
    digitalWrite(rightMotor2, LOW);
    delay(2);
    digitalWrite(leftMotor1, LOW);
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, LOW);
    digitalWrite(rightMotor2, LOW);
    delay(1);
  }

  // turn till the sensor comes back to the black line
  while(analogRead(leftSensor) < 200) {
    digitalWrite(leftMotor1, LOW);
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, HIGH);
    digitalWrite(rightMotor2, LOW);
    delay(2);
    digitalWrite(leftMotor1, LOW);
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, LOW);
    digitalWrite(rightMotor2, LOW);
    delay(1);
  }

  if(replaystage==0){
      path[pathLength]='R';
      //Serial.println("r");
      pathLength++;
  }
}


// TURNING AROUND FUNCTION
void turnAround() {

  // go back till all sensors are on the black line
  while(analogRead(rightSensor) < 200 || analogRead(leftSensor) < 200){
```

```
  digitalWrite(leftMotor1, LOW);
  digitalWrite(leftMotor2, HIGH);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, HIGH);
  delay(3);
  digitalWrite(leftMotor1, LOW);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, LOW);
  delay(1);
}

// take a pause of 100 ms
digitalWrite(leftMotor1, LOW);
digitalWrite(leftMotor2, LOW);
digitalWrite(rightMotor1, LOW);
digitalWrite(rightMotor2, LOW);
delay(100);

// do the turning of 180 degrees
while(analogRead(leftSensor) > 200) {
  digitalWrite(leftMotor1, HIGH);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, HIGH);
  delay(500);

  while(analogRead(centerSensor) < 200) {
    digitalWrite(leftMotor1, HIGH);
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, LOW);
    digitalWrite(rightMotor2, HIGH);
    delay(1);
    digitalWrite(leftMotor1, LOW);
    digitalWrite(leftMotor2, LOW);
    digitalWrite(rightMotor1, LOW);
    digitalWrite(rightMotor2, LOW);
    delay(2);
```

```
    }
  break;
  }

  if(replaystage==0){
      path[pathLength]='B';
      //Serial.println("b");
      pathLength++;
  }
}


// STOP ALL THE MOVEMENT
int stopall(int stop_time) {

  digitalWrite(leftMotor1, LOW);
  digitalWrite(leftMotor2, LOW);
  digitalWrite(rightMotor1, LOW);
  digitalWrite(rightMotor2, LOW);
  delay(stop_time);
}


// FRONT GOING FUNCTION
int front(int time){

      digitalWrite(leftMotor1, HIGH);
      digitalWrite(leftMotor2, LOW);
      digitalWrite(rightMotor1, HIGH);
      digitalWrite(rightMotor2, LOW);
      delay(time);
}


//---------------- SERIAL DISPLAY FUNCTIONS --------------//

// IR SENSORS DISPLAY
void irInfos(){
```

```
  Serial.print("L C R = ");
  Serial.print(analogRead(leftSensor));
  Serial.print(" ");
  Serial.print(analogRead(centerSensor));
  Serial.print(" ");
  Serial.println(analogRead(rightSensor));
}

// PING SENSORS DISPLAY
void infos(){

  frontPing = sonar[0].ping_cm();  delay(ii);
  leftPing = sonar[2].ping_cm();   delay(ii);
  rightPing = sonar[1].ping_cm();  delay(ii);

  Serial.print("F L R = ");
  Serial.print(frontPing);
  Serial.print(" ");
  Serial.print(leftPing);
  Serial.print(" ");
  Serial.println(rightPing);
}

// PATH DISPLAY
void printPath(){

  Serial.println("++++++++++++++++++");
  int x=0;
  while(x<=pathLength){
  Serial.println(path[x]);
  x++;
  }
  Serial.println("++++++++++++++++++");
}

// PATH DISPLAY NEW
void printNew(){
```

```
  for(i=0; i<routeLength; i++){
    Serial.print(" ");
    Serial.println(route[i]);
  }
}
```