EE 40290 SENIOR DESIGN II:
Soldering Irons

FINAL REPORT

By: Emily Govea, Kristen Kelly, Hunter Koch, Jacob Riem

SENIOR DESIGN SPRING 2026

Submitted May 6, 2026

ND Department of Electrical Engineering
Notre Dame, IN 46556

Contents

1 Introduction

The Micromouse Competition is a long-running IEEE robotics challenge involving autonomous vehicles navigating an unknown 16x16 maze. The objective is for the “mouse” to navigate the maze during the 10-minute race duration and record the fastest completion time from start to finish.

Over the course of two semesters, we have collaborated as a team to work through the design process from start to finish. For autonomous operation, the system must rely on internal sensors, mechanical components, and onboard processing. Achieving such a system required design efforts in compact hardware, control algorithms, and navigation algorithms, which will be discussed throughout this final report.

2 Key System Requirements

2.1 Explicit competition requirements

2.1.1 Maze size and features

The competition maze consists of a 16x16 unit grid composed of 18x18cm squares. The walls are 1.2cm thick, 5 cm high, and white on the faces. There is a wall at each lattice point of the maze grid, and there will be more than one possible route to reach the 2x2 unit finish square located at the center of the maze from the designated starting corner. There is only one entrance to the 2x2 finish square. The floor of the maze will be black with a non-gloss finish, and the tops of the walls will be red.

2.1.2 Competition run time

Each mouse has a total of 10 minutes in the maze. There is no explicit requirement detailing how much time must be allotted to specific tasks. The 10-minute period starts once the mouse crosses the open grid line out of the starting corner unit. Possible tasks the mouse could complete, in any order and duration, include maze exploration, mapping, and completing “runs.” A “run” is a complete journey from the designated starting corner cell to entering the center 2x2 finish square. The official time recorded for a “run” starts the moment the front edge of the mouse exits the start cell to the moment the front edge of the mouse crosses into the finish cell. An unlimited number of “runs” could be made by the mouse within the allotted 10-minute time. Between consecutive “runs,” the mouse must return to the starting corner autonomously. A mouse that is picked up by a team member and placed back in the starting

corner incurs a 30-second penalty added on to the next “run.” The minimum “run” time recorded by a mouse at any point during the 10 minutes will count as its final time

2.1.3 Autonomy

The micromouse must be self-contained, prohibiting the use of a remote control. Additionally, the micromouse must remain intact throughout its 10-minute period, not leaving behind parts in the maze. The mouse may not be reprogrammed after the reveal of the competition maze, indicating that the mouse must be capable of learning/discovering the maze and solving the maze without external assistance.

2.2 Design requirements arising directly from the explicit competition requirements

2.2.1 Micromouse chassis size

(i) With the listed maze dimensions, the resulting width of a maze corridor is 16.8cm. This accounts for the width of the wall on either side encroaching half (0.6cm) of the overall wall thickness (1.2cm). Thus, the widest dimension of the chassis (the diagonal in the case of a rectangular shape) must be shorter than 16.8cm with enough tolerance to make a 360-degree turn on the axis in the center of the corridor. With a larger chassis size, there is less room for error, meaning more precise controls are necessary. Due to this, we made a design choice to prioritize minimizing the size of the micromouse chassis to have maximum clearance.

(ii) Maze walls are 5cm tall. Any sensors used to detect walls from within the corridor must be between 0 and 5cm from the floor of the maze. Ideally, the sensors will sit close to half the height of the wall (2.5cm) from the maze floor. If the sensors incur a tilt relative to the floor, this will allow us the most range. Additionally, this will prevent optical signals from reflecting off the floor of the maze, resulting in inaccurate distance readings.

(iii) The walls are white, which has a higher reflectivity than the non-glossy black floor. This provides an advantage for a design using optical sensing methods, resulting in our decision to use optical sensors rather than acoustic. Additionally, the longest possible straight path is 288cm (2.88m). For the micromouse to discover such a path, either the sensing device must have a range greater than 2.88m, or use a navigation strategy in the software that accommodates a device with a shorter range. This weighed into our sensor selection.

2.2.2 Micromouse interaction with maze features

(i) The micromouse must have a mechanical solution for movement. Many designs from previous competitions utilize some form of direct current (DC) motor with plastic and rubber wheels. The choice of motor must consider housing size, power requirements, and motor shaft rotations per minute (RPM). We explicitly investigated variable speed motors. This allows the speed of each motor to be set by a pulse with a modulation (PWM) frequency.

(ii) The number and placement of motors are also design choices influenced by maze features. We decided on two motors placed slightly behind the center of the chassis. This allows for all the basic maneuvers necessary to traverse the maze, which include:

   (a) Forward drive.

   (b) Backward drive.

   (c) On-axis pivot by any angle and in either direction, allowing for left turn, right turn, 180-degree turn, and even smooth arcs during turns.

   Using two motors and wheels works better than other options, such as four, due to cost restraints, increased mobility in small spaces, size, and ease of control.

2.2.3 Micromouse operation

1. The micromouse must be autonomous. This requires that there is a controller onboard to execute instructions, incorporating information input by other onboard devices, and output signals to control other peripherals.

2. The micromouse must have onboard power. The power must have a high enough voltage to power the selected DC motors, sensors, and the selected microcontroller. Additionally, the battery must be capable of powering the entire system for a minimum of 10 minutes for the micromouse to complete the competition period without being removed from the maze.

2.3 Additional design requirements

2.3.1 Physical features

(i) The chassis of the micromouse is the printed circuit board (PCB) on which the components are mounted. Thus, the minimum size is limited by the area required to

mount all the necessary hardware components. Due to this, we aimed to make the board as compact as possible, with the absolute maximum size being as described in 2.B.a.i.

2.3.2 Movement

(i) Part of the competition is speed; therefore, the motors should be capable of moving the micromouse quickly. At higher speeds, it is more difficult to control the micromouse. Choosing a motor gearbox with a reasonable gear ratio (as determined through live testing) was a design decision made by balancing speed and control.

(ii) Motor voltage and drive current are limited by the capabilities of the onboard battery power. Thus, we restricted the voltage to 6V and the maximum current pull to 1.5A per motor.

(iii) DC motors can have built-in encoders, allowing us to access pulse counts that can be used as an input to the control system to keep track of the distance traveled by each motor. This can be used as feedback for driving in a straight line, or as input to turn the mouse by any given range of degrees. Encoders were a required feature of our motors.

(iv) Motors need to be independently controlled and capable of rotating in both directions. This requires an H-bridge for each motor, and a PWM pin on the microcontroller for each motor control. The H-bridge must accommodate the motor voltage, meaning a supply voltage range including 6V. Additionally, the H-bridge must be compatible with a PWM frequency sufficient to drive the motor. Based on research, 20kHz is typically sufficient for micromouse competitions. Thus, the H-bridge must handle about 20kHz PWM frequency.

(v) The wheel diameter must be large enough to keep the mouse off the ground when level. Therefore, the radius must be greater than half the height of the motor housing plus the thickness of the PCB. However, the closer the PCB is to the ground, the closer the center of gravity is to the floor, allowing better control. Smaller wheels also introduce less error when counting motor rotations with encoders.

2.3.3 Sensors

(i) Optical sensing was favored due to the capability to correct for external factors. Ultrasonic sensors could have been affected by external sound waves from HVAC and conversation, which would be difficult to account for.

(ii) Infrared emitters and phototransistor detector measurements can subtract the noise caused by ambient light for each measurement. The coloration of the maze features supports optical sensing. IR emitters and detectors are analog and can be sampled at high frequencies, making them advantageous for this application. IR sensing must work for short-range sensing (1-25cm) to be used for detecting corridor walls and openings.

(iii) Time of flight (ToF) sensors are accurate at longer distances. We wanted to be able to detect around 1-1.5m, roughly half the overall width of the maze. Detecting longer paths without traversing them seemed advantageous for maze exploration and micromouse control. Using an input from sensing an obstacle farther away would enable better movement control. The chosen sensor must operate on the microcontroller logic voltage of 3.3V.

2.3.4 Microcontroller

(i) Must support I2C communications for sensor interface.

(ii) Must have analog-to-digital conversion (ADC) pins for IR detector reading.

(iii) Must be capable of PWM frequency 20kHz to drive motors.

(iv) Must interface with motor encoders and count encoder pulses efficiently (PCNT).

(v) Must complex support hardware operation and maze-solving algorithms, usually requiring a dual-core microcontroller.

2.3.5 Power

(i) The onboard battery must have >6V to drive the 6V motors, which will also be sufficient for the 3.3V microcontroller logic voltage. 3.3V also interfaces with the encoders and distance sensors.

(ii) The 3.3V and 6V regulators should have a maximum input voltage higher than what the battery is capable of (>6-10V).

(iii) The battery life must also run the entire system for longer than 10 minutes for the micromouse to utilize the full competition period without failure.

(iv) The battery power density should be taken into consideration, as the weight of the mouse will affect how much torque the motors must put out and the ease of starting and stopping movement.

(v) USB-C connection will be needed for uploading code to the microcontroller, but it also supplies 5V when plugged in. Power distribution design must prevent backfeeding of battery power into USB-C to protect the host computer when connected.

(vi) The system should have a physical shutoff (switch) for the battery.

3 Detailed Project Description

3.1 Hardware Design Overview

3.1.1 Physical features

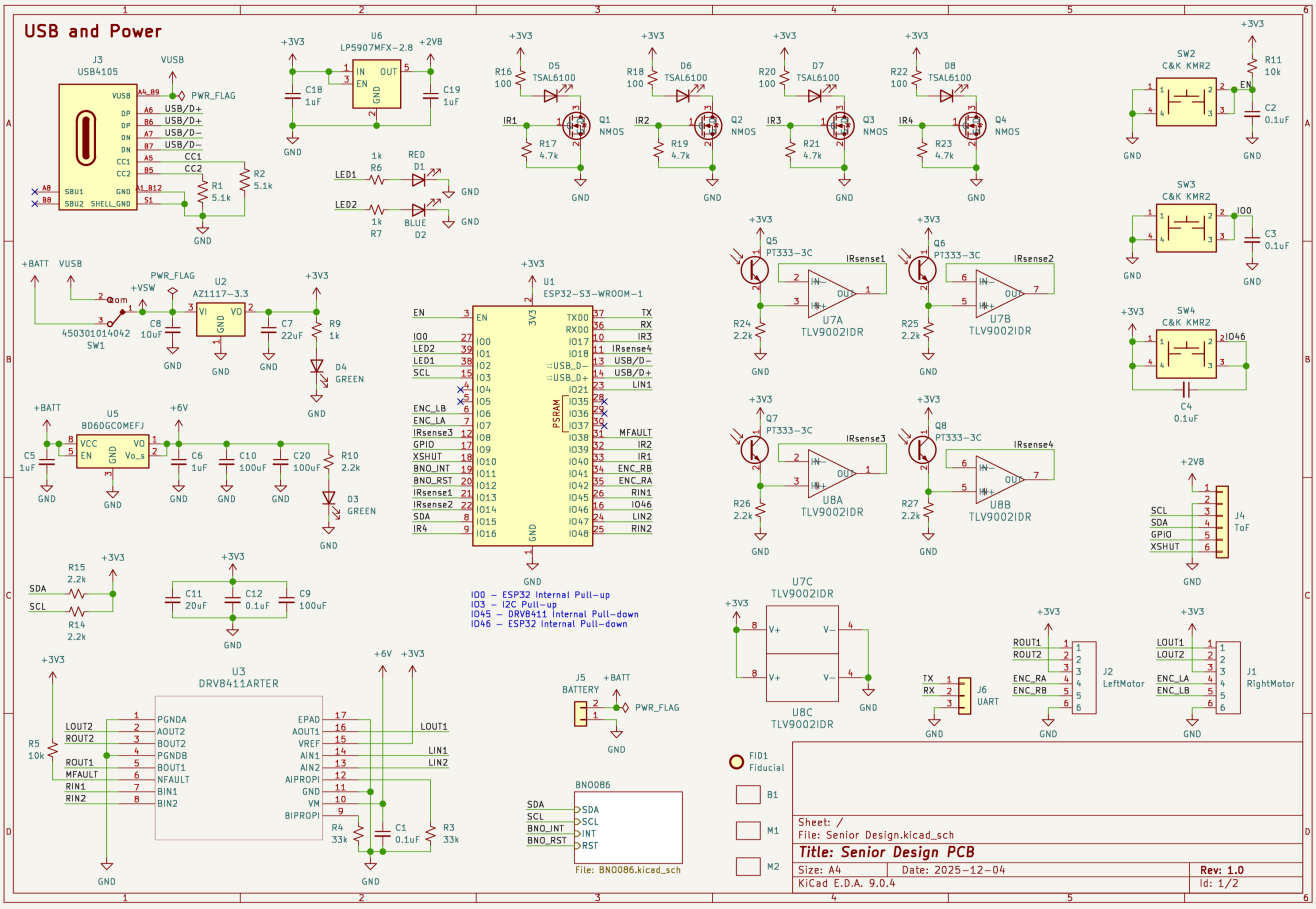
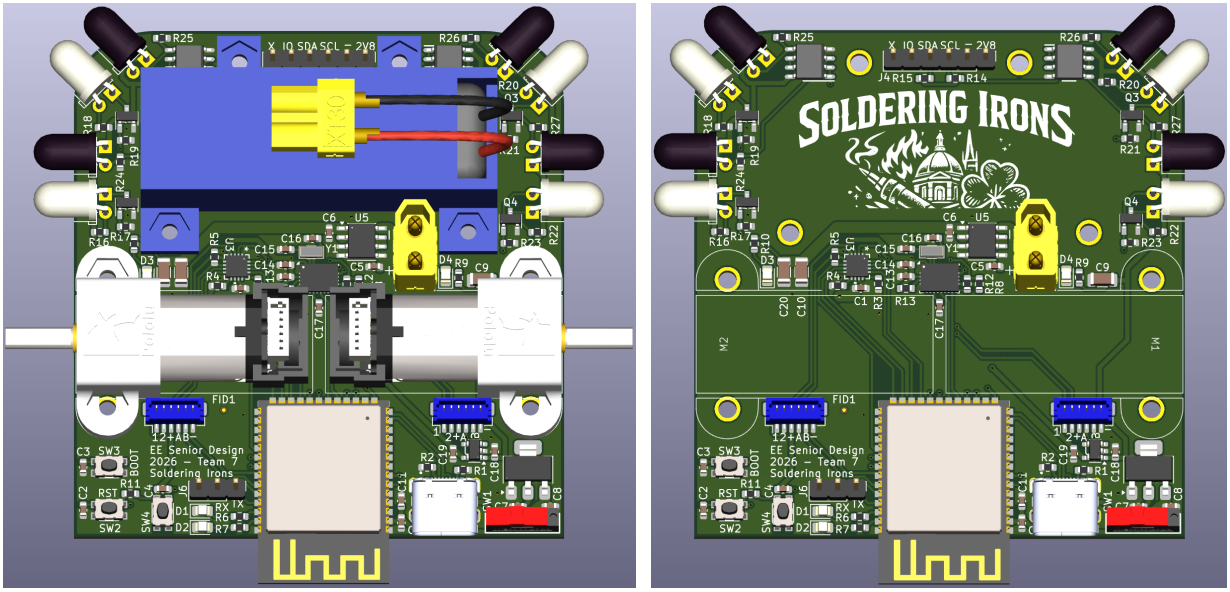


Figure 1: Final Project Schematic

Above in Fig. 1 is our full schematic for our custom PCB.

The PCB is a 2-layer board with a physical size of 67.3x70.5x1.6mm. The longest dimension of our board is the diagonal, measuring 97.5mm (9.75cm), which is well under the design requirement we set at <16.8cm. Screw holes were made for motor clamps and a battery holder.

3.1.2 Mechanical Layout



(a) 3D PCB View with Battery and Motors **(b)** 3D PCB View Bare

Figure 2: 3D PCB View in KiCad

Component placement was guided by size constraints, current flow, and signal integrity. The PCB is organized into functional regions: power entry and regulation, microcontroller, motor driver and motor outputs, and sensor/IMU interfaces. This separation simplifies routing, reduces noise coupling, and supports reliable assembly inside the chassis.

Components were positioned to avoid interference with each other, while connectors and switches were placed near board edges for accessibility. Adequate spacing was maintained around the motors and regulators to support heat dissipation and simplify assembly, and all footprints were verified in the KiCad 3D viewer to ensure proper mechanical fit before fabrication.

Motor connectors are grouped near the motor driver to minimize high-current trace length and reduce internal wiring complexity. High-current and high-generating components, including the 6V regulator and motor driver, are clustered to confine motor paths and limit voltage drop. These components are supported by ground vias and ground copper pours. Ground copper pours fill in both the top and bottom copper layers to spread heat into the PCB and improve thermal performance during sustained or high-load operation.

Sensitive circuitry is physically separated from the motor power region. The ESP32 module is placed to preserve antenna keep-out requirements and reduce EMI from motor switching. The IMU is positioned away from the high-current paths and closer to the mouse's center of rotation to reduce vibration-induced error. Time-of-flight sensor connectors are placed to support perimeter mounting with clear line-of-sight to maze walls while keeping I2C routing short and robust.

3.1.3 Microcontroller

We selected the ESP32-S3-WROOM-1-N16R8 microcontroller. The S3 operates at a logic voltage of 3.3V and boasts low power consumption, making it ideal for interfacing with our sensors and encoders, and ideal for conserving onboard power from the battery. With 16 MB of flash memory and 8 MB of PSRAM, we have plenty of memory for storage of mapping, hardware operations, and maze-solving algorithms. The PWM frequency can be set as high as 40MHz, far exceeding the required 20kHz for the motor speed control. The S3 has three separate UART buses and two I2C buses for sensor integration. Additionally, the pulse count (PCNT) function is supported for reliable encoder counts.

3.1.4 Motor Assembly

(i) Motor:

We used two high-power, carbon brush (HPCB) 6V Pololu Micro Metal Gearmotor. The no-load current is 150mA, and the stall current is 1.5A. The physical dimensions of the motor housings, including the attached encoder and gearbox, are 10x12x32mm, making the absolute minimum width of the micromouse 64mm. These specifications satisfy our set requirements.

(ii) Gearbox

The original gearbox had a 30:1 ratio, making the maximum speed at 6 V 1000 rpm. With a 32mm diameter wheel, this mouse would travel over 9m in just 5 seconds. After testing with this gearbox, we decided a higher gear ratio would allow for better control of the mouse during testing and initial software design. Therefore, during testing and most of the software creation, our design used 75:1 Micro Metal Gearmotor HP 6V with 12 CPR Encoder, Side Connector. This substantially aided software development, especially during the practice mouse phase with no IMU. After the robust software was mostly complete, the team switched back to our original 30:1 Pololu motors. Our final design :1 Micro Metal Gearmotor HPCB 6V with 12 CPR Encoder, Side Connector.

(iii) Encoder

A 12 count per rotation (CPR) encoder is attached to the motor shaft, meaning after the 75:1 gearbox (75.81:1 exactly) there is approximately 909.72 CPR. For the 30:1 gearbox motors (29.86:1 exactly), the encoders have a final CPR of 358.32:1. This allows for high precision reading of the motors to determine length traveled. Additionally, while encoders were initially used for turning and driving straight (PID algorithm), the switch to a more accurate and responsive IMU superseded the need for encoders for straightening. Encoders are always used to measure the distance moved.

3.1.5 Wheels

The final wheels are 32 mm in diameter with rubber treads. The clearance height beneath the board with these wheels is 9.4mm, keeping the board off the ground but still close enough as required.

3.1.6 Motor driver

The mouse uses a DRV8411A Dual H-Bridge Motor Driver by Texas Instruments. The recommended supply voltage is 1.65-11V, which will support the 6V needed for the motors. The logic voltage range is 0-5.35V, which will interface with the 3.3V logic of the S3. The maximum PWM frequency is 100kHz, which exceeds the 20kHz design requirement.

3.1.7 Sensors

(i) IR Sensors

We used TSAL6100 IR emitters, which emit at 940nm. The phototransistors are PT333-3C with peak sensitivity at 940nm. These two components were placed in pairs at 4 locations on the board. The IR emitters and detectors sit roughly 15mm from the ground, which is below our specified ideal; however, this does not affect their performance, as the range is not long enough for errors from reflections off the ground.

Two of the IR pairs are on either side of the board (left and right), and they are perpendicular to the side of the board. The side sensors are used to measure a range of 10-180mm, which is used for centering the micromouse in the corridor and detecting openings as the micromouse travels in a straight corridor. These side sensors are typically used for distances between 10 and 100mm.

The other two pairs of IR sensors are on the two front corners at a 30-degree angle from the side of the. These angled sensors on the front corners are used to detect walls on either side of the cell ahead of the micromouse. This allows for the micromouse to both

know the walls for the cell it is entering, and decide whether to stop, to turn, or to continue with its path. Additionally, since the ToF sensor discussed later was found to be inaccurate for distances less than 50mm, while also having a slow update rate, these angled sensors could be used to determine the distance of the micromouse from a wall directly ahead (using a different calibration look-up table). The angled IR sensors are typically used for distances of 50-200mm for sensing side walls ahead or 10-75mm for sensing a wall directly ahead.

(ii) Time of Flight Sensor

We used one VL53L1X Time of Flight (ToF) sensor on the front of the micromouse. The range is 50mm to 1200mm (1.2m), which is short of our desired 1.5m range. UPDATE The sensor is on the front of the board, used for detecting obstacles directly ahead of the path of travel. This VL53L1X runs only at 50 Hz with inaccurate readings under 50mm (due to photon scattering). The sensor sits roughly 2cm from the ground, which is satisfactory, and after live testing has worked sufficiently.

3.1.8 Battery

Instead of using the two provided 1-cell LiPo batteries used on the practice mouse board, the team decided to utilize a combined 2S (7.4 nominal) battery with 300 mAh capacity. This simplified the power input to 1 plug and reduced the weight and size of the battery. The team believes this was an improvement incorporated in our design, not included by other teams.

3.1.9 Power

The board supports dual power sources: a 2-cell LiPo battery for full operation and a USB-C connection for development and programming. A manual power switch selects between battery input (+BATT) and USB VBUS, preventing backfeeding of battery voltage into the USB port. This ensures safe operation when connected to a host computer while still allowing the microcontroller to be powered independently of the battery.

Power distribution is divided into 2 regulated rails. A 3.3V rail supplies the ESP32-S3 microcontroller, encoders, IMU, IR emitters, and time of flight sensor, while a separate 6V rail supplies the motor driver and motors. Separating the high-current motor supply from the logic supply reduces noise coupling into digital and sensory circuitry.

The 6V regulator was selected to support the expected motor current requirements. Each motor has a stall current of 1.5A, and the regulator is sized to provide sufficient headroom above nominal operating current while tolerating short stall events. This ensures stable motor

operation without excessive voltage droop during acceleration or turning maneuvers.

Bulk and decoupling capacitors are placed on both the 3.3V and 6V rails to stabilize the supplies and suppress transient voltage spikes caused by motor switching and load changes. Local bypass capacitors are placed near the ESP32 and regulators to ensure stable operation during dynamic load conditions.

The manual power switch that switches between +BATT power and USB power acts as a power switch on the battery input to allow the system to be powered down without unplugging the battery. This provides a safe and practical method to disable the motors and logic during testing, handling, and storage. This is especially helpful when the micromouse acts unexpectedly or uncontrollably, and the user wants to quickly shut down the micromouse.

The motor power path is intentionally isolated from the logic rail. The +6V rail feeds only the motor driver and motor connectors, while all control signals remain referenced to the 3.3V logic domain. This separation provides robustness and simplifies debugging.

Linear regulators were chosen for simplicity and layout robustness during the preliminary design iteration. The slight power loss was deemed allowable with the design requirement of only running for a maximum of 10 minutes.

3.2 Software

3.2.1 Overall software structure

The software has a layered architecture consisting of drivers, control, navigation, and main/wireless layers.

At the first level, the drivers layer handles interactions with the hardware, including the motors, encoders, IR sensors, ToF sensor, and the IMU, which comprises a triple-axis accelerometer, gyroscope, magnetometer, and a sensor-fusion algorithm to provide an absolute position and rotation vector with quaternions. This layer primarily handles raw data collection for the race, serving as the first layer between hardware and software (hardware abstraction).

The second layer, the control layer, allows the micromouse to move in a consistent way and achieve fast, stable, and accurate movements throughout the maze. This layer includes a speed control loop, a motion control loop that uses a trapezoidal profiler, a blended IMU and wall-follow PID, and the code needed for turns to be arched rather than sharp pivots.

The third layer, the navigation layer includes a stored map of the 16x16 maze that is updated during each race (maze.c), a cell-to-cell navigation to move the mouse through the maze cell-by-cell (cell_nav.c), a solver that explores the maze using a flood-fill (Dijkstra) algorithm (solver.c), and finally an algorithm that races through the maze using the fastest path it can find (race.c). This layer contains all the logic needed to complete the maze rather than merely drive straight or turn.

The last layer includes the main functions that initialize the entire code and control the start of the solver and racer. This layer includes all the functions at the start of the code to initialize every layer below(main.c), the buttons that allow for user interaction with the micromouse (buttons.c), and the wireless system that allowed for robust user input during testing but was eliminated before raceday (wireless.c and webui.c).

Finally, the code also includes some tool that contains additional code mainly used for debugging and programming, but not used during the race itself.

All of the code used for this project can be found on the team's shared GitHub. (Note that our GitHub is private and not linked here per class rules to not spoil future project years)

3.2.2 Low-level software decisions

3.2.2.1 Drivers Layer

(a) Motors Interface (motors.c)

- The motor driver code takes in values between -100 and +100 from higher-level software to determine the desired motor speed. Positive values make the micromouse go forward, and negative values make it go backward, and a value of 0 means that the mouse should come to a stop. These values are converted to PWM duty cycle values that the motor can process. The PWM frequency used is 20 kHz, and the clock speed used is 2 MHz. These values were chosen so that the PWM frequency was fast enough for the motor to run at a consistent speed and the timer resolution was still high enough to precisely control the speed used. When initially setting up the motors, the motors were tested to determine their deadband. The values [-100,100] passed into the motor driver automatically apply the deadband so that all values [-100,100] are usable as inputs. This is described further below:
- One issue that had to be rectified using this code was that small PWM values originally did not move the motors at all, likely due to issues overcoming friction and torque forces. To rectify this issue, apply a deadband that was measured

experimentally. For the 75:1 motors, the deadband was determined to be below 53% duty cycle. For the 30:1 motors, the deadband was determined to be below 42% duty cycle. Therefore, the inputs were rescaled from mapping linearly between 1% to 100% of the duty cycle to mapping to the motor's effective operating range of about 42% to 100% of the duty cycle. In this way, it is ensured that every motor speed request by a higher-level code has a meaningful response.

(b) Encoders Interface (encoders.c)

- The encoder code uses quadrature encoding (with A and B signals) rather than single-channel encoding to allow for the detection of both the distance and direction traveled. The two signals are phase-shifted from each other so that the relative timing between them can give the program information about the direction the micromouse is moving in. The code counts every edge of the signals: A rising, A falling, B rising, and B falling. This choice was made to allow for higher position accuracy than simply counting once per cycle. Note, the 12 CPR advertised by Pololu is only achieved if every rising and falling edge of both channel A and B is counted.
- Rather than counting these pulses in the software, the code uses the ESP32's built-in PCNT peripheral. This decision was made because the hardware counter can track pulses more reliably than the software, as it does not depend on CPU timing. To account for the limits of the ESP32's ability to count encoder pulses, an accumulator is used. To ensure the counter always remained within safe limits, a maximum of 10,000 pulses and a minimum of -10,000 pulses were set. When the maximum or minimum value is reached, an interrupt fires. This interrupt adds the hardware's counter value to the software's accumulator, and then the hardware's counter value can be set back to zero. The accumulator can then hold the number of encoder counts corresponding to the number of motor rotations completed.
- A glitch filter was also included to protect against electrical noise, so any pulse shorter than 1 microsecond is rejected. This length of time was chosen so that it was long enough to detect these unwanted spikes but short enough to know that they are truly unwanted. In this way, the reliability of the encoder counts is protected.

(c) IR Sensors (ir_sensors.c)

- The IR sensor driver controls the left, right, front-left, and front-right wall sensors.

When the PCB was designed, the front-right sensor was wired to ADC1, and the other three were wired to ADC2. Each ADC channel is sampled multiple times to reduce the noise of each sample. ADC values are measured both with the emitter on and with the emitter off, and the value with the emitter off is subtracted from the value with the emitter on so that ambient light is removed. This method of subtracting ambient light increased the reliability of the sensor in various environments.

- The raw values that the ESP32's ADCs read are then converted to their true distances using a unique lookup table (LUT) calibrated to each sensor through experimentation. Values not directly stated in the lookup table are estimated using linear interpolation between two values. Higher raw values correspond to closer wall distances, and lower raw values correspond to further wall distances. This code also uses a front-wall composite lookup table. It uses both the front-left and front-right sensor readings to estimate how far the micromouse is from the wall directly in front of it. In this way, the mouse is better able to use these two sensors to determine if there is a wall directly in front of it rather than just diagonally right or diagonally left of it. This ability to determine wall distances from each sensor placed in different locations around the front of the mouse helps tremendously with maze navigation.

(d) Time of Flight Sensor (tof_sensor.c)

- In this code, the sensor has been configured to be in short-distance mode. It can measure at most 1.3 meters away to allow for minimal interference from ambient light and greater reliability for close-by walls. The timing budget is set to 15 ms, the time the sensor spends collecting light each time it fires. This low value was chosen to allow for quick updates to the sensor. The inter-measurement period is 20 ms, as this value must be at least 4 ms greater than the timing budget as specified by the manufacturer. Finally, the Region of Interest (ROI) is set to 6x6. This variable is set to be a relatively small value again to reduce noise from outer fields of view, such as side walls, and allow this sensor to point as straight ahead as possible to only see walls directly in front of the micromouse and down a long corridor.

- The ToF sensor code uses the STMicroelectronics VL53L1X Ultra Light Driver (ULD) API library, a public API distributed by the manufacturer, to generate preliminary distance measurements. Like the IR sensors, error values are checked for rejection if found. A unique lookup table generated through experimentation is

also used to correct minor errors in linearly mapped raw value and distance readings. This ToF sensor works in conjunction with the front-wall composite lookup table for the IR sensors in that the IR sensors can quickly estimate if a wall is present for alignment purposes, and the ToF sensor can accurately and directly determine if a wall is present directly in front of the micromouse at longer distances.

(e) Inertial Measurement Unit (IMU) (bno086.c)

- The ESP32's IMU driver communicates with the BNO086 sensor over a shared I2C bus. For communication, the code uses the manufacturer's sh2 library, which translates between the ESP32 and the IMU, stopping the code from needing to manually handle the low-level communication information. For this project, the sensor is only configured to output one main type of data: game rotation vector (GRV) quaternions, which represent the direction of the micromouse in 3D space. The BNO086 utilizes its own robust fusion algorithm to robustly combine sensor values from its onboard accelerometer, gyroscope, and magnetometer to compute absolute orientation data. By easily converting the quaternions to yaw angle, the micromouse extremely and quickly determines the absolute angle the micromouse is compared to the direction the micromouse was calibrated at upon initialization of the micromouse in the maze. The code receives this data, interprets it, and stores the most recent values so that other parts of the system can use them.
- The code converts the quaternion—how the orientation in 3D space is measured—into a yaw angle—which depicts how much the micromouse has turned left or right. The yaw angle can vary from -180 degrees to 180 degrees, so yaw unwrapping is used to keep track of times that the yaw angle increases or decreases past these values. In this way, the code can methodically and reliably track the micromouse's turns. This sensor is an extremely capable device that allows for precision control of the micromouse.

3.2.2.2 Control Layer

(a) Speed Control Loop (speed_control.c)

- The speed control code implements a closed-loop speed PID controller for each drive motor. In this way, the motor behavior is continuously monitored through the encoder count, and the motor speed is adjusted as needed. Encoder counts are

converted into measured wheel speed in millimeters per second, and this measured speed is compared to a target speed set by the higher-level motion-control software. If there is a difference between the desired and actual speed, the PID adjusts the motor input to account for that difference.

- The PID combines variables to create the motor command. It has the standard proportional, integral, and derivative variables, as well as target variables. These variables are reasonable guesses for where the PID should start based on what the desired speed is. In this way, the micromouse can more quickly determine what speed the motors should be moving at.
- To prevent the integral term from becoming too large over time, the code uses clamping. This technique places limits on how large the integral term can grow. Similarly, the integral is reset to zero when the target speed is set to zero. In this way, the PID avoids errors resulting from this buildup and allows the motor speeds to remain stable.

(b) Motion Control Loop (motion_control.c)

- This loop is the higher-level code that determines what the wheel speeds should be. When the micromouse is driving straight, the motion profile, set by higher-level code, provides the desired forward speed, and a PID generates a correction value needed to keep the micromouse moving in the intended direction. When needed, the code will increase the speed of one wheel and decrease the speed of the other to get the micromouse moving straight once again. This correction is determined using information from multiple sensors, including the IR sensors and the IMU, using multiple PIDs. There is also a wall-follow PID. If no walls are seen, this PID does not influence how the micromouse drives straight, and only the IMU is used to determine the correction needed. If one wall is seen, it makes up some of the determination, and if two walls are seen, they make up more of the determination. In this way, the micromouse uses as much data as it can from its surroundings to determine how to accurately drive straight.
- When the micromouse is turning, the system uses a separate motion profile that defines how quickly the micromouse should rotate. This code does not use data from the encoders because they are not accurate during turns due to varying angles and slippage. The code instead just uses the IMU's yaw measurement to track how far the micromouse has turned. It then compares the turn angle set by the

higher-level motion profile code to the actual yaw angle measured by the IMU and corrects for error.

- The code also has arc motion, which allows the micromouse to follow curved paths without having to take multiple turns with stops in between them. The code sets a radius and angle for the arc determined by higher-level code and then converts them into a distance that the micromouse should travel to complete that arc. The inner wheel must travel a shorter distance than the outer wheel, so the code continuously sets different target speeds for the left and right motors based on this difference in distance. This code is especially helpful when trying to complete the maze as quickly as possible during the race.
- Finally, this code also has an extra safety measure for front-wall detection. Since there can be slight errors in the higher-level motion profile set values, the micromouse can potentially come closer to a wall than desired or expected. This code continuously checks the distance to a wall in front of the micromouse using both IR sensors and the ToF sensor to prevent these accidental collisions. Since the IR sensors update more quickly, they are used whenever possible. However, the IR sensors cannot detect walls as far as the ToF sensor can, so if the IR sensors cannot detect a wall, the code defers to the ToF sensor. If any of these sensors come too close to a wall, they override the rest of the driving code and stop the micromouse to stop it from colliding with the wall.

(c) Trapezoidal Motion Profiler (motion_profile.c)

- The higher-level motion profile code determines the desired velocity for the micromouse throughout its movement. It uses a trapezoidal profile, meaning that the micromouse accelerates to a maximum velocity, stays at that speed, and then decelerates before reaching its desired location. This maximum velocity is provided from the lower-level motion control system, and is determined by set speed limits, entrance and exit velocities, and the PID tuning parameters, so that consecutive movements can be chained together easily. During each update, the code calculates how much distance the micromouse has left to travel and compares it to the distance needed to slow down accurately, based on velocity and a set deceleration rate. When the micromouse comes within this distance from the target location, it decelerates by reducing its velocity at a fixed rate for each cycle. In this way, the micromouse is prevented from making extremely jerky starting and stopping motions.

3.2.3 Higher-level software design

3.2.3.1 Navigation Layer

1. Cell Navigation (cell_nav.c)

    - To navigate the maze properly, the mouse must first be able to properly position, or center, itself within each cell. To do so, this code keeps track of the micromouse's current position in the maze and checks that movements are carried out, keeping in mind the maze's 16x16 structure. This code uses wall sensing at a fixed distance after leaving the center of each cell. The data from the IR and ToF sensors allows the code to determine the presence of walls on the left, right, and front sides of the cell. It assumes there is no wall behind the micromouse, as it came from this location. The code records this information in the maze map and calls a callback function to the maze-solving algorithm, which returns the next action the micromouse should take. This code continuously updates the micromouse's position as it crosses cell centers and notes cells within the maze that it has already crossed through. By combining position tracking, real-time sensing, and continuous motion, this layer allows motion control and maze navigation code to work well together and allows the micromouse to explore the maze efficiently.

    - If the higher-level maze-solving code determines that the micromouse should continue moving forward, the cell navigation code continues the current motion using the aforementioned motion control system rather than stopping and starting a new movement. This allows the micromouse to pass from one cell to the next without stopping, which is especially important for maintaining speed. If the micromouse needs to turn, this system allows the motion profile to naturally decelerate the micromouse to the center of the next cell before turning, allowing all turns to be specific and meaningful. This system allows the micromouse to navigate the maze extremely efficiently without unnecessary stops.

2. Maze (maze.c)

    The maze code stores the micromouse's map of the maze. Only the 16x16 perimeter of the maze is present from startup to stop the micromouse from attempting to drive outside the maze. As the micromouse navigates through each cell, it records whether walls are present on its north, east, south, and west sides, whether the cell has been visited, and whether the cell is the target destination location. When a wall is found, the code also updates the cell that uses this same wall to allow for faster collection of data

about the whole maze. As a result, the code also keeps track of whether the walls it has found have been directly observed or not. This code allows the cell navigation and solver code to have an updated map as the micromouse moves and to determine where the micromouse can and should move next.

3. Solver (solver.c)

   The solver code implements a modified flood-fill algorithm based on Dijkstra's algorithm, used to choose the micromouse's next movement through the maze. Each time new wall information is gathered, the algorithm computes a new cost map where lower-cost cells are closer to the desired target destination. During the first phase, the target is the maze's goal. Once the goal is reached, the solver enters a scanning phase where it continues navigating until all goal cells have been visited, after which the target switches to the starting cell so the micromouse can return to the beginning of the maze. The algorithm adds a penalty for moving into already visited cells, encouraging the micromouse to explore new cells whenever possible rather than backtracking. After calculating the cost map, the solver tells the rest of the code to move into the open neighboring cell with the lowest cost by converting that direction into an action such as continue forward, turn left, turn right, turn around, or stop. When two neighboring cells tie on cost, the solver breaks the tie using a configurable priority scheme. Three options are available: a cardinal-direction priority of N > E > S > W, its opposite S > W > N > E, or a heading-relative priority of straight > right > left > turn around. The default used in this project was the heading-relative straight priority, which minimizes unnecessary turns by preferring to continue in the current direction whenever costs are equal.

4. Racer (racer.c)

   - The racer code is responsible for executing the fastest possible run through the maze once it has already been fully explored. Rather than deciding movement one cell at a time, the racer uses Dijkstra's algorithm to pre-plan an entire path from start to goal through the known maze before moving. Unlike the exploration solver, this Dijkstra includes a configurable turn penalty added to the cost each time the path changes direction, allowing it to prefer routes with fewer turns even if they are slightly longer in distance. This turn penalty currently adds 0.5 "cells" of length per turn taken on each path. Once the optimal cell sequence is found, it is compressed into an ordered list of motion segments — straight drives and turns — which are then executed in sequence. Consecutive non-turn segments are chained together at a reduced handoff speed rather than stopping between them, keeping the

micromouse moving as continuously as possible. During straight segments, the racer also reads the front time-of-flight sensor near the end of each move to detect and correct any accumulated odometry error before the next turn.

- A late addition to the code, when smooth arc mode is enabled, 90-degree turns are replaced with continuous curved arcs at a fixed radius of 90 mm rather than stopping in place to rotate. To accommodate this, the path compression step shortens the preceding straight segment so the micromouse begins curving at the correct entry point, and the arc and following straight segment are chained together at speed so the micromouse never fully decelerates through the corner. No more than two consecutive arcs are allowed in a row to avoid compounding geometry errors, and 180-degree U-turns always use an in-place rotation regardless of arc mode. Since these arcs were added at the end, not enough time was spent perfecting and dialing in the turns, and sensor readings (IR and ToF) are not used to correct any errors in arcs.

4 System Testing

4.1 Practice mouse

4.1.0.1 Microcontroller

The provided practice mouse was equipped with an ESP32-S3 microcontroller. We ran test programs such as “blinky” to verify the board’s ability to program and communicate with an LED onboard.

4.1.0.2 Sensors

(i) The initial optical sensor we chose to purchase was the Adafruit VL53L0X. This came on a breakout board. We installed it using 90-degree pins and soldered it to the front left sensor location. The associated shorting jumpers were modified to accommodate the digital sensor. Simple code was written to receive input from the device and display the measured distance in millimeters on the OLED display provided with the practice mouse. The sampling frequency was 10 Hz for initial verification and testing of accuracy. Through this testing, we discovered that the ToF sensors work well with ±10mm accuracy at distances over 5cm. With a 7cm board width and 16.8cm corridor width, the side sensors would need to be accurate at shorter distances. Additionally, we were

unsure if we could sample fast enough with the ToF sensor to make adjustments for drive centering.

(ii) We installed IR emitters and phototransistors on the left and right sides of the practice micromouse. These were calibrated by plotting an ADC vs. distance scatter plot and fitting a curve to the data. The function defining the curve could then be used to convert the ADC value to a distance in millimeters. The IR sensors could be sampled faster than ToF. We created a software solution for making IR measurements, which subtracted ambient light and averaged three individual measurements in order to increase the accuracy and reduce error from noise or odd reflections. The side sensors were used to help the micromouse stay centered as it drove down the corridors and to prevent collisions with walls.

4.1.0.3 Motors

We tested the HPCB 6V Pololu Micro Metal Gearmotor 12 CPA Encoder motors on the practice micromouse. The motors were driven by a 5V regulator provided by Professor Schafer, which is 1V less than our final design, which we accounted for when evaluating results. We ran tests to determine key PWM values, such as minimum start and minimum keep running PWM. Additional tests were performed to determine the number of encoder counts needed to make maneuvers, including 90- and 180-degree turns, as well as encoder counts for the micromouse to travel a specified linear distance. This testing allowed us to evaluate the motor and encoder performance, leading to changes for our final design. We adjusted the gear ratio for finer speed control and added an IMU to more accurately measure micromouse turns and rotations.

4.1.0.4 Movement “S-shape”

To test preliminary progress for the micromouse navigating its way through the maze, we made it traverse through an S-shape section of the maze. We made sure that the mouse did not hit any walls, even when the maze involved right turns at the end of the first corridor and a left turn at the end of the second. Successful completion of this test indicated that the choice of sensors and control algorithm would work well in our final micromouse design.

4.2 “Soldering Irons” Custom PCB Mouse

4.2.0.1 Solder-Rework

The team spent multiple hours testing every component on our custom PCB board once it was assembled. The first step for the team was to create our own pins.h file to define the hardware connections to each device on the PCB. After, the team tested each component separately to determine if it would need solder rework. In a few cases, some components had to be resoldered or reworked with the hot air gun and magic gel flux. The testing and reworking of various components is described below:

(i) **ESP32:**

Originally, the pins on the ESP32-S3 were not all connected to the pads under it. We believe that not enough solder paste was applied to these pads. To rectify these poor connections, magic gel flux was applied to each pad connector on the ESP32, and a soldering iron was used with additional wire solder to add solder between each connection to the PCB. While most connections were originally solid, each connection was soldered again to ensure continued reliability.

(ii) **USB-C Connector:**

The USB-C connector has very small pins that must transmit accurate data for programming between our computers and the ESP32. Originally, the team was having various programming issues where sometimes the board would not be discovered by PlatformIO, sometimes the programming would fail, and other times, the programming would work just fine. After the direction of Professor Schafer, magic gel flux was applied to all the pins, and a soldering iron was applied over all the pins. This drastically improved the connection for the USB-C connector, allowing for programming each time attempted.

(iii) **DRV8411 Motor Driver:**

The DRV8411 motor driver was tested by trying to run each motor separately for 1 second each on and off at full power. This was used to discount any PWM problems, but instead just tested the raw motor connection and driver functionality. Originally, when trying this test, only one of the two motors would turn, so the connections between the motor driver, ESP32, and motor connectors were tested. The motor driver had to be reapplied with the hot air gun and magic gel flux. The DRV8411 was slightly difficult to

troubleshoot since most of the pins are under the device. After resoldering the motor driver with the hot air gun, each motor worked properly.

(iv) **BNO086 IMU:**

Like the motor driver, when testing the BNO086, the IMU connections to the ESP32 appeared to be unreliable, so solder hot air rework was used to resolder and apply the BNO086. All of the BNO086's pins are completely under the device, inaccessible to a regular soldering iron. After using the magic gel flux with the hot air rework station, the BNO086 connections worked properly, and the ESP32 received accurate measurements at high speed over I2C.

(v) **VL53L1X Time of Flight:**

Like the BNO086, all the pins for the time of flight sensor are under the device, which would require hot air rework. However, unlike the IMU, the VL53L1X sensor worked on the first try over I2C with the ESP32. The team believes a correct amount of solder paste was applied to the custom PCB time of flight breakout board, while too little solder was applied to the main custom PCB

4.2.0.2 Post-assembly IR calibration

After verifying the basic connections and functionality of all the devices on our custom PCB, the IR sensors had to get a new lookup table calibration. It was at this point that the ir_sensor.c code was created to speed up the calibration of each sensor. The team measured the raw ADC value received by the ESP32 for various wall distances from the sensor to populate a new lookup table.

4.2.0.3 Motor Calibration

The practice board code relied on various known hardware values to achieve straight-line motion and turning using encoders. At this point, the code still used encoders for both, so the known dimensions and other hardware values were added to the code and tested using code blocks used previously in design reviews. The main set of code used to calibrate the motors again was the drive 720mm and turn repeatedly code.

4.2.0.4 S-Shaped Maze (5x3)

The first serious test to integrate all the pieces together was putting our custom mouse design in the S-shaped maze, again used on the practice mouse. By continually running our

micromouse through the S-shaped maze, the team was able to dial in PID values for the encoders and wall-following while also starting to develop code to keep track of the maze (maze.c). Here, cell_nav.c was also born to more efficiently control the movement of the micromouse from cell to cell instead of simple straights and turns. Finally, wireless.c and webui.c were created here to aid in changing PID values and visualizing what the micromouse was seeing and recording for cells. Once all this code and testing were complete before Spring Break, the team took a long testing break, waiting for a larger portion of the maze to test in.

4.2.0.5 5x5 Maze

The next and significant testing occurred in the 5x5 maze portions left in the senior design room. This allowed for further testing of maze.c and cell_nav.c while also testing how the micromouse could traverse a larger grouping of cells. The larger the maze the micromouse could explore, the greater the potential to add error as the micromouse moved from cell to cell.

In addition, the next big addition here was the beginning of the use of the IMU data, specifically the game rotation vector yaw angle, to control the heading of the micromouse and to aid in precise turns. The IMU allowed for an initial direction to be set, and each turn would set a target change of 90 degrees so the micromouse would always snap to be perfectly perpendicular on each turn. This was a monumental achievement that allowed for better control to go straight and make very accurate turns.

4.2.0.6 Full Maze

Finally, the step we've all been waiting for... a full maze was created to allow for better testing of the micromouse. In particular, this full maze enabled more accurate testing of maze sensing and recording. This tested the flood-fill algorithm to accurately traverse a large maze and get the micromouse to the center.

Lastly, the full maze design enabled the addition of a faster race mode. Once the team was confident in the solver/explorer portion of the code, a completely explored (or sufficiently explored) maze could be converted into the fastest path from the start to the finish. By turning the cell paths into segments that ignored simple cell bounds, the micromouse was able to complete the maze faster than ever before.

In the final few hours of testing, an additional experimental mode was added to the code that allowed for smooth arcs and turns for the micromouse. In theory, this would allow the mouse to not stop at any turn and instead chain together segments to achieve a faster race time.

However, since this mode was not entirely necessary, little time was spent refining it, but during most runs, the smooth race mode would work to complete a very fast race time.

This experimental arc mode ultimately proved to be one of the most impactful additions to the project, contributing directly to the team's winning 38-second competition run — a 34% improvement over the initial cell-by-cell time. The full impact of this mode on competition performance is discussed in further detail in the Section 5 Conclusions.

5 Conclusions

The Soldering Irons team successfully designed, built, and tested a high-performance autonomous micromouse that completed the IEEE Micromouse Competition maze in 38 seconds—the fastest time among all competing teams. This exceptional result was achieved through disciplined design practices, early and frequent component testing, iterative software development, and strategic hardware choices that prioritized both precision and speed.

5.1 Competition Performance and Results

Of the twelve teams competing, only five were able to successfully navigate and explore the maze to reach the center. The Soldering Irons team not only completed the maze exploration but achieved a first-run completion time of 58 seconds using standard cell-by-cell navigation with precise turns. After implementing smooth arc motion in the final hours of testing, the team reduced this time to 38 seconds—a 34% improvement in race time. The second-place team finished in 1 minute 8 seconds, followed by the third-place team at about 1 minute 40 seconds, demonstrating a significant performance gap.

This competitive advantage stemmed directly from the team's commitment to avoiding procrastination and conducting rigorous testing throughout the design cycle. While many teams spent the weeks leading up to competition debugging fundamental issues with motor control and maze navigation, the Soldering Irons team had validated core hardware functionality early in the semester using the practice mouse board, allowing the custom PCB to be deployed with confidence and minimal rework.

5.2 Key Technical Achievements

5.2.1 IMU Integration

The most impactful technical decision was the integration of the BNO086 Inertial Measurement Unit (IMU). Initially, the team relied on motor encoder feedback for both straight-line motion and turning control. However, encoders proved unreliable during turns due to wheel slippage and variable ground friction. The switch to absolute orientation tracking via the IMU's game rotation vector fundamentally transformed the micromouse's capability.

By tracking the yaw angle with quaternion-based sensor fusion, the team achieved precision

heading control within a fraction of a degree. This allowed the micromouse to execute perfect 90-degree turns, align itself accurately along corridor walls, and maintain consistent paths through the maze. The IMU's onboard accelerometer, gyroscope, and magnetometer fusion algorithm eliminated the need for complex odometric calibration and made the system robust to variations in floor surface and wheel wear. This single upgrade was responsible for enabling the smooth arc mode and the substantial final-run time improvement.

5.2.2 Hardware Design and Manufacturing

The custom PCB was successfully designed, fabricated, and integrated with minimal critical rework, all before the start of Spring Break! The two-layer 67.3 × 70.5 × 1.6 mm board fit well within the design constraint of 16.8 cm diagonal clearance, leaving ample safety margin. Component placement prioritized signal integrity and thermal performance: power entry and regulation were clustered with motor drivers to minimize high-current trace lengths, and sensitive circuitry (ESP32, IMU) was physically separated from motor switching noise. The use of dual regulated rails (3.3 V and 6 V) isolated logic-level signals from motor transients, reducing EMI-induced errors in sensor readings and IMU data.

The bill of materials totaled $212.65 per unit, including prototype PCB manufacturing and shipping. This cost-effective design leveraged readily available components (Pololu motors, Texas Instruments drivers, STMicroelectronics sensors). The choice to use a single 2S LiPo battery instead of two 1-cell batteries simplified the power distribution, reduced weight, and improved reliability. Our BOM table below shows our detailed bill of materials. Note: This bill of materials reflects unit cost for the minimum parts necessary; the team ordered additional quantities of many components for redundancy, and the cost does not include stock components provided by the course.

Item	Qty	Unit	Total
30:1 Micro Metal Gearmotor HPCB 6V with 12 CPR Encoder	2	$32.45	$64.90
Pololu Motor Bracket Pair - Black	1 pair	$2.95	$2.95
Pololu 32×7mm Wheel Pair - Black	1 pair	$4.24	$4.24
TATTU 2S LiPo Battery 300mAh 7.6V	1	$9.99	$9.99
MAIN PCB ASSEMBLY			
ESP32-S3-WROOM-1-N16	1	$6.56	$6.56
BD60GC0MEFJ Linear Regulator 6V 1A	1	$1.40	$1.40
LP5907MFX Linear Regulator 2.8V 250mA	2	$0.57	$0.57
DRV8411 Motor Driver H-Bridge 1.65-11V 4A	1	$2.30	$2.30
BNO086 IMU Accelerometer/Gyro/Compass	1	$13.40	$13.40
PT333-3C IR Photodiode 940nm	4	$0.21	$0.84
TSAL6100 IR LED 940nm 100mA	4	$0.33	$1.32
Tantalum Capacitor 100µF	3	$0.48	$1.44
Various Stock Capacitors 0603	-	—	-
Various Stock Resistors 0603	-	—	-
9HT10-32.768KBZF Crystal 32.768kHz SMD	1	$0.68	$0.68
BSS138AKS Logic-level MOSFET	2	$0.26	$0.52
Slide Switch SPDT 500mA 12V	1	$2.03	$2.03
XT30 Connector Gold Plated Battery	1	$1.90	$1.90
TLV9002 Op-Amp Dual Opamp 8-SOIC	2	$0.59	$1.18
LEDs, Logic Transistors, Push Buttons (Stock)	—	—	-
PCB Manufacturing (JLCPCB)	1	$3.20	$3.20
PCB Solder Paste Stencil (JLCPCB)	1	$7.06	$7.06
Main PCB Subtotal			$44.4
TIME OF FLIGHT SENSOR BREAKOUT PCB			
VL53L1CXV0FY ToF Distance Sensor 4M I2C	1	$5.77	$5.77
Various Stock Capacitors 0603	-	—	-
Various Stock Resistors 0603	-	—	-
Stock Dual NMOS Transistor (SOT-363)	1	—	-
PCB Manufacturing (JLCPCB)	1	$4.20	$4.20
PCB Solder Paste Stencil (JLCPCB)	1	$3.03	$3.03
ToF Breakout Subtotal			$13.00
PCB Tariffs	1	—	$13.42
Total Item Cost			**$152.90**
Pololu Shipping	1	—	$11.45
Amazon Shipping	1	—	$0.00
PCB Shipping	1	—	$48.30
TOTAL MICROMOUSE COST			**$212.65**

5.2.3 Multi-Layered Software Architecture

The four-layer software architecture—drivers, control, navigation, and main—proved highly maintainable and scalable. The driver layer abstracted hardware complexity, the control layer managed motion dynamics with trapezoidal profiling and blended sensor fusion, and the navigation layer implemented maze exploration and race planning. This separation allowed different team members to work in parallel and enabled rapid iteration on high-level algorithms without disrupting low-level hardware functionality.

The modified flood-fill algorithm (based on Dijkstra) efficiently explored the maze, and the racer module's path compression and arc mode added sophistication without introducing fragility. The ability to visualize maze state in real-time via the wireless web UI during early testing phases accelerated debugging and tuning.

5.3 Lessons Learned

5.3.1 Early Hardware Validation is Critical

Testing core components (motors, sensors, microcontroller) on the practice board before committing to the custom PCB design eliminated major design flaws and informed component selection. The team identified motor deadband characteristics, IR sensor calibration procedures, and encoder behavior patterns months before the final competition. This early work meant the custom board could be deployed with confidence.

5.3.2 Sensor Fusion Outperforms Single-Sensor Reliance

The initial encoder-only approach was abandoned in favor of a blended system: IMU for absolute heading, encoders for odometry validation, IR sensors for wall centering, and ToF for long-range obstacle detection. No single sensor was perfect, but by weighting them appropriately, the team achieved robust and responsive motion control. The lesson is that complexity in sensor fusion is justified when it improves reliability and performance.

5.3.3 Iterative Testing on Progressively Larger Mazes Was Essential

Testing progression—S-shaped maze, 5×5 maze, full maze—allowed the team to isolate and debug problems at manageable scale. A full-maze test with broken maze-exploration code would have been overwhelming; incrementally growing the test scope caught errors early and validated each layer of the control hierarchy.

5.4 Final Design Assembly

Figure 3 shows the final assembled Soldering Irons micromouse from multiple angles, demonstrating the compact integration of all hardware components on the custom PCB with motors, sensors, and battery mounted.

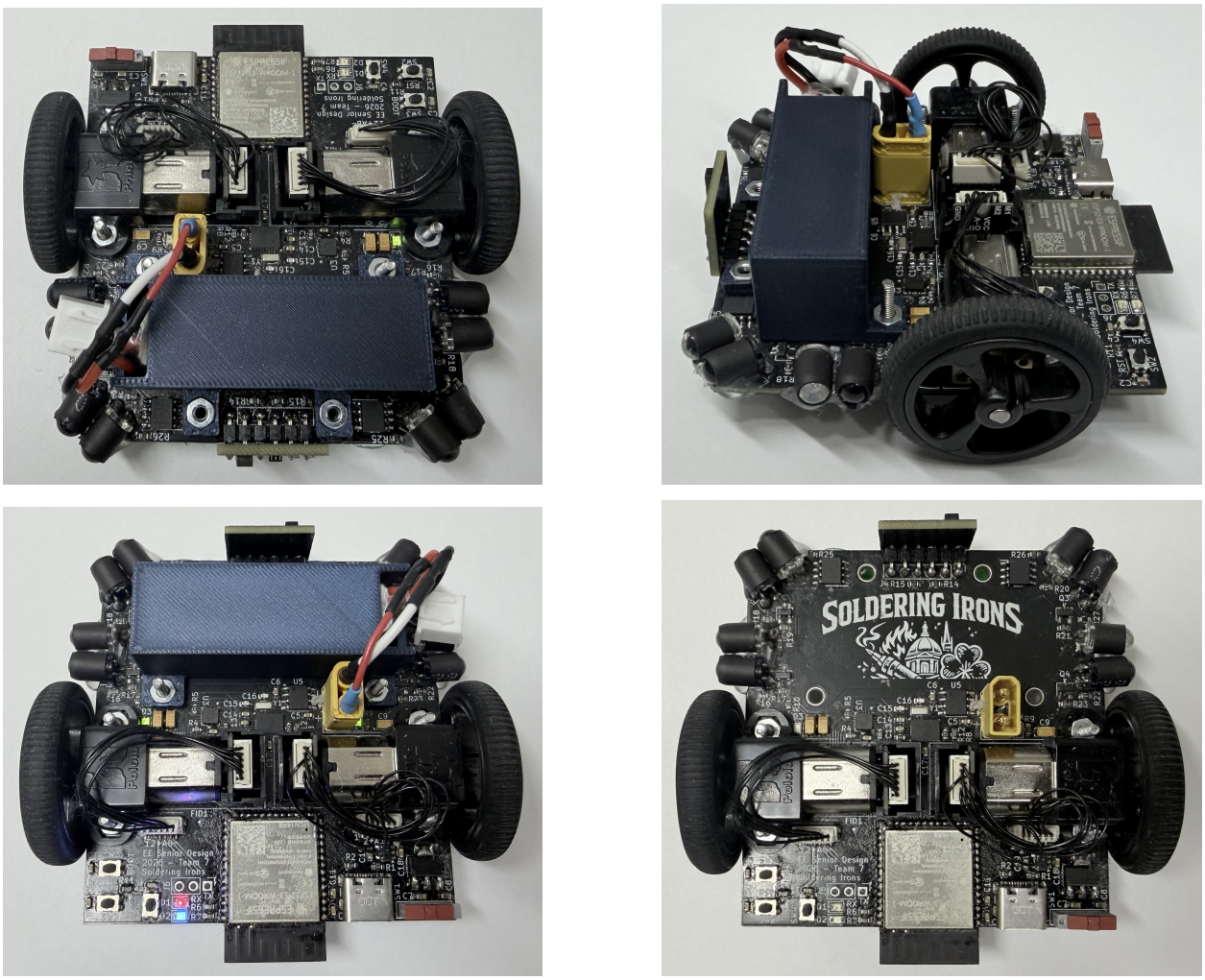


Figure 3: Final Assembly of the "Soldering Irons" Micromouse

5.5 Areas for Future Improvement

While the design was successful, several opportunities for enhancement exist:

- The ToF sensor's 50 Hz update rate and 50 mm minimum range limited its utility for fine motion control. A faster, shorter-range distance sensor would improve obstacle detection and emergency stopping response without relying on angled front IR sensors.
- The smooth arc mode was implemented in the final hours and not fully tuned. Given more time, arc-based navigation could potentially reduce race time further by eliminating acceleration and deceleration transients between turns.
- Power consumption was not extensively optimized. The system ran the motors continuously at full speed and had no sleep modes, consuming battery capacity inefficiently. Adaptive speed profiling or reduced-duty-cycle sensing could extend operational time.

5.6 Summary

The Soldering Irons micromouse project demonstrates that disciplined engineering practice, early component testing, and thoughtful sensor fusion yield competitive results. By integrating an IMU for absolute orientation tracking, designing a compact, robust PCB, and implementing a multi-layer control architecture, the team created a system capable of navigating a complex maze faster than any competitor. The 38-second completion time and 34% improvement from initial to final run reflects not only sound hardware design but also the refinement that comes from robustly testing the project and each part of the micromouse continuously throughout the engineering lifecycle.

The project's success underscores the value of avoiding last-minute panic and instead building confidence through incremental validation and iteration. Future micromouse competitors would benefit from adopting similar practices: validate cores early, embrace sensor fusion, test incrementally, and prioritize reliability over premature optimization. Micromouse was a great project to conclude our Notre Dame EE careers.