ABSTRACT

Micromouse is a worldwide event where small robot mice compete to solve a 16*16 maze. Our goal is to build a small mouse-like robot which is designed to solve the maze with the shortest effective route. It uses proximity sensors like VL53L0X using VCSEL to detect walls and maps out the maze, encoder motors for movement and feedback so that a closed loop control system can be constructed for the implementation of PID and modified flood fill algorithm to traverse the maze through shortest path possible. Initially, it will be tested in various maze configurations in simulation rather than in an actual maze to ensure that it is able to consistently find the center of the maze in a timely manner. PCB designing, System Identification, PID tuning using MATLAB, ROS simulation, microcontroller programming and embedded systems are some of the fields of engineering that we got to explore during this project.

Keywords: Control System, Encoder Motor, Floodfill, MATLAB, Maze, Micromouse, PCB, PID, ROS, System Identification, VL53L0X

Table of Contents

Lists of Figures

List of Tables

List of Abbreviations

AVR	Alf and Vegard's RISC processor
DC	Direct Current
IC	Integrated Circuit
IR	Infrared Radiation
PID	Proportional-Integral-Derivation
ROS	Robot Operating System
SDF	Simulation Description Format
STM	Super-True-Microcontroller
TOF	Time-of-Flight
URDF	Unified Robot Description Format
VLSI	Very Large Scale Integration

1. INTRODUCTION

Micromouse is a popular robotics device in which a small robot is tasked with finding its way to the center of a maze as quickly and efficiently as possible. It uses infrared sensors to detect the walls of the maze and create a map of the environment while trying to reach the center of the maze. Encoder motors help us to keep track of the robot in the maze. After it reaches the center, it travels back to the starting cell searching for the shortest path. Once the optimal path is determined, the micromouse navigates through the maze and reaches the center in the lowest time possible.

In this project, we describe the design and construction of our micromouse, the algorithms used for maze mapping and navigation, and the results of our testing. We will also discuss potential improvements and future directions for the project.

1.1. Background

Recent years have seen significant progress in the field of maze-solving algorithms and robotic mice. A variety of computational and biological approaches have been explored, leading to a better understanding of the mechanisms and strategies that can be used to navigate complex environments.

One of the key technical challenges in the design and implementation of maze-solving robots is the integration of sensors, actuators, and computational algorithms. By combining these components in novel and effective ways, researchers have been able to develop robotic mice that are capable of solving mazes with impressive accuracy and speed.

1.2. Motivation

The main factor that motivated us on choosing this project is the fact that it requires a greater understanding about hardware, control systems as well as precise implementation of the algorithms. This project assists us in broadening our knowledge on hardware components and its integration to make capable robots and to compile the algorithms like FloodFill into the robot. The project is motivated by the need

for autonomous robots that can operate in complex and unstructured environments, such as those found in industrial settings or disaster zones.

1.3. Problem Definition

There are numerous components and algorithms available for making a single robot. But the main problem statement is to research and choose the best available sources which makes the robot faster, precise and economical among different computational approaches we need to select for the best possible outcome. After integration of best available and possible hardware and software PID control algorithms are to be implemented to ensure the robot is precisely functional. Design and construction of our micromouse, the algorithms used for maze mapping and navigation must succeed to navigate and find the shortest path to the center of the maze.

1.4. Project Objectives

The objective of our project are as follows:

- To design the functional Micromouse robot that is capable of solving the maze and finding the shortest path to reach the center of the maze.

1.5. Project Scope and Applications

This type of project is typically used to test the skills and knowledge of engineering students in the areas of robotics, computer science, and electrical engineering. Some common applications of micromouse technology include search and rescue operations, warehouse automation, and mine detection. Micromouse robots can be used in search and rescue operations to navigate through collapsed buildings and other complex environments to find survivors. They have also been used in warehouse automation to help locate and transport items within a warehouse, and in mine detection to help locate and identify hidden mines, to search the lost labor in mines.

By designing and building robots that can navigate complex environments, engineers can help to improve safety, efficiency, and accuracy in a variety of industries. This type of technology is constantly evolving, with new innovations and developments

being made all the time. As a result, the scope and applications of micromouse technology are likely to continue to expand in the future.

1.6. Report Organization

The presented project report has been categorized into nine chapters.

- Chapter 1 provides the general introduction of the project. It includes all the background information, motivation, objectives, and scope of the project.
- Chapter 2 is the literature review; it contains summaries of the previous works performed and theories related to this project. It gives us idea of how others went about solving the problem that was ahead of us, the methodologies used before, and the results achieved be them.
- Chapter 3 describes the different requirements necessary during the development and implementation of project. It includes hardware requirements as well as software requirements.
- Chapter 4 discusses about the hardware architecture of our system, algorithm used, working principle of our robot and flowchart of the working principle.
- Chapter 5 discusses about the implementation details of both hardware components and algorithm used in the robot. It also includes the control system design of the micromouse robot.
- Chapter 6 presents the result of control system design of our robot and different graphs to visualize the output of the sensors used in our robot. It also presents results from our simulation software.
- Chapter 7 describes how the system can be further improved from its present states so that we can increase its scope or application.
- Chapter 8 discusses the overall project in summary and gives a brief conclusion about it.
- Chapter 9 contains additional information like the project budget, timeline of the projects, code snippets, used commands and PCB designs.
- The references which were taken throughout the course of this project development are included at the end of the report.

2. LITERATURE REVIEW

Micromouse competition has been around since the early 80s hence numerous researches have been done and different kinds of methodologies have been proposed. The present top competing mouse are able to produce acceleration and deceleration of around $8m/s^2$ with a turning acceleration as high as 2g[1]. With the addition of a suction fan 200g additional downward force is applied to the mouse for better grip and less slippage increasing the maximum acceleration limit[2].

[3] proposes using micromouse for teaching or introducing students to the concepts of autonomous mobile robots. Exposure to various topics like real time interrupt driven control, Integrated development environment, maze solving algorithm, stepper motors, high intensity IR sensors can be achieved while building micromouse bot. Stepper motors were used to provide motion but stepper motors are actually not suitable for high-speed movement applications. They can provide precise steps of motion but compared to the normal brushed dc motor they have low power to size ratio. They are bulky and is harder to move it to high speed efficiently although being an open loop control system they are simple to use. For maze solving bellman or floodfill algorithm is used. After full traversal of the maze, we might find more than one path to reach to the center. The optimal path is the one with the largest number of 90 degrees turn when diagonal turn is implemented.

Use of infrared sensors is common practice in the competition. Six pairs of sensors are generally used. Two facing directly sideways, two facing directly forward, and two angled in between the front and side sensor at around 45 degrees. The IR pair consists of an IR emitter and a phototransistor and works based on the intensity of the reflected light. The output obtained from the sensors is dependent upon the type of surface of the walls and is easily affected by the ambient light. To increase the immunity of the ambient light differential method is used. Here first the receiver is read when the emitter is turned off then the receiver is read when the emitter is turned on and the actual signal/output is the difference between the two. The main drawback of the use of IR pairs is their reliability. Since it is affected by ambient light and dependent on the surface type for real world situations and actual application of Micromouse (autonomous terrain exploration) use of IR pair isn't

ideal. The main advantage of this method might be its high speed but there is a tradeoff of reliability. The alternative for the IR pair and the solution for its poor reliability might be TOF sensors. Instead of relying on the intensity of the light one can rely on the time the sent signal requires to make a round trip. Use of six pairs of sensors might be a little redundant for beginners trying out constructing a mouse. In addition to this if use of sensors like vl53lox preferred which might be reliable but considerately slow, discarding two side sensors and using only angled sensors for side wall detection might improve performance[4].

Dead end detection and calibration is also another important algorithm to detect dead ends (walls at front and both sides). After dead end detection the mouse has to align itself laterally and longitudinally. Use of front sensors can align it longitudinally but for lateral alignment it has to move one wheel backwards first and then another wheel according to which wall it is close to. Then only it can do 180 degrees turn. But this method isn't reliable and might take many attempts to successfully align the mouse. Hence using only front sensors might be a better option. The method includes first using the front sensor to align longitudinally and then rotation 90 degrees and again using the front sensors to align longitudinally. After the alignment mouse can turn 90 degrees again to make a successful U turn[4].

Making a fastest micromouse robot using a single microcontroller for all calculations as well as decision making is impossible. A microcontroller when used alone has to do computations for PWM generation as well as perform logical operations to decide next move to traverse maze and find optimal path. Hence using FPGA for computation and leaving microcontroller only for decision making will definitely make the mouse run faster and smoother[5].

The mouse navigates using two motions i.e., rotational and translational. Using the combination of both it can make smooth turns or using only either it can move forward, backward or perform stationary pivot turning. The difference between stationary turning and smooth turning is that in stationary turning the mouse has to stop at a certain pivot point and move the two wheels in opposite direction for turning whereas in smooth turning the mouse has to decrease speed of one wheel and increase the speed of another wheel without stopping. Smooth turning significantly increases

performances as it doesn't have to stop but is also significantly complicated. The mouse has to decide where to start turning and by how much by creating a difference in speed of two wheels. Small offset in the calculation might cause the mouse to crash so precision is a must. Distance before turn, distance after turn, turning velocity and turning angle are some of the things the bot should consider to avoid collision Front correction feature is also a. compulsion as it helps to prevent collision after turning. After straight path travel error might accumulate and the mouse mightn't be exactly at the center of the cell and if in such a condition the mouse tries to turn avoiding collision is impossible. Using front sensors can help to realign the mouse at the center position[6].

[7]describes micromouse can be quite useful for mechatronics learners. As the mouse needs to fit in the single small sized cell careful mechanical design of the mouse is of utmost importance. Smaller the mouse, easier it is to navigate and larger the margin of error becomes. Smaller mouse also lowers the center of gravity as well as moment of inertia decreasing the friction and rocking of the mouse. If the robot is bulky and huge even smallest of offset can cause the mouse to crash. Encoder motors use are extremely popular as they are fast and compact. With the addition of encoder of large resolution one can easily track the mouse creating a closed loop control system where encoder sensor provides the feedback. The encoder motors are controlled using a motor driver and speed is controlled using a pulse width modulation. This paper mentions the use of MATLAB for simulation but micromouse being a widely popular competition there are many other dedicated software for simulations like ROS, mackorone mms, gazebo etc.

Micromouse has to perform extremely precise movements. There might be obstacle any direction hence all the movements should be calculated and any real-world problems like difference in coefficient of friction of the surface, certain inclination of the platform, ambient light, difference in the size of the cell or walls shouldn't affect the bot's performance. Hence a control scheme is required and for such task PID controller is the most popular. But as our robot is moving at a high speed there won't be any steady state error hence the I term is excluded. With the removed I term the PID control becomes PD controller. The difference between setpoint and output is calculated and fed to the controller and ultimately to the

motor. [8] propose addition of another derivative term for improved accuracy. The goal of the PID controller is to provide a trapezoidal motion profiling. In ramp profiling the motor is fed with the maximum voltage it can have and the motor is accelerated rapidly. This causes the bot to slip and reduces life expectancy of motor. Hence trapezoidal motion profiling is used where the robot is slowly accelerated until it reaches the maximum distance and then also slowed gradually to reach the destination[8].

The Flood fill algorithm is often coded for micromouse robot as it has helped to won many micromouse competition. But to save the computation time as well as memory space, engineers have shifted towards the use of modified flood fill algorithm. In the flood fill algorithm, the maze is updated with new flood values each time our bot reaches a new cell. But the modified flood fill algorithm does not update maze with new flood values every time a new cell is reached. The flood values are updated only when the flood value of the current cell of our robot is less than or equal to the flood value of the accessible cell. Doing this, the total number of flood values updated to find the shortest path is drastically reduced than that of using flood fill algorithm[9]. So, the use of modified flood fill algorithm on the robot will help it to traverse from cell to cell in a higher speed.

3. REQUIREMENT ANALYSIS

The hardware components used in this project are:

- N20 geared encoder motor
- L293d motor driver
- STM32 blue pill
- Vl53lox TOF sensor

3.1. Encoder Motor

The motor is a brushed dc motor with rated voltage of 6v. It contains a metal gear attached to it with a gear ratio of 1:10. The gear increases the torque by decreasing speed so that the mouse can move easily. The speed at free load is 1500 rpm whereas the rated torque is 0.07 kg.cm with the rated current of 170mA.

The encoder sensor attached to it works on the principle of hall effect, hence also called hall effect sensors. A magnetic disk is attached to the motor which rotates along with the shaft when certain voltage is given to the motor. The magnetic disk is made up off 7 pole pair. In other words, it contains 14 poles containing north and south poles (7 north, 7 south). Along with the magnet the encoder sensor also contains quadrature hall sensors, that senses and informs microcontroller which pole of the magnet it is sensing/facing. Those two sensors are placed in such a way that the phase difference between the signal they output is 90 degrees.

Figure 3-1: Encoder Motor Dimension

The hall sensors provide a way to construct closed loop control system so that the mouse can know how much the mouse has travelled forward, backward or rotated clockwise or counter-clockwise. When the magnetic disk rotates along with the motor shaft the hall sensor senses the change in the pole of magnet it faces and outputs a sequence of pulses. One pole of a magnet represents high portion of the pulse whereas opposite pole represents low portion of the pulse. By counting the total number of pulses, we can easily determine the distance moved by the robot using the information like gear ratio and diameter of the wheel.

If we use only one hall sensor a problem might arise that Is the mouse won't know whether it is moving forward or backwards. As the hall sensor will output pulse regardless of the direction of motion using only one sensor makes the mouse incomplete. Hence to deduce the direction of the motion as well we need two hall sensors angled at 90 degrees hence called quadrature sensors.

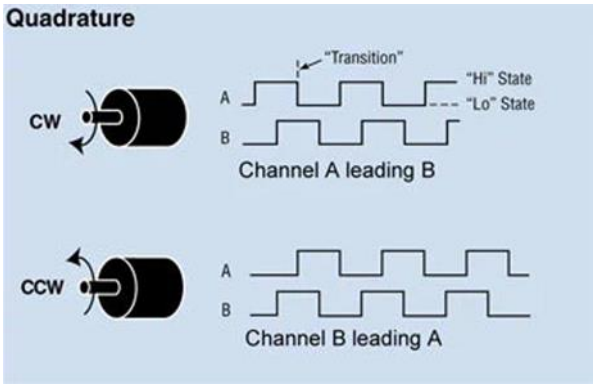


Figure 3-2: Quadrature Hall Sensor Out

The two hall sensors are named say A and B. With the spin off motor both A and B outputs sequence of pulses independently. They are identical but just phase shifted by 90 degrees. When a transition occurs in one sensor by reading the state of another sensor gives us idea of the direction of rotation. Say in a transition of A we read B then we might find the B is high. Now if it the direction doesn't change the condition also remains the same. But as soon as the direction changes well see that the value of B will be low. In this way by this logic, we can deduce the direction and by counting pulse of either A or B we can easily track our mouse in the maze.

Now after we get the pulse from the sensors, we can deduce the distance travelled using formula:

$$Distance = \frac{total\ pulse}{poles\ per\ pair} * \frac{1}{gear\ ratio} * Circumference$$

3.2. L293D motor driver

STM32blue pill can only output max 25mA current from its GPIO pins which is not

sufficient to drive the motor. If the motor is directly connected the large current drawn by the motor will definitely damage the microcontroller. Hence a motor controller is required. Similarly, directly connecting two terminals of the motor to the power sources will limit the functionality of direction control as the wires needs to be reversed or the current must be flown to the opposite direction to change the direction. Such task is also performed by the motor controller.

The motor controller does the task using transistors arranged in h bridge configuration. The arrangement looks like a letter hence called h bridge.

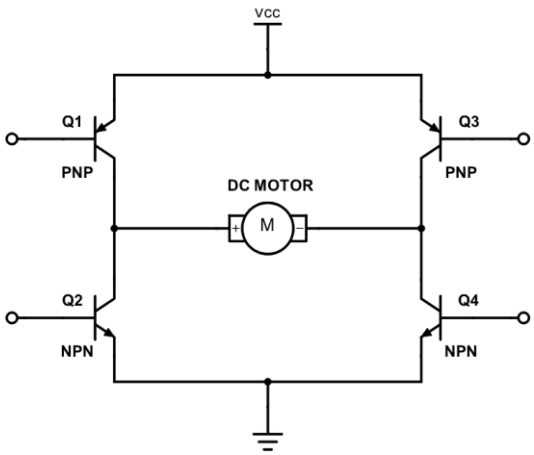


Figure 3-3: H Bridge

say X(left) and Y(right). When 1 is applied to X and 0 to Y then transistor Q2 and Q3 is turned on causing the current to flow from right to left and turning the motor in one direction. When X is 0 and Y is 1 opposite happens and motor turns in another direction. Vcc provides power to the motor. In this way a h bridge is used to drive motor.

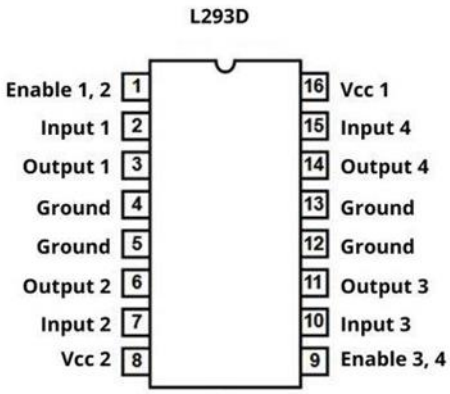


Figure 3-4: L293D Pin Configuration

L293D consists of two half bridges and can drive two motors independently. The inputs from microcontroller are given to pin 2, 7 and pin 10, 15. Vcc1 is for the logic supply whereas Vcc2 is for supplying current to the motor. Enable pins are uses to control the speed of the motor. By applying PWM signals of certain duty cycle and certain frequency we can vary voltage supplied to the motor and hence speed.

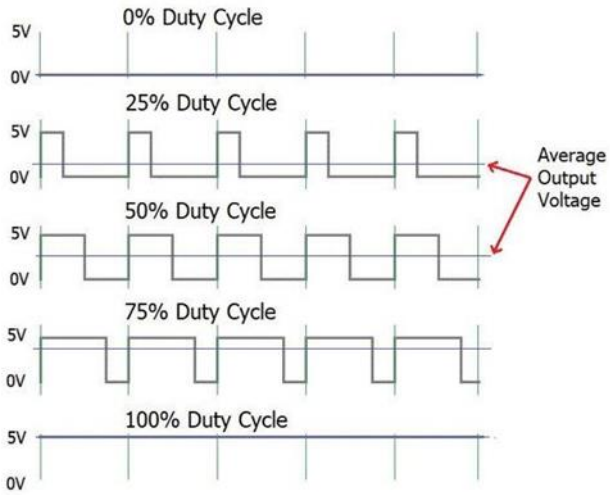


Figure 3-5: Duty Cycle

$$Duty\ Cycle = \frac{on\ time}{total\ time} * 100$$

3.3. STM32 Blue pill

The blue pill uses 32 bit 72 MHZ STM32F103C8T6 microcontroller using arm cortex m3 architecture. Having higher clock frequency, more PWM channels, higher resolution ADC, larger flash memory, SRAM, higher number of GPIO pint STM32F103C8T6 is superior to Atmega328p in every way. Even in terms of pricing blue pill is cheaper than Arduino uno. Hence picking this to Arduino was an obvious choice.

The only drawback of using the blue pill instead of Arduino was the programing. Arduino is beginner friendly and provides all the abstraction for the user for easy use in the cost of performance whereas STM32 HAL is complicated but extremely powerful. The library support is also lacking compared to Arduino.

For the sake of simplicity using blue pill's hardware in Arduino language was the approach we used. Using stm32duino library the programming was simplified.

3.4. VL53L0X

VL53L0X is a time-of-flight sensor that uses high intensity IR laser beam for obstacle detection. It uses 940 nm VCSEL (Vertical Cavity Surface Emitting Laser). Since it uses time of flight method and high intensity laser beam it is highly reliable compared to other methods of detection. It uses i2c method for communicating with the microcontroller and has a unique i2c address.

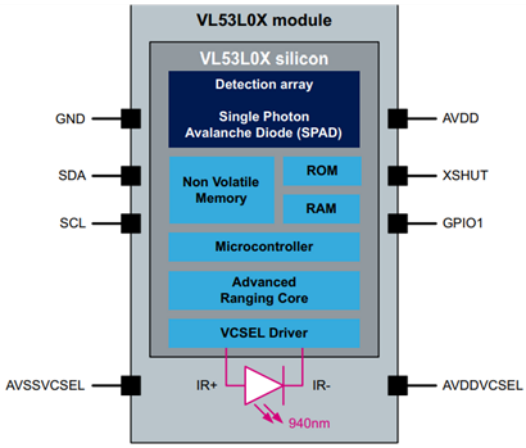


Figure 3-6: VL53L0X Architecture

It uses VCSEL for emitting 940 nm laser and single photon avalanche diode for the detection and calculation of time of flight of sent signal is computed by the inbuilt microcontroller. Hence the only thing the main microcontroller needs to do is to read the data through i2c channel.

The device is initially turned off. Then after application of power the hardware stays in standby mode. Afterward firmware boots and software is in standby mode. Then only the range mode is selected.

There are three modes of operation i.e., single continuous and continuous timed. In single measurement the sensor reads once and then goes to standby. In continuous the device continuously reads until host commands to stop. In continuous timed the device reads waits for a specific period of time and then reads again, the cycle is also ended by host like continuous measurement.

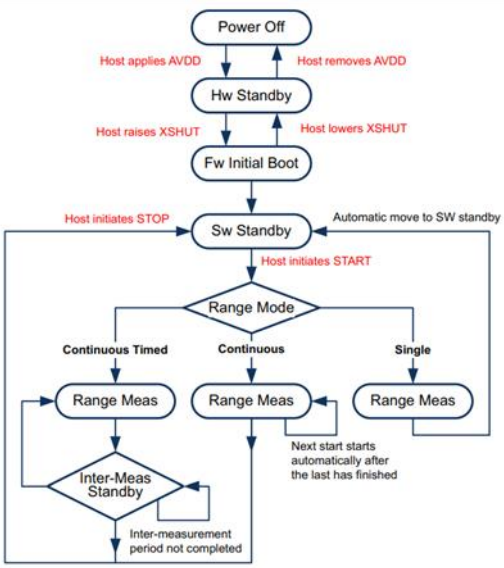


Figure 3-7: Modes of Operation

The device can be used for different purposes and according to the purpose appropriate settings can be set. There is a tradeoff between speed and accuracy.

Range profile	Range timing budget	Typical performance	Typical application
Default mode	30 ms	1.2 m, accuracy as per *Table 12*	Standard
High accuracy	200 ms	1.2 m, accuracy < +/- 3%	Precise measurement
Long range	33 ms	2 m, accuracy as per *Table 12*	Long ranging, only for dark conditions (no IR)
High speed	20 ms	1.2 m, accuracy +/- 5%	High speed where accuracy is not priority

Figure 3-8: Timing Budgets

Here we can see default mode is the best as our mouse requires accuracy as well as speed. The main drawback of use of this sensor instead of IR pair can be the timing budged. For default mode the time required is 30ms which is way slower than IR pair which uses microcontroller ADC to read the values.

3.5. Timers

Timers are essential for our mouse. At a certain interval of time an interrupt must be generated periodically to perform various computation and decision making like whether to accelerate of decelerate, when to read the sensors etc. Hence the interrupt required is generated by microcontroller's timer/counter. When it is timed by some external event it acts as a counter whereas when it its timed through system clock it acts as a timer. By using pre-scaling, setting overflow value and configuring modes of timer we can generate timer of different intervals.

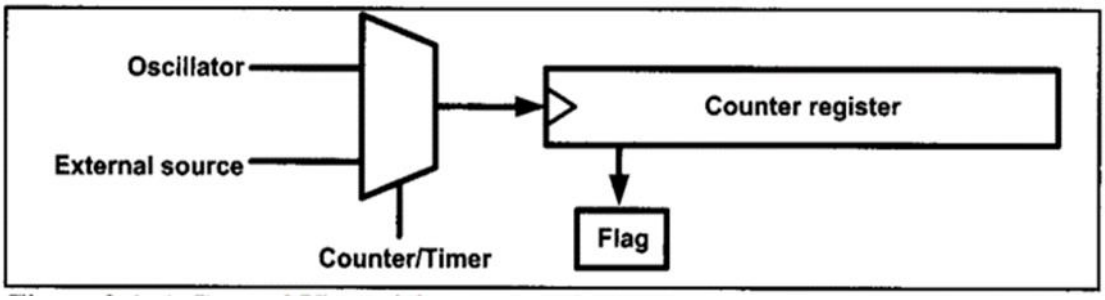


Figure 3-9: Timer/Counter

3.6. MATLAB

Many robotic applications involving motion are characterized as dynamic systems. Working on a control of a dynamical system can be very tedious to find correct gains for a control system by hit and trials. One might spend a lot of time tuning the system in real hardware just to discover that the control law was not good enough. There are also other problems such as having to write code, debug and upload it to hardware multiple times. It is not time efficient to manually tweak the gains and run the code in the hardware system throughout the project. The process might even damage the sensitive and expensive hardware components. There is an alternative to this hassle which is to perform simulation of the control system before working with real hardware thus saving time and money. Finding the transfer function of the system we are working with allows us to perform the simulation.

Transfer function is a mathematical description of the dynamical system which represents the system as the ratio of output to input. Basically, it means that if we have a transfer for a system, we can simply get the output by multiplying the transfer function with input. It is not easy to derive a transfer function for any dynamical system using the first principles. Difficulty increases with the complexity of the system and the presence of extra sources of force and torques which are difficult to model. It is not best practice to derive the transfer function manually because the equations of motion, with different sources of friction should be considered, after all the work the transfer function is not guaranteed to be precise.

3.7. System Identification Toolbox

MATLAB is equipped with an application called *System Identification Toolbox* which eliminates the complexity in identifying the transfer function of any dynamic system. It is a data driven system which estimates the best transfer function for given input-output data. The response of the system at constant input (i.e., step response) is recorded from the hardware system itself from the actual experiment. The step response is fed to the application which determines the best transfer function for input values.

3.8. Robot Operating System (ROS)

ROS is a flexible framework for writing and executing code for robots. It provides a standardized communication interface, allowing different components of a robotic system to communicate and interact with each other.

3.9. Gazebo

Gazebo is a separate software from ROS, but is tightly integrated with ROS and is widely used for simulation and testing of robots. It provides realistic physics and sensor simulations, including support for 3D rendering. It can be used to simulate robots and their environments including testing and debugging of control algorithms. Gazebo provides a different ROS plugin that allows it to interact with ROS nodes and

topics, enabling ROS nodes to control robots in the simulation and receive sensor data from the simulated robots. We are using two different ROS plugins for our simulation of micromouse in the Gazebo environment. They are: differential_drive_controller and gazebo_ros_head_hokuyo_controller.

4. SYSTEM ARCHITECTURE AND METHODOLOGY

4.1. Hardware Architecture

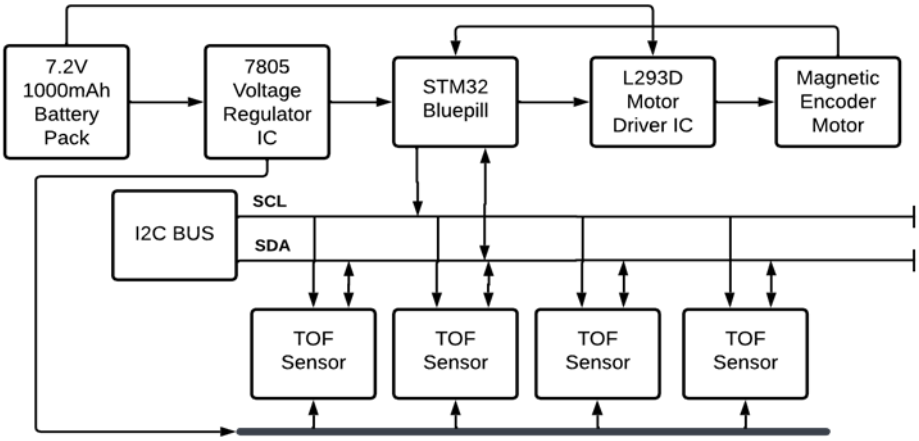


Figure 4-1: Hardware Architecture

4.1.1. 7.2V 1000 mAh Battery Pack

This is the power source for our micromouse robot. Two lithium-ion cells of 3.6V 1000mAh are connected in series to make a suitable battery pack for our project.

4.1.2. 7805 Voltage Regulator IC

This IC is used to provide +5V regulated power supply to our microcontroller and TOF sensors.

4.1.3. STM32 BluePill

We are using a Stmicroelectronics microcontroller for our project. This is small in size and runs at a magnificent frequency of 72MHz max. It can be programmed easily using Arduino IDE and also has large memory space which makes it a best choice for our project.

4.1.4. L293D Motor Driver IC

It is a commonly used 16-pin motor driver IC which can control two dc motors simultaneously in any direction. It can handle current up to 600mA in each channel at voltages from 4.5V to 36V. As our micromouse robot is a small light-weight robot, there will be no current related issues by using this IC.

4.1.5. Magnetic Encoder Motor

It is very critical to keep track of the distance traveled by our robot in the maze. For that, dc motors with magnetic encoders will constantly provide us with feedback on distance covered by our robot as well as the direction of rotation of the motor shaft.

4.1.6. TOF Sensors

The micromouse robot should be able to detect walls around it as flawlessly as possible to avoid hitting walls of the maze while moving. So we are using four VL53l0X Time-of-Flight sensors which calculate distance based on the time difference between emission of the laser and its return after being reflected by an obstacle. It is immune to ambient lights as well as has fast response.

4.2. Algorithm

4.2.1. Modified FloodFill Algorithm

The main objective of the micromouse robot is to traverse through an unknown maze completely by itself and reach the center of the maze in the shortest possible time. This job requires an algorithm that will guide our robot on its movement inside the maze to find the center/destination while consuming less memory as well as time. Modified FloodFill Algorithm is widely used all over the world as the most efficient maze solving algorithm.

First of all, we flood the maze with numeric values starting from the center (as 0) to all directions until the four last corners of our maze evenly assuming there are no walls inside the maze. It is analogous to pouring the water from the center of the maze as it flows towards all directions evenly. We refer to these values as initial flood values. Now, we update these values according to the current location of our robot in

the maze and its surrounding wall information. These values provide the direction to our robot for traversing the maze. After the robot has completely traversed the maze, these flood values will be arranged in such an order that they provide us the shortest path from start to the center of the maze.

4.2.2. Queue Data Structure

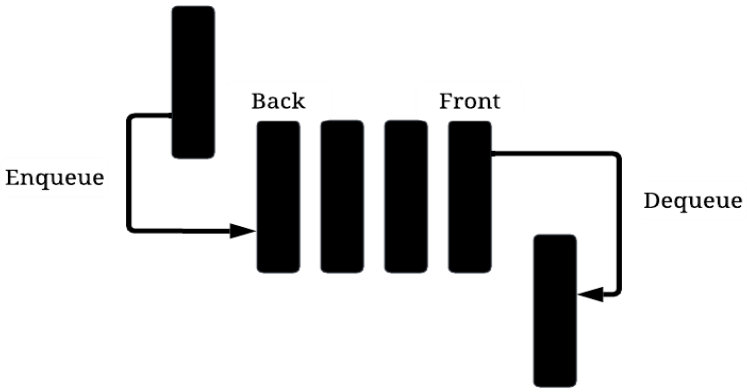


Figure 4-2: Queue Data Structure

Queue is the First In, First Out data structure where you can put things inside from one end and take things out from another end. The process of adding an item to the back of the queue is called enqueue and taking an item from the front is called dequeue. In our case, we will use the queue as the list of cells to examine while using floodfill algorithm.

4.2.3. Modified FloodFill Algorithm Pseudocode

- Add current cell to the queue
- While queue is not empty
    - Take front cell from queue
    - Get front cell's minimum values among accessible neighbors
    - If front cell's value <= minimum of its neighbors
        - set front cell's value to minimum+1
        - add all accessible neighbors to queue
    - Else continue.

4.3. Working Principle of Micromouse Robot

The working of our robot starts as soon as we put it in the starting cell of the maze. The first step is to detect the surrounding walls of the cell. There are four tof sensors used to detect front, right and left walls respectively. The output from the sensors when processed by our microcontroller, provides the information on the availability of walls in three different directions. The wall information is then updated and stored in memory. Now, the FloodFill algorithm generates the next move of our robot inside the maze. The robot responds correspondingly and makes a move using motors. Our software also keeps track of orientation and position of the motor inside the maze. After that it analyzes whether it has reached the center or not. If not, the process continues right from detecting walls to the generation of new moves. If it has reached the center, it will now traverse back to the start and perform the fast run.

4.4. Flowchart Of Working Principle

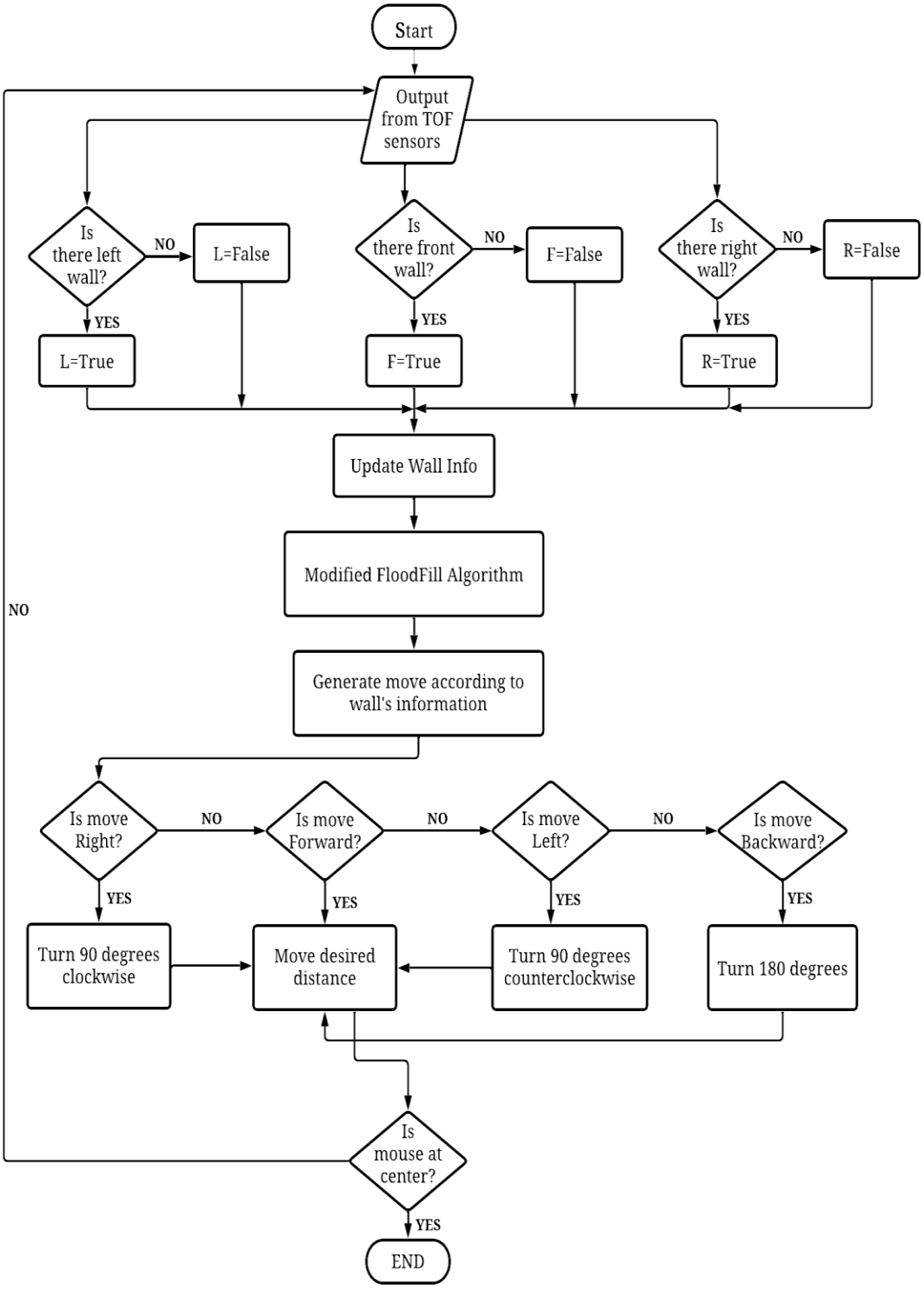


Figure 4-3: Flowchart Working Principle

5. IMPLEMENTATION DETAILS

5.1. Sensors

The mouse uses four sensors two facing at 45 degree and two directly facing forward. Due to high timing budget of around 30ms the use of two side sensors were discarded as they both add 30ms more time each.

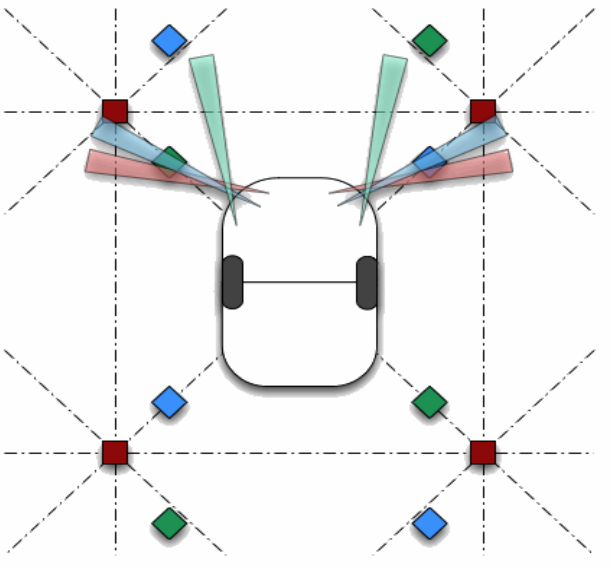


Figure 5-1: Sensor Position

The sensors are placed such that when at center the 45-degree sensor senses the post or the corner of the cell.

There are four identical sensors. Even though the sensor model has unique i2c address

since all sensors are of same model all have same i2c address. Now to obtain data from them without miscommunicating certain method is used which is explained by the following points.

- Reset all sensors by setting all of their XSHUT pins low for delay (10), then set all XSHUT high to bring out of reset.
- Keep sensor #1 awake by keeping XSHUT pin high
- Put all other sensors into shutdown by pulling XSHUT pins low

- Initialize sensor #1 with lox.begin(new_i2c_address) Pick any number but 0x29 and it must be under 0x7F. Going with 0x30 to 0x3F is probably OK.
- Keep sensor #1 awake, and now bring sensor #2 out of reset by setting its XSHUT pin high.
- Initialize sensor #2 with lox.begin(new_i2c_address) Pick any number but 0x29 and whatever you set the first sensor to
- Repeat for each sensor, turning each one on, setting a unique address

5.2. Encoder motor

The motor is controlled by L293D motor driver. The microcontroller provides PWM signal using analogWrite(pin_no,dutycycle) which ultimately controls the speed.

ISR (Interrupt Service Routine) is written and is called whenever a pulse is obtained from the encoder sensor. One hall sensor is connected from each motor to the hardware interrupt pin of stm32 bluepill which supports 16 hardware interrupts at a time.

Pseudo code:

```
// interrupt is attached to sensor A
ISR {
Digital read sensor B;
If B==high;
Dir=clockwise; ++total pulse;
Else Dir=anticlockwise; --total pulse;
}
```

5.3. PID controller

$$u(t) = kp * e(t) + ki \int e(t)dt + kd * \frac{de(t)}{dt}$$

Pseudo code:

```
//distance controller
Void loop ()
 {
Define kp, kd, ki = 0;
Error = target distance-distance travelled; //e(t)=setpoint-output
Output_to_motor = Pid_fun(kp, kd, ki, error);
PWM = Limit (Output_to_motor, -255,255);
AnalogWrite(pin,PWM);
If distance travelled = = distance travelled
End loop;
}
```

5.4. System Identification of the Micromouse robot

In order to make sure that the method and logic that we are following are correct, we took our fully functional Micromouse robot and tried to perform its system identification. The dynamic system in our case is the the robot with two encoder motors. For any system we need to be clear about the input and output of the system because these play an important role in the definition of the transfer function. In our case the input to the Micromouse is voltage whereas the output is the angular velocity of the bot which is also the average velocity of the encoder motors. The angular velocity feedback was taken using the encoder of the motor which gives a constant number of pulses on each round. As the requirement for the system identification application is the output of the motor for constant input, we set up the motor and its encoder into a microcontroller and logged the angular velocity for step input of 7.52V.

5.5. Data collection and System Identification of Robot

We can compute the angular velocity of the motor by sampling the pulses from the encoder in specific sample times. Formula to calculate the velocity of the motor using the hall sensors is.

$$Angular\ Velocity = \frac{Circumference * 1000 * ds}{dt * N}$$

Where,

PI = 3.14

dt = Sample time

ds = Number pulses from encoder during sample time

N = No of pulse at 360-degree rotation of motor shaft

After code is uploaded into a microcontroller including the implementation of the above equation, we get the rise of velocity from zero to some constant value after supplying constant input voltage when the motor is at rest. Using the program CoolTerm we accessed the serial port for the microcontroller and logged the angular velocity data into csv format.

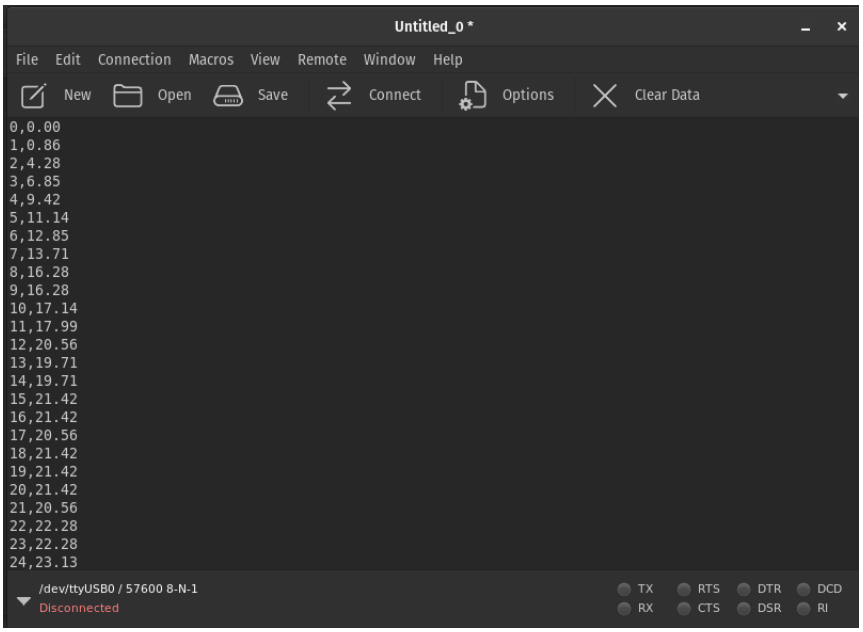


Figure 5-2: Serial Data Logging

The system identification of the Micromouse robot is now performed using the System Identification Toolbox in the matlab. The values we logged from arduino are stored as matrices as all matlab computations are performed in matrices and vectors. After feeding the response of the motor to the toolbox we will be provided with the transfer function for our motor. Now we can perform a simulation of the control system using this transfer function as if it were a real motor.

5.6. Maze Mapping

The robot should know the exact configuration of a real maze and store it in the software. The standard micromouse maze is a 16*16 maze with 256 cells. Each cell may have any of the 16 different configurations of walls. So, we assign each wall configuration with an integer number.

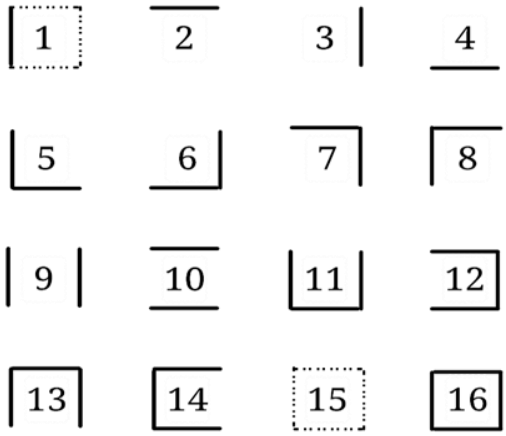


Figure 5-3: Maze Mapping

5.6.1. Robot Current Position

It is of utmost importance to keep track of robot position inside the maze. As we have to update the wall information for each cell, we need to know exactly which cell our robot is currently at and perform updates on that cell only.

5.6.2. Orientation Of Our Robot

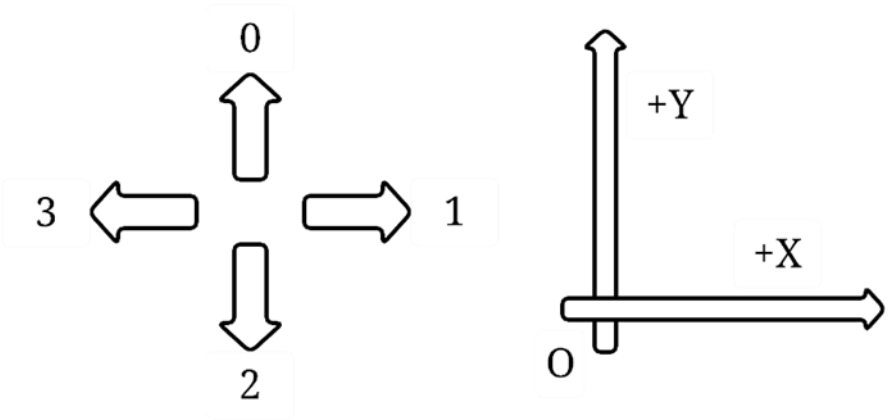


Figure 5-4: Orientation

We represent different orientations of our robot by integer number from 0 to 4. 0 is for the north facing, 1 for the east facing, 2 for the south facing and 3 for the west facing of the robot. The wall configuration differs according to the orientation of our robot inside the maze. The position of our robot also depends on the orientation of the robot. If the robot is facing north and moving forward, then the y coordinate will increase keeping the x coordinate constant. Also, if the robot is facing east and moving forward, then the x coordinate will increase keeping the y coordinate constant and so on for other orientations.

5.6.3. Storing Info in Memory

Now, we need to store the maze information in memory for further computations. We define a 16*16 integer array where we store the integer value that we assigned for 16 different configurations of walls at each cell of the maze. That array will be updated as soon as our robot makes a movement and traverses to a new cell.

The 16*16 integer array is initialized to store the flood value of each cell in the maze. Flood values are those values that represent the distance between a particular cell and the destination. The initial flood values are generated assuming there are no walls on any of the 256 cells.

```
uint16_t flood[16][16] = {{14, 13, 12, 11, 10, 9, 8, 7, 7, 8, 9, 10, 11, 12, 13, 14},
                          {13, 12, 11, 10, 9, 8, 7, 6, 6, 7, 8, 9, 10, 11, 12, 13},
                          {12, 11, 10, 9, 8, 7, 6, 5, 5, 6, 7, 8, 9, 10, 11, 12},
                          {11, 10, 9, 8, 7, 6, 5, 4, 4, 5, 6, 7, 8, 9, 10, 11},
                          {10, 9, 8, 7, 6, 5, 4, 3, 3, 4, 5, 6, 7, 8, 9, 10},
                          {9, 8, 7, 6, 5, 4, 3, 2, 2, 3, 4, 5, 6, 7, 8, 9},
                          {8, 7, 6, 5, 4, 3, 2, 1, 1, 2, 3, 4, 5, 6, 7, 8},
                          {7, 6, 5, 4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 5, 6, 7},
                          {7, 6, 5, 4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 5, 6, 7},
                          {8, 7, 6, 5, 4, 3, 2, 1, 1, 2, 3, 4, 5, 6, 7, 8},
                          {9, 8, 7, 6, 5, 4, 3, 2, 2, 3, 4, 5, 6, 7, 8, 9},
                          {10, 9, 8, 7, 6, 5, 4, 3, 3, 4, 5, 6, 7, 8, 9, 10},
                          {11, 10, 9, 8, 7, 6, 5, 4, 4, 5, 6, 7, 8, 9, 10, 11},
                          {12, 11, 10, 9, 8, 7, 6, 5, 5, 6, 7, 8, 9, 10, 11, 12},
                          {13, 12, 11, 10, 9, 8, 7, 6, 6, 7, 8, 9, 10, 11, 12, 13},
                          {14, 13, 12, 11, 10, 9, 8, 7, 7, 8, 9, 10, 11, 12, 13, 14}
};
```

Figure 5-5: Floodfill Array

5.6.4. Generation of next move

Our robot detects the surrounding walls of the current cell using its TOF sensors and knows about the surrounding cells which it can easily access. All neighboring cells have flood values assigned to them already. It compares the flood values among all accessible neighbors and moves towards that neighbor which has the value one less than that of the cell where our bot is currently at. If there are multiple choices, the priority is given for the one that takes the bot closer to the destination.

5.7. Making Flood Values Consistent

Such a situation may occur where the robot cannot access the cell whose flood value is one less than that of the robot's current cell. But the cells that our robot can access have a comparatively greater flood value. In this case, We have to update the flood value of our current cell by making it one more than the minimum value among the accessible neighbors. Here, In the Figure 5-6, the flood values are not consistent if the robot is at the lower left corner. The Figure 5-7 represents the array after making it consistent.

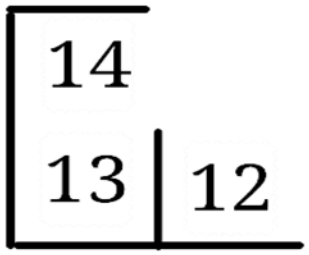


Figure 5-6: Inconsistent Array

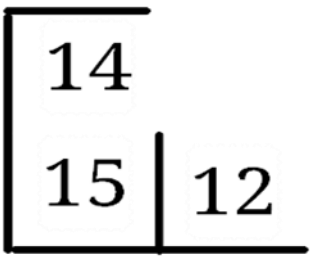


Figure 5-7: Consistent Array

5.7.1. Update the Flood Array

There will occur a situation where multiple flood values should be made consistent. In such a case, we initialize a queue and add the current cell to the queue. Then, we take the front cell of the queue for consideration. If that cell is not consistent, we make it consistent by using the above-mentioned method and add all accessible neighbors to the queue. Else, we omit that step. The process continues until the queue becomes empty.

6. RESULT AND ANALYSIS

6.1. Result of Wall Detection

The Figure 6-1, 6-2 show the wall detection graph for the front right sensor and front left sensor. At first, the distance detected by the sensors are around 300 mm and 270 mm respectively for a particular wall configuration but when it moves close to the wall, the distance decreases and settled at around 50 mm .

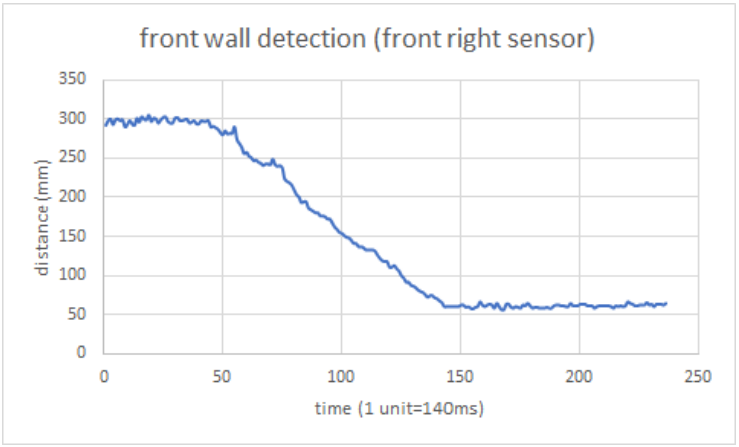


Figure 6-1: Front Right Sensor

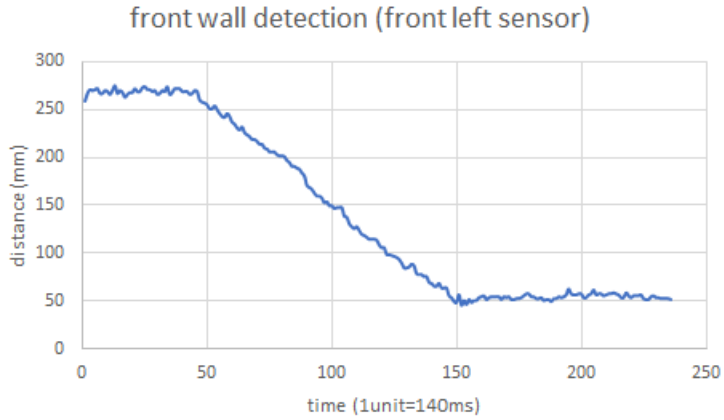


Figure 6-2: Front Left Sensor

The Figure 6-3, 6-4 show the transition phase of left and right sensor when it detects left and right wall respectively to no wall detection. The distance obtained for detecting left wall is around 160 mm and for right wall is around 140 mm. When the robot moves forward it detects the front wall, the left and right sensor also detect the same wall so the value is settled at around 80 mm for left and right sensors too.

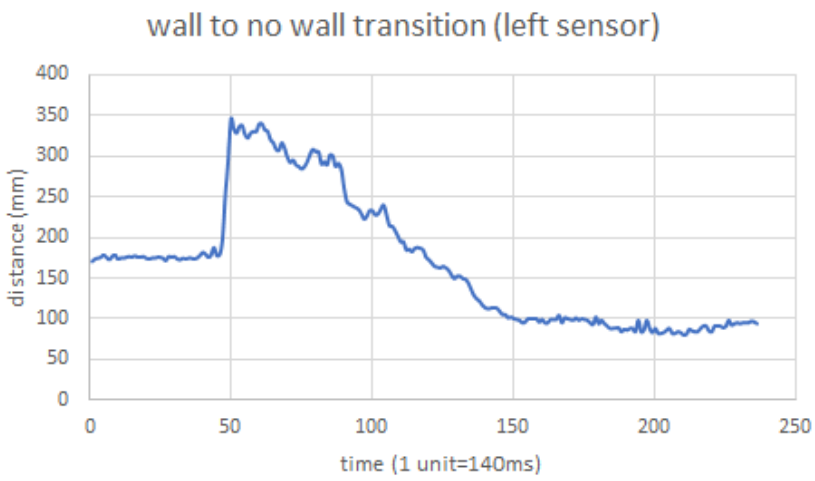


Figure 6-3: Left Sensor

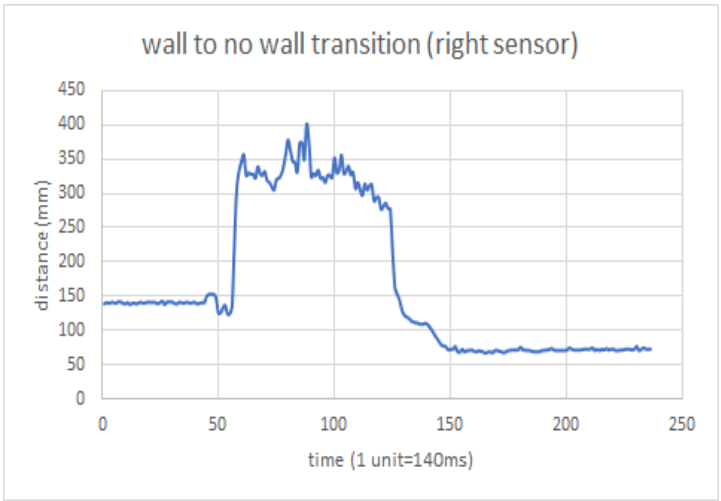


Figure 6-4: Right Sensor

6.2. Result of System Identification

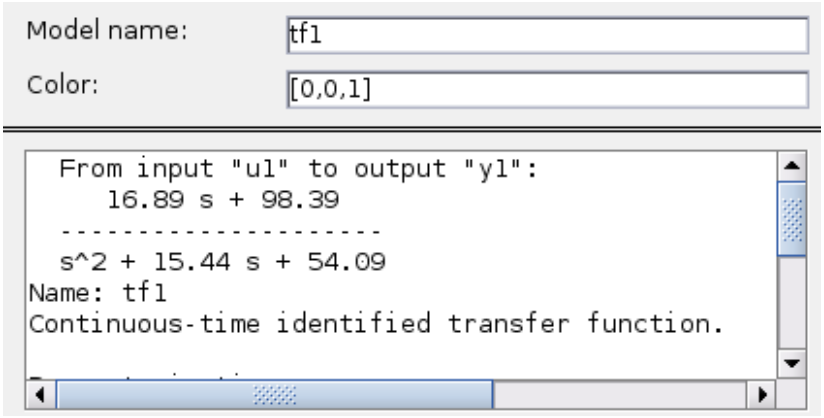


Figure 6-5: System Identification of Micromouse

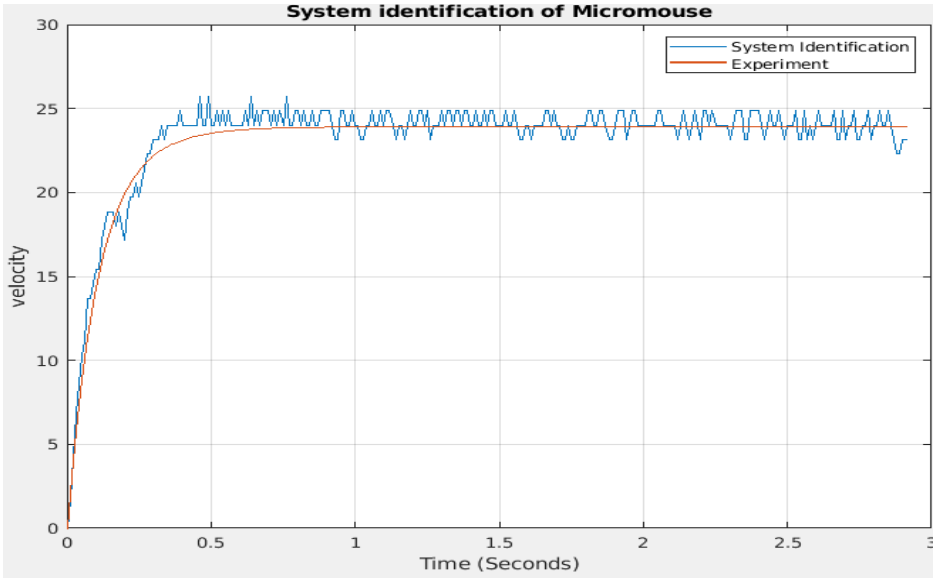


Figure 6-6: System Identification Plot

Transfer function for our motor generated by the System Identification Toolbox is:

$$Transfer\ Function = \frac{16.89\ s + 98.39}{s^2 + 15.44\ s + 54.09}$$

Blue plot is the response of the motor from the experiment. Irregular pattern in the plot is due to the low resolution in the encoder of the motor. Similarly, the red plot is the 12V step response plot of the transfer function. It can be clearly seen that the graph of experiment and the system identification fits to a good degree. This validates our

process hence giving us confidence to continue control system design using the transfer function.

6.3. Control System Design of Motor

The goal of system identification is the control system design of the dynamical system. Here the transfer function of the motor represents the *plant* and the PID control system is used to control the angular velocity of the motor. It should be noted that the ultimate goal is to control the position, distance, and rotation of the Micromouse using the control system.

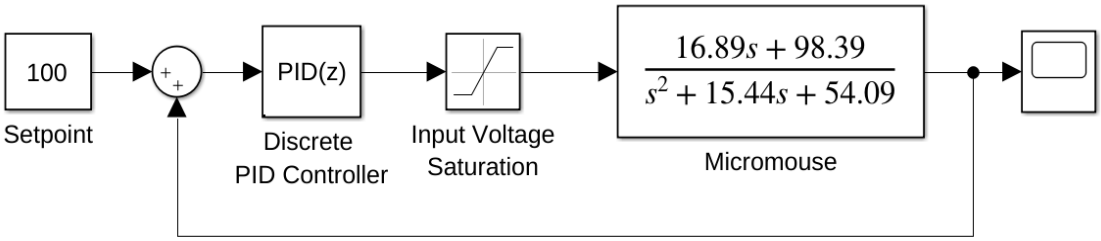


Figure 6-7: Control System

Above block diagram demonstrates the implementation of the PI controller in the plant.

Plant is the same transfer function identified using System Identification Toolbox. Above system is provided with a constant set point of 100. Voltage saturation of -12 V to 12 V is assigned to maintain the output from PI control in between that saturation. After completing the design of the control system, we use the auto PID tuning feature of MATLAB. The result from the control system with untuned PI and tuned PI is examined. Result below shows that our Robot is a linear system and we can use PID controller to control our robot. The obtained control parameters for our tuned PID response are shown in Figure 6-8.

6.3.1. Results of velocity control of Encoder DC motor

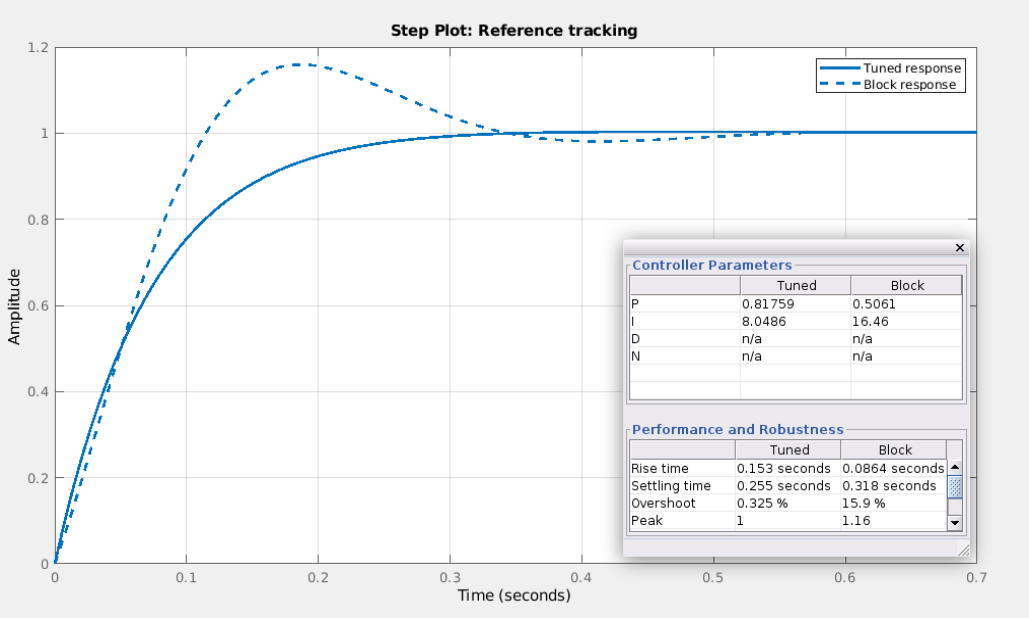


Figure 6-8: Tuned and Untuned PID Plot

6.4. Gazebo Simulation

6.4.1. Micromouse Arena

The environment of our simulation is a 16*16 maze where each cell is a square of size 16.8 cm. We create a SDF (Simulation Description Format) file that describes objects and environments for robot simulators.

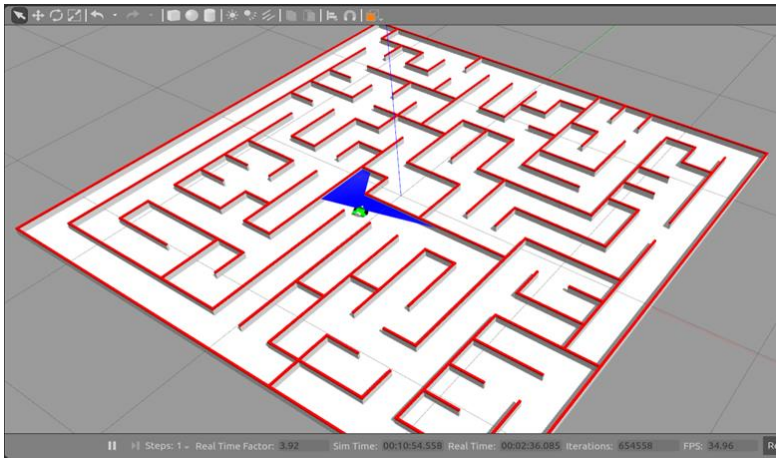


Figure 6-9: Simulation Arena

6.4.2. Micromouse Robot

The robot in the simulation should also be small enough to navigate properly in the maze without hitting any walls. To make such custom robots' model in gazebo, we make use of URDF (Unified Robotic Description Format) files. We describe all the elements of our robot: chassis, wheels and sensor, inside this file and also add ROS plugins for communication with ROS nodes later to perform navigation.

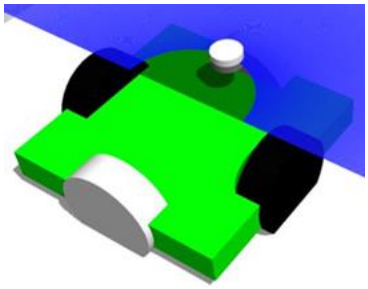


Figure 6-10: Simulation Bot

6.4.3. Nodes and Topics

A node is a single process in ROS that communicates with other nodes to send andreceive the messages. A topic is a named communication channel that nodes can use to publish and subscribe to messages. Here, we made a node named '/botMovement' which publishes a message to a topic, '/cmd_vel_x' and subscribes from the topic, '/laser/scan'.

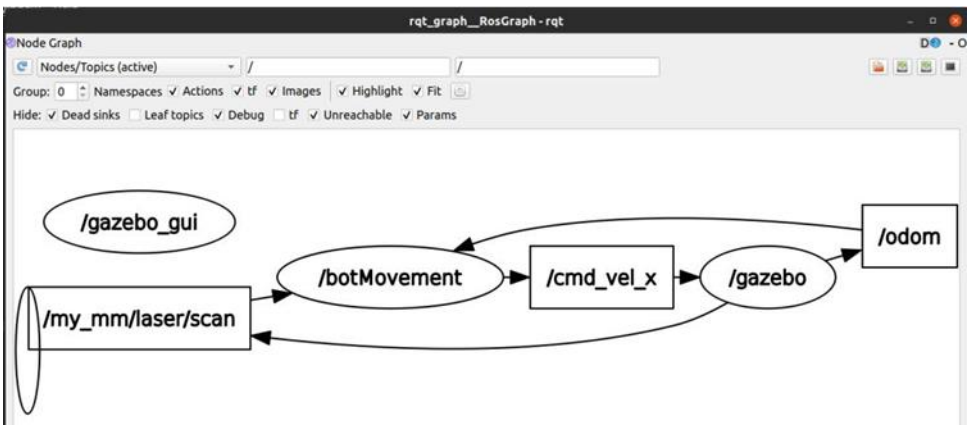


Figure 6-11**:** Nodes and Topic

6.5. Result of Maze Solving

The figures below show the updated flood values in the maze and the shaded parts represent the shortest path in the maze from starting position to the center of the maze. The starting position is at the lower left corner of the maze. We have tested the algorithm in multiple mazes and it works well on all of them.

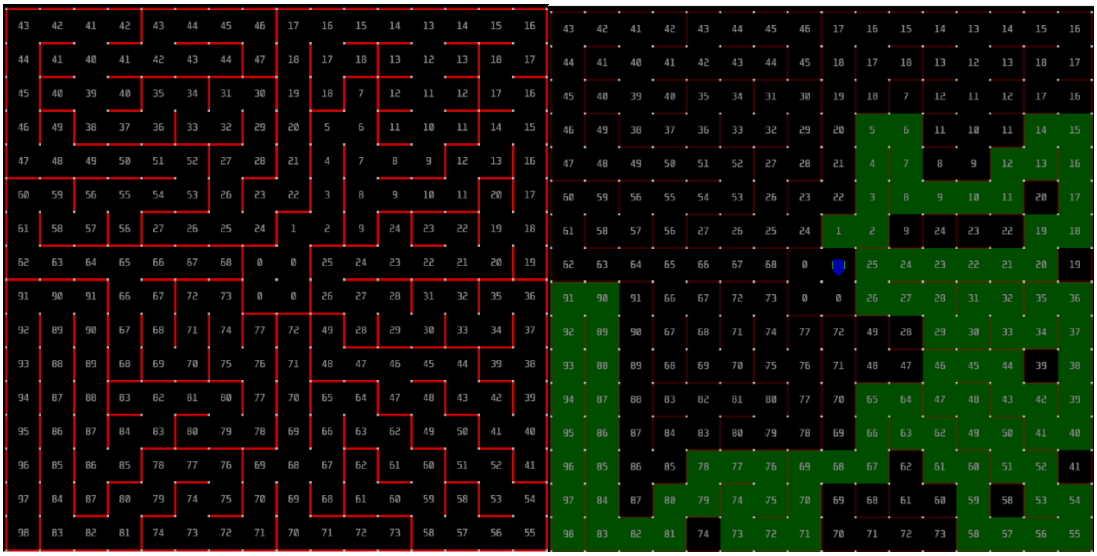


Figure 6-12: Shortest Path in Maze 1

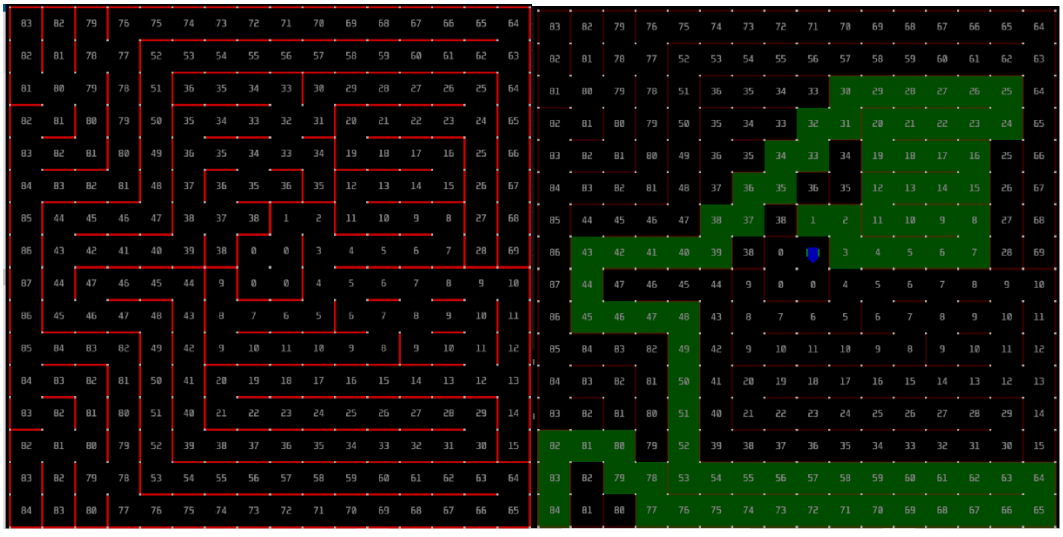


Figure 6-13: Shortest Path in Maze 2

7. Future Enhancement

The first thing that we can do is the addition of curve turns and diagonal runs. Now the bot performs a pivot turn to change the direction of motion which is it first stops at the center of the cell and then rotates the two wheels in opposite direction. This method of turning is slow or time consuming as the robot has to slow down and then completely stop first then only it can turn then it has to accelerate again for forward motion. To tackle this problem curve turning can be implemented which is the robot senses whether it has to turn in next cell or not and then it slows down a little then changes the speed of two wheels to create a turning effect. For example, if it has to turn right it increases the speed of left and then decreases the speed of right wheel and the net forward and rotation effect will turn the robot without stopping. For this a velocity profile is precalculated which gives the reference velocities to the robot that it should have at certain position so that it doesn't crash to the walls.

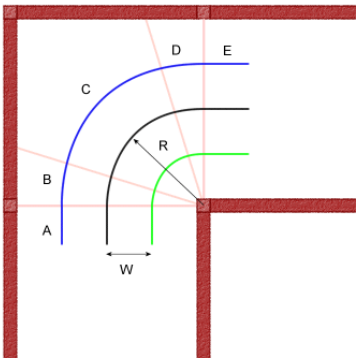


Figure 7-1: Curve Turn

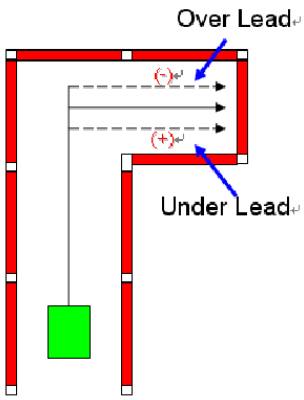


Figure 7-2: Pivot Turn

The next thing is the diagonals detection. When multiple consecutives turning path is present in the maze then it is optimal to travel the Euclidean distance rather than Manhattan distance. Rather than turning multiple times we can just move in a straight path with diagonal path detection which will definitely reduce the travel time during fast run.

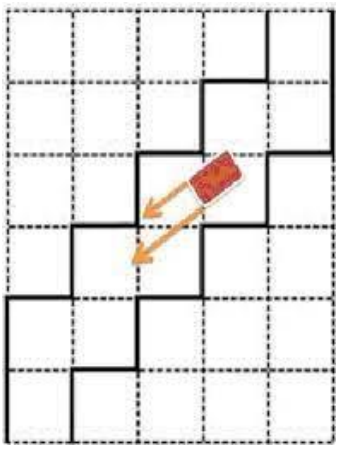


Figure 7-3: Diagonal Detection

Another important aspect to increase the performance is the use of suction decrease the slippage and increase the grip on the maze. In fact almost all the fastest mouse competing in the international competitions definitely use a suction fan on the mouse to prevent slippage. The grip is so strong that depending upon the surface the robot can withhold its whole weight upside down

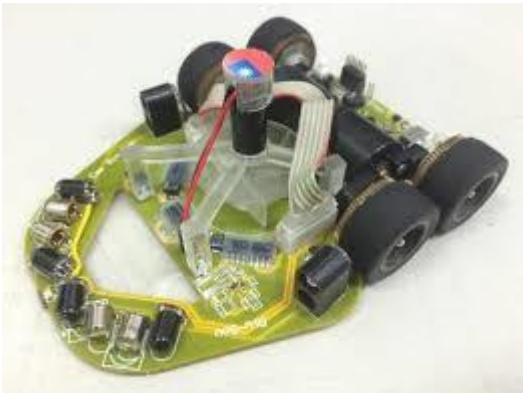


Figure 7-4: Mouse with Suction Fan

Implementation of FPGA is also promising to improve the performance. When only one microcontroller is used all the computation as well as the logical or decision making is done solely by the microcontroller. And as microcontroller are designed to do so much its performance isn't optimal when used alone hence using a FPGA for computation of PWM generation or another microcontroller along with the main microcontroller to divide the work as decision making (algorithm processing) and robot motion control can help to boost the performance greatly.

8. Conclusion

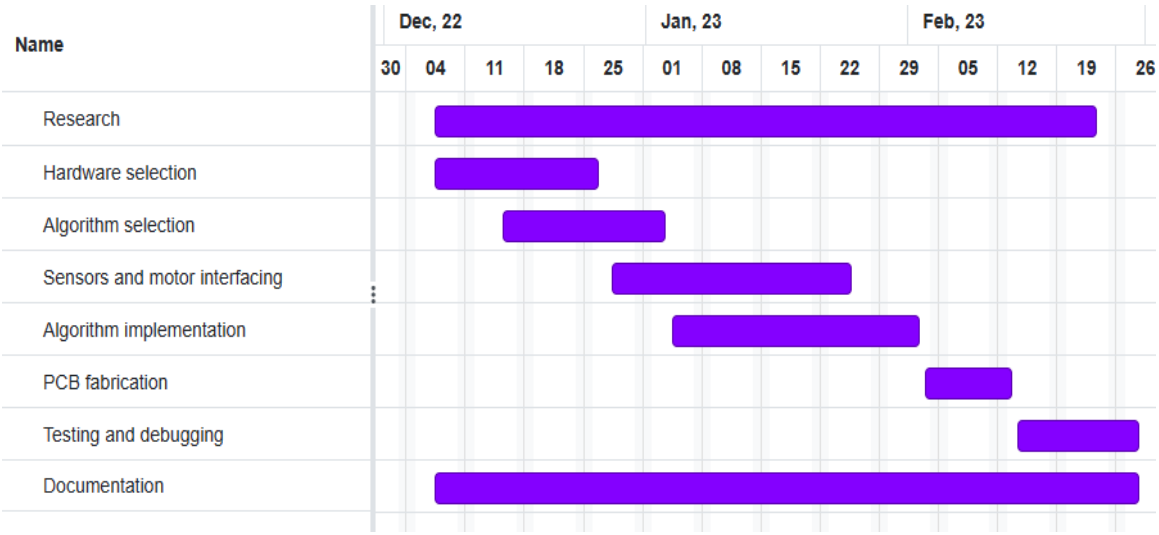
Maze-Solving involves quality control engineering and efficient algorithms. In this project, we designed a robot which is small in size and can solve the 16*16 maze unaided.

Using modified flood fill algorithm, we were able to find the shortest path between the starting point and the center point of the maze. The PID control algorithm helped us to navigate our robot efficiently without hitting any walls of the maze. The algorithms were also tested in the simulated environment before putting it into the robot.

9. APPENDICES

9.1. Appendix A: Project Schedule

Table 9-1: Project Schedule



9.2. Appendix B: Project Budget

Table 9-2: Project Budget

S.N	COMPONENTS	QUANTITY	PRICE (in Rs.)
1	PCB	1	180.00
2	Encoder Motor	2	1200.00
3	Wheel	2	100.00
4	VL53L0X	4	4,800.00
5	L293D	1	120.00
6	STM32 BluePill	1	1,200.00
7	3.6V 1000 mAh Lithium-ion cell	2	800.00
8	7805 Voltage Regulator	1	50.00
	TOTAL		**8,450.00**

9.3. Appendix C: Code Snippets

```
void floodfill(int16_t botx, int16_t boty) {
  enqueue(botx);
  enqueue(boty);

  while (q.front != -1) {
    botx = dequeue();
    boty = dequeue();
    if (isConsistant(botx, boty)) {
      continue;
    } else {
      makeConsistant(botx, boty);
      getSurrounds(botx, boty, &botx0, &boty0, &botx1,
          &boty1, &botx2, &boty2, &botx3, &boty3);
      if (botx0 >= 0 and boty0 >= 0) {
        if (isAccessible(botx, boty, botx0, boty0)) {
          enqueue(botx0);
          enqueue(boty0);
        }
      }
```

Figure 9-1: Floodfill Algorithm

```
//  exporting the response value for system identification
currentMillis = millis();
if (currentMillis - previousMillis > dt) {
  previousMillis = currentMillis;


  //
  M1.drive(125);
  M2.drive(125);

  float ratioA = float( encoderValueA) / pulseperRotation;
  float ratioB = float( encoderValueB) / pulseperRotation;
  float angularA = 2*PI*2.1*((1000 * ratioA) / dt);
  float angularB = 2*PI*2.1*((1000 * ratioB) / dt);
  float angularAVG = (angularA + angularB) / 2;

  Serial.print(samples);
  Serial.print(",");
  Serial.println(angularAVG);

  samples++;
  //
  encoderValueA=0;
  encoderValueB=0;
```

Figure 9-2: Serial Data Logger

9.4. Appendix D: System Identification

```
input_voltage = 13.15;
input = input_voltage * ones(1, length(time));
data = readmatrix('C:\Users\luffy\Desktop\velocity_data.csv');
velocity = data';
output = velocity(2,:);

systemIdentification();
s=tf('s');
plant2 = (-6167*s + 3.21e5)/(s^2 + 463.5*s + 7119);

plotoutput = lsim(plant2, input, time);
plot(time, output,'linewidth',1.2);
hold on;
plot(time, plotoutput,'linewidth',1.5);
```

Figure 9-3: System Identification Code

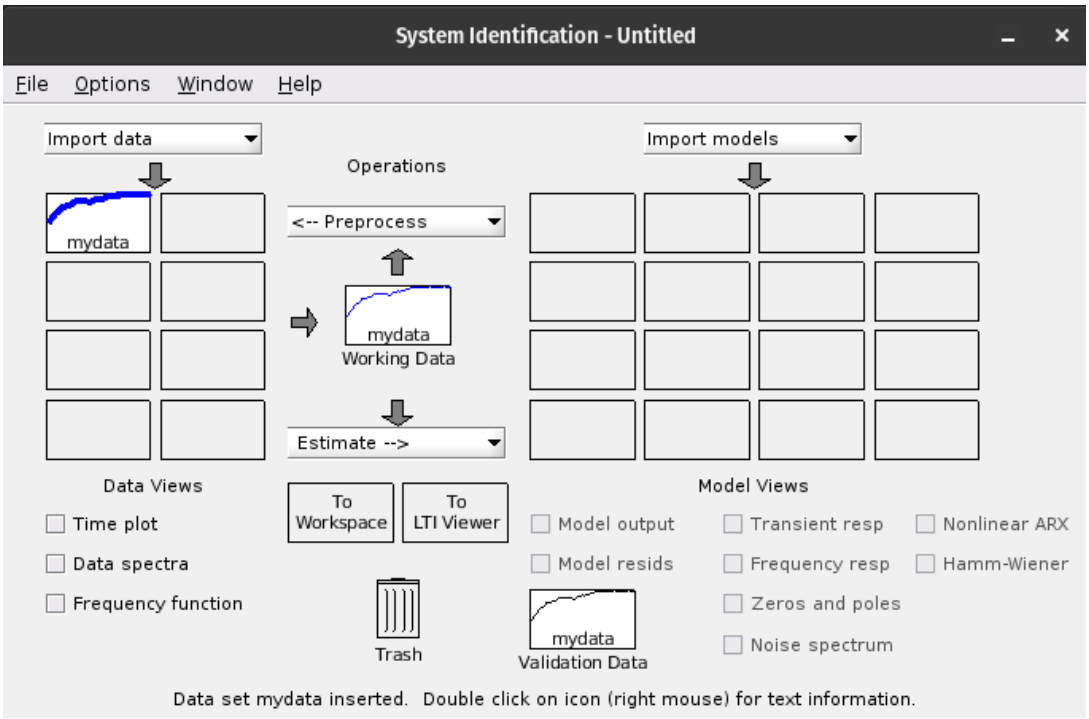


Figure 9-4: System Identification Toolbox

9.5. Appendix E: Used Linux Commands

General Commands

- *clear*: Clears the terminal of previously run commands and text.
- *exit*: Closes the linux terminal.

File/Directory Commands

- *cd /abc/xyz*: Changes the current directory to the *abc/*xyz directory.
- *ls*: Lists files in current directory.
- *mkdir name*: Creates a new directory named ‘name’ inside the current directory.
- *touch file_name*: Creates a new empty file of name ‘file_name’.
- *chmod +x filename*: Changes file permission to make it executable.

ROS Commands

- *catkin_make*: Builds the code inside catkin workspace.
- *rosnode list*: Displays the nodes running on ROS.
- *rqt_graph*: Provides GUI for visualizing ROS computation graph.
- *rosrun* <*package_name*> <*file_name*>: Runs the node.
- *rostopic list*: Displays the topics of the nodes.
- *rostopic info* <*topic_name*>: Provides publishers and subscribers of the topic.
- *rosmsg show* <*type_of_topic*>: Displays what inside the message that is being sent.
- *roslaunch* <*package_name*> <*file_name*>: Opens launch files of Gazebo.

9.6. Appendix F: Schematic

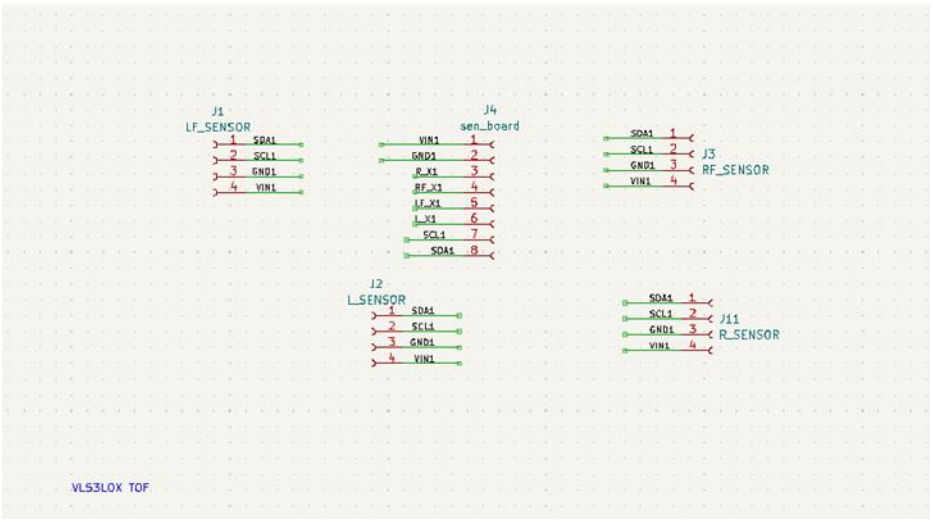


Figure 9-5: TOF Sensors

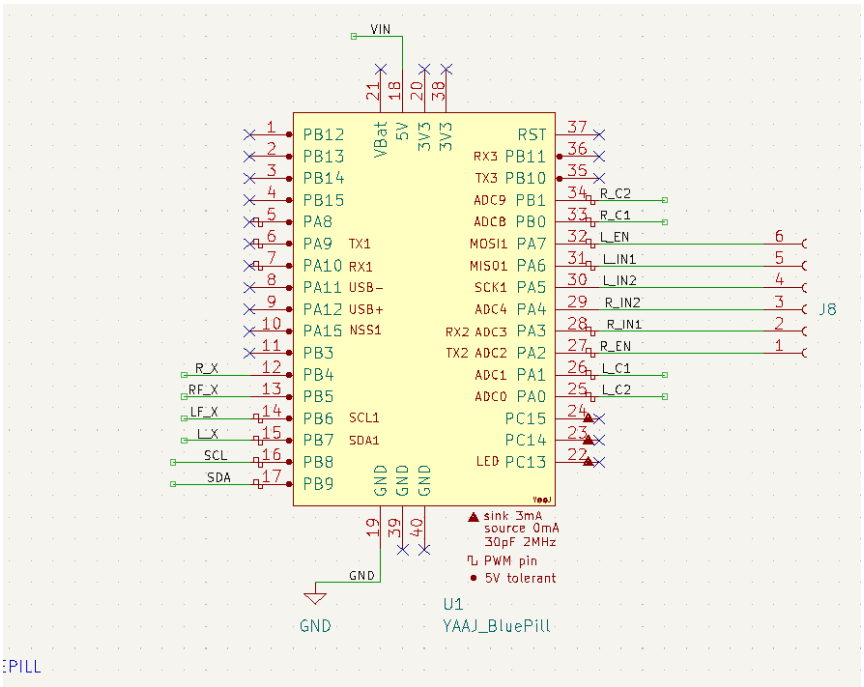


Figure 9-6: STM32 Bluepill

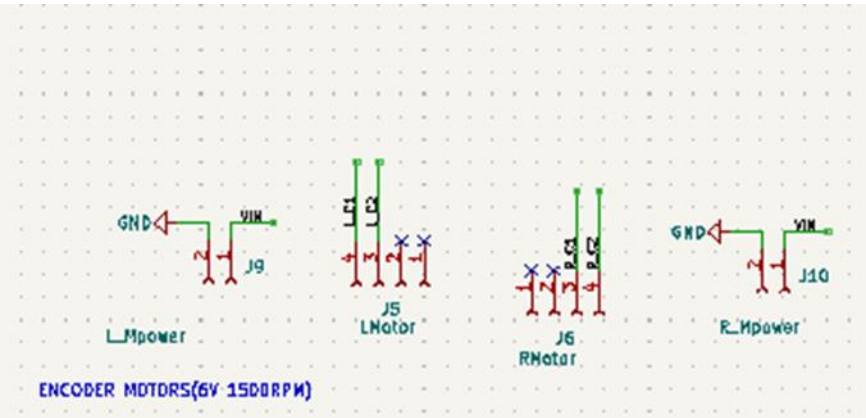


Figure 9-7: Motor Connections

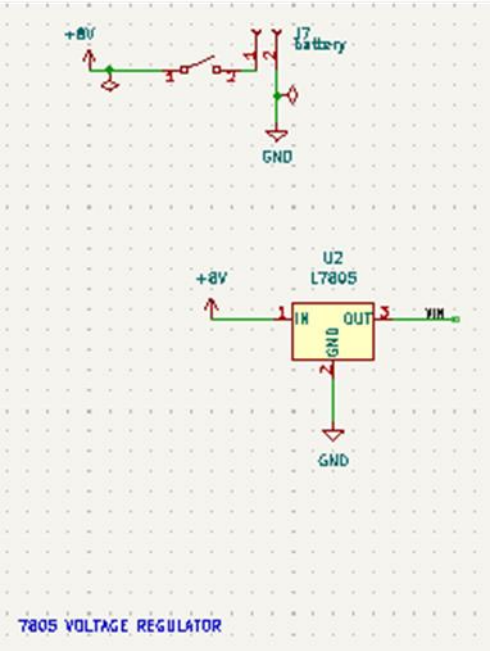


Figure 9-8: Power Source

9.7. Appendix G: PCB

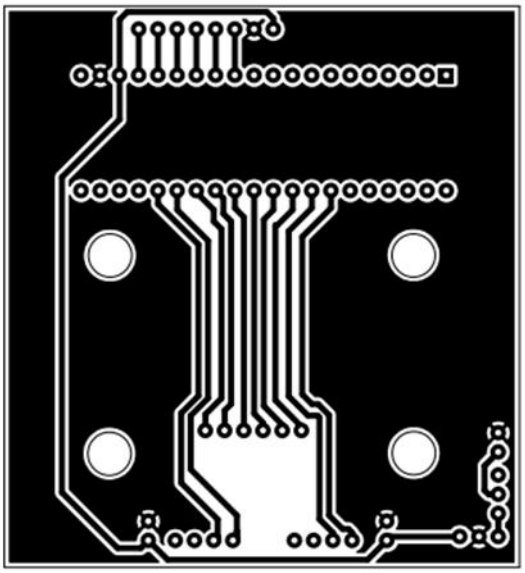


Figure 9-9: Main Board

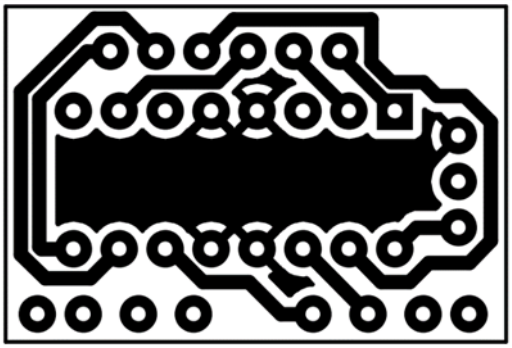


Figure 9-10: L293D IC

s

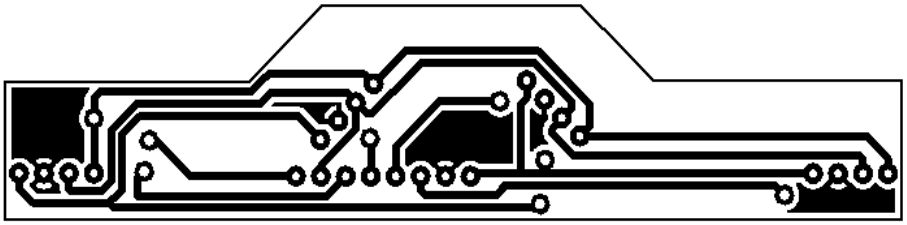


Figure 9-11: Sensor Module

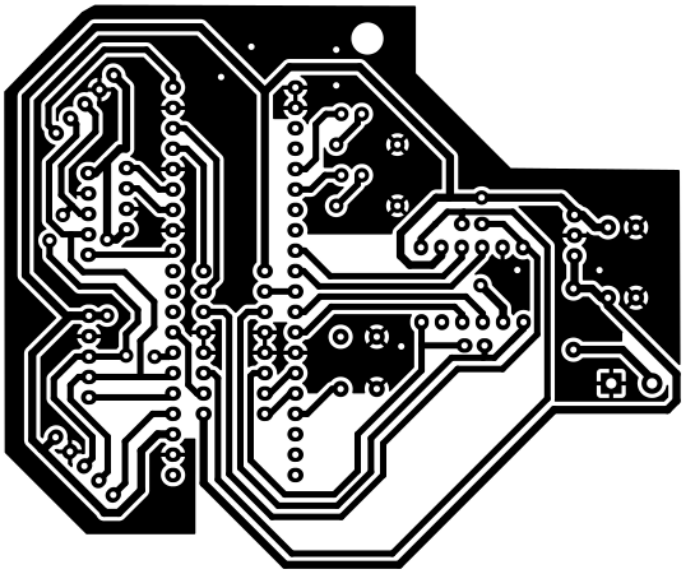


Figure 9-12: Optional Board

10. References

[1] P. Harrison, “Micromouse Online - Everything for Micromouse and Line Follower Robots,” *Micromouse Online*, 2022. https://micromouseonline.com/ (accessed Mar. 05, 2023).

[2] P. Harrison, “More suck, less slip - Micromouse Online,” *Micromouse Online*, 2018. https://micromouseonline.com/2018/02/18/more-suck-less-slip/ (accessed Mar. 05, 2023).

[3] J. H. Su, C. S. Lee, H. H. Huang, and J. Y. Huang, “A Micromouse Kit for Teaching Autonomous Mobile Robots,” *http://dx.doi.org/10.7227/IJEEE.48.2.6*, vol. 48, no. 2, pp. 188–201, Oct. 2012, doi: 10.7227/IJEEE.48.2.6.

[4] Z. Haoming, W. Yinghai, and P. L. Soon, “Auto-navigation of micromouse based on infrared sensor,” *Sensors and Transducers*, vol. 170, no. 5, pp. 67–72, 2014.

[5] H. M. Zhang, L. S. Peh, and Y. H. Wang, “Two Axes Micromouse PWM Generator Based on FPGA,” *Appl. Mech. Mater.*, vol. 496–500, pp. 1674–1680, 2014, doi: 10.4028/WWW.SCIENTIFIC.NET/AMM.496-500.1674.

[6] Z. Haoming, P. E. H. Lian Soon, and W. Yinghai, “Turnings and its calibration of micromouse based on ARM9 and FPGA,” *Sensors and Transducers*, vol. 168, no. 4, pp. 120–126, 2014.

[7] S. G. Kibler, A. E. Hauer, D. S. Giessel, C. S. Malveaux, and D. Raskovic, “IEEE Micromouse for mechatronics research and education,” *2011 IEEE Int. Conf. Mechatronics*, pp. 887–892, 2011, doi: 10.1109/ICMECH.2011.5971240.

[8] H. M. Zhang, L. S. Peh, and Y. H. Wang, “Software Design of Micromouse Motion Control,” *Appl. Mech. Mater.*, vol. 602–605, pp. 989–992, 2014, doi: 10.4028/WWW.SCIENTIFIC.NET/AMM.602-605.989.

[9] G. Law, “Quantitative Comparison of Flood Fill and Modified Flood Fill Algorithms,” *Int. J. Comput. Theory Eng.*, pp. 503–508, 2013, doi: 10.7763/IJCTE.2013.V5.738.