Senior Design Final Report

Kieran Cosden, Samantha Sebastian, Kuwunda Shamambo, Ethan Stone

University of Notre Dame

EE 41440 Senior Design II

Professor Mike Schafer

05/06/2026

Table of Contents

1 Introduction

For the first time in Notre Dame's history, the electrical engineering seniors took part in a new Senior Design project: designing and constructing a micromouse! The Micromouse Competition, first announced in 1977 by IEEE Spectrum and then held in New York City in 1979, challenges teams to design and build a fully autonomous robot capable of independently navigating and solving a 16x16 maze without any human intervention. Each team is given a ten minute exploration phase, during which the robot traverses the maze, builds an internal map, and identifies the fastest route to the center. The robot then either must travel back from the center to its starting point or be brought back by the competitors to be reset for racing. The micromouse must then travel back to the center as quickly as possible.

In our year-long Senior Design Course, we tested out skills through both hardware and software design and implementation. During the fall semester, we researched the system and components for the micromouse under the requirement of using an ESP32 chip with an Arduino framework for the code. In tandem with class lectures about both hardware and software fundamentals, we took the time to narrow down our choices for the sensors and motors to be tested in the spring semester. Then using KiCad we designed a custom PCB for the components to interface with, and through small tweaks and changes based on design feedback we printed the first and only iteration of our board by early-March. Then throughout the last few weeks of the semester we troubleshooted our main board and also worked out the logic behind our code.
In this report, we will dive deep into our solution to this challenge, first talking about the system requirements and how we went about the design process in Section 2 and Section 3 respectively. Then Section 4 we explain the results of our testing and troubleshooting with both the hardware and software. Finally, Section 5 covers the results of our design and construction and ways we would improve it in the future.

2 Key System Requirements

2.1 Competition Requirements

The Micromouse Competition requires that the micromouse is a fully autonomous robot that can reach the center of a maze in the shortest possible time. The maze is built up by a 16 x 16 grid of square cells, each measuring 18cm x 18cm. The walls are white/red (high reflectivity) with a height of 5cm with a width of 1.2cm while thFe floors are black for low reflectivity. With the ending in the center, the micromouse would start on one of the four corners and traverse the maze without prior knowledge of the maze layout, including external control, communication, or preloaded maps. The navigation part of the competition is limited to ten minutes, and once either the time is done or the micromouse must then run the maze as quickly and optimally as possible. Also, the micromouse must remain within a 25cm x 25cm footprint and operate without damaging the maze or learning any components behind.

2.2 Component Requirements

Alongside the competition requirements, we based our component selection and board design on the following requirements:

<u>Sensor</u>

- Polling Update Rate: We required the sensors to have a maximum sampling time of 250ms per reading. Since the micromouse would spend around 1 second traversing a single grid space, all three sensors must be polled fast enough. With this, we also require our software to synchronize the motor speed with the chosen polling rate so it doesn't outpace the sensors.
- Millimeter-Level Accuracy & Calibration: We required the sensors to be able to detect at least 168mm (or 6.6 inches) while also having very similar calibrations to ensure it can have accurate readings. The maze runs on a 180mm square grid with 12mm walls, so the sensors should be accurate enough to where it is exactly in the square.
- Signal Processing & Environment: With the white walls and black floors, the sensors must be able to filter out corner reflections/noise and operate reliably against the reflective white walls in ambient roomlighting.

Locomotion & Power

- Torque: We required a mid-range gear ratio to provide sufficient torque to move the chassis, especially with the added weight of the batteries and potentially a third wheel, without sacrificing high-end linear speed. Also, with a 38.5mm wheel and through calculations, the gear ratio should provide enough force to start the robot within a "reasonable time."
- Voltage limits: Through our investigation and decision to use the DRV8833 H-bridge, the system must not exceed 10.8V due to its operating limits. The Pololu motors we were looking into are intended for 12V, and to combat the inductive "kickback" of the motors the driver must utilize Slow Decay Mode to maintain smoother current flow and improve control at lower speeds.
- Motor-Driver and Encoder Pairing: We required an H-bridge that can allow for independent dual-motor control, especially one that supports motor asymmetry since they don't spin the exact same or may require slightly different PWMs. Also, the encoder should provide feedback necessary for complex maneuvers both for linear and rotational precision.

3 Detailed Project Description

3.1 Project Timeline & Goals

The bulk of our construction, testing, and troubleshooting of both our hardware and software took place during the spring semester. Our main goal before spring break (mid-March) or earlier was to prepare our PCB design for printing, finalize and order the required system components, and develop the code for the mouseboard's movement to be in sync with the motor encoders and distance sensors. After that, we aimed to develop and implement a Flood Fill algorithm for our mouseboard's maze navigation by mid-April, leaving us a few weeks before the Race Day on May 1st to polish up the code and board as needed. Below are the goals for each part of the project we made:

Hardware Development

- PCB Iteration: Redesign the board based on design review feedback from the fall semester. Then print and test the board with the current hardware we have on hand, reserving budget for a potential second reprint or component upgrades based on testing results.

Core Movement & Path Finding

- Locomotion Control: Development movement functions that integrate wheel encoders and sensors to navigate the specific physical dimensions of the maze safely. Also, decide which sensor to change to from ultrasonic distance sensors.
- Algorithm Integration: Research past micromouse algorithms, test potential algorithms via software emulation, and implement the best pathfinding logic into the board.

Programming & Processing Requirements

- Timing and Synchronization: Calibrate motor speeds to align closely with sensor polling rates and processing delays to prevent data latency.
- Sensor Logic & Thresholds: Define strict distance ranges (e.g. open path, off-center, dead-end) to trigger the correct movement state. Use the dominant sensor signal to dictate immediate navigational choices.
- Dynamic Mapping & Memory: Build a robust data structure to map the maze in real-time. The system also must continuously log explored pathways, node connections, and track absolute distance traveled from reference points.

- Spatial Tracking & Auto-Correction: Continuously calculate the robot's exact position within a cell, implementing real-time closed-loop corrections to center the robot while dynamically adjusting safe linear and angular speeds to ensure precise, collision-free maneuvers and turns.

3.2 Hardware Design & Implementation

Hardware Overview

Our main board hardware system is built around a custom four-layer PCB, designed with dedicated power and ground planes on the inner layers to allow for clean signal paths, and two outer layers dedicated to signal routing. The board supports two input power sources, either 7.4V supplied from two 3.7V LIPO batteries, or 5V supplied from the USB-C connection. A main power switch allows the user to switch between these two inputs, feeding our 3.3V voltage regulator that supplies our ESP32 and other onboard logic. The 7.4V battery rail is routed directly to our motor driver, allowing the motors to operate at full battery voltage. Along with battery voltage, our dual H-bridge motor driver receives pwm and direction controls from the ESP32 module, then drives our two motors. Each of our motors has a built in encoder, which is then routed back to the ESP32 for speed monitoring. Our ESP32 is programmable through a series of strapping pins, including push-buttons connected to our Enable and Boot pins.

In addition to our main PCB, three custom two layer Time-of-Flight (ToF) sensor breakout boards were designed and built. Rather than integrating the ToF sensors directly on the board, our design focuses on flexible placement of our sensors, allowing us to easily replace our sensors and avoid using additional long signal traces across our main board. Each breakout board receives 3.3V supply from the main board, and performs onboard voltage conversion down to 2.8V required for VL53 sensor operation. This means that each sensor breakout board is self-contained with respect to its power requirements, and the main board only needs to supply a single 3.3V rail to each sensor. Communication between the sensor boards and the ESP32 are handled over the two dedicated I2C lines, with each sensor's XSHUT pin connected back to a dedicated GPIO pin, allowing for individual sensor control and address assignment on setup.

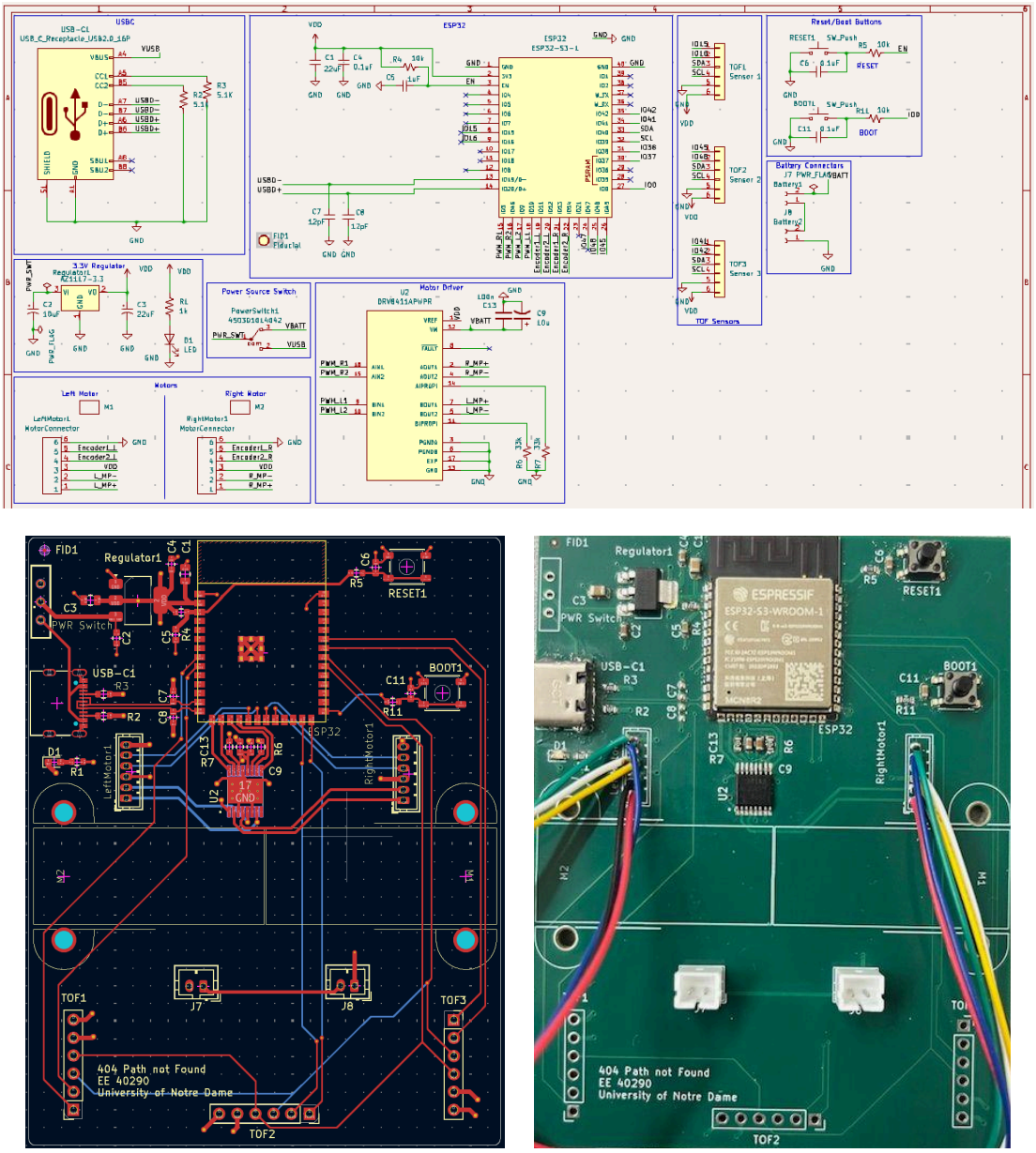

Figure 1. Board 1 Schematic, PCB Layout, and Picture

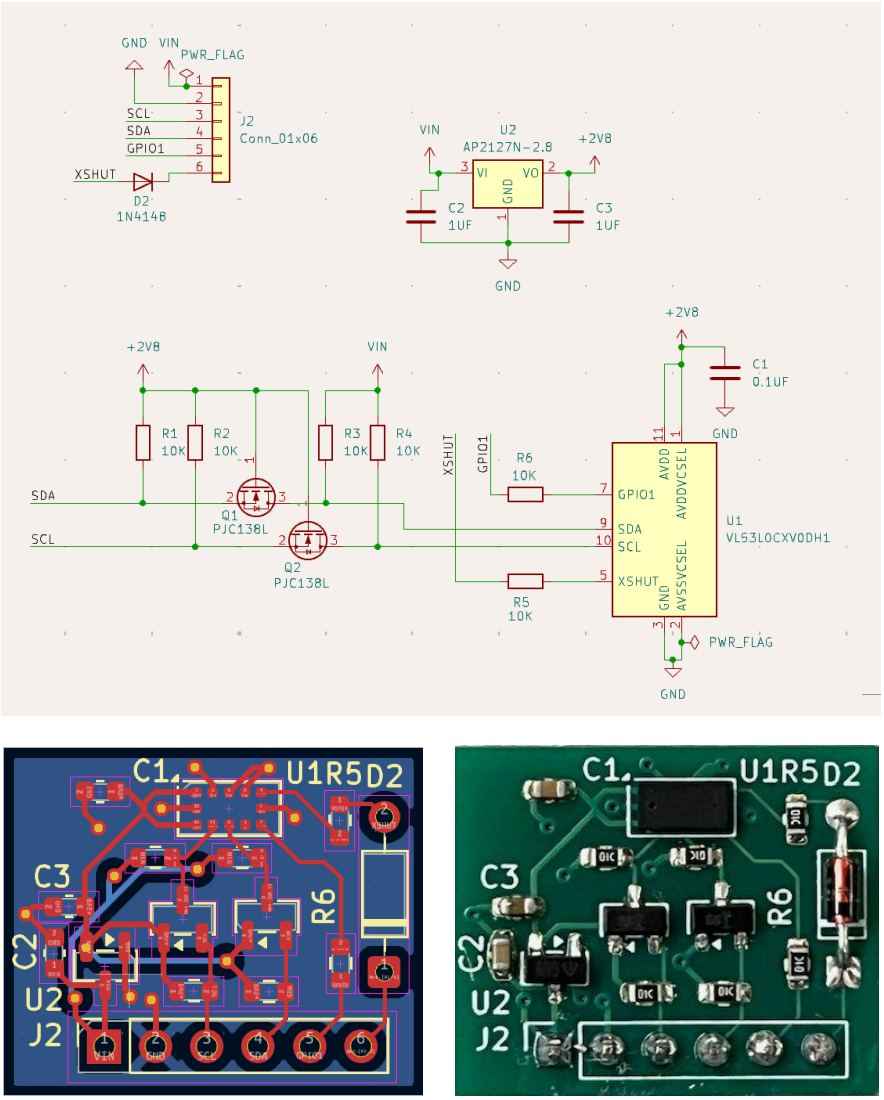

Figure 2. Time-of-Flight VL530X Board Schematic, PCB Layout, and Picture

Main Board Component Selection

Microcontroller - ESP32-S3-WROOM-1

The ESP32-S3-WROOM-1 was selected as the main microcontroller for our system. The module integrates a dual core processor, providing sufficient computational power for simultaneous real-time tasks, including sensor polling, motor control, and maze mapping. A key driver in this selection was the module's Bluetooth and wifi capabilities. The ESP32-S3 also provided an ample number of GPIO pins, allowing us to interface with dual motor encoder lines, sensor channels, PWM motor controls, and XSHUT sensor initializations all on a single compact module.

Motor Driver - DRV8411APWPR

The DRV8411APWPR from Texas Instruments was selected as the motor driver for our system. The device utilizes two independent H-bridge channels in a single compact 16-pin HTSSOP package, allowing both motors to be driven from one IC and reducing total board area compared to using two separate dedicated single-channel drivers. The driver supports a wide power supply range of 1.65V to 11V, making it fully compatible with the 7.4V battery rail used in our design, and accepts standard PWM and directional inputs directly from the ESP32 GPIO pins. This device accepts logic input levels of 1.8V to 5V, making it compatible with our Vdd signal of 3.3V. The DRV8411APWPR is capable of 4A peak output current per channel, which comfortably exceeds the stall current of our DC micro metal motors.

Vdd Voltage Regulator - AZ117C-3.3

The AZ1117C-3.3 LDO linear regulator was selected in order to generate the 3.3V logic supply voltage from either the 7.V battery or 5V USB input. The device supports a voltage differential of up to 10V, meaning both our battery and USB sources lie comfortably within its operating range. The regulator has a typical dropout voltage of 1.2V at 0.8A output, which ensures the regulator will maintain a steady 3.3V output even as the LiPo battery cells discharge. The device has a current limit of 1A minimum, which is sufficient to supply the ESP32 and all onboard logic under full load.

Motors - Pololu Micro Metal Gearmotor, 50:1, 6V

Our drive system uses two Pololu Micro metal gearmotors with a 50:1 gear ratio, rated at 6V. The gear ratio was selected as a balance between output torque and rotational speed. At the rated 6V, these motors comfortably function with our 7.4V battery voltage election. Each motor includes a quadrature encoder, providing feedback for closed-loop speed control and monitoring. Encoder counts are read directly by the dedicated GPIO pins on the ESP, and require 3.3V logic operation, further aligning with our previous design choice.

USB Connector - USB4105-GF

The USB4105-GF USB-C connector was selected for programming, debugging and bench power input. The connector supplies the 5V required for operation when the batteries are disconnected. Additionally, the connector routes D+/D- differential signal lines directly to the ESP32 for native USb communication.

Push Buttons - PTS64 Tactile

Two PTS64 tactile buttons are included on the board for bootloader control. One button connected to the EN (reset) pin and one to the BOOT (GPIO0) pin. This allows the user to manually enter the ESP32 download mode by holding the boot button while toggling enable.

Breakout Board Component Selection

ToF Sensor - VL53LOCXV0DH/1, ST Microelectronics

The VL53LOCXV0DH/1 time-of-flight sensor from ST microelectronics was selected for wall detection. The device uses an infrared VCSEL (Vertical Cavity Surface Emitting Laser) to measure absolute distance by timing the flight of emitted pulses, providing extremely accurate wall distance measurements. The sensor supports sensing distance of up to 2 meters, which far exceeds the maximum wall distance in the 16x16 Micromouse maze where each cell is 180mm wide. Communication to the ESP is handled over I2C at up to 400kHz, allowing the ESP to poll all three sensors rapidly within the control loop. Each sensor is powered at 2.8V, supplied by an onboard AP2127N-2.8 regulator on each board. Since all three sensors share the same default I2C address, the XSHUT pin on each sensor is connected back to a dedicated GPIO on the ESP32, allowing the sensors to initialize sequentially and receive individual address assignments.

2.8V Regulator - AP2127N-2.8

The AP2127N-2.8 low-fropout regulator was selected to provide the 2.8V supply required by the VL53 ToF sensors. The device accepts input voltage between 2.5V and 6V, making it fully compatible with out 3.3V supply rail provided by the main board. The most critical selection criteria for this regulator was its dropout voltage, since there is only .5V of headroom between the 3.3V input and 2.8V output. The AP2127N-2.8 specifies a maximum dropout of 300mV and a typical value of 170mV, providing a sufficient margin to maintain a stable 2.8V even at operating extremes.

Level Shifting MOSFETS - BSS138-G

The BSS138-G N-channel MOSFETs are used on each ToF board to perform bidirectional I2C level shifting between the 3.3V logic of the main board and the 2.8V operating voltage of the sensor itself. The I2C bus is an open-drain interface, and a pair of MOSFETs are required to shift logic levels in both directions without requiring a dedicated level-shifting IC.

3.3 Software Design & Implementation

As shown in Figure 5 below, the system operates on a continuous, closed-loop cycle that allows the micromouse to read its physical environment, updates its internal digital map, recalculates the shortest path, and executes motion control for it to move to the next cell. The software is broken up into two parts which are explained below.

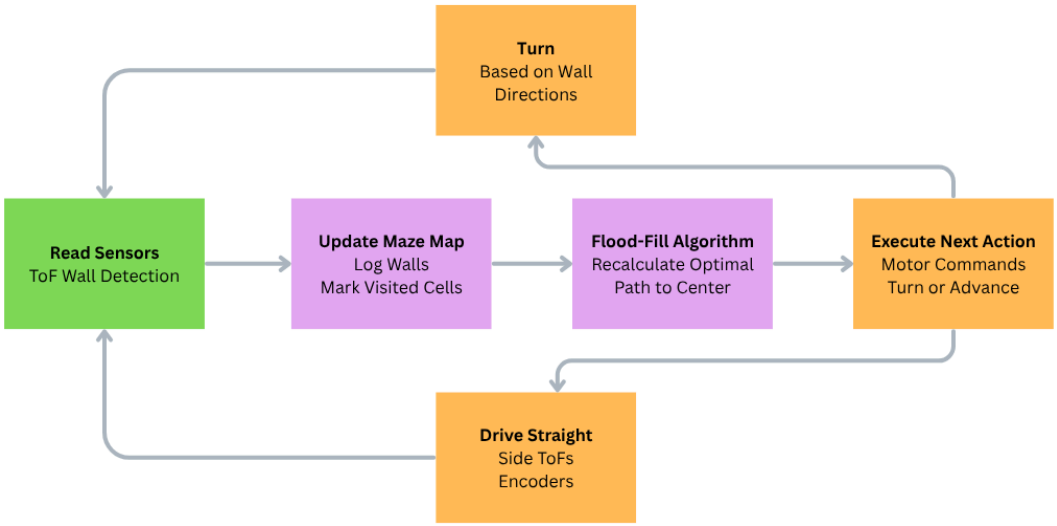

Figure 3. Control Loop Block Diagram

Low-Level Abstraction Layers

This part of the software bridges the gap between the ESP32 microcontroller and the physical hardware (e.g. sensors, motors) to ensure that the data inputs and outputs work together.

Sensor Interfaces

The micromouse perceives its environment through asynchronous, non-blocking polling of the VL53L0X ToF sensors, two of which are mounted on the right and left and one in the front/front-left of the board. As shown in the Read Sensors block in Figure 5, the interface systematically requests millimeter-accurates distance readings from the front, left, and right walls. To maintain system stability, the sensor interface includes a robust I2C recovery mechanism that automatically resets the data bus during a hardware lockup, ensuring uninterrupted wall detection.

Control Algorithm

An encoder-driven, closed-loop control system determines how the micromouse's locomotion functions. When the robot enters the Drive Straight state, it continuously cross-references the left and right ToF sensor readings, and a Proportional-Derivative (PD) control algorithm applies real-time micro-corrections to the motor PWM signals which adjusts the wheel speeds to center the mouseboard in a corridor. This ensures that the mouseboard won't drift and also scrap any walls.

High-Level Navigational Logic

To process the raw data from the low-level side, this layer acts as the "brain" of the micromouse, allowing for it to gain spatial awareness and optimize a path for solving the maze.

Motion Control

The mouseboard's movement is turned into discrete, cell-by-cell operations that run under a state machine, tracking if it is either stopped, moving forward, or turning. As shown in the Execute Next Action block, the system translates algorithmic decisions into exact physical maneuvers using the wheel encoders to determine how exactly it should move from one cell to the next.

Maze Navigation

Alongside motion control, this level covers synchronization between where the micromouse physically is in the maze and also the digitally mapped coordinate maze that is kept in the Flood-Fill Algorithm block. When the mouseboard is in a new cell and stops, it executes the Update Maze Map phase, logging in newly discovered walls in the 16x16 grid memory and updating where it is currently at.

Maze Search Algorithm

The Flood-Fill algorithm we use simulates water flowing from the target center back to the robot's current position, assigning a distance value to every accessible cell based on the known wall layout. Then, it evaluates the adjacent open cells and selects the node with the lowest flood value, dictating the optimal direction for the robot's next move. The 16x16 dimensions of the maze are hard-coded into the algorithm, as well as the location of the 2x2 target in the center of the maze. When powered on, the mouse expects to be facing north in the southwest corner of the maze. This way, the mouse will never make its first turn left. This algorithm stores the explored

maze as an array, which can be visualized by printing an ASCII representation of the maze through the serial monitor. This was a valuable method of debugging the program during testing.

Breakdown of the Project File Structure

The following part of this report contains a summary of each project file that executes the different parts of the Control Loop Block Diagram in Figure 5. We would like to note that we did not end up using the MouseWifi files for the last few weeks of debugging and racing, but we included it here since we originally planned to implement it into our design

main.cpp

Serves as the central point of all the files listed below, combining the sensor data acquisition, motor control, state management, and Flood Fill algorithm. Through the logic continuously executed in loop(), the mouse is able to exhibit high-level behavior.

Mouse.h

Encompasses the global robot state! It doesn't contain any executable logic since it establishes the fundamental data structures, enumerations, and global variables that are used across various other files to ensure that there are no re-initializations or inaccessible variables.

Direction.h

Similar to Mouse.h, it is shared amongst several files as well including Mouse.h. However, it only contains a strictly typed, scoped enumeration that establishes the four cardinal directions to ensure proper directional logic throughout the mouseboard's maze navigation.

FloodFill.cpp and FloodFill.h

Implements the core pathfinding and navigation logic for the micromouse to track where it is currently at in the maze while also mapping out the maze. It utilizes a Flood Fill algorithm that determines the shortest path to the center of the grid. The maze class maintains a 2D representation of the maze with coordinates (x, y), tracks the robot's current state, and recalculates the optimal route as new walls are discovered.

MotorsAndEncoders.cpp and MotorsAndEncoders.h

Bridges the gap between high-level navigation with the Flood Fill algorithm and low-level movements done through the control algorithm. It utilizes the ESP32's hardware PWM (ledc API) to drive the motors and hardware interrupts to read wheel encoders, ensuring that the robot moves precise distances, turns accurately, and centers itself within the maze corridors.

SensorAndDriving.cpp and SensorAndDriving.h

Manages the initializes and continuous data acquisition for the Time-of-Flight distance sensors, utilizing the Adafruit_VL53L0X library to access the VL53L0X modules. Also, it handles the I2C bus communication required to run multiple sensors simultaneously. The Time-of-Flight distance sensors were able to be accurately read so that the micromouse can have real-time spatial awareness of the maze walls.

MouseWifi.cpp and MouseWifi.h

Provides wireless communication for the ESP32-based board to provide easy debugging and programming. Because it connects the microcontroller to a local WiFi network, two remote capabilities are enabled: Over-The-Air (OTA) firmware updates and live wireless telemetry via a Telnet server.

4 System Testing

In order to determine how well selected components and chosen software design fit for our key system requirements, we conducted a number of tests throughout the semester. System testing began in the fall as a way to help us choose among our top component options which would be best to move forward with for our draft of our board design. For the motors we ended up using for the final board and the ultrasonic distance sensor we did not end up using for the final board, we developed code to test to observe and compare the real performance of our component choices.

In our test code for the motors, we counted the number of pulses per revolution and set the PWM for both motors. The PWM was initially set to 100, which was not enough to start both motors, for both motors but through the serial monitor we are able to change the values. After testing various values for the PWM starting from 100 and increasing it by 1 or 5, we determined that both motors have a starting PWM of 150. Through our testing approach, we verified that the previous measurement makes sense based on the gear ratio of your motor and the counts/revolution of your encoder. The speed (rpm) of each motor was running for the same PWM signal, so we tested performance for the PWM values of 150, 180, and 210. The results of testing were as follows:

PWM Value 1: 150

Right motor speed : 52.5 RPM

Left motor: 70 RPM

PWM Value 2: 180 RPM

Right motor: 99 RPM

Left motor: 114 RPM

PWM Value 3: 210

Right motor: 153 RPM

Left motor: 160 RPM

We tested all four of the ultrasonic distance sensors on the mouse board, using the digital SDL and SCA sensor pins and switching the solder jumpers to digital. Unfortunately, the sensors did not support I2C, and each needs two GPIO pins, so we were not able to test all of the sensors simultaneously on the mouse board. We verified accurate distance readings for each sensor by checking serial output with the wall at various distances, both for cases where the sensor was moving and where the sensor was still. We found that the sensors were able to read accurately up to 20cm away and were easily able to handle 5 ms increments of change between measurements.

In the spring, after we made changes to our design by using time of flight sensors instead of ultrasonic distance sensors, we tested and verified the performance of the new sensors. We bought ToF sensors that we tested in a way very similar to how the ultrasonic sensors were tested and verified. Though the sensors were not as accurate as we had expected, based on specifications on the data sheet, we found that the sensors met the standards that we had based on system requirements. After overcoming difficulties in using multiple ToF sensors, which persisted largely due to a hardware issue with our PCB that was eventually replaced, we printed the breakout board PCBs we designed for ToFs and assembled our own. With individual sensor test code, we checked the performance of our assembled ToF boards and sensors, finding that we got better uncalibrated measurement accuracy than with the ToF boards we had bought with sensors already attached. Once we were able to successfully run our code with several ToF sensors, we tested the precision of the distance measurements of each by setting the mouse still and collecting readings of each over time to see the range and average of the readings of each for given distances. The side sensors were tested slightly differently from the front sensors, with each the same distance–down to 0.5 mm–from a wall on either side of the practice mouse, so that they would be calibrated to get readings that were not only as accurate as possible for short and long distances, but also readings that were precise (consistently proximal in value). Especially after calibrating each sensor according to the offsets they presented through the data we gathered from several trials, the ToFs that we put together were rather accurate and functioned well.

As for the software side of things, we developed our code in steps in order to verify functionality at each stage and make sure we had minimum reliable function in areas deemed most important before moving forward and adding the more complex aspects of our software design. We developed the basic driving/navigation capabilities of the micromouse primarily, ensuring that it would be able to drive in paths as straight as possible and making proper 90-degree and 180-degree turns by running our functional navigation code and iterating the gain and PWM parameters. After numerous iterations, we determined the best values. Over time, however, as our code changed and the demands on the system grew, we continually iterated these parameters in order to keep the mouse's movement as consistent and accurate as possible. Once PID control was implemented, there was much more iterating that took place, especially since the mouse's performance was not wholly consistent for given parameters from one day to the next or from one trial to another. For the floodfill algorithm implementation and maze solving capabilities, we worked similarly, building slowly by changing the logic for case switching and determining which path options to prioritize at every junction the mouse encountered. Additionally, we used the serial monitor in order to visualize how much of the 16 x 16 cell maze the mouse had discovered and where it understood the paths it had traveled to be. An example of the visualization printed in the serial monitor is displayed below, in Figure 4. This assisted us in understanding how well the software we implemented enabled the mouse to determine its coordinates within the grid and determine which cell it was in, which allowed it to determine when and where it should choose to take one path or another at the next junction.

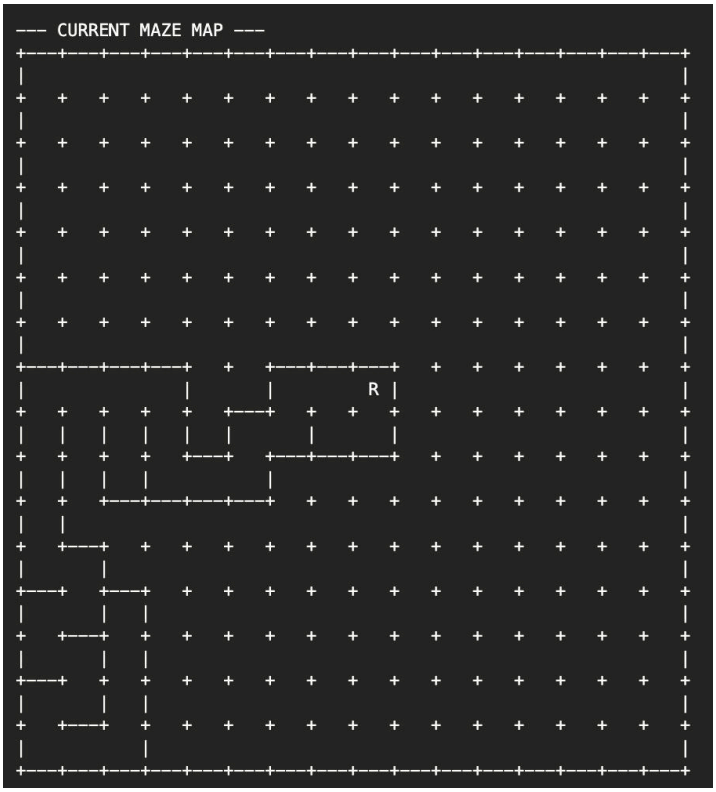

Figure 4. Serial read-out from successfully solving the maze during testing

Note, the maze in Figure 4 was a path without T-intersections, so it provided a false positive.

5 Conclusions

5.1 Finalized Micromouse

Since we couldn't utilize our constructed PCB, we went ahead and utilized the practice board provided to us at the start of the semester. Each of our customized ToF sensors were mounted on the sides and front left, and we used the same batteries, wheels, and battery holder from the class. This is due to the fact that we spent the last few weeks before the Race Day to reconstruct our main board, hoping that it will work as soon as possible. However, with less than two weeks left of the project we opted to focus solely on getting the software portion to work with the practice board to at least have something for race day. We would like to note that in Figure 5 the right ToF is not shown but it was used for the Race Day.

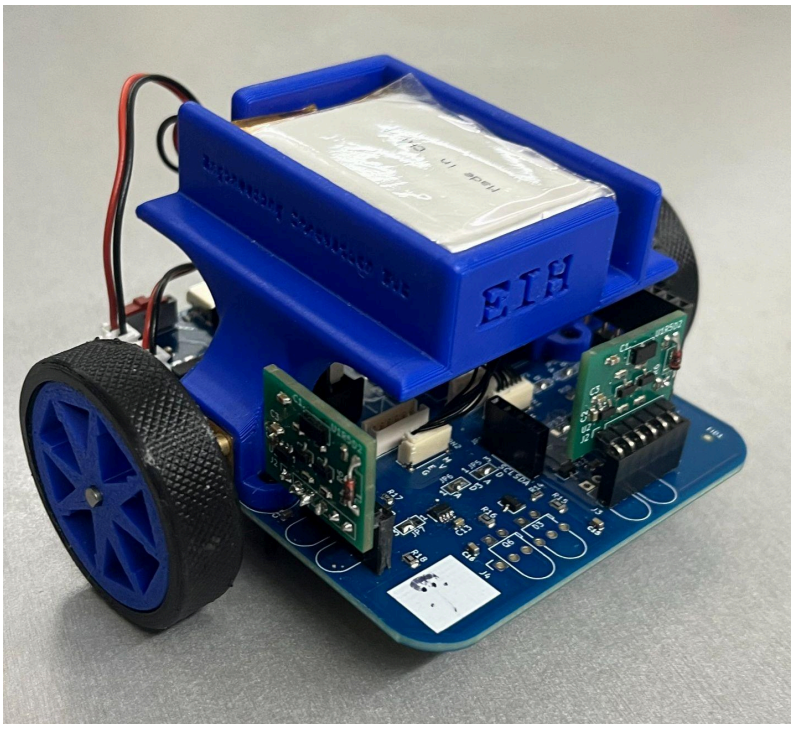

Figure 5. Final Micromouse for Race Day

5.2 Race Day Results

While our micromouse was successful in solving the maze during test runs, it unfortunately did not reach the center during the Race Day. During the ten minute exploration phase, it traveled to 17 cells before struggling to get past the first T-intersection. We believe that since our front ToF

sensor was more on the left front of the practice board and there was a lack of side walls once it encountered the T-intersection, the micromouse wouldn't be able to properly drive straight and stop in the middle of the cell before turning. Instead, the lack of side walls forced the robot to only use the readings from the front left sensor to determine where it should go, and as a result it would turn too early. With a set of two front facing sensors, we believe that the mouse would have been able to navigate t-intersections and explore much more of the maze. The flood fill exploration algorithm was well-executed and functioned properly, so the point of failure was the ability of the mouse to orient itself without full ability to orient itself with only three distance sensors.

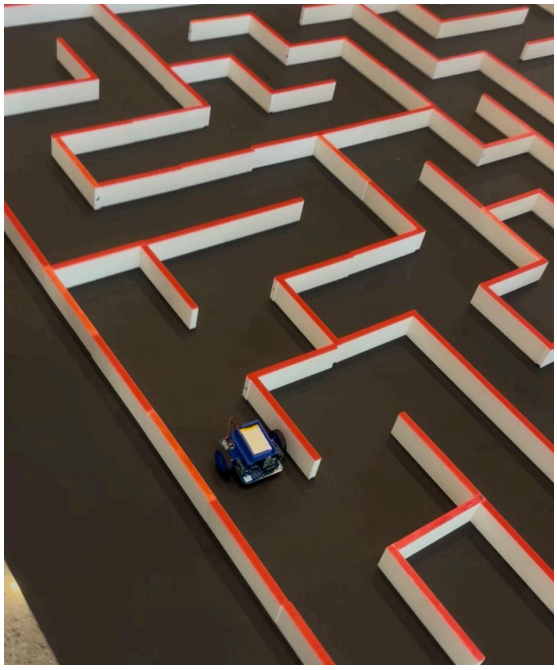

Figure 6. T-Intersection from Race Day

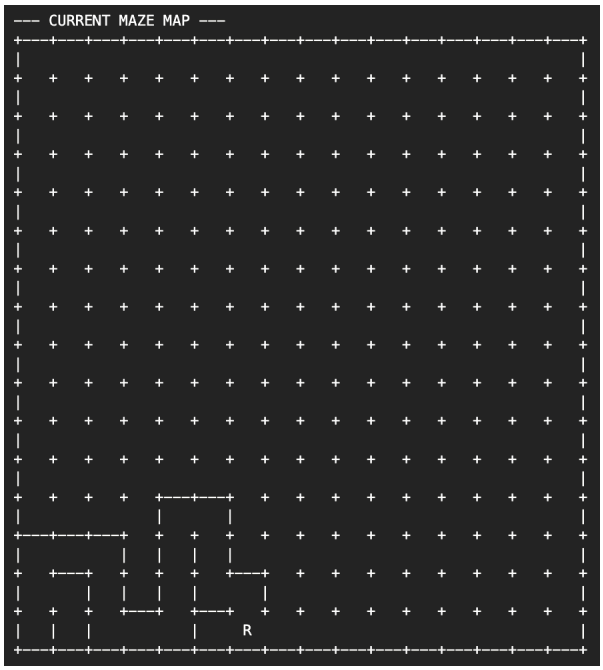

Figure 7. Serial read-out of the maze

5.3 Future Improvements

Hardware Improvements

While the custom PCB was successfully fabricated and constructed, two main hardware issues prevented the main board from functioning as intended during the competition. The first was related to the strapping pins, EN and BOOT, which control the bootloader entry sequence required to begin USB enumeration. The pins were not sufficiently pulled up to Vdd, causing the board to fail, and making USB enumeration impossible. This was temporarily resolved by adding external pull-up resistors, however a more permanent fix would require a board redesign involving properly sized pull-up resistors on both the EN and BOOT lines. The second issue with the board was USB signal integrity. The USB 2.0 requires tightly controlled differential impedance on the D+/D- lines, and the combination of manual bodge wire and trace stubs introduced during strapping pin debugging created significant signal reflections and parasitic capacitance, making USB communication impossible.

Beyond the issues encountered in this revision of the board, several additional hardware improvements have been identified for future board iterations. The current design places a 2.8V regulator on each of the three ToF breakout boards, which adds to total component count. A cleaner solution would be to utilize a single 2.8V regulator directly onto the main board and route the 2.8V supply to each sensor, simplifying the breakout board design significantly. Additionally, a future consideration would be to replace the side ToF sensors with IR sensors. While functional, the VL53L0X sensors operate less reliably at short ranges encountered when closely following side walls. Replacing these side ToF sensors with dedicated short-range IR emitter/receiver pairs would improve accuracy and responsiveness during straight-line driving and centering. The front facing sensor configuration could also be improved by adding an additional forward-facing ToF sensor, allowing for triangulated distance measurements. Finally, the current motor connections on the main board use through hole connections, which are mechanically fragile and prone to failure. Dedicated JST motor connections in a future revision would improve the reliability and simplify the construction process.

Software Improvements

Several software improvements have been identified following our race day experience with the current system. The most obvious issue during race day was the robot’s handling of T-intersections, where the maze presents an open path to both the right and the left of the mouse.

When approaching said intersection, the mouse had a difficult time recognizing both side openings before the centering function fired, causing the mouse to rapidly turn to one side before any flood-fill logic had occurred. A future software revision would implement more explicit T-intersections detection logic, potentially having a fine-tuned motor response for when this exact type of wall configuration is detected.

A second planned software improvement involves utilizing bluetooth communication. The MouseWifi.cpp and MouseWifi.h modules were created in order to handle serial monitoring and code upload wirelessly; however, these modules required connection to the SDNet wifi network, which was not accessible in the actual competition day environment. By developing bluetooth low energy code for this robot, real-time communication during trial maze runs would have been possible, allowing for more accurate testing and bug diagnosis.