ASR Balloon Launch Paper

Group: Ballooney Toons

Bodie Callaghan (SE)

Sophie Housser (ED)

Hannah Bernthal (PL)

Grant Wilson (RE)

Braden Rock (CE)

Spersh Goyal (EE)

Veer Vohra (ME)

I. Abstract

A payload designed to carry equipment to space, as well as the data collected by said payload, is examined in this paper. The payload itself was one cubic foot in size and weighed 2.8 lbs. A 9V battery powers the overall system which uses an Arduino Mega 2560, an Adafruit Rev B Data Logger Shield (with an onboard PCF8523 RTC and an SD interface), two cameras, three 3V, 25F capacitors, analog temperature sensors, a heating pad, a GPS chip, a CO monitor, a PICO and a FOX beacon, a pressure sensor, and a humidity sensor. While the system successfully meets its design goals, several improvements could have enhanced its effectiveness.

II. Introduction

Earth’s atmosphere is a layer of gases surrounding the planet that makes life possible by regulating temperature, protecting organisms from harmful solar radiation, and providing the air needed for biological processes. It is composed of 78% nitrogen, 21% oxygen, and the final 1% is made up of small amounts of argon, carbon dioxide, water vapor, and other trace gases. The atmosphere is divided into several layers based on temperature changes with altitude. From closest to farthest from Earth's surface, there are the troposphere, stratosphere, mesosphere, thermosphere, and exosphere. As altitude increases through these layers, different atmospheric properties change significantly, which is important for understanding how the atmosphere

behaves. In this experiment, sensors carried in the payload measured variables such as pressure, temperature, humidity, and GPS altitude, allowing us to study how these quantities change as the payload travels upward through the atmosphere.

An important motivation for studying the atmosphere is the greenhouse effect. The greenhouse effect is the process by which certain gases, primarily carbon dioxide, methane, and water vapor, trap heat in the atmosphere by absorbing infrared radiation emitted from Earth's surface. This process is why Earth is warm enough to support life. However, it can create a positive feedback loop where warming leads to even more warming. For example, higher temperatures increase atmospheric water vapor, which strengthens the greenhouse effect and traps more heat. It is important to study the atmosphere so that scientists can better understand phenomena like the greenhouse effect and learn how to keep our planet safe and healthy.

III. Temperature in the Atmosphere

A. Why Temperature Changes as Height Increases

Earth's atmosphere is split into four main layers, each determined based on how temperature changes with altitude.[1] While pressure and density both decrease with altitude, the changes in air temperature with distance—or the temperature gradient—are not consistent. In fact, each layer of the atmosphere alternates between increasing and decreasing temperatures depending on the location of the heat source for that particular atmospheric layer, as seen in Figure 1. If the energy is being absorbed at the bottom of a layer, temperature will decrease as altitude increases (troposphere and mesosphere), but if energy is being absorbed from the top of an atmospheric layer, temperature will increase along with altitude (stratosphere and thermosphere). The atmosphere doesn't have just one source of heat; it comes from the ground, the ozone layer, and the upper thermosphere. These pieces are all at different altitudes, resulting in inconsistent temperature gradients.

[1] This paper does not discuss the exosphere—the highest layer of the atmosphere—as it begins at 700,000 meters, and our weather balloons only reach a maximum of around 120,000 meters.

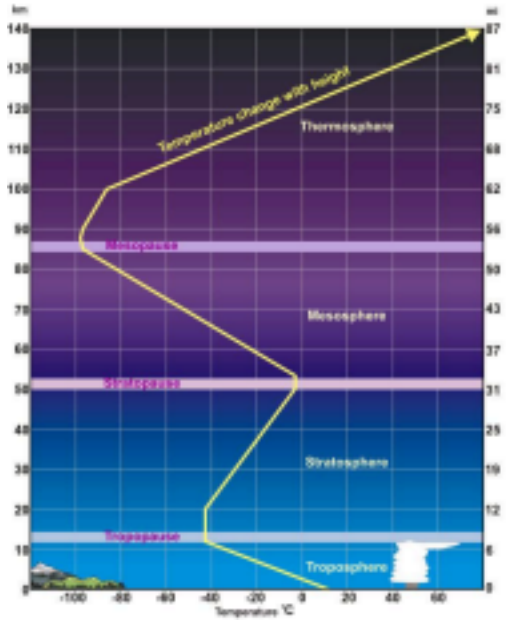

Figure 1: A model from Lumen Learning depicting how temperature changes as altitude increases in the dif erent layers of the atmosphere

The troposphere is the lowest layer of Earth's atmosphere, spanning from the surface to about 20 kilometers above the equator. When a parcel of air rises in the troposphere, the atmospheric pressure decreases, causing the air to expand. As it expands, it does work on the surrounding molecules and loses kinetic energy, causing the temperature to drop by about 6.5˚C for every kilometer of altitude gained. This process is adiabatic, meaning no heat is exchanged with the surrounding environment; the cooling occurs purely because the expanding air does work on its surroundings. The troposphere's decreasing temperature with altitude stems from the fact that its primary heat source is Earth's surface. The surface absorbs visible and near-UV solar radiation and warms, and this heat is moved upward through convection. In this process, warm, less-dense air rises and is replaced by cooler air from above, resulting in rapid vertical mixing and causing weather to occur. Because the heat source is at the bottom and weakens with distance, temperature naturally decreases going up.

The next layer up is the stratosphere, which extends from around 20-50 kilometers above the equator. In the stratosphere, the pattern reverses: whereas temperature decreases with altitude in the troposphere, it increases with altitude in this layer. The reason for this is photochemistry. Described by British scientist Sydney Chapman in 1930, photochemistry describes when a molecule absorbs a photon of light directly, and the energy snaps its chemical bonds. In the atmosphere, this happens with O_2. A photon of light will hit a molecule of oxygen, breaking it

into two highly reactive, single oxygen atoms. These atoms will bond with other O_2 molecules, forming O_3 (ozone) and releasing heat. The UV rays will then split up this newly formed ozone molecule, breaking it back into its original parts, and the reaction restarts. This process occurs over and over again, and the cycle is what generates heat in the stratosphere.

Above the stratosphere lies the mesosphere (about 50 to 85 kilometers up), and things start to cool down again for two main reasons. First, the ozone concentration decreases dramatically above the stratosphere. Solar UV absorption falls off along with the diminishing ozone concentration, since ozone is the primary absorber of the UV wavelengths that survive passage through the layers above. Furthermore, the increasing distance from the ozone layer below contributes to significantly lower heat input. The second reason that the mesosphere cools down is CO_2 radiative cooling. In the lower atmosphere, CO_2 absorbs infrared radiation emitted by Earth's surface, acting as a greenhouse gas. In the mesosphere, however, it plays the opposite role. The air at this altitude is so thin that infrared radiation re-emitted by CO_2 molecules is far less likely to be reabsorbed by neighboring molecules, but instead, much of it escapes directly into space, resulting in a net loss of heat that cools the surrounding atmosphere. Because of these two mechanisms, temperature continues to decrease until the mesopause (boundary between the mesosphere and the thermosphere), which ranges from altitudes of about 85 to 100 kilometers. This is the coldest region in Earth’s atmosphere, reaching temperatures of -100˚ Celsius.

This brings us to the highest layer of the atmosphere, the thermosphere (85 km+). Here, the temperature rises quickly and abruptly with altitude due to direct absorption of highly energetic solar radiation—extreme UV and X-rays that were not absorbed by other layers. In this region, temperatures can reach 2,000˚C or more during periods of high solar activity. Temperatures can rise so quickly in the thermosphere because of the very strong heat source and ineffective cooling mechanism (removing heat through conduction down into the mesosphere). However, despite these extremely high temperatures, any individual in the thermosphere would feel intense cold. This contradiction between measured and felt temperature is due to the definition of temperature itself. Temperature is defined as the average kinetic energy of the molecules in a substance, not the amount of heat energy or how much heat would be transferred. Temperature measures the speed of particles, while heat (what we experience) is the transfer of energy.

In the atmosphere, temperature and altitude do not have a direct or even consistent relationship.

Instead, each layer alternates between an increasing and decreasing temperature depending on the location and intensity of its heat source and the concentration of its molecules.

IV. Payload Design

The Ballooney Toons payload has a multilevel design for optimal space efficiency, insulation, and organization. Measuring from the inside of the walls, the payload has a volume of 25 cubic inches, requiring a very intentional interior design. Using a 3D printed structure with three 4.5” x 4.5” levels separated by 2.6” in between them, each component of the payload was separated according to weight distribution, ease of connections, and specific location requirements. The bottom level of the 3D printed structure included the FOX beacon, PICO, both cameras, and the battery for the cameras. The middle level housed our Arduino, two solder boards, two 9V batteries, and a buzzer zip-tied to the bottom of the platform. The top level contained our cut-down capacitors, one solder board, our GPS tracker, and one 9V battery. Many of our components required exposure to the outside environment and thus had holes in the payload. The humidity sensor required shade and was threaded through a hole at the bottom of the payload to shield it from sunlight. Both antennas needed to poke out of the payload and point either sideways or downwards to avoid popping the balloon. In our design, we opted to have the PICO pointed downwards for optimal data retrieval during flight, and the FOX pointing perpendicular to the sides of the payload for maximized safety during landing. The FOX beacon was required for finding the payload on the ground, so it was necessary that it would survive the fall. Two circular holes were cut for the cameras, just enough for clean images, but still minimizing heat loss and the risk of losing a camera. The external temperature sensor and CO sensor were poked out of the side of the payload closest to their respective Arduino pins to mitigate any unnecessary wire tangling. The cut-down wires were poked through the lid of the payload and strung up to circle around the string attaching the payload to the balloon.

A. CAD Design

The idea behind the 3-level structure was to create a secure system for organizing all of the payload components while still allowing for disassembly and relatively easy access to the lower levels of the payload. This proved to be extremely helpful on launch day as we realized 5 minutes before launch that a camera had accidentally been switched from taking photos to video,

and we were able to fix it fairly quickly. The structure was 3D printed with PLA, which yielded a strong and light structure, perfect for strict weight requirements from the FAA.

Each platform has 7 rectangular cutouts to make it as easy as possible to thread wires between layers. On the top of the base and middle platform there are 4 rectangular poles on each corner, 0.25” x 0.25” x 2.43”. The top 0.17” of each rectangle decreases in side length to 0.12” to fit into the corresponding slots on the bottom of the platform above—the poles of the bottom platform slot into the underside of the middle platform, and the poles of the middle platform slot into the underside of the top platform.

Designing the 3D structure warranted much trial and error. In the first few trials, the pole inserts were too large for their respective holes, despite accounting for PLA expansion, so their side lengths were decreased by 0.03 in less than the size of the hole to accommodate. Unfortunately, this overcorrected the problem, and on the day of launch, the middle and top levels could not remain attached with the packed wires of the Arduino pushing up so much. In the future, it would be important to do further testing to find an exact fit to allow for moderately easy disassembly, while still ensuring a secure connection.

The CAD drawings of the internal structure of the payload can be seen in Figures 2 and 3.

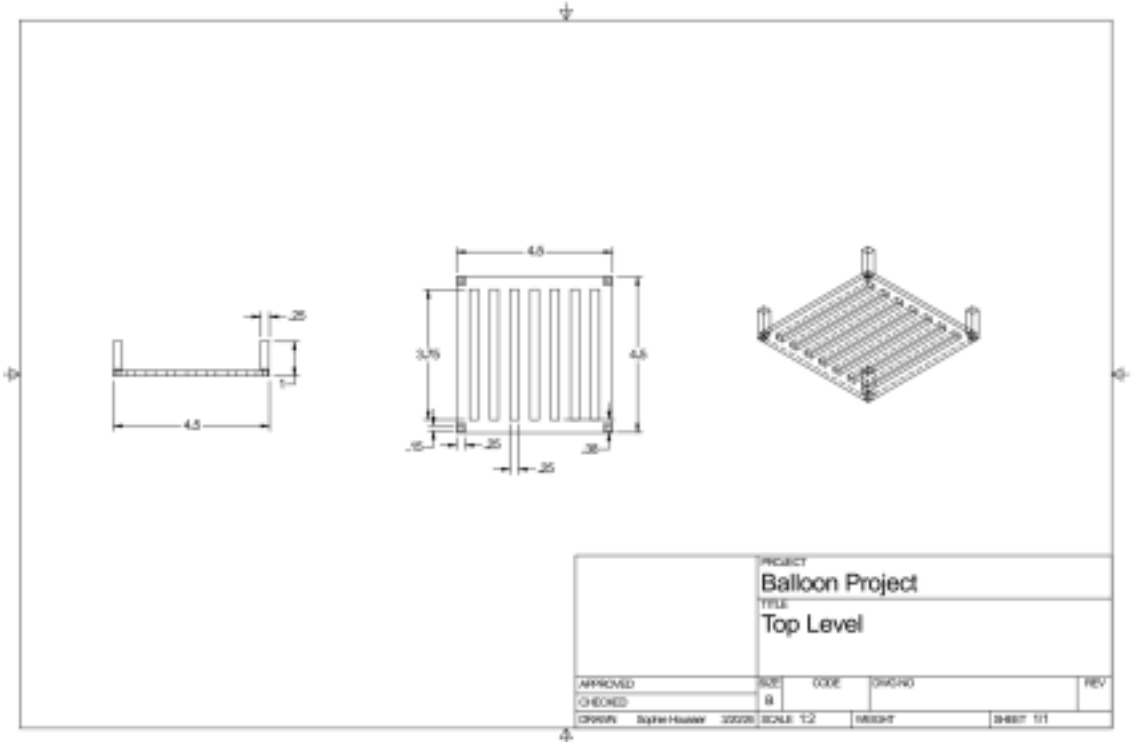

Figure 2: 3D model of the top platform of the payload structure

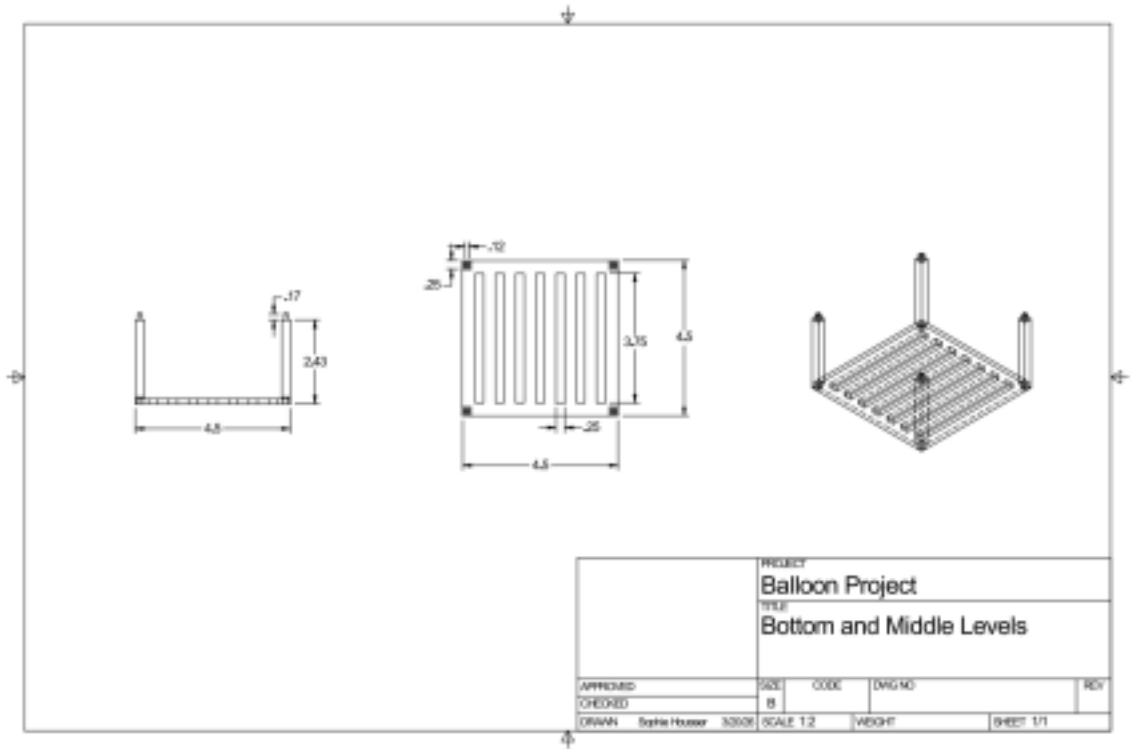

Figure 3: 3D model of the bottom and middle platforms of the payload structure

B. Heating Pad Circuit

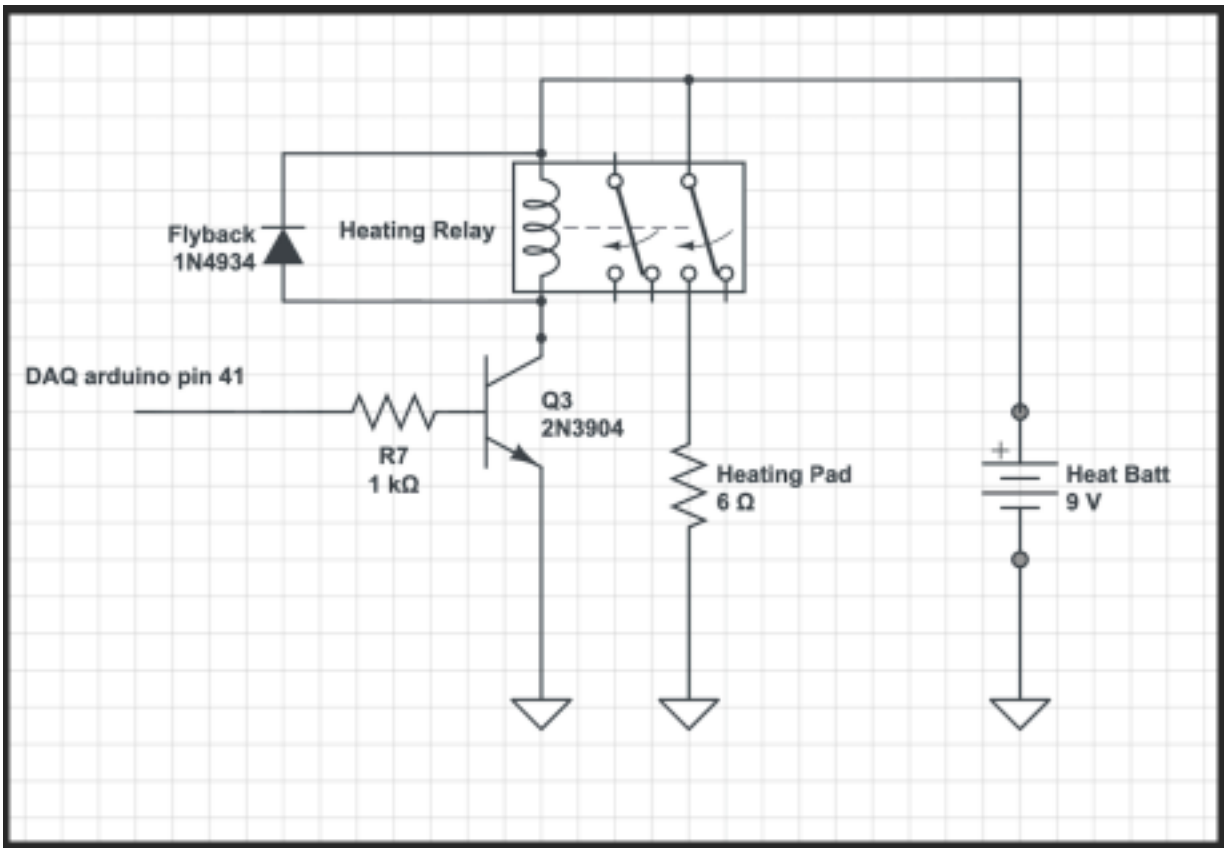

Figure 4: heating pad circuit

The heating pad circuit is designed to allow the Arduino to control the high-current heating pad without exceeding the limits of its digital pins. The heating pad requires approximately 1 amp of current to operate, but the Arduino's digital pins can only safely supply 40 milliamps. To bridge this gap, the circuit uses a dedicated 9V battery as a separate power source for the heating pad and uses the Arduino signal purely as a low-power control signal rather than a direct power source. The core of the control side is a 2N3904 NPN transistor (Q3), whose base is connected to Arduino digital pin 41 through a 1kΩ current-limiting resistor (R7). When pin 41 goes high, a small base current flows through the transistor, switching it on and allowing current to flow from collector to emitter. The transistor acts as an electronically controlled switch, activated by the Arduino's small signal but capable of handling the larger current needed to drive the relay coil.

The relay serves as the interface between the low-power control circuit and the high-current heating circuit. When the transistor switches on, the relay coil is grounded

and

current flows through it, generating a magnetic field that physically closes the relay's contacts and connects the 9V battery to the heating pad, turning it on. When the Arduino pin goes low, the transistor switches off, the coil loses power, and the contacts spring open, disconnecting the heating pad. A flyback diode (1N4934) is placed across the relay coil to protect the circuit from a potentially damaging voltage spike. When the transistor switches off and current through the coil is suddenly interrupted, the coil's collapsing magnetic field generates a large reverse voltage spike. Without the diode, this spike would travel back through the transistor and into the Arduino, likely destroying both. The flyback diode provides a safe path for this spike to dissipate back through the coil, protecting the rest of the circuit.

C. Buoyancy Force Calculations

In order for our payload to rise in the air, the buoyant force must be greater than the mass of all of the components. To calculate this, we assumed a spherical balloon with a diameter of 3 feet (0.9144 m) and used the density of air at sea level, 1.225 kg/m^3. Buoyant force is equal to the weight of the fluid displaced, in this case, air, by the balloon:

$$\frac{4}{3}\pi ◻◻^3 = \frac{4}{3}\pi \times (0.9144)^3 = 3.2◻◻^3$$

$$◻◻ = \rightarrow$$

$$◻◻_{◻◻} = ◻◻ = \rho◻◻◻◻ = (1.225◻◻◻◻/◻◻^3) \times (9.81◻◻/◻◻^2) \times (3.2◻◻^3)$$

This gives us a buoyant force of

$$◻◻_{◻◻} = 38.5◻◻$$

To determine the constraints of the weight of the payload, we must subtract the weight of the helium in the balloon and the weight of the balloon itself $(1.2◻◻◻◻ \times 9.81◻◻/◻◻)$: $^2 = 11.77◻◻$

$$\rho_{◻◻◻◻} = 0.1785\ ◻◻◻◻/◻◻^3$$

$$◻◻ = \rho◻◻◻◻ = (0.1785\ ◻◻◻◻/◻◻^3) \times (9.81◻◻/◻◻^2) \times 3.2◻◻^2$$

$$◻◻_{◻◻◻◻} = 5.6◻◻$$

$$F_{net} = F_b - F_g - F_{payload}$$

$$F_{net} = 38.5V - 5.6V - 11.77V$$

$$F_{net} = 21.1V \rightarrow 2.15kg$$

In order for there to be optimal upwards acceleration, the gravitational force of the payload should not exceed 50% of the buoyant force, so we were constrained to a payload mass of about 1.075kg. As detailed in Table 1, the final mass of our payload was about 1kg, so we just barely made the constraint.

Item	Mass (kg)
Balloon	1.2
Grant's Pico & Fox	0.15716
Weight of Helium Used	0.5
2 cameras	0.148
3 9V batteries	0.103
Camera battery	0.1777
Arduino and Servo w/ 3D structure	0.1867
Spersh's Capacitors	0.0665

Payload w Airtag 0.15

Mass without balloon and helium: ≈1 kg

Table 1: Total mass of components in the payload

D. Converting from Pressure to Altitude

NASA's Empirical Model gives us pressure as a function of altitude, so in order to convert altitude to pressure, we must reverse the equation. The pressure can fall into three different zones: the troposphere, where $p < 22.65$ kPa, the lower stratosphere, where $2.488 \text{ kPa} < p \leq 22.65$ kPa, and the upper stratosphere, where $p \leq 2.488$ kPa.

In the troposphere, the given equations are:

$$\text{[illegible]} = 15.04 - 0.00649h$$

$$\text{[illegible]} = 101.29 \left(\frac{\text{[illegible]}}{288.08}\right)^{5.256} \text{[illegible]} + 273.1$$

Then substitute in the temperature:

$$\text{[illegible]} = 101.29 \left(\frac{(15.04 - 0.00649h) + 273.1}{288.08}\right)^{5.256}$$

Isolate h in a fraction:

$$\frac{\text{[illegible]}}{288.08} = \frac{288.14 - 0.00649h}{288.08} \qquad \left(\frac{\text{[illegible]}}{101.29}\right)^{\frac{1}{5.256}}$$

Then isolate h:

$$h = \frac{288.14 - 288.08 \times \left(\frac{\text{[illegible]}}{101.29}\right)^{0.190}}{0.00649}$$

In the lower stratosphere, the forward equation is:

$$\text{[illegible]} = 22.65 \times \text{[illegible]}^{1.73 - 0.000157h}$$

$$\frac{22.65}{\text{[illegible]}} = \text{[illegible]}^{1.73 - 0.000157h}$$

Then we must take the natural log of both sides:

$$\ln\left(\frac{\text{[illegible]}}{22.65}\right) = (1.73 - 0.000157h)$$

And then we can isolate h:

$$h = \frac{1.73 - \ln\left(\frac{\text{[illegible]}}{22.65}\right)}{0.000157}$$

In the upper stratosphere, the forward equations are: [illegible] =−

$$131.21 + 0.00299h$$

$$\left(\frac{}{216.6}\right)^{-11.388}$$

$$\text{}\text{} = 2.488 \times^{\text{}\text{}+2.73.1}$$

We can then substitute temperature and divide by 2.488: $\left(\frac{}{216.6}\right)^{-11.388}$

$$\frac{}{2.488} = {}^{(-131.21+0.00299h)+273.1}$$

Then isolate h:

$$\left(\frac{}{2.488}\right)^{-1}$$

$$\frac{}{11.388} = (-131.21 + 0.00299h) + 273.1$$

Then solve:

$$h = \frac{216.6 \times \text{}\text{}}{\left(\frac{}{2.488}\right)^{-0.0878}} - 141.89$$
$$0.00299$$

$$216.6 \times^{\text{}\text{}}$$

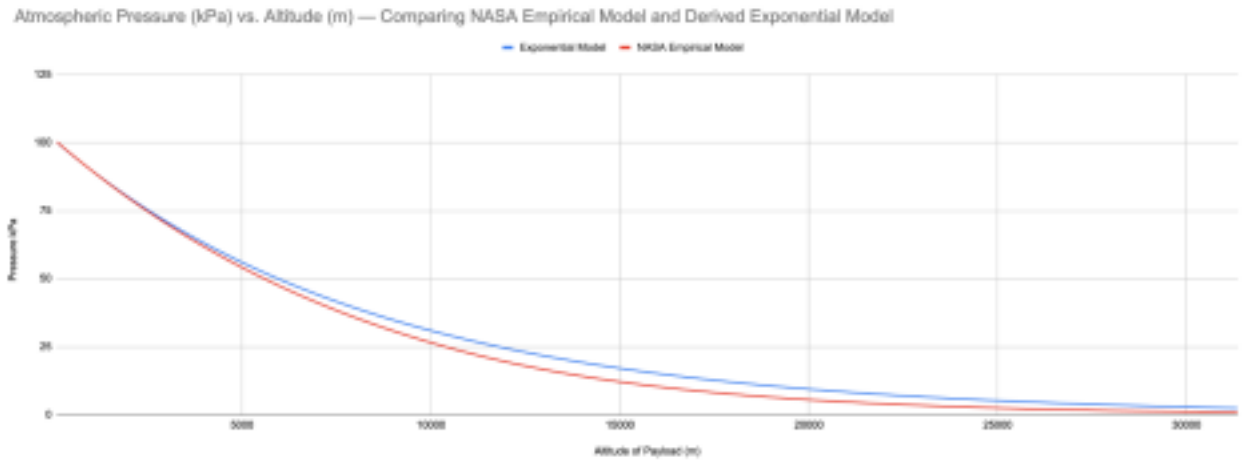

Figure 5: Graphing expected pressure vs. altitude using the derived exponential model and the NASA Empirical Model

E. Derivation of the Exponential Model of Pressure vs. Altitude:

Consider a thin horizontal slab of air at altitude h with thickness dh. Refer to the figures

below for a visual graphic. The pressure difference across this slab must support the weight of the air within it. This gives the hydrostatic equation:

$$\frac{dP}{dh} = -\rho g$$

where P is pressure, ρ is air density, and g is gravitational acceleration (9.81 m/s²). The negative sign indicates pressure decreases with increasing altitude.

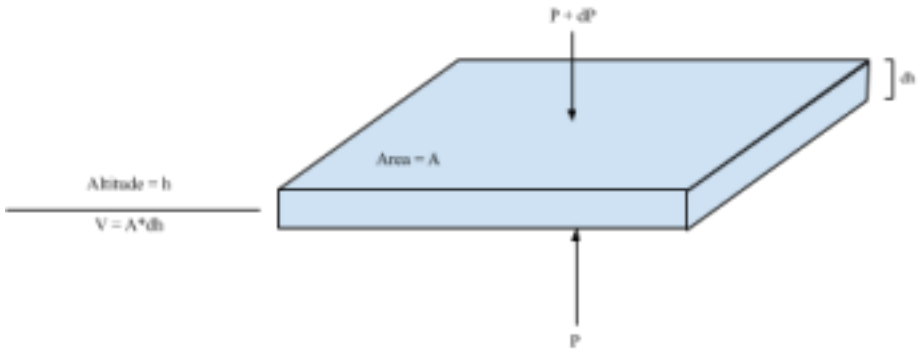

*Figure 6: a model of the pressure acting of the slab of air at height h with width dh, area A, and volume A*dh*

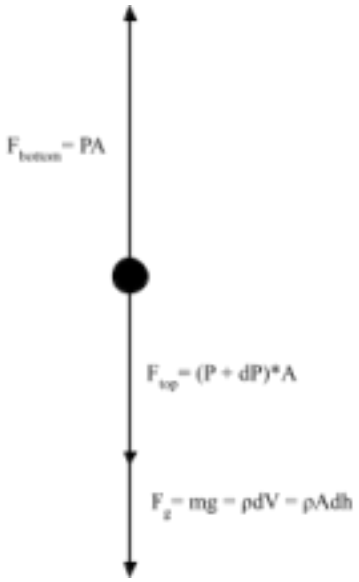

Figure 7: Free body diagram depicting forces acting upon a slab of air in equilibrium, whether the force of pressure from the bottom of slab the air is equal to the force of pressure from the top of the slab of air and the weight of the air

The Ideal Gas Law states: $PV = nRT$, and we can rewrite this to isolate pressure: $P = \frac{nRT}{V}$

We can then substitute n for (mass of gas / molar mass of air): $n = \frac{m}{M}$

$$\frac{m}{V} = \rho \quad \rightarrow \quad \frac{nM}{V} = \rho \quad \rightarrow \quad \frac{PM}{RT} = \rho$$

We can rearrange this to get $\rho = \frac{PM}{RT}$. Substituting this into the hydrostatic equation, we get:

$$\frac{dP}{dh} = -\frac{PMg}{RT}$$

→ rearrange to get: $\frac{dP}{P} = -\frac{Mg}{RT}dh$

1

We then assume that T is constant with altitude (and M, g, R, and T are all constants) and integrate both sides:

$$\int_{P_0}^{P} \frac{dP}{P} = -\frac{Mg}{RT}\int_{0}^{h} dh$$

$$\int_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h} dh$$

We can put this into exponential form, yielding this equation as the final result:

$$f(h) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{h - \mu}{\sigma}\right)^2}$$

With these constants:

P_0 = 101,325 Pa (sea level pressure) M = 0.02897 kg/mol (molar mass of dry air)

g = 9.81 m/s²

h = altitude in meters

R = 8.314 J/mol·K

T = 288 K (assumed constant/sea level standard temperature)

While the exponential model does great at predicting the pressure at lower altitudes, as height increases, the model breaks down. This is because the exponential model assumes constant temperature, and as discussed in section II, atmospheric temperature is not only changing, but also increasing and decreasing with an increase in altitude.

The NASA Empirical Model offers a more accurate prediction because not only does it describe the relationship between altitude and pressure as a piecewise function (offering more variation with altitude), it also provides temperature as its own function, not just a constant.

V. Reel-in System

The goal of the reel-in system was to serve as a test run before the full space launch. The payload, loaded with all of our sensors and technology, is tethered to a 1000-foot line wound around the reel. The balloon is sent up to the full length of the tether and then pulled back down in a controlled manner. The purpose of this tethered launch is to verify that all systems are working, such as data collection, GPS tracking, and sensor logging, so that any necessary changes can be identified and made before committing to the real launch.

When designing the reel-in system, there were three main considerations. First, the structure needed to be stable enough to handle the tension of a 1000-foot tether without shaking or shifting during operation. Second, the design aimed to use as few materials as possible to keep

it lightweight and sustainable. Third, the build was constrained by the fact that glue and laser cutting were not permitted, meaning every structural decision had to rely on mechanical fastening alone.

With those constraints in mind, the frame was built around triangular geometry. Triangles are inherently rigid and resist deformation under load, making them ideal for a structure that needed to stay stable while the motor was running and tension was on the line. The base was kept square for a wide, grounded footprint, and two triangular side frames were built up from it. A thin crosspiece connected the two triangles at the top, eliminating horizontal wobble and tying the whole structure together. The result was a frame that was sturdy, material-efficient, and built entirely without adhesives.

Overall, the mechanical structure performed well under load, with the frame remaining stable and the belt-drive transmission successfully transferring torque from the motor to the spool. However, during initial testing, the motor was unable to generate enough force to lift even a roll of tape off the ground, which indicated a problem with the power supply rather than the mechanical design itself. After switching to the correct power supply, the system demonstrated the ability to lift up to 33 pounds. The mechanical advantage was calculated by dividing the diameter of the driven gear (5.686 in) by the diameter of the driving gear (0.989 in), giving a theoretical mechanical advantage of 5.75. This means the system multiplies the motor's torque by 5.75 while reducing its speed by the same factor. The motor operates at 300W and 2750 RPM, producing a torque of approximately 1.04 N·m. After passing through the gear system at an estimated efficiency of 70%, the output torque at the spool is approximately 4.2 N·m. In practice, the actual mechanical advantage was confirmed through load testing, where the motor alone could lift 9.7 pounds, and the full belt-drive system lifted 33 pounds, giving an actual mechanical advantage of approximately 3.40. The difference between the theoretical value of 5.75 and the measured value of 3.40 can be attributed to real-world losses such as friction in the bearings, slippage in the belt, and mechanical inefficiencies in the drivetrain that are not accounted for in the theoretical calculation.

The primary failure point was the directional switch. The design called for a three-way on/off/on

switch to control motor direction, allowing the reel to both pay out and retrieve the line from a single switch. One side of the switch was non-functional, so reversing the motor required manually swapping the positive and negative leads rather than simply flipping the switch.

In terms of improvements, the most immediate fix would be replacing the faulty three-way switch with a fully functional one, allowing the motor to reverse direction without manually swapping the leads. On the structural side, adding two parallel rods running through the middle of the triangular frames would further reduce horizontal shaking and increase overall rigidity. Finally, to reduce weight and make the system more compact, the gear ring could be scaled down in size, reducing both the material used and the overall footprint of the system without compromising the mechanical advantage.

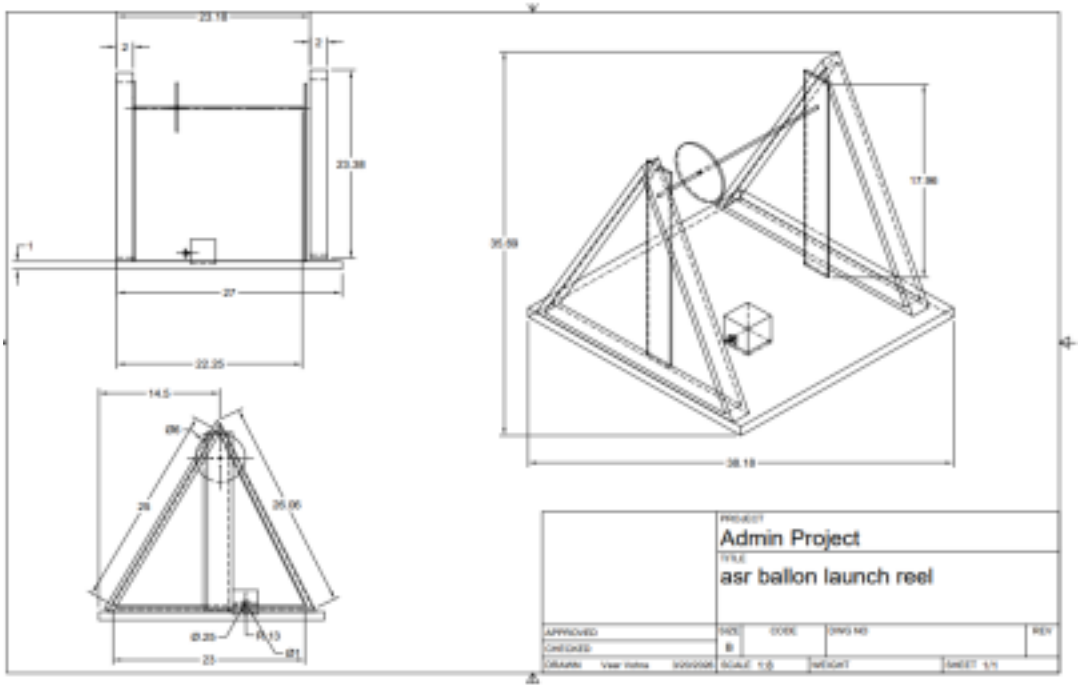

Figure 8: drawing of reel in system with dimensions

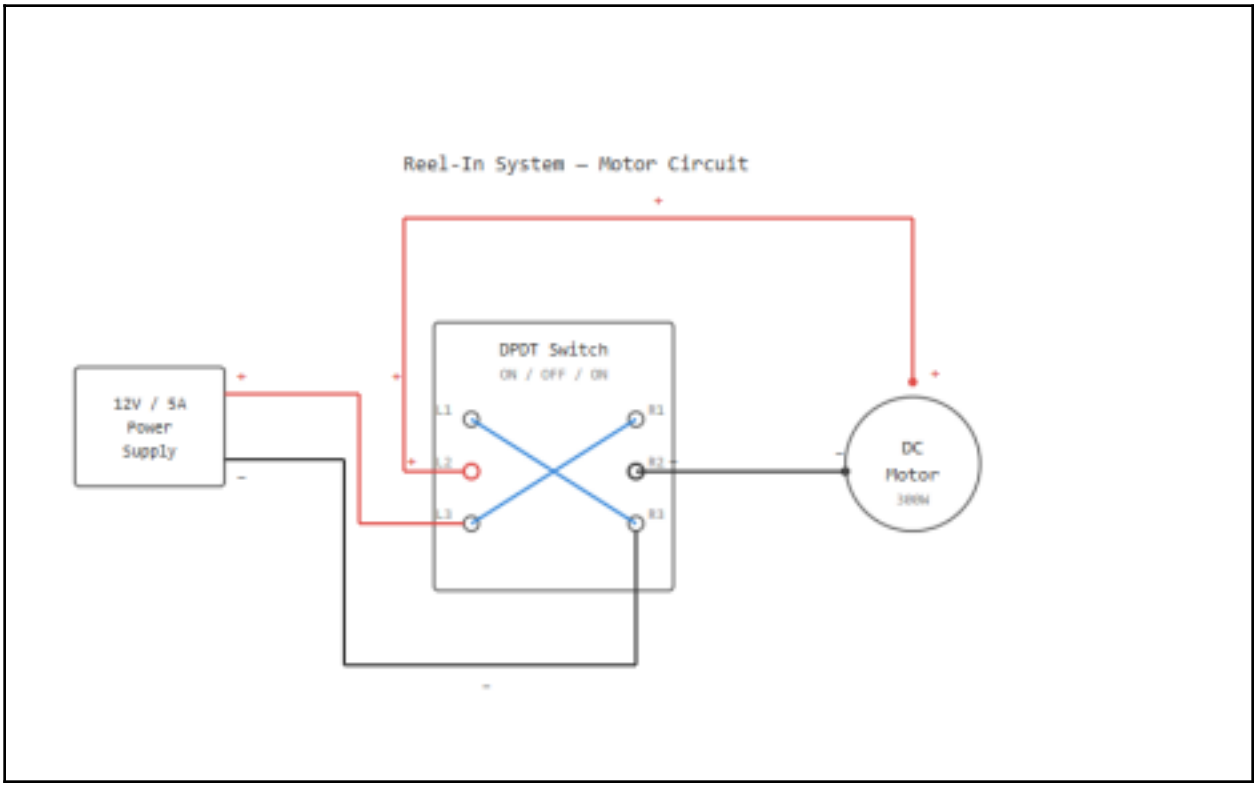

Figure 9: Electrical circuit diagram of reel in system

VI. Cut-down Mechanism

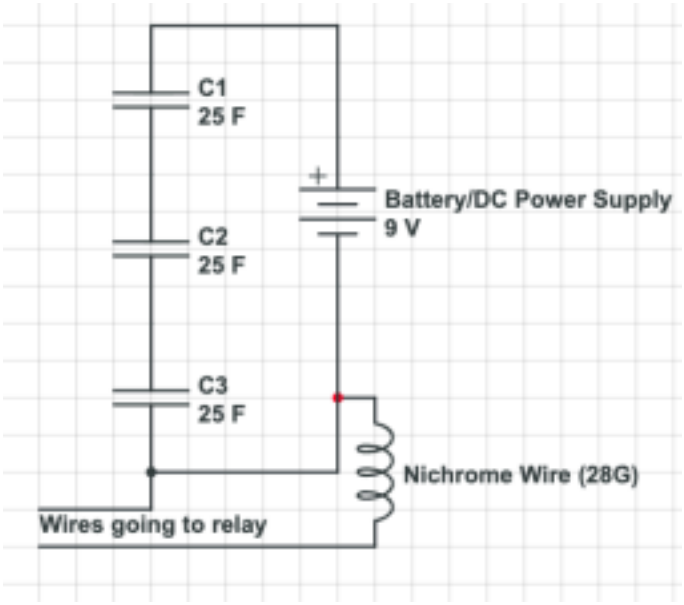

Figure 10: Circuit diagram of the cutdown sub circuit with wires going to the DAQ Cutdown Sub circuit pictured below.

The cutdown circuit relies on three 3-volt, 25-farad capacitors in series. These capacitors discharge through a nichrome wire and are triggered by an 8-pin relay. Pin 22 on the DAQ Arduino is an output and set high when the pressure reading on the Sparkfun pressure sensor is less than 10 millibars and the time since the Arduino got power (from a 9V battery) is greater than 120 minutes. The 120-minute minimum prevents premature cutdown caused by a faulty pressure reading or sensor glitch. Pin 22 on the Arduino is wired to the base of an NPN transistor that grounds the coil of an HKF-19 relay (rated at 2 amps). When the relay closes, the cutdown circuit in Figure 10 is closed, allowing the capacitors to discharge through the nichrome wire.

The decisions made to design the circuit in the above fashion hinged on a test conducted at the beginning of the project. The nichrome wire was attached to alligator clips, which were directly attached to the positive and negative terminals of a DC power supply. Then, using the red and white string that was eventually used for the final launch, the string is wrapped around the nichrome wire, with a weight (a thick metal washer) hanging. Then, the power supply is switched on, and different current values (and how much voltage they need) are tested. After observing that 0.5 amps would draw 5V, and would cut down in approximately 4 seconds, the current was increased to 1.5 amps, which drew 8 volts and cut down in roughly 2.5 seconds.

Then, to achieve a virtually instantaneous cutdown time, current was increased to 4 amps, which drew 9 volts and cut down the washer instantaneously. To get 4 amps or more of current going through the nichrome wire, it was decided to put the three capacitors in series. By putting the three capacitors in series, we were able to get 8.33 Farads in capacitance, 9 Volts, 75 Coulombs of charge, and a peak current of 6.52 amps.

To get total capacitance, we know that 1/C_total = 1/C1 + 1/C2 + 1/C3. Substituting values into the equation, and doing the reciprocal to get C_Total, we get C_total = (1/25 + 1/25 + 1/25)^-1 = 8.33 F. To get the total voltage, we just add all the voltages of each individual capacitor, so V_total = 3V + 3V + 3V = 9 Volts. To get charge, Q = C_total * V_Total = 8.33 * 9 = 75 Coulombs. 6.52 amps of current were determined experimentally by putting an Ammeter in series with the nichrome wire as the capacitors discharged. Ultimately, this capacitor setup worked phenomenally well, allowing the nichrome wire to get red hot every time the system was tested, and allowing for the least amount of time needed to actually perform the cut down.

Capacitors in series were determined to be the better option than capacitors in parallel, as the same three capacitors in parallel only produced a peak current of 0.83 amps, which, admittedly, still got cut down to work, though it took much longer (approximately two seconds). 0.83 amps was also determined experimentally by putting an ammeter in series before the nichrome wire. 28-gauge nichrome wire was determined to be the best option as it was thick enough not to break when handled roughly, was thin enough to still get red hot, and survived the wear-and-tear of the system throughout the project. The nichrome wire was fastened to the rest of the cut-down circuit by stripping a portion of the 35-inch-long insulating wires going up past the parachute, tightly coiling a portion of the nichrome wire around the exposed part, and then adding shrink wrap to the exposed wire and coil to ensure no exposed wires and keep the nichrome wire in place. The resulting arc of Nichrome wire was then wound along the red and white kite string. Though this ended up interweaving the 35-inch insulating wires, after inspecting the soldered connections and the presence of slack in the wires, the mounting system was deemed acceptable multiple times by the electrical engineer, project leader, and other group members.

Another significant design decision that was made was to use a relay to trigger the cutdown. The Hui Ke HKF19F-DC5V-SHG relay was picked to ensure that the cutdown circuit

was electrically isolated from the rest of the electronics system. This was crucial as our group elected to only use one Arduino, and had all data acquisition/software running off of the one Arduino. If the capacitors somehow discharged back into the Arduino, there would be a high likelihood that something would break, whether it's a single pin, the internal power rails of the Arduino, a voltage regulator, or the entire chip. The consequences of such an occurrence would be devastating, so it was important to ensure that the cutdown was electrically isolated. With the relay, we have a hardware solution that ensures the capacitors do not discharge anywhere *but through* the nichrome wire, as the pins (5 and 7) that the cut-down wires go to literally don't complete the circuit unless they get a signal from the Arduino. But even then, the Arduino doesn't actually share a loop with the cutdown, as they connect to different pins on the relay (1 and 8) that never complete a circuit with the cutdown circuit.

When building and soldering the circuit, wires that didn't have both ends on the board (the 35-inch insulating wires, for example) were mounted in a way that, if stressed at any time, would put stress on the corner of the board, not the actual connection. This was achieved by soldering the wire in from the bottom of the board, curling the other end around the bottom, and then up to the top of the cut-down mechanism. When charging the capacitors, a power supply set to 9 volts was used, or a 9V rechargeable battery. The battery/power supply would be connected to two wires, one coming from the positive bus and the other from the negative bus on the board.

The system was also tested rigorously in the pressure and temperature chambers to ensure reliability before the final launch. Tests were done with hung weights in the pressure chamber, and the DAQ Arduino to ensure that the trigger and subsequent cutdown would work based on the code thresholds of time/pressure. Though a test with the 120-minute timer and 10 millibar threshold was not done due to time constraints, the electrical and software engineers tested the cutdown circuit dozens of times at different timers and pressure thresholds to ensure system reliability. Furthermore, the electrical engineer would insert the cut-down circuit into the freezer and then directly into the pressure chamber to observe how the system behaves in an environment similar to space. This was done to ensure that the wire would not become brittle and snap due to the rapid cooling of the nichrome in space, and subsequent rapid heating due to the capacitors' discharging. Regrettably, the electrical engineer on the team was testing so rigorously and often that one of the vacuum chamber glass bell jars broke. The electrical engineer is still

looking for a suitable replacement, per Dr. Dann's constraints.

Unfortunately, the cut-down circuit was not working properly by the date of the tethered launch, due to a late pivot away from driving the trigger with a MOSFET and NPN transistor combination (pivoted away due to complexity, inconsistency with it working or not, and because the relay offered better electrical isolation from the rest of the system). After debugging the circuit rigorously after the tethered launch, we found that parts of the breadboard didn't work anymore, though the circuit logic and implementation were correct. After moving to a new breadboard, the circuit ended up working.

Upon retrieving the payload, it was clear that the cutdown was not successful, as the carabiner and remnants of the balloon remained attached to the payload. The nichrome wire also appeared to have snapped at two places, roughly an inch away from each connection to the 35-inch insulated wires. It was clear that the capacitors discharged, as they were depleted when checked with a voltmeter when the payload was back in the lab, and the code reached the trigger threshold (120 minutes, and pressure less than 10 millibars). This was a bizarre and perplexing result because the likely error was that the nichrome wire underwent cooling (due to the vacuum of space) and rapid heating (in the form of the capacitors discharging), and ultimately broke due to the rapid, uneven expansion/contraction of the wire due to the extreme temperatures. This was perplexing, though, since the system had been tested immediately after being placed in the freezer to ensure that the nichrome wire would not snap.

VII. Software Design and Data Acquisition System

A. Arduino Overview

Arduino is both a type of compact computer known as a microcontroller and an open-source programming platform; together, Arduino is used as the heart of many hobbyist programmable circuits and electronics. The board itself contains a Central Processing Unit (CPU) which runs the uploaded code and performs the arithmetic, logic, comparison, and data processing functions of the board. The board also has a memory built into the microcontroller for storing programs. The Arduino ATMega 2560 has 256 kilobytes of flash memory for storing code, but only 8 kilobytes of RAM. RAM is the short-term, quick access memory that processors like the Arduino use to hold and work with data while running. A program must take up less than

8 kilobytes of memory on the ATmega 2560; otherwise the Arduino might have difficulty running it smoothly. The third main component of the Arduino is its set of input and output pins, which can be wired to external circuits. The multitude of pin types on an Arduino makes it immensely powerful as a controller and processor. Below is an overview of each type used in this project:

Digital Input/Output pins (numbered 0 though 53) can occupy a HIGH state (5V) or a LOW state (0V). As an output, the binary signal of digital pins can be used to power external components; as an input, these pins can read the binary voltage (HIGH/LOW) of external components or other pins. The digital I/O pins are the main avenue for integrating external components into the logic of an Arduino program.

Analog pins (numbered A0-A15) can only function as inputs: they can read a variable voltage between 0 and 5V, as opposed to the binary signal of digital pins. This voltage is converted into an Analog to Digital Converter (ADC) value, which uses a resolution between 0 and 1023 bits to indicate the voltage relative to 5V. Analog pins are useful for reading nodes of voltage dividers due to their 1024-bit resolution. Often, the voltage readings from analog pins are used to convey some other value or metric (such as temperature, sunlight, or gas concentration), though they can be used to integrate external components into the controller's logic.

Serial Communication pins (RX and TX) are a form of digital pins that use a communication protocol known as UART (Universal Asynchronous Receiver-Transmitter) to communicate data with other UART pins. The Transmit (TX) pin of one device connects to the Receive (RX) of the other, and vice versa. UART standardizes a way for communicating and interpreting data through bytes. A baud rate is set in the code over which the microcontroller and component use as the agreed-upon sampling rate of bits per second. A bit is just a high or low signal, and UART allows components to reassemble each bit into bytes (8-bit strings that correspond to a character). Different baud rates communicate at different speeds: the more data needed to be communicated, the higher the rate. In this project, the GPS chip communicates with the DAQ Arduino via RX1 and TX1 using UART, and a baud rate of 9600 (i.e., 9600 bits per second) is used for cross communication (see Appendix I, line 221).

Beyond UART, Arduino has another, separate communication protocol known as I2C for elements like the SparkFun humidity and pressure sensors used in this project. I2C (an

abbreviation for Inter-Integrated Circuit) also uses two pins to communicate with components. I2C involves a master component (Arduino) and slave components (SparkFun sensors). The master and slave streamline communication using two lines: SDA (Serial Data) and SCL (Serial Clock). The slave components each have a unique 7-bit address, which the master calls over SDA to initiate communication. Whichever slave device has its address called is then the only device on the I2C bus that can communicate over the SDA line. The SCL line acts as a clock to synchronize communication; bits are sent over SDA and are synchronized by HIGH/LOW pulses over SCL. Due to the master–slave relationship with specialized device addresses, I2C acts as a bus such that any compatible device can be added in parallel to other components on SDA and SCL.

For logging to the SD card, Arduino uses a communication protocol similar to I2C, known as SPI (Serial-Peripheral Interface). It also has the master–slave relationship between the Arduino and the component; SPI doesn't use specialized addresses for components. SPI instead uses four wires: MOSI (master out slave in, for communicating data from master to slave), MISO (master in slave out, communicating data from slave to master), SCK (Serial clock), and SS (slave select). SCK functions as a synchronization clock just like SCL. SS consists of individual wires to each slave device; rather than calling out multiple addresses on one wire, SPI uses individual wires for each device and sets them high to initiate communication. In a sense, SPI is a simpler format of I2C, which makes communication faster. The only catch is that it needs an individual SS wire for every device. For this project, though, SPI was only used for writing to our SD card, which was already hardwired on the Adafruit Data Logger Shield. **B.** Schematic Overview

Virtually all of the power connections to electrical components in this payload were soldered into the DAQ Arduino via a solderboard. Four male headers were soldered into either the Adafruit logger shield or a mini protoboard, which was placed into the SDA, SCL, GND, and AREF pins of the DAQ Arduino.[2] Though some components like the GPS chip couldn't be soldered due to the pre-soldered headers on Arduino pins such as RX1 and TX1, a vast majority of connections to the Arduino were soldered first to one of the two boards, and then to external components: this anchoring mechanism helped prevent wires from coming undone—either by

[2] Note: AREF was not used electrically; this connection was just for stability.

being pulled out during payload packing or by in-flight forces.

There are a couple of analog sub-circuits that were used repeatedly in this project: voltage dividers were used to read the voltage across our temperature thermistors, and transistors were used in conjunction with relays on three occasions to toggle certain components on and off with electrical isolation.

Note: CircuitLab, the software used for drawing schematics, has a component limit that restricts the entire Ballooney toons circuit from fitting into one sketch. Instead, all pins used on the DAQ Arduino have been labeled in the primary schematic below, and their various "breakout circuits" have been drawn out separately in the following figures.

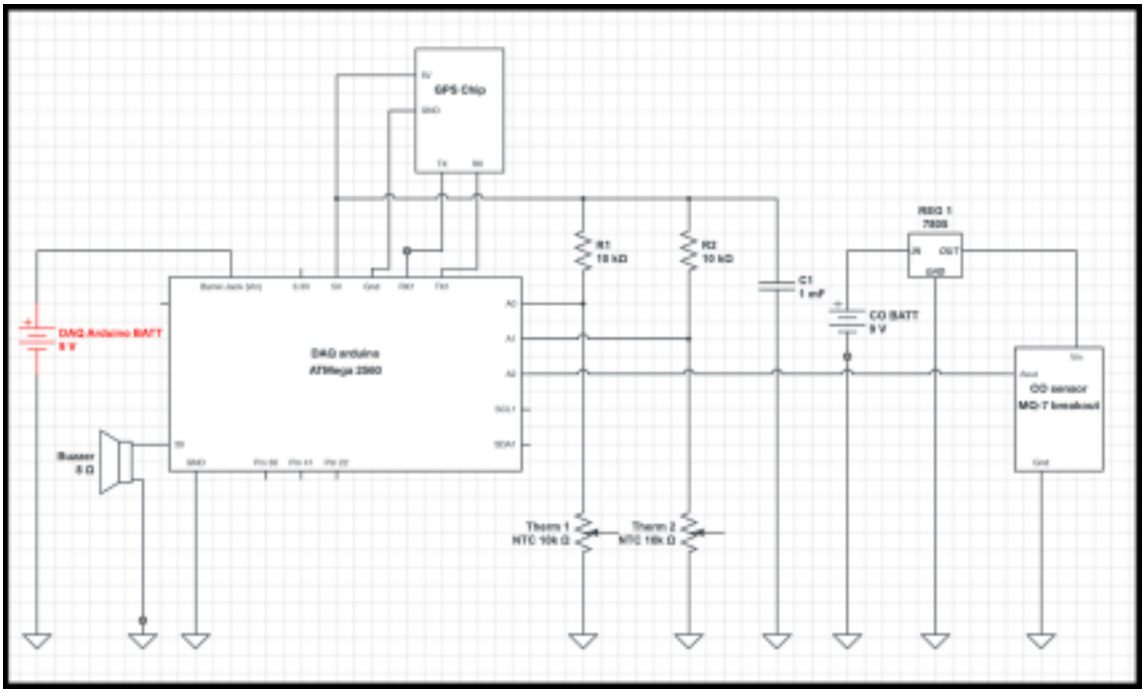

Figure 11: Schematic of Data Acquisition (DAQ) Arduino, Main Battery, GPS circuit, Temperature Circuits, Buzzer, Carbon monoxide independent project, and breakout pins for other components.

C. Data Acquisition Circuit Overview

The heart of data collection in this project is an Arduino ATMega 2560 equipped with the Adafruit Rev B Data Logger Shield. This shield has an SD card interface in addition to a PCF8523 Real Time Clock (RTC) chip onboard. These components were not drawn out as they are hardwired in the circuitry of the board and do not use any of the Arduino's available pins.[3]

The Arduino communicates with, gathers data from, and manages a slew of breakout circuits and components. Included in this schematic are the temperature sensors, the CO sensor, the buzzer, and the GPS chip. Due to the large load on the Arduino's 5V power rail, a 1000 µF bulk capacitor is used to smooth out voltage dips.

The temperature sensor circuits are seen to the right of the DAQ Arduino above and consist of two identical yet independent voltage dividers: a 10k ohm fixed resistor connects to the 5V power rail from the Arduino, and sits in series above the thermistor, which goes to ground. *R1* and *Therm1* make up the internal temperature circuit, while *R2* and *Therm2* make up the external temperature circuit. The node voltages of the divider circuits for internal and external temperature are read by analog pins A0 and A1, respectively.[4]

The CO detector consists of an MQ-7 gas sensor in a voltage divider on a breakout module. Due to a current draw of nearly an amp from this component, we decided to power it separately using another 9V lithium-ion battery, named *CO BATT* in Figure 11 above. However, the MQ-7 requires a supply voltage of 5V, so a 7805 5V linear voltage regulator was used to supply a clean 5V input to the CO sensor module from the 9V *CO BATT*. The MQ-7 breakout module outputs both a digital signal and an analog signal. The digital output is used as a trigger that becomes set HIGH upon reaching a chosen ppm, while the Analog output (Aout) give a traditional 0-1023 signal indicative of the node voltage in the sensor's divider; since we only wanted to calculate the resistance of the sensor, the digital output was not used and the analog output was wired to pin A2 to be read by the DAQ Arduino.

The buzzer uses very little current (less than a milliamp) to vibrate a small diaphragm thousands of times per second; this oscillation produces the resulting high-pitched sound wave. In our circuit, the buzzer was connected to digital pin 39, which would be set to HIGH by a state machine after three hours had passed since boot-up. The objective of the buzzer was to help us locate the payload once in close proximity.

[3] Aside from the SD interface, which uses pin 10 as the SS pin for the component.
[4] For information on how the temperature dividers are used to calculate a reading, refer to Section IX.

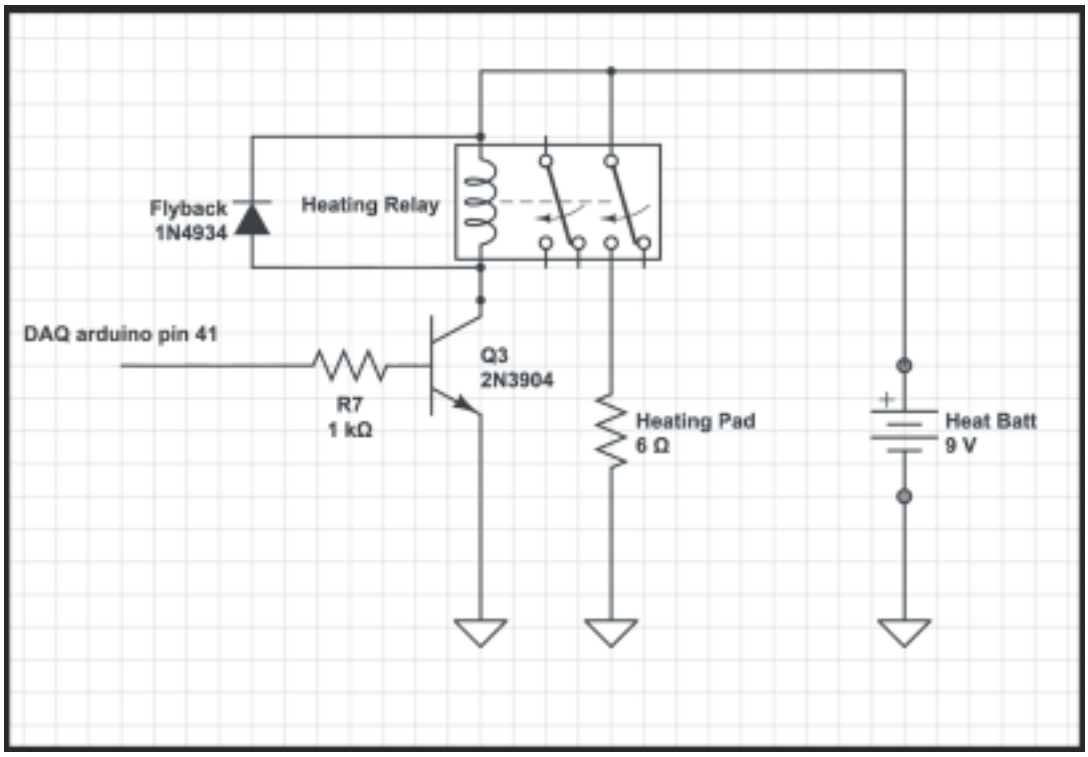

Figure 12: Heating Pad "Breakout" schematic

The heating pad used requires an amp of current. To be able to toggle it on and off with a digital Arduino pin, a transistor–relay circuit was used. The collector of an NPN transistor (labeled Q3) is tied to the bottom of the relay's coil, the emitter is wired to ground, and the base signal comes from Pin 41 on the DAQ Arduino. A 1k ohm resistor was used to keep the transistor from drawing excessive current from the digital pin in order to prevent damage to the microcontroller. The relay used is an HK-19F; its coil demands roughly 70 mA at 5V, and its connections can support up to 2A of current. To additionally limit power draw from the DAQ Arduino's battery, a separate lithium-ion 9V battery was used both for powering the relay and the heating pad. All grounds are common.[5]

[5] For a more detailed overview of the heating pad, see section IV, B.

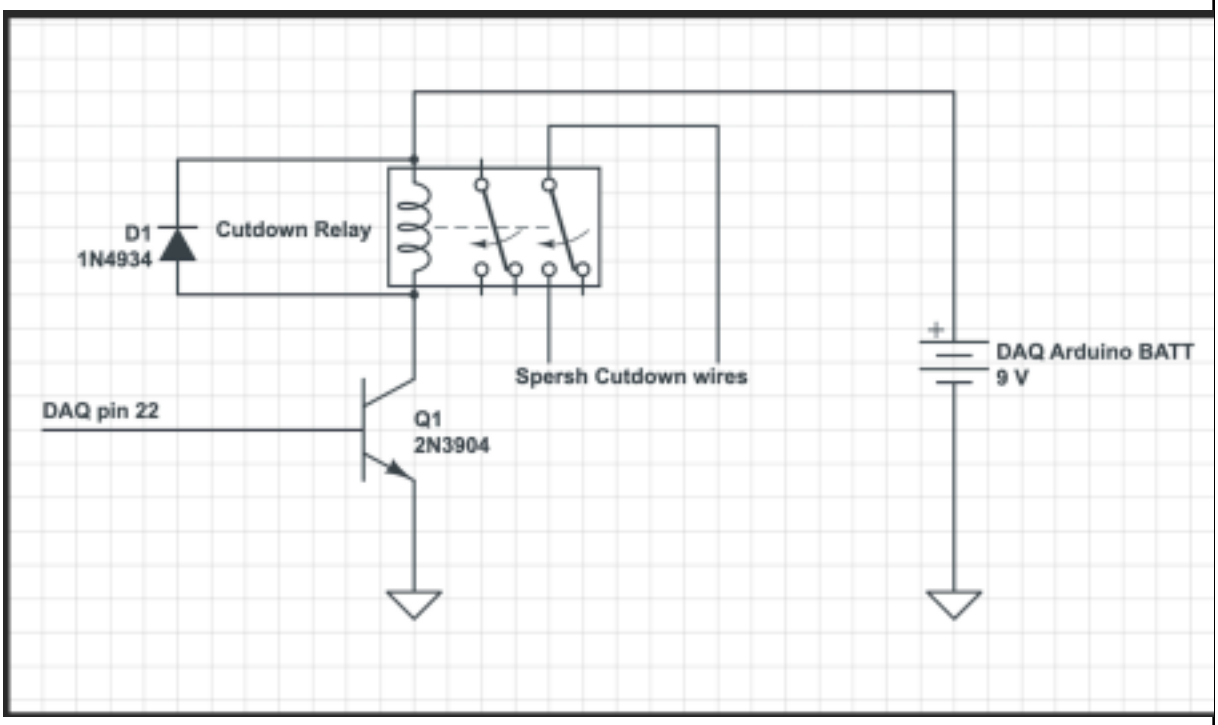

Figure 13: Cutdown Relay "breakout" Circuit

Similar to the heating pad circuit in Figure 10, the cutdown circuit above uses the transistor–relay circuitry to turn the HIGH/LOW signal of digital pin 22 into an electrically isolated on/off switch. When pin 22 is set HIGH, the Cutdown relay coil (a second HK-19F) is grounded by the NPN transistor (Q1) and the relay connections are closed; this allows the cutdown capacitors to discharge through the nichrome wire.[6]

[6] See Section VI for an overview of the cutdown mechanism.

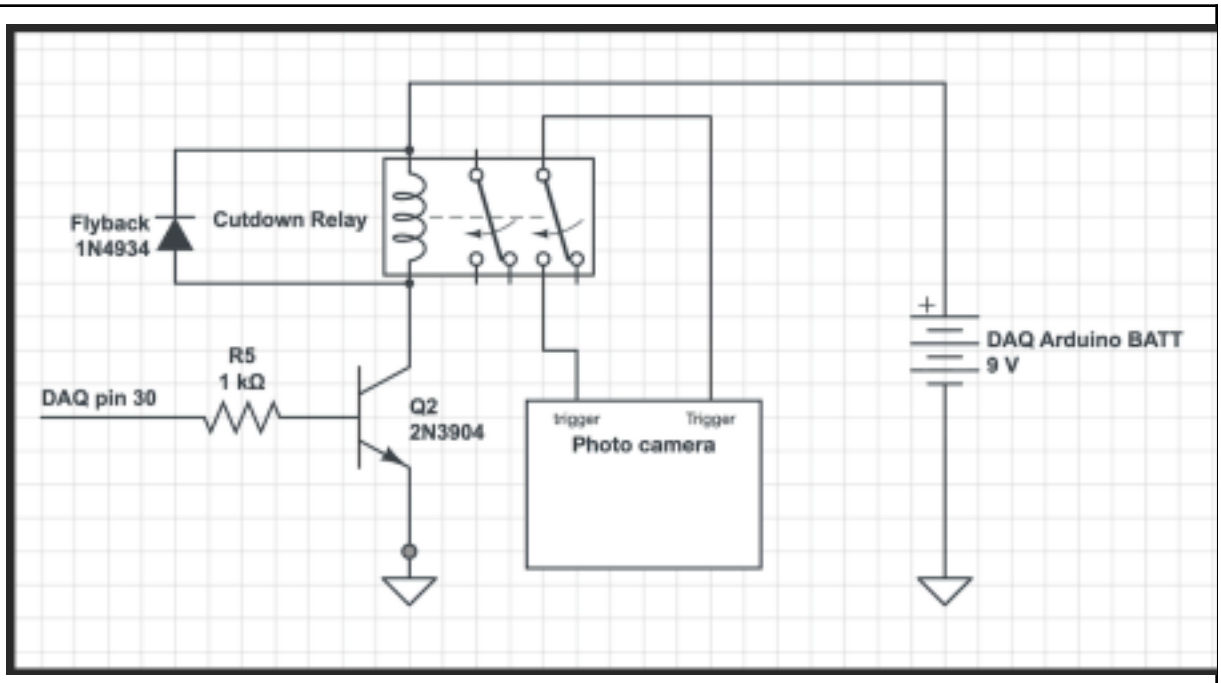

Figure 14: DAQ Arduino: Photo Trigger circuit

Once again, the Photo Trigger Circuit above leverages the electrical isolation and mechanical switch of a relay to trigger the camera's shutter button. To see a more in-depth explanation of the Camera system, see Section IX.

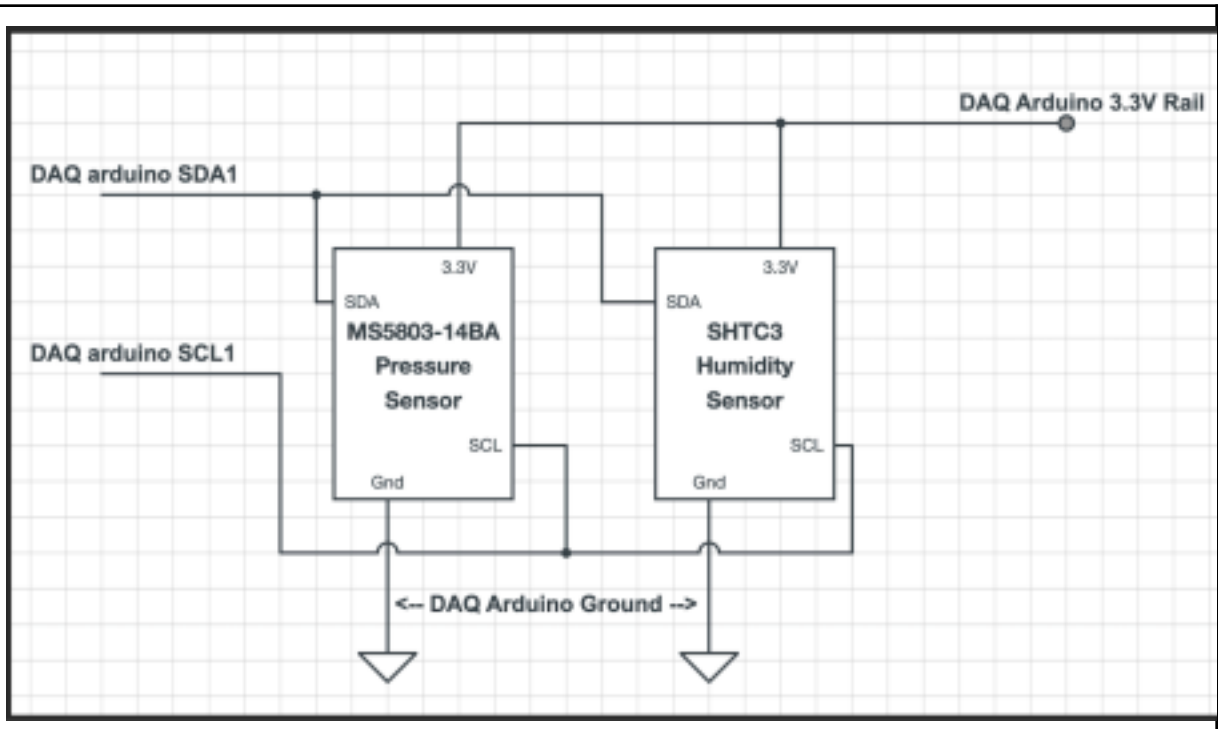

Figure 15: DAQ Arduino: Humidity and Pressure "Breakout" Circuit

The SparkFun Humidity and Pressure sensors communicate using the I2C bus described earlier in this section. These devices operate using the DAQ Arduino's 3.3V signal. While the pressure sensor was soldered into place on the DAQ Arduino, the humidity sensor was connected with a header wired to allow for quick assembly and placement of the sensor outside of the payload. The connections were made secure using solder on the male connections to increase the friction of the joint. The header wires from both the sensor and the DAQ Arduino were soldered.

D. Code and Logic Overview

The program for this project is fairly simple in theory, but proved to be complex in practice due to the integration of a number of separate components and their respective codes. To help keep the sketch organized, top and bottom comment headers were created for global, setup, loop, and functions to clearly group each component's separate code.[7]

Figure 16: Flowchart of logic

Figure 16 above depicts the logic of the entire program from boot up. The structure of the loop consists of updating values (reading GPS, reading thermistors, and calculating temperature, updating humidity and pressure, reading CO sensor) and checking time statements (such as cutdown and the logging interval of 30 seconds). All sensors were coded to update regularly during each loop, which not only allowed the Arduino to write to the SD card as quickly as possible when logging, but also captured the values of all sensors at the same timestamp. When the time condition for SDLog() is satisfied (every 30 seconds), one consecutive write would

[7] See Appendix I, lines 21 & 24.

occur, after which the program would return to the loop to keep servicing its sensors.

One issue that came up during the integration of the many different sub-codes was GPS interference. The Adafruit GPS hardware serial Parsing example sketch makes up the foundation of the GPS code in this program. It uses a library to read the raw NMEA sentences and parse them into local variables with quick access and easy storage. The GPS chip must be checked for incoming sentences very frequently; characters can get lost and garble the data. Due to many sensors needing to be updated each loop, a separate *GPSread()* function was created and called multiple times throughout the loop, including at the top. This function was implemented after the tethered launch; at first, the integrated code for the pressure circuit would interrupt GPS parsing, which prevented us from sending pressure up in our payload for the tethered launch.

VIII. Tracking Section

A Yagi Antenna is a directional antenna used to send or receive radio signals(My yagi is meant to receive) in one direction. It focuses in a forward direction rather than receiving signals in all directions. It works by using three types of elements: a driven element, which is the only part connected to the radio and generates the signal, a reflector placed behind it that bounces energy forward, and one or more directors placed in front that focus and compress the signal into a narrow beam. The combination of these elements creates constructive interference in the forward direction but destructive interference everywhere else; this means that the waves add together in one direction, amplifying the signal but in only the forward direction. I used tape measures for these three elements and connected them with pieces of PVC pipe. Tape measures are conductive(metal), flexible to make it more compact, and easy to find, while PVC is non-conductive and lightweight, so it keeps the structure stable without interfering with the signal. The element lengths were based on the wavelength using the relation $\lambda = c/f$, and for my frequency of 146.705 MHz, the wavelength is about 2.04 meters, so the driven element is about half a wavelength, which ends up being 18 inches on each side. My reflector is 5% longer than the driven element, and my director is about 5% shorter than the driven element. My reflector to drive spacing was about 0.3 of my wavelength, and my director to drive spacing was about 0.4 of my wavelength, which should have provided optimal gain. The reflector is slightly longer than the driven element, which makes it inductive and causes the current on it to lag, so it reflects

energy back to the driven element and stops the signal from behind. The directors are slightly shorter, which makes them capacitive and causes the current to lead, pulling the wave forward and focusing it. In line-of-sight tests, where there were no obstacles, the antenna reached a range of about 105 meters at 65db of attenuation; however, since VHF radio waves travel mostly in straight lines and are limited by Earth's curvature and terrain, the line of sight ends up being around 7,100 meters. Although the range while the balloon is high in the air is calculated to be around 100 miles. In obstruction tests with trees and buildings, the attenuated range dropped to about 45 meters because materials like wood, water, and concrete absorb and scatter radio waves, which weakens the signal. My antenna also has 41% more range than a normal handheld radio antenna. Radio signals in general work by converting sound or data into an electrical signal that modulates a carrier wave, which is then radiated by an antenna as an electromagnetic wave and received by another antenna, where it is demodulated back into usable information. Specifically, FM radio uses frequency modulation, where the frequency of the carrier wave changes based on the signal, which makes it more resistant to noise. The picoAPRS system uses this same idea but for digital data, taking GPS location information and encoding it into packets, then transmitting those packets into a network where they can be picked up in real time on the aprs.fi website. GPS itself works by receiving signals from multiple satellites that broadcast their position and precise time, allowing the receiver to calculate its distance from each satellite based on signal delay and use trilateration with at least four satellites to determine its exact latitude, longitude, and altitude, and this position data is then packaged by the picoAPRS and transmitted as radio packets that can be received and decoded by other stations. The radio transmission sends data packets by first organizing the information into a structured frame that includes a callsign, position data, and error-checking bits. Then, this digital packet is turned into two distinct audio frequencies (representing binary 0s and 1s), which are fed into the radio and used to modulate the carrier signal. This process is then reversed to reconstruct the original packet.

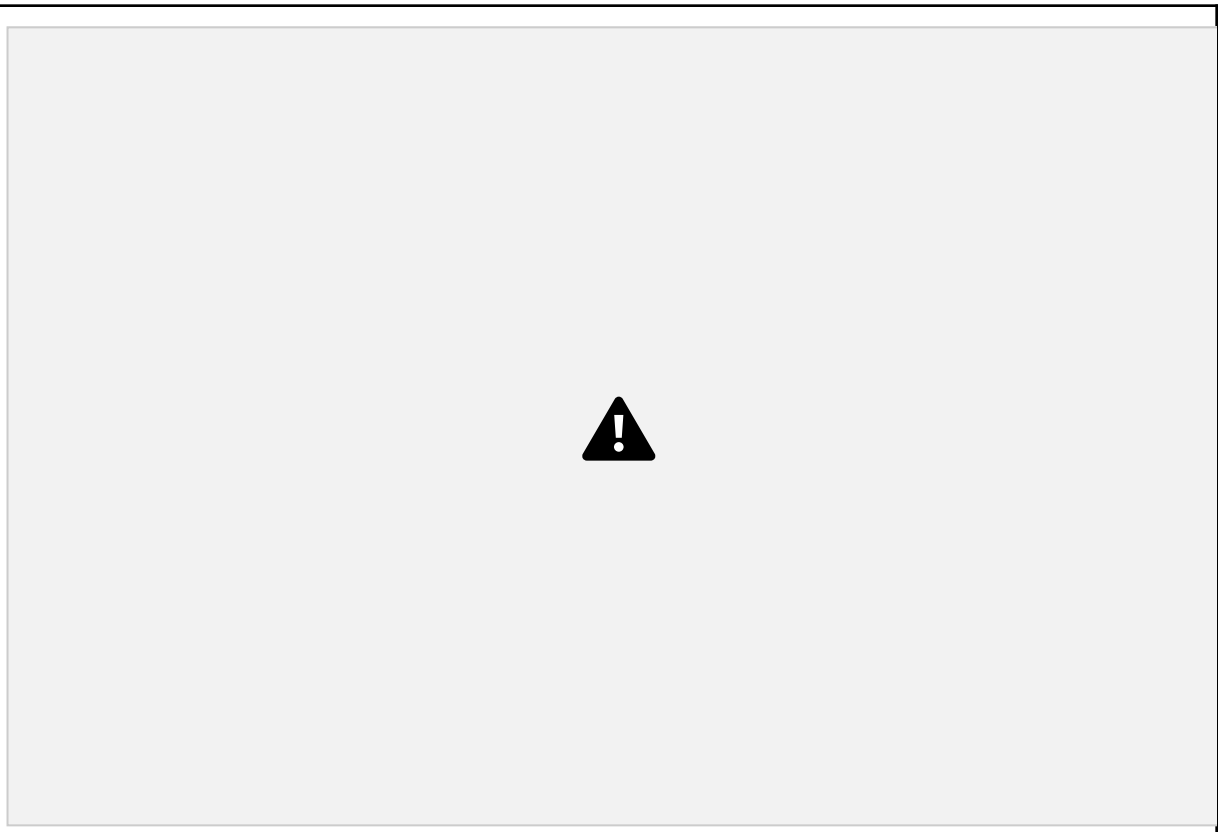

Figure 17: CAD drawing of Yagi Antenna

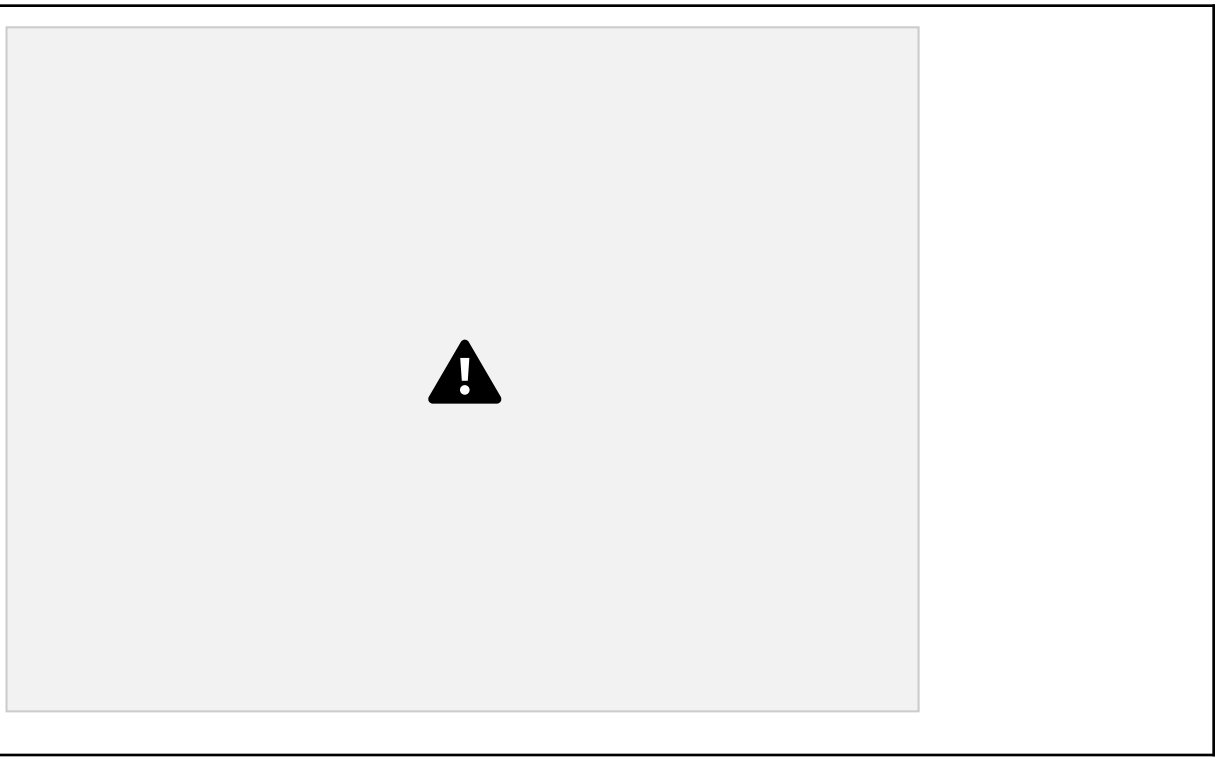

Figure 18: Drawing of Yagi Antenna

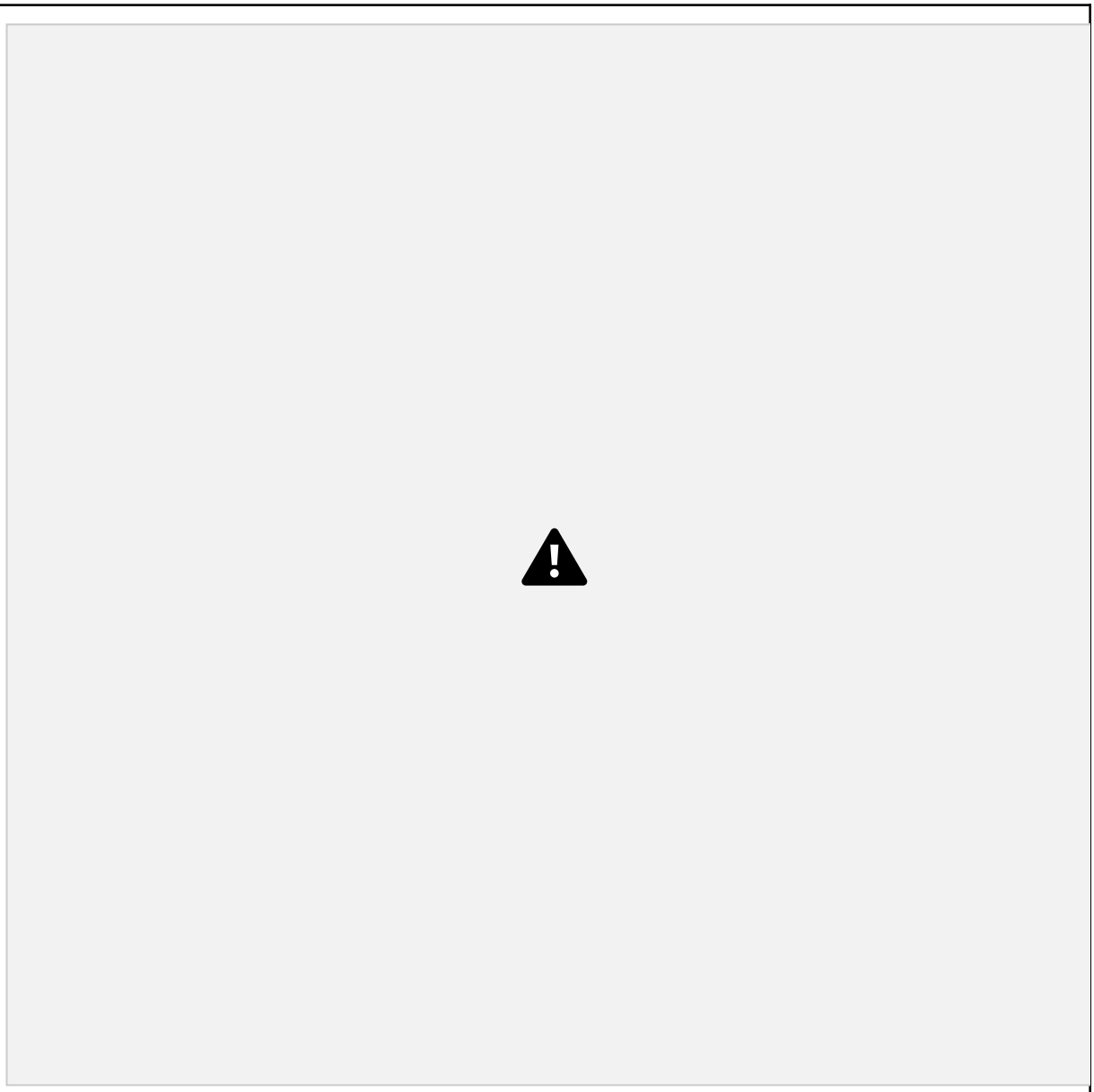

Figure 19: Flight Path and Simulation Path
Red: Flight path

Blue: Simulation path

IX. Camera Section

For the camera system, an AKASO V50 Pro was used to take photos periodically, and an AKASO V50 X was used to take one continuous video as a backup. A timing-based system was used to trigger the V50 Pro to take 2 photos every 2 minutes, as opposed to triggering it based on

altitude. To do this, the Arduino digital pin 4 was wired to the base of an NPN transistor, with the collector connected to 5V and the emitter connected to one side of a relay coil. The other side of the relay coil to ground and put a flyback diode in parallel with the relay coil. The camera housing was then disassembled to expose the button that triggers photos and the relay was wired in parallel with the button. With this setup, the Arduino sets pin D4 high every 2 minutes, which allows current to flow through the transistor, causing the relay coil to turn on. This completes the connection between either side of the button. This basically simulates a press of the button, but allows the button to be bypassed, such that it can be triggered electronically rather than physically. Every time the Arduino sets pin D4 high, it will make a note in the data with a timestamp and corresponding pressure data for use in the estimation of the Earth's radius.

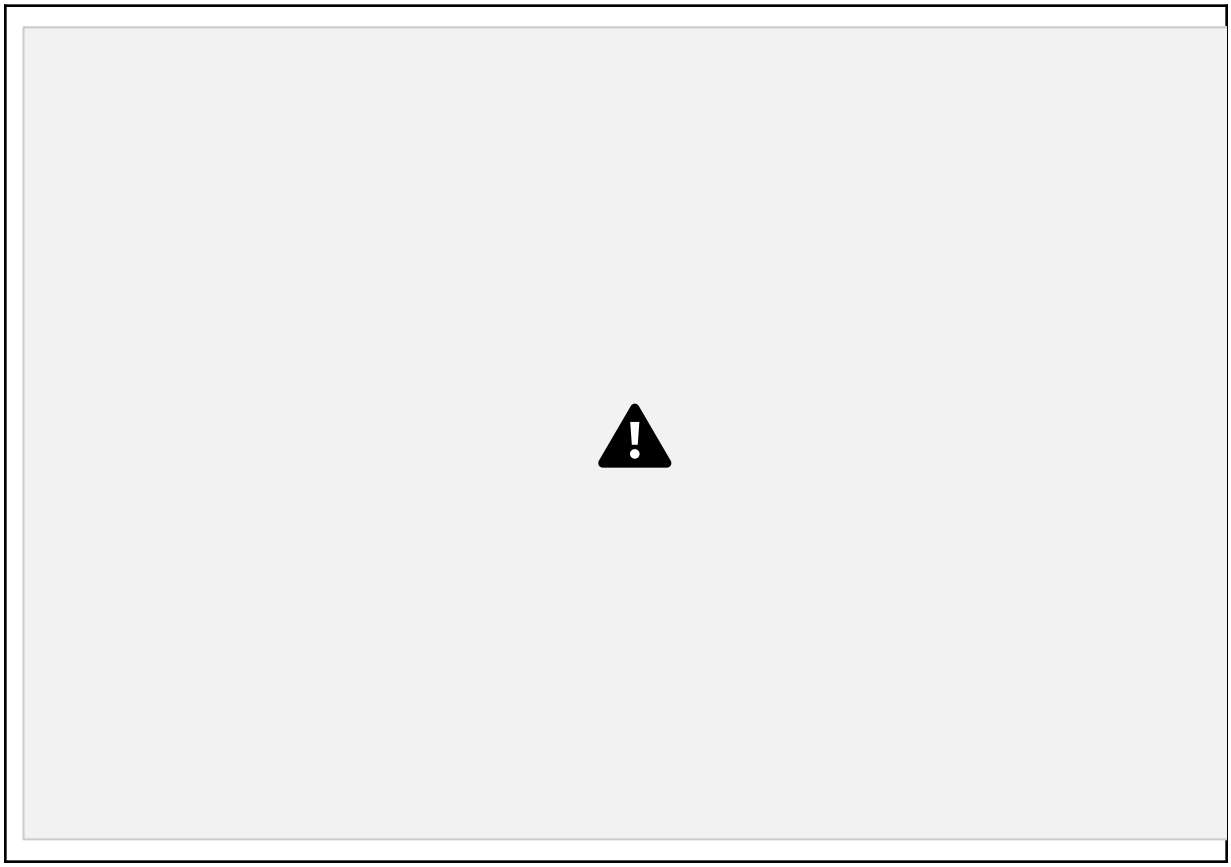

Figure 20: Camera Circuit Diagram

A. How Does a Digital Camera Work

Digital cameras are able to capture images through the use of specialized image sensors that are covered in a grid of highly light-sensitive tiles called photosites, where each photosite

makes up one pixel of the image. When the shutter is activated, a mechanical window is opened for a brief period of time, during which light is able to enter. The camera lens is used to bend incoming light rays, focusing them such that all the light is able to hit the sensor. As the light hits the sensor, each photosite generates an electrical charge that is proportional to the intensity of the light to which it is exposed. Then the image processor is able to analyze the electrical charge across each photosite to determine the brightness of each pixel.

To capture color, camera sensors usually have filters laid over the photosite that contain layers of red, green, and blue. These filters are made to replicate the sensitivity of the human eye in order to produce accurate coloration. Each photosite is only able to capture the color of the filter that is laid directly over it. As a result of this, the intensity of the brightness on each photosite is directly proportional to the intensity of that color of light. The image processor is able to use this data to determine the intensity of each color for each photosite, and combine them to obtain an RGB (red, green, blue) value that accurately reflects the color of each pixel.

B. Estimating Earth's Radius

The radius of the Earth was first estimated by Eratosthenes in 240 BCE. He was able to do this by identifying two landmarks that were a known distance away and analyzing the angles of their shadows and the position of the sun. Eratosthenes identified that the sun was shining directly at the bottom of a deep well. Through this, he was able to deduce that the sun was directly overhead. However, 800km to the north, a vertical stick cast a shadow at an angle of about 7.2˚. Since he knew that the sun was far enough away from the earth that the rays were basically parallel, he realized that the earth must be curved. Since 7.2˚ is about 1/50 of 360˚, the distance between the location of the well and the stick must be 1/50 of the circumference of the earth. Then he just divided 800 km by 1/50 (or multiplied by 50), and you get a circumference of 40,000 km.

X. Pressure and Temperature Sensor Calibration

For the temperature sensor, we used Negative Temperature Coefficient 10 kilohm thermistors (NTC-10K thermistors). These are analog temperature sensors that are made from a specific mixture of oxides, including manganese and copper. Unlike metals, these oxides have a band gap. A band gap is an energy zone between the valence band (where electrons are bound)

and the conduction band (where electrons flow freely). NTC refers to the inverse relationship between temperature and resistance. At low temperatures, very few electrons have enough thermal energy to jump this gap, so resistance is high. As the temperature rises, more electrons gain enough energy to cross the gap, and the resistance drops sharply. These thermistors have a base resistance of 10 kilohms at room temperature.

To read resistance values, each thermistor was put in series behind a 10k resistor, and the node voltage between the resistors was read by an analog pin on the Arduino. The raw ADC reading can then be converted into the actual voltage at the nodes, from which we can calculate the resistance of the thermistor using ohms law:

$V = ▯▯ \cdot ▯▯$

$V_{R1} = 5▯▯ - V_{node}$

The voltage across R1 is equal to the delta voltage between Vcc and the node of the divider. We can use the voltage across R1 to solve for the current through the divider: $I_{divider} = V_{R1} \div R_1$

In this case, we know the resistance of the fixed resistor R_1 and the voltage across R_1. We can now calculate the resistance of the thermistor using the node voltage and the current through the divider:

$R_{therm} = V_{node} I \div _{divider}$

A total of two thermistors (one internal, one external) were onboard the payload, each with their own identical voltage divider. 10k ohm resistors were used for the fixed resistors R_1 and R_2in the temperature voltage dividers (refer to Figure 9) however their true resistance values were measured and used for the temperature calculations in order to be as accurate as possible. The Steinhart-Hart equation $[1/▯▯ = ▯▯ + ▯▯ \cdot ▯▯▯▯(▯▯) + ▯▯ \cdot (▯▯▯▯(▯▯))^3]$ was used to convert raw resistance readings into temperature values. To calibrate our SH-H constants *A,B,* and *C,* we gathered three resistance values for each thermistor at appropriate temperatures which characterize each thermistor's unique resistance-temperature curve. The constants for the internal sensor were calculated using the resistance at 23º C, 10º C, and 0º C, while the constants for the external sensor were calculated using 10º C, 0º C, and -40º C. We strategically chose these temperatures to match the expected operating range.

For the pressure sensor, we used SparkFun MS5803-14BA Pressure Sensors. These pressure sensors utilize a piezoresistive micro-electro-mechanical systems (MEMS) sensing element. In simple terms, piezoresistivity is the property of certain materials to change their electrical resistance when mechanically stressed. So, when pressure is applied to the sensor, the diaphragm reacts, and the stress changes the electrical resistance, allowing us to measure the pressure at a given time.

To calibrate the pressure sensor, a pressure chamber as well as a Vernier pressure sensor to mark our “true” value, were utilized in tandem. We put our pressure sensor into the chamber with the Vernier one, and reduced the chamber to a near-vacuum. After that, we slowly released pressure, making discrete pressure increments in millibars that would be visible on a graph. After this process, we collected the Vernier “true” data (Figure 21) and our SparkFun pressure sensor (Figure 22) and plotted them together to determine a line of best fit (Figure 23).

Figure 21: Vernier “True” Pressure values

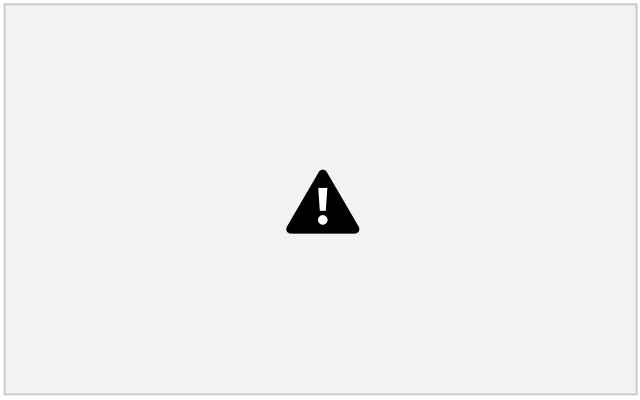

Figure 22: Our SparkFun Pressure values

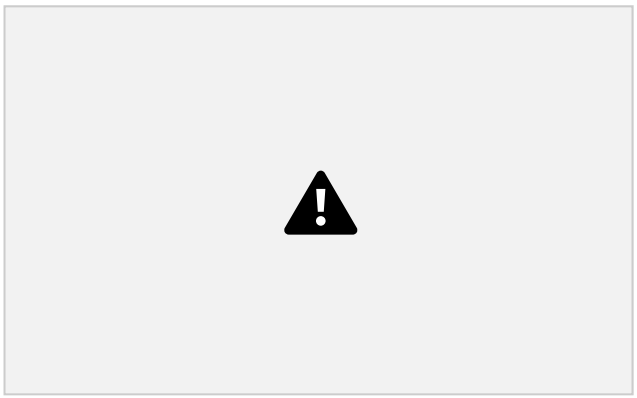

Figure 23: Vernier vs. SparkFun Pressure values

The line of best fit (.997x - 2.49R^2) provided us with a calibration equation that we could apply to the raw data in the code to get the true value. This equation is provided above in Figure 23. We then checked this calibration by running a second test, and utilizing a Vernier graphing analysis app to determine that our pressure was accurate to .04%.

XI. Experimental Results

PL: *Graph of pressure vs. altitude (overlaid with NASA and exponential model), graph of pressure vs. time, graph of GPS vs. time, and results.*

Figure 24 below is the graph of pressure vs. altitude with the NASA and exponential model overlaid, and Figure 25 is the same with the outliers removed. In addition, Figures 26 and 27 are the supporting graphs of pressure vs. time and GPS vs. time, respectively, with their error bars.

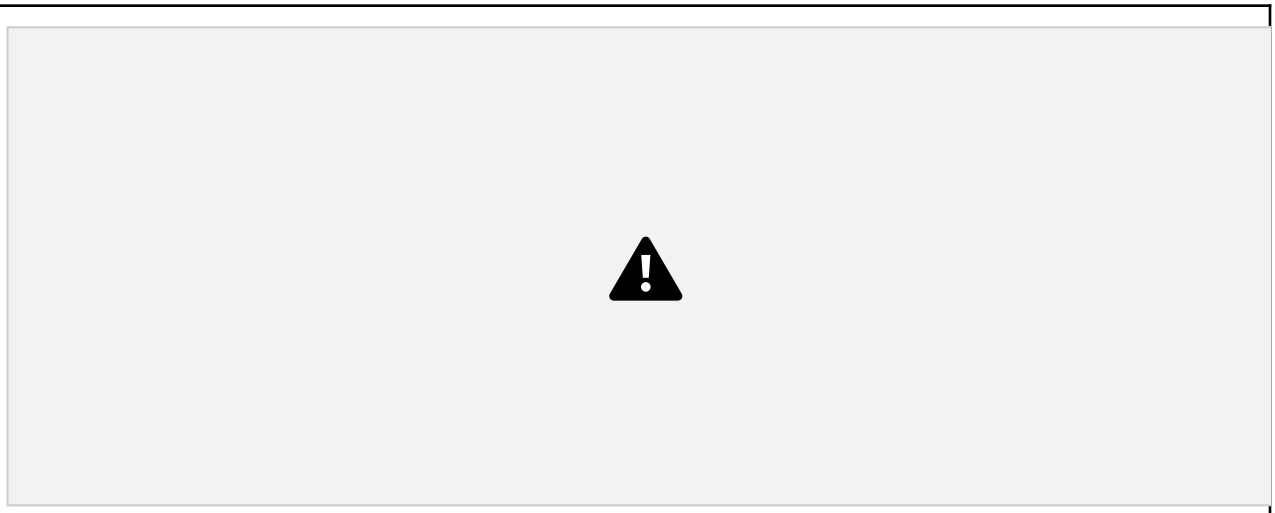

Figure 24: NASA Empirical Model, derived exponential model, and our data mapped onto the same chart graphing the expected vs. observed relationship between pressure and altitude. This chart includes the outlier data where the collected pressure would drop to exactly -186.519 KkPa, while the rest of the data conformed closely with the expected relationship.

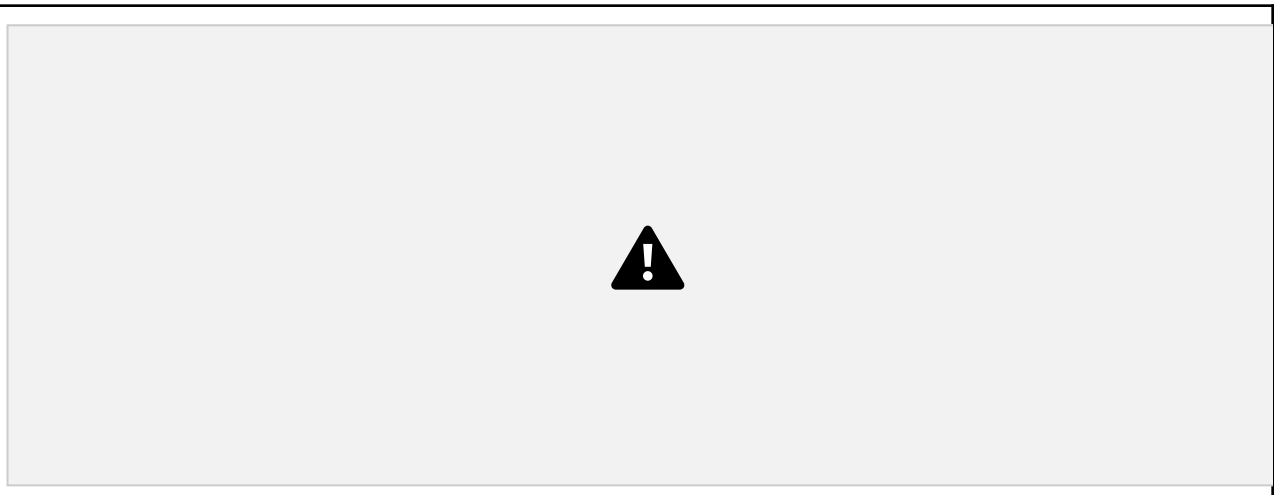

Figure 25: NASA Empirical Model, derived exponential model, and our data mapped onto the same chart graphing the expected vs. observed relationship between pressure and altitude. This chart excludes the outlier data points where
collected pressure would drop to exactly -186.519 kPa, resulting in a graph that aligns very closely with
NASA's Empirical Model of pressure vs. altitude.

In our collected data, our pressure sensor would periodically read exactly -186.519 kPa before returning back to the expected data. Because it was consistently the same value, we can

reason that this is the bottom threshold of the pressure sensor, and this reading occurs when there is either low power or a loose wire. Prior to launch, we tested the pressure sensor multiple times in temperatures down to -20˚C and for extended periods of time, so we can assume that this was not an issue with the code or inability to properly function in below-freezing conditions. Because we are able to assume that every data point of -186.519 kPa is an outlier due to a data collection issue, we can ignore these points in an analysis of our data and thus we will refer to Figure 25 which excludes all of the redundant data.

The NASA Empirical Model is a simplified, piecewise curve version of the U.S. Standard Atmosphere published in 1976, which represents the statistical average of atmospheric conditions under moderate, mid-latitude conditions. While the full Standard Atmosphere divides the atmosphere into seven layers, the NASA Empirical Model uses only the first three for its calculations, with its curve fit to observed data from weather balloons, radiosondes, rocket soundings, and satellite measurements. However, as you'll see below, that is still more accurate than the Exponential Model. Above ~10,000 m, the Exponential Model begins to overestimate pressure relative to the measured data, while the NASA Empirical Model continues to track the measurements well. This divergence occurs because the Exponential Model uses a single exponential scale height across the entire atmosphere, an assumption that only holds in the troposphere, where temperature decreases at a roughly constant lapse rate. Once the balloon crosses the tropopause, where the temperature profile changes, the single-scale-height approximation breaks down, and the model loses accuracy.

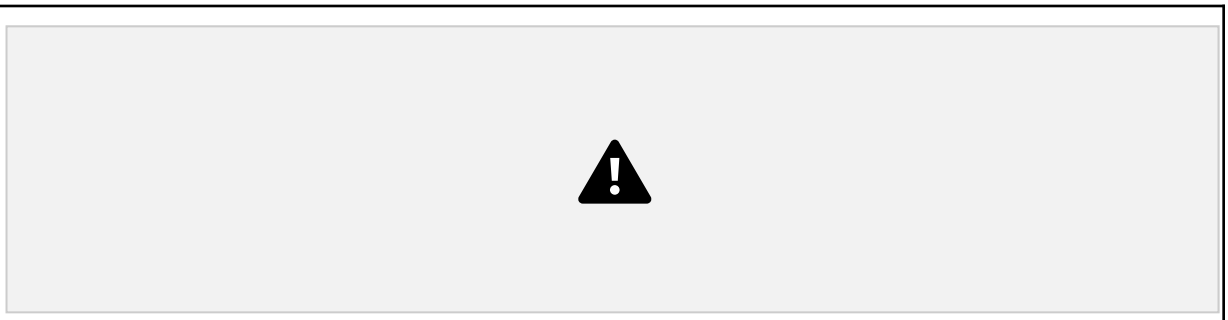

Figure 26: Atmospheric Pressure vs. Time.
The flat plateau between roughly 11:34 and 11:53 was when the balloon was still on the ground/ascending very slowly at the start of the launch. The sharp drop beginning around 11:53 marks the point of rapid ascent. After this, pressure decreases steadily from ~1,000 mbar at launch to near 0 mbar by the end of the flight, consistent with what is expected as the balloon ascends through the atmosphere.

Figure 27: GPS Altitude vs. Time.
From takeof at ~11:53, there is a steady increase in altitude over time. This shows that the balloon consistently ascended through the air until ~31,000 m. However, as shown in the graph, the data stops after it hits that peak, and we did not collect data upon descent. This will be discussed later.

SE: *CO Concentration vs. Altitude*

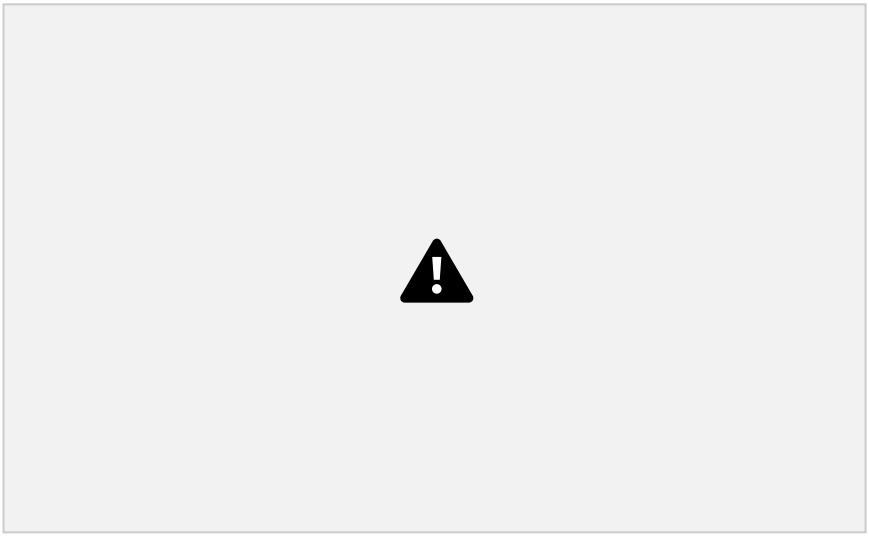

Figure 28: graph of carbon monoxide sensor resistance vs. altitude

Carbon Monoxide is a colorless, odorless gas. Trace amounts exist naturally in our atmosphere as a result of natural events such as wildfires or volcanic eruptions. The primary sources of CO live in the troposphere: combustion engines, the burning of fossil fuels, and industrial plants all release CO as an exhaust gas. Carbon monoxide has a weight of 28 g/mol, making it equal to nitrogen and slightly heavier than oxygen at 32 g/mol. Theory would predict that as altitude increases, the concentration of CO decreases since the sources of CO live in the low troposphere and the heavier weight of the gas would keep it from diffusing much higher in

the atmosphere.

For this experiment, I used an MQ-7 gas sensor breakout module. The breakout board consists of a voltage divider with a 10k load resistor $R_{L,}$ with the MQ-7 sensor in series behind it. In higher concentrations of CO, the resistance of the MQ-7 sensor will decrease—it is inversely proportional to CO concentration. The sensor works by running current through a metal oxide surface. In normal air (very low CO ppm), oxygen ions occupy the surface, which restricts current. When exposed to CO, the oxygen ions react to form CO_2, which frees up the sensor material and allows more current to flow, thus decreasing the resistance. The metal oxide material must be warm in order for the oxygen ions to react with CO. The sensor has a built-in heater to keep the sensor active, and this causes the sensor to draw 230 mA of current. To give accurate readings, the heater is intended to work in a high-low duty cycle. During the high cycle (when 5V is supplied) the heater is at its hottest; it burns off any accumulated CO_2 and also reestablishes the oxygen ion layer on the sensor element for future readings. During the low cycle (around 1.5V supplied) the sensor is in a state warm enough to react with CO such that the resistance gives an accurate reading relative to CO ppm.

Figure 28 above depicts the graph of the MQ-7 sensor resistance over altitude. Unfortunately, the breakout module I implemented was not capable of the heater duty cycle, which means all readings were taken at 5V. This causes the resistance readings to be largely misrepresentative of CO ppm. However, the general trend of the sensor's resistance as altitude increases can reveal the trend of CO ppm. Figure 26 shows a clear increase in resistance up to 10,000 meters, suggesting a decrease in CO concentration that aligns with the theory above. After 10,000 meters, the sensor resistance begins to decrease which is inconsistent with our predictions. This decrease is explained by the independent 9V battery being fully discharged by this point in the flight. We used a new 9V lithium ion battery on the day of the launch, which would've been able to supply around 1 Ah of charge. The entire CO circuit—which included a 7805 linear regulator in addition to the MQ-7 module—was powered long before the launch due to delays during packing; this would've prematurely drained the battery for the mission. The colder temperatures at altitude could've caused the battery's voltage to sag, further affecting output.

ED: *Inside/Outside Temperature vs. Altitude and results.*

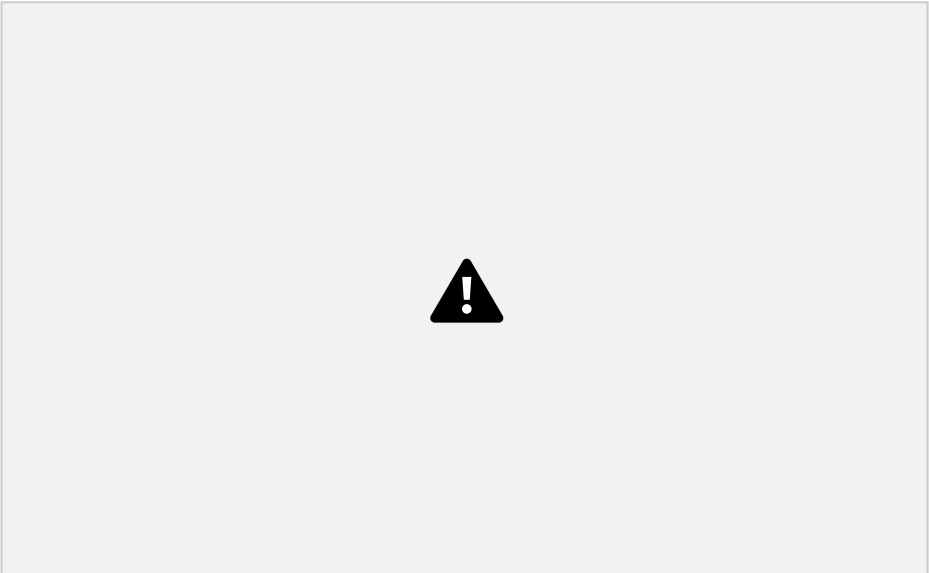

Figure 29: graph of temperature vs. altitude, internal and external

Figure 29 depicts our measured internal and external temperature data with respect to altitude. The external temperature data behaves as atmospheric science predicts, beginning around 28–35˚C at the surface and decreasing steadily through the troposphere at roughly the expected lapse rate, crossing 0˚C around 4,000–5,000 m and reaching approximately −35˚C near 12,000–15,000 m. Above around 15,000–20,000 m the external temperature stabilizes in the −20 to −30˚C range, consistent with entering the lower stratosphere where temperature stops decreasing and holds relatively constant. The increase in temperature after entering the stratosphere is also expected, as the change in temperature inverts with each new layer of the atmosphere.

The internal temperature reflects the combined effect of the heating circuit, the insulation of the styrofoam enclosure, and the heat generated by the tightly packed internal components. The payload box was well insulated by the styrofoam, which slowed heat loss to the surrounding atmosphere significantly, and the dense packing of electronics and batteries inside the enclosure contributed additional heat. From around 9,000 m onward the internal temperature displays a clear oscillating pattern between roughly 0˚C and +10˚C, which is a signal that the heating pad was switching on and off to remain in the calibrated equilibrium. The heater, combined with the insulation and component heat, successfully maintained a roughly 25–35˚C temperature

difference between the interior and exterior throughout the majority of the flight, demonstrating that the thermal management system performed as intended across the full ascent to 30,000 m.

The sharp spikes in the data can be attributed to measurement errors rather than real data, because both internal and external sensors spiked at around the same time despite being in different environments and exposed to different sources of heat.

RE: *Air Resistance data on descent, graph of Speed vs. Altitude, and results.* At the start, speed is moderate and somewhat scattered. This is likely due to low altitude winds, especially because we launched it on a windy day. As altitude increases up to 11,000ft, the speed increases linearly up to about 60-70 mph, this tells you the payload enters stronger, more stable wind currents. However, there is a noticeable drop after this, meaning that the payload entered a slower layer of wind in the atmosphere. The peak in mph likely refers to a jet stream or a high wind layer, while the troughs in speed are likely due to slower, calmer winds. It is important to note that we do not have GPS data significantly above 30,000 ft, which is very similar to our picoAPRS hits.

Figure 30: APRS graph
X-axis: Altitude (ft)
Y-axis: Speed (mph)

Air Resistance

Air resistance was calculated using the effective drag area equation $CdA = 2mg / (\rho \times v^2)$, derived from the condition that drag force equals gravitational force at terminal velocity. Air density was calculated at each altitude using measured pressure and temperature values from the dataset where pressure was converted to Pascals and temperature to Kelvin. Velocity was taken from recorded speed data during descent. As altitude increased, air density decreased, which reduced drag and allowed the payload to move faster; as the payload descended, increasing air density caused drag to increase and speed to decrease. The effective drag area remained

relatively consistent across the descent, confirming that drag depends primarily on air density and velocity rather than changing physical properties of the payload.

Results:

Cd= 0.58

CdA= 0.83 m^2

EE: *Graph of Humidity vs. Altitude and results.*

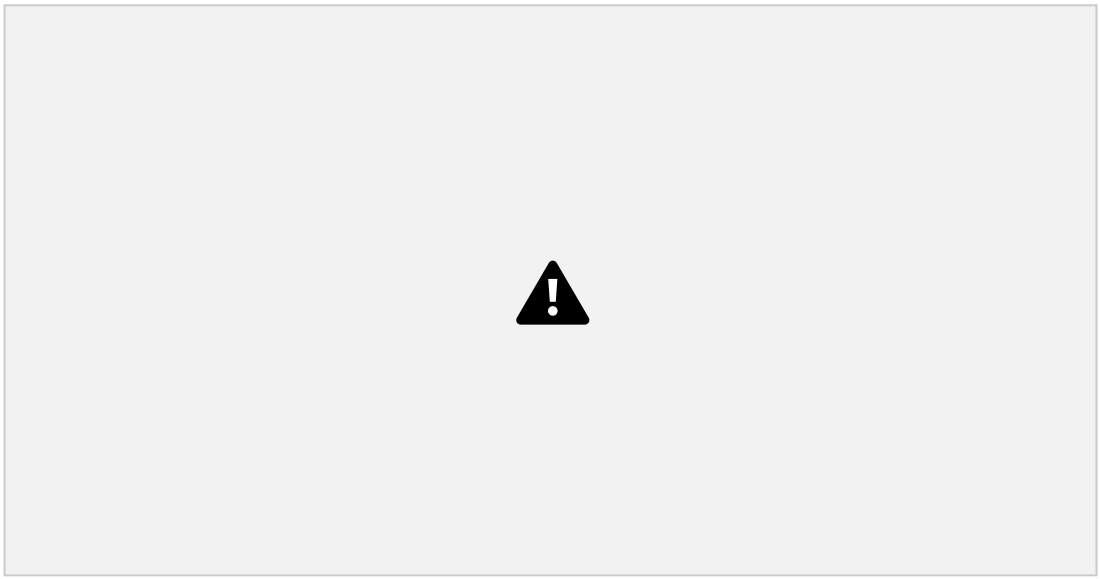

Figure 31: graph of humidity vs altitude

The humidity versus altitude graph was obtained by reading relative humidity from the SparkFun SHTC3 breakout humidity sensor (operating range of 0-100% with accuracy at plus or minus 2% relative humidity), and altitude data from GPS chip. The graph of the Ballooney Toones data mapped closely to the expected graph should have looked like.

Relative humidity doesn't just decrease monotonically. By the definition of humidity, we know that the humidity vs altitude graph shouldn't be linear. Relative humidity is the ratio of the actual water vapor pressure in the air to the saturation vapor pressure at the temperature that the sensor reads. Saturation vapor pressure is the maximum amount of water vapor the air can support before condensation starts, and saturation vapor pressure is controlled by temperature through the Clausius-Clapeyron equation which says that for approximately every 10 degrees

Celsius in temperature, the saturation vapor pressure roughly *doubles.* This means that as the temperature drops with altitude, the maximum possible moisture content of the air drops exponentially.

So we can observe what the expected, ideal graph should look like at different altitude ranges. From zero to 500 meters (the planetary boundary layer), we would expect relative humidity to trend upward, but slowly. This is a turbulent layer of our atmosphere that has convection currents moving everything around due to sunlight heating the ground. Due to the "well mixed" nature of the air, relative humidity would tend to stay fairly constant, or increase slightly with altitude. We might see a slight increase because even though the actual moisture content of the air stays constant (because of the turbulence), temperature is dropping, so the saturation vapor pressure drops, thus increasing the relative humidity reading. At this point in the launch, our data readings were fairly chaotic, fluctuating between 24% and 44%, which we attribute to the sensor not stabilizing, and being affected by nearby heat sources (electronics, direct sunlight, body heat from the group, etc) as the payload was not that high off the ground at this point in time (the 43.9% reading occurred at a measly 146 meters off the ground). Then, between 100 and 235 meters, our data starts to make more sense where the relative humidity values were 25-29%, and the altitude data was increasing rapidly. The rapid altitude changes with relatively stable humidity suggested that the sensor was finally reading the actual boundary layer air, and was fully stabilized.

Then, from 500-1500 meters (the upper boundary layer of the atmosphere), the air is still being mixed by convection but less vigorously. Temperature continues to drop, and saturation vapor pressure continues to drop, leading to an increasing humidity reading. At this point, we would expect a fairly steep increase in humidity. But humidity might also decrease due to subsidence. In regions with high pressure systems, air in the free troposphere is slowly sinking. When air sinks, it moves into regions of higher pressure and gets compressed. Compression without adding heat causes the air to warm adiabatically, dramatically lowering humidity since an increase in saturation vapor pressure occurs (since warmer air can hold more moisture) without adding any actual water vapor. This is also why our data showed a dramatic decrease in humidity between 1400 and 2300 meters, crashing from 31% to 8.5%.

From 1500-3500 meters, subsidence inversions may occur, where temperature actually

increases briefly with altitude. The inversion is created by subsidence, and because the sinking
air warms enough above it the temperature actually increases with height, making a stable “lid” form in high-pressure systems.

From 3000-6000 meters (mid troposphere), the absolute amount of water vapor is declining exponentially with altitude. Temperature also declines. Relative humidity in this region depends on whether water vapor or temperature is dropping faster, typically having relative humidity declining gradually. But the decline is hardly smooth because the payload encounters layers of different air masses stacked on top of each other. This is why our data showed an oscillation between 17% and 23% through this region.

From 6000-10,000 meters (upper troposphere), relative humidity continues to drop because the amount of water vapor and saturation vapor pressure are both shrinking. In terms of how our data compares to what we would expect, this is where the data gets somewhat suspicious. The humidity readings flatten out dramatically, hovering between roughly 13 and 14% for roughly 3500 of altitude gain, when in reality, humidity should still be steadily declining. At this point, though, the sensor is operating at temperatures between -30 to -50 degrees Celsius, and the capacitive polymer element in the sensor is becoming sluggish. For context, how the sensor works is that inside the SHTC3 chip, a capacitive polymer element absorbs water vapor from the surrounding air, and the resulting change in capacitance is converted into a percentage of relative humidity. At such low temperatures, we believe that the sensor is starting to have difficulty keeping up with the balloon’s ascent rate, and can’t adjust to the rapidly thinning, cooling air fast enough. The end result is that the sensor basically “stalls”, reporting the same value multiple times.

From 10,000-12,000 meters (tropopause), temperature stops declining (or even increases since we’re entering the stratosphere where ozone absorption heats the air). Regardless, compared to the earlier categories, air rising through the troposphere is ridiculously cold by this altitude that virtually all of its remaining moisture has frozen out and fallen as ice crystals. We would expect relative humidity to be in the 10-20 percent range, and falling. This was ultimately the case, and reflected in our data and graphs above. There was a little bit of oscillation as humidity continued to drop in this altitude range in our data, but overwhelmingly, the downward

trend was clear.

At 12,000-20,000 meters (lower stratosphere), the air is extremely dry, saturation vapor pressure is extremely low, and relative humidity should continue to decline through this region.

However, it does not drop to zero yet because the oxidation of methane in the stratosphere produces water vapor. $CH_4 + 2O_2$ produces CO_2 and $2H_2O$. It's a small amount of water, but ensures that there is some humidity reading here. This was also reflected in our data as the humidity reading continued to decline, but remained non-zero.

At 20,000-31,000 meters (mid stratosphere), the atmosphere is about 50 times thinner than at the surface. Temperatures here should be around -50 to -40 degrees celsius, with the water vapor mixing ratio staying roughly constant (compared to the last region) at 3-5 parts per million, meaning that relative humidity is *extremely low,* and hardly changing. At this point, the SHTC3, an over the counter humidity sensor, would struggle to make meaningful measurements, and the numbers it reports (3-6% in our case) would most likely just be the sensor's electronic noise being interpreted as a humidity value, not an actual measurement.

If we had a perfect sensor, the expected relative humidity profile would look something like the following: starting at 40-50% humidity at the surface, rising to 60-80% by 1500 m, dropping sharply if the payload hit a subsidence inversion (down to 20-30%), declining gradually through the free troposphere (15-25% around 5000-7000 m), continuing to decline through the upper troposphere (10-15% around 10000 m), crossing into the tropopause around 12,000 m (5-10%), and then becoming virtually unmeasurable (under 1-2%) through the stratosphere above 15000 meters. Ultimately, given our hardware, our graph matches fairly well with the expected graph of humidity versus altitude.

CE: *Calculation of the curvature of the Earth, supporting photos, estimations, assumptions, calculations, and results.*

My method of estimation for the radius of the earth using photos taken at high altitude requires the FOV of the camera, the dimensions of the photo, and the angle of drop from the center of the horizon to the point where the horizon meets the edge of the photo. When a photo is taken from a high enough height, the curvature of the Earth is visible as a subtle arc across the frame of the camera. The points where the horizon meets the edge of the camera's frame are

identified and assigned pixel coordinates (x,y). When a straight line is drawn between these points, the horizon bulges upward in the center, demonstrating the arc. The vertical distance between the middle of the arc and the line is called the sagitta. For any arc of a circle, the radius that it corresponds to can be calculated using the length of the line (L) and the height of the bulge (S):

$$R = L^2 / 2S$$

From the image, these measurements can be directly extracted using the number of pixels, where L_{px} is the distance in pixels from the centre of the image to the edge of the arc, and S_{px} is the distance in pixels from the middle of the line connecting the two ends of the arc, up to the highest point of the arc. The FOV of the camera can then be used in order to convert L_{px} and S_{px} into real dimensions. In order to do this, you need to convert the diagonal field of view (D_{FOV}) into horizontal field of view (H_{FOV}) and vertical field of view (V_{FOV}) using the sensor dimensions in pixels (W × H) and the diagonal size of the sensor in pixels (d_{px}),:

$$d_{px} = \sqrt{(W^2 + H^2)}$$

$$H_{FOV} = 2 \times \arctan(\tan(D_{FOV} / 2) \times (W / d_{px}))$$

$$V_{FOV} = 2 \times \arctan(\tan(D_{FOV} / 2) \times (H / d_{px}))$$

For the AKASO V50 Pro at a 140˚ diagonal FOV setting with a 3840×2880 pixel sensor:

$d_{px} = \sqrt{(3840^2 + 2880^2)} = \sqrt{23,040,000} = 4,800$ px

$H_{FOV} = 2 \times \arctan(\tan(70^\circ) \times (3840 / 4800))$

$= 2 \times \arctan(2.747 \times 0.800)$

$= 2 \times \arctan(2.198)$

$= 2 \times 65.5^\circ$

$= 131^\circ$

$V_{FOV} = 2 \times \arctan(\tan(70^\circ) \times (2880 / 4800))$

$= 2 \times \arctan(2.747 \times 0.600)$

$= 2 \times \arctan(1.648)$

$= 2 \times 58.7^\circ$

$= 117^\circ$

Next, the FOVs need to be adjusted to account for the cropping of the image. For a crop that retains W_c pixels of the original W wide image, the new horizontal field of view is:

$$H_{FOV\ Cropped} = 2 \times \arctan(\tan(H_{FOV} / 2) \times (W_c / W))$$

Then three points are marked are marked on the arc of the horizon, including the left end of the arc (x_1, y_1), the right end (x_2, y_2), and the midpoint of the arc at horizontal position $x_m = (x_1 + x_2) / 2$, with vertical position y_m. These are used to convert the pixel measurements into angles:

$$L_{px} = (x_2 - x_1) / 2$$
$$S_{px} = ((y_1 + y_2) / 2) - y_m$$
$$\alpha = (L_{px} / W_c) \times (H_{FOV\ Cropped} / 2)$$
$$S_{angle} = (S_{px} / H_c) \times V_{FOV\ Cropped}$$

Finally the angles are converted to physical distances and the radius of the earth is calculated. (d) is the actual distance from the camera for an altitude (h):

$$d = \sqrt{(2Rh)}$$
$$L = d \times \sin(\alpha) \approx d \times \alpha$$
$$S = d \times \tan(S_{angle}) \approx d \times S_{angle}$$

	Photo 1	Photo 2	Photo 3
Time	13:42:26	13:34:20	13:20:07
Altitude (m)	30,708	28,047	23,900
Dip angle (°)	1.463	1.385	1.358
Sagitta s (in)	0.064	0.06	0.059
Line length 2L (in)	5.003	5	5
Half line length L (in)	2.502	2.5	2.5

Figure 32: Data and measurements corresponding with each photo

Camera model	Image quality	Resolution	Diagonal FOV	Horizontal FOV (calc)	Vertical FOV (calc)	Distortion correction	Aspect ratio	Crop applied	Horizontal FOV after crop	Vertical FOV after crop	Working image (px)	Working image (in)	Pixel scale	Angular scale
AKASO V50 Pro	Fine	4K (3840 × 2880)	140°	131°	117°	Enabled (in-firmware)	4:3	3840 to 2880 px wide	117°	117°	2880 × 2880	5 × 5 inches	576 px/in	0.0406° /pixel

Figure 33: Relevant camera specs

Using this data for 3 different photos, I was able to calculate 2,867 km, 2,918 km and 2,738 km for the radius of the earth. When compared to the earth's real radius, the above calculations have a percent error of 55.0%, 54.2% and 57.0% respectively, which is fairly far off. However, each of the calculations are fairly close to each other, meaning that the source of the error is not in the calculations or the method, but in the camera itself. Specifically, the distortion caused by the camera's lens. Digital cameras use lenses that focus light onto the sensors, but these lenses also expand the FOV of the camera by allowing it to collect light from wider angles. The AKASO V50 Pro uses a wide angle lens to achieve its 140° diagonal FOV. Unfortunately, this also has the side effect of distorting the image. This type of distortion is called barrel distortion, and it is where straight lines that do not pass through the center of the image appear to bow outward toward the edges of the frame. Although this is not super noticeable over short distances on the ground, when photographing the horizon from 100,000 feet, it can cause the arc of the horizon to bow significantly. This makes the ends of the horizon appear to curve downward more sharply than they actually do. This increases the distance between the center to the lines connecting the ends of the horizon and the center point of the horizon, leading to an overestimation of S. This overestimation leads to an underestimation of the radius, which is consistent with the findings.

Formulas Used:

Diagonal to horizontal field of view conversion:

$$d_{px} = \sqrt{(W^2 + H^2)}$$

$$H_{FOV} = 2 \times \arctan(\tan(D_{FOV} / 2) \times (W / d_{px}))$$

$$V_{FOV} = 2 \times \arctan(\tan(D_{FOV} / 2) \times (H / d_{px}))$$

Adjusting field of view:

$$H_{FOV\ cropped} = 2 \times \arctan(\tan(H_{FOV} / 2) \times (W_c / W))$$

Pixel to angle conversion:

$$\alpha = (L_{px} / W_c) \times (H_{FOV\ cropped} / 2)$$

$$S_{angle} = (S_{px} / H_c) \times V_{FOV\ cropped}$$

Horizon distance:

$$d = \sqrt{(2Rh)}$$

Physical half line length and sagitta:

$$L = d \times \sin(\alpha)$$

$$S = d \times \tan(S_{angle})$$

Earth's radius:

$$R = (L^2 + S^2) / 2S$$

Percent error:

$$\%\ error = | (R_{calculated} - R_{accepted}) / R_{accepted} | \times 100$$

where $R_{accepted} = 6{,}371$ km.

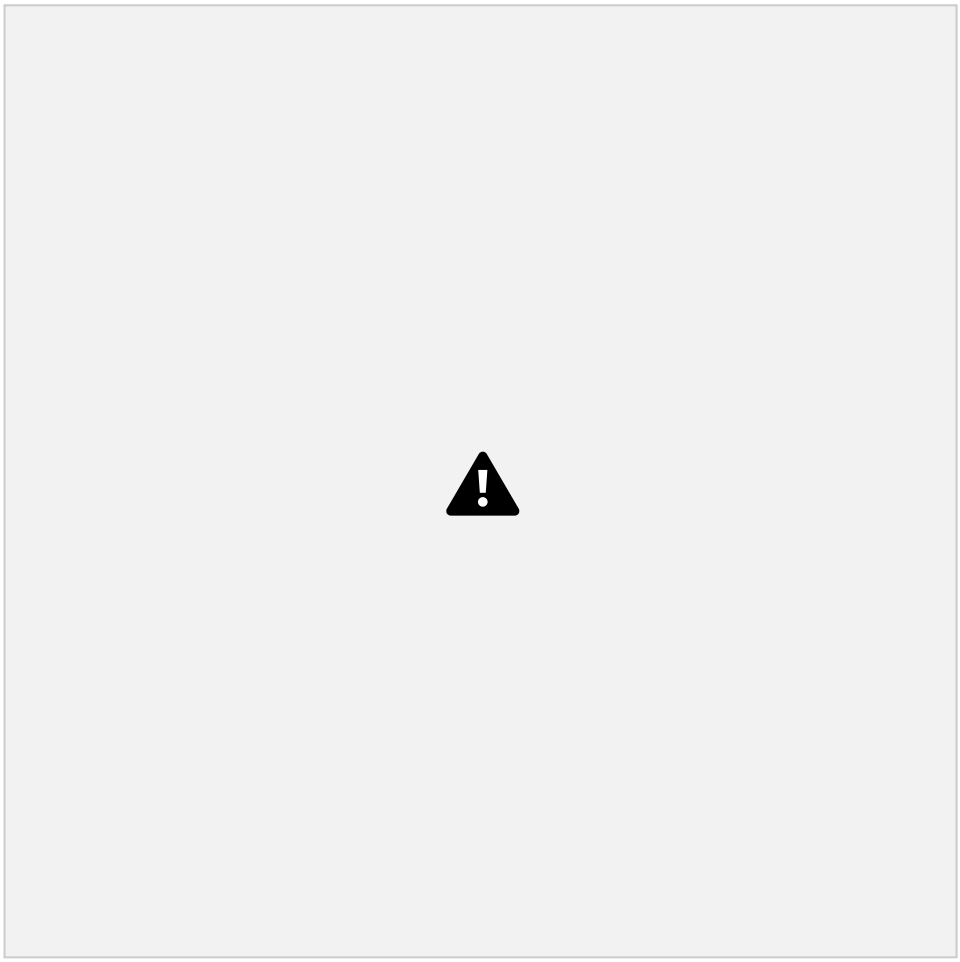

Figure 34: Photo 1

Calculations for Photo 1:

$\alpha = (1{,}441 / 2{,}880) \times (117 / 2) = 0.5003 \times 58.5^\circ = 29.27^\circ$

α (in radians) $= 29.27 \times \pi/180 = 0.5109$ rad

$S_{angle} = (36.9 / 2880) \times 117^\circ = 0.01281 \times 117^\circ = 1.498^\circ$

S_{angle} (in radians) $= 1.498 \times \pi/180 = 0.02614$ rad

$d = \sqrt{(2 \times 6{,}371{,}000 \times 30{,}708)} = \sqrt{(391{,}319{,}112{,}000)} = 625{,}555$ m

$L = 625{,}555 \times \sin(29.27^\circ) = 625{,}555 \times 0.4889 = 305{,}834$ m

$S = 625{,}555 \times \tan(1.498^\circ) = 625{,}555 \times 0.02615 = 16{,}358$ m

$L^2 = 305{,}834^2 = 93{,}534{,}435{,}556$

$S^2 = 16{,}358^2 = 267{,}584{,}164$

$L^2 + S^2 = 93{,}802{,}019{,}720$

$2S = 32{,}716$

$R = 93{,}802{,}019{,}720 / 32{,}716 = 2{,}867{,}000$ m $= 2{,}867$ km

% error $= |2{,}867 - 6{,}371| / 6{,}371 \times 100 = 55.0\%$

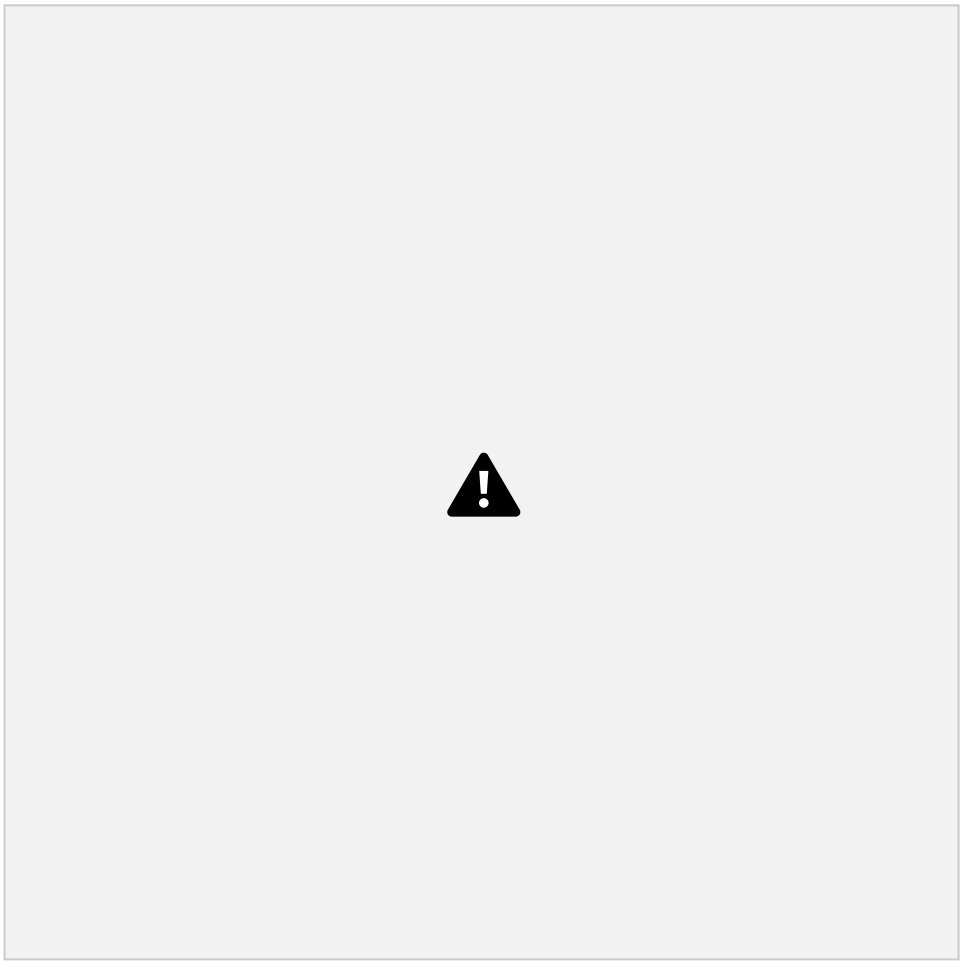

Figure 35: Photo 2

Calculations for Photo 2:
$\alpha = (1{,}440 / 2{,}880) \times 58.5^\circ = 0.500 \times 58.5^\circ = 29.25^\circ$

α in radians = 0.5105 rad

S_{angle} = (34.6 / 2880) × 117° = 0.01201 × 117° = 1.405°
S_{angle} (in radians) = 0.02452 rad

d = √(2 × 6,371,000 × 28,047) = √(357,303,894,000) = 597,749 m

L = 597,749 × sin(29.25°) = 597,749 × 0.4886 = 292,122 m S =

597,749 × tan(1.405°) = 597,749 × 0.02452 = 14,657 m

L^2 = 292,122² = 85,335,262,884
S^2 = 14,657² = 214,827,649
$L^2 + S^2$ = 85,550,090,533
2S = 29,314
R = 85,550,090,533 / 29,314 = 2,918,000 m = 2,918 km

% error = |2,918 − 6,371| / 6,371 × 100 = 54.2%

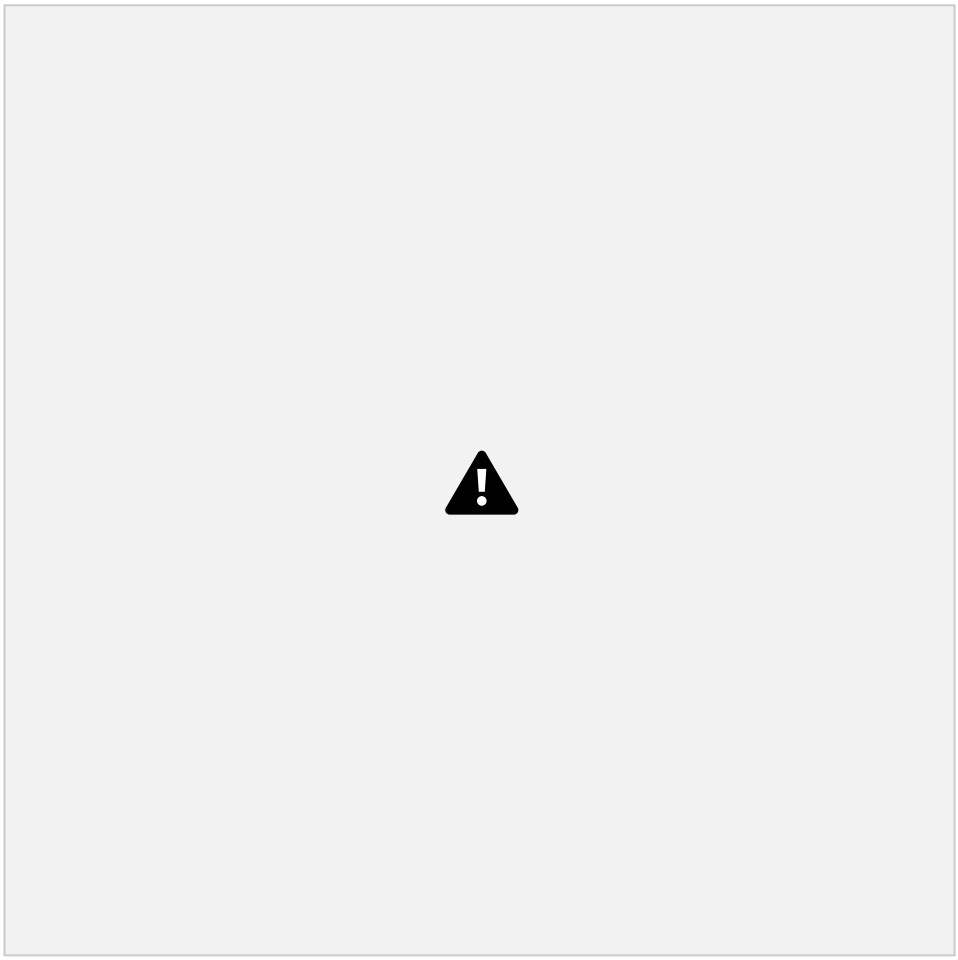

Figure 36: Photo 3

Calculations for Photo 3:

$\alpha = (1{,}440 / 2{,}880) \times 58.5^\circ = 29.25^\circ$

α (in radians) = 0.5105 rad

$S_{angle} = (34.0 / 2880) \times 117^\circ = 0.01181 \times 117^\circ = 1.382^\circ$

S_{angle} (in radians) = 0.02412 rad

$d = \sqrt{(2 \times 6{,}371{,}000 \times 23{,}900)} = \sqrt{(304{,}530{,}800{,}000)} = 551{,}844$ m

$L = 551{,}844 \times \sin(29.25^\circ) = 551{,}844 \times 0.4886 = 269{,}631$ m

$S = 551{,}844 \times \tan(1.382^\circ) = 551{,}844 \times 0.02412 = 13{,}310$ m

$L^2 = 269{,}631^2 = 72{,}700{,}869{,}161$

$S^2 = 13{,}310^2 = 177{,}156{,}100$

$L^2 + s^2 = 72{,}878{,}025{,}261$

$2S = 26{,}620$

$R = 72{,}878{,}025{,}261 / 26{,}620 = 2{,}738{,}000$ m $= 2{,}738$ km

% error $= |2{,}738 - 6{,}371| / 6{,}371 \times 100 = 57.0\%$

ME: *Study of Temperature Absorption*

The goal of this experiment was to investigate how surface color affects the rate and degree of temperature absorption when exposed to sunlight. Three identical thermistor temperature sensors were each coated with a different color, black, white, and a color of choice (blue), and tested under the same conditions to determine whether color alone had a measurable effect on heat absorption. Based on the known physics of radiation absorption, the hypothesis was that the black sensor would absorb the most heat, followed by blue, with white absorbing the least due to its higher reflectivity.

To conduct the experiment, three thermistor sensors were tested before they were painted to give a baseline resistance reading. Each sensor was then spray painted a different color and left to dry for a full day before testing. Once dry, each sensor was placed in direct sunlight for one minute and then immediately measured using a multimeter to record its resistance in kilo ohms. The sensors were tested one at a time under the same conditions and in the same location to keep things as controlled as possible. Since thermistors decrease in resistance as temperature increases, the change in resistance between the before and after readings was used to measure how much each sensor heated up.

The physics behind color absorption comes from how we perceive color in the first place. What we see as color is not the object itself, but rather the wavelengths of light that it reflects back to our eyes. Every other wavelength is absorbed, so when we see something as blue, that means the surface is absorbing almost every wavelength of light except blue light, which is reflected back.

Black, on the other hand, absorbs every wavelength across the visible spectrum and reflects none of it back, which is why it appears black to our eyes. Because it is absorbing all incoming wavelengths rather than reflecting them, black surfaces convert more light energy into heat than any other color. White does the opposite, it reflects nearly all wavelengths back, absorbing very little and therefore heating up the least. This relationship between color and heat absorption is directly tied to the wavelength of light. Shorter wavelengths like ultraviolet carry more energy, while longer wavelengths like infrared carry less. A surface that absorbs across a wider range of wavelengths, like black, will accumulate more total energy and reach a higher temperature than one that reflects most of them back. Which is why in theory the black color temperature sensor should have decreased the most, while the white should have decreased the least and blue should be somewhere in between.

The results of the experiment were surprising, based on the change in resistance after one minute in direct sunlight, all three sensors heated up by a similar amount. The black sensor dropped from 11.13 kΩ to 7.70 kΩ, a change of 3.43 kΩ. The white sensor dropped from 11.21 kΩ to 7.79 kΩ, a change of 3.42 kΩ. The blue sensor dropped from 11.52 kΩ to 8.37 kΩ, a change of 3.15 kΩ. Rather than black absorbing the most heat and white the least as predicted, all three sensors performed very similarly, with black and white essentially tied and blue absorbing slightly less. The differences between the sensors were so small (within 0.28 kΩ of each other) that it is difficult to conclude that color had a meaningful effect on heat absorption under these testing conditions. One potential explanation is that the surface area of the sensors was too small for the color to make a significant difference. Because the painted surface was so minimal, the amount of additional heat generated by one color over another may have been too small to produce a measurable difference in resistance, basically eliminating any effect that color absorption would have had on a larger surface.

Overall, the results of the experiment went against my hypothesis. I expected that black would absorb the most heat, white the least, and blue somewhere in the middle, but the data showed all three sensors performing nearly identically with no meaningful difference between them. There are a few things that could have been done differently to improve the experiment. First, running the test again with a fresh set of sensors would help determine whether the sensors themselves were functioning properly or whether there was something inconsistent about the way they were

reading. Second, applying more layers of spray paint could have made a real difference in the outcome. A thicker coating would increase the surface area of paint on the sensor and create a denser barrier between the sensor and the sun, giving each color more material to either absorb or reflect light. This could have produced a larger difference between the three colors and results that more closely matched what I predicted. The thin single layer of paint may simply not have been enough to produce a measurable effect.

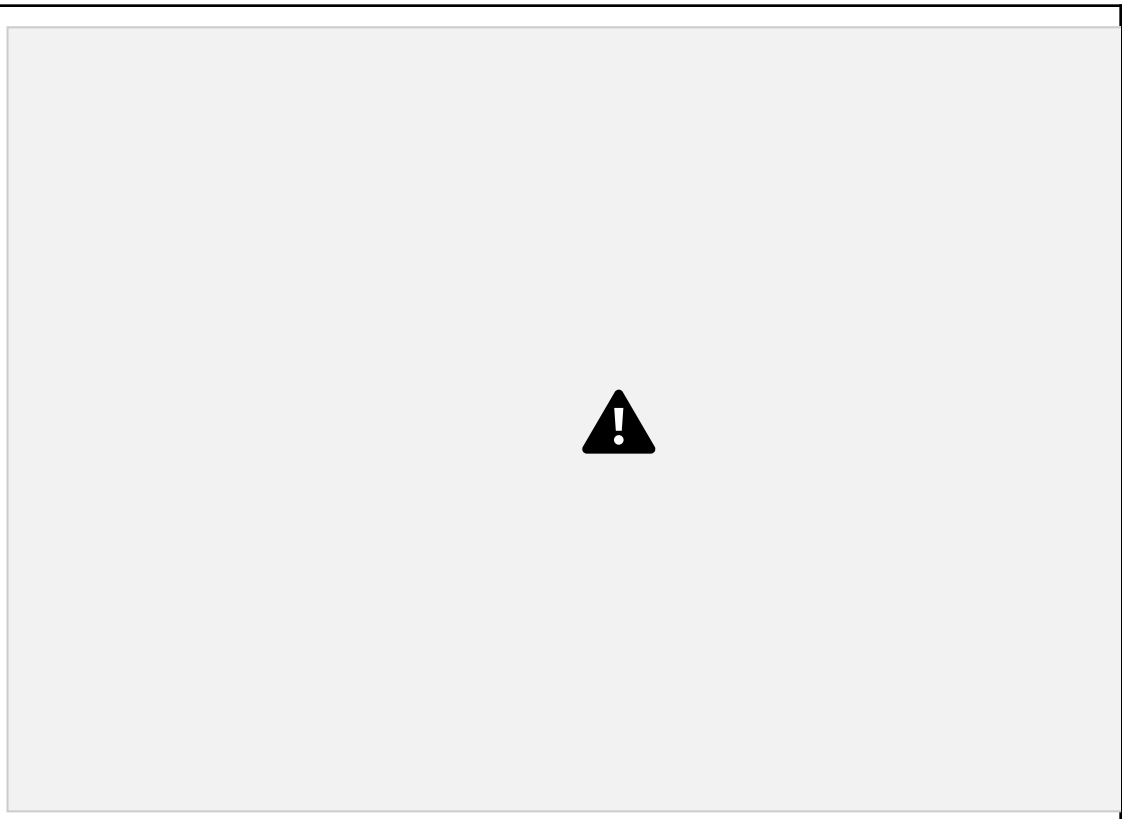

Figure 37: Bar graph of change between original and paint temperature sensors

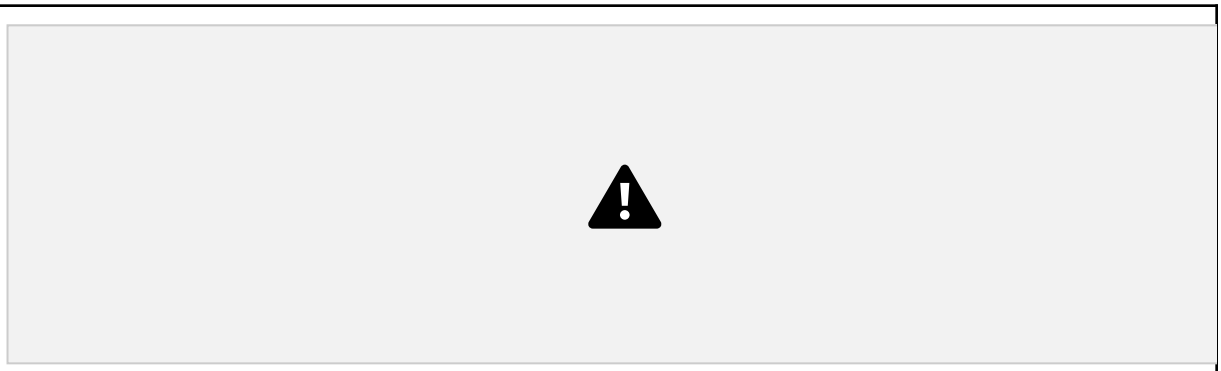

Figure 38: Bar graph of Resistance Change by Sensor Color After One Minute of Sun Exposure

XII. Conclusion

The data collected during this high-altitude balloon flight provides a detailed look at how atmospheric pressure, temperature, humidity, and altitude interact during a real ascent through the troposphere and into the stratosphere, and how well theoretical models capture that behavior.

The pressure vs. altitude data confirmed the expected exponential decay of atmospheric pressure with increasing altitude, and the comparison between the two models revealed that the NASA Empirical Model was more accurate than the Exponential Model after ~10,000 m. This was consistent with the known breakdown of the single-scale-height assumption, where the atmospheric temperature lapse rate changes. This was discussed in more detail in Section III.

The GPS altitude vs. time data demonstrated a remarkably consistent ascent rate of approximately 277 m/min, confirming stable buoyancy and a well-controlled flight. This number was calculated by dividing the maximum recorded altitude (~31,000 m) by the approximate flight duration (~112 minutes, from roughly 11:53 to 13:45).

One key issue that occurred during our launch was the loss of data post-cutdown. It is impossible to determine the real cause of this, but after much speculation, we believe that it was due to a loose connection between the Arduino's battery and the Arduino itself. This was the only option because there is no other single wire that would stop the entire data logging. Additionally, this would explain why data suddenly started logging ~8 hours again after it had hit the ground, meaning that the wire had perhaps slipped back into place. Our only other idea was that the SD card may have come out, but this seems highly unlikely because an SD card inserting itself back in seems more difficult than a wire slipping back into place.

The cutdown itself also did not perform as intended, with the nichrome wire snapping at altitude. This was likely due to the extreme thermal cycling between the cold vacuum of the upper atmosphere and the rapid heating caused by the capacitor discharge.

Despite these setbacks, the flight was overall successful and provided us with data that closely aligns with established atmospheric science. While the balloon launch offered us meaningful insight into how different data points (such as temperature, pressure, and humidity) evolve through the lower and middle atmosphere, it also provided us with an enriching opportunity to

collaborate and learn from others. Collectively, our group bonded throughout this project and emerged having learned lessons and worked hard together. As project leader, I deeply enjoyed getting to know each team member individually and will never forget both the highs and lows that came from this experiment.

XIII. References

American Chemical Society. (2024). Digital imaging. Celebrating Chemistry. https://www.acs.org/education/celebrating-chemistry-editions/2024-ncw/digital-imaging.html

Dastrup, R. A. (n.d.). *Layers of the atmosphere*. Lumen Learning / SUNY Physical Geography. https://courses.lumenlearning.com/suny-geophysical/chapter/layers-of-the-atmosphere/

Earth Sciences New Zealand | NIWA. (n.d.). *Layers of the atmosphere*. https://niwa.co.nz/atmosphere/layers-atmosphere

Elsevier. (n.d.). *Photochemistry*. ScienceDirect Topics. https://www.sciencedirect.com/topics/chemistry/photochemistry

Eratosthenes calculates the circumference of the Earth. In Science and its times: Understanding the social significance of scientific discovery. Encyclopedia.com. https://www.encyclopedia.com/science/encyclopedias-almanacs-transcripts-and-maps/eratosthenes-calculates-circumference-earth

Hall, N. (Ed.). (2021, May 13). *Earth atmosphere model – metric units*. NASA Glenn Research Center. https://www.grc.nasa.gov/WWW/K-12/airplane/atmosmet.html

Hazardous Gas Monitor - SparkFun Learn. (2025). Sparkfun.com. https://learn.sparkfun.com/tutorials/hazardous-gas-monitor

Hernández, J. C., Machucho, M. A., & Ramírez, J. E. (2025). Adiabatic lapse rate estimation using a Van Der Waals-type equation of state. *Brazilian Journal of Physics, 56*, 15. https://doi.org/10.1007/s13538-025-01937-0

Intel. (n.d.). *Computer and Laptop RAM*. Intel.

https://www.intel.com/content/www/us/en/tech-tips-and-tricks/computer-ram.html

Machine Applications Corporation. (2021, May 7). *What is the density of air at STP?* MAC Instruments. https://macinstruments.com/blog/what-is-the-density-of-air-at-stp/

NOAA Office of Response and Restoration, US GOV. (2010). *CARBON MONOXIDE | CAMEO Chemicals | NOAA*. Noaa.gov. https://cameochemicals.noaa.gov/chemical/335

National Oceanic and Atmospheric Administration. (2026, February 17). *Layers of the atmosphere*. https://www.noaa.gov/jetstream/atmosphere/layers-of-atmosphere

What Is Arduino? | Arduino Documentation. (2022, January 25). Docs.arduino.cc. https://docs.arduino.cc/learn/starting-guide/whats-arduino/

XIV. Appendix I

```
#include <RTClib.h>
#include <SPI.h>
#include <SD.h>
#include <math.h>
#include <Adafruit_GPS.h>
#include <Wire.h>
#include <SparkFun_MS5803_I2C.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ST7735.h>
#include "SparkFun_SHTC3.h"

bool HeatPad = false; //used for indicating the state of the Heating pad in the data file. 0 when off, 1 when on.
bool CutDownHappen = false; //used for only running cutdown 'if' statement once. After cutdown, set to TRUE so that cutdown statement isn't checked again.
bool HumidityFailed = false; //used for initializing the humidity sensor in the loop if for some reason it failed during boot up.
unsigned long buzzertimer; //clock variable for buzzer. If greater than 3 hrs, buzzer state machine is activated (beeps 3 consecutive times after every minute)
const int buzzer_pin = 39;
```

```
bool buzzerActivate = false;
unsigned long Humrtc;


// --------------------------------SD GLOBAL CODE ---------------------------------
File dataFile;
const int chipSelect = 10; //pin 10 is SD card on adafruit shield
//^^^^^^^^^^^^^^^^^^^^^^SD GLOBAL CODE^^^^^^^^^^^^^^^^^^^^^^^^^^^^


// --------------------------------RTC GLOBAL CODE ---------------------------------
RTC_PCF8523 rtc;
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};
DateTime now;

unsigned long Datapullrtc; //Data pulling timer:
unsigned long CUTDOWNrtc; //Cutdown timer
unsigned long reinitialize; //timer for checking to reinitializing sensors in loop.
const int Datapull = 30; //Data pulling interval, in seconds
const int CUTDOWN = 7200; //CutDown interval--- 2 hrs since plug in.

//^^^^^^^^^^^^^^^^^^^^^^RTC GLOBAL CODE^^^^^^^^^^^^^^^^^^^^^^^^^^^^


// --------------------------------TEMP GLOBAL CODE ---------------------------------
//pow(2,3) = 2 cubed
//log() = logbase e of whatevers in parenthesis

//defining internal temp variables:
const int THERMin_PIN = A0; //A0 = internal thermistor read pin 45 double tempcurrentin; //tempcurrentin = current through internal thermistor voltage divider,
double therm_resistance_in; //resistance of the internal resistor
double temperaturein; //internal temperature (calculated using SHH) 48 int r1in = 9900;
//r1in = resistance of fixed resistor in internal temp circuit 49 //Steinhardt constants for
```

internal temp circuit (A, B, C):

50 const double tinshhA = -0.00024775;

51 const double tinshhB = 0.00044917;

52 const double tinshhC = -0.0000006388;

53

54 //defining external temp variables:

55 const int THERMex_PIN = A1; //A1 = external thermistor read pin

56 double tempcurrentex; //tempcurrentex = current through external thermistor divider 57

double therm_resistance_ex; //resistance of the external thermistor

58 double temperatureex; //External temperature (calculated using SHH) 59 int r1ex =

9960; //r1ex = resistance of fixed resistor in external temp circuit 60

61 //Steinhardt constants for external temp circuit (A,B,C), precalculated prior using calibration datapoints:

62 const double texshhA = -0.00053760;

63 const double texshhB = 0.00046323;

64 const double texshhC = -0.000000523811;

65

66 //steinhardt calculation variables;

67 int adcin; //adc reading of internal temperature node 68 float voltagein;

//node voltage of internal temp divider

69 int adcex; //adc reading of external temp node

70 float voltageex; //voltage of external temp node

71 double lntinR; //ln of the internal temperature resistance 72 double lntexR;

//ln of the external temperature resistance

73 //^^^^^^^^^^^^^^^^^^^^^^^^TEMP GLOBAL CODE^^^^^^^^^^^^^^^^^^^^^^^^

74

75 // --------------------------------GPS GLOBAL CODE --------------------------------

76 #define GPSSerial Serial1

77 Adafruit_GPS GPS(&GPSSerial);

78 #define GPSECHO false

79 //^^^^^^^^^^^^^^^^^^^^^^^^GPS GLOBAL CODE^^^^^^^^^^^^^^^^^^^^^^^^

80

81 //----------------------------------PRESSURE GLOBAL--------------------------------------

```
82 // Sensor I2C address: ADDRESS_HIGH = 0x76, ADDRESS_LOW = 0x77 83
MS5803 sensor(ADDRESS_HIGH);
84
85 char dataFilename[15]; // Current log filename
86 float baselinePressure = 0; // First pressure reading (for delta)
87 int sampleCount = 0; // Number of samples taken
88 bool sdOK = false; // SD card status
89 bool sensorOK = false; // Sensor status
90
91 float pressure = NAN; //Global pressure value
92 float deltaP;
93 float raw;
94 float temperature;
95 //^^^^^^^^^^^^^^^^^^^^^^PRESSURE GLOBAL CODE^^^^^^^^^^^^^^^^^^^^^^
96
97 //Cutdown variables for keeping the relay closed for 60 seconds before never closing again (to
save battery):
98 bool relayClose = false;
99 unsigned long Relaytime = 0;
100
101 //---------------------------------Humidity GLOBAL CODE--------------------
102 SHTC3 mySHTC3;
103 float humidity;
104 float HUMtempC;
105 float HUMtempF;
106 //^^^^^^^^^^^^^^^^^^^^^^^^^^Humidity GLOBAL CODE^^^^^^^^^^^^^^^^^^^^^^^^^^
107
108 // -------------------------------CO GLOBAL CODE --------------------------------
109 const int COanalogPin = A2; // AOUT from MQ-7
110 int COanalogValue; //the ADC reading from Aout into Pin A2
111 float COvoltage; //the calculated voltage using COanalogValue
112 float COresistance; //the calculated resistance of the MQ-7 sensor
113 //^^^^^^^^^^^^^^^^^^^^^^^^CO GLOBAL CODE^^^^^^^^^^^^^^^^^^
114
```

```
// ---------------------------------Camera Global ---------------------------------
DateTime Phototakentime; //timestamp for when photos were taken
const int CAMERA_PIN = 30; // Relay module IN pin
const int HEATING_PIN = 41; //heating relay pin is pin 41

const bool RELAY_ACTIVE_LOW = false;

const unsigned long PERIODIC_S = 120; // 2 minutes (unix seconds) 123
const unsigned long RELAY_ON_TIME_MS = 500; // 0.5 s ON 124 const
unsigned long PAUSE_TIME_MS = 5000; // 5.0 s pause 125 const int
PULSE_COUNT = 3; // 3 connections

enum SeqState {
IDLE,
PULSE_ON,
PAUSE
};

SeqState state = IDLE;

unsigned long stateStartTime = 0;
unsigned long lastPeriodicTime = 0;
int currentPulse = 0;
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^Camera GLOBAL CODE^^^^^^^^^^^^^^^^^^^^

// ------------------------------ Buzzer Global ---------------------
const unsigned long buzzPERIOD = 60000; // 1 minute
const unsigned long buzzBEEP_TIME = 500; // buzzer on time 143
const unsigned long buzzPAUSE_TIME = 500; // gap between beeps 144
unsigned long buzzlastTrigger = 0;
unsigned long buzzstateStart = 0;

int buzzbeepCount = 0;
```

```
149 int buzzstate = 0;
150 //^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^BUZZER GLOBAL CODE^^^^^^^^^^^^^^^^^^
151
152 //functions:
153 void GPSread();
154 void SDLOG();
155
156 // ------------------------------ Camera functions --------------------
157 void setRelay(bool on) {
158 digitalWrite(CAMERA_PIN, on
159 ? (RELAY_ACTIVE_LOW ? LOW : HIGH)
160 : (RELAY_ACTIVE_LOW ? HIGH : LOW)
161 );
162 }
163
164 void startSequence() {
165 if (state != IDLE) return;
166
167 currentPulse = 1;
168 state = PULSE_ON;
169 stateStartTime = millis();
170 setRelay(true);
171 Phototakentime = rtc.now();
172 }
173 //^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^Camera functions^^^^^^^^^^^^^^^^^^^^^^^^
174
175
176 void setup() {
177 Serial.begin(115200);
178 //pin setup:
179 pinMode(22, OUTPUT); //cutdown pin is output, set low in next line
180 digitalWrite(22, LOW);
181 pinMode(10, OUTPUT); //SD pin is an output for writing to SD card
```

```
182 pinMode(HEATING_PIN, OUTPUT);
183 digitalWrite(HEATING_PIN, LOW);
184 pinMode(buzzer_pin, OUTPUT);
185 digitalWrite(buzzer_pin, LOW);
186 // --------------------------------SD SETUP CODE ---------------------------------
187 if (!SD.begin(chipSelect)) {
188 Serial.println("SD card failed");
189 while (1) { delay(1000); }
190 }
191 Serial.println("SD card OK");
192 //^^^^^^^^^^^^^^^^^^^^^^^^^SD SETUP CODE^^^^^^^^^^^^^^^^^^^^^^^^^
193
194 // --------------------------------RTC SETUP CODE ---------------------------------
195 #ifndef ESP8266
196 while (!Serial);
197 #endif
198
199 if (! rtc.begin()) {
200 Serial.println("Couldn't find RTC");
201 Serial.flush();
202 while (1) delay(10);
203 }
204 now = rtc.now();
205 Datapullrtc = now.unixtime();
206 CUTDOWNrtc = now.unixtime();
207 reinitialize = now.unixtime();
208 buzzertimer = now.unixtime();
209 Humrtc = now.unixtime();
210
211 if (! rtc.initialized() || rtc.lostPower()) {
212 Serial.println("RTC is NOT initialized, let's set the time!"); 213
rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
214 Serial.print("Adjusted RTC to compile time: ");
```

```
215 Serial.println(DateTime(F(__DATE__), F(__TIME__)).timestamp()); 216
}
217 rtc.start();
218 //^^^^^^^^^^^^^^^^^^^^^^^^RTC SETUP CODE^^^^^^^^^^^^^^^^^^^^^^^^^
219
220 // --------------------------------GPS SETUP CODE ---------------------------------
221 GPS.begin(9600);
222 GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
223 GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ); 224
GPS.sendCommand(PGCMD_ANTENNA);
225 delay(1000);
226 GPSSerial.println(PMTK_Q_RELEASE);
227 //^^^^^^^^^^^^^^^^^^^^^^^GPS SETUP CODE^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
228
229 // --------------------------------PRESSURE SETUP CODE ---------------------------------
230 Wire.begin();
231 if (!SD.begin(chipSelect)) {
232 sdOK = false;
233 } else {
234 sdOK = true;
235 }
236 sensor.reset();
237 delay(10);
238 sensor.begin();
239 float testPressure = sensor.getPressure(ADC_4096);
240 if (testPressure > 0 && testPressure < 20000) {
241 sensorOK = true;
242 } else {
243 sensorOK = false;
244 }
245
246 if (sdOK) {
247 strcpy(dataFilename, "LOG00.CSV");
```

```
248 for (uint8_t i = 0; i < 100; i++) {
249 dataFilename[3] = '0' + i / 10;
250 dataFilename[4] = '0' + i % 10;
251 if (!SD.exists(dataFilename)) {
252 dataFile = SD.open(dataFilename, FILE_WRITE);
253 break;
254 }
255 }
256 } //for printing the CSV header row:
257 if (dataFile) {
258 dataFile.println(F("Time (rtc),Temp_internal,Thermistor_internal_voltage,
Thermistor_internal_resistance,Temp_external,Thermistor_external_voltage,Thermistor_external_resistan
ce,pressure_mbar,pressure_sensor_temp_C,delta_P,GPS_UTC,GPS_fix,Fix_quality,Latitude,Longitude,A
ltitude,speed,angle,satellites,antenna_status,humidity,HUM_temp_C,HUM_temp_F,CO_Analog_Value,C
O_Voltage,CO_Resistance,photo_taken,HeatPad?"));
259 dataFile.close();
260 Serial.println(dataFilename);
261 } else {
262 sdOK = false;
263 }
264 //^^^^^^^^^^^^^^^^^^PRESSURE SETUP CODE^^^^^^^^^^^^^^
265
266 //--------------------------------Humidity Setup-----------------------------------
267 SHTC3_Status_TypeDef result = mySHTC3.begin();
268 if (result != SHTC3_Status_Nominal) {
269 Serial.println("ERROR: SHTC3 sensor not found!");
270 HumidityFailed = true;
271 } else{
272 Serial.println("SHTC3 HUMIDITY sensor ready.");
273 }
274 //^^^^^^^^^^^^^^^^^^^^^^Humidity Setup^^^^^^^^^^^^^^^^^^^^^^^^
275
276 //-------------------------------------------------Camera Setup---------------------------------------
```

```
pinMode(CAMERA_PIN, OUTPUT);
setRelay(false);
DateTime now = rtc.now();
lastPeriodicTime = now.unixtime();
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^Camera Setup^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
}

//Function for reading the actual supply voltage.
long readVcc_mV() {
ADMUX = _BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1);
delay(2);
ADCSRA |= _BV(ADSC);
while (bit_is_set(ADCSRA, ADSC)) {}
uint16_t adc = ADC;
long vcc = 1125300L / adc;
return vcc;
}


void loop() {
now = rtc.now();
GPSread();

//--------------------------------------------------------TEMP LOOP--------
adcin = analogRead(THERMin_PIN);
voltagein = adcin * (5.0 / 1023.0);
tempcurrentin = (5 - voltagein) / r1in;
therm_resistance_in = voltagein / tempcurrentin;

adcex = analogRead(THERMex_PIN);
voltageex = adcex * (5.0 / 1023.0);
tempcurrentex = (5 - voltageex) / r1ex;
therm_resistance_ex = voltageex / tempcurrentex;
```

```

lntinR = log(therm_resistance_in);
temperaturein = (1.0 / (tinshhA + tinshhB * lntinR + tinshhC * pow(lntinR,3)) - 273);

lntexR = log(therm_resistance_ex);
temperatureex = (1.0 / (texshhA + texshhB * lntexR + texshhC * pow(lntexR,3)) - 273);

//----------------HEATING PAD------------//
if (temperaturein <= 3) {
digitalWrite(HEATING_PIN,HIGH);
HeatPad = true;
}
if (temperaturein >= 10){
digitalWrite(HEATING_PIN, LOW);
HeatPad = false;
}
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^Temp Loop^^^^^^^^^^^

//--------------------------------------------------------PRESSURE LOOP------
if (!sensorOK) {
//skip
} else {
temperature = sensor.getTemperature(CELSIUS, ADC_4096);
raw = sensor.getPressure(ADC_4096);
pressure = .997 * raw - 2.49;
if (sampleCount == 0) {
baselinePressure = pressure;
}
sampleCount++;
deltaP = pressure - baselinePressure;
}
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^PRESSURE Loop^^^^^^

```

```
343 if (now.unixtime() - Datapullrtc >= Datapull) {
344 SDLOG();
345 }
346
347 //----------------------------------------------------------CUTDOWN TRIGGER----
348 if (!CutDownHappen) {
349 if (!isnan(pressure) && pressure <= 10 && now.unixtime() - CUTDOWNrtc >= CUTDOWN) {
350 digitalWrite(22, HIGH);
351 Relaytime = now.unixtime();
352 relayClose = true;
353 CutDownHappen = true;
354 }
355 }
356 if (relayClose){
357 if (now.unixtime() - Relaytime >= 60) {
358 digitalWrite(22, LOW);
359 relayClose = false;
360 }
361 }
362 //^^^^^^^^^^^^^^^CUTDOWN TRIGGER^^^^^^^^^^^
363
364 //-----------------------Humidity LOOP---------------------
365 if (!HumidityFailed) {
366 if (now.unixtime() - Humrtc >= 5) {
367 Humrtc = now.unixtime();
368 mySHTC3.update();
369 humidity = mySHTC3.toPercent();
370 HUMtempC = mySHTC3.toDegC();
371 HUMtempF = mySHTC3.toDegF();
372 }
373 }
374 //^^^^^^^^^^^^^^^^^^^^^^^Humidity LOOP^^^^^^^^^^^^^
375
```

```
376 //-------------------------CAMERA LOOP----------------
377 now = rtc.now();
378 unsigned long unixNow = now.unixtime();
379 unsigned long msNow = millis();
380
381 if (state == IDLE && (unixNow - lastPeriodicTime >= PERIODIC_S)) {
382 lastPeriodicTime = unixNow;
383 startSequence();
384 }
385
386 switch (state) {
387 case PULSE_ON:
388 if (msNow - stateStartTime >= RELAY_ON_TIME_MS) { 389
setRelay(false);
390 if (currentPulse >= PULSE_COUNT) {
391 state = IDLE;
392 } else {
393 state = PAUSE;
394 stateStartTime = msNow;
395 }
396 }
397 break;
398
399 case PAUSE:
400 if (msNow - stateStartTime >= PAUSE_TIME_MS) { 401
currentPulse++;
402 setRelay(true);
403 state = PULSE_ON;
404 stateStartTime = msNow;
405 }
406 break;
407
408 case IDLE:
```

```
409 default:
410 break;
411 }
412 //^^^^^^^^^^^^^^^^^^^^^^^^^^^^CAMERA LOOP^^^^^^^^^^^^^^^
413
414 //------------------------------BUZZER State machine---------------
415 if (buzzerActivate) {
416 unsigned long buzznow = millis();
417
418 if (buzzstate == 0 && buzznow - buzzlastTrigger >= buzzPERIOD) { 419
buzzlastTrigger = buzznow;
420 buzzbeepCount = 1;
421 buzzstate = 1;
422 buzzstateStart = buzznow;
423 digitalWrite(buzzer_pin, HIGH);
424 }
425
426 if (buzzstate == 1 && buzznow - buzzstateStart >= buzzBEEP_TIME) {
427 digitalWrite(buzzer_pin, LOW);
428 if (buzzbeepCount >= 3) {
429 buzzstate = 0;
430 } else {
431 buzzstate = 2;
432 buzzstateStart = buzznow;
433 }
434 }
435
436 if (buzzstate == 2 && buzznow - buzzstateStart >= buzzPAUSE_TIME) { 437
buzzbeepCount++;
438 digitalWrite(buzzer_pin, HIGH);
439 buzzstate = 1;
440 buzzstateStart = buzznow;
441 }
```

```
442 }
443 //^^^^^^^^^^^^^^^^^Buzzer State machine^^^^^^^^^^^^^^^
444
445 //reinitialize humidity if failed
446 if (HumidityFailed) {
447 humidity = NAN;
448 HUMtempC = NAN;
449 HUMtempF = NAN;
450 if (now.unixtime()- reinitialize >= 10){
451 reinitialize = now.unixtime();
452 SHTC3_Status_TypeDef result = mySHTC3.begin(); 453
if (result != SHTC3_Status_Nominal) {
454 Serial.println("ERROR: SHTC3 sensor not found!"); 455
HumidityFailed = true;
456 } else{
457 Serial.println("SHTC3 HUMIDITY sensor ready."); 458
HumidityFailed = false;
459 }
460 }
461 }
462
463 // --------------------------------CO LOOP CODE ---------------------------------
464 COanalogValue = analogRead(COanalogPin);
465 float VCC = readVcc_mV() / 1000.0f;
466 COvoltage = COanalogValue * (VCC/ 1023.0f);
467 if (COanalogValue <= 1 || COanalogValue >= 1022) {
468 COresistance = NAN;
469 } else {
470 COresistance = (VCC / COvoltage - 1.0) * 10000.0; 471
}
472 //^^^^^^^^^^^^^^^^^^^^^^^^CO LOOP CODE^^^^^^^^^^^^^^^^^^^^^^^^
473
474 if (now.unixtime() - buzzertimer >= 10800){
```

```
475 buzzerActivate = true;
476 }
477 }
478
479
480 void GPSread() {
481 while (GPSSerial.available()) {
482 char c = GPS.read();
483 if (GPSECHO && c ) Serial.write(c);
484 }
485 if (GPS.newNMEAreceived()) {
486 if (!GPS.parse(GPS.lastNMEA())) {
487 }
488 }
489 }
490
491 void SDLOG() {
492 GPSread();
493 Datapullrtc = now.unixtime();
494 if (sdOK) {
```