Micromouse Competition

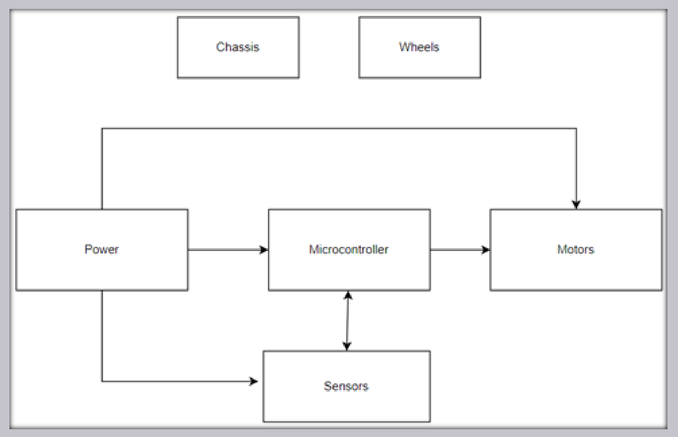
Brayton Niccum, Nicholas Rogge, Spencer Seim
Advisor: Sean Carroll, PhD
Trine University
One University Avenue, Angola, Indiana 46703

Trine University
Electrical and Computer Engineering

Introduction:

Over the last two semesters, we have worked on a micro-mouse robot for use in the NRC micro-mouse robotics competition. A micro-mouse is defined as a robot that fits within a 7” cube, and is capable of solving a 10x10 tile maze, where each tile is 10”x10”. The design consists of a few main parts. First is the distance sensors which are used to determine where walls are and how close the robot is to these walls. Next, we have the gyroscope which is used to drive the robot straight down robot corridors as well as help turn the robot. There are also quadrature encoders which are used for determining linear distances traveled throughout the maze. The robot is driven by two great motors in the rear and a centered ball caster in the front. The software takes data from the sensors described above and determines what decisions the robot is allowed to make and then uses that data to create data for the motors to drive the robot through the maze. The budget for this project was $600 of which we used about $400 including lodging for the robot competition. The team consists of three members, one that is studying electrical engineering and the other two are studying software engineering.

Figure 1: Basic Design of Micromouse Robot



Rules and Specifications:

The competition judges grade on two bases, time-to-finish if any robot successfully solves the maze, and total tiles traveled if fewer than 3 do. Each team is limited to 10 minutes with the maze, during which any number of runs can be completed. The robot must also be fully autonomous.

The maze itself consists of 100, 10”x10” tiles. These tiles are arranged in a 10x10 grid. The walls themselves are 0.75” thick, giving us an effective usable tile space of 9.25”x9.25”. The walls of the maze are painted white, and the floors painted black. The colors cannot be guaranteed due to wear from prior competitors however, and as such should not be relied upon.

Each robot starts in one of the four corners of the maze, selected at random. The end point of the maze is the four center squares. Timing starts after leaving the starting square and ends when you reach the end squares. More runs can be completed after this point, though a 30-second penalty will be applied. Multiple paths to reach the end squares are to be expected.

Using this information, we stated that the worst possible route would consist of 50 necessary tiles to reach the end of the maze. We also specified a maximum of 1 minute and 30 second time limit to finish this worst possible route for use in required speed calculations. Furthermore, a maximum weight was not specified by the competition rules, and as such a maximum weight of 3 lbs was decided upon for use in torque calculations.

Standards included making sure that our parts were mostly rohs compliant. Coding standards including making sure that all classes had their own header and source files. Limited use of globals when possible and specific naming conventions such as using m_ in front of class member variables and using camelCase for member functions.

Sensors and Code Design:

All of the software was written in an object oriented fashion using the C++ programming language. We used an extension to the Visual Studio Code IDE called Platform IO to program our Teensy 3.5 microcontroller consisting of a 32-bit ARM Cortex M4 processor running at a clock speed of 120 Mhz. Platform IO allowed us to program the microcontroller by just directly accessing registers, but it also supported the use of the Arduino framework to allow us to use libraries so we could focus more on the software related to our sensors and motors rather than reinventing things such as an I2C driver, though many of these still required some manual configuration due to the microcontroller having many more peripherals and features compared to standard 8-bit arduino board

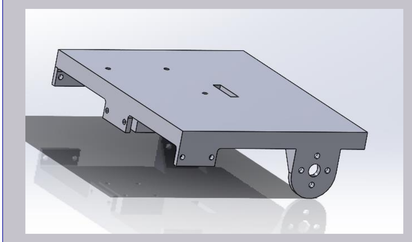
Distance Sensors
The distance sensors are arguably one of the most important sensors of this entire product. Distance sensors are what allows the robot to see the walls that are inside the maze so that it can avoid hitting them or make decisions based on what they see in general. The distance sensor we choose is the VL53L0X - Time-of-Flight Distance sensor made by STM.

Linear Distance and Rotational Sensors
For other sensors a quadrature encoder and the MPU6050 which contains an accelerometer, gyroscope, and temperature sensor. The first sensor to discuss is the MPU6050 more specifically the gyroscope portion. For this sensor we utilized the mpu6050_light library by rfetick on github.

Motor and Motor Driver Software
The software for the motor is again its own class much like the other components thus far and is concerned with sending a desired PWM (Pulse Width Modulation) signal to the motor driver. Each instance of the class has three main functions The first is to set up a timer with the appropriate resolution which in this case we use a TOP value of 255 making it an 8-bit timer. The next function of the motor is to update the PWM duty cycle or in other words the width of our logic high part of the signal. The final function is the update method which is where the motor grabs a new duty cycle from its respective queue and updates the proper register regarding the width accordingly.

Queues
The robot’s software relies almost completely on queues for moving data between sections of the software and it also serves as a way to decouple our software. Decoupling our sensors from the system and logic software is important because if we were to change a sensor, we have very little changes to make to the overall software. If the software wasn’t as decoupled, changing to a different type of sensor could require a very large software change even though it's most likely just a change related to how the data is gathered. In this case we have 3 different types of queues one for the gyro, TOF, and motors.

Driving Software
The driving software is the software that takes in all of the sensor data and using logic it can turn that data into a duty cycle to send to each motor for driving. This is all done through a class called Motor Controller. This class has a few main functions. First it will grab data from all of the sensors and save it to private memory local to the class. Next the class does checks and calculations to determine if the robot is driving straight or not as well as save the information that informs whether the front, right, or left sides have openings or not. Next the class checks if there were any openings and if there is more than one decision a random number is generated. The number is always generated and analyzed in a way that allows for an even probability of going any possible direction that is available. If there is not more than one option available, the software will only take that option. The possible options in that case are continuing straight or turning around at a dead end. The motor controller also contains a state machine. This state machine is there to ensure that only the correct logic is executed for a given state. The states in this machine are Start, Straight, Stop, TurnLeft, TurnRight, Backwards, and SlowForwards.



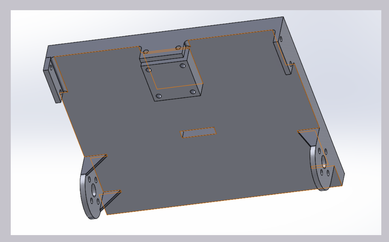


Figure 2: The 3D Design used for the chassis.

Figure 3: Robot in maze at competition

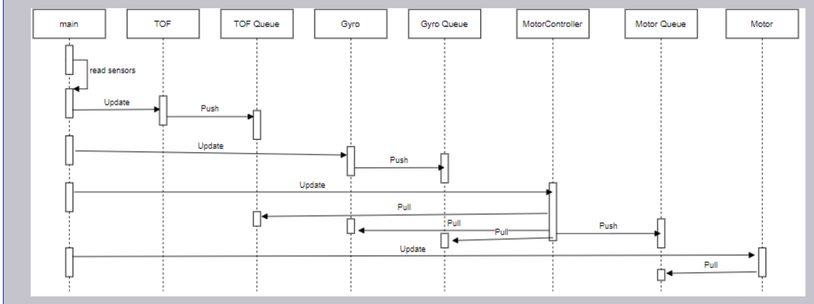


Figure 4: The sequence diagram of the Micromouse

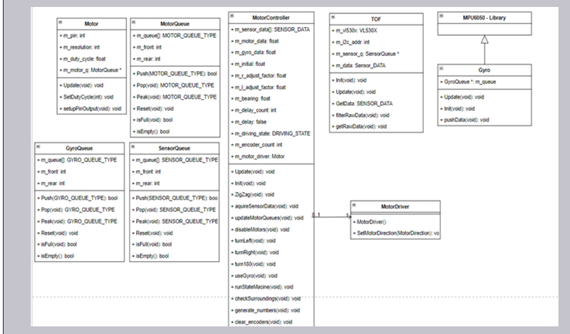


Figure 5: The software Class Diagram

Conclusion:

The purpose of the micromouse robot was to compete in the National Robotics Contest with a functioning robot. We achieved first place in the Post-Secondary part of the micromouse competition. Our robot can successfully complete a maze but could use some improvements with its turning logic.

Acknowledgements:

We would like to extend a special thank you to:

- The Fawick Extended Family
- Alex Menelli
- EduEverything.inc for hosting the National Robotics Contest
- Dr. Sean Carroll