High Altitude Weather Balloon Research

ASR C Block
November 2014

The A-Team
Software Engineer - Megs Malpani
Design Engineer - Brittney Kidwell
Quality Control - Elaine Wong
Mechanical Engineer - Leo Jaimez
Radio Engineer - Nick Seidl
Experiment Designer - Sam Rubin
Project Leader - Jeff Barratt

I. Abstract

In this research project, a tethered and free launch of a balloon with a data-recording payload were performed. The tethered launch consisted of a 1000-foot string attached to the bottom of the payload and to a ground-based reel-in system, with a goal of testing all of the systems for the free launch. The free launch started at Timm Ranch in Vacaville, travelled into the lower stratosphere, and was recovered about 2 hours later north of Sacramento. The main goal of this project was to record data about our lower atmosphere as well as practice optimizing the balance between weight and data-taking devices.

II. Table of Contents

III. Introduction

§III.1 – Motivation

The main goals of this project were to record data about the lower atmosphere as well as practice optimizing the weight and data balance in the payload we designed. The motivation for these goals originate from the challenge given by Dr. Dann to build a successful payload that would accurately take data on the change of temperature, pressure, and humidity relative to altitude in the earth's atmosphere. We carried out these experiments with a team of seven people, each person with different sets of responsibilities.

§III.2 – Atmospheric Composition and Theory

The atmosphere is composed of nitrogen, oxygen, and a small amount of other gases. An atmosphere surrounds a large mass, like Earth, because of gravity. The atmosphere becomes thinner and thinner the higher up you go. It is composed of five layers the troposphere, stratosphere, mesosphere, thermosphere, and exosphere. The shallowest layer of the atmosphere is the troposphere, which extends from about 4 to 12 miles above sea level. This layer is where all the weather occurs, and is typically where weather is at its warmest. This is because the Earth, which is heated by the Sun's energy, heats the air. The troposphere also contains about 80% of the atmospheres mass. Above the troposphere lies the stratosphere. The stratosphere ranges from 12-31 miles above the Earth's surface. Here, temperatures increase due to the energy absorption of the ozone. The stratosphere contains a large amount of ozone, which absorbs large quantities of Ultraviolet light given off by the sun, and prevents some from reaching the Earth's surface. Above the stratosphere is the mesosphere. This layer extends from 31 miles to 53 miles above the Earth's surface. It, along with the stratosphere, is considered the middle part of the atmosphere. In the mesosphere, the gases in the atmosphere are just dense enough to be able to stop meteors from hurtling into the atmosphere. Next comes the thermosphere, 53-375 miles from the earth's surface. This is known as the upper atmosphere and is by far larger than all previously mentioned layers. Gases here are extremely dense and temperature ranges from -184 degrees Fahrenheit at the bottom to 3,600 degrees Fahrenheit at the top. However, because of the miniscule amount of molecules left in the atmosphere it feels cold. The last layer is called the Exosphere, and is where atoms and molecules escape into space. It is located at approximately 375 miles away from Earth's surface and above.

Atmospheric pressure is the force exerted on a surface by the air above it as gravity pulls it to Earth. The atmosphere is pressed against the earth by atoms further up. Atmospheric pressure is measured using a barometer. Atmospheric pressure drops exponentially as altitude increases. Atmospheric Temperatures, for the entirety of the troposphere, lower approximately linearly at a rate of 2°C per 1000 feet of altitude.

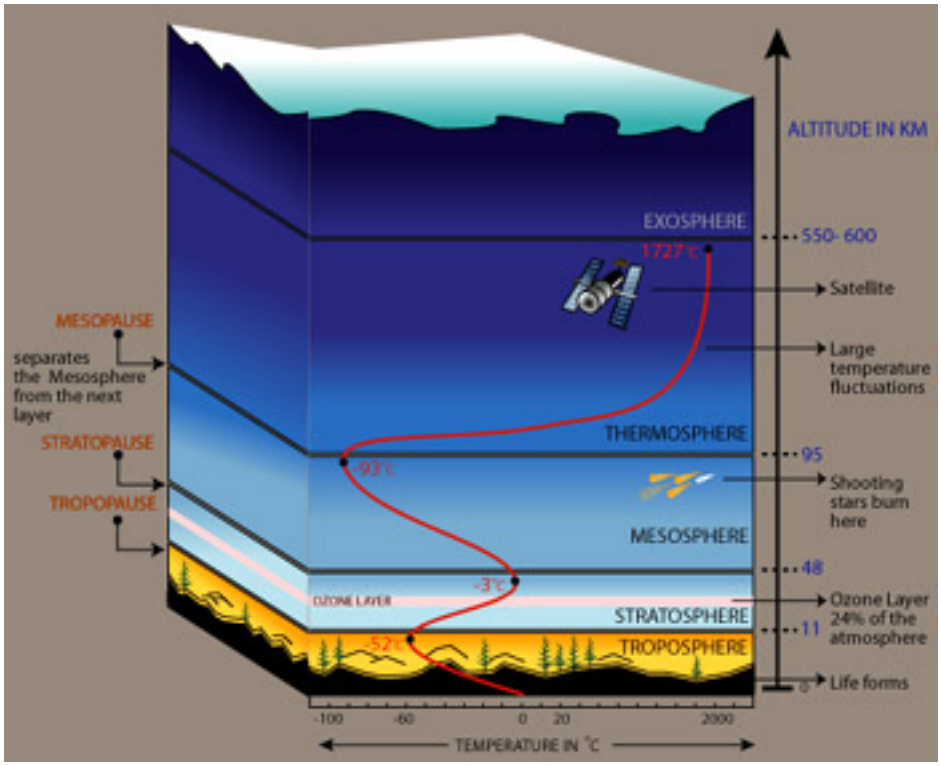


Figure 1: Depicted are the different levels of the atmosphere as well as the relationship between temperature and altitude. The red curved line indicates the change in temperature.

§III.3 – Greenhouse Effect

The greenhouse effect, at its simplest form, is a process in which thermal radiation from a planet's surface is absorbed by greenhouse gases in the air. Greenhouse gases are molecules such as water vapor, methane, carbon dioxide, and ozone. These gases absorb the earth's thermal radiation and cause our planet's atmospheric temperature to increase. Greenhouse gases are a vital part of life, and without them, we would not be able to maintain the warmth necessary for the survival of most species on the planet. However, with the increased usage of fossil fuels, the emission of carbon dioxide into the atmosphere is leading to higher than average amount of greenhouse gases, and consequently is leading to warmer-than-average temperatures. If the temperature continues to rise, it will lead to a variety of environmental issues such as the melting

of polar caps and the warming of oceans, both of which are extremely bad for the earth's environmental balance. This makes atmospheric science a very important matter to study.

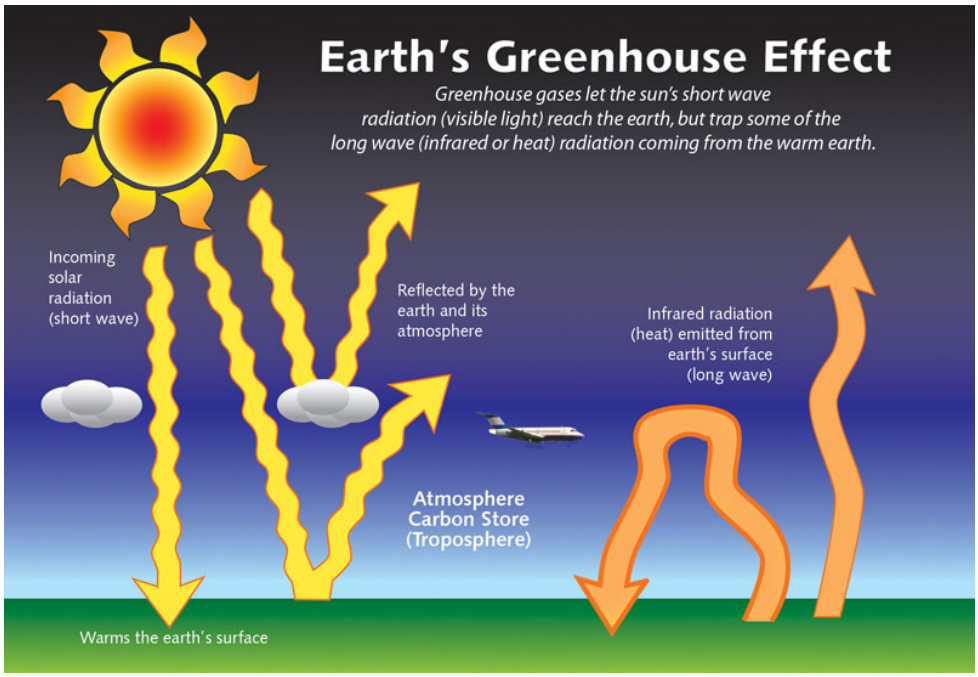


Figure 2: Graphic depiction of the greenhouse effect.

§III.4 – Aurora Borealis

Figure 3: Image of the Aurora Borealis

The Aurora Borealis, or the Northern lights, and the Aurora Australis, the Southern lights, are phenomena that occur when supercharged atoms and molecules in the ionosphere

interact with elements in our atmosphere. These lights are only seen above the northern and southern hemispheres. The variation in colors are a result of the variation in gases in which molecules are colliding with, and the height at which it collides with the molecules. In the 1950s, scientists were able to conclude that the reason for the electron and proton flow to the Earth's atmosphere is because of solar wind. Solar wind is composed of charged particles such as electrons and protons that flow outward from the sun and through the solar system. These charged particles then reach Earth and collide with atoms like water, oxygen, nitrogen, and create the beautiful colors seen.

List of colors seen:

1. Green – collision with oxygen, from 150 miles and down
2. Red – collision with oxygen, from 150 miles and up
3. Blue – collision with nitrogen, from 60 miles and down
4. Purple/violet – collision with nitrogen, from 60 miles and up

§III.5 – Experiments Conducted

Five experiments were chosen to be conducted for the free launch: Temperature, Pressure, Humidity, UV light, and ambient light. See Appendix E for more information on each of the sensors and how they work.

IV. Physical Design

§IV.1 – Payload

The payload was designed for strength and durability. There were three main components of the payload: The Styrofoam box, the Styrofoam lid, and the outside wooden box. The Styrofoam box is 11” wide, 9” deep, and 7 ¼” tall. The Styrofoam walls are 1 ½” thick, for maximum insulation. Inside, the box is 8” wide, 6” deep, and 4 ¼” tall. The payload weighed 4.45 lbs., which was 51% of the buoyant force.

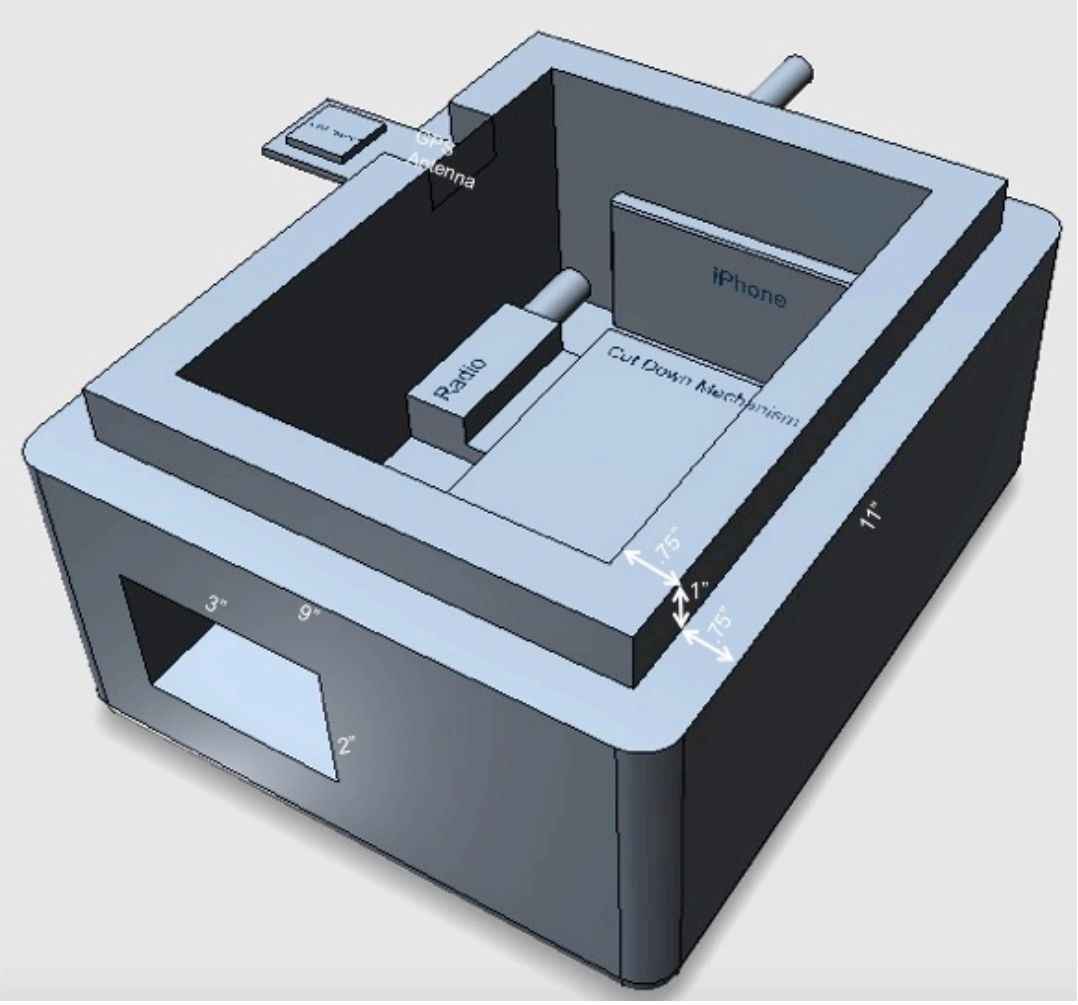


Figure 4: Styrofoam bottom of payload. CAD drawing done on 123D Design

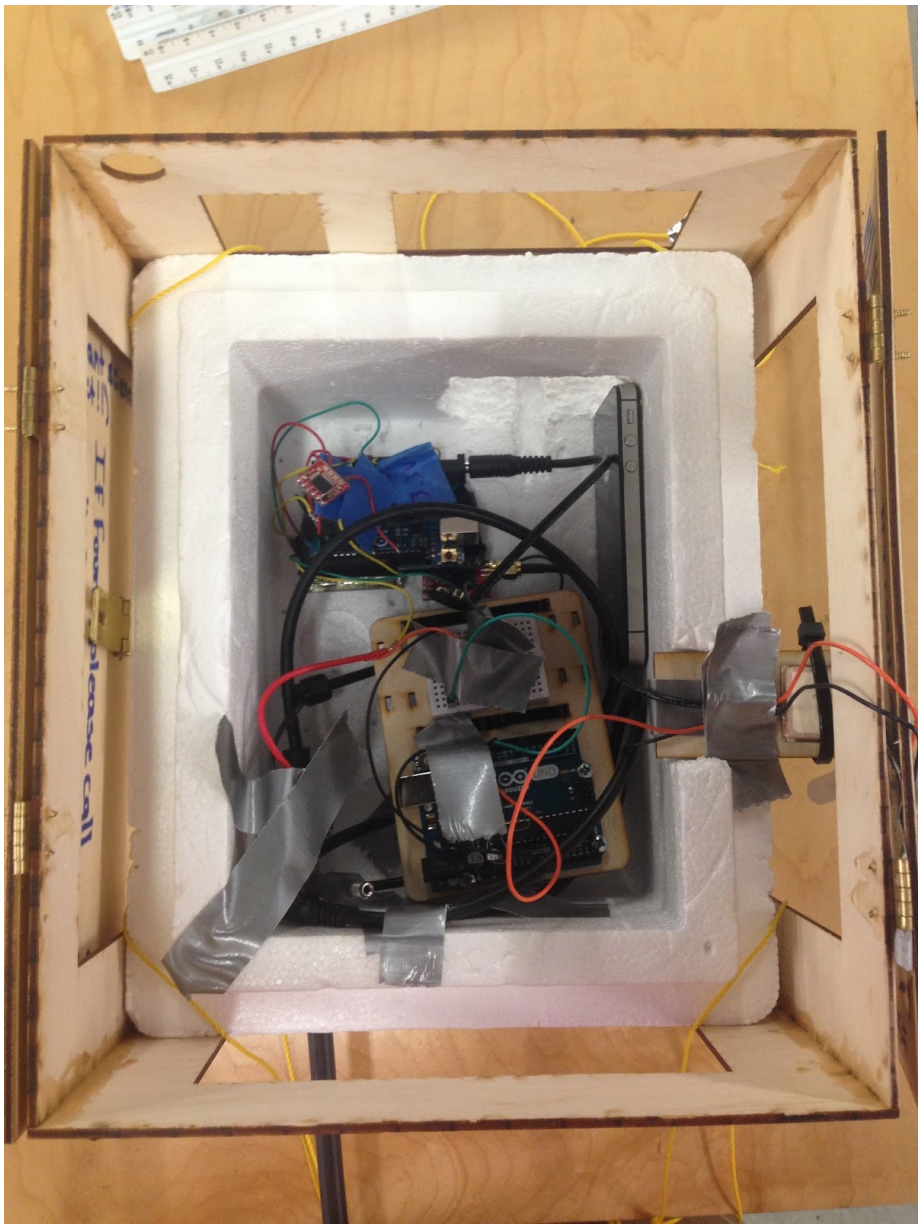


Figure 5: Top view of payload.

The Styrofoam bottom of the payload held the arduino, module, and antennae for the radio system, the arduino for the cutdown mechanism and the cutdown mechanism itself, the iPhone, and the GoPro. The radio system arduino and module were taped inside the box. There were two holes in the payload: one for for the radio antenna and one for the GPS antenna. The radio antenna stuck out of the payload, and the GPS antenna was taped to a small wooden laser-cut platform that also stuck outside of the payload for maximum satellite visibility. The cutdown mechanism was also taped inside of the box, with the wire used to actually cut down the parachute and payload strung through the hole that the GPS antenna was in. An iPhone was

taped in the box, to ease the process of finding the payload by using Find My iPhone, should we need it. A hole the size of the GoPro was cut in the side of the Styrofoam box to house the GoPro, which ultimately did not end up in the final launch because it had no charge when the free launch was about to start.

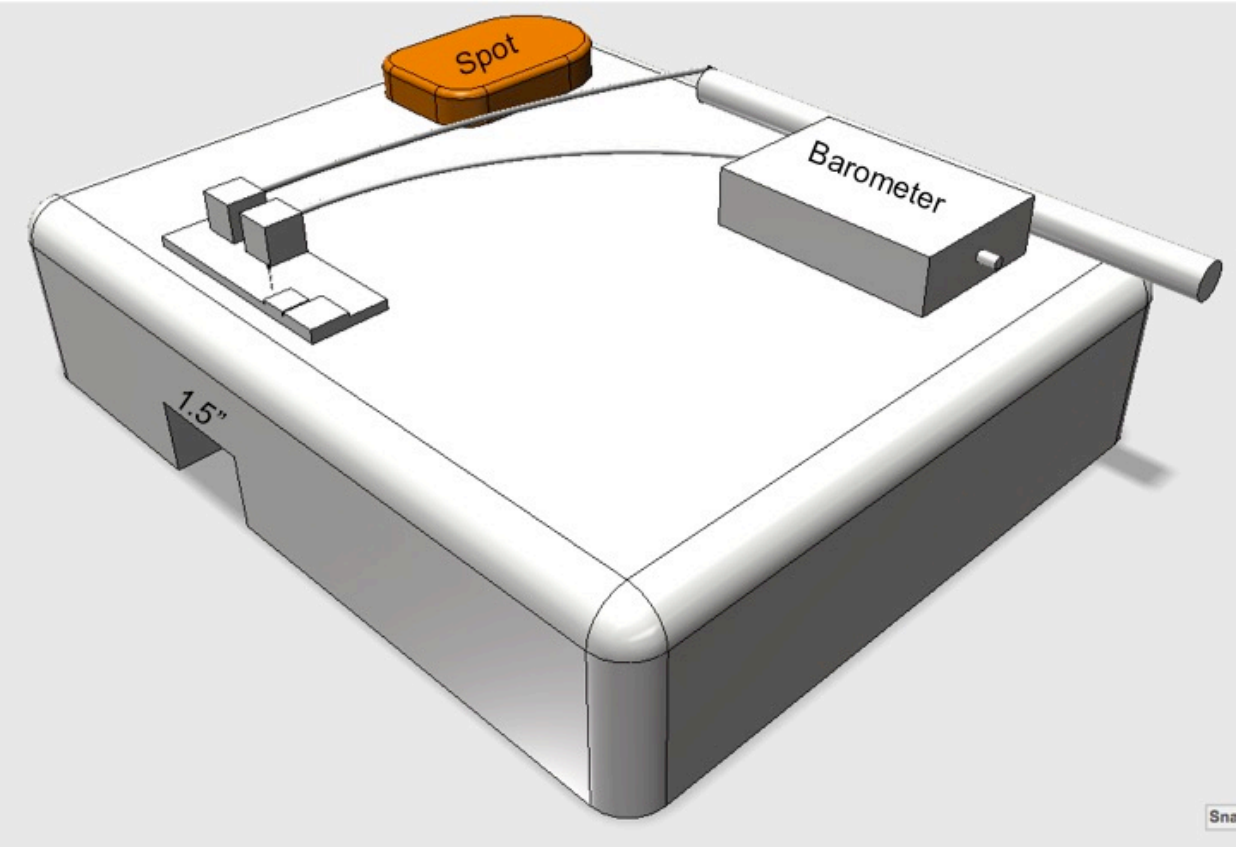


Figure 6: Styrofoam lid of payload, top view. CAD drawing done on 123D Design.

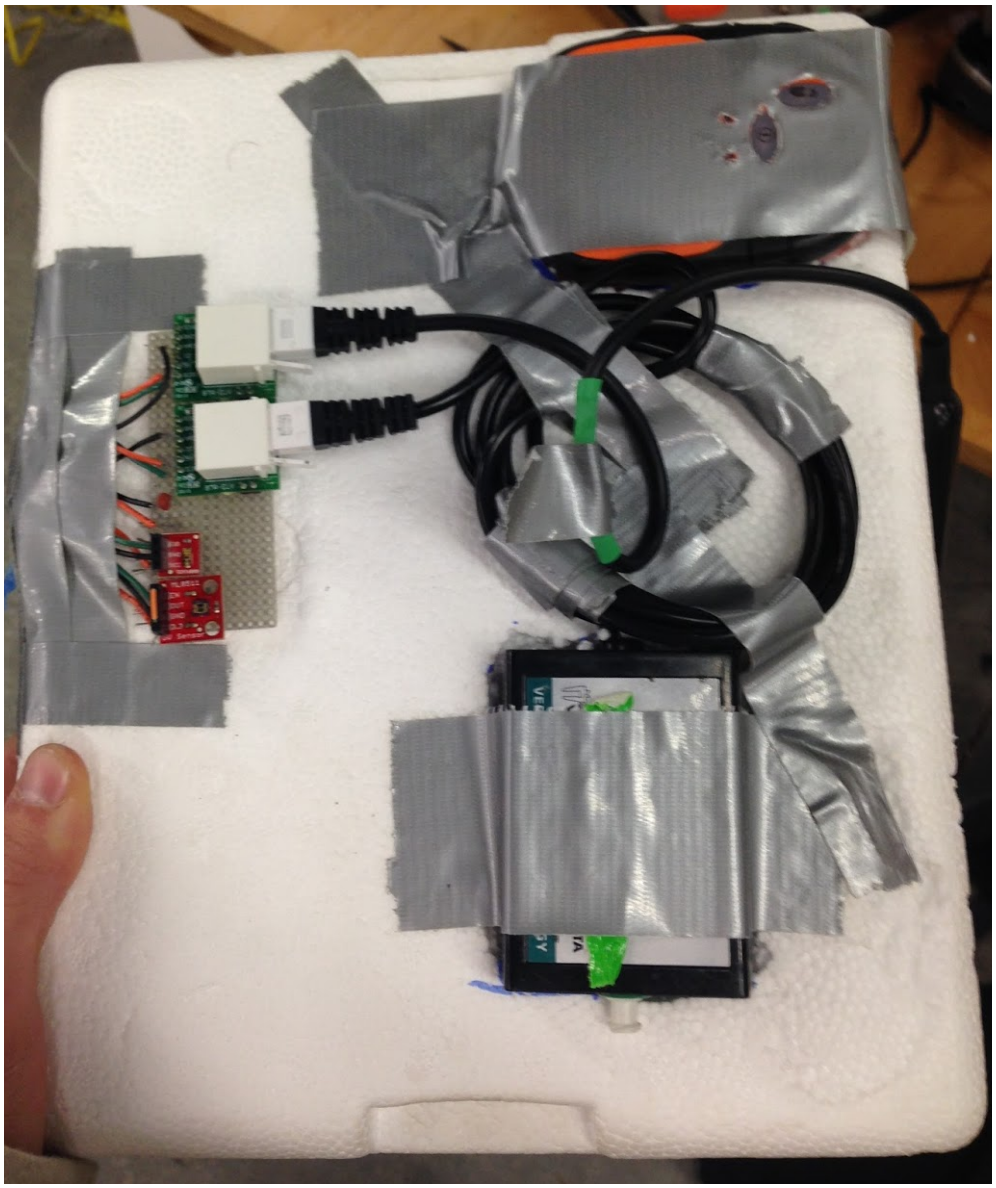


Figure 7: Top view of lid of payload.

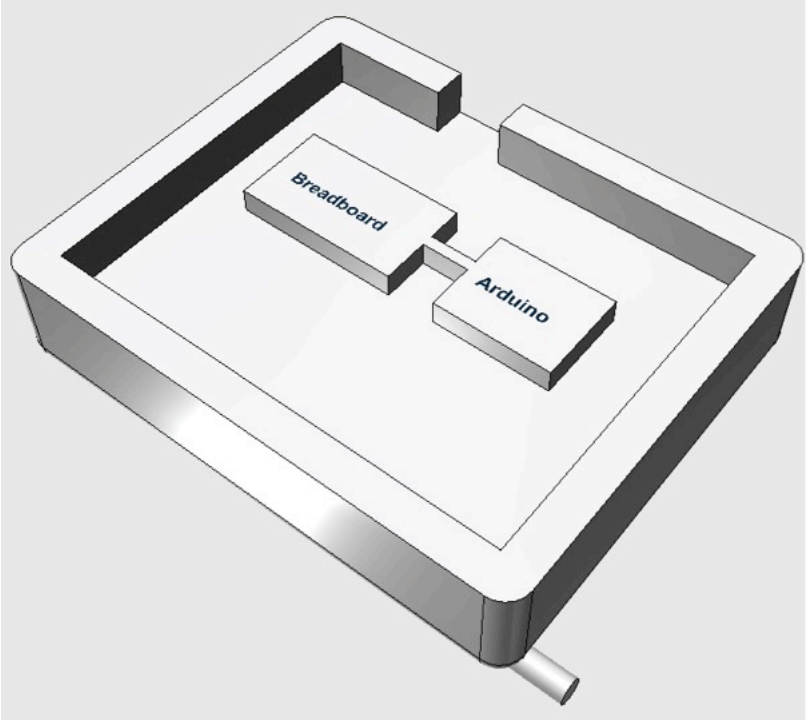


Figure 8: Styrofoam lid of payload, bottom view. CAD drawing done on 123D Design.

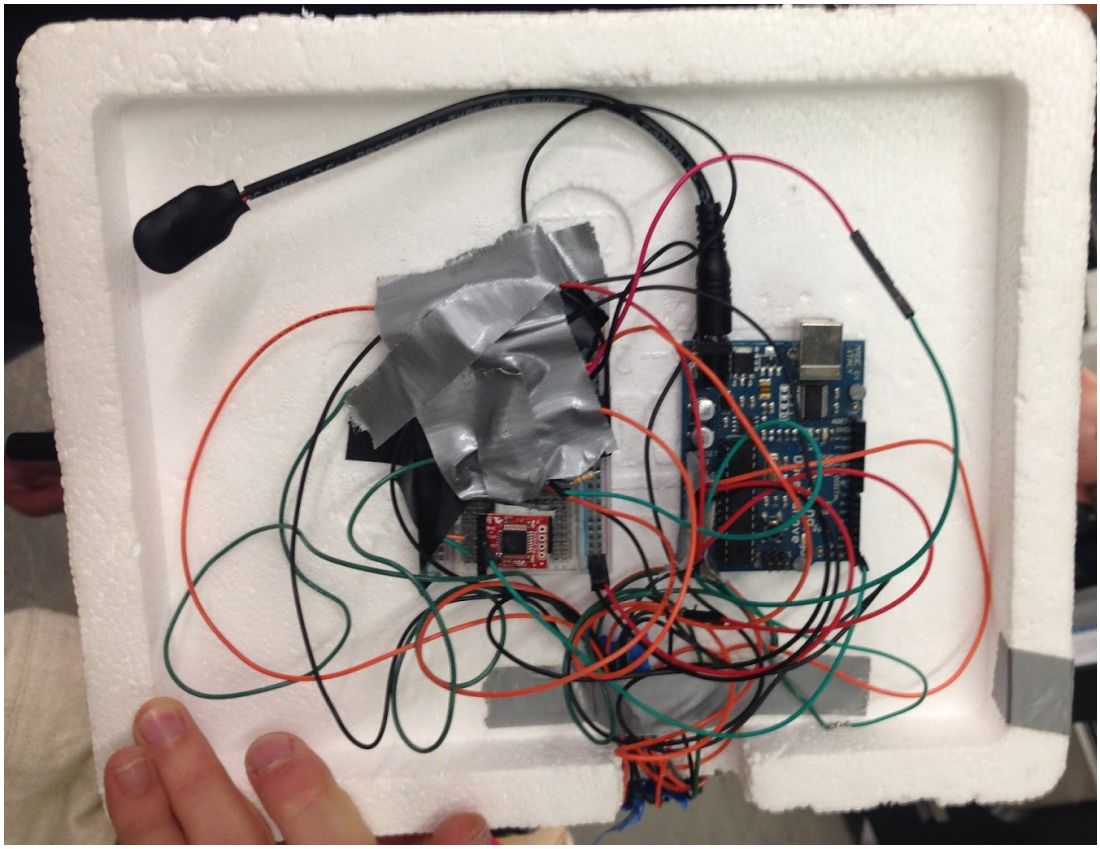


Figure 9: Styrofoam lid of payload, bottom view.

The Styrofoam lid of the payload held the experiments and the Spot tracker. On the inside of box top was the arduino and breadboard for the experiments. Along with being taped in securely, the arduino was secured with nails stuck through holes in the arduino into the Styrofoam. All the above items placed inside the Styrofoam box were done so because they were the most sensitive to temperature and the styrofoam offered insulation. The outside of the box top held the temperature, ambient light, UV, gas pressure, and humidity sensors, all securely taped down. The gas pressure sensor was placed in an indent on the box to further secure it. The Spot tracker was also taped on the Styrofoam top in an indent in the Styrofoam. For the tethered launch, the Spot tracker was placed inside the box, but no signal was obtained because the GPS satellite reception wasn't able to get through the Styrofoam box, so, after rigorous testing, for the space launch, it was placed outside the box.

Figure 10: The wooden box. CAD drawing done on 123D Design.

In addition, a wooden box was laser cut to house the Styrofoam box. The wooden box was 11 ¼ wide, 9 ½” deep, and 8 ½” high, with walls ⅛” thick. To reduce the weight of the box, each side had a large square hole, which removed all wood up to 1 ½” from the edge of each side. Both the bottom and one side had an extra cross of wood, making eight square holes instead of two. The bottom had this feature for extra support, and the side had it so that a acrylic panel could be attached so the GoPro would stay in the box but also be able to film the outside. The same side also had a small circular hole cut out so that the humidity sensor could stick out of the box. The top of the box had a 1 ½”x3 ½” rectangular hole so that the temperature, light, and UV sensors would have access to the outside and have more accurate data. The top of the box also had a smaller hole with an LED protruding (not shown in the CAD drawing) that would be used for ease in locating the box, should it turn dark before we find the payload. The wooden box had three main purposes. The first was to keep the Styrofoam box and the items inside protected should it have landed in a tree or had any type of rough landing. The second was to make it easier to open and close the Styrofoam box and ensure it stayed closed throughout the flight. The top of the wooden box consisted of two panels connected to the rest of the box by hinges, acting as doors to provide access to the payload and items inside. Also, the wooden box was a little bigger that the Styrofoam box for ease of removing the top, and it allowed space for wires and the experiments. The third purpose of the box was to have a way to attach the payload to the balloon. A small hasp was attached to the top of the panels, and a carabineer was placed through the metal loop, both holding up the payload and ensuring it remained closed.

Figure 11: Final Payload on launch day.

§IV.2 – Reel-In System

Although it was not used for the space launch, a reel-in system was designed for the tethered launch. The reel-in system facilitated the process of lifting the balloon and the payload into the air, as well as helped to reel them back in. This system greatly improved the efficiency of the tethered launch because it improved the speed and amount of work required to complete the trial.

§IV.2.1 – Background

It was necessary that the reel-in system be able to withstand the upward force of both the balloon and the payload. To do this, a DC electric motor was used. However, the motor by itself could not generate enough force to counteract those of the balloon and the payload. Therefore, a system taking advantage of mechanical advantage had to be created.

§IV.2.2 – Mechanical Advantage

The original plan was to use a chain drive. This design required two sprockets connected by a chain. The smaller sprocket would be attached to the motor, while the larger sprocket would be attached to a wooden axle. When the motor was turned on, the smaller sprocket would turn with the motor, and because of the chain, would also cause the larger sprocket and the axle to rotate. The balloon and the payload would be attached by a string to the dowel, and so when the motor was turned on, the string would wind around the axle and reel in the balloon and payload. However, it was very difficult to locate two sprockets and a chain that matched, so the design for the reel-in system had to be altered.

Ultimately, a friction belt drive was used. Two wooden blocks were secured onto a large piece of wood that functioned as the base. A wooden axle was then inserted into the blocks using ball bearings. A large wooden wheel was attached to the axle using wood glue, while a smaller wheel was attached to the motor and secured with a bolt. A serpentine belt from a car engine was then placed around the two wheels. Since the motor was cylindrical, a wooden stand had to be built so that the motor wouldn't roll. This stand was then secured to the base wood at such a distance from the axle to provide enough tension for the belt so that it wouldn't slip. The friction belt drive then worked identically to the chain drive. When the motor was turned on, the small wheel would turn with the motor. Because of the tension acting on the belt, as well as the friction between the belt and the wheels, the movement of the smaller wheel caused the larger wheel to rotate as well. Since the axle was securely fastened to the wheel, it would also spin, causing the string to wrap around it and thus effectively reel in the balloon and the payload.

In order to calculate how much mechanical advantage was generated by the reel-in system, it was first necessary to determine the amount of torque generated by the system. To do this required measurements of the amount of force generated by the reel-in system. Ideally, this would have been done by measuring the amount of acceleration the system exerted while lifting a weight using a motion detector. However, due to setup of this reel-in system, this proved very difficult to measure accurately, and so the force had to be determined through another method. Instead, a force probe was used. The string used to reel in the balloon was first wound around the wooden axle, and a small loop was tied on the end. The force probe was hooked through the loop on the string, and the probe was then attached to LabQuest device so that the force measured could be displayed. The reel-in system was then tested at different amperages and the corresponding forces recorded. This same process was then repeated with just the motor to

determine how much force it exerted. A piece of string was tied around the axle of the motor, and the force probe was looped through that string. It was then possible to measure the force generated by the motor at different amperages. The forces generated by both the reel-in system and the motor proved to be approximately linear, as shown on the graph below.

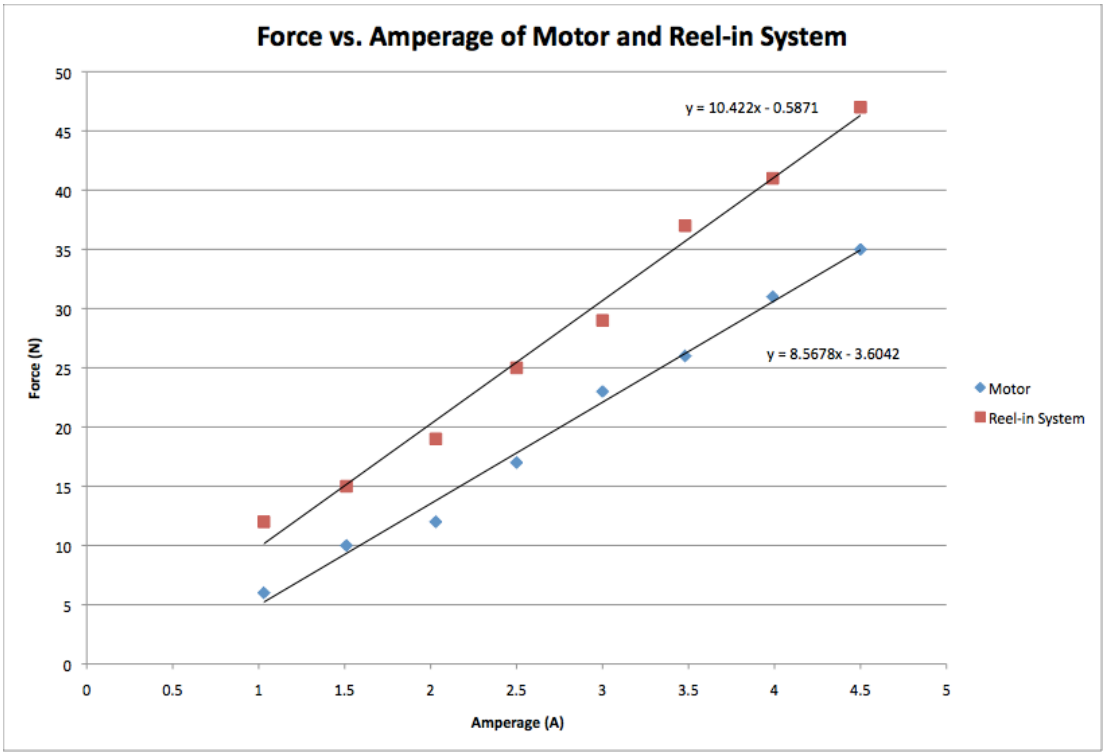


Figure 12: Force versus Amperage for both the motor and the reel-in system. These trends show a linear relationship, with the slope of the best-fit line for the motor being 8.5678 N/A and the slope of the best-fit line for the reel-in system being 10.422 N/A. This graph also demonstrates how the force generated by the reel-in system is always greater than that generated by just the motor because of mechanical advantage.

Once the force generated by the motor and the reel-in system was measured, it was possible to calculate torque. For these calculations, only the force values given at the highest amperage was used.

$$\tau_{motor} = F_{motor} \times r_{motor\ axle}$$

$$\tau_{motor} = (35\ N) \times (0.014\ m)$$

$$\tau_{motor} = 0.49\ N \cdot m$$

$$\tau_{reel\text{-}in\ system} = F_{reel\text{-}in\ system} \times r_{reel\text{-}in\ system\ axle}$$

$$\tau_{reel\text{-}in\ system} = (47\ N) \times (0.041\ m)$$

$$\tau_{reel\text{-}in\ system} = 1.927\ N \cdot m$$

The observed mechanical advantage can then be calculated by making a ratio of the two torque values.

$$\tau_{\text{reel-in system}} / \tau_{\text{motor}} = 1.927 \text{ N·m} / 0.49 \text{ N·m} = 3.93265306$$

The observed mechanical advantage is 3.93.

However, the theoretical mechanical advantage can also be calculated by simply making a ratio of the radius of the large wheel attached to the axle to the radius of the small wheel attached to the motor.

$$r_{\text{large wheel}} / r_{\text{small wheel}} = 105.21 \text{ mm} / 26 \text{ mm} = 4.04653846$$

The theoretical mechanical advantage is 4.05.

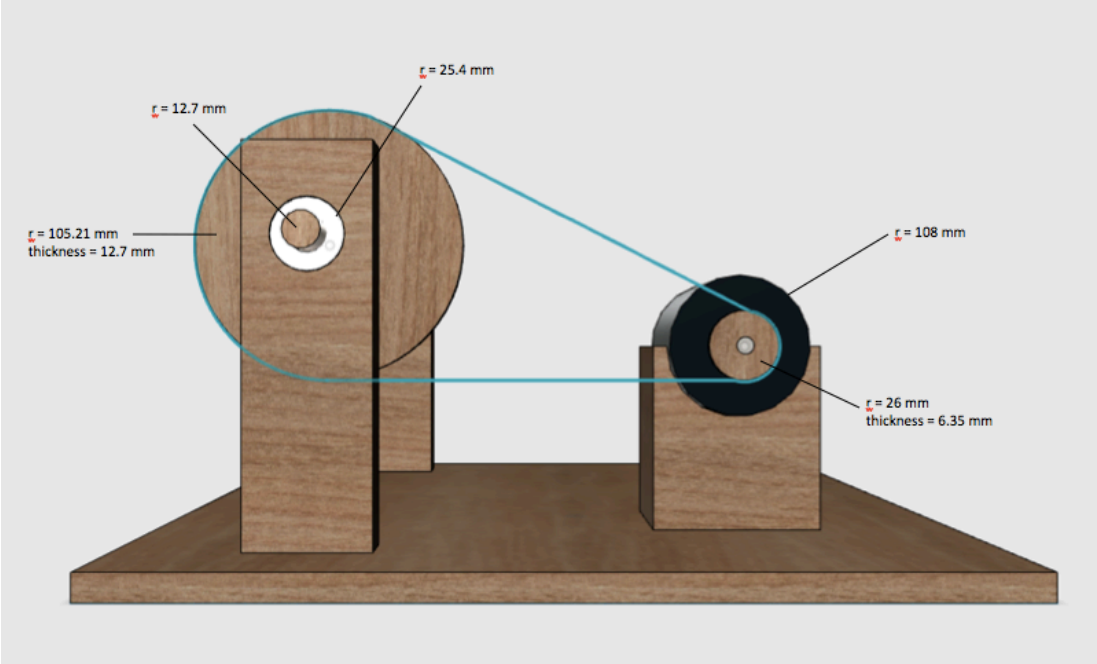


Figure 13: Side view of reel-in system. Cyan loop is the placement of the belt. CAD drawing done on 123D Design.

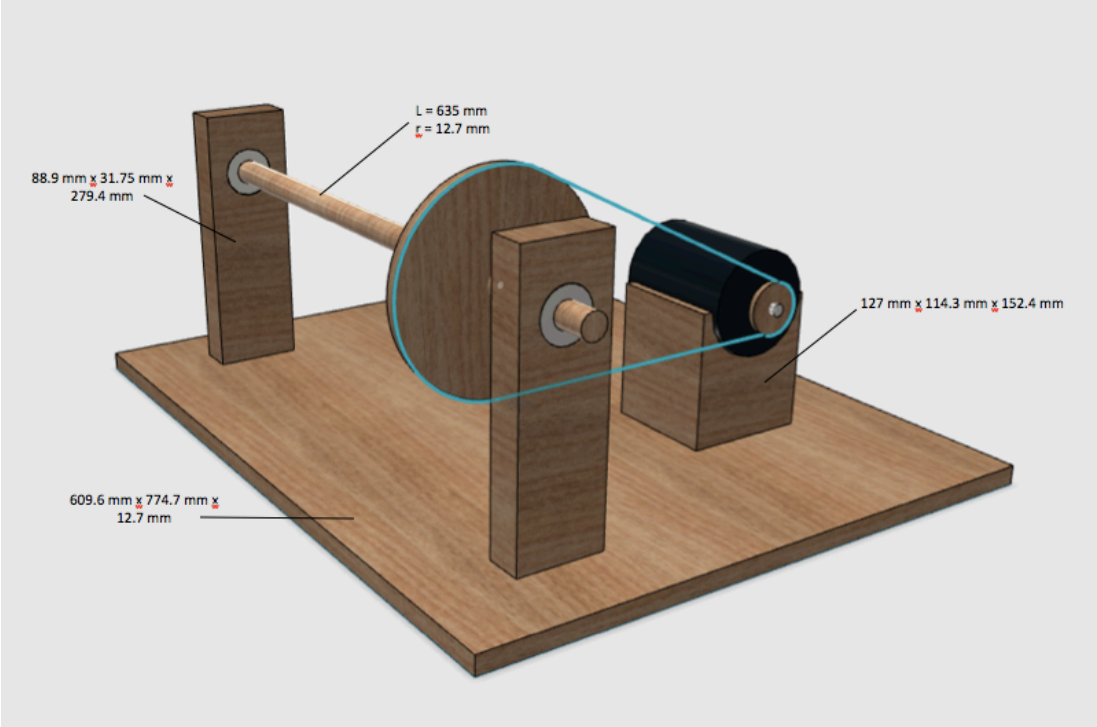
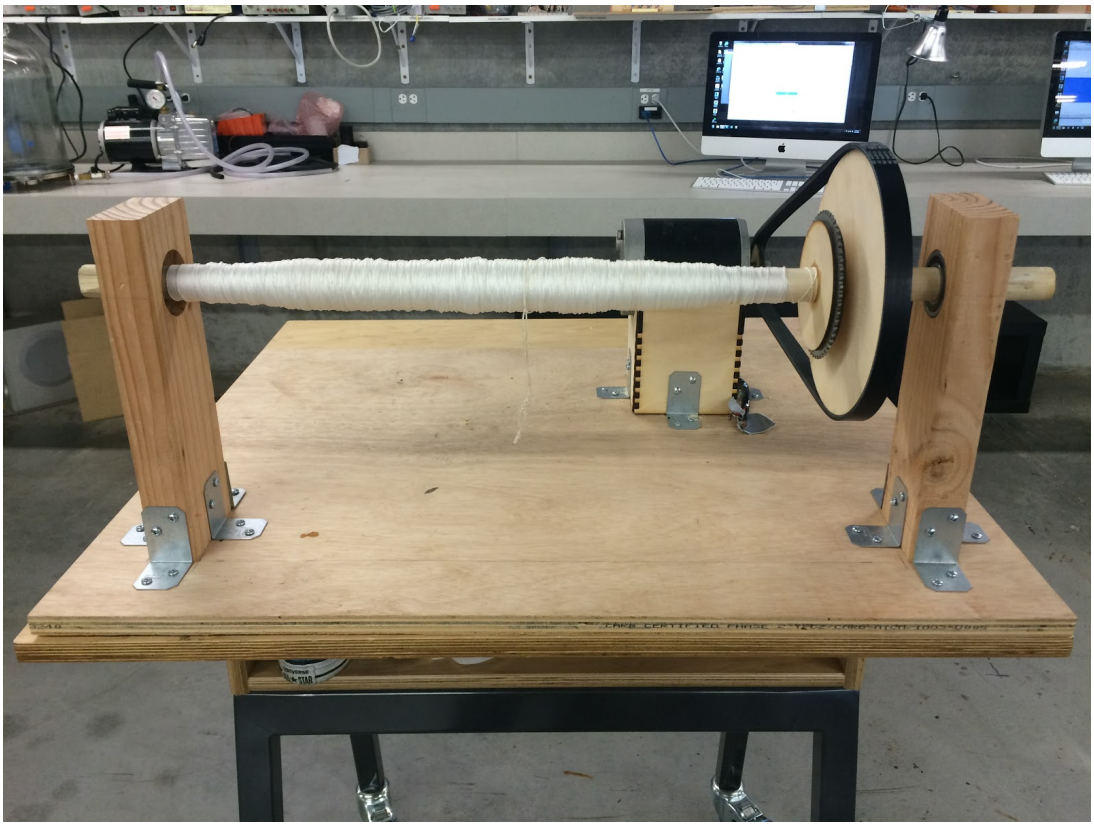


Figure 14: Top diagonal view of reel-in system. CAD drawing done on 123D Design.



Fixure 15: Picture from front of reel-in system.

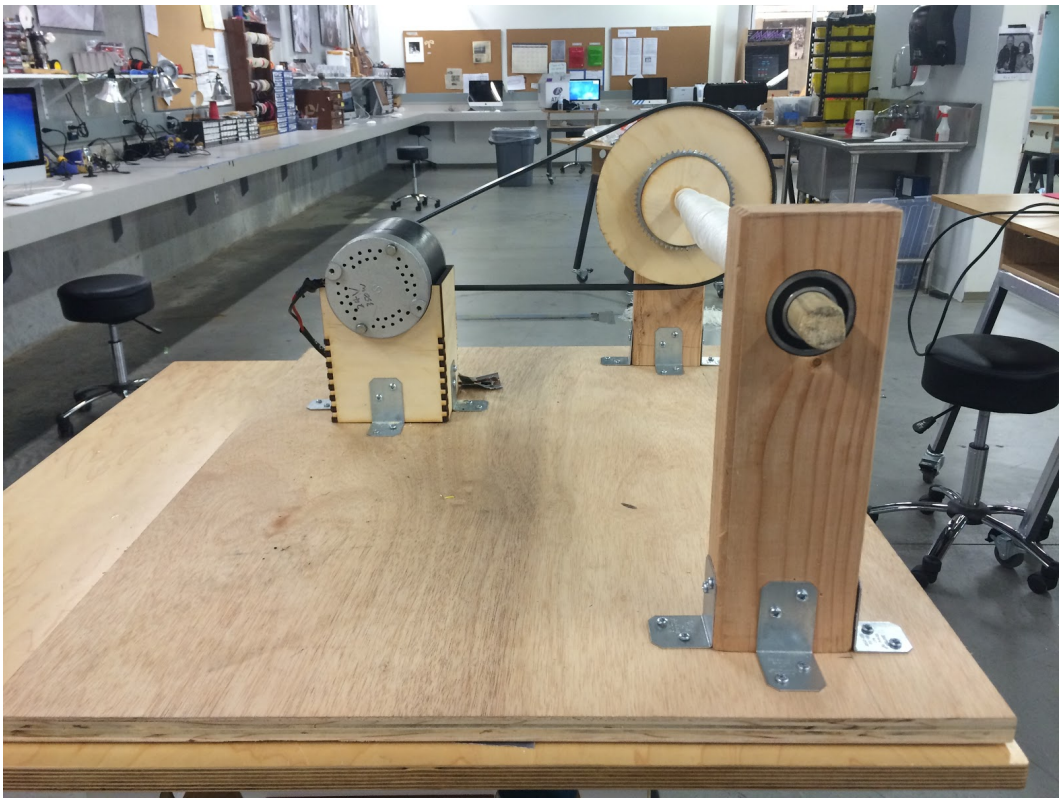


Figure 16: Picture from side of reel-in system.

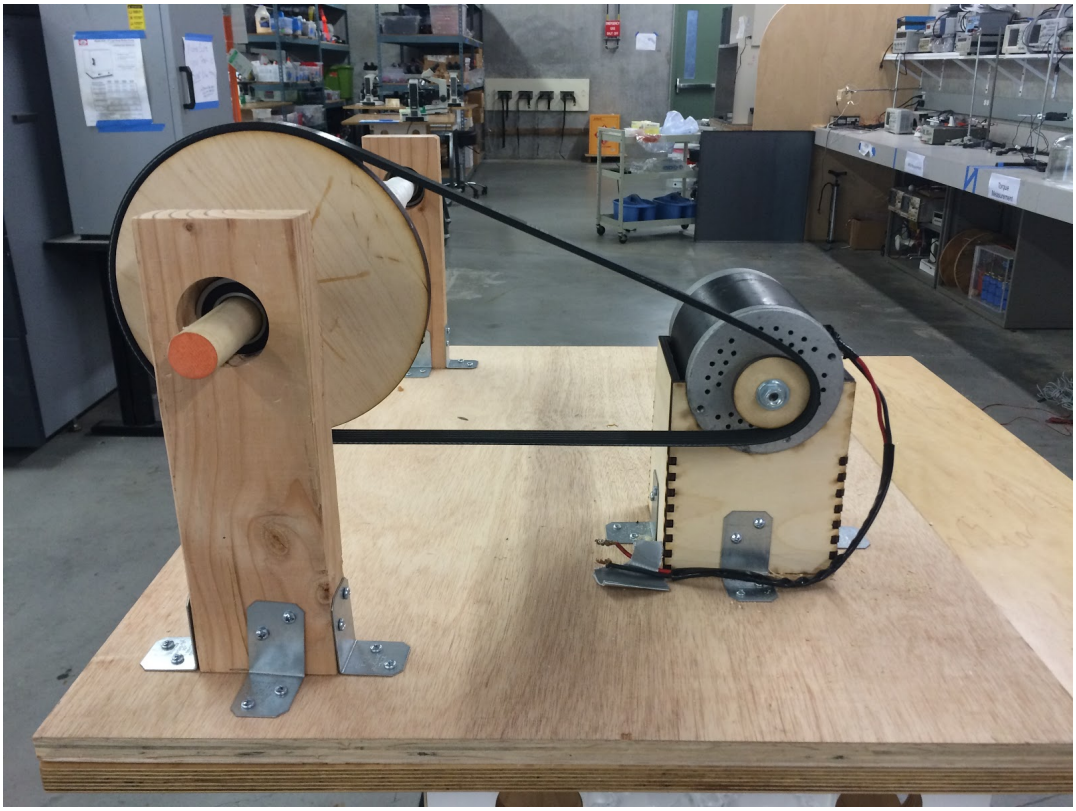


Figure 17: Picture from opposite side of reel-in system.

§IV.3 – Cut Down Mechanism

A cut down mechanism was designed for the free balloon launch. The cut down system was used to detach the payload and parachute from the balloon at the target height of 100,000 feet. This system increased the probability of a successful flight by allowing us to control when we wanted the payload to be detached as opposed to just letting the payload travel until the balloon popped due to the expansion of the helium.

The cut down mechanism was originally planned to use both temperature and time as the factors to trigger it. As it got closer to launch date however, the team decided that if we took a simpler approach and only used time as the trigger for the cut down mechanism, we would be able to achieve a error-free cut. The cut down mechanism was set to trigger two hours after the launching of our balloon, using past years' flight duration calculations as the basis of the trigger time.

The cut down mechanism involved four main parts. It was composed of an arduino for commands, a set of capacitors for power, a breadboard for circuitry, and nichrome wire for the heating of the string. All of these pieces were mounted onto a placeholder that was designed in order to minimize the space taken up in the payload and also make it neater. On one side of the placeholder was the arduino and breadboard. These two pieces were placed together because they are needed to make each other work. The arduino is what triggers the demands given by the code and makes the whole cut down mechanism function, while the circuit is what makes the flow of energy possible. On the other side of the breadboard was a 7 volt battery and two capacitors wired in series for a total voltage of 5.3. The battery is what powered the arduino, while the capacitors were what gave energy to the circuit when triggered. The electronics involved in the cut down mechanism were quite simple. After two hours, the arduino sent 5 volts of energy into the circuit, this voltage switched on a relay, putting current from the capacitors into a nichrome wire wrapped around a piece of string that was keeping the balloon and parachute/payload attached. This string was then heated up to the point of melting and resulted in the release of the payload from the balloon. The cut down mechanism worked as intended on the free launch.

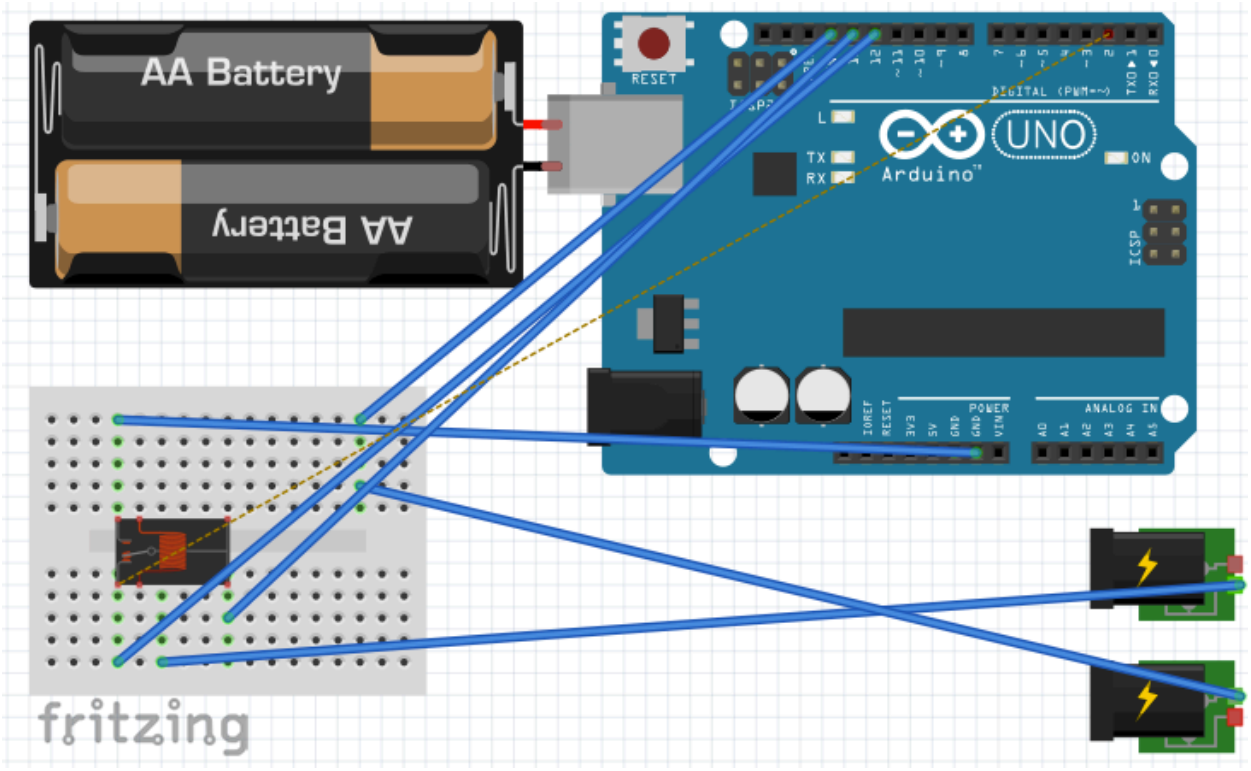


Figure 18: Diagram of circuit for cut down mechanism. Here, the 2 AA batteries represent the 9V battery used in the actual experiment to power the arduino, and the two figures with lighting logo on them represent the capacitors used for the melting of the string.

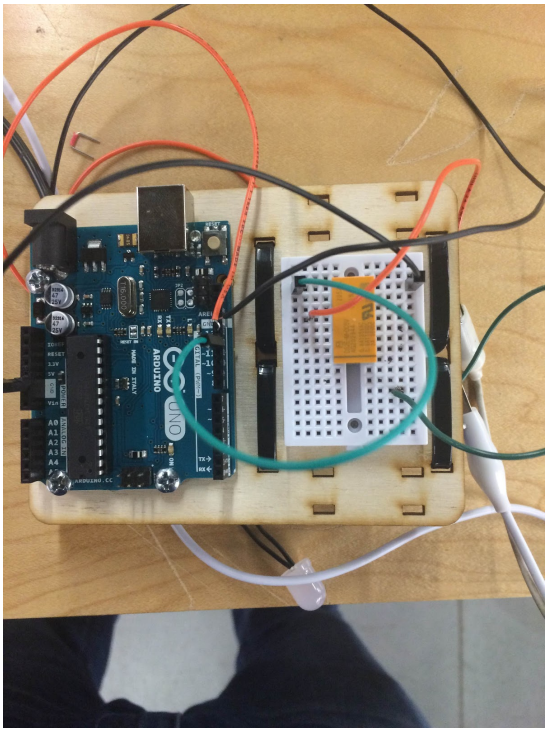


Figure 19: Image of one side of cut down.

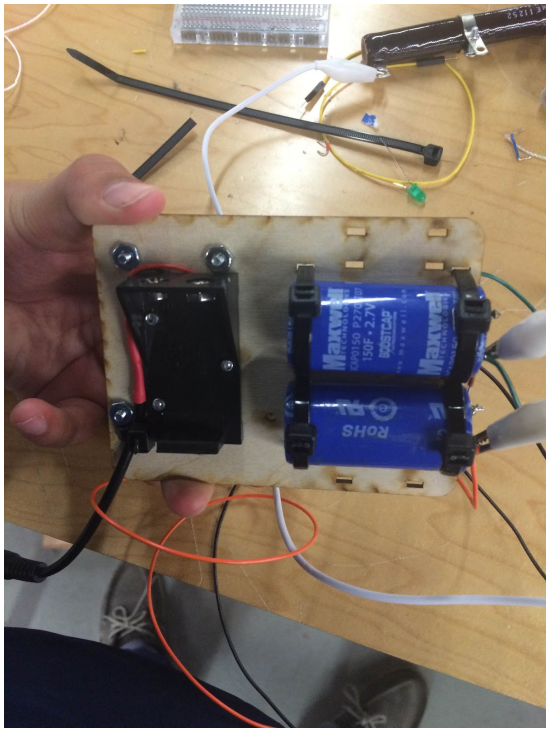


Figure 20: Image of the other side of the cut down.

§IV.4 – Force and Buoyancy Calculations

§IV.4.1 – Pressure vs. Altitude

To find the exponential model of pressure vs. altitude, two equations are needed. The ideal gas equation, in terms of pressure and density:

$$\rho = \frac{pM}{RT}$$

where M is the molar mass of the atmosphere, and along with T, is assumed to be the same at all elevations. Also needed is the relationship between pressure and density in a fluid in equilibrium:

$$\frac{dp}{dy} = -\rho g$$

Assume g is the same at all elevations. Combining the two equations yields:

$$\frac{dp}{p} = -\frac{gM}{RT}\,dy$$

To solve for p, the integral is taken from p_1(pressure at sea level) to p_2(pressure at certain elevation) on the left side and from y_1(sea level) to y_2(elevation):

$$\int_{p_1}^{p_2} \frac{1}{p}dp = -\frac{gM}{RT}\int_{y_1}^{y_2} dy$$

$$ln(\frac{p_2}{p_1}) = -\frac{gM}{RT}(y_2 - y_1)$$

Let y_1= 0, and p_1= 101.325 kPa. solving for p_2, the equation is:

$$p=101.325 * e^{-\frac{gMy}{RT}}$$

plugging in g = 9.81m/s^2, M = .029 kg/mol, R= .8314 m^3*hPa/K*mol, and T=298K, the equation is (solving for p in kPa):

$$p=101.325 * e^{-.000115y}$$

Nasa's model differs from this equation (solving for p in kPa):

$$for\ y \leq 11{,}000m\ (Troposphere):$$

$$p = 101.29 * \left[\frac{(288.15 - .0069y)}{288.08}\right]^{5.256}$$

$$for\ 11{,}000m < y \leq 25{,}000m\ (Lower\ Stratosphere):$$

$$p = 22.65 * e^{(1.73-.00157y)}$$

$$for\ y > 25{,}000m\ (Upper\ Stratosphere):$$

$$p = 2.488 * \left[\frac{141.89 + .00299y}{216.6}\right]^{-11.388}$$

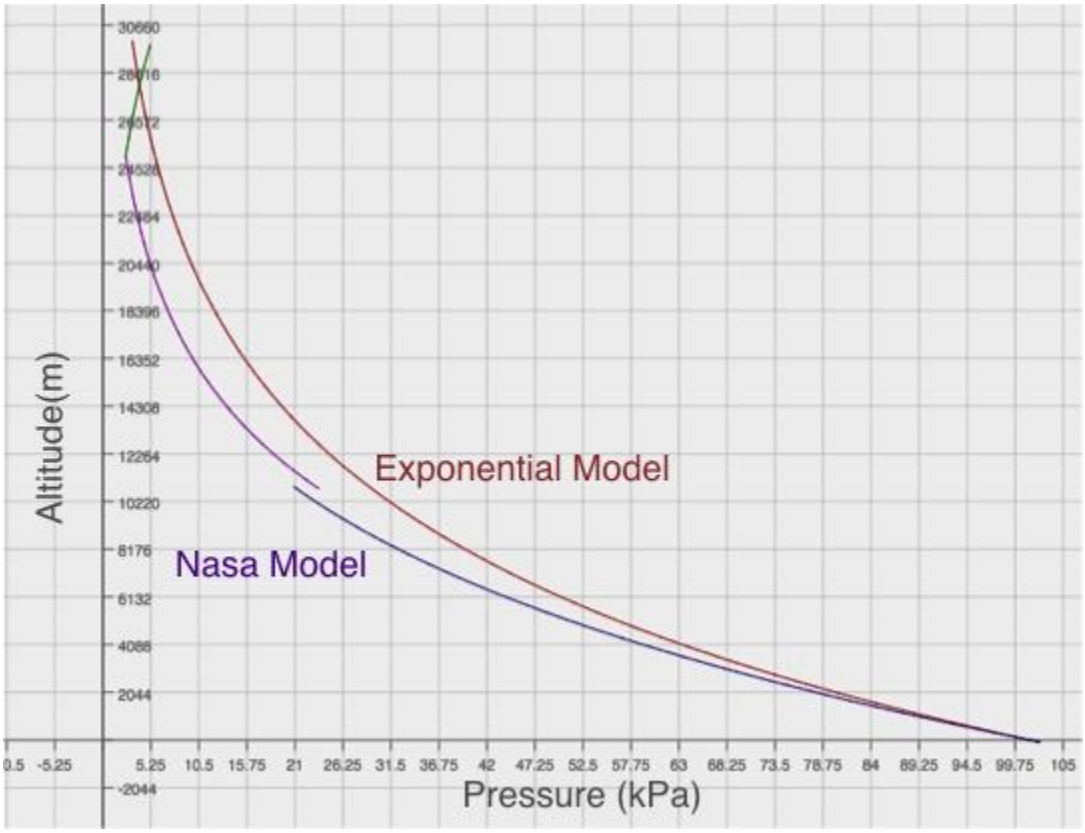


Figure 21: Comparing exponential and Nasa models of altitude vs. pressure. There is not a graph for our data because we did not measure GPS altitudes. The exponential model breaks down at higher altitudes because the model assumes constant temperature. Graph made on www.meta-calculator.com. [s6]

§IV.4.2 – Buoyancy Force

For the calculation of the buoyant force on the balloon at its highest point, $\rho_{air} * V_{balloon}$ is the same as $\rho_{air} * V_{balloon}$ on the ground because as the balloon gets higher, density of air (ρ_{air}) decreases but the volume of the balloon ($V_{balloon}$) increases. The only difference is gravity, which at our maximum altitude of 15.4km is 9.76m/s^2

$$F_{buoyant} = \rho_{air} * V_{balloon} * g$$

$$F_{buoyant} = 1.225\frac{kg}{m^3} * 3.198m^3 * 9.76\frac{m}{s^2}$$

$$F_{buoyant} = 38.24N$$

$$F_{payload} = m_{payload} * g$$
$$F_{payload} = 2.018kg * 9.76\frac{m}{s^2}$$
$$F_{payload} = 19.70N$$

$$F_{He} = \rho_{He} * V_{balloon} * g$$
$$F_{He} = .1786\frac{kg}{m^3} * 3.198m^3 * 9.76\frac{m}{s^2}$$
$$F_{He} = 5.597N$$
$$\Sigma F_{upward} = F_{buoyant} - (F_{payload} + F_{He})$$
$$\Sigma F_{upward} = 38.24N - (19.70N + 5.597N)$$
$$\boldsymbol{\Sigma F_{upward} = 12.943N}$$

§IV.5 – Camera

A GoPro Hero 3 silver edition was used in the tethered launch of our experiment. The GoPro was able to give us 1080p video of the launch and captured the surrounding areas of Menlo Park. The GoPro's battery life varied based on the temperature that it was exposed to.

Table 1: GoPro Hero 3 No Battery Pack Recording Times

Temperature (°C)	Battery Life Recording at 1080p, 30fps (H:MM:SS)
22.2 (Room Temp.)	2:10:08
-25	2:09:00
-40	1:35:43
-55	0:53:20
-70	0:41:51

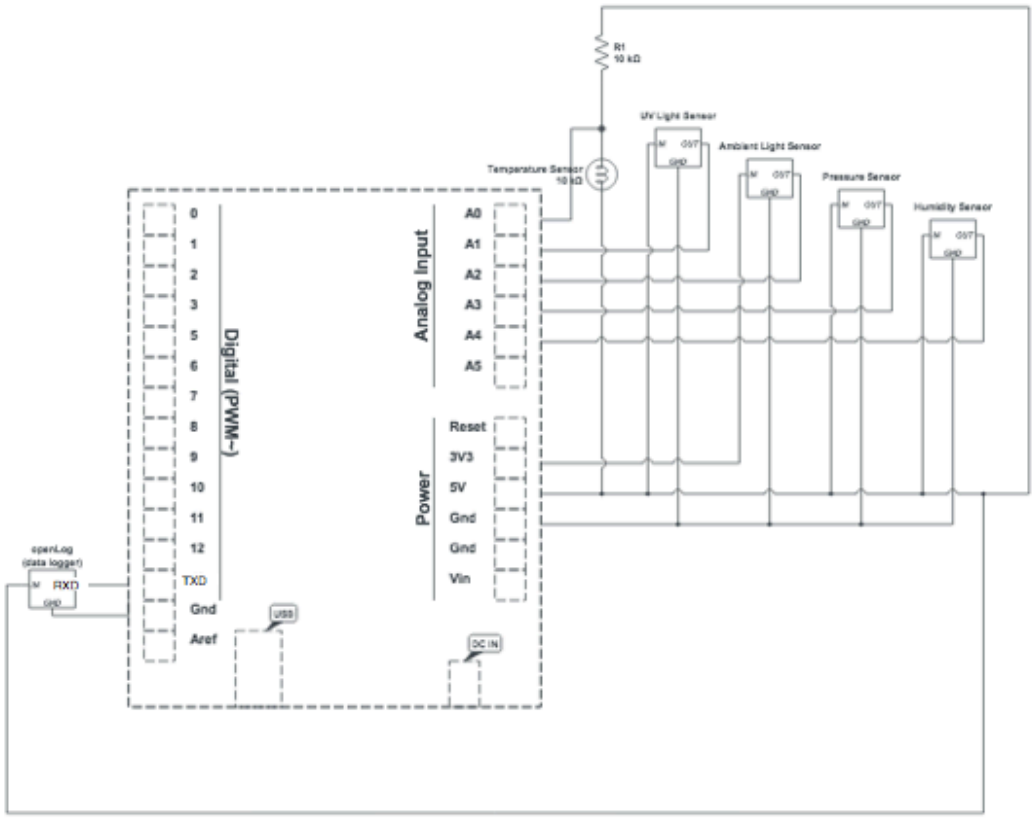
§IV.6 – Batteries

For this project, 600 mAh 9V lithium-ion rechargeable batteries were used to power the arduinos that controlled the temperature sensor, humidity sensor, UV sensor, ambient light sensor, gas pressure sensor, GPS, and the cut down mechanism. Lithium-ion batteries can work in extremely cold temperatures. This was important since the balloon was going to be reaching a height of 100,000 feet, or approximately 30 km, which is well into the stratosphere. Although not as cold at this height (only about -40 °C), temperatures between 10 km and 20 km above the ground can reach -60 °C [b1]. Since most batteries don't work at such cold temperatures, it was important to conduct research and determine which batteries could continue to function at such high altitudes and in such cold conditions. Based on our research, we determined that a lithium battery would be the best solution for providing power during the space launch. Ultimately, the lithium-ion batteries were selected because they functioned at the necessary temperatures, were more cost efficient, and were rechargeable, making it possible to test the batteries before being used for the final launch.

V. Electrical and Software Design

§V.1 – Data Collection

The Data Acquisitioning system made use of an arduino microcontroller given its easy programming advantage and its reliability under extreme conditions (i.e. low temperature, low pressure, etc). The code for the data-taking arduino can be found in Appendix D.1. A pressure sensor, humidity sensor, temperature sensor, uv light sensor, and ambient light sensor were all connected to different inputs on the arduino. Each sensor served as potentiometer, changing its resistance with changes in pressure, humidity, temperature, uv light, and external light. The arduino recorded the voltage in each input every second by sending the information to the OpenLog data logger, where it was stored in a .txt file on a micro SD card.



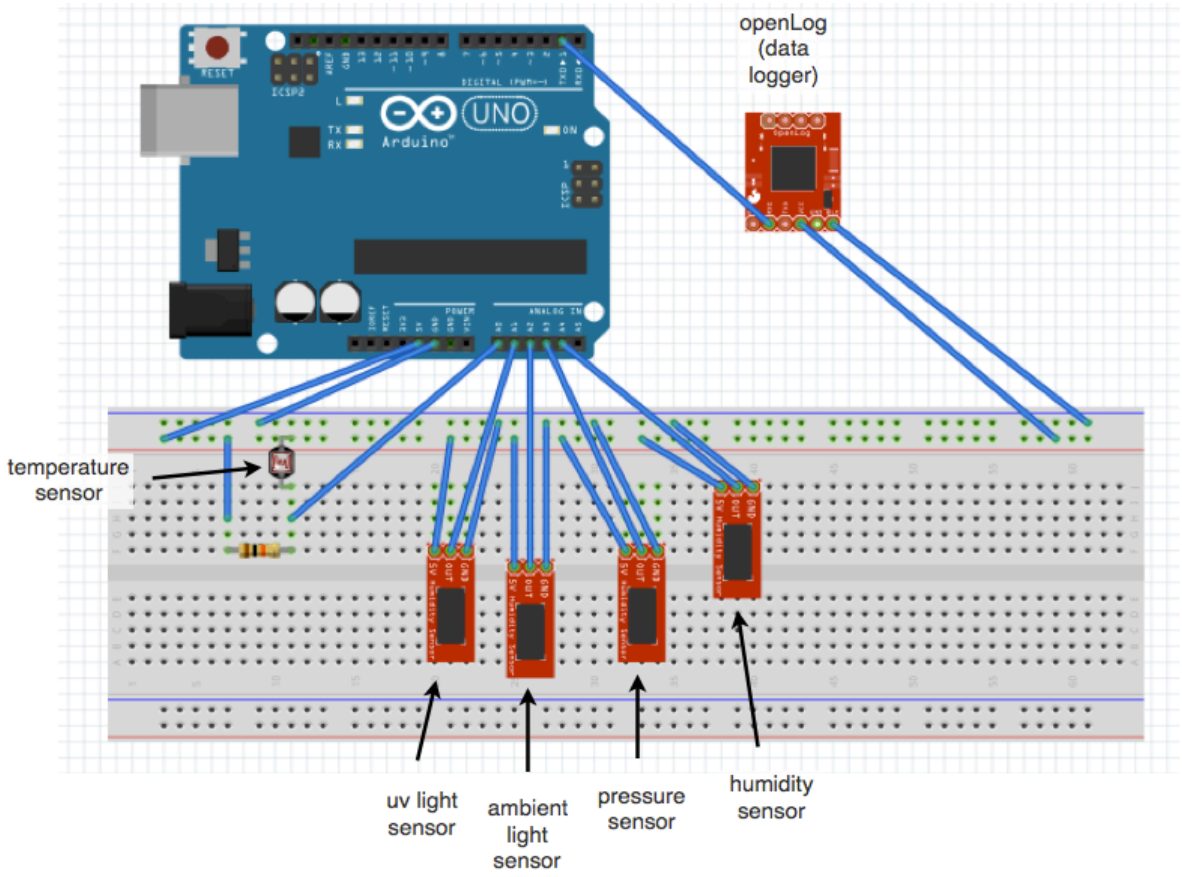


Figure 22 and 22.5: A schematic and visual diagram of the data acquisitioning circuit. While each sensor had varying appearances, they all had the same input and output configuration. Each sensor was connected to power, ground, and a serial port. The arduino transmitted the voltage to the OpenLog (data logger) where it was recorded in a .txt file.

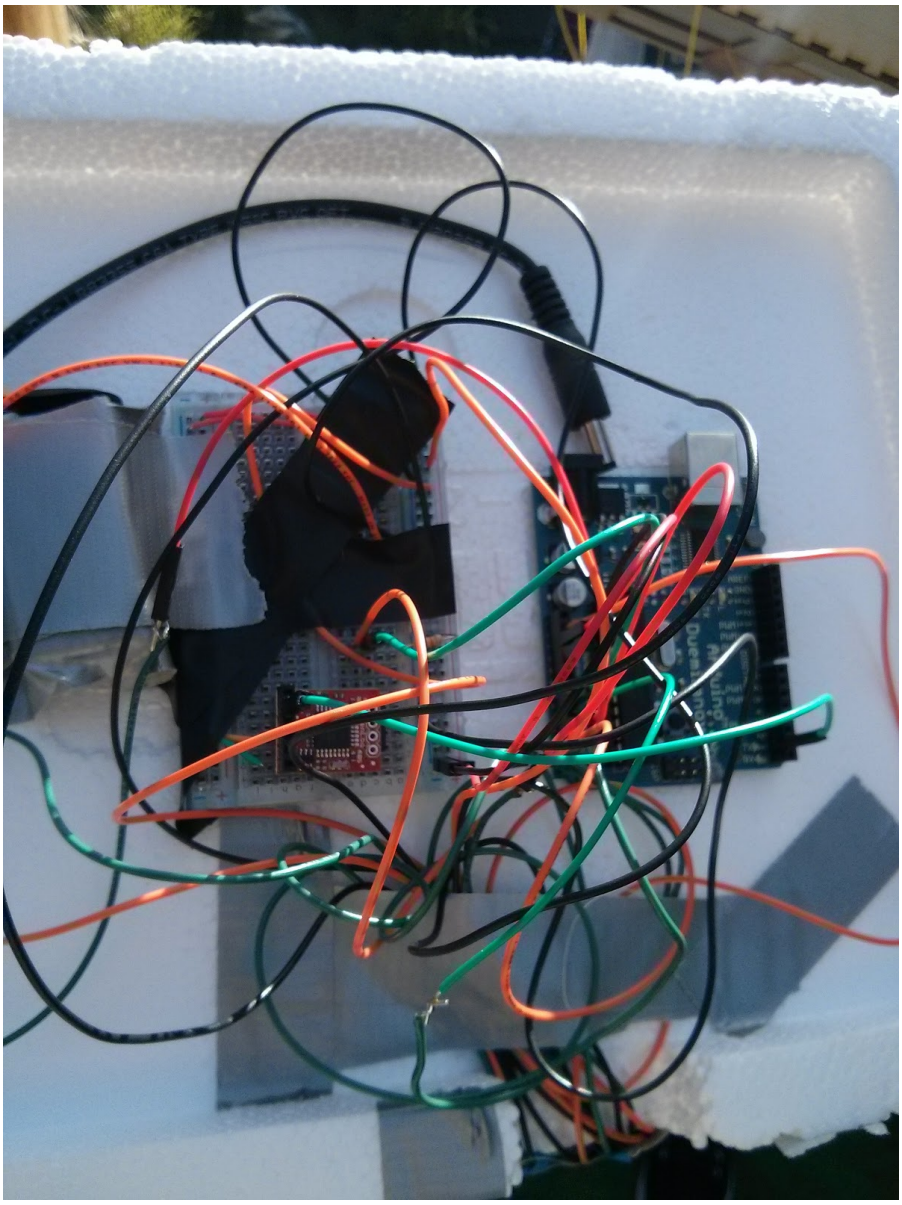


Figure 23: A Photograph of the arduino connected to the sensors on the lid of the payload.

§V.2 – GPS

GPS data can be received in various formats, such as GPGGA, GPGSV, GPRMA, and GPRMC. Each contains different amounts of information. GPRMC is the simplest and the one that we decided to parse. GPS satellite data in the form of GPRMC contains the GPS antenna's time of fix, health status, latitude, longitude, speed over ground in knots, date of fix, magnetic variation, and a mandatory checksum. For our purposes, using solely the latitude and longitude was adequate [qq4].

Figure 24: Pictured above are the Radiometrix NTX2 and NRX2 radio transmitter and receiver modules. For the NTX2, pin 1 was connected to the ground lead coming from the transmitter antenna and pin 2 to the input lead coming from the transmitter antenna. Pin 3 was connected to ground, pin 4 was connected to the Arduino's digital pin 3, pin 5 was connected to the Arduino's 5V pin, pin 6 was connected to the Arduino's ground pin, and pin 7 was connected to the Arduino's TX0 port. Whenever the arduino printed something, it would be processed within the NTX chip and output to the antenna. For the NRX2, pin 1 was connected to the positive lead coming from the yagi antenna and pin 2 to the negative lead coming from the yagi antenna. Pin 4 was connected to the arduino's ground pin, pin 5 was connected to the arduino's 5V pin, and pin 7 was connected to the arduino's RX0 port. Whenever the antenna received radio waves, it would be processed through the NRX chip, and the output data would be sent through pin 7 to the arduino.

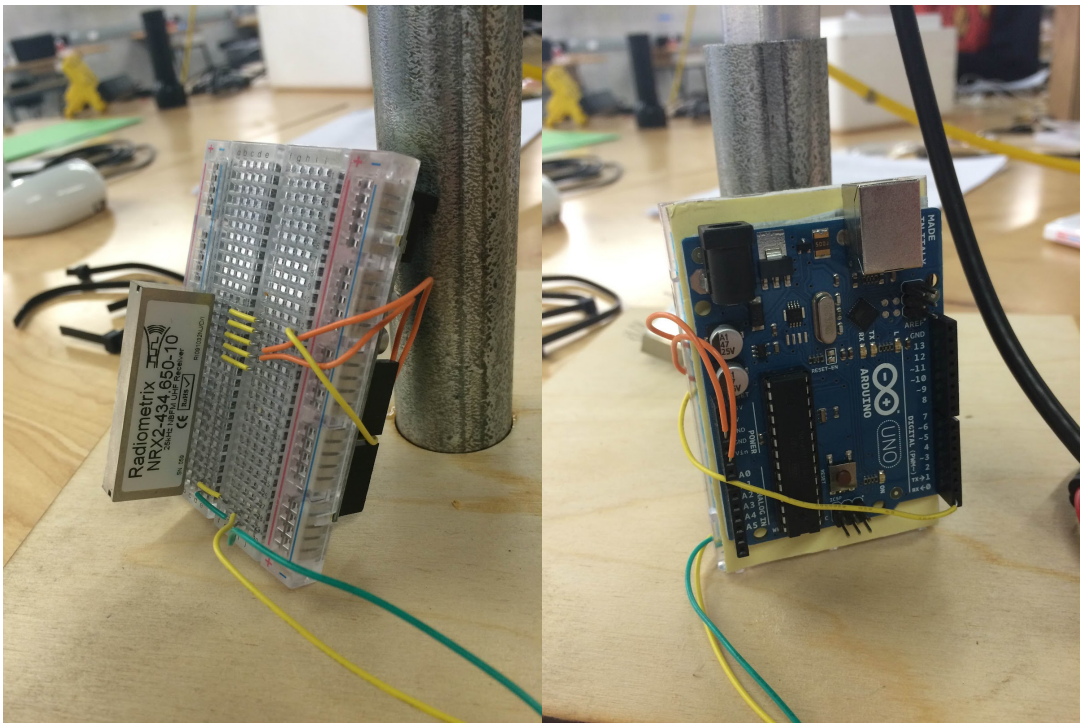
The transmitter module inside the payload was a Radiometrix NTX2, which transmitted at a frequency of 434.650Mhz. We used a matching receiver module in order to receive the transmissions at 434MHz, a Radiometrix NRX2.

The code referred to in this paragraph can be found in Appendix D.2. The goal of the code loaded onto the transmitter arduino inside of our payload was to transmit the payload's latitude and longitude down to ground in real time. The SoftwareSerial line in the aforementioned code imports the Arduino Software Serial library so as to enable communication between the arduino and the Sparkfun GPS chip. The purpose of the starter and ender sequences are to identify the data string from the rest of the miscellaneous radio transmissions that may be on the same frequency as we are, such as other groups in ASR. Inside the loop in the code, if

there is GPS data available from the GPS antenna, the program will read it in and store it. After all of the data from the GPS is done being read in, the GPS data string is translated into a readable, comprehensible string, seen in the long, final "if" structure in the code. The transmitter chip is given time to turn on, given time to transmit the string, and then is turned off to conserve power.

The method displayGPS1(), seen in Appendix D.3, is crucial to GPS transmission. It is responsible for fetching data from the GPS chip, extracting the latitude and longitude, and adding that information to the string that is transmitted. First, it calls the getField(char* buffer, int index) method in order to determine if the string from the GPS chip is in the GPRMC format(detailed above). If the original string is indeed in the correct format, it continues to call the getField() method, each time with a shorter and shorter version of the original string. The purpose of this is to extract the latitude and longitude. sendString, a variable containing the information that the antenna ends up transmitting, is concatenated (adds to itself) each time the shorter and shorter string is determined.

The code that was loaded onto the arduino on the ground, seen in Appendix D.4 is more complicated to understand because it is responsible for decoding radio waves into a comprehensible string with relevant data in it. The loop is responsible for reading in radio wave data and storing it in a string. After the data is stored, it sends the string to the extract() method in order to get the information we are looking for. If nothing could be extracted, it prints "no message".



Source: http://www.radiometrix.com/content/ntx2
Figure 25: Pictured above is the receiver module. It is comprised of an arduino and a breadboard. Nothing had to be soldered because if a wire became disconnected, the wire could easily be reconnected. Rigidity and reliance weren't extremely crucial here. The long yellow and green wires led to the leads of the yagi antenna we used. The Arduino and breadboard were taped together so that it would be easier to move and use. The combined module was zip tied to the antenna stand for improved portability.

The extract(string, string, string) method, seen in Appendix D.5 is responsible for extracting the desired information from a given string (which is read in via the receiver chip). The basic methodology is to loop through every character of the given string, and if the starter and ender sequence are found, extract the data between the two sequences.

One problem that occurred during the free launch was that transmitted data was only received for the first half hour. Intermittent latitude and longitude transmissions were being received, but stopped receiving them relatively early into the balloon's flight. This may have been because the transmitting antenna and receiving antenna were simply too far away from each other; after all, the transmit antenna was only powered by a 9V battery, which gave power to it through the Radiometrix transmitter chip, which in turn was powered by the 5V Arduino Pin. Another problem for the early reception termination could have been that our antennas weren't pointed at each other. Although we tried to point the receiving yagi antenna at where we

estimated the balloon to be, the transmit antenna may quite possibly have been pointing in the opposite direction.

Another problem that occurred was that the GPS data that was gleaned from the microSD card cuts off a little after halfway through the flight. The most logical explanation for this error is that the batteries ran out early. We determined that this theory was incorrect, as when the payload as opened, about five minutes after retrieval, the LEDs on the arduino, GPS chip, and SD card adapter were all still on. This means that all of components part of the transmitting procedure on the payload had power throughout the whole flight. The GPS Chip's led was blinking on and off, which means that it had a fix. By this logic, the problem was narrowed down to either an error in the code or an inherent problem in saving data to an SD card. After a careful review of the code, our group determined that the issue was indeed in the transfer of data from the arduino to the SD card. There is no reason why the arduino would stop outputting data after a certain amount of time within the code. There are no if statements that are time based that influence whether or whether or not data is/should be written. The code that was loaded onto the arduino in the payload that was responsible for transmitting the Latitude and Longitude can be seen in Appendix E.2 through Appendix E.5. As you can see, the actual values of whatever was in the string from the GPS chip never get changed.

There was another problem with the GPS data on the SD card. All of the latitude and longitude values are of by a constant, fixed amount. After careful review of the code in Appendix E.2 through Appendix E.5, it was determined that there is no reason for the interpreted latitude and longitude to be different from the payload's actual latitude and longitude. The numbers are never actually changed, and are merely transferred from variable to variable.

§V.2.1 – GPS Syntax:

eg1. $GPRMC,081836,A,3751.65,S,14507.36,E,000.0,360.0,130998,011.3,E*62
eg2. $GPRMC,225446,A,4916.45,N,12311.12,W,000.5,054.7,191194,020.3,E*68

225446 Time of fix 22:54:46 UTC
A Navigation receiver warning A = OK, V = warning
4916.45,N Latitude 49 deg. 16.45 min North
12311.12,W Longitude 123 deg. 11.12 min West
000.5 Speed over ground, Knots
054.7 Course Made Good, True
191194 Date of fix 19 November 1994

020.3,E Magnetic variation 20.3 degrees East [qq4]
*68 mandatory checksum

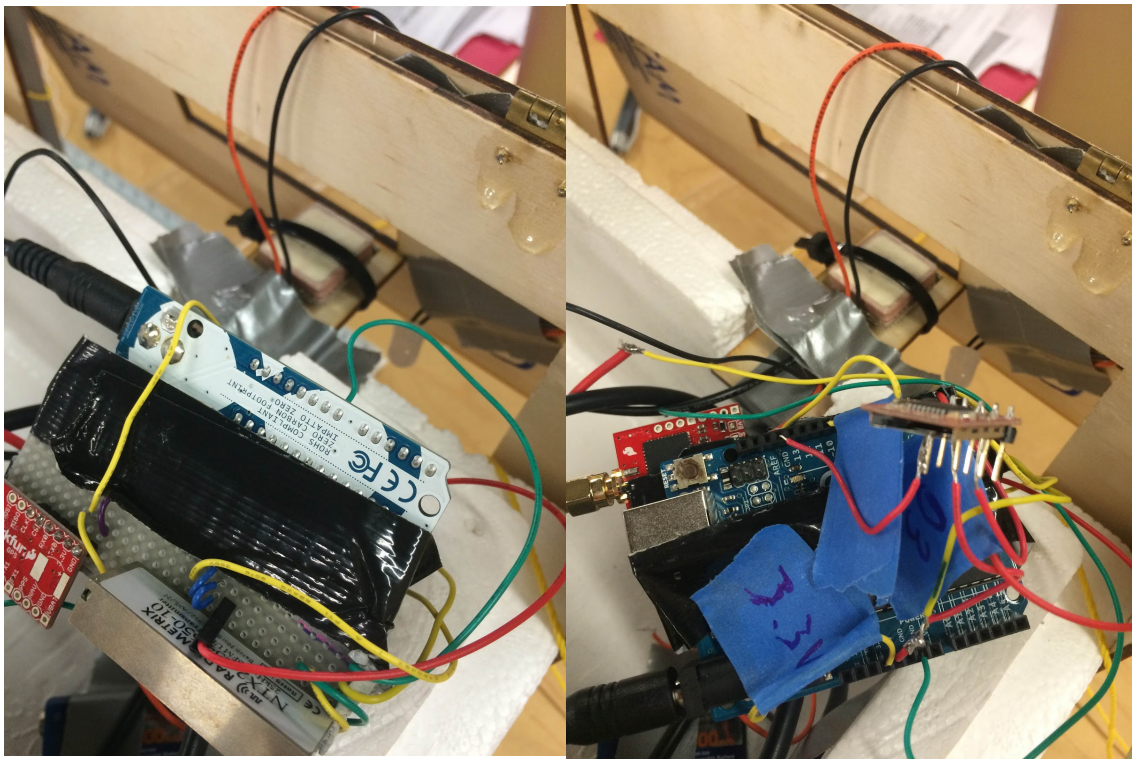


Figure 26: Above is a combined photo of the transmit module inside of our payload. An Arduino was taped to a solder board in order to conserve space, and in an effort to minimize the chances that any cables become unplugged. The GPS antenna, seen near the middle-top of the photos, is on the exterior of the payload. It's located on the outside because it wasn't able to get a GPS fix while within the Styrofoam box, covered by the wooden frame.

VI. GPS and Radio Tracking

Radio waves can transmit data, pictures, music, and a variety of different information through the air while at the same time invisible to the human eye. The actual technology behind a radio isn't complicated either [qq2]. Transmitters send electromagnetic energy in radio waves; two objects send information from one place to another without any wired, physical connection. Transmitted from a wire, a radio wave has three aspects to it: wavelength, the distance between adjacent crests, frequency, the number of waves that arrive every second, measured in hertz, and amplitude, how much distance there is from the top of a crest to the bottom of a trough divided by two [qq2].

Information is transmitted by adding it to a larger radio wave called a carrier; this process is called modulation. Sometimes, the information is added in a way that it causes fluctuation in the larger wave's frequency, which causes frequency modulation, commonly known as FM. One problem with FM radio wave transmission is that the integrity of the signal relies upon the carrier radio wave not getting interefered with. If there are any other radio waves at the same frequency of the carrier wave between the transmitter and receiver, the carrier wave will be misaligned and lose the signal radio message [qq3].

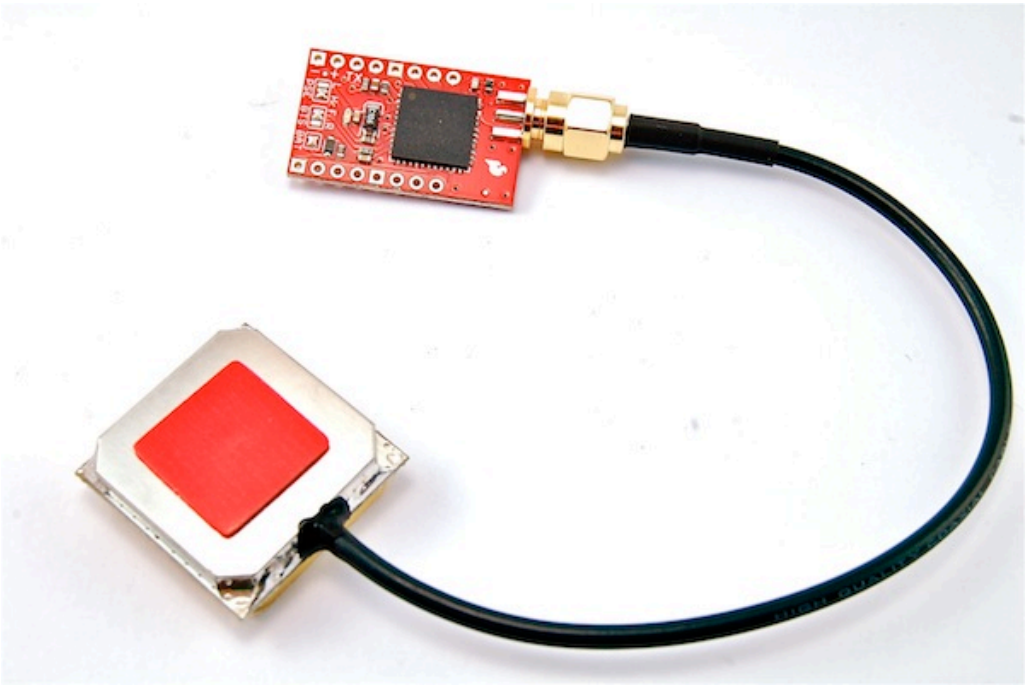


Figure 27: Pictured above are the GPS antenna and the Sparkfun Venus GPS chip. The GPS chip is the small red object in the photo, with places for pins to go along either side. The first three pins on the Sparkfun module closest to the left on the side farthest from the top of the photo were used. The first pin was connected to ground, the second pin was connected to the Arduino's

3.3V pin, and the third pin was connected to the Arduino's RX0 pin. This way, when the chip output information through its TX0 port, the Arduino would receive it.

The payload was equipped with a radio wave transmitter chip. The goal of its implementation was to transmit GPS data in real time to a receiving station on the ground, which, in our case, was a yagi antenna connected to a receiver chip, connected to an Arduino, and connected to a laptop. After the GPS data was transformed by the Sparkfun Venus GPS module, it was parsed to extract the latitude and longitude, and used to locate the payload. The radio transmitter module was connected to an Arduino and a dipole antenna. Every half second, the radio transmitter would receive a string from the Arduino and transmit it through the antenna. The square-shaped GPS antenna got a "fix" from GPS satellites, and thus changed the red LED on the GPS decoding chip from a constant on to blinking on and off. It sends the signal it gets from the GPS satellites to the Sparkfun GPS module, which is responsible for decoding the signal into what can be interpreted by the Arduino.

The GPS, or global positioning system, is a network of about 50 satellites that orbit the earth. Regardless of where you are on the surface it is likely that, at least four satellites have a line of sight to you. The satellites transmit information about themselves, such as latitude and longitude, and our GPS antenna recieves these signals. Then, the GPS chip (the one we used was designed by Sparkfun) interprets the signals from at least three satellites through a process called Trilateration. The chip then calculates its distance from each of the three satellites. Then, using the satellites' positions and respective distances, it determines a position that fits the distances from each of the three satellites [qq1].

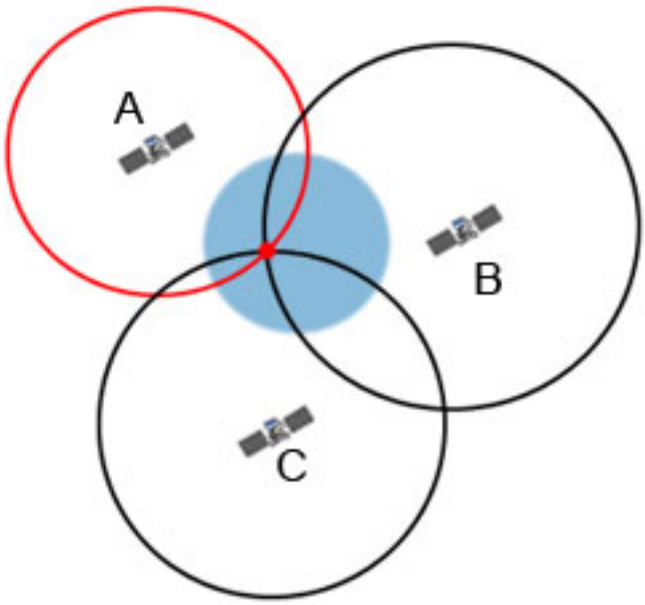


Figure 28: Above is a diagram outlining the trilateration method. The GPS unit, in our case, our payload, would be represented by the intersection of the three circles. Satellites A, B, and C transmit their distance away from the GPS unit to the GPS unit, represented by the circles of radius(distance from the GPS unit to the satellite). The GPS unit must be in the place where all three circles intersect.

VII. Sensor Calibration

§VII.1 – Calibration

§VII.1.1 – Pressure

To calibrate the Vernier Gas Pressure Sensor used to take pressure data, the sensor was attached to an Arduino microprocessor, which took the voltage across the Arduino every second and recorded it on a micro-SD card, and placed in a vacuum chamber with an Extech Datalogger SD2700, which recorded data every five seconds and was used as the calibration standard. The pressure inside the vacuum chamber was then lowered from the surrounding atmospheric pressure, 1010.5 hPa, to 56 hPa. Then, the pressure inside the vacuum chamber was slowly increased. In order to ensure the separation of individual data points for ease of calibration, the release valve was opened to allow air into the vacuum chamber, closed for a time to maintain a constant pressure that would appear in the trends of the data from the Arduino and Extech Datalogger, and then opened again, repeating this process multiple times. These constant pressure points were then plotted together (Figure X) to get an equation that would convert the Arduino reading to pressure in hPa, with five Arduino points averaged for every Extech Datalogger point.

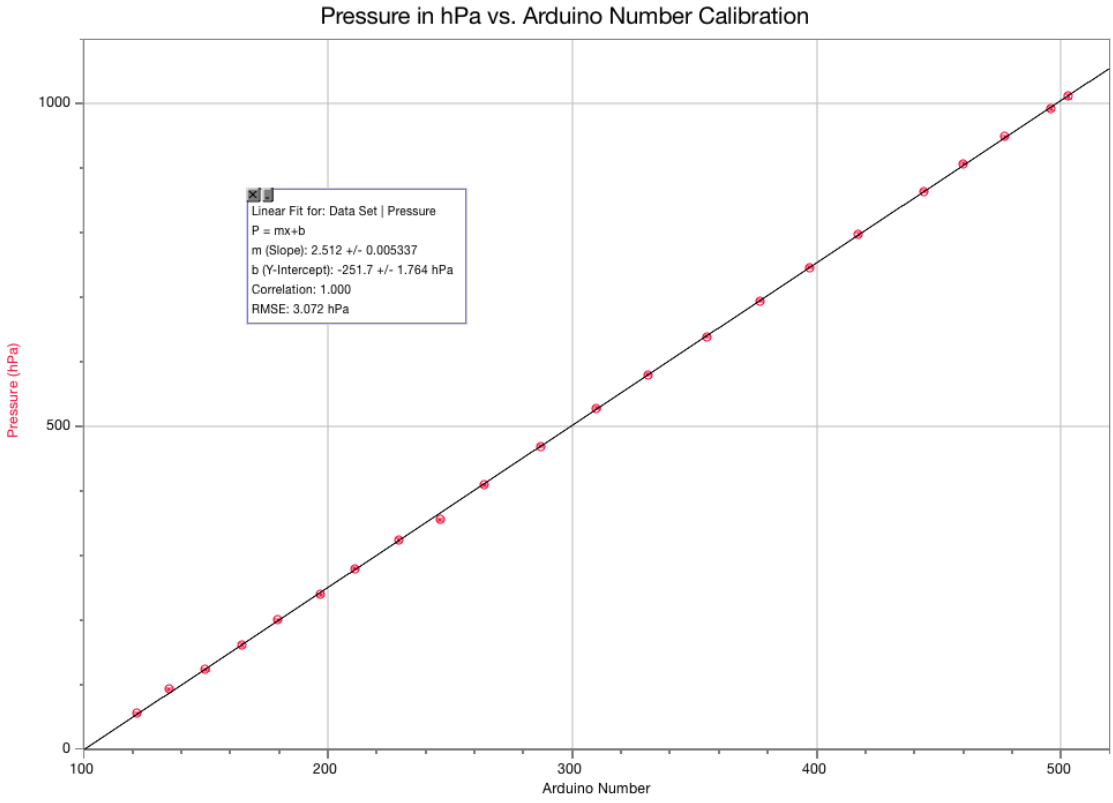


*Figure 29: Calibration of Vernier Gas Pressure Sensor. The Y-axis shows the pressure in hPa, given by the Extech Datalogger, while the X-axis displays the Arduino number given for that pressure. The best-fit line equation therefore gives the conversion from Arduino reading to pressure in hPa: Pressure = (2.512 ± 0.005337) * (Arduino number) + (-251.7 ± 1.764).*

It is important to note that pressure calibration was done at room temperature. Due to the relationship between pressure and temperature, pressure readings during the launch may have been influenced by colder temperatures at high altitude conditions. Because temperature is directly proportional to the kinetic energy of a substance, warm air moves faster and more freely than cold air. Cold temperatures would therefore lead to a pressure reading slightly less than would be accurate, as the molecules would not hit the pressure sensor as frequently since they move more slowly and are more compacted together (i.e. dense) because of their limited movement. However, the effect of temperature on pressure was not factored into the pressure data taken on the balloon launch due to the fact that the temperature calibration and data was much less accurate than the pressure data, and applying a temperature correction factor would decrease the accuracy of our pressure data rather than providing any benefit.

§VII.1.2 – Temperature

For temperature measurements, a Jameco thermistor was wired up in series with a 10k-ohm resistor and again connected to an Arduino. The calibration standard used was a Fieldpiece Dual Temperature Digital Thermometer ST4. To calibrate the Arduino readings, temperature data was taken with the thermistor and Fieldpiece Thermometer simultaneously at room temperature, inside a refrigerator, inside a freezer, and at multiple different temperature settings in a temperature chamber. The readings for the thermistor were then plotted against the readings from the Fieldpiece Thermometer (Figure X+1).

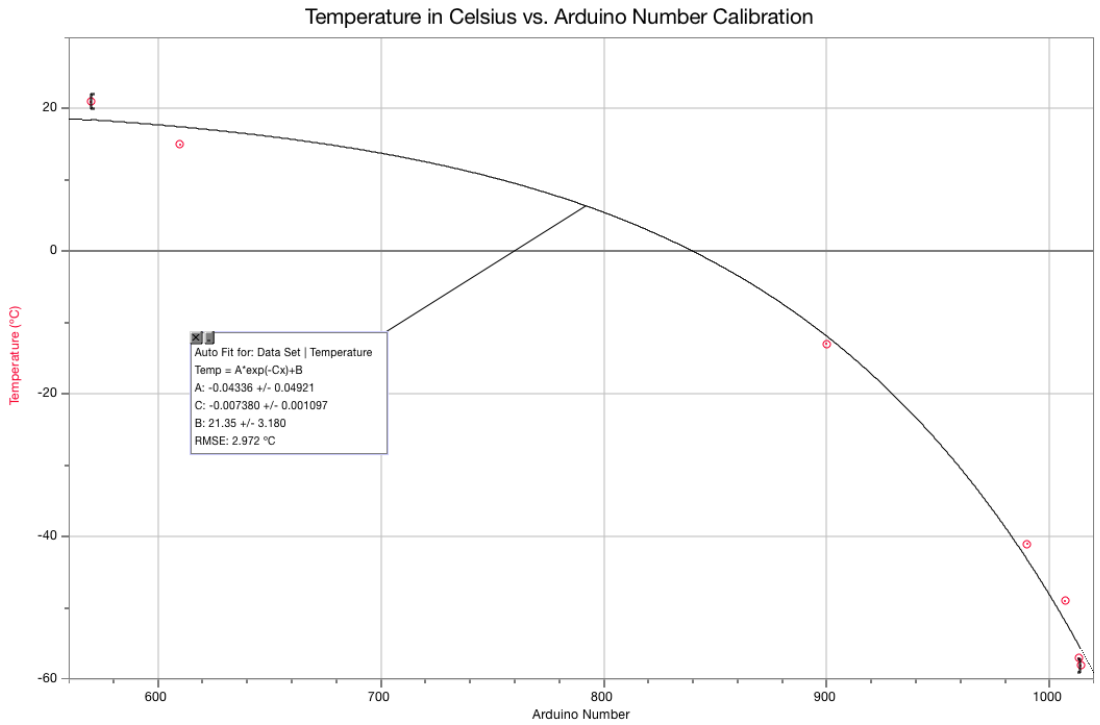


*Figure 30: Calibration of thermistor-Arduino setup. The Y-axis shows the temperature in Celsius, given by the Fieldpiece Thermometer, while the X-axis displays the Arduino number given for the same temperature. The curve-fit equation therefore gives the conversion from Arduino reading to temperature in Celsius: Temperature = (-.04336 ± 0.04921) * $e^{(-0.007380 \pm 0.001097)*(\text{Arduino number})}$ + (21.35 ± 3.180).*

§VII.1.3 – Humidity

Humidity data was taken by a Vernier Relative Humidity (RH) Sensor and calibrated with the Extech Datalogger as the standard. In order to calibrate the sensor, a method similar to the one used for the pressure sensor was employed: after hooking the humidity sensor up to the Arduino, it was placed atop a bathroom sink next to the Extech while the door to the bathroom was closed and hot water was run through the faucet. The faucet was left on for about twenty minutes, then turned off for a few minutes before the door was opened. Because the Arduino did not take data exactly every second and the Arduino and Extech Datalogger did not begin taking data at the exact same time, the point where both the Extech Datalogger and Arduino data began to drop (i.e. when the door was opened) was aligned in each data set, every five Arduino points averaged, and then the averaged Arduino numbers plotted against the Extech readings for relative humidity (Figure X+2).

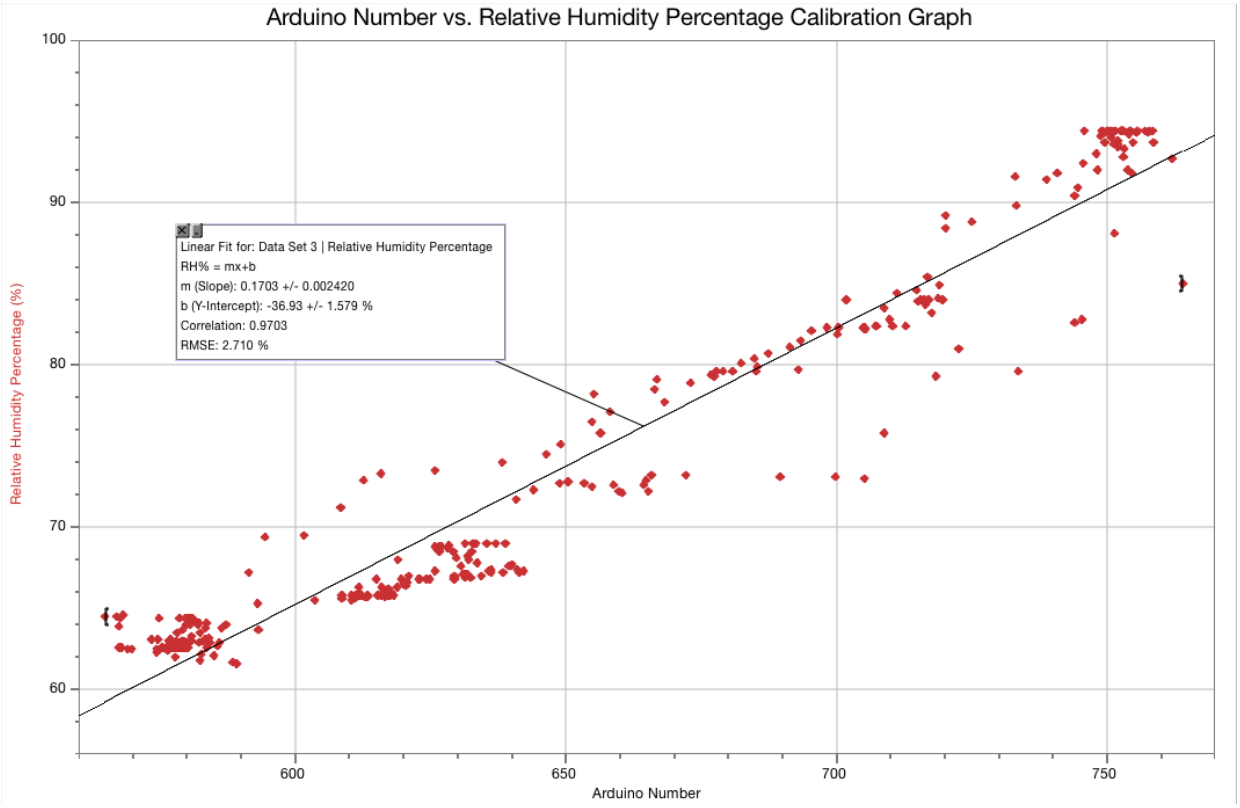


*Figure 31: Calibration of Vernier Relative Humidity Sensor. The Y-axis shows the relative humidity percentage, given by the Extech Datalogger, while the X-axis displays the Arduino number given for the same humidity. The best-fit line equation therefore gives the conversion from Arduino reading to relative humidity percentage: Relative Humidity = (0.1703 ± 0.002420) * (Arduino number) - (36.93 ± 1.579).*

§VII.1.4 – Ultraviolet (UV)

To take data on the intensity of ultraviolet light, a Sparkfun ML8511 UV Sensor was connected to an Arduino and calibrated with a Vernier UVA and UVB sensor, which output ultraviolet light intensity to a LabQuest 2. Both the Arduino and LabQuest 2 took data every second. The Sparkfun sensor and Vernier sensors were set in the same direction at the same distance from the light source in a darkroom, afterwhich a UV lamp producing UVA light was turned on towards the sensor. To vary intensity, the light source was distanced from the sensors and held at that point for several seconds, and then distanced more. The Arduino points were then plotted against the data given by the LabQuest 2 when it was connected to the Logger Pro software (Figure X+3).

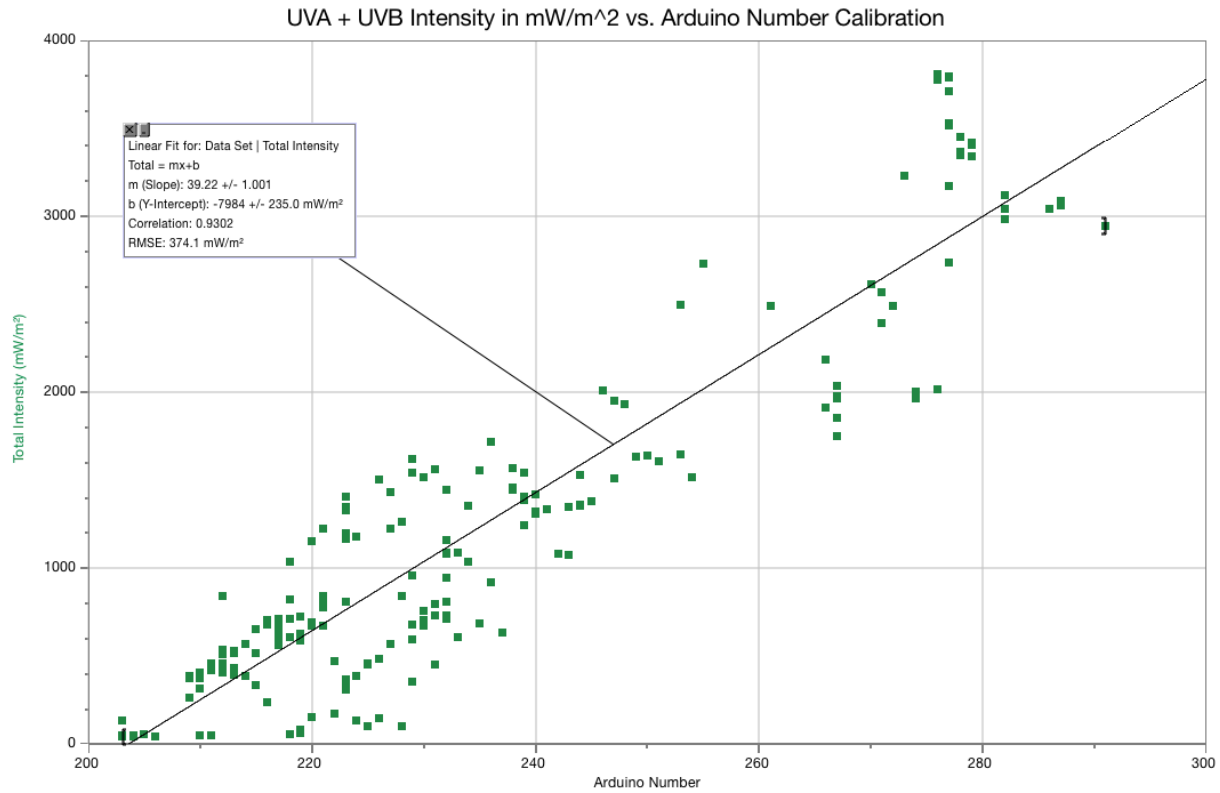


*Figure 32: Calibration of Sparkfun ML8511 UV Sensor. The Y-axis shows intensity of UV A + UV B light in mW/m^2, given by the LabQuest 2, while the X-axis displays the Arduino number given for the same intensity. The best-fit line equation therefore gives the conversion from Arduino reading to UV intensity: UVA + UVB Intensity = (39.22 ± 1.001) * (Arduino number) – (7984 ± 236.0).*

§VII.1.5 – Ambient Light

Originally, a Sparkfun TEMT6000 Ambient Light Sensor was going to be used to take ambient light data. However, due to time constraints, the sensor could not be calibrated before the final balloon launch, and upon attempting to calibrate the sensor with a Vernier Light Sensor as the standard, it was found to be impossible: the conversion from lux—the units that the Vernier Light Sensor used for measuring light—to miliwatts per meter squared—the units used in the remaining light data comprised of UV readings—was dependent on the wavelength of light used to calibrate the sensor, which was impossible to measure since the light source used (an electric bulb with a sliding dimmer switch) was comprised of several different wavelengths of light. In addition to the fact that this conversion might change when the light source was changed from the limited spectrum of an electric light to the spectrum wavelengths emitted by the actual sun (which would render any calibration useless), breaking this reading down into its individual wavelengths and converting the intensity of light was beyond the capabilities of the

person responsible for calibration, as well as the capabilities of the available instruments, and therefore the ambient light sensor was never officially calibrated, nor a specifications sheet for the Vernier Light Sensor or the Sparkfun sensor included in Appendix B. However, it was confirmed for the sake of analyzing the UV data that the Arduino number and intensity of light were directly proportional by shining an iPhone flashlight on the sensor and watching the Arduino number go up or down accordingly.

§VII.1.6 – Barometer as an Altimeter

To prove that the barometer could be used as an altimeter, the pressure sensor was brought to the top of Stent Hall to take data, and then set at the bottom to give a pressure difference that could calculate the height of Stent using the pressure-altitude equation (altitude = $(1 - (P_1/P_0)^{0.190284}) * 145366.45$). The actual height of Stent Hall was measured, using string, to be 27.6 feet tall. The pressure at the bottom of Stent Hall was measured to be 1016.86 hPa, while the pressure at the top was measured to be 1014.348 hPa. These values were then plugged into the equation, and a height of 68.40 feet was obtained. Although the percent error was greater than 50%, the values required to obtain a height of 27.6 feet were within the range of the calculated error for the pressure sensor (see *Barometer as an Altimeter* in the Error section, below). In addition, pressure reading only changed by one Arduino number, and the error can therefore be attributed to the low resolution of our pressure sensor (2.512 hPa per Arduino number, where all Arduino readings are integers). At higher altitudes, the readings would therefore become more accurate because the pressure differences would be much greater between sea-level and, say, 50,000 feet, than at 27.6 feet above the ground. In addition, the pressure-altitude relationship curve is steeper nearer to sea level than it is at the example of 50,000 feet, which means that a slight error in the pressure reading would result in a much lower accuracy at a lower altitude than at a higher one.

§VII.2 – Error

§VII.2.1 – Pressure

In order to calculate error bars for the pressure sensor, three different sources of technical error had to be added together for the total range of the error bar. The first was the error from the Extech Datalogger used as the calibration standard. On the data sheet given by Extech, values from 10.0 to 1000.0 hPa were declared to have an accuracy of ± 2 hPa, and values from 1000.1 hPa and above to have an accuracy of ± 3 hPa. The second component of error was given by the

Vernier Gas Pressure Sensor used to take the data, which, on the Vernier data sheet, was said to be accurate to ± 0.25% of the pressure value given. The final component was the percent error given by the slope of our calibration graph, which could be calculated through dividing the error on the slope by the best-fit slope—in the case of pressure, ± 0.005337 / 2.512 = .21%. To add this to the error bar, each data point would be multiplied by this percentage, as was the percentage of error given by the Vernier data sheet.

If *P* represents the data point taken by the pressure sensor, the final error bar calculation used was as follows:

$$\text{If } 10.0 \text{ hPa} \leq P \leq 1000.0 \text{ hPa, error} = \pm (2 + 0.0025*P + 0.0021*P)$$
$$\text{If } P \geq 1000.1 \text{ hPa, error} = \pm (3 + 0.0025*P + 0.0021*P)$$

§VII.2.2 – Temperature

To calculate error bars for the temperature sensor, a similar method was used. The error given in the Fieldpiece data sheet was ± 0.3% rdg (i.e. 0.3% of the given reading) + 1ºF (5/9ºC) from -50ºC to 600ºC, and the error of the sensor itself was not given on the data sheet provided by Jameco. The calculations for the error from the calibration graph, however, was far more complicated than the others due to the fact that the relationship between Arduino number and temperature was not linear. For each point, because the error bars were not symmetrical, the most reasonable error from the data point on the curve was calculated and added to the error from the standard. Errors on any of the constants in the calibration were not taken into account, as only the error in the exponent made a significant difference. Thus, the error for a given temperature point (represented by *T*) uses the following formula:

$$\text{error} = \pm (0.0003 * T + (5/9) + [(-.04336) * e^{(-0.007380 + 0.001097)*(\text{raw Arduino number})} + (21.35) - T]$$

It is important to note that because the error bars were so large after adding the error from the calibration graph, the calibration error factor was not incorporated into the analyzed graphs because it made the error bars so large that they obscured the viewing of the data. Instead, the graphs containing the full error bars will be put in Appendix C, and to account for calibration error, the root-mean-square error (2.972ºC) of the calibration curve was added to the error in place of the expression inside the rectangular brackets, resulting in the following error bar:

$$\text{error} = \pm (0.0003 * T + (5/9) + 2.972)$$

§VII.2.3 – Humidity

For the humidity sensor, the error bars again comprised of the error from the calibration standard, the error from the sensor itself, and the percent error from the slope on the calibration graph. The error given for the calibration standard on the Extech data sheet was ± 4% rdg + 1 %RH (note: %RH is a unit and 1 %RH is therefore a constant error, not a percentage), for the sensor on the Vernier data sheet, ± 10 %RH, and from the slope error, ± 1.4% of the given data point. Thus, if *RH%* represents the corresponding data point recorded by the humidity sensor, the final error bar calculation is as shown:

$$\text{error} = \pm (0.04*RH\% + 1 + 10 + 0.014*RH\%)$$

§VII.2.4 – Ultraviolet

Due to the fact that the Vernier website did not contain information on the data sheets for the accuracy of its UVA and UVB sensors, which were used as a calibration standard, the error from calibration could not be included in the error bars for UV data. However, the percentage error from the slope was included for each non-calibration graph containing UV data, and was calculated solely by using the slope from the calibration graph, yielding the following equation: error = ± (0.026*UV).

§VII.2.5 – Ambient Light

Because the ambient light sensor was never properly calibrated, there are no error bars on the ambient light data. Instead, the ambient light graphs were used simply to tell whether or not the light sensors were being covered by the balloon at the time of the measurement (i.e. if covered, the ambient light Arduino reading would drop). As this was a observation of the relationship between ambient light data points, error bars did not need to be calculated.

§VII.2.6 – Barometer as an Altimeter

In using the barometer as an altimeter that gave the x-axis altitude in other graphs, horizontal error bars had to be factored into the altitude readings to represent the uncertainty in reading how high the payload actually was at the time the data was taken. In order to calculate this error, the altitude was calculated twice: once with the given pressure datum, and once with the pressure error (including the RMS error of 10.266, which was found by taking the square root of the averages of the squared error as the pressure reading fluctuated at a constant altitude over ten minutes) factored into the calculation. The difference was then taken and used as the value

for the error bar. Thus, if *P* again represents the data point recorded for pressure, the final error bar calculation for altitude is as shown below:

$$\text{If } 10.0 \text{ hPa} \leq P \leq 1000.0 \text{ hPa, error} = \pm ((1 - (((P - (2 + 0.0025*P + 0.0021*P)) - 10.266)/1013.25)^{0.190284})*145366.45) - (1 - (P/1013.25)^{0.190284})*145366.45)$$

$$\text{If } P \geq 1000.1 \text{ hPa, error} = \pm ((1 - (((P - (3 + 0.0025*P + 0.0021*P)) - 10.266)/1013.25)^{0.190284})*145366.45) - (1 - (P/1013.25)^{0.190284})*145366.45)$$

VIII. Results

§VIII.1 – Balloon Path

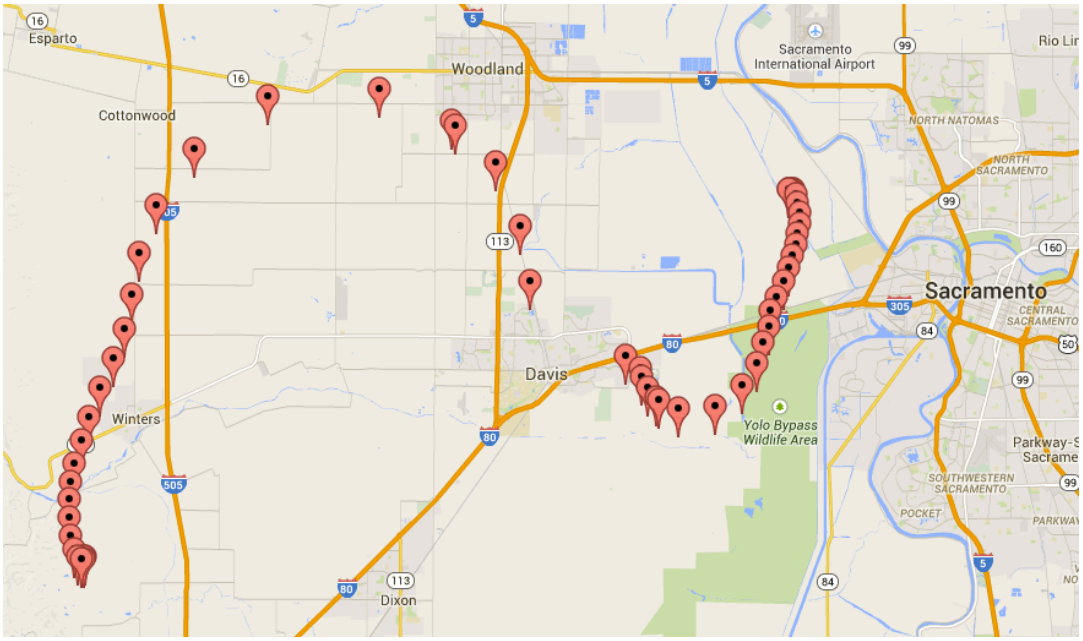


Figure 33: Expected path of balloon according to balloon simulation ran before launch.

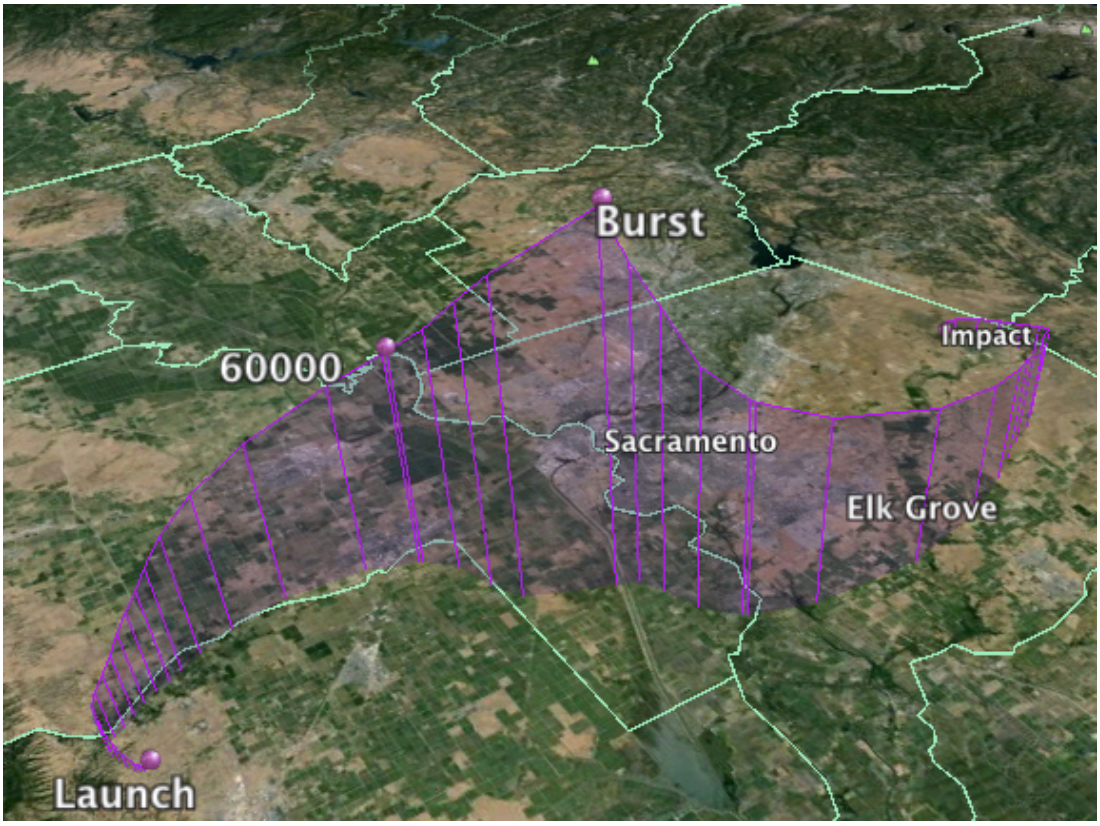


Figure 34: Expected path of balloon, with altitudes. Map made on Google Earth.

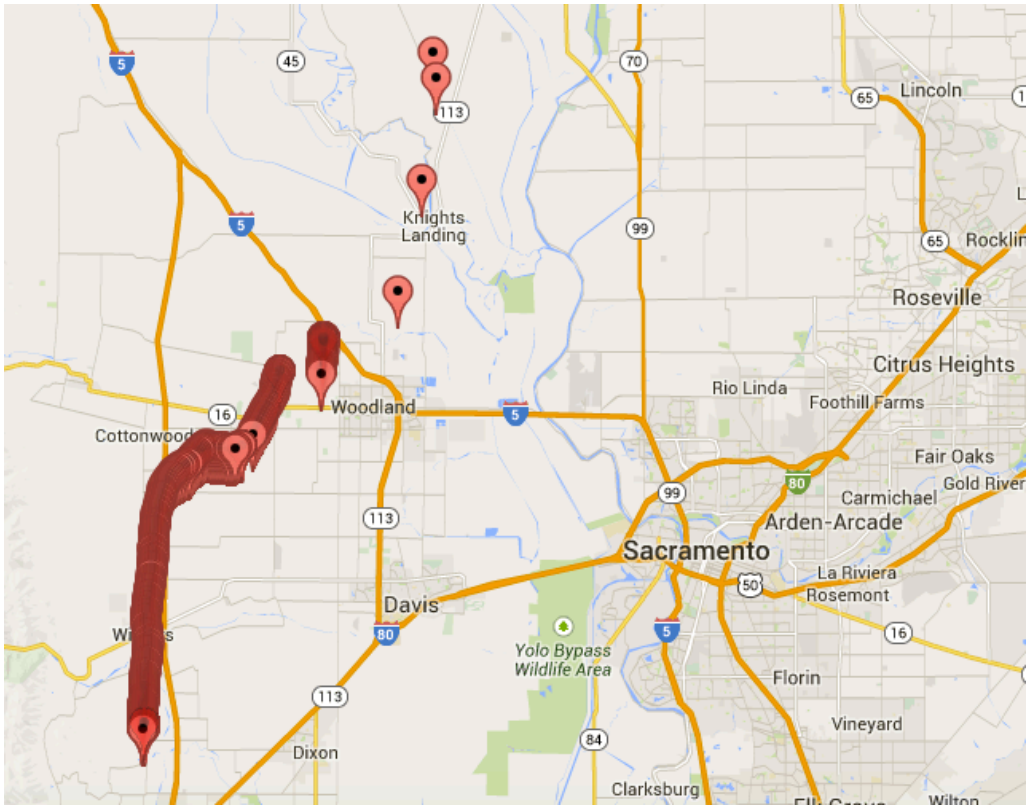


Figure 35: Actual path of balloon. All data was collected by the radio transmitter, except for the last 4 locations, which were collected by the Spot Tracker, due to the radio cutting out early.

§VIII.2 – Experimental Results

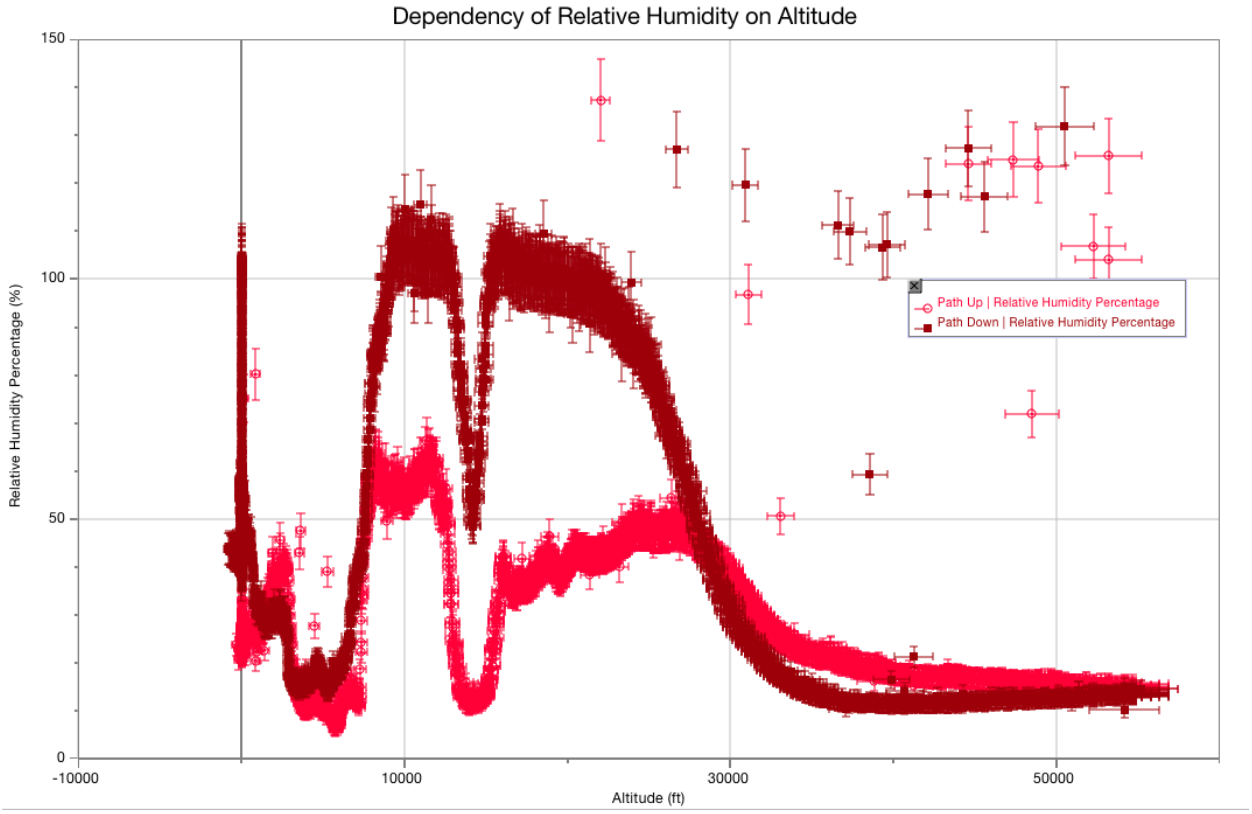


Figure 36: Illustrates humidity with respect to the balloon's altitude. Note the spike in humidity at approximately 10,000 ft, likely due to the moisture in clouds. This also gives significance to the drops between the two major peaks on the graphs, as they must represent altitudes where there are gaps between clouds. The farther up the balloon went, the less moisture there was in the atmosphere (as expected). This graph has two sets of data that form two distinctive curves. The significantly lower humidity trend is from the balloon's trip into space while the higher humidity trend was from the data recorded on the fall back to Earth. While we have no definitive explanation for the difference, it is possible that as the payload fell, the air was moving more rapidly, hitting the humidity sensor faster and subsequently quickening its response time (as illustrated in the Vernier data sheet, which states a response time of 60 minutes in still air and a response time of 40 seconds in vigorously moving air, it is highly probable that the rapid motion of the sensor contributed significantly to the comparatively large spikes in humidity on the way down).

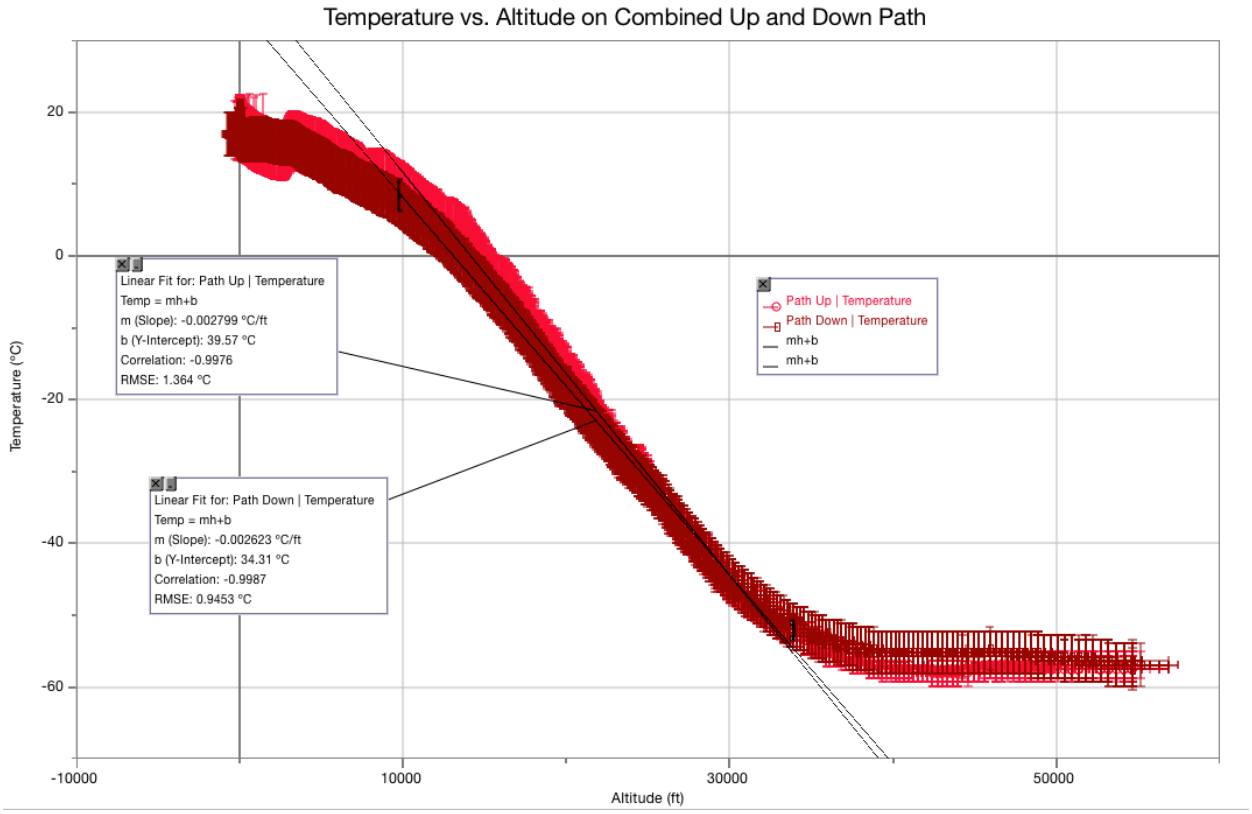


Figure 37: Illustrates the change in temperature with an increase in altitude. The temperature increases approximately linearly in relation to altitude until about 40,000 feet, where the temperature levels off at -58 degrees Celsius. This can be attributed to either the Arduino number reaching its maximum (1023), the curved nature of the calibration between Arduino number and temperature, and/or the actual slowing of temperature change at this altitude.

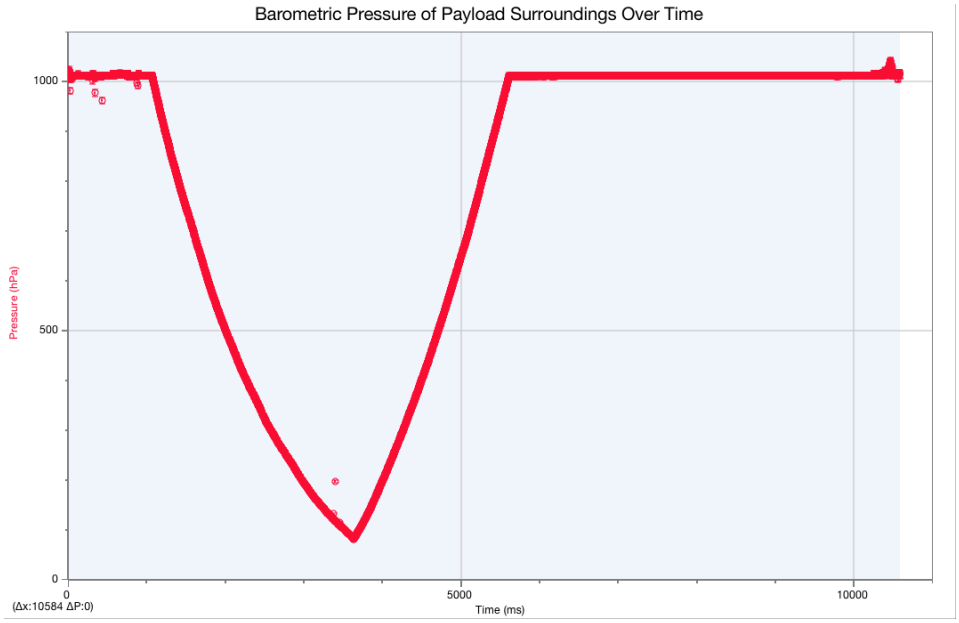


Figure 38: Illustrates the change in pressure with respect to time. From this figure, we can extrapolate the balloon reached its maximum altitude approximately an hour after the battery was attached to the data acquisitioning system's Arduino. The payload reached a minimum pressure of approximately 80 hPa.

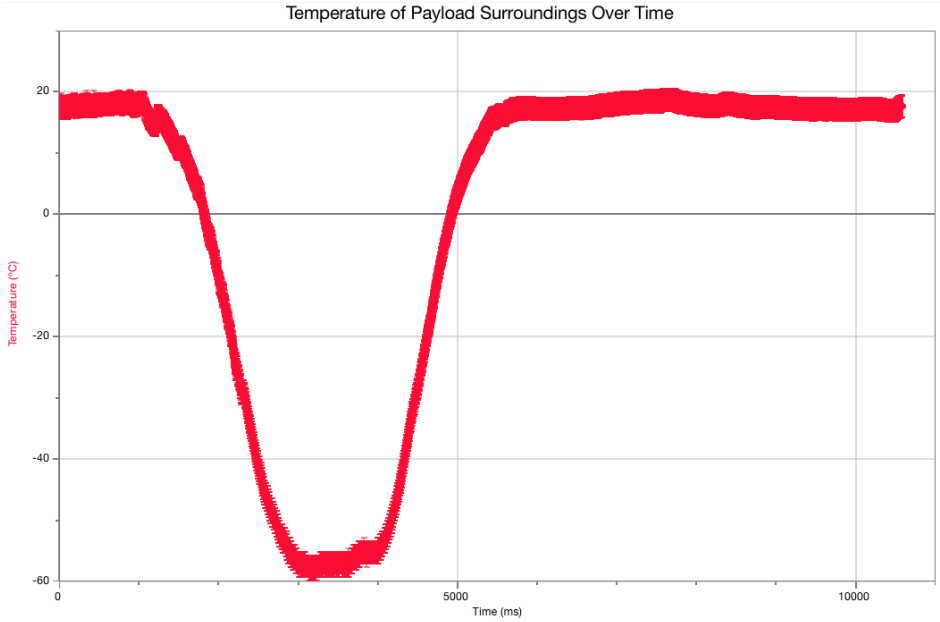


Figure 39: Illustrates the change in temperature with respect to time. Given that temperature and altitude are inversely related, we can assume the balloon reached its maximum height at the lowest temperature (approximately an hour after the battery was attached to the data acquisitioning system's Arduino).

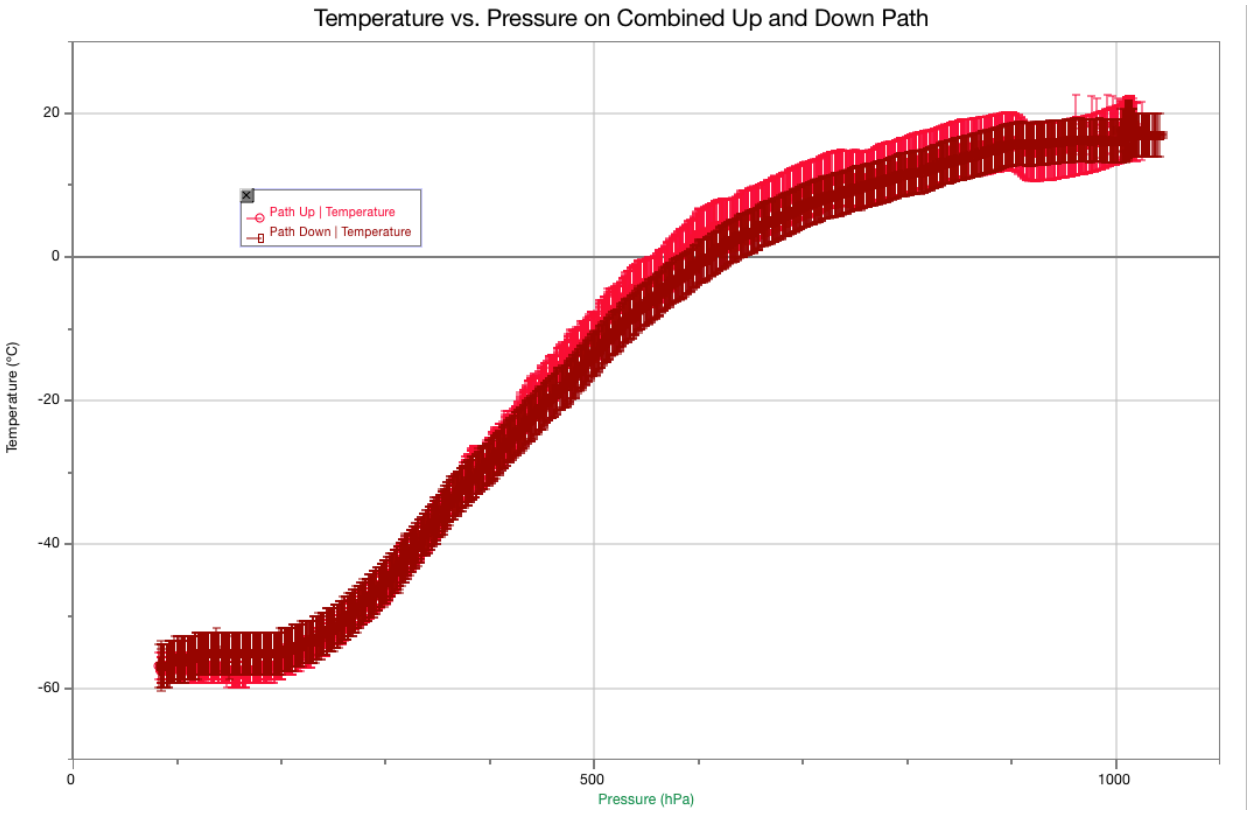


Figure 40: Illustrates the almost linear relationship between temperature and pressure based on the data collected by the payload. As the temperature decreases with altitude, which has an indirectly proportional relationship with pressure, this trend is to be expected given the observations taken from Figure (whatever figure the Temperature vs. Altitude graph is).

Figures 41 and 42: The two graphs above display both the data taken by the UV sensor as well as the ambient light sensor. As can be seen when comparing the two graphs (note that they are

on different scales), the ambient light sensor appears to register less light whenever the UV sensor is sensing more UV light. Without this observation, it appears that there is a downward trend in UV exposure as the balloon travels upwards, but with the comparison, it begins to seem that the data is not logical—if the ambient light sensor reads less light, the sensors must have been covered by the balloon—in which case the UV sensor could not possibly have registered a higher intensity of UV light.

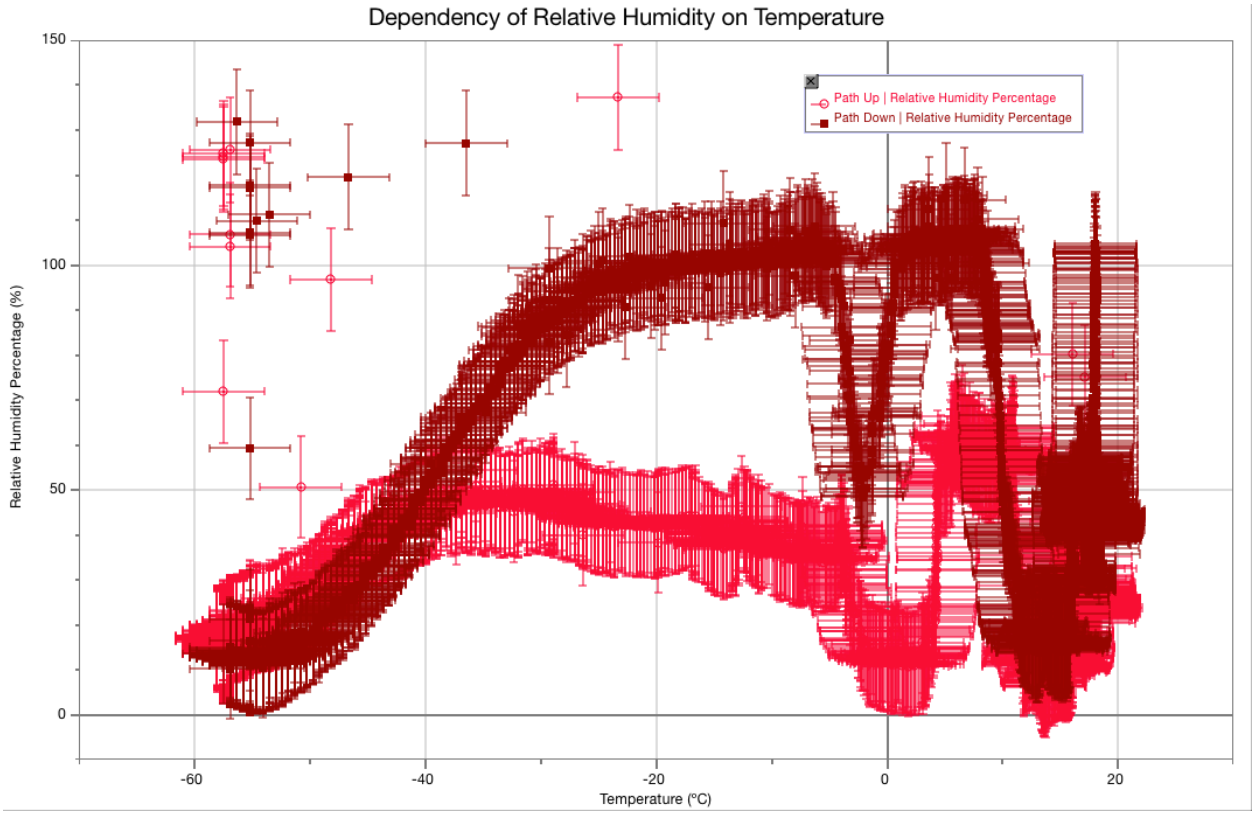


Figure 43: The graph above depicts the relationship between humidity and temperature on the up and down paths of the balloon. Due to temperature's high, negative correlation with altitude, it can be observed that the humidity vs. temperature graph is a somewhat mirrored version of Figure (whatever figure humidity vs. altitude is). However, it is important to note that this effect can also be partially attributed to temperature itself rather than just altitude—as the temperature gets colder, the majority of water in the atmosphere is also frozen, thus resulting in a lower level of humidity in the air.

IX. Conclusion

Overall, our space launch experience was for the most part a success. We were able to take relatively meaningful and complete data, our cut-down mechanism worked as expected, and we were ultimately able to locate the payload and retrieve our data after the free launch. In the future, we would extend the time before the cut-down as our balloon was released somewhat early - it likely could have gone much higher. In addition, it would have been very interesting to have a working GoPro in our payload so that we could view earth from space. Other sensors that would be interesting to integrate in future balloon projects include a Geiger counter, as well as an accelerometer to track the movements of the payload as it travels into the edge of space.

Over the course of our launch, we were able to take 10,585 data points, all of which contributed to a better understanding of either space or the scientific process. While some data-taking attempts like the UV and ambient light data fell short and did not provide too much insight into the data we were hoping to attain, they contributed to our strength as researchers by teaching us to recognize inconsistent data and identify a corresponding error. For the most part, the other data we wanted to take—pressure, temperature, humidity, and GPS—confirmed prior information or provided new insight into high-altitude conditions. As an example, our relative humidity with respect to altitude graph provided a general picture of the altitudes at which clouds could and couldn't be found. Both pressure and temperature conformed to accepted trends already established by the scientific community, but allowed us to confirm this for ourselves via hands-on experimenting. Finally, the GPS data we were able to map and compare to the simulated path provided a firsthand account of the unpredictability of wind patterns. All in all, the data we took resulted in findings that allowed us to look at accepted trends with methods we developed and planned ourselves, or that opened doors to new insight or facets of high-altitude conditions that we had not previously considered.

As a team, we learned to work together throughout the course of this project. Having each person responsible for certain aspects of the project and integrating it into one final product made the experience very applicable to real life.

X. Bibliography

[qq1] “How Does GPS Work?” Physics.org http://www.physics.org/article-questions.asp?id=55

[qq2] “How Radio Works” howstuffworks.com http://electronics.howstuffworks.com/radio.htm

[qq3] “Radio” explainthatstuff.com http://www.explainthatstuff.com/radio.html

[qq4] “GPS – NMEA sentence information” gids.nl http://aprs.gids.nl/nmea/

[s1]"Jameco Electronics." NTC-103-R: JAMECO VALUEPRO: ICs & Semiconductors. N.p., n.d. Web. 23 Nov. 2014.

[s2]"SparkFun UV Sensor Breakout - ML8511." - SEN-12705. N.p., n.d. Web. 24 Nov. 2014

[s3]"1960 - Epitaxial Deposition Process Enhances Transistor Performance - The Silicon Engine." Computer History Museum. N.p., n.d. Web. 25 Nov. 2014.

[s4]"Humidity Sensors." Innovative Sensor Technology. N.p., n.d. Web. 25 Nov. 2014.

[s5]Vernier. "Gas Pressure Sensor." (n.d.): n. pag. 16 Feb. 10. Web. 23 Nov. 2014.

[s6] Benson, Tom. "Earth Atmosphere Model - Metric Units." Nasa.gov. N.p., 12 June 2014. Web. 23 Nov. 2014. <http://www.grc.nasa.gov/WWW/k-12/airplane/atmosmet.html>.

[b1] Streete, John. "The Earth's Atmosphere." *The Sun-earth System*. Sausalito: University Science Bks., 1996.

[Layers of atmosphere] *National Weather Service*. National Weather Service, 22 July 2013. Web. 24 Nov. 2014. http://www.srh.noaa.gov/jetstream/atmos/layers.

[Auroras] *Northern Lights*. Northern Lights Centre, n.d. Web. 24 Nov. 2014. http://www.northernlightscentre.ca/northernlights.

[GPS syntax] *GPS-NMEA Sentence Information*. Glenn Baddeley, 20 July 2001. Web. 24 Nov. 2014. http://aprs.gids.nl/nmea/.

[Atmospheric Pressure] *Kids.Earth.NASA*. Nasa Official, 22 Jan. 2003. Web. 24 Nov. 2014. http://kids.earth.nasa.gov/archive/air_pressure/.

XI. Acknowledgements

Thank you to Dr. Dann for giving us help in our project.

Thank you to Ari Holtzman for his NTX2/NRX2 Documentation Paper.

Thank you Jessica Fry for having the movie “Easy A” so that we could be entertained on the long bus ride

Appendix A

Appendix A.1: Launch Protocol

1. **Arduino for Sensors**
    1. ____ Ensure SD card is set to 4800 baud
    2. ____ Insert SD card into the SD card writer
    3. ____ Connect arduino to batteries
    4. ____ Check that it is wired correctly
    5. ____ Turn it on and ensure data is being collected
    6. ____ Ensure it is secure and taped down
2. **Arduino for Radio**
    1. ____ Connect arduino to batteries
    2. ____ Check that it is wired correctly
    3. ____ Turn it on and ensure data is being collected
    4. ____ Ensure it is secure and taped down
3. **Tracking Systems**
    1. ____ Turn on spot tracker, and ensure it is working
    2. ____ Place spot tracker in box and tape it down
    3. ____ Ensure find my iPhone is on phone
    4. ____ Place iPhone in box and tape it down
4. **GoPro**
    1. ____ Turn on GoPro, ensure it is working
    2. ____ Place in box and tape
5. **Cut-Down Mechanism**

1. ____ Connect arduino to batteries
2. ____ Check that it is wired correctly
3. ____ Ensure it is secure and taped down
4. ____ Record time battery connected to arduino

6. **Payload**
    1. ____ Close box and place carabiner through hasp
    2. ____ Attach strings on corners to carabiner
    3. ____ Shake around to ensure it won't break and that items are stable
7. **Final**
    1. ____ Connect payload to balloon
    2. ____ Send balloon up!

Appendix A.2: Pre-Launch Form

[Pre-Launch Form is attached on next pages]

[Filler Page for Pre-Launch Form]

[Filler Page for Pre-Launch Form]

[Filler Page for Pre-Launch Form]

[Filler Page for Pre-Launch Form]

Appendix B: Sensor Technical Specification Tables

Note: Specifications marked with an asterisk (*) were taken from the corresponding data sheet. Calibration standard specifications will not be marked, as all specifications were taken from the corresponding data sheet. Resolutions of individual sensors were not used in data analysis, and therefore are not included in sensor specification sheets.

Calibration Standard	Extech Barometric Pressure/Humidity and Temperature Datalogger (Model SD2700)
Relative Humidity Range	10 to 90%
Relative Humidity Resolution	0.1%
Relative Humidity Accuracy	± (4% of rdg + 1% RH) @ 70 to 90% ± (4% RH) @ 10 to 70%
Barometric Pressure Range	10.0 hPa to 1100.0 hPa
Barometric Pressure Resolution	0.1 hPa
Barometric Pressure Accuracy	± 2 hPa @ 10.0 to 1000.0 hPa ± 3 hPa @ 1000.1 to 1100.0 hPa

Calibration Standard	Fieldpiece Dual Temperature Digital Thermometer (Model ST4)
Temperature Range	-50 to 1300ºC (-58 to 2000ºF)
Temperature Resolution	0.1º
Temperature Accuracy	± (0.3% rdg + 1ºF) @ -50 to 600ºC ± (0.5% rdg + 1ºF) @ 600 to 1300ºC

Calibration Standard	Vernier UVA Sensor with LabQuest 2
Approximate Wavelength Range	320 to 390 nm
12-bit Resolution (LabQuest 2)	5 mW/m^2
Response Time	2 seconds to 95% of final reading
Intensity Accuracy	Not provided

Calibration Standard	Vernier UVB Sensor with LabQuest 2
Approximate Wavelength Range	320 to 390 nm
12-bit Resolution (LabQuest 2)	0.25 mW/m^2
Response Time	2 seconds to 95% of final reading
Intensity Accuracy	Not provided

Sensor	Jameco LM-103-R NTC Thermistor
Temperature Range*	-55 to 150ºC
Temperature Accuracy	± (0.3% rdg + 1ºF + 2.972ºC)**
Response Time*	20 seconds
Calibration Function	Temperature = (-.04336 ± 0.04921) * $e^{(-0.007380 \pm 0.001097)*(\text{Arduino number})}$ + (21.35 ± 3.180)

**Refer to *Temperature Sensor* portion of Error section for more details

Sensor	Jameco LM-103-R NTC Thermistor
Temperature Range*	-55 to 150ºC
Temperature Accuracy	Varies per point
Response Time*	20 seconds
Calibration Function	Temperature = (-.04336 ± 0.04921) * $e^{(-0.007380 \pm 0.001097)*(\text{Arduino number})}$ + (21.35 ± 3.180)

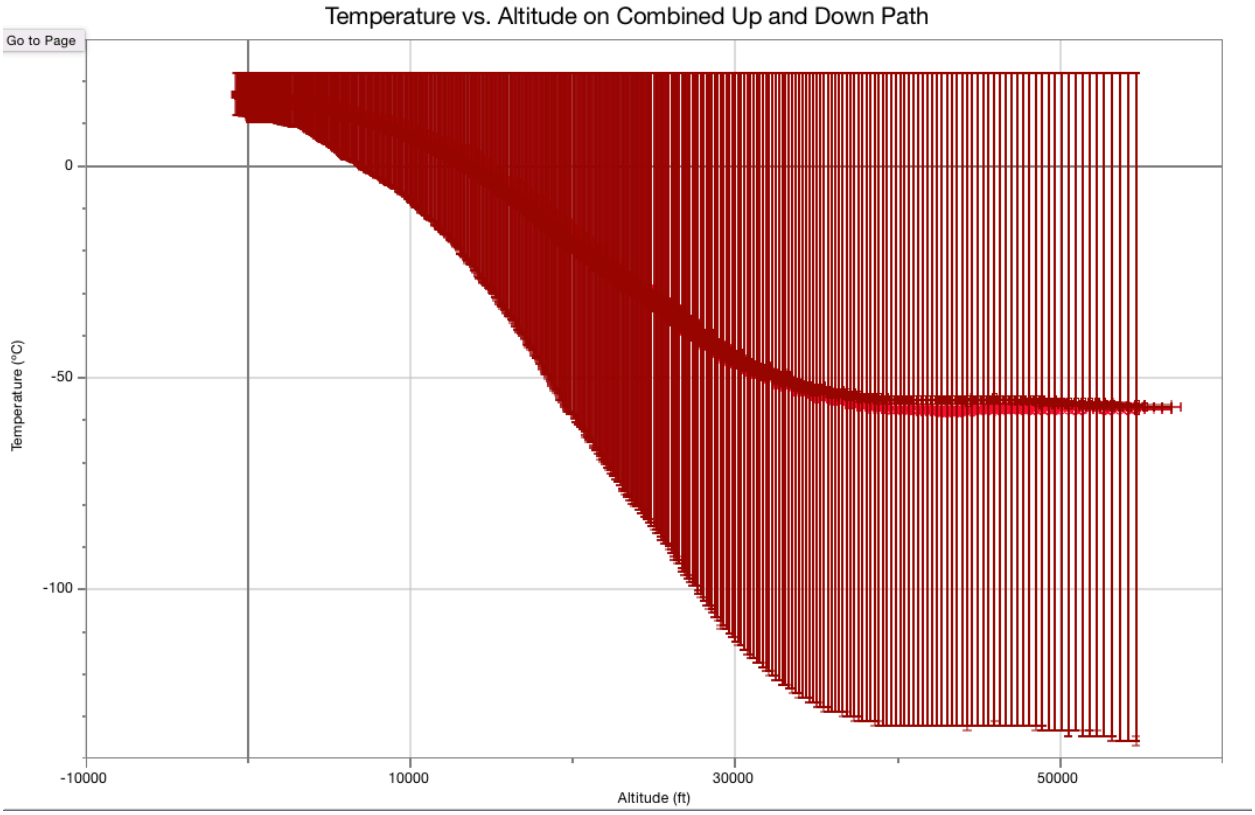
Sensor	Vernier Relative Humidity Sensor
Relative Humidity Range*	0 to 95%
Relative Humidity Accuracy	± (5.4% rdg + 11% RH)
Response Time*	40 seconds (with vigorous air movement)

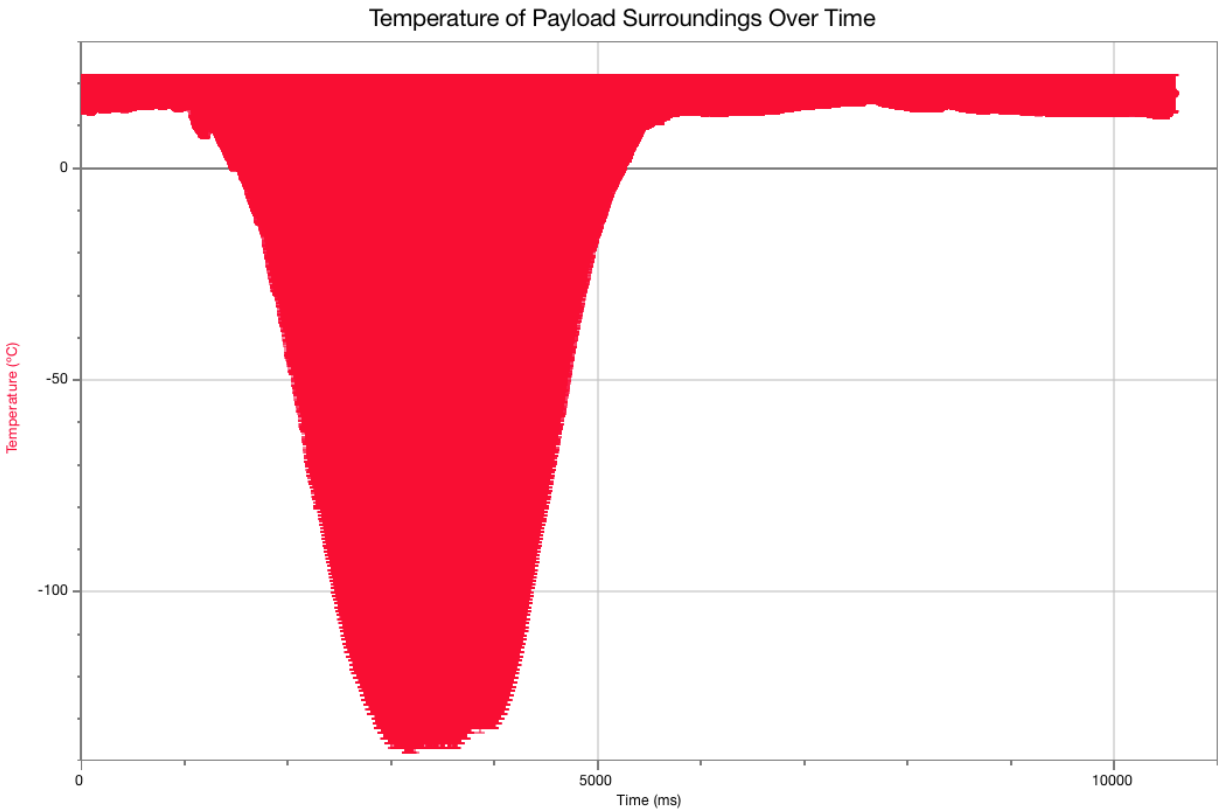
Calibration Function	Relative Humidity = (0.1703 ± 0.002420) * (Arduino number) – (36.93 ± 1.579)

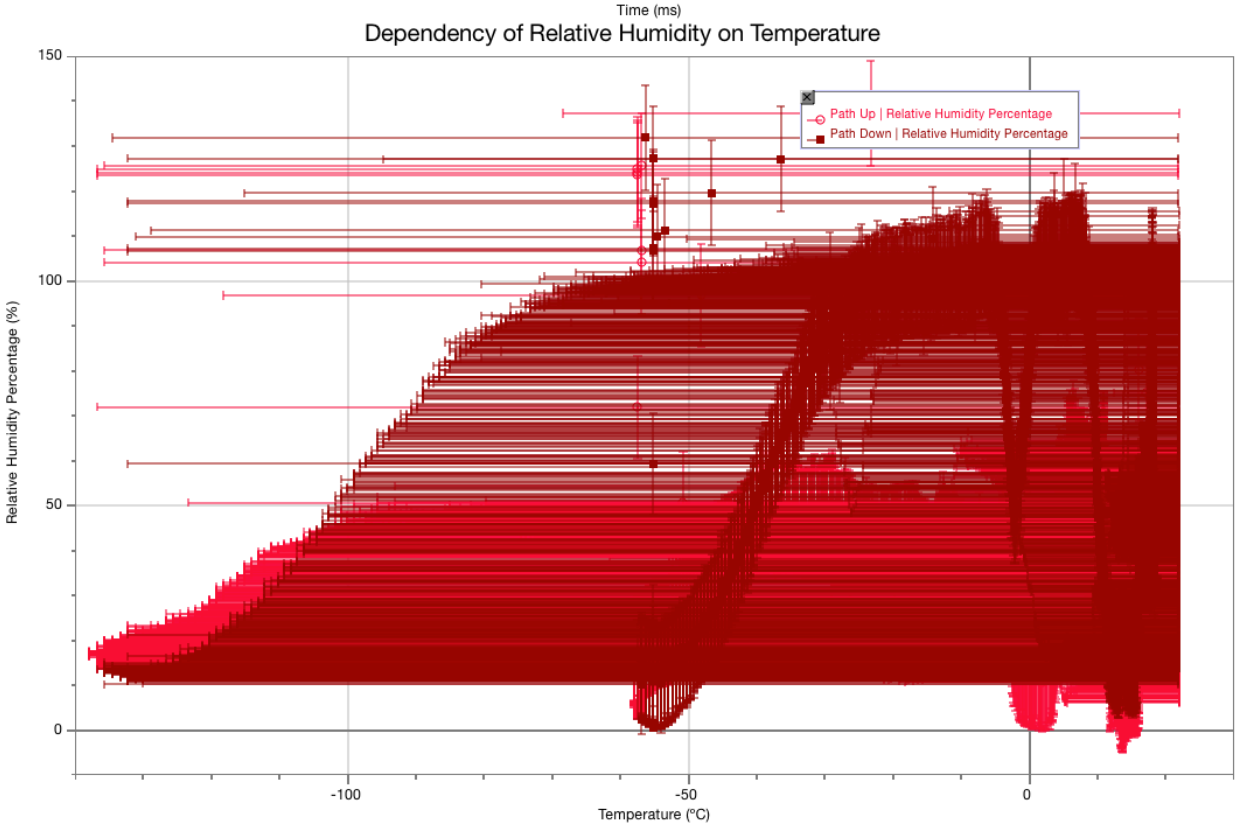
Sensor	Sparkfun ML8511 Breakout UV Sensor
Approximate Wavelength Range*	280 to 390 nm
Intensity Accuracy	± 2.6% rdg
Calibration Function	UVA + UVB Intensity = (39.22 ± 1.001) * (Arduino number) – (7984 ± 236.0)

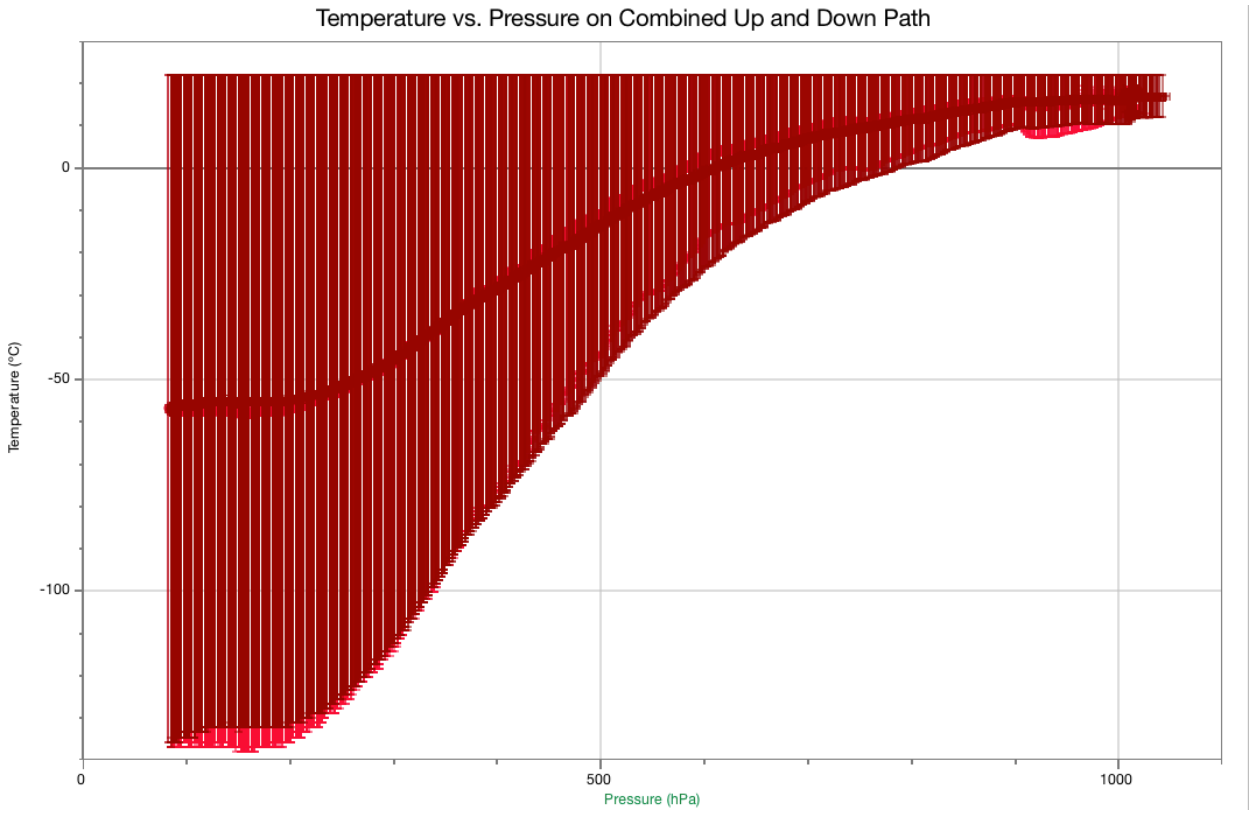
Appendix C: Temperature-Related Graphs With Full Calibration Error

Below are the graphs mentioned in the Temperature Sensor portion of the Error section. As mentioned, the full calibration curve component of the error bars were not included in the graphs used to analyze data due to the fact that the error was exponential and therefore had a large effect on the data that a) obscured the data to the point where it was unreadable and b) was obviously highly nonsensical and is purely included for the sake of scientific integrity.




Temperature of Payload Surroundings Over Time
Temperature (°C)
0
-50
-100
0
5000
10000
Time (ms)


Dependency of Relative Humidity on Temperature
Path Up | Relative Humidity Percentage
Path Down | Relative Humidity Percentage
Relative Humidity Percentage (%)
150
100
50
0
-100
-50
0
Temperature (°C)


Temperature vs. Pressure on Combined Up and Down Path
0
-50
-100
Temperature (°C)
0
500
1000
Pressure (hPa)

Appendix D: Arduino Code

Appendix D.1: Data Acquisition

```
/*
Data Acquisition Code for the Balloon Launch (note: this code does not include uv light,
ambient light, or humidity sensor calibrations because they were not known at the time of
launch)
Megs Malpani
11/15/14
*/

void setup()
{
 Serial.begin (4800);
 Serial.println("temp raw,temp calibrated,uv light raw,ambient light raw,pressure
raw,pressure calibrated,humidity raw");
}

void loop()
{
 double tempRaw = analogRead(A0);
 double uvLightRaw = analogRead(A1);
 double ambientLightRaw = analogRead(A2);
 double pressureRaw = analogRead(A3);
 double humidityRaw = analogRead(A4);

 //to calculate the calibrated temp
 double tempExponent = pow(2.71828, 0.00738*tempRaw);
 double tempCalibrated = (-.04336*tempExponent)+21.35;
 //to calculate the calibrated pressure
 double pressureCalibrated = (2.512*pressureRaw) - 251.7;

 //printing the data in a .csv file format
 Serial.print(tempRaw);
 Serial.print(",");
 Serial.print(tempCalibrated);
 Serial.print(",");
 Serial.print(uvLightRaw);
 Serial.print(",");
 Serial.print(ambientLightRaw);
 Serial.print(",");
 Serial.print(pressureRaw);
 Serial.print(",");
 Serial.print(pressureCalibrated);
 Serial.print(",");
 Serial.println(humidityRaw);

 //to record data every second
 delay (1000);
}
```

Appendix D.2: Transmitter Code

```
/*
The following code goes on the arduino inside of the payload. It gets data from the GPS chip
(which gets data from the GPS antenna), changes the format of the data, and then transmits
it through an antenna.
Nicholas Seidl
11/25/14
*/
       #include <SoftwareSerial.h>//import the arduino serial library
       String STARTER = "Tem1";//starter and ender sequences for identification
       String ENDER = "Tem2";

       SoftwareSerial gpsSerial(10, 11); // RX, TX (TX not used)
       const int sentenceSize = 80;//the maximum length of the GPS input string
       char sentence[sentenceSize];//an array of characters for the gps string
       String sendString = "";

       void setup()
       {
        gpsSerial.begin(9600);  // baud rate of the GPS chip
        Serial.begin(4800); //baud rate of the transmitter chip
       }

       void loop() {
        static int i = 0;
        if (gpsSerial.available())  {//if there is data available
          char ch = gpsSerial.read();//then read it in one character at at a time
          if (ch != '\n' && i < sentenceSize)  {//if the character isn't the last one and we
aren't at the end
            sentence[i] = ch;//add it to the gps sentence and increment character on by one
            i++;
          }
          else {//if there is no more gps data to read in
            sentence[i] = '\0';//put a zero at the end
            i = 0;
            displayGPS1();//change the gps input string into a readable format
            if(sendString != "")  {//if sendString isn't empty
              digitalWrite(3,HIGH);//turn on transmitter chip
              delay(50);  //wait for it to turn on
              sendString = STARTER + sendString + ENDER;//compile the final send string
              Serial.println(sendString); //send the string
              delay(300); //wait for the whole string to be transmitted
              sendString = "";//reset the send string
              digitalWrite(3,LOW); //turn off the transmitter
              delay(20); //wait to conserve battery
            }
          }
        }
       }
```

Appendix D.3: GPS Receiver Code

```
/*
The following code is part of the code that was loaded onto the transmitter arduino.
displayGPS1() takes the GPS data string, takes certain parts of it, and puts those parts
into the string that will end up bring printed or transmitted. getField() parses the whole
raw GPS string, each time shortening it.
Nicholas Seidl
11/25/14
*/
        void displayGPS1()
        {
         char field[20];//create a new array of charaters to store the sentence in
         getField(field, 0);//grab information from the gps input
         if (strcmp(field, "$GPRMC") == 0) {//if the gps string is in GPRMC format
           sendString = "Lat: ";
           getField(field, 3);  // number
           sendString = sendString + field;

           getField(field, 4); // N/S
           sendString = sendString + field;

           sendString = sendString + " Long: ";
           getField(field, 5);  // number
           sendString = sendString + field;

           getField(field, 6);  // E/W
           sendString = sendString + field;
         }
        }

        void getField(char* buffer, int index){
         int sentencePos = 0;
         int fieldPos = 0;
         int commaCount = 0;
         while (sentencePos < sentenceSize) {//if we aren't at the end of the sentence
           if (sentence[sentencePos] == ',')  {//if the character we're on is a comma
             commaCount ++;//increase commacount and sentence pos by one
             sentencePos ++;
           }
           if (commaCount == index) {//if the commacount is at the target index
             buffer[fieldPos] = sentence[sentencePos];
             fieldPos ++;
           }
           sentencePos ++;
         }
         buffer[fieldPos] = '\0';//add a zero
        }
```

Appendix D.4: Decoder Code

```
/*
The following code is the code that goes on the arduino on the ground connected to a laptop.
It gets data from a radio wave, interpreted by the radiometrix receiver chip, and prints
certain data from the original data received.
Nicholas Seidl
11/25/14
*/

String STARTER = "Tem1"; //starter and ender sequences for identification
String ENDER = "Tem2";
String NULL_TOKEN = "0000"; //what if printed if data cannot be extracted from GPS string

#define INLENGTH 500 // the largest string the Arduino can handle from the GPS chip

char inString[INLENGTH+1];
int inCount;
String s;

void setup()
{
 Serial.begin(4800); //start up serial ports TX0 and RX0 at the baud rate of the transmitter
}

void loop()
{
 Serial.flush(); //clear out the excess junk coming through the serial port
 inCount = 0;
 do
 {
   while(!Serial.available());//don't do anything if there isn't anything available from
serial
   inString[inCount] = Serial.read(); //once there is, read in data one char at a time
 }
 while (++inCount < INLENGTH);

 inString[inCount] = 0;
 s = String(inString);//convert the array characters into string
 s = extract(STARTER, ENDER, s);//extract the data from String

 if(s == NULL_TOKEN){
   Serial.println("No message");
 }
 else {
   Serial.println(s);
 }
}
```

Appendix D.5: Transmission Receiver Code

```
/*
The following code is part of the code that was put onto the receiving arduino. extract()
extracts the information between a given starter and ender sequence; in our case, it
extracted the latitude and longitude.
Nicholas Seidl
11/25/14
*/

String extract(String starter, String ender, String s) {
 int startCount = 0;
 int endCount = 0;
 int startPos = 0;
 int endPos = 0;

 boolean inside = false;
 boolean ending = true;

 for (int i = 0; i < s.length(); i = i + 1) {
   if (!inside) {//if we aren't inside the info part of our string
     if (starter.charAt(startCount) == s.charAt(i)) { //if the char in the starter sequence
mathces the char in the string
       startCount = startCount + 1;
     }
     else {
       startCount = 0;
     }

     if (startCount == starter.length())  {//if the numbers counted in the start sequence is
equal to the length of the starter sequence
       inside = true;
       startPos = i + 1;
     }
   }
   else  {
     if(ender.charAt(endCount) == s.charAt(i)) {//same logic as the logic structures for
testing starter sequence
       endCount = endCount + 1;
       ending = true;
     }
     else if (!(ender.charAt(endCount) == s.charAt(i)) and (ending == true)) {
       endCount = 0;
       ending = false;
     }

     if (endCount == ender.length()) {//if we are at the end of the sequence
       endPos = i - ender.length() + 1;
       i = s.length()+1;
     }
   }
 }
 if (s.substring(startPos,endPos).length() < 2) {//if there isn't anything between the
starter and ender sequence
   return NULL_TOKEN;
 }
 else  {
   return s.substring(startPos,endPos);
 }
}
```

Appendix D.6: Cut-Down Mechanism Code

```
/*
The following code runs the cut-down mechanism and also the LED that is helpful for finding
the payload at night.
Nicholas Seidl and Leo Jaimez
11/25/14
*/

int timeInMinutes = 0;//time starts at 0
int stringPin = 12;//pin which the plus to the LED is connected to
int ledPin = 13;
int timeForCutdownInMinutes = 120;//time in minutes when to cut down
int timeForLedInMinutes = 180;//time in minutes when LED turns on

void setup()
{
 pinMode(stringPin, OUTPUT);
 pinMode(ledPin, OUTPUT);
}

void loop()
{
 if(timeInMinutes = timeForCutdownInMinutes)
 {
    digitalWrite(stringPin, HIGH);
 }

 if(timeInMinutes >= timeForLedInMinutes)
 {
     blinkLed();
 }

 timeInMinutes++;

 delay(60000);//60,000 ms = 1 minute

 digitalWrite(stringPin, LOW);
}

void blinkLed()
{
  while(true)
 {
   digitalWrite(ledPin, HIGH);   // turn the LED on (HIGH is the voltage level)
   delay(15000);               // wait for a second
   digitalWrite(ledPin, LOW);    // turn the LED off by making the voltage LOW
   delay(5000);
 }
}
```

Appendix E: Sensor Descriptions

Temperature

An NTC thermistor was used to measure temperature. The NTC thermistor is made up of a pressed disc of a semiconductor. NTC stands for Negative Temperature Coefficient, meaning as temperature increases, the resistance provided by the thermistor decreases, thus changing the voltage detected and recorded by the arduino. The thermistor has a base resistance of 10 kΩ. [s1]

UV

To measure UV, the SparkFun UV Sensor Breakout - ML8511 was used. The UV sensor contains an amplifier that converts photo-current to voltage, outputting an analog signal, in accordance to the UV light detected. The arduino converts the analog signal into a digital signal to detect the level of UV. The sensor detects part of the UVB spectrum and most of the UVA spectrum, from 280-390nm.[s2]

Ambient Light

To measure ambient light, the SparkFun Ambient Light Sensor Breakout - TEMT6000 was used. The TEMT6000 is a silicon NPN epitaxial planar phototransistor. A phototransistor is a type of photodiode, which uses a semiconductor to convert light into current, when photons are absorbed. The epitaxial layer of silicon is added between the base and collector of the transistor to increase its switching speed. The ambient light sensor is connected to the arduino to record its data. [s3]

Gas Pressure

A Vernier Gas Pressure Sensor was used to detect the outside gas pressure. The sensor inside the housing is a SenSym SDX30A4 pressure transducer. This transducer has a membrane, with a vacuum on one side and the open end to the atmosphere on the other. The membrane flexes with a change in pressure, producing an output voltage that is linearly related to pressure. [s4]

Humidity

A Vernier Humidity sensor was used to detect the outside humidity. The sensor inside the housing is the Hy-Cal Engineering IH-3602-L Integrated Circuit Humidity Sensor. This sensor has a capacitive polymer that absorbs moisture, which changes the relative permittivity of the electric field between two electrodes in the sensor. This change in capacitance can be detected to measure the humidity. [s5]