High Altitude Weather Balloon Launch IV for Measuring Environmental Pollution

Major Qualifying Project

A Major Qualifying Project Report Submitted to the faculty of the Electrical and Computer Engineering Department at Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science

Submitted by:

Connor Borsari

Nicholas Chantre

Jonathan Lopez

Drew Solomon

Advised by:

Professor Gregory Noetscher

March 24, 2023

Table of Contents

Abstract

As the levels of greenhouse gas pollution rise globally, global warming continues to affect ecosystems, societies, and negatively impact human health. To better monitor and aid in the effort to mitigate the effects of environmental pollution, our team designed and constructed a high-altitude balloon system capable of traveling to the lower regions of Earth's upper atmosphere and recording sensor data on greenhouse gas levels. Our system makes use of an electronic payload that collects and transmits greenhouse gas levels among other data to a base station in real time.

Acknowledgements

The HAB 4.0 team would like to thank Professor Gregory Noetscher for his guidance in the process of designing and executing this project. His advice was essential in making core decisions on new functionality, keeping the project pace, and continually testing and evaluating our system. The team wishes him the best in any future final projects he advises at WPI.

The team would also like to thank William Appleyard for assisting with the acquisition of hardware, and for supplying necessary tools and machinery.

Finally, the team would also like to acknowledge the help we received from Alexander Kobsa and Madeline Kasznay of the HAB 3.0 team. Throughout the project they were both available and willing to share their experiences working through the project themselves and gave us valuable insight going into the next iteration.

Executive Summary

At the center of this project was the 4th Generation of the High-Altitude Balloon with a payload full of sensors capable of collecting data from the atmosphere. High Altitude Balloons, or HABs for short, are used all over the aerospace industry for taking data related to weather and climate. It has been this project's long-term goal to use a HAB to collect data specifically related to climate change in the New England area. In order to achieve this and collect meaningful data at altitudes ranging from 20,000 to 80,000 feet the HAB payload needs to be robust and reliable. As previously stated, this is the fourth generation of this project and with each generation a new set of improvements was made. Previous generations have focused heavily on selecting proper payload sensors to take readings on things such as ultraviolet light and carbon dioxide as well as collecting data with said sensors. HAB4.0 took an alternative path and made it the team's top priority to improve the overall quality and functionality of the system. These improvements were organized into three overarching project goals. The first of these goals was the addition of an early flight termination system, the second was to enable the transfer of images from the payload to the base station during a flight and the third being a complete reconstruction of the base station with payload tracking capabilities.

Authorship

Section	Primary Author
Abstract	Nicholas
Acknowledgements	Drew
Executive Summary	Connor
Introduction	Drew & Nicholas
Background	Drew
Pre-Flight Major Objectives	Drew
Development of the Early Termination System (ETS)	Connor
ETS Computer System and Software (3.3.1-8)	Drew
Image Transmission (3.3.9)	Nicholas
Base Station Development	Jonathan & Nicholas
Results	Drew, Jonathan & Nicholas
Conclusion & Recommendations	Jonathan & Nicholas
References	Drew
Appendices	Drew & Connor

Table of Figures

List of Tables

Introduction

1.1 Significance

High Altitude Weather Balloon 4.0 (HAB 4.0) is the 4th iteration of a senior design project to record the presence of harmful greenhouse gases (GHG) within the Earth's atmosphere. Historically, this project has contributed to a larger environmental effort to obtain extensive data monitoring of changes to greenhouse gas levels. In this iteration of the project, our team's goal was to build upon the existing systems developed and integrated by prior teams and design and develop meaningful contributions of our own. Upon meeting with the HAB 3.0 Team and discussing the state of the project, we found the following to be the main areas for improvement: strengthening the system's communication link, designing a retrieval system, and adding the ability for our payload to capture and transmit images during flight. As a result, our team collectively conducted research and designed systems to achieve these objectives and new functionalities to ensure the collection of environmental data. In summary, we contributed the following functionalities: images can now be communicated from the air to the ground, the base station used to receive data from the HAB in flight was overhauled to adjust its antenna towards the payload, an Early Termination System (ETS) was implemented to end the flight early should the balloon fly further than expected. These improvements will open new possibilities in what data can be gathered during the flight, while providing stronger and more reliable communication, and ensuring the retrieval of the payload post-flight.

1.2 Scope

As it ascends into the atmosphere, HAB 4.0 collects data on the presence of greenhouse gases. It also records its own GPS location and transmits all this information to the grounded base station in real-time. The payload is equipped with a carbon dioxide, ozone, and methane sensor, as well as a GPS module, and radio which form our main data collection and transmission system.

To achieve the wireless transmission of images from our payload to the ground station, we integrated a camera module into our payload system and wrote and implemented the necessary software to periodically capture, encode and transmit images taken throughout our flight. For the image encoding of our captured images, we made use of an existing image encoding library that would compress images to allow for our system to transmit them.

Our base station is a key component to this project as it serves as the collection center for all data captured and transmitted by our payload. In this iteration of the project, we wrote software to better organize our recorded data into a excel file that can be more easily parsed and interpreted for our analysis and added the respective software to decode and store the transmitted images. In addition to the software redesign, we constructed a new base station now equipped with the ability to autonomously turn to point towards the payload as it moves through the sky.

Implementing the ETS required both new software and hardware, using a MOSFET trigger paired with a nichrome wire heater to melt through the payload balloon connection line. On account of the complexity of the project, a start-up guide was created in order to help future HAB teams get acquainted with the project's current capabilities, components, ideas for future improvements, and directions for proper maintenance.

Initially the project was intended to have at least one flight as each year before had done, but the rising cost of helium [1] unfortunately prevented the team from flying even once. Since helium is a

non-renewable resource and worldwide supplies of it are running thin, the cost has increased rapidly over the last few years, Figure 1 below displaying the current price per cubic foot in several locations around the world [1].

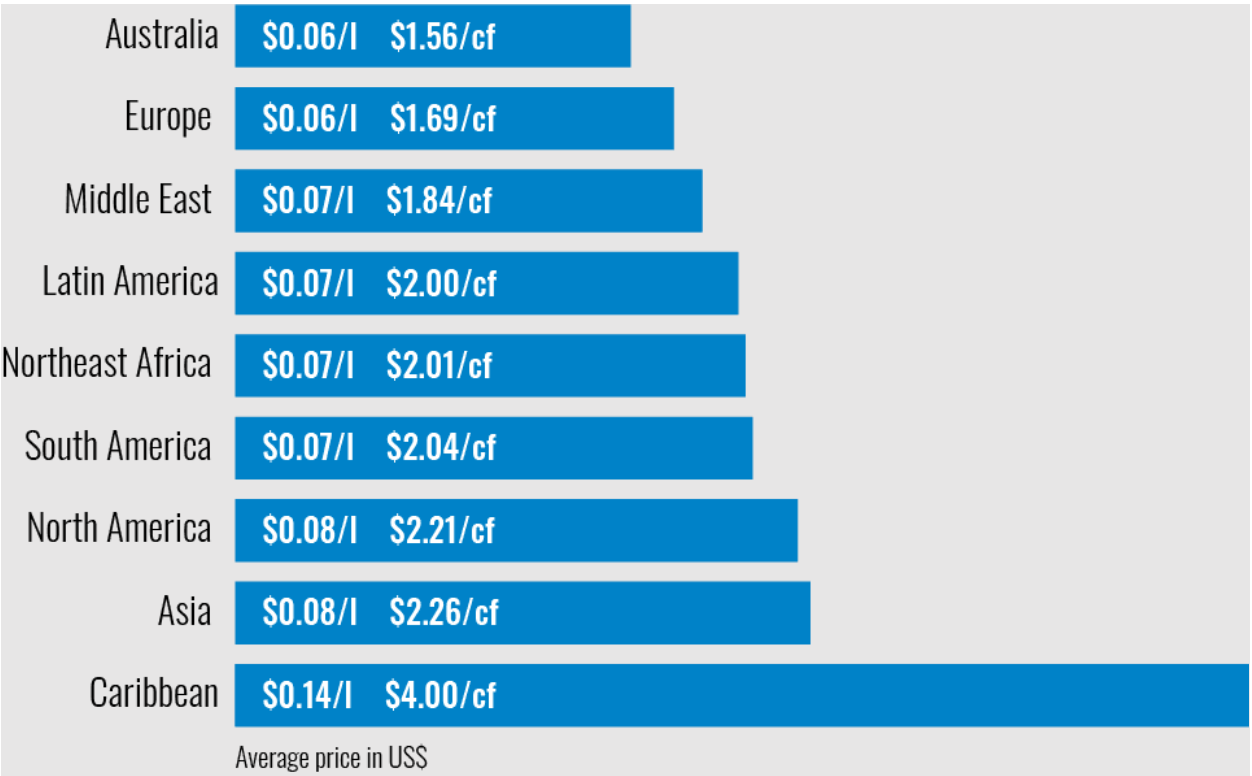


Figure 1: Worldwide Cost of Helium [1]

With the project needing around 250 cubic feet of helium for a single flight, the extremely high price of helium in North America as seen in Figure 1 prevented a flight from occurring during this project year. The team decided that it would be more worthwhile to make advancements and set the framework such that future years could fly rather than have our own flight without adding anything to the project. The remainder of the team budget at the end of the project was used to purchase equipment and supplies for future years to use so they could focus their budget on acquiring helium in the face of rising costs.

Background

2.1 The Current State of the Atmosphere

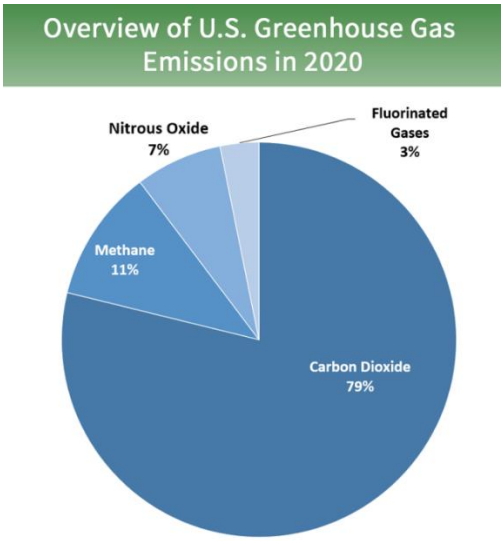


Figure 2: Overview of Greenhouse Gas Emissions in 2020 [2]

The majority of greenhouse gas emissions consist of three main gases: carbon dioxide, methane, and nitrous oxide, as seen in Figure 2. Carbon dioxide is easily the most common, taking up 79% of all emissions, with methane in 2nd place at 11% and nitrous oxide in 3rd at 7% [2]. Fluorinated gases, while still contributing to the problem of global warming, are not present nearly as much as others listed [2].

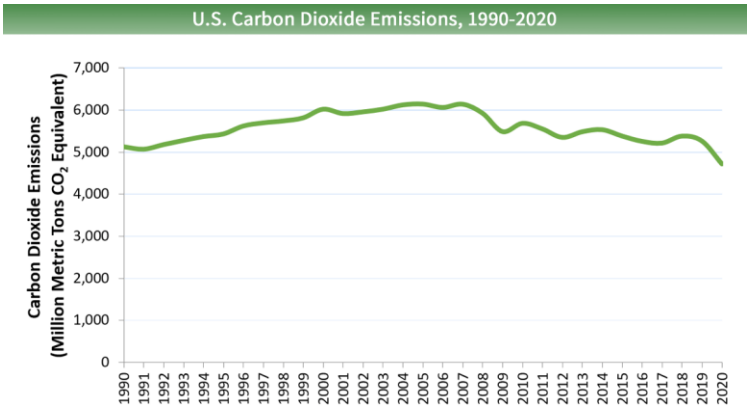


Figure 3: U.S. Carbon Dioxide Emissions, 1990-2020 [2]

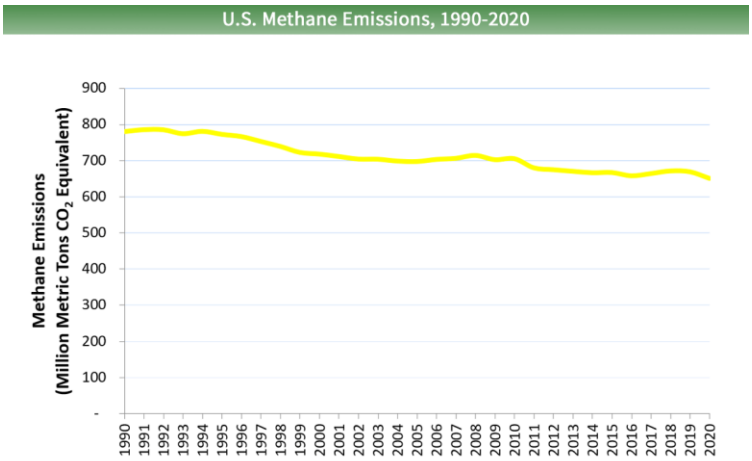


Figure 4: U.S. Methane Emissions, 1990-2020 [2]

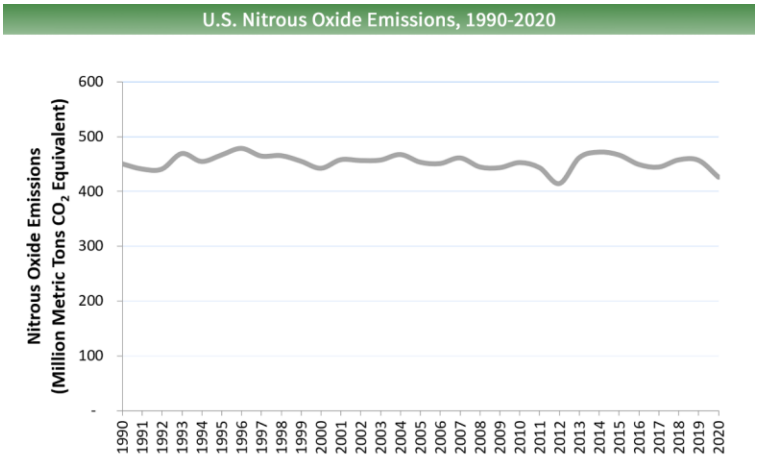


Figure 5: U.S. Nitrous Oxide Emissions, 1990-2020 [2]

As seen in Figures 2, 3, and 4, in the United States emissions of the three major greenhouse gases have either been steady or in a gradual decline.

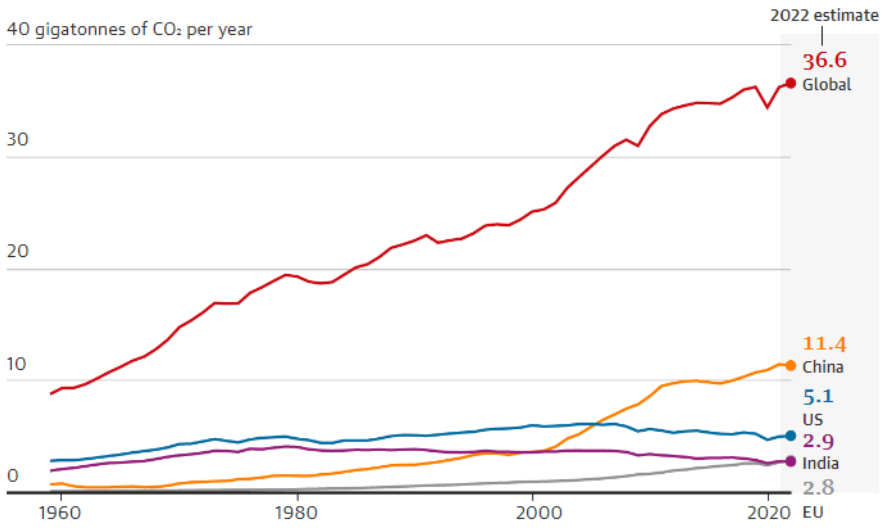


Figure 6: Global Carbon Emissions, 1960-2022 [3]

Taking a look at carbon emissions (the most common by far) on the global scale, the situation is beginning to look more dire. Since 1960, the world's collective carbon emissions have been on a steady rise with no sign of stopping [2]. Some large contributors have been able to lower their rates over the years such as the United States and India, but China and the EU have been greatly rising especially over the past 20 years [2]. The task of reducing emissions is not simple and has direct economic implications, but if action isn't taken the Earth's climate may be irreversibly damaged.

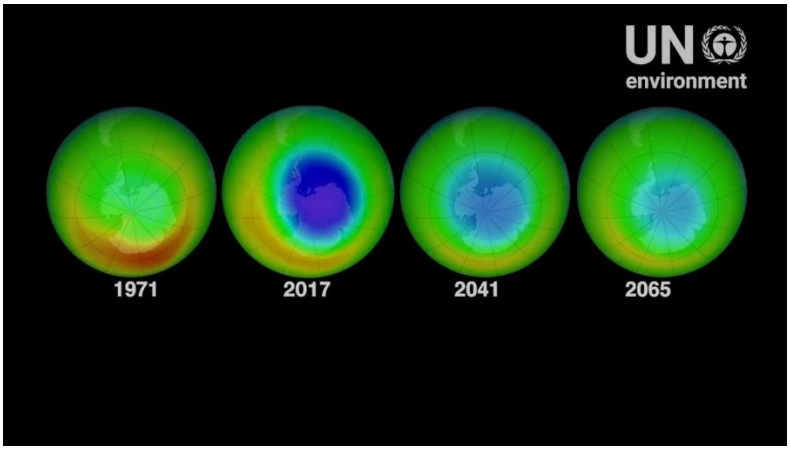


Figure 7: Predicted Healing of the Ozone Layer, 1971-2065 [4]

Aside from greenhouse gases, another gas of interest to this project is ozone. The ozone layer of the atmosphere protects the Earth from harmful UV rays and was in danger of being destroyed up until the worldwide Montreal Protocol was established in 1987 to save it [4]. As seen in Figure 7, a hole in the ozone layer began forming over Antarctica, but due to the guidelines established in the Montreal Protocol, the ozone layer has begun to heal and is on the way to a complete recovery within our lifetime [4].

2.2 The Flight of HAB 3.0

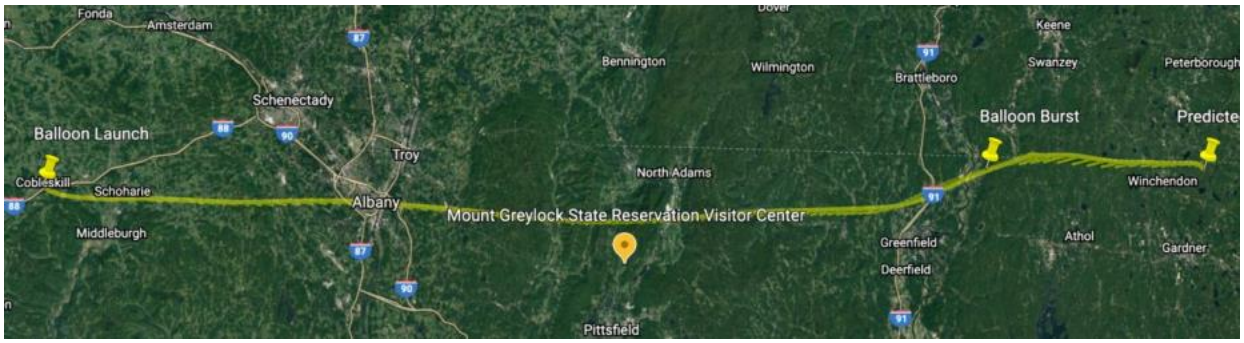


Figure 8: HAB 3.0 Predicted Flight Path, 12/5/21 [5]

For their 2nd launch of the year in December 2021, the HAB 3.0 team had predicted the flight path of the payload using an online tool. As seen in Figure 8, the flight was predicted to stretch no further than Peterborough, MA from their launch point of Cobleskill, NY [5]. Unfortunately, the weather forecast incorrectly predicted the air conditions of the flight day, and the payload was blown all the way into the Atlantic Ocean [5].

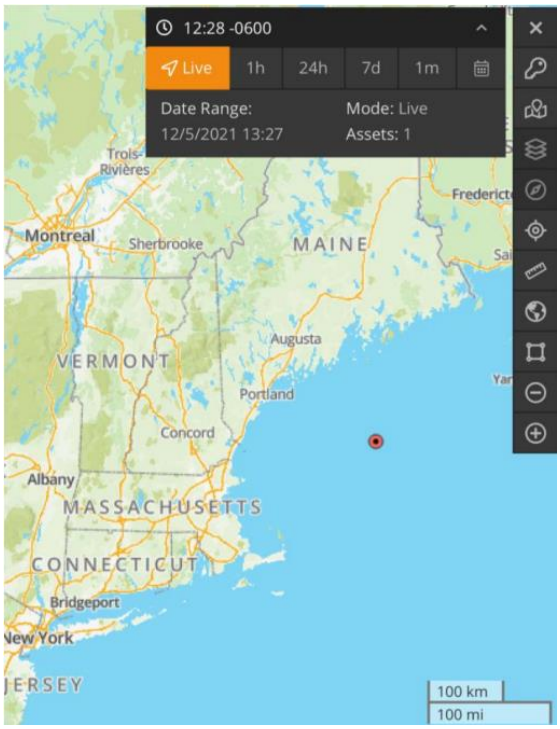


Figure 9: HAB3.0 Payload Landing Site, 12/5/21 [5]

The previous iteration of the HAB project, HAB 3.0, was equipped with a GPS module to track its location (Figure 9) but the coordinates were not used for anything other than tracking the payload in flight [5]. In order to prevent such a loss from occurring again, our team decided to utilize the GPS module a step further and implement it as part of the Early Termination System. With the outcome of HAB 3.0's last flight in mind, we wanted to make a system which would stop the payload from approaching the coast, no matter the weather conditions on the day of flight.

Methodology

3.1 Pre-Flight Major Project Objectives

Given the shortcomings of HAB 3.0 and its final flight, our team decided that the most important developments to make this year were the following:

1. Create an Early Termination System to ensure payload retrieval

2. Enable transfer of images from the payload to the base station during flight

3. Reconstruct the base station and develop payload tracking capability

3.2 Development of the Early Termination System (ETS)

3.2.1 Why HAB 4.0 Needed an ETS

During the second HAB 3.0 flight test, in December of 2021, the payload was never recovered as it touched down in the Atlantic Ocean. This complication occurred as a result of a delayed test date which exposed the HAB to much higher windspeeds. Despite the team moving its launch site west to accommodate for the troubling predictions made by the flight path simulator, the HAB still traveled further than expected and was lost. This unfortunate sequence of events exposed a major vulnerability which would need to be remedied in future HAB iterations in order to reduce the potential of losing another payload. The HAB 4.0 team decided that these preventative measures were of the utmost importance given the limited budget allotted to this project. Additionally, a significant amount of time was spent in the early stages of the HAB 4.0 project returning the replacement payload that the HAB 3.0 team had begun working on to a functional state which impacted the amount of time that our team could commit to the development of new features and optimizations. These delays were a direct result of the previous payload being lost, further emphasizing the need for a system that will prevent additional loss of technology.

Based on recommendations given by the HAB 3.0 team, the HAB 4.0 team was quickly able to identify one plausible remedy to the issue of losing payloads as a result of unpredictable weather. This remedy would be the development and installation of an Early Termination System (ETS) that would allow for automatic termination of flight in the event of that the HAB begins to move too far away from its predicted flight path. This system would need a way of continuously monitoring its location in relation to some pre-defined boundaries and have the hardware needed to instantaneously pop the latex balloon carrying the payload without damaging and of the other technology on board including the parachute system. The team divided this system into two distinct sections, including the positioning related software and the balloon popping hardware. HAB 3.0 already included a GPS module which meant that the bulk of the software side would be related to defining boundaries and establishing a response to the HAB exiting these boundaries. Ideally the ETS would be triggered if the HAB were to fly too close to any significant body of water.

3.2.2 Initial Calculations and Considerations

The HAB 3.0 team had initially played with the idea of using nichrome wire [5] to pop the balloon, so the team chose to push forward with this concept and develop a prototype system which would use nichrome at its core. Nichrome wire is a particular type of wire consisting of nickel and chromium which has a very high resistivity. For example, a single foot of nichrome 80 wire in 16 AWG has a resistance of 0.2499Ω compared to the 0.004Ω that would be expected from a foot of 16 AWG copper wire [7]. This property of high resistance makes nichrome wire ideal for use in coiled heaters. These heaters often appear in things such as hair dryers and can produce a range of temperatures. One fairly low risk way of popping the balloon in the event of a deviation from the flight plan is to use this nichrome wire to heat a portion of the balloon to its melting point. One of the immediate concerns which developed in the early stages of the ETS development was the change in temperature with increased elevation. As temperature decreases, materials tend to experience consistent shifts in their resistivity which could affect a nichrome heater's ability to reach the proper temperature to pop a balloon. In order to alleviate these concerns, the following calculations were performed using a formula which relates resistivity to temperature.

$$R = Rref[1 + \alpha(T - Tref)]$$

(1)

$$Peak\ elevation = 32km \rightarrow T = -44.5℃$$

$$Temperature\ coefficient\ of\ nichrome\ wire \rightarrow\ \alpha = 0.00017$$

$$ohms\ per\ foot\ of\ 26AWG\ nichrome\ 60\ wire \rightarrow Rref =\ 2.670\Omega$$

$$Rref\ measurent\ temp \rightarrow Tref = 20℃$$

$$R\ @\ peak\ elevation = 2.64\Omega$$

$$Change\ in\ resistance = 0.03\Omega$$

From the calculations above, it can be seen that the resistance of a single foot of 26AWG nichrome wire will only drop 0.03Ω at the highest possible point that the balloon could reach. For the purpose of this heater, 2.845 feet of nichrome wire will be used (see page 15 for more information) so the total drop in heater resistance will be roughly 0.09Ω. Given that heater temperature is directly related to the current, this subtle drop in resistance will increase the output temperature of the heater. As long as this increase does not draw excessive current from the DC power supply this increase in heat will only help to ensure that the latex melting point is reached, and the balloon is popped. With this concern elevated it was possible to move forward in the design process of the ETS using nichrome wire.

Apart from the ETS's ability to effectively pop the balloon, it was essential to confirm that this system could be effectively powered by the DC power supply included in the payload. The TalentCell 24V Lithium-ion Battery used in this iteration of the payload is advertised as being able to provide 22400mAh which is a tight budget considering that this supply is responsible for powering every piece of onboard electronics. An initial power consumption analysis was performed on the systems included onboard the HAB 3.0 payload to identify a concrete power budget. The table below summarizes the current draw of all the components included in the HAB 3.0 payload.

HAB 3.0 Payload Current Consumption

Payload Component	**Current Consumption (mA)**
Raspberry Pi	3000
Arduino Uno	500
GoPro Camera	717.65
GPS Antenna Module	150
JBtek BMP180	1
UV Light Sensor	1
Gravity Analog Co2 Gas Sensor	200
Gravity I2C Ozone Sensor	6.5
MICS-2714 NO2 Sensor	26
Xbee Radio	229
Total	**4831.15**

Table 1: Per Component Payload Current Consumption

With the knowledge of the HAB 3.0 payload total current draw the following calculation were done.

$$Worst\ case\ flight\ time\ (based\ on\ previous\ years) = 4\ hours$$

$$mAh = mA * h \rightarrow 4831.15mA * 4hours = 19324.6mAh$$

$$Total\ required\ system\ mAh \approx\ 19500mAh$$

$$Budget = DC\ supply\ rating - Total\ used\ mAh \rightarrow 22400mAh - 19500mAh = 2900mAh$$

(1)

$$Budget\ for\ ETS = 2900mAh$$

It should be noted that the total current draw of the payload varied as the payload systems were continuously improved and modified by the HAB 4.0 team. This calculation was done in order to identify a general budget for the ETS as well as any other additional current consuming elements that could be added to HAB 4.0. Identifying this ETS mAh budget was an essential step in the development of the ETS and with this information the team was able to move on to the selection of components that would be used in this system.

3.2.3 Nichrome Wire Selection

In terms of electrical components, the ETS ultimately included a nichrome wire heating element, a power MOSFET, a fuse, and several resistors. At the core of all these components is the nichrome heating element and it was essential to first select the specific nichrome wire which would be used in order to select all the other components listed above. It was identified that there were three major constraints that would play into the selection of a variety of nichrome - these constraints are listed below:

1. Power supply output voltage
2. Power supply maximum current draw
3. Physical space required for heating element

The previously mentioned TalentCell 24V Lithium-ion Battery included three possible voltage output options. These options include two coaxial power connector outputs capable of producing 24V or 12V and a singular 5V USB output. Each of these outputs has a unique maximum current draw. The HAB 3.0 payload was currently making use of the 5V USB output to power the entire existing system and the team decided to keep this for the HAB 4.0 payload. In order to keep the ETS isolated from the rest of the payload system the usable options were limited to the two coaxial power outputs. The 12V coaxial output has a maximum current draw of two amps while the 24V coaxial output has a maximum of 3.5A. For the sake of keeping both the current and the voltage on the lower side in the ETS, increasing overall system safety, it was decided that the ETS would make use of the 12V coaxial power connector. In order to not exceed the 2A maximum current draw of this output a nichrome wire that could reach 347°F, the melting point of the latex balloon, at a current below this limit. Relating current through a given type of nichrome to a specific output temperature was identified to be a complicated material science problem but this was luckily circumvented through the use of the online resource document, found on wiretron.com [7], specifically developed to aid in designing nichrome heaters. The specific table used to relate temperature to current is shown below.

CURRENT TEMPERATURE CHARACTERISTICS OF
NICHROME 60
STRAIGHT WIRE

Showing approximate amperes necessary to produce a given temperature.
Applying only to straight wires stretched horizontally in free air.

No. B.& S.	Diam. Inches	Temp.400°F Temp.204°C	600 316	800 427	1000 538	1200 649	1400 760	1600 871	1800 982	2000° F. 1093° C.
1	.289	75.10	104.00	134.0	169.0	210.0	257.0	309.0	363.0	417.0
2	.258	62.81	87.45	112.4	142.1	176.5	215.9	259.5	304.9	350.3
3	.229	52.54	73.53	94.33	119.4	148.5	181.4	217.9	256.0	294.2
4	.204	43.94	61.82	79.15	100.4	124.8	152.4	183.0	215.0	247.2
5	.182	36.75	51.98	66.41	84.36	104.9	128.0	152.7	180.6	207.6
6	.162	30.74	43.72	55.72	70.91	88.16	107.5	129.0	151.7	174.3
7	.144	25.71	36.75	46.75	59.60	74.12	90.35	108.3	127.4	146.4
8	.128	21.50	30.90	39.23	50.10	62.30	75.90	91.00	107.0	123.0
9	.114	18.00	25.90	33.20	42.10	52.40	64.20	76.80	90.20	104.0
10	.102	15.43	22.03	28.15	35.63	44.30	54.39	65.08	76.40	88.00
11	.091	13.23	18.72	23.89	30.16	37.54	46.07	55.15	64.70	74.4
12	.081	11.34	15.91	20.27	25.53	31.77	39.03	46.73	54.80	63.01
13	.072	9.73	13.53	17.21	21.61	26.89	33.06	39.60	46.41	53.31
14	.064	8.34	10.50	14.59	18.30	22.76	28.01	33.56	39.31	45.11
15	.057	7.15	9.78	12.38	15.50	19.26	23.73	28.44	33.30	38.17
16	.051	6.13	8.31	10.50	13.11	16.30	20.10	24.10	28.20	32.30
17	.045	5.31	7.18	9.13	11.30	13.90	16.90	20.30	23.60	27.00
18	.040	4.66	6.26	7.90	9.75	11.96	14.51	17.37	20.48	23.08
19	.036	4.09	5.46	6.84	8.41	10.30	12.45	14.87	17.78	19.73
20	.032	3.58	4.77	5.92	7.25	8.86	10.69	12.72	15.43	16.87
21	.0285	3.14	4.16	5.13	6.26	7.63	9.17	10.88	13.40	14.40
22	.0253	2.76	3.63	4.44	5.40	6.56	7.87	9.31	11.63	12.33
23	.0226	2.42	3.16	3.84	4.67	5.65	6.76	7.97	10.09	10.54
24	.020	2.12	2.76	3.32	4.01	4.86	5.80	6.82	8.76	9.01
25	.0179	1.84	2.42	2.90	3.44	4.15	4.97	5.86	6.96	7.72
26	.0159	1.58	2.09	2.52	3.00	3.61	4.31	5.06	5.97	6.63
27	.0142	1.37	1.80	2.19	2.62	3.14	3.73	4.37	5.12	5.69
28	.0126	1.18	1.55	1.90	2.28	2.73	3.23	3.77	4.39	4.88
29	.0113	1.02	1.34	1.65	1.99	2.37	2.80	3.25	3.76	4.19
30	.010	.875	1.16	1.43	1.74	2.06	2.43	2.81	3.22	3.59
31	.0089	.754	1.00	1.24	1.51	1.79	2.10	2.42	2.76	3.09
32	.008	.650	.860	1.08	1.32	1.56	1.82	2.09	2.37	2.65
33	.0071	.560	.750	.940	1.13	1.34	1.56	1.80	2.04	2.26
34	.0063	.493	.658	.820	.978	1.16	1.34	1.54	1.74	1.93
35	.0056	.434	.577	.715	.846	1.01	1.16	1.32	1.49	1.66
36	.005	.382	.506	.623	.732	.872	.998	1.13	1.27	1.42
37	.0045	.337	.444	.543	.633	.755	.860	.970	1.09	1.21
38	.004	.296	.390	.474	.548	.654	.741	.831	.931	1.04
39	.0035	.261	.342	.413	.474	.567	.638	.712	.796	.888
40	.0031	.230	.300	.360	.410	.491	.550	.610	.680	.760

Table 2: Nichrome 60 Wire Current Temperature Characteristics [7]

The table above relates current to temperature ranges for specific nichrome 60 wire gauges. It should be noted that there are two varieties of nichrome wire which are commonly used in heater elements, these varieties include nichrome 80 and nichrome 60. Both variations of wire are similar with the major difference between them being that nichrome 60 has a slightly higher resistivity and corrosion resistance, making it more expensive. There was more abundant data in this document related to nichrome 60 and when this was combined with its slightly more desirable characteristics it made the most sense to select nichrome 60 for use in the ETS. It can be seen in the leftmost column that different Gauges of nichrome are listed followed by the wire diameter in the next column and the various temperature ranges in the remaining columns. Under each temperature range is the required current in Amps that must be flowing through that specific gauge of nichrome 60 to enter that range. Given the previously discussed 2A limit on the 12V coaxial power connector it was determined that a current of 1.58A would be safe. A nichrome wire gauge of 26 AWG was selected on the table by cross referencing the required heating range of 300°F to 400°F to pop the balloon with the safe current draw of 1.58A. The following calculations were done to identify the total run time of the ETS based on the mAh budget calculated earlier.

$$Run\ time = \frac{mAh\ budget}{ETS\ current\ draw} \rightarrow \frac{2900mAh}{1580mA} = 1\ hour\ and\ 50\ minutes$$

(2)

Ideally the ETS would only ever need to run for a few minutes to pop the balloon, but this large maximum run time is an excellent indication of how using this gauge of nichrome and the selected current of 1.58A will have minimal impact of the remaining mAh budget for other additions made to the HAB 4.0 payload. This gauge and variety of nichrome satisfies constraints 1 and 2 but does not consider constraint 3. With a constant supply voltage, the size of the heater coil depends purely on the resistance of a singular foot of 26 AWG nichrome 60 wire. The resistance of this variety of nichrome was extracted from another table found in the resource document [7] which is shown below.

NICHROME 60 WIRE

AWG	DIAMETER IN INCHES	OHMS/FOOT AT 68° F.	LBS. PER 1000	FEET PER POUND
8	.128	.04120	46.00	21.74
9	.114	.05194	36.49	27.40
10	.102	.06488	29.21	34.23
11	.091	.08151	23.25	43.01
12	.081	.1029	18.42	54.29
13	.072	.1302	14.56	68.68
14	.064	.1648	11.50	86.96
15	.057	.2078	9.123	109.6
16	.051	.2595	7.304	136.9
17	.045	.3333	5.686	175.9
18	.040	.4219	4.493	222.6
19	.036	.5208	3.639	274.6
20	.032	.6592	2.875	347.8
21	.0285	.8310	2.281	434.4
22	.0253	1.055	1.797	556.5
23	.0226	1.322	1.434	697.4
24	.0201	1.671	1.134	881.8
25	.0179	2.107	.8997	1,111.
26	.0159	2.670	.7099	1,413.
27	.0142	3.348	.5662	1,766
28	.0126	4.251	.4458	2,243.
29	.0113	5.286	.3586	2,789.
30	.010	6.750	.2808	3,561.
31	.0089	8.523	.2224	4,496.
32	.0080	10.55	.1797	5,565.
33	.0071	13.39	.1416	7,062.
34	.0063	17.00	.1114	8,977.
35	.0056	21.52	.08806	11,360.
36	.0050	27.00	.07020	14,250.
37	.0045	33.33	.05686	17,590.
38	.0040	42.19	.00493	22,260.
39	.0035	55.10	.03440	29,070.
40	.0031	70.24	.02698	37,060.
41	.00275	89.29	.02124	47,080.
42	.00250	108.0	.01755	56,980.

Table 3: Nichrome 60 Wire Resistivity Chart [7]

It can be seen in the table above that 26 AWG nichrome has a resistance of 2.67Ω per foot of wire. The following calculations were performed to identify the amount of wire needed to generate the required current of 1.58A.

$$R = \frac{V}{I} \rightarrow \frac{12v}{1.58A} = 7.595\Omega$$

(3)

$$Length\ nichrome\ of\ wire = \frac{R}{\frac{\Omega}{foot}} \rightarrow \frac{7.795\Omega}{2.67\frac{\Omega}{foot}} = 2.845\ feet$$

(4)

It was calculated that it would only require 2.845 feet of 26AWG nichrome 60 wire to generate the desired 1.58A at a voltage of 12v. Although there is no concrete limitation of the size of the heating element it was listed as constraint 3 as too much nichrome would not only be more expensive, but it would also be cumbersome to fix to the balloon. Ideally the smaller the length of nichrome wire the better and given that this 2.845 feet can be turned into a coil to further reduce its size it was decided that this length of wire is acceptable for the design. With the three constraints considered 26AWG nichrome 60 wire was an ideal candidate for the ETS heater element and with this wire selected it was then possible to select the other components of the ETS listed earlier that were dependent on this selection.

3.2.4 MOSFET and Complimentary Component Selection

Other than the Nichrome heater itself the most important component in the ETS is the power MOSFET that controls the flow of current through the heater element. Given the digital pin on the Arduino Uno, being used to control the ETS, are incapable of sourcing 1.58A, and that the ETS heater draws its current from the independent 12V coaxial power connector it is essential to use a power MOSFET in the triode region as a switch to turn the system on or off. In this configuration the Arduino digital pin applies a voltage, above the threshold voltage, to the gate terminal of the power MOSEFT while the heater is connected to the drain terminal and the source terminal is grounded. There are several very important constraints that had to be considered when selecting this MOSFET that have been listed below.

1. Must operate in the triode region
2. Must have a threshold voltage below 5V (Arduino logic)
3. Must be able to tolerate a continuous drain current of 1.58A
4. Must be able to operate effectively at -44.5°C
5. Must have a very small R_{DS}(ON) in order to not affect the output current

Each of these five constraints are important to ensuring that the ETS operates reliably. In order to begin the search for an ideal power MOSFET filters were applied to a general search for power MOSFETs on Mouser.com [8]. These filters covered constraint 2 as a restriction on Vgs th, constraint 3 as a restriction on id continuous drain current, and through hole mounting style. After applying these filters, several different candidates were compared in order to ensure that the best possible MOSFET was selected. Since only two of the constraints were applied in the filter it was necessary to check the data sheets of each candidate for values like minimum continuous operating temperature and R_{DS}(ON) in
order to satisfy constraints 4 and 5. Ultimately the Toshiba TK040N65Z,S1F was found to have the lowest cost while still satisfying constraints 2 through 5. The constraint related parameters are shown below.

Mounting Style:	Through Hole
Package / Case:	TO-247-3
Transistor Polarity:	N-Channel
Number of Channels:	1 Channel
Vds - Drain-Source Breakdown Voltage:	650 V
Id - Continuous Drain Current:	57 A
Rds On - Drain-Source Resistance:	40 mOhms
Vgs - Gate-Source Voltage:	- 30 V, + 30 V
Vgs th - Gate-Source Threshold Voltage:	3 V
Qg - Gate Charge:	105 nC
Minimum Operating Temperature:	- 55 C
Maximum Operating Temperature:	+ 150 C

Figure 10: Toshiba TK040N65Z,S1F Data Sheet [8]

With constraints 2 through 5 satisfied, it was necessary to validate that constraint 1 was also satisfied with this MOSFET selection. Ensuring that the MOSFET will stay in the triode region is essential to ensuring that the MOSFET acts as a predictable switch and has a consistent R_{DS}(ON). The MOSFET region of operation is related to the circuit in which it is being used rather than the design characteristics of the MOSFET alone, so this portion of analysis required additional calculations. It is common practice to plot a circuit specific “Load line” against an Id vs. Vds curve for a specific Vgs value; the intersection of these two lines will show the Vds value at which the MOSFET operates, verifying that the MOSFET is in the correct region of operation. In order to generate this plot, a MATLAB script was written to generate an Id vs. Vds curve for a Vgs of 5V(Arduino logic). This MATLAB script took in parameters that were found in the Toshiba TK040N65Z,S1F datasheet and generates a said curve. Additionally, the script plots the load line for the ETS circuit against the previous plot. Finally, the script adds a dotted line to indicate the overdrive voltage of the MOSFET which separates the triode region

and saturation regions. The plot specific to the Toshiba TK040N65Z,S1F can be seen below, and the script used to generate the plot can be found in Appendix D.

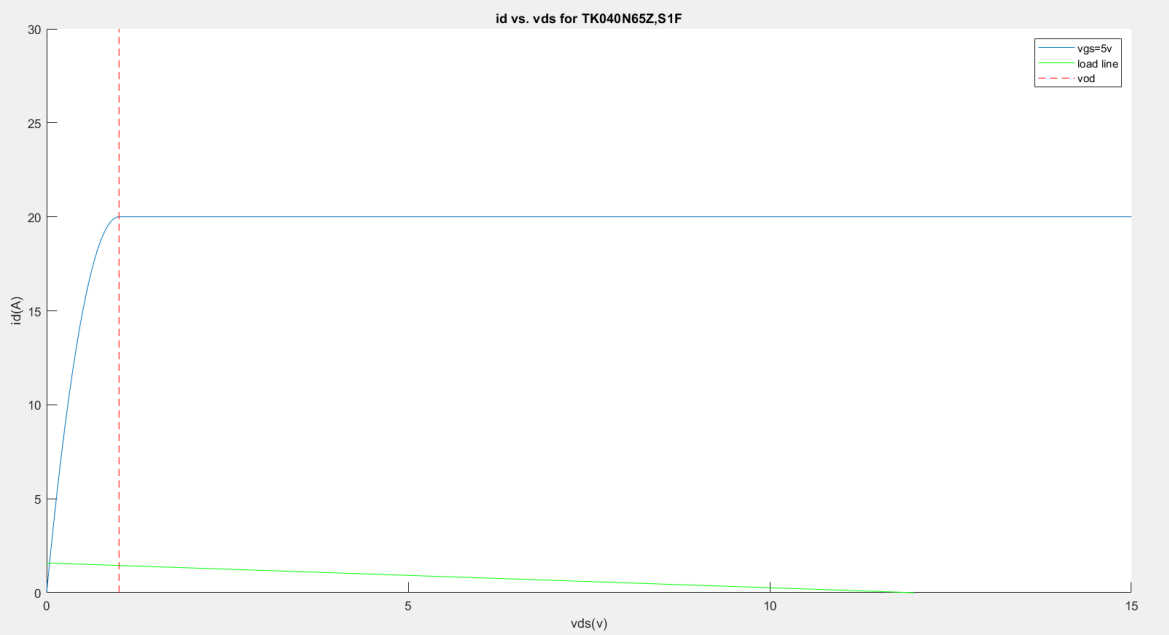


Figure 11: Id vs. Vds curve for Toshiba TK040N65Z,S1F

It can be seen in the figure above that the load line(green) intersects with Id curve(blue) before the devices overdrive voltage (red dotted line) with a Vgs of 5V. Given that the Toshiba TK040N65Z,S1F remains in its triode region for this specific application it was possible to move forward with the ETS circuit design and the selection of complimentary components.

Although the nichrome heater and the power MOSFET are the two core components of the electrical portion of the ETS, there were several complimentary components that were added to the circuit to ensure optimal performance. These components include the following.

1. Protective fuse for power supply
2. Pull down resistor for MOSFET gate
3. Current limiting resistor on Arduino logic pin

All the components listed above were selected based on simple circuit parameters or common practice. For the protective relay the only requirement was that the relay will blow at 2A to ensure the current limit of the 12V 2A coaxial output on the battery is never exceeded. Additionally, it was important to select a slow blow fuse in order to ensure that no instantaneous spikes in current draw would cause an immediate system failure as these may occur when the system is initially powered on. The Littlefuse 0215002.MXP was found on Mouser.com [9] and meets the previously stated requirements. The relevant specifications for this fuse are shown below.

Current Rating:	2 A
Fuse Type:	Time Delay / Slow Blow
Fuse Size / Group:	5 mm x 20 mm
Voltage Rating AC:	250 VAC
Interrupt Rating:	1.5 kA at 250 VAC
Mounting Style:	Holder / Clip
Termination Style:	Clip
Minimum Operating Temperature:	- 55 C
Maximum Operating Temperature:	+ 125 C
Packaging:	Bulk
Resistance:	56.6 mOhms

Figure 12: Littlefuse 0215002.MXP Data Sheet [9]

In addition to the fuse the proper holder/clip hardware was also purchased on Mouser.com. For the two resistors listed earlier, there are standard values that are recommended by a number of online sources [10] making the selection of these components straight forward. It was found that the ratio between the two resistors is what matters most, and it was recommended that the current limiter resistor is roughly 100 times smaller than the pull-down resistor. Armed with this information a 1kΩ resistor was selected for logic pin current limiting and a 100kΩ resistor was selected for a pull-down resistor.

3.2.5 ETS Circuit Simulation

With all the major elements of the ETS electrical system selected the following schematic was drafted using LT Spice.

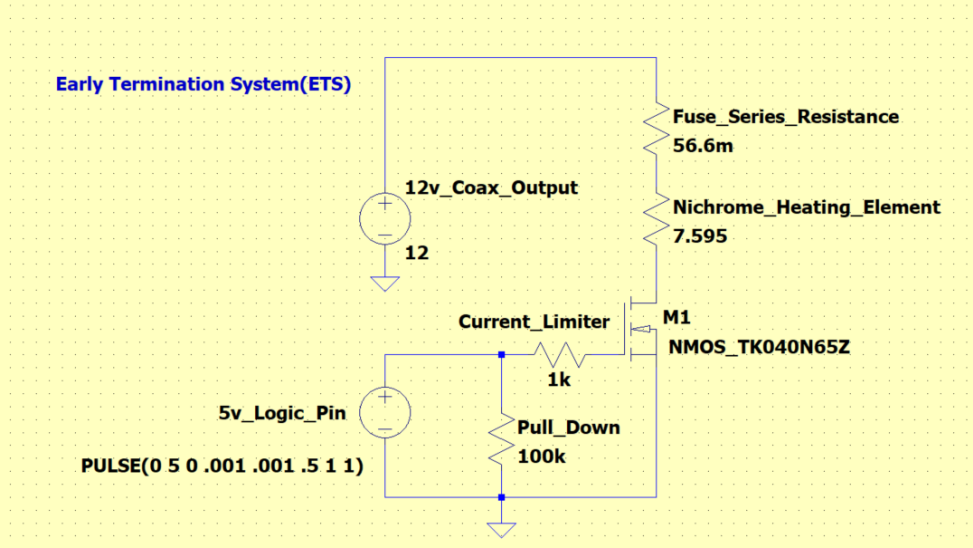


Figure 13: ETS LTSpice Schematic

Using this circuit, a basic simulation was performed to ensure that the nichrome had the correct current running through it when the gate of the MOSFET was supplied with 5V. The results of this simulation can be seen below.

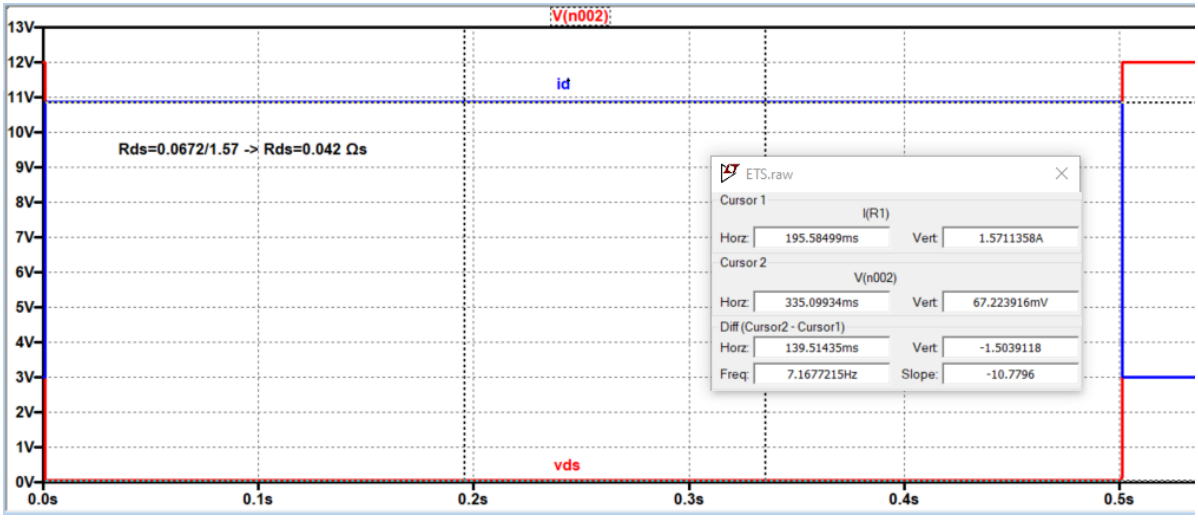


Figure 14: ETS LTSpice Output Simulation

The simulation confirmed that when 5V was applied to the gate the voltage across the MOSFET was near zero, as seen in the left half of the plot above, which aligns with the low Rds ON value of the Toshiba TK040N65Z,S1F. It can also be seen in the left half of the plot that id reaches a maximum current of 1.571A when the gate was supplied with 5V. The right side of the plot displays vds and id

when the gate was pulled to ground, and as expected the output current was 0A and vds rests at 12V. With this schematic drafted, simulated, and all the components selected and ordered the team was able to move onto assembly and testing of the first physical prototype of the ETS.

3.2.6 ETS Prototyping and Testing

Although it was not until later in the prototyping process that the final design for the mechanical elements of heating element was selected an initial prototype heater was developed for the purpose of testing the electrical components selected for the system and ensuring that the nichrome wire was effectively reaching the required temperature. The initial heater design included the necessary 2.845 feet of nichrome wire wrapped around a circular insulated metal core. The insulated core included a 3-inch diameter aluminum ring wide enough to be fixed to the bottom of the balloon with the neck of the balloon running through the ring. This ring was wrapped in an insulating self-adhering tape, with a maximum temperature of 500°F, to prevent shorting between the nichrome and the metal core. This 500°F maximum temperature far exceeds the expected 347°F temperature of the nichrome wire. An image of this prototype heater has been provided below.

Figure 15: Prototype ETS Heating Element

It can be seen in the image above that the leads(red) that connect the heater to the rest of the ETS circuit are joined to the nichrome wire(silver) using short steel screws and lock nuts. This was done as conventional solder can melt at temperature anywhere from 190 to 840°F making it an unreliable method for joining the nichrome to its leads. This solution was used consistently across of iterations of the heating element. The remainder of electrical portion of the ETS was assembled on a protoboard and can be seen in its entirety in the image below.

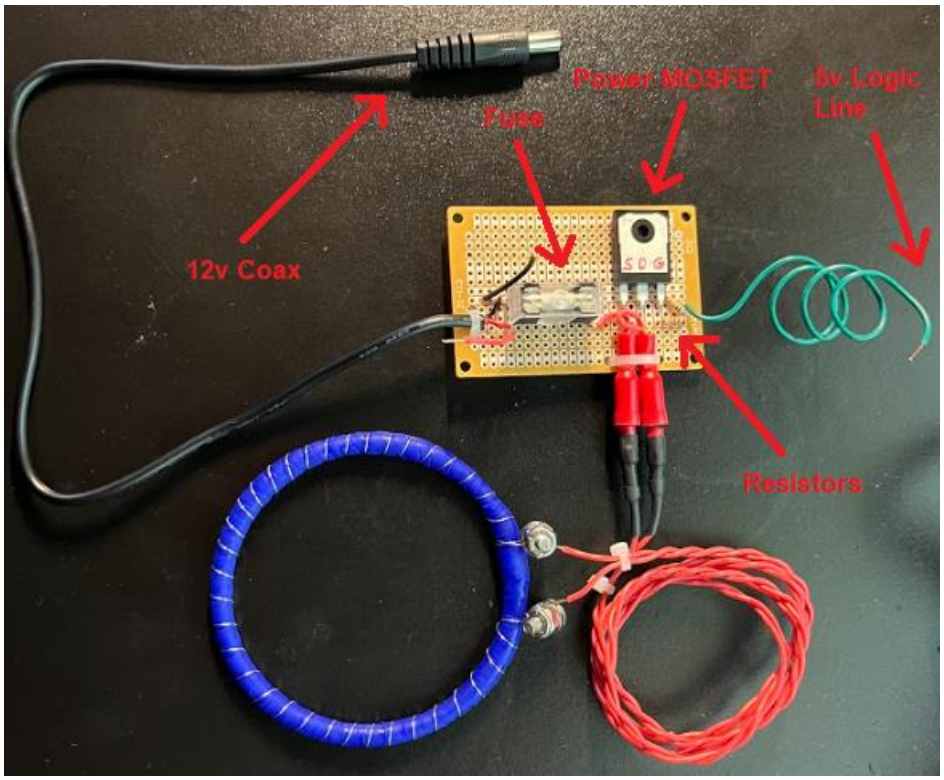


Figure 16: Complete ETS With Prototype Heating Element

As previously stated, the purpose of creating a prototype heater was to evaluate the performance of the remainder of the ETS including all the sections labeled in red in the image above. Additionally, the use of a prototype heater allowed for the verification of the nichrome wire related calculations through heat measurements.

In order to test the ETS combined with the prototype heater a bench top DC power supply was used to provide both 12V to the coaxial connector, which will eventually be attached to the payload's battery, and 5V to the logic line in order to toggle the ETS into its ON state. In addition to providing power to the various parts of the circuit a FLUKE multimeter paired with a temperature probe was used to measure the temperature of the nichrome wire. From the moment that the logic line was provided 5v a timer was started in order to measure the amount of time that it would take for the nichrome wire to reach its calculated temperature of 347°F. Images of the test setup are shown below.

Figure 17: ETS Heating Element Temperature Test

It was found that 5 minutes and 50 seconds after 5v was applied to the gate of the MOSFET the nichrome wire had reached a steady temperature of 341.1°F. This temperature is slightly below the desired 347°F but can be adjusted by varying the length of the nichrome wire segment slightly. This test provided proof that the calculations done to ensure the nichrome reached the correct temperature were accurate. The small error in the steady state temperature of the heater would have resulted from unanticipated resistance in the leads connecting the nichrome wire to the rest of the ETS. In addition to the temperature related data, this test proved that the MOSFET did function correctly as a switch and remained in its triode region as id was measured to be a constant 1.47A according to the DC power supply used to power the system. From this point forward the ETS circuit remained untouched with the exception of the heating element which required further iterations to ensure the desired heat was reached consistently and delivered to the surface in need of melting in the most efficient way possible.

3.2.7 Final Iteration of the ETS

With a proof-of-concept protype of the ETS heater completed and a functioning ETS circuit the focus was shifted to fixing the heating element to the balloon itself which would ultimately help to guide the design of the final heating element. To identify how the heater would be fixed to the balloon it was first necessary to develop an understanding of how exactly the HABs parachute system would interact with the heater. For obvious reasons it is very important to ensure that the heating element will not damage the parachute when activated in the event of an emergency. Prior to this point in the design process this consideration had not been taken into account so initially, through communication with last year's team, it was identified that the HAB 3.0 parachute worked as shown in the following image.

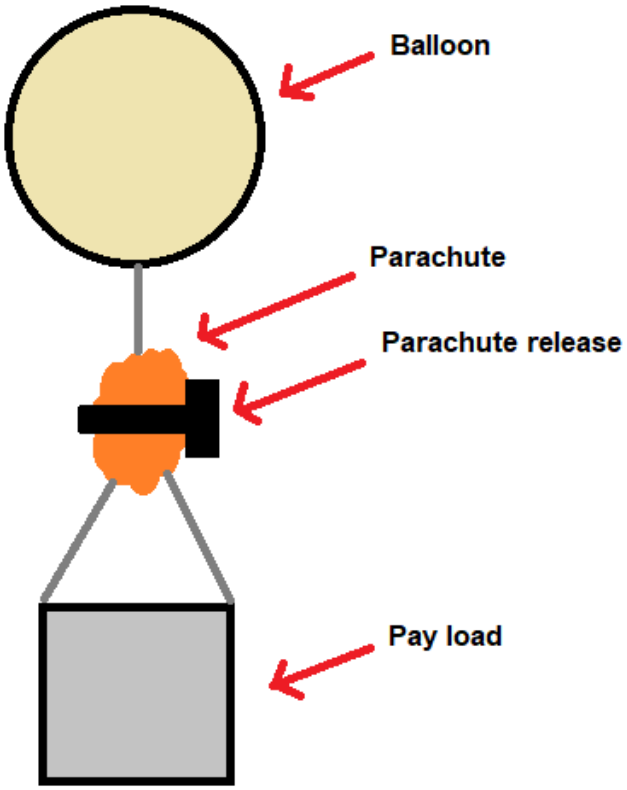


Figure 18: HAB3.0 Parachute Configuration

In the simple diagram above it can be seen that both the payload and balloon are actually fixed to the parachute with independent lines (shown in dark grey). Prior to this research it was not known that this was the case and it was initially assumed that the parachute lines and balloon line would be entirely independent of one another. Although this assumption did not greatly impact the development of the electrical components of the ETS it did in fact shape the initial design of the heating element. In can be seen from the diagram above that the ETS heater would need to be fixed to the balloon all the way at the top of the system, above the parachute and all of its associated lines. It was determined that with the current heater design there would be a risk of the heater leaving part of the balloon still attached to the parachute impeding its ability to effectively properly deploy. To ensure that the entirety of the balloon is detached from the parachute, the design was shifted from a balloon popper to a line cutter which required only a few slight modifications.

First a line material was identified that could be cut using the same nichrome wire at its original designed temperature. Ultimately the Polyethylene line was selected as it has a melting point of 230°F which is well within the original design temperature of 300°F. This line would be used to link the balloon to the parachute and the ETS heater would be fixed to this line. A modified version of the previous diagram in which the ETS modification is shown can be seen below.

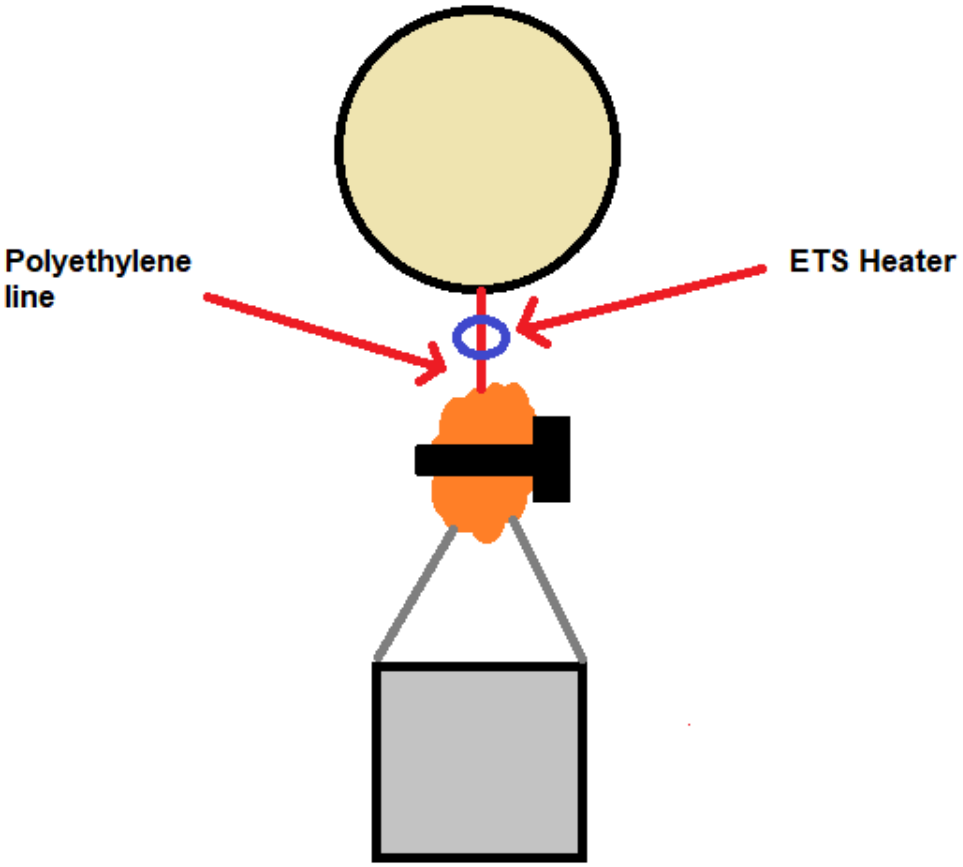


Figure 19: ETS Positioning on Payload

The diagram above shows the modified heating element (blue ring) fixed to the polyethylene line(red). This modified heater system cuts the entire balloon free in the case of an emergency rather than popping the balloon and risking any remaining balloon debris from interfering with the deployment of the parachute.

For the heater to effectively cut the line connecting the balloon and the parachute it was important to ensure direct contact between the nichrome wire in the heater and the line. This was achieved with an alternate design that utilized the same components but in a slightly different configuration. This new heating element can be seen in the image below.

Figure 20: Final ETS Heating Element Configuration

It can be seen in the image above that the nichrome wire runs across the metal wring and intersects perpendicularly with the polyethylene line. At this intersection point the nichrome is wrapped around the line four times which ensures constant contact between the nichrome wire and the line at the bottom of the ring. The line is fixed to the ring as a form of strain relief to prevent any unexpected external forces from damaging the nichrome line. Apart from this transition from the initial heater design very little has changed in terms of how the nichrome is fixed to the leads that connect to the ETS circuit in the payload. As an additional precaution the remaining nichrome which is not used in the cutting section (spanning across the ring) is covered by heat resistant tape which will ideally reduce the risk of the nichrome coming into contact with the parachute below it.

3.2.8 Effects of the ETS on parachute deployment

As described in the previous section, fixing the Early Termination System to the payload required a strong understanding of the parachute deployment system that has been used in previous years. This system was largely undocumented in the HAB 3.0 report as it was a recycled element of HAB1.0 and at the time this part of the system required no design changes. With this year's addition of the ETS a significant part of how the parachute was to be deployed had to be changed.

The most significant modification to this system related to a change in the way that the balloon's tether physically attaches to the parachute. In previous years the parachute was connected directly to the balloon with a small loop found at the apex of the chute. This element of the parachute system was ultimately changed in the HAB 4.0 payload as part of the ETS may have complicated the successful deployment of the parachute. In previous years when the chute was not wrapped up and instead was pulled taught and the chute release was purely there to keep the chute from inflating

before it was needed. This presented a problem as leaving the chute taut and having its line fully unraveled put a large distance between the balloon line and the payload. This significant gap between the payload and the point where the ETS needs to be attached would require the wires used to power the heater to be very long in order to reach the ETS circuit in the payload. Longer wired could directly result in a failed chute deployment if they became tangled with any of the cute lines. In order to avoid excess wire length and streamline the parachute system as a whole the way in which the balloon is linked to the payload as the chute folding technique were altered on the HAB 4.0 payload.

The alteration made in the parachute system of the HAB 4.0 payload are as follows:

- Properly rolled up parachute held in place by the chute release
- Parachute chords are rolled into the chute rather than being full extended at launch
- Balloon connection point separated from parachute apex in order to avoid excessively long power wires to compensate for fully expanded and unraveled parachute
- ETS cuts balloon away from parachute at maximum altitude, regardless of emergency in order to shed excess balloon material that could result in tangling

All these modifications to the parachute system not only reduce the need for unneeded risk increasing components but also utilize existing hardware in a more professional manner. The parachute release is not designed to be used as a way of keeping a chute from inflating when taut but was actually intended to keep a fulling wrapped up parachute in the compact state up until deployment. Additionally having the parachute lines contained within the parachute during flight decreases the risk of wind induced tangling prior to deployment. It is worth noting that this altered method of parachute storing was not possible prior to the implementation of the ETS as it is needed in order to disconnect the balloon and prevent it from interfering with the unraveling chute when the payload is losing altitude.

These alterations resulted in a slightly more complicated deployment sequence. In order to better explain this, images of the complete parachute system are provided below.

Figure 21: Payload Parachute Line Configuration

The provided images of the parachute system depict the parachute in its unrolled state(left) and in its rolled state(right) contained by the chute release. It can be seen that the parachute and balloon line are connected to a common silver loop which links both to the payload lines. The ETS wires are twisted together and are fixed to one of the payload lines all the way up to the point of the silver ring at which point it jumps to the balloon line and ultimatly to the ETS further up on said line. It can be seen that when the parachute is in its rolled state that all of the black chute chords are contained within the roll. In the image of the unrolled chute it is clear that the parachute apex and the balloon line(with the ETS fixed to it) are entirely separate and that there will only be a short legth of balloon line remaining after the ETS has been triggered. Even with the parachute folded over on its lines in the unrolled image, it can be seen that the ETS wires would need to be significantly longer is the balloon line was connected to the chute apex when bull taught. The steps below described the exact sequence that the parachute system would following in a non-emergency situation.

1. Payload reaches peak altitude and balloon pops
2. Payload begins to lose altitude rapidly with balloon line still intact and popped balloon still connected
3. Payload reaches software defined decrease in altitude and ETS is triggered
4. Balloon line is cut allowing excess line and popped balloon to separate and blow away leaving the payload, ETS, and rolled up chute in a continued free fall
5. Altitude continues to drop until preprogrammed chute release altitude is reached
6. The parachute is released allowing it to unroll and inflate
7. Added length of the parachute chords puts ETS well below the inflated parachute
8. Payload slowly continues to lose altitude until touch down

When broken down into steps, it can be seen that the parachute release system, even after these modifications, is quite simple. Overall, this improved chute release system allows everything to be more compact and greatly reduces the risk of any damage being done to the parachute and its chords during the flight. It is essential that this system be properly reset before every new payload HAB flight, and all the steps required to do so will be described in detail within the startup guide. This section of the guide will aid in both parachute rolling as well as ETS resetting to ensure successful operation every time.

3.3 ETS Computer System and Software

3.3.1 Functionality and Equipment

Decisions on the functionality of the payload were made based on the most abundant greenhouse gases, the Early Termination System additions, goal to implement image transmission, and the equipment passed on from the HAB 3.0 team.

Sensor	Product Name	Operation Range (°C)	Voltage Input (V)	Sensing Range	Output Type
Carbon Dioxide (CO2)	Gravity: Analog Electrochemical Carbon Dioxide Sensor	-20 - 50	3.7 - 5	0-10000ppm ± 100ppm @ 400ppm	Analog
Ozone (O3)	Gravity: Electrochemical Ozone Sensor	-20 - 50	3.3 - 5.5	<=10ppm, 10ppb resolution	Digital (I2C)
Methane (CH4)	Methane CNG Gas Sensor - MQ-4	-10 - 50	5±.1	300 −10000ppm	Analog
Barometric Pressure	SparkFun Pressure Sensor Breakout - MS5803-14BA	-40 - 85	1.8 - 3.6	0 - 14 Bar, 0.2 mBar Resolution	Digital (I2C)
One Wire Thermometer	Temperature Sensor - High Temperature, Waterproof (DS18B20)	-55 - 125	3 - 5.5	-10°C to +85°C ±0.5°C accuracy	Analog
Ultraviolet Light (UV)	SparkFun UV Light Sensor Breakout - VEML6075 (Qwiic)	-40 - 85	1.7 – 3.6	280-400nm Light Wavelength	Digital (I2C)
Barometric Pressure, Temp, Altitude (Used as Backup)	JBtek BMP180 Barometric Pressure, Temperature and Altitude Sensor	-40 - 85	3.3 - 5	Pressure: 300hPa - 1100hPa Altitude: <= 30,000ft	Digital (I2C)
GPS Module	GPS & GLONASS Antenna Module YIC93030PGMFUGG-U8	-40 - 85	2.8 - 5.5	~2.5m Max Position Accuracy, Altitude: <= 50,000m	Digital (NMEA 0183)
Camera Module	Raspberry Pi Camera Module v2	-20 - 60	3.3	3280 x 2464 Still Picture Resolution	MIPI Camera Serial Interface (CSI-2)

Table 4: HAB 4.0 Payload Sensors

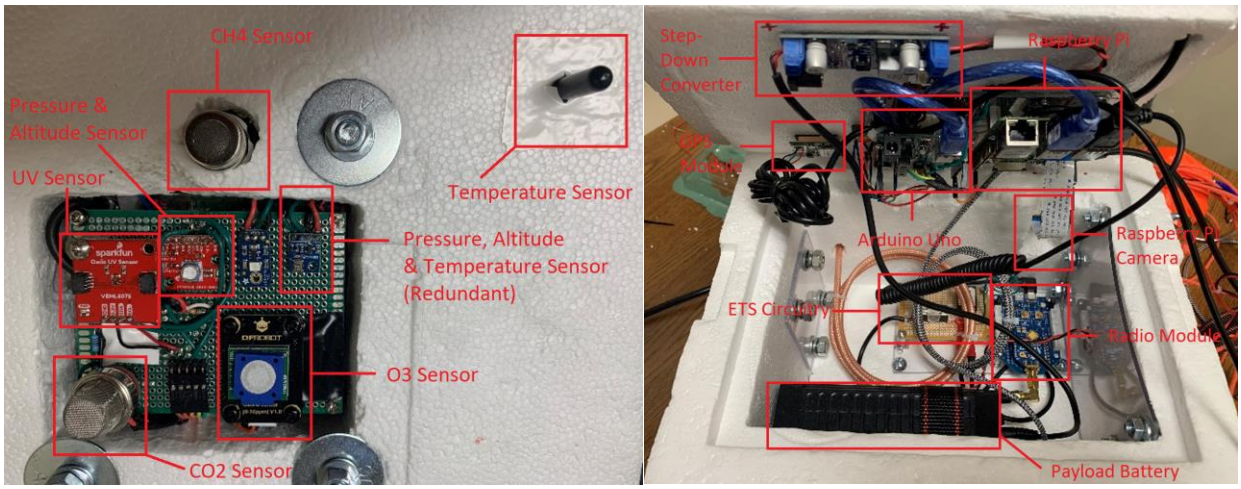


Figure 22: Payload Anatomy

As can be seen in the figure above, most of the sensors are mounted on the outside of the payload box such that they can be exposed to the outside air and sun. The rest of the computer system is located inside the box to insulate it from cold and precipitation, both of which could cause potential malfunctions in the system.

3.3.2 Equipment Acquisitions for HAB 4.0

Although a methane sensor was listed in the payload sensors list of HAB 3.0, in the payload left for our team to work with, we did not find such a sensor [5]. It is likely that this sensor was lost in the flight that was lost in the Atlantic Ocean, and a replacement was not purchased. For our payload to have the same level of functionality, we purchased a MQ-4 Methane sensor and breakout board. Additional equipment acquisitions included a dedicated monitor to use while testing and setting up the computer system for launches, and a Raspberry Pi camera unit for sending images to the base station during flight. The UV sensor attached to the HAB 3.0 payload was nonfunctional, so a different one was purchased and used in the system in place of it.

3.3.3 Electrical Schematic

In the HAB 4.0 team's efforts to make a cohesive repository of project information for future teams, we devised multiple schematics to help familiarize new members with the structure of the project computer system and its electrical connections.

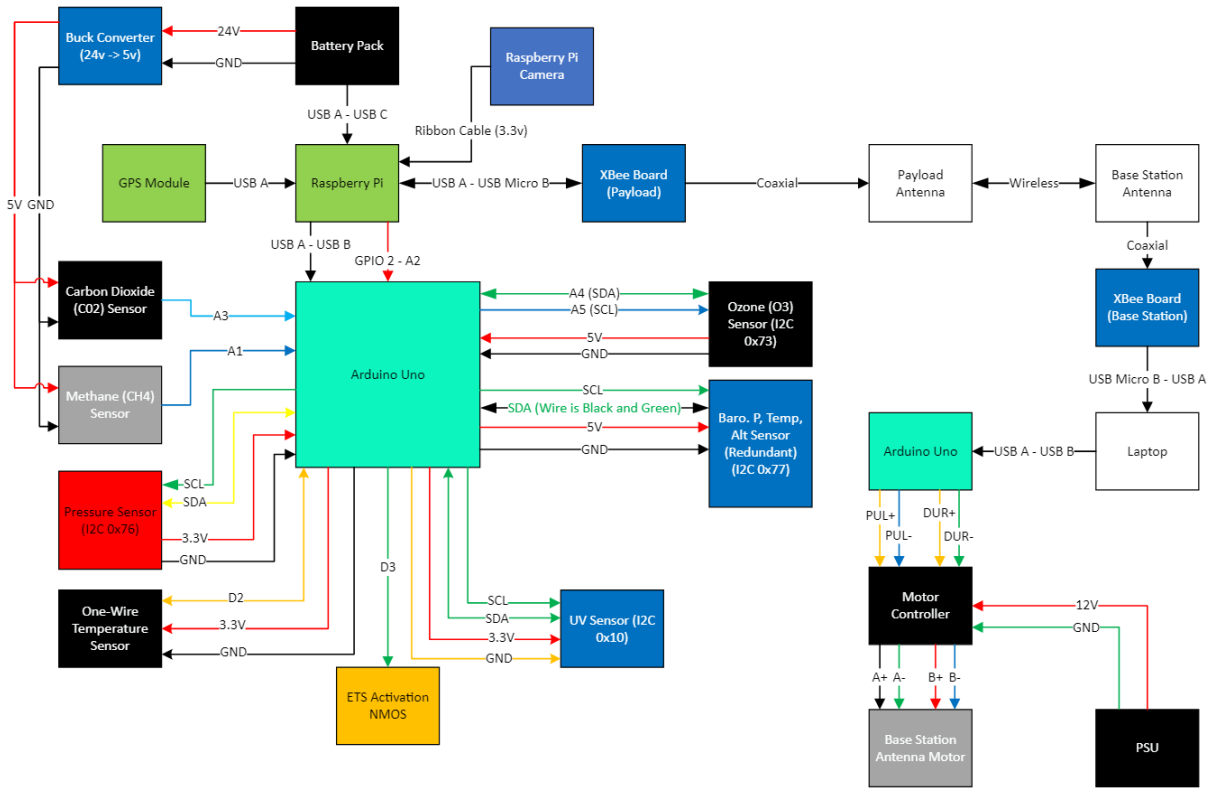


Figure 23: Block Diagram of Payload Computer System

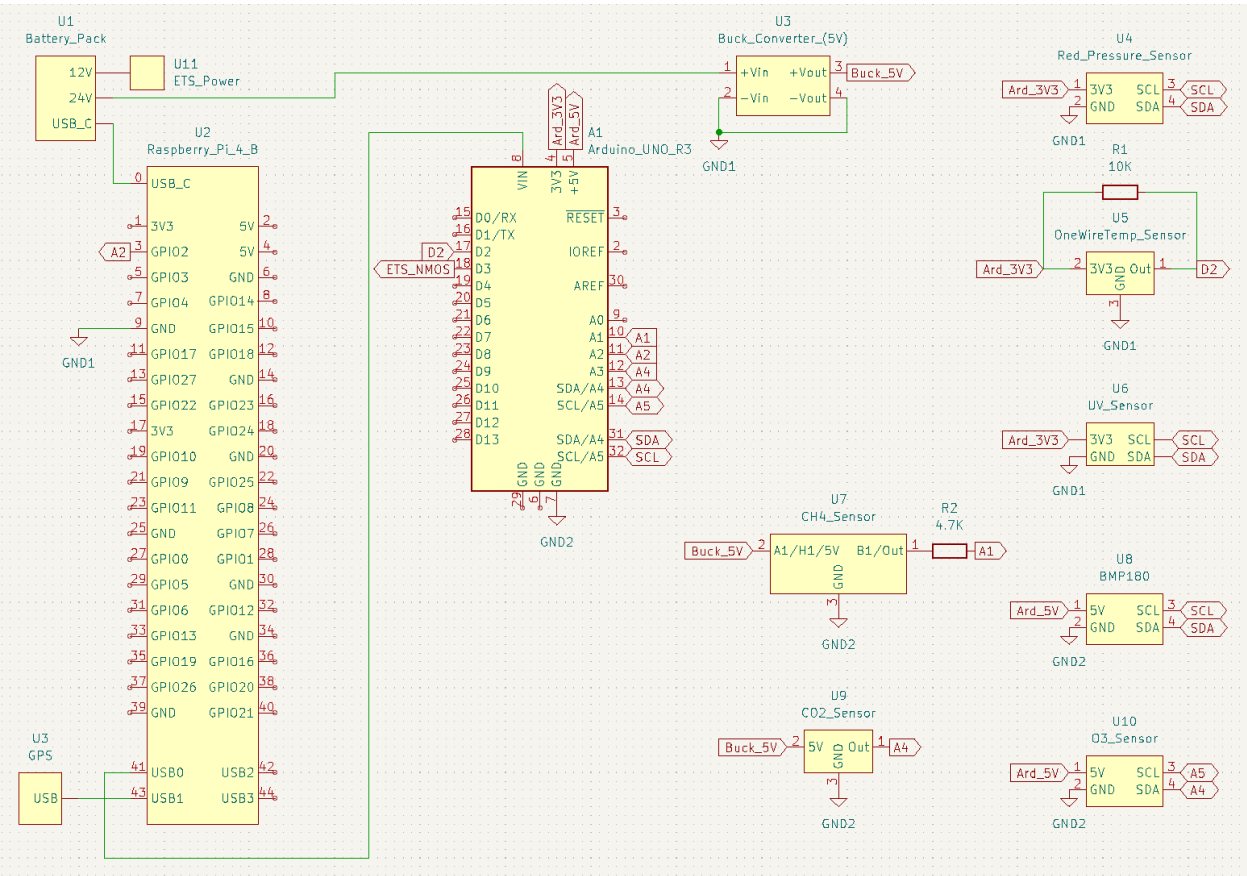


Figure 24: Payload Electrical Schematic

As can be seen in the above diagrams, the Arduino handles the gathering of all sensor data other than the GPS module. This data is communicated to the Raspberry Pi, which is then broadcasted to the base station through the payload radio module and antenna. The carbon dioxide and methane sensors have a heater circuit which requires more current than the Arduino can output, so a buck converter was attached to the 24V port of the payload battery and supplies the sensors with the current they need to function. The Arduino is powered by a USB port of the Raspberry Pi.

The Raspberry Pi is powered by a USB port on the battery, and the 12V port is used to power the ETS, and during setup and testing, powers the computer monitor. GPIO2 of the Raspberry Pi is used to trigger the ETS power NMOS, which dumps current into the nichrome wire, heating it up so it can cut through the balloon rope. Since the Raspberry Pi can only output 3.3V GPIO and our NMOS functions on 5V logic, a level shifter was made through the Arduino to convert this into a 5V signal using 2 of its GPIO ports. The GPS module is connected to the Raspberry Pi via USB.

3.3.4 Gathering Gas Data

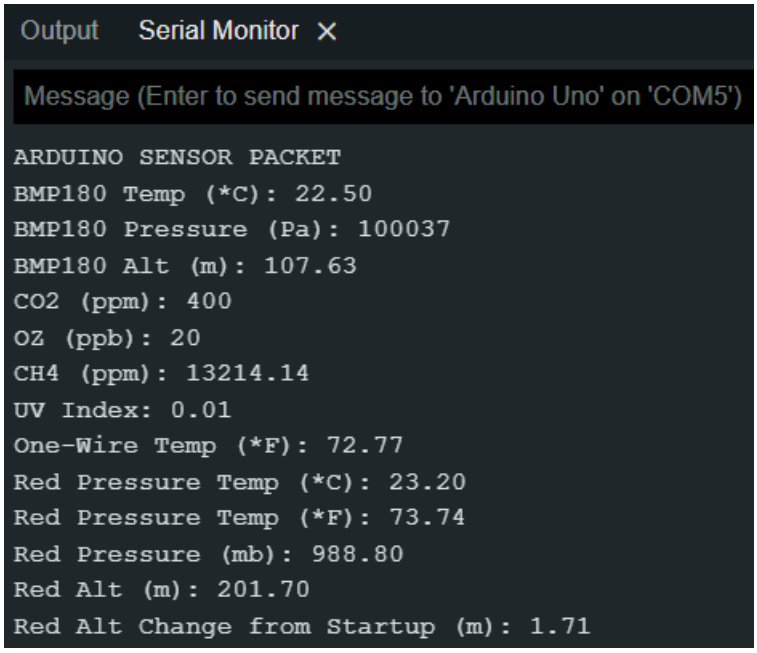


Figure 25: Format of Sensor Data Transfer

After the Arduino is powered on and initializes its sensors, it begins a loop of gathering data from each of the sensors and packing it into a text packet to send to the Raspberry Pi through serial communication. The header “ARDUINO SENSOR PACKET” is used to signify the base station code that the messages that follow are a new packet of sensor data. The order of data gathering corresponds to the order of the sensors in the above figure. The format of the sensor data sent can be seen in the above Figure 25. From here on, Red Pressure and Red Alt refers to the Sparkfun MS5803-14BA, used as the primary pressure sensor.

3.3.5 GPS Boundary Shape

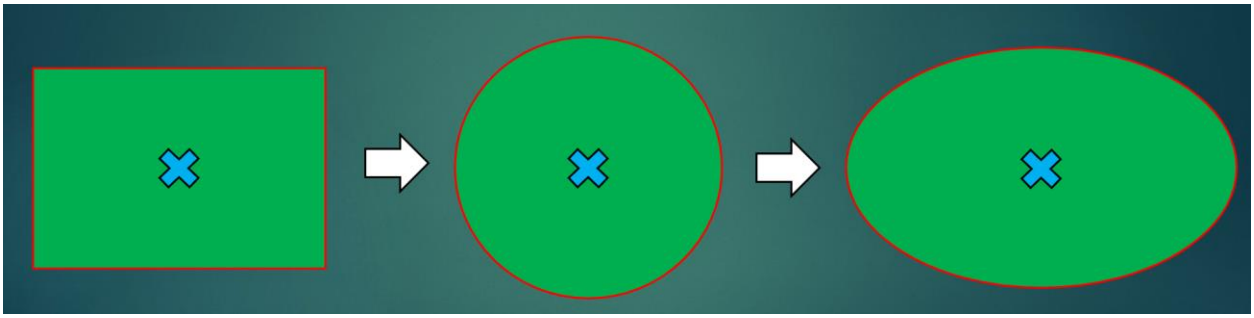


Figure 26: Development Sequence of the Final ETS Boundary Shape

One of the most important decisions in programming the ETS was what the shape of the GPS boundary should be. As depicted in Figure 26, the initial idea was to use a rectangular shape, and over time it evolved into the ellipse featured in the final product. The first idea was to use a rectangle since it was simple and easy: all you needed to do was define a top, bottom, left and right boundary line and the area of flight was defined. However, we realized this shape would prove less reliable than our other options, specifically on account of the rectangle’s corners. If the balloon were to fly into one of the

corners of the rectangle boundary, the flight would be significantly longer than if it flew directly towards the middle of one of the sides.

In order to have more control over the allowed flight area, we opted next for a circle. With an area based on a radius distance from the start point, the payload would only be allowed to fly a single, set distance from the starting point. This would prevent the inconsistencies present in the potential flights that could come from using the rectangular version, but also limits our control of how far in each cardinal direction we would allow the payload to fly. In pursuit of finding the best shape to make the boundary, we reviewed past years' flight paths and researched wind patterns. The Coriolis effect causes winds to be stronger in the west-east axis than north-south, which means we should expect a flight that will move far more horizontally than vertically on a map [6]. With this information in mind, we were ready to make an educated pick of the final shape, which was chosen to be an ellipse. Carrying the qualities we sought in both the rectangle and circle boundaries, this shape lets us define our flight boundary the best.

$$\frac{(x-h)^2}{r_x^2} + \frac{(y-k)^2}{r_y^2} \leq 1. \tag{2}$$

Equation (2) above is used to calculate whether the current GPS position of the payload is within the defined ellipse or not, with all coordinates inside the ellipse making the inequality true. Once the payload has flown outside of the ellipse, the left side of the equation will be greater than 1, and the inequality becomes false. *x* and *y* represent the current longitude and latitude of the payload, *h* and *k* represent the starting longitude and latitude of the payload, and r_x and r_y represent the longitudinal and latitudinal radii, with all variables being in decimal degrees.

3.3.6 ETS Software Testing Hardware Setup

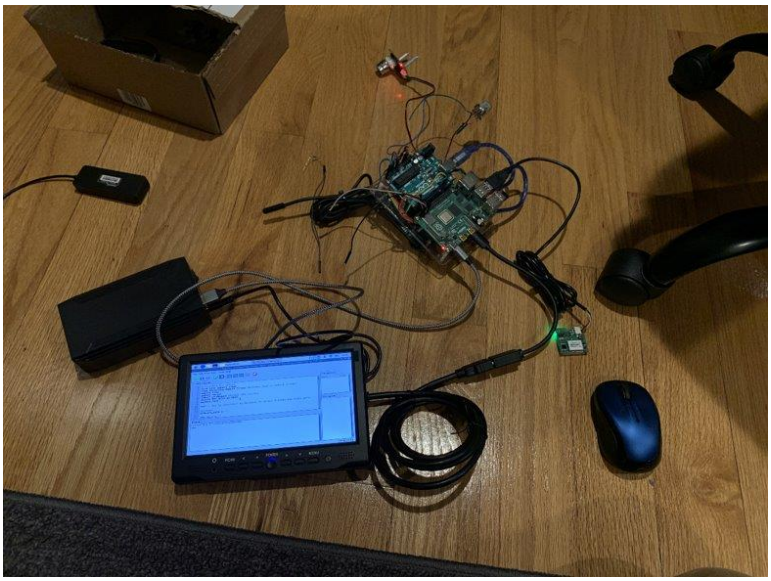


Figure 27: Hardware Setup Used to Test ETS Software

In order to test the ETS software in its early stages, the loose system was placed into a box and the monitor was connected to the 12V port of the battery instead of the ETS system. With the help of skillfully placed alert messages in the code, the system could be tested without the ETS hardware connected. In place of the power NMOS used with the ETS, a blue LED was connected to its port to have a visual aid when it activates and confirmation that the port was outputting correctly. This setup would be used in the testing done in the following section.

3.3.7 Payload Data Organization

Initially, HAB 4.0 used solely text-based transmissions to record sensor data and communicate it to the base station as past years had. However, since this format proved unnecessarily difficult to parse, alternate filetypes which were compatible with our radio hardware were explored. Since the application was about data logging, the obvious choice was a spreadsheet-type file, so the team opted to use CSV files.

Date	Time	Latitude (DD)	Longitude (DD)	GPS Alt (m)	ETS Triggered	BMP180 Temp (*C)	BMP180 Pressure (Pa)	BMP180 Alt (m)	CO2 (ppm)	OZ (ppb)	CH4 (ppm)	UV Index	One-Wire Temp (*F)	Red Pressure Temp (*C)	Red Pressure Temp (*F)	Red Pressure (mb)	Red Alt (m)	Red Alt Change from Startup (m)
2/24/2023	11:36:52	42.2812965	-71.78287617	206.9	0	1.5	98192	264.09	400	20	10.94	-0.02	36.89	2.48	36.46	972.6	200	0
2/24/2023	11:36:54	42.2812965	-71.78287617	206.9	0	1.4	98181	263.92	400	20	10.94	-0.01	36.89	2.48	36.46	972.6	200	0
2/24/2023	11:36:55	42.28129667	-71.78287583	207	0	1.4	98176	265.03	400	20	10.94	-0.01	36.89	2.47	36.45	972.6	200	0
2/24/2023	11:36:57	42.28129683	-71.7828755	207.1	0	1.4	98187	265.28	400	20	10.94	-0.03	36.89	2.46	36.43	974	187.92	-12.14
2/24/2023	11:36:58	42.28129683	-71.7828755	207.1	0	1.4	98190	264.43	400	20	10.94	-0.03	36.89	2.45	36.43	972.7	199.14	-0.87
2/24/2023	11:37:01	42.28129683	-71.7828755	207.1	0	1.4	98177	265.28	400	20	10.94	-0.03	36.89	2.45	36.43	972.4	201.73	1.74
2/24/2023	11:37:02	42.28129717	-71.78287417	207.4	0	1.4	98160	266.14	400	20	10.94	-0.05	36.89	2.45	36.41	973.9	188.78	-11.27
2/24/2023	11:37:05	42.28129717	-71.782873	207.4	0	1.3	98176	265.28	400	20	10.94	-0.06	36.89	2.45	36.41	972.6	200	0

Figure 28: Payload Data File Example

When the Raspberry Pi code begins, one of the first actions it takes is creating a new CSV file in which to store the sensor data it will receive from the Arduino. Each filename holds the date and time at which the code was run so that files produced by subsequent runs are easily differentiated from each other. As seen above in Figure 28, each packet received is organized and saved to the CSV file in the order:

1. Date – Timestamp of current Raspberry Pi system date
2. Time - Timestamp of current Raspberry Pi system time
3. Latitude (DD) – GPS latitude in decimal degrees
4. Longitude (DD) – GPS latitude in decimal degrees
5. GPS Alt (m) – GPS altitude (secondary altitude reading)
6. ETS Triggered – Boolean indicator of when the ETS is first triggered during a run of the code
7. BMP180 Temp (°C) – Secondary temperature reading
8. BMP180 Pressure (Pa) – Secondary pressure reading
9. BMP180 Alt (m) – Secondary altitude reading
10. CO2 (ppm) – Amount of carbon dioxide sensed in the air
11. OZ (ppb) – Amount of ozone sensed in the air
12. CH4 (ppm) – Amount of methane sensed in the air
13. UV Index – Level of UV light sensed
14. One-Wire Temp (°F) – Primary temperature sensor
15. Red Pressure Temp (°C) – Secondary temperature sensor
16. Red Pressure Temp (°F) – Secondary temperature sensor
17. Red Pressure (mb) – Primary air pressure reading
18. Red Alt (m) – Primary altitude reading
19. Red Alt Change from Startup (m) – Amount the altitude read by the Red Pressure sensor has changed since the code started

The CSV file format proved extremely useful in processing data after each test, since it could be exported to Excel without much difficulty and graphed there.

3.3.8 ETS Software Testing Results

12/9/22 Test

On December 9, 2022 the first round of testing to ensure functionality of the ETS was done. This testing was done on the Worcester Polytechnic Institute football field, and tested if the elliptical GPS boundary could be tripped by walking away far enough from the starting point, and at the correct distance. In both tests a boundary with an X-axis radius of .000417 decimal degrees and a Y-axis radius of .000139 decimal degrees was used, emulating a shape similar to that which would be used in an actual flight albeit much smaller.

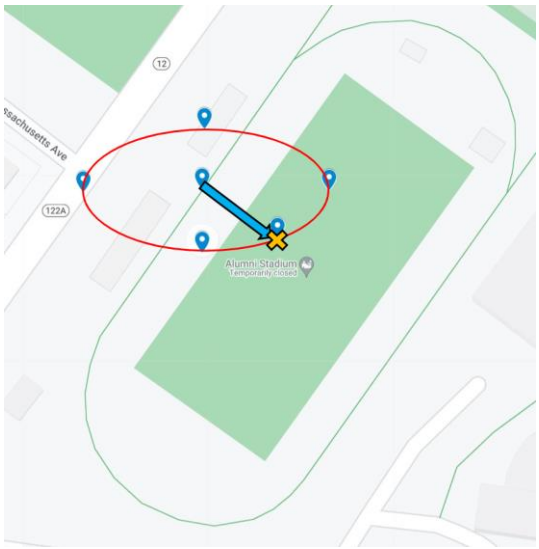


Figure 29: GPS Boundary Testing Trial 1, 12/9/22

The first trial, depicted in Figure 29 had the payload starting at the bleachers of the football field (at the base of the light blue arrow line). As we carried the payload towards and across the football field, the alert message indicating the ETS has been activating appeared on the monitor and the blue LED at the ETS power NMOS port lit up. The location where the ETS was activated is denoted by an orange X mark.

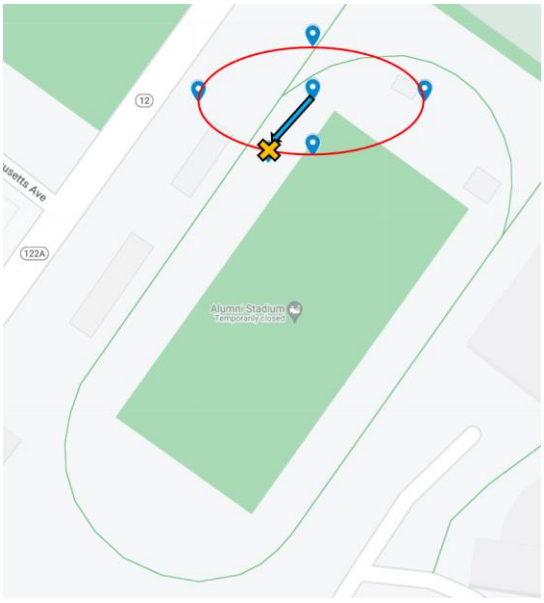


Figure 30: GPS Boundary Testing Trial 2, 12/9/22

A second trial was done the same day but starting further north and with the payload being moved southwest instead of southeast. As with the prior test, the ETS was activated after moving far enough away from the starting point, the path depicted by the light blue arrow line. After overlaying all test information and results onto a map of the football field, the tests were deemed successful. The point at which the ETS activated during each test aligned with the boundary that was set beforehand.

2/8/23 Test

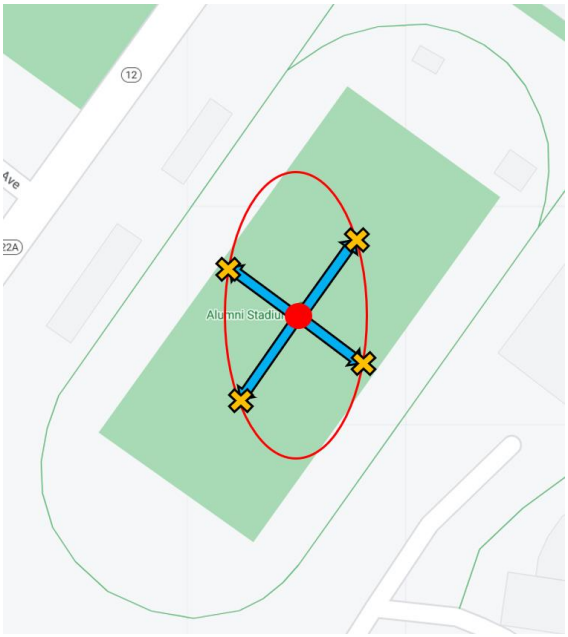


Figure 31: GPS Boundary Testing Trial, 2/8/22

The third trial of the ETS system to confirm consistency in results took place on February 8, 2023. During this time the CSV data organization system was also tested. In this test, the same radii

lengths were used but were flipped so that a vertical ellipse was produced for the boundary instead of a horizontal one (X-axis = .000139 DD, Y-axis = .000417 DD). To test the boundary, the code was started while the payload was in the center of the WPI football field, and then walked toward an end or side of the field as seen in the figure above. The payload responded as planned, activating the ETS when the line is roughly crossed. There was some variation in how far past the line the ETS was triggered, since the accuracy of the GPS module is somewhat inconsistent but accurate enough for the project. Given that the payload will normally be flying miles away from the location of the base station and team, on this larger scale of distance the GPS accuracy is satisfactory.

3.3.9 Image Transmission

For the capturing of images, we acquired a raspberry pi camera module. When configuring the camera module, we selected a relatively low resolution so that we would have images whose sizes were reasonable for our communication system's data rate. Through applying an image encoding library, we were able to prepare our captured images for transmission. The process of image encoding essentially compresses and encodes our image files, reducing their size. This was an integral and necessary component to our image transmission functionality as the image file sizes needed to be within the capabilities of our communication system. Without using an encoding algorithm, we would be spending large amounts of time transmitting images. Additionally, we are increasing the likelihood that we drop too many image packets which will result in a failure to reconstruct the image at the base station due to missing a significant portion of data.

For our application of a high-altitude balloon, we made use of the Slow Scan Digital Video (SSDV) encoding library written in the C language by Philip Heron that we discovered in the early stages of our research on image transmission techniques [11]. To implement the encoding library into our project, we installed the SSDV (Slow Scan Digital Video) library onto our payload's raspberry pi and base station computer for the respective encoding and decoding of our captured images.

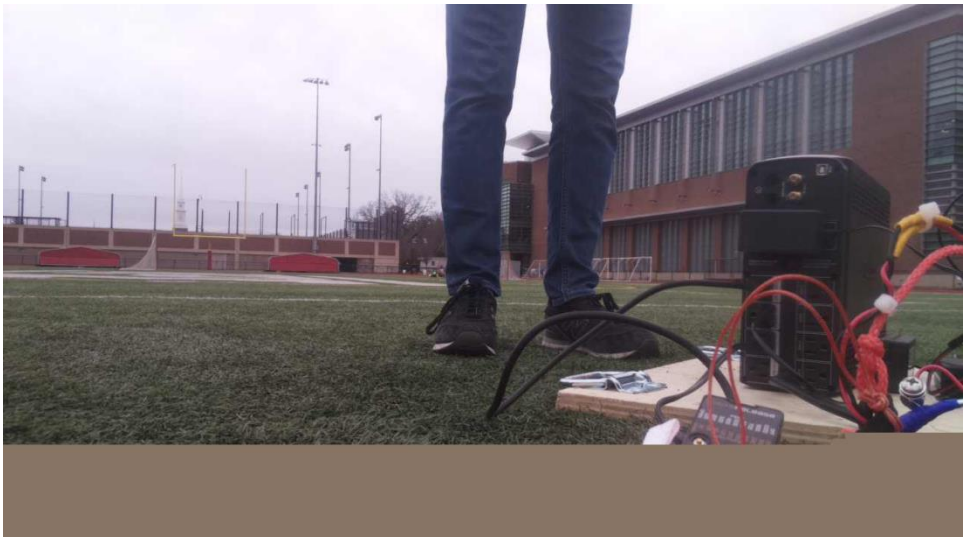


Figure 32: Football Field Test 2/9/23

The program developed by Philip Heron essentially encodes the specified images by applying the Huffman coding to create a significantly smaller file in the form of a binary file. Due to the fact the library was written in C, we had to call the program as a subprocess in the python language to encode captured images ahead of transmission.

In developing this functionality, we had to redesign our transmission system such that we could accurately and consistently read image packets which are 256 bytes in size. To achieve this, we added packet headers to the beginning of image packets that our base station program would recognize after being read from the serial buffer, then write its contents to a binary file which would later be decoded. This was due to the concern of creating a significant delay in our base station program which would affect our tracking system and our communication link. While redesigning the communication aspect of our base station program, we considered employing other data integrity techniques to monitor the performance of our communication system and provide insightful metrics. However, due to time constraints we did not develop many of these features more accurately.

When deciding how to schedule the capturing and transmission of our images, we decided to capture an image once the prior image had been sent. Additionally, rather than sending all image data at once, we sent an image data packet among the other payload data (GPS & Sensor Readings). This is due to the concern that this stream of data being sent over long distances has the potential to disrupt our antenna tracking system which is reliant on receiving GPS coordinate data. In testing our image transmission functionality in our two field tests, we did receive some images where packets had dropped.

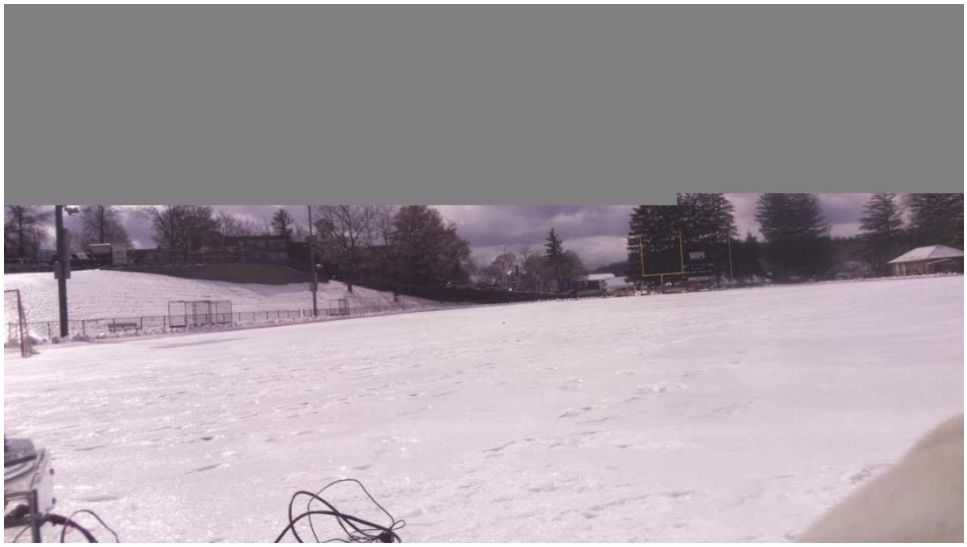


Figure 33: Football Field Test - Image with missing packets 2/19/23

To ensure the proper allocation and storage of received image data, we inserted a byte object prefix/header that is concatenated to each split segment of image packets that are sent so that the base station can differentiate between the received data. This file is then to be passed back to the program as an input binary file which will be decoded and used to produce the original image.

3.4 Base Station Development

Previous HAB teams used a base station to communicate with the payload while in the air. HAB 3.0 added to this aspect of the project by implementing an upgraded antenna sat upon a wooden box with a stepper motor inside to control the rotation of the antenna. This team had planned to implement an automatic adjustment system, where the antenna would move as the payload moved through the sky, however, this effort was left unfinished, as the HAB 3.0 team had to focus on reconstructing the payload after losing the first one in the Atlantic Ocean. As part of our project, we went on to finish this effort and redesign the base station for it to be able to rotate and point towards the payload while in flight. As part of the recommendations formulated by the HAB 3.0 team, the base station should have a microcontroller controlling the stepper motor which will control how far the motor turns. In addition to implementing the rotation of the antenna, the HAB 4.0 team is adding a linear actuator to control the angle at which the antenna is pointing.

Figure 34: Old Base Station on Quad for Preliminary Testing

The parts left behind by the previous team included the unfinished wooden base station with a 1.8° per step, 2.4 N*m, 24V stepper motor and the stepper motor controller. This specific stepper motor was chosen by the previous team for its strong holding torque. This is particularly important for battling winds as most launches occur in February or March. At the start of the year the base station was

evaluated for future use and the team decided to make a new base station as the current one was made of frail plywood and the motor was not mounted properly.

3.4.1 Base Station Software Redesign

To develop and fully incorporate the ability to transmit and receive encoded images, and enable antenna tracking, we had to redesign the base station software to support these capabilities. This involved creating a system for recognizing and differentiating between image and payload packet data and creating a process to periodically output GPS coordinates to our Arduino to effectively track our payload. Our base station software can be understood through the following software diagram.

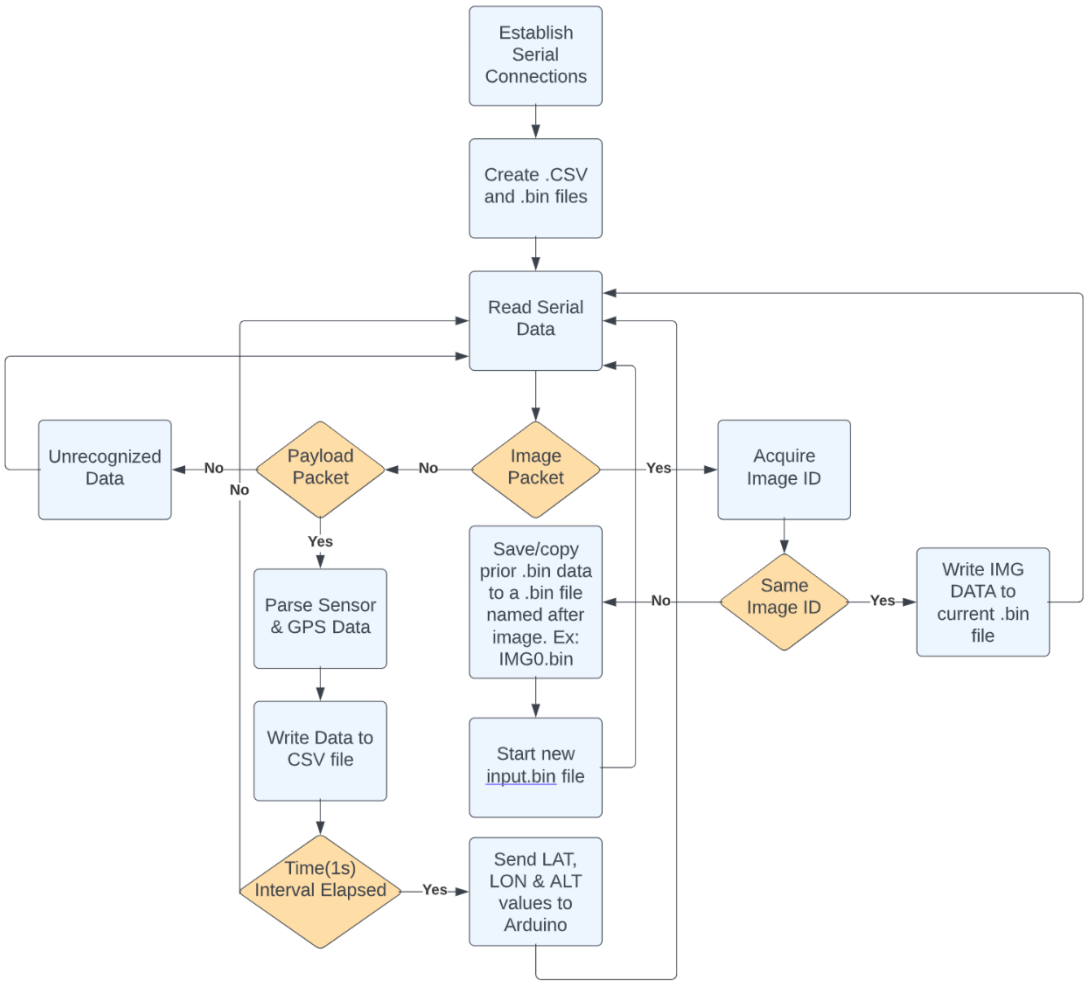


Figure 35: Base Station Software Diagram

Our base station computer connects via USB to both our radio module and Arduino. The radio module is receiving data that is passed via a serial connection to our program which checks whether the received data is an image or payload data packet and proceeds accordingly. Received payload packets are parsed for their GPS coordinates whose values are periodically (every 1 second) sent to the Arduino via a USB connection. The GPS coordinates are sent periodically to the Arduino to allow the motor

enough time to complete its rotation without creating a large queue in the serial buffer resulting in delayed responses to payload movement. Given the time which exists between sending the GPS data via serial connection, computing the antenna rotation value, and actuating the motor, there exists an additional delay beyond the 1 second timer which further ensures an adequate window for our Arduino program to respond accordingly.

3.4.2 Rotation

To ensure the functionality of the stepper motor, getting the motor to rotate was our priority. The stepper motor controller had some switches on the side to set the current limits and the pulses per revolution. According to our power supply, left from HAB 3.0, the amount of current it will supply is 2.5A. In turn, we will set our current limit to 2.37A on switches one through 3. Switch 4 is set to on to use full current, and finally switches 5-8 are set to the code for 6400 Pulses per Revolution, a reasonable pace, so the antenna does not move violently when the payload is moving through the air. With all these set, the armature resistances for both coils of the stepper were checked. A high resistance value shows that the motor is dead. Both coils came in at a resistance of 2.7Ω.

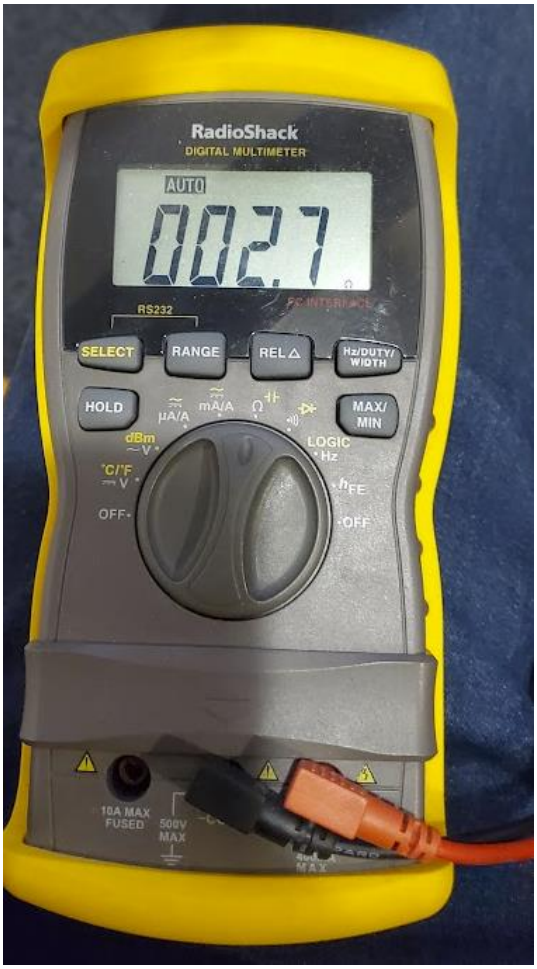


Figure 36: Resistance of the Stepper Motor Coils

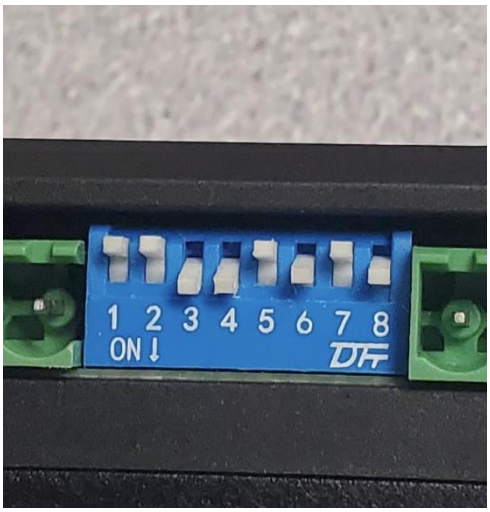


Figure 37: Dipswitch Configuration for Speed

Wiring up the motor to the controller required the 2 coils to be plugged into A+/A- and B+/B- respectively for each coil pair. The V+ barrel plug was connected to the 24V/2.5A power supply. Next came signals from our microcontroller of choice. To keep things familiar, we chose an Arduino Uno. There are digital I/O pins needed for the DIR- and PUL- ports. Arduino ports D6 and D7 are used for those 2, respectively. PUL+ and DIR+ are connected to the Arduino's 5V pin. The test code was a simple sketch setting the DIR- to HIGH and PUL- to HIGH as well to make the stepper motor rotate. Reversing the direction requires the DIR- pin to be set LOW.

```
/*
MQP Team 5
J. Lopez
Stepper Motor Test

Max Current Output from PSU 2.5A

Below is the Dip Switch Configuration on the DM542T Digital Stepper Driver

Max Current Table:
SW | Status
---+-------
1  | Off
2  | Off
3  | On      2.37A Max Draw

Current Limiter:
4  | On      Use Full Current

Pulses/Rev Table:
SW | Status
---+-------
5  | Off
6  | On
7  | Off
8  | On      6400 Pulses/Rev

*/
```

Figure 38: Explaining Each Switch

3.4.3 AccelStepper Library

Utilizing one of the many stepper motor libraries was necessary to get ours to turn to a specific degree angle. AccelStepper is a popular choice for this sort of application [12]. Taking us over to C++, we need to create an object to instantiate our stepper motor with the class already defined in the library.

```
// Define a stepper and the pins it will use
AccelStepper stepper(1,7,6);
```

Figure 39: Stepper Object

The stepper is defined by the first number declaring the mode. The second and third numbers correspond to the pins that the DUR- and PUL- are connected to on the Arduino. From here, we needed to specify the speed and acceleration of the stepper motor in the void setup. We will use 100 as a baseline. There are a variety of functions to get the stepper motor to move. We will use the turn run to new position function as we have a math calculation based on the number of degrees we want to rotate, and our pulses/rev being set to 6400. If we want 1 revolution, we will need to input 6400 for the integer intake of the function.

```
void setup()
{
    // put your setup code here, to run once:
    //Speed and Acceleration
    stepper.setMaxSpeed(1000.0);
    stepper.setAcceleration(1000.0);
}

void loop()
{
    // put your main code here, to run repeatedly:

    //stepper.runToNewPosition(0);
    stepper.runToNewPosition(6400);
    stepper.stop();
    // delayMicroseconds(500);
    // stepper.runToNewPosition(0);
    // stepper.runToNewPosition(6400);
}
```

Figure 40: Test Code

When creating the final Arduino program to run and control the rotation of the motor, we made use of the bearing function written by the HAB 3.0 team, which takes in two coordinates in the form of latitude and longitude pairs and computes the bearing between one point [5] to the other. This function was used in conjunction with our motor rotation functions to map bearing return values to rotation values and adjust our antennas by these particular amounts.

3.4.4 Pitch of the Antenna and Final Design

Many ideas surfaced on how the antenna was going to pitch upwards towards the payload. Pneumatics was one of our first ideas but the complexity of having a closed loop system along with having a compressor was too complicated and it would have made our base station more difficult to transport. Not only that, the cost of pistons, solenoids, tubing/fittings, and the compressor would have added up to our entire budget. Power is also a big requirement. The compressor would use most of our portable battery's charge. Another idea was the use of chains and sprockets, like on a bicycle. Ideally, a gear box is needed for this design and with some of us not being mechanically inclined we decided on choosing a linear actuator. Linear actuators work by having a motor that drives a lead screw. On the inside, a lead nut on screw extends shaft outwards. Upward and downward motion make it comparable to a piston but with no air [13]. To select a placement for it our final base station design must be made. A few key decisions need to be made.

1. The Rotation must be independent from the Pitch.
2. It must be sturdy but portable enough to transport to various sites.

Previously, the antenna was fixed to a PVC tube by a bracket which was then connected to a lazy Susan. The stepper motor turns the lazy Susan and is directly connected. To implement the pitch, either the entire station would have to tilt, or the fixed mounting of the antenna would have to be modified. The latter is smaller, allowing more flexibility to ensure portability. A new hole for a dowel was drilled through the existing bracket and the PVC tube. This gives the antenna freedom of rotation about this axis. Completing the triangle, the linear actuator will hold the antenna 90° when it is in a home position (not extended). We found this home distance to be about 10”, therefore, a linear actuator with a stroke of 10” will pitch the antenna out a further 10” diagonally. This axis will be fixed, and the extended length will make the antenna rotate about the axis with the PVC tube. The other end of the linear actuator will be connected to the PVC tube. This will make both systems independent. Final placement of the linear actuator was moved lower on the PVC tube and on the antenna itself with an L bracket. This was done to have the full length of the linear actuator. As before it would only expand 3" and then the antenna would fold back on itself. The new alignment will make a 4-bar linkage. The 4 bars in this linkage are the PVC tube, the top aluminum L bracket, the antenna, and the linear actuator. This makes our system more stable and easier to move, converting linear motion to rotational motion. With the linear actuator mounted lower, the antenna can go from 0 -> 90 degrees with 90 being the upper limit length of the linear actuator.

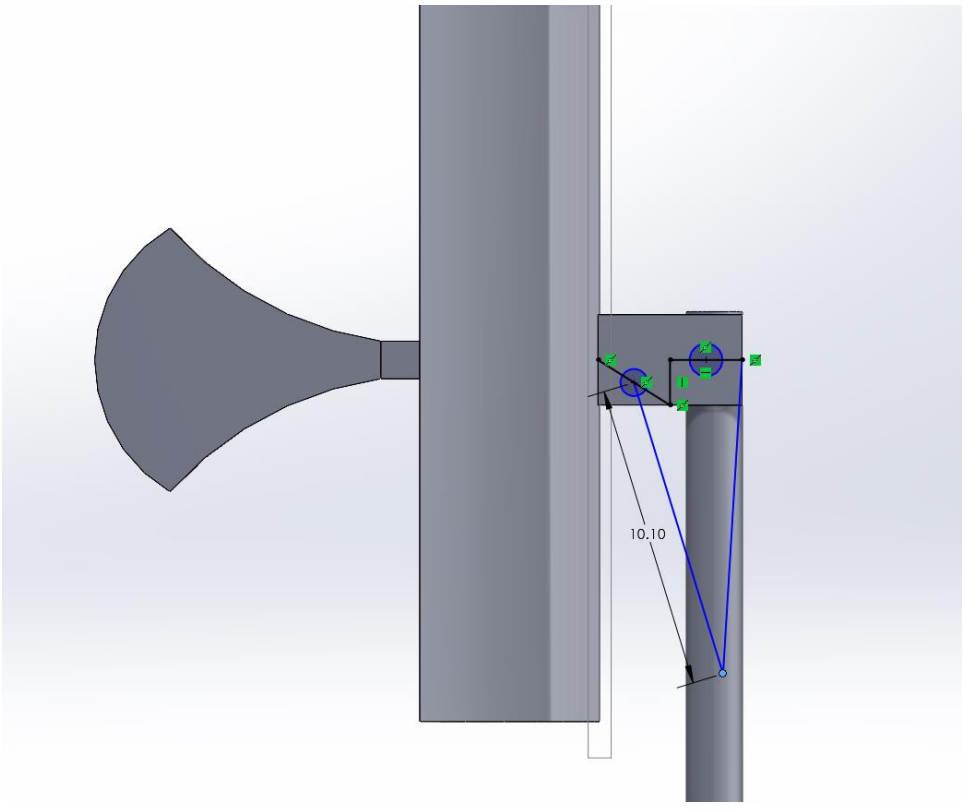


Figure 41: Old Mounting

Figure 42: New Mounting

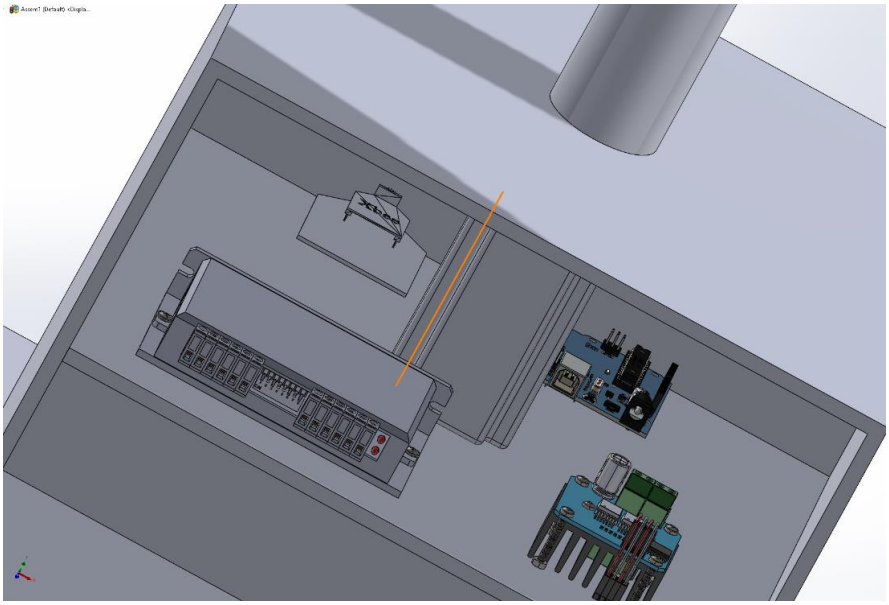


Figure 43: Electrical Board CAD

3.4.5 Linear Actuator

The 10” linear actuator we chose has the following specifications: 12V, 35 lbs., and a 10” stroke length. Like the stepper motor, we need a motor controller to take in our Arduino digital inputs. The IBT2 Hight Current H-Bridge Motor Driver is our controller of choice due to compatibility with our linear actuator and ease of use with Arduinos. We must assign digital Arduino ports like the stepper motor controller. D10 and D11 were used to go to LPWM and RWPM, respectively. These are for Right Pulse Width Modulation and Left Pulse Width Modulation which are the digital signals going from the Arduino to the controller. R_EN and L_EN are right and left enable pins - these will be tied to the Arduino’s 5V pin as well as VCC. GND will be connected to the Arduinos GND pin. V+ will be connected to the 12V power supply, and finally Motor+ and Motor- will be connected to the motor on the linear actuator. A KiCAD schematic was made for the whole base station.

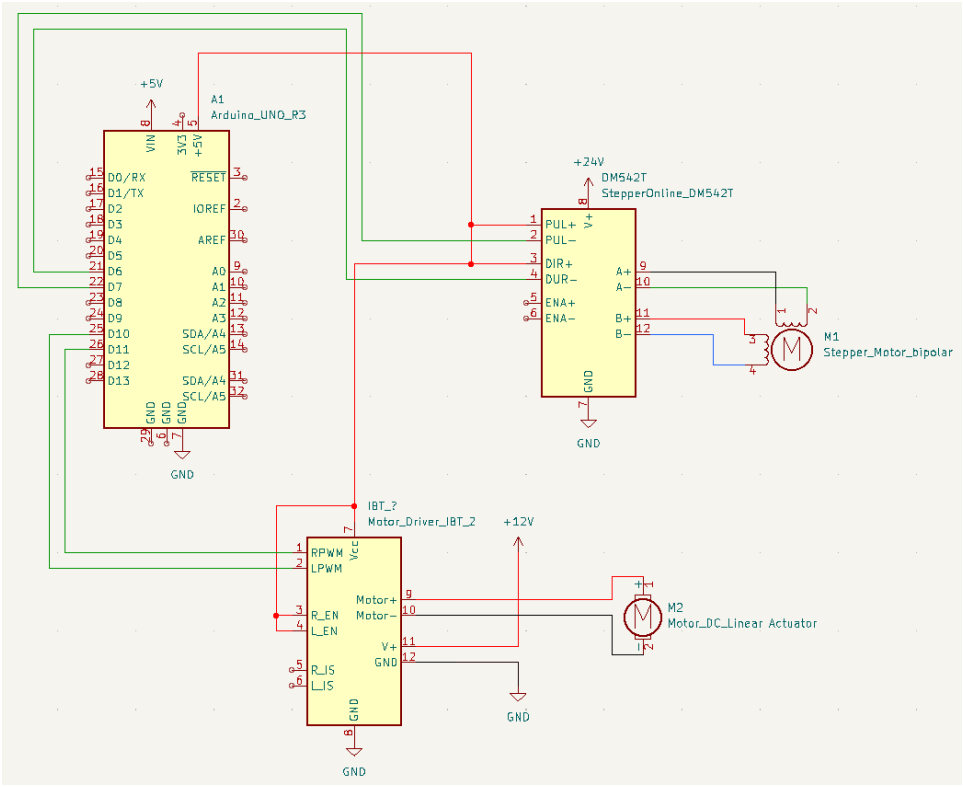


Figure 44: Base Station KiCAD

Like with the stepper motor, we needed to declare a speed for the linear actuator. Going for a smooth transition between angles, we set this speed to 100 pulses per sec. Speeds lower than this would not move the antenna far. This was determined by trial and error. The default speed was 256. Using the analogWrite() function built into Arduino, we need to specify a port and an integer value. In our case, it is the speed value, which is declared as a global variable. The system is time–based, as there is no feedback on position in the middle. There is feedback when it reaches either end. There are two internal limit switches, one on each end to tell the motor to not rotate anymore once it reaches that point. Early testing consisted of moving the linear actuator to its fully extended length, then retracting it and determining a safe speed.

Figure 45: Linear Actuator Fully Extended using a Variable Supply

A time-based system is not the best in terms of control. Final implementation went as follows. The stroke of the linear actuator is 10” and since it reaches a full 90°. We divided the angles into distinct positions. Increments of 90°/10in = 9°/in. This yields to having the linear actuator extend 1" for every 9° the payload altitude angles increase. We know the time it takes for the linear actuator to extend and retract 1". It takes 1.7s to extend 1" and 1.3s to retract 1" due to help from gravity. The angles are calculated by taking the diagonal distance between the payload coordinates and the base station coordinates. This then forms a triangle with the payload in the sky. Using the altitude of the payload and the diagonal distance we found on the 2d triangle on the ground, using latitude and longitude differences, we can use the inverse tangent (arctan(X)) function to find the angle. However, the inverse tangent function is only positive in the 1st and 3rd quadrants. If we have a negative in one of our latitude, longitude, or altitude differences, we will get a negative angle. This is why the values are put in an abs() function for absolute value. In this case, we used fabs() for floating point numbers. Converting to degrees from radians will yield the angle at which to point.

This angle was then taken by the split function to split it into 10 distinct positions. The last 4 positions are commented out due to the length of our coaxial cable not being long enough to sustain anything more than 54 degrees. In the main program, the default position is facing straight horizontal (0°). The pointer function runs, and it returns a new angle to point at. An old position variable is initialized to the current position. The position is then updated with the new angle from the split function. Then an if statement compares the two and if it is greater, it uses the extension rate. If it is less, then it uses the retraction rate. It only increases or decreases by 1", so we have complete, stable control on the antenna pointing.

```
// 10in stroke
// 90deg/10in = 9deg per in
// 17 second up
// 13 seconds down
// 17/10 = 1.7s/in up
// 13/10 = 1.3 s/in down
// 3ft offset from the ground
float pointer(float pay_alt, float pay_lat, float pay_lon, float bs_alt1){
  //take the difference of the payload coords to make a triangle
  float lonDiff1 = pay_lon - bs_lon;
  float latDiff1 = pay_lat - bs_lat;


  // Absolute Value for positive angle
  float lonDiffFabs = fabs(lonDiff1);
  float latDiffFabs = fabs(latDiff1);


  // solve for Hypotenuse or the diagonal distance which will be the adjacent side when find the angle.
  float Hypotenuse1 = (pow(latDiffFabs,2)) + (pow(lonDiffFabs,2));
  float diagdist=sqrt(Hypotenuse1); //adjacent
  //now that we havre the adjacent side we can use tangent for opposite(payload_alt)/adjacent(diagdistance)
  float angle = atan((pay_alt-bs_alt1)/diagdist); //returns angles in radians opposite/adjact (TOA)
  float angledegree = angle * (180/pi); //conversion to degrees
  return angledegree;
}
```

Figure 46: Angle Calculation

```
int split(float angledegrees){
  int position;
  if (angledegrees <= 9){
    position = 1;
  }
  else if (angledegrees >= 10.0 && angledegrees <= 18.0){
    position = 2;
  }
  else if (angledegrees >= 19.0 && angledegrees <= 27.0){
    position = 3;
  }
  else if (angledegrees >= 28.0 && angledegrees <= 36.0){
    position = 4;
  }
  else if (angledegrees >= 37.0 && angledegrees <= 45.0){
    position = 5;
  }
  else if (angledegrees >= 46.0 && angledegrees <= 54.0){
    position = 6;
  }
//    else if (angledegrees <= 55 && angledegrees >= 63){
//      position = 7;
//    }
//    else if (angledegrees <= 64 && angledegrees >= 72){
//      position = 8;
//    }
//    else if (angledegrees <= 73 && angledegrees >= 81){
//      position = 9;
//    }
//    else if (angledegrees <= 82 && angledegrees >= 90){
//      position = 10;
//    }
  return position;
}
```

Figure 47: Position Split

```
  //Linear Actuator
  int oldposition = position;

  float angledegs = pointer(pl_alt, pl_lat, pl_lon, bs_alt);
  position = split(angledegs);
  //Serial.println(position);

  if (position > oldposition){
    //Extend Actuator 1in
    analogWrite(RPWM, Speed);
    analogWrite(LPWM, 0);
    delay(1700); //1.7 seconds to go up 1in

    // Stop Actuator
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
    delay(1000);

  } else if(position < oldposition){
    // Retract Actuator
    analogWrite(RPWM, 0);
    analogWrite(LPWM, Speed);
    delay(1300); //1.3 rate doing down 1in

    //Stop Actuator
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
    delay(1000);
  }
}
```

Figure 48: Linear Actuator Logic

3.4.6 Construction of the Base Station

Construction first began on January 29th, 2023. A 2'x4'x.75" piece of plywood was purchased to be the base of the base station. For the walls and the roof, our team had various materials in mind, but we ended up choosing Lexan, a non-shatter polycarbonate material. This gives the base station a sleeker look with a view of the inside brackets and electrical components. With this decision in mind, a 2'x4'x.25" piece of Lexan was purchased and cut into 1'x1' square tiles to make the box and the door. Cutting these tiles was done using a Sawzall. L-Brackets were used to hold the 3 walls, the roof, and the

base together. All hardware for the base station was 10-32. 10-32 is an imperial screw size with the 10 for the size designator and 32 meaning 32 threads per inch.

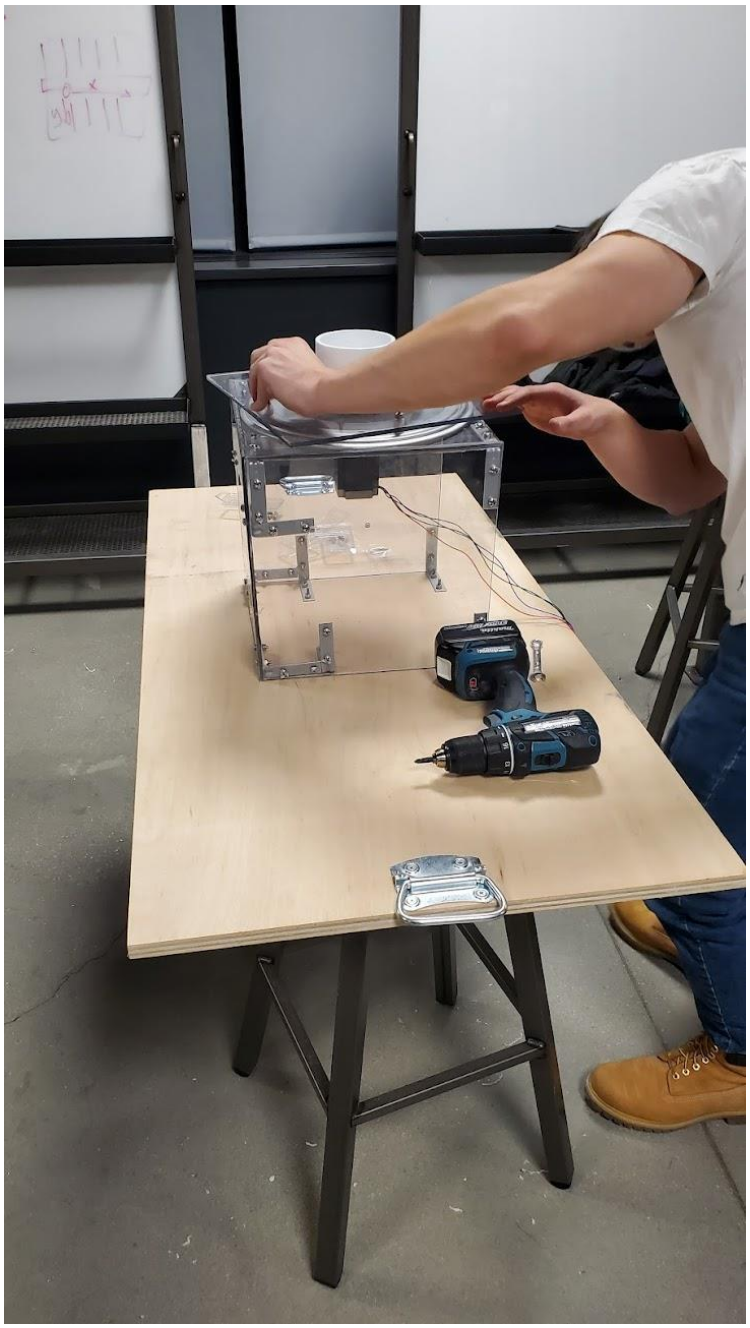


Figure 49: Construction of the Base Station

A later edition of a door and handle was added for more security for the storage unit underneath the electrical board. The lazy susan was mounted to another 1'x1' tile of Lexan. The motor hub and toilet flange for the PVC tube were mounted to this tile. On the roof of the Lexan box, there is an indexing hole right above the stepper motor controller to take off the top tile to service the motor hub, toilet flange, and the lazy susan. You need to spin the lazy susan to align the indexing hole with the screw holes for the top tile. External holes were drilled for the USB-B Port for the Arduino, Micro-USB

for the X-Bee radio, Coaxial Connection also for the XBee, and two-barrel plugs for the linear actuator and the stepper motor.

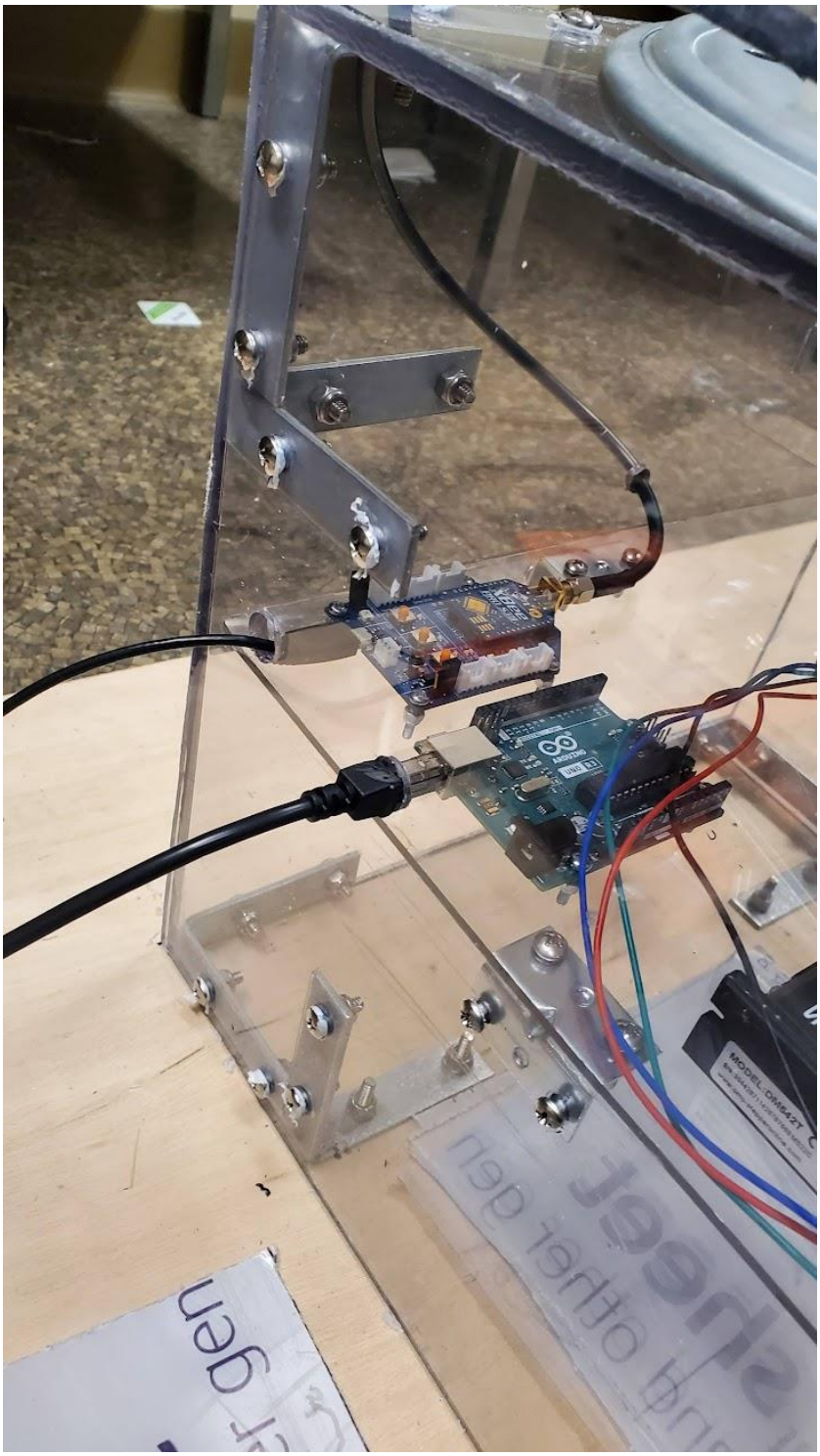


Figure 50: Base Station Port View

The electrical board was made from a thinner sheet of Lexan (only 1/8” thick) as it did not need to be as thick as the rest of the base station. It was also easier to manipulate. Each component on the board was mounted with assorted sizes of hardware. The XBee and Arduino used M3 (Metric screw with a width of 3mm), The Stepper Motor Controller used 6-32 (Imperial Size 6 and 32 threads/in), and the protoboard as well as the linear actuator motor controller used 4-40 (Imperial Size 4 and 40 threads/in). The side L-brackets holding up the electrical board are 8-32 (Imperial Size 8 and 32 threads/in).

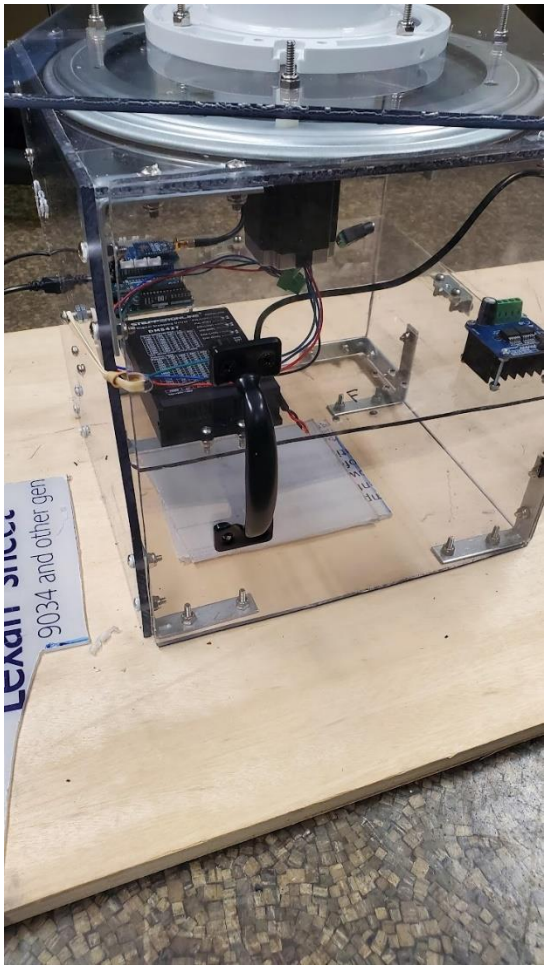


Figure 51: Door and Storage Bay

Two handles were added to each end of the base station for ease of transport. This was increased from one on each end as the base plate was rotating when picked up due to the antenna. The antenna is also detachable during transport. The coaxial cable can be unscrewed, and the linear actuator power wires are on quick disconnectors call Anderson Power poles. In addition, the handles flipped down also proved a place to put the garden stakes when the base station is on the road and needs some support. Four corners were on the top tile to alleviate cable strain and the interior is cable managed.

Figure 52: Handles

3.4.7 Tracking the Payload

To track the payload, our team made use of accurate trigonometric formulas to determine the distance and respective angle between our payload and the base station. After many iterations of researching and experimenting with formulas of our own, our group decided to incorporate a bearing formula written by the HAB 3.0 team to obtain the bearing for which the antenna should be oriented to point in the direction of the payload given its coordinates relative to the base station's coordinates. This was mainly due to a slight difference in accuracy. At an elevated level, our tracking system functions in the following manner:

Our base station computer, which is attached via USB to our radio module, reads in all transmitted data via a serial connection to a radio module and parses the longitude and latitude coordinates for our payload. After parsing these values, the base station computer program then writes them to the Arduino Uno via USB serial. Our Arduino program then receives these values and computes the respective angle for rotation. This value is then mapped to an integer value input for our respective motor function, which ultimately changes the position of the antenna.

After each rotation, we implemented a delay on our Arduino Program which stalls the program upon the serial buffer not having any data in its queue. Once a new set of coordinates is received, then the calculation for determining rotation amount is performed and sent to the motor for actuation.

Upon testing our final system, we confirmed the tracking system's functionality as expected. Due to the high winds at the test site and friction of the lazy susan plate of our base station, some computed rotations were unable to fully complete which resulted in all future rotations being inaccurate as well. This is because the station program would work off the assumption that the prior rotation was fully completed.

4. Results

4.1 First Full System Test (WPI Football Field)

The first full system test of HAB 4.0 was conducted at WPI's football field, since it provided an open, flat space upon which to do a basic test.

4.1.1 GPS Coordinates

Figure 53: Full System Test #1 GPS Coordinates

The above figure displays the recorded GPS path of the payload. Along with earlier tests done to see if the ETS triggered correctly, plotting the recorded coordinates put into perspective the accuracy of the GPS module. The shape of the walked path was a circle around the base station, which was placed along the middle of the field a few yards southwest of the center. However, the coordinates recorded don't seem to reflect this path for the most part. This shows us that on a small scale, the GPS module has limited accuracy due to its degree of position variability.

4.1.2 Pressure and Altitude Sensor Readings

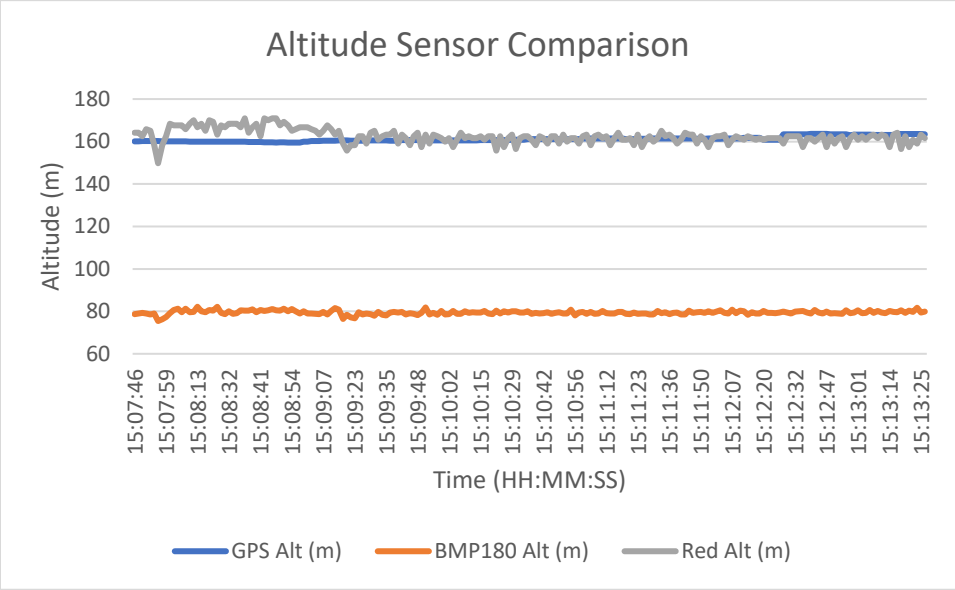


Figure 54: Full System Test #1 Altitude Sensor Readings Comparison

As seen in Figure 54, the altitude sensors output near consistent values, which is good considering the payload's altitude never changed during this test. However, the BMP180 sensor read a constant value of roughly 80m altitude, which is incorrect given the altitude of the WPI football field is between 160 - 170m. In the legacy code left from HAB 3.0, the BMP180 did not have any calibration code, but it seems as though this sensor was in need of tuning to work correctly (The Red Alt sensor needed a manual input of the starting altitude to function). The two other altitude readings, from the Red Pressure sensor and the GPS module, can both be considered valid, but it seems as though the Red Pressure sensor was less variable and thus slightly more accurate. GPS altitude in general is not as accurate as an actual processed reading of air pressure, so this was to be expected. Another point of note in the altitude sensor graphs is a sharp dip in altitude at 15:07:58. This dip was experienced by all the pressure-based sensors, so it may have been a gust of wind or other air phenomenon that caused the slight deviation. During this test, it was confirmed that the Red Pressure sensor should be the primary altitude sensor going forward.

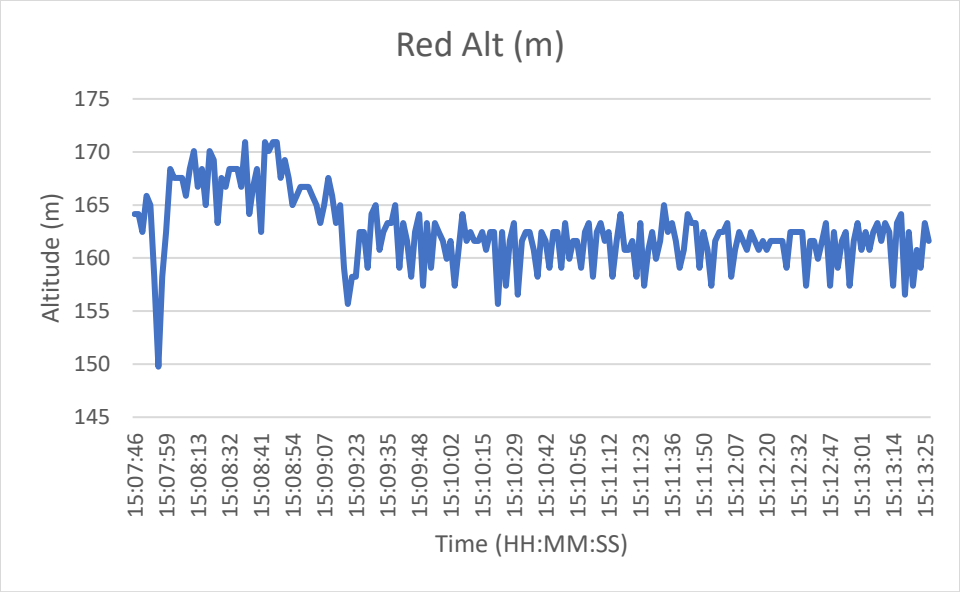


Figure 55: Full System Test #1 Red Altitude Readings

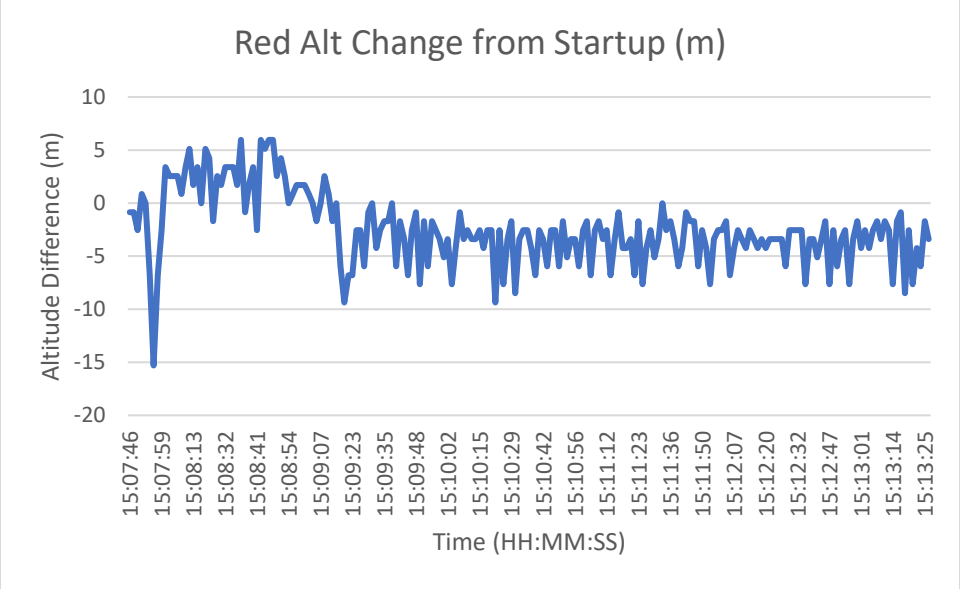


Figure 56: Full System Test #1 Red Altitude Change from Startup

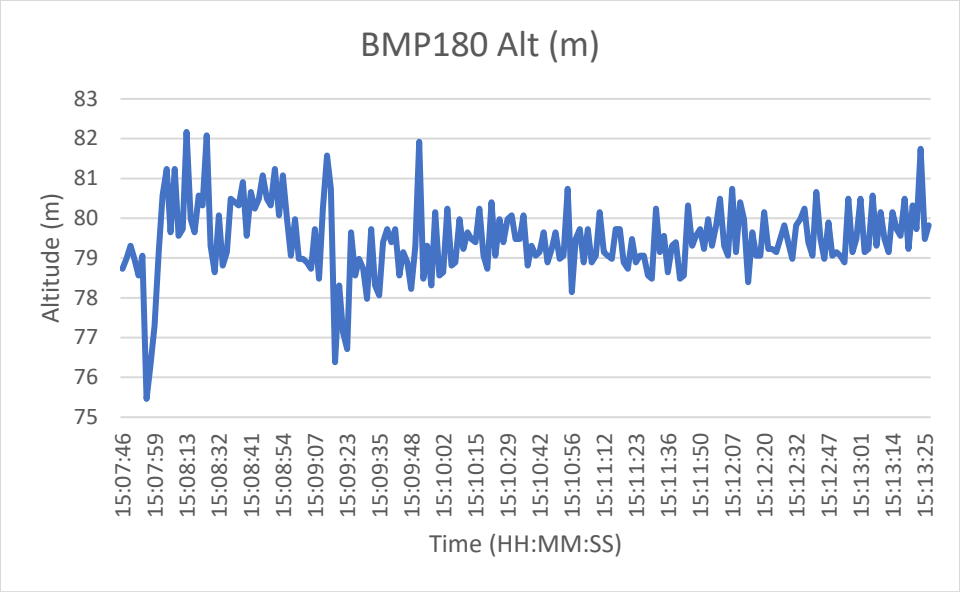


Figure 57: Full System Test #1 BMP180 Altitude Readings

The BMP180 readings were shifted down about 90m from the actual altitude, and thus were not considered for analysis.

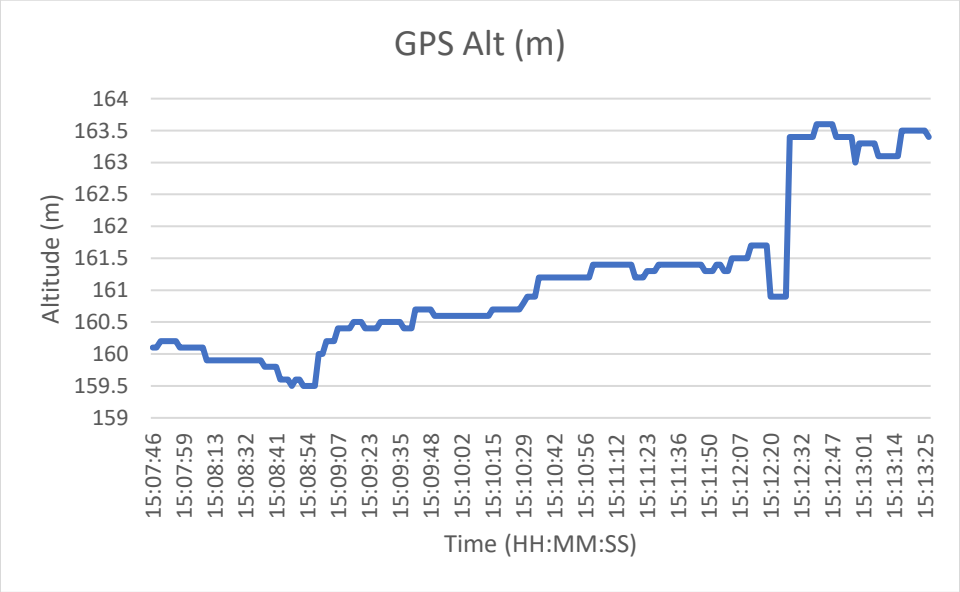


Figure 58: Full System Test #1 GPS Altitude

The GPS altitude steadily rose over time, but generally its readings stayed accurate enough to the current altitude.

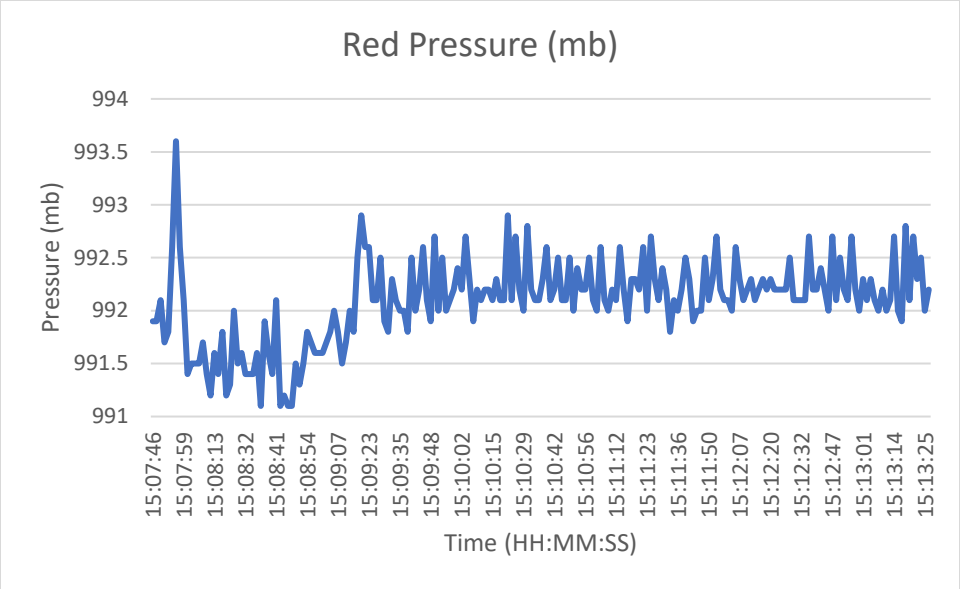


Figure 59: Full System Test #1 Red Pressure Readings

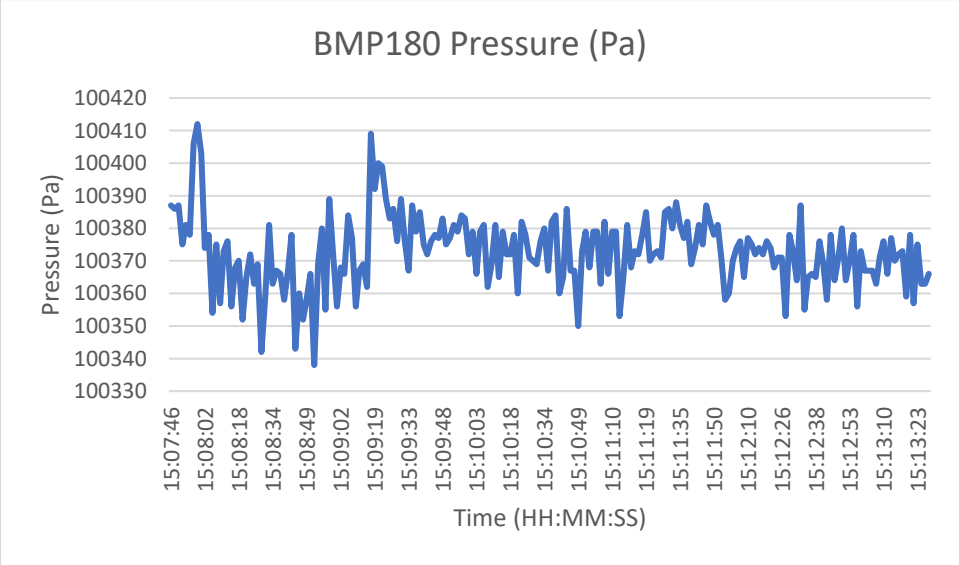


Figure 60: Full System Test #1 BMP180 Pressure Readings

Since pressure is inversely proportional to altitude, it makes sense that the pressure graphs for the altitude sensors have mirrored shapes compared to the altitude graphs.

4.1.3 Gas Sensor Readings

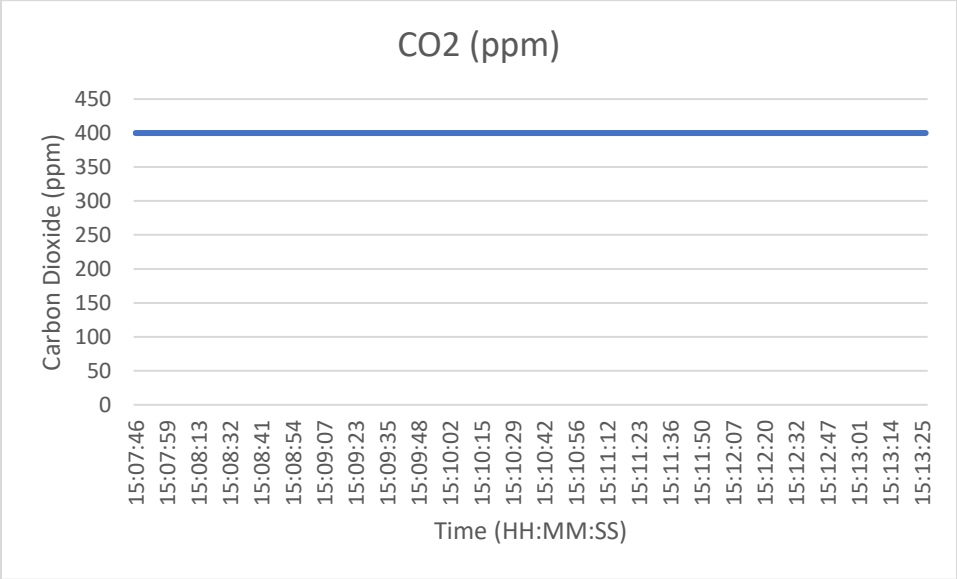


Figure 61: Full System Test #1 CO2 Readings

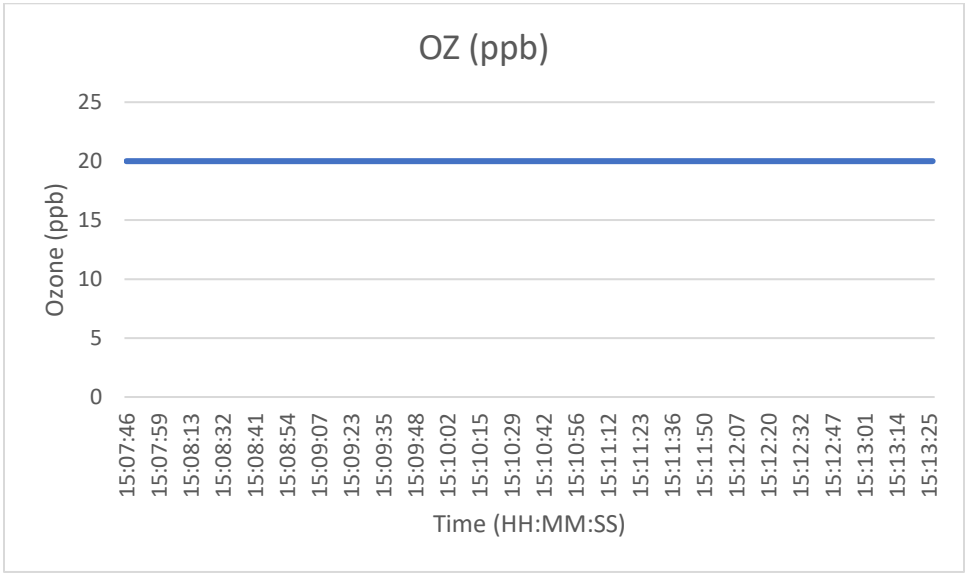


Figure 62: Full System Test #1 OZ Readings

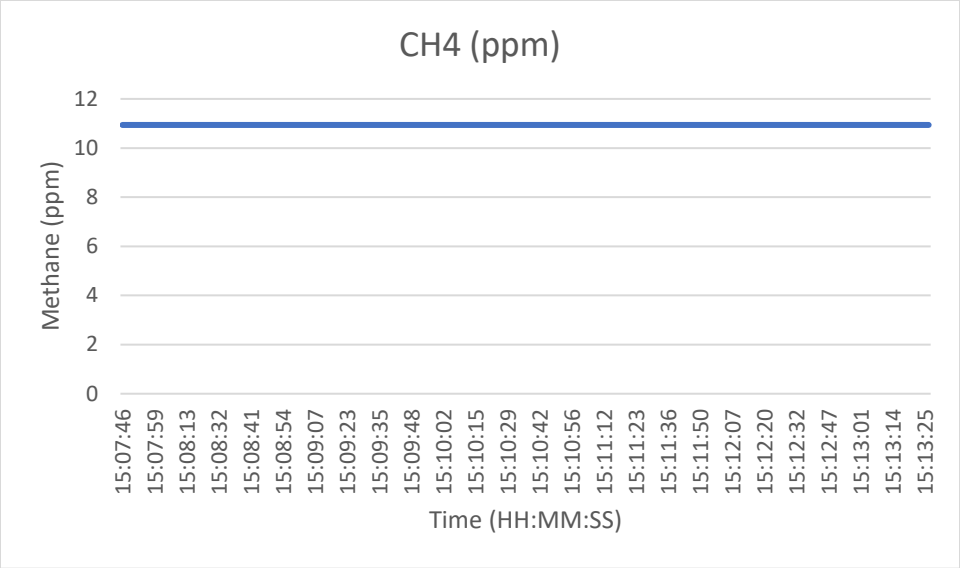


Figure 63: Full System Test #1 CH4 Readings

All of the gas sensor readings were constant since there simply was no exposure to these gases during the test. Any non-zero values seen are default values due to the nature of the sensors not being able to reliably sense less than that amount of gas.

4.1.4 Miscellaneous Sensor Readings

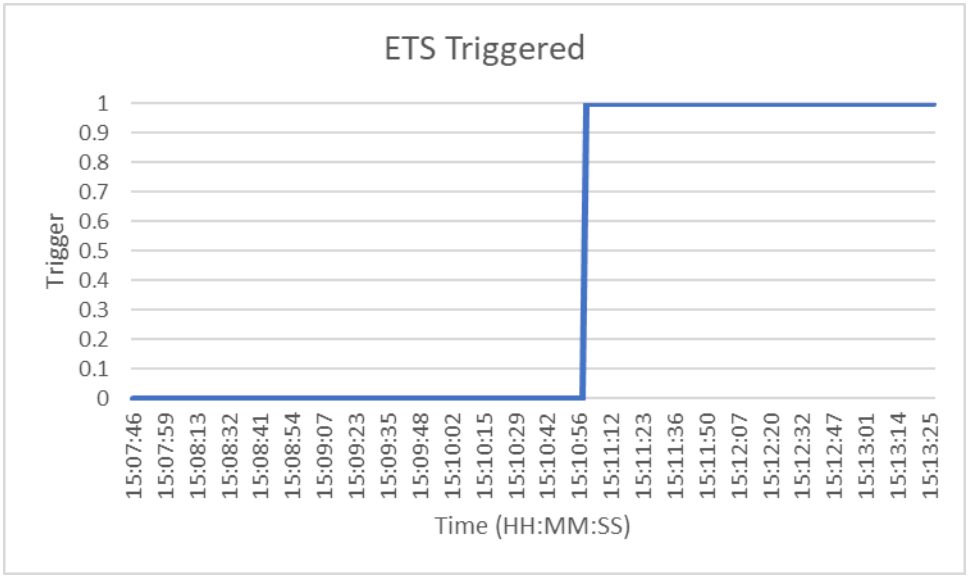


Figure 64: Full System Test #1 ETS Trigger Status Over Time

The above figure represents the status of the ETS, changing from 0 to 1 when it is triggered during a run of the payload code. During this run, the ETS would have been triggered by exiting the GPS barrier at 15:10:56.

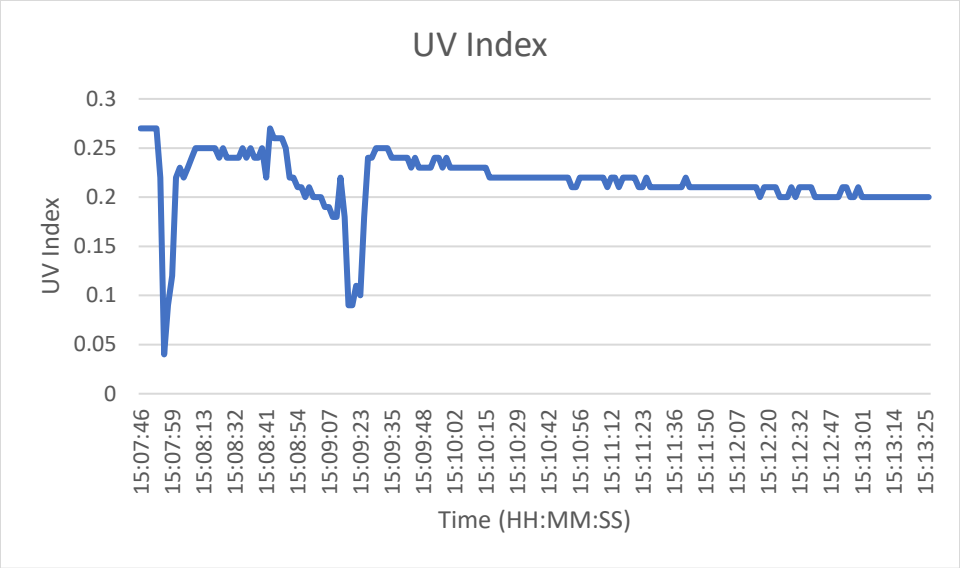


Figure 65: Full System Test #1 UV Index Readings

The UV index read by the payload generally stayed constant, likely only dipping when the sensor was covered by the payload carrier's hands or when it was out of view of the sun.

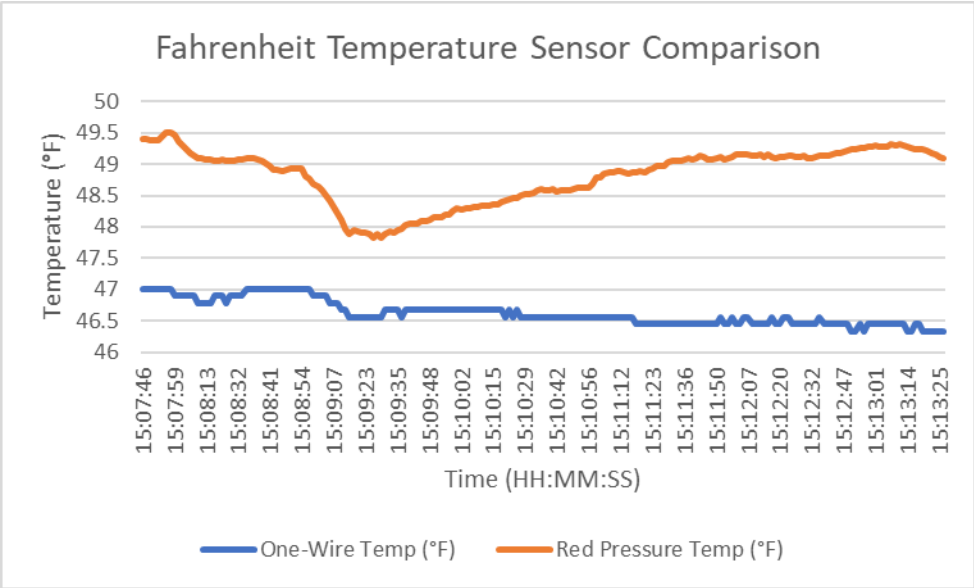


Figure 66: Full System Test #1 Fahrenheit Temperature Sensors Reading Comparison

In a comparison of the temperature sensors which output Fahrenheit values, the one-wire temperature sensor won out over the Red Pressure sensor's readings. The one-wire temperature sensor was both more consistent, and closer to the actual temperature of the day. The hypothesis behind why the Red Pressure sensor reads higher is because it is situated next to other sensors turning electricity into heat, mainly the gas sensors, which drives up its readings slightly.

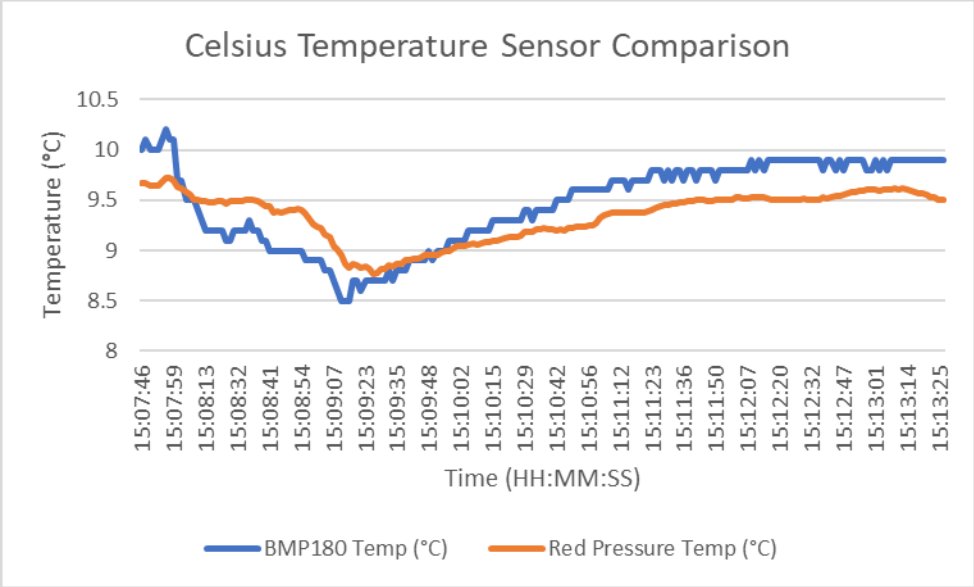


Figure 67: Full System Test #1 Celsius Temperature Sensors Reading Comparison

The Celsius temperature sensors were similar in performance, and likely are influenced by the heaters of the gas sensors as mentioned previously. When comparing all the sensors, the one-wire temperature sensor clearly takes the cake for having the most accurate readings.

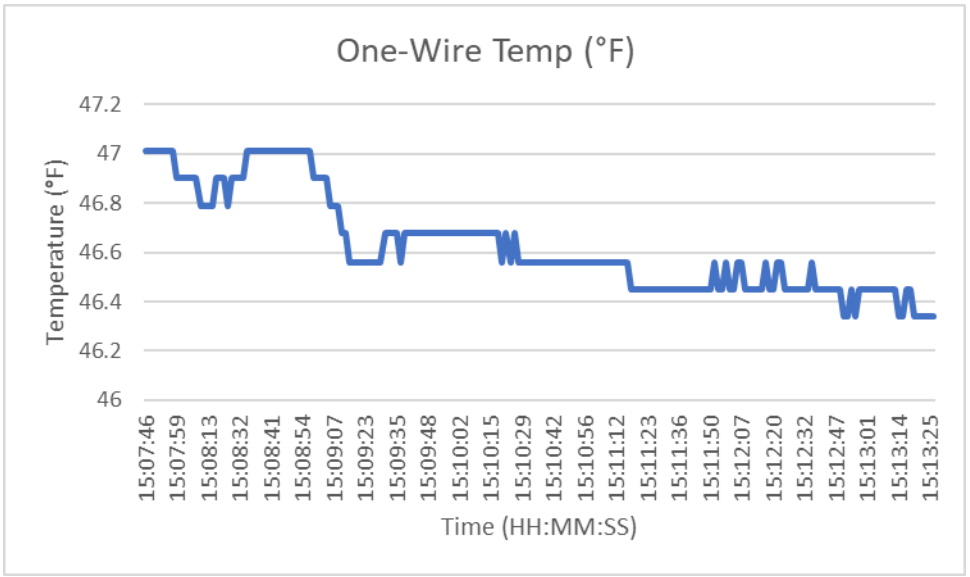


Figure 68: Full System Test #1 One-Wire Temperature Sensor Readings

The one-wire temperature sensor outputs a digital signal, hence why the graph seems to alternate between discrete values. The temperature drops as the test goes on, which makes sense given this test was done during the afternoon as sunset was approaching.

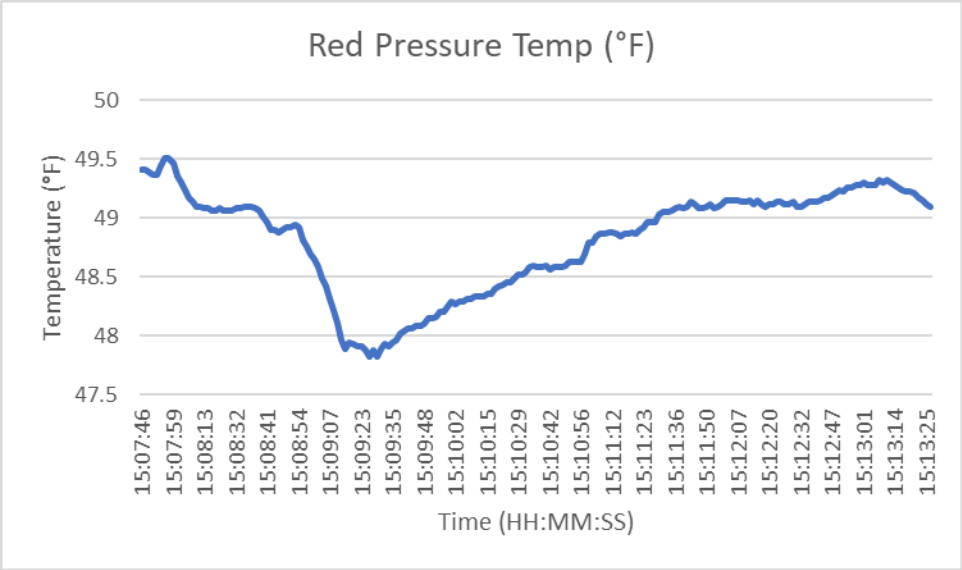


Figure 69: Full System Test #1 Red Pressure Temperature Readings in Fahrenheit

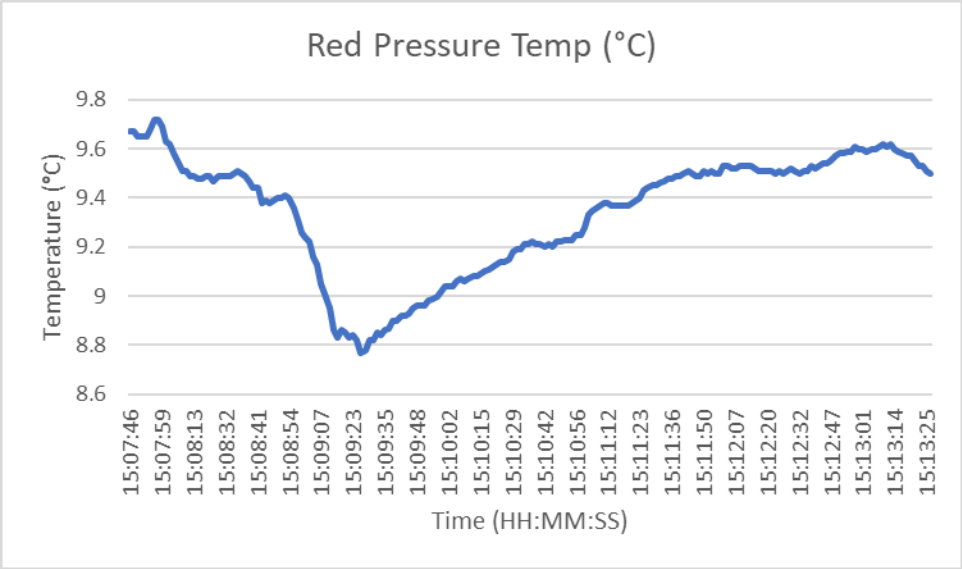


Figure 70: Full System Test #1 Red Pressure Temperature Readings in Celsius

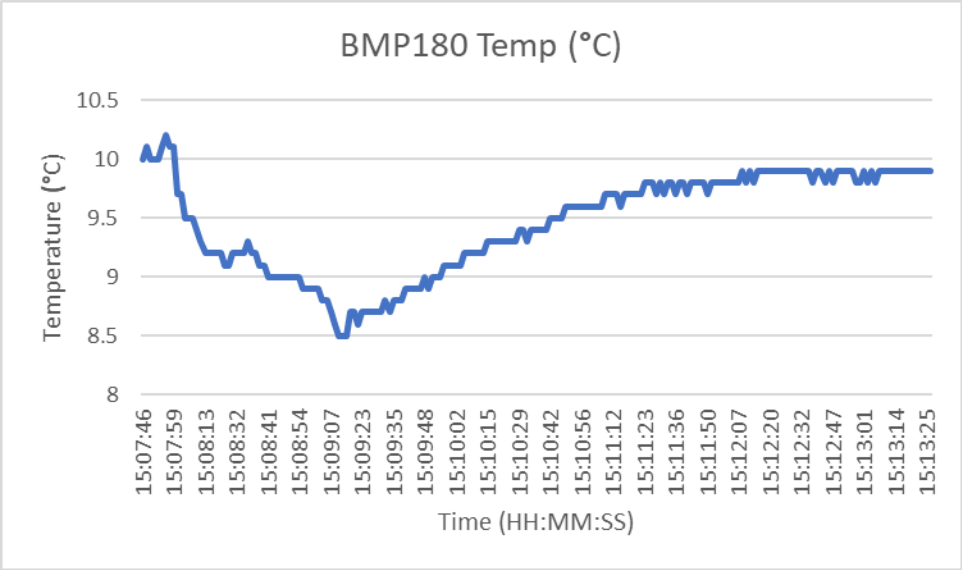


Figure 71: Full System Test #1 BMP180 Temperature Readings

The other temperature sensors did not capture the gradual drop in temperature over the course of the test, and thus cannot be considered as accurate as the one-wire sensor.

4.1.5 Base Station Performance

The base station did not perform up to expectations. The error in the linear actuator was due to our physical location on planet Earth. Being in the Western Hemisphere yields a negative longitude value with our location being to the left of the Prime Meridian. This means that some of our differences in the diagonal distances were outputting negative values into our inverse tangent function. A lesson back in high school Geometry will yield that tangent is only positive in the first and third quadrants. A positive angle will result from either positive values or both negative values. One of our values was negative and was making our angle return a negative value.

4.2 Second Full System Test (Green Hill Park, Worcester MA)

The second full system test was conducted at Green Hill Park in Worcester, Massachusetts since the system still needed to be tested over longer distances and also over an altitude difference. The park offered us altitude differences of up to 30m, and a distance of more than 260m. During this test, the payload was run for approximately 1 full hour, and all graphs were generated from this hour of data. To differentiate between the testing that occurred on ground level near the base station and the later testing with the payload taken to a nearby hill, a separate graph for each scenario is given for each measurement.

4.2.1 GPS Coordinates

Figure 72: Full System Test #2 GPS Coordinates

During this test, the payload was taken around the base station many times at ground level first, and then taken up a hill to the west and walked around atop it. Unfortunately, it seems as though the GPS stopped providing accurate messages to the payload code about when the team entered a car to bring it up the hill. For the entire hour of the test, all coordinates recorded were in the general area of the base station, as shown above in Figure 72. This is definitely a problem to investigate in the future, as it could compromise the tracking of the payload during an actual flight.

As for the accuracy of the GPS before the car was entered, it was generally accurate save for some minor miscalculations. For example, the south and southwest of the map in Figure 72 was walked with the payload twice or more, but there are no coordinates that indicate that area was entered. All other routes have recorded coordinates. Notably, there was more cover by trees in the unrecorded route which may have influenced the GPS module. The hill to the west was clear of trees however, so this excuse would not apply to the hill's false coordinates.

4.2.2 Pressure and Altitude Sensor Readings

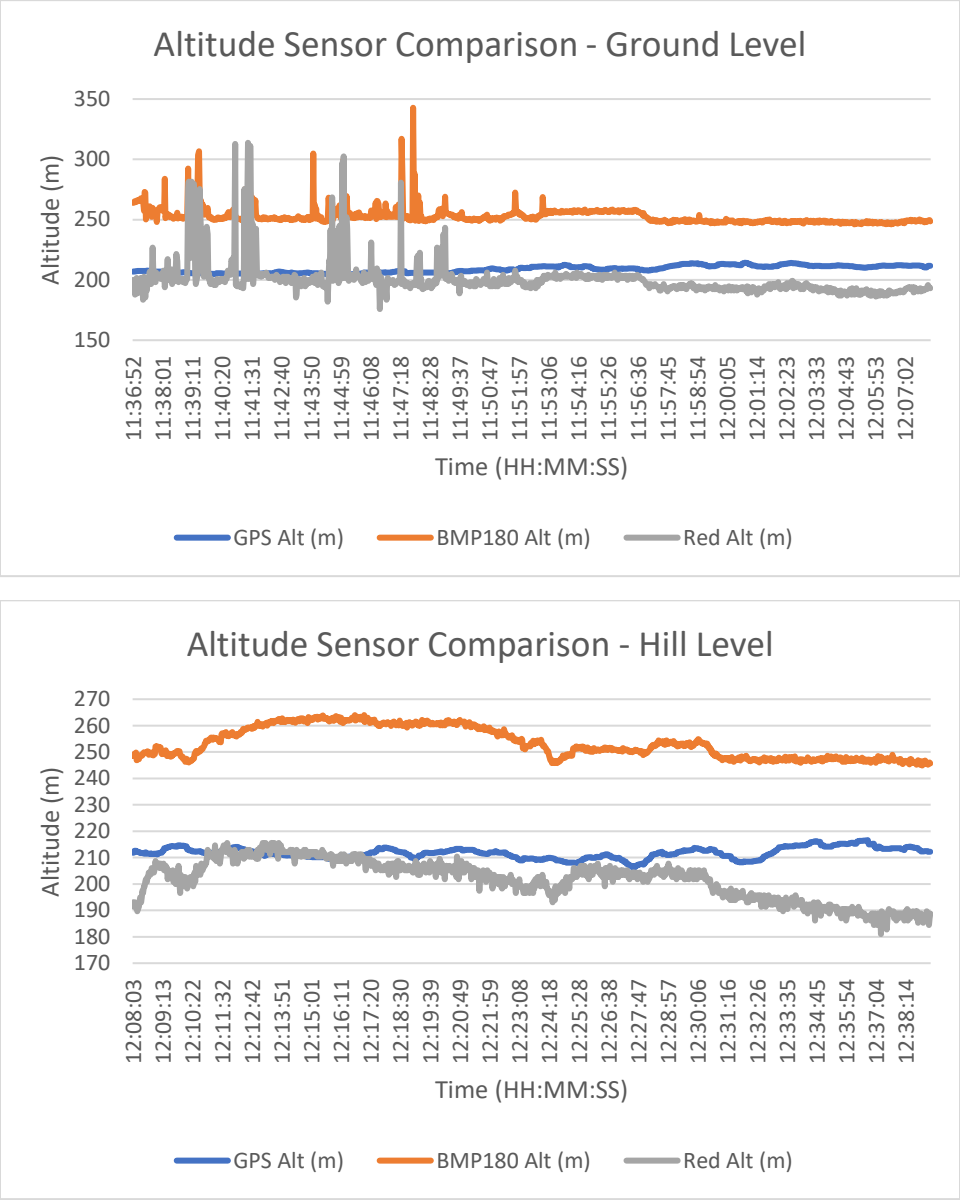


Figure 73: Full System Test #2 Altitude Sensor Readings Comparison

The altitude sensors worked almost as expected based on the results of the first test, with the exception of the BMP180 sensor now outputting higher rather than lower. In the beginning of the run, there were a few hiccups in altitude, which were again probably caused by wind given that the GPS sensor seems to have stayed completely unaffected. This proves that the GPS sensor may be able to be used for primary readings in the event of a dramatic increase in the pressure-based altitude sensor readings. The elevation was not constant around the base station so there is some fluctuation in the readings through the first half, and shortly after the halfway mark is when the payload was taken atop a roughly 20m tall hill. The Red Alt continued to output the best results overall, tracking the changes in altitude perfectly with the exception of the wind gust spikes.

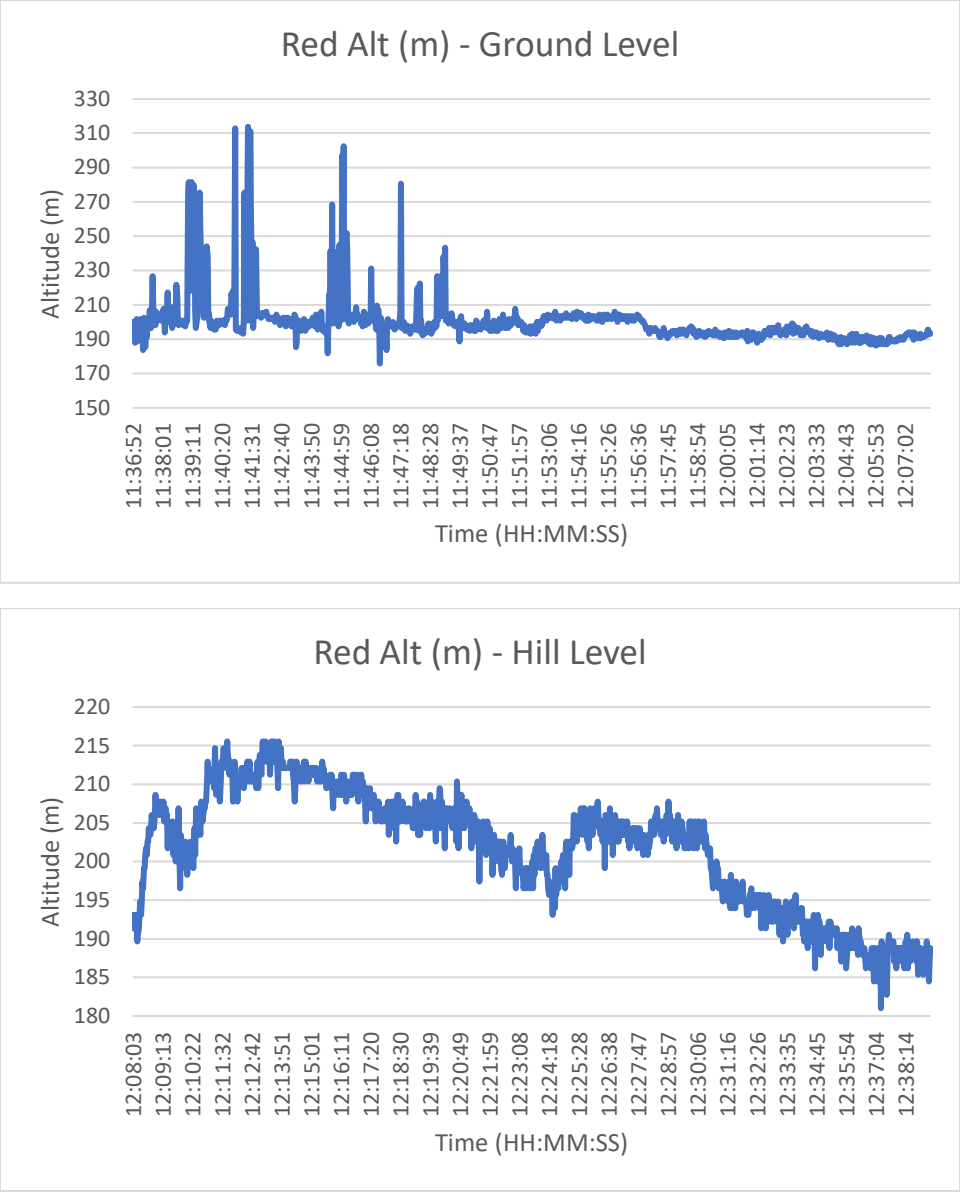
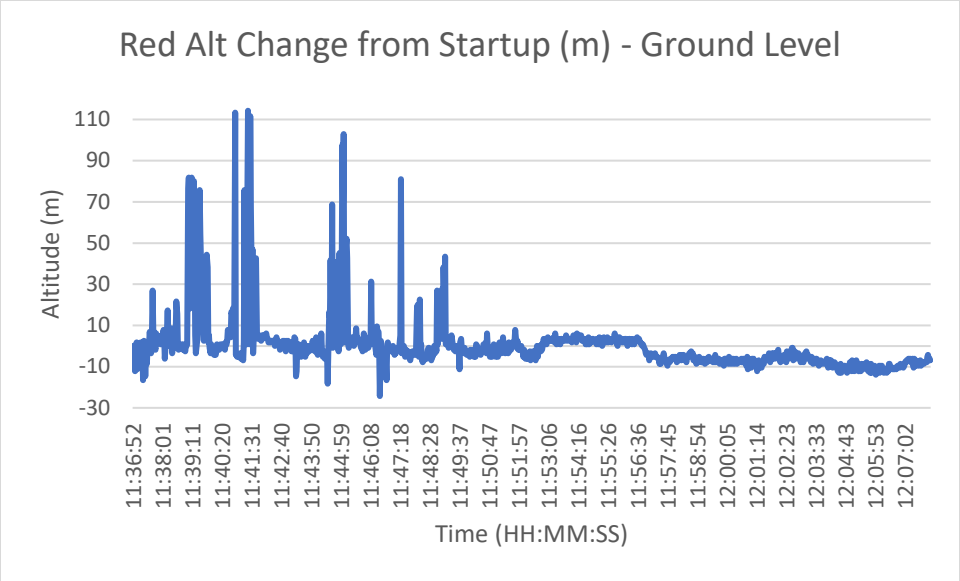


Figure 74: Full System Test #2 Red Altitude Readings



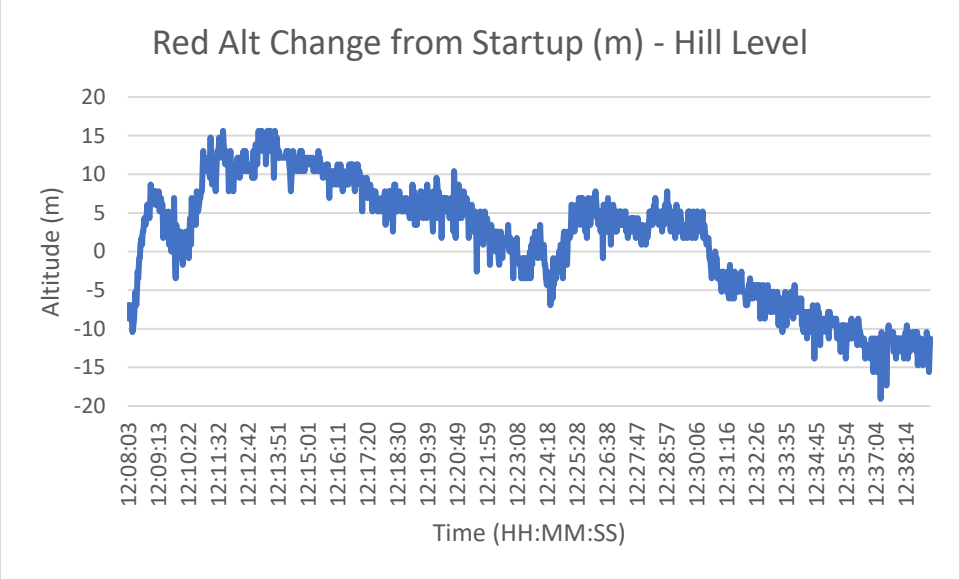
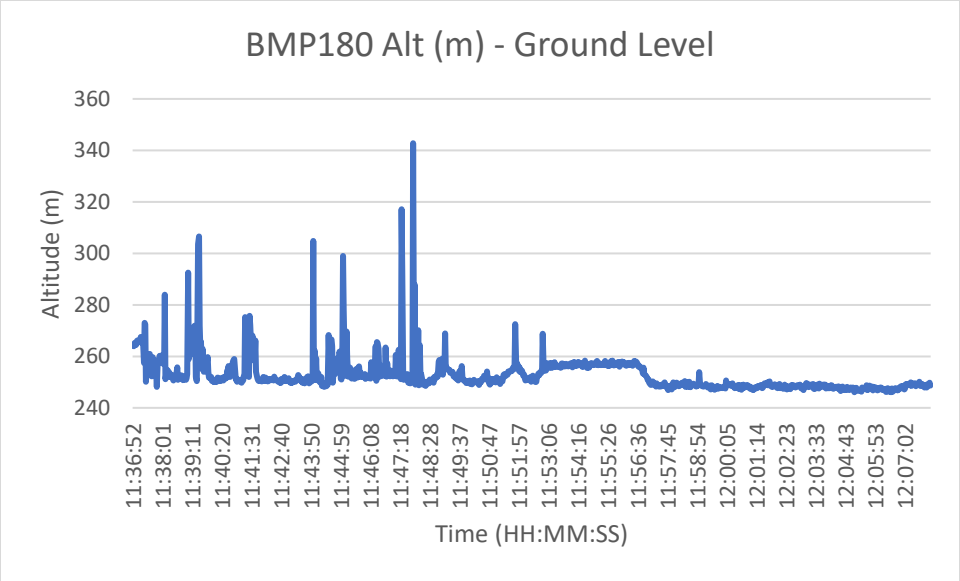


Figure 75: Full System Test #2 Red Altitude Change from Startup



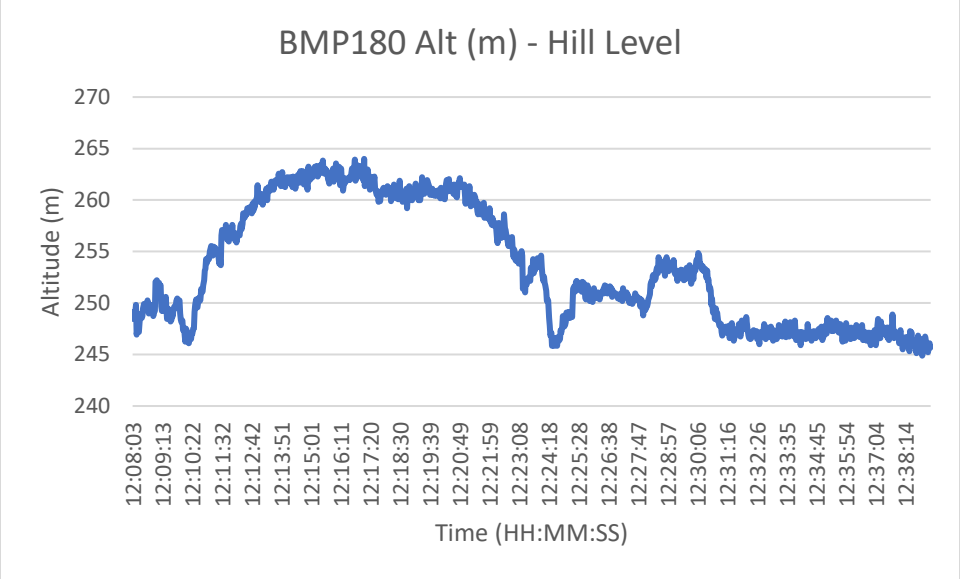


Figure 76: Full System Test #2 BMP180 Altitude Readings

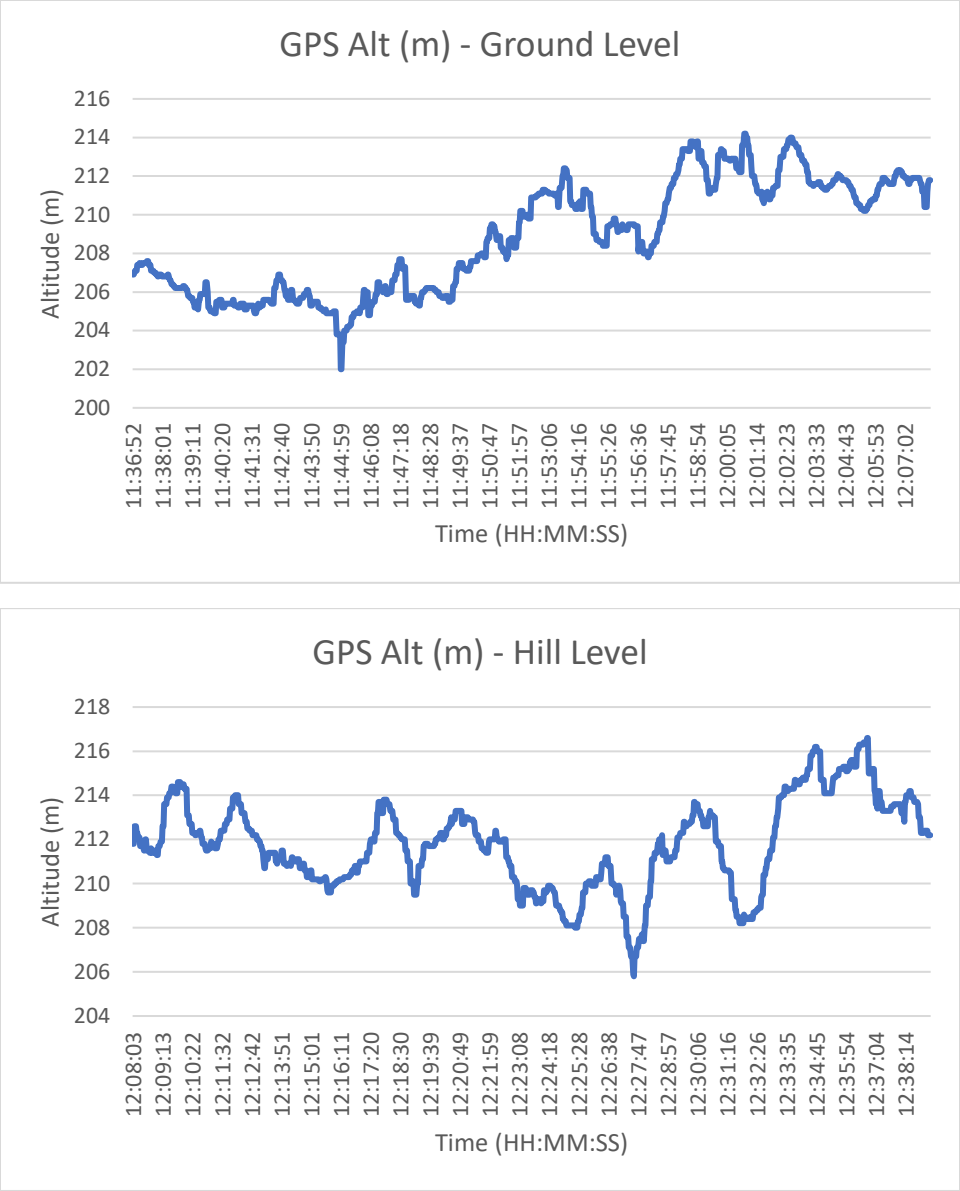
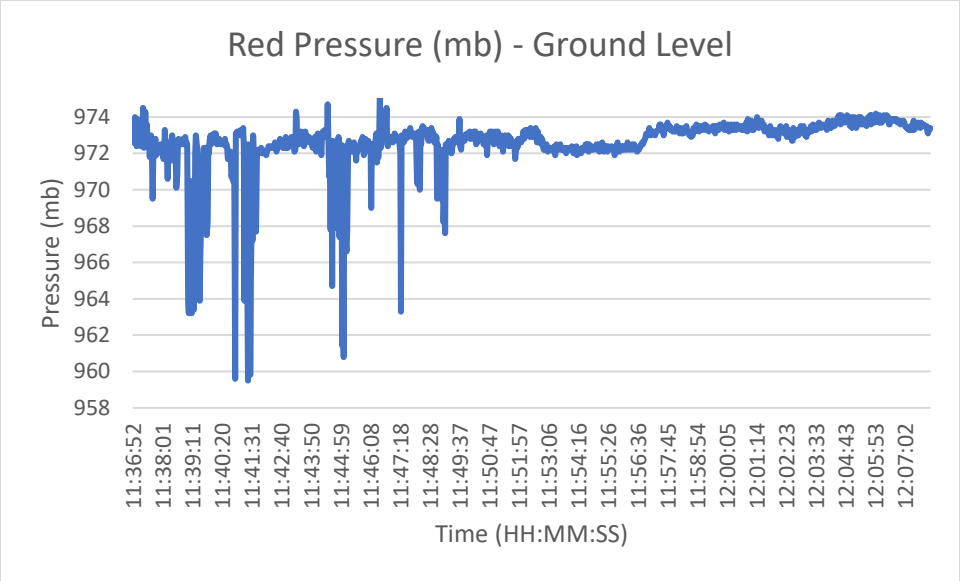


Figure 77: Full System Test #2 GPS Altitude

Like the first full system test, the GPS altitude seems to trend upwards as time goes on regardless of the actual altitude of the payload. The highest altitude recorded was at the end of the test when the payload was back at ground level, which makes no sense in actuality. The GPS sensor will be considered only for when the pressure-based altitude sensors are giving unreliable outputs.



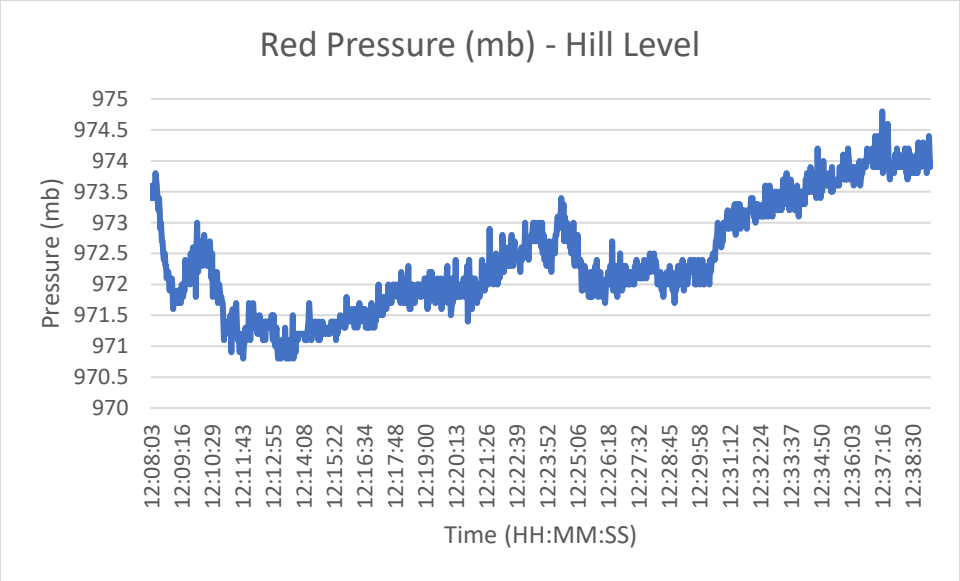


Figure 78: Full System Test #2 Red Pressure Readings

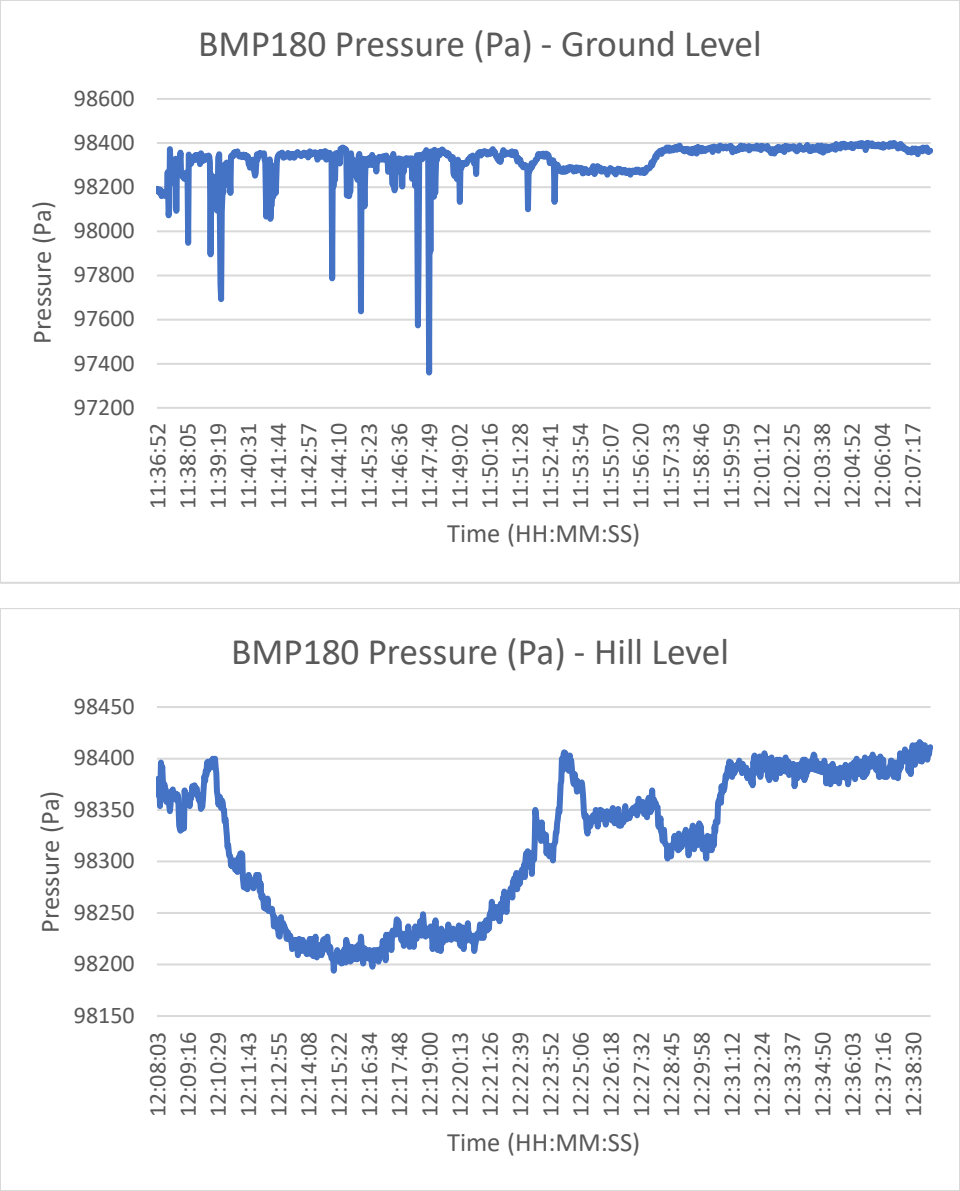


Figure 79: Full System Test #2 BMP180 Pressure Readings

4.2.3 Gas Sensor Readings

As with the first test, for the most part the gas sensors output their constant default value. However, the carbon dioxide sensor had a spike around when the car was entered, as well as the ozone sensor. The carbon dioxide sensor may have had its single spike due to being exposed to a teammate's breath in the enclosed space of the car, but the ozone sensor's multiple spikes after the car was entered are unexplainable. The ozone amount sensed seemed to gradually rise and fall rather than spike at a single datapoint, which makes it seem likely that it actually sensed ozone.

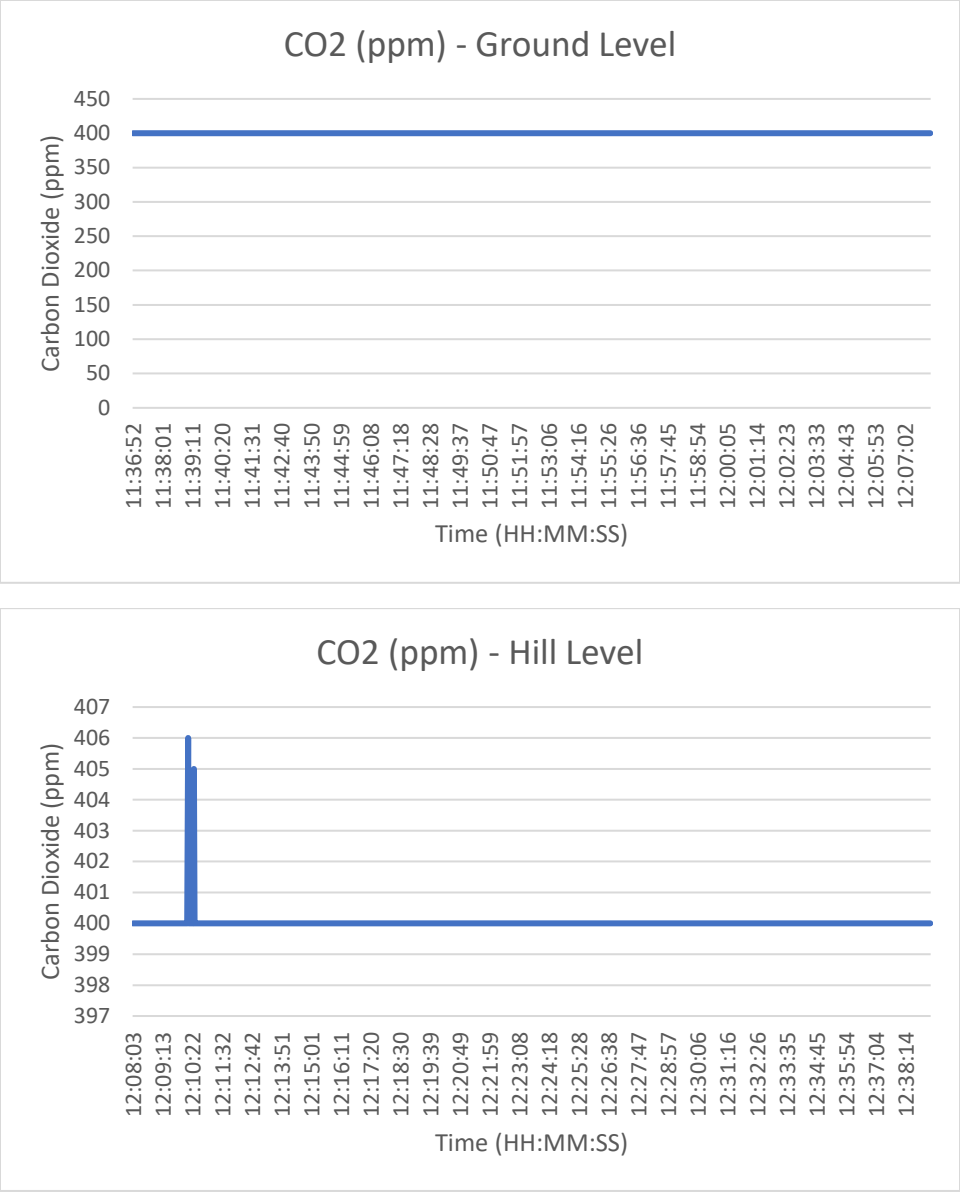
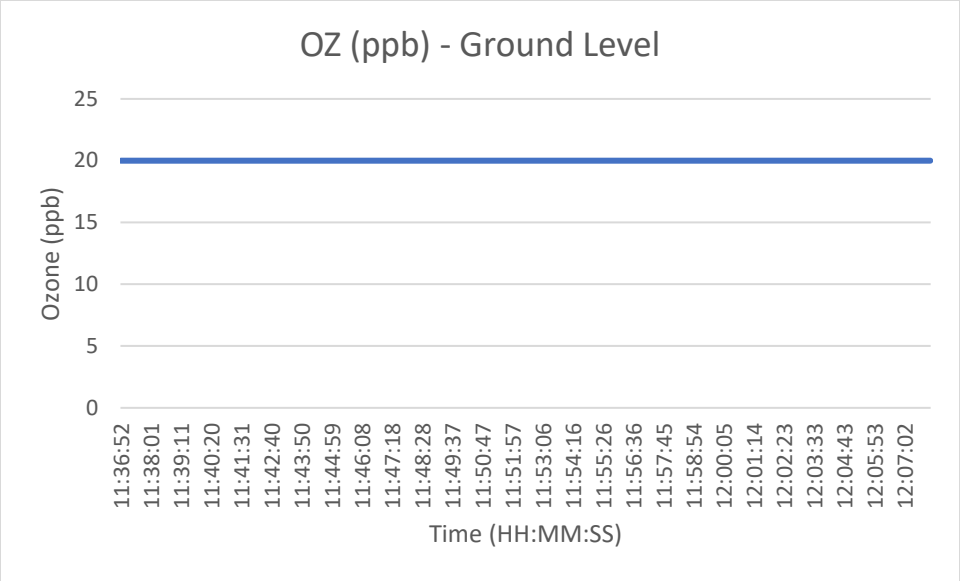


Figure 80: Full System Test #2 CO2 Readings



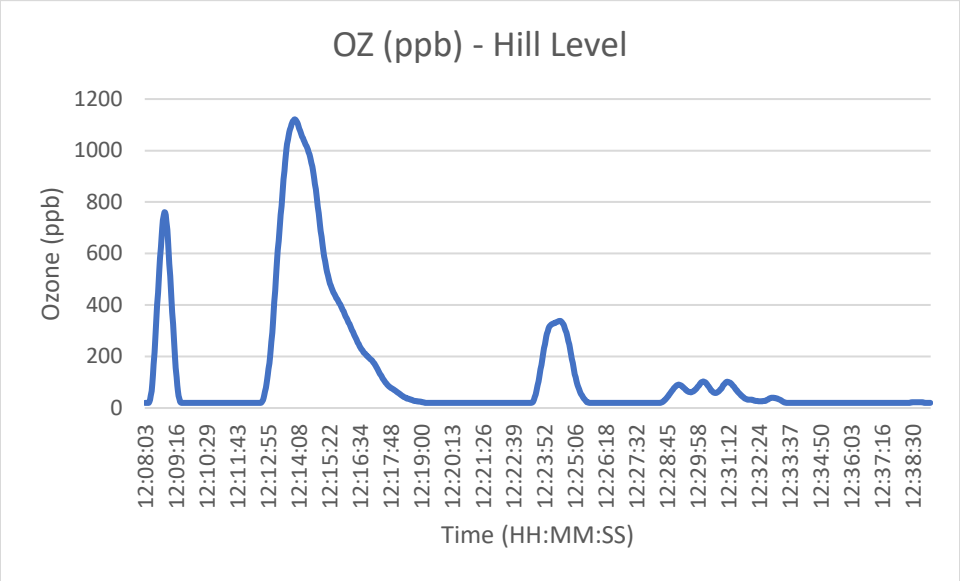


Figure 81: Full System Test #2 OZ Readings

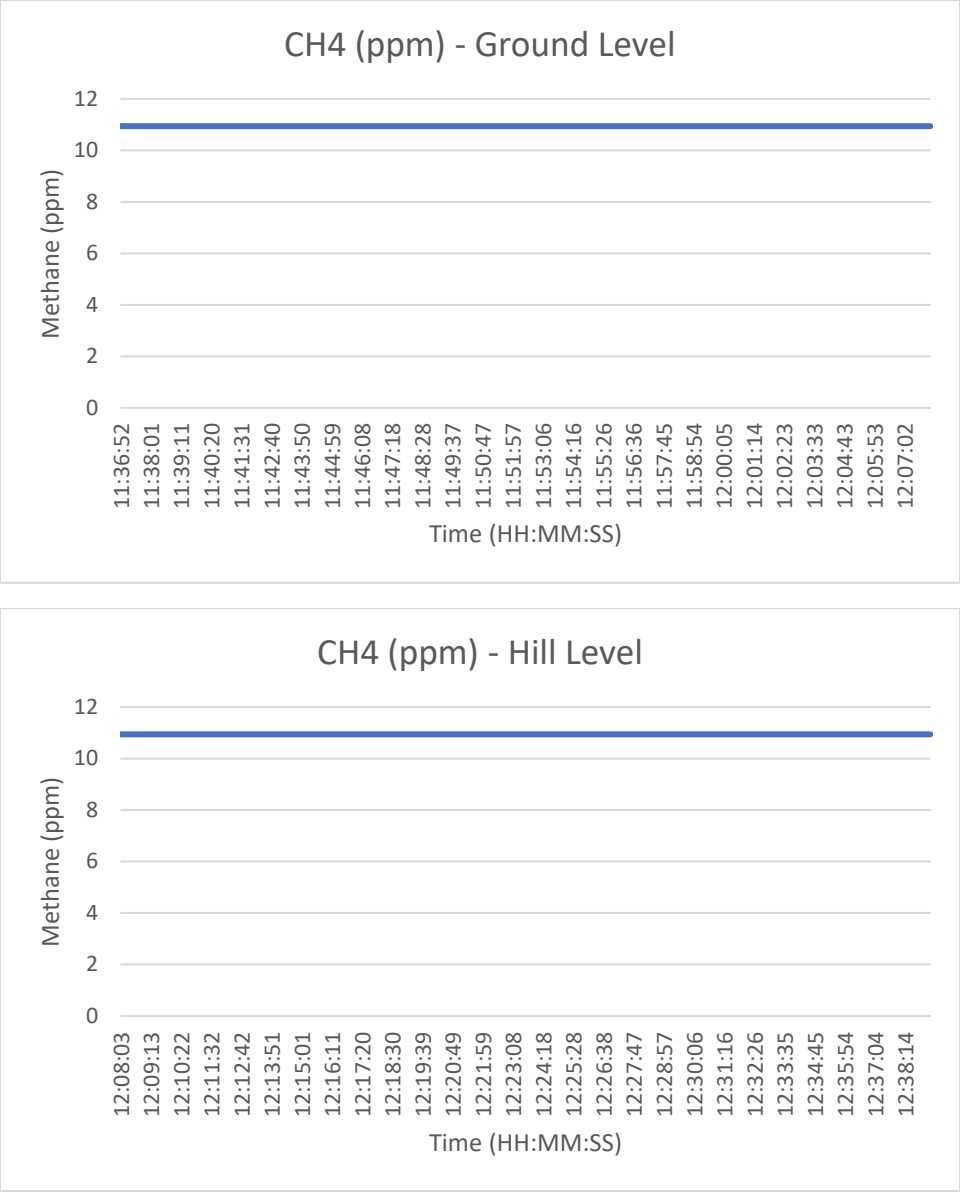


Figure 82: Full System Test #2 CH4 Readings

4.2.4 Miscellaneous Sensor Readings

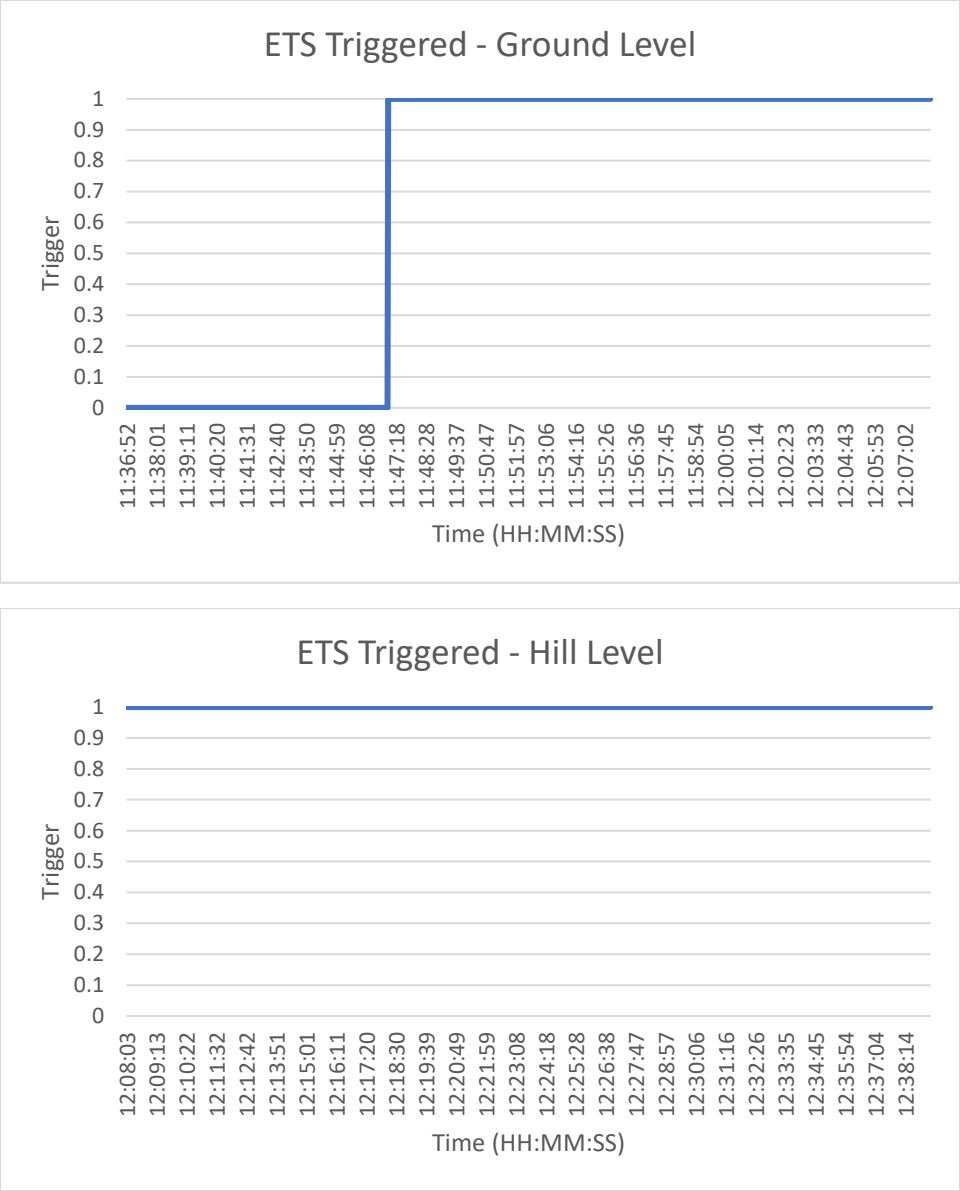


Figure 83: Full System Test #2 ETS Trigger Status Over Time

The ETS triggered early in the test, during the first of the laps around the base station.

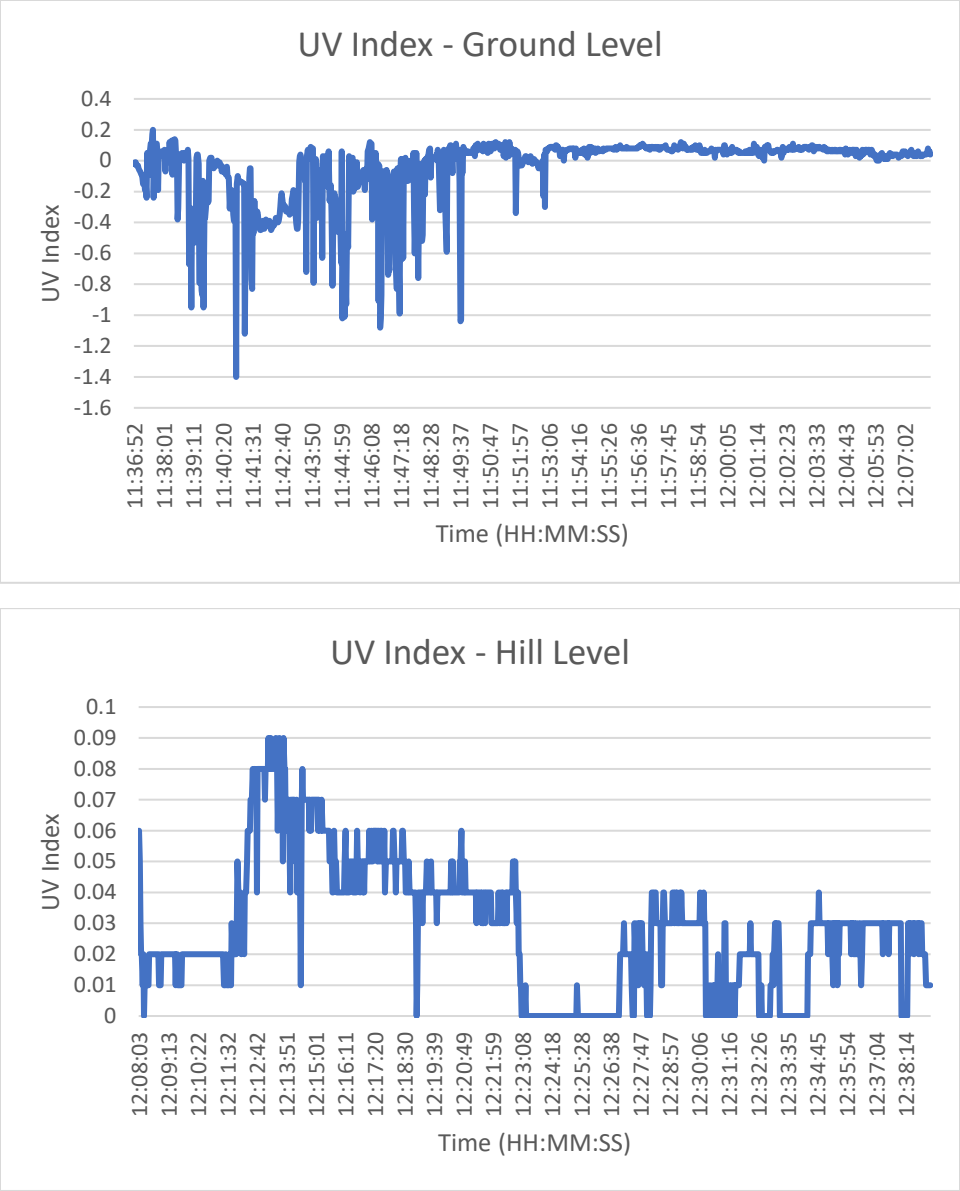


Figure 84: Full System Test #2 UV Index Readings

During the first half of the ground test, the UV sensor output negative values which was unexpected given that the UV scale is made up entirely of positive numbers. However, upon further testing and documentation reading, the VEML6075 sensor we used may output a negative value under low UV conditions due to the nature of the calculations made using the Arduino library. Values below 0 should be regarded as UV index 0, not as negative UV index values.

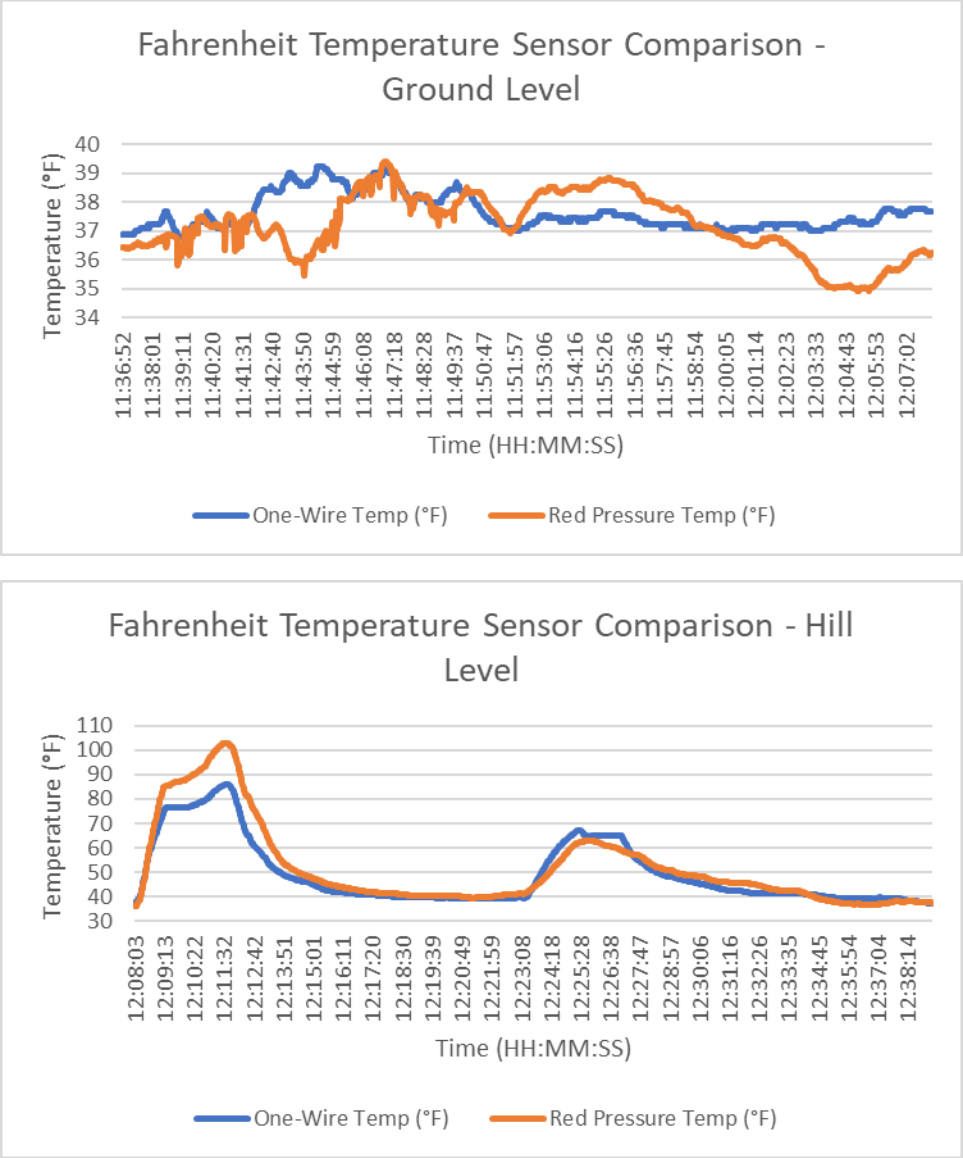


Figure 85: Full System Test #2 Fahrenheit Temperature Sensors Reading Comparison

The Fahrenheit temperature sensors functioned almost identically during this test save for some differences in their peak readings. Neither sensor tended to read consistently higher or lower than the other except for these peaks, which occurred in the second half of the test. However, since the one-wire temp proved to output a more accurate value during these times of difference, it still wins as the most reliable temperature sensor.

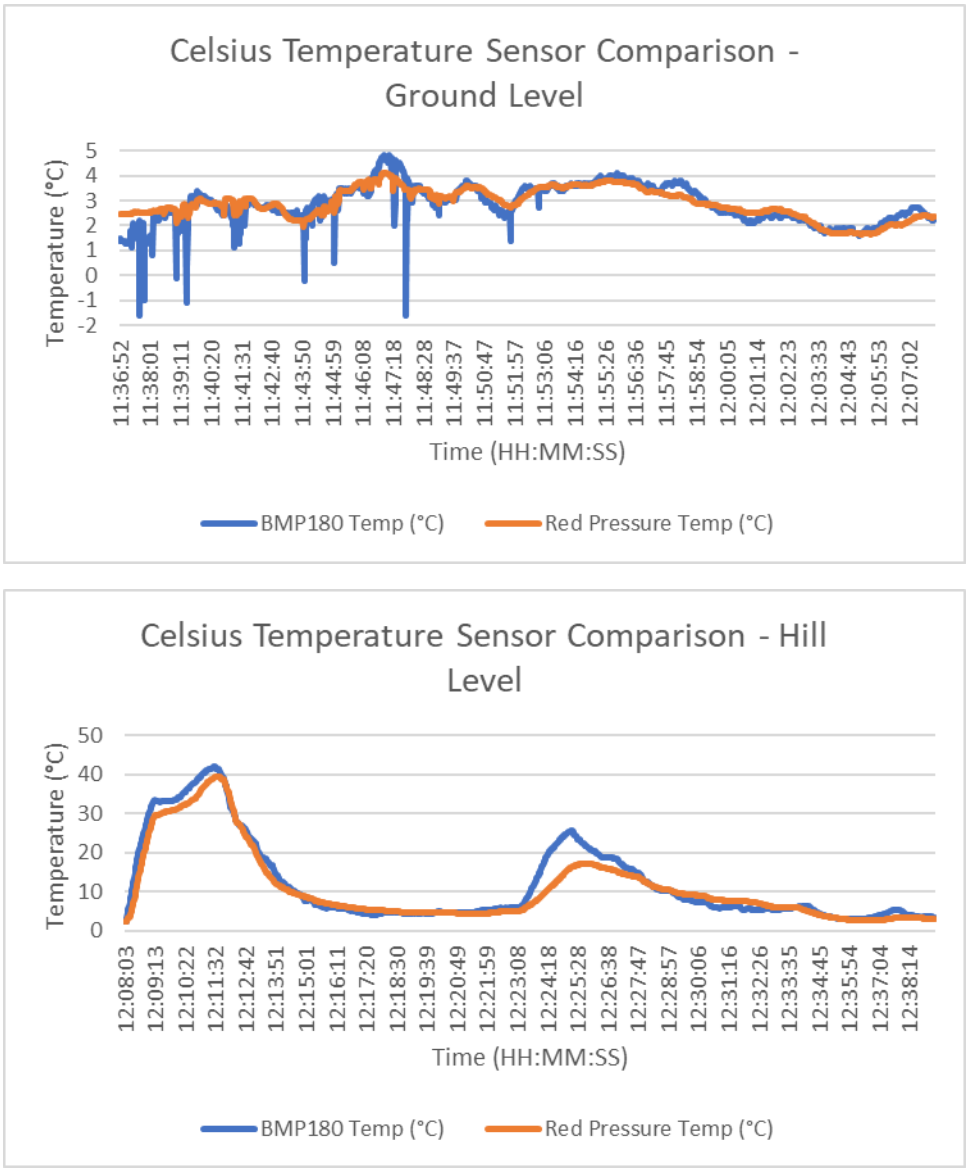


Figure 86: Full System Test #2 Celsius Temperature Sensors Reading Comparison

As with the Fahrenheit temperature sensors, the Celsius sensors had a few differences. The BMP180 sensor had some glitches in the beginning of the test, but later worked accurately, and the Red Pressure sensor readings ran lower during peaks. In this comparison the Red Pressure readings seem to have been more accurate.

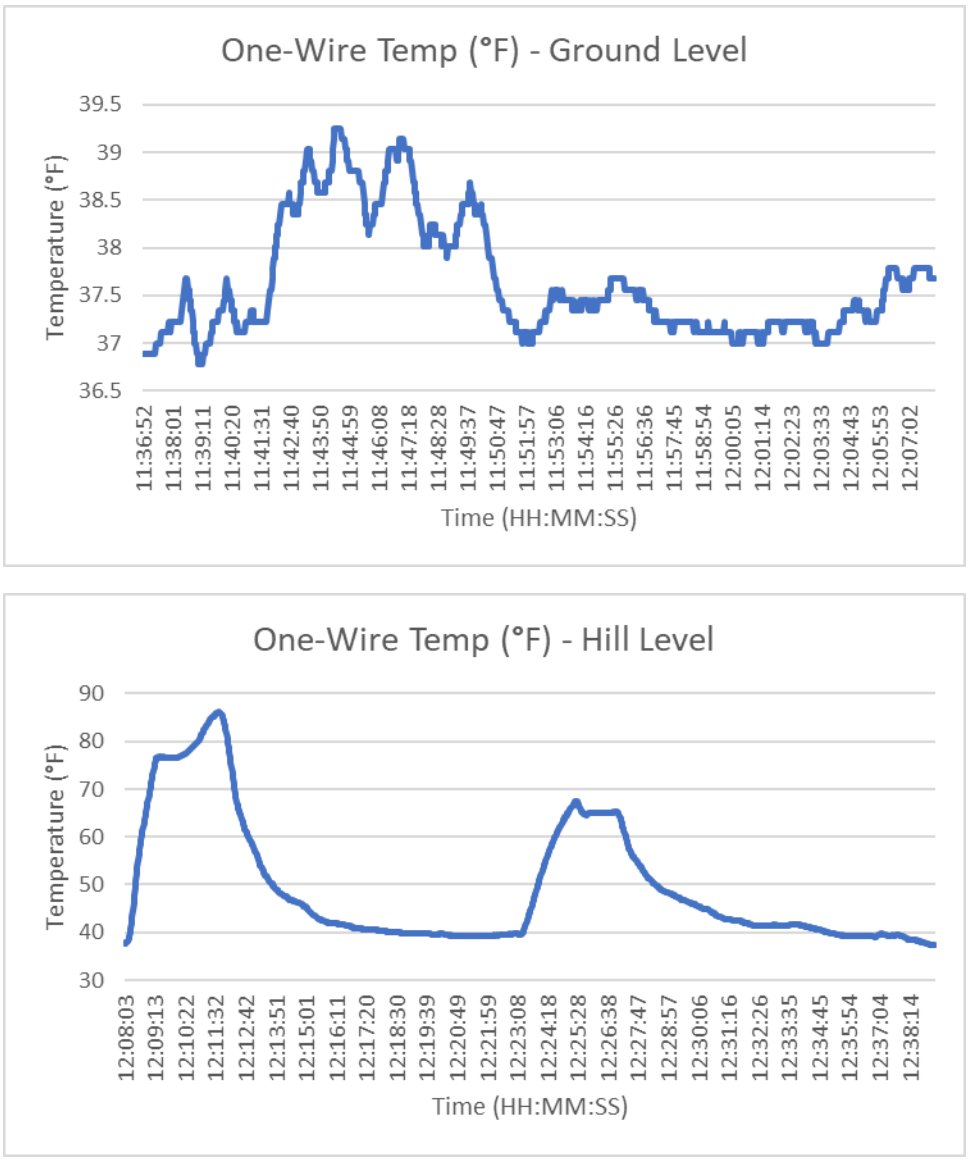


Figure 87: Full System Test #2 One-Wire Temperature Sensor Readings

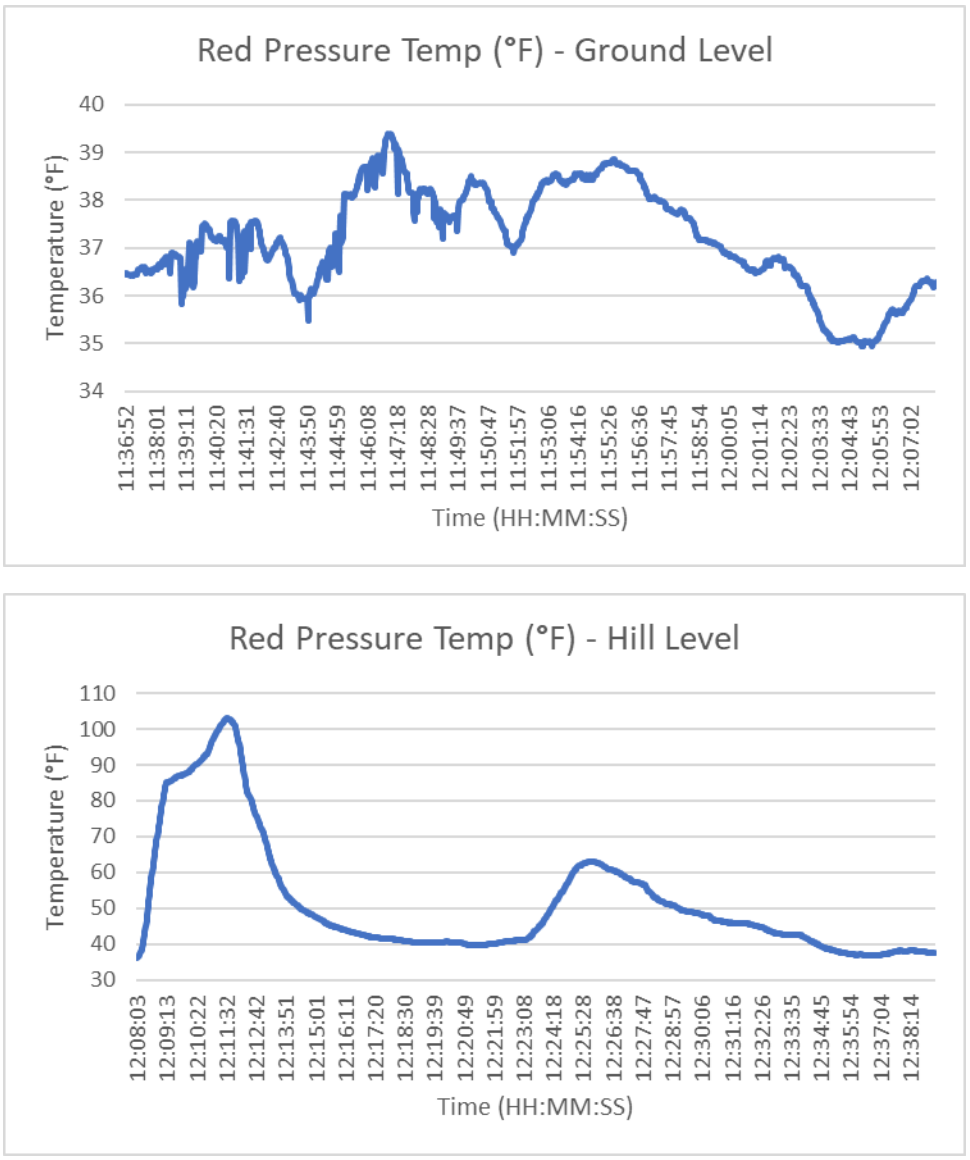


Figure 88: Full System Test #2 Red Pressure Temperature Readings in Fahrenheit

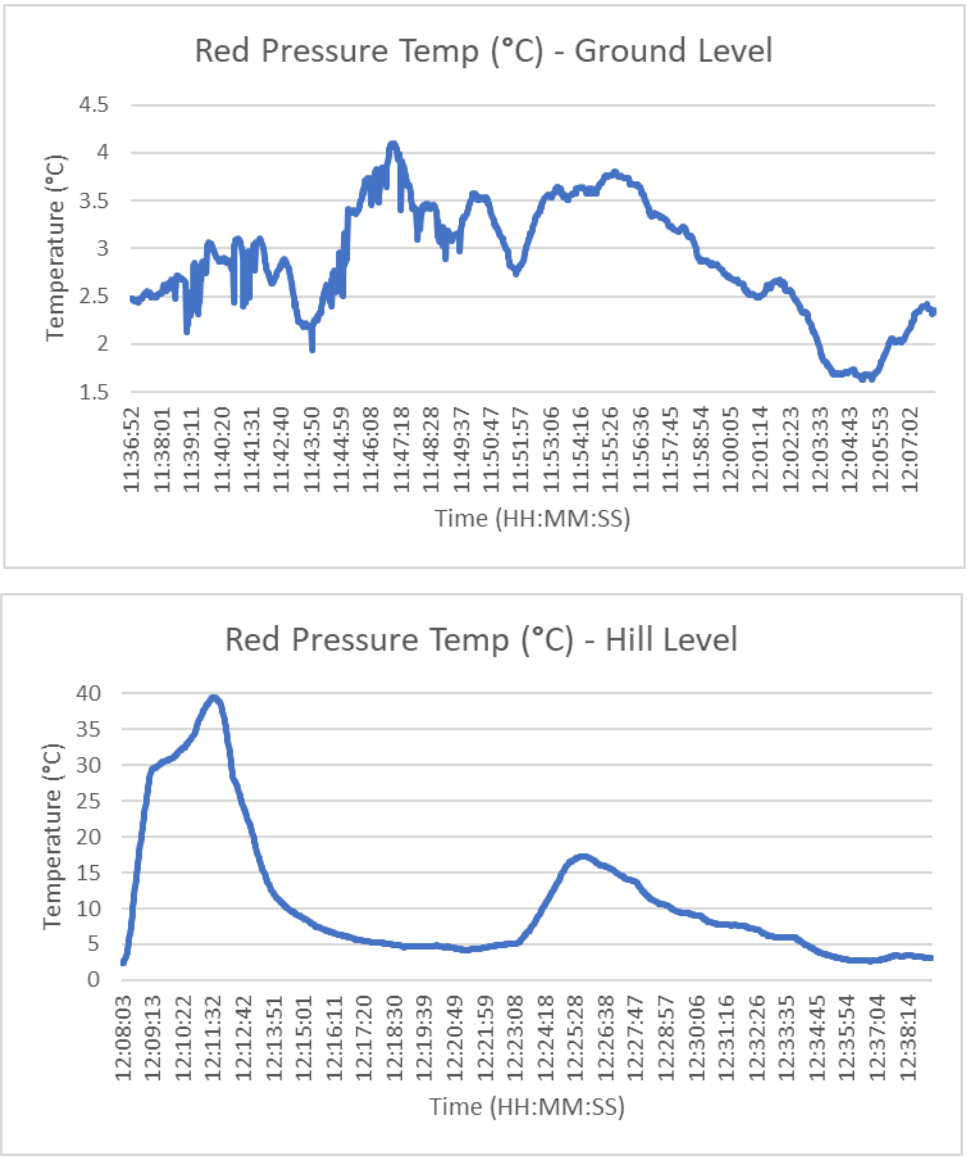


Figure 89: Full System Test #2 Red Pressure Temperature Readings in Celsius

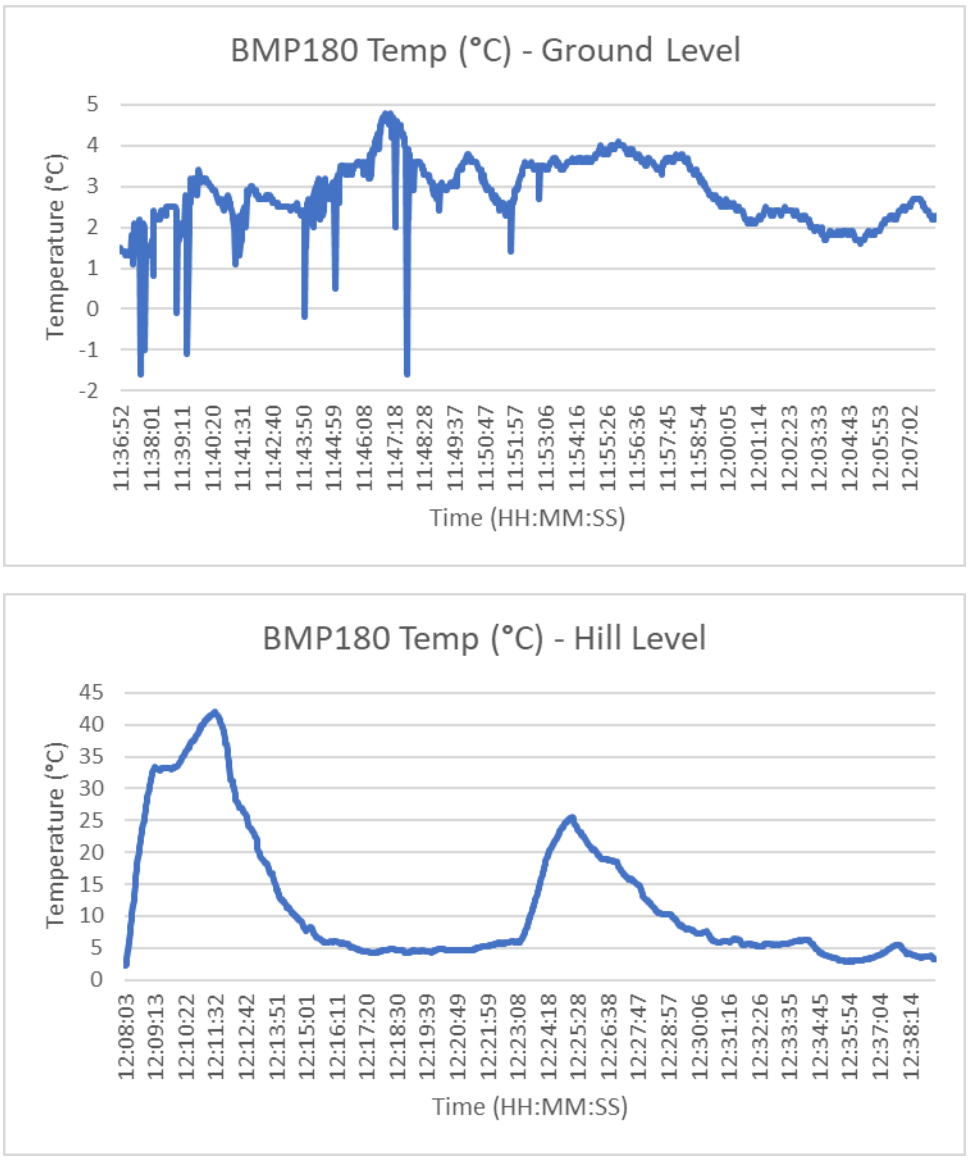


Figure 90: Full System Test #2 BMP180 Temperature Readings

4.2.5 Base Station Performance

For our base station system, we were able to collect sensor, GPS, and image data for most of our final test. There were instances during our final test where our base station program crashed due decoding errors when reading in the transmitted data from our radio. We suspect these errors were the result of packet fragmentation where our packets were being received with their identifying packet headers missing, causing our program to crash.

As for our antenna tracking system, our system performed well overall. However, there was a noticeable delay between the time it took our system to respond to the movement of our payload. We believe this was a result of the processing and storage of image data within our base station code in conjunction with the fact that one GPS coordinate packet would send after roughly four image data packets had been sent, causing another undesired delay. Additionally, due to the friction in the turning plate (lazy Susan) our antenna tracker stalled during some rotations, being unable to fully turn due to

the strong friction force. This resulted in our program making incorrect future turns as it was in a position different to what the Arduino expected. Overall, our system performed well.

5. Conclusion & Recommendations

In our project iteration, our team was able to successfully implement an early termination system in both hardware and software, enable image transmission, enable .csv file output, redesign the physical structure of our base station, and create a software system responsible for its control. Focusing on our achievements this term, we have compiled a list of recommendations for future iterations of the High-Altitude Weather Balloon Project that could apply in the coming years.

In our experience with using the base station hardware responsible for the antenna tracking system, we believe it would be immensely helpful to have a feedback system present in both the linear actuator and the stepper motor, whether it is via an encoder, limit switch or another form of control. This is because it would be useful for both parts and their mechanisms to know where they are to maximize the overall accuracy of the system. A time-based linear actuator and an origin direction-based stepper are always very uncertain at times to control. A PID (proportional–integral–derivative controller) system would be nice to see, or even in an interrupt service routine (ISR) when the computed angle gets out of range of the position and instructs the linear actuator to correct itself. Robotics Engineering students would be the best fit for this task. With this C/C++ code we would also look to switching the payload from a python-based system to a C/C++ system for better power efficiency due to python's high use of memory.

Further modifying the base station would be a terrific addition. Making it a whole table would be nice for the tests and debugging that is done mostly outdoors, on the floor. Adding some legs and ensuring that it is level would be a topic of interest of mechanical engineering or robotics students or an ECE major who is mechanically inclined.

Beyond the physical infrastructure of the project, we believe it would be beneficial for a team member to acquire an amateur (HAM) radio license as this would permit communication on frequency bands capable of providing greater coverage and a much more reliable communication link. Acquiring a amateur license would allow the team to purchase components to communicate on these licensed frequency bands. This would be a significant addition to the project beyond the antenna tracking that would help ensure a near full flights' worth of real time for data transmission including images.

Using better quality sensors will yield better data based on the choice of sensor. Ensure that a given sensor will function at altitude, where the temperature is very cold. In addition to getting better data, data processing would be a huge addition to this project. Either a Data Science or Computer Science student dedicated to processing the data achieved during flight to test flight to create nice plots, tables, etc. in a genuinely nice, comprehensible, readable, and neat style.

References

[1] A. Stewart, “The price of helium is up in the Air,” InDepth, 08-Oct-2022. [Online]. Available: https://gue.com/blog/the-price-of-helium/. [Accessed: 20-Mar-2023].

[2] EPA, “Overview of Greenhouse Gases,” EPA. [Online]. Available: https://www.epa.gov/ghgemissions/overview-greenhouse-gases. [Accessed: 08-Jan-2023].

[3] D. Carrington, “Carbon emissions from fossil fuels will hit record high in 2022,” The Guardian, 11-Nov-2022. [Online]. Available: https://www.theguardian.com/environment/2022/nov/10/carbon-emissions-from-fossil-fuels-will-hit-record-high-in-2022-climate-crisis. [Accessed: 08-Jan-2023].

[4] UN, “Ozone on track to heal completely in our lifetime, UN Environment Agency declares on World Day. | UN news,” United Nations, 16-Sep-2019. [Online]. Available: https://news.un.org/en/story/2019/09/1046452. [Accessed: 08-Jan-2023].

[5] A. Kobsa, “High Altitude Weather Balloon Launch III for Measuring Environmental Pollution,” Worcester Polytechnic Institute, 25-Mar-2022. [Online]. Available: https://digital.wpi.edu/concern/student_works/9593tz27b?locale=en. [Accessed: 08-Jan-2023].

[6] “Wind,” National Geographic Society. [Online]. Available: https://education.nationalgeographic.org/resource/wind. [Accessed: 22-Jan-2023].

[7] “NiCrTechTips,” Wiretron. [Online]. Available: https://wiretron.com/nichrome-resistance-informational-charts/. [Accessed: 01-sep-2022].

[8] Mouser. [Online]. Available: https://www.mouser.com/ProductDetail/Toshiba/TK040N65ZS1F?qs=w%2Fv1CP2dgqoOzYU83sO0WA%3D%3D. [Accessed: 01-sep-2022].

[9] Mouser. [Online]. Available: https://www.mouser.com/ProductDetail/Littelfuse/0215002.MXP?qs=TSScP84zSEreW3iQYIRNnQ%3D%3D. [Accessed: 12-sep-2022].

[10] “MOSFET Gate Resistor,” build-electronic-circuits. [Online]. Available: https://www.build-electronic-circuits.com/mosfet-gate-resistor/. [Accessed: 12-sep-2022].

[11] “A robust version of the JPEG image format, for transmission over an unreliable medium. [Online] Available: https://github.com/fsphil/ssdv [Accessed: 10-Oct-2022].

[12] “AccelStepper library for Arduino,” Accelstepper: Accelstepper Library for Arduino. [Online]. https://www.airspayce.com/mikem/arduino/AccelStepper/ [Accessed: 21-Mar-2023].

[13] F. A. Team, “How do you control a linear actuator with an Arduino?,” *Firgelli Automations*, [Online]. https://www.firgelliauto.com/blogs/tutorials/how-do-you-control-a-linear-actuator-with-an-arduino. [Accessed: 21-Mar-2023].

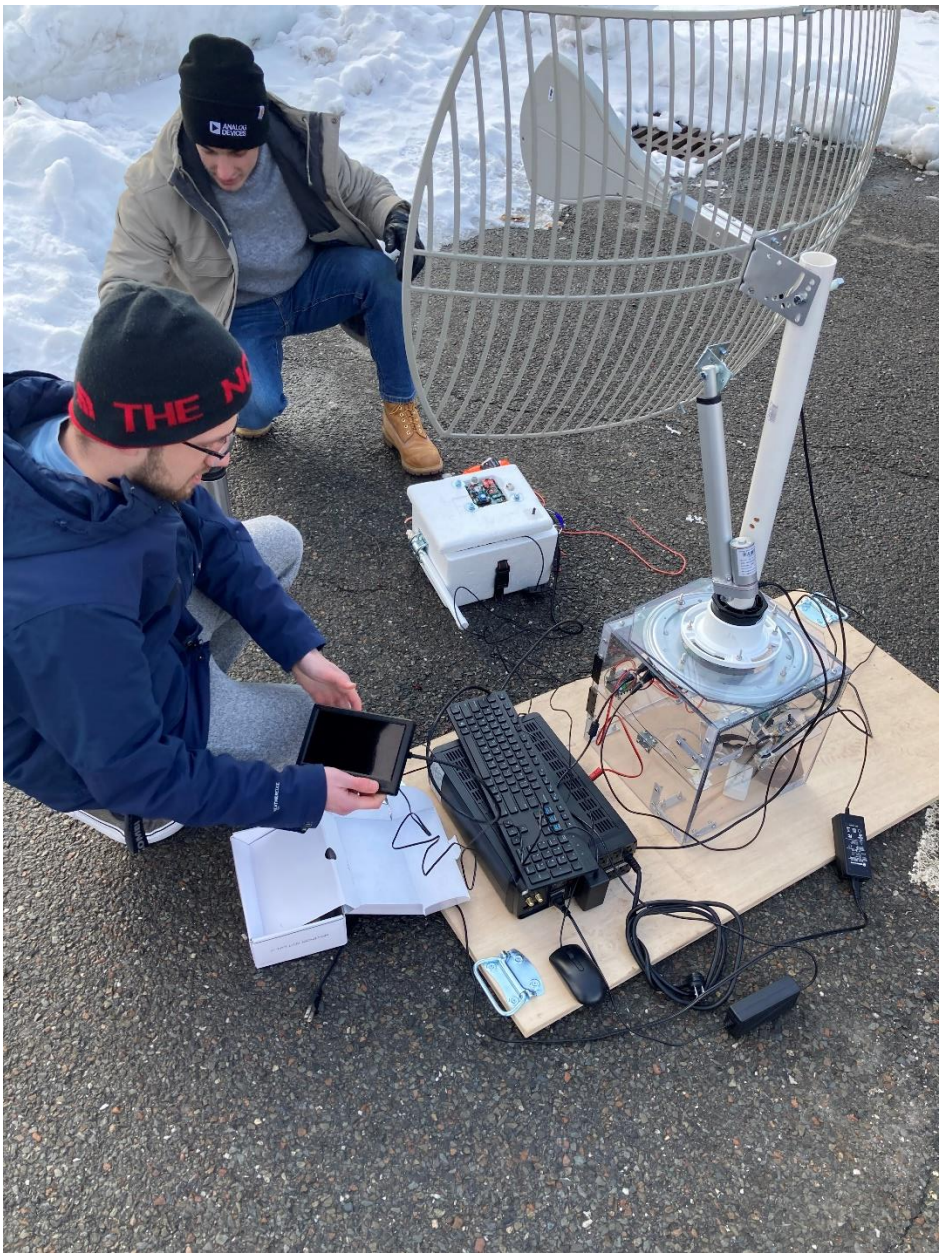
Appendices

Appendix A: Git Repository Link

https://github.com/HAB-MQP-WPI/HAB-IV-Code

Appendix B: Full System Test #1 Picture

Appendix C: Full System Test #2 Pictures


THE

Appendix D: Full System Test #2 Video Link

Test Vid (Working).mov

Appendix E: MOSFET Id vs. Vds MATLAB Code

```
close all;
clc
clear

syms id(vds)
K=20;
vth=4;
vgs=5;
vdsTest=linspace(0,20,1000);

id1=K*2*(vgs-vth-(vds/2))*vds;
id2=K*(vgs-vth)^2;

id(vds) = piecewise(vds<(vgs-vth),id1,vds>=(vgs-vth),id2);

id5 = id(vdsTest);%when vgs=5v

hold on;
plot(vdsTest,id5);
line([0 12],[1.58 0],'Color','green');%load line
line([1 1],[0 30],'Color','red','LineStyle','--')
xlim([0 15]);
ylim([0 30]);
legend('vgs=5v','load line','vod')
xlabel("vds(v)")
ylabel("id(A)")
title("id vs. vds for TK040N65Z,S1F")
hold off
```