MicroMouse

Blase Fencil, Frank Drugge

1 Motivation and Overview

MicroMouse is a popular robotics competition where participants design and build small autonomous robots capable of navigating a maze. The goal of the competition is for the robots, known as MicroMice, to find the optimal route from a predetermined starting point to the center of the maze in the shortest possible time.

MicroMice are equipped with sensors, such as infrared, ultrasound sensors, or time of flight sensors, to detect walls and obstacles in the maze. These sensors help the robots make real-time decisions about the direction in which to move. The robots are typically programmed using algorithms that enable them to explore the maze efficiently and determine the best path to the center.

Competitions like MicroMouse provide an opportunity for students, hobbyists, and engineers to apply their knowledge of robotics, programming, and problem-solving skills in a fun and competitive environment. Participants often compete at local, national, and international levels, showcasing their creativity and technical expertise in designing these small autonomous robots.

We want to test and hone our mechanical, electrical, and software skills through this project(creating a MicroMouse), meaning that we are striving to pick and stick-to a project that will(through its implementation) both allow us to strive towards a meaningful goal, and to be challenged by the goal.

The goal of making a functional MicroMouse will be that of creating a device that acts autonomous(e)ly with **repeatable** and **precise** actions such that it can reliably follow an algorithm in order to solve a **complex** puzzle like a maze.

2 Functional Definition

The primary objective of the MicroMouse project is to create a functional autonomous robot capable of traversing a maze. This section details the key functionalities and performance expectations of the MicroMouse:

- Maze Exploration: The MicroMouse will be able to autonomously explore a maze with the intent of reaching the maze's center. The robot will enter the maze from a predetermined start point, navigate through corridors, make decisions at intersections, and efficiently locate the maze's center.
- Obstacle Detection and Avoidance: The MicroMouse must possess the ability to detect and avoid obstacles, including walls and dead ends. It will rely on its sensors to assess its immediate surroundings, allowing it to adjust its path when necessary.
- Basic Decision-Making Algorithms: The robot will employ elementary decision-making algorithms to determine its path within the maze. Initial algorithms may include wall-following techniques or random exploration. The MicroMouse's design will allow for algorithm enhancements in future iterations.
- Efficiency: While the primary goal is maze traversal, the micromouse will strive to do so efficiently. It should aim to minimize the time required to reach the maze's center or other assigned goal, even though optimal pathfinding algorithms are not a strict requirement for the initial version

Creating the maze structure is a critical component of the MicroMouse project, as it directly influences the robot's ability to navigate effectively. The maze construction process involves careful consideration of materials, and dimensions, to ensure a consistent testing area for the MicroMouse.
To begin with, the maze structure can be assembled using a variety of materials, depending on the project's available resources and objectives. Common options include wood, 3D-printed parts, or even Lego bricks. Each material choice brings its own advantages and considerations. Wood offers durability and stability, while 3D-printed parts allow for more intricate designs. Lego, on the other hand, provides modularity and flexibility for changing maze layouts. We ended up using wood for the maze construction, the maze was just a few turns and a couple branching paths so we didn't need that much material. It's important to keep in mind that the material chosen should be tall enough to trigger the sensors.
However, regardless of the chosen material, maintaining consistency in maze dimensions is of paramount importance. Ensuring uniformity in the dimensions of the maze corridors, including the width of the passages, is crucial for several reasons. First and foremost, consistency in maze structure simplifies the development of algorithms for maze traversal. A consistent maze layout minimizes the number of unknown variables the MicroMouse has to contend with, which is particularly important when starting with a basic maze-solving algorithm. We want the maze to be as close to a theoretically ideal maze

as possible or else we have to implement more and more non-ideal correcting factors into the algorithim design.

3 Parts and Re-usability

The main parts that we will use in this project are going to be our sensors, chassis, and motors. All of these, with the possible exception of the chassis, will be highly reusable in Phys 124.

- 4 IR Sensors: We used the *Pololu Carrier with Sharp GP2Y0D805Z0F Digital Distance Sensor 5cm* sensor in class, during lab four. Its range was from 0.5 cm to 5cm. There is another sensor from the same family *Pololu Digital Distance Sensor 15cm*, which has a range of 0.5 cm to 15 cm. This sensor would be appropriate for our project as we would like to make a maze that maps to the tiles on the lab floor which are 1 square foot.(because 1 foot is $\approx 30cm$.) This second part is of comparable price (Unit price(15cm) (US) 1 for 12.95 5 for 11.91 each 25 for 10.96 each), compared to Unit price(5cm) (US) 1 for 10.95 5 for 10.07 each. This sensor can also be utilized for lab 4, but more importantly, it has the appropriate range for sensing walls in our project.
  Pololu Digital Distance Sensor 15cm Specs:
  Operating voltage: 3.0 V to 5.5 V
  Current consumption: 30 mA (typical) when enabled, 0.4 mA when disabled
  Maximum range: 15 cm (6 inches)
  Minimum range: $<$ 5 mm
  Minimum update rate: 95 Hz (10.6 ms period)
  Field of view (FOV): 15° typical; can vary with object reflectance and ambient conditions
  Output type: digital signal (low when detecting an object, high otherwise)
  Dimensions: 0.85in $\times$ 0.35in $\times$ 0.122in (21.6 $\times$ 8.9 $\times$ 3.1 mm)
  Weight: 0.014 oz (0.4 g)

- L298N Motor Driver: We used this motor driver over the adafruit motor driver due to only having to use two 12V DC motors and the need for more pins on the arduino.
  Both of the motor leads of one dc motor can be plugged into the two ports on the sides of the motor driver, the order in which they are plugged in will only change the direction that the motor spins.

- Chassis: For this MicroMouse, *Premium Robot Tracked Car Chassis Starter Kits with 2pcs 12V DC Motor, Caterpillar Moving Robotic Tank Platform with 2pcs Tracks for Arduino Raspberry Pie Microbit Python DIY Steam Remote Control RC Toy*
  Chassis Specs:
  Tank Chassis: Tank chassis is different from car chassis, which has the track and wheels. By this way, tank chassis can move smoothly in the complex environment, like grassland, sand, and and many little stone. If the car, will be easy to rollover. But the tank chassis has much more width wheel, so can easily pass these cases.

  Metal Panel: This smart robot car chassis kit adopts metal panel design, with 2 plastic track and 4 wheels. But the price is relatively affordable.

- Two 12V DC Motors: With respect to Motors, we prized accuracy and the ability to perform a full rotation over speed or precise angular positioning. Since we used a tank chassis with trends, DC motors were an easy choice for controlling the movement.

- IMU/gyroscope: We used the HiLetgo 3pcs GY-521 MPU-6050, not because of any particular reason, it was just what we had available.

- In the event of adding Time of Flight Sensors, we should probably use the *HiLetgo VL53L0X Time-of-Flight Flight Distance Measurement Sensor Breakout VL53L0X ToF Laser Range Finder for Arduino.*

4 Sensors

The basic MicroMouse will be equipped with a range of sensors essential for effective maze navigation and obstacle avoidance. The selection of sensors will balance functionality with cost-effectiveness, keeping in mind that this project's primary focus is on providing a foundational experience in robotics.

- Infrared (IR) Sensors: The infrared sensors work based on sending out infrared light and detecting how much is bounced back. The further an object is away the less light that bounces back. The IR sensor then reads high or low values based on the amount of infrared light bounced back relative to a threshold value. Which in turn determines the detection range of the sensor. The IR sensors in turn return a boolean value, which by itself does not provide that much information.

- Gyroscope/Compass:
  Although we did not consider the utility of a Compass in our initial understanding of this project, we found that navigating in any sort of intelligent way using just the IR sensors was

neigh impossible. So, we included a Gyroscope which was able to give us the compass directions that were integral to the function of our final iteration.

- Time of Flight Sensors:
  While our project did not include these, we believe in order for this project to work seamlessly, it would be very helpful to include these, because being able to accurately measure the distance to objects around the micromouse provides significantly more information than just what IR sensors can do.

The sensors integrated into the MicroMouse were selected with the intent to provide a basic yet functional setup for maze navigation. This selection provides a solid foundation for understanding sensor data processing, which is a fundamental aspect of robotics and autonomous navigation.

5 Mechanical Considerations

- Chassis Design: We used the box the metal chassis came in as a container for the battery, Arduino, and other components.
  Note that the disconnected red and gray wires are for the battery pack that powers the IR sensors, so that they can be easily turned off and on just by connecting/disconnecting the wires.

- Wheel and Motor Assembly: The tank trends were stable, but they do pose a variety of problems. The first of which is the slipping problem. The trends could slip pretty easily on tile floor, this wasn't much of a problem when going straight, but when turning it would end up being very easy slide a bit further after the motors had already stopped. This wouldn't be that large of error most of the time for most purposes. But even a 4 degree overshoot from a perfect 90 degree is a death sentence when trying to solve a maze accurately. We did tests on carpet, but that ended up being on the opposite end of problems. since the carpet was too rough for the rover to be able to move at low speeds. So it would overshoot turns on carpet too, while also requiring significantly higher motor speeds just to move. In the end we stuck with low speeds on tile, and solved the slipping problem with a software remedy.
  There was also another issue that when we got the trends, initially they were too loose when wrapped around the wheels. We then had to take off a about 6 trend links in the end, before it fitted correctly.
  The motors and chassis also don't fit that well, even though they were sold together. As can be seen in the underneath view of the rover, the 12V motors are too large to fit right next to each other, so the motors are always anti-symmetric to each other. This posed a variety of minor problems. For instance the trends behave slightly differently going each motor wheel, when the motors are spinning in the same direction. This causes some error in the movement of the rover. The anti-symmetric position of the motors is convenient in one way. The center of mass relative to the motors can easily be placed at about the center of the metal chassis frame. This is why the box frame is shifted up a bit so that it hangs over the front, since we placed the 12V battery pack at the back of the box, so we are simply balancing out the weight. It is worth noting that the free spinning wheels are kept in place with a double-nut configuration.
  The the only thing the tank trends are good at is turning, and they aren't accurate enough for the purposes of MicroMouse, so it would be better to use omni-directional wheels, which have all the upsides and fewer of the downsides.

- Motor Power/Torque: The two 12V DC motors we received were defiantly not exactly the same. The left motor was noticeably weaker than the right motor. Although some of this issue may have been because of inconsistent power supply through the motor drivers we used. There also wasn't a good way to calibrate the motors since the voltage to torque output was not a purely linear relationship. However in the end we had our motors set to different definite speeds(we had the left motor set 2-2.5 times the right motor) for making the maze algorithm work more consistently.

- Turning Radius: The tank trends effectively allow a 360 degree turn radius. However there typically is some movement when turning, as the rover won't turn perfectly in place. There isn't a good way to get around this, but making smaller turns limits this error.

- Enclosures for Electronics: All the electronics are inside of or attached to the outside of the box that is zip tied and screwed down to the metal chassis.
  The Arduino is attached with screws and nuts on the inside wall of the box. The screws can then easily be tightened from the outside.
  The 12V battery pack in the back of the rover is easily secured with tape and its own weight. Likewise the other battery packs use either zip ties or tape to keep them secure to the box.
  The IR sensors are just kept attached to the box with a screw and nut, they tend to be a

little bit more difficult to tighten because of the nut being on the inside of the box.

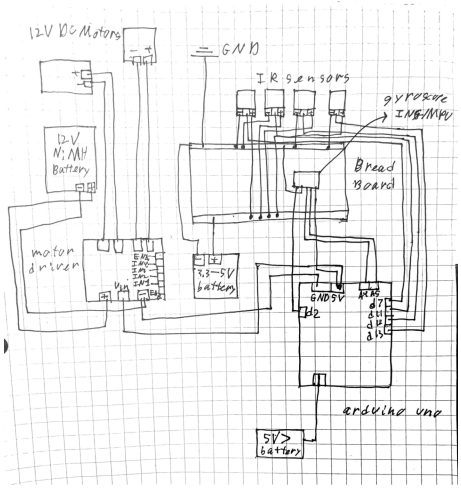
- Customization Potential: The current box container design is very convenient for customization. Since you can always attach a new sensor to the side of the of box and get the wires to where they need to be in the interior of the box. We went through quite a few modifications and redesigns the component layout. At one point the rover had a bumper made out of relatively thin aluminum bar. The most obvious evidence of prior customization is the variety of holes in the box there were previous places for screws/wires to go through.

The MicroMouse's mechanical design should prioritize a balance between simplicity and functionality. It should be easy to assemble, disassemble, and adapt, making it an ideal platform for educational purposes while providing a foundation for future enhancements or more complex mechanical designs as the project progresses.

6 Electrical Considerations

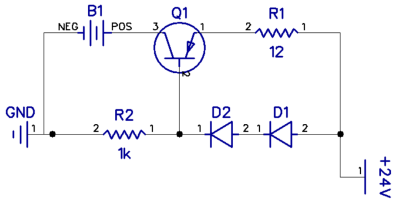
We used an arduino uno with an ATmega328P microcontroller chip, as the the brain of the MicroMouse. For more efficient designs it would be better to make your own PCB with all the necessary components(microcontroller, IMU, motor driver, etc). However the arduino uno is pretty easy to use, the only problem is getting it to integrate well with all the other components being used.

Here is the general block diagram for this project:



Many of the specific connections are also explained in the component section.
Power Supply:

- Battery Selection: The 12V NiMH battery we used provided enough power to run the motors for quite awhile before it needed to be recharged. Using a rechargeable battery with a relatively high amount of milliAmp hours is very useful for this project that will require a lot of power consumption over the testing and usage phases. Since we didn't have a battery charger, we had to make our own circuit for charging the batteries:
(Image from circuit basics, link in references)



This circuit can be used to charge the 12V rechargeable NiMH battery packs pretty easily. We supplied about 16V from a power supply and monitored the current going in the battery which we kept at about 60mAmps, and monitored the voltage across the battery using two multimeters. The battery will charge pretty slowly so expect the to spend a few hours charging it if you want to use the motors for a sizable length of time.
We also used one 9V battery for powering the arduino, along with 3 triple batteries for power the IR sensors. We used the extra battery pack for the IR sensor since the arduino ended up being able quite inconsistent in being able to supply power to the 4 IR sensors and the gyroscope, and the voltage in for the motor driver.

Components:

- L298N Motor Driver:
ou need to plug in your power source into the +12V pin and along with the negative lead of your battery pack into the GND of the motor driver. The GND on this motor driver doesn't work properly so also plug a GND from the arduino into the GND of the motor driver. Also in order for this motor driver to function properly you need to put in 5V into voltage in port(this is the blue port that is the last one to the right of the 3 blue ports in the picture above). Then 4 digital pins and 2 PWM pins from the arduino need to be connected to the remaining black pins. The specific pin layout we used is in the code, but the ENA and ENB should be connected to the PWM pins. This motor driver also doesn't support the motor libraries that the adafruit motor shield does, so we had to write our own control code. Which was relatively simple, although inconsistent torque output of the motors does become a more noticeable problem when this done. This motor driver also comes with a heat sink, so we never had any problems with heat genera-

tion.

For making the wire connections to the IR sensors, it is best to solder wires to the Vin, GND, and Out connections. The out connection should go to one of the arduino's digital pins, as the IR sensors give high/low values. Its best to keep the battery supply power to the sensors at about 4.5 volts so that the sensors can run for a fairly long time before the battery drops below the 3.3 volt threshold. We also made sure to have a convenient way to unplug battery pack, so the IR sensors aren't on when the rover isn't in use.

We used the Vcc,GND,SCL,SDA, and INT pins on the gyroscope. You plug 5V into the Vcc pin to power the gyroscope, we did this from the arduino 5V pin. The SCL and SDA should go to two analog pins on the arduino, while the INT pin should go to a digital pin.

The arduino is the most straightforward component. It works mostly like it should, which is good considering it is the core of our project. It has been mentioned that we used a 9V battery for power the arduino. We used a 9V battery since it would be able to power the arduino for fairly long time. Its safe to due this since the recommended input voltage on the arduino uno is 7-12V, while its operating voltage is 5V. We only had to change out the 9V battery once over the two weeks while we were using a battery pack to power the arduino.

7 Interfaces

The MicroMouse project will include various interfaces to facilitate programming, control, and data analysis, with a particular emphasis on using an Arduino microcontroller for the robot's control system.

Programming Interface:
The primary programming interface is essential for uploading code to the MicroMouse's microcontroller, which in this case is an Arduino. This interface will allow developers to modify and improve the robot's software:

- Arduino IDE: The Arduino Integrated Development Environment (IDE) will be the primary software tool for programming the MicroMouse. It is a beginner-friendly platform that supports C/C++ programming and provides a user-friendly interface for code development, uploading, and debugging.
- USB Connection: A USB interface will be provided for connecting the Arduino to a computer running the Arduino IDE. This connection enables developers to upload code to the Arduino's microcontroller, facilitating code development and testing.

Control Interface:
The control interface allows for remote operation of the MicroMouse and real-time adjustments during testing and experimentation, with the Arduino playing a central role:

- Arduino Control Logic: The Arduino microcontroller will run the control logic responsible for executing the MicroMouse's movements and making decisions based on sensor data. Developers can upload and modify the control code via the Arduino IDE.

Data Interface:
Monitoring and analyzing sensor data is crucial for understanding the MicroMouse's behavior and improving its navigation algorithms. The Arduino is instrumental in collecting and processing sensor data:

- Sensor Data Processing: The Arduino processes data from the IR sensors. It converts analog sensor readings into digital information that informs the robot's decision-making process.
- Data Logging: The Arduino can store sensor data locally in variables or arrays for real-time analysis and debugging. Data can be logged during maze runs and retrieved for post-run performance evaluation using the Arduino's memory storage capabilities.
- For the Gyroscope, we use code from a library for the MPU-6050 in order to get the data from the Gyroscope.

By using the Arduino as the central control system for the MicroMouse, the project ensures ease of programming. While also allowing users to implement complex features as part of the expansion options. The Arduino IDE, USB connection, and data processing capabilities of the Arduino provide a strong foundation for the project's success.

8 Software

The software component of the MicroMouse project is a critical aspect of achieving autonomous maze navigation. The following sub-sections provide a more detailed breakdown of the software components and their functions.

Maze Navigation Algorithms: The basic MicroMouse will implement fundamental maze navigation algorithms, primarily wall-following and random exploration techniques. These algorithms are essential for autonomous maze traversal. More advanced maze algorithms are listed in the expansion section.

- Right Wall-Following Algorithm: This algorithm will instruct the MicroMouse to follow the walls of the maze, using data from the IR sensors. The robot will(try to) maintain a consistent distance from the walls while making decisions to navigate the maze. The specifics of how we implemented this algorithm goes like this:

  First: If the back right IR sensor is triggered, there is a wall directly to the right of the MicroMouse, so we go forward.

  Second: In order to keep going at a consistent from the wall, the MicroMouse needs to keep going in approximately a straight line. We can use the angle readings from the gyroscope to initialize the the MircoMouse with a set heading and then take more angle measurements to see if the MicroMouse is deviating from the set heading. We can then check to see if this error is greater than some threshold(we used 2 degrees of maximum allowed error) and then adjust the rover to head back towards the correct heading.

  Three: To adjust back to the correct heading, since we are using tank trends, you can do two things. The first is to adjust the speed of just one motor so that one side spins faster than the other and will eventually lead back to the correct heading. This approach is best done with a PID. However, because the rover is still moving forward a little bit in this adjustment phase, by the time the rover is once again pointing in the correct direction. There is a good chance it will be on a parallel line to its original heading. Which there isn't a good way to correct for with the sensors we have. So the only way to get around this, is to make the error correction of the PID very aggressive. such that an error of 1-2 degrees will immediately be corrected by the the motors being ramped up/down appropriately. This then creates the problem that if the error is too big, the PID terms will grow far too large and cause the motors to do unpredictable things. This approach is also entirely dependent on the gyroscope readings being reliable all the time, which isn't realistic from what we found with the gyroscopes we tested. We therefore decided to go with the second approach of using a TurnAdjustment function, that will stop the rover when the error is to big and then make small micro turns by turn the motors on for about 40ms(this value works the the motor speed we used, higher motor speeds may need lower active times) and then immediately turning them off again. Then we just loop and do micro left and right turns until we are pointing in the right direction again. This does have the slight affect of moving onto parallel lines like the first method, but the affect is basically negligible in this case.

  Four: When the back right sensor is no longer triggered we know that there is a right turn there, so we now attempt to do a 90 degree turn. However, due to trend slipping and the general overshooting of the rover, we just about always overshoot the 90 turn by 10 or more degrees. So we change the current heading angle by 90 degrees in the appropriate direction, and do a TurnAdjustment, that will take us back to an almost perfect 90 degree turn.

  Five: Now that we have made the perfect 90 degree turn, the rover should be lined up with maze and reading to go forward. But it isn't, the back right sensor will be not be triggered since it is now pointing in the direction of the path where the rover was moving from. So we instead move forward in small time steps while periodically doing a TurnAdjustment until the back right sensor is triggered again. So now the rover has cleared the turn completely and keep going until it reaches another obstacle.

  Six: Now what if the rover is at a dead end or at left only turn? We can employ a similar tactic to the right hand turn. But now we use one of the forward IR sensors to see if anything is in front of the rover. So if both the the front sensor and back right sensor are triggered we can just turn left by 90 degrees, update the current heading and do turn adjustment to make sure we did a perfect 90 degree turn. If there is now nothing in front of the rover, the rover will go forward. If there is again something in front of the rover we will turn left again and be pointing in the same direction that the rover came from but will be attached to the other side of the maze wall. Thus we have surmounted the left turn and dead end situations. It's worth noting that the front sensor isn't placed on the front of the rover, because other wise the rover would detect the front walls too earlier and attempt to turn left only to have the right sensor no longer detecting anything, so it will then try to turn right, and then left again and so on. So moving the front sensor closer to the back of the rover gets around this issue. Since the rover will only

trigger the left turn when it is close enough to the wall for the right sensor to then be in range of the wall after turning.
Given enough time this algorithm will be able to solve any right handed solvable maze. The algorithm also assumes that the rover isn't perfect, if it did, the rover would never be able to solve any maze.

Sensor Data Processing:
Efficient processing of sensor data is crucial for accurate decision-making. The software will include:

- IR Sensor Data: The IR sensors read either high(nothing is there) or low(something is there), so the IR sensors can only give information as a boolean value. the limited IR sensor range of 15cm. This greatly limited what we could do in the maze, because of the relatively small margin of error. Since if the rover was perfectly centered in the maze it would only take a couple centimeters of movement in one direction to to deactivate the sensors and throw us into a case where the rover no longer thinks it is next to a wall. This introduces a lot of inconsistently, so a lot of our code and mechanical considerations was in an attempt to remedy this problem. If we had some IR sensors with a range of about 30cm. Then we could of use the short range IR sensors to detect when we are too close to a wall, and the long range ones to detect if there is a wall. Which provide significantly more information than using just one range of IR sensor.

- IMU Data: We used the gyroscope in an IMU to measure the direction of the rover, since we only needed one angle measurement this basically functioned as a compass. With the orientation of the gyroscope when hooked up to the arduino, we used the yaw measurement. To actually be able to use the gyroscope however, you can either attempt to write your own code from scratch(not recommended), or use some pre-existing libraries. The sources for the libraries we used are in the GitHub links in the references. Once the code is working, the gyroscope will then output data in an array of length 3, where the entries are yaw, pitch, and row. The gyroscope will always have some small perturbations and errors in its measurements so we used a simple average to clear away some of the noise. That worked well enough most of the time, however sometimes the gryoscope would output a string of bad readings(something like over 100 degrees off) for no discernible reason. We could only chalk it up to some kind of divine retribution(this was a very rare problem over the course of testing). In this case the gyroscope effectively timed out, so it did an automatic reset and begun its initialization process again. It's also worth noting that the gyroscope uses I2C communication with the arduino, but the libraries take care of the specifics.

Testing and Debugging Tools:
To facilitate the testing and debugging process, additional software tools will be developed, including:

- Simulation Environment: A simulated maze environment where the MicroMouse can be tested in a controlled setting before attempting real-world maze trials. More specific information is available here https://www.nathanlam.com/projects /MicroMouse-simulator. We didn't end up using this since we spent a great deal of time getting the mechanical aspect working. We also didn't test algorithm's more advanced than just hug the right wall.

- Logging and Visualization: Software for recording and visualizing sensor data and the robot's path during testing, aiding in debugging and analysis. Using serial readings is good enough for a lot of this project, but that requires the arduino to be plugged in to a computer, so being able to have the rover send that data remotely would of been a great convenience for testing.

The software for the basic MicroMouse project will be structured in a modular and extensible manner, allowing for future enhancements and upgrades in response to the project's expansion options. It forms the core of the MicroMouse's decision-making, navigation, and interaction capabilities.

9 Safety

While the robot is intended to operate in controlled environments and may not pose little to no significant risks, several safety considerations must be taken into account:

- Battery Safety: Rechargeable batteries used to power the MicroMouse can pose a risk if mishandled. To mitigate this risk, the project will ensure that the batteries are properly stored, charged, and maintained. The batteries especially the 12V ones shouldn't be shorted between their positive and negative terminals. The rechargeable batteries should also be charged at a rate far lower than the maximum charging rate, we aimed for about 1/20 the maximum.

- Heat Management: The electrical components of the MicroMouse, such as the motors and

motor drivers, may generate heat during operation. Adequate ventilation and thermal management will be considered to prevent overheating and reduce the risk of burns.

- Electrical Safety: Safety measures will be put in place to protect participants from electrical hazards. The MicroMouse will be powered by low-voltage batteries, minimizing the risk of electrical shock. Wiring will be appropriately insulated and secured to prevent short circuits.
- Secure Maze Design: The physical maze used for testing the MicroMouse should be designed with safety in mind. We don't want the MicroMouse potentially falling off a ledge.

Emergency Stop: The MicroMouse will be equipped with an emergency stop mechanism to halt its movement in case of unexpected behavior. This ensures immediate control if the robot behaves unpredictably.

10 Expansion and De-scope options

The fundamental goal of this project is for the MicroMouse to traverse the maze, even if it uses a rather simple algorithm. Such as random traversal to eventually make it through the the maze. This leaves the opportunity for expansion options, to make the project more complex and sophisticated.

Expansion options:

- Mapping and Exploration: Develop a mapping feature that allows the micromouse to create a map of the maze as it navigates. This advanced capability can be used for maze-solving, making the micromouse more efficient.
- Additional Sensors: While the basic micromouse's design incorporates the sensors mentioned above, future expansion options may include other types of sensors, such as ultrasonic sensors for more precise obstacle detection, light sensors for detecting ambient light conditions. The decision to integrate additional sensors can be part of the expansion phase of the project.
- Vision-Based Navigation: Integrate a camera and computer vision algorithms to enable the micromouse to recognize maze patterns and intersections, providing a higher level of accuracy in maze traversal.
- Depending on the specific chassis design of the MicroMouse. It may be possible to use a fan/vacuum attached to the underneath of the chassis to essentially increase the friction. As the fan/vacuum will suck the MicroMouse into the floor, allowing it to travel at far higher speeds. The down sides of this are the increased weight and power consumption. But for short fast maze traversals these variables are negligible.
- Multi-Maze Compatibility: Make the micromouse compatible with various maze layouts or changeable maze configurations. This expansion allows for more diverse testing scenarios.
- Dynamic Maze Adaptation: Implement adaptive algorithms that can adjust the micromouse's behavior based on the maze's layout and obstacles, making it more adaptable to different situations.
- Multi-Robot Collaboration: Develop the micromouse to work in collaboration with other micromouse robots, enabling cooperative maze navigation or competition.
- Bluetooth Control: An optional Bluetooth module can be integrated with the Arduino. This module allows for wireless control from a mobile device or computer, enhancing the micromouse's flexibility during initial testing or debugging. It enables real-time adjustments to the robot's movements, making troubleshooting and fine-tuning more accessible.

There is also the possibility of having to tone down some of our initial goals for the project. These are mostly concerned with hardware over software. As getting the software to a minimal functional product should be achievable.

De-scope options:

- Basic Power Management: Simplify the power management system, such as using disposable batteries instead of rechargeable ones, to reduce the complexity and cost of the robot.
- Noisy Sensor Data: Accept less accurate or noisy sensor data and forgo calibration or advanced filtering techniques, simplifying the sensor subsystem.
- Manual Control Option: Remove autonomous navigation capability and focus on a remote-controlled micromouse, effectively making it an RC robot.
- Fixed Chassis: Use a fixed chassis design without modularity or the ability to easily adapt to different scenarios, reducing mechanical complexity.
- Minimalistic User Interface: Provide only basic LED indicators or a simple command line interface for controlling and monitoring the micromouse.

11 Lessons Learned

11.1 Overview

While there were many issues within making this project work, like issues with motor shields, motors, batteries, sensors, treads slipping, direction, and integration, the largest challenge was that of making the device fully autonomous. The reason why having this project function autonomously was so difficult is that having the device complete almost any autonomous task with success, means that all of the aforementioned issues must be solved, or mitigated to within an acceptable error bound such that our errant movements would not compound over the length of our task and lead to game-breaking amounts of error.

11.2 Instructions

In order to accomplish our stated mission of making a bot which could solve a maze autonomously, we employed a general set of instructions which the bot would have to be able to perform in order to have the capability to solve a maze.
The first of these Instructions was for the bot to be able to move in a straight line.
The second of these Instructions was for the device to be able to turn 90 degrees either right or left without loss, i.e. if the bot were to turn less or greater than 90 degrees, the maze solving attempt would be all but ruined.
These were two instructions that if paired with appropriate sensing and algorithm would be able to solve any maze.
By the end of our project, we were able to have these two instructions work with an acceptable(not game-breaking) amount of error.

11.3 Sensors

Firstly, I believe that we underestimated the variety of sensors that would have been appropriate for this project. As mentioned in the Motivation and Overview, many MicroMouse projects use Time of Flight sensors, which would have been extremely helpful in making our project work both more seamlessly and robustly. This is because these sensors would help us overcome one of the main issues that were inherent in the construction of this project, which was that the IR sensors were not as potent of a source of information as was necessary to correctly understand the positioning of the Bot.

12 References

ElectronicCats. “ElectronicCats/MPU6050: Mpu6050 Arduino Library.” GitHub, github.com/ElectronicCats/mpu6050. Accessed 16 Dec. 2023.

Jarzebski. “Jarzebski/Arduino-MPU6050: Mpu6050 Triple Axis Gyroscope & Accelerometer Arduino Library.” GitHub, github.com/jarzebski/Arduino-MPU6050. Accessed 16 Dec. 2023.

Lambert, Graham. “A Guide to Building Battery Chargers.” Circuit Basics, 30 Nov. 2021, www.circuitbasics.com/battery-chargers/.

A Appendix A