Table of Contents

1. Abstract1
2. Introduction2
3. Design3
4. Hardware5
   1. Microcontroller5
   2. Motors and Drivers5
   3. Sensors7
   4. Battery and Power Supply 9
   5. Printed Circuit Board11
5. Software12
   1. Error Correction12
   2. Control System13
   3. Flood Fill Algorithm14
6. Conclusion17
7. Recommendations17
8. References18
9. Appendix19
   1. Cost Analysis19
   2. Figures20
   3. Tables23

**1. Abstract**

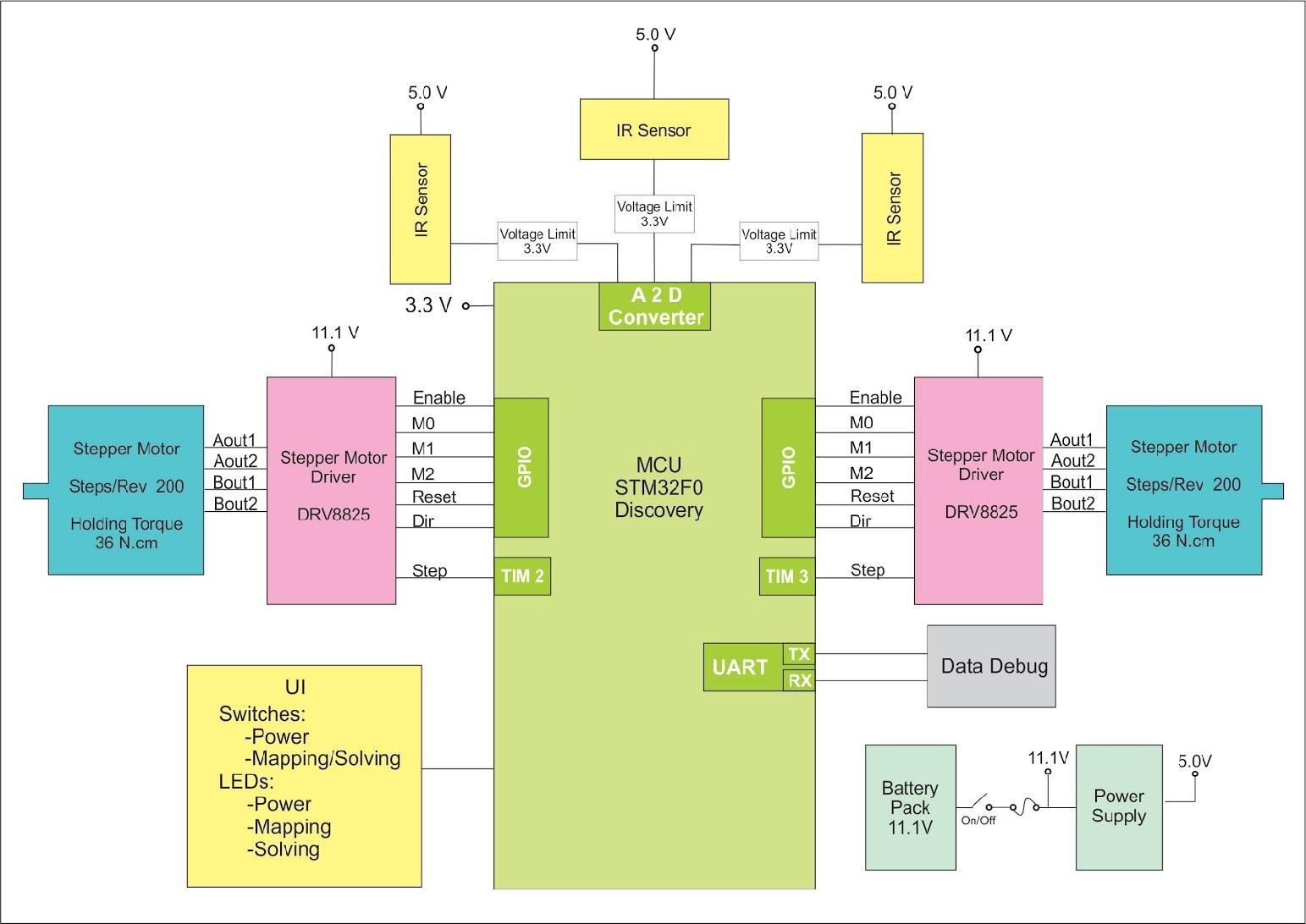
The objective of this project was to build a Micromouse, which is an autonomous robot able to navigate inside a maze and find its center in the shortest time possible. To accomplish this, AutoMouse’s design utilizes an STM32F0 microcontroller in combination with three infrared analog sensors and two stepper motors. The closed-loop feedback provided by the sensors allows the robot to perform error correction, so it can safely map and solve the maze at a high speed. The design has resulted in a reliable robot which is capable of reaching the center of the maze within the time limit.

**2. Introduction**

Micromouse is a competition hosted by IEEE since the late 1970’s in order to foster involvement of young engineers. In order to participate in this competition, the robots must meet certain criteria. First of all, the robot must be self-contained and autonomous. The total cost of the mouse should not exceed $500.00. The robot must not damage the walls of the maze in any way. All entries are allowed ten minutes to map and solve the maze [1].

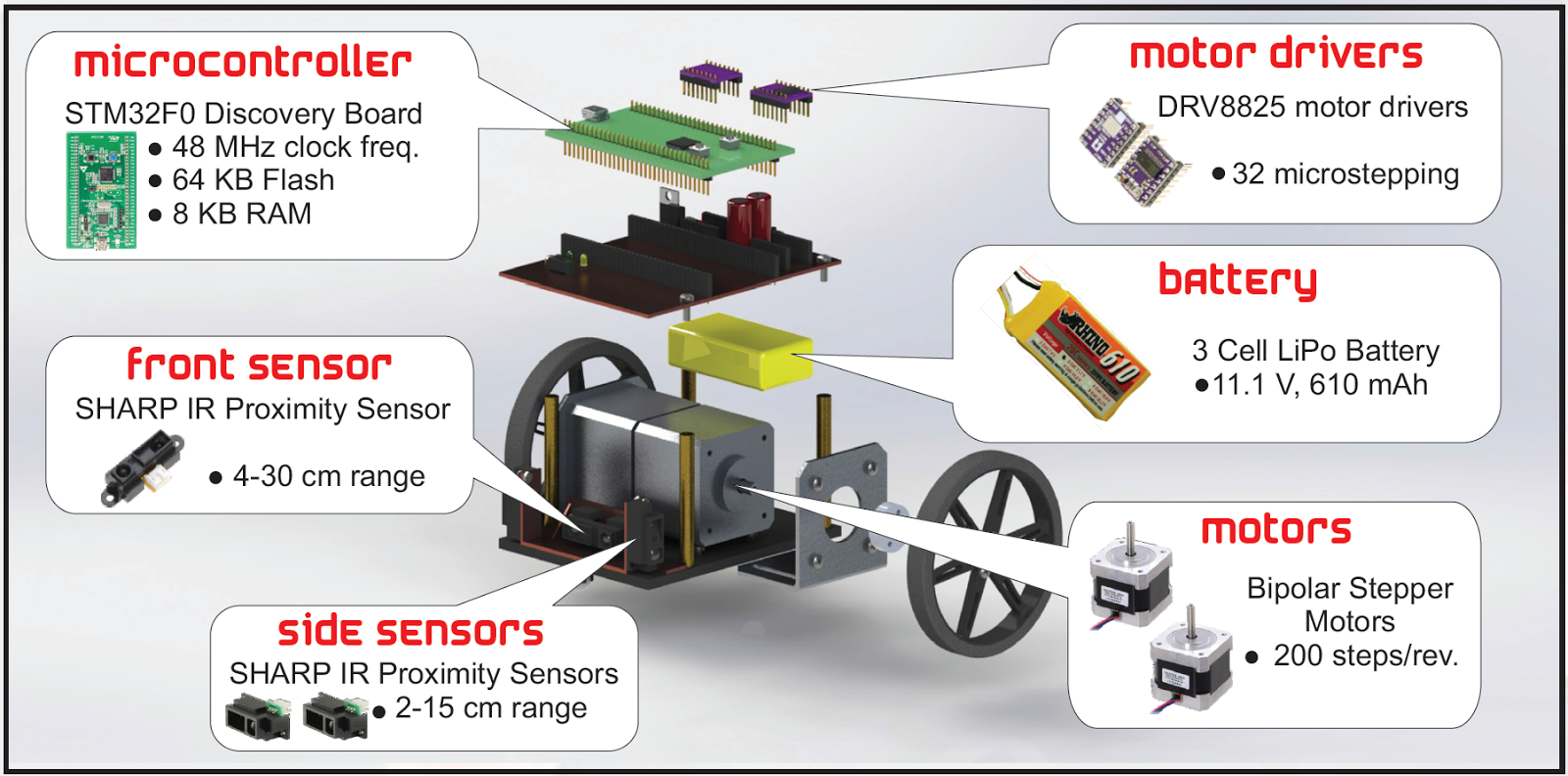
Team AutoMouse designed a robot that possesses unique characteristics which sets it apart from its competitors. For instance, the robot utilizes large bipolar stepper motors in order to achieve a high speed. To accommodate these motors, the chassis was designed in a compact square layout which results in a small turning radius. The addition of the SHARP infrared sensors provides the robot with accurate distance readings. Furthermore, the long range front sensor, as opposed to the shorter range side sensors, allows for the detection of walls from a far distance. To map the maze, the flood algorithm is used in order for the robot to keep track of its position and to find the shortest route to the center.

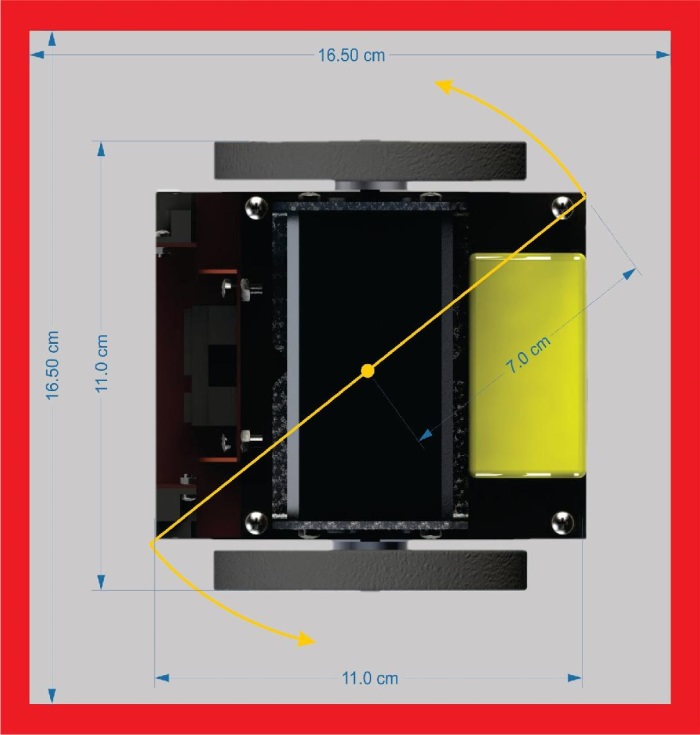
**3. Design**



***Figure 1. Block diagram***

Figure 1 shows the main aspects of the design; each block is interconnected in a different manner according to its particular function. The microcontroller directly maintains a constant relationship with the drivers, sensors, and power supply depending of the maze solving code and its specifications. Drivers control the speed and directional rotation of the motors depending on the type of commands received from the microcontroller. The motors operate at a frequency determined by the microcontroller. The sensors constantly report the distance of the robot from the walls to the microcontroller in order to detect obstacles and recognize the location of the robot inside the maze. To interact with the robot, a number of user interface peripherals were required such as switches to toggle power to the robot and to select between mapping and solving modes.

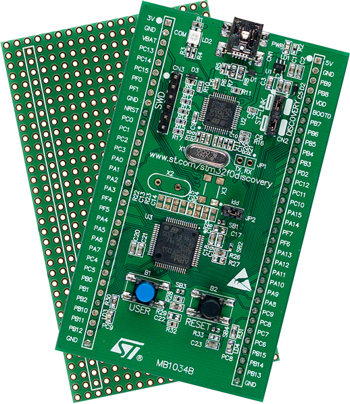
***Figure 2. Exploded view of the robot***

Figure 2 portrays the physical appearance and the components of the design. The sensors are located on the front of the robot in order to detect walls as early as possible. The large diameter size wheels are necessary in order to provide enough clearance for the ball casters that are on the bottom. By positioning the motors in the center of the chassis, the robot can achieve a turning radius of 7.0 cm as shown in Figure 3. This small turning radius gave robot the ability to maneuver the maze without colliding into the walls.

***Figure 3. Clearance of robot inside a cell***

**4. Hardware**

**4.1 Microcontroller**

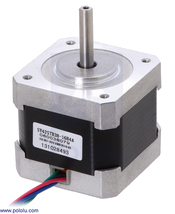
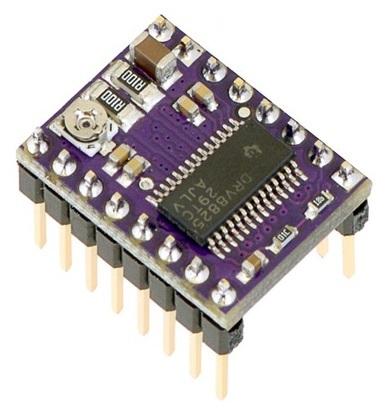
 The STM32F0-Discovery Board was chosen because it provides powerful processing capabilities and large memory storage for embedded systems. This microcontroller provides necessary peripherals for the robot such as: timers (TIM), analog-to-digital conversion (ADC), direct memory access (DMA), and general purpose input-output pins (GPIO).

***Figure 4. STM32F0-Discovery Board***

The signals that are sent to the motor drivers are generated by the timers which operate in output-compare toggle mode. The ADC and DMA are used to continuously transfer converted data from ADC Channel 1 to memory. The converted data from the three sensors is stored in memory after the DMA transfer in circular mode.

**4.2 Motors and Drivers**

In order to provide locomotion, the robot uses a bipolar stepper motor and a DRV8825 driver for each wheel shown in the images below. This motor was chosen due to its holding torque of 36 N∙cm, rated voltage of 2.8 V, 1.68 A rated current, and its ability to function within the range of 8.2 V to 45 V. Since the motor is needed to work nonstop between runs for at least ten minutes, it is important that the motor achieves the best control at the fastest speed when required. These drivers not only control the direction of rotation of the wheels, but also the stepping mode for the motors at a stepping ratio of 32 micro-stepping (6,400 steps per revolution). This mode gives the best resolution for higher frequencies, and also reduces the motor vibrations.

***Figure 5. Bipolar stepper motor Figure 6. DRV8825 driver***

The drivers require a minimum of 8.2 V for the motors and 3.3 V for the digital logic. As the frequency increases, the current passing through the driver decreases. To avoid overheating at lower frequencies or due to overuse, the drivers were tuned at a current rating of 1.5 A [2]. The configuration of motors and drivers is shown in the Figure I of the Appendix [3].

The bipolar stepper motors are driven by square waves of a certain frequency. As the frequency increases, the speed also increases. To obtain square waves, the code uses two timers called TIM2 and TIM3 for the right and left motor respectively.

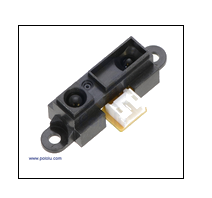
Since the motors and drivers are set in the 32 microstepping mode for better stability, the frequency needed to be varied in a controlled way in a range from 6 kHz up to 30 kHz to avoid missing steps and even unwanted breaks. The increment of frequency from the resting point of the motor to a constant and stable speed (6.144 kHz = 24.1274 cm/s) was acquired with a piece of code that changed the period of the square waves by a factor of approximately 983.04 Hz. It was decided to start the motors at about 3.78 kHz as a point when their shafts were no longer stationary. Using the procedure stated above, the speed of the robot can increase from approximately 14.85 cm/s to 24.13 cm/s in less than 0.423 ms. A similar process was approached to diminish the speed when the robot needed to stop. Table 1 in the Appendix shows the speed obtained for a range of frequencies.

The frequencies generated were sent from the microcontroller’s PB5 and PB3 to the right and left drivers’ STEP pin respectively. The 32 microstepping mode was established by sending 3.3 V from assigned microcontroller’s pins to the M0, M1, and M2 pins of both drivers (Mode 111). Table 2 in the Appendix describes the configuration for the different types of modes.

The output capture function integrated in the TIM2 and TIM3 peripherals is used to count the number of pulses being sent to the drivers. A total of 6,400 pulses are needed to complete a full rotation of the wheel; furthermore, to achieve a perfect 90° and a 180° turns, 3,850 and 8,000 pulses are implemented to a single wheel regarding the direction of turning in order to rotate in the same axis.

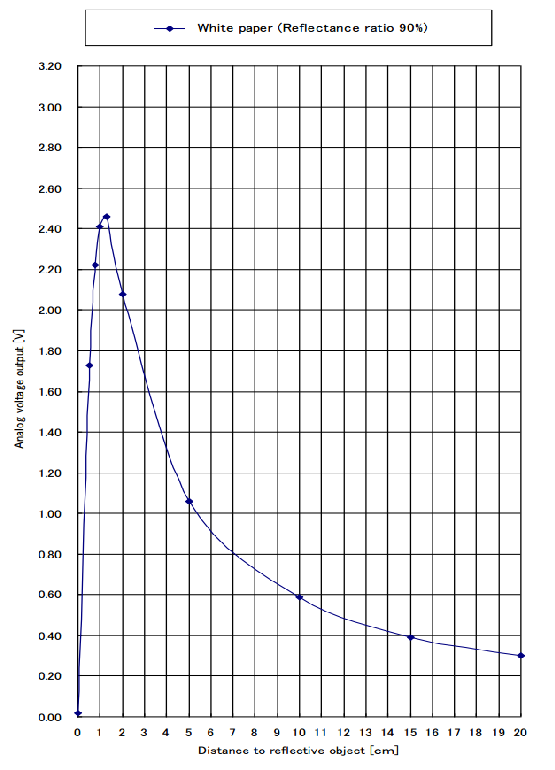
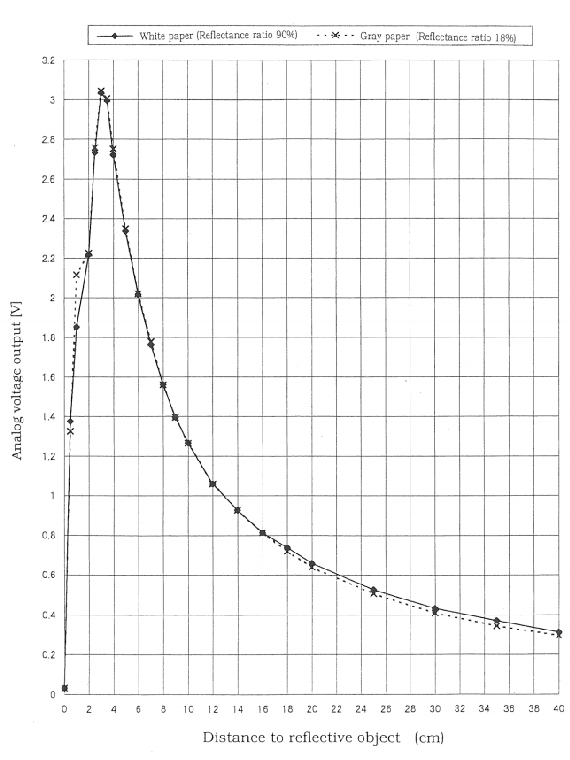
**4.3 Sensors**

A crucial feature of a Micromouse is the ability to determine if there are walls obstructing its path or if there are openings instead. It also must be able to indicate its proximity from the walls in order to avoid hitting them. This cannot be accomplished without eyes or any form of sensor. Multiple factors were taken into consideration when deciding which type of sensor would be implemented. One other type of sensor that was taken into consideration was the ultrasonic sensor. These sensors work with sound waves, but since the Micromouse maze contains small cells, the signals of an ultrasonic sensor could get muddled and produce the wrong readings. It was a known factor that the maze would be located indoors, so interference from the sunlight did not need to be accounted for, hence an ultrasonic sensor was not necessary. With various sensors and factors taken into consideration, the optimal choice for sensors was the Sharp IR Proximity Sensors. This type of sensor uses infrared light beams which bounce off of the walls to indicate the presence of walls, as well as proximity, through the method of triangulation, as seen in Figure II of the Appendix.

***Figure 7. Front sensor of 4-30 cm range Figure 8. Side sensor of 2-15 cm range***

Two types of Sharp IR Proximity Sensors were implemented in the design. The front sensor, seen in Figure 7, has a range of 4-30 cm which enables the robot to sense walls far ahead. The side sensors, seen in Figure 8, are of 2-15 cm range because the range only needed to be far enough for the robot to see walls and openings within the current cell. The ranges were chosen based on the 18 x 18 cm dimension of each cell because it was not ideal to have either too short or too long of a range. The sensors’ range being too short is not a problem since the robot takes up most of the space in a cell, leaving very little space between the side sensors and the walls. However, if the range is too long, it means that the sensors would have to be offset even farther into the chassis, consuming more space.

The correlation between the distance of the reflective object and the output voltage is seen in Figure 9.(a) [4] and Figure 9.(b) [5]. The farther the wall or reflective object gets from the sensor, the lower the output voltage becomes. However, when the distance of the object from the sensor falls below minimum range, the output voltage also decreases, but more drastically. These characteristics of the sensors are evident in Figures 9.(a) and 9.(b). Since the output voltage decreases in both instances, the sensor would provide inaccurate output when below the minimum range, thus the solution of offsetting the sensors to accommodate this problem was implemented.

***(a) 4-30 cm range (b) 2-15 cm range***

***Figure 9. Range correlation between distance of reflective object and output voltage***

In order for the output of the IR sensors to become manageable in code, the output voltages of the three sensors are converted to digital values using the analog to digital converter of the STM32F0 Discovery Board. Three channels of the analog to digital converter are used to accommodate all three sensors since one channel alone cannot handle inputs from three sensors simultaneously. The sampling rate is set to about 60 Hz. The ADC values are then stored into three sample variables, allowing ease of use of the sensors’ outputs in software. In terms of software, the sensors are used in mapping to detect walls and openings throughout the maze. The sensors are also used in the error correction function which allows the robot to avoid hitting walls by using thresholds with the sensor data.

**4.4 Battery and Power Supply**

The standard allotted time for a Micromouse competition is a maximum of ten minutes to map out and solve the maze with the fastest route possible. As such, it is required to use a battery that would be reliable enough to last at least those ten minutes, preferably longer for testing purposes, and ideally needed to be rechargeable so they could be reused. Lithium polymer batteries are the best bet at fulfilling all of the requirements for testing and the actual competition. These high capacity batteries are light, compact, and are easily rechargeable.

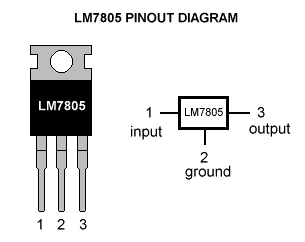
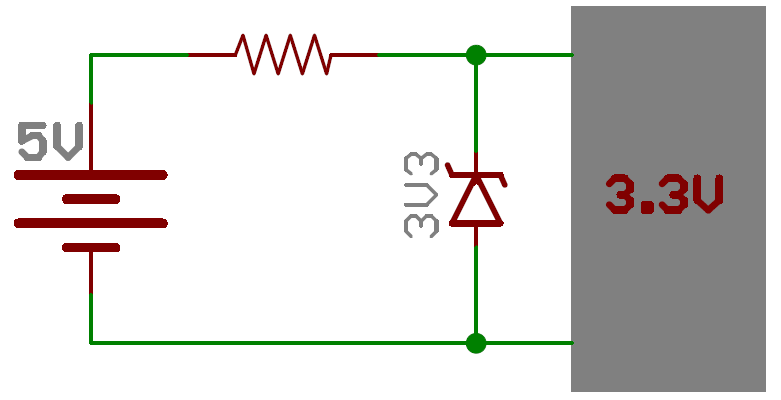
The maximum voltage draw of the robot is around 9 V for the stepper motors; meaning it is necessary to have a battery that can output at least that voltage. Lithium polymer batteries are composed of cells, each one being 3.7 V, which meant that it was required to have a minimum of three cells to accommodate the motors, providing at least 11.1 V total. While the voltage is important, the size, discharge rate, and capacity of the battery are subjects to take into account as well. The robot is very compact with the addition of the stepper motors, so there is little extra space for other components. With all of these key factors in mind, the Rhino Lithium Polymer Battery Pack, illustrated in Figure 10, is the best choice for these requirements. This battery pack can supply a total of 11.1 V, and has a capacity of 610 mAh at 20 C with a maximum discharge current of 12.2 A. The dimensions of the battery are 55 x 31 x 16 mm, and it weighs 50.9 g, allowing the battery to fit nicely behind the motors at the rear of the robot. Rechargeability is the most important factor of this battery. With a 0.6 A current charge rate, the battery itself charges at a fairly quick rate, making it ideal for testing. With the robot constantly running, the battery lasts long enough for the ten minutes required without the need to switch out for a new battery.



***Figure 10. Rhino Li-Po battery, 11.1 V, 610 mA***

To avoid damaging components of the robot with the voltage supplied by the battery, some form of voltage regulation is required to step down the voltage directly from the battery. The major components in question are the microcontroller and the sensors since the motor drivers have their own regulators. Both the microcontroller and the sensors can operate normally with an input voltage of 5 V. Considering the maximum current draw of the robot and overall costs, a LM7805 voltage regulator, depicted in Figure 11, is used. In this case, it was unnecessary to embellish what could be simplistic, leaving this standard voltage regulator as an option for both cost and ease of use. Through heat dissipation, these voltage regulators step down input voltages. While this runs the risk of overheating, the current draw of the robot is low enough to leave this fact as a non-issue.

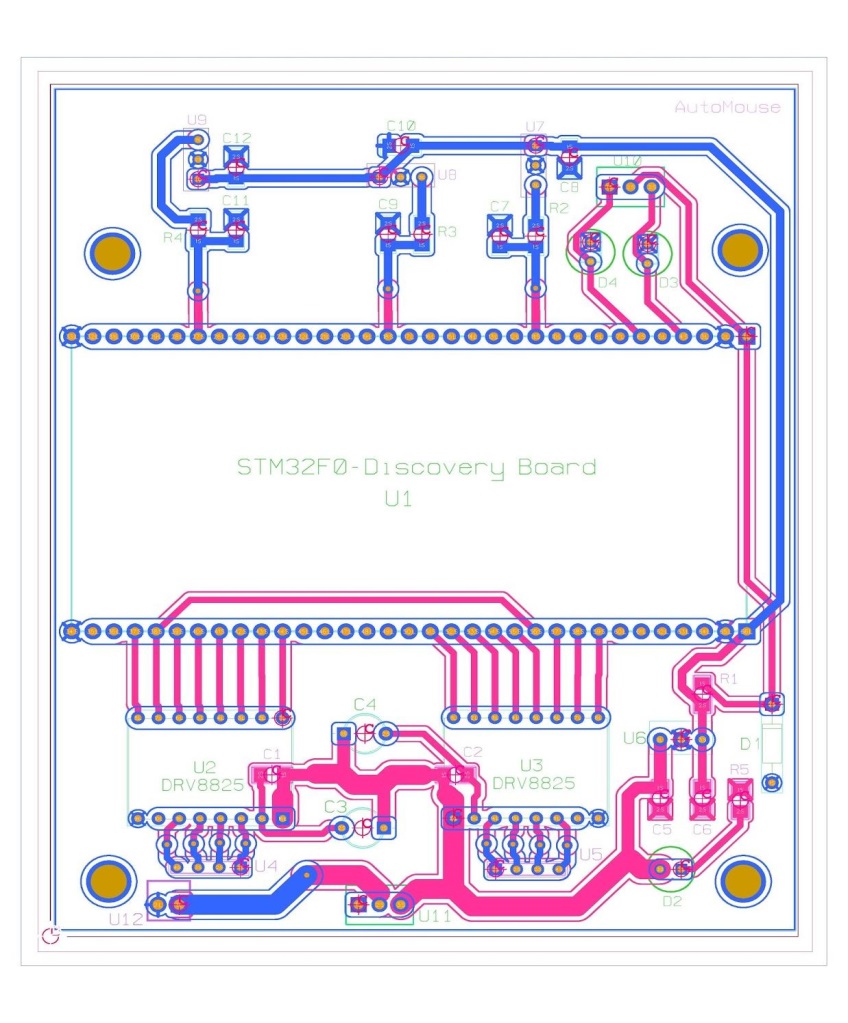
It is also necessary to use a zener diode in order to create a voltage divider that further steps down the 5 V to 3.3 V to provide a safe voltage for the GPIO pins of the microcontroller and avoid potentially damaging them as well. Figure 12 displays the setup of the voltage divider.

***Figure 11. Voltage regulator, LM 7805*** ***Figure 12. Voltage divider with Zener diode***

**4.5 Printed Circuit Board**

A two-layered PCB was designed with a size of 11 x 9 cm to contain all the required electrical components and devices as shown in Figure 13. Most of the through-hole devices are soldered on the top layer except for some headers as well as some surface-mount electrical components. In addition, a variety of connectors were soldered onto the board in order to easily connect or disconnect the battery, microcontroller, and drivers. Two electrolytic capacitors of 1000 μF were placed between VMOT and GND of each motor driver in order to protect the drivers from being permanently damaged by LC voltage spikes. Since the current from the drivers was limited to 1.5 A, the width of the traces routed from the battery into the drivers were required to be twice as large as the traces routed from the sensors into the microcontroller. Vias were used to connect the traces of the top layer to the ones of the bottom layer. For the full schematic of the PCB layout, refer to Figure III in Appendix.

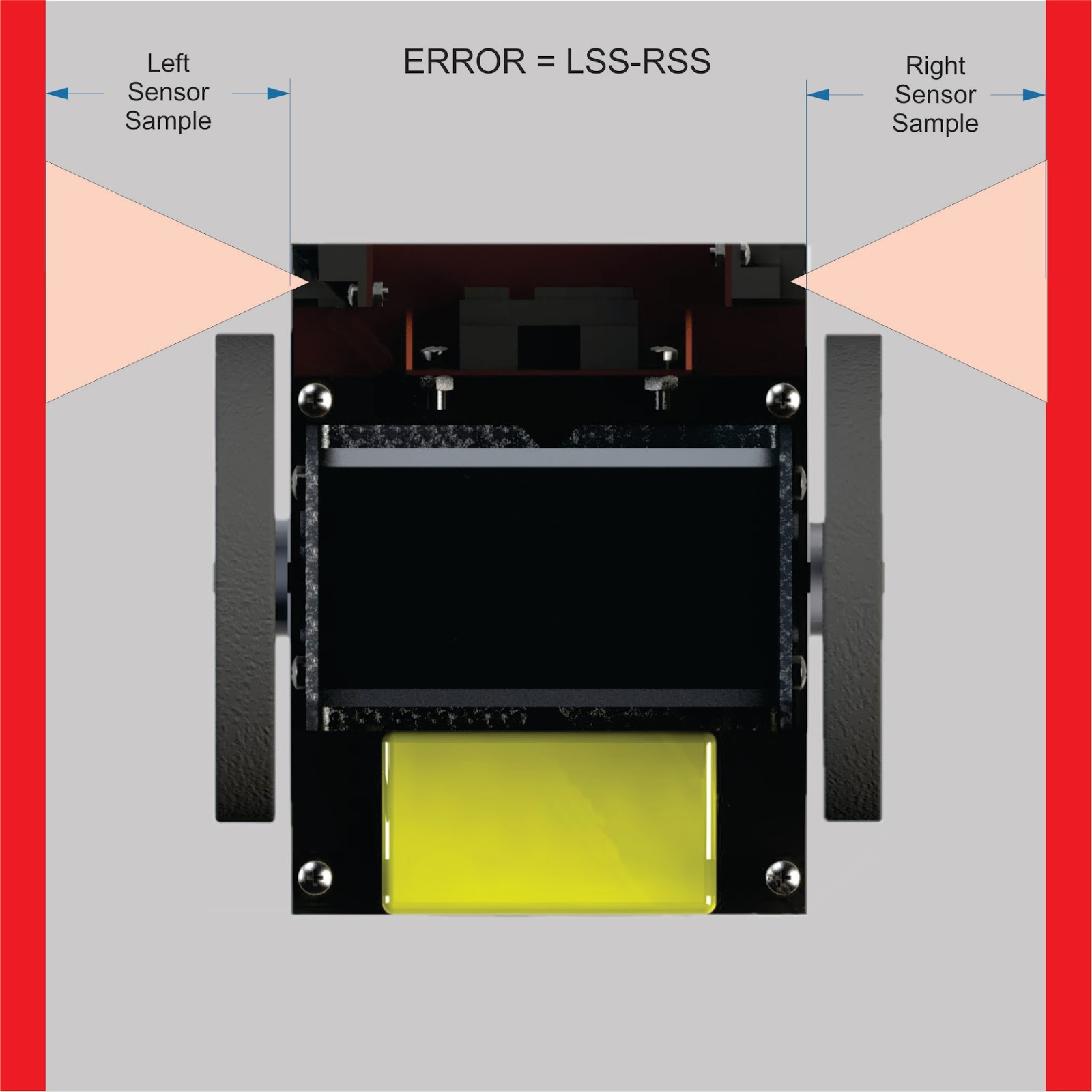


***Figure 13. Printed circuit board design***

**5. Software**

**5.1 Error Correction**

Even with the straightest wheels and perfect synchronization, there is always some error involved when trying to get the robot to drive straight. Without some form of error correction, the robot would drift to one direction and eventually hit the wall. Therefore, a simple error correction procedure is implemented in order to keep the robot aligned in the center of the cells. When the robot drifts closely to the right wall, the readings of the right sensor increase while the readings of the left sensor decrease. The error control is performed when one of these readings is greater than the other. When the robot drifts to the right wall, the right wheel accelerates for a short amount of time and then decelerates back to its original speed to realign the robot inside the cell. On the other hand, when the robot drifts to the left wall, the process is reversed.



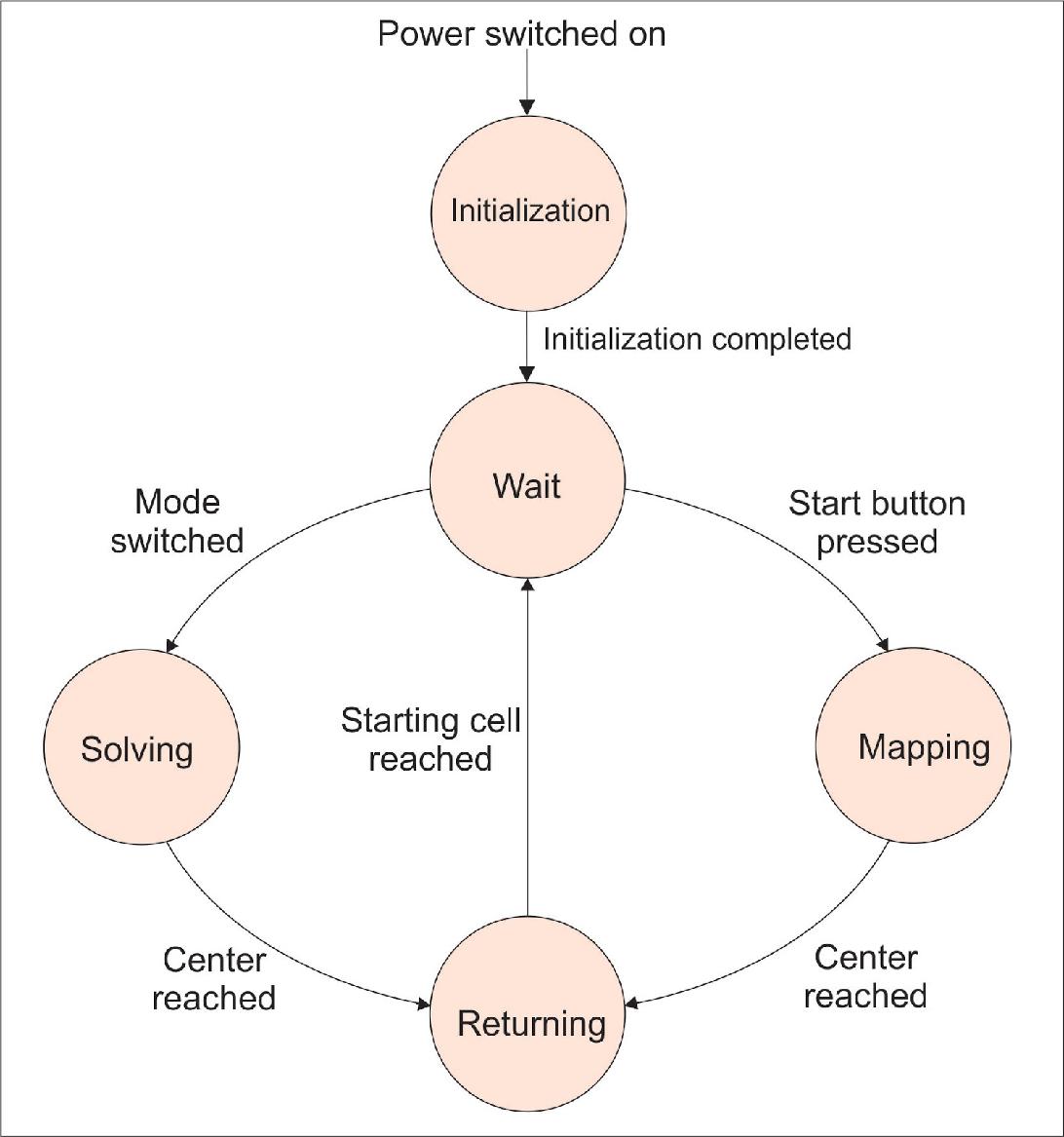
***Figure 14. Error control diagram***

In the case when the robot senses the wall on one side only, the error control is performed according to the readings of only the side that has the wall.

**5.3 Control System**

By structuring the software using a state machine, the complexity of the code can be reduced while increasing its maintainability. This allowed for rapid feature addition and improvement of the code base during development. Using the state machine structure, all the actions which the robot can perform are clearly defined and segregated. Also, the transitions between them based on their events (throwing switches, action completed, etc.) are easily observable.

A state diagram of the machine is shown in Figure 15. Upon turning the robot on via the power switch, the robot enters the Initialization state. This is a preliminary setup state where the microcontroller’s peripherals are configured (ADC, DMA, timers, etc.) and data structures used in the mapping and solving states are created and initialized. Once initialization is completed, the robot waits for the operator to press the start button in order to begin the mapping process. This intermediate state was added so that the operator can accurately position the robot, which is important for the mapping process. There is also a delay of one second after the start button is pressed, so that the operator’s hand does not change the direction of the robot when it moves.



***Figure 15. State diagram of control system***

During the mapping procedure, the robot traverses the maze in order to determine the shortest path to the center. Using the Flood Fill algorithm, the robot can keep track of its position in the maze and intelligently move from cell to cell. In short, each cell is assigned a value which indicates the distance to the center of the maze. As the robot discovers more walls, these distance values are updated. By continuously updating the walls and distance values, the robot is able to find the center by following these values in descending order. Upon successfully entering the center, the path is stored in memory, which is then used in the solving state. For an in-depth description of this routine, refer to Section 5.4 Flood Fill Algorithm. After the maze has been mapped and the robot has entered the center, the mouse proceeds to the Returning state. This action has the robot returning from the center of the maze back to the starting cell. This time, by following the distance values in ascending order the robot can find its way back to the start. This is more so a matter of convenience than necessity, since manually retrieving the robot from the center is difficult given the large dimensions of the maze.

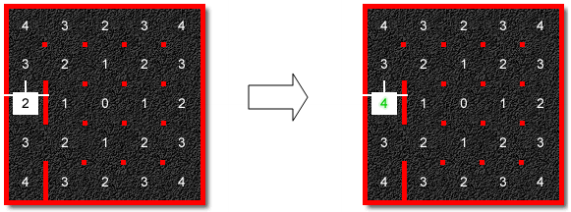
When the robot has returned to the starting cell, it once again waits for user input. At this point, the operator instructs the robot to perform solving by selecting the solving mode with the mode switch. In this state, the robot quickly drives to the center of the maze, using the path previously determined in the mapping state, in order to perform a “speed run”. The time it takes for the robot to arrive to the center is the time that is recorded for the competition. When it finishes the speed run the robot once again returns to the starting cell.

**5.4 Flood Fill Algorithm**

The procedure that was implemented for the mapping state is the Flood Fill algorithm. This well-known algorithm is very dependable in producing the shortest route to the center of the maze. It can resolve more difficult mazes which contain loops, islands, and dead ends, which are commonly found in the competition. It is also relatively simple to implement compared to other algorithms. While there are more sophisticated algorithms available, it is commonly the case that multiple robots discover the optimal route, regardless of which algorithm they use. As such, the competition is more dependent on the time it takes for the speed run rather than the mapping time itself. For this reason, it was determined to use the reliable Flood Fill algorithm for mapping and depend on the larger stepper motors for speed.

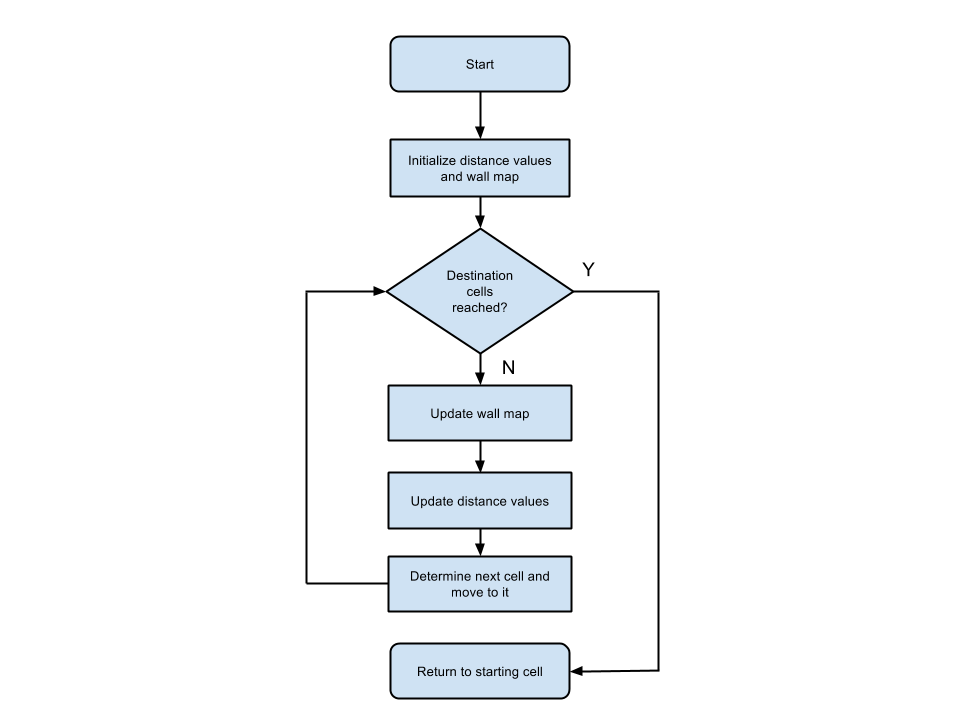
At the heart of the algorithm are the following three principles:

1. Assume no internal walls initially
2. Follow distance values in descending order
3. Prefer straight movement over turns

The first principle indicates that the maze is a blank slate when the robot begins mapping. Thus, it is up to the robot to discover all the internal walls. Second, the robot chooses destination cells by following the distance values in descending order. As shown in the left diagram of Figure 16, the maze is initialized with distance values in every cell, with the center containing zero. Using this rule, the robot will tend to move toward the center. Lastly, given the choice between two cells of equal value, the robot will prioritize straight movement over turns because turns take more time to complete.

***Figure 16. Updating distance values [6]***

In order for the robot to keep track of its position, walls, and distance values, data structures for these metrics need to be created and operated on. The distance values are stored in a two-dimensional, 16x16 integer array which corresponds to size of the maze. Similarly, another two-dimensional character (byte) array is used to store the walls present in each cell. For this array, each byte follows the .MAZ format,[7] where each bit of the lower-half indicates the presence of a wall as follows: (MSB) 0 0 0 0 **W** **S** **E** **N** (LSB). For example, in Figure 16, for the cell that the robot is currently located in, the byte would be 0xA.



***Figure 17. Flood fill algorithm flowchart***

For every new cell that the robot enters, several tasks need to be performed. An overview is shown in Figure 17. First, the robot determines if it has reached the center of the maze. If it has not, it performs the steps necessary for implementing the Flood Fill algorithm. This includes updating the walls array for any newly discovered walls, updating the distance values based on any new walls, and determining the next cell to move to. At the end of this process, the entire maze will have been mapped and the distance values will indicate the shortest path to the center.

Because the route to the center is dependent on the distance values, it is critical that they are updated correctly. Figure 15 demonstrates how this process occurs. When the robot arrives to a new cell and has finished updating the walls, it validates the distance values by placing the current cell and its open neighbors onto a stack to examine. For the current cell that the robot is in, [0,2] (if the upper-left corner is [0,0]), its open neighbors are the cells directly north and south of it. In this case, the minimum distance value of its open neighbors is 3. Since its current cell is 2, it must change its value to 1 + the minimum value of its open neighbors. As shown in the diagram on the right-hand side, the value is correctly updated to 4. This process is repeated until all of the open neighboring cells have been validated (i.e. when the stack is empty).

**6. Conclusion**

Team AutoMouse experienced many challenges throughout the development of this project. The original design was improved in multiple stages in order to meet the established goals. Similar to other Micromouse teams, the initial plan for the physical design of the robot was to have the wheels towards the back and the overall size slightly larger to accommodate all the parts. However, during the experimental phase, it was evident that this design was not going to work since the turn radius was too large and the robot was hitting walls during turns. Hence, the wheels were moved to the center of the robot and the chassis size decreased, drastically improving the robot’s maneuverability inside the maze. Another aspect of the design that changed was the driving of the motors. Initially, the motor was driven through bit-banging, which was inefficient due to its blocking nature, so the method of driving the motors was changed to utilizing the timers in output capture mode. In the end, team AutoMouse designed and constructed a reliable robot which can navigate to the center of the maze at a speed of 23.56 cm/s. Team AutoMouse would like to thank the Department of Electrical and Computer Engineering for sponsorship.

**7. Recommendations**

The prototype produced by team AutoMouse can be used as a foundation for further development. However, there are certain aspects of the robot which can be improved upon. First, the design shape can be more compact by reducing the size of the components. Second, the quality of the PCB could be higher in order to prevent failures in the components. This could be obtained by having the board professionally fabricated. Most importantly, as with most Micromouse projects, it is very important to allocate sufficient time for software development.

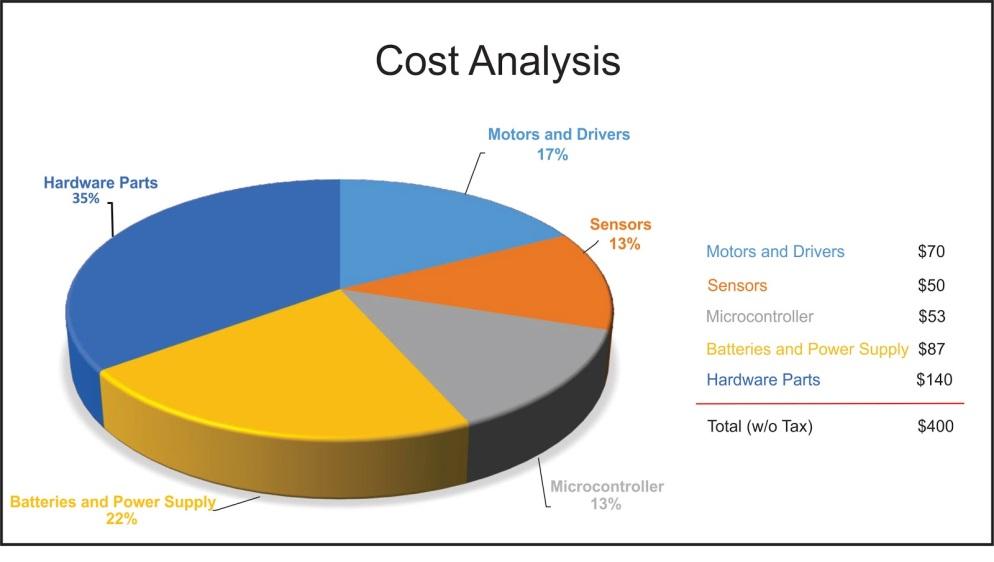
**8. Reference**

1. Micromouse Rules. IEEE Region 6, n.d. Web. 17 Dec. 2014. <http://ieee.ucsd.edu/projects/micromouse/rules.php>
2. Texas Instruments. Web. 17 Dec. 2014. <http://www.pololu.com/file/0J590/drv8825.pdf>
3. Pololu. Web. 17 Dec. 2014. <http://www.pololu.com/product/2133\>
4. "GP2Y0A41SK0F." SHARP, n.d. Web. 17 Dec. 2014. <http://www.pololu.com/file/download/GP2Y0A41SK0F.pdf?file_id=0J713>
5. "GP2Y0A51SK0F." SHARP, n.d. Web. 17 Dec. 2014. <http://sharp-world.com/products/device/lineup/data/pdf/datasheet/gp2y0a51sk_e.pdf>
6. "Laboratory #4-A (Micromouse Algorithms)." [*http://www.csee.wvu.edu/classes/cpe313/spring2010/04\_A.pdf*](http://www.csee.wvu.edu/classes/cpe313/spring2010/04_A.pdf). West Virginia University Lane Department of Computer Science and Electrical Engineering, 2010. Web. 10 Dec. 2014.
7. “MazeFileFormats.” [*https://code.google.com/p/maze-solver/wiki/MazeFileFormats*](https://code.google.com/p/maze-solver/wiki/MazeFileFormats). maze-solver, 24 Apr. 2010. Web. 10 Dec. 2014.
8. **Appendix**

**9.1 Cost Analysis**

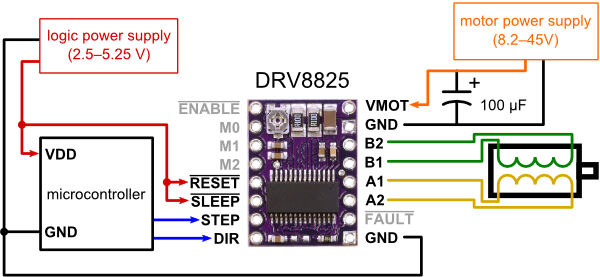
A standard Micromouse competition typically allows $500 for each participant in constructing a robot to compete. However, for this project, a total budget of $600 was generously allotted by the Department of Engineering for completion of the Micromouse robot. The extra $100 was provided as leeway for any extra hardware needed in constructing the robot, considering the competition was not an official one with experienced competitors. Figure I below displays a pie chart of the budget actually used to complete the project. Each section represents a major component utilized in the construction of the robot, and they comprise all the minor and extra parts ordered for that major component.

The last section on the cost analysis list, “Hardware Parts”, is a filler piece that indicates hardware parts including wheels, brackets, wire casings and connectors, etc. Despite all the extra costs put into this project, the budget at the end totaled approximately $400, meaning about $200 under the provided budget.

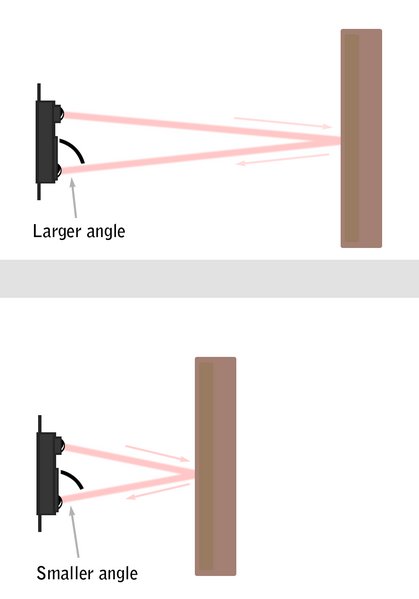


***Figure I. Pie chart of budget and cost analysis***

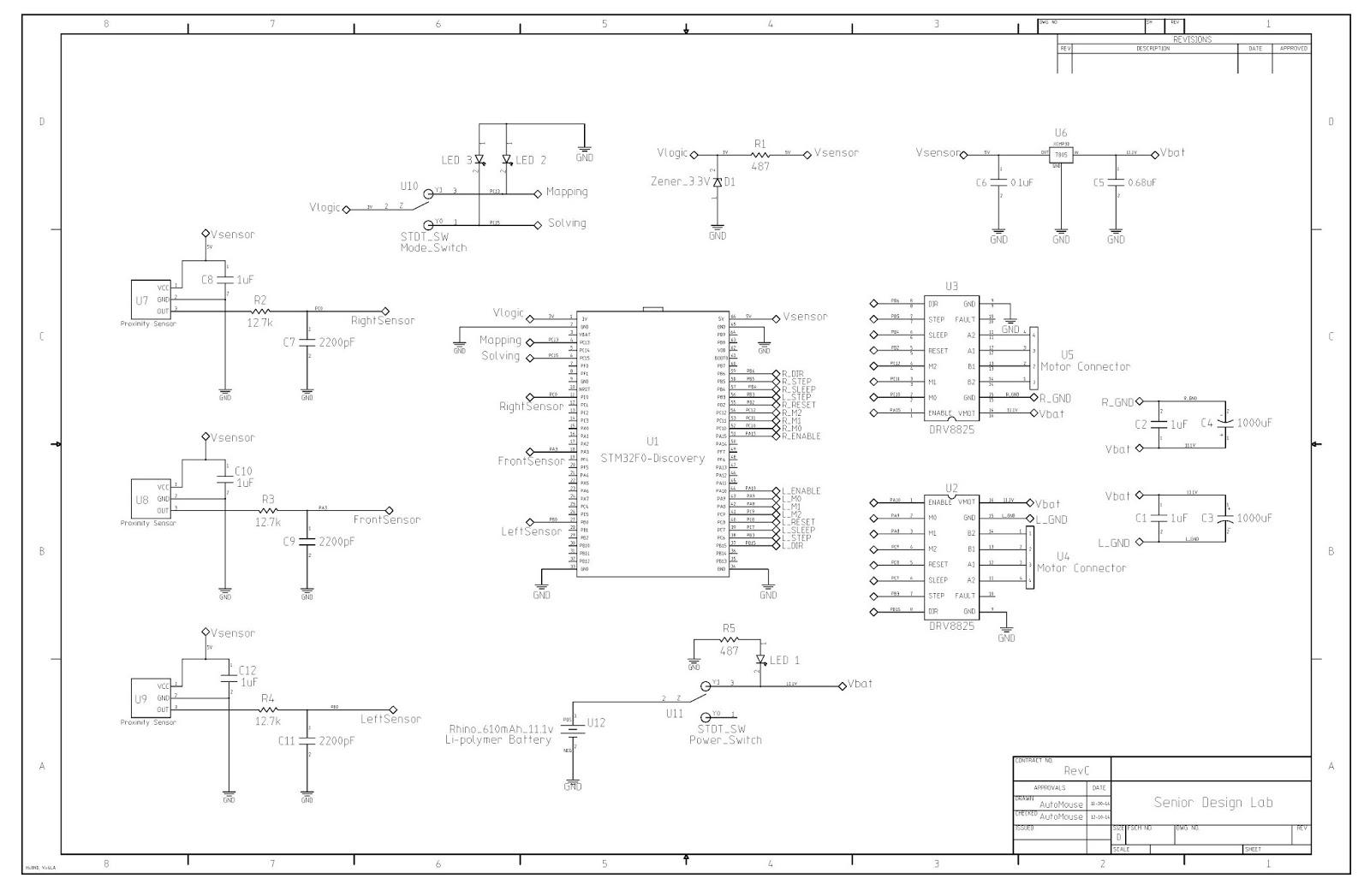
**9.2 Figures**



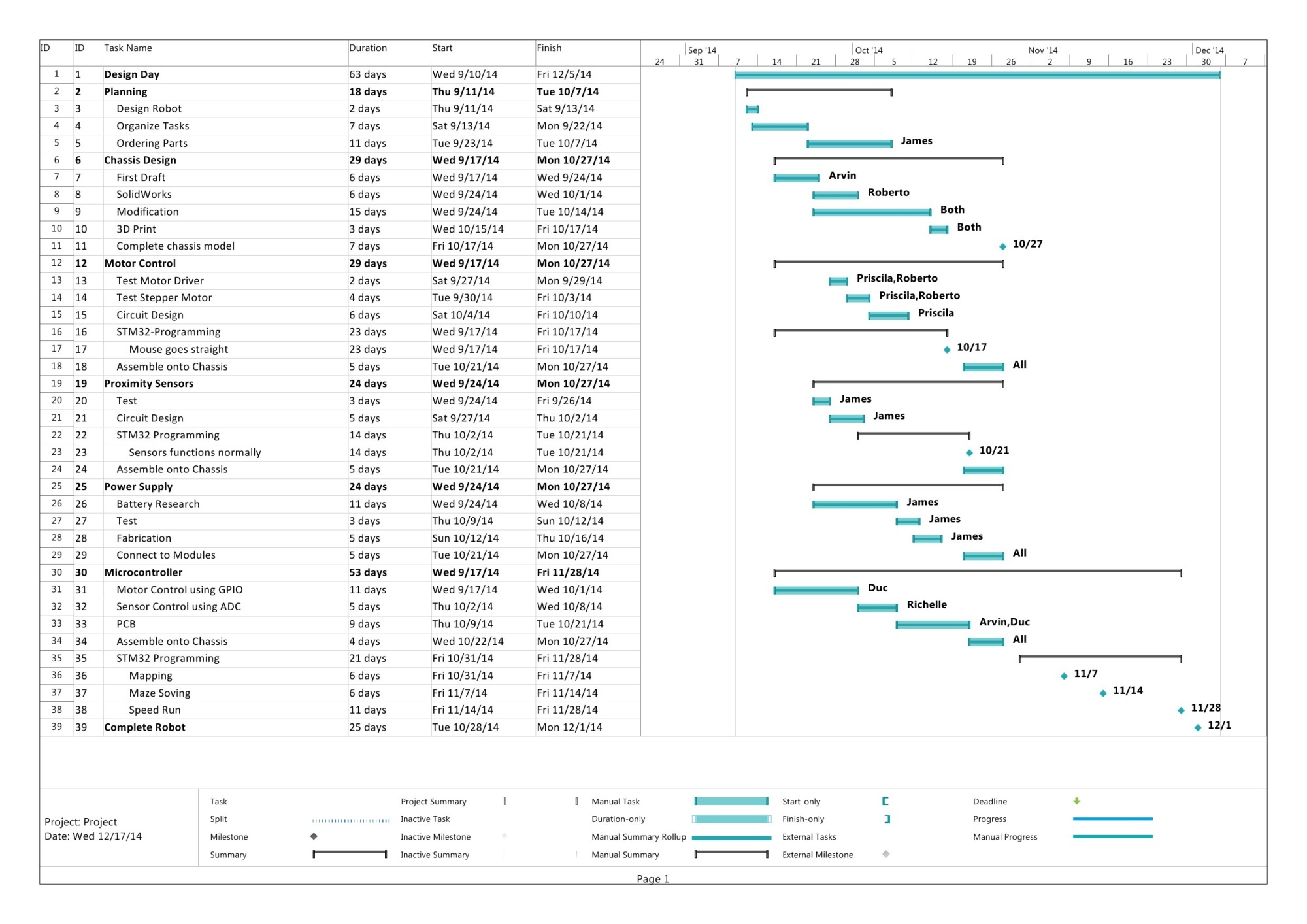
**Figure II.** Motor and driver configuration



**Figure III.** Method of Triangulation



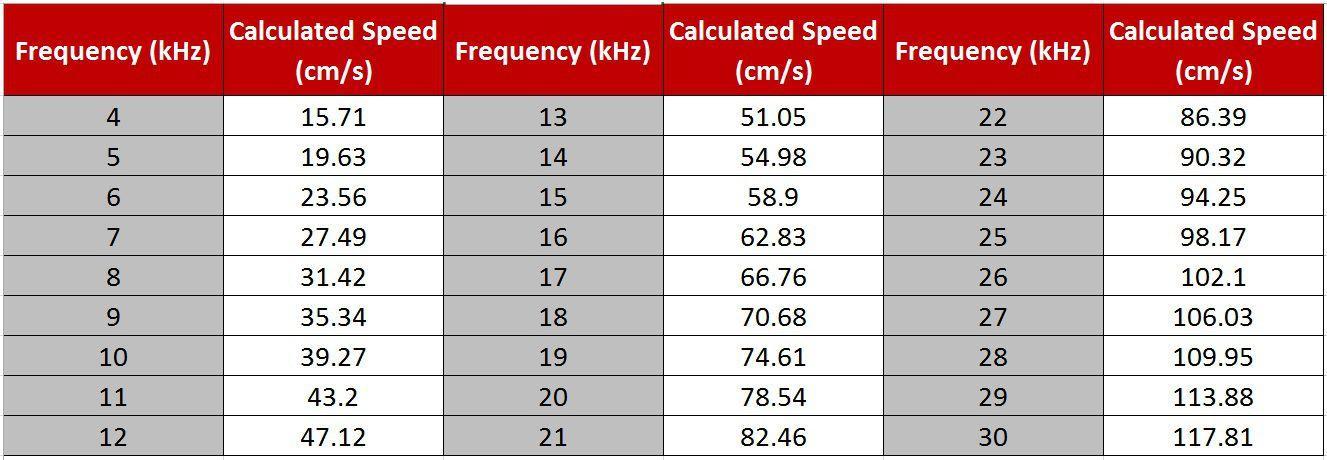
**Figure IV.** Final schematic for PCB design



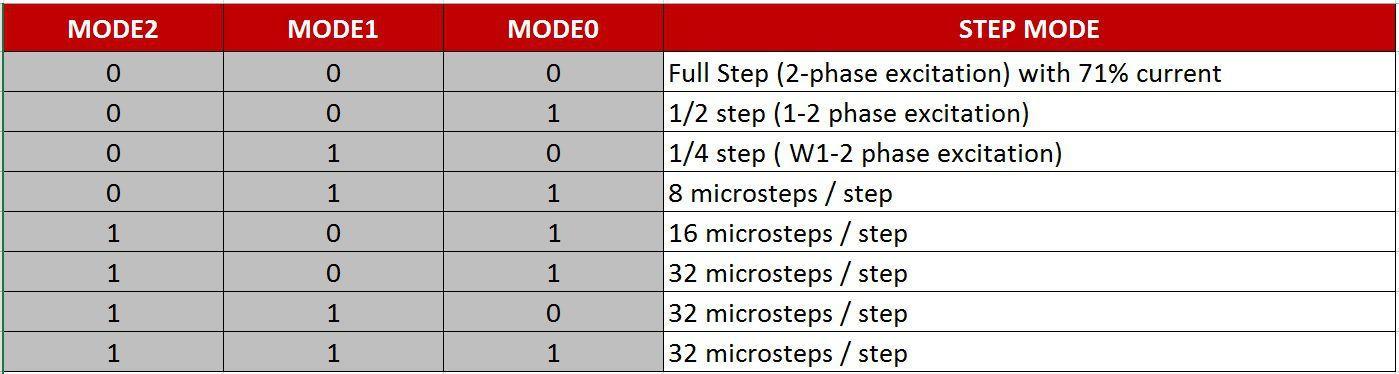
**Figure V.** Gantt Chart

**9.3 Tables**

Table 1 uses the following equation:



**Table 1.** Correlation of speed and frequency



**Table 2.** Stepping format [2]