Trinity College Fire Fighting Robot Competition

Conner Sheets – Electrical Engineering
Jared Sutphin - Electrical Engineering

Project Advisor: Dr. Blandford

Trinity College International Robot Contest
Hartford, Connecticut
April 13th, 2019

Acknowledgements

We would like to thank several people for helping us along the way through our senior project. The people we would like to thank are: Jeff Cron, Dr. Howe, Dr. Blandford, Dr. Mitchell, and Mitch Steinkamp. We would like to thank Jeff for attending the competition with us, answering our endless questions, and always willing to help with anything needed; Dr. Howe for constantly encouraging us and supporting us through the entire project with answers to our electronics questions; Dr. Blandford for advising us and supporting our project through all programming questions; Dr. Mitchell for helping us with miscellaneous questions regarding our project; lastly Mitch for supplying the project with high quality springs so our line sensors could be used effectively and would have shock absorption.

Table of Contents

List of Figures

List of Tables

Appendices

I. Introduction

With today's technology why are we still risking countless firefighters' lives to run into burning buildings, put out fires, and save lives? Firefighters are at constant risk of being burned, becoming trapped, inhaling smoke, and so many more things that could be avoided. An autonomous robot being the first responder to a fire could greatly reduce the risk of losing human lives. The goal of the Trinity College Fire Fighting Robot Competition is to create an autonomous robot that can navigate around a replica house to search for sources of fire and extinguish them. To further replicate real life scenarios, the competition requires the robot to be able to detect a tone like a fire alarm, avoid obstacles, and maneuver on different flooring. While the competition is not creating the exact model that would be used in real houses, it creates a small-scale replica that serves as a proof of concept to an actual autonomous home firefighting robot one day. The Trinity College Fire Fighting Home Robot Competition is the first step in the direction fighting fires with autonomous robots that will allow us to save lives without risking lives.

II. Statement of Problem

The Trinity College Home Fire Fighting Robot competition has three different levels of difficulty that the robot will be tested in. The robot may only progress onto the next level by passing the previous trial and by not exceeding the five trials allotted to each team for the competition. If the robot fails to recognize the frequency to start or if the robot begins prematurely moving, that trial will be recorded as a failure. Each team may do a max of three trials on Saturday or Sunday.

A. Robot Inspection Table

Before beginning any trial, each robot must pass inspection at the robot inspection table. The robot inspection table will be checking to ensure the following parts of the robot are compliant with the competition guidelines. The robot must pass size inspection by fitting into a bounding box, which has a base of 31 x 31 cm square and a height of 27 cm or

(12.2 in x 12.2 in x 10.63 in)(Base x Width x Height). Each robot must operate untethered and be powered within its chassis. Robots can use air, inert gas, water, mist spray, or other mechanical methods to extinguish the flame. The inspection will ensure no robot is using any type of powder extinguishers. Robots must have a carrying handle for judges to easily transport the robot without damaging the robot in any way. Direction of movement must be signaled by an arrow on top of the robot. Microphones for detecting the 3.8 kHz +/- 13% start frequency played by judges must be visible on the top surface of the robot and easily accessible. The microphone must also have a blue background and clearly be labeled by the abbreviation MIC. Lastly, the robot inspection table will be ensuring each robot is conforming to the rules and guidelines of the competition.

B. Control Panel Requirements

Robots are required to have a control panel on the handle in a horizontal orientation. The panel must include the checkpoint LEDs, kill motor plug, microphone, and arrow indicating direction of movement. A main power switch must be included in the robot design somewhere not on the control panel in case of an electrical failure. The checkpoint LEDs required in the control panel are the blue sound detect LED, red flame detect LED, and green video detect LED. The blue LED is supposed to illuminate when the correct frequency sound is detected signifying the start of the robot's trial. The red LED is supposed to illuminate when the robot has detected the flame and turned off after the flame has been extinguished. The last component required in the control panel is the kill motor plug that allows for judges to easily stop the robot in case of emergency.

C. Trial Runs and Layout Explanation

After the robot has passed all requirements at the robot inspection table, it will be allowed to start a trial at that difficulty level. All robots are required to start at the first level for the Trinity College Fire Fighting Robot Competition. Level 1 is a 244 cm x 244 cm maze that is supposed to represent a simple model of a house with high-contrast floors and walls. Robots must not rely on precise dimensions because measurements for the maze can have up to a tolerance of 2.5 cm. All hallway widths and door openings are 46 cm wide. The doorways are marked by white tape on the floor that goes across the entire door opening. The only obstacle present for Level 1 of the competition is a dog obstacle that will block a hallway. Robots are not allowed to touch the dog and must find another hallway to maneuver around the maze. The robot will be placed in a 30 cm diameter solid white circle for the start of the trial. To successfully complete Level 1 the robot must autonomously maneuver through the maze and extinguish the flame in under 3 minutes. The layout for the Level 1 maze is shown in Figure 1.

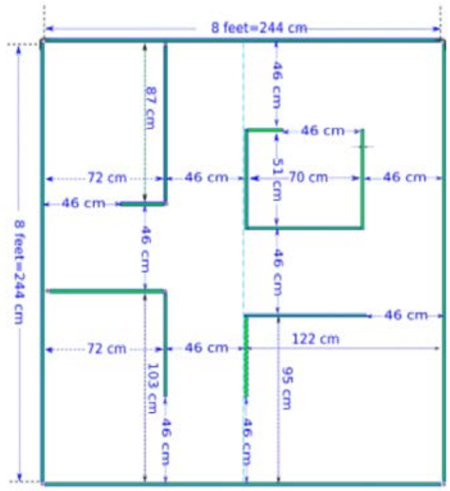


Figure 1: Level 1 maze with dimensions [1]

After successfully completing Level 1, teams may choose to progress on to the Level 2 maze. Level 2 is meant to mimic a more realistic house with different types of flooring and other decorations. Level 2 has four different potential configurations but relatively similar. The maze will now have rug placed in some or all the areas shaded in Figure 2. The robot must be able to navigate through the house over different types of flooring. The rug will be 1 cm thick and each maze will have different colors/locations of rugs. Level 2 also has wall decorations such as pictures, tapestries, and mirrors. These wall obstacles will not stick out more than 1 cm. The second level has some similarities with Level 1 like the dog obstacle and the same goal of finding the candle flame and extinguishing it. Level 2 will be considered a success if the robot is able to autonomously navigate through the maze, overcome obstacles, and put out the fire in under 4 minutes.

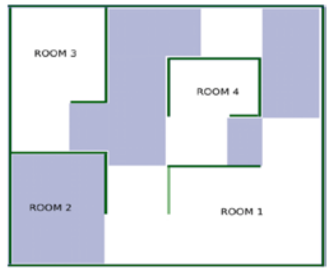


Figure 2: One possible Level 2 orientation with possible rug locations [1]

After successfully completing Level 2, teams may choose to attempt Level 3. Level 3 attempts to add another level of difficulty by incorporating search and rescue. Level 3 is made of two Level 2 mazes that the autonomous robot must navigate through avoiding the same obstacles as Levels 1 and 2 such as dogs, furniture, mirrors, rugs, and paintings. The robot is required to use visual recognition to find a baby trapped in a crib and transport it out of the maze then come back and extinguish an unknown amount of fires. The two Level 2 mazes are connected by a 1 m hallway that can have a maximum pitch angle of 15 degrees. To be successful on Level 3 the robots must first save the baby, then extinguish all the candles, and lastly return to the start pad before 5 minutes is up. A full breakdown of the rules and requirements for the competition can be found at the Trinity College Fire Fighting Home Robot's webpage [1].

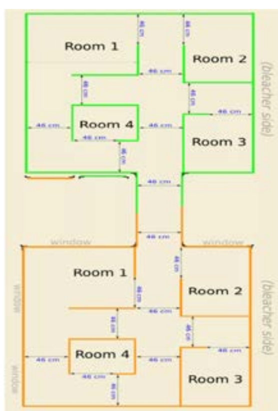


Figure 3: Level 3 maze orientation [1]

D. Breakdown of Scoring

The teams get five trials to try and complete all the challenges presented in the different levels. The final score per trial is calculated based on the Actual Time (AT) the trial takes the robot to complete, the Mode Factor (MF) used in the run, Room Factor (RF), and Penalty Points (PP). Mode Factor (MF) is the product of all the Operating Modes (OM) used from Table 1. Room Factor is used to adjust the Actual Time based on the amount of rooms searched before fire is found. Trial Score (TS) is the Actual Time with the addition of Penalty Points from Table 3. The Operating Score is calculated by multiplying the Trial Score, Mode Factor, and Room Factor. The equation used to calculate each trials score is:

$$(TS) = [(AT) + (PP)] * (MF) * (RF)$$

Table 1: Possible Operating Modes for Mode Factor

Operating Mode	**Multiplier**	**Short Description**
Arbitrary Start	0.80	Robot will be started in one of the four rooms without fire and facing a random direction
Return Trip	0.80	After extinguishing flame robot returns to the start location
Non-Air Extinguisher	0.75	Robot uses an extinguishing method other than fan or blower
Furniture	0.75	Obstacles replicating furniture can be placed in the rooms
Candle Location Mode(Level 1)	0.75	Candle can be placed anywhere within the room and no white circle surrounding it on the floor

The multipliers we attempted from Table 1 above at the competition were the Arbitrary Start, Candle Location Mode, and Return to Start.

Table 2: Room Factor Breakdown

Number of Rooms Searched	Multiplier
1	1.0
2	0.85
3	0.5
4	0.35

The room factor is intended to adjust the actual time taken per run based on the amount of rooms the robot must search before finding the flame.

Table 3: Possible Penalty Points

Penalty Name	Points Added	Short Description
Touching Candle	50	Any robot that touches the candle or base will have 50 points added to that trial's time score
Continuous Wall Contact	1point/2cm	Robots that are in continuous contact with a wall will have a point added per 2cm
Kicking the Dog	50	Robots must not move the dog obstacle more than 1cm

Penalty points are added to the Actual Time(AT) of any robot that performs any of the possible actions in Table 3 above.

III. Client Requirements

A. Dimensions

i. Fit in a box with a base 31 x 31 cm square and 27 cm high

B. Power

i. Draw less than 10 A from a single US-standard 15-amp outlet

C. Tasks

i. Be self-controlled

ii. Start at 3.8 kHz frequency

iii. Navigate the arena without leaving anything behind or causing damage

iv. Identify a candle before extinguishing

v. Extinguish the candle within three minutes(Level 1) and four minutes(Level 2)

D. Robot

i. Contain a control panel

   a. Turn on a red light only when flame is detected

   b. Turn on a blue light only when sound activation is detected

   c. Have a kill motor plug that removes power from the robot’s sensor, control and drive systems

ii. Have a microphone

   a. Located on the top of the robot and accessible from above

   b. Labeled “MIC” on a blue background

iii. Have a durable carry handle

   a. Have an arrow pointing in the start direction of the robot

iv. Have a method of extinguishing the flame

v. Have a main power switch not on control panel

IV. Project Design

A. Microprocessor Selection

The STM32F446RE board [3] will be the brain of the robot containing the maze navigating and firefighting program. This microprocessor was chosen due to its speed and versatility to be used within almost any project. Both team members had experience using this board before and understood how to maneuver the data sheet to program on it. The versatility will allow the different sensors to be controlled by one microprocessor. The processor has three ADC ports, two of which have the capability of running up to 16 different channels. This would allow us to use all the analog sensors we had planned and have room for extra if needed.

B. Sensor Selection

The next task of this project was deciding the way we wanted out robot to function within the maze and then to begin picking sensors to allow us to implement that functionality. Based off our design to implement a right wall following algorithm for the main navigation of the maze we knew we were going to need some distance sensors. Upon researching and testing distance sensors we discovered the best choice for our design was going to be the IR Proximity Sensor Short Range – Sharp(Figure 4 below). These sensors were chosen due to their short-range capabilities of 4cm to 30cm, whereas alternatives were not accurate at such low distances. The other main deciding factor was due to their cost being low compared to alternative proximity sensors. Our project design implemented two of these sensors on the front of the robot for checking for potential obstacles or obstructions and two on the right side for implementing the right wall following algorithm.

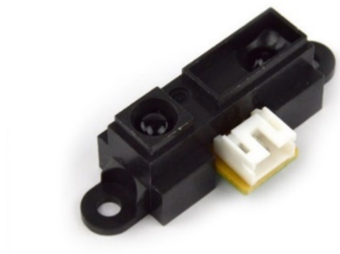


Figure 4: Infrared Proximity Sensor Short Range – Sharp GP2Y0A41SK0F [2]

The robots are required to have sound detection capabilities to start upon the detection of a 3.8 kHz start tone. Upon looking into microphones, we bought a simple analog electret microphone with amplifier breakout board from SparkFun(Figure 5 below). The filter we implemented with this microphone was a digital bandpass filter to simplify the hardware design and decrease any additional purchases.

Figure 5: SparkFun Electret Microphone Breakout [3]

The rooms within the maze are represented by white tape lines on the floor at the doorways. To signal to the robot that it had found a room we needed a way to detect the change from dark black flooring to white tape. To accomplish that we chose to use two-line sensors mounted on the bottom of the robot. The line sensors we chose for the robot were the RedBot Line Sensors from Robot Shop, as seen in Figure 6 below, due to their low cost.

Figure 6: RedBot Line Sensor [4]

The flame detection portion of this project required the implementation of two different sensors. The initial detection of the flame within a room was done using a UV-Tron sensor such as those shown in Figure 7 below. The problem we hit with using this single sensor for the flame detection was there was no way to hone in on the actual location of the flame. To solve this closing in on flame problem we implemented two analog UV sensors that had adjustable sensitivity on the sides of our motor with propeller. We set up one of the sensors to be extremely sensitive and the other was set up to be only tripped when within several centimeters of a flame.

Figure 7: Different UV Tron Flame Detectors [5]

C. Additional Components

For the driving power within the robot we decided to use two motors with wheels attached to the shaft along with two ball caster wheels to keep the robot balanced while in motion. The motors we chose to use were the Pololu 12V, 100:1 Gear Motor in combination with the Cytron 10A Dual Channel DC Motor Driver, both of which can be seen in Figure 8 below. We chose these motors due to easily controllable max RPM, not too large for contest requirements, and price was comparable to alternatives.

Figure 8: Pololu Motor and Dual Motor Driver [6] [7]

D. Printed Circuit Board

To help decrease the amount of wiring on the inside of the robot we decided to design a printed circuit board that would house most of the connections necessary for our robot. The design of the printed circuit board is just connections for all the distance sensors, UV sensors, line sensors, motors, extinguisher fan, and power nodes for the 5 and 12V sources. The design for our printed circuit board can be seen below in Figure 9.

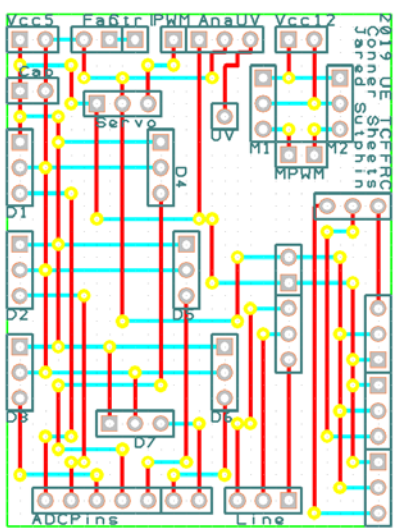


Figure 9: Printed Circuit Board Design

E. Power Planning

After the sensors and driving power for the robot had been decided the next decisions in the design were how to power the robot. All our sensors and microprocessor needed 3.3 – 5V supply while our motor driver needed 12V. After testing different setups, we decided the best solution was to have two power sources along with two buck converters to step down the voltage, as well as, add some safety to the chance of supplying too much current. The two sources we decided to use were a 9V battery and a LiPo battery that fully charged can supply around 12.5V. We chose to use a rechargeable 9V battery and LiPo to help lower the cost of the robot. The 9V battery would be stepped down by a buck converter and used to give the microprocessor and our sensors their own dedicated battery to avoid any spikes caused by the motors. The LiPo battery would be used to supply 12V directly to the motor driver and stepped down to 5V to be used for the extinguisher fan, H-bridge power, and dual motor driver power. The use of two sources seemed to decrease the amount of potential sensor error from power spikes or drops.

F. 3-D Print Designs

We decided to try and 3-D design and print several of our parts for a couple reasons. They could be easily altered or improved, the materials needed to print were relatively cheap, and the plastic was durable enough to withstand slight collisions during practicing. The first part of the robot we 3-D designed was the chassis to house all our robot’s components while also allowing us to implement the program plan we had planned. To utilize the right wall following algorithm we chose to implement we needed to place two distance sensors on the right face of the robot. Along with those right-side sensors we decided to best avoid obstacles and walls we would need two distance sensors on the front face. The front distance sensors would have one directly in the center and the other off to the right. We chose this layout, so the middle sensor would allow us to detect the dog obstacle whereas if either was tripped it would signal a wall was found. The distance sensors for right wall following were placed so that one was located towards the front face and the other directly at the back. This set up would allow us to detect when the robot was fully within a doorway or hall before turning. The other design considerations when creating the chassis were to pull the distance sensors slightly inside the body to give them some shielding from potential interference. Holes were placed so the motors and ball casters could be mounted directly to the chassis. The chassis design can be seen in Figure 10 below.

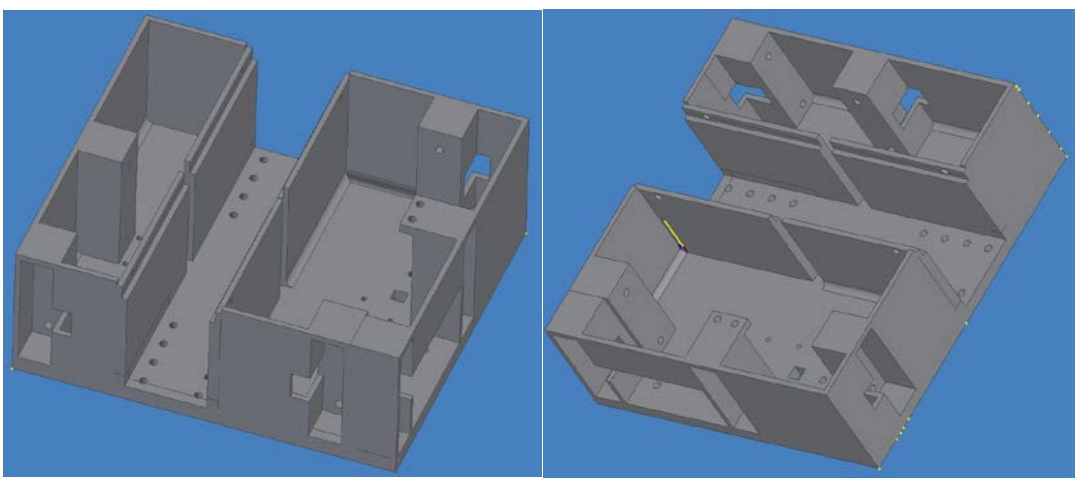


Figure 10: Chassis Design

Upon testing the UVTron we began to quickly realize the sensor was extremely sensitive and would be tripped by almost any small flame or spark inside of the room. To solve this issue, we decided to design a shield to cover up most of the sensor other than a small area in front of the sensor. The design for this shield can be seen in Figure 11 below.

The shield helped make the sensor less sensitive to the point it would only be tripped when the robot had entered the room containing the fire.

Figure 11: UVTron Cover

The line sensors picked for our robot had a very short range and had to be almost touching the floor. The issue with having them that low was the possibility of breaking the sensor when the robot encountered a rug or small bumps on imperfect maze floor. We decided to try and design a rounded holder on the bottom of our robot and add springs between the line sensor holder and chassis to give the sensor shock absorption which can be seen in the left picture below. The newest curved line sensor holder can be seen in the right image below.

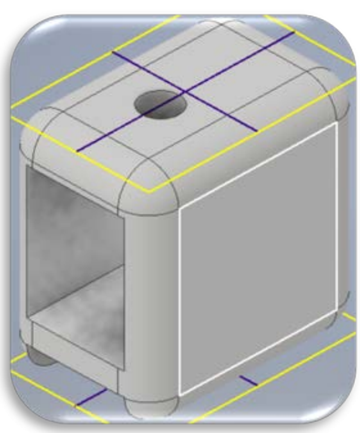


Figure 12: Shock Absorption Spring and Line Sensor Holder

The competition requires each robot to have a sturdy carrying handle that the judges can use for moving around the robots without risking damage. We decided to construct a handle out of aluminum brackets that would attach to the bottom side of the robot. These would then run up the sides of the robot to a height 2cm above the max height the propeller blade could reach. These brackets would then attach to a 3-D printed control panel we designed to meet the competition requirements. The design of the handle and control panel can be seen in Figure 13 below.

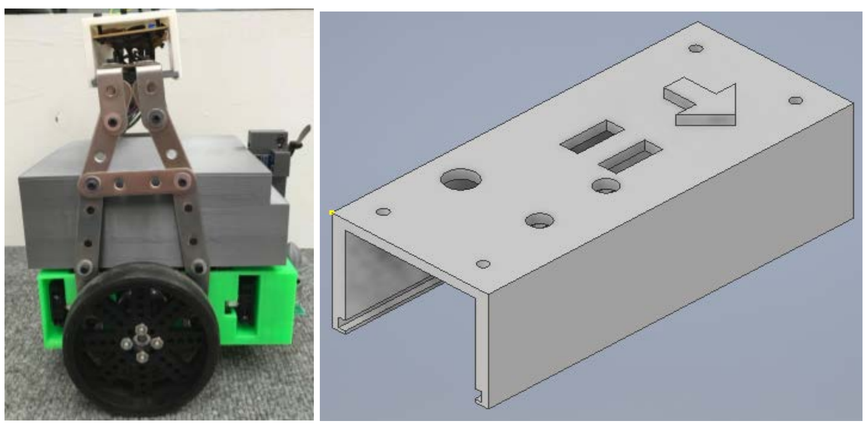


Figure 13: Bracket Handle and Control Panel

The last component we designed was a piece to attach to the front of our chassis to hold our extinguishing fan and the two UV Sensors for closing in on the flame. This piece was designed with the same considerations made when designing the UVTron cover to basically help decrease the sensitivity of the UV Sensors, so they would not be tripped as easily. We created this piece because the original plan of using a servo motor was causing issues with our power and messing up other sensors. This piece can be seen in Figure 14 below holding the extinguishing fan and UV Sensors.

Figure 14: Fan and UV Sensor Mount

G. Concept Design

The autonomous robot will be signaled to start by a 3.8 kHz tone played into the robot's microphone. After the robot has been signaled to start the function to exit the start room, which is outlined in Figure 15 below, will be executed. This function begins with the robot spinning in a circle while taking a distance measurement with the front center distance sensor every eighteen degrees. After the robot has spun a full circle it will spin back to face the direction that the shortest distance was measured. Next the robot will progress forward until the robot is within ten centimeters of the wall and turn left. Using the two distance sensors on the right face the robot will align on the wall and then begin right wall following while searching for a line below. Once the line had been found it meant the robot was at the doorway of the start room and ready to progress on to the maze exploration stage of the software.

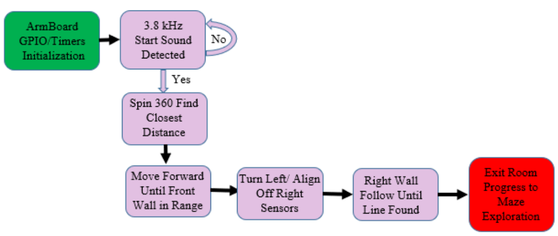


Figure 15: Start Room Software Flow Chart

Once the robot has exited the start room the robot will enter the main portion of the software which is the maze exploration. This maze exploration is outlined in the software flow chart below in Figure 16. The robot will continuously move forward adjusting the left motor speed to keep the front right distance sensor within ten to eleven centimeters. If the distance sensor is further than eleven centimeters the left motor speed is increased to push the robot closer to the wall or if the distance sensor is less than ten centimeters away from the wall the left motor is slowed to pull the robot away from the wall. While continuously adjusting the speed of the motors to stay close to the right wall the robot is checking the front distance sensors as well. If one of the front distance sensors reads less than ten centimeters the robot has found an obstacle and will turn left before progressing forward again. The last thing the robot is checking for is the right distance sensors suddenly reading a distance greater than twenty centimeters. If this happens it means, there is either a door way or a hallway and the robot will turn right before slowly moving forward while checking the line sensors. After the robot has moved forward about five inches if no line has been found the software will go back to the right wall following exploration. However, if a line has been found the robot will increment the variable(LineCount) we are using to keep track of lines found and execute a check of the room utilizing the UVTron to see if a fire is present. If no fire is detected the explore maze loop will be reentered, but if a fire has been detected the robot will enter the room and move into the fire extinguishing section of the code.

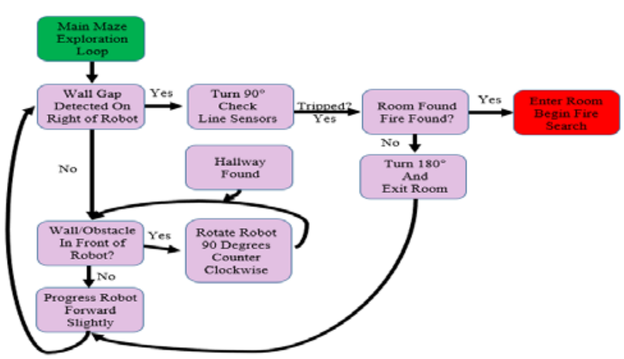


Figure 16: Maze Exploration Software Flow Chart

Once the robot has found the room containing the fire it will begin to execute the software loop outlined below in Figure 17. This is also the point the software determines the amount of lines it must cross to return home by subtracting 5 minus our variable(LineCount) we used to keep track of lines found. We stored the difference in another variable(ReturnLineCount) to use when navigating back to start room. The robot will spin taking a scan of the room using the sensitive UV Sensor to locate the location of the candle flame. After locked onto the direction of the flame the robot will slowly progress forward and check for the two cases that would signal that the flame is within extinguishable range which are: non-sensitive UV Sensor is being tripped or the front middle-distance sensor is being tripped. This loop repeats until the UV Tron is no longer registering that there is a fire present in the room. Once the flame has been extinguished the robot executes the same loop that it did when leaving the original start room. After the robot has left the room containing the fire it will continue the right wall following searching for lines and decrementing the ReturnLineCount variable each time one is found until the variable equals zero. When the variable equals zero it means the robot has found the line in the doorway of the original start room. The robot then crosses over the line into the room and then stops signaling the end of the trial.

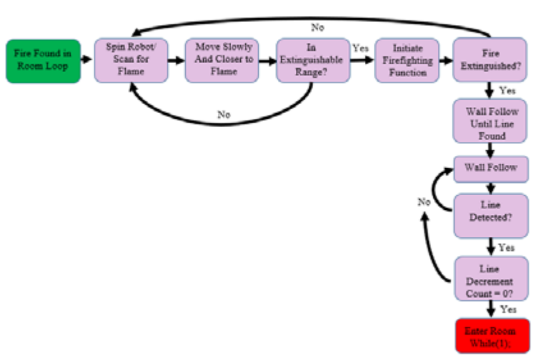


Figure 17: Extinguish Flame and Return Home Software Flow Chart

The hardware block diagram in Figure 18 represents a simplified description of what the project contains as far as main components. The more in-depth schematic can be found in Appendix C. The robot's main chassis houses most of the components such as the range finder sensors, the motors with wheels, 9 and 12V batteries, and most of the wiring for the robot. The main chassis is also attached to a reinforced handle for easy transportation of the robot. The handle contains the control panel with a kill motor plug, microphone, and the status LEDs such as: sound detect, and flame detect. The robot has a front portion of the chassis that contains two UV sensors and extinguisher motor/propeller combo for adjustable flame extinguishing direction. The last components are two ball caster wheels to help the robot move and remain upright while being propelled by the two-back motor/wheel combos.

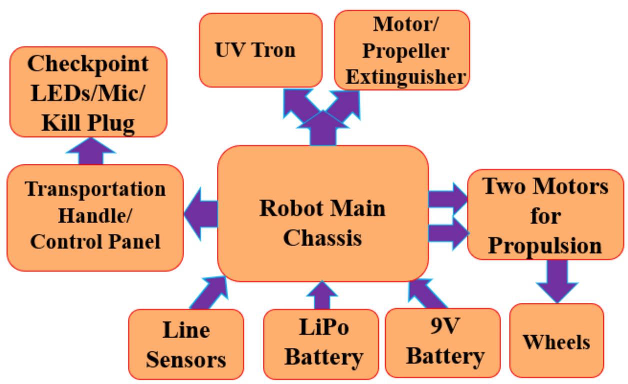
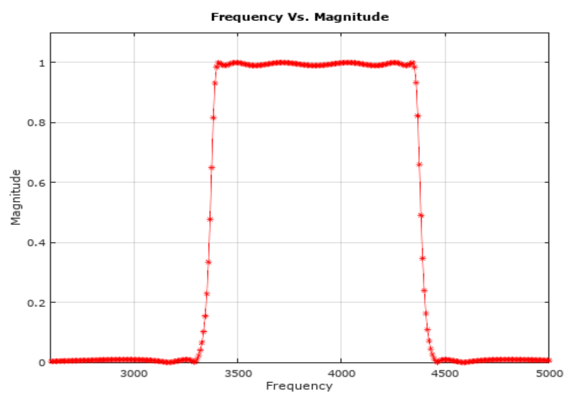


Figure 18: Hardware Block Diagram

H. Bandpass Filter Design

One of the project requirements is that the robots only startup upon the detection of a 3.8kHz ± 13% start tone. This tone is meant to resemble a fire alarm. To accomplish this requirement the robot needed a filter for the microphone. We could have used either a digital or a hardware filter but upon further thought we chose to implement a digital filter due to no extra required components and easily adjustable if needed at the competition. We used Octave to find the required coefficients to plug in for the code to implement the digital filter. The filter that we designed was a sixth order elliptic bandpass filter. The Octave script can be found in Appendix A and the plot of the frequency response with coefficients generated can be seen in Figure 19.




```
Elliptic Order is 6
SOS =

   0.011522   0.012323   0.011522   1.000000   0.284777   0.979202
   1.000000   0.457263   1.000000   1.000000   0.230837   0.913693
   1.000000   0.358901   1.000000   1.000000   0.056960   0.809374
   1.000000  -0.548337   1.000000   1.000000  -0.236649   0.810769
   1.000000  -0.641882   1.000000   1.000000  -0.416775   0.915080
   1.000000  -1.204521   1.000000   1.000000  -0.475228   0.979594
```

Figure 19: Frequency Response/Coefficients of Digital Bandpass Filter

I. Other Project Constraints/Considerations

The economic constraints for this project were limitations set by the budget given by the school for the funding of the project. All the components needed for the project and materials used in fabrication were within the allotted budget. Parts such as the range finders, motors with drivers, UV Tron, wheels, PCB, microphone, and contest entry fee all remained within the financial constraint given to the project by the school.

The environmental constraints were the power consumption of the robot due to the power being used is coming from non-recyclable batteries and the materials used for the robot could be wasteful. For this reason, the robot was made as power efficient as possible to have the least negative impact on the environment as possible. The waste of materials was minimized by using rechargeable batteries and thoroughly planning the design of the chassis to avoid multiple trial prints resulting in wasted material.

Manufacturability and sustainability were considered in the design and implementation portions of the project. The economic constraints led to a design that has a definite price for each robot in the case of manufacturing and price may be subject to slight changes in the case of bulk ordering components. Manufacturability was also considered when designing the portions of the project that are 3-D printed. These parts were thoroughly planned before printing to ensure less trial prints as well as using the most material efficient design.

The ethical portion of the project in all stages is introduced with the obstacle of the dog. Instead of potentially harming the dog by trying to pass through, or potentially over, the dog our robot finds an alternative route to ensure no harm is done to a living creature. The contest introduces the ethical constraint of prioritizing human and animal life over losing physical possessions.

The motors for this robot that are attached to the wheels have the potential of being dangerous due to their maximum RPM. The robot could potentially start driving around very quickly and pose a health and safety risk if the robot runs into someone or something. The robot utilizes accurate range finders for detecting any potential animal, person, or wall in the way to ensure the safety of all life in the house. The range finders are also used for shutting down the power to the motors in the situations where the robot is quickly approaching a living creature or wall. The last precaution to ensure health and safety is a kill switch that is easily accessible on the top of the robot.

The other safety constraint in the project is the risk of fire. The robot can find and effectively put out the fire without causing more harm such as spreading the flame using a poor method of extinguishing or by knocking the candle over. As far as avoiding knocking over the candle our program is designed to extinguish the flame from as far away as possible. Once the

UV Tron has detected the flame in the room the robot will spin until the UV Sensors have locked on the flame source to orient the robot to face the flame. Next the robot begins a loop of inching forward until the distance sensor is tripped or both UV Sensors are tripped meaning the robot is within extinguishing range. The robot then turns on its drone motor to put out the flame. The robot will continue this loop until the UVTron no longer detects the flame.

V. Results

We represented the University of Evansville at the Trinity College Fire Fighting Robot Competition on April 13th and 14th. The robot passed the initial judges table inspection and was awarded a participation award for being compliant with all the competition rules and requirements. On the first day of the competition the gymnasium lights caused lots of interference with our distance sensors causing us to redesign the program to work even with random sensor trips. We were successful on our first attempt at level one and accomplished the arbitrary start, candle location, and return to start multipliers giving us a score of 24.44 for that level. We attempted level two to round out the first day however the robot got caught suspended in the air on the ball casters and the wheels were unable to propel the robot. On day two we made slight hardware adjustments to try and allow the robot to shift off the carpets. We removed one of the spacers on the back-ball caster and shifted it back more on the robot chassis to allow the robot to rock back and forth more. Trial three we managed to completely make it around the maze and over the carpets but upon entering the room containing the fire the line sensors missed the line and therefore the robot did not know to check for the flame and was unsuccessful in putting out the flame. Trial four we were finally successful on level 2. The time for the run was long because the robot got hung up on the carpet for a while but managed to rock off it. We accomplished the arbitrary start multiplier for this run giving us a score of 109. Since we had no visual recognition components we decided to use our last trial on level two again to try for a lower score. The robot ran into the candle however and failed the final trial. Wrapping up the weekend our robot placed third in the senior unique division and first out of the North American senior unique robots. We also placed first in the poster and presentation competition. Final pictures of the robot can be seen in Appendix D.

VI. Costs

The budget for this project is broken into two different sections. The first portion of the budget is travel expenses for the team to be able to travel to the competition. This portion of the project costs pitched to the Academic Fund Board (AFB). The second part of the budget is for all the components that the project required. Some portions of the project were reused parts from around Koch center and from extras of previous robots to lower costs. The total estimated costs broken down for the travel and robot construction costs can be seen in Table 2. In conclusion the UE Fire Fighting Home Robot Team has been funded the total $1000 requested for the project budget and $1853.96 for travel.

Table 4: Travel Budget

Travel (for two)	
Description	**Cost ($)**
Competition Registration Fee	90.32
Team shirts	52
Air Flight	1,132.62
Checked Bag	60
Hotel	275.98

Car Rental	68.01
Fuel	30
Poster for Competition	25
LiPo Batteries (if not allowed on plane)	120
Food	60
Total	**$1,913.93**

Table 5 b6: Robot Budget

Robot	
Description	**Cost ($)**
Motors (x2)	79.90
Motor Drivers	23.49
Motor Brackets	7.95
Mounting Hubs	7.95
Ball Caster	4.99
Quadcopter Motor	7.99
Wheels	10.90
IR Sensor	49.74
IR Jumper Wire	9.00
Mic	5.95
11.1 V Battery	38.75
XT-60 Battery Connector	7.25
Buck Converter	12.99
Wheels	6.99
Wheels	13.98
UV Sensor	7.89
Line Sensor	5.90
9V Battery Snap	1.30
IR Sensor	9.55
LED Assortment	6.15
IR Jumper Wire	1.50
PCB	55.57
9V	5.99
TRX Converter	6.29
Corner Braces	7.89
11.1 V Battery	48.99
Ball Caster	4.99
Total	**$490.28**

Overall Total = $2,404.21

Received Funding from AFB: $1,853.96

Received Funding from CECS: $1000

Remaining Funding from CECS: $509.72

VII. Conclusion and Recommendations

As a team we had many successes with this senior project and placed better than we could have ever hoped at the competition. We managed to get the robot done and attend the competition through all the setbacks and delays. The team placed third out of the senior unique division and first out of the North American senior unique robots by points. We were also the winners of the poster presentation competition with a $200 cash prize.

Looking back and thinking about what could have been done differently to make recommendations for potential future teams we thought of a few major things. First the main problem we had with the robot was the line sensors used to detect the rooms. They had to be too close to the ground and would constantly clip on the rugs or just completely not see the lines. It would be best to find a way to function the robot without line sensors or find more reliable ones. Second our motors were too large and heavy. There are many smaller options that can be used and improve the robot's overall speed. Third UVTrons are outdated and we may have been one of the only teams at the competition using one. Most teams used a pyro sensor instead. Lastly incorporating a gyro into the robot to help with turning the robot and overall navigation would make calibration much easier. Instead of just throwing a set number into delays once the motors are spinning opposite directions to get roughly a ninety degree turn the gyro would allow you to always turn exactly the amount you want.

VIII. IEEE Safety Standard Considered:

“IEEE C37.14-2002 - IEEE Standard for Low-Voltage DC Power Circuit Breakers Used in Enclosures”

This standard was considered in the production of the firefighting robot, because by the competition rules the robot is required to have 78ijjsome sort of kill switch to easily shut down the robot’s motor function. The easiest way to replicate a kill switch is setting up a circuit replicating a circuit breaker to shut down all power supply to the motor. A circuit breaker type circuit will be implemented to act as the kill switch on the robot.

Appendix A

```
%TCFFRC Mic Elliptic Bandpass Filter Matlab Code    By:Conner Sheets
fs = 16000;
fpass = [3400 4350]; %Rules specify start freq. = 3.8kHz +/- 13%
fstop = [3300 4550]; %Not sure where to put the stop at yet...
Rpass = 0.01;RpassDB = -20*log10(1-Rpass);
Rstop = 0.01;RstopDB = -20*log10(Rstop);
[NE fp] = ellipord(fpass/(fs/2), fstop/(fs/2),RpassDB,RstopDB);
[numE denE] = ellip(NE, RpassDB, RstopDB, fpass/(fs/2));
[HE f] = freqz(numE, denE, 1024, fs);
fprintf(1,['Elliptic Order is ' num2str(NE) '\n']);

fpass1 = [3300 4300]; %Rules specify start freq. = 3.8kHz +/- 13%
fstop1 = [3100 4500]; %Not sure where to put the stop at yet...
Rpass1 = 0.01;RpassDB1 = -20*log10(1-Rpass1);
Rstop1 = 0.01;RstopDB1 = -20*log10(Rstop1);
[NE1 fp1] = ellipord(fpass1/(fs/2), fstop1/(fs/2),RpassDB1,RstopDB1);
[numE1 denE1] = ellip(NE1, RpassDB1, RstopDB1, fpass1/(fs/2));
[HE1 f1] = freqz(numE1, denE1, 1024, fs);

figure(1);
plot(f, abs(HE),"r-*",10);
hold on;
axis([2600 5000 0 1.1]);
%plot(f1, abs(HE1));
title('Frequency Vs. Magnitude');
xlabel('Frequency');
ylabel('Magnitude');
SOS = tf2sos(numE,denE)
```

Appendix B

```
#include "stm32f446.h"
#include <stdint.h>
#include <math.h>
#include <stdlib.h>

/*
This is the finished code for the 2019 Trinity College Fire Fighting Robot
College Team: University of Evansville
Programmers: Conner Sheets and Jared Sutphin
Date of Competition: April 13th - 14th
*/

void Setup(void);
void Tim3Setup(void);
void TurnLeft90(void);
void TurnRight90(void);
void DMASetup(void);
void DataToCm(void);
void Left_Mtr(signed int speed);
void Right_Mtr(signed int speed);
void Delay(unsigned int i);
void CheckObstruction(void);
void ExploreMazeFrontRightSensor(void);
void ExploreMazeBackRightSensor(void);
void MotorStop(void);
void CheckRoom(void);
void CheckLine(void);
void AlignOnLine(void);
void ExitStartRoom(void);
void ExitAlignOnLine(void);
void AlignAfterTurn(void);
void ScanForFlame(void);
void FindFlame(void);
void Extinguish(void);
void LeaveRoom(void);
void MicSetup(void);
void Start_Sound(void);
void Check_Mic_Input(void);
```

```
void candleInDoorwayCheck(void);
uint16_t ADC_Data[8];
uint16_t ADC_DataCm[4];
uint16_t ClosestWall = 0x32;
uint16_t uv = 0, uv2 = 0;
int FIRE = 0, count = 0, cMax = 20, yInt, maxY;
int x, UV_Tron_Tick = 0, StartSound = 0, ActualObstacle,FrontRight, InRange, Alignable;
int ClosestWallDegree, CandleFound = 0, LineCount = 0, ReturnHomeLineCount = 5,
ReturningHome = 0;
int ScanDirection = 0, FirstCheck = 0, FalseStop = 0;
int SecondCheck, InsideRoom = 1, BreakVariable = 0, RightDelay, LineAligned = 0;
int rotationDirection = 0, flameAhead = 0, OffSet = 1, MoveForwardReady = 0;
int ClosestDist, DistCheck, TooClose, TooFar, Opening, LineAlignBypass = 0;
int main()
{


    Setup();        //Call Function for setup of the ADC
    DMASetup(); //Call Function for setup of the DMA
    Tim3Setup(); //Call Function for setup of Timer3
    MicSetup();   //Setup all needed registers for Mic Operation
    ADC1_CR2 |= 0x300; //DMA keeps requesting (DDS) and DMA Enable


while(1)
{
        while(StartSound == 0)     //Loop to keep checking mic until correct freq. detected
        {
        Start_Sound();
        }
```

```
while(InsideRoom == 1)   //calloc function for Arbitrary Start
{
ExitStartRoom();
}

ADC1_CR2 |= (1<<30);        //ADC1 Start for all channels
DataToCm();          //Converts AD data from dist sens to cm
if(ADC_DataCm[2] <= 0x14) //If Front Right is within Range of a Wall
{
 ExploreMazeFrontRightSensor();//Follow wall with front right Sensor
 }
 else if(ADC_DataCm[2] > 0x14) //If Front Right is not within Range of wall
 {
 ExploreMazeBackRightSensor(); //Follow wall with back right Sensor
 }
 CheckObstruction();         //Function for checking in front of robot
 }
}

void Setup()
{
RCC_AHB1ENR |= 0x3F;      //Enable GPIOA/B/C Clock Bits
GPIOA_MODER |= 0x3F0F;  //PA[0,1,4,5,6] are Analog
GPIOB_MODER |= 1<<(2*10);
GPIOB_MODER &= 0;         //UVTron Input Pin
GPIOB_PUPDR |= 1<<(2*8);//Pulling up PB8 because UVTron will pull it low when fire detected
GPIOB_MODER |= 1<<(2*14);        //UVTron LED Indicator Pin
GPIOB_MODER |= 1<<(2*15);        //UVTron LED Indicator Pin
GPIOB_MODER |= 0xF;       //PB[0,1] Dist_Sens
GPIOC_MODER |= 0xFF;      //PC[0,1,2,3] are analog
GPIOB_MODER |= 1<<(2*9);          //Propeller output
ADC_CCR |= 0x30000;         //PCLK Divided by 8
RCC_APB2ENR |= (1<<8);    //ADC1 Clock Enable
ADC1_CR2 |= 0x1;  //Enable ADC1
ADC1_CR1 |= (1<<8);          //Scan Mode Enabled ***Page 385 Ref. Manual***
ADC1_SQR1 |= 0x700000;  // Regular Channel Sequence Length: 8 for 9 ADCs ***Mic is on ADC2***
```

```
                                        //SQR1 change above is always 1 less than amount of conversions
                                        ADC1_SQR3 |= 0<<(5*0);    //Dist_Sens[0] Front Right
                                        ADC1_SQR3 |= 1<<(5*1);    //Dist_Sens[1] Front Middle
                                        ADC1_SQR3 |= 6<<(5*2);    //Dist_Sens[2] Right Front
                                        ADC1_SQR3 |= 8<<(5*3);    //Dist_Sens[3] Right Back
                                        ADC1_SQR3 |= 9<<(5*4);    //Dist_Sens[4] UV Sensor
                                        ADC1_SQR3 |= 10<<(5*5);   //Dist_Sens[5] UV Sensor
                                        ADC1_SQR2 |= 11<<(5*0);   //Left Line Sensor
                                        ADC1_SQR2 |= 12<<(5*1);   //Right Line Sensor
                                        }

void Tim3Setup(void)
{
//GPIO Setup
GPIOB_MODER |= 1<<(2*5);    //PB5 Output for Direction 1
GPIOB_MODER |= 1<<(2*6);    //PB6 Output for Direction 2
GPIOC_MODER |= 2<<(2*7);    //PC7 Set to Alternate Function
GPIOC_AFRL |= 0x20000000;   //PC7 Set to Alt. Funct. 2 - TIM3 Ch2
GPIOC_MODER |= 2<<(2*6);    //PC6 Set to Alternate Function
GPIOC_AFRL |= 0x2000000;    //PC6 Set to Alt. Funct. 2 - TIM3 Ch1
GPIOC_MODER |= 2<<(2*8);    //PC8 Set to Alternate Function
GPIOC_AFRH |= 0x2;   //PC8 Set to Alt. Funct. 2 - TIM3 Ch3
//Timer3 Setup
RCC_APB1ENR |= 1<<1;        //Timer 3 clock enable
TIM3_CCMR1 |= 0x6C6C;       //PWM mode output compare 1, preload and fast enable for Ch1,2
TIM3_CCMR2 |= 0x6C;//PWM mode output compare 1, preload and fast enable for Ch3
TIM3_CR1 |= (1<<7);   //ARPE Pg 526
TIM3_PSC |= 15;       //16 Mhz/15+1 = 1 MHz
TIM3_ARR |= 19999;    //PWM Period = (19999 + 1) * (1/1Mhz) = .02Sec
TIM3_CCR1 |= 0;       //Duty cycle starts at 0
TIM3_CCR2 |= 0;       //Duty cycle starts at 0
TIM3_CCR3 |= 0;       //Duty cycle starts at 0
TIM3_CCER |= 0x111; //Capture/Compare 1 output enable for Ch1,2,3
TIM3_EGR |= 1;        //Update generation
TIM3_CR1 |= 1;        //Counter enabled

GPIOB_BSRR = 1<<6;        //PB6 High (Left Mtr Forward) PB5 Low (Right Mtr Forward)
}
```

```
void TurnLeft90(void)
{
Left_Mtr(-50);      //Mtrs turned on in opp directions
Right_Mtr(50);
Delay(19500);       //Delay obtained from testing for 90 degree turn
MotorStop();
}

void TurnRight90(void)
{
Left_Mtr(50);       //Mtrs turned on in opp directions
Right_Mtr(-50);
Delay(19500);       //Delay obtained from testing for 90 degree turn
MotorStop();
}

void DMASetup()     //Function for setting up DMA
{
RCC_AHB1ENR |= (1<<22);  //DMA2 Clock Enable Channel 0 Stream 0 or 4 is ADC1 ***Page 207***
//DMA2_S0CR Bit 25-27 Chan. Sel. Default to 000 which is Chan. 0 and SxCR -> x = Stream #
DMA2_S0CR |= (1<<17);      // Priority Level High
//Set Peripheral Data Size: ADC_DR = 16bits
DMA2_S0CR |= (1<<11);      // 01 = 16 Bits setting
//Set Memory Data Size: Match the Peripheral = 16 Bits
DMA2_S0CR |= (1<<13);      //01 = 16 Bits
//Peripheral Increment Mode or Memory Increment Mode
DMA2_S0CR |= (1<<10);      //Memory Increment After Each Data Transfer
//Circular Mode ***Page 212***
DMA2_S0CR |= (1<<8);
//Set Transfer Direction... Want: Peripheral -> Memory
//DMA2_S0CR |= 00 in bit 6,7 but defaults to that so commented out
```

```c
//# of Data Sequences to Transfer: 7 for the Distance sensors
DMA2_S0NDTR = 8;//Number for amount of ADC Conversions
//Link DMA to Peripheral (ADC1_DR)
DMA2_S0PAR = 0x4001204C;          //The address of ADC1_DR
//Memory Address Register
DMA2_S0M0AR = (uint32_t)ADC_Data;// The register is 32 bits so cast to 32???
DMA2_S0CR |= 0x1;          //DMA2 Channel Enable
}
void DataToCm()
{
for(int i=0;i<5;i++)
{
//Rounded to 50: Data->cm Equations derived using Vref = 3V
ADC_DataCm[i] = (17350/ADC_Data[i]) - 0.42;
}
}
void Left_Mtr(signed int speed)       //This funct. just allows speed of Mtr to be set
{
if(speed > 0)
{
GPIOB_BSRR |= (1<<6);
TIM3_CCR1 = (uint16_t)(320000.0*speed/100.0);
}
if(speed < 0)
{
GPIOB_BSRR |= (1<<22);
speed = abs(speed);
TIM3_CCR1 = (uint16_t)(320000.0*speed/100.0);
}
if(speed == 0)
{
TIM3_CCR1 = (uint16_t)(0);
}
}
void Right_Mtr(signed int speed)      //This funct. just allows speed of Mtr to be set
{

if(speed > 0)
```

```
{
GPIOB_BSRR |= (1<<21);
TIM3_CCR2 = (uint16_t)(320000.0*speed/100.0);
}
if(speed < 0)
{
GPIOB_BSRR |= (1<<5);
speed = abs(speed);
TIM3_CCR2 = (uint16_t)(320000.0*speed/100.0);
}
if(speed == 0)
{
TIM3_CCR2 = (uint16_t)(0);
}
}
void MotorStop(void)          //Function to cut off the motors
{
TIM3_CCR1 = (uint16_t)(0);
TIM3_CCR2 = (uint16_t)(0);
}
void CheckObstruction()       //Function to check in front of the robot for obstacle
{
    if(InsideRoom == 0)     //Set closest range based on where robot is
    {
    ClosestDist = 0xA;
    DistCheck = 10;
    }
    else if(InsideRoom == 1)
    {
    ClosestDist = 0x8;
    DistCheck = 6;
    }
    if((ADC_DataCm[1] <= ClosestDist) || ((ADC_DataCm[1] <= ClosestDist) &&
    (ADC_DataCm[0] <= ClosestDist)))
    {
    MotorStop();
    ADC1_CR2 |= (1<<30);
    DataToCm();
    if((ADC_DataCm[1] <= 0xF) && (ADC_DataCm[0] <= 0xF)) //Is wall going Alignable after
    turn
    {
    Alignable = 1;
    }
    else
```

```
{
Alignable = 0;
}

for(int i = 0; i < DistCheck; i++)        //Takes 10 trips to try and avoid false obstacle
readings
{
ADC1_CR2 |= (1<<30);
DataToCm();
if((ADC_DataCm[1] <= ClosestDist) || ((ADC_DataCm[1] <= ClosestDist) &&
(ADC_DataCm[0] <=
ClosestDist)))
{
ActualObstacle++;
}
else if((ADC_DataCm[1] > ClosestDist) && (ADC_DataCm[0] > ClosestDist))
{
ActualObstacle = 0;
break;
}
}
if(ActualObstacle >= DistCheck)      //If there truly is an obstacle turn and align if
possible
{
ActualObstacle = 0;
TurnLeft90();
if(Alignable == 1)
{
AlignAfterTurn();
}
ADC1_CR2 |= (1<<30);
DataToCm();
CheckObstruction();
}
}
else          //No obstacle continue on with Right Wall Following
{
ActualObstacle = 0;
ADC1_CR2 |= (1<<30);
```

```
DataToCm();
if(ADC_DataCm[2] <= Opening)
{
ExploreMazeFrontRightSensor();
}
else if(ADC_DataCm[2] > Opening)
{
ExploreMazeBackRightSensor();
}
}
}
void ExploreMazeFrontRightSensor(void)
{
ADC1_CR2 |= (1<<30);
DataToCm();
if(ADC_Data[4] <= 0x200)        //This loop was added at Comp. To try and still see fire even if a Line was missed
{
for(int b=0; b<5; b++)  //If the sensitive UV Sensor is tripped Bypass the need to find a line and begin
{        //the CheckFlame and FindFlame process
ADC1_CR2 |= (1<<30);
DataToCm();
uv = uv + ADC_Data[4];
}
uv = uv/5;
if(ADC_DataCm[1] <= 0x7)
```

```
{
uv = 0xFFF;
}
if(uv <= 0x200)
{
LineAlignBypass = 1;
uv = 0;
AlignOnLine();
LineAlignBypass = 0;
}
}
if(InsideRoom == 0)     //If no flame set paramaters for wall following 10-11cm is ideal outside rooms
{
TooClose = 0xA;
TooFar = 0xB;
Opening = 0x10;
}
else if(InsideRoom == 1) //If no flame set paramaters for wall following 8-9cm is ideal inside rooms
{
TooClose = 0x8;
TooFar = 0x9;
Opening = 0x10;
}
if(ADC_DataCm[2] <= TooClose)          //Robot is getting too close to wall
        {
Left_Mtr(50); //Slow down left Motor to correct away from wall
Right_Mtr(75);          //Leave Right at 75        //Convert to Cm
ADC1_CR2 |= (1<<30);
DataToCm();
CheckLine();
}
else if(ADC_DataCm[2] >= TooFar && ADC_DataCm[2] <= Opening)      //Robot is getting too far from wall
{
Left_Mtr(75); //Left Motor Left at 75 to correct towards wall
Right_Mtr(50);          //Drop Right Motor Speed to correct
ADC1_CR2 |= (1<<30);
DataToCm();
```

```
        CheckLine();
        }
        else if(ADC_DataCm[2] > Opening)      //Front right sensor suddenly jumps to large distace... Means door or hall
        {
        ExploreMazeBackRightSensor();        //Start Following wall off back right sensor
        }
        else if(ADC_DataCm[2] > TooClose && ADC_DataCm[2] < TooFar)
        {       //MotorStop is within desired distance
        Left_Mtr(75); //Set Left to 75% speed
        Right_Mtr(75);          //Set Right to 75% speed
        ADC1_CR2 |= (1<<30);
        DataToCm();
        CheckLine();
        }
        }

void ExploreMazeBackRightSensor(void)   //Front Right is out of range use back right
{
if(ADC_DataCm[3] < 0x8) //Robot too close to wall
{
 Left_Mtr(50);    //Slow down left Motor to correct away from wall
 Right_Mtr(75);   //Leave Right at 75                                   //Keep at 75%
    ADC1_CR2 |= (1<<30);
    DataToCm();
    }
    else if((ADC_DataCm[3] > 0x10) && (ADC_DataCm[3] <= 0x14))
    {
    Left_Mtr(75);        //Left Motor Left at 75 to correct towards wall
    Right_Mtr(50);       //Drop Right Motor Speed to correct
    ADC1_CR2 |= (1<<30);
    DataToCm();
    }
    else if(ADC_DataCm[3] > 0x14)
    {
    Left_Mtr(50);        //Continue forward slightly to be fully within gap
     Right_Mtr(50);
      Delay(10000);
      MotorStop();       //Kill motor to prepared for turn
      TurnRight90();     //Turn Right to progress into room or hallway
     Delay(5000);
            Left_Mtr(30);   //Motors to 30% speed to slowly search for a line or reattach to wall
            Right_Mtr(30);
```

```
for(int i = 0; i < 50000; i++)	//Loop to constantly be checking for Line or if no line wall follow
{
ADC1_CR2 |= (1<<30);
DataToCm();
if(ADC_Data[6] < 0xC00 || ADC_Data[7] < 0xC00)	//If either line sensor is tripped
{
if(InsideRoom == 1)
{
ExitAlignOnLine();
if(LineAligned == 1)
{
LineAligned = 0;
break;
}
else
{
Left_Mtr(30);
Right_Mtr(30);
}
}
else if(InsideRoom == 0)
{
AlignOnLine();
if(LineAligned == 1)
{
LineAligned = 0;
break;
}
else
{
Left_Mtr(30);
Right_Mtr(30);
}
}
}

	else if(ADC_DataCm[1] <= 0xA && ADC_DataCm[0] <= 0xA)
	{
	ActualObstactle = 0;
```

```
for(x = 0; x < 15; x++)
{
ADC1_CR2 |= (1<<30);
DataToCm();
if(ADC_DataCm[1] <= 0xA && ADC_DataCm[0] <= 0xA)
{
ActualObstactle++;
}
else if(ADC_DataCm[1] > 0xA || ADC_DataCm[0] > 0xA)
{
Left_Mtr(30);
Right_Mtr(30);
ActualObstactle = 0;
break;
}
}
if(ActualObstactle >= 15)
{
ActualObstactle = 0;
CheckObstruction();
break;
}
else if(ActualObstactle < 15)
{
Left_Mtr(30);
Right_Mtr(30);
ActualObstactle = 0;
}
}
}
}
else if(ADC_DataCm[3] >= 0x8 && ADC_DataCm[3] <= 0x10)
{       //Wall is within the desired range
Left_Mtr(75);           //Set Left to 75% speed
Right_Mtr(75);          //Set Right to 75% speed
ADC1_CR2 |= (1<<30);
DataToCm();
}
}

                    void CheckRoom(void)
                    {
```

```
int z = 1300;
if(SecondCheck == 1)
{
z = 1000;
}
for(x=0;x<45000;x++)
{
if((GPIOB_IDR & 0x100) != 0x100) //UVTron Pulls Pins Low
{
UV_Tron_Tick++;            //Increment the Tick Counter for UVTron and X because it will get stuck in here if it keeps seeing flame
x++;
}
if((GPIOB_IDR & 0x100) == 0x100) //UVTron not detecting flame reset count
{
}
}
if(UV_Tron_Tick >= z)      //This is the amount of ticks the UV_Tron was detecting fire
//Increasing this adjusts makes the UV_Tron harder to trigger
{
x = 0;      //Reset Counter
UV_Tron_Tick = 0;
FIRE = 1;//Returns 1 if fire has been found
}            //Reset the UV_Tron_Tick
else
{
x = 0;
UV_Tron_Tick = 0;
FIRE = 0;//Returns 0 if fire not found
}
}
```

```
void CheckLine(void)
{
if(ADC_Data[6] < 0xC00 || ADC_Data[7] < 0xC00)          //If either line sensor is tripped
{
if(InsideRoom == 1)       //Call the correct Function to Leave or Check Room
{
ExitAlignOnLine();
}
else if(InsideRoom == 0)
{
AlignOnLine();
}
}
}
void AlignOnLine(void)
{
MotorStop();
Delay(2000);
for(int i = 0; i < 11600; i++)
{
ADC1_CR2 |= (1<<30);
DataToCm();
if(ADC_Data[6] <= 0xC00 && ADC_Data[7] > 0xC00)      //If left line sens tripped but right not
{
if(ADC_Data[7] > 0xC00)         //Adjust right mtr to put right sens over line
{
Right_Mtr(10);
Left_Mtr(0);
}
}
else if(ADC_Data[6] > 0xC00 && ADC_Data[7] <= 0xC00)//If Right Line sens tripped but left not
{
if(ADC_Data[6] > 0xC00)         //Adjust left mtr to put left sens over line
{
Right_Mtr(0);
Left_Mtr(10);
}
}
else if(ADC_Data[6] < 0xC00 && ADC_Data[7] < 0xC00 || LineAlignBypass == 1)
//If both sensors are over line or UV Sensor has detected flame and
```

```
bypassing need to find a line
{
LineAligned = 1;
if(ReturningHome == 1)
{
ReturnHomeLineCount = ReturnHomeLineCount - 1; //If returning home and line found decrement amount of lines needed until Home
if(ReturnHomeLineCount <= 0)            //If Last Line has been found meaning at start room Go Inside and wait forever
{
Left_Mtr(75);
Right_Mtr(75);
Delay(14000);
MotorStop();
while(1);
}
else if(ReturnHomeLineCount > 0)        //If not back at the Start Room Exit Room and Keep searching
{
LeaveRoom();
}
}
else if(ReturningHome == 0)     //If not returning home
{
LineCount++;  //Increment the amount of lines found
MotorStop();  //Kill Motors
Delay(2000);
CheckRoom();//Check room for UVTron trip if line detected
if(FIRE == 1)     //If Fire is detected
{
ReturnHomeLineCount = 5 - LineCount;           //This point we can determine amount of lines left to cross to get back home
if(LineAlignBypass != 1)
{
candleInDoorwayCheck();
}
if(flameAhead == 1 && LineAlignBypass != 1) //flameAhead checks to see if room can be fully or only half entered
```

```
{
Left_Mtr(75); //Enter room Half Because Candle is obstructing full entrance
Right_Mtr(75);
Delay(13500);
MotorStop();
}
else if(flameAhead == 0 && LineAlignBypass != 1) //FlameAhead check confirms that robot can fully enter room
{
Left_Mtr(75); //Enter room fully because candle is not obstructing entrance
Right_Mtr(75);
Delay(23500);
MotorStop();
}
flameAhead = 0;         //Reset the flameAhead variable
GPIOB_ODR |= (1<<14);         //Turn on LED to signal fire found
InsideRoom = 1;         //Signal that the robot is inside of a room
while(CandleFound == 0)         //While the Candle has not been found
{
ScanForFlame();         //Scan the room using Sensitive UV Sensor for flame location
if(BreakVariable == 0)
{
FindFlame();   //Inch closer to the flame
}
}
while(FIRE!= 0)         //While the fire has not been extinguished
{
Delay(5000);
CheckRoom();//Check the UVTron to see if Fire still present
if(FIRE != 0)
{
```

```
BreakVariable = 0;      //Reset Variables to allow checking and honing in on flame
CandleFound = 0;
SecondCheck = 1;
while(CandleFound == 0)        //Continue searching for the candle until found
{
ScanForFlame();        //Scan the room using Sensitive UV Sensor for flame location
if(BreakVariable == 0)
{
FindFlame();   //Inch closer to the flame
}
}
}
else if(FIRE == 0)        //If UVTron is not sensing flame anymore
{
GPIOB_ODR &= ~(1<<14);       //Turn off RED LED
ReturningHome = 1;   //Set ReturningHome Variable to change program function for going home
while(InsideRoom == 1)
{
ExitStartRoom();        //Robot needs to get out of room so reuse ExitStartRoom
}
ExploreMazeFrontRightSensor();
break;
}
}
}
else if(FIRE == 0)        //If fire is not initially detected following Line Trip
{
LeaveRoom(); //Back out and leave room to continue maze exploration
```

```
break;
}
}
}
else if(i == 11599)       //If the Robot has searched for so long to line up line sensors and not finding line with
{       //other sensor probably a false trip so move forward and Wall follow again
for(int i = 0; i < 25000; i++)
{
Left_Mtr(30);
Right_Mtr(30);
if(ADC_DataCm[1] <= 0xA && ADC_DataCm[0] <= 0xA)
{
ActualObstactle = 0;
for(x = 0; x < 15; x++)
{
ADC1_CR2 |= (1<<30);
DataToCm();
if(ADC_DataCm[1] <= 0xA && ADC_DataCm[0] <= 0xA)
{
ActualObstactle++;
}
else if(ADC_DataCm[1] > 0xA || ADC_DataCm[0] > 0xA)
{
Left_Mtr(30);
Right_Mtr(30);
ActualObstactle = 0;
break;
}
}
if(ActualObstactle >= 15)
{
ActualObstactle = 0;
CheckObstruction();
break;
}
else if(ActualObstactle < 15)
{
Left_Mtr(30);
Right_Mtr(30);
```

```
ActualObstactle = 0;
}
}
}
MotorStop();
if(ADC_DataCm[2] <= 0x14)     //If Front Right is within Range of a Wall
{
ExploreMazeFrontRightSensor();//Follow wall with front right Sensor
}
else if(ADC_DataCm[2] > 0x14) //If Front Right is not within Range of wall
{
ExploreMazeBackRightSensor();//Follow wall with back right Sensor
}
break;
}
}
}



void ExitStartRoom(void)
{
InsideRoom = 1;
ClosestWall = 0x99;
ClosestWallDegree = 0;
for(int i = 0; i < 10; i++)          //Spin and take 10 measurements so the robot can find the closest wall to drive to
{
ADC1_CR2 |= (1<<30);
DataToCm();
if(ADC_DataCm[1] <= ClosestWall)
{
ClosestWall = ADC_DataCm[1];
ClosestWallDegree = i;
}
Left_Mtr(-50);
Right_Mtr(50);
```

```
            Delay(8050);
            MotorStop();
        }
        if(ClosestWallDegree > 4)
        {
        for(int x = 9; x > ClosestWallDegree; x--)  //Decide quickest way to spin back to face closest wall
        {           //Idea is to reverse the spin to go back to facing original closest wall

            Left_Mtr(50);
            Right_Mtr(-50);
            Delay(8050);
            MotorStop();
        }           //Robot should now be facing the closest wall
        }
        else if(ClosestWallDegree <= 4)   //Decide quickest way to spin back to face closest wall
        {
        for(int x = 0; x < ClosestWallDegree; x++)
        {           //Idea is to reverse the spin to go back to facing original closest wall

            Left_Mtr(-50);
            Right_Mtr(50);
            Delay(8050);
            MotorStop();
            } //Robot should now be facing the closest wall
            }
            for(int z = 0; x < 200000; z++)       //This for loop is to drive towards the wall and breaks when within range
            {
            ADC1_CR2 |= (1<<30);
            DataToCm();
            if(ADC_DataCm[1] <= 0xA && ADC_DataCm[0] <= 0xA)
            {
            ActualObstactle = 0;
            for(int i = 0; i < 10; i++)     //Take 10 checks to avoid false obstacle trips
            {
            ADC1_CR2 |= (1<<30);
            DataToCm();
            if(ADC_DataCm[1] <= 0xA && ADC_DataCm[0] <= 0xA)
```

```
{
ActualObstacle++;
}
else if(ADC_DataCm[1] > 0xA || ADC_DataCm[0] > 0xA)
{
ActualObstacle = 0;
}
}
if(ActualObstacle >= 10) //If the front wall is within range turn the robot and align using the wall
{
MotorStop();
ActualObstacle = 0;
TurnLeft90();
AlignAfterTurn();
ADC1_CR2 |= (1<<30);
DataToCm();
break;
}
else if(ActualObstacle < 10)        //If wall not within range reset variable and continue progressing forward
{
ActualObstacle = 0;
}
}
Left_Mtr(50);
Right_Mtr(50);
ADC1_CR2 |= (1<<30);
DataToCm();
}
while(InsideRoom == 1)   //Once robot has found a wall to attach to right wall follow until doorway found
{
ExploreMazeFrontRightSensor();
CheckObstruction();
}
}
void ExitAlignOnLine(void)
{
MotorStop();
Delay(2000);
for(int i = 0; i < 15000; i++)   //Once line at doorway of room has been found
```

```
Align on it
        {
        ADC1_CR2 |= (1<<30);
        DataToCm();
        if(ADC_Data[6] <= 0xC00 && ADC_Data[7] > 0xC00) //If left line sens tripped but right not
        {
        if(ADC_Data[7] > 0xC00)     //Adjust right mtr to put right sens over line
        {
        Right_Mtr(10);
        Left_Mtr(0);
        }
        }
        else if(ADC_Data[6] > 0xC00 && ADC_Data[7] <= 0xC00)//If Right Line sens tripped but left not
        {
        if(ADC_Data[6] > 0xC00)     //Adjust left mtr to put left sens over line
        {
        Right_Mtr(0);
        Left_Mtr(10);
        }
        }
        else if(ADC_Data[6] < 0xC00 && ADC_Data[7] < 0xC00) //If both sensors are over line
        {
        MotorStop();        //Kill Motors
        LineAligned = 1;
        Delay(2000);
        Left_Mtr(50);
        Right_Mtr(50);
        Delay(7000);        //Turn Motors on Long Enough Just to Get off the white line
        MotorStop();
        InsideRoom = 0;     //Changes which Align function will start being called
        break;      //Robot has made it outside of the room at this point RightWallFollow now
        }
        }
        }

void AlignAfterTurn(void)
```

```
{
uint16_t FrontRight;
uint16_t BackRight;
        for(int i = 0; i < 20000;i++)       //Loop to line robot up with wall to help smoothen out after turns
        {
        ADC1_CR2 |= (1<<30);
        DataToCm();
        FrontRight = ADC_DataCm[2] - OffSet;
        BackRight = ADC_DataCm[3];
        if(FrontRight > BackRight)       //Front Right is closer than Back Right
        {
        Right_Mtr(-7);
        Left_Mtr(7);
        }
        else if(FrontRight < BackRight)//Front Right is closer than Back Right
        {
        Right_Mtr(7);
        Left_Mtr(-7);
        }
        else if(FrontRight == BackRight)          //Front Right and Back Right at equal distance
        {
        MotorStop();
        break;
        }
        }
        }

  void ScanForFlame(void)
  {
  int c = 0;
  if(rotationDirection % 2 == 0) //Alternate the direction the robot spins to scan
  {
            Left_Mtr(-50);
            Right_Mtr(50);
            Delay(10000);
            MotorStop();
            if(BreakVariable == 0)
            {
            for(int a=0; a<1000000; a++) //Rotate right until uv is found
            {
            uv = 0;
            uv2 = 0;
            Left_Mtr(7);
```

```
Right_Mtr(-7);
for(int b=0; b<5; b++)
{
ADC1_CR2 |= (1<<30);
DataToCm();
uv = uv + ADC_Data[4];
if(b%2 == 0)
{
uv2 = uv2 + ADC_Data[5];
}
}
uv = uv/5;
if(ADC_DataCm[1] <= 0x7)
{
uv = 0xFFF;
}
uv2 = uv2/3;
if(uv2 < 0x200)
{
MotorStop();
Extinguish();
BreakVariable = 1;
CandleFound = 1;
break;
}
if(uv < 0x200)
{
MotorStop();
break;
}
}
}
if(BreakVariable == 0)
{
for(int a=0; a<1000000; a++) //Rotate right until uv is not found
{
uv = 0;
uv2 = 0;
Left_Mtr(7);
Right_Mtr(-7);
for(int b=0; b<5; b++)
{
ADC1_CR2 |= (1<<30);
```

```
DataToCm();
uv = uv + ADC_Data[4];
if(b%2 == 0)
{
uv2 = uv2 + ADC_Data[5];
}
c++;
}
uv = uv/5;
uv2 = uv2/3;
if(uv2 < 0x200)
{
MotorStop();
Extinguish();
BreakVariable = 1;
CandleFound = 1;
break;
}
if(uv > 0x200)
{
MotorStop();
break;
}
}
}
if(BreakVariable == 0)
{
for(int a=0; a <= c/2; a++)
{
Left_Mtr(-7);
Right_Mtr(7);
}
MotorStop();
MoveForwardReady = 1;
rotationDirection++;
}
}
else if(rotationDirection % 2 == 1)//Alternate the direction the robot spins to scan
{
  Left_Mtr(50);
  Right_Mtr(-50);
  Delay(10000);
  MotorStop();
```

```
if(BreakVariable == 0)
{
for(int a=0; a<1000000; a++)        //Rotate left until uv found
{
uv = 0;
uv2 = 0;
Left_Mtr(-7);
Right_Mtr(7);
for(int b=0; b<5; b++)      //Take 5 Samples of Sensitive UV Sensor
{
ADC1_CR2 |= (1<<30);
DataToCm();
uv = uv + ADC_Data[4];
if(b%2 == 0)
{
uv2 = uv2 + ADC_Data[5];
}
}
uv = uv/5;
if(ADC_DataCm[1] <= 0x7)          //UV Sensor can also be tripped by close wall this avoids that false trip
{
uv = 0xFFF;
}
uv2 = uv2/3;
if(uv2 < 0x200)   //If nonsensitive UV Sensor is tripped fire in range so extinguish
{
MotorStop();
Extinguish();
BreakVariable = 1;
CandleFound = 1;
break;
}
if(uv < 0x200)     //If aligned on flame stop on it and ready to inch closer
{
MotorStop();
break;
}
}
}
```

```
if(BreakVariable == 0)
{
for(int a=0; a<1000000; a++) //Rotate left until uv not found
{
uv = 0;
uv2 = 0;
Left_Mtr(-7);
Right_Mtr(7);
for(int b=0; b<5; b++)     //5 Samples taken
{
ADC1_CR2 |= (1<<30);
DataToCm();
uv = uv + ADC_Data[4];
if(b%2 == 0)
{
uv2 = uv2 + ADC_Data[5];
}
c++;
}
uv = uv/5;
uv2 = uv2/3;
if(uv2 < 0x200)   //If nonsensitive UV Sensor tripped extinguish
{
MotorStop();
Extinguish();
BreakVariable = 1;
CandleFound = 1;
break;
}
if(uv > 0x200)     //If Sensitive UV is no longer detecting break... Ready to find Middle point = FLAME
{
MotorStop();
break;
}
}
}
if(BreakVariable == 0)     //Spin the robot back to the mid point which should be the fire or very close
{
for(int a=0; a <= c/2; a++)
{
```

```
                Left_Mtr(7);
                Right_Mtr(-7);
                }
                MotorStop();
                MoveForwardReady = 1;
                rotationDirection++;
                }
                }
                }



    void candleInDoorwayCheck(void)
    {   //Function to determine if the candle is in the way of robot fully entering the room
    for(int a=0; a<30000; a++) //Rotate left until uv is found
    {
    uv = 0;
    Left_Mtr(-7);
    Right_Mtr(7);
    for(int b=0; b<5; b++)          {
       ADC1_CR2 |= (1<<30);
       uv = uv + ADC_Data[4];
       }
       uv = uv/5;
       if(uv < 0x200)        {
          MotorStop();
          flameAhead = 1;         //set variable
          for(int d=0; d<a; d++) //reverse back
          {
          Left_Mtr(7);
          Right_Mtr(-7);
          }
          MotorStop();
          break;
          }
          }
          if(flameAhead != 1)
          {
          for(int c=0; c<30000; c++)
          {
          Left_Mtr(7);
```

```
Right_Mtr(-7);
}
MotorStop();
}
for(int a=0; a<30000; a++) //Rotate left until uv is found
{
uv = 0;
Left_Mtr(7);
Right_Mtr(-7);
for(int b=0; b<5; b++)
{
ADC1_CR2 |= (1<<30);
uv = uv + ADC_Data[4];
}
uv = uv/5;
if(uv < 0x200)
{
MotorStop();
flameAhead = 1;//set variable
for(int d=0; d<a; d++) //reverse back
{Left_Mtr(-7);
Right_Mtr(7);}
MotorStop();
break;
}
}
if(flameAhead != 1)
{
for(int c=0; c<30000; c++)
{
Left_Mtr(-7);
Right_Mtr(7);
}
MotorStop();
}
}
void FindFlame(void)
{
ADC1_CR2 |= (1<<30);
DataToCm();
if(MoveForwardReady == 1 && ADC_DataCm[1] > 0x9)
{
Left_Mtr(50);
Right_Mtr(50);
MoveForwardReady = 0;
}
```

```
for(int z = 0; z < 7000; z++)        //This for loop is to drive towards the wall and breaks when within range
{
ADC1_CR2 |= (1<<30);
DataToCm();
if(ADC_DataCm[1] <= 0xA)        //While moving continue checking for obstacle
{
        for(int i = 0; i < 10; i++)     //10 readings to avoid false trips
        {
        ADC1_CR2 |= (1<<30);
        DataToCm();
        if(ADC_DataCm[1] <= 0xA)
        {
        ActualObstacle++;
        }
        else if(ADC_DataCm[1] > 0xA)
        {
        ActualObstacle = 0;
        break;
        }
        }
        if(ActualObstacle >= 10) //If there is an actual obstacle Candle is found
        {
        MotorStop();
        ADC1_CR2 |= (1<<30);
        if(ADC_Data[5] <= 0x200)//Check the Nonsensitive UV Sensor and extinguish if needed
        {
        CandleFound = 1;
        Extinguish();
        Delay(5000);
        }
        else if(ADC_Data[5] > 0x200)
        {
        BreakVariable = 0;
        CandleFound = 0;
        }
        ActualObstacle = 0;
        break;
        }
        else if(ActualObstacle < 10)        //If False Tripped reset variable
        {
        ActualObstacle = 0;
```

```
}
}
else if(ADC_Data[5] <= 0x200)      //If Nonsensitive UV Sensor is tripped
{
for(int m=0; m<3; m++)    //Triple check to make sure the Fire has been found
{
ADC1_CR2 |= (1<<30);
uv2 = uv2 + ADC_Data[5];
}
uv2 = uv2/3;
if(uv2 <= 0x200)
{
CandleFound = 1;
MotorStop();
Extinguish();
Delay(5000);
uv2 = 0;
break;
}
else if(uv2 > 0x200)
{
uv2 = 0;
}
}
else if(ADC_DataCm[1] > 0xA && ADC_Data[5] > 0x200) //Otherwise fire is not within range
{
if(ADC_DataCm[2] <= 0x8 && ADC_DataCm[3] <= 0x8)
{
InRange++;
if(InRange >= 5)
{
OffSet = 2;
AlignAfterTurn();
InRange = 0;
OffSet = 1;
}
}
Left_Mtr(50);
Right_Mtr(50);
ADC1_CR2 |= (1<<30);
DataToCm();
}
```

```
                    else if(z == 6999)
                    {
                    MotorStop();
                    break;
                    }
                    }
                    }



void Extinguish(void)
{
GPIOB_BSRR |= 1<<9; //propeller on
Delay(40000);
GPIOB_BSRR |= 1<<25;        //propeller off
Delay(5000);
}

 void LeaveRoom(void)
 {
 Left_Mtr(-50);
 Right_Mtr(-50);
 Delay(21000);          //Idea here is to back out of the room a bit
 MotorStop();
 Delay(5000);
 Left_Mtr(-50);          //Mtrs turned on in opp directions
 Right_Mtr(50);
 Delay(21500);          //Delay obtained from testing for 90 degree turn
 MotorStop();
                    Alignable = 0;
                    Delay(5000);
                    for(int i = 0; i < 200000; i++)
                    {
                    ADC1_CR2 |= (1<<30);
                    DataToCm();    //Progress forward until wall found to attach to
                    Left_Mtr(50);
                    Right_Mtr(50);
                    if(ADC_DataCm[1] <= 0xA && ADC_DataCm[0] <= 0xA)
                    {
                    ActualObstactle = 0;
                    for(x = 0; x < 15; x++)
                    {
                    ADC1_CR2 |= (1<<30);
```

```
DataToCm();
if(ADC_DataCm[1] <= 0xA && ADC_DataCm[0] <= 0xA)
{
ActualObstacle++;
}
else if(ADC_DataCm[1] > 0xA || ADC_DataCm[0] > 0xA)
{
Left_Mtr(30);
Right_Mtr(30);
ActualObstacle = 0;
break;
}
}
if(ActualObstacle >= 15)
{
ActualObstacle = 0;
CheckObstruction();
break;
}
else if(ActualObstacle < 15)
{
Left_Mtr(50);
Right_Mtr(50);
ActualObstacle = 0;
}
}
else if(ADC_DataCm[2] <= 0xD && ADC_DataCm[3] <= 0xD)
{
for(int x = 0; x < 15; x++)
{
ADC1_CR2 |= (1<<30);
DataToCm();
if(ADC_DataCm[2] <= 0xD && ADC_DataCm[3] <= 0xD)
{
Alignable++;
}
else if(ADC_DataCm[2] > 0xD || ADC_DataCm[3] > 0xD)
{
Alignable = 0;
//break;
}
}
if(Alignable >= 15)
```

```
{
Delay(4000);
MotorStop();
Delay(5000);
Alignable = 0;
AlignAfterTurn();
Delay(5000);
ADC1_CR2 |= (1<<30);
DataToCm();
break;
}
else if(Alignable < 15)
{
Left_Mtr(50);
Right_Mtr(50);
Alignable = 0;
}
}
else if(i == 35000)
{
AlignAfterTurn();
}
else if(i == 54000)    //If the front right sensor is within range of a wall begin using it again to explore
{
MotorStop();
AlignAfterTurn();
if(ADC_DataCm[2] <= 0x14) //If Front Right is within Range of a Wall
{
ExploreMazeFrontRightSensor();//Follow wall with front right Sensor
}
else if(ADC_DataCm[2] > 0x14) //If Front Right is not within Range of wall
{
ExploreMazeBackRightSensor();//Follow wall with back right Sensor
}
break;
}
}
}

void MicSetup(void)
{
//Clock bits
RCC_APB1ENR |= (1 << 29); //Bit 29 is DAC clock enable bit
RCC_APB2ENR |= (1<<9);    //Bit 8 is ADC 2 clock enable bit
```

```
RCC_APB1ENR |= (1 << 4);  //Enable peripheral timer for timer 6
//I/O bits
GPIOA_MODER |= 0x4000; //Bits 15-14 = 01 for digital output on PA7
//OTYPER register resets to 0 so it is push/pull by default
GPIOA_OSPEEDER |= 0xC000; //Bits 15-14 = 11 for high speed on PA7
//PUPDR defaults to no pull up no pull down
GPIOA_MODER |= 0xC00;  //PA5 is MIC analog
GPIOB_MODER |= 1<<(2*15); //PB15 is output LED for mic

//DAC bits
DAC_CR |= 0x3E;      //Bits 3, 4, 5 = 111 for software trigger ch1
//Bit 2 = 1 for Ch 1 trigger enabled
//Bit 1 = 1 for Ch 1 output buffer enabled
DAC_CR |= 1;         //Bit 0 = 1 for Ch 1 enabled
//ADC bits
ADC2_CR2 |= 1;       //Bit 0 turn ADC on
ADC2_CR2 |= 0x400;//Bit 10 allows EOC to be set after conversion
ADC2_SQR3 |= 0x5;  //Bits 4:0 are channel number for first conversion
// Channel is set to 5 which corresponds to PA5
//Timer 6 bits
TIM6_CR1 |= (1 << 7);          //Auto reload is buffered
      TIM6_CR1 |= (1 << 3);  //One pulse mode is on.
      TIM6_PSC = 0;  //Don't use prescaling
TIM6_ARR = 1000;   //Math explanation below
//^^Did not reset sysclk so clk is 16Mhz(HSI) Math Changed to:
// [(16Mhz)/TIM6_ARR] = 16,000   The  16,000 is from fs set in Octave
program
//Gives Tim6_ARR = 1000;
TIM6_CR1 |= 1;       //Enable Timer 6
}

void Check_Mic_Input(void)
{
if(count < cMax)
{
```

```
if(yInt > 2300)     //This number is adjustable and had to set higher so the lower freq. Button
//would stop tripping the mic...
{
maxY++;
count++;
}
}
else
{
count = 0;
if(maxY >= 16)    //This number is set to try and cut out short bursts played at correct frequency
{
GPIOB_ODR |= (1<<15); //Toggle LED for now but start signal later
StartSound = 1;
Delay(5000);
maxY = 0;
}
else if(maxY < 16)           //Can be adjusted to cut out short bursts of 3.8kHz tones
{
StartSound = 0;
maxY = 0;
}
}
}

void Start_Sound(void)
{
//First Section Contants: Row 1 Above
const float b10 = .011522;
const float b11 = .012323;
const float b12 = .011522;
const float a11 = .284777;
const float a12 = .979202;
//Second Section Constants: Row 2 Above
const float b20 = 1.0000;
const float b21 = .457263;
const float b22 = 1.0000;
const float a21 = .230837;
```

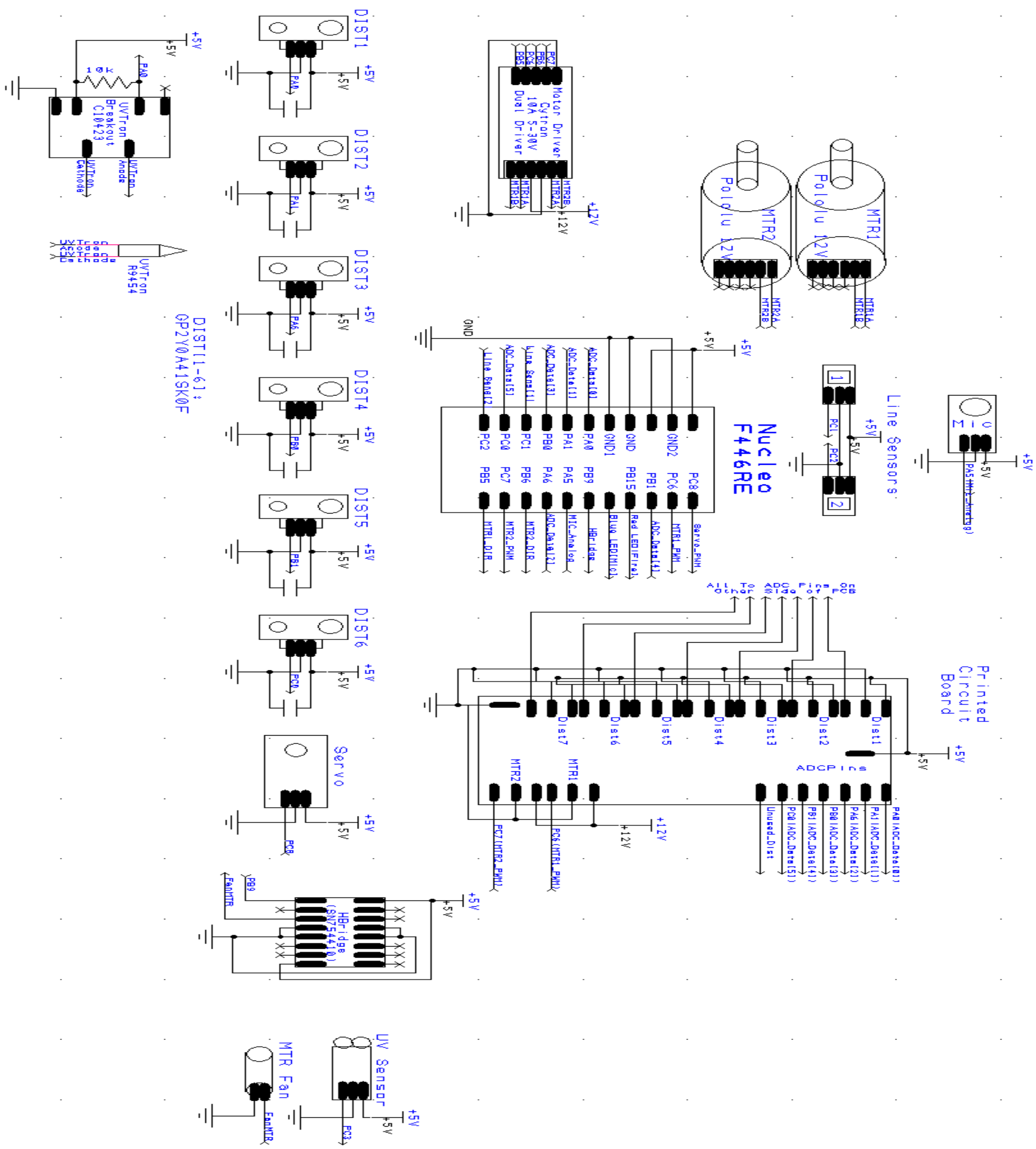
```
    const float a22 = .913693;
    //Third Section Constants: Row 3 Above
    const float b30 = 1.0000;
    const float b31 = .358901;
    const float b32 = 1.0000;
    const float a31 = .056960;
    const float a32 = .809374;
    //Fourth Section Constants: Row 4 Above
    const float b40 = 1.0000;
    const float b41 = -.548337;
    const float b42 = 1.0000;
    const float a41 = -.236649;
    const float a42 = .810769;
    //Fifth Section Constants: Row 5 Above
    const float b50 = 1.0000;
    const float b51 = -.641882;
    const float b52 = 1.0000;
    const float a51 = -.416775;
    const float a52 = .915080;
    //Sixth Section Constants: Row 6 Above
    const float b60 = 1.0000;
    const float b61 = -1.204521;
    const float b62 = 1.0000;
    const float a61 = -.475228;
    const float a62 = .979594;

     unsigned int xInt;
     float x, y10, y20, y30, y40, y50, y60;
     float w10, w11, w12;
     float w20, w21, w22;
     float w30, w31, w32;
     float w40, w41, w42;
     float w50, w51, w52;
     float w60, w61, w62;

        while(StartSound == 0)
        {
        ADC2_CR2 |= 0x40000000;     //Bit 30 does software start of A/D conversion
        while((ADC2_SR & 0x2) == 0); //Bit 1 is End of Conversion
        xInt = ADC2_DR;
        x = ((float)(xInt & 0xFFF))/(float)4095.0;
```

```
//first section
w10 = x-a11*w11-a12*w12;
y10 = b10*w10+b11*w11+b12*w12;
//second section with 'y10' as the input, and 'y20' as the output
w20 = y10-a21*w21-a22*w22;
y20 = b20*w20+b21*w21+b22*w22;
//third section with 'y20' as the input, and 'y30' as the output
w30 = y20-a31*w31-a32*w32;
y30 = b30*w30 + b31*w31 + b32*w32;
//fourth section with 'y30' as the input, and 'y40' as the output
w40 = y30-a41*w41-a42*w42;
y40 = b40*w40+b41*w41+b42*w42;
//fourth section with 'y40' as the input, and 'y50' as the output
w50 = y40-a51*w51-a52*w52;
y50 = b50*w50+b51*w51+b52*w52;
//sixth section with 'y50' as the input, and 'y60' as the output
w60 = y50-a61*w61-a62*w62;
y60 = b60*w60+b61*w61+b62*w62;

    yInt = (int)(1500*(y60+1));           //Data to D/A

DAC_DHR12R1 = yInt & 0xFFF; //Converted number to D/A
DAC_SWTRIGR |= 0x1;    //Start the D/A conversion
w12 = w11;
w11 = w10;
w22 = w21;
w21 = w20;
w32 = w31;
w31 = w30;
w42 = w41;
w41 = w40;
w52 = w51;
w51 = w50;
w62 = w61;
w61 = w60;

Check_Mic_Input();        //Call function to determine if Start_Frequency Detected
while((TIM6_CR1 & 1) != 0); //Wait here until timer runs out
TIM6_CR1 |= 1; //Restart timer
}
}
```

```
void Delay(unsigned int z)//Function for variable delay based on the unsigned int that is sent
{
unsigned int x;
int y;//Declares variables to be used in loops
for(x=0;x<z;x++)
{for(y = 0;y < 256; y++);}}
```

Appendix D

Appendix C



DIST1
DIST2
DIST3
DIST4
DIST5
DIST6
DIST[1-6]: GP2Y0A41SK0F
Servo
UVTron Breakout C10423
UVTron R9454
Motor Driver Cytron 10A 5-30V Dual Driver
MTR1 Pololu 12V
MTR2 Pololu 12V
Nucleo F446RE
Line Sensors
Printed Circuit Board
ADCPins
HBridge (SN754410)
UV Sensor
MTR Fan

References

1. Trinity College. (2018, September 18). [Online].
   Available: http://www.trinityrobotcontest.org/rules.html

2. "Sharp GP2Y0A41SK0F IR Range Sensor – 4 to 30cm"
   https://www.robotshop.com/en/sharp-gp2y0a41sk0f-ir-range-sensor.html

3. "SparkFun Electret Microphone Breakout." *BOB-12758 - SparkFun Electronics*, www.sparkfun.com/products/12758.

4. *RedBot Line Sensor*, www.robotshop.com/en/redbot-line-sensor.html.

5. "STM32F446RE"

   https://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32-high-performance-mcus/stm32f4-series/stm32f446/stm32f446re.html

6. "Pololu 12V, 100:1 Gear Motor w/ 64 CPR Encoder." *Www.robotshop.com*, www.robotshop.com/en/pololu-12v-1001-gear-motor-64-cpr-encoder.html?gclid=CjwKCAjw0oveBRAmEiwAzf6_rMMimJqBrek9OUvlEtH5p4FHLSQvPCSDs-aYXzHrbn9va2kll57gtxoCjvsQAvD_BwE.

7. "Cytron 10A 5-30V Dual Channel DC Motor Driver." *Www.robotshop.com*, www.robotshop.com/en/cytron-10a-5-30v-dual-channel-dc-motor-driver.html.

8. "FLAME SENSOR UVTRON"
   https://www.hamamatsu.com/resources/pdf/etd/R9454_R9533_TPT1019E.pdf

9. "IEEE Standards Association"
   https://standards.ieee.org/standard/C37_14-2002.html