# Helicopter Emergency and Rescue Operations (HERO) - FireBot Arduino Project

Christian M. Schrader[1]

*Worcester Polytechnic Institute, Worcester, Massachusetts, 01609, United States*

Mary Radke[2]

*University of Nebraska - Lincoln, Lincoln, Nebraska, 68588, United States*

Saanjali Maharaj[3]

*University of Toronto, Toronto, Ontario, M5S 1A1, Canada*

[1] Aeromechanics Intern, NASA Ames Research Center, Massachusetts Space Grant Consortium
[2] Aeromechanics Intern, NASA Ames Research Center, Nebraska Space Grant Consortium
[3] Aeromechanics Intern, NASA Ames Research Center, International Internship Program

## Acknowledgments

C. M. Schrader thanks the Massachusetts Space Grant Consortium and M. Radke thanks the Nebraska Space Grant Consortium for funding this research. S. Maharaj thanks the NASA International Internship Program and NIHERST Trinidad and Tobago for facilitating her participation on this project. Special thanks to Dr. William Warmbrodt for this internship opportunity, and to Shannah Withrow and Lee Kohlman for their invaluable mentorship. Thank you to the HERO FireBot Design Team – Hima Patel, Alicea Hyland, and Devony Whiting – for collaborating with us to create the concept, and the HERO DemoBot Team – Kaitlyn Eckhart, Reid Smith, Quentin Frederick, and David Bombara – for sharing ideas and advice. We wish to acknowledge Harmit Chahal for her work on the base station.

## Abstract

**The HERO FireBot Project is an initiative by NASA Ames Research Center that strives to assist firefighting professionals in controlling wildland fires. FireBot aims to eliminate spot fires to maintain the integrity of a fireline, allowing firefighting professionals to focus on other sections of the ongoing fire. The goal of this project is to develop a terrestrial robot which will eventually be adapted to a final drone version, capable of extinguishing such fires. FireBot will operate from a base station, where it prepares for deployment. Once given the coordinates of a spot fire from either an operator or a detection system, FireBot will navigate to the fire's general location. After pinpointing the precise location of the spot fire using an infrared camera, FireBot will deploy its fire suppressant payload before returning to base where it can be resupplied for another sortie.**

## I. Contents

## II. List of Figures

## III. List of Tables

## IV. List of Equations

## V. Nomenclature

| | |
|---|---|
| 4WD | Four-wheel Drive |
| $A$ | Surface Area of Region to be Insulated |
| $D$ | Distance |
| GPS | Global Positioning System |
| GSM | Global System for Mobile communication |
| GUI | Graphical User Interface |
| $h$ | Convective Heat Transfer Coefficient [W/m$^2$ K] |
| $H$ | Heading [°] |
| HERO | Helicopter Emergency and Rescue Operations |
| IO | Input and Output |

| | |
|---|---|
| IR | Infrared |
| JSON | JavaScript Object Notation |
| $k$ | Thermal Conductivity [W/m K] |
| LED | Light Emitting Diode |
| LiPo | Lithium Polymer |
| LoRa | Long Range |
| $R_{cond}$ | Conductive Thermal Resistance [K/W] |
| $R_{conv}$ | Convective Thermal Resistance [K/W] |
| $R_{rad}$ | Radiative Thermal Resistance [K/W] |
| $R_{tot}$ | Total Thermal Resistance [K/W] |
| $t$ | Thickness of Insulating Material [m] |
| $x_b$ | $x$ coordinate of the base station |
| $x_t$ | $x$ coordinate of the target |
| $y_b$ | $y$ coordinate of the base station |
| $y_t$ | $y$ coordinate of the target |

## VI. Introduction

Wildland fires were responsible for $5.1 billion in losses in the U.S. from 2007 to 2017 [1] and an average of 3.5 thousand deaths per year in the same time frame [2]. Emergency responders work to contain wildland fires by creating regions of space without combustible material, called firelines. Wind can cause hot embers to drift over firelines, igniting small spot fires outside the contained area. If spot fires are not extinguished, they expand and compromise the fireline. The FireBot Arduino Project strives to improve technology so that eventually a readily-available drone is used by firefighting professionals in wildland fires. It is hoped that this project will lead to a reduction of lives and dollars lost during wildland fire fighting efforts.

This project is dedicated to the service and sacrifice of fallen firefighters who have given their lives ensuring the safety of the communities they serve.

## VII. Mission

FireBot is designed to operate from a base station which serves to maintain operational status. Specifically, the base station must include a computer and antenna to communicate with the drone, electricity to recharge it, and fire suppressant to resupply it. The GPS coordinates of a spot fire are identified and given to FireBot. Next, FireBot exits standby mode, initializes its systems, and flies to the coordinates. There, it identifies the fire with an IR (infrared) camera and positions itself next to the fire to deploy its fire-suppressing payload. After this, or if at any point FireBot runs low on battery or is in danger of being destroyed, it returns to the base station to resupply and prepare for its next sortie.

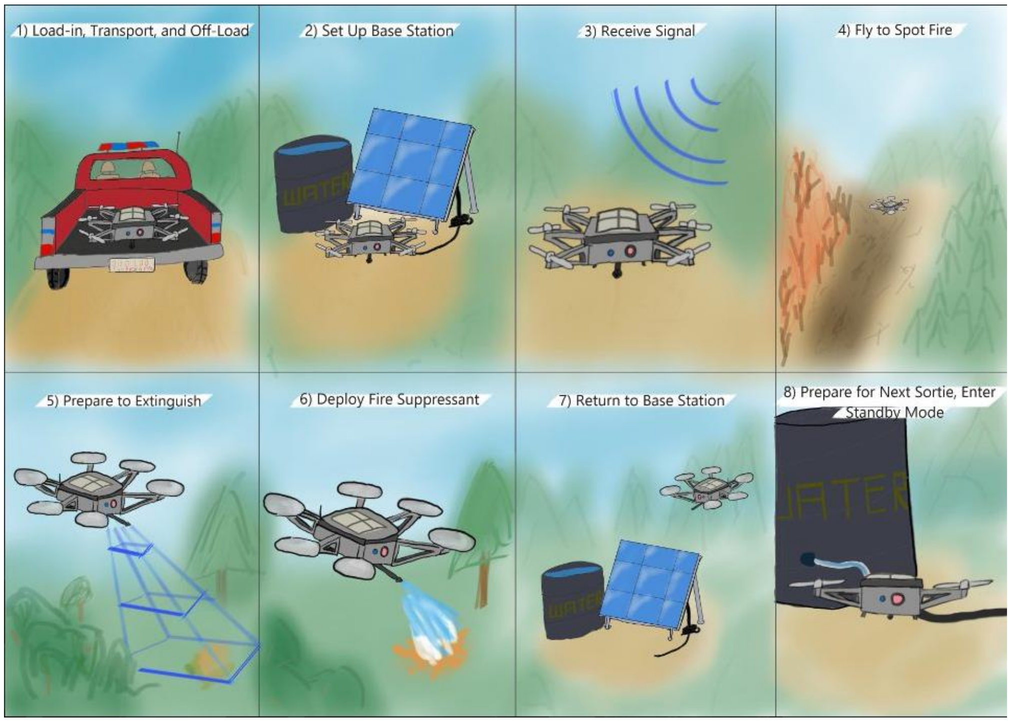
1) Load-in, Transport, and Off-Load
2) Set Up Base Station
WATER
3) Receive Signal
4) Fly to Spot Fire
5) Prepare to Extinguish
6) Deploy Fire Suppressant
7) Return to Base Station
WATER
8) Prepare for Next Sortie, Enter Standby Mode
WATER

*Figure 1 Concept of Operations*

This project focused on the design and construction of a terrestrial proof of concept for a spot-firefighting drone. This has allowed the team to work on demonstrating the feasibility of the system at a low cost and without the need for flight clearance. This initial version of FireBot uses a drive system with four omni-wheels. It operates from a similar base station. Once coordinates are received as it activates, the robot exits standby mode and initializes its systems before driving to the coordinates while avoiding obstacles. It identifies a simulated fire and positions itself next to the heat source and deploys suppressant before returning to the base station.

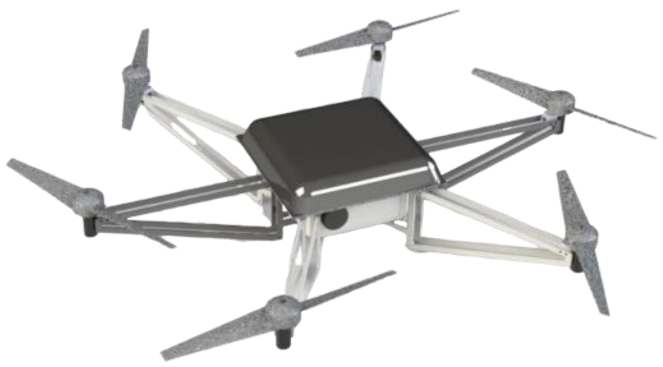

*Figure 2 FireBot Conceptual Model [3]*

A conceptual design of the drone version of FireBot was done by the FireBot Design team. The design uses a hexacopter configuration with a fiberglass airframe [3]. This configuration helps the design meet the payload requirements necessary to carry the suppressant system used on the proof of concept robot. The Design team has run many simulations to identify the aerodynamic properties of the vehicle to ensure that it will perform as intended.

## VIII. Hardware

### A. Chassis

FireBot is built using a 4WD 58mm Omni Wheel Arduino Robot [4]. Using four independently controlled Omni-wheels, it can move left and right in addition to forward and backward. This allows it to simulate drone movement. While it cannot emulate pitch, roll, and heave, the chassis provides a platform that can be transitioned to a drone with minimal complications. The chassis has its own Arduino which is controlled by sending commands via $I^2C$ from the primary Arduino. The wiring has been slightly changed from the factory standard which has one of the motor encoders connected to the analog 4 and analog 5 pins. Because these pins are required for $I^2C$, the encoder was switched to pins digital 6 and digital 13.

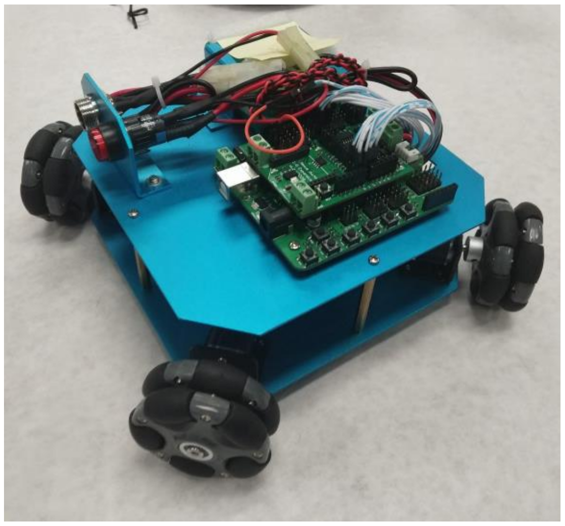

*Figure 3 Robot Chassis*

## B. Sensors

After being dispatched, FireBot must use an array of sensors to safely navigate to its target coordinates. For most of the travel, the robot relies primarily on a BNO055 accelerometer. To avoid objects en route, an HC – SR04 ultrasonic time-of-flight range finder is used. It detects the presence of obstructions in the path allowing FireBot to turn and reroute itself around them.

Once at its target coordinates, FireBot identifies the exact location of a fire using an IR sensor array, specifically an AMG8833 Grid-EYE. Once the fire is located, the Grid-EYE allows the robot to position itself to effectively deploy suppressant. To confirm it is near the fire, the robot is equipped with a flame sensor. This is a photoresistor with a filter that reduces noise caused by light from other sources.

## C. Communications

To communicate with the base station, the robot utilizes an HC-06 Bluetooth serial module. This was selected due to the ease of use of Bluetooth. It communicates with the primary Arduino using a serial connection that only uses two IO pins. Bluetooth does have the downside that it is only suited to short range communications, however, because the data being sent is unrelated to the communication standard it is sent over, it can easily be swapped out for another wireless standard in future versions. To interface with the robot, the only requirement for the base station is that it has Bluetooth connectivity.

| Chassis Arduino | |
|---|---|
| **Pin** | **Connection** |
| 0 | N/A |
| 1 | N/A |
| 2 | Motor 1 |
| 3 | Motor 1 |
| 4 | Motor 1 |
| 5 | Motor 1 |
| 6 | Motor 4 |
| 7 | Motor 4 |
| 8 | Motor 3 |
| 9 | Motor 3 |
| 10 | Motor 4 |
| 11 | Motor 2 |
| 12 | Motor 2 |
| 13 | Motor 4 |
| 14 | Motor 2 |
| 15 | Motor 2 |
| 16 | Motor 3 |
| 17 | Motor 3 |
| A4 | $I^2C$ SDA |
| A5 | $I^2C$ SCL |

| Primary Arduino | |
|---|---|
| **Pin** | **Connection** |
| 0 | Hardware Serial |
| 1 | Hardware Serial |
| 2 | HCSR04 Trigger |
| 3 | HCSR04 Echo |
| 4 | N/A |
| 5 | N/A |
| 6 | N/A |
| 7 | N/A |
| 8 | N/A |
| 9 | N/A |
| 10 | Bluetooth Receive |
| 11 | Bluetooth Transmit |
| 12 | Payload Trigger |
| 13 | N/A |
| A4 | $I^2C$ SDA |
| A5 | $I^2C$ SCL |

*Table 1 Arduino Pin Layout*

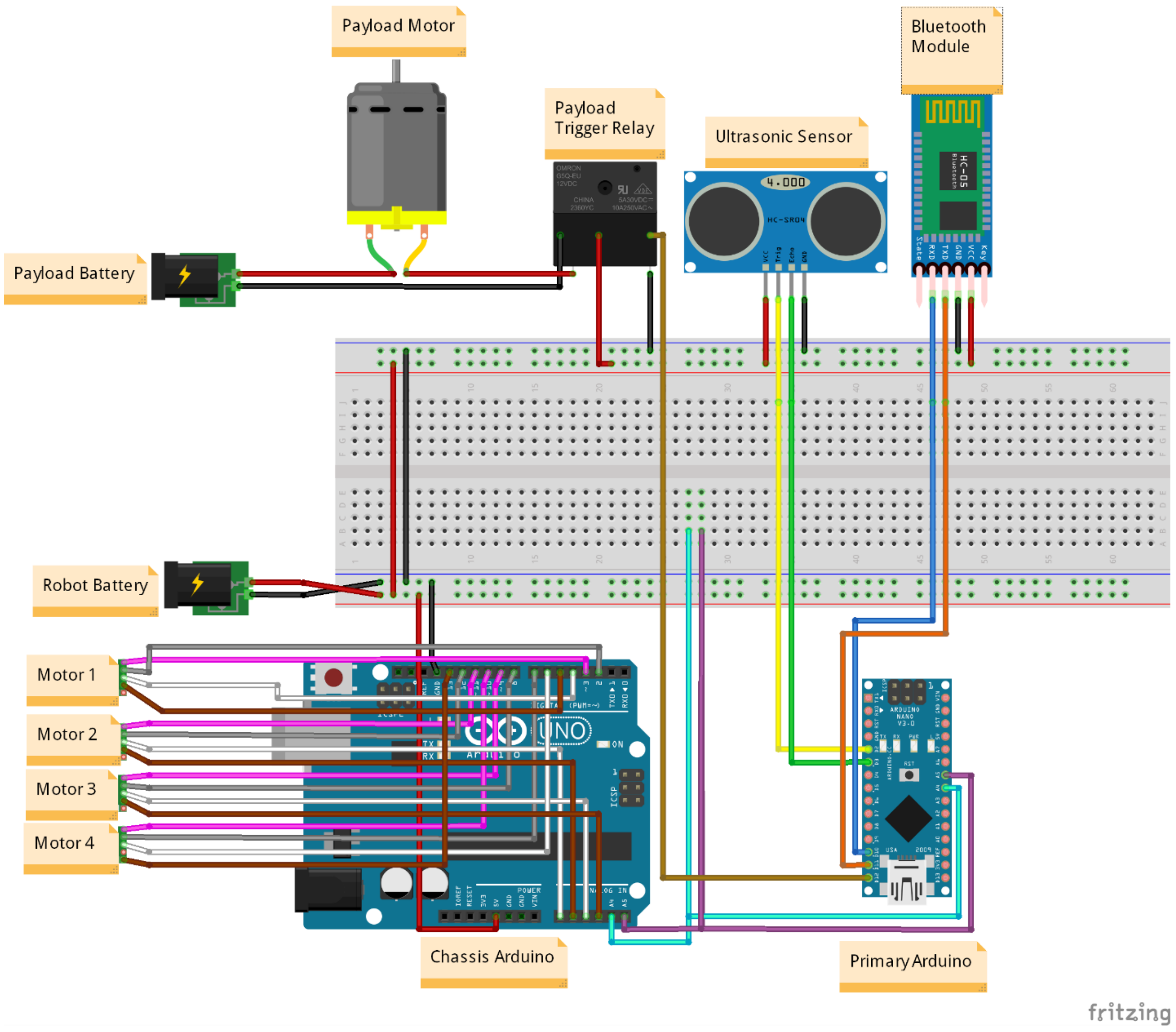

*Figure 4 Wiring Diagram*

## IX. Software

### A. Program Structure

FireBot's decisions are managed by a state machine. Based on the state, different actions are taken along with different criteria that are checked to determine when to switch states. The states are as follows.

I. STANDBY: The robot does nothing. It waits until it receives target coordinates before switching to INIT.
II. INIT: All sensors are initialized in preparation for the sortie. Once all checks are complete, the robot switches to TAKE_OFF.
III. TAKE_OFF: The robot instantly switches to CRUISE. This state is left as an empty place holder that, while not needed on the initial version of FireBot, will be important in the eventual drone version. When properly implemented, it should cause the drone to ascend to its cruise altitude before switching to CRUISE.
IV. CRUISE: The robot navigates to its target coordinates. It switches to SEARCH once it arrives.
V. SEARCH: The switch from navigation by coordinates to navigation by IR is made. The robot searches for a heat source and approaches it. Once close enough, it switches to SUPPRESS.

VI. SUPPRESS: The suppression payload is activated. After the time set to deploy, it switches to RECALL.
VII. RECALL: The robot navigates back to its base station. Once it arrives, it switches to STANDBY.

Within the structure of the state machine, most of the actions of the robot involve interacting with sensors and the chassis. Every component on the robot is represented by a class. These classes are then organized into .ino files. At compile time, every file is combined to one large file in order. All sensors have update(), connect() and disconnect() functions. The files and their classes are as follows.

I. FireBot.ino (Appendix 0): This file contains only initialization operations. This consists primarily of instantiating constants and arrays.
II. 1_Chassis.ino (Appendix C): A file containing classes needed to utilize the drive system.
    a. Chassis: This class is a base class intended to be inherited by other movement classes.
    b. OmniDrive: A class used to drive the current omni-wheel robot.
    c. Hexacopter: An unimplemented class intended to control a hexacopter chassis.
III. 2_Range.ino (Appendix D): A file containing classes related to running range finder sensors.
    a. RangeFinder: The general base class for other range finders to inherit from.
    b. HCSR04: A subclass of RangeFinder for interacting with an HCSR04 sensor.
IV. 3_Thermistor.ino (Appendix E): A file containing classes for operating thermistors. It is unimplemented.
    a. Thermistor: A base class for thermistors to inherit from.
    b. TMP36: A class intended to drive a TMP36 sensor. The team did not receive one so this class is unimplimente3d.
V. 4_Navigation.ino (Appendix F): A file with classes for controlling navigation sensors and navigating to coordinates.
    a. NAV: A base class for navigation devices. It contains functions for coordinate navigation math.
    b. Accelerometer: A class for using accelerometer only navigation. Coordinates are tracked internally with the only input being heading from the BNO055 accelerometer.
VI. 5_IR.ino (Appendix G): This file has classes that control IR sensors and navigation by IR.
    a. IR: A general base class intended to be inherited by other IR classes.
    b. GridEye: A class used to drive the Grid-EYE. Its update() function takes a new image and, based on the image, makes a new decision about which movement to take to track the heat source.
VII. 6_Suppression (Appendix H): This file is contains classes used to operate the payload.
    a. Suppression: A class for deploying the suppressant payload. The implementation of this class is likely to change depending on what method is used to activate the payload.
VIII. comms.ino (Appendix I): This file contains one function for reading Bluetooth and one for writing to Bluetooth. These will likely have to be modified depending on how commands are made to be received and what telemetry is to be sent.
IX. main.ino (Appendix J): This file contains the main decision making code. It instantiates objects, completes setup, and has the main loop that contains the state machine.

## B. Navigation

When given target coordinates, such as the fire or the base station, FireBot navigates based on an internal coordinate system. In the INIT state, the navigation system is initialized. It records the position it is in as (0, 0) with a heading of 0°. The heading and travel distance necessary to reach its target coordinates are calculated.

$$H = \tan^{-1}\frac{x_t - x}{y_t - y}$$

*Equation 1 Heading*

$$D = \sqrt{(x_t - x)^2 + (y_t - y)^2}$$

*Equation 2 Distance to Target*

FireBot then turns to heading H and travels the distance D. During this movement, the accelerometer is periodically checked to ensure FireBot is still on the correct heading, making corrections as necessary. It also checks the ultrasonic sensor to make sure there are no obstructions in its path. If there are, it turns 90° and attempts to get around it. After making this movement, the navigation system marks the bot as having arrived at the target coordinates.

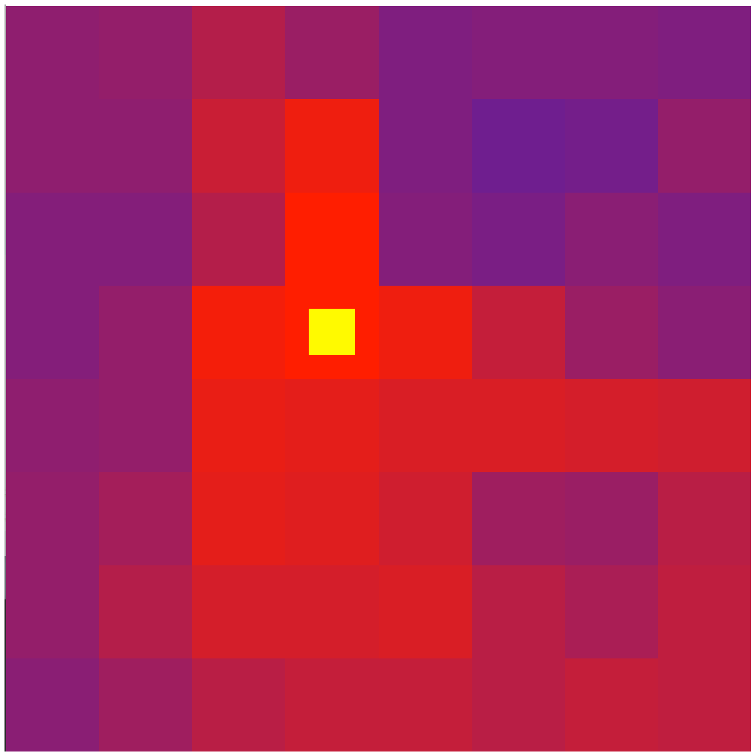

*Figure 5 Image captured using the Grid-EYE sensor array*

To do final positioning, the Grid-EYE's image is analyzed and the pixel with the highest temperature is identified. In Figure 5, the highest temperature pixel is marked with a yellow square. Based on its location in the image, FireBot rotates either left or right to move the high temperature location into the center of its field of view. Because the Grid-EYE is not linked to the coordinate navigation system, movements made while in the SEARCH state are not recorded. This leads to an offset of the base station equal to the displacement of the robot while it is in SEARCH.

### C. Communications

To communicate with FireBot, the base station initializes a Bluetooth connection between itself and the HC-06 module on the primary Arduino. This acts as a wireless serial port. As FireBot gathers data from its sensors, it creates a packet formatted in JavaScript Object Notation (JSON). The JSON packet is then sent through the serial connection to the base station.

The base station itself runs a program written in Java. It accesses the Bluetooth serial connection to receive the packets transmitted from the robot. These packets are then interpreted as objects and the values are displayed to the operator on a graphical user interface (GUI). In real time, the GUI displays whether or not FireBot is connected, percentage of remaining battery capacity, temperature of the robot, payload status, and coordinates.

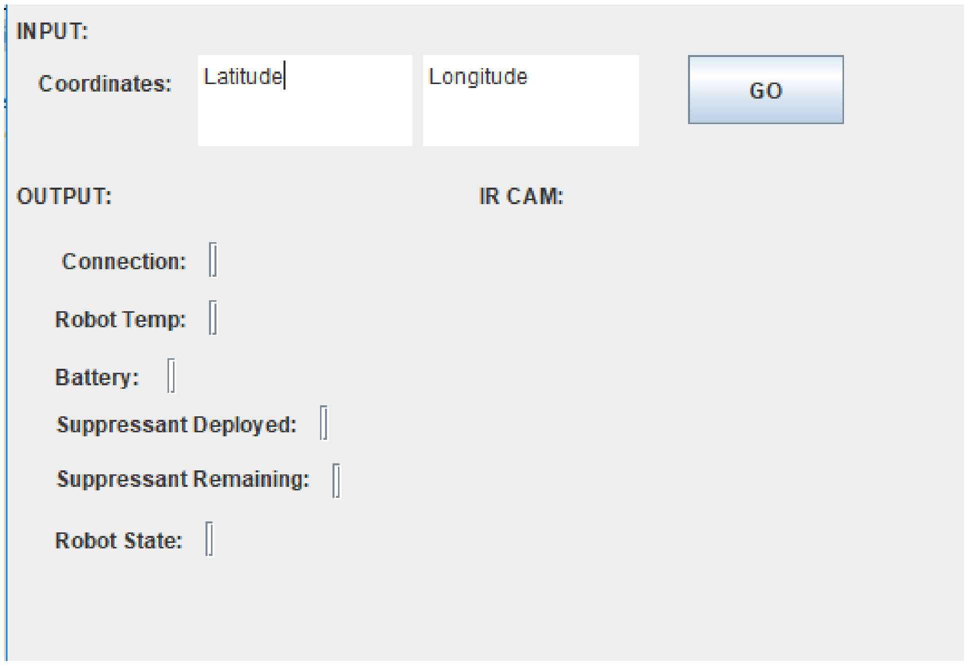

*Figure 6 Base Station GUI*

## X. Thermal Analysis

For an average wildfire, the maximum temperature reached is 800°C (1073K) [5]. However, for extreme wildfires, the maximum temperature can be up to 1200°C (1473K) [5]. Ideally, FireBot should be able to operate in the latter maximum temperature. In particular, the components most sensitive to extreme temperatures are the electronics, especially the battery.

In order to withstand the extreme temperatures it will be subject to in a wildfire, FireBot requires sufficient insulation. Due to the small size of the terrestrial proof of concept, a passive cooling system was chosen instead of an active system to meet weight constraints. The insulation material selected for the proof of concept robot was ceramic paper. The thickness required was determined using the following theoretical calculations.

### A. Material Properties

$Surface\ Area\ of\ Region\ to\ be\ Insulated\ (A) = 1183\ cm^2$

$Thermal\ Conductivity\ of\ Insulating\ Material\ (k) = 0.2\ W/m^2\ K$ [6]

The thickness of insulating material was selected to be 1/8in and calculations were continued to determine whether this thickness is sufficient.

$Thickness\ of\ Insulating\ Material\ (t) = 0.3175\ cm$

### B. Temperatures

$Maximum\ Temperature\ of\ a\ Wildfire = 1473\ K$

$Maximum\ Operating\ Temperature\ of\ Electronic\ Components = 313\ K$

$Temperature\ Difference = 1473\ K - 313\ K = 1160\ K$

$Maximum\ Heat\ Flux\ of\ Wildfire = 454000\ W/m^2$ [7]

The heat transfer problem will be modeled as a thermal resistance circuit, with convective ($R_{conv}$) and radiative ($R_{rad}$) resistances from the air between the wildfire and FireBot in parallel with each other, in series with the conductive resistance ($R_{cond}$) from the insulating material.

$$R_{tot} = 1 \Big/ \left(\frac{1}{R_{conv}} + \frac{1}{R_{rad}}\right)$$

*Equation 3 Total (Effective) Resistance*

$$R_{cond} = \frac{t}{kA} = \frac{0.003175\ m}{0.2 \frac{W}{m^2K} * 0.00001183\ m^2} = 0.134 \frac{K}{W}$$

*Equation 4 Conductive Resistance*

$$R_{conv} = \frac{1}{hA} = \frac{1}{120 \frac{W}{m^2K} * 0.00001183\ m^2} = 0.00256 \frac{K}{W}$$

*Equation 5 Convective Resistance*

$$R_{rad} = R_{conv}(R_{tot} - R_{cond}) \Big/ (R_{cond} + R_{conv} - R_{tot}) = -0.0456\ K/W$$

*Equation 6 Radiative Resistance*

$$Total\ Resistance\ (R_{tot}) = \frac{Maximum\ Heat\ Flux}{Temperature\ Difference} = \frac{454000\ W/m^2}{1160\ K} = 0.00256\ K/W$$

*Equation 7 Total Resistance*

The total resistance represents the thermal resistance required to keep the electronics safe at 313K from a 1473K fire. It can be concluded that, regardless of the convective and radiative resistance in the space between the wildfire and FireBot, the electronics will be maintained at a safe temperature since the conductive resistance provided by the insulation exceeds the required total thermal resistance.

From the calculations above, a thickness of 1/8in should be more than sufficient to withstand the maximum temperatures of extreme cases of wildfires. Hence, the material was ordered in this thickness and physical testing was implemented as explained in section XII.

## XI. Fire Suppression

A number of suppressant mechanisms were researched and three selected to test their effectiveness. The first mechanism investigated is based on what helicopters typically use to help suppress fire from the air called the Bambi Bucket. Essentially, a Bambi Bucket is a bucket that drops water all at once so that the majority of it makes it to the fire and does not entirely evaporate as it nears the heat. For a drone-sized application the bucket needs to be scaled down, resulting in the name Baby Bambi Bucket. The Bambi buckets can also be dipped into any body of water such as lakes or pools which makes refill convenient. Due to the nature of water drop method, the Baby Bambi Bucket would only have a single use at the fire before needing to return to water to refill. To do a basic test for this method a bucket was used to drop water by hand. If this idea was implemented in the future it would be converted to a canvas type material that could be dipped into a body of water nearby. Depending on the type and size of the water nearby, there may be complications with the refill ability. Other complications such as wind over a lake may create some problems for the drone flying above it. This makes the Baby Bambi Bucket less of a favorable alternative since there is a possibility the drone may fall in the water.

Another technique tested was based on edible water bottles (bubbles) and investigating whether water contained in a membrane allows the water to fully reach the base of the fire without evaporating. Since fire is best put out by smothering the base, the membrane may allow the bubbles to get closer before bursting. The edible water bubbles were developed using a reaction of calcium lactate and sodium alginate, which creates a membrane that incases the water. Since these ingredients are biodegradable, it is plausible that these could be dropped in wildland setting and not have adverse effects on the nature. A challenge with the water bubbles is that they need to remain cool in order to maintain their membrane type layer. This makes it difficult to transport them to and store them at the scene of a fire. This could be solved by creating them on scene, except currently the water bubbles are created by hand at a slow rate. At this time the manufacturability of these edible water bubbles is not very applicable for use but machines that create them in the future could make this idea plausible.

The third method is a sprayer device. The sprayer is advantageous due to its ability to target a fire more directly and move while spraying. For proof of concept a hand pump pressure sprayer was used to test the suppression ability. Pumping the sprayer builds pressure within the tank, which allows the sprayer to dispense water at 0.4-0.5 gallons per minute [gpm].

Types of fire suppressing foam were also researched since foam has proven to make water more effective at putting out fire [8]. The foam comes in concentrate form and small amounts can either be mixed into water initially, or can be added via a specific type of pressure nozzle that allows some to be added with the spray. Class A foam is the type of foam used in wildland fire fighting applications because it is safe for the environment. Essentially, the foam just helps the water stick to surfaces better instead of quickly evaporating away or sinking into the ground. The fire then cannot light the wet surfaces, ergo its suppressed. With the continuation of this project, adding a foam nozzle would be a fairly simple improvement that could be made.

## XII. Test Methods

### A. Insulation

To investigate the effectiveness of the insulation sheet a test was completed. A sheet of insulation was marked with a test grid which consisted of nine evenly spaced points spanning the dimensions of the chassis of FireBot. The temperature of the heat source, a bonfire, was allowed to stabilize. Water was kept on hand to extinguish the fire.

#### a. Procedure

I. Hold a sheet of insulation of the chosen thickness 4ft away from the heater. (This distance was determined to be the minimum distance between FireBot and the fire for suppression)
II. Measure the temperature of each point on the test grid using a non-contact IR thermometer.
III. Repeat the process using two then three layers of insulation

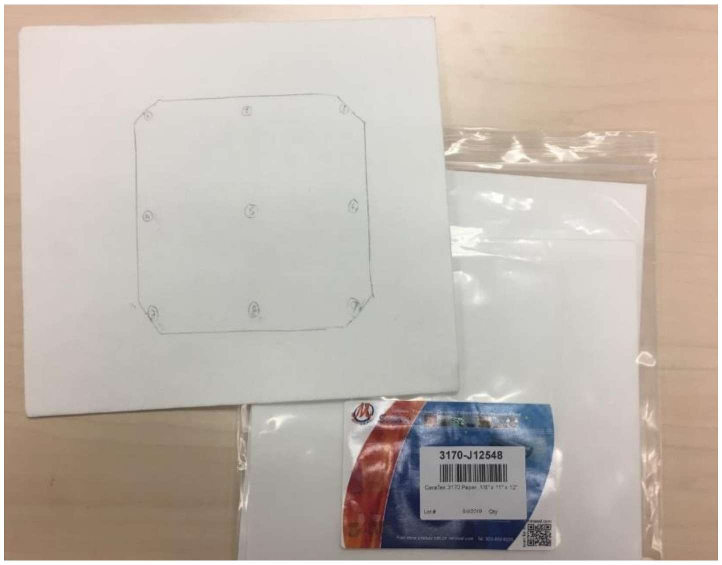

*Figure 7 Test grid on ceramic paper*

*Figure 8 Using non-contact IR thermometer on ceramic paper over heat source*

#### b. Evaluation

The results were used to calculate the maximum allowable temperature of wildfire that FireBot can withstand, using a comparison of temperature gradients. If this maximum allowable temperature is below the average wildfire temperature of 1073K, the test is considered a failure. Ideally, the maximum allowable temperature should exceed both the average and maximum (1473K) wildfire temperatures.

### B. Fire Suppression

To provide more data on which suppressant system would be most effective for the drone applications a test was completed. The test set up included three different fires of consistent amounts of fuel to maintain size. To simulate a spot fire fuel, small dried sticks in bundles of 3in by 10in were used for each fire. Each dispersion method was given half a cup of water to put out the fire, and held 2ft above the top of the fire. Three suppressant systems were tested: a sprayer, a bucket, and water bubbles. All tests were video recorded for reference.

#### a. Procedure

I. Set up the dispersion method is set up 2ft above the fire
II. Light the fire and allow a few seconds to light most of the fuel
III. Deploy (drop or spray, depending on the method being testing) the water accordingly
IV. Visually inspect the fire response

#### b. Evaluation

Success was defined based on whether or not fire was completely suppressed with only the half cup of water. The results of this test determined what mechanism would be used for the robot and future drone fire suppression applications.

### C. Demonstration One

This demonstration was intended to show the navigation abilities of the system. It consisted of fifteen simulated sorties. To setup, a starting point was marked that specified the initial location and orientation of FireBot. A waypoint was then marked. Finally, five heat source locations were marked. For this demonstration, a seated person was used as the heat source. The coordinates of each point were measured using the waypoint as the origin.

| **Point Locations** | | |
|---|---|---|
| **Point** | **x (in.)** | **y (in.)** |
| Waypoint | 0 | 0 |
| Base Station | -48 | -48 |
| Point 1 | 36 | 48 |
| Point 2 | -36 | 48 |
| Point 3 | 0 | 84 |
| Point 4 | 0 | 144 |
| Point 5 | 0 | 204 |

*Table 2 Point Locations for Demonstration One*

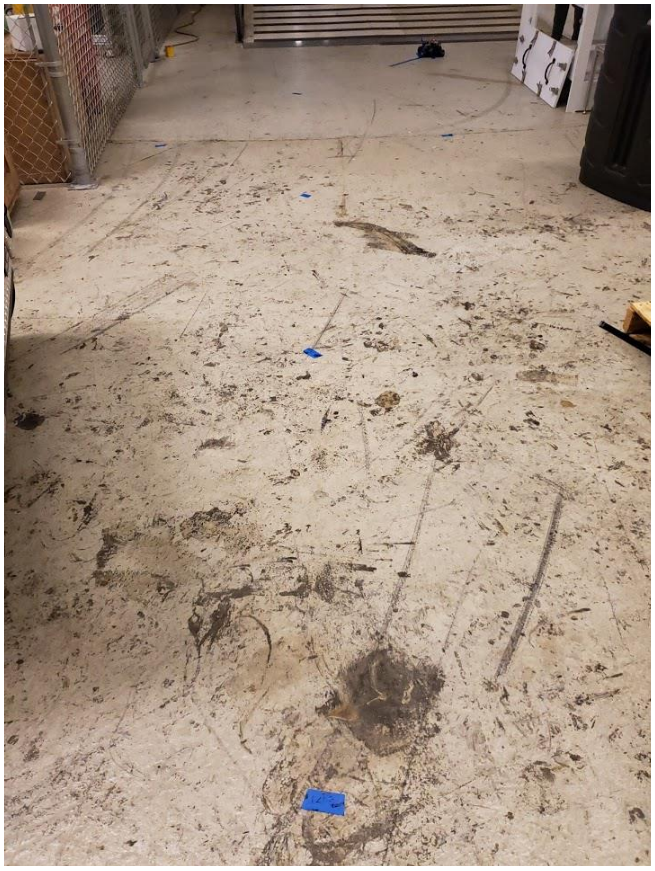

*Figure 9 View of the test area from point 5*

Figure 9 shows the test area used. All the marked points can be seen marked by blue tape and the robot itself can be seen at its base station in the back.

**a. Procedure**

I. Set the robot at its starting point and orientation
II. Turn on the robot.
III. Wait for the robot to run its task.
    a. Drive to the waypoint.
    b. Search and approach the heat source.
    c. Activate the status LED to indicate simulated suppressant deployment.
    d. Drive back to the base station.
    e. Come to a full stop.
IV. Turn off the robot.
V. Record its coordinates using the waypoint as the origin.

**b. Evaluation**

The test was considered successful if FireBot arrived at the location of the heat source, turned on its LED to signal that the suppressant was being deployed, and returned to the general area of the base station. A partial success occurred if FireBot arrived at the location of the heat source and turned on the LED to signal suppressant deployment, but did not return to the general area of the base station. It was considered a failure if FireBot did not arrive at the location of the heat source and turn on the LED.

### D. Demonstration Two

This demonstration was a final proof of concept of the system. It comprised six simulated sorties. To setup, a starting point was marked that specified the initial location and orientation of FireBot. A waypoint was then marked. Finally, one heat source location was marked. For this demonstration, a seated person was used as the heat source. While Demonstration One involved simply turning on an LED to signal the deployment of suppressant, Demonstration Two involves actually spraying water from the tank at the heat source. For the first three trials, the course was obstacle-free, but for the following three trials, an obstacle, a 27in by 16in bin was placed in the path between the initial position and the waypoint. Coordinates were measured using the waypoint as the origin.

| Point Locations | | |
|---|---|---|
| **Point** | **x (in.)** | **y (in.)** |
| Waypoint | 0 | 0 |
| Base Station | -32 | -37 |
| Point 1 | 0 | 38 |
| Obstacle | -8 | -14 |

*Table 3 Point Locations for Demonstration Two*

#### a. Procedure

I. Set the robot at its starting point and orientation
II. Turn on the robot.
III. Wait for the robot to run its task.
    I. Drive to the waypoint, attempting to avoid any obstacles present.
    II. Search and approach the heat source.
    III. Deploy suppressant by turning on sprayer directed at heat source
IV. Turn off the robot.
V. Record its coordinates using the waypoint as the origin.

#### b. Evaluation

The test was considered successful if FireBot arrived at the location of the heat source avoiding any obstacles in its path and deployed its suppressant on the heat source. A partial success occurred if FireBot arrived at the location of the heat source but missed the heat source when it attempted to deploy the suppressant. It was considered a failure if FireBot did not arrive at the heat source at all.

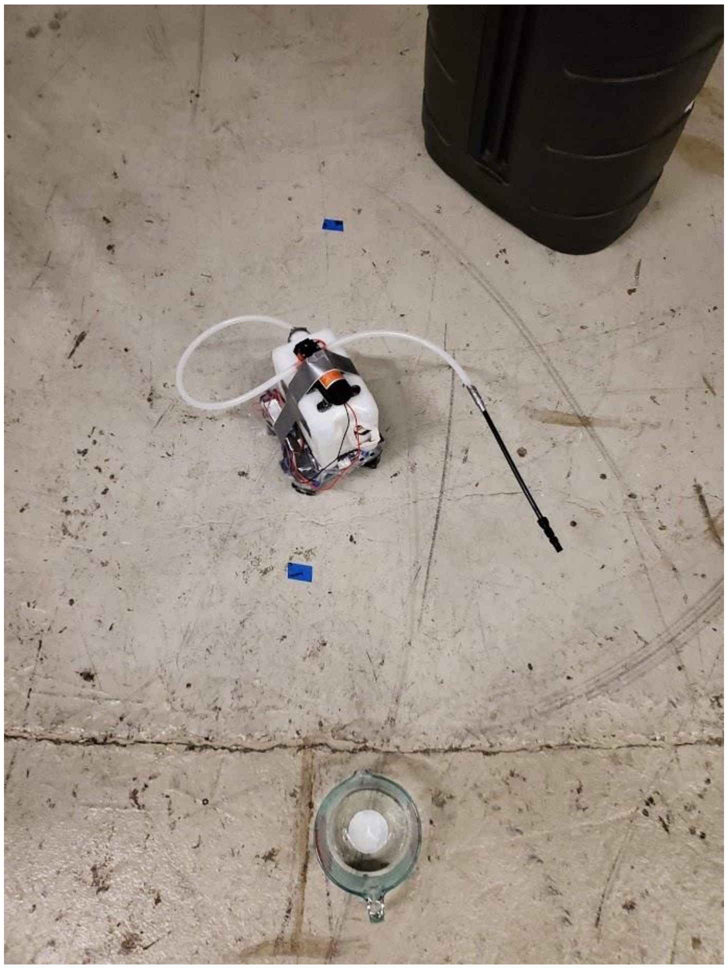

*Figure 10 FireBot in the test area*

## XIII. Results

### A. Insulation

The temperature of the bonfire was 938K and the distance between the bonfire and the insulation was 1.22m (4ft) during all 3 trials. The maximum grid point temperature was used in calculating the temperature gradient because the electronics must be protected at any position on FireBot. The temperature gradient was calculated using the following formula.

$$Temperature\ Gradient\ of\ Test\ (\nabla T_{test}) = \frac{bonfire\ temperature - maximum\ grid - point\ tempature}{distance\ between\ insulation\ and\ bonfire}$$

*Equation 8 Temperature Gradient of Test*

| | Temperature at Grid-point (°C) | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Number of Sheets** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **Average** | **Maximum** | **Temperature Gradient (°C /m)** |
| 1 | 24.0 | 24.2 | 23.9 | 23.7 | 22.6 | 22.3 | 22.5 | 22.7 | 22.6 | 23.2 | 24.2 | 525.59 |
| 2 | 16.4 | 16.6 | 16.8 | 16.3 | 16.2 | 16.3 | 16.0 | 16.1 | 16.2 | 16.3 | 16.8 | 531.66 |
| 3 | 16.2 | 16.0 | 15.9 | 15.8 | 15.7 | 15.6 | 15.5 | 15.6 | 15.5 | 15.8 | 16.2 | 532.15 |

*Table 4 Temperatures Measured for Insulation Testing*

The relationship between the temperature gradient of the test and the actual temperature gradient is as follows:

$$\frac{q'_{fire}}{\nabla T_{fire}} = \frac{q'_{test}}{\nabla T_{test}}$$

*Equation 9 Heat flux to temperature gradient ratio*

Where $q'_{fire}$ is 434,000 W/m2, as used previously, and the value of $q'_{\text{test}}$ was used as 121,278 W/m$^2$ [9].
Thus, the temperature gradient of the fire can be calculated by multiplying the temperature gradient test values by $\frac{q'_{fire}}{q'_{test}}$.

$$\nabla T_{fire} = \frac{q'_{fire}}{q'_{test}} * \nabla T_{test}$$

*Equation 10 Wildland fire temperature gradient*

Finally, the maximum wildfire temperature can be determined as follows, by setting the temperature of the electronics as 40°C (313K), their maximum operating temperature.

$$\nabla T_{fire} = \frac{maximum\ wildfire\ temperature - temperature\ of\ electronics}{distance\ between\ wildfire\ and\ FireBot}$$

$$\therefore\ maximum\ fire\ temperature = \left(\nabla T_{fire} * distance\ between\ fire\ and\ bot\right) + temp\ of\ electronics$$

*Equation 11 Maximum wildland fire temperature*

| **Number of Sheets** | **Maximum Wildfire Temperature (°C)** |
|---|---|
| 1 | 2438.81 |
| 2 | 2466.51 |
| 3 | 2468.75 |

*Table 5 Maximum Wildfire Temperature for Different Numbers of Sheets of Insulation to Protect Components*

It can therefore be concluded that one sheet of insulation is sufficient to ensure that the temperature of the electronics are kept in a safe operating range. The experimental results verify the theoretical conclusion.

## B. Fire Suppression

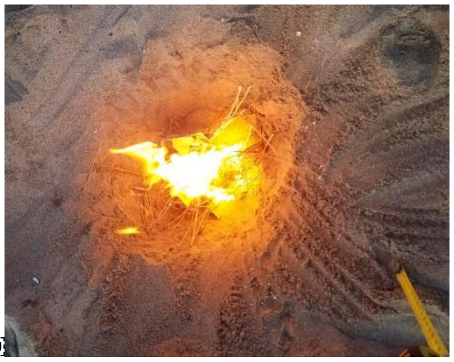

*Figure 11 Image of initial fire size prior to test*

For testing purposes a hand pressure sprayer was used to simulate the spraying effect of an electric pump sprayer. After spraying for only four seconds completely extinguished the fire. In addition to quickly dousing the fire, the sprayer method is also useful because it has the ability to continuously spray while changing angles to spread the water more effectively. In changing conditions such as gusts of wind, the ability to continue to spray while shifting position could become key in eliminating the fire. The sprayer method was most effective in eliminating the fire.

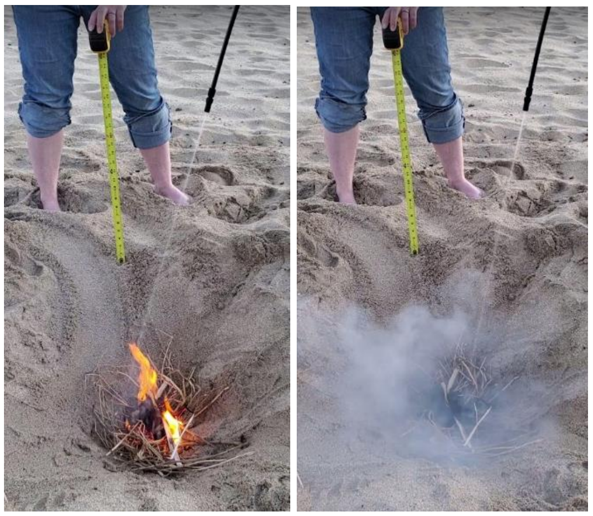

*Figure 12 Image of sprayer system in action, during (left), and after (right)*

To test the Baby Bambi Bucket method a container was used to drop the half cup of water on the fire. The fire was completely reduced on the locations where the water landed, but it only landed on some portions of the fire. Due to its inability to quickly refill and disperse again, the fire would most likely have spread during the lost time. It took an additional cup of water to fully put out the fire since it was hard to properly target a specific section of fire with a lone cup device. It was discussed that while this has some to do with human error, the drone dropping water would likely perform similarly in terms of targeting the fire in real conditions. Any wind causing the fire to shift quickly would create a condition where it would be difficult to put a quick stop to the fire if the drone missed with a single bucket drop.

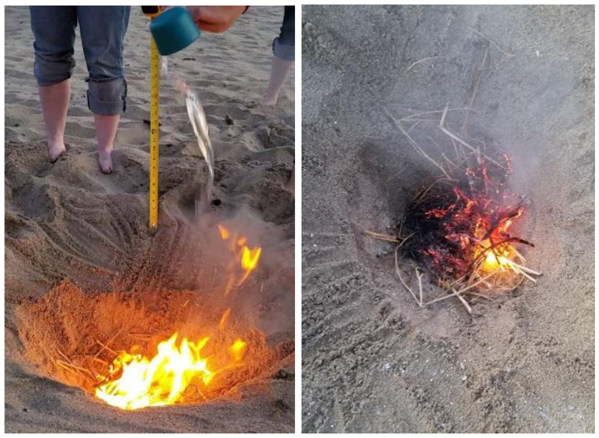

*Figure 13 Image of bucket (cup), during (left) and the residual fire (right)*

The final method tested was the water bubbles method. The bubbles were held and dropped over the fire one a time. It was interesting to test because the spots at which the water bubbles landed the fire was doused. This method was unique in that the water bubbles would land and then burst shortly after landing. The theory that the water bubbles would be useful in reaching the bottom of the fire without evaporation seems correct. Since, fire is best put out at the base this application could continue to be investigated in the future. Unfortunately, due to the size and amount of water bubbles, they were not big enough or there were not enough to put out the entire fire. More would need to be dropped to completely put out the fire. For the purpose of this test eight tablespoon sized water bubbles were used, which totals a half cup. If they were larger in size they would have covered a larger area. While effective where they landed, overall, the bubbles do not seem very practical when compared to the sprayer.

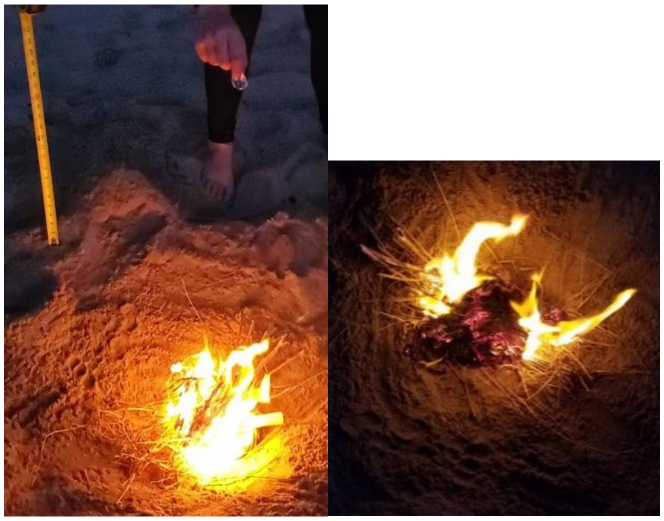

*Figure 14 Image of water bubble drop method, during (left) and the residual fire (right)*

The results of the fire suppression test were reviewed to verify which method of suppressant was best. A trade study was completed using criteria such as weight, ability to reach fire base, multiple drops, clog considerations, ease of refill, cost, and targetability (Appendix L). According to the trade study, the best method of fire suppression, out of the three methods tested, was the sprayer. This aligns with the team’s assessment after testing was carried out.

As a result of the success of the sprayer during tests, the project moved forward with a sprayer design. To be fitted with a drone in the future, the weight constraint for the sprayer system, including water, is 15lbs. Utilizing the lightest possible components, the resultant sprayer design involved a 1.3gal tank, which carries about 11lbs of water. The pump utilized is a SEAFLO 12-series water pressure diaphragm pump which sprays at 1.2gpm at 35psi. This allows the pump to discharge the tank in about 52 seconds. Using the amount of fuel on fire that the sprayer was able to put out from the test, 30 square inches of fire with 0.5cup of water, a rough calculation for the amount of fire the drone would ideally be able to douse was estimated to be approximately 8.67 square feet of fire. This number easily meets the criteria for putting out small spot fires jumping a fireline, meeting the project goal. These numbers are all rough approximates due to a number of factors that could affect the real life application such as fuel material, wind, and other environmental conditions., but the test served as a sufficient proof of concept.

$$Area\ of\ Fire\ to\ be\ Suppressed\ (ft^2) = \frac{30\ in^2}{.5\ cup} * \frac{16\ cup}{1\ gal} * \frac{1\ ft^2}{144\ in^2} * 1.3\ gal$$

*Equation 12 Area of Fire to be Suppressed*

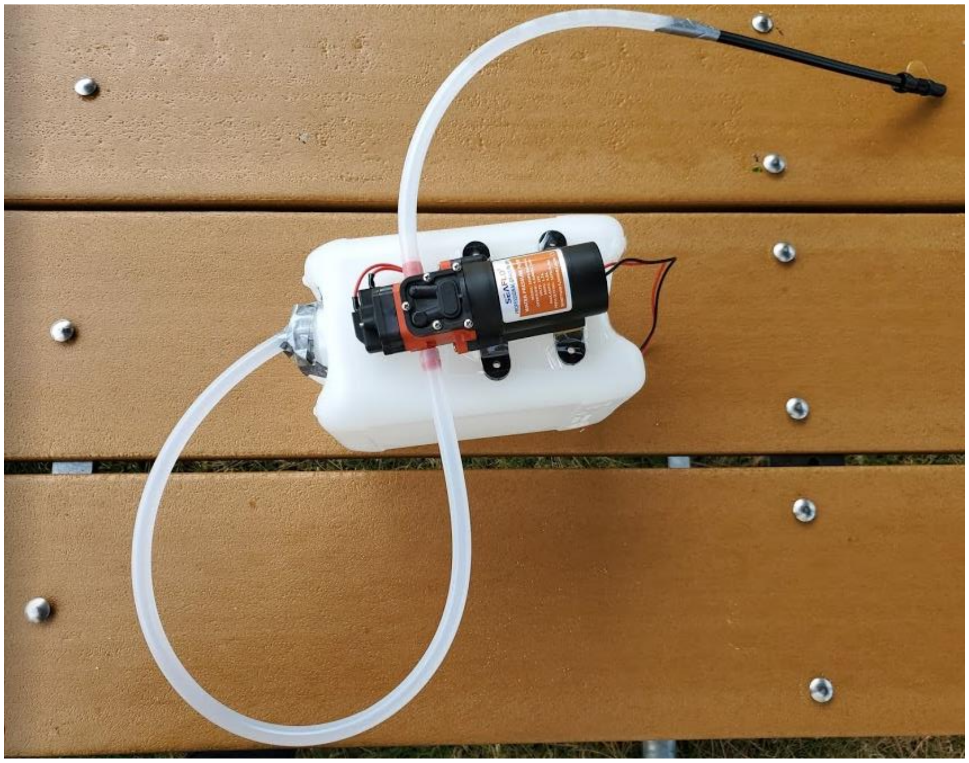

*Figure 15 Suppression System*

The battery size and type was determined based on the pump and weight constraints. The SEAFLO pump runs on 12 V (volts) and 4 A (amps). To meet the pump requirements, the battery selected was a 4S LiPo 45C, which operates at 14.8 V and holds 2200 mAh. LiPo batteries often used in drone applications are useful due to their lightweight nature. Based on what the pump draws and what the battery contains, an ideal total of 26 trips could be completed before recharging. However, it is recommended by the manufacturer that a LiPo battery is not completely drained, so this reduces the number of trips. Additionally, for future drone applications, the entire drone could run off of the same battery which would also reduce the number of sorties.

A majority of the weight contained in the suppressant system comes from the amount of water carried which uses about 11lbs of the allotted 15lbs. The remaining pump, battery, tubing, and mounting device could not exceed 4 lbs. The small SEAFLO pump was chosen because it only weighs 1.2lbs and the light weight LiPo battery only used 0.6lbs of weight. To also conserve weight a 3D printed mounting device was used to support the tank and sensors. The current design is fit for proof of concept using a small robot but continuation of this project would result in another 3D printed mounting device designed to attach to a drone. Since the mounting device, nozzle, and tubing are all plastic the weight is well under the 2lbs left over. The collective materials fit within the weight requirements for FireBot's suppressant system,  and as a result the objective is met.

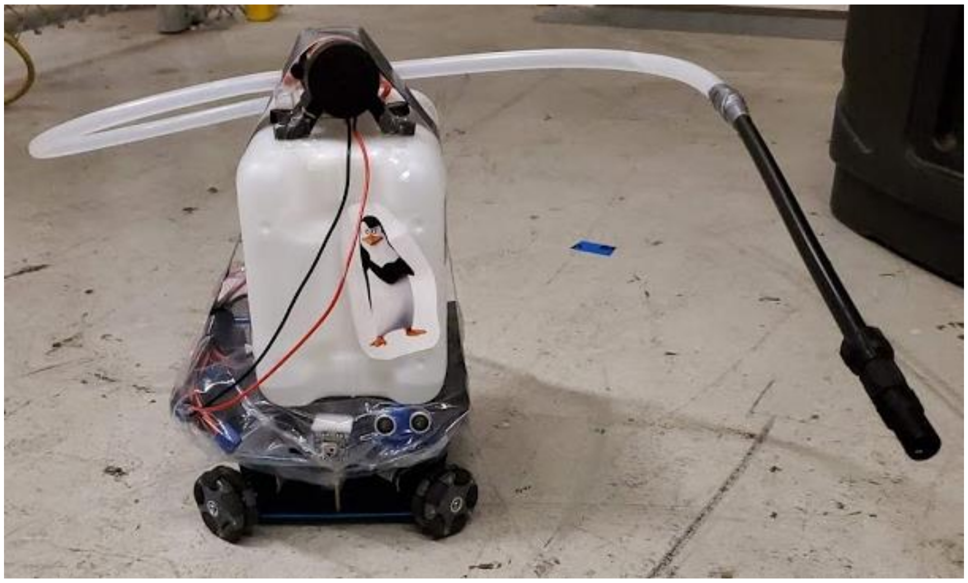

*Figure 16 FireBot with its suppression system mounted*

## C. Demonstration One

Overall, the test was a partially successful demonstration the capabilities of the system. FireBot reached the waypoint successfully in all tests. Due to the fact that the navigation system uses an internal coordinate system, not based off any sensors such as a GPS module, coordinate changes are not tracked while following IR sources. This issue meant that each test had an expected error equal to the displacement of FireBot while it was in the SEARCH state. Of the 15 sorties, 60% of them returned to base station coordinates within the expected error. Two additional sources of error due to the test environment were noted. At $x = 46in$ there was a small lip in the floor. FireBot did not have a problem driving straight over it, but it occasionally impeded turns. Additionally, the floor was dusty. After seven sorties, it was observed that the wheels were so dusty that they were losing traction. Before resuming testing, the wheels were washed. Finally, in three trials, point 4 in trial #1, and point 5 in trials #2 and #3, FireBot tracked almost all the way to the IR target but drifted off to the left to the corner. The team believes that there may have been an unanticipated heat source, such as a vent, that was not visible.

* - Collision, movement off course, or failure to acquire heat source.

† - Trial occurred after cleaning the wheels.

| Trial #1 | | |
|---|---|---|
| **Point** | **x (in.)** | **y (in.)** |
| 1 | -32 | -26 |
| 2 | -68 | -26.5 |
| 3 | -55.5 | 3 |
| 4* | -41 | 201 |
| 5* | -24 | 205 |

*Table 6 Trial #1 End Locations for Demonstration One*

| Trial #2 | | |
|---|---|---|
| **Point** | **x (in.)** | **y (in.)** |
| 1† | 3 | -41 |
| 2† | -42 | -31 |
| 3† | -44 | 7 |
| 4 | -46 | 64 |
| 5* | -78 | 162 |

*Table 7 Trial #2 End Locations for Demonstration One*

| Trial #3 | | |
|---|---|---|
| **Point** | **x (in.)** | **y (in.)** |
| 1† | 4 | -39 |
| 2† | -52 | -28 |
| 3† | 36 | 46 |
| 4 | -40 | 62 |
| 5* | -20 | 231 |

*Table 8 Trial #3 End Locations for Demonstration One*

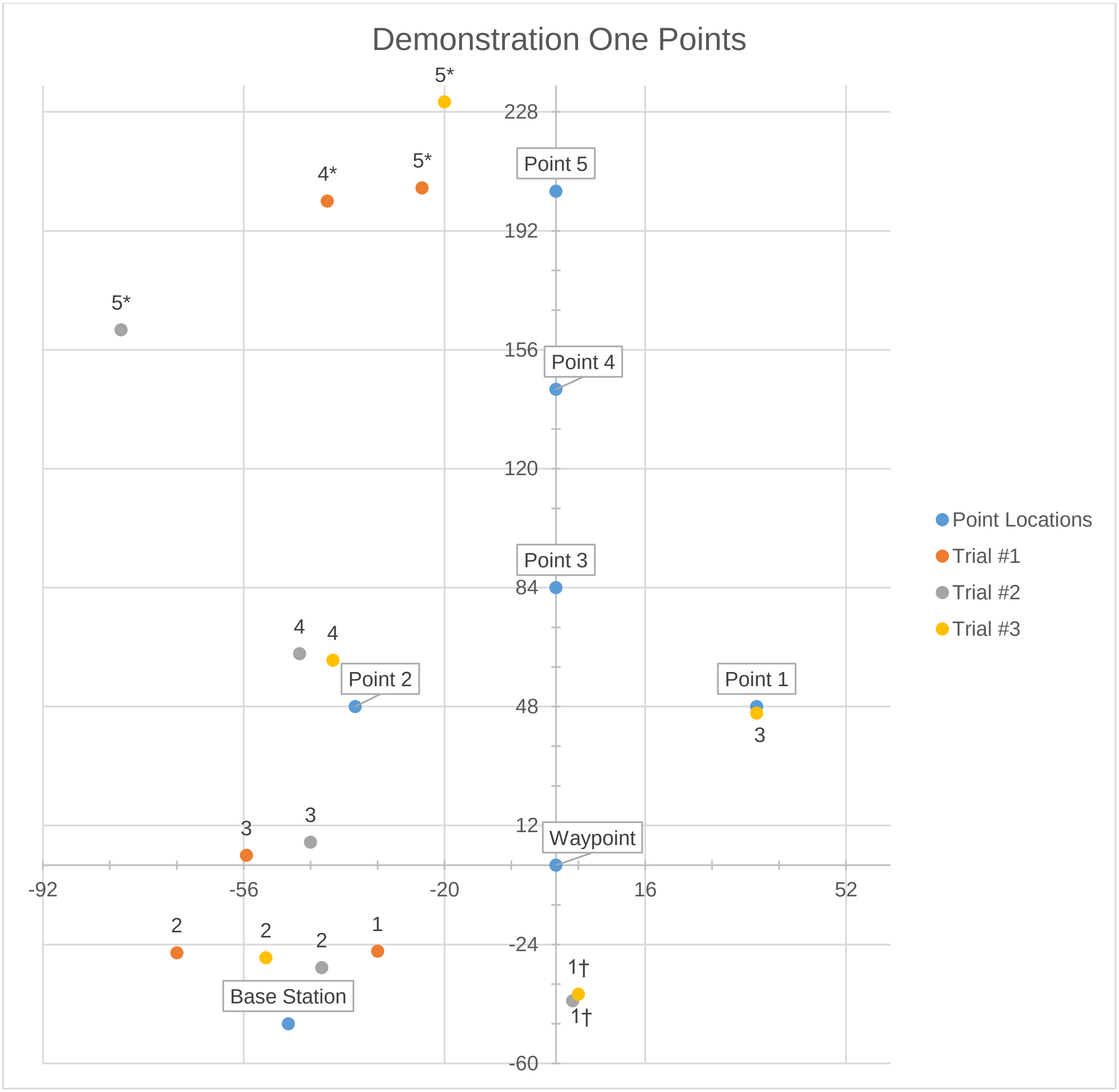

*Figure 17 Demonstration One Points*

**D. Demonstration Two**

This demonstration was mostly successful. FireBot was able to navigate accurately to the heat source and deploy its suppressant at the source. There was some difficulty with the clearance of the tubing with respect to the obstacle, however this is an easy fix by adding additional ultrasonic sensors to make it more aware of its surroundings and minimizing the size of the hose. Additionally, FireBot was not able to navigate back to the base station after the suppressant was deployed. This issue was resolved when the suppression system was taken off. The reason this happened is unclear but is very likely not a software issue. It should be noted that the results are more consistent as can be seen in Figure 18.

| Trial #1 | | |
|---|---|---|
| Obstacle? | x (in.) | y (in.) |
| Y | 0 | 31.5 |
| N | 1 | 17 |

*Table 9 Trial #1 End Locations for Demonstration Two*

| Trial #2 | | |
|---|---|---|
| Obstacle? | x (in.) | y (in.) |
| Y | 2.5 | 31 |
| N | 0 | 21 |

*Table 10 Trial #2 End Locations for Demonstration Two*

| Trial #3 | | |
|---|---|---|
| Obstacle? | x (in.) | y (in.) |
| Y | 3 | 25 |
| N | 0 | 20 |

*Table 11 Trial #3 End Locations for Demonstration Two*

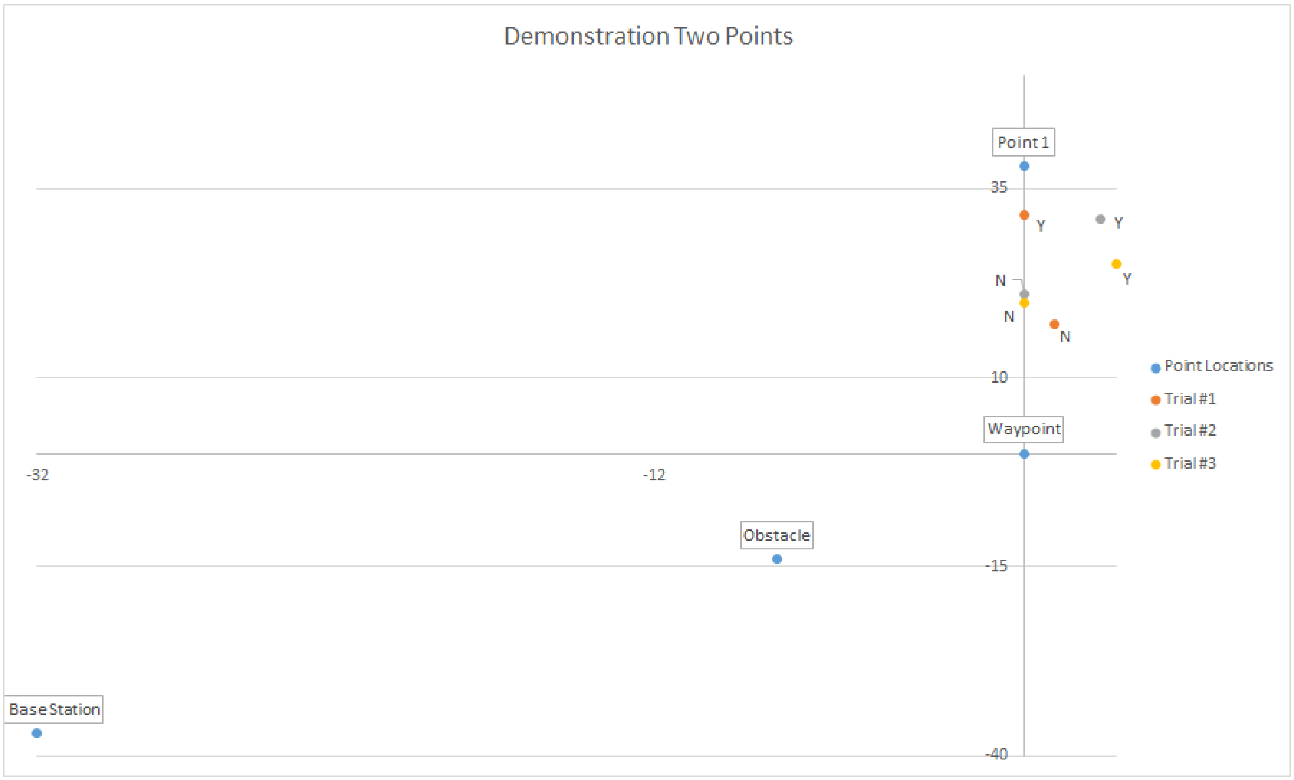

*Figure 18 Demonstration Two Points*

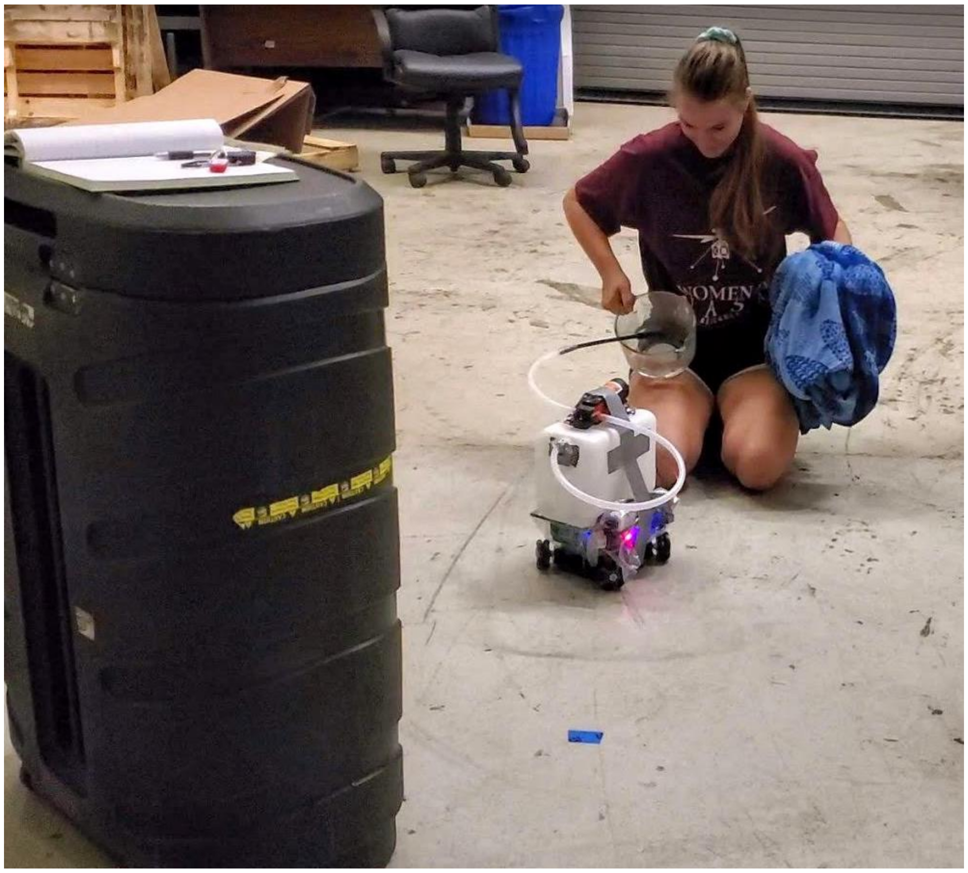

*Figure 19 FireBot spraying water on a simulated fire (a Person)*

## XIV. Future Work

### A. Flight

When FireBot transitions from being a terrestrial robot to an aerial drone, a new chassis must be selected. The first option to consider is the design created by the FireBot design team [3]. Because this would involve creating a custom chassis, a study of existing drones should be conducted to identify if there are any suitable commercial-off-the-shelf options. Some existing designs to consider are the Alta 8 and the Yuneec H250. The former is an octocopter with a flight time of 15 minutes, a maximum payload of 20lbs and an auto-land feature. The latter is a hexacopter with a longer flight time of 30 minutes but without auto-land capabilities.

### B. Software

The software created for FireBot makes extensive use of object-oriented programing. All of its components are represented as objects. In the future, when components such as the chassis or sensors are inevitably changed, this allows for minimal disruption to the main body of the code. Rather than edit preexisting classes, a new class can be made for the new component. Keeping as much of the specific implementation of features in classes, as opposed to the main loop function, ensures that the code is organized and easily extendable. In the code itself, there are also several functions, classes, and states, such as TAKE_OFF and Thermistor, that are unimplemented. While they were not relevant to the initial version of FireBot, they were created in anticipation of necessary features. The code is also documented with Doxygen style comments. While Doxygen was not able to be used to generate documentation due to technical issues, the standardized comment structure makes it significantly easier to understand comments. It is recommended that future additions continue to document the code with Doxygen style comments and continue with the object oriented paradigm for consistency, standardization, and ease of maintenance.

### C. Navigation

The current navigation system does not utilize sensors to determine its x and y coordinates. If there is an anomaly which affects how far or in what direction it travels, such as uneven terrain for the terrestrial bot or gusts for the drone, the drone will not be able to correct and will arrive at a different set of coordinates. It is necessary on future versions to add a GPS module. Typical GPS systems are accurate to around 16ft [10] which is approximately the outer range of the Grid-EYE. With this module, x and y can be replaced by latitude and longitude which will allow the system to account for unexpected movements. This will also resolve the issue caused by the fact that coordinates are only tracked while the navigation system is being actively followed. Using GPS, FireBot will be able to return to the base station with minimal error regardless of how far it traveled while following IR sources.

One issue encountered in testing was that the singular ultrasonic sensor was not able to tell when the robot had fully cleared an obstruction. This caused an issue in the second demonstration where the nozzle would collide with obstructions due to how far out it was from the area the ultrasonic sensor scanned. This could be alleviated by adding more ultrasonic sensors to create a wider field of view that is scanned for obstructions.

FireBot will initially acquire coordinates from human input but eventually it is hoped that another drone will identify the coordinates of a spot fire and send the GPS location to FireBot. In the initial research of this project it was found that a few different groups are working on locating spot fires [11] [12] [13]. This projects goal was to put out those spot fires so that when working in conjunction with the other type of drone, the system would become self-sufficient.

In the future, it will also be necessary to develop a more intelligent IR tracking algorithm. While it has not been tested, the current algorithm is likely to be thrown off by background sources. Since, FireBot is intended to operate near a larger wildland fire, it will need to be capable of distinguishing the smaller spot fire through the noise. This problem could be solved using an algorithm that can quantify the size of a fire. One property that could be used for classifying heat sources is the width of the source. The main wildland fire will span a wide area where as a spot fire will be localized, appearing as a much more circular blob. This may be improved by using a higher resolution IR sensor array or by switching to a thermal camera. Another possibility would be adding a camera to utilize machine vision to identify the location of the fireline. In addition to better navigation, adding a standard or thermal camera will improve telemetry sent to the operator, allowing them to better identify what the drone is looking at.

### D. Communications

One of the major changes between the current terrestrial version and future drone versions of FireBot will be a drastic increase in range. Where current versions are limited to within 30ft [14], the range of the Bluetooth module, the final iteration will be expected to extinguish fires far from the operator. This is an intrinsic limitation of Bluetooth as it is only intended to be used over short range. Because of this, future versions will need to switch from Bluetooth to a longer range standard.

One option for a long range communications standard is WiFi. While typically used for local area networks, WiFi can communicate over longer ranges if the base station is connected to a larger external antenna. Similar to WiFi, there is also the option to use LoRa, which is similar to WiFi. The key difference is that LoRa sacrifices bandwidth for reduced battery usage and increased range. The relatively small packet size may make LoRa a good option. Both WiFi and LoRa are still limited by line of sight which may cause problems as FireBot descends to get close enough to the spot fire.

To avoid the line of sight issue, it would also be possible to use a GSM module to connect to a cellular network. This has the advantage of minimizing the extra hardware needed on both FireBot and its base station as it does not require a bulky directional antenna. Firefighting agencies already use cellular networks to communicate and allocate resources which may make integrating it into existing firefighting command vehicles easier [15]. Another similar option would be to connect to a satellite network. While this has the advantage of allowing for connectivity everywhere, even in places without cellular coverage, it has some disadvantages compared to GSM. Because these satellite networks are operated in geostationary orbit, communication is slower. The other downside to satellite networks is that they are expensive, often charging a significant amount for the modem required to access them and for all the bandwidth used.

While not necessary, adding a buzzer would be a nice quality of life feature. Different tones and combinations of beeps could be easily used to tell an operator what state FireBot is currently in along with enabling it to communicate things such as battery capacity to a person not looking at the base station computer. This is very common on other drones and other robots.

### E. Control Board

Currently, FireBot has one Arduino for operating the chassis and one for all sensor-reading and decision-making. This choice was made due to the limited number of IO pins available on the chassis' Arduino. The only pins not used by motor encoders and $I^2C$ are digital 0 and digital 1. This is not ideal because it adds code complexity, additional hardware, and wiring connections. All of these create additional possible failure points. Even though the primary Arduino makes the decisions about motor control, it does not have direct control over the chassis. It is recommended that future versions switch to a new control board that has sufficient IO capabilities and processing power to connect to all the sensors and directly control the chassis such as an Arduino Mega or possibly a single board computer.

### F. Base Station

It is vital that future versions of the base station allow for two way communications. There is a function, readBluetooth(), in comms.ino (Appendix I) that was intended to be implemented but was not due to time constraints. It would likely be the best option to utilize JSON for this to keep it consistent with the outgoing communications.

Hardware wise, as described in the Communications section, the base station will require either an external antenna, GSM module, or satellite modem depending on which long range communications method is selected. The base station used in tests only included the minimum hardware needed to communicate with the bot and did not include items necessary to resupply it. For field applications the base station also includes space for the drone or multiple drones to be kept and maintained as necessary. The base station should include battery replacements and charging, and either the ability to swap suppressant tanks with ease or refill on-board tanks. Ideally, these things would be done mostly autonomously but could also be accomplished with a person manually refilling and replacing components. The manual option is only applicable with the caveat that the drone would be outperforming the human assisting it in extinguishing spot fires.

The base station is an integral part of making FireBot a useful tool in a real emergency. Therefore, it is imperative that it provides an intuitive GUI and user experience. This will make it easier to train operators for the system in addition to minimizing possible mistakes that the operator can make. For FireBot to make use of its full feature set, it must accessible by the operator at the base station.

When the finalized drone version is completed, it would be beneficial to enable the base station to control multiple FireBots. This would allow the system the capability to fight multiple spot fires simultaneously. The most straightforward method for accomplishing this would be to modify the ground station to allow the control of multiple FireBots without changing how each FireBot operates. This would be simple and effective but it prevents individual drones from working together autonomously. This could be taken farther by allowing individual FireBots to communicate with one another. This would create capabilities such as being able to deploy a small swarm to work together to extinguish a brush fire. While swarm technology would make this the ideal version of FireBot, it would pose a significant technical challenge. It may be far more effective to simply have the base station route the drones to ensure they never get too close to each other while extinguishing multiple spot fires and leaving larger fires to firefighting professionals.

### G. Thermal Protection

In the case of the terrestrial proof of concept, the size and weight limits of the robot restricted the type of cooling system used. For the final hexacopter drone version that can be much bigger and carry more weight, an active cooling system should be considered, instead of just insulation. Some active systems that can be tested are water or air coolant pumped-loop systems, heat exchangers and radiators. A more efficient thermal protection system can allow FireBot to get closer to the wildfire and increase the time period that it can operate near the flames, thus increasing the effectiveness of fire suppression.

### H. Fire Suppression

A future version of FireBot's suppression system would probably include less duct tape. Since this version of FireBot was an early prototype, no actual holes were put to fasten the tank and tubing. This was so these parts could be reused in the future. Mounted properly to a device all of the components purchased could go on the actual final drone version in the future. The system itself is precisely built to become the flight payload, meeting the 15 lbs weight limit. The nozzle also could use a proper male-male tubing connector as well as tubing clamps to secure it. Tubing clamps can also be added to the pump for the inlet and outlet hose barbs.

In the event the future drone payload is larger it is recommended battery capacity and tank size be increased first. As a result, FireBot would be able to run more sorties and stay active in the field longer. Other suppressant methods could be looked into for alternate applications where the drone is deployed differently into the field. One possible alternate application would be for the drone to be packed in with a smoke jumper, a firefighting professional that parachutes into a wildland fire. The alternate drone would need the ability to use varying sources of water and charge its batteries using solar power. Finally, as mentioned earlier, a foam nozzle is an easy addition that could make the sprayer method even more effective.

For future testing of FireBots sprayer system it is recommended that a sizable plastic bag is secured over the nozzle so any spray is contained. Additionally, adding an LED that turns on before spraying commences would assist anyone in the line of spray to be, at the very least, notified that they are located in an active splash zone.

## XV. Conclusion

As a proof of concept, the initial version of FireBot has demonstrated the feasibility of tracking and extinguishing a fire given only its general area. Using a heading derived from an accelerometer for tracking its coordinates along with a low resolution IR array for heat source tracking, FireBot is able to reach its intended target. Using a state machine for decision making, the entire mission is organized within the software. In order to protect FireBot from the extreme temperatures of wildfires, a thermal analysis was done to select an insulation material. Ceramic paper with a thermal conductivity of 0.2 $W/m^2 K$ was selected in a thickness of 1/8in based on calculations. The insulation was also tested experimentally to verify the theoretical calculations and it proved sufficient to withstand wildfire temperatures. FireBot's suppressant system proves that an effective system can fit within the weight requirements of a future drone application. Together the abilities to arrive at coordinates, locate a fire with IR, protect on-board components, and suppress fire make FireBot a key step in the future of technology when it comes to fighting wildland fires.

## XVI. References

[1] Samanta, A, "Key Findings from the 2017 Verisk wildfire risk analysis," Verisk, 12 7 2017. [Online]. Available: https://www.verisk.com/insurance/visualize/key-findings-from-the-2017-verisk-wildfire-risk-analysis/. [Accessed 31 7 2019].

[2] U.S. Fire Administration, "U.S. Fire Statistics," [Online]. Available: https://www.usfa.fema.gov/data/statistics/. [Accessed 31 7 2019].

[3] H. Patel, A. Hyland and D. Whiting, "Helicopter Emergency and Rescue Operations (HERO) - FireBot Design Project," NASA Aeromechanics, 2019.

[4] RobotShop, "4WD 58mm Omni Wheel Arduino Robot," [Online]. Available: https://www.robotshop.com/en/4wd-58mm-omni-wheel-arduino-robot.html?utm_source=rb-community&utm_medium=forum&utm_campaign=4wd-58mm-omni-wheel-arduino-robot-one-wheel-speed-problem. [Accessed 31 7 2019].

[5] Natural History Museum of Utah, "Wildefires: Interesting Facts and F.A.Q," [Online]. Available: https://nhmu.utah.edu/sites/default/files/attachments/Wildfire%20FAQs.pdf. [Accessed 31 7 2019].

[6] Mineral Seal Corporation, "Ceramic Fiber Paper," 2019. [Online]. Available: https://minseal.com/ceramic-fiber-paper/.. [Accessed 31 7 2019].

[7] B. Butler, "Characterization of convective heating in full scale wildland fires," in *VI International Conference on Forest Fire Research*, Viegas, 2010.

[8] National Wildfire Coordinating Group, "FOAM VS FIRE Class A Foam for Wildland Fires 2nd Edition," 1993.

[9] A. Vanaparti, "Experimental Study of Radiative Properties of Charcoal Combustion in Grills," Auburn University, 2016.

[10] National Coordination Office for Space-Based Positioning, Navigation, and Timing, "GPS Accuracy," 5 12 2017. [Online]. Available: https://www.gps.gov/systems/gps/performance/accuracy/. [Accessed 31 7 2019].

[11] M. J. Logan, L. J. Glaab and T. Craig, "Use of Small Unmanned Aircraft System for Autonomous Fire Spotting at the Great Dismal Swamp," NASA Langley Research Center, Hampton, Virginia.

[12] T. Giitsidis, E. G. Karakasis, A. Gasteratos and G. C. Sirakoulis, "Human and fire detection from high altitude UAV images," in *Euromicro International on Parallel, Distributed, and Network-Based Processing*, 2015.

[13] L. Savvides, "California's Fires Face a new, high-tech foe: Drones," *c|net,* 2018.

[14] Intorobotics, "The Guide to Bluetooth Modules for Arduino," 2 2018. [Online]. Available: https://www.intorobotics.com/pick-right-bluetooth-module-diy-arduino-project/. [Accessed 31 7 2019].

[15] R. Suppe, "Verizon throttled 'unlimited' data of Calif. fire department during Mendocino wildfire," *USA Today,* 23 8 2018.

[16] L. J. Marchetti, "Fire Dynamics Series: Estimating Fire Flame Height and Radiant Heat Flux From Fire," 2012. [Online]. [Accessed 31 7 2019].

# XVII. Appendices

## A. Omni-4WD-Driver.ino

```
/***************************************************************************
****

                                   Omni-4WD-Driver

  Script used to control the 4WD 58mm Omni Wheel Arduino robot.  It listens
to
  the I2C address 8 for commands.  Each command must consist of the
following:

  Interrupt: A char indicating whether or not to add the new command to the
queue
             or to dump the queue so it can take priority.
  Command:   A char indicating the desired move command.  The availible
commands
             are listed in the definitions.
  Distance: An unsigned short integer indicating the length of the
command.  For
             linear motions, this is interpreted as millimeters but for
rotational
             motions they are interpreted as degrees.

  Commands are added to the queue to be performed in order.

                                  //[[!WARNING!]]\\
                     MOST MOVEMENT CONSTANTS ARE UNCALIBRATED!
              THEY MUST BE ADJUSTED FOR THE ROBOT TO MOVE ACCURETLY!

  @file: Omni-4WD-Driver
  @author: Christian M. Schrader [cmschrader@wpi.edu]


****************************************************************************
***/

#include <Queue.h>
#include <Wire.h>
#include <PinChangeInt.h>
#include <PinChangeIntConfig.h>
#include <EEPROM.h>
#include <fuzzy_table.h>
#include <PID_Beta6.h>
#include <MotorWheel.h>
#include <Omni4WD.h>
#include <fuzzy_table.h>

// Move Commands: Used to Indicate Desired Motion
#define HALT 'H'                  // No Movement
#define FORWARD_ADVANCE 'F'       // Direct Forward Movement
#define RIGHT_ADVANCE 'R'         // Direct Right Movement
#define LEFT_ADVANCE 'L'          // Direct Left Movement
```

```
#define TURN_CLOCKWISE 'C'         // Rotate Clockwise
#define TURN_COUNTERCLOCKWISE 'W' // Rotate Counterclockwise
#define REAR_ADVANCE 'B'           // Direct Rearward Advance

// Interrupt: Used to Indicate Top Priority of a Command
#define INTERRUPT '!'               // Empties the queue and imedietly
executes the move.

// Movement Constants: Used as Control Function Constants
#define MMPS 100                   // Standard Millimeters Per Second
#define MMPS_TURN 10               // Standard Millimeters Per Second When
Turning
#define UPTIME 250                 // Time Length of Each Individual Movement
#define MS_PER_MM .1               // Miliseconds Taken to Move 1 Millimeter
at 100 MMPS.  [!UNCALIBRATED!]
#define MS_PER_DEGREE 1            // Miliseconds Taken to Turn 1 Degrees at
100 MMPS.     [!UNCALIBRATED!]

// Controller Objects: Used to Drive the Motors.
irqISR(irq1, isr1);
MotorWheel wheel1(3, 2, 4, 5, &irq1);
irqISR(irq2, isr2);
MotorWheel wheel2(11, 12, 14, 15, &irq2);
irqISR(irq3, isr3);
MotorWheel wheel3(9, 8, 16, 17, &irq3);
irqISR(irq4, isr4);
MotorWheel wheel4(10, 7, 6, 13, &irq4);
Omni4WD Omni(&wheel1, &wheel2, &wheel3, &wheel4);

class Command
/**
   @brief Defines a movement command.
*/
{
  public:
    char cmd;               // Indicates what type of movement to make.
    unsigned short dur;     // Duration of the command.
    Command() {}            // Default Constructor Must exist for commands to
be in the array.
    Command(char cmd, unsigned short dur) {
      /**
         Instantiates a command.
         @param cmd   Desired motion.  Must use defined motion commands.
         @param dur   Duration of the motion.  Must not be more than UPTIME.
      */
      this->cmd = cmd;
      this->dur = dur;
    }
};

// Instantiate the Command Queue
DataQueue<Command> queue(100);

void setup() {

  /*
     6 and 13 are declared as inputs because of a difference in wiring
    from the factory default to free up I2C.
  */
  pinMode(6, INPUT);
  pinMode(13, INPUT);
  Wire.begin(8);  //Begin I2C
  Wire.onReceive(receiveEvent);

  // Controller Setup
  TCCR1B = TCCR1B & 0xf8 | 0x01; // Pin9,Pin10 PWM 31250Hz
  TCCR2B = TCCR2B & 0xf8 | 0x01; // Pin3,Pin11 PWM 31250Hz
  Omni.PIDEnable(0.31, 0.01, 0, 10);
}

void loop() {
  move();
}

void move() {
  /**
    Executes the next command in the queue.
  */
  if (!queue.isEmpty()) { // If there is a command in the queue, execute it.
    Command action = queue.dequeue();
    if (action.cmd == HALT) {
      Omni.setCarSlow2Stop(UPTIME);
      Omni.setCarStop();
    }
    else if (action.cmd == FORWARD_ADVANCE) {
      Omni.setCarAdvance(0);
    }
    else if (action.cmd == RIGHT_ADVANCE) {
      Omni.setCarRight(0);
    }
    else if (action.cmd == LEFT_ADVANCE) {
      Omni.setCarLeft(0);
    }
    else if (action.cmd == TURN_CLOCKWISE) {
      Omni.setCarRotateRight(0);
    }
    else if (action.cmd == TURN_COUNTERCLOCKWISE) {
      Omni.setCarRotateLeft(0);
    }
    else if (action.cmd == REAR_ADVANCE) {
      Omni.setCarBackoff(0);
    }
    if (action.cmd != TURN_CLOCKWISE && action.cmd != TURN_COUNTERCLOCKWISE)
{
      Omni.setCarSpeedMMPS(MMPS, UPTIME);
    }
    else {
      Omni.setCarSpeedMMPS(MMPS_TURN, UPTIME);
    }

  }
  else {  // If there are no commands, halt.
    Omni.setCarSlow2Stop(UPTIME);
    Omni.setCarStop();
  }
  Omni.delayMS(UPTIME);   // Omni.delayMS sends the command to the motors.
}

void emptyQueue() {
  /**
     Empties the queue.  Intended to be used to clear the queue
     when a priority command is received.
  */
  while (!queue.isEmpty()) {
    queue.dequeue();
  }
}

void queueCommand(char cmd, unsigned short dur) {
  /**
     Adds commands to the queue.  If dur is greater than UPTIME,
     multiple commands will be queued such that they are never
     longer than UPTIME.
     @param cmd A defined Movement Command indicating desired
                motion.
     @param dur The length of the desired motion in milliseconds.
  */
  while (dur > UPTIME) {
    queue.enqueue(Command(cmd, UPTIME));
    dur -= UPTIME;
  }
  queue.enqueue(Command(cmd, dur));
}

void receiveEvent(int numBytes) {
  /**
     I2C receive function.  Interprets every 4 bytes as a command.
     Byte 1    : INTERRUPT - If it equals INTERRUPT, the command is given
                             priority.
     Byte 2    : cmd       - Used to the the desired motion of the command.
     Bytes 3/4: dist       - Used to indicate distance of the motion.  For
                             regular commands this is in milimeters but for
                             turns this is in degrees.
     @param numBytes  The number of bytes received.
  */
  for (int i = 0; i < numBytes; i += 4) { // For each 4 byte command
    // Read all 4 bytes from I2C
    char priority = Wire.read();
    char cmd = Wire.read();
    unsigned short dist = Wire.read() << 8 | Wire.read();

    // Dumps queue if INTERRUPT is in the priority byte.
    if (priority == INTERRUPT) {
      emptyQueue();
    }

    // Convert the distance into ms from degrees or mm.
    if (cmd == TURN_CLOCKWISE || cmd == TURN_COUNTERCLOCKWISE) {
      dist *= MS_PER_DEGREE;  // Convert Degrees to ms
    }
    else {
      dist *= MS_PER_MM;  // Convert mm to ms, figure out this conversion
factor, I don't think 1 is correct
    }
    queueCommand(cmd, dist);
  }
}
```

**B. Firebot.ino**

```
/***************************************************************************
****

                                   FireBot
                         Spot Fire Suppression Robot

  FireBot is a robot, currently terrestial but eventually a drone, designed
to
  extinguish spot fires that may comprosise firelines.  It waits in standby
until
  given coordinates.  After navigating to the coordinates, it will seek out
the
  fireu using an IR Sensor Array and deploy a fire suppressing payload.

  This file contains the definitions and imports used by the rest of the
program.

                                //[[!WARNING!]]\\
                  MOST MOVEMENT CONSTANTS ARE UNCALIBRATED!
           THEY MUST BE ADJUSTED FOR THE ROBOT TO MOVE ACCURETLY!

  @file: FireBot
  @author: Christian M. Schrader [cmschrader@wpi.edu]


****************************************************************************
***/

#include <ArduinoJson.h>
#include <Adafruit_AMG88xx.h>
#include <Adafruit_GPS.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include<SoftwareSerial.h>
#include<Wire.h>

// Pins
#define TRIGGER_PIN 2                    // Range Finder TRIGGER_PIN Pin
#define ECHO_PIN 3                       // Range Finder ECHO_PIN Pin
#define FIRE_PIN 4                       // Fire Detector Data Pin
#define GPSTX 7                          // GPS Transmit Pin
#define GPSRX 8                          // GPS Receive Pin
#define TEMP_PIN 9                       // Thermistor Data Pin
#define BLUETOOTH_RX 10                  // Bluetooth Receive Pin
#define BLUETOOTH_TX 11                  // Bluetooth Transmit Pin
#define SUPPRESSION_PIN 12               // Suppression Payload Pin

// Sensor Constants
```

```
#define RANGERPS 20                       // Range Finder Refresh Rate in
Readings Per Second
#define IRFPS 4                           // IR Array Refresh Rate in Frames
Per Second
#define TIMEOUT 5000                      // Time in milliseconds given for
sensors to boot up

// Move Commands
#define HALT 'H'                          // No Movement
#define FORWARD_ADVANCE 'F'               // Direct Forward Movement
#define RIGHT_ADVANCE 'R'                 // Direct Right Movement
#define LEFT_ADVANCE 'L'                  // Direct Left Movement
#define TURN_CLOCKWISE 'C'                // Rotate Clockwise
#define TURN_COUNTERCLOCKWISE 'W'         // Rotate Counterclockwise
#define REAR_ADVANCE 'B'                  // Direct Rearward Advance
#define ASCEND 'A'                        // Upwards Ascent
#define DESCEND 'D'                       // Downwards Descent

// Priority Commands
#define INTERUPT '!'                      // Imedietly Execute a Command and
Dump the Queue
#define QUEUE '.'                         // Queue the Next Move Command.

#define ACQUIRE_HEADING_TOLERANCE 1.0   // Maximum Deviation from Target
Heading While Acquiring Target in Degrees
#define MAINTAIN_HEADING_TOLERANCE 5.0  // Maximum Deviation from Target
Heading After Acquiring Target in Degrees
#define LOCATION_TOLERANCE 250           // [!UNCALIBRATED!] Maximum Deviation
from Target Coordinates When Considering Being Arrived
#define UNIT_DISTANCE 250                // [!UNCALIBRATED!] Standard Movement
Distance in mm
#define UNIT_ROTATION 5                  // [!UNCALIBRATED!] Standard Rotation
Distance in degrees
#define IR_MOVEMENT_THRESHOLD 1600       // [!UNCALIBRATED!] Minimum Sum of
All Pixels that the IR Array Consideres to be Arived
#define IR_DETECTION_THRESHOLD 23        // [!UNCALIBRATED!] Minimum
temperature in degrees celcius that is persued
#define AVOIDANCE_THRESHOLD 20           // [!UNCALIBRATED!] Distance at Which
the Navigation Class Will Reroute Around an Obstacle
#define MOVE_INTERVAL 250                // Time Between Standard Move
Commands in ms.  Recomended to be Equal to UPTIME in Omni-4WD-Driver.ino

/*
   Bot State Enumeration
   Used to indicate what state the bot is in.
*/
enum bot_state {
  STANDBY,
  INIT,
  TAKE_OFF,
  CRUISE,
  SEARCH,
  SUPPRESS,
  RECALL
};
bot_state state = STANDBY;               // Default state
```

```
// Initialize variables
float *pixels;                          // Pointer to pixel array used by IR
cameras.
int lastMove = millis();                // Time the last move command was
sent.
bool newCoordinates = true;             // Indicates New Coordinates Have
Been Received

// Begin Bluetooth Connection.
SoftwareSerial Bluetooth(BLUETOOTH_RX, BLUETOOTH_TX);
```

### C. 1_Chassis.ino

```
/*****************************************************************************
****

                                      Chassis

  Chassis contains classes relevant to control of the movement.  New chassis
types
  should have their own class that inherits from Chassis.  Chassis objects
are
  controlled by using the transmit function which sends a command to the
drive
  system.

  @file: 1_Chassis
  @author: Christian M. Schrader [cmschrader@wpi.edu]


*****************************************************************************
***/
class Chassis
/**
   @brief Base class used to interface with movment systems.
*/
{
  public:
    // Transmit a command to the drive system
    virtual void transmit(char command, unsigned short distance, bool
interupt) = 0;
};

class OmniDrive : public Chassis
/**
   @brief Interfaces with a quad omni wheel drive running Omni-4WD-
Driver.ino.
*/
{
  public:

    void transmit(const char command, unsigned short distance, bool interupt)
{
      /**
         Transmits a command to the drive system.  Sends it over I2C to be
         interpreted by the chassis Arduino using Omni-4WD-Driver.ino.
         @param command   Const Char indicating the type of command.  Must be
```

```
                          one of the commands defined under Move Commands in
                          FireBot.ino.
         @param distance  Number indicating either degrees for a rotational
                          command or mm for a linear command.
         @param interupt  If true, will dump the queue and imedietly execute
                          the command.  If false, the command will be put
into
                          the queue.
      */
      Serial.print("CHASSIS: ");
      Serial.print(command);
      Serial.print(" > ");
      Serial.println(distance);

      Wire.beginTransmission(8);
      if (interupt) {
        Wire.write(INTERUPT);
      }
      else {
        Wire.write(QUEUE);
      }
      Wire.write(command);
      // Convert short to bytes
      Wire.write(distance >> 8);    // Transmit 1st byte of short.
      Wire.write(distance & 0xFF);  // Transmit 2nd byte of short.
      Wire.endTransmission();
    }
};

class Hexacopter : public Chassis
/**
   Interfaces with a six-rotor drone system.
   This class serves as an empty place holder.  In future drone versions
   this class should control all drone movments.
*/
{};
```

**D. 2_Range.ino**

```
/*****************************************************************************
****

                                    Range

  Range contains classes used to interface with range finders.  To use them,
  instantiate them and call their connect() function.  Include update() in
your
  main loop.  Whenever you call range(), it will return the last distance
read by
  the sensor.

  @file: 2_Range
  @author: Christian M. Schrader [cmschrader@wpi.edu]


*****************************************************************************
***/
```

```
class RangeFinder
/**
   @brief Base class used to interface with Range Finders.
*/
{
  public:
    bool connected = false;          // Indicates if the sensor is connected
and ready.
    virtual bool connect() = 0;      // Connects the sensor.
    virtual void disconnect() = 0;  // Disconnects the sensor.
    virtual float range() = 0;       // Returns the last read value from the
sensor.
    virtual void update() = 0;       // Activates the sensor if it is due for
an update.
};

class HCSR04 : public RangeFinder
/**
   @brief Reads data from an HC-SR04 Ultrasonic Range Finder.

   It should be noted that the HC-SR04 has not inherent need to connect.  The
   connect() function does nothing other than change its status.  The reason
   this is done is to keep it consistant with the other sensors which all
   operate in this manner.
*/
{
  public:
    HCSR04(int trigger, int echo, int readings_per_second)
    {
      /**
         Instantiates an HCSR04.
         @param trigger                The sensor's trigger pin.
         @param echo                   The sensor's ech pin.
         @param readings_per_second   The maximum number of readings per
second
                                       to perform.
      */
      this->trigger = trigger;
      this->echo = echo;
      this->interval = 1000 / readings_per_second;  // Convert RPS to time
between readings
      pinMode(this->trigger, OUTPUT);
      pinMode(this->echo, INPUT);
    }

    bool connect()
    /**
       Turns the sensor on so it can be read.
       @return: true
    */
    {
      connected = true;
      return connected;
    }

    void disconnect()
    /**
```

```
       Turns off the sensor.
    */
    {
      connected = false;
    }

    void update()
    /**
       Reads a new sensor value if it is due.
    */
    {
      if (connected && millis() - lastUpdate >= interval) {
        lastUpdate = millis();
        /*
           For information on how the HCSR04 works, see the following URL:
           https://randomnerdtutorials.com/complete-guide-for-ultrasonic-
sensor-hc-sr04/
        */
        digitalWrite(this->trigger, LOW);
        delayMicroseconds(2);
        digitalWrite(this->trigger, HIGH);
        delayMicroseconds(10);
        digitalWrite(this->trigger, LOW);
        this->dist = uSTOcm * pulseIn(this->echo, HIGH);
      }
    }

    float range()
    /**
       Returns the last measured distance to an object.
       @return The last read distance as a float.
    */
    {
      return dist;
    }
  private:
    unsigned int trigger;     // The trigger pin
    unsigned int echo;        // The echo pin
    unsigned int interval;    // Minimum interval between sensor reads in ms
    unsigned int lastUpdate;  // The last time the sensor was read
    float uSTOcm = .01723;    // Conversion of mircroseconds to cm for the
sensor ping.
    float dist;               // The last read distance in cm
};
```

**E. 3_Thermistor.ino**

```
/****************************************************************************
****

                                  Thermistor

  Thermistor contains classes used to interface with thermistor.  To use
them,
  instantiate them and call their connect() function.  Include update() in
your
```

```
  main loop.  Whenever you call readTemp(), it will return the last distance
read
  by the sensor.

  @file: 3_Thermistor
  @author: Christian M. Schrader [cmschrader@wpi.edu]


*****************************************************************************
***/
class Thermistor
/**
   @brief Base class used to interface with thermistors.
*/
{
  public:
    virtual float readTemp() = 0;   // Returns the last read value from the
sensor
    bool connected = false;         // Indicates if the sensor is connected
and ready
    virtual bool connect() = 0;     // Connects the sensor
    virtual void disconnect() = 0;  // Disconnec the sensor
    virtual void update() = 0;      // Activates the sensor if it is due for
an update
  protected:
    int pin;                        // The data pin used by the sensor
};

class TMP36 : public Thermistor
/**
   Reads data from a TMP36.
   [!UNIMPLEMENTED!] - There are not implementations that override the pure
                       virtual functions in its super class.
*/
{
  public:
    TMP36(int pin)
    {
      /**
         Instantiates a TMP36
         @param pin   The pin used for reading data.
      */
      this->pin = pin;
    }
};
```

### F. 4_Navigation.ino

```
/*****************************************************************************
****

                                 Navigation

  Navigation contains classes used to interface with navigation sensors and
to
  make navigation decisions.  To use them, instantiate them and call their
  connect() function.  Include update() in your main loop.  Whenever update
is
  called, they will update their sensor(s) if nessesary and make a movement
  decision, calling the nessesary functions to do that.

  @file: 4_NAVIGATION
  @author: Christian M. Schrader [cmschrader@wpi.edu]


*****************************************************************************
***/
class Nav
/**
   @brief Base class used to interface with sensors and make navigation
decisions.
*/
{
  public:
    bool onTarget = false;           // Indicates that the robot is on the
correct heading.
    bool connected = false;          // Indicates that the sensor is connected
and ready.
    virtual void update() = 0;       // Reads sensors if nessesary and makes
navigation decisions.
    virtual bool connect() = 0;      // Connects to the sensors and prepares
for navigation.
    virtual void disconnect() = 0;   // Disconnects from the sensors.
    virtual bool arrived() = 0;      // Determines if the robot is at its
target coordinates.

    void execmd(char cmd, int dist, bool override) {
      /**
         Executes a movement command.
         @param cmd       The char command to execute.  Must be from the
defined
                          values in FireBot.ino.
         @param dist      The distance in degrees (for rotation) or mm (for
                          translation).
         @param override  If true, forces the execution.
      */
      if (millis() - lastMove > MOVE_INTERVAL || override) {
        lastMove = millis();
        drive->transmit(cmd, dist, true);
      }
    }

    void setTarget(int x, int y) {
      /**
         Sets the target coordinates for the navigation system.
         @param x The x coordinate that is also the latitude.
         @param y The y coordinate that is also the longitude.
      */
      onTarget = false;
      target[0] = x;
      target[1] = y;
    }
```

```
void returnToBase() {
  /**
     Sets the base station as the target coordinates.
  */
  onTarget = false;
  target[0] = base[0];
  target[1] = base[1];
}

float findTargetHeading(float x, float y) {
  /**
     Finds the heading nessesary to point straight at the target
coordinates
     from a given set of coordinates.
     @param x The x coordinate.
     @param y The y coordinate.
     @return The heading in degrees clockwise from the +y axis.
  */
  float dx = target[0] - x;
  float dy = target[1] - y;
  float angle = atan(dx / dy) * 180 / M_PI;

  /*
     Because atan() only returns posative values, the value is modified
     depending on what qaurter of the graph the angle lies in.
  */
  if ( dy > 0 && dx < 0) { // q2
    angle *= -1;
  }
  else if (dy < 0 && dx < 0) { // q3
    angle += 180;
  }
  else if (dy < 0 && dx > 0) { // q4
    angle *= -1;
    angle += 180;
  }
  // Convert negative angles to posative ones.
  if (angle < 0) {
    angle += 360;
  }
  return (float) angle;
}

float distToTarget(float x, float y) {
  /**
     Calculates the remaining distance to the target from a given set
     of coordinates.
     @param x   The x coordinate.
     @param y   The y coordinate.
     @return    The distance remaining as a float measured in coordinate
                grid units (whatever units were given to the grid).
  */
  int dx = target[0] - x;
  int dy = target[1] - y;
  return sqrt(pow(dx, 2) + pow(dy, 2));
}
```

```
    float getLat() {
      /**
         @return  The x or latitude coordinate.
      */
      return current[0];
    }

    float getLon() {
      /**
         @return  The y or longitude coordinate.
      */
      return current[1];
    }
  protected:
    float current[2];           // The current coordinates of the robot.
    float target[2] = {0, 0};   // The target coordinates.
    float base[2] = {0, 0};     // The current coordinates of the base
station.
    float reroutHeading = -1;   // The override target heading used to rerout
around obstacles.
    Chassis* drive;             // Pointer to the chassis object that is
given commands.
    RangeFinder* rangeForward;  // Pointer to the forward range finder.
};

class Accelerometer : public Nav
/**
   @brief Uses an internal coordinate system along with accelerometer heading
to navigate.

   The internal coordinate system is laid out as follows:


                                     0°
     Q2                               |                         Q1
                                      |   [+y]      /\
                                      |            /__\
                                      |             ||   Forward
                                      |             ||
                                      |
                                      |
                           //   _______|_______   \\
                          // \/   ____|__     \/ \\
                             |   [    |  ]     |
                             |   [____|__]\    |
   270°--------------------------------------------------------------- 90°
       [-x]                  |  =====-|-----|] |            [+x]
                             |  ===== |  \--|{ |
                          \\ /\_______|_______/\ //
                           \\         |         //
                                      |
                                      |
                                      |
                                      |
                                      |
                                      |   [-y]
     Q3                               |                         Q4
```

```
                                        180°

*/
{
  public:
    Accelerometer(Chassis* drive, RangeFinder* forward) {
      /**
         Instantiates the class.
         @param drive     Pointer to the chassis object that is given
commands.
         @param forward   Pointer to the forward range finder.
      */
      this->drive = drive;
      this->rangeForward = forward;
    }
    bool connect() {
      /**
         Connects the sensor and readies the system.
         @return  Boolean indicating whether or not the system initialized
correctly.
      */
      current[0] = 0;
      current[1] = 0;
      bno = new Adafruit_BNO055(55);
      connected = bno->begin();
      bno->setExtCrystalUse(true);
      return connected;
    }

    void disconnect() {
      /**
         Disconnects from the sensor.
      */
      delete bno;
      connected = false;
    }

    void update() {
      /**
         Makes a new navigation decision and acts on it.
      */
      // Get heading
      sensors_event_t event;
      bno->getEvent(&event);
      int heading = event.orientation.x;
      int targetHeading = findTargetHeading(current[0], current[1]);

      // Shift target heading to 180 degrees.
      heading += 180 - targetHeading;
      if (heading < 0) {
        heading += 360;
      }
      else if (heading > 360) {
        heading -= 360;
      }
      targetHeading += 180 - targetHeading;
      // Determine whether to already on target or acquiring.
      int tolerance = ACQUIRE_HEADING_TOLERANCE;
      if (onTarget) {
        tolerance = MAINTAIN_HEADING_TOLERANCE;
      }

      /*
         Make movement decision.  The if determines that the robot is
         on target and can move forward if the following statements
         return true in the following order:

         detectCollision: If there are no obstructions, returns true
                          else it sets a rerout heading and goes
                          around.
         seekHeading():   If on target returns true else it rotates
                          towords the correct heading.
         !arrived():      Returns true if the robot is not at the
                          target.
         Interval:        Returns true if it has been at least
                          MOVE_INTERVAL ms since the last move.
      */
      if (detectCollision(heading, event.orientation.x) &&
          seekHeading(targetHeading, heading, tolerance) &&
          !arrived() &&
          millis() - lastMove > MOVE_INTERVAL) {
        onTarget = true;
        coordinateChange(event.orientation.x, UNIT_DISTANCE);
        execmd(FORWARD_ADVANCE, UNIT_DISTANCE, false);
      }
    }

    bool arrived() {
      /**
         Determines if FireBot has arrived at its target coordinates.
         @return  true if it has arrived.
      */
      if (distToTarget(current[0], current[1]) < LOCATION_TOLERANCE) {
        return true;
      }
      return false;
    }

  private:
    bool seekHeading(int targetHeading, int currentHeading, int tolerance) {
      /**
         Checks if the robot is pointed towords the target heading.  If it is
         not, it rotates towords it.
         @param targetHeading   The desired heading in degrees clockwise from
                                the +y axis.  Should be 180°.
         @param currentHeading  The current heading in degrees clockwise from
                                the +y axis.
         @param tolerance       The maximum deviation allowed between
                                targetHeading and currentHeading
         @return                Boolean indicating whether or not the robot
                                is done rotating, meaning that it is on
target.
      */
      bool doneRotating = true;
      if (abs(currentHeading - targetHeading) > tolerance) {
        onTarget = false;
        doneRotating = false;
        if (currentHeading > targetHeading) {
          execmd(TURN_COUNTERCLOCKWISE, UNIT_ROTATION, false);
        }
        else {
          execmd(TURN_CLOCKWISE, UNIT_ROTATION, false);
        }
      }
      return doneRotating;
    }

    bool detectCollision(int heading, int trueHeading) {
      /**
         Detects if the robot must be rerouted to avoid a collision.
         @param heading      The translated heading where the targetHeading
has
                            been translated to be 180°.
         @param trueHeading  The raw accelerometer heading.
         @return             true if there is no rerouting to be done.
      */
      bool clear = false;
      // If rerout is less than 0, it is considred on target.
      if (reroutHeading > 0) {
        if (seekHeading(reroutHeading, heading, ACQUIRE_HEADING_TOLERANCE)) {
          if (rangeForward->range() < AVOIDANCE_THRESHOLD) {
            reroutHeading -= 180;
          }
          else {
            reroutHeading = -1;
            coordinateChange(trueHeading, UNIT_DISTANCE);
            execmd(FORWARD_ADVANCE, 10 * UNIT_DISTANCE, true);
            delay(3000);
          }
        }
      }
      // If too close, check the right for a clear path.
      else if (rangeForward->range() < AVOIDANCE_THRESHOLD) {
        reroutHeading = heading + 90;
      }
      else {
        clear = true;
      }
      return clear;
    }

    void coordinateChange(int heading, int dist) {
      /**
         Modifies the internal coordinates.
         @param heading    The raw accelerometer heading.
         @param dist       The distance to translate.
      */
      current[0] += sin(heading * M_PI / 180) * dist;
      current[1] += cos(heading * M_PI / 180) * dist;
    }
```

```
    Adafruit_BNO055* bno; // Accelerometer object
};

class UltimateGPS : public Nav
/**
   @brief Reads data from a Adafruit Ultimate GPS.
   [!UNTESTED!]
*/
{
  public:
    UltimateGPS(byte RX, byte TX)
    /**
       Instantiates an UltimateGPS.
       @param: RX Receiving Pin
       @param: TX Transmitting Pin
    */
    {
      this->RX = RX;
      this->TX = TX;
    }

    void update()
    /**
       Makes a new navigation decision and acts on it.
       [!UNIMPLIMENTED!] - This function currently does not make any
                           decisions or even recording of the coordinats.
    */
    {
      if (adaGPS->newNMEAreceived() && !adaGPS->parse(adaGPS->lastNMEA())) {
        return; // Exit the function if the GPS fails to update
      }
    }

    bool connect()
    /**
       Connect to an Ultimate GPS over software serial.
       @return: Bool  Wheather or not the opperation succeded.
       [!UNTESTED!]
    */
    {
      bool success = true;
      gpsSerial = new SoftwareSerial(TX, RX);
      int waitTime = millis();
      while (!gpsSerial && success) {
        // Wait for gpsSerial or timout
        if (millis() - waitTime <= TIMEOUT) {
          success = false;
        }
      }
      if (success) {
        adaGPS = new Adafruit_GPS(gpsSerial);
        gpsSerial->begin(115200);
        adaGPS->begin(9600);
      }
      if (gpsSerial && adaGPS && success) {
        adaGPS->sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
        gpsSerial->println(PMTK_Q_RELEASE);
        gpsSerial->println(PMTK_SET_NMEA_UPDATE_1HZ);
        connected = success;
      }
      else {
        success = false;
      }
      connected = success;
      return success;
    }

    void disconnect()
    /**
       Disconnects from GPS.
    */
    {
      delete gpsSerial;
      delete adaGPS;
      connected = false;
    }
  private:
    byte RX;                        // GPS Receive Pin
    byte TX;                        // GPS Transmit Pin
    SoftwareSerial* gpsSerial;      // GPS Serial Port
    Adafruit_GPS* adaGPS;           // GPS Object
};
```

**G. 5_IR.ino**

```
/****************************************************************************
****

                                        IR

  IR contains classes for interfacing with thermal cameras and IR arrays and
for
  making movement decisions.  To use them, instantiate them and call their
  connect() function.  Include update() in your main loop.  Whenever update
is
  called, they will update their sensor if nessesary and make a movement
  decision.  The decision must be acted on by the main code.

  @file: 5_IR
  @author: Christian M. Schrader [cmschrader@wpi.edu]


*****************************************************************************
***/
class IR
/**
   @brief Base class used to interface with IR camras.
*/
{
  public:
    char cmd = FORWARD_ADVANCE;        // The most recent command decided on.
    bool connected = false;            // Indicates if the sensor is connected
and ready.
```

```
    virtual void update() = 0;         // Activates the sensor if it is due
and decides on a new cmd.
    virtual bool connect() = 0;        // Connects the sensor.
    virtual void disconnect() = 0;     // Disconnects the sensor.
    virtual float *getPixels() = 0;    // Get latest pixel readout
    virtual int getResolution() = 0;   // Get the resolution of the readout
    virtual float temp();              // GetAmbient tempurature
};

class GridEye : public IR
/**
   Reads data from a 8x8 IR GridEye sensor.
*/
{
  public:
    GridEye(int fps)
    /**
       Instantiates a GridEye IR camera.
       @param: fps  The frames per second to read the camera at.
    */
    {
      this->interval = 1000 / fps; // Convert FPS to millisecond interval.
    }

    bool connect()
    /**
       Connects to a GridEye via I2C
       @return: Bool  Whether or not the operaiton succeded.
    */
    {
      amg = new Adafruit_AMG88xx;
      updateTime = millis();
      connected = amg->begin();
      return connected;
    }

    void disconnect()
    /**
       Disconnects from the GridEye
    */
    {
      delete amg;
      connected = false;
    }

    void update()
    /**
       Updates the pixels and thermistor values if connected and it is due
for an update.
       Also makes a decision about which movement to make.
    */
    {
      // If it has been long enugh, update the sensor.
      if (connected && millis() - updateTime >= interval) {
        updateTime = millis();
        thermistorTemp = amg->readThermistor();
        amg->readPixels(pixels);
      }

      // Identify hottest pixel
      int maximum = 0;
      for (int i = 0; i <  getResolution(); i++) {
        if ( getPixels()[i] >  getPixels()[maximum]) {
          maximum = i;
        }
      }
      int maxcol = (maximum - 1) / int(sqrt(getResolution()));

      // Make Movement Decision
      if ( getPixels()[maximum] < IR_DETECTION_THRESHOLD) {
        if (gueseleft) {
          cmd = TURN_COUNTERCLOCKWISE;
        }
        else {
          cmd = TURN_CLOCKWISE;
        }
      }
      else if (maxcol < 3) {
        cmd = TURN_CLOCKWISE;
        gueseleft = true;
      }
      else if (maxcol > 4) {
        cmd = TURN_COUNTERCLOCKWISE;
        gueseleft = false;
      }
      else if ( colsSum(0, 7) < IR_MOVEMENT_THRESHOLD) {
        cmd = FORWARD_ADVANCE;
      }
      else {
        cmd = HALT;
      }
    }

    float *getPixels()
    /**
       @return A pointer to an array of tempurature floats for each pixel.
    */
    {
      return pixels;
    }

    float colsSum(int start, int end)
    /**
       Finds the sum of all temperatures in the pixels of a set of columns.
       @param start The start of the set.
       @param end   The end of the set.
       @return      The sum of the temperatures of every pixel in the
                    column set as a float.
    */
    {
      int sum = 0;
      int col;
      for (int i = 0; i < getResolution(); i++) {
        col = (i - 1) / int(sqrt(getResolution()));
```

```
        if (col >= start && col <= end) {
          sum += pixels[i];
        }
      }
      return sum;
    }

    float colsAvg(int start, int end)
    /**
     * Finds the average temperature of a set of columns.
     * @param start The start of the set.
     * @param end   The end of the set.
     * @return      The average temperature of all pixels in the set as
     *              a float.
     */
    {
      return colsSum(start, end) / (end - start + 1) /
int(sqrt(getResolution()));
    }

    int getResolution()
    /**
       @return Get the size of 1 row of the pixel array.
    */
    {
      return AMG88xx_PIXEL_ARRAY_SIZE;
    }

    float temp()
    /**
       Reads the on board thermistor.
       @return: The float temp in deg C.
    */
    {
      return thermistorTemp;
    }
  private:
    Adafruit_AMG88xx* amg;                   // IR cam object
    int updateTime;                          // Time the sensor was last
updated
    int interval;                            // Time between updates
    float thermistorTemp;                    // Last recorded thermistor temp
    float pixels[AMG88xx_PIXEL_ARRAY_SIZE]; // Last recorded pixel array
    bool gueseleft = true;                   // Whether or not left is the
best guess in absence of a visible heat source.
};
```

### H. 6_Suppression.ino

```
/*****************************************************************************
****

                                  Suppression

  Suppression is used to control the fire suppressing payload using the
  suppression class.  To use it, instantiate the class and call deploy.
  @file: 6_Suppression
  @author: Christian M. Schrader [cmschrader@wpi.edu]


*****************************************************************************
***/
class Suppression
/**
   @brief Base class used to interface with fire suppression systems.
*/
{
  public:
    void deploy() {
      /**
         Deploys the fire suppressing payload.
      */
      digitalWrite(SUPPRESSION_PIN, HIGH);
      delay(4000);
      digitalWrite(SUPPRESSION_PIN, LOW);
    }
};
```

### I. comms.ino

```
/*****************************************************************************
****

                                    comms

  comms contains functions used to communicate with the ground station. Both
ways
  utalize JSON objects to contain data.

  @file: comms
  @author: Christian M. Schrader [cmschrader@wpi.edu]


*****************************************************************************
***/

void readBluetooth()
/**
   Unpacks the data within a JSON string from Bluetooth.
   [!UNIMPLIMENTED!]
*/
{
  StaticJsonDocument<JSON_OBJECT_SIZE(1) * 4> doc;          // 7 is the
number of variables being received.
  DeserializationError err = deserializeJson(doc, "input"); // input is the
input string, should be the bluetooth.
  if (!err) {
    const char* stringvar = doc["stringvar"];
  }
  else {
    //Serial.print("JSON Error: ");
    //Serial.println(err.c_str());
  }
}

void report(float temp, bool fire, bool fireClose, int battery, bool
suppressant, float lat, float lon)
/**
   Returns the recorded data as a JSON packet to the base station over
Bluetooth.
   @param: temp         Temperature of the bot.
   @param: fire         Whether or not a fire can be identified.
   @param: fireClose    Whether or not a fire is in position to be
suppressed.
   @param: battery      0-100, battery percentage left.
   @param: suppressant  Whether or not suppressant can be deployed.
   @param: lat          GPS Latitude.
   @param: lon          GPS Longitude.
*/
{
  StaticJsonDocument<JSON_OBJECT_SIZE(1) * 7> doc; // 7 is the number of
variables being sent back.
  doc["temp"] = temp;
  doc["fire"] = fire;
  doc["fireClose"] = fireClose;
  doc["battery"] = battery;
  doc["suppressant"] = suppressant;
  doc["lat"] = lat;
  doc["lon"] = lon;
  serializeJson(doc, Bluetooth); // Outputs doc to Serial Bluetooth
}
```

### J. main.ino

```
/*****************************************************************************
****

                                      main

  main contains the main loop function, main setup, and the state
machine.  Its
  primary purpose is to control the objects created in other files.  The code
  for each state should be limited to a few lines to keep things
readable.  In the
  case that complex behaviour is needed, this should be put into a function
of the
  relevant class.

  @file: main
  @author: Christian M. Schrader [cmschrader@wpi.edu]


*****************************************************************************
***/

// Declare Objects
IR* ircam;
Nav* navigation;
Suppression* nozzle;
Thermistor* temp;
```

```
Chassis* movement;
RangeFinder* tofsensor1;
//RangeFinder* tofsensor2;
//RangeFinder* tofsensor3;

void setup()
{
  // Start Communication Channels and Pins
  Serial.begin(9600);        // Connects to the Standard Serial Port for
Debugging.
  Bluetooth.begin(9600);     // Connects to Bluetooth Serial.
  Wire.begin();              // Connects to i2c for motor.
  pinMode(FIRE_PIN, INPUT);
  pinMode(SUPPRESSION_PIN, OUTPUT);

  // Instantiate Objects
  ircam = new GridEye(IRFPS);
  tofsensor1 = new HCSR04(TRIGGER_PIN, ECHO_PIN, RANGERPS);
  movement = new OmniDrive();
  nozzle = new Suppression();
  navigation = new Accelerometer(movement, tofsensor1); //Replace with "new
UltimateGPS(GPSRX, GPSTX);"

  Serial.println("Setup Done");

  /*
     For debugging.  In the final version, this sohuld only ever be done
     by the receive command.
  */
  navigation->setTarget(2500, 2500);
}

void loop()
{
  readBluetooth();  // [!UNIMPLEMENTED!] Get commands from Bluetooth

  /*
     ACT ON RECEIVED COMMANDS
     [!UNIMPLEMENTED!]
     It is nessesary to link these to get data from readBluetooth().  This is
probably easiest done
     by creating an integer command array that readBluetooth() writes
to.  The first integer represets
     the received command (Using defined constants similar to the movement
code) while the other two
     can represent the arguments given to the command (such as the x an y
coordinates).
  */
  // Receive Coordinates
  if (false) { // If received command == set target coordinates
    //[!UNIMPLEMENTED!]
    // GPS_SET_COORDINATES();
    newCoordinates = true;
  }
  // Recall
  if (false) { // If received command == recall or TEMP > DANGER TEMP or
BATTERY < LOW BATTERY PERCENT
```

```
    //[!UNIMPLEMENTED!]
    // GPS_RECALL();
  }

  // Change States
  switch (state) {
    case STANDBY:
      if (newCoordinates) {
        newCoordinates = false;
        state = INIT;
        Serial.println("STATE: INIT");
      }
      break;
    case INIT:
      if (ircam->connected &&
          navigation->connected &&
          tofsensor1->connected) { // Add other system connection checks as
nessesary.
        state = TAKE_OFF;
        Serial.println("STATE: TAKE_OFF");
      }
      break;
    case TAKE_OFF:
      if (true) { // Replace condition with if ALTIMETER.READ() >=
CRUISE_ALTITUDE
        movement->transmit(HALT, 0, true);
        state = CRUISE;
        Serial.println("STATE: CRUISE");
      }
      break;
    case CRUISE:
      if (navigation->arrived()) {
        state = SEARCH;
        Serial.println("STATE: SEARCH");
      }
      break;
    case SEARCH:
      if (ircam->cmd == HALT ) { //&& digitalRead(FIRE_PIN)) { // This
digital read detects fire using the fire sensor
        state = SUPPRESS;
        Serial.println("STATE: SUPPRESS");
      }
      break;
    case SUPPRESS:
      if (true) { //Implement SUPPRESSANT_OUT()
        movement->transmit(ircam->cmd, UNIT_DISTANCE, true);
        navigation->returnToBase();
        state = RECALL;
        Serial.println("STATE: RECALL");
      }
      break;
    case RECALL:
      if (navigation->arrived()) {
        state = STANDBY;
        Serial.println("STATE: STANDBY");
      }
      break;
  }

  // State Actions
  switch (state) {
    case STANDBY:
      break;
    case INIT:
      ircam->connect();      // Add more connections as nessesary.
      navigation->connect();
      tofsensor1->connect();
      break;
    case TAKE_OFF:
      movement->transmit(ASCEND, UNIT_DISTANCE, true);
      break;
    case CRUISE:
      navigation->update();
      break;
    case SEARCH:
      if (millis() - lastMove > MOVE_INTERVAL) {
        lastMove = millis();
        movement->transmit(ircam->cmd, UNIT_DISTANCE, true);
      }
      break;
    case SUPPRESS:
      nozzle->deploy();
      break;
    case RECALL:
      navigation->update();
      break;
  }

  // Send Bluetooth Packet
  bool fire = false;
  bool fireClose = false;
  bool suppressant = true;
  if (state == SEARCH) {
    fire = true;
  }
  else if (state == SUPPRESS) {
    fire = true;
    fireClose = true;
    suppressant = false;
  }
  int battery = 100;
  report(ircam->temp(), fire, fireClose, battery, suppressant, navigation-
>getLon(), navigation->getLat());    // Report via bluetooth

  // Update
  ircam->update();
  tofsensor1->update();
}
```

### K. Base Station

import java.awt.BorderLayout;

```
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import com.fazecast.jSerialComm.SerialPort;

import javax.swing.GroupLayout;
import javax.swing.GroupLayout.Alignment;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JProgressBar;
import javax.swing.LayoutStyle.ComponentPlacement;
import javax.swing.JEditorPane;
import javax.swing.JButton;
import java.awt.Color;
import javax.swing.JTable;
import javax.swing.JTextArea;
import java.awt.SystemColor;
import javax.swing.UIManager;
import javax.swing.JTextPane;
import javax.swing.JFormattedTextField;
import javax.swing.JTextField;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.awt.event.ActionEvent;

public class GroundStationWb extends JFrame {

	private JPanel contentPane = new JPanel();
	private JTextField txtFieldConnection = new JTextField();
	private JTextField txtFieldTemp = new JTextField();
	private JTextField txtFieldBattery = new JTextField();
	private JTextField txtFieldDeployed = new JTextField();
	private JTextField textField_5 = new JTextField();
	private JTextField txtFieldState = new JTextField();


	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					GroundStationWb frame = new GroundStationWb();
					frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
	}


	/**
	 * Create the frame.
	 */

	public GroundStationWb() {
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setBounds(100, 100, 781, 677);
		contentPane = new JPanel();
		contentPane.setBackground(UIManager.getColor("ComboBox.buttonBackground"));
		contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
		setContentPane(contentPane);

		JLabel lblBattery = new JLabel("Battery: ");

		JLabel lblConnection = new JLabel("Connection:");

		JLabel lblCoordinates = new JLabel("Coordinates:");

		JEditorPane dtrpnLatitude = new JEditorPane();
		dtrpnLatitude.setText("Latitude");

		JEditorPane dtrpnLongitude = new JEditorPane();
		dtrpnLongitude.setText("Longitude");

		JLabel lblOutput = new JLabel("OUTPUT:");

		JLabel lblRobotTemp = new JLabel("Robot Temp:");

		JButton btnNewButton_1 = new JButton("GO");

		JLabel lblInput = new JLabel("INPUT:");

		JLabel lblSupressantDeployed = new JLabel("Suppressant Deployed:");

		JLabel lblHowMuch = new JLabel("Suppressant Remaining:");

		JLabel lblIrCam = new JLabel("IR CAM:");

		JLabel lblRobotState = new JLabel("Robot State:");

		// just GUI stuffs
		GroupLayout gl_contentPane = new GroupLayout(contentPane);
		gl_contentPane.setHorizontalGroup(
			gl_contentPane.createParallelGroup(Alignment.LEADING)
				.addGroup(gl_contentPane.createSequentialGroup()
					.addGroup(gl_contentPane.createParallelGroup(Alignment.LEADING)
						.addGroup(gl_contentPane.createSequentialGroup()
							.addContainerGap()
							.addComponent(lblCoordinates,
GroupLayout.PREFERRED_SIZE, 76, GroupLayout.PREFERRED_SIZE)
							.addPreferredGap(ComponentPlacement.RELATED)
```

```
                                                                        .addComponent(dtrpnLatitude,
GroupLayout.PREFERRED_SIZE, 118, GroupLayout.PREFERRED_SIZE)
                                                                        .addPreferredGap(ComponentPlacement.RELATED)
                                                                        .addComponent(dtrpnLongitude,
GroupLayout.PREFERRED_SIZE, 119, GroupLayout.PREFERRED_SIZE)
                                                                        .addGap(27)
                                                                        .addComponent(btnNewButton_1,
GroupLayout.PREFERRED_SIZE, 86, GroupLayout.PREFERRED_SIZE))
                                                                .addComponent(lblInput)
                                                                .addGroup(gl_contentPane.createSequentialGroup()

        .addGroup(gl_contentPane.createParallelGroup(Alignment.LEADING)
                                                                                .addComponent(lblOutput,
GroupLayout.PREFERRED_SIZE, 225, GroupLayout.PREFERRED_SIZE)

        .addGroup(gl_contentPane.createSequentialGroup()
                                                                                        .addGap(21)

        .addGroup(gl_contentPane.createParallelGroup(Alignment.LEADING)

        .addGroup(gl_contentPane.createSequentialGroup()

        .addComponent(lblRobotTemp)

        .addPreferredGap(ComponentPlacement.RELATED)

        .addComponent(txtFieldTemp, GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
GroupLayout.PREFERRED_SIZE))

        .addGroup(gl_contentPane.createSequentialGroup()

        .addPreferredGap(ComponentPlacement.RELATED)

        .addComponent(lblSupressantDeployed)

        .addPreferredGap(ComponentPlacement.RELATED)

        .addComponent(txtFieldDeployed, GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
GroupLayout.PREFERRED_SIZE))

        .addGroup(gl_contentPane.createSequentialGroup()

        .addPreferredGap(ComponentPlacement.RELATED)

        .addGroup(gl_contentPane.createParallelGroup(Alignment.LEADING)

        .addGroup(gl_contentPane.createSequentialGroup()

        .addComponent(lblRobotState)

        .addPreferredGap(ComponentPlacement.RELATED)

        .addComponent(txtFieldState, GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE,
GroupLayout.PREFERRED_SIZE))

        .addGroup(gl_contentPane.createSequentialGroup()
        .addComponent(lblHowMuch)

        .addPreferredGap(ComponentPlacement.RELATED)

        .addComponent(textField_5,        GroupLayout.PREFERRED_SIZE,        GroupLayout.DEFAULT_SIZE,
GroupLayout.PREFERRED_SIZE))))

        .addGroup(gl_contentPane.createSequentialGroup()

        .addPreferredGap(ComponentPlacement.RELATED)

        .addComponent(lblBattery)

        .addPreferredGap(ComponentPlacement.RELATED)

        .addComponent(txtFieldBattery,     GroupLayout.PREFERRED_SIZE,     GroupLayout.DEFAULT_SIZE,
GroupLayout.PREFERRED_SIZE))

        .addGroup(gl_contentPane.createSequentialGroup()
                                                                                .addGap(4)

        .addComponent(lblConnection)

        .addPreferredGap(ComponentPlacement.RELATED)

        .addComponent(txtFieldConnection,   GroupLayout.PREFERRED_SIZE,   GroupLayout.DEFAULT_SIZE,
GroupLayout.PREFERRED_SIZE)))))
                                                .addGap(30)
                                                .addComponent(lblIrCam)))
                                    .addContainerGap(309, Short.MAX_VALUE))
                  );
                  gl_contentPane.setVerticalGroup(
                        gl_contentPane.createParallelGroup(Alignment.LEADING)
                              .addGroup(gl_contentPane.createSequentialGroup()
                                    .addGap(5)
                                    .addComponent(lblInput)
                                    .addPreferredGap(ComponentPlacement.RELATED)
                                    .addGroup(gl_contentPane.createParallelGroup(Alignment.LEADING)
                                          .addComponent(btnNewButton_1,
GroupLayout.PREFERRED_SIZE, 38, GroupLayout.PREFERRED_SIZE)
                                          .addGroup(gl_contentPane.createSequentialGroup()

        .addGroup(gl_contentPane.createParallelGroup(Alignment.LEADING, false)
                                                  .addComponent(lblCoordinates,
GroupLayout.PREFERRED_SIZE, 30, GroupLayout.PREFERRED_SIZE)
                                                  .addComponent(dtrpnLongitude,
GroupLayout.DEFAULT_SIZE, 50, Short.MAX_VALUE)
                                                  .addComponent(dtrpnLatitude))
                                                .addGap(19)

        .addGroup(gl_contentPane.createParallelGroup(Alignment.BASELINE)
                                                  .addComponent(lblOutput)
                                                  .addComponent(lblIrCam))
                                                .addGap(18)
```

```
		.addGroup(gl_contentPane.createParallelGroup(Alignment.BASELINE)
								.addComponent(lblConnection)
								.addComponent(txtFieldConnection,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))

		.addPreferredGap(ComponentPlacement.UNRELATED)

		.addGroup(gl_contentPane.createParallelGroup(Alignment.BASELINE)
								.addComponent(lblRobotTemp)
								.addComponent(txtFieldTemp,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))

		.addPreferredGap(ComponentPlacement.UNRELATED)

		.addGroup(gl_contentPane.createParallelGroup(Alignment.BASELINE)
								.addComponent(lblBattery)
								.addComponent(txtFieldBattery,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))
							.addPreferredGap(ComponentPlacement.RELATED)

		.addGroup(gl_contentPane.createParallelGroup(Alignment.BASELINE)
								.addComponent(txtFieldDeployed,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
								.addComponent(lblSupressantDeployed))

		.addPreferredGap(ComponentPlacement.UNRELATED)

		.addGroup(gl_contentPane.createParallelGroup(Alignment.LEADING)
								.addComponent(textField_5,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
								.addComponent(lblHowMuch))

		.addPreferredGap(ComponentPlacement.UNRELATED)

		.addGroup(gl_contentPane.createParallelGroup(Alignment.BASELINE)
								.addComponent(lblRobotState)
								.addComponent(txtFieldState,
GroupLayout.PREFERRED_SIZE, GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))))
						.addContainerGap(342, Short.MAX_VALUE))
			);
			contentPane.setLayout(gl_contentPane);


			/**
			 * Open Connector parts.
			 */
			while (true)
			{

			JSONParser parser = new JSONParser();
			for (SerialPort port : SerialPort.getCommPorts()) {
				System.out.println(port.getPortDescription());
			}
			SerialPort sp = SerialPort.getCommPort("COM5");
			sp.setComPortParameters(9600, 8, 1, 0);
sp.setComPortTimeouts(SerialPort.TIMEOUT_READ_BLOCKING, 0, 0);

if (sp.openPort()) {
	System.out.println("Connected to" + sp.getPortDescription());
} else {
	System.out.println("Failed to connect to " + sp.getPortDescription());
	return;
}

int received = 0;
char receivedChar;
String line = "";
int temp = 0;
int battery  = 0;
int botState = 0;
int suppressant = 0;
int connect = 0;

// try catch finally
try {
	received = sp.getInputStream().read();
}
catch (IOException e) {

}
finally {
	line = "";
}

//tells variables what they are, grabbing them from json)
if (received != -1) {
	receivedChar = (char) received;
	if (receivedChar == '\n') {
		System.out.println(line);
		try {
			Object obj = parser.parse(line);
			JSONObject jsonObject = (JSONObject) obj;

			temp = (int) jsonObject.get("temp");
			// long temp = (long) jsonObject.get("temp");
			// System.out.println("Temperature is: " + temp);

			battery = (int) jsonObject.get("battery");
			// long battery = (long) jsonObject.get("battery");
			// System.out.println("Battery Level is: " + battery);

			suppressant = (int) jsonObject.get("suppressant");
			// boolean suppressant = (boolean) jsonObject.get("suppressant");
			// System.out.println("Suppressant Deployed: " + suppressant);

			botState = (int) jsonObject.get("robotState");
			// String botState = (String) jsonObject.get("robotState");
			// System.out.println("Robot State: " + botState);

			connect = (int) jsonObject.get("connection");
			// String connect = (String) jsonObject.get("connection");
```

```
                                        // System.out.println("Connection: " + connect);

                                        long lat = (long) jsonObject.get("lat");
                                        // System.out.println("Latitude: " + lat);

                                        long lon = (long) jsonObject.get("lon");
                                        // System.out.println("Longitude: " + lon);

                                } catch (ParseException e1) {
                                        e1.printStackTrace();
                                }finally {
                                        line = "";
                                }
                        } else {
                                line += receivedChar;



                //String connect = (String) jsonObject.get("connection");
                // Labels the blank spots in the gui fields using the variables set by json get
                txtFieldConnection = new JTextField(connect);
                txtFieldConnection.setColumns(10);
                txtFieldConnection.setEditable(false);

                txtFieldTemp = new JTextField(temp);
                txtFieldTemp.setColumns(10);
                txtFieldTemp.setEditable(false);

                txtFieldBattery = new JTextField(battery);
                txtFieldBattery.setColumns(10);
                txtFieldBattery.setEditable(false);

                txtFieldDeployed = new JTextField(suppressant);
                txtFieldDeployed.setColumns(10);
                txtFieldDeployed.setEditable(false);

                textField_5 = new JTextField();
                textField_5.setColumns(10);
                textField_5.setEditable(false);

                txtFieldState = new JTextField(botState);
                txtFieldState.setColumns(10);
                txtFieldState.setEditable(false);

}
        }
}
        }
}
```

### L. Fire Suppression Trade Study

| Trade Study | | | Ranking Description | Sprayer | Baby Bambi | Water Bubbles |
|---|---|---|---|---|---|---|
| **Criteria:** | **Mandatory (Y=1/N=0)** | **Weight** | **SCALE** | | | |
| **Reaches Fire Base:** | 1 | 20 | 3 = Most, 1 = least | 3 | 2 | 3 |
| **Targeted:** | 1 | 10 | 3 = Most, 1 = least | 3 | 2 | 2 |
| **Multishot:** | 0 | 20 | 3 = Most, 1 = least | 3 | 1 | 2 |
| **Ease of Refill:** | 1 | 10 | 3 = Easiest, 1 = More difficult | 2 | 2 | 1 |
| **Clog:** | 0 | 5 | 3 = not likely, 1 = likely | 1 | 3 | 2 |
| **Cost:** | 0 | 5 | 3 = less expensive, 1 = expensive | 2 | 3 | 1 |
| **Weight:** | 1 | 20 | 3 = light, 1 = heavy | 2 | 3 | 2 |
| **proximity to heat:** | 0 | 10 | 3 = farthest, 1 = closest | 1 | 3 | 3 |
| **Total:** | | | | 235 | 220 | 215 |
| **Weighted Totals in %:** | | **100** | **Percent:** | 78.333333 | 73.3333333 | 71.6666667 |