Trinity Fire Fighting Robot Competition

Jacob Stratman, Electrical Engineering
Isaiah Frey, Electrical Engineering

Project Advisor: Mark Randall

Trinity College International Robot Contest
Hartford, CT
April 7-8, 2018

Acknowledgements

We would like to thank the University of Evansville and the Academic Fund Board for funding this project and travel. We would also like to thank Mark Randall for advising us. Special thanks also to Jeff Cron for traveling with us to the contest, Vicky Hasenour for assistance in planning, Dr. Lotfalian for his advice, and Dr. Howe for her feedback.

Table of Contents

List of Figures

List of Tables

I. Introduction

The objective of this project is to create an autonomous robot capable of starting at a signal, navigating a simple floor plan with four rooms, extinguishing a candle flame, and returning to the starting room. The robot was required to start in an arbitrary room, navigate hallways (with rugs and an obstacle in level two), and extinguish the flame using CO_2. LEDs had to be lit when the signal was detected and the fire extinguished. The total dimensions of the robot were not to exceed one cubic foot.

To complete this objective both hardware and software solutions were used. A digital band-pass filter detected the starting signal. Positional memory and right wall following code were used in conjunction with IR SHARP Distance sensors to navigate the hallways. Line sensors detected doorways marked by solid white lines and stored the room being searched in memory. A Hamamatsu UV Tron identified if a flame was present once in a room and a phototransistor mounted on a servo motor located the precise position of the flame. A compressed CO_2 canister regulated by a Versa Valve® dispensed CO_2 to extinguish the flame. The robot then recalls the room it began in and selects a return path to that room, completing its objective. In level two front distance sensors, additional positional memory, and detour paths allow the robot to navigate around the dog obstacle.

II. Background

A. Problem Definition

The competition consisted of three levels. In order to move on to the next level, the previous level had to be completed. Each team was allowed a total of 5 trials for the entire competition. Before each trial, the robot had to pass a judge's inspection to verify that the robot was within the regulations of the competition. Robots could not be larger than 31 cm long by 31 cm wide by 27 cm tall. Robots had to employ water, air, CO_2 canisters, or mechanical means, to extinguish the flame. If competing in the Versa Valve® Challenge, teams had to prove the Versa Valve® is in use on the robot. Robots had to run off a DC power source. Robots had to have a power switch which connects and disconnects the battery power to the robot [1].

Robots had to have a control panel which included an LED indicating when the flame had been located. The control panel had to have the microphone labeled "MIC" on the control panel for easy identification. The control panel had to include a labeled kill switch that would shut off power to all systems. The control panel had to have a labeled arrow indicating the front of the robot.

Robots had to have a carrying handle located above the robot enabling contest employees to safely move each robot without touching any components. Upon having the power switch turned on, robots could not move until sensing a 3.8 kHz signal which indicated a round had begun. Robots could not false start by responding to a dummy 2 kHz signal [1].

Once the robot successfully passed the judge's exam, the robot could make an attempt at the current level round. The Level 1 maze layout, as outlined in the Firefighting Robot Competition Rulebook, is shown in Figure 1 [1].

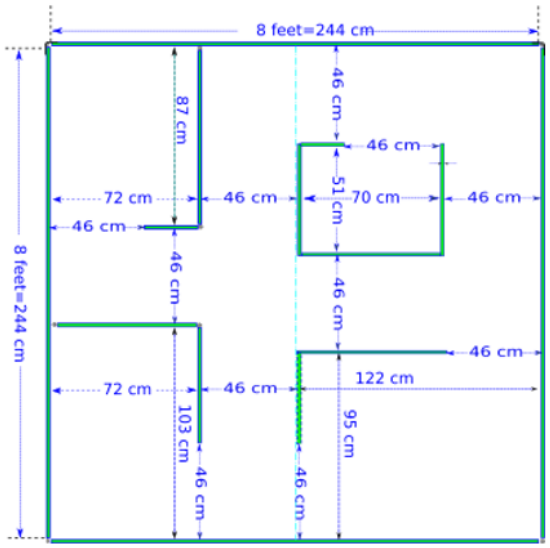


Figure 1: Level 1 maze including dimensions [1]

The dimensions shown for Level 1's maze have a tolerance of 1 cm, so robots were expected to account for slight error. The entire maze was 244 cm by 244 cm (8 ft. by 8 ft.). All hallways were approximately 46 cm wide. Each level had 4 possible maze layouts, but each configuration follows the same general layout with four rooms and hallways connecting them. Robots would start in an arbitrary room. Each maze configuration had white walls with black wooden flooring. There were solid white lines in each room doorway. For Level 1, robots only needed to navigate the maze, extinguish a candle located in one of the four rooms, and return to the starting circle. There were no obstacles in the Level 1 maze, and there was a time limit of three minutes for Level 1 [1].

The Level 2 maze had the same layout as the Level 1 maze, but there were obstacles for the robot to avoid [1]. There was areas of the maze that were carpeted. The shaded grey areas shown in Figure 2 were potential carpeted areas.

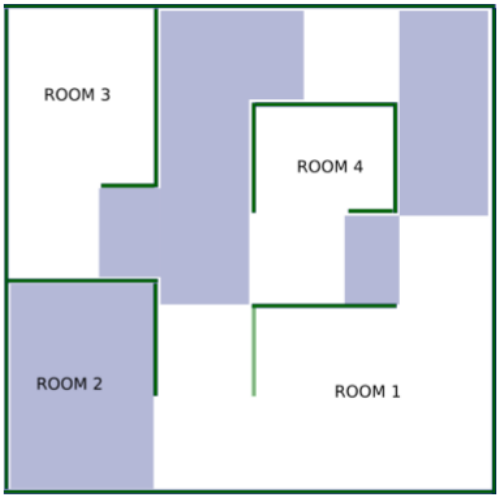


Figure 2: Level 2 maze displaying areas of possible carpeting (shaded grey) [1]

Pictures, mirrors, and tapestries could be hung on the walls in Level 2. A plush puppy was sitting in one hallway and had to be avoided by the robot. Furniture was placed in rooms and corners of the hallways and had to be avoided. Just like in Level 1, the robot had to begin in an arbitrary room, locate and extinguish the candle, and return to the starting circle to complete Level 2. Level 2 had a four minute time limit [1].

B. Summary of Requirements

- Robot could not exceed 31cm long by 31cm wide by 27cm tall
- Robot could use CO2, air, water, or mechanical means to extinguish candles
- Robot could use a Versa Valve® to compete in Versa Valve Challenge
- Robot had to have a power kill switch connected to DC power source
- Robot had to have a control panel with labeled microphone and LEDs identifying when specific functions such as fire detection and vision tracking occur

- Robot had to have a carrying handle to allow contest workers to move the robot safely and securely
- Robot had to activate navigation upon sensing a 3.8 kHz starting signal and ignore a 2 kHz false starting signal
- Robot had to navigate an 8 ft. by 8 ft. maze, locate and extinguish a candle flame, and return to starting position
- Robot had to avoid obstacles in the maze
- Robot had to complete each level within the allotted time limit

C. Summary of Scoring

1. Actual Time

Each team had five chances, or trials, to complete the required tasks. Each trial was assigned a score based on the robot's performance, with teams competing for the *lowest* score. To complete level 1 and level 2, some core requirements had to be met:

1. The robot had to start at a 3.8 kHz ± 10% signal
2. The robot had to navigate the maze
3. The robot had to extinguish the fire.

The robot was required to find and extinguish the candle in three minutes for level 1 and four minutes for level 2. This time limit included the amount of time taken from the starting signal to the candle fire being extinguished. This time in seconds is called the Actual Time (AT).

2. Operating Modes

If a team wishes to improve their score by taking on other challenges, they can attempt different operating modes. The team must decide before the trial which operating modes they

will attempt. If they are successful, a Mode Factor (MF) will be calculated by multiplying the operating mode's multipliers together. Operating modes are listed in Table 1.

Table 1: Operating Modes and descriptions

Operating Mode	Multiplier	Description
Standard	1.0	Start at the starting signal and extinguish the fire
Arbitrary Start	0.8	The robot will begin in a random room at an arbitrary angle instead of the start circle
Return Trip	0.8	The robot must return and stop fully at its starting location or starting room
Non-air Extinguisher	0.75	The robot must use inert gas, water, or mechanical means to extinguish the fire
Furniture	0.75	The robot must avoid large cylindrical pieces of furniture in the rooms
Candle Location (Level 1 only)	0.75	The robot must find and extinguish the candle without the assistance of a candle circle

Our robot attempted and completed the Arbitrary Start, Return Trip, Non-air Extinguisher, and Candle Location Operating Modes for a Mode Factor of 0.36 for level 1 and 0.48 for level 2.

3. Room Factor

The Room Factor (RF) is determined by how many rooms must be searched before the candle is found (excluding the starting room in Arbitrary Start Operating Mode). The Room Factor is a fractional multiplier which reduces the final score. Room Factors are shown in Table 2.

Table 2: Room Factors based on when the candle is located

Room Containing Candle	Room Factor
First Room	1.0
Second Room	0.85
Third Room	0.5
Fourth Room	0.35

4. Penalty Points

Some actions will incur Penalty Points (PP) which add to the Actual Time for a worse score. The penalties are shown in Table 3.

Table 3: Penalties

Action	Penalty
Touching the candle	50
Continuous Wall Contact	(Contact cm)/2
Touching the Dog Obstacle	50

5. Final Score

Again, teams are seeking the lowest score. Teams may use multiple trials on a single level in an attempt to improve their score. The lowest score of any of the trials may be used, but only five trials are allowed in total for all levels. The final score is calculated using the following equation:

$$Score = (Actual\ Time[seconds] + Penalty\ Points) * Mode\ Factor * Room\ Factor$$

If a robot does not complete a level, the team will be given a default score of 600.

III. Design Solution

A. Hardware Design

The first step of the hardware design was choosing the type of components suitable for the chassis. For the main body plates, 0.25" marine starboard was used. Starboard was chosen for its low weight, durability, and ease of shaping. The body of the robot was composed of rectangular plates of starboard and held in place using galvanized metal plumber's tape and size 8 machine screws. Plumber's tape's ability to be bent into shape but still hold that shape while load bearing made it a strong choice for the skeleton of the robot.

For traversing the maze, two Pololu 12 V DC motors were used, as shown in Figure 3. These motors generate the torque necessary to move the weight of the robot through the maze with no issues. The motors were mounted using machined aluminum motor mounts to ensure they remained in the correct position at all times.

Figure 3: 12 V DC motors from Pololu [2]

The DC motors were driven using a single 10A dual channel DC motor driver shown in Figure 4. This driver provided the power required to operate the motors as well as an effective way to control them with the Raspberry Pi. Four GPIO pins were connected to the driver to control the motor system. One pin each was used to control the direction of each motor rotation and another pin to control the motor speed via PWM.

Figure 4: 10 A dual channel DC motor driver [3]

IR SHARP sensors were used to determine where the robot was in the maze. For close range navigation, 4-30 cm range sensors were used shown in Figure 5a. For longer distance, the 20-150 cm range sensors were used shown in Figure 5b. The long-range sensors enabled the robot to measure rooms it was in, allowing it to determine its starting location. The short-range

sensors allowed for PID-controlled wall following and seeing obstacles such as the dog and oncoming walls. All SHARP sensor outputs were read using an external ADC chip for the Raspberry Pi. Capacitors were connected in parallel with the sensor voltage inputs to minimize chance of power spikes which may cause inaccuracies in input data.

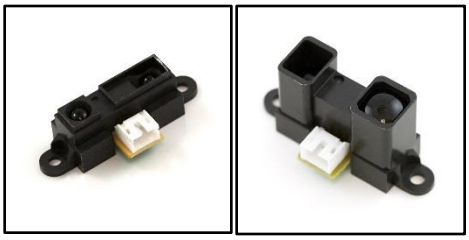


Fig. 5a **Fig. 5b**

Figure 5: Short (a) and long (b) range SHARP distance sensors [4]

Three TCRT5000 line sensors (Figure 6) were used to recognize the white lines located in the entrances of each room. These sensors normally output an analog signal based on the surface it is pointed at, but a higher resistance value was used with their receivers to make the output binary instead. This allowed a single, 8-channel ADC chip to be used since standard GPIO ports could be used to read the binary digital output of the three TCRT5000s.

Figure 6: TCTR5000 line sensor [5]

The flame detection strategy was two-fold. Initial detection of the candle in a room was covered using the Hamamatsu UV TRON along with the Hamamatsu C10807 driving circuit board both shown in Figure 7. When combined, these components make a very sensitive UV

flame detector. The configuration generated a pulse train at the output upon detecting fire. The Raspberry Pi waited for this pulse train to know if the fire is in a given room.

Figure 7: The Hamamatsu UV TRON (right) and its driving circuit (left) [6]

Once the fire was detected, the second fire-detection phase begins. An IR phototransistor was mounted to a Futaba DC servo (Figure 8). This servo used PWM to rotate the phototransistor in a 180 degree semicircle. The phototransistor was tuned using a resistor to detect UV light coming off the candle flame. Upon recognizing where the fire was in the room, the robot used the CO_2 to extinguish it.

Figure 8: Futaba DC servo motor [7]

In order to compete in the Versa Valve challenge, an EZ-1 series Versa Valve®, shown in Figure 9, had to be incorporated into the project. This model of valve is normally closed until a voltage of 9-12V is applied to it. To control the valve with the Raspberry Pi, a MOSFET switch

circuit was used. A GPIO pin was connected at the gate, ground at the source, and the valve and 12V at the drain. Setting the GPIO pin high closes the drain-to-source and opens the valve.

An air regulator and CO_2 canister were connected to the valve's input, and the output was connected to a hose running adjacent to the phototransistor on the servo. When the servo is aimed at the flame, the GPIO pin at the gate turns on, allowing the air regulator and valve to release a stream of compressed CO_2 to extinguish the flame.

Figure 9: EZ-1 Versa Valve® [8]

The robot's circuitry was contained on a PCB. The PCB minimized the amount of space wasted internally and the possibility of unexpected shorts or disconnections. Figure 10 shows the schematic of the PCB on *PCB Artist*. A full schematic of the robot circuitry can be found in Appendix A.

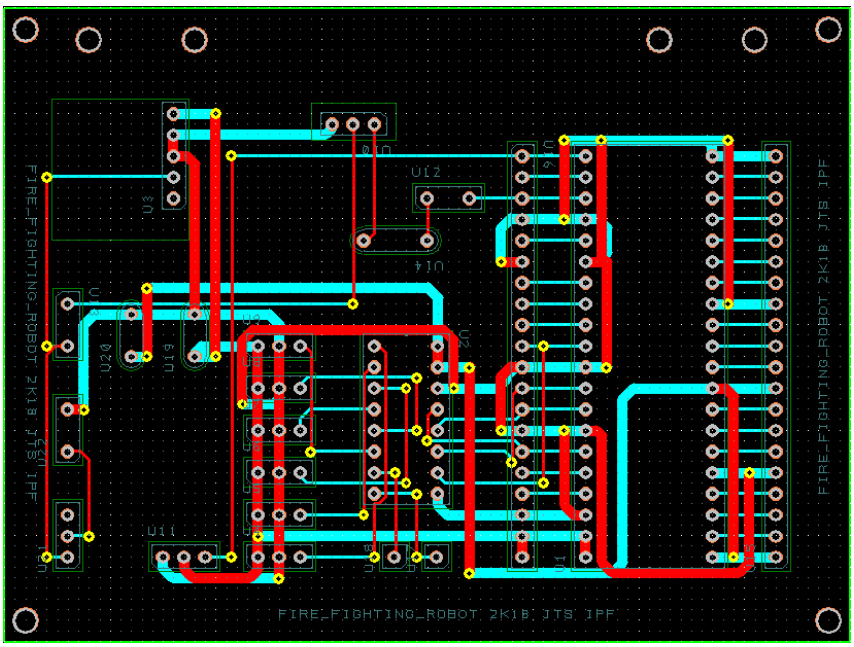


Figure 10: PCB Layout

Figure 11 shows a complete block diagram for the robot's hardware design and connections.

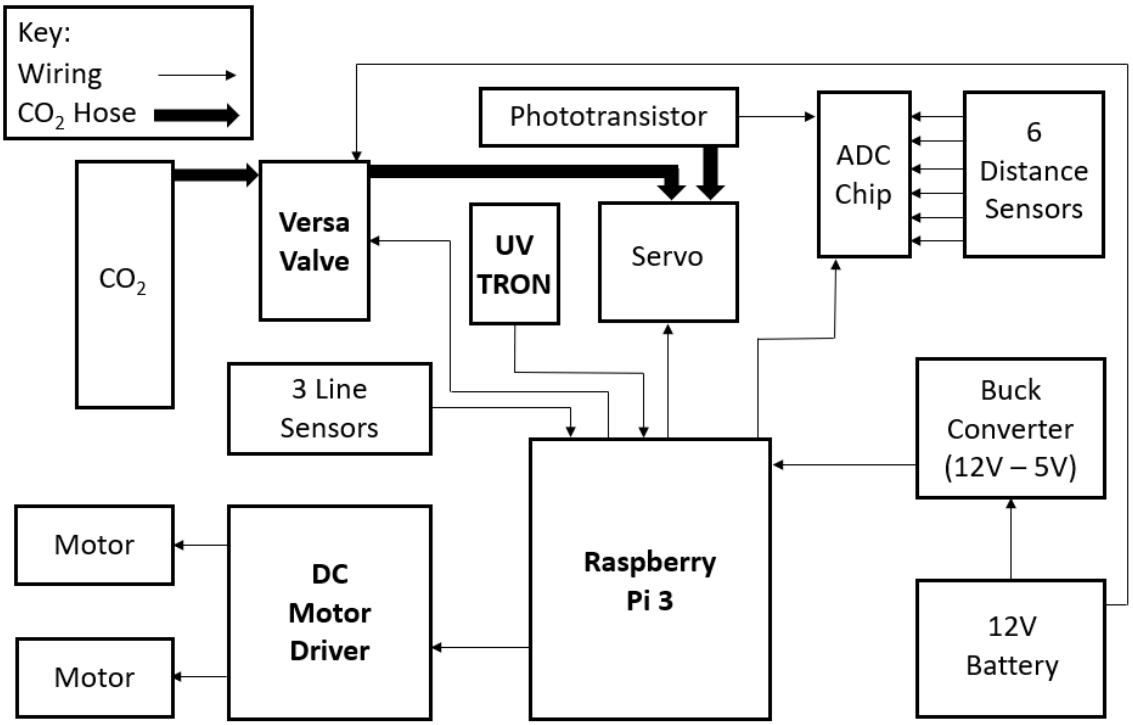


Figure 11: Hardware Block Diagram

B. Software Design

The robot functioned entirely from a Python script loaded onto a Raspberry Pi 3B, which served as the robot's computer. To complete the tasks outlined in the competition rules, the problems were broken down into five sections. One: detect the starting 3.8 kHz signal while ignoring all other signals outside a 10% margin of error. Two: navigate out of the starting room from and direction and angle offset, then determine which room it began in. Three: navigate the floor plan one room at a time while avoiding obstacles. Four: Search each room as it is entered and extinguish the fire once it is found. Five: Return to the starting room.

To successfully detect a 3.8 kHz signal, several libraries were used to reduce the number of functions that would have to be programmed. The alarm detection was modeled after an alarm detector that returned the frequency for a chunk of input sound using the Fast Fourier Transform, by Benjamin Chodroff [9]. This code samples an input sound of 0.1 seconds in duration at a rate of 44100 Hz. The intensity of the signal is calculated using the Fast Fourier Transform and quadratic interpolation around the maximum intensity. Once the frequency is determined, it is compared to the acceptable range of frequencies (3.8 kHz ± 10%). To prevent accidental triggering, the signal must persist for at least 0.2 seconds, that is, two consecutive chunks must be within the acceptable range.

```
Def Sound_start():
    While (An acceptable signal hasn't been detected twice):
        While (Frequency < 3420 Hz or Frequency > 4180 Hz):
            Obtain a 0.1 second chunk of sound input
            Perform the Fast Fourier Transform on the input
            Frequency = quadratic interpolation of the FFT
    Return Start
```

Once the robot recognized the starting tone, it then began a loop to navigate out of the arbitrary starting room. This loop searched the room for a wall and then followed that wall to the door.

```
While (Inside the starting room):
    If (A wall is visible ahead):
        Drive to the wall
        Align to the wall
        Make a 90 degree turn left
        Follow the right wall to the door
    Else:
        Turn 45 degrees until a wall is visible
    If (Robot has completed a full rotation and not seen a wall)
        Drive forward some and repeat the process
```

Once this loop was completed and the robot stopped at the white line marking the doorway, the robot slowly reversed while recording values of two long range distance sensors: one on the back of the robot and one on the left side of the robot. The average of the values received by the back sensor was the length of the room, and the average of the values on the left sensor was the width of the room. An area was calculated with the length and the width given by the sensors plus the length and width of the robot. Since all four rooms in the floor layout have unique areas, this discovered area could be used to determine which room the robot began in. Furthermore, the left, right, and front distance sensors could also be checked once the robot was realigned with the door to determine which entrance of the room the robot was at if there were multiple entrances (such as in room 3 and 4).

At this point, the robot was positioned at the exit of the starting room and had recorded which room it started in. Next the robot had to carefully navigate the floor plan, proceeding from one room to the next, avoiding the dog obstacle if it was in level two. The general plan for

navigation was a checkpoint to checkpoint, six position system. In other words, the robot would mark its position, then have a plan to navigate to the next position. The robot used the following general plan to navigate to the next position for each of the six positions.

```
While (The fire is not extinguished AND the robot is not in its starting room):
    If (Exiting room 1):
        Right wall follow to room 2
        Search room 2 for a fire
        Navigate to room 2's exit
    If (Exiting room 2):
        Right wall follow to room 3
        Search room 3 for a fire
        Navigate to room 3's exit
    If (Exiting the left exits of room 3):
        Right wall follow to the top exit of room 3
        If (A dog is in the hallway):
            Take a detour route to room 4
        Else:
            Align self to top exit of room 3
    If (Exiting the top exit of room 3):
        Right wall follow to the top of room 4
        If (A dog is in the hallway):
            Take a detour route to room 4
        Else:
            Check if the entrance to room 4 is at the top
            If (Entrance is at the top):
                Search room 4 for a fire and navigate to its exit
            Else:
                Right wall follow to the bottom entrance of room 4
                Search room 4 for a fire
                Navigate to its exit
    If (Exiting room 4 at the top)
        Drive out to the top wall and make a left turn
        Right wall follow to room 1
        If (A dog is in the hallway):
            Take a detour around room 4 to room 1
    If (Exiting room 4 at the bottom):
```

Drive out to top wall of room 3
Turn right and drive diagonally up to the right wall of room 1
Wall follow to room 1
Search room 1 for a fire
Navigate to room 1's exit

This checkpoint navigation system with the three detours for the dog obstacle is also illustrated in Figure 12.

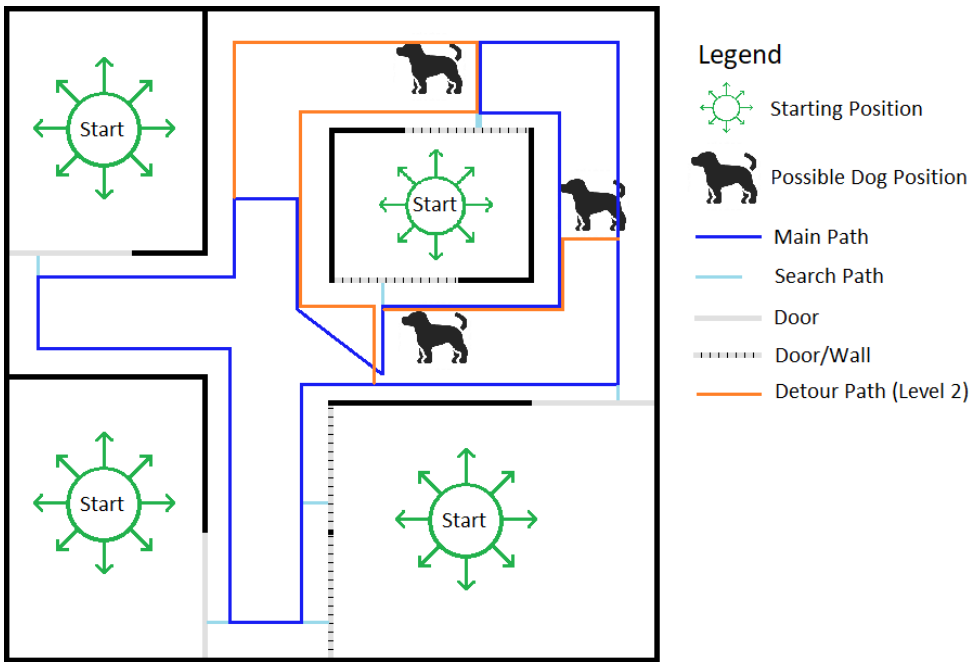


Figure 12: General Navigation Plan

In most of the position's navigation plans, a line mentions searching the room the robot has navigated to for a fire. Searching the room consisted of two general parts: first, using the Hamamatsu UV Tron to determine if there was a fire anywhere in the room, and second, using the phototransistor to determine the exact location of the fire. A search room function was programmed to find the exact location of the robot and it was passed which room it was searching.

```
Def Search_room(room):
    Use UV Tron to determine if a fire is in the room
    Drive forward until the robot is entirely in the room
    If (The UV Tron detected a fire):
        Execute a function to extinguish it
        Record that the fire has been extinguished
    Locate a wall
    Drive to the wall and follow it out of the room
```

The extinguishing function used a phototransistor to slowly approach the flame before extinguishing it once it was close enough to do so with confidence.

```
Def Extinguish_flame():
    Stop the robot
    While (A fire is detected):
        Sweep the servo motor 180°, record the phototransistor's output every 9°
        Record the position of the servo at the min. output (when fire is brightest)
        Turn the robot to face that position and turn the servo back to that position
        If (The fire is within a few inches of the robot):
            Open the CO2 canister and extinguish the flame
            Check the UV Tron to make sure the fire is out
            If (The fire is out):
                Break from the while loop
        Else:
            Approach the flame proportional to how far away the robot is
```

Now the fire has been located and extinguished, and the only remaining objective is to return to the starting room. This problem is very easily solved with the positional navigation code. Once the fire is extinguished, the robot will continue to navigate the maze from position to position until it reaches the starting room. This approach is acceptable since the return time is not counted against the total run time for that trial. The only requirement is that the robot return to the starting room within two minutes, an easily achievable time since the robot is not fighting

fires in the rooms along its return. Once it reaches the starting room, it will reverse fully into the room and stop, concluding the trial.

C. Standards

1. Economic

An economic standard for the robot is a relatively low cost to manufacture it. Certain parts used for the robot such as the motors, drivers, and regulator, are expensive so the costs of the rest of the robot's parts must be minimized. Starboard and relatively few sensors help minimize the cost of the robot. Furthermore, since the code can be duplicated, the time cost for manufactured robots would be almost zero.

2. Environmental

The environmental standards for the robot is to have a positive impact on the environment. The robot uses CO_2 as the means of extinguishing candles, which has a negative effect on the environment. This will be handled by mitigating how much CO_2 is used during the extinguishing process. Furthermore, the use of batteries in the robot which are non-recyclable, impact the environment. However, these will be offset by the reduction of smoke, and pollution that the robot will prevent by extinguishing household fires before they burn down the house.

3. Ethical

Ethical standards for the robot include prioritizing human life. The robot places the value of human life over its own safety by rushing immediately to the fire at the sound of an alarm and

extinguishing it. There is a chance the robot may be damaged by the fire, but this is unimportant since the robot is replaceable while humans are not.

4. Safety

One of the safety standards for the robot include avoiding the possibility of the robot knocking the candle over and spreading the fire even more. The chance of this happening will be mitigated by having a two-fold searching system for the fire. Upon entering a room, the robot will scan the room using the UV Tron. If the fire has been detected, the robot will then use phototransistors attached to the extinguisher barrel to hone in on the fire's location. The robot will then slowly approach the flame until it gets in range before extinguishing fire, minimizing the chances of the robot running into the candle while it is still lit.

Another safety standard is avoiding harming people or pets while moving through hallways searching for the fire. The robot will have distance sensors on the front and sides of the frame in order to track proximity to objects. If the robot is too close to an object, such as a pet or person, it will back up and find a new path. Another safety constraint is that the robot cannot go up and down stairs, only up and down a ramp. If the robot is halted by a stairway, the flame would not be extinguished.

5. IEEE Standards

The robot will also adhere to the IEEE standards in its construction and operation. Most importantly this project will "Hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment" [5]. The robot will function properly and consistently and will prioritize human life (saving the baby) above the

extinguishing of the fire or saving the house from harm. Should any malfunction of the robot be discovered, it will be immediately reported, publicized, and corrected so as to avoid harm to anyone. The robot also adheres to the IEEE "Standard for Autonomous Robotics (AuR) Ontology" by unambiguously identifying the basic hardware and software components which allow the robot to operate autonomously.

IV. Results

Our Fire Fighting Robot team won the Versa Valve® competition and placed third in the senior unique division. Our team applied the Arbitrary Start, Return to Start, and Non-air Extinguisher modifier for level 1 and level 2. The Candle Location modifier was also applied for level 1 giving an operating mode modifier of 0.36 for level 1 and 0.48 for level 2. The robot failed the first two trials on level 1 before completing the third trial with a time of 88 seconds giving a score of 88*0.36 = 32 for round one. The robot proceeded to level 2 and failed the first trial before succeeding on the final trial with a time of 197 seconds for a score of 95. Our team also presented a poster at the contest which was chosen as a top 10 poster.

1. Cost

The cost for this project was justified because the robot was used to successfully compete in an international competition. The cost to build the robot itself was well within the allotted budget. Table 4 shows the cost of the parts for the robot as well as the travel costs to get to the competition.

Table 4 - Cost

Section	Part	Total ($)
Motors and Wheels	**Motors**	**81.30**
	Main Wheels	44.30
	Motor Drivers	49.99
Main Body	Chassis	28.00
Battery	LiPo 12V Battery	69.54
Sensors	Distance sensor	69.20
PCB	**PC Board**	**104.00**
Registration	Registration	85.00
	Poster Registration	30.00
Extinguisher	**UV TRON**	**129.18**
	Regulator	61.49
Software	Raspberry pi	70.00
Travel	Flights and hotels	1939.61
Total		**Robot: 822.00** **Final: 2761.61**

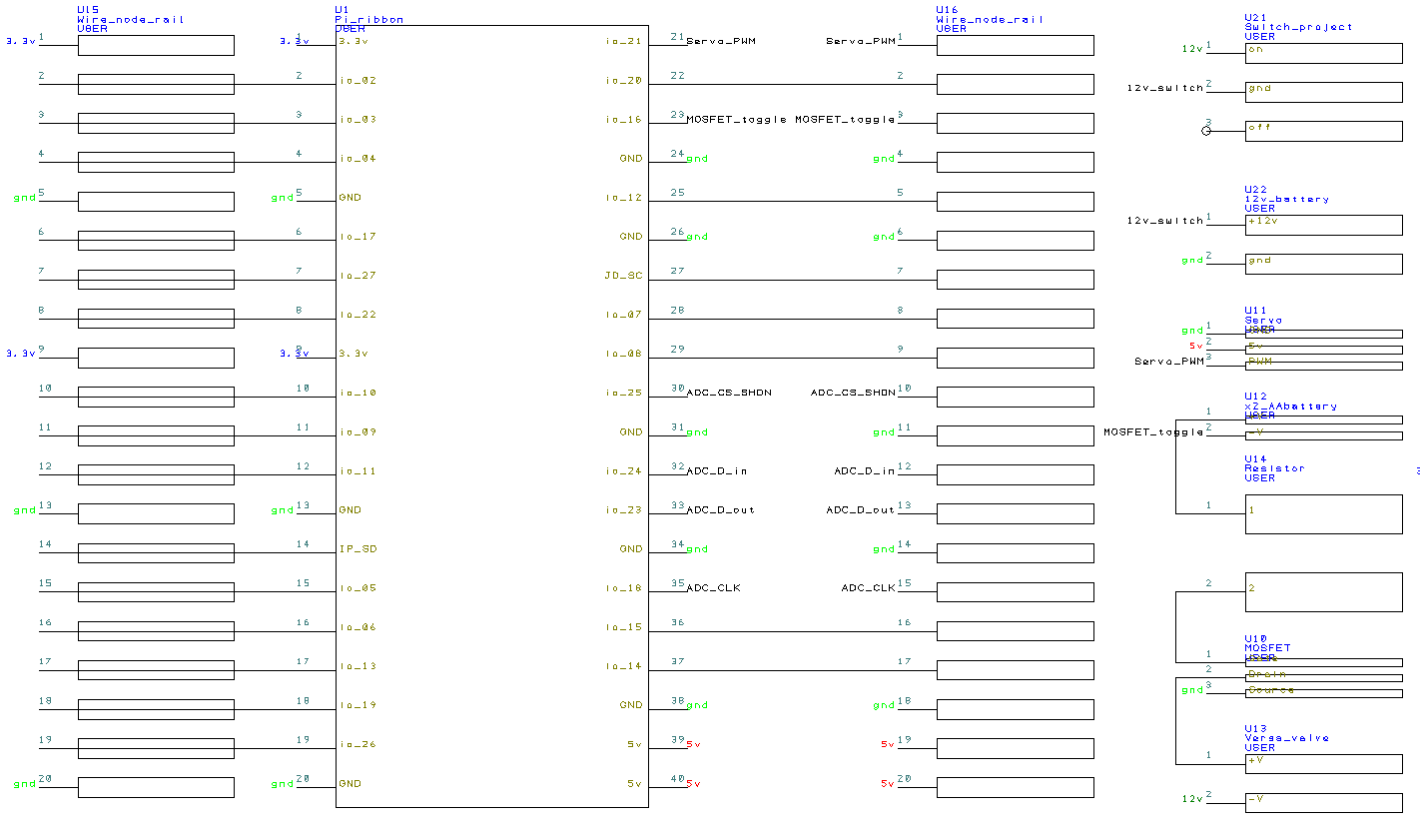
V. Conclusion

The Fire Fighting Robot Competition was created to allow young engineers to test their creativity and problem solving skills. While the robot that is built for the contest would be incapable of extinguishing actual house fires, it serves as a protoype for a full scale fire fighting robot. The general principles of sound detection, navigation, obstacle avoidance, and fire detection could be applied to a full scale robot similarly to how they were applied to this prototype.

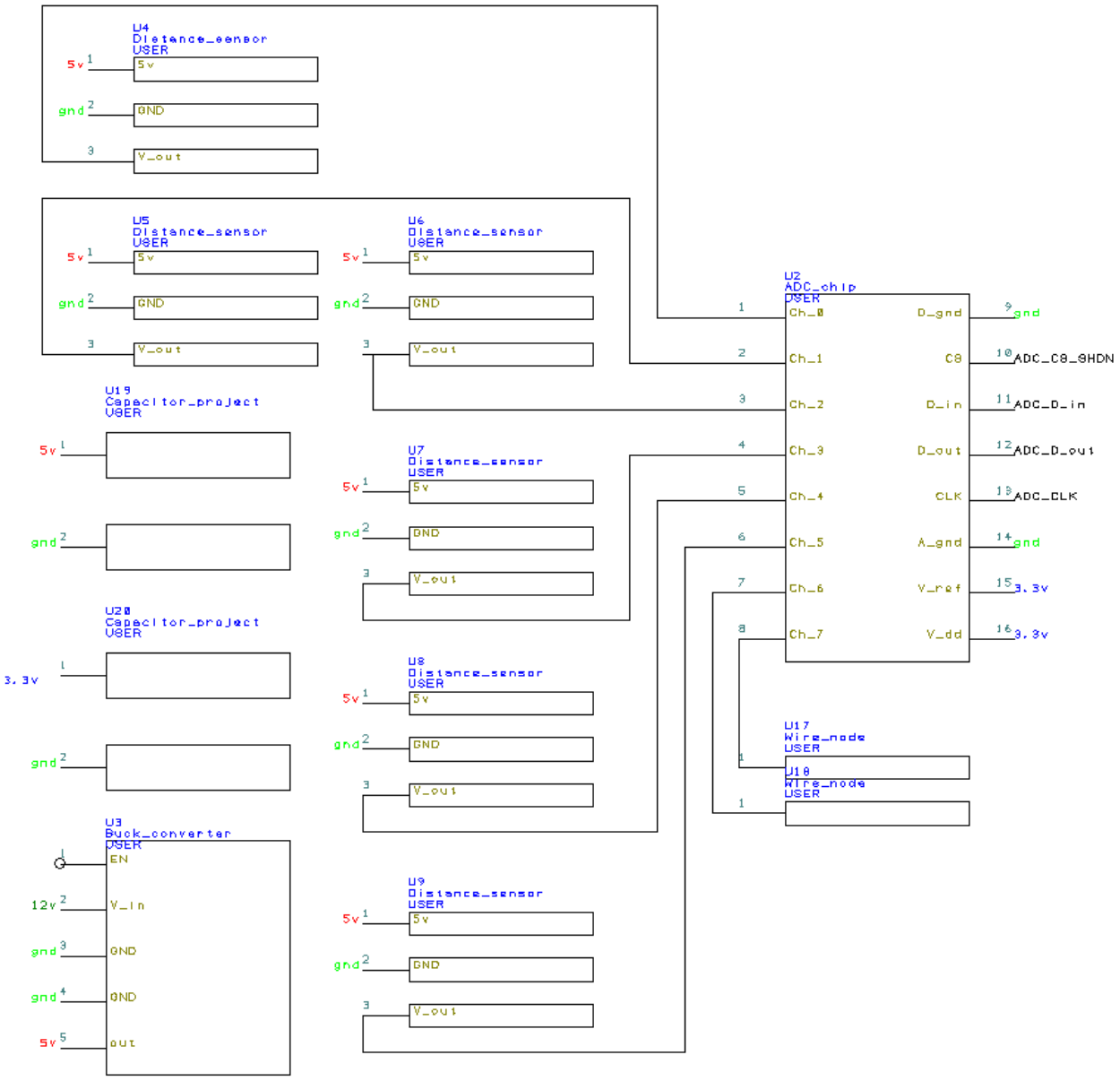
The main recommendation we would make for a second iteration of this design would be to increase the speed of the robot. The robot reliably navigated the maze, but did so slowly, which seems counterintuitive for a fire fighting robot. Increasing the base speed and making other necessary changes to compensate for the increased speed would lead to a more effective, better scoring robot. We might also recommend an LCD or similar screen be installed on the top of the robot to give the user feedback and allow for easier troubleshooting.

VI. Appendices

A. Hardware Design



A. Hardware Design Continued



B. Code

```
#Fire Fighting Robot Code
#Isaiah Frey and Jacob Stratman
#This robot will navigate a maze and extinguish a fire.
#!/usr/bin/env python3

"""-----------------------------------Update Log---------------------------------
Version
Major Update.Medium Update. Minor Update/Bug fix
V 1.0 - Initial programming
V 2.0 - Alpha testing
V 3.0 - Beta testing
V 1.0.0    12/1/17
   Added 4 pwm ports to move motors forward and backward
V 1.1.0    12/2/17
   Changed to 2 pwm pins and a directional variable
V 1.2.0    1/8/18
   Fixed encoder code and added Move_forward_distance Function
V 1.3.0    1/22/18
   Added ADC setup code to main program and Find_short_dist Function
V 1.3.1    1/23/18
   Fixed incompatibility between ADC setup and main setup code
   Switched to BCM mode for consistency
V 1.4.0    1/23/18
   Programmed read short distance sensor function and turn until aligned
   Added main loop and test loop
V 1.5.0    1/24/18
   Programmed right wall following
   Programmed read long distance sensor
V 1.5.1    1/26/18
   Made program run upon boot
V 1.5.2    1/29/18
   Programmed test loop to test all movement functions
V 1.6.0    2/2/18
   Programmed room measuring and room determination
V 1.6.1    2/4/18
   Fixed Turn_until_aligned bounding bug
V 1.6.2    2/5/18
   Improved wall following code
V 2.0.0    2/8/18
   Robot now wall follows.
V 2.1.0    2/12/18
   Robot exit first room code tested and works successfully
   This includes Turn_until_aligned, Move_forward, and Turn_left
V 2.1.1    2/14/18
   Edited wall following to allow for right turns as well as left turns
V 2.1.2    2/16/18
   Began construction of beginning sequence in main loop
V 2.1.3    2/19/18
   Added Determine_entrance function to allow the robot to determine which door it is
at
V 2.2.0    2/20/18
   Added programming mode to keep the robot from running during programming if right
back sensor is blocked
   Added config variables to allow the robot to remember what configuration the maze
is in.
   Set a general plan for how to navigate the maze in the main loop
V 2.3.0    2/21/18
   Added line sensors and line sensor function Check_line
   Robot now checks for a line during a right turn approximately once per millimeter
V 2.4.0    2/23/18
   Fixed edge cases for navigating out of first room
   Added Search_for_flame function and double check feature
V 2.4.1    2/24/18
```

```
   Added instructions for navigation from position 1, 2, and 3
   Changed Move_forward_while_checking function to also check for a line
   Added checking for a dog to Right_wall_follow_until_door function
V 2.4.2    2/25/18
   Fixed multiple variable passing errors with Navigate_out_of_room function
   Adjusted area calculation in Determine_room to be width*length instead of adding up
sensors
   Added an additional 2 cm buffer for line sensors on length of Determine_room
V 2.4.3    2/26/18
   Edited and tested Measure_room function
   Robot successfully  moves from room 1 to room 2 and room 2 to room 3
V 3.0.0    2/27/18
   Finished all basic positions for naviagation
   Created a Align_to_line function to line up the bot whenever it passes over a door
V 3.0.1    2/28/18
   Edited wall following function, no longer delays for .02 seconds between samples
   Added line checking between each sensor reading during wall following
   Made line checking 3X as frequent during right turns during wall following
   Added room checked array to help determine which rooms the robot has already been
in
V 3.0.2    3/1/18
   Edited Search_room function to be room specific
V 3.0.3    3/1/18
   Edited Search _room to apply to all rooms and always search for a left wall
   Edited position 3 code to more accurately align with hall 4
   Edited position 2 code to include the entrance position 2 ends at in room 3
V 3.1.0    3/1/18
   Edited position 6 code to move the robot from the left wall of room 4 to the right
wall of room 1
   for greater accuracy
   Robot completes  navigation of the maze of the first time
V 3.1.1    3/12/18
   Adjusted room searching code for room 3
V 3.1.2    3/14/18
   Fixed entrance code in room 3
   Adjusted  position 5 to work if robot starts in room 4
V 3.1.3    3/16/18
   Programmed and tested UV Tron
   Programmed and created a test for MOSFET for fire extinguisher
V 3.2.0    3/17/18
   Programmed Detect_flame and Extinguish_flame functions
   Improved Search_room function to actually search for a fire
   Fixed error in Align_to_line function where bot would continually align if it
missed the line
V 3.2.1    3/17/18
   Improved Extinguish_flame function to turn toward the fire and approach it between
scans
V 3.2.2    3/18/18
   Improved Search_room function to return robot to previous position after searching
for fire
   Improved Extinguish_flame function to approach proportional to the square root of
the distance
V 3.3.0    3/24/18
   Added Sound_start functionwhich detects a 3.5-4.1 kHz signal and starts the robot
"""

#---------------------------------------Assignments---------------------------------#
#Wheel diameter approximately 10cm
#Room areas 1: 6264 cm^2
#           2: 7416 cm^2
#           3: 11590 cm^2
#           4: 3570 cm^2

```

```
#Motor Assignments
#GPIO 17 - Left motor
#GPIO 27 - Right motor
#GPIO 22 - Left motor direction
#GPIO 10 - Right motor direction

#Encoder Assignments
#GPIO 23 - Left encoders
#GPIO 5 - Right encoders

#8 Channel ADC
#Channel 0 - Front Left Sensor
#Channel 1 - Front Right Sensor
#Channel 2 - Right Front Sensor
#Channel 3 - Front Middle Sensor
#Channel 4 - Long Range Left Sensor
#Channel 5 - Long Range Back Sensor
#Channel 6 - Servo
#Channel 7 -

#Line sensors
#GPIO 4 - Left
#GPIO 14 - Middle
#GPIO 15 - Right

#Servo Motor
#GPIO 21 - Servo PWM

#UV Tron
#GPIO 12 - UV Tron (Default high, pulses when detects fire)

#Extinguisher switch
#GPIO 16 - MOSFET

#-------------------------------------Import Libraries---------------------------------
#
import RPi.GPIO as GPIO           #Import a library to setup GPIO
import time                       #Import a library to setup clocks
from time import sleep            #Import the sleep function for delays
import Adafruit_GPIO.SPI as SPI   #Import SPI (Serial Peripherial Interface)
import Adafruit_MCP3008           #Import a library for ADC
from math import atan             #Import the arc tangent function from math
from math import sqrt             #Import the square root funtion from math
import pyaudio                    #Import a library for sound detection
from numpy import zeros,linspace,short,fromstring,hstack,transpose,log  #import
functions for  signal analysis
from scipy import fft             #Import a function to perform the fast fourier
transform

#-------------------------------------Initializations---------------------------------
#
GPIO.setmode(GPIO.BCM)     #Set the pins to the BOARD configuration
GPIO.setwarnings(False)      #Do not warn the user about pin initialization

GPIO.setup(17, GPIO.OUT)     #Set GPIO 17 as output
GPIO.setup(27, GPIO.OUT)     #Set GPIO 27 as output
GPIO.setup(22, GPIO.OUT)     #Set GPIO 22 as output for left motor direction
GPIO.setup(10, GPIO.OUT)     #Set GPIO 10 as output for right motor direction

GPIO.setup(4, GPIO.IN,)      #Set GPIO 4 as the input for the line sensors
GPIO.setup(14, GPIO.IN,)     #Set GPIO 14 as the input for the line sensors
GPIO.setup(15, GPIO.IN,)     #Set GPIO 15 as the input for the line sensors

```

```
GPIO.setup(23, GPIO.IN)      #Set GPIO 23 to input for encoders
GPIO.setup(5, GPIO.IN)       #Set GPIO 5 to input for encoders

GPIO.setup(21, GPIO.OUT)     #Set GPIO 21 as output for servo

GPIO.setup(12, GPIO.IN)      #Set GPIO 12 as input for UV Tron

GPIO.setup(16, GPIO.OUT)     #Set GPIO 16 as output for extinguisher switch

pwm_left = GPIO.PWM(17,100) #Initialize GPIO 17 at a 100 Hz frequency
pwm_right = GPIO.PWM(27,100)#Initialize GPIO 27 at a 100 Hz frequency
pwm_servo = GPIO.PWM(21,90)#Initialize GPIO 21 at a 100 Hz frequency

###TEST
GPIO.setup(16, GPIO.OUT)

# Setup software SPI configuration:
CLK  = 18
MISO = 23
MOSI = 24
CS   = 25
mcp = Adafruit_MCP3008.MCP3008(clk=CLK, cs=CS, miso=MISO, mosi=MOSI)

#Variable Initializations
start = False          #Set this true when a 3.8 kHz signal is detected
fire_extinguished = False  #Set to true when the fire has been put out
level = 1              #Set this for the level of the maze the robot is running
room3_config = 0       #0 for unknown, 1 for lower entrance, 2 for upper entrance
room4_config = 0       #0 for unknown, 1 for lower entrance, 2 for upper entrance
dog_config = 0         #0 for unknown, 1 for lower placement, 2 for right, 3 for upper
position = 0           #Set to a number 1-5 for based on the part of the sequence the
robot is at
hall = 0               #Set to a number 1-6 based on which hallway the robot is in
room = 0               #Initialize the room to 0 (none of the rooms)
inside_room = True     #Set to false when in a hall and true when in a room
line_detected = False #Set to true when a line is detected
room_checked = [False,False,False,False] #Set an array to mark off which rooms were
checked

#---------------------------------------Definitions-----------------------------------
-#
#Speeds
TURNSPEED = 13.6  #Speed robot moves while turning
SPEED = 35        #Speed robot moves while going forward (optimal at 35)

#Delays
TURN = .02        #Time delay the motors must run to turn 1 degree
CONVCM = .2       #Convert the time delay to centimeters
PAUSE = .2        #Wait time between movements
TRANS = .5        #Set the transition delay
EXTINGUISH = 1    #Set the time the Versa Valve will be open

#Constants
PI = 3.1415926535 #Define pi

#Tolerances
ALIGNTOL = .5     #Tolerance in cm that front sensors must be before driving forward
OBSTACLETOL = 3   #Tolerance in cm that the side sensors must differe from the center
sensor to be an obstacle
MAXSHORT = 25     #Tolerance for how far in cm short range distance sensor can pick up
WALLDIST = 12     #Tolerance between robot and wall while wall folowing
FRONTDIST = 12    #Tolerance between robot and obstacles in front

```

```
#Dimensions
WIDTH = 20        #Define the robot width in cm
LENGTH = 23       #Define the robot length in cm

#PD Controller
P = 6             #Define proportional constant
D = .12           #Define derivitave constant
T = .02           #Define time constant

#---------------------------------------FUNCTIONS-----------------------------------
-#
#-------------------------------------Sound Detection-------------------------------
-#

def Sound_start():
    #Volume Sensitivity, 0.05: Extremely Sensitive, may give false alarms
    #             0.1: Probably Ideal volume
    #             1: Poorly sensitive, will only go off for relatively loud
    SENSITIVITY= 1.0
    # Alarm frequencies (Hz) to detect (Use audacity to record a wave and then do
Analyze->Plot Spectrum)
    TONE = 3500
    #Bandwidth for detection (i.e., detect frequencies within this margin of error of
the TONE)
    BANDWIDTH = 30
    #How many 46ms blips before we declare a beep? (Take the beep length in ms, divide
by 46ms, subtract a bit)
    beeplength=8
    # How many beeps before we declare an alarm?
    alarmlength=5
    # How many false 46ms blips before we declare the alarm is not ringing
    resetlength=10
    # How many reset counts until we clear an active alarm?
    clearlength=30
    # Enable blip, beep, and reset debug output
    debug=False
    # Show the most intense frequency detected (useful for configuration)
    frequencyoutput=True


    #Set up audio sampler -
    NUM_SAMPLES = 2048
    SAMPLING_RATE = 44100
    pa = pyaudio.PyAudio()
    _stream = pa.open(format=pyaudio.paInt16,
                      channels=1, rate=SAMPLING_RATE,
                      input=True,
                      frames_per_buffer=NUM_SAMPLES)

    print("Alarm detector working. Press CTRL-C to quit.")

    blipcount=0
    beepcount=0
    resetcount=0
    clearcount=0
    alarm=False
    thefreq = 0

    while (thefreq < 3420 or thefreq > 4180):
        while _stream.get_read_available()< NUM_SAMPLES: sleep(0.01)
        audio_data  = fromstring(_stream.read(
             _stream.get_read_available(), exception_on_overflow = False),
dtype=short)[-NUM_SAMPLES:]
```

```
        # Each data point is a signed 16 bit number, so we can normalize by dividing
32*1024
        normalized_data = audio_data / 32768.0
        intensity = abs(fft(normalized_data))[:int(NUM_SAMPLES/2)]
        frequencies = linspace(0.0, float(SAMPLING_RATE)/2, num=NUM_SAMPLES/2)
        if frequencyoutput:
            which = intensity[1:].argmax()+1
            # use quadratic interpolation around the max
            if which != len(intensity)-1:
                y0,y1,y2 = log(intensity[which-1:which+2:])
                x1 = (y2 - y0) * .5 / (2 * y1 - y2 - y0)
                # find the frequency and output it
                thefreq = (which+x1)*SAMPLING_RATE/NUM_SAMPLES
            else:
                thefreq = which*SAMPLING_RATE/NUM_SAMPLES
            print("\t\t\t\tfreq=",thefreq)
        if max(intensity[(frequencies < TONE+BANDWIDTH) & (frequencies > TONE-
BANDWIDTH )]) > max(intensity[(frequencies < TONE-1000) & (frequencies > TONE-2000)])
+ SENSITIVITY:
            blipcount+=1
            resetcount=0
            if debug: print("\t\tBlip",blipcount)
            if (blipcount>=beeplength):
                blipcount=0
                resetcount=0
                beepcount+=1
                if debug: print("\tBeep",beepcount)
                if (beepcount>=alarmlength):
                    clearcount=0
                    alarm=True
                    print("Alarm!")
                    beepcount=0
        else:
            blipcount=0
            resetcount+=1
            if debug: print("\t\t\treset",resetcount)
            if (resetcount>=resetlength):
                resetcount=0
                beepcount=0
                if alarm:
                    clearcount+=1
                    if debug: print("\t\tclear",clearcount)
                    if clearcount>=clearlength:
                        clearcount=0
                        print("Cleared alarm!")
                        alarm=False
        sleep(0.01)
    print("Start")
    start = True
    return start

#-----------------------------------Movement Functions-----------------------------
-#
def Move_forward(delay, speed): #Move forward for a set time with speed 0-100
    GPIO.output(22,GPIO.HIGH)   #Set left motor to forward
    GPIO.output(10,GPIO.LOW)    #Set right motor to forward
    pwm_left.start(speed)
    pwm_right.start(speed)
    time.sleep(delay*CONVCM)
    Stop(PAUSE)

def Move_forward_while_checking(delay, speed, line_detected): #Move forward for a set
time with speed 0-100
```

```
        GPIO.output(22,GPIO.HIGH)    #Set left motor to forward
        GPIO.output(10,GPIO.LOW)    #Set right motor to forward
        pwm_left.start(speed)     #Start the left motor with a correction factor
        pwm_right.start(speed)
        for x in range (0,delay*10):
            time.sleep(.1*CONVCM)
            sensor0 = Find_short_dist(0)
            sensor1 = Find_short_dist(1)
            line_detected = Check_line(line_detected)
            if (line_detected):
                break
            if ((sensor0 < (MAXSHORT - 5)) or (sensor1 < (MAXSHORT - 5))):
                break
        Stop(PAUSE)
        return line_detected

    def Move_forward_distance(clicks, speed):
        GPIO.output(22,GPIO.HIGH)    #Set left motor to forward
        GPIO.output(10,GPIO.LOW)    #Set right motor to forward
        pwm_left.start(speed)
        pwm_right.start(speed)
        right = 0
        while(right < clicks):
            GPIO.wait_for_edge(5, GPIO.FALLING)
            right += 1
        Stop(PAUSE)

    def Move_forward_until_dist(distance, speed):  #Move robot forward until a distance
    from a wall
        sensor0 = Find_short_dist(0)
        sensor1 = Find_short_dist(1)
        escape = False    #If the bot moves forward don't have it move backward
        if (sensor0 > (distance) and sensor1 > (distance)):
            escape = True
        while (sensor0 > (distance) and sensor1 > (distance)):
            GPIO.output(22,GPIO.HIGH)    #Set left motor to forward
            GPIO.output(10,GPIO.LOW)    #Set right motor to forward
            pwm_left.start(speed)
            pwm_right.start(speed)
            sensor0 = Find_short_dist(0)
            sensor1 = Find_short_dist(1)
        if (escape):
            pwm_left.stop()
            pwm_right.stop()
            Stop(PAUSE)
            return None
        sensor0 = Find_short_dist(0)
        sensor1 = Find_short_dist(1)
        while (sensor0 < (distance) and sensor1 < (distance)):
            GPIO.output(22,GPIO.LOW)    #Set left motor to backward
            GPIO.output(10,GPIO.HIGH)    #Set right motor to backward
            pwm_left.start(speed)
            pwm_right.start(speed)
            sensor0 = Find_short_dist(0)
            sensor1 = Find_short_dist(1)
        pwm_left.stop()
        pwm_right.stop()
        Stop(PAUSE)

    def Transition_forward(delay, speed):
        GPIO.output(22,GPIO.HIGH)    #Set left motor to forward
        GPIO.output(10,GPIO.LOW)     #Set right motor to forward
        pwm_left.start((speed/2)+2) #Start the left motor with a correction factor
```

```
442         pwm_right.start(speed/2)
443         time.sleep(delay*CONVCM)
444 
445     def Transition_backward(delay, speed):
446         GPIO.output(22,GPIO.LOW)    #Set left motor to forward
447         GPIO.output(10,GPIO.HIGH)    #Set right motor to forward
448         pwm_left.start((speed/2)+2)     #Start the left motor with a correction factor
449         pwm_right.start(speed/2)
450         time.sleep(delay*CONVCM)
451 
452     def Move_backward(delay, speed): #Move forward for a set time with speed 0-100
453         GPIO.output(22,GPIO.LOW)     #Set left motor to backward
454         GPIO.output(10,GPIO.HIGH)     #Set right motor to backward
455         pwm_left.start(speed)
456         pwm_right.start(speed)
457         time.sleep(delay*CONVCM)
458         Stop(PAUSE)
459 
460     def Stop(delay): #Stop the motors
461         pwm_left.stop()
462         pwm_right.stop()
463         pwm_left.stop()
464         pwm_right.stop()
465         time.sleep(delay)
466 
467     def Turn_right(angle,speed): #Turn right a specified angle
468         GPIO.output(22,GPIO.HIGH)     #Set left motor to backward
469         GPIO.output(10,GPIO.HIGH)    #Set right motor to forward
470         pwm_left.start(speed)
471         pwm_right.start(speed)
472         time.sleep(angle*TURN)
473         Stop(PAUSE)
474 
475     def Turn_left(angle,speed): #Turn left a specified angle
476         GPIO.output(22,GPIO.LOW)    #Set left motor to forward
477         GPIO.output(10,GPIO.LOW)     #Set right motor to backward
478         pwm_right.start(speed)
479         pwm_left.start(speed)
480         time.sleep(angle*TURN)
481         Stop(PAUSE)
482 
483     def Turn_until_aligned():
484         sensor0 = Find_short_dist(0)     #Get front sensor values
485         sensor1 = Find_short_dist(1)
486         while (sensor0 < (sensor1 - ALIGNTOL) or sensor0 > (sensor1 + ALIGNTOL)):  #If the
487     robot is misaligned
488             angle = atan((abs(sensor0 - sensor1)/WIDTH))*(180.0/PI) #Find the angle robot
489     should turn in degrees
490             if (sensor0 < sensor1):              #Turn towards the smaller sensor value
491                 Turn_left(angle,TURNSPEED)
492             if (sensor0 > sensor1):
493                 Turn_right(angle,TURNSPEED)
494             sensor0 = Find_short_dist(0)         #Check front sensor values to see if they
495     are aligned
496             sensor1 = Find_short_dist(1)
497         Stop(PAUSE)
498 
499     def Align_on_line():
500         x = 0
501         if ((not GPIO.input(4)) and GPIO.input(15)):   #If the left line sensor is over a
502     line and the right is not
503             while(GPIO.input(15) and x<10000):     #While the right line sensor is not on
504     a line
```

```
            GPIO.output(10,GPIO.LOW)    #Set right motor to forward
            pwm_right.start(SPEED/4)    #Turn on the right motor
            x += 1      #Increment a counter to prevent overshoot errors
        Stop(PAUSE)      #Stop the motors
    if (GPIO.input(4) and (not GPIO.input(15))):   #If the right line sensor is over a
line and the left is not
        while(GPIO.input(4) and x < 10000):      #While the left line sensor is not on
a line
            GPIO.output(22,GPIO.HIGH)    #Set left motor to forward
            pwm_left.start(SPEED/4)      #Turn on the left motor
            x += 1      #Increment a counter to prevent overshoot errors
        Stop(PAUSE)      #Stop the motors
    if (GPIO.input(4) and GPIO.input(15)):    #If neither sensor is above a line,
assume it was overshot while turning
        while(GPIO.input(14) and x < 10):     #While the middle sensor doesn't see a
line
            Transition_backward(.1,SPEED/2)    #Go backwards at 1/4th speed
            x += 1      #Increment a counter to prevent overshoot errors

def Right_wall_follow_until_door(inside_room, hall, dog_config):
    line_detected = False #Set this true when the line is detected
    Transition_forward(TRANS, SPEED)
    sensor_left_old = WALLDIST      #Initialize previous values as current values
    GPIO.output(22,GPIO.HIGH)    #Set left motor to forward
    GPIO.output(10,GPIO.LOW)     #Set right motor to forward
    while (not line_detected):
        #Find current sensor value and average 5 samples
        sensor2 = 0
        for x in range(0,5):
            sensor2 += Find_short_dist(2)
            line_detected = Check_line(line_detected)
            if (line_detected):    #Check for a line while checking sensors
                Stop(PAUSE)
                return line_detected, dog_config
        sensor_left = sensor2/5.0
        #Set the motor adjustment based on distance from target and previous distance
        proportional = -P*(WALLDIST - sensor_left)
        differential = D*(sensor_left - sensor_left_old)/T
        adjust_left = (proportional + differential)
        #Set the adjustment to be inside the duty cycle range
        if ((SPEED - adjust_left) <= 0 or (SPEED + adjust_left) <= 0):
            adjust_left = 0
        if ((SPEED + adjust_left) >= 100 or (SPEED - adjust_left) >=100):
            adjust_left = 50
        #Change the motor speeds according to the adjustment
        pwm_left.start(SPEED + adjust_left)
        pwm_right.start(SPEED - adjust_left)
        #Set the sensor_old to the current sensor values
        sensor_left_old = sensor_left
        #Check to see if the front sensor has dropped out of range, if so, turn right
        if (sensor_left > MAXSHORT):
            Transition_forward(TRANS, SPEED)
            for x in range (0,SPEED*14):
                Transition_forward(.01,SPEED*2)
                line_detected = Check_line(line_detected)
                if (line_detected):
                    Stop(PAUSE)
                    return line_detected, dog_config
            Stop(PAUSE)
            Turn_right(90,TURNSPEED)
            Transition_forward(TRANS, SPEED)
            for x in range (0,SPEED*14):
                Transition_forward(.01,SPEED*2)
```

```
                line_detected = Check_line(line_detected)
                if (line_detected):
                    Stop(PAUSE)
                    return line_detected, dog_config
            Stop(PAUSE)
            sensor_left_old = Find_short_dist(2)
        #Check to the middle sensor for a wall
        sensor6 = Find_short_dist(6)   #Get middle front sensor value
        if (sensor6 < (WALLDIST + .5)):      #See if a wall or obstacle has been
spotted in front of robot
            obstacle = False
            if (hall == 3 or hall == 4 or room == 4):
                obstacle = Scan_for_obstacle()
            if (inside_room and obstacle):
                Move_around_obstacle()
            if ((not inside_room) and obstacle):
                if (hall == 3):
                    dog_config = 1
                    ###
                    pwm_servo.start(5)
                    time.sleep(1)
                    ###
                    break
                if (hall == 4):
                    dog_config = 2
                    ###
                    pwm_servo.start(10)
                    time.sleep(1)
                    ###
                    break
                else:
                    dog_config = 3
                    ###
                    pwm_servo.start(15)
                    time.sleep(1)
                    ###
                    break
            else:
                Turn_left(90,TURNSPEED)
                Stop(PAUSE)
                GPIO.output(22,GPIO.HIGH)   #Set left motor to forward
                GPIO.output(10,GPIO.LOW)    #Set right motor to forward
                Transition_forward(TRANS, SPEED)
        #Check to see if a line is detected
        line_detected = Check_line(line_detected)
    Stop(PAUSE)
    return line_detected, dog_config

def Right_wall_follow_until_turn(inside_room, hall, dog_config):
    line_detected = False #Set this true when the line is detected
    Transition_forward(TRANS, SPEED)
    sensor_left_old = WALLDIST      #Initialize previous values as current values
    GPIO.output(22,GPIO.HIGH)   #Set left motor to forward
    GPIO.output(10,GPIO.LOW)    #Set right motor to forward
    while (not line_detected):
        #Find current sensor value and average 5 samples
        sensor2 = 0
        for x in range(0,5):
            sensor2 += Find_short_dist(2)
        sensor_left = sensor2/5.0
        #Set the motor adjustment based on distance from target and previous distance
        proportional = -P*(WALLDIST - sensor_left)
        differential = D*(sensor_left - sensor_left_old)/T
```

```
631             adjust_left = (proportional + differential)
632             #Set the adjustment to be inside the duty cycle range
633             if ((SPEED - adjust_left) <= 0 or (SPEED + adjust_left) <= 0):
634                 adjust_left = 0
635             if ((SPEED + adjust_left) >= 100 or (SPEED - adjust_left) >=100):
636                 adjust_left = 50
637             #Change the motor speeds according to the adjustment
638             pwm_left.start(SPEED + adjust_left)
639             pwm_right.start(SPEED - adjust_left)
640             #Set the sensor_old to the current sensor values
641             sensor_left_old = sensor_left
642             #Check to the middle sensor for a wall
643             sensor6 = Find_short_dist(6)   #Get middle front sensor value
644             if (sensor6 < (WALLDIST + .5)):      #See if a wall or obstacle has been
645     spotted in front of robot
646                 obstacle = False
647                 if (hall == 3 or hall == 4 or room == 4):
648                     obstacle = Scan_for_obstacle()
649                 if (inside_room and obstacle):
650                     Move_around_obstacle()
651                 if ((not inside_room) and obstacle):
652                     if (hall == 3):
653                         dog_config = 1
654                         ###
655                         pwm_servo.start(5)
656                         time.sleep(1)
657                         ###
658                         break
659                     if (hall == 4):
660                         dog_config = 2
661                         ###
662                         pwm_servo.start(10)
663                         time.sleep(1)
664                         ###
665                         break
666                     else:
667                         dog_config = 3
668                         ###
669                         pwm_servo.start(15)
670                         time.sleep(1)
671                         ###
672                         break
673                 else:
674                     Stop(PAUSE)             #Once a wall is found stop to prevent current
675     overload of motors
676                     Turn_left(90,TURNSPEED) #Make a left turn
677                     break   #Once a left turn is found exit the wall follow
678             #Check to see if a line is detected
679             line_detected = Check_line(line_detected)
680         Stop(PAUSE)
681         return line_detected, dog_config
682 
683     def Right_wall_follow_for_distance(distance):
684         Transition_forward(TRANS, SPEED)
685         sensor_left_old = WALLDIST      #Initialize previous values as current values
686         GPIO.output(22,GPIO.HIGH)   #Set left motor to forward
687         GPIO.output(10,GPIO.LOW)    #Set right motor to forward
688         distance_traveled = 0       #Set the current distance_traveled
689         while (distance_traveled < distance):     #While the distance still hasn't been
690     reached
691             #Find current sensor value and average 5 samples
692             sensor2 = 0
693             for x in range(0,5):
```

```
694             sensor2 += Find_short_dist(2)
695         sensor_left = sensor2/5.0
696         #Set the motor adjustment based on distance from target and previous distance
697         proportional = -P*(WALLDIST - sensor_left)
698         differential = D*(sensor_left - sensor_left_old)/T
699         adjust_left = (proportional + differential)
700         #Set the adjustment to be inside the duty cycle range
701         if ((SPEED - adjust_left) <= 0 or (SPEED + adjust_left) <= 0):
702             adjust_left = 0
703         if ((SPEED + adjust_left) >= 100 or (SPEED - adjust_left) >=100):
704             adjust_left = 50
705         #Change the motor speeds according to the adjustment
706         pwm_left.start(SPEED + adjust_left)
707         pwm_right.start(SPEED - adjust_left)
708         #Set the sensor_old to the current sensor values
709         sensor_left_old = sensor_left
710         distance_traveled += .1    #Add the distance traveled
711     Stop(PAUSE)
712     return line_detected, dog_config
713
714 def Move_around_obstacle():
715     Turn_left(360,TURNSPEED)
716     Stop(PAUSE)
717
718 def Enter_and_exit_room():
719     Transition_forward(TRANS, SPEED)
720     Move_forward(5, SPEED)
721     Turn_left(180, TURNSPEED)
722     sensor2 = Find_short_dist(2)
723     if (sensor2 < (WALLDIST + 4)):
724         Right_wall_follow_until_door(inside_room, hall, dog_config)
725     else:
726         Transition_forward(TRANS, SPEED)
727         Move_forward(5, SPEED)
728
729 #-----------------------------------Sensor Functions--------------------------------
730 -#
731 def Check_line(line_detected):
732     if ((not GPIO.input(4)) or (not GPIO.input(14)) or (not GPIO.input(15))):
733             line_detected = True
734     return line_detected
735
736 def Find_short_dist(sensor):
737     voltage = mcp.read_adc(sensor)#*(3/1023) #Convert digital value to voltage
738     if (voltage == 0):
739         print ("Sensor out of range")
740         voltage = .1
741         #Linearize the sensor value to find distance
742     voltage = voltage * 3.0
743     voltage = voltage / 1023.0
744     distance = (13.2/voltage) - 0.42
745     return distance                         #return the distance in cm
746
747 def Find_long_dist(sensor):
748     voltage = mcp.read_adc(sensor)*(3.0/1023) #Convert digital value to voltage
749     if (voltage == 0):
750         print ("Sensor out of range")
751         voltage = .1
752     distance = (60/voltage)
753     return distance
754
755 def Scan_for_obstacle():   #Return True if the sensor is seeing an obstacle, False for
756 a wall
```

```
757         sensor6 = Find_short_dist(6)
758         Turn_right(10,TURNSPEED)
759         sensor1 = Find_short_dist(1)
760         Turn_left(20,TURNSPEED)
761         sensor0 = Find_short_dist(0)
762         Turn_right(10,TURNSPEED)
763         if ((sensor1 > (sensor6 + OBSTACLETOL)) or (sensor0 > (sensor6 + OBSTACLETOL))):
764             print("Obstacle found")
765             return True
766         else:
767             print("Wall found")
768             return False
769 
770     def Measure_room():
771         sensor4_total = 0
772         sensor5_total = 0
773         sensor2_total = 0
774         for x in range(0, 40):
775             sensor4 = Find_long_dist(4)
776             sensor5 = Find_long_dist(5)
777             sensor2 = Find_short_dist(2)
778             sensor4_total += sensor4
779             sensor5_total += sensor5
780             sensor2_total += sensor2
781             Transition_backward(.1,SPEED)
782         sensor4_avg = sensor4_total/40
783         sensor5_avg = sensor5_total/40
784         sensor2_avg = sensor2_total/40
785         ###TEST
786         print("Left: ", sensor4_avg)
787         print("Right: ", sensor2_avg)
788         print("Back: ", sensor5_avg)
789         ###
790         width = (sensor4_avg + sensor2_avg + WIDTH)
791         length = (sensor5_avg + LENGTH + 3)        #Add 2 cm for line sensors sticking in
792     front of robot
793         area = (width)*(length)
794         line_detected = False
795         Move_forward_while_checking(5, SPEED/2, line_detected)
796         ###TEST
797         print("Width: ", width)
798         print("Length: ", length)
799         print("Area: ", area)
800         Stop(PAUSE)
801         return width, length, area
802 
803     #-----------------------------------Position Functions-------------------------------
804     -#
805 
806     def Determine_room(width, length, area):
807         room = 0
808         if (area < 4917):
809             room = 4
810             return room
811         if (area >= 4917 and area < 8000 and width < 85):
812             room = 1
813             return room
814         if (area >= 4917 and area < 8000 and width >= 85):
815             room = 2
816             return room
817         if (area >= 8000):
818             room = 3
819             return room
```

```
820         print ("Error with long distance sensors")
821         return room
822 
823     def Determine_entrance(room):
824         entrance = 0
825         sensor2 = Find_short_dist(2)    #Get front right sensor
826         sensor4 = Find_long_dist(4)     #Get left sensor
827         sensor5 = Find_long_dist(5)     #Get back sensor
828         if (room == 3 and sensor2 < MAXSHORT and sensor4 < (110 - (WIDTH + WALLDIST))):
829             entrance = 2  #room 3 left entrance is at the top
830         if (room == 3 and sensor2 >= MAXSHORT):
831             entrance = 1  #room 3 left entrance is at the bottom
832         if (room == 3 and sensor2 < MAXSHORT and sensor4 >= (110 - (WIDTH + WALLDIST))):
833             entrance = 3  #room 3 top entrance
834         if (room == 4):
835             Move_forward(8,SPEED)
836             sensor6 = Find_short_dist(6)
837             if (sensor6 < MAXSHORT):
838                 entrance = 5   #room 4 top entrance
839             else:
840                 entrance = 4   #room 4 bottom entrance
841         return entrance
842 
843     def Determine_hall(room, entrance):
844         hall = 0   #Catch all if room is not set
845         if (room == 1):
846             hall = 1
847         if (room == 2):
848             hall = 2
849         if (room == 3 and (entrance == 1 or entrance == 2)):
850             hall = 3
851         if (room == 3 and entrance == 3):
852             hall = 4
853         if (room == 4 and entrance == 4):
854             hall = 6
855         if (room == 4 and entrance == 5):
856             hall = 7
857         return hall
858 
859     def Set_config(room3_config, room4_config, dog_config, entrance):   #Set which
860     configuration the maze is in
861         if (entrance == 1):
862             room3_config = 1
863         if (entrance == 2):
864             room3_config = 2
865         if (entrance == 4):
866             room4_config = 1
867         if (entrance == 5):
868             room4_config = 2
869         return room3_config, room4_config, dog_config
870 
871     def Determine_position(room, hall, room3_config, room4_config, dog_config, entrance):
872         if (room == 1 or hall == 1):
873             position = 1
874         if (room == 2 or hall == 2):
875             position = 2
876         if ((room == 3 and (entrance == 1 or entrance == 2)) or hall == 3):
877             position = 3
878         if (room == 3 and entrance == 3):
879             position = 4
880         if (room == 4 and entrance == 5):
881             position = 5
882         if ((room == 4 and entrance == 4) or hall == 6):
```

```
            position = 6
        return position
    #------------------------------Fire Extinguishing Functions-------------------------
    #

    def Detect_flame():
        fire_check = 0
        for x in range(0,10000):
            fire_check += GPIO.input(12)
        if (fire_check < 10000):
            print ("Fire Detected")
            fire_detected = True
        else:
            print ("Fire not detected")
            fire_detected = False
        return fire_detected

    def Extinguish_flame(fire_detected):
        Stop(.1)              #Stop the robot while extinguishing
        return_angle = 0      #Initialize a value for how much the robot will turn so it
    can correct itself
        return_dist = 0       #Initialize a value for how far the robot will travel so it
    can correct itself
        while (fire_detected):
            location, sensor3 = Search_for_flame()
            pwm_servo.start(location)     #Move the servo to the face the flame
            if (sensor3 < 550):           #If the flame is close enough to the robot
                time.sleep(2)
                GPIO.output(16, GPIO.HIGH)#Open the CO2 canister
                time.sleep(EXTINGUISH)    #Hold open for a delay
                GPIO.output(16, GPIO.LOW) #Close the CO2 canister
            else:
                if(location > 10):
                    Turn_left((location-10)*9,TURNSPEED)
                    return_angle += (location-10)*9
                if(location < 10):
                    Turn_right((10-location)*9,TURNSPEED)
                    return_angle -= (10-location)*9
                Transition_forward(TRANS,SPEED)
                forward = (sqrt(sensor3)-23.4)*.8
                if (sensor3 < 900 and (location > 14 or location < 6)):
                    forward = forward/3      #If the sensor is close to the fire but still
    angled wrong, advance slowly
                print("forward:", forward)
                Move_forward(forward,SPEED)
                return_dist += forward
            fire_detected = Detect_flame()
        return return_dist, return_angle

    def Search_for_flame():   #A function using a phototransitor and servo to fine exact
    flame location
        Stop(.1)              #Stop the robot while searching
        #double_check = True   #Set a double check boolean to ensure proper positioning
        location_old = 1000   #Set a temporary location of the flame
        #while(double_check):  #While the function has not found the same value twice
        sensor3_temp = 10000            #Set a temporary phototransitor value
        pwm_servo.start(1)              #Start the servo at the far right side
        time.sleep(1)                   #Take a short pause
        for x in range (1,20):          #Sweep the servo right to left recording values on
    the way
            sensor3 = mcp.read_adc(3)
            print (sensor3)
            pwm_servo.start(x)
```

```
            time.sleep(.2)
            if (sensor3 < sensor3_temp):#Record the minimum value
                sensor3_temp = sensor3
                location = x - 1          #Set location to that value
            #if (location == location_old):  #Set the double_check if the same value was
obtained twice
            #    double_check = False
            #else:                          #Otherwise set the current location in the
temporary location
            #    location_old = location
        print(location)
        return location, sensor3_temp       #Return the position the robot should move its
extinguisher to and the strength of the flame

def Search_room(room, fire_extinguished, inside_room, hall, dog_config):
    line_detected = False
    fire_detected = Detect_flame()        #Use UV Tron to check for a fire
    Transition_forward(TRANS*3, SPEED)
    Move_forward(9, SPEED)
    if (fire_detected):              #If a fire was spotted
        return_dist, return_angle = Extinguish_flame(fire_detected)  #Extinguish the
fire
        fire_extinguished = True          #Set the fire to be extinguished
        Transition_backward(TRANS, SPEED)
        Move_backward(return_dist*.8,SPEED)
        if(return_angle < 0):
            Turn_left(return_angle*-1,TURNSPEED)
        if(return_angle >= 0):
            Turn_right(return_angle,TURNSPEED)
    sensor2 = Find_short_dist(2)
    #If at the bottom entrance of room 3 do some additional aligning
    if (sensor2 < WALLDIST + 6):
        Turn_right(90,TURNSPEED)
        Turn_until_aligned()
        Turn_left(155,TURNSPEED)
        fire_detected = Detect_flame()        #Use UV Tron to check for a fire
        if (fire_detected):                   #If a fire was spotted
            return_dist, return_angle = Extinguish_flame(fire_detected)  #Extinguish
the fire
            fire_extinguished = True          #Set the fire to be extinguished
            Transition_backward(TRANS, SPEED)
            Move_backward(return_dist*.8,SPEED)
            if(return_angle < 0):
                Turn_left(return_angle*-1,TURNSPEED)
            if(return_angle >= 0):
                Turn_right(return_angle,TURNSPEED)
    #Otherwise just turn left
    else:
        Turn_right(90,TURNSPEED)
        fire_detected = Detect_flame()        #Use UV Tron to check for a fire
        if (fire_detected):                   #If a fire was spotted
            return_dist, return_angle = Extinguish_flame(fire_detected)  #Extinguish
the fire
            fire_extinguished = True          #Set the fire to be extinguished
            Transition_backward(TRANS, SPEED)
            Move_backward(return_dist*.8,SPEED)
            if(return_angle < 0):
                Turn_left(return_angle*-1,TURNSPEED)
            if(return_angle >= 0):
                Turn_right(return_angle,TURNSPEED)
        Turn_left(180,TURNSPEED)
    Move_forward_until_dist(WALLDIST,SPEED/2)
    Turn_until_aligned()
```

```
    Move_forward_until_dist(WALLDIST,SPEED/2)
    Turn_left(90,TURNSPEED)
    line_detected, dog_config = Right_wall_follow_until_door(inside_room, hall,
dog_config)
    if (line_detected):
        Align_on_line()
    return fire_extinguished
    ###
    #Code for searching for candle and extinguishing it
    #return fire_extinguished
    ###

#-----------------------------------High Level Functions----------------------------
#

def Navigate_out_of_room(inside_room, hall, room3_config, room4_config, dog_config):
    Move_forward_until_dist(WALLDIST, SPEED)
    Turn_left(22.5, TURNSPEED)
    Turn_until_aligned()
    Move_forward_until_dist(WALLDIST, SPEED)
    Turn_left(90, TURNSPEED)
    line_detected, dog_config = Right_wall_follow_until_door(inside_room, hall,
dog_config)
    width, length, area = Measure_room()
    room = Determine_room(width, length, area)
    entrance = Determine_entrance(room)
    room3_config, room4_config, dog_config = Set_config(room3_config, room4_config,
dog_config, entrance)
    return room, entrance, room3_config, room4_config

#---------------------------------------Main Loop----------------------------------
#


###PROGRAMMING MODE
sensor2 = Find_short_dist(2)
if (sensor2 < 8):
    while(True):
        print ("Programming mode")
        Stop(1000)

#SIGNAL DETECTION
#Wait for 3.8 kHz singal
start = Sound_start()

#NAVIGATING OUT OF FIRST ROOM
#Initialize the motor speed
Move_forward(.1,SPEED)
Stop(PAUSE)
#Check if a there is a wall in front of robot
while (start):
    sensor0 = Find_short_dist(0)   #Get right front sensor
    sensor1 = Find_short_dist(1)   #Get left front sensor
    room_not_found = 0             #Set a variable to prevent an infinite loop while
searching for room
    #First navigate from the arbitrary starting room
    while(inside_room):
        if (sensor0 < MAXSHORT and sensor1 < MAXSHORT):
            room, entrance, room3_config, room4_config =
Navigate_out_of_room(inside_room, hall, room3_config, room4_config, dog_config)
#Navigate from the first room to a door
            if(line_detected):          #If a line was detected
                Align_on_line()         #Align on the line
```

```
                line_detected = False   #Reset the line sensors
            room_checked[room - 1] = True   #Mark off this room as checked for the
candle
            inside_room = False         #Tell the bot it's outside the room
            starting_room = room        #Remeber which room the robot began in so it
can return there
            hall = Determine_hall(room, entrance) #Decide which hall section the robot
is in
        else:
            Turn_left(45,TURNSPEED)             #Turn left 45 degrees
            room_not_found += 1                 #Increment a counter to prevent from
spinning in circles
            sensor0 = Find_short_dist(0)        #Check sensors again
            sensor1 = Find_short_dist(1)
        if (room_not_found > 7):                #If a wall is still not found after
checking all 8 directions
            Transition_forward(TRANS,SPEED)
            line_detected = Move_forward_while_checking(10,SPEED,line_detected)  #Move
forward looking for a line or wall
            sensor0 = Find_short_dist(0)   #Get right front sensor
            sensor1 = Find_short_dist(1)   #Get left front sensor
            if(line_detected):                  #If a line was detected
                Align_on_line()                 #Align on the line
                line_detected = False           #Reset the line sensors
                inside_room = False             #Tell the robot its outside the room
                width, length, area = Measure_room() #Measure the room
                room = Determine_room(width, length, area) #Determine which room it is
in
                starting_room = room            #Remeber which room the robot began in
so it can return there
                entrance = Determine_entrance(room)   #Set the entrance
                hall = Determine_hall(room, entrance) #Decide which hall section the
robot is in
                room3_config, room4_config, dog_config = Set_config(room3_config,
room4_config, dog_config, entrance)  #Set the configuration
                room_checked[room - 1] = True   #Mark off this room as checked for the
candle
            room_not_found = 2                  #Reset the robot to do 270 next check

    position = Determine_position(room, hall, room3_config, room4_config, dog_config,
entrance)   #Determine where in the maze the bot is
    while (not fire_extinguished or not (room == starting_room)):
        print("Position is: ", position)
        print("Hall is: ", hall)
        if (position == 1):   #If exiting room 1
            Transition_forward(TRANS*3, SPEED)  #Transition forward far enough to go
off line
            line_detected, dog_config = Right_wall_follow_until_door(inside_room,
hall, dog_config)  #Follow until the next room
            if(line_detected):                  #If a line was detected
                Align_on_line()                 #Align on the line
                line_detected = False           #Reset the line sensors
            inside_room = True                  #Tell the bot it's outside the room
            room = 2                            #Set which room bot is in
            Search_room(room, fire_extinguished, inside_room, hall, dog_config)
#Search the room the bot is in
            room_checked[room - 1] = True       #Mark off this room as checked for the
candle
            inside_room = False                 #Mark the bot is back in the hall
            hall = 2                            #Set the hall to be 2
            position = 2                        #Set the new position
        if (position == 2):  #If exiting room 2
```

```
            Transition_forward(TRANS*3, SPEED)  #Transition forward far enough to go
off line
            line_detected, dog_config = Right_wall_follow_until_door(inside_room,
hall, dog_config)  #Follow until the next room
            if(line_detected):                  #If a line was detected
                Align_on_line()                 #Align on the line
                line_detected = False           #Reset the line sensors
            inside_room = True                  #Tell the bot it's outside the room
            room = 3                            #Set which room bot is in
            Search_room(room, fire_extinguished, inside_room, hall, dog_config)
#Search the room the bot is in
            room_checked[room - 1] = True           #Mark off this room as checked for
the candle
            inside_room = False                 #Mark the bot is back in the hall
            hall = 3                            #Set the hall to 3
            sensor2 = Find_short_dist(2)        #Check the right front sensor
            if (sensor2 < (WALLDIST + 5)):      #If there is a wall to the right of
the bot
                entrance = 2                    #Set the entrance to be the left upper
entrance of room 3
            else:                               #Otherwise
                entrance = 1                    #Set the entrance to be the lower left
entance
            position = 3                        #Set the new position
        if (position == 3):  #If exiting room 3 on the left side
            Transition_forward(TRANS*3, SPEED)  #Transition forward far enough to go
off line
            if (entrance == 1):                 #If the robot is at the bottom
entrance of room 3
                Move_forward(3, SPEED/2)        #Move forward some to account for
earlier right wall drop off
            line_detected, dog_config = Right_wall_follow_until_door(inside_room,
hall, dog_config)  #Follow until the next room
            if (line_detected):                 #If there is no dog in hall 3, and
robot goes to entrance 3
                Align_on_line()                 #Align on the line
                line_detected = False           #Reset the line sensors
                Turn_left(90, TURNSPEED)        #Turn around to face hall 4
                Move_forward_until_dist(WALLDIST, SPEED/2) #Move to the wall of hall 4
                Turn_until_aligned()            #Turn until aligned to the wall
                Move_forward_until_dist(WALLDIST, SPEED/2) #Move to the wall of hall 4
                Turn_left(90,TURNSPEED)         #Turn to look down hall 4
                hall = 4                        #Set the hall to 4
                position = 4                    #Set the new position
            else:                               #If there is a dog in hall 3
                Turn_left(90, TURNSPEED)        #Turn left 90 degrees
                line_detected = Move_forward_while_checking(20,SPEED,line_detected)
#Go forward to look for room 4
                if (not line_detected):         #If lower entrance of room 4 isn't
found
                    hall = 8                    #Set the hall to 8
                    line_detected, dog_config =
Right_wall_follow_until_door(inside_room, hall, dog_config)  #Follow until the next
room
                    if(line_detected):                  #If a line was detected
                        Align_on_line()                 #Align on the line
                        line_detected = False           #Reset the line sensors
                    inside_room = True                  #Tell the bot it's outside the
room
                    room = 4                            #Set which room bot is in
                    Search_room(room, fire_extinguished, inside_room, hall,
dog_config)     #Search the room the bot is in
```

```
                    room_checked[room - 1] = True       #Mark off this room as checked
for the candle
                    inside_room = False                 #Mark the bot is back in the
hall
                    hall = 9                            #Set the hall to 9
                    entrance = 5                        #Set the entrance to be the
top entrance of room 4
                    position = 4                        #Set the new position
                if (line_detected):                     #If the entrance to room 4 was
found
                    Align_on_line()                     #Align on the line
                    line_detected = False               #Reset the line sensors
                    inside_room = True                  #Tell the bot it's outside the
room
                    room = 4                            #Set which room bot is in
                    Search_room(room, fire_extinguished, inside_room, hall,
dog_config)     #Search the room the bot is in
                    room_checked[room - 1] = True       #Mark off this room as checked
for the candle
                    inside_room = False                 #Mark the bot is back in the
hall
                    hall = 6                            #Set the hall to 6
                    entrance = 4                        #Set the entrance to be the
lower entrance of room 4
                    position = 6                        #Set the new position
        if (position == 4):                     #If the robot is at the top entrance
of room 3
            Transition_forward(TRANS*3, SPEED)  #Transition forward far enough to go
off line
            line_detected, dog_config = Right_wall_follow_until_turn(inside_room,
hall, dog_config)  #Follow until the top wall
            if (dog_config == 2):               #If there is a dog in hall 4
                Turn_left(90,TURNSPEED)         #Turn left 90 degrees
                Move_forward_while_checking(10,SPEED,line_detected)  #Move forward
looking for a wall
                Turn_left(90,TURNSPEED)         #Turn left 90 degrees
                hall = 7                        #Set the hall to 7
                position = 5                    #Set the new position
            else:
                Right_wall_follow_for_distance(46)  #Follow the right wall for 46 cm
                Turn_left(90,TURNSPEED)             #Turn right 90 degrees
                line_detected = Move_forward_while_checking(10,SPEED,line_detected)
#Move forward looking for a line or wall
                if (line_detected):                 #If the top entrance to room 4
was found
                    Align_on_line()                 #Align on the line
                    line_detected = False           #Reset the line sensors
                    inside_room = True              #Tell the bot it's outside the
room
                    room = 4                        #Set which room bot is in
                    Search_room(room, fire_extinguished, inside_room, hall,
dog_config)     #Search the room the bot is in
                    room_checked[room - 1] = True       #Mark off this room as checked
for the candle
                    inside_room = False                 #Mark the bot is back in the
hall
                    hall = 7                            #Set the hall to 7
                    entrance = 5                        #Set the entrance to be the
upper entrance of room 4
                    position = 5                        #Set the new position
                else:                                   #If room 4 entrance is on
bottom
                    Turn_left(90,TURNSPEED)             #Turn left 90 degrees
```

```
                    hall = 7                                #Set the hall to 7
                    position = 5                            #Set the new position
            if (position == 5):                         #If the robot is at the top entrance
of room 4
                if (room_checked[3] and (not dog_config == 3)):     #If the robot began in
room 4 and there isn't a dog in the top hall
                    Transition_forward(TRANS*3, SPEED)      #Move off the line
                    Move_forward_while_checking(30,SPEED,line_detected)  #Move forward
looking for a wall
                    Turn_until_aligned()                    #Turn until aligned with the
wall
                    Move_forward_until_dist(WALLDIST,SPEED/2) #Make sure the robot is
still WALLDIST away
                    Turn_left(90,TURNSPEED)                 #Turn left 90 degrees
                    line_detected, dog_config = Right_wall_follow_until_door(inside_room,
hall, dog_config)  #Follow until the next room
                    if (line_detected):                     #If the top entrance to room 4
was found
                        Align_on_line()                     #Align on the line
                        line_detected = False               #Reset the line sensors
                        inside_room = True                  #Tell the bot it's outside the
room
                        room = 1                            #Set which room bot is in
                        Search_room(room, fire_extinguished, inside_room, hall,
dog_config)     #Search the room the bot is in
                        room_checked[room - 1] = True       #Mark off this room as checked
for the candle
                        inside_room = False                 #Mark the bot is back in the
hall
                        hall = 1                            #Set the hall to 1
                        position = 1                        #Set the new position
                    else:
                        Turn_left(180,TURNSPEED)             #Turn left to face room 4
again
                        Move_forward_while_checking(200,SPEED,line_detected)
                        line_detected =
Move_forward_while_checking(10,SPEED,line_detected)  #Move forward looking for a wall
or lione
                        if (line_detected):                     #If the top entrance to
room 4 was found
                            Align_on_line()                     #Align on the line
                            line_detected = False               #Reset the line sensors
                            Enter_and_exit_room()               #Realign robot at the door
                        else:
                            Right_wall_follow_until_door(inside_room, hall, dog_config)
#Follow until door
                            Align_on_line()                     #Align on the line
                            Enter_and_exit_room()               #Realign robot at the door
                else:
                    Transition_forward(TRANS*3, SPEED)      #Transition forward far enough
to go off line
                    if(starting_room == 4):                 #If robot started in room 4
                        Turn_right(90,TURNSPEED)            #Turn right to avoid driving
into top wall
                    line_detected, dog_config = Right_wall_follow_until_door(inside_room,
hall, dog_config)  #Follow until the next room
                    if (line_detected):                     #If the entrance to room 4 was
found
                        Align_on_line()                     #Align on the line
                        line_detected = False               #Reset the line sensors
                        inside_room = True                  #Tell the bot it's outside the
room
                        room = 4                            #Set which room bot is in
```

```
                    Search_room(room, fire_extinguished, inside_room, hall,
dog_config)     #Search the room the bot is in
                    room_checked[room - 1] = True       #Mark off this room as checked
for the candle
                    inside_room = False                 #Mark the bot is back in the
hall
                    hall = 6                            #Set the hall to 6
                    entrance = 4                        #Set the entrance to be the
lower entrance of room 4
                    position = 6                        #Set the new position
        if (position == 6):                             #If the robot is at the bottom
of room 4
            Transition_forward(TRANS,SPEED)
            Move_forward_while_checking(30,SPEED,line_detected)  #Move forward looking
for a wall
            Move_forward_until_dist(WALLDIST,SPEED/2)   #Make sure the robot is still
WALLDIST away
            Turn_until_aligned()                        #Turn until aligned with the
wall
            Move_forward_until_dist(WALLDIST,SPEED/2)   #Make sure the robot is still
WALLDIST away
            Turn_right(100,TURNSPEED)                   #Turn  right 100 degrees to
face the center of the intersection
            Transition_forward(TRANS,SPEED)             #Transition forward slowly
            Move_forward(11,SPEED)                      #Move forward 11 cm
            Turn_right(80,TURNSPEED)                    #Turn down hallway 8
            Transition_forward(TRANS,SPEED)             #Transition forward slowly
            Move_forward(6,SPEED)                       #Move forward to make sure the
right wall of room 4 is there
            Right_wall_follow_for_distance(30)          #Follow up this wall for 30 cm
            Turn_left(90,TURNSPEED)                     #Turn left to face room 1
            Move_forward_until_dist(WALLDIST,SPEED/2)   #Make sure the robot is still
WALLDIST away
            Turn_until_aligned()                        #Turn until aligned with the
wall
            Move_forward_until_dist(WALLDIST,SPEED/2)   #Make sure the robot is still
WALLDIST away
            Turn_left(90,TURNSPEED)                     #Turn left to put the wall on
the right side of the robot
            line_detected, dog_config = Right_wall_follow_until_door(inside_room,
hall, dog_config)  #Follow until the next room
            if (line_detected):                         #If the  entrance to room 1 was
found
                Align_on_line()                         #Align on the line
                line_detected = False                   #Reset the line sensors
                inside_room = True                      #Tell the bot it's outside the
room
                room = 1                                #Set which room bot is in
                Search_room(room, fire_extinguished, inside_room, hall, dog_config)
#Search the room the bot is in
                room_checked[room - 1] = True       #Mark off this room as checked for
the candle
                inside_room = False                 #Mark the bot is back in the hall
                hall = 1                            #Set the hall to 1
                position = 1                        #Set the new position

        if (inside_room):    #If there is an error and the robot is stuck in a room
            room, entrance, room3_config, room4_config =
Navigate_out_of_room(inside_room, hall, room3_config, room4_config, dog_config) #Get
out of the room
            room_checked[room - 1] = True         #Mark off this room as checked for
the candle
            inside_room = False                   #Mark bot is outside of a room
```

```
1385                hall = Determine_hall(room, entrance) #Decide which hall section the robot
1386    is in
1387                Determine_position(room, hall, room3_config, room4_config, entrance)
1388    #Determine where in the maze the bot is
1389
1390        Transition_backward(TRANS,SPEED)  #Once returned to starting room, back into room
1391    a little
1392        Move_backward(3,SPEED)
```

VII. References

[1] Trinity College Robot Contest. (2018). *Rules*. [online] Available at: http://www.trinityrobotcontest.org/rules.html [Accessed 2 Apr. 2018].

[2] Encoder, R. (2018). *Robot DC Gearhead Motor - 6v 180rpm w/ Encoder*. [online] Trossenrobotics.com. Available at: http://www.trossenrobotics.com/p/Robot-DC-Gearhead-Motor-6v180rpm.aspx [Accessed 2 Apr. 2018].

[3] Robotshop.com. (2018). *Cytron 10A 5-25V Dual Channel DC Motor Driver*. [online] Available at: https://www.robotshop.com/en/cytron-10a-5-25v-dual-channel-dc-motor-driver.html [Accessed 2 Apr. 2018].

[4] GP2Y0A02YK0F, I. (2018). *Infrared Proximity Sensor Long Range - Sharp GP2Y0A02YK0F - SEN-08958 - SparkFun Electronics*. [online] Sparkfun.com. Available at: https://www.sparkfun.com/products/8958 [Accessed 2 Apr. 2018].

[5] Semiconductors, V. (2018). *TCRT5000 Vishay Semiconductors | Mouser*. [online] Mouser Electronics. Available at: https://www.mouser.com/ProductDetail/Vishay/TCRT5000/?qs=glpcD2KT6uaaYldHGIIt5g%3D%3D&gclid=EAIaIQobChMI-qaUwM2A2gIVDLjACh03GQEfEAQYAiABEgLQPPD_BwE [Accessed 2 Apr. 2018].

[6] Active Robots. (2018). *UVTron Flame Detector Package*. [online] Available at: https://www.active-robots.com/uvtron-flame-detector-package.html [Accessed 2 Apr. 2018].

[7] Tower Hobbies. (2018). *Futaba S3003 Standard Servo*. [online] Available at: https://www.towerhobbies.com/cgi-bin/wti0001p?&I=LXH288 [Accessed 2 Apr. 2018].

[8] Versa Home. (2018). *EZ Series Aluminum - Versa Home*. [online] Available at: https://www.versa-valves.com/portfolio/ez-series-aluminum/ [Accessed 2 Apr. 2018].

[9] Python, U. (2018). *Using a raspberry pi with a microphone to hear an audio alarm using FFT in python - benchodroff.com*. [online] benchodroff.com. Available at: https://benchodroff.com/2017/02/18/using-a-raspberry-pi-with-a-microphone-to-hear-an-audio-alarm-using-fft-in-python/ [Accessed 2 Apr. 2018].