University of Minnesota Twin Cities and MN Space Grant Consortium

AEM 1301 Freshman Seminar: Fall 2021
Introduction to Spaceflight
(with Stratospheric Ballooning Project)

Team Project Documentation

Team A: Team Sea

Written by:
Jayleigh Acree, Abbie Friessen, Jack Rogers, and James Mccabe

Report Submission Date: 12/03/2021
Revision C

Table of Contents

0.0 Team Member Assignments

General Roles:
Introduction: Jack R
Mission Overview: Jack R
Payload Design: James M
Project Management: Jack R
Project Budgets: James M
Payload Photographs: Jayleigh A
Test Plan and Results: Abbie F
Expected Science Results: Jayleigh A
Flight Day Narrative: James M
Results and Analysis: Abbie F
Conclusions and Lessons Learned: Jayleigh A
References: Jayleigh A
Program Listings: Jack R

Oral Presentation Assignments:
Conceptual Design Review: Jack and James
Flight Readiness Review (FRR): Abbie and Jayleigh

Payload Build Assignments:
Overall team lead and ground-testing lead: Jack R

Payload Box Build: Jayleigh A
PTERODACTYL basic sensor suite: James M
Programmer lead for the PTERODACTYl: Jack R
Camera experiment: Abbie F
Neulog modules: Abbie F
“Other” science experiments: All

Flight Day Assignments:
Documentation/ photographer: All
Flight prediction specialist: All
Payload handling/start-up specialist: Jayleigh A
Comms telemetry data specialist: All
Payload telemetry data specialist: James M
Aprs tracking specialist: Abbie F
SPOT tracking specialist: Jack R

1.0 Introduction

The one major limitation to everyone interested in launching things into space is cost. It is simply too expensive for the average person to pay for the materials, equipment, and manpower to run a unique, self-funded space mission. There is a reason it costs NASA 58 million dollars on average to send something into orbit. Money is what brings many amateurs to use near-space flight balloons.

Several conditions of near-space are especially important for planning a mission: temperature, pressure, and gravity. Temperature after launch initially decreases but then increases once the balloon hits the stratosphere. Pressure, on the other hand, only decreases with increasing altitude which has many interesting effects from the increased prevalence of cosmic rays to the change of color of the sky. Lastly, gravity does decrease as the balloon climbs altitude; however, this effect is very minimal, the change in gravity from an altitude of one foot to 100,000 feet is only 0.95% (Heckman).[1] While these conditions might not be exactly the same as space, they are closely related enough to get good scientific data that can be applied to other types of spaceflight.

The typical amateur near-space mission has two main pieces, the balloon and the payload, and two main stages of flight, ascent and descent. Balloons used for near-space flight are not typical party balloons. They are mostly helium or hydrogen filled, sealed weather balloons built to withstand the conditions of extremely high altitudes. The payloads themselves can be surprisingly simple, consisting of styrofoam boxes with the scientific equipment stored inside equipped to conduct experiments that need space-like conditions. The ascent is fairly simple, there is not much stress on the balloon or the payload and the balloon will continue to rise until about 100,000 feet until it bursts. After the balloon pops the parachute is deployed and descent begins. At high altitude the descent is extremely fast due to the low air density but once the payload approaches the ground it should slow to around 10 ft per second. There is a very good chance anything not inside the payload (ie: solar panels, shelves on the outside) will not survive descent but due to the low cost of materials this is perfectly acceptable.

2.0 Mission Overview

The primary objective of the mission is to examine how atmospheric characteristics change during the journey to and from near-space, such as the relationship between temperature and pressure. Our payload must be able to withstand the extreme temperature and pressure conditions as it ascends; however, our main data will be collected during ascent so we do not have to worry about some of our equipment surviving reentry.

There will be two main experiments conducted during the flight. The first experiment involves the efficacy of different seals when exposed to low pressure. As the payload ascends, pressure will decrease and the seals will likely expand. How will different types of seals (currently a chip bag, ziplock, and water bottle) respond differently? And which type of seal is most effective in resisting the pressure change? Does temperature play a role? These are the questions that will hopefully be

answered using data collected from the flight. These results will be interesting because they can be applied to the real world; there are supply chains that have to deal with altitudes ranging from cities at sea level to cities miles high in mountains. The second main experiment involves radiation in space. Sometimes a radioactive particle will hit a physical bit cell and give it enough energy to flip from a zero to a one. And the more radiation present, the greater chance of a bit change. The micro SD card on the payload will contain an empty file (i.e. filled with zeros) that will be checked for bit changes after the flight.

Finally, the payload will contain a camera, pointed to the side showing the flight's progression in relation to the earth's surface. While many near-space balloon flights have captured the same image we want a video that shows the curvature of the earth. There is nothing more synonymous with spaceflight than this image and it will be a symbol of our successful mission.

3.0 Payload Design

In order to conduct these experiments, as defined in the mission overview, there are certain limitations. Firstly, only systems twelve pounds and under are approved to fly. With the weight of the equipment to reach near-space and the addition of another team's payload, our group is allotted two and half pounds for our payload. This limits part of our experiment, as we can not send up heavy items, like a full water bottle. Then, another limitation comes from the design requirements. For our Neulog sensors to work, the battery must be able to be plugged in the final moments before launch to preserve battery life, and all record buttons must be accessible to start recording. This is why the Neulogs have been placed along one of the longer walls, which allows for easy access. The payload was also designed to have a larger surface area for the lid in order to place the seal experiment on top. We have collaborated with Team B for this, where their camera will face downwards and our payload will be beneath theirs, allowing us to use their video to analyze our experiment.

Parts Equipment List:

Seal Experiment:

- Personal size Lay's chip bag
- Empty mini water bottle
- Slightly inflated Ziploc bag

Radiation Experiment:

- SD card (housed on PTERODACTYL)

Inside payload:

- PTERODACTYL
- Lightdow Camera
- Anker Camera battery
- Neulog Sensors
    - Pressure
    - Temperature

    - Geiger Counter (with extended sensor stick)
    - Battery
- 9V battery for PTERODACTYL

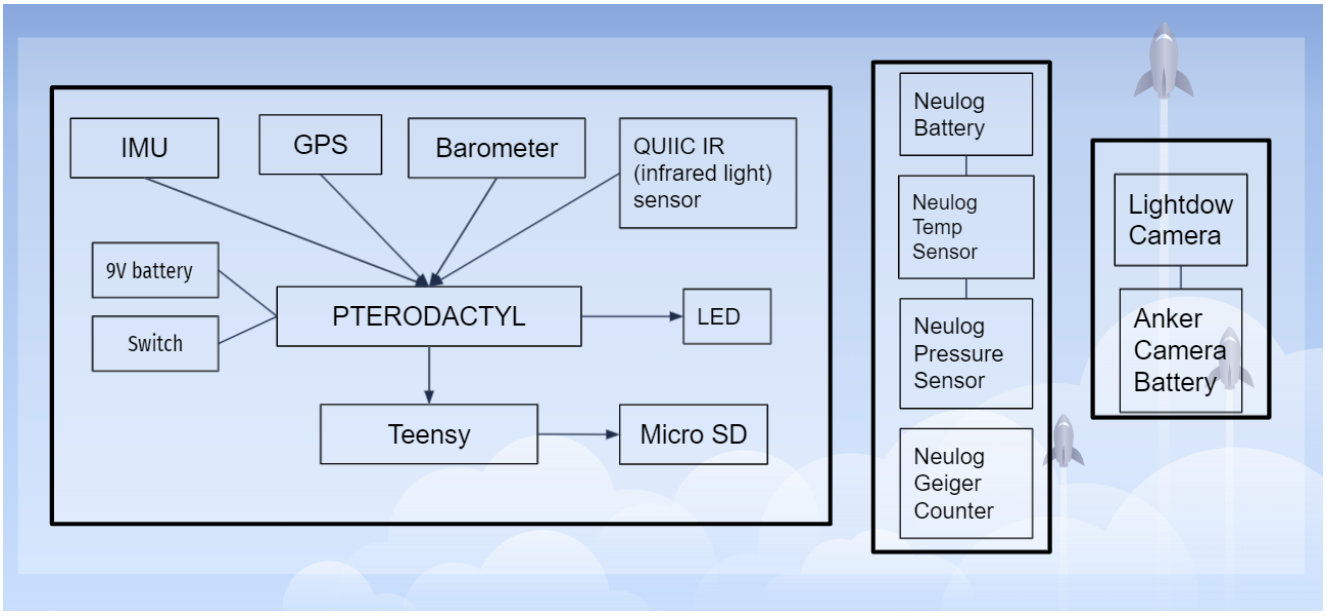
External Equipment:

- Switch (to turn on/off PTERODACTYL)

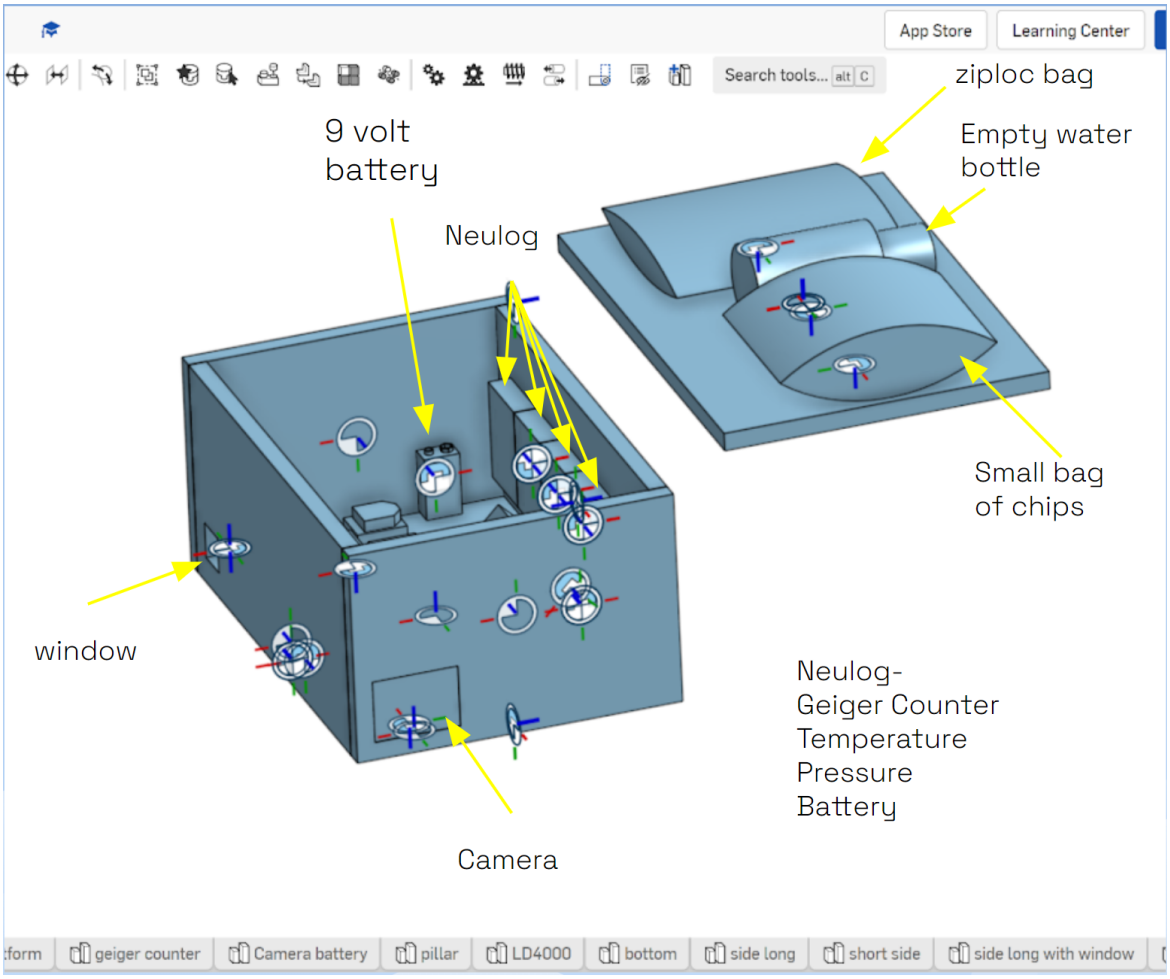
Building Materials:

- Styrofoam Insulation
- Strapping (filament) tape
- Electrical Tape

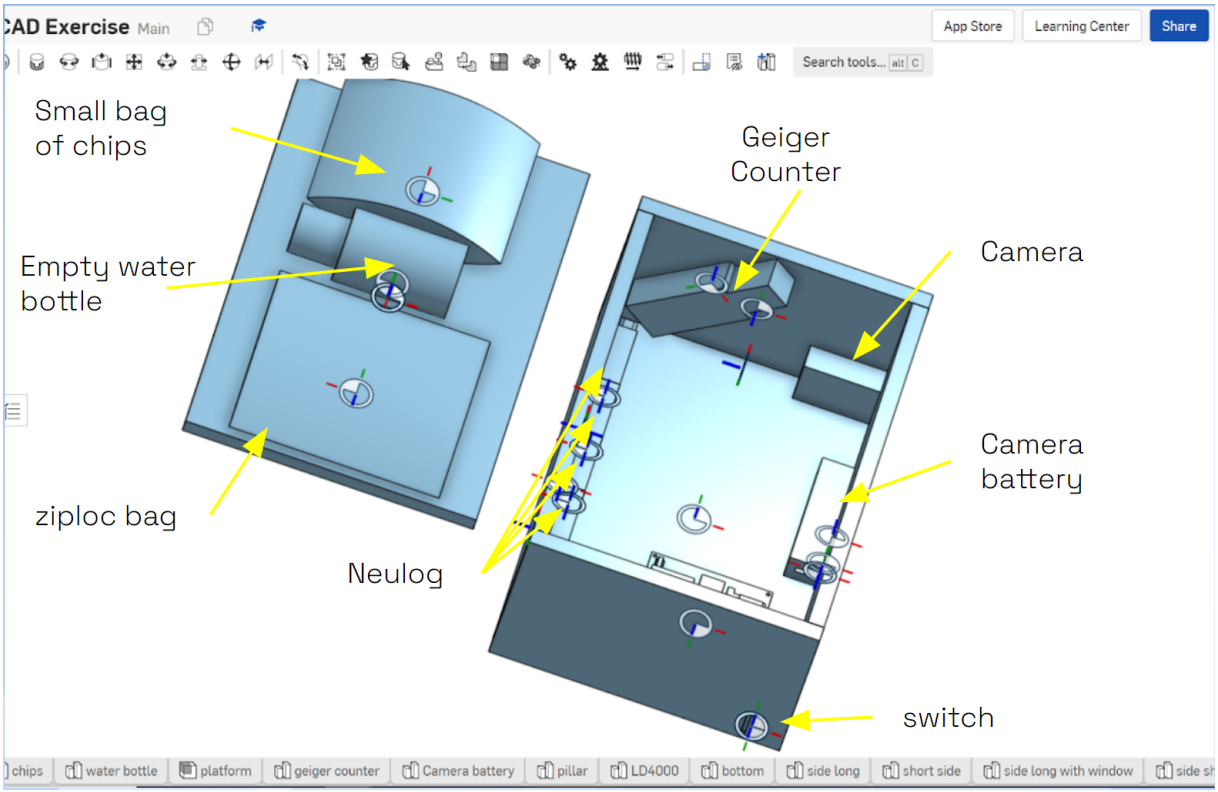
Functional Block Diagram:



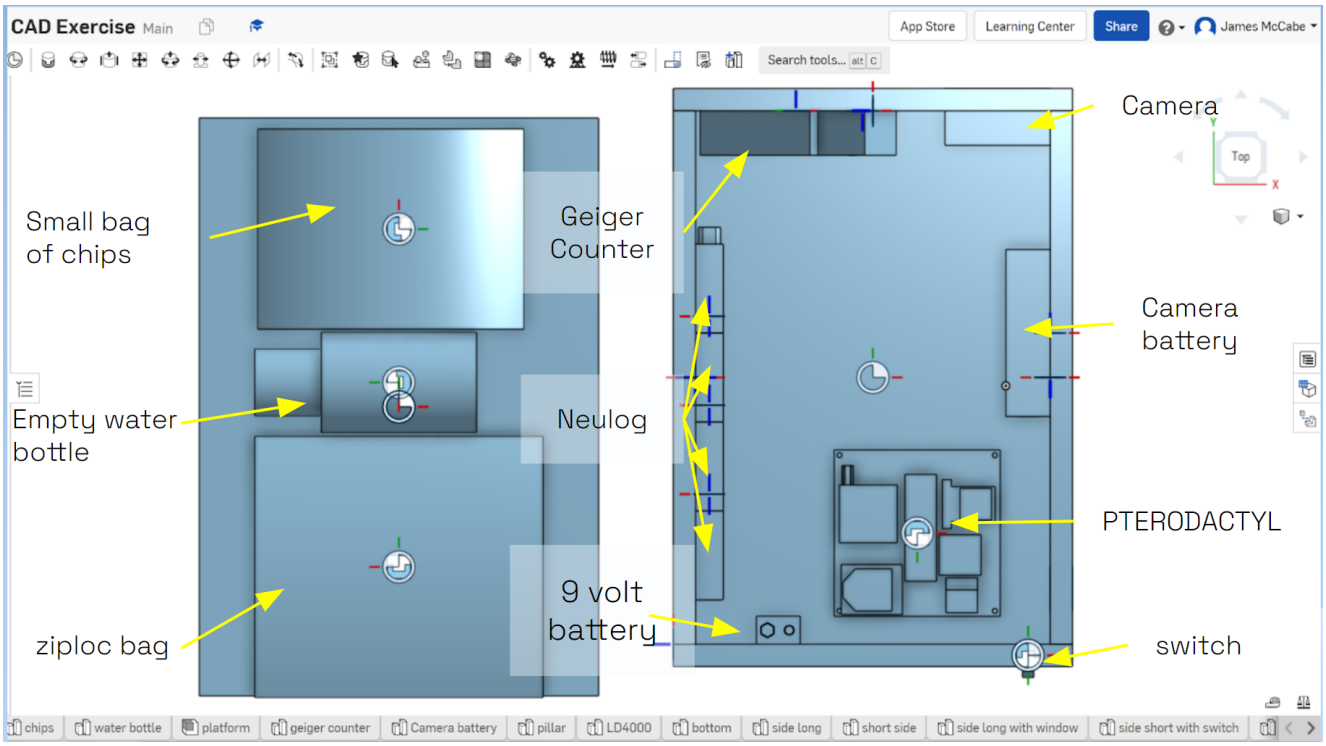
CAD:



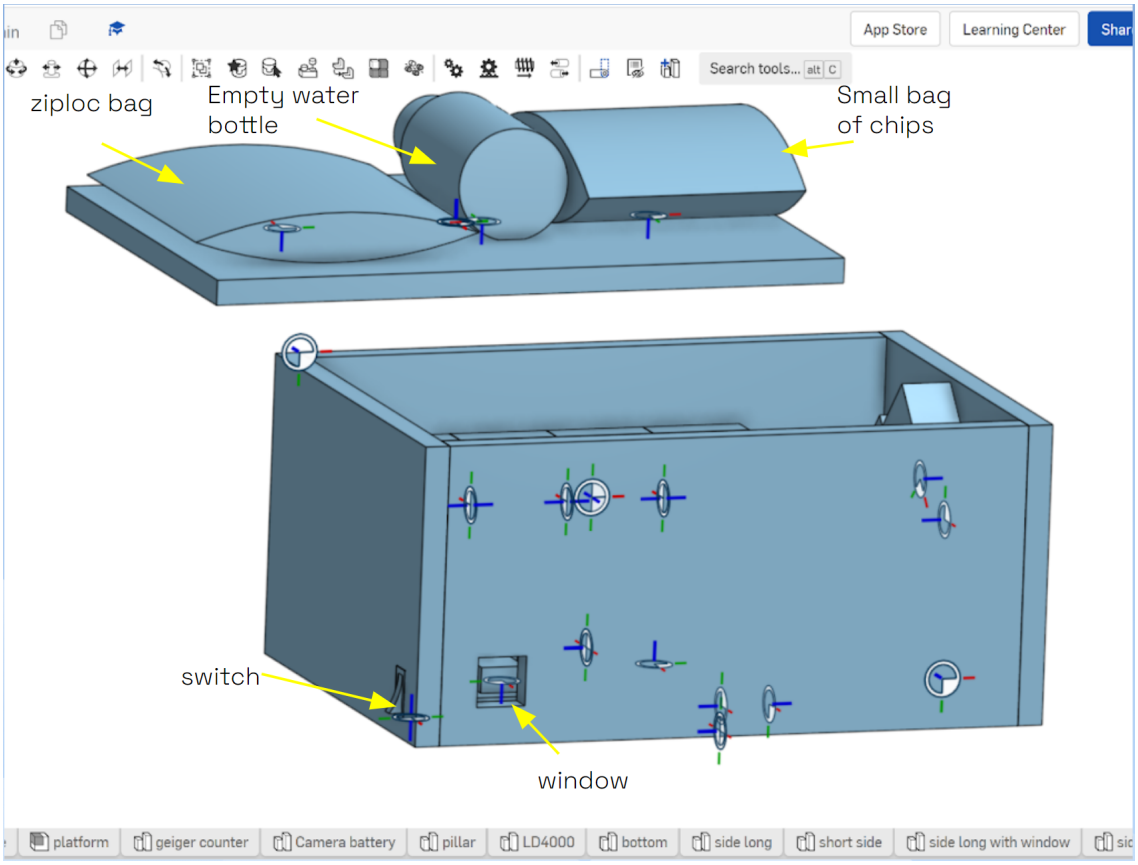
This is a side(ish) view of the payload; showing the camera and window for the PTERODACTYL on the sides of the box, the 9 volt battery, geier counter and Neulogs inside of the box, and the ziploc, water bottle, and bag of chips on top of the lid.



This is the isometric(ish) view of the payload; showing the switch to the PTERODACTYL on the sides of the box, the camera, camera battery and Neulogs inside of the box, and the ziploc, water bottle, and bag of chips on top of the lid.

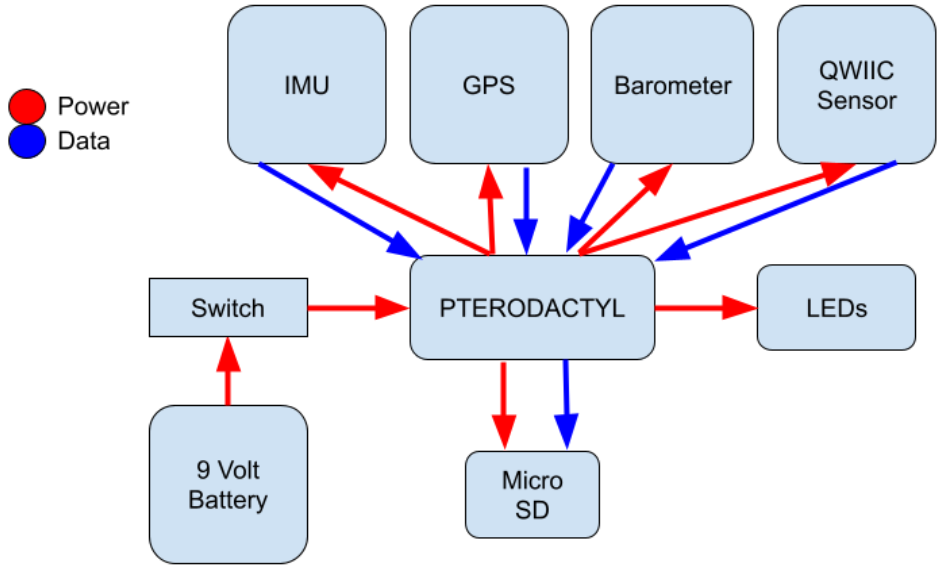


This is a top view of the payload; showing the switch to the PTERODACTYL on the sides of the box, the camera, camera battery, 9 volt battery, geier counter and Neulogs inside of the box, and the ziploc, water bottle, and bag of chips on top of the lid.



This is a side(ish) view of the payload; showing switch and window for the PTERODACTYL on the sides of the box, and the ziploc, water bottle, and bag of chips on top of the lid.

PTERODACTYL System Wiring Diagram:



4.0 Project Management and Schedule

Schedule (up to launch):

Oct 5: presentations, begin building payload

Oct 10: short meeting to discuss payload (meet at 3pm)

- Items discussed:
  - Change experiment → Test efficacy of different seals under different pressure conditions as they reach near-space
  - Use plexiglass window instead of LED lights to see if everything inside payload is working properly

Oct 12: FRR template, work on payload

Oct 14: meet up to work on Rev A (meet at 10pm), finished rev A

Oct 15: team project documentation **Rev A due @5pm!**

Oct. 18: Abbie and Jayleigh met up to finish building the payload box, James and Jack met to work on code

Oct 19: flight prediction software and monitoring

Oct 21: **meet up to work on FRR and testing (3pm)**

- Items discussed:
  - Change camera location: original CAD had it facing down, we decided to change it to face out the side of the box since the other payload in our stack has a downwards view
  - Ensured experiment items (bag of popcorn, empty water bottle, and snack size ziploc

bag) fit on box and meet weight requirements

Oct 24/25: finish FRR (meet at 3pm) finalize payloads

- Items discussed:
  - Conducted pre-flight testing
  - Each team member is responsible for revising their sections from CDR for the FRR slides by noon October 26
    - Assigned remaining sections as well

Oct 26: present FRR, miscellaneous

October 27: meet one last time

- Items to discuss:
  - Final testing
    - Ensure Neulogs are functional and set to proper settings (ie: not cumulative Geiger Counter measurements)
    - Review camera settings
    - Meet with Seyon to confirm code
  - Review PTERODACTYL code
  - Review roles and responsibilities for launch day

Oct 28: final weigh-in deadline

Oct 30: launch!

Schedule (post launch)

Nov 2: Receive payload post flight, and start downloading data and videos

Nov 4: work together to finish Rev B

Nov. 5: Rev B due @5pm

Nov:15: Work on data analysis

Nov.16: Data Analysis due @1 pm

Dec2: work to finish Rev C

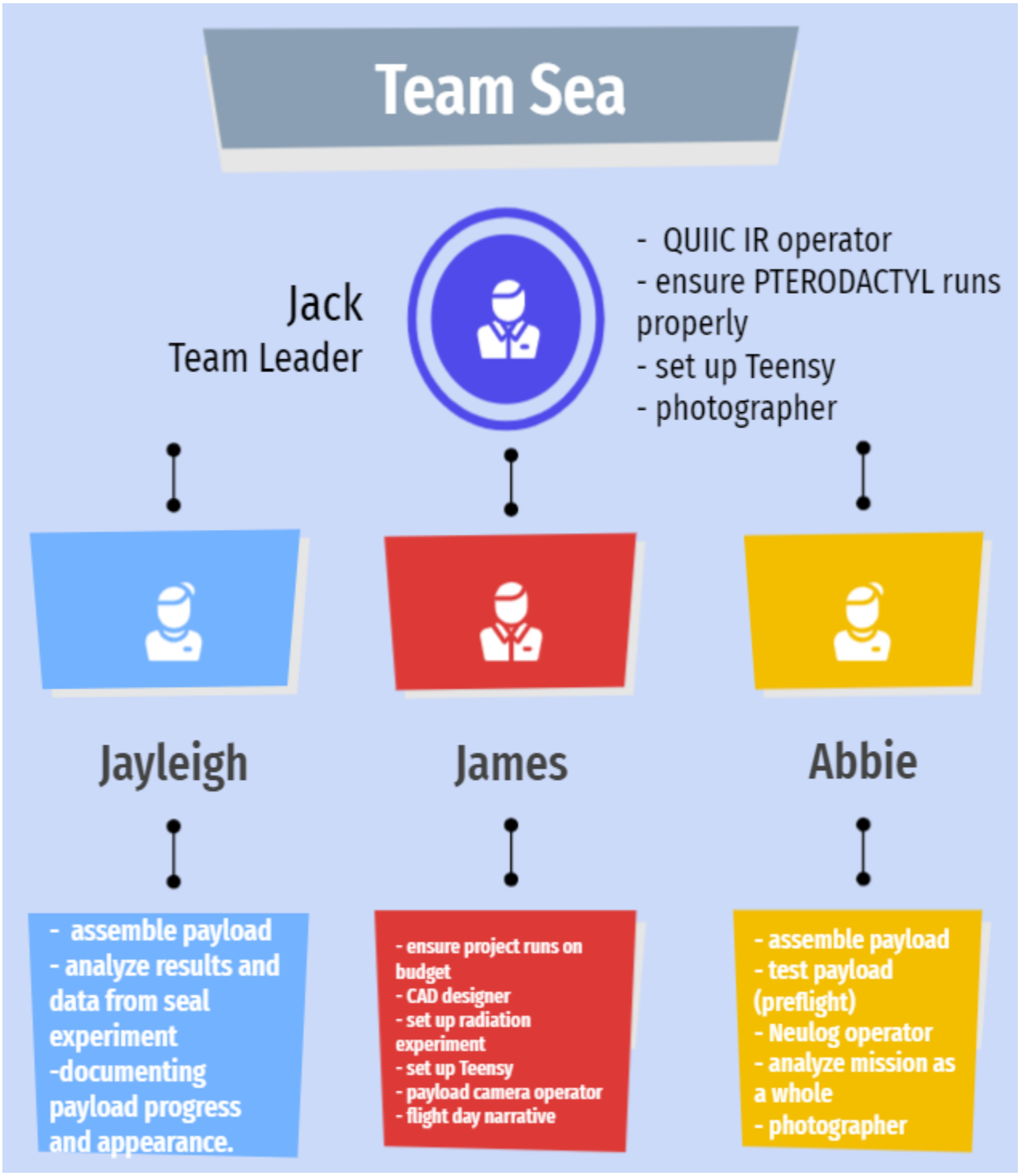
Dec. 3:Rev C due @5pm

***Bolded items:** out-of class meet-ups

** underlined items: important events

*** red items: deadlines

Team Responsibilities:



5.0 Project Budgets

Money Budget:

	Rigging Tubes x4	Strapping Tape	Duct Tape	Braided Mason Twine	Key Rings x4	32 Gig MicroSD Card	PTERODACTYL	breakout boards and components	9-volt battery x2
Money (in dollars)	1	1.50	3.50	1	1	8.74	200	178.60	16

	Neulog battery module	Neulog wide-range temperature module	Neulog pressure module	Neulog Geiger counter module	Lightdow LD4000 camera	Anker battery pack	Styrofoam 7”x7”x1/2” x10	QWIIC Heat Sensor
Money (in dollars)	45	64	83	500	40	20	10	35

	Water Bottle	Ziploc Bag	Bag of Chips
Money (in dollars)	0.5	0.05	.5

Total Cost
$1210.39

Mass Budget:

	Rigging Tubes x4	Strapping Tape	Duct Tape	Braided Mason Twine	Key Rings x4	32 Gig MicroSD Card	PTERODACTYL	breakout boards and components	9-volt battery x2
Weight (in oz)	0.5	2	8	0.4	0.7	negligible	4	Included	2.4

	Neulog battery module	Neulog wide-range temperature module	Neulog pressure module	Neulog Geiger counter module	Lightdow LD4000 camera	Anker battery pack	Styrofoam 7”x7”x1/2” x10	QWIIC Heat Sensor
Weight (in oz)	2.3	1.7	1.7	4.7	1.8	3.1	5	0.1

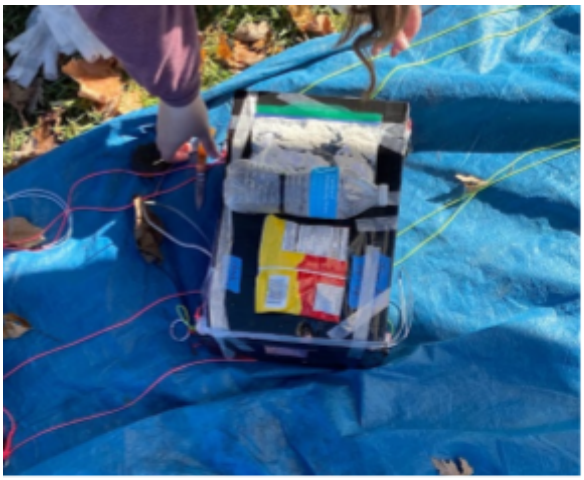
	Water Bottle	Ziploc Bag	Bag of Chips
Weight (in oz)	~0.3	~0.07	~1

Total Weight
39.77 oz **~2.45 lbs**

Actual Total Mass: 38.1 oz

6.0 Payload Photographs

The payload sitting outside during testing. Payload has not been painted black and items have not been fully secured yet.



Payload on Flight Day, fully secured and ready to go. Items on top of payload (from top to bottom) are: ziploc bag filled with flour and binder paper shreds, empty 16 oz plastic water bottle with lid tightened by hand, personal size bag of Cheese-its Snapped. Payload is secured to Team B's payload (left) and to the upwards facing camera (right).

The payload on flight day with all interior items secured (except batteries). Paracord strapping is being untangled in order to secure our payload to Team B's payload.

The stack after it landed. Team A's payload furthest back, then to the bottom left of that is Team B's payload, to the right of Team B's payload is the upwards pointing camera box, below that is Seyon's payload, and the closest payload is the SatCom.

The interior of our payload and the lid next to it on flight day. This is prior to adding the finishing touches of attaching our batteries, turing on the PTERODACTYL/camera/neulogs, finishing the riffing between payloads, and taping our payload completely shut.

7.0 Pre-Flight Ground Testing Plan and Results

Testing Plan:

The main components of this payload include three Neulog sensors, a camera, and the PTERODACTYL. The Neulog sensors include pressure, temperature, and a Geiger Counter. These require a battery attachment, which immediately starts running once plugged into the other sensors, even if the sensors have not started recording. This calls for careful execution on launch day, and plenty of planning beforehand on behalf of designing the structure of our payload and determining how we organize our flight day procedures. To test the entire process, at least two simulations will be performed to ensure the accuracy of our payload and thus the flight.

For testing the Neulogs, all sensors will be plugged together, then the battery will be plugged in, and finally two people will simultaneously press record on the three Neulog sensors. This system will run

for at least one hour, under different conditions throughout. Afterwards, the data will be reviewed to see how the battery life lasted and if valid data was collected for the correct length of time. More specifically, to test the pressure sensor a syringe will be attached to the sensor, allowing for easy manipulation of pressure by creating a make-shift vacuum. This will change the pressure to the best of our abilities without actually running a flight. For the temperature sensor, the box will be placed indoors, then moved outdoors, and possibly in the shade and then in the sun to assess the temperature in the different environments. For the Geiger Counter, a radioactive material will be placed directly in front of the sensor for the first twenty minutes, then one inch further for the next twenty minutes, and another inch for the last twenty minutes. This concludes the test planning for the Neulog sensors.

Next, the camera must be tested. In order to do so, a similar simulation to the Neulogs will be conducted. During every trial, the camera will be turned on and the recording will be started before the Neulogs are plugged in, and it will remain running until the trial is completed and nuelogs are turned off. Completing the trial in this order we will be attempting to simulate actual events that will occur on the flight day because due to battery power the camera and Neulogs will be the last things we turn on before flying. Additionally, throughout the camera trials, the camera will be facing sideways to simulate how it will be structured within our payload. This test will reveal data on how the camera functions in the same various environments that the Neulogs are tested in.

Next, the PTERODACTYL must be tested. To test the GPS system, two people will take the payload and walk around campus. Then, the other two people will use the GPS system to try and track down the payload and other two team members. This is good practice for locating our payload on Launch Day. The gyroscope on the PTERODACTYL will also be tested by changing the box's orientation during the different trial intervals. This will simulate the process of gathering data to record our payload’s orientation throughout the duration of the flight.

Test Plan List:

1. Camera on
2. Camera record
3. PTERODACTYL running (verify connections)
4. Neulogs connected
5. Add Neulog battery
6. Press record on all Neulogs
7. Begin hour long testing
    a. 0 min in:
        i. Box location (inside)
        ii. Box orientation (right-side up)
        iii. Radiation distance (0 inch)
    b. 20 min in:
        i. Change box location (outside, in sun)

   ii. Change box orientation (sideways)
   iii. Change radiation distance (1 inch total)
c. 40 min in:
   i. Change box location (outside, in shade)
   ii. Change box orientation (upside down)
   iii. Change radiation distance (2 inches total)

8. Unplug Neulogs
9. Stop Camera recording

\+ Hide-and-Seek GPS Scavenger Hunt
\+ Camera battery life test

Testing Results:

Note: Before discussing the results, it is important to note the modifications that were made to the testing plan.

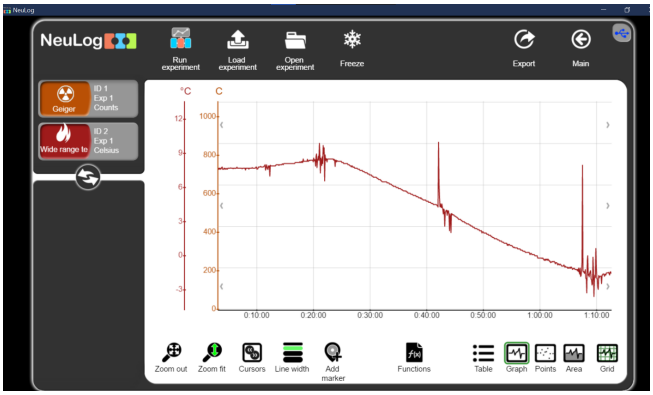
The first change involved the Neulog Pressure Sensor. While the primary test was still conducted, a Solder Sucker was used instead of a syringe. This meant that only two different pressure values were able to be achieved, but the results still gave enough information to ensure accuracy and functionality of the sensor. This change was made due to accessibility of materials. The second change that was made was to the Neulog Geiger Counter. During the hour-long test, no extra radioactive material was introduced. Instead, natural radiation was measured and the sensor was tested for longevity more than accuracy. Later, a separate test was conducted in which the Neulog was connected through a laptop to see live results. A radioactive material was placed directly in front of the sensor for ten seconds, then one inch further for another ten seconds, and another inch for the final ten seconds. This change was made due to accessibility of materials and practicality of the experiment, as it would be difficult to achieve a correct distance away from the Geiger Counter during the hour-long test and ensure that it remained throughout the duration of the test.

One test that was not able to be run was the GPS Hide and Seek test. Unfortunately, due to time restrictions and technology limitations we were unable to practice finding the payload, which would have simulated finding it after it landed. However, we are confident that with the help of Professor Flatten, our TA Seyon, and the extra-curricular ballooning team students, all of which have completed many balloon launches and recoveries, we will be successful in locating the final position of the payloads.

Results:

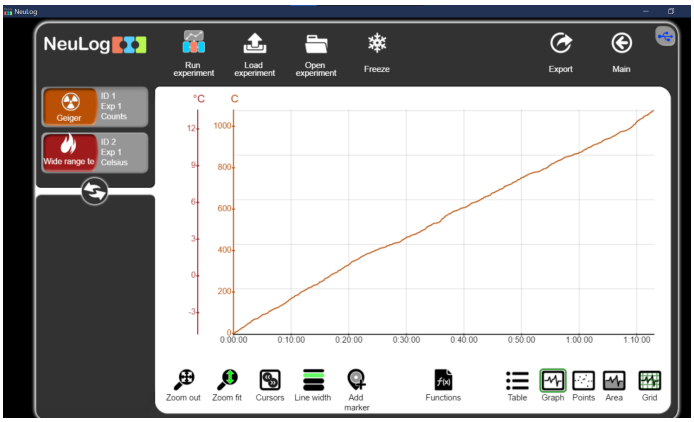
Neulog Temperature: The sensor shows that the temperature was steady and at its highest values during the first twenty minutes, which makes sense since that is when it was indoors. However, while the trend matches what we would expect, the Celsius values are too low. After the first twenty minutes, the sensor reads that the temperature continually decreased until we stopped the test, which

according to the Neulog was around 0℃. The temperature should be colder since it was outside the last forty minutes, but not that cold. After seeing these results, we concluded that the sensor was malfunctioning because it was gathering the wrong temperatures, so we replaced it with a new Neulog Temperature sensor.



Neulog Pressure: The pressure sensor turned off shortly after our experiment began, but this was not discovered until the test had been concluded. So, we ran the Solder Sucker test again directly after the hour-long test. Since the pressure consistently dropped with the Solder Sucker acting as a syringe, we concluded that the pressure sensor was functional. The next day, we also went into the Neulog application and changed the settings so it would gather data for 5 hours instead of 1 minute. We tested this by running it for ten minutes to ensure that it wouldn't turn off, and it passed this test.

Neulog Geiger Counter: From the hour long test, it was discovered that the Neulog Geiger Counter ran for the proper amount of time, but was set to an accumulative setting. To change this, we went into the Neulog application and changed the settings to count/second instead of just count. We also ran the secondary test with a radioactive material at various distances away from the sensor, in which the data correctly corresponded to. So, we concluded that the Geiger Counter was ready for Flight Day.



Camera:
The camera test went to plan. We tested it for an hour both indoors and outside and it worked perfectly. We also ended up doing a second test later with everything together and it continued to work.

PTERODACTYL:
The PTERODACTYL had a GPS, IMU, QWIIC sensor, thermometer, and a clock. The clock on the PTERODACTYL didn't work at all, it recorded the hours, minutes, and seconds as zero. The GPS also didn't work, recording latitude, longitude, and altitude as zeros. The QWIIC temperature sensor recorded data; however, the data does not seem reliable because according to the sensor the atmospheric temperature changed from -111.66 degrees to 25 degrees over the course of 100 data points which should correspond to about one and a half minutes of flight time. The included thermometer recorded data that seems reliable compared to our expected results. The IMU recorded data that seems reliable for the accelerometer, magnetometer, gyroscope, and pressure.

8.0 Expected Science Results

As a result of our payload testing we have accumulated different expectations as to what the outcome of our data will be from our flight. Primarily, we expect all of our Neulog Sensors to collect data throughout the flight. This expectation is derived from the failures we had through our testing, and how after having those revelations occur we were able to identify our problems and work together, as well as with Professor Flatten and our TA to create solutions.

In relation to actual quantitative data, our research and testing has led us to several expectations here as well. One expectation that pertains to the temperature data collected throughout the flight's duration. Through researching we found that throughout the stratosphere, temperature acts on a wave; steadily decreasing as you move up and through the troposphere when it turns and begins to increase in temperature through the stratosphere. This trend is one we expect to see reflected within our own data, pending on the highest altitude point our payload reaches.

Due to complications with our pressure Neulog Sensor we also had to turn to research to determine how pressure would be impacted throughout our flight. This is a very important factor as the basis of our experiment relies on how the altitude of our balloon and payload influence the pressure of the sealed objects. In our research we found that as altitude increases applied pressure decreases, and because of this we expect the pressure within our sealed items to increase with altitude because there will be less force pushing back against the seals giving them room to grow and expand their internal pressure until they (hopefully) reach a point where they burst/open because the internal pressure became too much.

From testing our payload we were able to see that our Giger Counter Neulog does sense and record radiation levels. In researching, we found that several forms of radiation can potentially be present during the flight, but the influence and occurence of the varies depending on extraneous variables. It can still be said that there is generally a positive increase of radiation moving up with altitude. After learning all of this we expect to see various influxes of radiation that overall increases throughout the flight's accent.

9.0 Flight Day Narrative

Flight day was a unique experience. It started off very slow, had a high speed chase in the middle, and ended as it started, with a long drive and a lot of waiting. We began early in the morning on campus with a quick check in and made final preparations for the flight. However, my team found out that for reasons unknown our gps wasn't working. There was nothing really we could do at the time but let the more experienced ballooners handle it. We sealed our box and Professor Flaten explained a few changes he had made since our final weigh in. After everyone had finished we left for the launch site which was about an hour and a half away.

As we all arrived at the launch site we began to get everything ready. We did some final checks on our camera as well as the cameras from the payload directly above us and, although our gps still wasn't working after being worked on by Paul (the head of the ballooning team) and Seyon, sealed the top on the box. After that, we attached the payloads on the stacks together with cables and then started to inflate the balloons. Inflating the balloons is a process that involves one person seated on a tarp with the balloon tied to it to ensure it does not fly away and several other people standing around with gloves (skin oils mess with the balloon material) to keep the balloon in place. The balloon is inflated with large helium canisters placed in the bed of a pickup truck. After inflating the balloon everyone holds one of the payloads and one person slowly starts letting the balloon up in the air, one payload at a time. After everything is up in the air the balloons are released.

After the balloons were released we watched them for a few minutes and then packed everything up so we would have time to hopefully reach the balloons before they reached the ground. We had a quick lunch and then began to track the balloons through two different tracking websites and attempted to predict their landing path based on the projections given to us by Professor Flaten. When we saw the balloons were almost at their max altitude we got out to watch them pop and then the race was on. Tracking the balloon was the most intense part of flight day. We had 2 people on the different

tracking websites, one comparing roadmaps with the trajectory of the balloon and communicating it with the driver, and one person communicating with the other team. The shape of the flight path was about the same as the projection but shifted almost 50 miles east. The most difficult part was attempting to take the right roads since we obviously can't just offroad in the university vans. We had to take several unpaved roads and, while we got there before the other van, unfortunately we did not get to see the landing because the balloon landed on someone's property.

If the tracking was the most intense part of the day retrieving the balloon was the most stressful. The balloon had landed on a farm and the landowner was very skeptical of us tracking it down while his bull was roaming free. Eventually we convinced him to let us on his property and we had to offroad across his property in our van. Luckily the balloon landed on the ground about six feet from a grove of trees so it was an easy recovery. We collected the payloads, shut off the cameras, and after the other van found the research balloon, called it a day and headed home.

Blowing up balloons

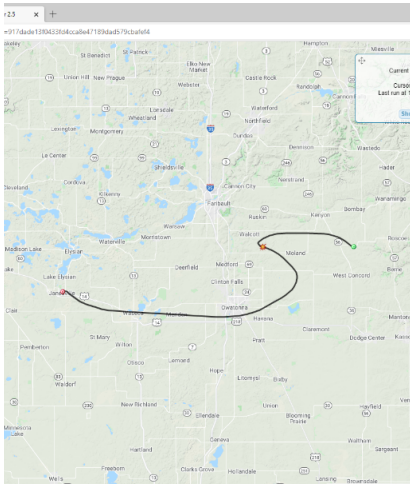
Releasing Balloon

Recovering balloon

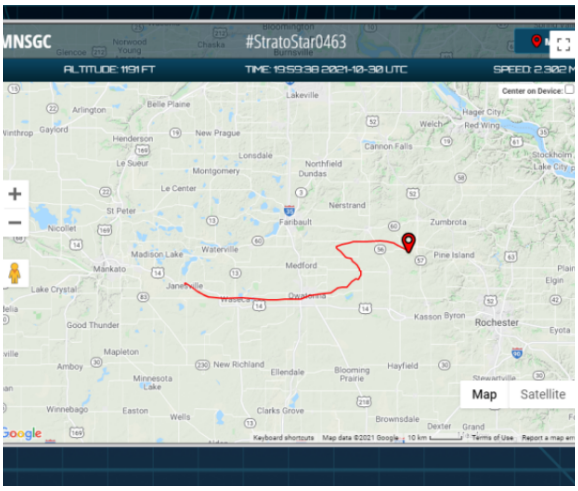
Team Picture with Payload

10.0 Results and Analysis

First, it is important to look at the balloon stack's predicted flight path and it's actual flight path. We released the balloons in Janesville, Minnesota and predicted that they would land between highway 56 and 57 near Kenyon, MN. It landed only a few miles east of where we predicted.



Predicted Flight Path
(CUSF Landing Predictor)

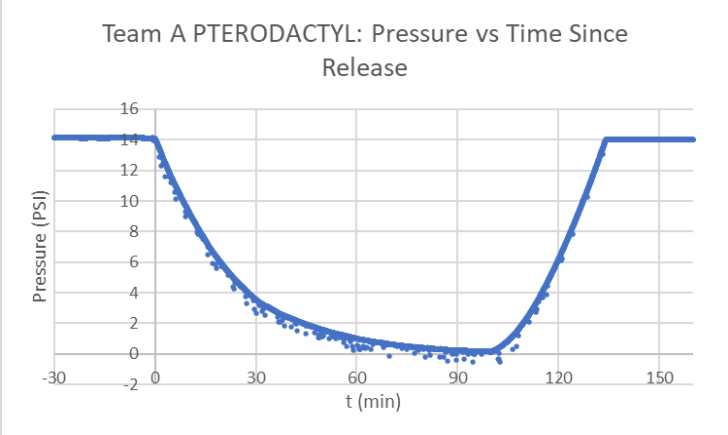


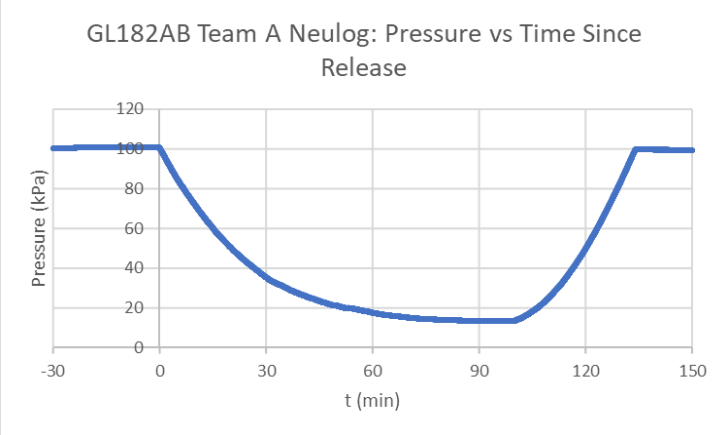
Actual Flight Path
(Stratostar)

After putting the flight in context, the results of experiments and data can be analyzed. There are two main ways to analyze data: compare it to time or compare it to altitude.

By comparing data with time, we can see how variables change throughout the duration of the flight. Some variables we can compare are: pressure, temperature inside the payload, temperature outside the payload, Geiger counts, rotation from accelerometers, and data from QWIIC sensors.

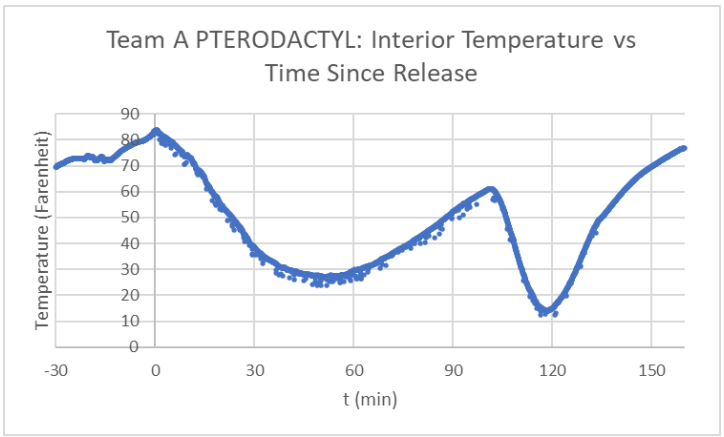
Pressure vs Time:

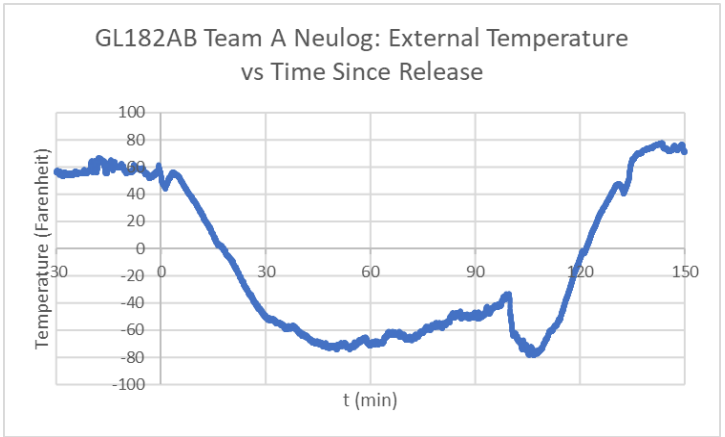




The PTERODACTYL and Neulog pressure sensors reveal similar trends. As time initially increases the pressure decreases, and then at a certain point it increases at a quicker rate than it was decreasing. In the context of ballooning, the pressure is stable before the balloon is released and then drops as the balloon climbs in altitude. Then when the balloon bursts the payloads fall back to the ground and pressure increases, stabilizing once again after it lands. So, it can be determined that the pressure was about 14 PSI or about 100 kPa and that the balloon burst at around 100 minutes. One discrepancy between the two graphs is that the PTERODACTYL measured a lower minimum pressure (about 0 PSI) than the Neulog sensor did (about 15 kPa). The Neulog Sensor was not designed to go to near-space, so it is likely that it loses accuracy at such extremes.

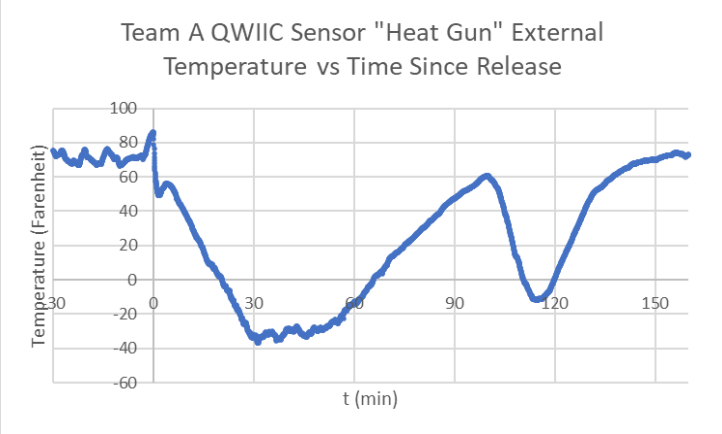
Internal and External Temperatures vs Time:

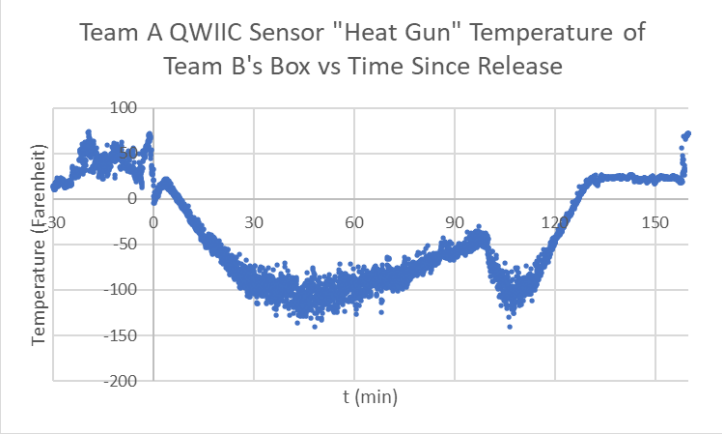




The interior and exterior temperatures varied greatly throughout the flight, but followed similar trends. However, the internal temperature remained significantly warmer than the external temperature did during flight. This makes sense since the payload was constructed from thermal styrofoam, covered in black tape to increase heat from the sun, and contained hand warmers to keep the batteries warm. However, it might not have been quite as warm as the Neulog suggests since the Neulog thermometer was not well calibrated or designed for such extreme conditions as seen in near-space. The external temperature is expected to be extremely cold as it is exposed to the elements of near-space. The trends of the various temperatures will be more thoroughly explored when compared to altitude.

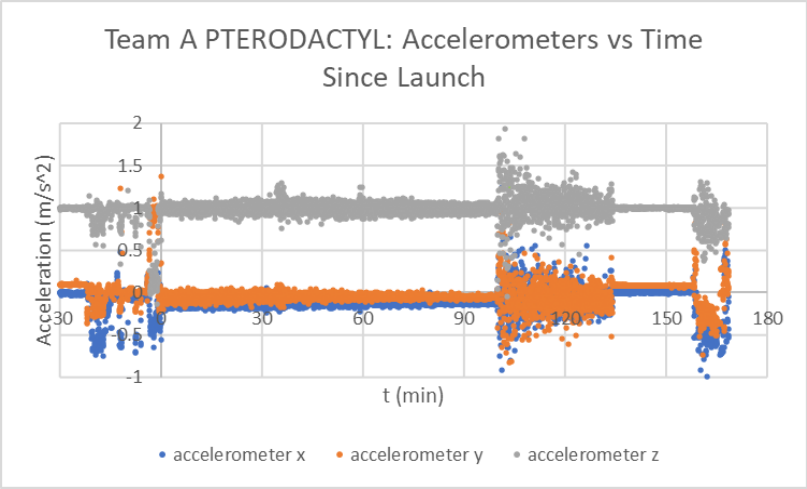
QWIIC Sensor Temperatures vs Time:





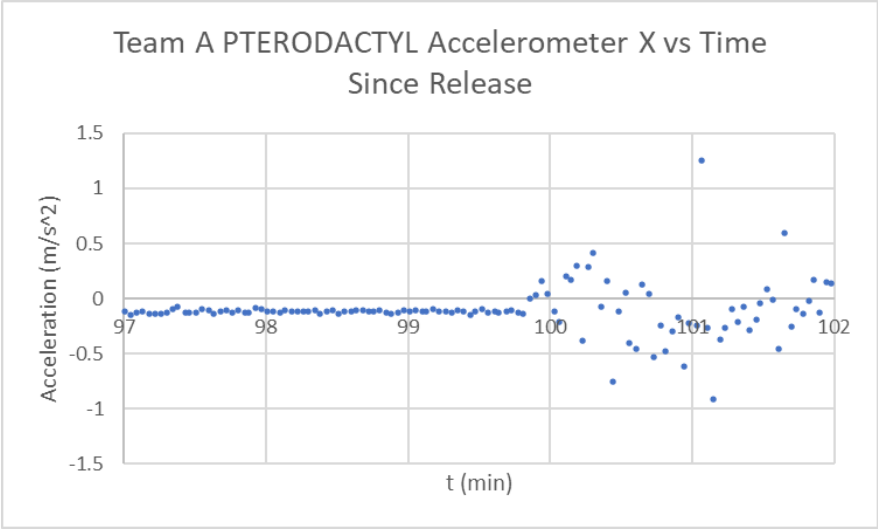
This was the first time this QWIIC sensor has been flown for this class. The QWIIC sensor temperatures followed similar trends to the internal PTERODACTYL and external Neulog sensor temperatures. However, the external ambient temperature from the QWIIC sensor gave a higher minimum temperature of about -40℉ compared to Neulog's -80℉. The QWIIC sensor temperature climbed a lot higher from 30 minutes to 100 minutes, reaching a relative maximum of 60℉ at 100 minutes; at 100 minutes the external Neulog sensor read -40℉, this shows a large discrepancy between the two sensors. The QWIIC sensor follows the same general trend of temperature differences that is expected when increasing the altitude but not the same temperature itself.[2] So, it is possible that the QWIIC sensor was miscalibrated or did not work. It also measured Team B's box to get to around -150℉, which is way too cold for this flight. It is possible that it didn't always point directly at their box since the payloads twist and bobble throughout the flight, but regardless it did not accurately work.

Accelerometers vs Time:

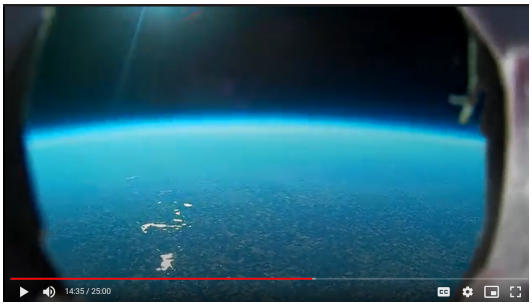


The payload experienced the most acceleration (about 1m/s^2) in the z direction which is up. As the payload flies, it twists and turns with wind and general turbulence causes random small accelerations in the other directions. The acceleration before release occurs when someone picks up or moves the payload and during the flight the accelerations are relatively constant. However, at about 100 minutes and about 160 minutes, there are significant amounts of movement. The first is when the balloon bursts and the payloads fall towards Earth, which comes to an end when the payloads hit the ground. The second is when we retrieve the payloads, picking them up and transferring them to a car.

By looking at one accelerometer more closely, the exact moment the balloon bursts can be determined.

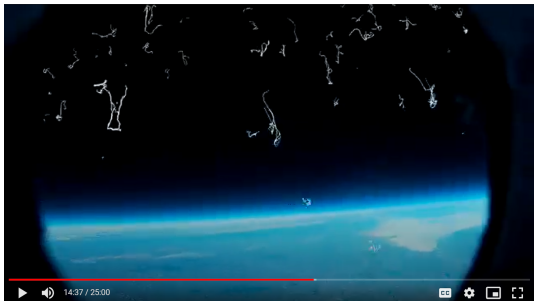


The acceleration is relatively constant until just before 100 minutes. This must be when the balloon bursts. Taking this information, we can then go look at the camera log and find footage of the box rotating and the balloon bursting.

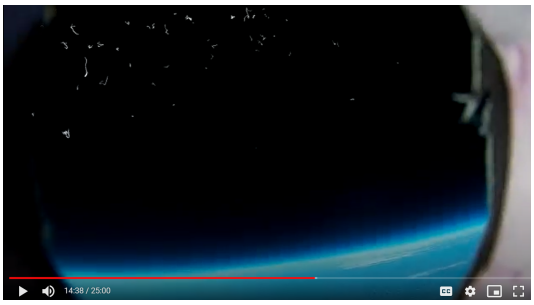


99:49
The box is relatively stable and the balloon has not yet burst.

99:50
The box is stable but the balloon has burst.



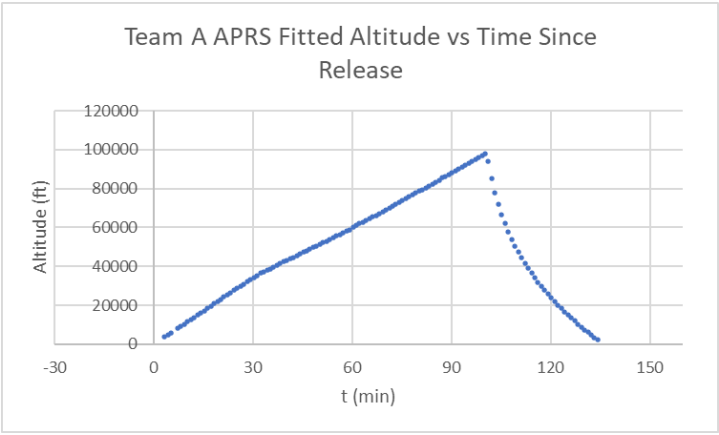
99:51
The box undergoes rotation.



99:52
The box undergoes more rotation.

To get an even greater understanding of the flight's movement, one can compare time and altitude.

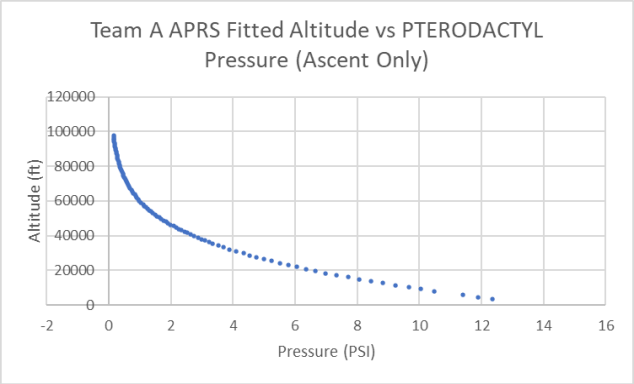
<u>Altitude vs Time:</u>

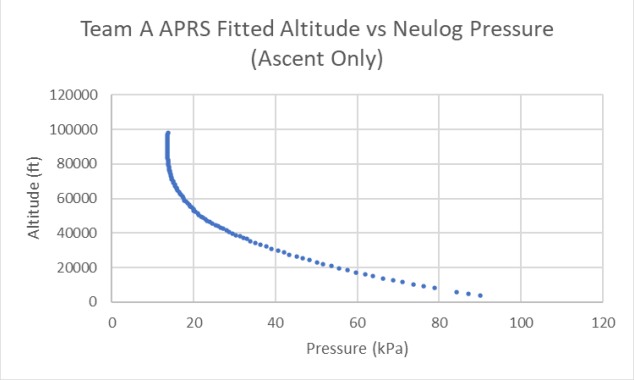


The GPS and altitude on the PTERODACTYL did not work properly, so data from APRS was fitted to give altitude. Based on this graph, the balloon increased altitude at a steady rate unit about 100 minutes when it burst. Then it decreased quickly and slowly decreased at a slower rate until it landed at about 130 minutes. It also took longer to ascend than it did to descend. This makes sense because the ascent is controlled by the helium in the balloon and the descent is violent and only countered by air resistance.

It is also important to look at data compared to altitude. When viewing data compared to altitude, it is often helpful to view the data during the ascent and descent separately.

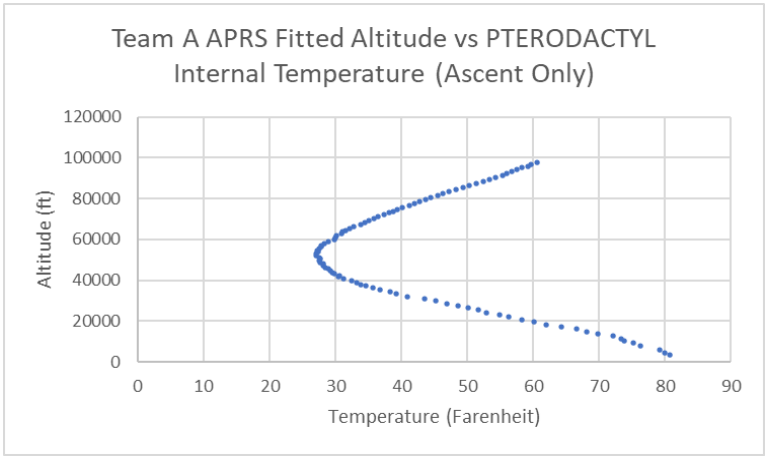
Altitude vs Pressure:

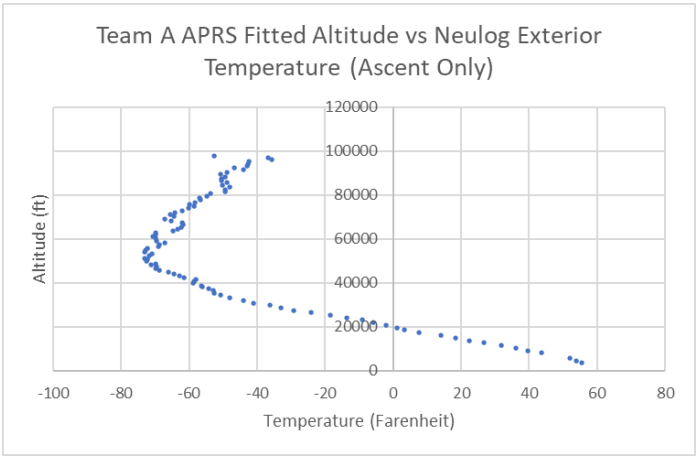




As the altitude decreases, the pressure increases. The air is denser, and thus has a higher pressure, towards the surface of the earth because of gravity. Less air molecules are present near the edge of the atmosphere in near-space. Similar trends can be similar between the PTERODACTYL and Neulog sensors, but just like when compared to time, the Neulog does not reach as low of pressures.

Altitude vs Internal and External Temperatures:

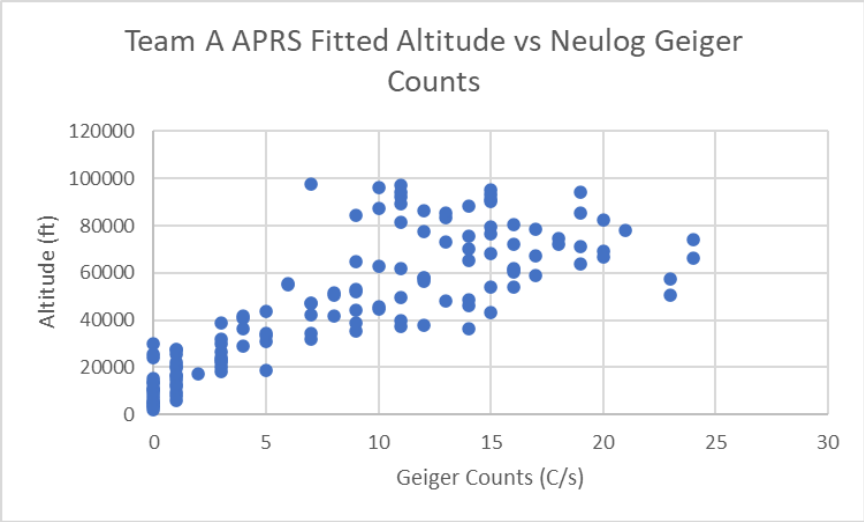




Both the internal and external temperatures show similar trends. Starting at the lowest altitude and working up, the temperature decreases at first. However, around 50,000 ft the temperature starts to

increase. This occurs because of the transition from the troposphere to the stratosphere. Once again, the internal temperature is warmer than the external temperature.

Altitude vs Geiger Counts:



As the altitude increases, the Geiger Counts/second also increases. This is because as the altitude increases the atmosphere gets thinner, leaving less particles to shield radiation.

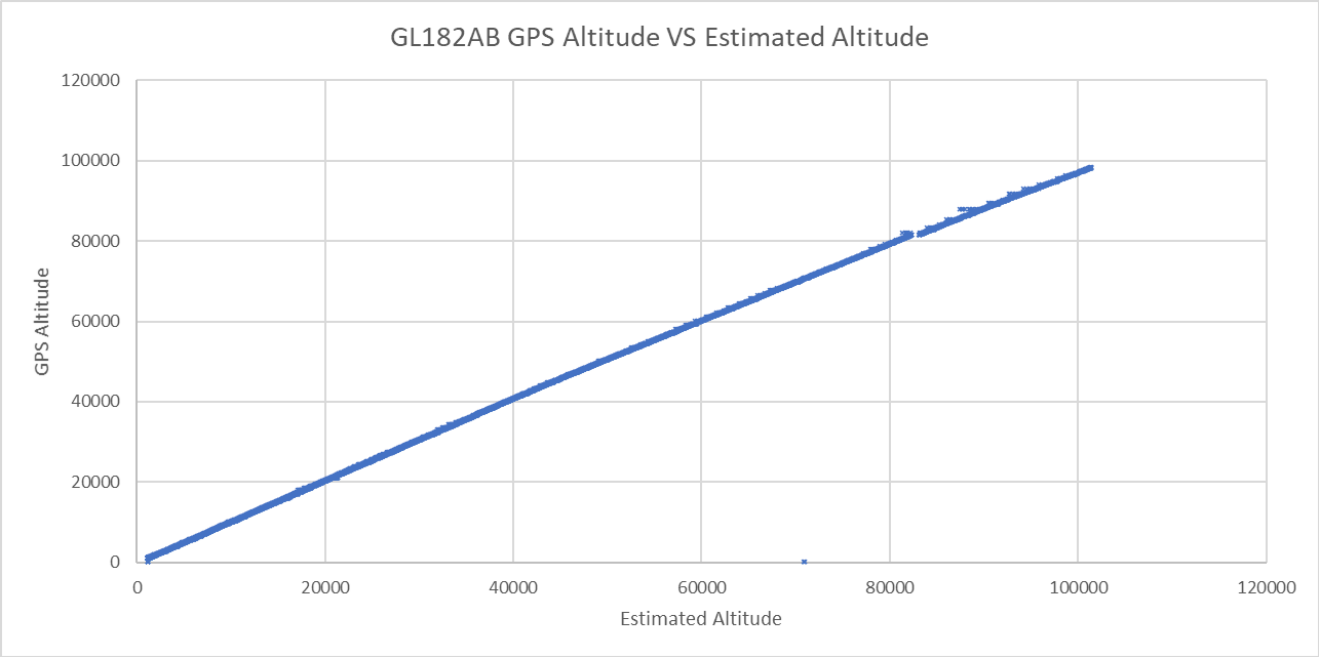
While most data was stored onboard the payload (the reason it is so important to recover the payload after the flight), we also sent some data down to the surface throughout the flight. This is called telemetry.

Telemetry:

Satcom: 10111111

We used 8 bits to tell us about the conditions and the status of our sensors about every 2 minutes. A 1 means a sensor is working and a 0 means it is not. This string from Satcom was prevalent throughout the entire flight. The first 1 is an identifier to distinguish our payload's string from other payloads. Then, the rest of the bits tell us that our gps wasn't working, but that the SD was logging, the internal temperature wasn't too high, the internal temperature wasn't too low, the QWICC sensor was gathering data on Box B and ambient temperature (although the accuracy of that data is less certain), and an accelerometer was working. This is accurate to what we found from the data logged onboard, meaning the information was correctly sent and parsed from the payload through multiple satellites and eventually to us on the ground

Real vs Estimated Altitude:



As can be seen from the above graph the estimated altitude is almost identical to the GPS altitude which confirms it was reading altitude correctly.

Bit Flip experiment:

In the bit flip experiment there were over 10.3 million zeros written in a txt file on the SD card in the PTERODACTYL. If an energized particle were to hit one of the physical bit cells in the part of the SD card that contained this file, then one of those zeros would have changed to a different character. The file after the flight only contained zeros in it. Thus, new energized particles hit one of the physical bit cells that the file was written on. Though theoretically the chance of a bit flip increases when altitude increases, due to the higher radiation, the likelihood of one happening is still low as shown by the absence of one in the file. Unfortunately we did not see evidence of any bit flips after the flight. This is to be expected, however, we realized that if we had more time it would have made sense to send the file through the satcom repeatedly so we could see the numbers live and compare it to radiation data.

Seal experiment:

In the seal experiment an empty water bottle, small bag of chips, and a ziploc bag were all sent up to 100,000 feet. During the flight there was a camera recording video of the three objects, and there was also a pressure sensor recording the pressure. The goal of this experiment was to use the video to determine when each object would burst open and then compare that to the pressure at which each burst. The results however, were surprising; none of the three objects bursted during the accent of the balloon. These results show that the seals of the water bottle (and the structure of the bottle itself) and the small bag of chips can withstand the near absence of external pressure. The ziploc bag however, never fully inflated either meaning that the seal was never fully sealed or that there was not enough air put into the bag before launch.

This shows the state of the three objects one second before the burst of the balloon.

Camera of Team B, 99 minutes and 49 seconds after release on the balloon.

11.0 Conclusions and Lessons Learned

Overall, as a group we were able to learn a lot about the change in the change of factors such as altitude and time compared to that of factors such as pressure, radiation, and rotation. We were able to assess how our internal and external temperatures followed the same trends over time and that our insulation tactics were successful as the internal temperatures were warmer. Additionally, we were able to see how over time the pressure recorded by the Neulog sensor and PTERODACTYL followed similar trends, decreasing temperature as the payload went deeper into the stratosphere and increasing as the payload was within the troposphere. We were even able to see trends in our acceleration, and how we can see exactly when the balloon popped just by looking at the graph and can even see how it rotates as it increases altitude, which can be visibly seen in our captured video footage. While all these things worked well for us and we were able to draw conclusions and analyze data there were some errors that if given the opportunity to redo we'd go back through and fix. One of which was that the altitude Neulog sensor and the altitude being recorded by the PTERODACTYL did not work so we had to adapt the APRS tracker's altitude as a substitute. Another aspect of our flight that didn't go as expected was the QUIIC IR sensor. Given, we weren't exactly sure what to expect flying them as they hadn't been flown within this class yet, the heat gun temperature sensor within it did not record accurate data that could be used despite it following the same trends as our other temperature sensors. If given the opportunity to do this again and retry would be the plastic ziplock bag portion of our pressure experiment. After looking over Team B's footage looking at our payload we were able to see some pressure reactions from the bag of chips and water bottle, and even though it wasn't the reaction we were expecting we were still able to see something happening with

them. This did not happen with the ziplock bag so given the opportunity to try again we'd make sure the ziplock was completely closed first and filled up with a bit more starting air before launching the payload. In the end while we didn't receive the results we were expecting exactly, we still had an overall successful flight.A final thing we could have improved on is developing a system for exporting our bit flip file with the satcom throughout the flight. This would have allowed us to know exactly if and when a bit flipped and compare it to radiation. Looking back there is definitely room for improvement dating all the way back to our original designs through actually constructing the payload to flight day that with the knowledge we have now we probably could have done better, but that just goes to show how much of a learning experience this was for us and how despite those novice mistakes we were able to ultimately have a successful flight and useful data.

Words of Wisdom to a Future Ballooning Team

Expect things to go wrong- It may seem like at the beginning of the class that building the payload and developing an experiment will be a pretty easy task. After all, so many balloons have already been flown that everything should be figured out at this point. This is just not true. Every flight, including the experimental payload flown by the actual ballooning team, had problems and it is important to anticipate that ahead of time.

Plan Ahead- Planning ahead is key to success. Continuing from the previous point, it is a lot easier to fix a problem two weeks ahead of the flight than on the day of. Staying ahead of schedule is key to success.

12.0 References

1 Heckman, Kurt. "Gravity Acceleration by Altitude." *VCalc*, 17 Mar. 2016, https://www.vcalc.com/wiki/KurtHeckman/Gravity+Acceleration+by+Altitude.

2 Engineering ToolBox, (2003). U.S. Standard Atmosphere. [online] Available at: https://www.engineeringtoolbox.com/standard-atmosphere-d_604.html [Accessed 12/1/2021].

13.0 Appendix: Program Listings

Main PTERODACTYL flight code:

The main PTERODACTYL code's purpose is to reference to all the other blocks of code and actually do something with them. This means that the main code will reference the SD and sensor blocks and then log the data from the sensor to the SD card. The main block will also send data by referencing the sensor block and the comms block. The main block also goes through the initial setup of the PTERODACTYL checking all the instruments.

```
// Main code block, contains setup(), loop(), and code to check the initial states of the slide switches and shorts
//  as well as variable declarations and logical code.
```

```
#define fixLED 26               // LED to indicate GPS fix
#define ppodLED 24
#define xbeeLED 27               // LED to indicate xbee communication
#define sdLED 25
#define serialBAUD 9600          // when using arduino serial monitor, make sure baud rate is set to this same value

#define ppodSwitchPin 30
#define satSwitchPin 31
#define commsSwitchPin 32
#define setAltSwitch 28
#define altSwitch 29
#define baroSwitchPin 9
#define id1SwitchPin 20
#define id2SwitchPin 21
#define id3SwitchPin 22
#define id4SwitchPin 23
#define pullBeforeFlightPin 16

int rfd900 = 1; // set true if you want this thing to operate with a rfd900 on serialX (declare above)
int ppod = 1; // set true if you want this thing to operate as ppod flight computer
int satCom = 1; // set true if you want to activate flight by waving a magnet over the IMU
int setAltVal = 1;
int altVal = 1;
int baroOn = 1;
int id1On = 1;
int id2On = 1;
int id3On = 1;
int id4On = 1;
int pullOn = 1;

//header that includes the data being logged.
String header = "Date, Hour, Minute, Second, Lat, Lon, Alt(ft), AltEst(ft), intT(F), extT(F), msTemp(F), msPressure(PSI), time since bootup (sec), magnetometer x, magnetometer y, magnetometer z, accelerometer x, accelerometer y, accelerometer z, gyroscope x, gyroscope y, gyroscope z, object temp, ambient temp";
unsigned long int dataTimer = 0;
unsigned long int dataTimerIMU = 0;
unsigned long int ppodOffset = 0;
```

```
int dataRate = 1000; // 1000 millis = 1 second
int dataRateIMU = 250; // 250 millis = .25 seconds
int analogResolutionBits = 14;
int analogResolutionVals = pow(2,analogResolutionBits);
float pressureBoundary1;
float pressureBoundary2;
float pressureBoundary3;
float pressureOnePSI;
float msPressure = -1.0;
float msTemperature = -1.0;
float altitudeFt = -1.0;
unsigned long int fixTimer = 0;
bool fix = false; // determines if the GPS has a lock

//////////////////// Sensor Global Variables ////////////////////////////

float thermistorInt;
float thermistorExt;
float magnetometer[3]; // {x, y, z}
float accelerometer[3]; // {x, y, z}
float gyroscope[3]; // {x, y, z}

float altitudeFtGPS;
float latitudeGPS;
float longitudeGPS;
String data;
String groundData;
String IMUdata;

String exclamation = "!"; // this needs to be at the end of every XBee message
String groundCommand;
String xbeeID = "NULL";
String xbeeProID = "AAA";
unsigned long int xbeeTimer = 0;
unsigned long int xbeeRate = 5000; // 10000 millis = 10 seconds
String xbeeMessage; // This saves all xbee transmissions and appends them to the data string

void setup() {

  Serial.begin(serialBAUD); //define baud rate in variable declaration above
```

```
 Serial.println("Serial online");
 pinMode(fixLED,OUTPUT);
 pinMode(xbeeLED,OUTPUT);
 pinMode(sdLED,OUTPUT);
 pinMode(ppodLED,OUTPUT);
 pinMode(13,OUTPUT);
 pinMode(setAltSwitch, INPUT_PULLUP);
 pinMode(altSwitch, INPUT_PULLUP);
 checkSwitches(); // slide and button switch status

 if(id1On==0)xbeeID="UMN1";
 if(id2On==0)xbeeID="UMN2";
 if(id3On==0)xbeeID="UMN3";
 if(id4On==0)xbeeID="UMN4";

 Serial.print("starting OLED setup... ");
 oledSetup();
 Serial.println("OLED setup complete");

 Serial.print("starting IMU setup... ");
 imuSetup();
 updateIMU();
 Serial.println("IMU setup complete");

 Serial.print("starting Altimeter setup... ");
 if(baroOn==1)msSetup();
 else{ updateOled("MS5611\nOffline.");
   delay(2000);
 }
 Serial.println("Altimeter setup complete");

 Serial.print("starting SD setup... ");
 sdSetup();
 Serial.println("SD setup complete");

 Serial.print("starting xbee setup... ");
 xbeeSetup();
 Serial.println("xbee setup complete");

 Serial.print("starting ublox setup... ");
```

```
  ubloxSetup();
  Serial.println("ublox setup complete");

 Serial.print("starting qtemp setup... ");
  qtempSetup();
  Serial.println("qtemp setup complete");

  pressureToAltitudeSetup();
  logData(header);
  if(pullOn==0) pullPin();
}

void loop() {
  updateData();
}

////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////// Functions  /////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////

void pressureToAltitudeSetup()
{
  float h1 = 36152.0;
  float h2 = 82345.0;
  float T1 = 59-.00356*h1;
  float T3 = -205.05 + .00164*h2;
  pressureBoundary1 = (2116 * pow(((T1+459.7)/518.6),5.256));
  pressureBoundary2 = (473.1*exp(1.73-.000048*h2)); // does exp function work??
  pressureBoundary3 = (51.97*pow(((T3 + 459.7)/389.98),-11.388));
}

void pressureToAltitude(){
  //float pressurePSF = (pressureOnePSI*144);
  float pressurePSF = (msPressure*144);

  float altFt = -100.0;
  //UNCOMMENT WHEN RELIABLE PRESSURE SENSORS ARE ON BOARD
  if (pressurePSF > pressureBoundary1)// altitude is less than 36,152 ft ASL
  {
    altFt = (459.7+59-518.6*pow((pressurePSF/2116),(1/5.256)))/.00356;
```

```
  }
  else if (pressurePSF <= pressureBoundary1 && pressurePSF > pressureBoundary2) // altitude is
between 36,152 and 82,345 ft ASL
  {
   altFt = (1.73-log(pressurePSF/473.1))/.000048;
  }
  else if (pressurePSF <= pressureBoundary2)// altitude is greater than 82,345 ft ASL
  {
   altFt = (459.7-205.5-389.98*pow((pressurePSF/51.97),(1/-11.388)))/-.00164;
  }
  else{altFt = -1.0;}

  altitudeFt = altFt;
  if(baroOn==0)altitudeFt = -1.0;
}

void updateData(){
  updateUblox();
  updateXbee();

  if(millis() - dataTimerIMU > dataRateIMU){
   dataTimerIMU = millis();
   updateIMU();
  }
  if(millis() - dataTimer > dataRate){
   dataTimer = millis();
   if(fix == true){
    digitalWrite(fixLED,HIGH);
   }
   digitalWrite(sdLED,HIGH);
   delay(30);
   digitalWrite(sdLED,LOW);
   digitalWrite(fixLED,LOW);
   pressureToAltitude();
   updateThermistor();
   updateqtemp ();
   if(baroOn==1) updateMS(); //Not every payload has one
   updateIMU();
   UpdateStatus();
   updateDataStrings();
```

```
    xbeeMessage="";
  }
}

void checkSwitches(){
  pinMode(ppodSwitchPin, INPUT_PULLUP);
  pinMode(satSwitchPin, INPUT_PULLUP);
  pinMode(commsSwitchPin, INPUT_PULLUP);
  pinMode(baroSwitchPin, INPUT_PULLUP);
  pinMode(id1SwitchPin, INPUT_PULLUP);
  pinMode(id2SwitchPin, INPUT_PULLUP);
  pinMode(id3SwitchPin, INPUT_PULLUP);
  pinMode(id4SwitchPin, INPUT_PULLUP);
  pinMode(pullBeforeFlightPin, INPUT_PULLUP);

  ppod = digitalRead(ppodSwitchPin);
  rfd900 = digitalRead(commsSwitchPin);
  satCom = digitalRead(satSwitchPin);
  baroOn = digitalRead(baroSwitchPin);
  id1On = digitalRead(id1SwitchPin);
  id2On = digitalRead(id2SwitchPin);
  id3On = digitalRead(id3SwitchPin);
  id4On = digitalRead(id4SwitchPin);
  pullOn = digitalRead(pullBeforeFlightPin);
}
```

PTERODACTYL flight code (Comms):

The Comms block job is to set up everything that is needed for the communication of the PTERODACTYL with the other PTERODACTYL on the stack, which will talk with the satCom. This means that this block prepares the XBee for communicating with the other one on the stack that then talks with the satCom.

```
// Contains all of the setup information and functions related to XBee communication.
// Some functions for converting numbers to the proper binary elements have been kept in case they're needed for the SatCom system

#include <RelayXBee.h> //Library can be found at https://github.com/MNSGC-Ballooning/XBee

#define xbeeSerial Serial5        // Serial communication lines for the xbee radio -- PCB pins: Serial3
```

```
String xbeeSendRequest = "Readyfordata";
String a0;

RelayXBee xbee = RelayXBee(&xbeeSerial, xbeeID);

float testFloat = 128.0;
int testInt = 1280;
int sentBytes = 0;

void xbeeSetup(){
  updateOled("Xbee Radio\nInit...");
  char xbeeChannel = 'A';
  xbeeSerial.begin(XBEE_BAUD);
  xbee.init(xbeeChannel); // Need to make sure xbees on both ends have the same identifier. "AAAA"
  xbee.enterATmode();
  xbee.atCommand("ATDL0");
  xbee.atCommand("ATMY1");
  xbee.exitATmode();
  Serial.println("Xbee initialized on channel: " + String(xbeeChannel) + "; ID: " + xbeeID);
  updateOled("Xbee\nChannel: " + String(xbeeChannel) + "\nID: " + xbeeID);
  delay(2000);
}

// Function required to convert floats into 4 byte hex vals
typedef union
{
  float number;
  uint8_t bytes[4];
}FLOATUNION_t;

// Function required to convert ints into 2 byte hex vals
typedef union
{
  int number;
  uint8_t bytes[2];
}INTUNION_t;

void sendFloat(float sendMe){
  FLOATUNION_t myFloat;
  myFloat.number = sendMe;
```

```
  for (int i=0; i<4; i++)
  {
    // Send converted floats here
  }
  sentBytes += 4;
}

// function that takes in an int value and prints it to the xbeePro serial as 2 bytes.
void sendInt(int sendMe){
  INTUNION_t myInt;
  myInt.number = sendMe;
  for (int i=0; i<2; i++)
  {
    // Send converted ints here
  }
  sentBytes += 2;
}

// rounds out the 21 byte xbee pro transmission with "0" bytes to fill packet
void finishSend(){
  for (int i=0; i<(21-sentBytes); i++){
    // Send completed packet here
  }
}

void updateXbee(){ // This is disgusting

//    For new satCom relay code
//    a0 = xbeeSerial.readString();
//    xbeeSerial.flush();
//    //Serial.println(a0);
//    if (a0 == xbeeSendRequest) {
//   delay(500);
//   xbeeSerial.print(xbeeID + "," + satData + "!" );
//   Serial.println(xbeeID + "," + satData + "!");
//    }
  if((millis() - xbeeTimer) > xbeeRate){

    xbeeTimer = millis();
```

```
    xbeeSerial.print(xbeeID + "," + groundData + "!");

    digitalWrite(xbeeLED,HIGH);
    delay(80);
    digitalWrite(xbeeLED,LOW);
    xbeeMessage = "DATA STRING TRANSMITTED";
  }

 if (xbeeSerial.available() > 10){
  groundCommand = xbeeSerial.readString();
  if(groundCommand.startsWith(xbeeID))
  {
    groundCommand.remove(0,xbeeID.length()+1);
    xbeeSerial.println(xbeeID + ", " + interpretMessage(groundCommand) + "!");
    xbeeMessage = xbeeID + " RECEIVED: " + groundCommand + "; SENT: " +
interpretMessage(groundCommand);
  }
 }
}

String interpretMessage( String myCommand ){

    if(myCommand.startsWith("ALT"))
    {
      xbeeMessage = "Altitude calculated from pressure: " + String(altitudeFt);
    }
    else if(myCommand.startsWith("DATA"))
    {
      xbeeMessage = data;
    }
    else if(myCommand.startsWith("MARCO"))
    {
      xbeeMessage = "POLO";
    }
    else if(myCommand.startsWith("FREQ="))
    {
      myCommand.remove(0,5);
      xbeeMessage = "New Send Rate: " + myCommand;
      xbeeRate = myCommand.toInt();
    }
```

```
  else{
    xbeeMessage = "Error - command not recognized: " + groundCommand;
  }
  if(xbeeMessage!="") return xbeeMessage;
 }
```

PTERODACTYL flight code (SD):

The job of the SD block of code is to set up for data logging on the SD card. This means that it creates a file name and prepares to write a data string.

```
// Contains setup and code for SD logging.
#include <SD.h> //Should come automatically with Arduino, although you may have to disable the native library.

#define chipSelect BUILTIN_SDCARD //Should highlight if you have teensy 3.5/3.6/4.0 selected

File datalog;
File datalogIMU;
char filename[] = "SDCARD00.csv";

bool sdActive = false;

void sdSetup(){
 pinMode(chipSelect,OUTPUT);
 if(!SD.begin(chipSelect)){
  Serial.println("Card failed, or not present");
  updateOled("Turn off\nand Insert SD card");
  for(int i=1; i<20; i++){
   digitalWrite(13,HIGH);
   digitalWrite(sdLED,LOW);
   delay(100);
   digitalWrite(13,LOW);
   digitalWrite(sdLED,HIGH);
   delay(100);
  }
 }
 else {
  Serial.println("Card initialized.\nCreating File...");
  for (byte i = 0; i < 100; i++) {
```

```
    filename[6] = '0' + i/10;
    filename[7] = '0' + i%10;
    if (!SD.exists(filename)) {
      datalog = SD.open(filename, FILE_WRITE);
      sdActive = true;
      Serial.println("Logging to: " + String(filename));
      updateOled("Logging:\n\n" + String(filename));
      delay(1000);
      break;}
   }
   if (!sdActive) {
    Serial.println("No available file names; clear SD card to enable logging");
    updateOled("Clear SD!");
    delay(5000);
   }
  }
}

void logData(String Data){
  datalog = SD.open(filename, FILE_WRITE);
  datalog.println(Data);
  datalog.close();
  Serial.println(Data);
}
```

PTERODACTYL flight code (Sensors):

The job of the sensor block is to collect all the data from the sensors and build strings that can be either sent to the satCom, be logged on the SD card, or both.

```
// Setup and code for all sensors mounted on PTERODACTYL board.
#include <UbloxGPS.h> //Library can be found at https://github.com/MNSGC-Ballooning/FlightGPS
#include <TinyGPS++.h>  //Library comes in the same download as UbloxGPS.h
#include <Wire.h> //This should come automatically with Arduino
#include <SPI.h>  //This should come automatically with Arduino
#include <SparkFunLSM9DS1.h> //Library can be found at
https://github.com/sparkfun/SparkFun_LSM9DS1_Arduino_Library
#include <OneWire.h>  //This should come automatically with Arduino
#include <MS5611.h> //Library can be found at
https://github.com/Patrikpcm/Arduino-MS5611-Library
```

```
#include <SFE_MicroOLED.h>  //Library can be found at
https://github.com/sparkfun/SparkFun_Micro_OLED_Arduino_Library
#include <SparkFunMLX90614.h>

//The library assumes a reset pin is necessary. The Qwiic OLED has RST hard-wired, so pick an
arbitrarty IO pin that is not being used
#define PIN_RESET 9
//The DC_JUMPER is the I2C Address Select jumper. Set to 1 if the jumper is open (Default), or set to
0 if it's closed.
#define DC_JUMPER 1

int lineNumber = 0;

byte Statusbyte;
String satData;
bool imucheck = false;

IRTherm therm;
#define thermIntPin A16
#define thermExtPin A17
#define ubloxSerial Serial3       // Serial communication lines for the ublox GPS -- PCB pins: Serial5

MS5611 baro;
MicroOLED oled(PIN_RESET, DC_JUMPER);    // I2C declaration
LSM9DS1 imu;
UbloxGPS ublox(&ubloxSerial);

/////////////// Thermistor constants //////////////////////

float adcMax = pow(2,analogResolutionBits)-1.0; // The maximum adc value given to the thermistor
float A = 0.001125308852122;
float B = 0.000234711863267;
float C = 0.000000085663516; // A, B, and C are constants used for a 10k resistor and 10k thermistor
for the steinhart-hart equation
float R1 = 10000; // 10k Ω resistor
float Tinv;
float adcVal;
float logR;
float T; // these three variables are used for the calculation from adc value to temperature
float currentTempC; // The current temperature in Celcius
```

```
float currentTempF; // The current temperature in Fahrenheit

void ubloxSetup(){
  ubloxSerial.begin(UBLOX_BAUD);
  ublox.init();

  byte i = 0;
  while (i<50) {
    i++;
    if (ublox.setAirborne()) {
      Serial.println("Air mode successfully set.");
      break;}
    if (i==50){
      Serial.println("Failed to set to air mode.");
      updateOled("Failed to set GPS Air Mode");
      delay(5000);
    }
  }
  updateOled("GPS init\ncomplete!");
  delay(1000);
}

void UpdateStatus(){
  bitWrite (Statusbyte, 7, 1);
  bitWrite (Statusbyte, 6, fix);
  bitWrite (Statusbyte, 5, sdActive);
  bitWrite (Statusbyte, 4, thermistorInt < 120);
  bitWrite (Statusbyte, 3, thermistorInt > 10);
  bitWrite (Statusbyte, 2, 0 -100 < therm.ambient() < 80);
  bitWrite (Statusbyte, 1, 0 < msPressure < 15);
  bitWrite (Statusbyte, 0, imucheck);
}


void qtempSetup(){
    if (therm.begin() == false){ // Initialize thermal IR sensor
    Serial.println("Qwiic IR thermometer did not acknowledge! Freezing!");
    while(1);
  }
  Serial.println("Qwiic IR Thermometer did acknowledge.");
```

```
  therm.setUnit(TEMP_F); // Set the library's units to Farenheit
  // Alternatively, TEMP_F can be replaced with TEMP_C for Celsius or
  // TEMP_K for Kelvin.

  pinMode(LED_BUILTIN, OUTPUT); // LED pin as output
}

void imuSetup(){
  Wire.begin();
  if(!imu.begin()){
    Serial.println("Failed to communicate with LSM9DS1.");
    updateOled("IMU\nOffline.");
    delay(5000);
  }
  else{
    updateOled("IMU init\ncomplete!");
    imucheck = true;
    delay(1000);
  }
}

void updateIMU(){
  if( imu.gyroAvailable() ) imu.readGyro();
  if( imu.accelAvailable() ) imu.readAccel();
  if( imu.magAvailable() ) imu.readMag();

  magnetometer[0] = imu.calcMag(imu.mx);
  magnetometer[1] = imu.calcMag(imu.my);
  magnetometer[2] = imu.calcMag(imu.mz);
  accelerometer[0] = imu.calcAccel(imu.ax);
  accelerometer[1] = imu.calcAccel(imu.ay);
  accelerometer[2] = imu.calcAccel(imu.az);
  gyroscope[0] = imu.calcGyro(imu.gx);
  gyroscope[1] = imu.calcGyro(imu.gy);
  gyroscope[2] = imu.calcGyro(imu.gz);
}

void oledSetup(){
  Wire.begin();
```

```
  oled.begin();    // Initialize the OLED
  oled.clear(ALL); // Clear the display's internal memory
  oled.display();  // Display what's in the buffer (splashscreen)
  //delay(1000);     // Delay 1000 ms
  oled.clear(PAGE); // Clear the buffer.

  randomSeed(analogRead(A0) + analogRead(A1));

  updateOled("Initializing...");
}

void updateOled(String disp){
  oled.clear(PAGE);
  oled.setFontType(0);
  oled.setCursor(0, 0);
  oled.println(disp);
  oled.display();
}

void msSetup() {
  //updateOled("initializing\nbaro...");
  while(!baro.begin()){
    updateOled("baro init failed!");
  }
  /*if(!baro.begin()){
    Serial.println("MS5611 Altimeter not active");
    updateOled("digital baro not active");
  }*/
  updateOled("baro init\ncomplete!");
  delay(1000);
}

void updateMS() {
  msTemperature = baro.readTemperature();
  msTemperature = msTemperature*(9.0/5.0) + 32.0;
  msPressure = baro.readPressure();
  msPressure = msPressure * 0.000145038;
}

void updateThermistor(){
```

```
  analogReadResolution(analogResolutionBits);
  adcVal = analogRead(thermIntPin);
  logR = log(((adcMax/adcVal)-1)*R1);
  Tinv = A+B*logR+C*logR*logR*logR;
  T = 1/Tinv;
  currentTempC = T-273.15; // converting to celcius
  currentTempF = currentTempC*9/5+32;
  thermistorInt = currentTempF;

  adcVal = analogRead(thermExtPin);
  logR = log(((adcMax/adcVal)-1)*R1);
  Tinv = A+B*logR+C*logR*logR*logR;
  T = 1/Tinv;
  currentTempC = T-273.15; // converting to celcius
  currentTempF = currentTempC*9/5+32;
  thermistorExt = currentTempF;
}

void updateUblox(){
  ublox.update();
}

void updateDataStrings(){
  altitudeFtGPS = ublox.getAlt_feet();
  latitudeGPS = ublox.getLat();
  longitudeGPS = ublox.getLon();
  lineNumber += 1;

//building a string for data logging
 groundData = String(ublox.getMonth()) + "/" + String(ublox.getDay()) + "/" + String(ublox.getYear())
+ "," +
           String(ublox.getHour()-5) + "," + String(ublox.getMinute()) + "," + String(ublox.getSecond())
+ ","
          + String(ublox.getLat(), 4) + ", " + String(ublox.getLon(), 4) + ", " + String(altitudeFtGPS, 4)
          +  ", " + String(altitudeFt) + ", " + String(thermistorInt) + ", " + String(thermistorExt) + "," +
          String(msTemperature) + ", " + String(msPressure) + ", " + String(millis()/1000.0) + ", ";

 data = groundData + String(magnetometer[0]) + ", " + String(magnetometer[1]) + ", " +
String(magnetometer[2]) + ", " +
```

```
        String(accelerometer[0]) + ", " + String(accelerometer[1]) + ", " + String(accelerometer[2]) +
", " +
        String(gyroscope[0]) + ", " + String(gyroscope[1]) + ", " + String(gyroscope[2]) + "," +
String(therm.object(), 2) + "," + String(therm.ambient(), 2) + "," +  xbeeMessage;

 satData = "|" + String(Statusbyte) + "," + String(lineNumber) + "," + String(therm.object()) + "," +
String(therm.ambient()) + "," + String(accelerometer[2]);


  updateOled(String(latitudeGPS,4) + "\n" + String(longitudeGPS,4) + "\n" + String(altitudeFtGPS,1)
+ "ft\nInt:" + String(int(thermistorInt)) + " F\nExt:"
  + String(thermistorExt) + " F\n" + String(msPressure,2) + " PSI");
  if(ublox.getFixAge() > 2000) fix = false;
  else fix = true;
  logData(data);
}

void pullPin(){
  updateOled("Pull Flight Pin to start timer");

  while(pullOn==0)
  {
    digitalWrite(fixLED,HIGH);
    digitalWrite(ppodLED,HIGH);
    digitalWrite(xbeeLED,HIGH);
    digitalWrite(sdLED,HIGH);
    pullOn = digitalRead(pullBeforeFlightPin);
  }
  ppodOffset = millis();
  updateOled("Timer Starting");
  digitalWrite(fixLED,LOW);
  digitalWrite(ppodLED,LOW);
  digitalWrite(xbeeLED,LOW);
  digitalWrite(sdLED,LOW);
  delay(2000);
}

void updateqtemp (){
  digitalWrite(LED_BUILTIN, HIGH);
```

```
  // Call therm.read() to read object and ambient temperatures from the sensor.
  if (therm.read()) // On success, read() will return 1, on fail 0.
  {
    // Use the object() and ambient() functions to grab the object and ambient
  // temperatures.
  // They'll be floats, calculated out to the unit you set with setUnit().
    Serial.print("Object: " + String(therm.object(), 2));
    Serial.println("F");
    Serial.print("Ambient: " + String(therm.ambient(), 2));
    Serial.println("F");
    Serial.println();
  }
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}

void ledGlissando() {
  digitalWrite(fixLED,HIGH);
  delay(50);
  digitalWrite(ppodLED,HIGH);
  delay(50);
  digitalWrite(fixLED,LOW);
  digitalWrite(xbeeLED,HIGH);
  delay(50);
  digitalWrite(ppodLED,LOW);
  digitalWrite(sdLED,HIGH);
  delay(50);
  digitalWrite(xbeeLED,LOW);
  delay(50);
  digitalWrite(sdLED,LOW);
  delay(150);
}
```