Spring 5-25-2022

CloudBots: Autonomous Atmospheric Explorers

Akash Binoj
akash.binoj@uconn.edu

CloudBots: Autonomous Atmospheric Explorers

Computer Science Honors Thesis

Akash Binoj (ME/CS)

Project Faculty Sponsor/Advisor: George Matheou (ME)

Honors Advisor: Jinbo Bi (CS)

Honors Thesis Advisor: Aswhin Dani (ECE)

ME Team 56: Akash Binoj, Ava Zahedi, Christopher Case

Final Report

i. Abstract/Executive Summary

Tornado warnings in the US are only issued after an early signal on weather radar is detected, resulting in an average of 13 minutes of warning rather than an advanced warning based on a numerical-model forecast. The lack of reliable extreme weather warning systems results in a preventable loss of human lives. To improve extreme weather forecasts, measurements within storms are needed to help develop and tune weather forecast numerical models. The CloudBot helps solve this problem by serving as a weather balloon carrying a robotic payload that can obtain live atmospheric measurements during storms. The CloudBot also acts as a proof-of-concept that a weather balloon can achieve both ascent and descent during flight.

To achieve our goals, the operating principle of the CloudBot is variable buoyancy. In essence, we can manipulate the density of the CloudBot by compressing air in an air cell below the payload. Doing so results in an increase of total weight and density of the CloudBot, causing the CloudBot to descend. Releasing the air then allows the CloudBot to ascend again. We can repeat this process as many times as desired to reach precise altitudes. Limitations of the CloudBot include being able to handle harsh storm conditions during flight and building within our $1000 budget. However, the most critical constraint of the CloudBot is that it must have a payload of at most 6 lbs in accordance with regulations set by the United States Federal Aviation Administration (FAA) to be legally defined as a weather balloon that we can fly, which we have adhered to.

We have fully built the CloudBot, featuring a helium balloon at the top and an air cell at the bottom. The air cell is connected to our robotic payload housing all of our electronics, sensors, and pump. Our helium connection allows us to refill and reuse our helium balloon rather than having to use a new one every flight. We performed tests in both calm and moderately windy conditions with altitude control. All tests were successful in achieving variable buoyancy, following our pre-programmed flight plans, and communicating live atmospheric measurements.

ii. Table of Contents

iii. Nomenclature/Glossary

- **Air cell**: The pressurizable, soft-walled container on the CloudBot used to increase weight of the CloudBot. We used a beach ball encased in ripstop nylon that will be attached to the bottom of the CloudBot. We adjust the internal pressure inside the ai cell during operation.
- **Buoyancy**: The upwards force allowing an object to float in a fluid.
- **FAA:** The Federal Aviation Administration. Provide guidelines and regulations pertaining to civil aviation to ensure safe operation of civilian aircraft and drones.
- **Helium balloon**: The helium-filled balloon at the top of the CloudBot. We used a 600 gram weather balloon.
- **Payload:** Positioned between the helium balloon and the air cell. It contains all of the electronics, sensors, the pump, and the air system.
- **Weight:** Downward force of a body with mass due to gravity.

I. Problem Statement

i. The Sponsor

This project is sponsored by Dr. George Matheou and the Department of Mechanical Engineering at the University of Connecticut (UConn). Professor Matheou both teaches in the Mechanical Engineering department and leads the Computational Fluid Dynamics Group at UConn. As such, this project is relevant to his interests relating to fluid flows and better understanding the environment so that future research may have a positive impact on society and the environment. Additionally, the Mechanical Engineering Department at UConn acts as a stakeholder in the project through funding and as a reflection of the department as a whole.

ii. Problem Definition

This project aims to provide the investigation, design, manufacture, and testing for a CloudBot weather balloon to quickly communicate atmospheric measurements, such as pressure, temperature, and altitude at desired locations. Currently, tornado warnings are sent out in the United States on an average of a mere 13 minutes before the tornado strikes, which does not leave much time for people to prepare and take shelter. In addition, it is not feasible to fly a fixed-wing or rotating-wing aircraft in the severe atmospheric conditions that accompany weather disasters. As such, it is difficult for these aircraft to gather data, and a common weather balloon has no means of navigation. The goal of the CloudBot is to quickly collect and communicate data at specific locations so that we can understand storms before they reach critical areas and cut the detection time for future ones.

iii. Deliverables

As is required, the CloudBot achieves bidirectional movement and is capable of collecting atmospheric measurements including pressure, temperature, and altitude. The CloudBot also houses a GPS, allowing us to note the precise location at which meteorological data is collected. The Federal Aviation Administration dictates that our payload weight may not exceed 6 pounds. Additionally, the CloudBot must be able to perform mid-flight transmission, which we have achieved through the use of a transceiver module present both on-board and on ground control.

iv. Codes and Standards

The codes and standards applying to our project are from the Federal Aviation Administration (FAA) and the Federal Communications Commission (FCC). The FAA regulations outlined in *FAA Part 101.1* limited our payload weight and size, so we designed our project around that. Namely, the payload has to weigh less than 6 lbs and the weight/size ratio cannot exceed 3 ounces per square inch on any surface of the payload [20]. The FCC regulations outlined in *FCC 22.925* dictate that cellular telephones installed in any type of aircraft (including balloons) must not be operated while such aircraft are airborne [21]. We abide by this regulation by utilizing a transceiver on a radio frequency to communicate our data and not any sort of cellular telephone.

v. Project use for Honors Thesis

The full senior project was completed with myself, as well as two other ME students, Ava Zahedi and Christopher Case. As the only CS student with hardware and software experience, I designed and developed the hardware components along with all of the electrical components. This report will showcase my individual contribution to the full Software Design and Hardware Design.

II. Results

i. Final Configuration Overview

Our final CloudBot design is able to ascend and descend in the atmosphere using variable buoyancy and is composed of relatively inexpensive components that are intended to be easily replaced in the event that they get damaged. To achieve variable buoyancy and collect atmospheric measurements, we need three main components: the weather balloon, the air cell, and the payload.

The Weather Balloon

First of all, we needed a way to provide a large upward buoyancy force on the CloudBot. We used a standard helium weather balloon to do so, because they are very accessible, inexpensive, and reliable. We chose a 600 gram professional weather balloon with a 20 foot maximum diameter. This way, we could inflate it to our required volume (roughly a five foot diameter) while leaving the balloon well below its limit to increase durability in flight. We also needed a way to quickly and easily add or remove Helium in order to precisely reach the necessary buoyancy force to lift the CloudBot without providing so much lift that we are unable to overcome it at all. To do this we used a series of PVC tubes and couplers along with a valve that we could plug directly into our Helium tank.

Figure 2.1.1 shows the parts used to interface between the helium balloon and the helium tank. The 3” PVC fits snugly in the open end of our helium balloon, and during testing we wrapped electrical tape around the neck of the balloon and the PVC when it was inserted in the balloon. We used a few zip ties to secure the connection. Then, to add Helium to the balloon we connect the half inch tube coming from the helium tank to the push connect inlet on the valve (all the way on the right in figure 2.1.1), turn the valve in the figure to the open position, and finally open the valve on the helium tank. While filling the balloon, we use a spring scale to measure the total upward buoyancy force provided by the balloon so we know precisely when

to stop. This system allows us to reuse our helium balloon, and to easily add and remove helium to precisely tune the system.

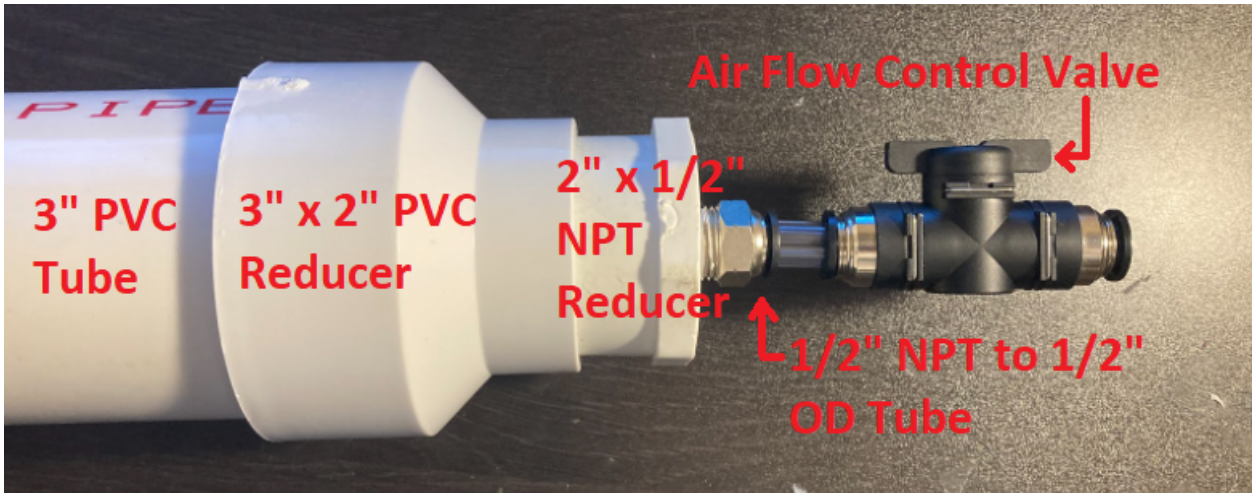


Figure 2.1.1 - Helium Adaptor Assembly

The Air Cell

Secondly, we needed a way to repeatedly increase and decrease the overall weight of the CloudBot in order to slightly overcome the buoyancy force and to bring the CloudBot back down. Our solution to this is our air cell. The air cell is a fixed volume sphere with a 30 inch diameter, which can be repeatedly pressurized up to 3 psi and depressurized again. Though it took several iterations to get a working air cell to meet our criteria, we ended up with a working design. A beach ball is used as an inflatable air bladder, and then around the beach ball, we stitched two ripstop nylon casings which keep the latex bladder from expanding in volume. This design is incredibly effective because it provides the large volume and the high tensile strength that we need while remaining lightweight.

Figure 2.1.2 - Ripstop Nylon Casings

The inner layer of the air cell (shown on right in figure 2.1.2) is stitched from a 70 Denier ripstop nylon with a little bit of stretch, and the outer layer (shown on left in figure 2.1.2) is stitched from 40 Denier ripstop nylon which does not stretch at all before ripping. With these two layers, the higher density 70 Denier layer perfectly conforms to the inside of the outer ripstop nylon layer when the beach ball bladder is pressurized. The 70D fabric provides more strength to the fabric covering and helps to reduce some of the load from the rigid 40D ripstop nylon layer.

Figure 2.1.3 - Template Pattern

After testing various designs, we found that the template pattern shown in figure 2.1.3 is the most effective for stitching the nylon casings. When assembled, there is only a maximum of four intersecting panels at any given point. In contrast, using six beach ball like templates that all intersect at each end did not work because of the higher density of stitches at a single point. With this four piece pattern, we were able to use a double welt seam with the fabric laid flat over each other which provided a much higher tensile strength than a regular simple seam that had to be used with the six template design.

To get a reliable interface between the air cell and the electronic air system in the payload, we used ⅜” NPT fittings that threaded snugly into both the air system tubing and the valve already included on the beach ball. Then, we put hose clamps over both connections to get a very reliable connection. Figure 2.1.4 shows the components of the connection before we added the hose clamps.

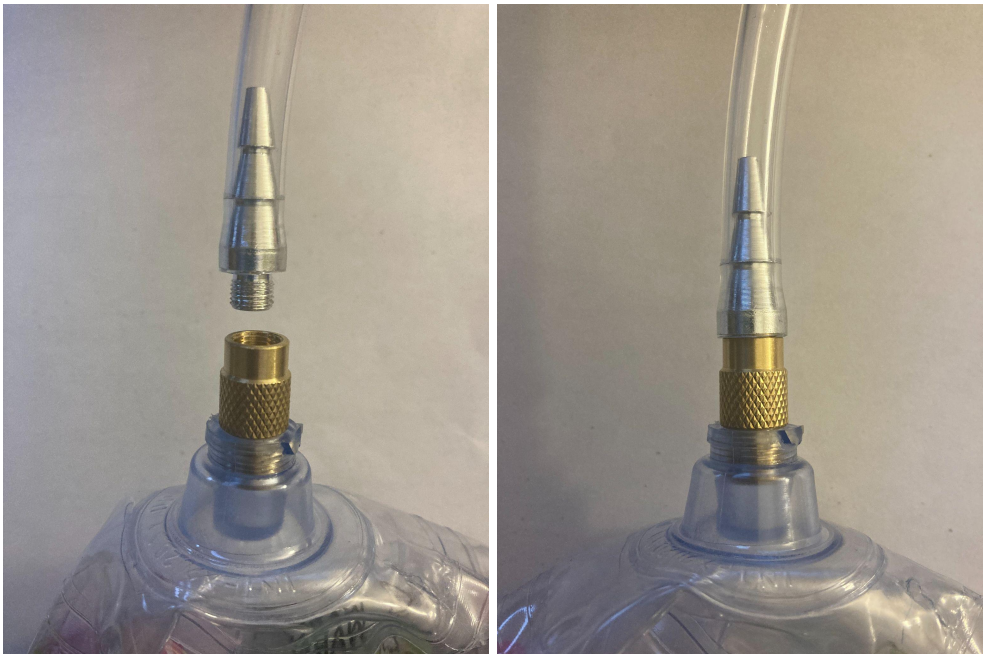


Figure 2.1.4 - Interface of Air Cell and Air System (No Hose Clamps)

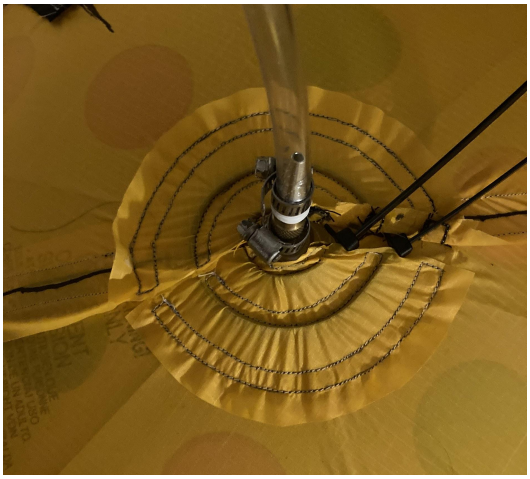


Figure 2.1.5 - Interface of Air Cell and Air System (Complete)

Figure 2.1.6 - Air Cell Pressurized to Three PSI During Test

As seen in figures 2.1.5 and 2.1.6, we incorporated channels on the ripstop nylon casing through which we routed two zip ties. By doing so, we were able to leave a large enough opening to easily swap out the beach ball if needed but when the zip ties were inserted, it created a really strong exit port for the air cell to interface with the air system.

The Payload

The third main component of the CloudBot is the payload. The payload houses the air system and the electronics. Together, the two systems are responsible for taking user input to direct air flow either into or out of the air cell.

The Air System

The air system is designed around a 12V DC pump and uses two solenoid valves to direct air flow depending on the user's input. Figure 2.1.7 below displays the three flow path options with the two valve system. Case 1 holds the pressure within the cell, Case 2 allows the pump to pressurize the air cell, and Case 3 releases pressure from the air cell to the atmosphere.

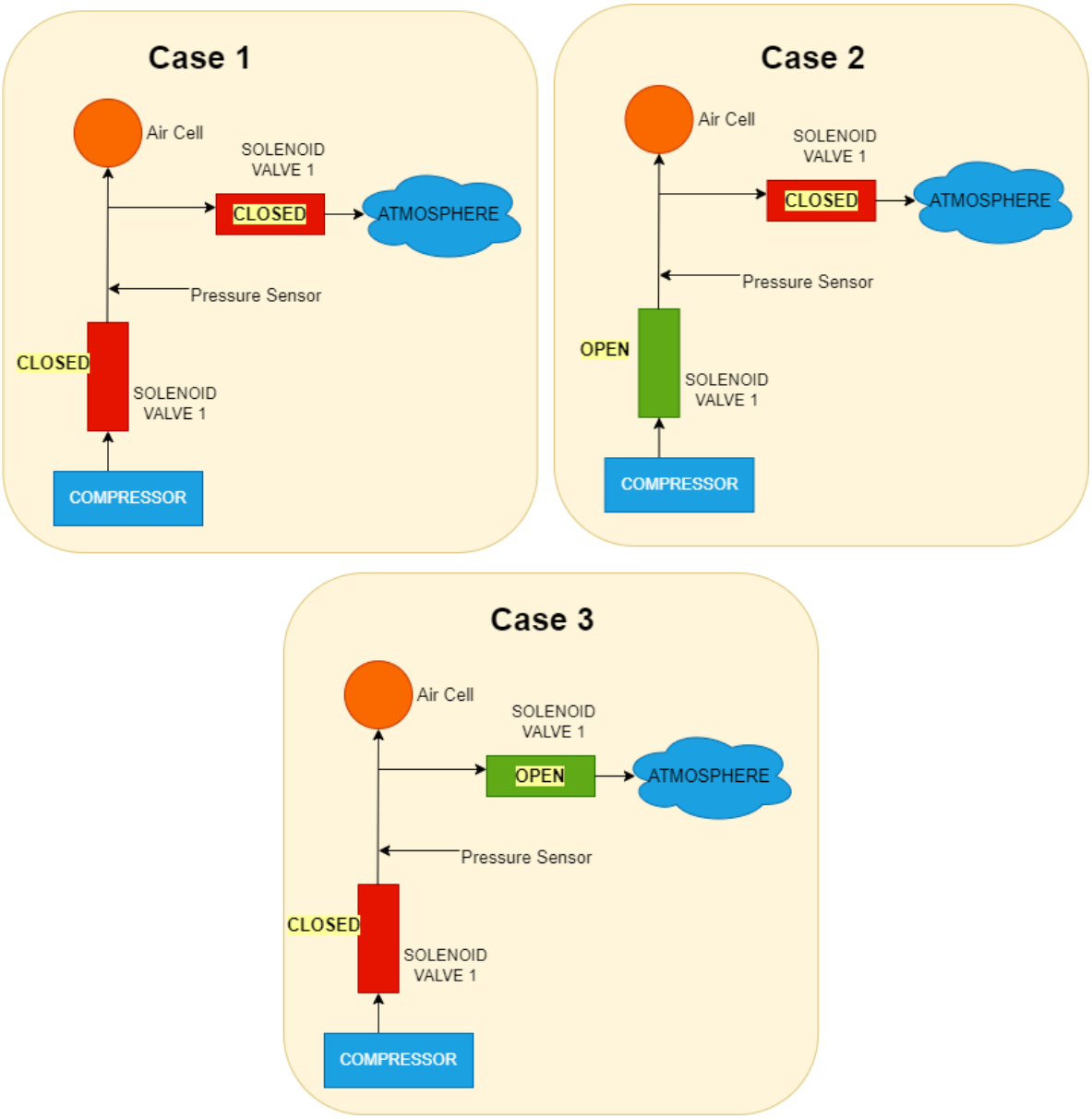
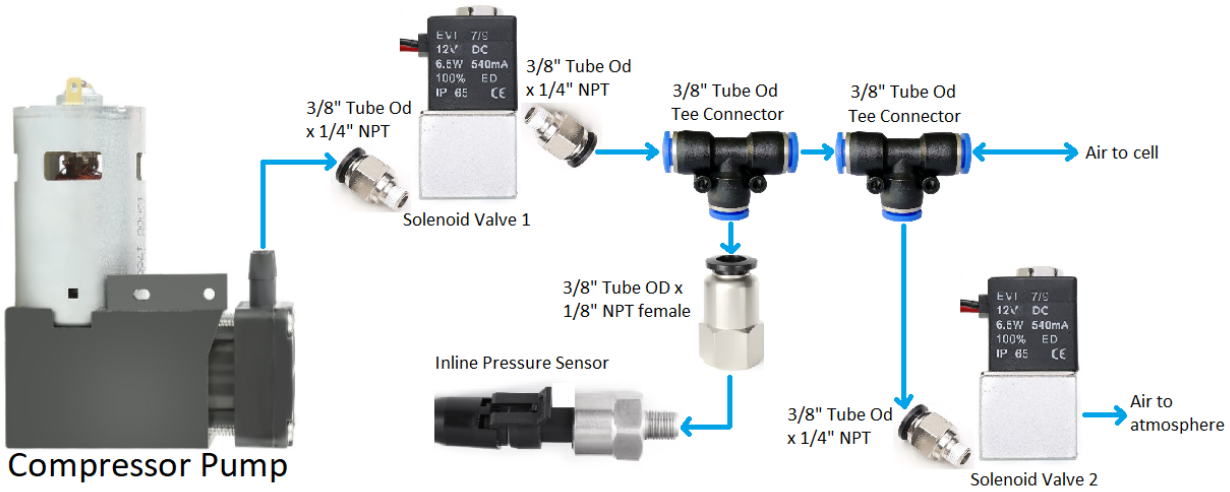


Figure 2.1.7 - All Three Flow Path Configurations



*Notes: All components are connected with 1/4" inner diameter, 3/8" outer diameter pvc tubing
Blue arrows indicate direction of air flow in system

Figure 2.1.8 - Flow Diagram of the Air System

Figure 2.1.8 displays the layout of the electrical components that we used to control which mode of operation is in use. All of the components in our air system are connected with ⅜” outer diameter vinyl tubing which is suitable for our pressurized system.

The Electrical System Design

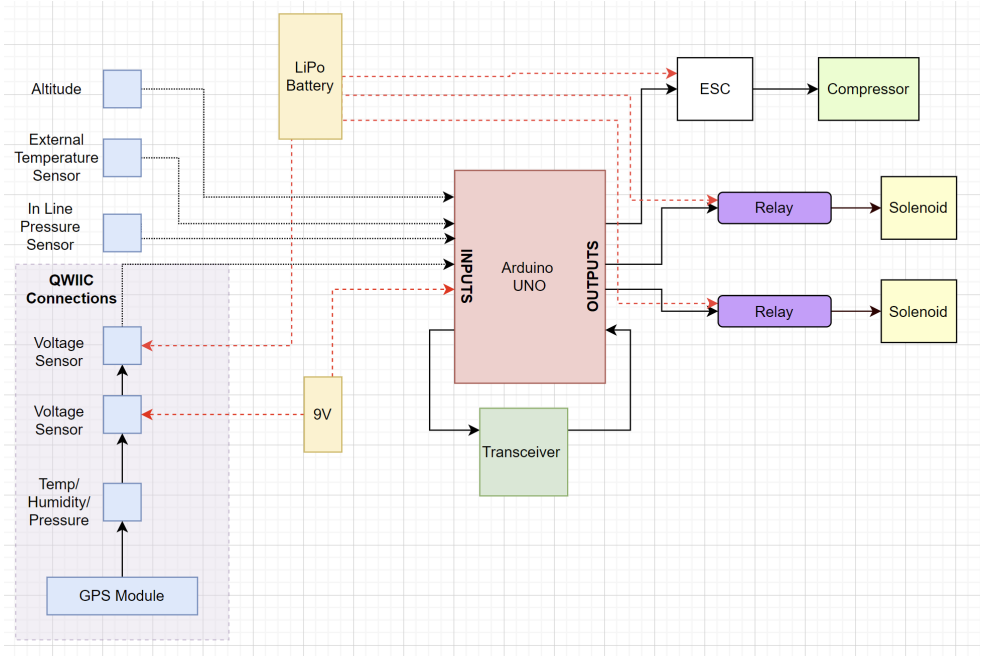


Figure 2.1.9 - Schematic of the Electrical Components within the CloudBot Payload

The left side of figure 2.1.9 lists all the sensors we will be adding to our payload. We have three sensors connected directly to the Arduino microcontroller. The four highlighted in purple on figure 2.1.9 use Sparkfun's QWIIC connect system, allowing them to be daisy chained together. The system has two power sources, the first being a 9V for running the Arduino and sensors and the second being a dedicated 12V LiPo battery to power solenoids and the pump. The devices we have listed in our electrical system were all selected for specific reasons and to help CloudBot achieve the functionality we desire. For the altitude sensor, we decided to use the Sparkfun MPL3115A2 chip, seen in figure 2.1.10 below, which can directly report the altitude to the Arduino with an accuracy of 30cm.

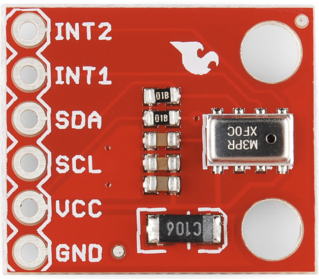


Figure 2.1.10 - Sparkfun MPL3115A2 Chip [28]

To measure the external temperature, we decided to use the Sparkfun DS18B20, shown in figure 2.1.11, which is waterproof and can function at temperatures -55°C to +125°C. The device has three wires, a +5V, a ground, and a third can be read through an Arduino Analog port and report a value from 0 (meaning -55°C) to 1023 (meaning +125°C). The device also functions with an accuracy of 0.5°C. All of these factors make it the ideal temperature sensor for the payload exterior.

Figure 2.1.11 - Atmospheric Temperature Sensor [27]

For monitoring the air cell internal pressure, we are using an air pressure transducer, shown in figure 2.1.12. This device operates on 5V and is wired similarly to the external temperature sensor, where there's a +5V and ground input with a data wire that can be read by the Arduino and be converted into a 0 psi to 30 psi scale.

Figure 2.1.12 - In-line Pressure Sensor for Air System [26]

When the CloudBot system is in flight, it is important that we are able to observe the battery levels for the LiPo and 9V, as they are critical for the operation of all components within the payload. To do this, we use the Zio Current and Voltage Sensor shown in figure 2.1.13, which can accurately read voltages from 0V to 26V and current from 0A to 3.3A. It is important for us to also be able to read the current, especially for the LiPo battery, so that we know the compressor is properly functioning and not being limited by the max current for the LiPo battery. Using the QWIIC communication protocol, we are able to daisy chain these sensors together and plug them directly into a QWIIC port on our Sparkfun Arduino.

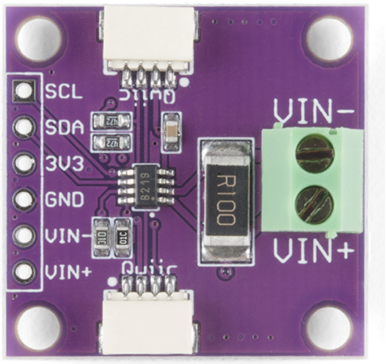


Figure 2.1.13 - Battery Voltage Sensor [25]

The CloudBot needs to be able to track variables such as Humidity and Pressure. To achieve this, we are using SparkFun's Atmospheric Sensor Breakout shown in figure 2.1.14. This device is ideal as it performs these functions as well as being operational and accurate in temperatures as low as -40°C. The wiring is also extremely easy as it also uses the QWIIC system.

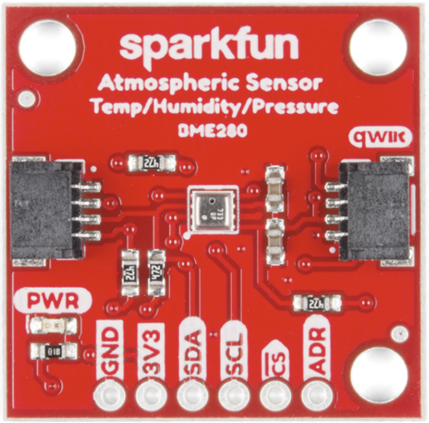


Figure 2.1.14 - Atmospheric Sensor Breakout [24]

For tracking the Cloudbot's movement and being able to locate it, we decided to use the SparkFun GPS Breakout SAM-M8Q, shown in figure 2.1.15. When powered on, this device takes up to 30 seconds for a satellite fix and is able to report longitude and latitude values every 30 nanoseconds after that with an accuracy of 2.5 meters. With a built-in gyroscope and accelerometer, this chip can also report velocity of the CloudBot, helping us monitor movement and further understand that state of the system while being on the ground. The device has QWIIC ports as well, letting us add it to the previously daisy chained system.

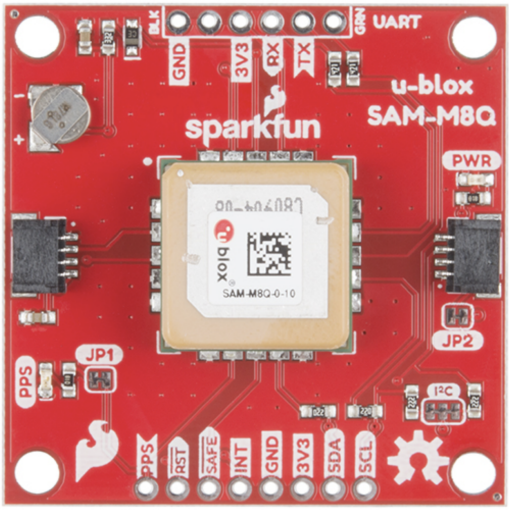


Figure 2.1.15 - GPS Breakout Sensor [13]

The pump we have is a DC 12V vacuum pump, capable of pushing at a rate of 40L/min (figure 2.1.16). Though this is the heaviest electronic component in our design, lighter pumps we considered with lower flow rates would not provide sufficient control response times.

Figure 2.1.16 - 12V DC Pneumatic Pump [23]

The pump we chose solely has two leads, one for +12V and the other for ground. Initially, we planned to incorporate a brushed motor electronic speed controller (ESC) for more precise control and to run the pump at varying speed (figure 2.1.17). Ultimately, we ended up just using a relay instead to simplify the controls. Since it took three minutes to fully pressurize the air cell we never needed to run the pump at less than its full speed. The input to this is from the 12V LiPo battery and a signal wire where we can write the power we want, from 0% to 100%. The device's output leads are connected to the pump.

Figure 2.1.17 - Brushed Motor Electronic Speed Controller [22]

For control over the solenoids and the pump, we are using 12V relays (figure 2.1.18), which can operate with an input signal from the Arduino. For the solenoids, when the relay provides no voltage they will be closed and air cannot flow. Then when there is a voltage difference they will open. Similarly, the pump will run when there is a voltage difference across it. .

Figure 2.1.18 - 12V Relays [29]

One of our main goals is for the CloutBot system to give live updates and sensor readouts while in flight. In order to do this, we use a set of transceivers, which can transmit and receive data, communicating with the Arduinos they are attached to. The specific NRF24L01P we are using is capable of sending and receiving signals at 2.4Ghz for a distance of 1100 meters (figure 2.1.19).

Figure 2.1.19 - Transceiver [14]

It is important to note that all of the electronics we have chosen are able to fully function in the environment ranges we have predicted. In case of precipitation, the electronics are in a protective waterproof housing within the payload structure.

Controller Electronics Assembly

For our ground controller system, we have retrofitted an Arduino Nano with a transceiver and three push buttons that can be used for user input (figure 2.1.20).

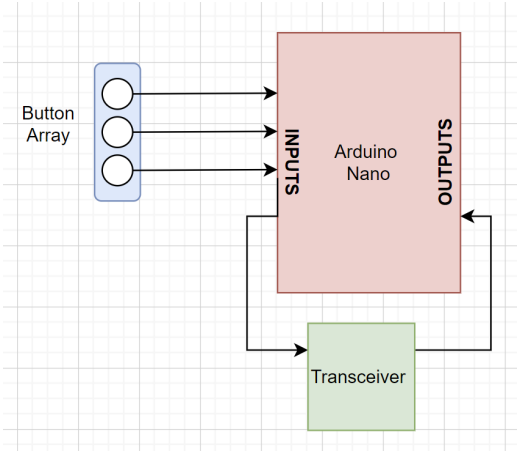


Figure 2.1.20 - Controller Electrical Schematic

After soldering all of these components to a Solderable Protoboard, we have created the controller system shown in the figure below.

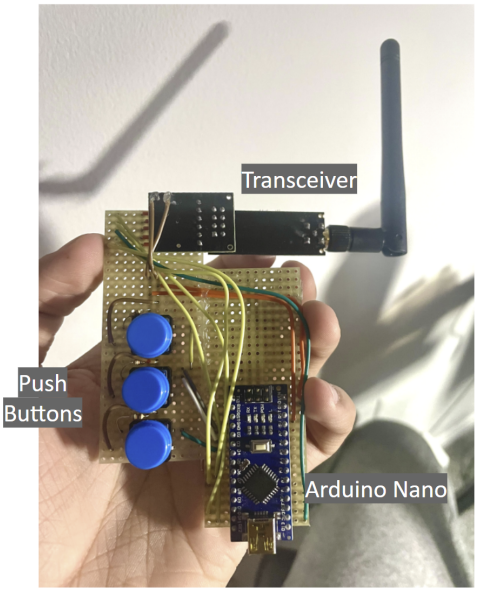


Figure 2.1.21 - Soldered Controller Build

We also designed a 3D-printed casing for it so that the sensitive electronics are enclosed and only the buttons and antenna are exposed. This design is shown below in Figure 2.1.22.

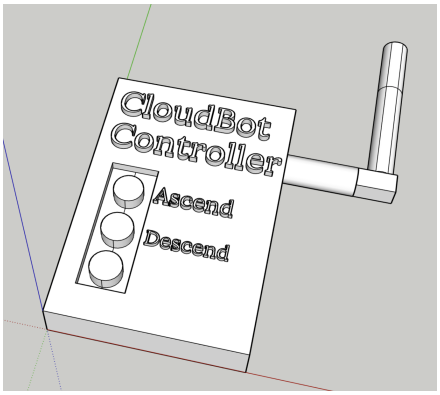


Figure 2.1.22 - 3D Model for Controller Enclosure

<u>CloudBot Electronics Assembly</u>

For our final configuration of the CloudBot, we replaced the ESC for the compressor pump with a relay, as we found out that we had no need to vary the speed of the pump due to the relatively slow pressurization time at maximum pump speed. The modified test schematic is shown below in Figure 2.1.23.

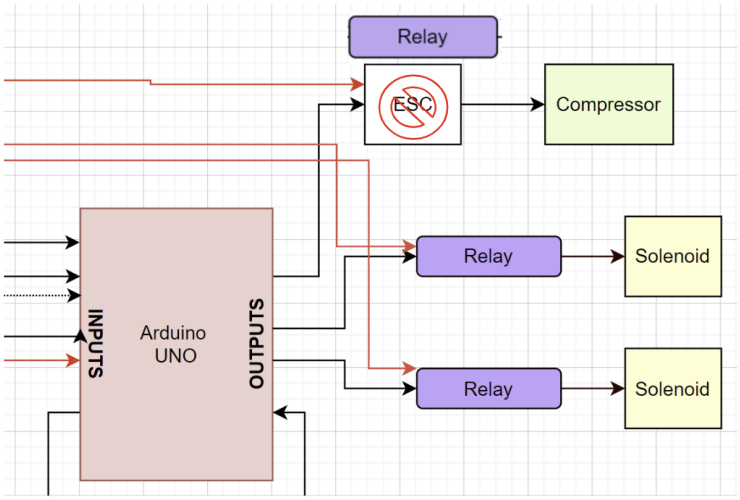


Figure 2.1.23 - Modified Test Electronics Schematic

In Figure 2.1.24, shown below, we have a schematic diagram of the power distribution. The solenoids, pumps, and relays are wired together with the help of two power rails, one for +12V and one for -12V. The rails essentially are a copper bar where all wires that are in contact with it conduct. Relays take an input of a signal and help close the power loop to its respective device.

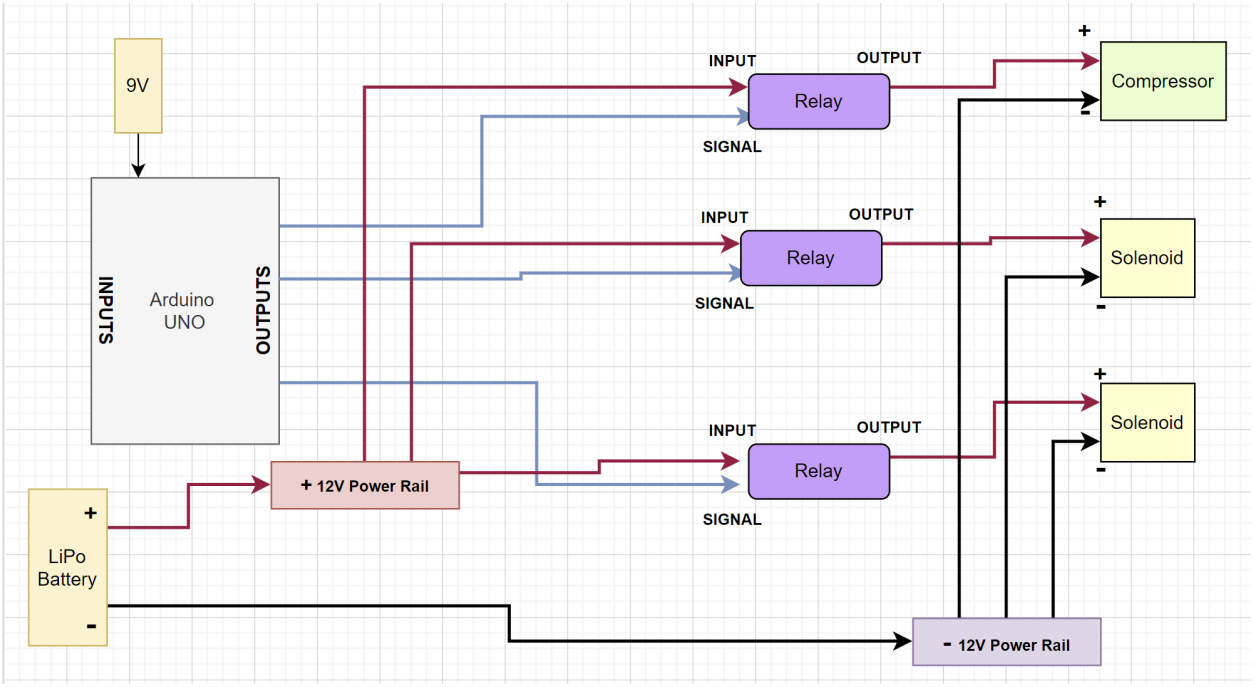


Figure 2.1.24 - CloudBot 12V Power Distribution

Shown below in Figure 2.1.25 is this system fully built with our air test setup.

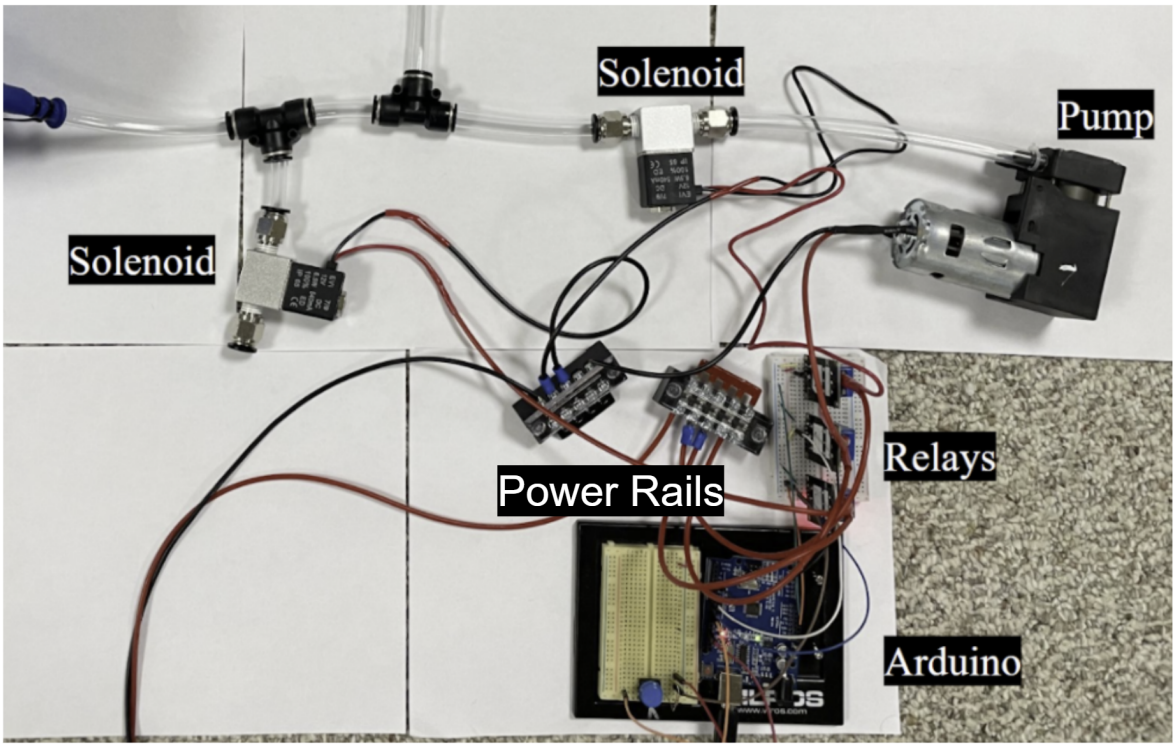


Figure 2.1.25 - Air System Build w/Power

Figure 2.1.25 also shows relays hooked up to a solderless breadboard for testing purposes, but we have created a soldered breadboard with mounted relays for the final build, shown in Figure 2.1.26.

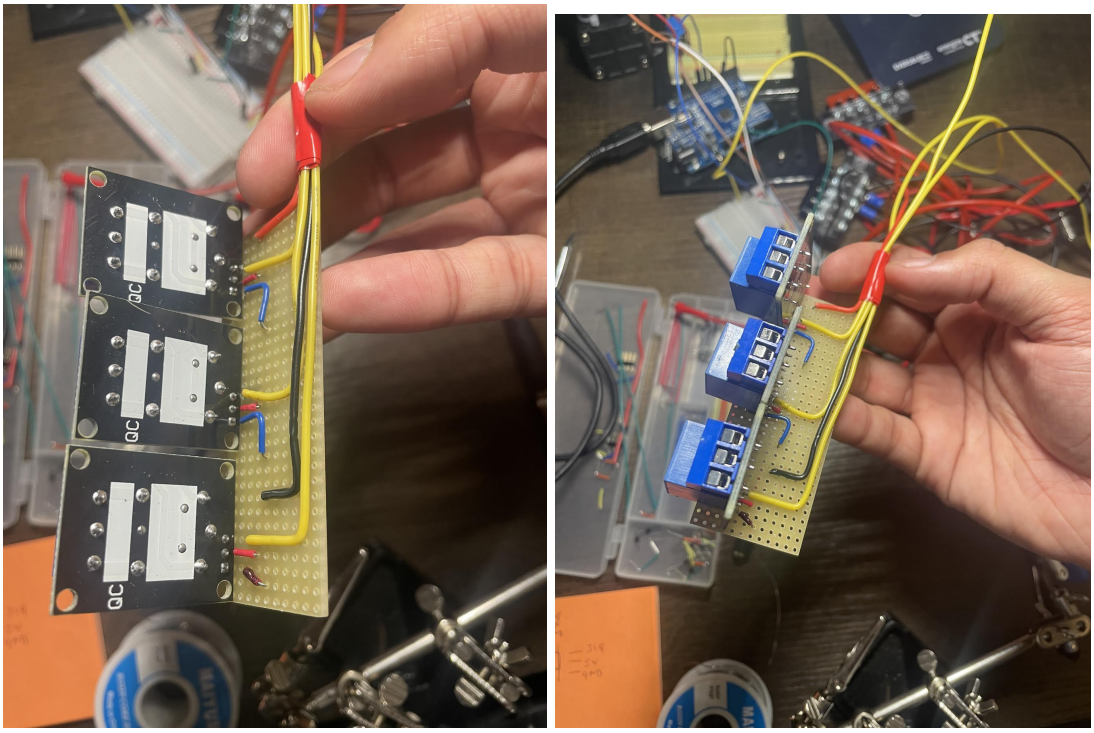


Figure 2.1.26 - Relay Wiring for Final Build

Component Housing

With our components selected, we modeled and 3D printed a component bed that neatly contains all of the components of the air system and the electrical system. This method helped us keep our components organized and compact, while also allowing us to manage some of the vibrations produced by the pump during operation.

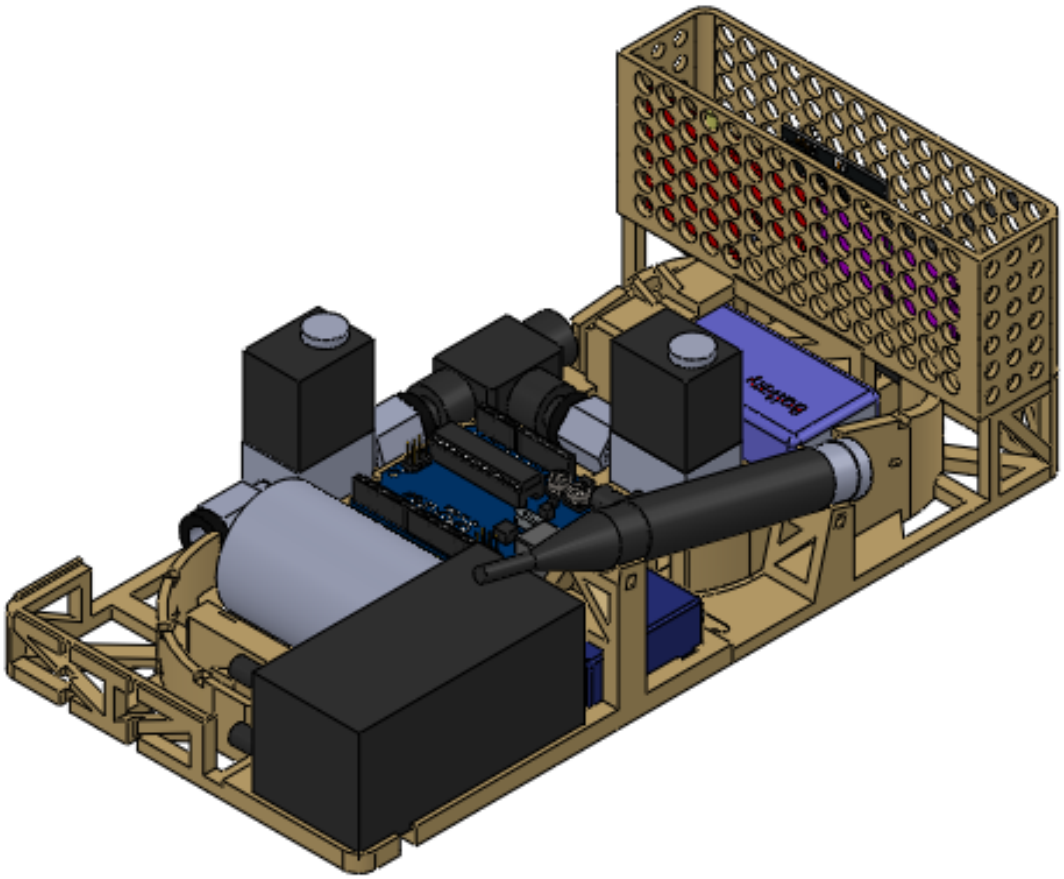


Figure 2.1.27 - Solidworks Model of Payload

Figure 2.1.27 shows how we positioned the components in the payload in order to fit them within a plastic box we selected with a latching lid. In our design, we incorporated a simple battery latch for the rechargeable LiPo battery, tube guides to keep the vinyl tube from getting kinked in the payload, an interface port for the air cell, a motor vibration damper, zip tie routing channels for securing components and tubing, an air channel to blow exhaust compressed air over critical electronics to assist cooling, and a sensor container which made it very easy to mount sensors and route wires throughout the payload. See figure 2.1.33 for a fully labeled payload diagram and BOM.

The bed we modeled is eleven inches long, which was too big to be able to print on most of the 3D printers that we had access to. To get around this, we split it into two parts with

puzzle-like connectors allowing us to get a strong connection between the two halves after printing, shown in Figure 2.1.28.

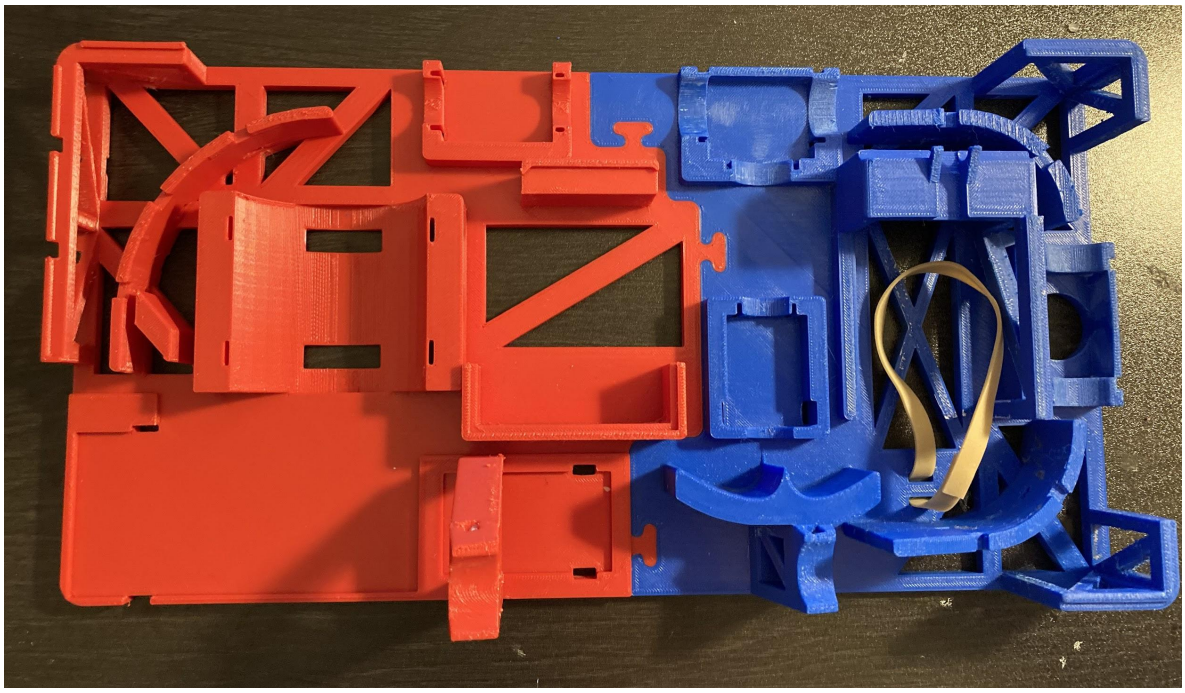


Figure 2.1.28 - Puzzle-Like Halves of Component Bed

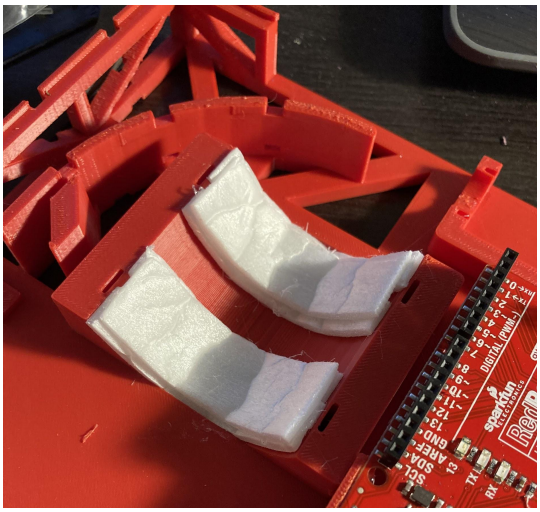


Figure 2.1.29 - Foam Vibration Damper for Motor

This foam damper (Figure 2.1.29) was critical in reducing vibrations from the motor which could have damaged other components in the payload. During testing, we noticed right

away that the motor produced significant vibrations but this design was effective in mitigating them.

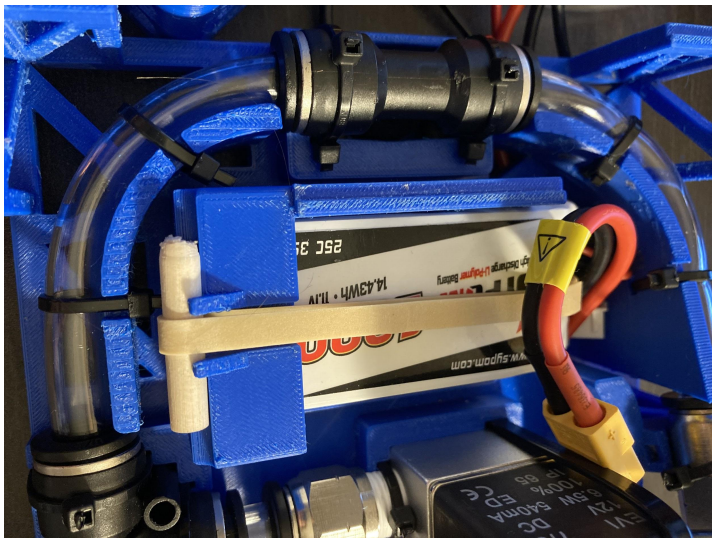


Figure 2.1.30 - Battery Rubber Band Latch

As seen in Figure 2.1.30, a rubber band and a piece of a wood dowel allowed us to make a secure battery latch system so that we could swap out the battery when it needed to be recharged, which occurs every flight.

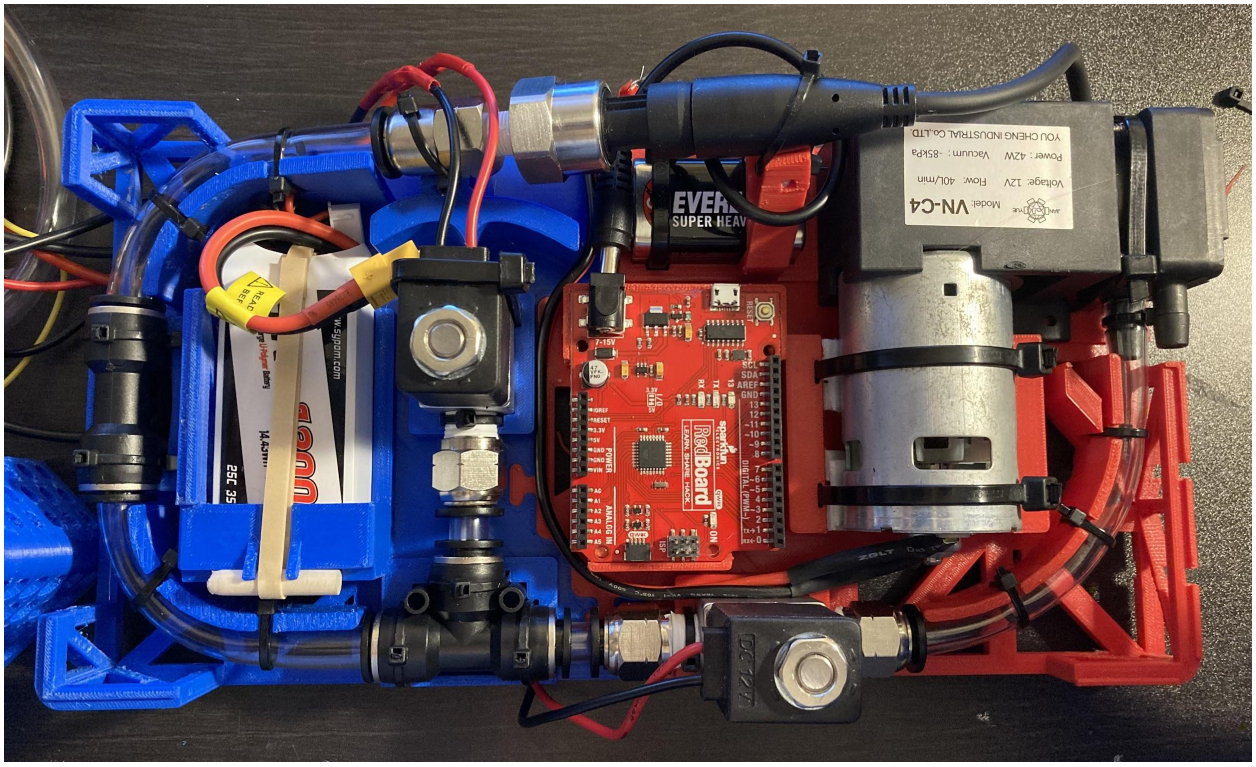


Figure 2.1.31 - Air System Components Mounted on Payload Bed

Figure 2.1.31 shows the complete air system mounted on the 3D printed bed. All components are securely mounted in place and do not become displaced during operation.

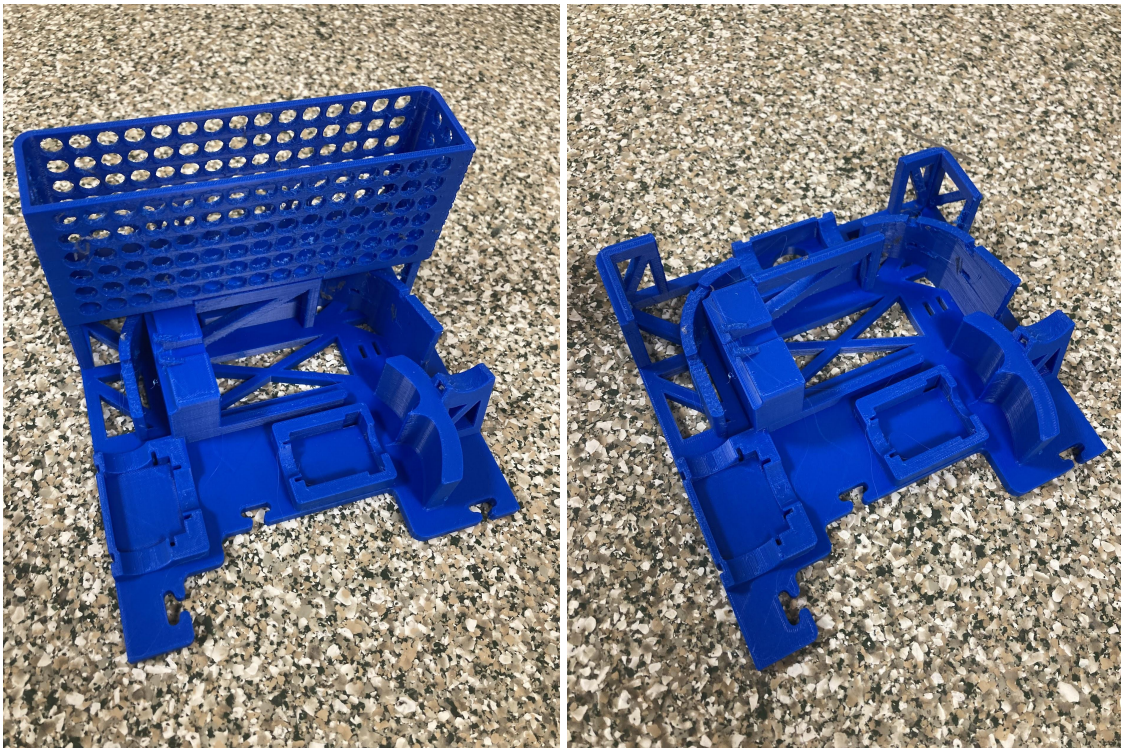
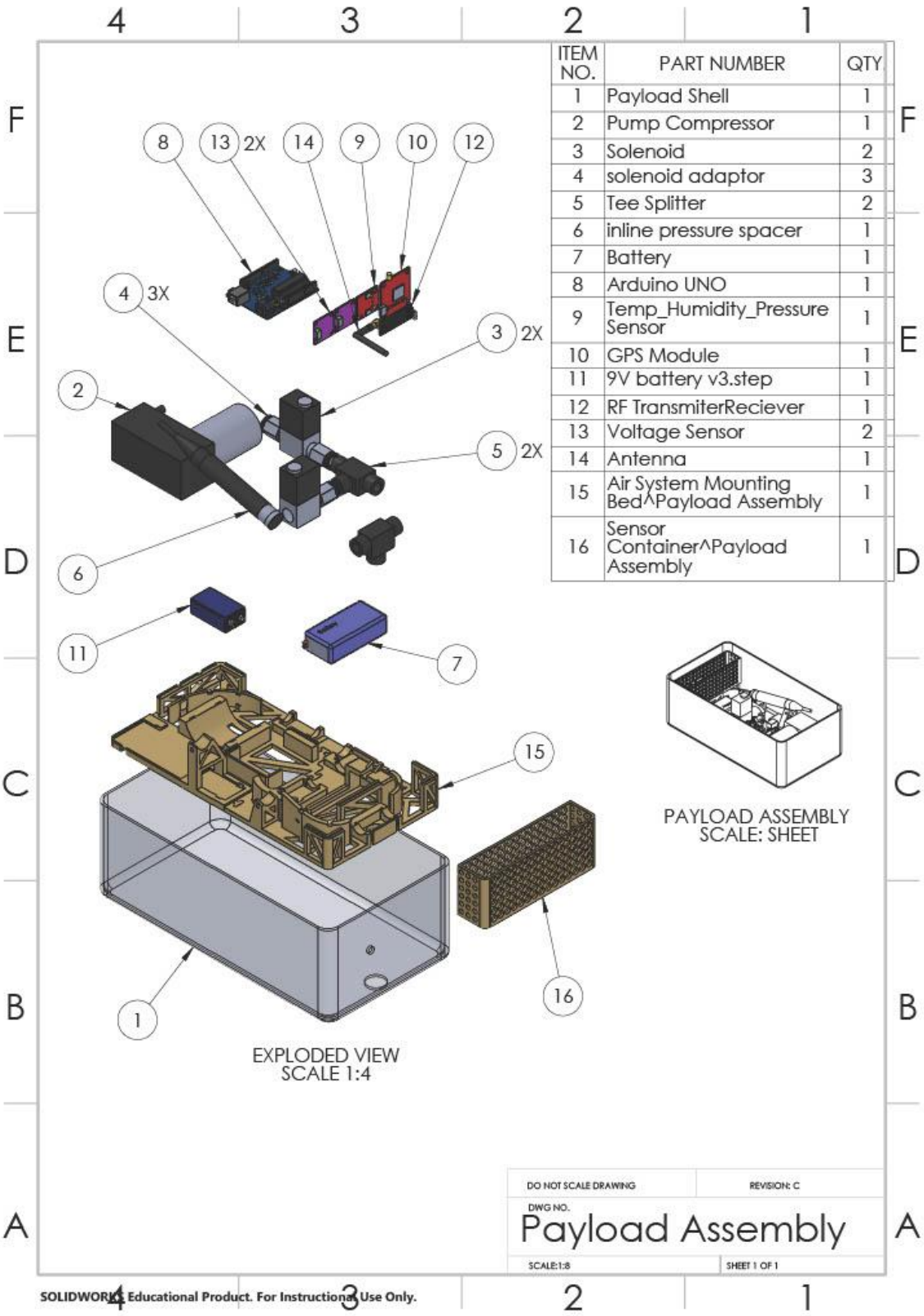


Figure 2.1.32 - The Snap-On Sensor Container

Figure 2.1.32 shows the sensor container, where we can easily zip tie on sensors and route cable to and from the rest of the payload. We had to make it removable so that we could install the air system as shown in Figure 2.1.31. Also note the “w”-shaped wind guide that directs air flow that exhausts from the air cell through the solenoid valve over critical components like the battery and motor for added cooling.



ITEM NO.	PART NUMBER	QTY
1	Payload Shell	1
2	Pump Compressor	1
3	Solenoid	2
4	solenoid adaptor	3
5	Tee Splitter	2
6	inline pressure spacer	1
7	Battery	1
8	Arduino UNO	1
9	Temp_Humidity_Pressure Sensor	1
10	GPS Module	1
11	9V battery v3.step	1
12	RF TransmiterReciever	1
13	Voltage Sensor	2
14	Antenna	1
15	Air System Mounting Bed^Payload Assembly	1
16	Sensor Container^Payload Assembly	1

Figure 2.1.33 - Exploded View of CAD Model and BOM of Main Parts

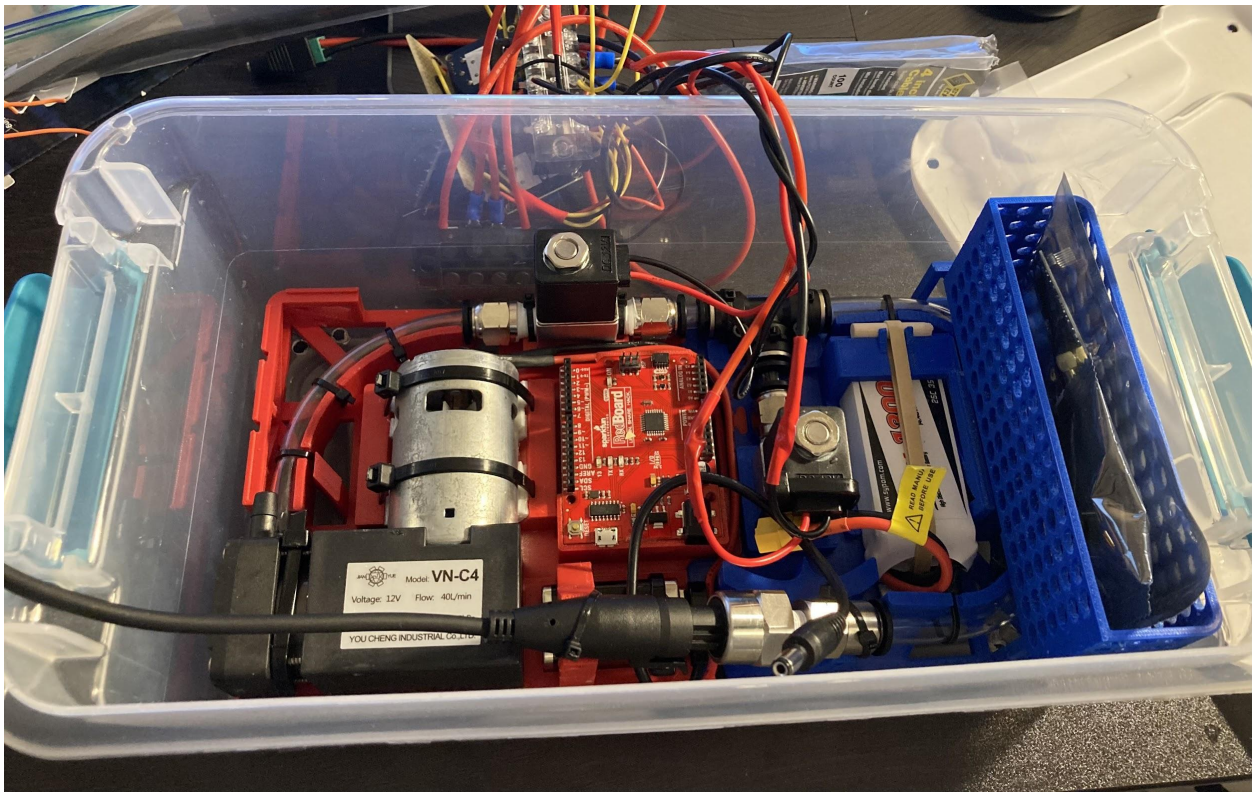


Figure 2.1.34 - Assembly in Payload Shell

ii. Software Design

It is easiest to explain the full software design by breaking it down into smaller components that control different parts of the system.

Cloudbot Onboard Controller Software

First, we look at the pressure management component.

```
pinMode(5, OUTPUT); // pump valve
pinMode(6, OUTPUT); // pump
pinMode(10, OUTPUT); // release
Serial.begin(9600);
```

Figure 2.2.1 - Pump Code Setup

The first step is to initialize the outputs and tell the program which pins are connected to relays controlling the pump solenoid valve, release solenoid valve, and the power for pump itself.

```
void pressurize(float targetPSI) {
  if(check < targetPSI){
    digitalWrite(5,HIGH);
    digitalWrite(6,HIGH);
    digitalWrite(10,LOW);
    delay(1000);
  else if(check > (targetPSI+0.3)){
    digitalWrite(5,LOW);
    digitalWrite(6,LOW);
    digitalWrite(10,HIGH);
    delay(1000);
    }
  digitalWrite(5,LOW);
  digitalWrite(6,LOW);
  digitalWrite(10,LOW);
  int sensorValue = analogRead(A0);
  float voltage = sensorValue * (5.0 / 1023.0);
  float psi = voltage - 0.5;
  psi = psi * 7.50;
  Serial.println(psi);
  check = psi;
  }
}
```

Figure 2.2.2 - Full Pressure Control Code

The *pressurize()* function controls all the pump and valve controls by continuously reading pressure values and changing what state the relays are in with the *digitalWrite()* command. It is important to note that all the relays are Normally Closed, so LOW state means solenoid valve is closed, and HIGH state means valve is opened. The full pressure code can be broken down into three parts.

```
if(check < targetPSI){
  digitalWrite(5,HIGH);
  digitalWrite(6,HIGH);
  digitalWrite(10,LOW);
  delay(1000);
```

Figure 2.2.3 - Increasing Pressure Code

When the actual pressure is below the target, it can be seen that pins 5 and 6 are turned to the HIGH state, meaning that the pump is powered on and the valve to allow air into the balloon is also open. Pin 10 is at LOW, which means that the release valve is closed. It also will run for 1 second (1000ms) before the loop continues.

```
else if(check > (targetPSI+0.3)){
  digitalWrite(5,LOW);
  digitalWrite(6,LOW);
  digitalWrite(10,HIGH);
  delay(1000);
  }
```

Figure 2.2.4 - Dropping Pressure Code

When the actual pressure is over the target with a 0.3 psi added safety to account for inaccuracies, pins 5 and 6 are changed to LOW, which turns the pump off and closes the valve for the pump. Pin 10 is changed to HIGH, opening the release valve. The code will leave this running for 1 second before continuing the loop.

```
digitalWrite(5,LOW);
digitalWrite(6,LOW);
digitalWrite(10,LOW);
int sensorValue = analogRead(A0);
float voltage = sensorValue * (5.0 / 1023.0);
float psi = voltage - 0.5;
psi = psi * 7.50;
Serial.println(psi);
check = psi;
}
```

Figure 2.2.5 - Reading Pressure Code

After the previously mentioned operations run, the system puts every pin to LOW, closing all valves and turning the pump off. This fully seals the aircell and creates a closed system, which is required for the inline pressure sensor to accurately read the pressure. This is done using the analogRead() function which reads a number between 0 and 1023. By multiplying the reading by (5.0/1023.0), we can convert that value to a scale of 0 to 5, which

represents a reading of 0 volts to 5 volts. Then, from the pressure sensor product information, we know that 0 psi is equivalent to a reading of 0.5V and 30 psi is equivalent to 4.5V, so the values are mapped to a scale of 0-30 psi, resulting in the variable *check*, containing the real pressure inside the aircell.

For all the sensors being used, I have included their respective libraries as seen in the below image.

```
#include <Wire.h>
#include <SPI.h>
#include <SparkFunLSM9DS1.h> //accelerometer
#include <nRF24L01.h> //transciever
#include <RF24.h> //transciever
#include "SparkFunBME280.h" //weather breakout board
#include <SparkFun_u-blox_GNSS_Arduino_Library.h> //http://librarymanager/All#SparkFun_u-blox_GNSS //GPS
SFE_UBLOX_GNSS myGNSS; //GPS
```

Figure 2.2.6 - Import libraries needed for sensors

Since many of the sensors are daisy chained together using the I2C communication protocol, I have implemented checks in the setup() loop to check for a good connection between sensors and readings.

```
BME280 mySensor;
LSM9DS1 imu;

void setup() {
  if (imu.begin() == false) // with no arguments, this uses default addresses (AG:0x6B, M:0x1E) and i2c port (Wire).
  {
    Serial.println("Failed to communicate with LSM9DS1.");
    Serial.println("Double-check wiring.");
    Serial.println("Default settings in this sketch will " \
                   "work for an out of the box LSM9DS1 " \
                   "Breakout, but may need to be modified " \
                   "if the board jumpers are.");
    while (1);

    if (mySensor.beginI2C() == false) //Begin communication over I2C
  {
    Serial.println("The sensor did not respond. Please check wiring.");
    while(1); //Freeze
  }

  if (myGNSS.begin() == false) //Connect to the u-blox module using Wire port
  {
    Serial.println(F("u-blox GNSS not detected at default I2C address. Please check wiring. Freezing."));
    while (1);
  }
```

Figure 2.2.7 - Checking for sensor connection

In the control loop, the first thing that happens is the transceiver turns into listening mode. Here, it looks for a connection with the ground controller, and if the connection is

present, it rewrites the targetPSI variable. It then uses this variable in the pressurize() function mentioned earlier.

```
radio.startListening();                    //This sets the module as receiver
if (radio.available())                     //Looking for incoming data
{
  radio.read(&targetPSI, sizeof(targetPSI));
  pressurize(targetPSI);
}
```

Figure 2.2.8 - Checking new Target PSI

Next, it no longer listens and gets ready to write information in the transmitting mode. It first builds an array 'value' which contains all the different sensor readouts collected including vertical acceleration, humidity, external pressure, altitude, external temperature, latitude, and longitude. It sends the array to the ground arduino unit with the radio.write() function.

```
float vertical_acceleration = imu.ay;
float humidity = mySensor.readFloatHumidity();
float external_pressure = mySensor.readFloatPressure();
float altitude = mySensor.readFloatAltitudeFeet();
float external_temp = mySensor.readTempF();
float latitude = myGNSS.getLatitude();
float longitude = myGNSS.getLongitude();
float values[] = {humidity,external_pressure,altitude,external_temp,latitude,longitude};
delay(5);

radio.stopListening();                  //This sets the module as transmitter
radio.write(&values, sizeof(values));   //Sending the data
}
```

Figure 2.2.9 - Collecting and Transmitting Sensor Readouts

Ground Arduino Controller Code

The next set of code shows what is needed to read the sensor readouts properly and use the installed controller buttons to modify and set new pressure targets. The first part shows libraries that are needed along with setting the button pinouts.

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(9, 10); // CE, CSN
const byte addresses [][6] = {"00001", "00002"};
int inc_button_pin = 2;
int dec_button_pin = 3;
int submit_button_pin = 4;
boolean inc_button = 0;
boolean dec_button = 0;
boolean sub_button = 0;
float targetPSI = 1.0;
```

Figure 2.2.10 - Setting up Ground Controller

Next, it turns the transceiver into writing mode. It first checks the state of the three buttons, checking if the increment up or down is pushed and changing the target value by 0.25 psi accordingly. Then if the third submit button is selected, the device will transmit the new value to the CloudBot onboard transceiver.

```
  radio.stopListening();
  inc_button = digitalRead(inc_button_pin);
  dec_button = digitalRead(dec_button_pin);
  sub_button = digitalRead(submit_button_pin);
  if (inc_button == HIGH)
  {
    targetPSI = targetPSI + 0.25;
  }
  if (dec_button == HIGH)
  {
    targetPSI = targetPSI - 0.25;
  }
  if (sub_button == HIGH)
  {
    radio.write(&targetPSI, sizeof(targetPSI));
  }
```

Figure 2.2.11 - Modifying and Transmitting Pressure Target

After, it changes to reading mode, where the array is passed through the radio.read() function and then printed into the system console, allowing the user to read the information. This can also be copied from the console and pasted into excel to make graphs, like outputs generated from testing shown later in this paper.

```
radio.startListening();
while(!radio.available());
float values[6];
radio.read(&values, sizeof(values)); //Reading t
println("Humidity: " + str(values[0]);
println("external_pressure: " + str(values[1]);
println("altitude: " + str(values[2]);
println("external_temp: " + str(values[3]);
println("latitude: " + str(values[4]);
println("longitude: " + str(values[5]);
```

Figure 2.2.12 - Reading and Printing Sensor Readings

Full Ground Arduino Code

```
//nRF24L01 communication 2 ways transmitter

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(9, 10); // CE, CSN
const byte addresses [][6] = {"00001", "00002"};  //Setting the two
addresses. One for transmitting and one for receiving
int inc_button_pin = 2;
int dec_button_pin = 3;
int submit_button_pin = 4;
boolean inc_button = 0;
boolean dec_button = 0;
boolean sub_button = 0;
float targetPSI = 1.0;

void setup() {
  pinMode(inc_button_pin, INPUT);
  pinMode(dec_button_pin, OUTPUT);
  pinMode(submit_button_pin, OUTPUT);
  radio.begin();                          //Starting the radio
communication
```

```
  radio.openWritingPipe(addresses[1]);     //Setting the address at
which we will send the data
  radio.openReadingPipe(1, addresses[0]);  //Setting the address at
which we will receive the data
  radio.setPALevel(RF24_PA_MIN); //You can set it as minimum or
maximum depending on the distance between the transmitter and
receiver.
}

void loop()
{

  radio.stopListening();                                //This sets the
module as transmitter
  inc_button = digitalRead(inc_button_pin);
  dec_button = digitalRead(dec_button_pin);
  sub_button = digitalRead(submit_button_pin);
  if (inc_button == HIGH)
  {
    targetPSI = targetPSI + 0.25;
  }
  if (dec_button == HIGH)
  {
    targetPSI = targetPSI - 0.25;
  }
  if (sub_button == HIGH)
  {
    radio.write(&targetPSI, sizeof(targetPSI));  //Sending the data
  }

  delay(5);

  radio.startListening();                               //This sets the
module as receiver
  while(!radio.available());                            //Looking for
incoming data
  float values[6];
  radio.read(&values, sizeof(values)); //Reading the data
  println("Humidity: " + str(values[0]);
```

```
  println("external_pressure: " + str(values[1]);
  println("altitude: " + str(values[2]);
  println("external_temp: " + str(values[3]);
  println("latitude: " + str(values[4]);
  println("longitude: " + str(values[5]);
}
```

Full CloudBoat Onboard Arduino Code

```
//nRF24L01 communication 2 ways cloudbot

#include <Wire.h>
#include <SPI.h>
#include <SparkFunLSM9DS1.h> //accelerometer
#include <nRF24L01.h> //transciever
#include <RF24.h> //transciever
#include "SparkFunBME280.h" //weather breakout board
#include <SparkFun_u-blox_GNSS_Arduino_Library.h>
//http://librarymanager/All#SparkFun_u-blox_GNSS //GPS
SFE_UBLOX_GNSS myGNSS; //GPS

RF24 radio(9, 10); // CE, CSN
const byte addresses [][6] = {"00001", "00002"};    //Setting the two
addresses. One for transmitting and one for receiving


float targetPSI;
void printGyro();
void printAccel();
void printMag();
void printAttitude(float ax, float ay, float az, float mx, float my,
float mz);

BME280 mySensor;
LSM9DS1 imu;

void setup() {
  if (imu.begin() == false) // with no arguments, this uses default
addresses (AG:0x6B, M:0x1E) and i2c port (Wire).
  {
```

```
    Serial.println("Failed to communicate with LSM9DS1.");
    Serial.println("Double-check wiring.");
    Serial.println("Default settings in this sketch will " \
                   "work for an out of the box LSM9DS1 " \
                   "Breakout, but may need to be modified " \
                   "if the board jumpers are.");
    while (1);

    if (mySensor.beginI2C() == false) //Begin communication over I2C
  {
    Serial.println("The sensor did not respond. Please check
wiring.");
    while(1); //Freeze
  }

  if (myGNSS.begin() == false) //Connect to the u-blox module using
Wire port
  {
    Serial.println(F("u-blox GNSS not detected at default I2C
address. Please check wiring. Freezing."));
    while (1);
  }

  radio.begin();                          //Starting the radio
communication
  radio.openWritingPipe(addresses[0]);      //Setting the address at
which we will send the data
  radio.openReadingPipe(1, addresses[1]);   //Setting the address at
which we will receive the data
  radio.setPALevel(RF24_PA_MIN);            //You can set it as
minimum or maximum depending on the distance between the transmitter
and receiver.
}

void loop()
{
  radio.startListening();                    //This sets the module
as receiver
  if (radio.available())                     //Looking for incoming
```

```
data
  {
    radio.read(&targetPSI, sizeof(targetPSI));
    pressurize(targetPSI);
  }

  if ( imu.accelAvailable() )
  {
    // To read from the accelerometer, first call the
    // readAccel() function. When it exits, it'll update the
    // ax, ay, and az variables with the most current data.
    imu.readAccel();
  }
}
  float vertical_acceleration = imu.ay;
  float humidity = mySensor.readFloatHumidity();
  float external_pressure = mySensor.readFloatPressure();
  float altitude = mySensor.readFloatAltitudeFeet();
  float external_temp = mySensor.readTempF();
  float latitude = myGNSS.getLatitude();
  float longitude = myGNSS.getLongitude();
  float values[] =
{humidity,external_pressure,altitude,external_temp,latitude,longitude
};
  delay(5);

  radio.stopListening();                  //This sets the module as
transmitter
  radio.write(&values, sizeof(values));   //Sending the data
  }
}


void pressurize(float targetPSI) {
  if(check < targetPSI){
    digitalWrite(5,HIGH);
    digitalWrite(6,HIGH);
    digitalWrite(10,LOW);
    delay(1000);
```

```
  else if(check > (targetPSI+0.3)){
    digitalWrite(5,LOW);
    digitalWrite(6,LOW);
    digitalWrite(10,HIGH);
    delay(1000);
    }
  digitalWrite(5,LOW);
  digitalWrite(6,LOW);
  digitalWrite(10,LOW);
  int sensorValue = analogRead(A0);
  float voltage = sensorValue * (5.0 / 1023.0);
  float psi = voltage - 0.5;
  psi = psi * 7.50;
  Serial.println(psi);
  check = psi;
  }
}
```

iii. Testing and Results

Air Cell Iterations/Testing

The air cell was a particularly tricky part of this build because of the strength and weight requirements, and the lack of documentation about any similar approaches or material strengths. As a result, it required a few different iterations and tests before we got a working design.

First, we tested a beach ball on its own. This did not work because the beach ball kept expanding and eventually burst at roughly zero psi. This is when we came up with the idea of using a ripstop nylon casing over the beach ball to contain pressure and prevent the beach ball from expanding. Our first design was composed solely of a single layer of 40 Denier rated ripstop nylon with 6 panels all intersecting at the top and the bottom, similar to a beach ball. The Denier rating corresponds to the density of the nylon, and 40D fabric is around 1.4 ounces per square yard. In theory, higher Denier ratings correspond to a stronger material, however we

discovered that not all ripstops are created equally. At one end, we left an open slit in the material (figure 2.3.1) so that we could put the beach ball bladder in once stitching was completed. The panels were connected using a simple stitch (figure 2.3.2). A simple stitch is done by placing the outside faces of two pieces together, stitching them together, and then turning them inside out once completed.

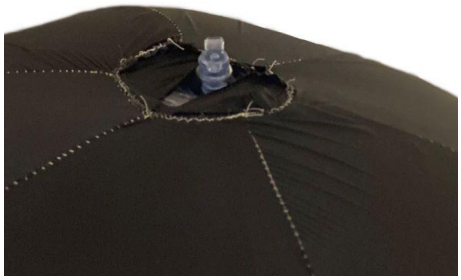


Figure 2.3.1 - Valve Interface Slit

Figure 2.3.2 - Simple Seam (Interior View)

We tested this design to failure by connecting it to our compressor, and we monitored the internal pressure in the cell via an in-line pressure sensor and an arduino board. At around 1.92 psi, we already spotted some critical signs of failure and ended the test. The main points of failure were at the top and bottom where all of the six panels intersected (figure 2.3.3). This is because at these spots, there is a higher density of stitches which weaken the strength of the nylon. It is similar to creating a perforated edge in the material.

Figure 2.2.3 - Failure in Nylon During Test

After carefully analyzing the failure of the first iteration, we came up with a new idea. Again, we used the same 40D ripstop nylon, except this time we used an entirely different stitching template.

Figure 2.3.4 - Iteration 2 Stitching Template

This pattern shown in figure 2.3.4 only requires four pieces, but more importantly it reduces the number of panels intersecting at any point to four (from six), which significantly reduces the stitching concentration at any given point. Another change that we made was that we used a double welt seam, rather than a simple seam. This is done by laying the two panels flat on top of each other, and then stitching two parallel lines along the seam (figure 2.3.5). This idea was inspired by climbing equipment which uses a similar approach to obtain a higher tensile strength.

Figure 2.3.5 - Double Welt Seam at Panel Intersection Point

As shown in Figure 2.3.5, this is a much neater way to compose the casing. To address the failure at the inlet to the casing, we incorporated zip tie routing channels around the inlet so that after inserting the beachball, zip ties would provide a stronger interface than the overlapping panels used in the first iteration.

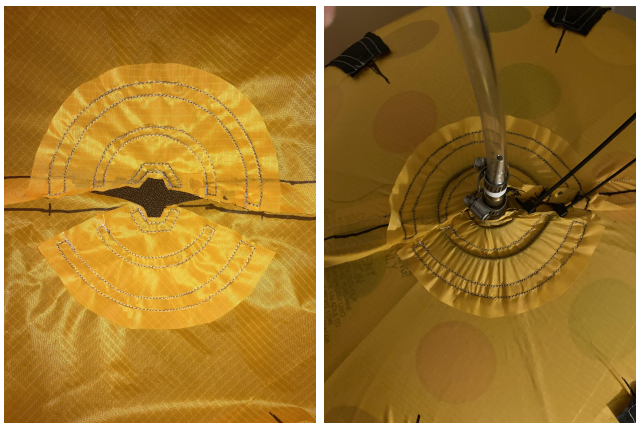


Figure 2.3.6 - Zip Tie Air System Interface

We then tested this design to failure to find the ultimate strength of this design. This time, we reached 3.375 psi in the air cell when the ripstop nylon casing burst open.

Figure 2.3.7 - Air Cell Iteration II Testing

This test gave us a really good idea of the limitations of this design, which we took into account for our final iteration. Based on the max pressure in this test at failure, we set the maximum allowable pressure of our design to 3 psi. This value would give us adequate control of the robot, and was an attainable goal for our air cell design. It also provided us with a factor of safety of at least 1.125.

While analyzing the failure of this air cell, we could not identify exactly where the failure occurred first because it was nearly instantaneous. However, it was most likely caused either by the stitches in the seam weakening the material or by a fabrication defect.

For our third and final iteration, we used the same panel template from the second iteration except we added a second layer of ripstop nylon. The inner ripstop nylon layer is made from 70 Denier fabric, weighing 1.9 ounces per square yard, and the outer layer is the exact same design as the second interaction. Upon receiving our 70D fabric, we noticed that it had slightly more stretch and flexibility to it than the 40D fabric we used previously. This made it a good choice for the inner layer as it would be able to mesh into the outer layer when we added pressure into the beach ball. A subtle change that we made in this iteration was to use a slightly

longer stitch. This would help to lessen the perforating effect of the sewing needle and to maintain more of the material strength.

Figure 2.3.8 - The Two-Layer Air Cell Design

We offset the seams of the inner and outer layers to most evenly distribute the load across the surface of the air cell.

Figure 2.3.9 - Air Cell Final Iteration Pressurized at 3 psi

For this test, instead of testing to failure we sought to pressurize the air cell to the goal pressure of 3 psi and then to depressurize it and repeat this process three more times (figure 2.2.9). This test was successful, proving that it would provide adequate pressure for our design. During this test, we timed how long it took to fill to three psi from zero, and vice versa to depressurize. Going in either direction took roughly three minutes which provides us with a fast enough response time for our design.

Electronics Testing

Our electronics design rig has different states that includes pressurizing, air release, and maintaining pressure. This is programmed on the Arduino IDE, and it uses the buttons on our controller to send commands to change between our defined states. The states are controlled by sending a HIGH signal to the signal port of each relay to power it on or off. We have also tested the measurement readings from all of our sensors individually as well as having multiple

measurements like air cell pressure, internal temp, and external weather variables be transmitted and read from the ground system.

Our final system is capable of merging all of these variables together, creating an array that is sent from the CloudBot transceiver to the controller transceiver. This data is then read and displayed on the Serial Output, which is available through the Arduino IDE software. For sending commands back, the states for the buttons on the controller are sent from the controller to the CloudBot, and it is able to appropriately change to the proper state. The final electronics wiring is shown below in Figure 2.3.10, with the sensors and transceiver.

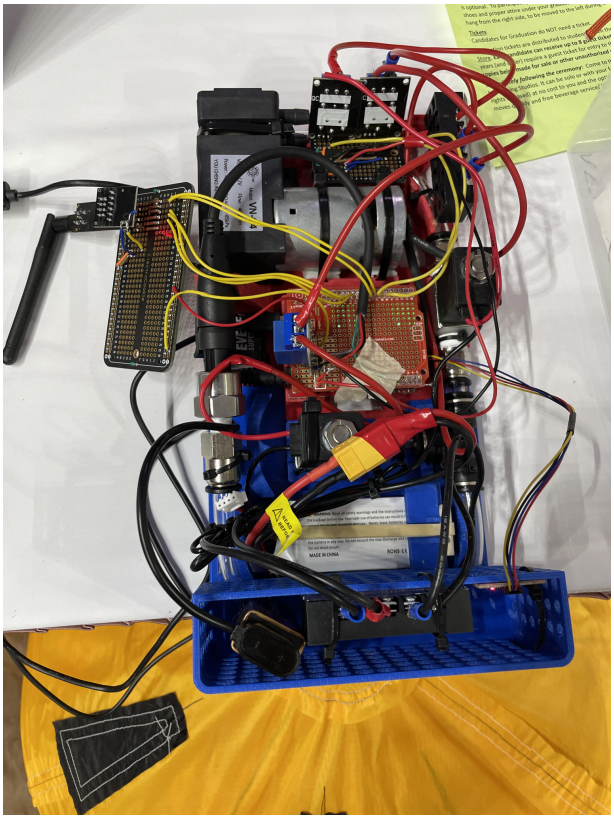


Figure 2.3.10 - Final Electronics Wiring

Full Rig Testing

Our full rig consists of the helium balloon, electronics payload, and air cell. For our first test with this setup, we prepared our system outdoors. First, we filled the air cell to a pressure

of 1.5 psi (½ the maximum pressure) using our electronics system. Next, we began to fill the helium balloon and continued until the entire system was neutrally buoyant. Due to a 5.4 knot wind, we took extra precautions by setting up tether connected to the payload. After setting the command to release air cell pressure to 0 psi, the CloudBot began to rise into the air, as seen in Figure 2.3.11.

Figure 2.3.11 - Full Rig Test Outdoors

We next sent the command to fully pressurize the air cell to 3 psi, which resulted in a slower ascend rate and then, after reaching the 1.5 psi point, the system began to descend back to the ground. Due to the wind, the balloon was getting knocked horizontally, so we decided to conduct our next test indoors. Here, we had very successful results, clearly seeing the balloon move upwards with an unpressurized air cell, hovering with 1.5 psi, and returning to the ground when pressurized more. The video of this test can be seen here (https://youtube.com/shorts/Ccup9WjAxtI). The image below shows the balloon perfectly hovering at a target altitude.

Figure 2.3.12 - Indoor Hover Test

With sensor readings occurring every 10 seconds, we graphed the external pressure, air cell pressure, external temperature, and humidity during our test flights. The results are shown below in Figure 2.3.13.

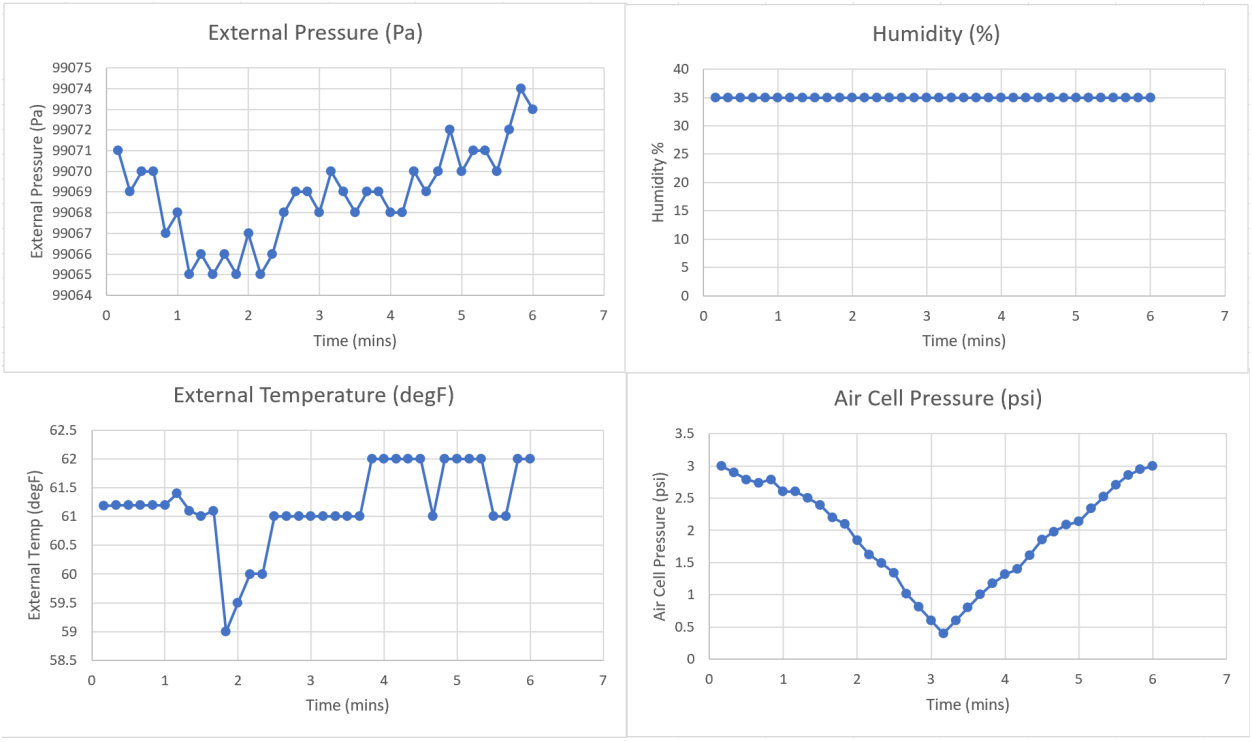


Figure 2.3.13 - Sample of Mid-Flight Sensor Measurements

All in all, our tests successfully demonstrated our proof-of-concept that the CloudBot can achieve variable buoyancy and communicate live atmospheric measurements. Our tests included both indoor and outdoor settings, featuring different levels of wind. In future tests, we could test for more weather conditions, including precipitation, different temperature ranges, and different humidity levels. Further, our helium connection functioned exactly as designed, allowing us to fine-tune the amount of helium inside the weather balloon and reuse the balloon for future tests.

iv. Design and Analysis

The primary inspiration for our design came from the Phoenix Unmanned Aerial Vehicle (UAV), a small autonomous airship that utilizes variable buoyancy and is designed to serve as an atmospheric satellite [1]. The Phoenix UAV's fuselage contains helium lifting gas and a "separate inflatable 6 cubic meter cell containing heavier air…To increase buoyancy, air in the inflatable cell is released to the atmosphere" [1]. Inspired by their work which shows that variable

buoyancy aircraft are possible on the small unmanned scale, we adopted a similar approach with some important distinctions. Rather than using a different “heavier air” in the separate air cell, we compress atmospheric air to increase the weight of the CloudBot so that we can repeatedly ascend and descend without the limitation of quantities of stored gasses. We also used a helium balloon to provide the buoyancy force. By compressing air into the air cell we can achieve variable buoyancy similar to the Phoenix by either having the weight slightly larger or slightly lower than the buoyancy force. Another important distinction to make is the means of navigation. Where the Phoenix actually used the upward and downward motion to propel itself, our CloudBot mainly relies on the different air current directions already present in the atmosphere.

Alternate Design Consideration

In the earlier stages of our design process, we were concerned with the feasibility of the air cell design. With the failure of our first two air cells, we began booking for alternate solutions. One idea was to use canisters which we would compress helium into from the weather balloon to descend. Initially, we decided against this design because of the complexity involved in getting a reliable interface between the canisters and the balloon, and because it is much more difficult to get within the FAA payload weight limit. We looked to this design again as a solution after our past failure to see if it could offer a better solution, and came up with a design for this approach. For our design, we assumed an ideal weight of 1.714 kg which is 200 grams less than our current design. This difference was to account for the air cell being replaced by 3 two liter plastic bottles. 2L bottles are designed to hold pressure, and by their nature they are pretty light weight, so it seemed to make sense. Then, if we compress our entire air system to a maximum of 20 psi which is the maximum pressure our lightweight system could reliably go up to, we would get a resulting adjustability range of 0.1072 N of Force from the equilibrium configuration of the CloudBot (0.0536 N upwards with no pressure, and 0.0536 N downwards at 20 psi). This is a huge decrease in performance in comparison to our current design at only 3 psi. Despite this being an ideal case for the compressed helium approach in our weight range

since our weight would likely be over that estimate when factoring in new plumbing and other new components needed, and because it is a challenge to get an air system of our scale and weight to reliably hold that pressure, it is out of the question because of the low performance potential. The calculations to obtain these values are shown in **Appendix i.**

Final Design Calculations

When coming up with the mechanical design of the CloudBot, there were a few governing parameters that we had to adhere to and design around. Most importantly, the payload weight cannot exceed 6 pounds, and for our design the maximum pressure in the air cell is 3 psi. We also had to consider the possibility that the decreasing air density as the CloudBot ascends could impact the performance. We looked into this concern and observed data from the NRLMSISE-00 Atmosphere Model provided by NASA. At our CloudBot's operation range of 182 to 272 meters above sea level (up to 90m or ~300ft above the ground), the air density decreases from 1.206 to 1.195 kg/m^3 (according to data recorded in past years at this location and time of year).

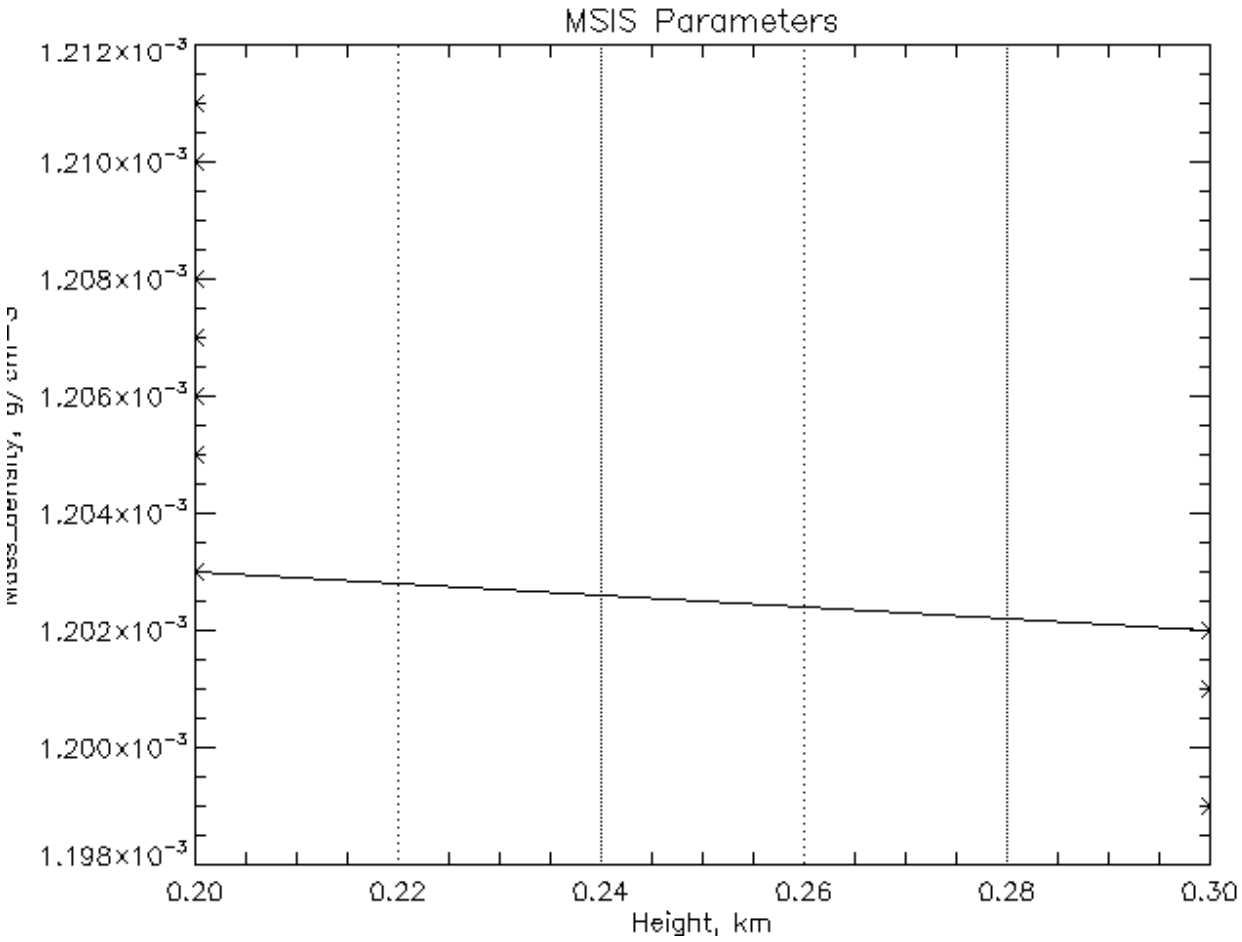


Figure 2.4.1 - NRLMSISE-00 Predicted Atmosphere Model on Test Day in Storrs

Figure 2.4.1 shows a fit trendline to the data to give an idea of the density changes with altitude. To help account for this decrease, we performed our calculations using the mean air density value of around 1.2 kg/m^3. However, it should be noted that the effects of changing air density will be negligible since as the CloudBot rises, the atmospheric pressure decreases, and the Helium balloon will expand. The resulting buoyancy force on the balloon stays constant because of this. This is the reason that the velocity of regular weather balloons stays constant throughout their entire ascent into the atmosphere until they finally burst from the balloon expanding. Likewise, in the air cell the pressure determining the increased downward force on the CloudBot is the gauge pressure, or the pressure difference between the air cell and atmospheric pressure. Therefore, regardless of the altitude we will be able to compensate for

pressure changes with our pump and should have roughly the same ranges of upward and downward forces.

With our constant parameters determined, we performed the calculations shown in **Appendix i** to estimate the performance of the final CloudBot design.

With a total weight adjustability of 0.83 Newtons, we can expect to reach a terminal velocity of 1.584 m/s. With this design, the CloudBot accelerates very slowly, however it is enough to validate our proof of concept.

For our air cell, because the tensile strength of the ripstop nylon casing could vary greatly depending on where we purchased the material and how we stitched the casing together, we could not analytically predict at which internal pressure we would reach failure. However, after testing, we could calculate the tensile strength of our design based on the point of failure from experimentation.

For a thin-walled pressure vessel w/p_0 = 0:

$$\sigma_{t,\,max} = \frac{p_i(d+t)}{2t} \qquad \text{(eqn 2.4.1)}$$

From testing, we know $\sigma_{t,\,max}$ was reached at ≈ 3.3 psi, gauge.
Using diameter d = 30 in and thickness t = 0.005 in:

$$\sigma_{t,\,max} = \frac{(3.3)(30+0.005)}{2(0.005)} = 9902\ psi$$

This is the tensile strength of our 2nd iteration air cell with the 40D ripstop nylon casing. Theoretically, if we have a known maximum tensile strength, we can rearrange *eqn. 2.4.1* to calculate maximum allowable internal pressure.

Regarding our air cell design iterations, we saw a substantial improvement from our first iteration to the second.

1st iteration: $\sigma_{t,\,max}$ was reached at ≈ 1.92 psi, gauge.

$$\sigma_{t,\,max} = \frac{(1.92)(30+0.005)}{2(0.005)} = 5761\ psi$$

2nd iteration:

$$\sigma_{t,\,max} = 9902\ psi$$

The 2nd iteration had a 72% increase in max tensile strength.

For our electronics layout, we had to modify certain parts and rework wirings. One part that has been swapped is the transceiver set we purchased. The first one had no actual code on it, requiring us to flash code onto the device to be able to use it with our Arduino. We were able to find a new one instead, that contained driver code which simplified the process.

One limitation with the new one however is that it cannot transmit messages and receive them at the same time. This means that on the ground side, we can either listen to sensor readouts or send pump commands, not both. This raises concern since the two transceivers would need to be changing states at the same time in order to be synchronized to move data between one another. To solve this, we are adding an encoding for time that is included for every transmission and will count time for our swap. Currently, we are planning on using a change from read to write every 30 seconds. Another solution that we will be adding is having pump regulation internally rather than from the ground. This means that instead of telling CloudBot to run the pump and stop it, we send our desired pressure value instead and the device will automatically regulate the pumps/valves to reach that value.

There are several ways by which we could increase the performance of CloudBot in the future. First of all, by reducing the weight of the CloudBot we would see faster acceleration of the CloudBot. Another improvement would be to increase the inner diameter of the adaptor from the beach ball to the air system. There is a narrow restriction in it that likely significantly

decreases the flow, and thus increases the time it takes to pressurize the cell. Lastly, being able to increase the maximum pressure of the air cell would allow us to reach higher terminal velocities in either direction and to have more control in high wind environments.

v. Build Specifications and Cost

Table 2.5.1 - Condensed Bill of Materials for the Entire Project

Item	Cost
Air System	$348.17
Sensors	$75.94
Electronics	$187.00
Payload	$43.29
Helium Connection	$39.61
Extra Unused Material	$42.73
Total	$736.74

Table 2.5.2 - Condensed Bill of Materials for the Final CloudBot

Item	Cost
Air System	$293.94
Sensors	$75.94
Electronics	$164.01
Payload	$43.29
Helium Connection	$39.61
Total	$616.79

Table 2.5.3 - Total Metrics of Our Design

Final CloudBot Cost*	$617
Payload Weight	5.7 lbs

*Cost does not include helium

See **Appendix iii.** Comprehensive Build Specifications and Cost Breakdown for the full breakdown of parts.

CloudBot Dimensions

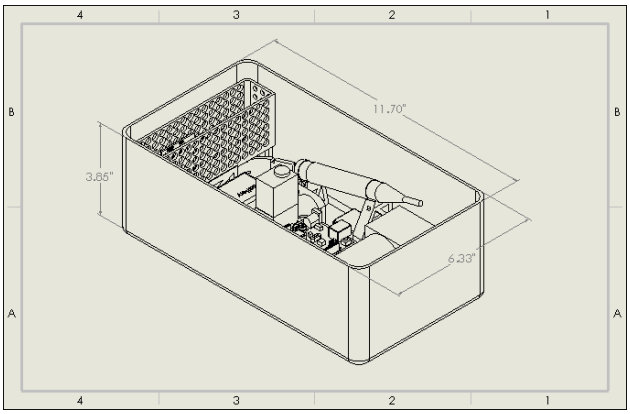


Figure 2.5.4 - Payload Dimensions

The payload shell alone has dimensions shown above in Figure 2.5.4.

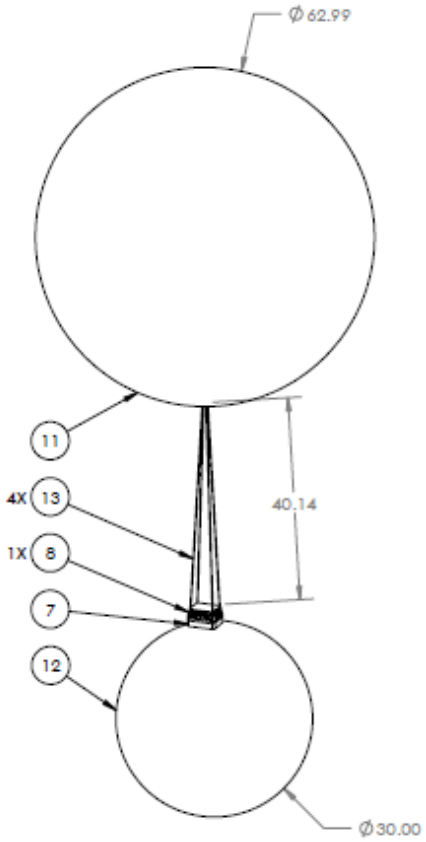


Figure 2.5.5 - Top Level Model Dimensions

Fully assembled and filled with helium, the CloudBot will stand just under eleven feet tall, as shown in figure 2.5.5.

III. Summary and Conclusions

In this project, UConn Professor George Matheou tasked our group with building a CloudBot: an autonomous weather balloon capable of collecting live atmospheric measurements while being lightweight, low-budget, and durable. The CloudBot aims to improve extreme weather forecasts by being able to collect measurements within storms and serves as a proof-of-concept that a weather balloon can achieve both ascent and descent mid-flight.

To achieve this goal, the operating principle of the CloudBot is variable buoyancy. By compressing or releasing air in the air cell, we can manipulate the density of the CloudBot, resulting in the CloudBot's ascent or descent. This makes the CloudBot unique from traditional weather balloons, which typically only ascend in the atmosphere and then pop once reaching a certain altitude. Furthermore, prior to launch, the CloudBot can be pre-programmed with a flight plan; the CloudBot can also be manually controlled during flight with the ground control. The main limitation of the CloudBot is that it must have a payload of 6 lbs or less in order to adhere to FAA regulations and be legally defined as a weather balloon.

Our air and electronics systems are fully functional both on their own and in conjunction with each other. Our helium connection allows us to securely inflate and deflate the overhead helium balloon without having to pop the balloon after each flight. This ability to refill is a critical part of our design that distinguishes the CloudBot from other weather balloons and allows us to perform test flights efficiently. Our custom 3D printed bed securely houses our electronics, sensors, and pump within our payload shell. We have successfully tested the CloudBot in calm and controlled indoor conditions as well as moderate windy conditions, and we have been able to collect data throughout test flights. We have demonstrated proof-of-concept for the overall operating principles of the CloudBot.

Next steps for the CloudBot would be improving communications and altitude control, cutting down payload weight, implementing a fail-safe parachute with servo deployment, and designing test plans for utilizing wind currents for navigation. Different altitudes have specifically directioned wind currents, so it is possible to pre-program a flight plan into the CloudBot with predetermined altitude levels in order to steer the CloudBot where we want it to

go. Given its versatility, we are confident the CloudBot has a future in collecting meteorological data and providing invaluable insight during weather disasters.

IV. References

[1] P. Lobner, “Phoenix Makes Its First Flight With Variable Buoyancy Propulsion. What’s Old is New Again!,” *The Lyncean Group of San Diego*, 01-May-2019. [Online]. Available: https://lynceans.org/all-posts/phoenix-makes-first-flight-with-variable-buoyancy-propulsion/. [Accessed: 17-Oct-2021].

[2] “Weather Balloon Safety & Regulations: Olhzn high altitude balloons,” *OLHZN High Altitude Balloons | Canandaigua, NY*, 13-Sep-2019. [Online]. Available: https://www.overlookhorizon.com/flight-safety/. [Accessed: 06-Dec-2021].

[3] “How Does the Coriolis Effect Influence Wind Direction at Different Heights?,” *How does the coriolis effect influence wind direction at different heights?* [Online]. Available: https://geography.name/how-does-the-coriolis-effect-influence-wind-direction-at-different-heights/. [Accessed: 06-Dec-2021].

[4] “What is the Coriolis Effect?,” *What is the coriolis effect?* [Online]. Available: https://geography.name/what-is-the-coriolis-effect/. [Accessed: 06-Dec-2021].

[5] D. Czernia and B. Szyk, “Air Density calculator,” *What is the Density of Air?*, 18-Jan-2021. [Online]. Available: https://www.omnicalculator.com/physics/air-density. [Accessed: 06-Dec-2021].

[6] “Density of the elements,” *Density for all the elements in the Periodic Table*. [Online]. Available: https://periodictable.com/Properties/A/Density.al.html. [Accessed: 06-Dec-2021].

[7] A. Krauchi and G. Romanens, “(PDF) Controlled weather balloon ascents and descents for atmospheric research and climate monitoring,” *ResearchGate*, Dec-2015. [Online]. Available: https://www.researchgate.net/publication/285629102_Controlled_weather_balloon_ascents_and_descents_for_atmospheric_research_and_climate_monitoring. [Accessed: 15-Oct-2021].

[8] D. Ward, “Severe Weather,” *University of Arizona ATMO336*, 2009. [Online]. Available: http://www.atmo.arizona.edu/students/courselinks/fall10/atmo336/lectures/sec2/hurricanes.html. [Accessed: 15-Oct-2021].

[9] B. R. Munson, T. H. Okiishi, W. W. Huebsch, and A. P. Rothmayer, *Fundamentals of Fluid Mechanics*. Hoboken, NJ: John Wiley & Sons, Inc., 2013.

[10] M. J. Moran, H. N. Shapiro, D. D. Boettner, and M. B. Bailey, Fundamentals of Engineering Thermodynamics. Hoboken, NJ: Wiley, 2014.

[11]“Jean Baptiste Meusnier,” *Military Wiki*. [Online]. Available: https://military.wikia.org/wiki/Jean_Baptiste_Meusnier. [Accessed: 17-Oct-2021].

[12] “GNSS / GPS technology differences,” TerrisGPS. [Online]. Available: http://www.terrisgps.com/gnss-gps-differences-explained/. [Accessed: 17-Oct-2021].

[13] “Sparkfun GPS breakout - NEO-M9N, SMA (Qwiic),” GPS-17285 - SparkFun Electronics. [Online]. Available: https://www.sparkfun.com/products/17285. [Accessed: 17-Oct-2021].

[14] “MakerFocus 2pcs NRF24L01P+PA+LNA RF Wireless Transmission Module 2.4ghz ML01DP5 22dBm 100MW 2300M measured distance SPI interface with antenna anti theft anti-interference,” Amazon.in: Computers & Accessories. [Online]. Available: https://www.amazon.in/MakerFocus-nRF24L01P-Transmission-Interface-Anti-Interference/dp/B07QC1SXJ8. [Accessed: 17-Oct-2021].

[15] “Benefits of Radio Frequency Technology,” Stalam. [Online]. Available: https://www.stalam.com/eng/technology-and-benefits/benefits. [Accessed: 17-Oct-2021].

[16] S. Liberty, “Weather Balloon Payload Box,” *Central Washington University*, 2017. [Online]. Available: https://digitalcommons.cwu.edu/cgi/viewcontent.cgi?referer=&httpsredir=1&article=1047&context=undergradproj. [Accessed: 15-Oct-2021].

[17] A. the A. Sushil, “Ultra-durable uavs fly with variable buoyancy technology: FRP Roofing Sheet , flat panel making machine manufacturer,” *FRP roofing sheet , flat panel making machine manufacturer | Leading manufacturer of FRP sheet making machines*, 27-Apr-2019. [Online]. Available: https://frptitan.com/variable-buoyancy-technology/. [Accessed: 06-Dec-2021].

[18] “Whitworth University,” *High-Altitude Weather Balloon Research | Engineering & Physics | Whitworth University*. [Online]. Available: https://www.whitworth.edu/cms/academics/engineering-and-physics/high-altitude-weather-balloon-research/. [Accessed: 06-Dec-2021].

[19] “Flight Environment - Prevailing Winds,” *Prevailing winds*. [Online]. Available: https://www.weather.gov/source/zhu/ZHU_Training_Page/winds/Wx_Terms/Flight_Environment.htm. [Accessed: 06-Dec-2021].

[20] “The Federal Register,” *Federal Register :: Request Access*. [Online]. Available: https://www.ecfr.gov/current/title-14/chapter-I/subchapter-F/part-101. [Accessed: 10-April-2022].

[21] “The Federal Register,” *Federal Register :: Request Access*. [Online]. Available: https://www.ecfr.gov/current/title-47/chapter-I/subchapter-B/part-22/subpart-H/section-22.925. [Accessed: 10-April-2022].

[22] “Hobbywing QuicRun 1060 Brushed : Toys & Games - Amazon.com.” *Amazon*, https://www.amazon.com/Hobbywing-HWI30120201-Quicrun-1060-Brushed/dp/B01LZHBJ85. [Accessed: 10-April-2022].

[23] “GP.” *Amazon*, Goettsche Partners, 2011, https://www.amazon.com/gp/product/B07H4R7QNC/ref=ox_sc_act_title_2?smid=A28ZWXW3ZSVNZU&psc=1. [Accessed: 10-April-2022].

[24] dmaners1. “Sparkfun Atmospheric Sensor Breakout - BME280 (Qwiic).” *SEN-15440 - SparkFun Electronics*, 1 Jan. 1969, https://www.sparkfun.com/products/15440. [Accessed: 10-April-2022].

[25] #789266, Member, et al. “Zio Current and Voltage Sensor - INA219 (Qwiic).” *SEN-15176 - SparkFun Electronics*, https://www.sparkfun.com/products/retired/15176. [Accessed: 10-April-2022].

[26] Kuklovskaiâ Elizaveta. “DP.” *Amazon*, "Books by Mail" Pub. Co., https://www.amazon.com/dp/B0748BXFYQ/ref=sspa_dk_detail_1?pf_rd_p=9fd3ea7c-b77c-42ac-b43b-c872d3f37c38&pd_rd_wg=Nml4i&pf_rd_r=W2XGSNAMDW2NNXK6GBB6&pd_rd_w=NMKi3&pd_rd_r=59d605d8-76bc-4a4e-9c87-ba1c2dda7a77&smid=A36ZH2MCHPKXUA&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEzSUlOS1ZBMTk5TUxTJmVuY3J5cHRlZElkPUEwNTEyODk1M0NFTDJVUDNCWUtNQiZlbmNyeXB0ZWRBZElkPUEwNzY4NzA2MTlPMlJIS1VSMVdDUCZ3aWRnZXROYW1lPXNwX2RldGFpbCZhY3Rpb249Y2xpY2tSZWRpcmVjdCZkb05vdExvZ0NsaWNrPXRydWU&th=1. [Accessed: 10-April-2022].

[27] c2builder, et al. “Temperature Sensor - Waterproof (DS18B20).” *SEN-11050 - SparkFun Electronics*, https://www.sparkfun.com/products/11050. [Accessed: 10-April-2022].

[28] #505814, Member, et al. “Sparkfun Altitude/Pressure Sensor Breakout - MPL3115A2.” *SEN-11084 - SparkFun Electronics*, https://www.sparkfun.com/products/11084. [Accessed: 10-April-2022].

[29] “Amazon.com: Keeyees 5pcs 1 Channel 5V Relay Module Board Shield Ky-019 ...” *Amazon*, https://www.amazon.com/KeeYees-Channel-Indicator-Arduino-Raspberry/dp/B07L6J6FHH. [Accessed: 10-April-2022].

V. Appendices

i. Literature Review

The purpose of the literature review was to gain familiarity with the scope of our project and be better prepared for designing a feasible product. In our case, this meant studying weather balloons in depth, as well as how we could use buoyancy to accomplish our goal. We also used the literature review to explore the electronics we would need for this project.

Background, Setting, and the Need for an Autonomous, Buoyancy-Regulated Balloon

Weather balloons, also known as sounding balloons, began to be used for meteorological research over a century ago. French meteorologist Léon Teisserenc de Bort pioneered the use of weather balloons, specifically those unmanned and instrumented, in the late 1800s. Radiosondes--telemetry devices that collect data about altitude, pressure, temperature, humidity, and wind--were invented in France by Robert Bureau in 1929 [7]. Given strong evidence of climate change and greater knowledge regarding atmospheric composition playing a role in the radiative effects in Earth's climate system, in situ climate observations become increasingly important and serve as motivation for our Senior Design project. Weather balloons are currently the preferred method of collecting in situ atmospheric measurements due to complications in harsh weather conditions, hence the decision to go with a weather balloon as opposed to a drone or conventional aircraft for the CloudBot.

Currently, there are two primary techniques for weather ballooning as discussed in this article: a single balloon with a valve to release helium from the balloon once it reaches a certain ambient pressure, and a double-balloon consisting of a hydrogen-filled carrier balloon and a helium-filled balloon to assist with descent after the carrier balloon bursts [7]. However, our project wishes to change vertical directionality mid-flight, rather than just at the end. We drew inspiration for variable buoyancy from these ideas and implemented it in the form of two balloons, but with one varying atmospheric air content. This way, we can retrieve the balloon without having it burst.

We look to another source to gain insight regarding storm conditions and the expected operating conditions for the CloudBot. This comes from lecturer Dr. Dale Howard of the University of Arizona's Department of Hydrology and Atmospheric Sciences. The lowest storm pressure ever recorded is 882 millibars (12.8 psi) from Hurricane Wilma in 2005 [8]. Atmospheric pressure is known to be ~14.7 psi. Keeping these in mind, we set our pressure range for the CloudBot to be between 12.8-15 psi. We also learn that wind speeds and surface

pressure have an inverse linear relationship [8]. Though meteorologists assign the category of the hurricane based on wind speed, we can see that a more severe storm is caused by a lower barometric pressure. Understanding the inner workings of how storms form and behave is critical to effectively designing the CloudBot to read measurements and properly interpret results.

Buoyancy Concept / Calculations

What makes our CloudBot unique from other weather balloons is its ability to change its weight mid-flight to ascend or descend in the atmosphere. Essentially, the CloudBot will alternate between being lighter than air and heavier than air. To understand how our design leverages the concept of buoyancy, we must be able to calculate the buoyancy force on the aircraft. According to the *Fundamentals of Fluid Mechanics* textbook, the buoyancy force acting on an object fully submerged in one fluid is calculated as follows:

$$F_B = V_{object} * g * \rho_{fluid} \qquad (eqn.\ 5.1.1)$$

Using the volume in cubic meters, the density of the surrounding fluid in kilograms per cubic meter, and a gravitational constant in meters per second squared, we can obtain the buoyancy force acting upwards on the object in Newtons [9]. The other main component that we must consider in the control of the CloudBot is the total weight acting downwards. This is simply calculated using this equation:

$$W = m * g \qquad (eqn.\ 5.1.2)$$

With these two forces acting on a body, the resulting vertical force is as follows:

$$F_{net} = F_B - W_{total} \qquad (eqn.\ 5.1.3)$$

Thus, by adjusting either the buoyancy force of an object or its weight we can determine if it will move up or down. The three main components of the CloudBot to consider are the helium balloon, the payload shell which contains all the electronics, and the external air cell. All masses and volumes of these components remain constant, as well as the amount of air in the helium balloon. Therefore, the buoyancy force remains constant, so we can only change the aircrafts overall buoyancy by compressing air into the lower air cell to increase the density of the air within it. This relationship of pressure to density of a gas was estimated using the ideal gas equation:

$$n = \frac{PV}{RT} \qquad (eqn\ 5.1.4)$$

Equation 4 was used to solve for the number of moles in our air cell by inputting absolute pressure in Pascals, volume of the cell in cubic meters, the absolute temperature in Kelvin, and

the ideal gas constant (R=8.314 kJ/kmol-K). Then, knowing the number of moles in the cell with a given pressure, we can find the resulting mass by multiplying by the molar mass of the air that was compressed into the cell [10]. These relationships were used to design critical aspects of the CloudBot including volume of the helium balloon, volume of the air cell, and necessary pressure in the air cell to reach appropriate performance metrics.

Our CloudBot is not the first vehicle to make use of this concept to reach various altitudes. In fact, Jean Baptiste Meusnier first introduced a very similar invention in 1783 known as the ballonet. This was a separate bag of air that was added inside of the main balloon of his helium airship. The idea was that by compressing atmospheric air into the ballonet, one could increase the weight of the aircraft and bring it down in the atmosphere and then release it to gain altitude again. In 1784, he built and flew a working 84-meter-long airship piloted by an 80 man crew which used this general concept [11]. Though this idea is very similar to the air cell we decided to incorporate on the CloudBot, ours is external to the helium balloon to improve simplicity of the design and to streamline repairs since our aircraft will eventually be subjected to far more harsh conditions. We are also building ours on a much smaller scale with the goal of reducing cost since it is likely we may lose or damage the balloon or the air cell.

A more modern example of an aircraft that uses this principle is the Phoenix UAV which is currently being developed by an array of professional teams and UK Universities. The Phoenix is a smaller scale variable buoyancy airship at roughly 49 feet long, and it uses its changing buoyancy as means to propel itself forwards. The Phoenix made its first successful unmanned flight in 2019 and eventually it will serve to transmit information long range for a variety of applications [1]. This flight proved the possibility of an unmanned variable buoyancy aircraft for the first time which encouraged us to use a similar approach for designing our CloudBot even though they are for very different purposes.

Our CloudBot project takes inspiration from both of these designs but will find a new application for variable density aircraft. From the research we have done, there have not been any weather balloons that operate using variable density. This aspect is advantageous since it would allow us to observe more precise locations in the atmosphere, particularly in severe storm conditions that many conventional aircraft are unable to operate in. Our solution of using an air cell, similar to a ballonet, seems to be very effective in this environment since we can ascend and descend as many times as desired unlike other airships that must release mass or their lifting gas into the environment to achieve variable densities. In addition, it is a relatively simple design allowing us to build it more robustly and with easily interchangeable parts in the case where we do not get the CloudBot back in one piece.

<u>Final design: Compressing atmospheric air into air cell, keeping helium balloon volume constant</u>

$P_{cell,max} = 3\ psi$

$Altitude\ range\colon\ 182\ -\ 272\ m\ above\ sea\ level$

$\rho_{air,mean} = 1.2\ kg/m^3$

$V_{cell} = 0.338\ m^3$

$m_{cell} = 0.46\ kg \quad m_{payload} = 0.854\ kg \quad m_{balloon} = 0.60\ kg \quad m_{total} = 1.914\ kg$

$\rho_{He} = 0.179\ kg/m^3$

$V_b = Volume\ of\ balloon\ m^3$

- Force balance with zero psi of compressed air in air cell
  Weight: $w = V_b * \rho_{He} * 9.81 + m_{total} * 9.81 = 1.756V_b + 18.78$
  Buoyancy: $F_b = V_b * 9.81 * 1.2 = 11.772V_b$
  Net Force: $|F_y| = F_b - w = 10.016V_b - 18.78$

- Force balance with 3 psi of compressed air in air cell
  $3\ psi = 20.684\ kPa \quad P_{atm} = 99.238\ kPa \quad P_{cell,absolute} = 119922.86\ Pa$
  $\rho_{3psi,air} = P_{cell,absolute}/R_sT = 1.45\ kg/m^3$
  Weight: $w = (V_b(\rho_{He}) + V_{cell}(\rho_{3psi,air}) + m_{total}) * 9.81 = 1.756V_b + 23.584$
  Buoyancy: $F_b = V_b(\rho_{air,mean}) * 9.81 + V_{cell}(\rho_{air,mean}) * 9.81 = 11.772V_b + 3.98$
  Net Force: $|F_y| = F_b - w = 19.605 - 10.016V_b$

- Set upward and downward net forces equal
  $19.605 - 10.016V_b = 10.016V_b - 18.78$
  $V_b = 1.916\ m^3\ of\ helium\ required\ for\ this\ design$
  $|F_y| = 0.415\ N\ of\ force\ in\ either\ direction$

- Terminal velocity of this design
  The majority of the drag coefficient is caused by the large helium balloon. If we assume the balloon to be a sphere, the drag coefficient of this design is 0.47. Then, terminal velocity occurs when $|F_y| = F_{drag}$, where $F_{drag} = 0.5\rho u^2 C_d A$
  Frontal area is found by calculating the diameter of the helium balloon from its volume found above.
  $0.415 = 0.5(1.2)u^2(0.47)\pi(0.864/2)^2$
  $u_{max} = 1.584\ m/s$

<u>Alternate Design: Compressing helium from balloon into plastic 2L bottles</u>

$\rho_{He} = 0.179\ kg/m^3$

$\rho_{air,mean} = 1.2\ kg/m^3$

$m_{total} = 1.714\ kg$

$P_{compressed,max} = 20\ psi$

$V_{3\ bottles} = 6L = 0.006\ m^3$

$V_b = Volume\ of\ balloon\ m^3$

- For no compressed helium (upward motion)
  Weight: $w = V_b(\rho_{He})(9.81) + m_{total}(9.81) = 1.756V_b + 16.814$
  Buoyancy: $F_b = V_b(\rho_{air,mean})(9.81) = 11.722V_b$
  Net Force: $F_y = 10.016V_b - 16.814$

- Compressing the bottles to 20 psi with helium from the balloon
  $MM_{He} = 4g/mol$
  $R_s = R/MM = 8314/4 = 2078.5$
  $P = 20\ psi = 237.13\ kPa$
  $\rho_{compressed\ He} = P/RT = 0.396\ kg/m$

- Resulting change in volume of Helium balloon
  $n = PV/RT = 99238 * 0.006/(8314 * 288.15) = 2.485 * 10^{-4}\ moles\ initially\ in\ 3x\ 2L\ bottles\ (P_{atm})$
  $n = PV/RT = 237130 * 0.006/(8314 * 288.15) = 5.939 * 10^{-4}\ moles\ in\ 3x\ compressed\ 2L\ bottles$
  $\Delta n = 3.454\ moles\ removed\ from\ helium\ balloon$
  $\Delta V_b = \Delta nRT/P_{atm} = 0.00834\ m^3\ of\ Helium\ removed\ from\ balloon$

- New force balance with 2L bottle compressed to 20 psi with helium from balloon
  Weight:
  $w = ((V_b - 0.00834)(0.179) + 0.006(0.396) + 1.714)9.81 = 1.756V_b + 16.823$
  Buoyancy: $(V_b - 0.00834)(1.2)(9.81) = 11.772V_b - 0.0982$
  Net Force: $F_y = 1.756V_b + 16.823 - 11.772V_b + 0.0982 = 16.921 - 10.016V_b$

- Now set net force magnitude for upward and downward motion equal to find volume of Helium
  $20.032V_b = 33.735$
  $V_b = 1.684\ m^3\ of\ helium\ required$
  $F_y = 0.0536\ N\ of\ force\ in\ either\ direction$

- Terminal Velocity of this design

  $C_d = 0.47$

  $A = 1.713m^2$

  $\rho = 1.2\ kg/m^3$

  $F_y = 0.0536N$

  $0.0536 = 1/2(1.2)(u^2)(0.47)(1.713)$

  $u_{max} = 0.333m/s$

Positioning System

Our entire system needs to be able to operate autonomously and thus make decisions on its own. For this, we will be using an Arduino microcontroller. A crucial part of CloudBot is the ability for the balloon to track its location to associate with temperature and pressure sensor readings. There are two primary types of positioning modules, which include GNSS and GPS. While GPS solely uses the Global Navigation Satellite System, a GNSS module will be able to read those same signals, as well signals from GLONASS, Galileo, and BeiDou. [12] We plan on using the GNSS module to be able to have access to more satellites, thus giving us increased accuracy and reliability for position coordinates. Specifically, we have decided to move forward with a NEO-M9N chip, which has a built-in GNSS receiver. This chip also includes an onboard power supply and memory, allowing it to continue collecting and storing positional data that can be retrieved in the case of damage to other electrical components within the payload [13].

Communication Protocol

An important requirement for our device is that it needs to be able to transmit and receive data in real-time. The balloon will need to frequently send latitude and longitude coordinates, as well as pressure and temperature readings, while receiving commands such as manual altitude target or initiating a landing. In order to do this, we plan on having a different Arduino on the ground connected to our operator laptop. Rather than adding a separate transmitter and receiver for each Arduino for the laptop and balloon to communicate, both microcontrollers will be connected to their own transceivers. A transceiver is able to use radio waves to transmit and receive data. We have decided to use a RF ML01DP5 transceiver module which can operate below freezing, which is ideal for our high altitude needs, and communicate from a distance of 2300 meters. While this is a higher range than our current testing plan, it will ensure a stronger connection at closer distances and allow us to properly test the communication protocol we will be coding for monitoring and controlling the balloon [14].

There are many benefits for choosing an RF (Radio Frequency) transmission module over other types of communication methods. The biggest factor is that radio frequency allows us to communicate with the balloon at larger distances, while also keeping the module dimensionally small and light. [10] RF also has negligible effects due to weather, which is especially important since CloudBot is designed to collect weather data from tornados. The ability to communicate in extreme weather and constantly relaying position and sensor readouts is a necessity for the system. These design and communication differences make RF the ideal candidate for the communication aspect of our design.

ii. Theory

i. Governing Equations

The most important equation governing our project is the buoyancy force equation:

$$F_B = V * g * \rho \qquad \textit{(eqn 5.2.1)}$$

In this equation, F_B is the buoyancy force upwards of an object submerged in a fluid in Newtons. *V* is defined as the volume of the submerged object in cubic meters. ρ is the density of the fluid that the object is submerged in in kilograms per cubic meter. In our case, this is atmospheric air at the CloudBot's location. *g* is the gravitational constant in meters per second squared.

Our second governing equation defines the downward force, or weight of the CloudBot which is changed to be either larger or smaller than the upward buoyancy force:

$$W = (V_{balloon} * \rho_{He} + V_{cell} * \rho_{compressed\ air} + m_{cell} + m_{balloon} + m_{payload}) * g$$

(eqn. 5.2.2)

We go into further detail on the components of the CloudBot later on, but this equation yields the weight downwards in Newtons. The section in parenthesis represents the total mass of the CloudBot in kilograms. This includes the masses of the gases existing in the helium balloon and the compressed air cell, which is found by multiplying the density in kilograms per cubic meter by the volume that it occupies in meters cubed. The remaining masses are

indicated by their subscript and are in kilograms. *g* is the gravitational constant in meters per second squared.

Now, we combine these two equations to find the total force balance of the CloudBot:

$$F_{net} = F_B - W \qquad \textit{(eqn. 5.2.3)}$$

The last governing equation we relied on is the ideal gas law:

$$PV = nRT \qquad \textit{(eqn. 5.2.4)}$$

In equation 1.4, P is absolute pressure in Pascals, V is the volume of the gas in cubic meters, n is the number of moles of the gas, R is the ideal gas constant, and T is the absolute temperature in Kelvin. This equation is required in order to find the density of the compressed air in the air cell at various pressures which is necessary to find the downward force on the CloudBot.

ii. Unit Problems

The unit problems were designed to help our team better understand the system we are working with by breaking it down into basic problems from which we can build upon to mimic our actual project. Our group did two unit problems, namely 1) The Velocity Profile of 2-D Laminar Flow in a Pipe, and 2) Examining Pressure Losses due to Friction from 2-D Laminar Flow in a Pipe.

Introduction

When analyzing fluids, laminar flow occurs when the flow moves in straight streamlines parallel to one another. Because of this property, laminar flows are much easier to predict than turbulent flows which are more random and by finding the velocity profile of the fluid we can go on to find other properties including velocity at some radial point in a tube, friction factor, pressure losses, and more. By analyzing the pressure differential required to keep the same volumetric flow rate when using tubes of various diameters, we can use concepts of laminar flow to optimize the air system from the pump to the valves and air cell on our CloudBot.

Part 1

Methods

For our base problem, we have 2-D, fully-developed laminar flow in a circular pipe with no-slip condition at the wall.

The velocity profile for fully developed laminar flow in a circular pipe is as follows:

$$u(r) = u_c\left(1 - \left(\frac{r}{R}\right)^2\right) \qquad \textit{(eqn. 5.2.5)}$$

$$u_c = \frac{R^2}{4\mu}\left(-\frac{\partial p}{\partial z}\right) \qquad \textit{(eqn. 5.2.6)}$$

$$\frac{\partial p}{\partial z} = -\frac{8\mu Q}{\pi R^4} \qquad \textit{(eqn. 5.2.7)}$$

$$Q = VA = V\left(\pi\frac{D^2}{4}\right) \qquad \textit{(eqn. 5.2.8)}$$

Where u is the fluid flow velocity (m/s), u_c is the centerline velocity (m/s), r is the radius variable (m), R is the fixed radius of the pipe (m), $\frac{\partial p}{\partial z}$ is the pressure differential (Pa/m), Q is the volumetric flow rate (m^3/s), and V is the inlet velocity (m/s).

For Part 1, we have these values:

Inlet velocity V	1 m/s
Dynamic viscosity μ	0.002 kg/m-s
Diameter D	0.2 m
Density ρ	1 kg/m^3

Table 5.2.1 - Flow Conditions and Properties for Unit Problem Part 1

Results

Using the values in table 5.2.1, we find the analytical solution for Part 1:

$$u(r) = 2(1 - (\frac{r}{0.1})^2) \qquad \textit{(eqn. 5.2.9)}$$

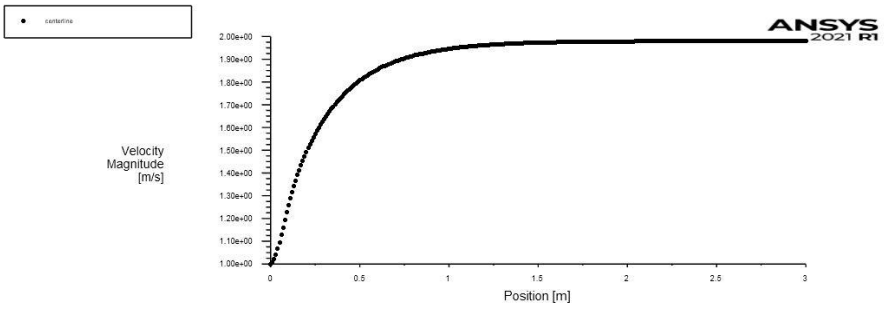


Figure 5.2.2 - Velocity Profile for Unit Problem Part 1

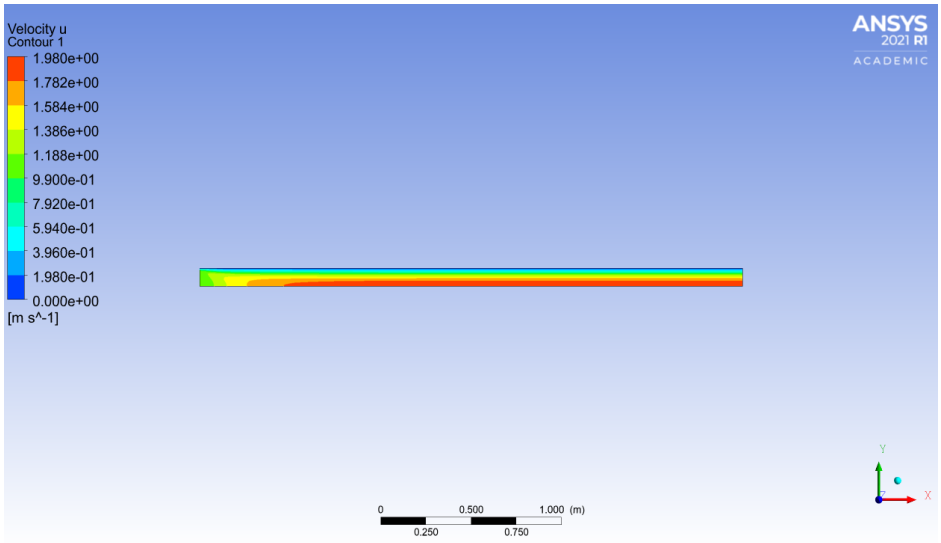


Figure 5.2.3 - Contour Plot of Velocity Profile for Unit Problem Part 1

We find 1.0% error between the simulation and the analytical solution. This is due to the simulation having a finite number of iterations (namely, 500). If we were to increase the number of iterations in the simulation, we would find higher accuracy to the analytical solution. In the simulation we find u_c = 1.980 m/s while the analytical solution has u_c = 2.000 m/s.

Part 2

Methods

For the second part of the unit problem, we want to learn about the pressure losses due to friction in laminar flow in a pipe which may be useful to know when designing our CloudBot. To do this, we used the dimensions of our tubing that we will use on our air system along with the viscosity and density properties of air, and calculated the Darcy Friction Factor and pressure loss per unit length using the Darcy-Weisbach equation. To calculate the theoretical inlet velocity, we used the volumetric flow rate of a pump option we were considering (14 L/min), and the cross sectional area of two different tube options. The first tube size we analyze has an inner diameter of a quarter of an inch because this is what we used in our design. Then we looked at a tube with a half inch diameter to see any potential improvements in efficiency of the system.

This problem also has 2-D, fully-developed laminar flow in a circular pipe with no-slip condition at the wall.

$$Friction\ factor,\ f = \frac{64}{Re} = \frac{64\mu}{\rho V_m D} \qquad \textit{(eqn. 5.2.10)}$$

$$f = \frac{2\Delta\rho D}{\rho V_m^2 L} \qquad \textit{(eqn. 5.2.11)}$$

$$\frac{\Delta p}{L} = \frac{dp}{dz} = -\frac{8\mu Q}{\pi R^4} \qquad \textit{(eqn. 3.2.8)}$$

The following values are similar to those of our project conditions:

	¼ Inch (6.35 mm) ID Tube Design	½ Inch (12.7 mm) ID Tube Design
Inlet velocity V	7.3678 m/s	1.8420 m/s
Dynamic viscosity μ	$1.7894*10^{-5}$ kg/m-s	$1.7894*10^{-5}$ kg/m-s
Diameter D	1/4 in = 0.00635 m	1/2 in = 0.0127 m
Flow rate Q	14 L/min = 0.00023333 m^3/s	14 L/min = 0.00023333 m^3/s

Air Density ρ	1.225 kg/m^3	1.225 kg/m^3

Table 5.2.4 - Flow Conditions and Properties for Unit Problem Part 2

Results

	¼ Inch (6.35 mm) ID Tube Design	½ Inch (12.7 mm) ID Tube Design
Friction factor f	0.01998	0.03996
Pressure loss $\frac{\Delta p}{L}$	104.63 Pa/m	6.5393 Pa/m
Centerline Velocity u_c	14.74 m/s	3.68 m/s

Table 5.2.5 - Analytically Calculated Friction Factor and Pressure Loss for Unit Problem Part 2

We also simulated the velocity profiles for each diameter in Ansys.

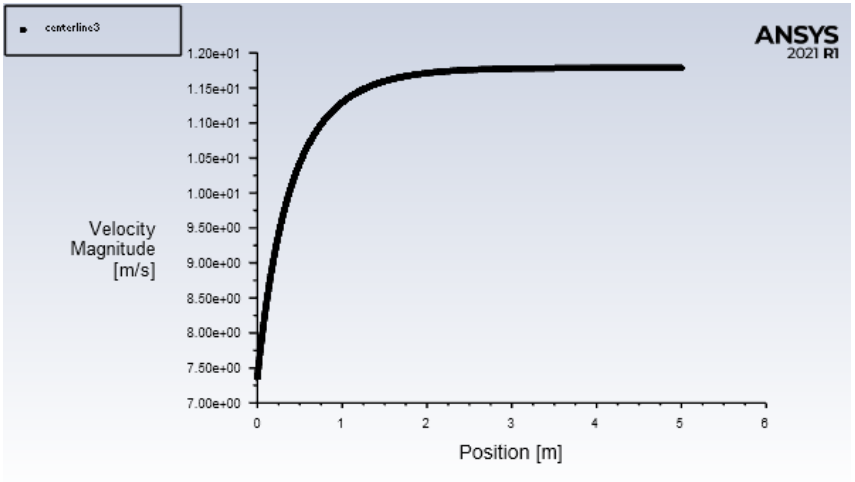


Figure 5.2.6 - Velocity Profile for an Inner Diameter (ID) of 0.25 in

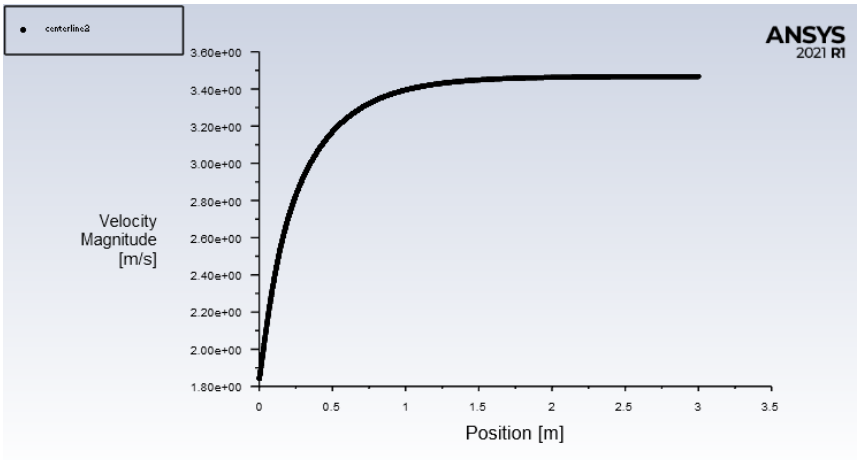


Figure 5.2.7 - Velocity Profile for an Inner Diameter (ID) of 0.5 in

Considering the analytical calculations, a pressure loss of 105 Pa/m is equivalent to about 0.005 psi/ft in the quarter-inch tube, and in the half-inch tube 6.5 Pa/m equates to 0.0003 psi/ft which seems reasonable. The pressure drop per unit length is roughly 17 times with the larger tube indicating a large potential to save energy with this configuration.

Regarding the simulated velocity profiles, we see some deviation from the calculated values. For the quarter-inch tube, we expect to see a centerline velocity of 14.74 m/s, but the simulation outputs a centerline velocity of roughly 11.8 m/s. This is an error of 19.9%.
For the half-inch tube, we expect to see a centerline velocity of 3.68 m/s, but the simulation outputs a centerline velocity of roughly 3.48 m/s. This is an error of 5.43%.

While helpful for visualizing, the Ansys outputs produce varying amounts of error. This is important to consider when relying on a simulation to understand a system better. More iterations would decrease this error, but at the cost of more time and more resources.

The most important information that we took away from this unit problem for our design is that keeping the diameter as large as possible will improve the efficiency and speed of our system. Any connections that restrict the air flow could be costly and any precautions should be taken to avoid restrictions.

iii. Key Concepts

Another important concept becomes relevant for the navigation of our CloudBot: wind currents. There are multiple factors that contribute to the way air moves in the atmosphere. First, the force caused by a pressure gradient from high to low pressure. Second, the Coriolis effect. And third, friction with Earth's surface, which does not itself change wind direction, but affects the previous two forces. Pressure gradients initiate air motion, but the direction is largely influenced by the Coriolis effect [3]. The Coriolis effect is one whereby a mass moving in a rotation experiences a force (the Coriolis force) acting perpendicular to the direction of motion and to the axis of rotation. On Earth, this refers to the apparent deflection in the path of a moving object in response to the Earth's rotation and axis tilt [4]. In the Northern Hemisphere, the Coriolis effect tends to deflect moving objects to the right; in the Southern Hemisphere, to the left. Due to friction near the surface of the Earth, wind speeds are slower. Slower wind speeds correlate to a weaker Coriolis force. The higher the altitude (i.e. the further from the Earth's surface), the weaker the friction force, allowing wind speeds to increase. Higher wind speeds increase the Coriolis effect, resulting in a change in wind direction as altitude increases [3]. The takeaway for navigating the CloudBot is that wind currents move in different directions at different altitudes. By sending the CloudBot to a specific height, we can make it move in the direction we want. We will take advantage of this atmospheric effect and discuss more in our electronics design, notably how we will program flight plans based on altitude.

iii. Comprehensive Build Specifications and Cost Breakdown

Air System						Sensors			
	Cost Each	Qt.	Cost				Cost Each	Qt.	Cost
Solenoid Valves	$9.80	2	$19.60			Pressure Sensor	$14.95	1	$14.95
Lipo Battery for Pumps	$29.99	1	$29.99			Temp Sensor	$9.95	1	$9.95
Solenoid Valve Adapters (Set of 2)	$8.99	2	$17.98			Altimeter Sensor	$14.95	1	$14.95
⅛" NPT Female to ⅜" OD Female Tube Adapt	$15.99	1	$15.99			In-Line Pressure Sensor	$20.19	1	$20.19
3-Way Push-to-Connect Fittings	$7.91	1	$7.91			Voltage Sensor	$7.95	2	$15.90
9V Battery	$4.00	1	$4.00					Total Cost	$75.94
Balloon	$69.99	1	$69.99						
Air Cell	$36.91	1	$36.91						
Ripstop Nylon 40D	$8.95	7	$62.65			Electronics			
Ripstop Nylon 70D	$6.05	2	$12.10				Cost Each	Qt.	Cost
Pump	$52.09	1	$52.09			Qwiicc Cables	$7.95	1	$7.95
Main Tubing (uxcell)	$6.49	2	$12.98			16 AWG Wire Spool	$17.89	1	$17.89
Thread for nylon casing	$2.99	2	$5.98			Terminal Blocks	$9.99	1	$9.99
		Total Cost	$348.17			Arduino Shields	$12.88	1	$12.88
						USB A to Mini-B Arduino Cable	$1.49	1	$1.49
						Solderable Breadboard	$12.99	1	$12.99
Payload						9V Adapter for Arduino	$2.95	1	$2.95
	Cost Each	Qt.	Cost			XT60 lipo connectors	$6.99	1	$6.99
14" Cable Ties	$3.16	1	$3.16	MS		Transmitter/Receiver	$19.99	1	$19.99
4" Cable Ties	$4.04	1	$4.04	MS		Arduino/Microcontroller	$19.95	1	$19.95
Shell	$4.00	1	$4.00			GPS Module	$39.95	1	$39.95
Paracord	$9.99	1	$9.99			Pump ESC	$22.99	1	$22.99
Kevlar Tether (50 lbs, 200 ft)	$11.99	1	$11.99			Relay	$10.99	1	$10.99
Spring Scale	$10.11	1	$10.11					Total Cost	$187.00
		Total Cost	$43.29						
						Extra Unused Material			
Helium Connection							Cost		
	Cost Each	Qt.	Cost			Tubing we didn't use	$21.23		
1/2" Quick Connect Ball Valve	$9.99	1	$9.99			Transceiver (unused)	$17.99		
1/2" Quick Connect to 1/2" NPT Adapter	$9.98	1	$9.98			Extra 9V Battery	$3.51		
3"x2" Coupling PVC	$9.19	1	$9.19			Total	$42.73		
2"x1/2" Reducer Bushing PVC	$5.27	1	$5.27						
Vinyl tube 5/16" ID 1/2" OD (per ft)	$0.52	0.5	$0.26	Mansfield Supply (MS)					
PVC Cement 4oz	$4.92	1	$4.92	MS					
		Total Cost	$39.61						

Table 5.3.1 - Full Breakdown of Build Specifications and Cost

Air System

	Cost Each	Qt.	Cost
Solenoid Valves	$9.80	2	$19.60
LiPo Battery for Pumps	$29.99	1	$29.99
Solenoid Valve Adapters (Set of 2)	$8.99	2	$17.98
⅛" NPT Female to ⅜" OD Female Tube Adapter	$15.99	1	$15.99
3-Way Push-to-Connect Fittings	$7.91	1	$7.91

9V Battery	$4.00	1	$4.00
Balloon	$69.99	1	$69.99
Air Cell	$36.91	1	$36.91
Ripstop Nylon 40D	$8.95	7	$62.65
Ripstop Nylon 70D	$6.05	2	$12.10
Pump	$52.09	1	$52.09
Main Tubing (uxcell)	$6.49	2	$12.98
Thread for nylon casing	$2.99	2	$5.98
		Total Cost	$348.17

Table 5.3.2 -Air System Build Specifications and Cost

Sensors

	Cost Each	Qt.	Cost
Pressure Sensor	$14.95	1	$14.95
Temp Sensor	$9.95	1	$9.95
Altimeter Sensor	$14.95	1	$14.95
In-Line Pressure Sensor	$20.19	1	$20.19
Voltage Sensor	$7.95	2	$15.90
		Total Cost	$75.94

Table 5.3.3 - Sensors Build Specifications and Cost

Electronics

	Cost Each	Qt.	Cost
Qwiicc Cables	$7.95	1	$7.95
16 AWG Wire Spool	$17.89	1	$17.89
Terminal Blocks	$9.99	1	$9.99
Arduino Shields	$12.88	1	$12.88
USB A to Mini-B Arduino Cable	$1.49	1	$1.49
Solderable Breadboard	$12.99	1	$12.99
9V Adapter for Arduino	$2.95	1	$2.95
XT60 LiPo connectors	$6.99	1	$6.99
Transmitter/Receiver	$19.99	1	$19.99
Arduino/Microcontroller	$19.95	1	$19.95

GPS Module	$39.95	1	$39.95
Pump ESC	$22.99	1	$22.99
Relay	$10.99	1	$10.99
		Total Cost	$187.00

Table 5.3.4 - Electronics Build Specifications and Cost

Payload

	Cost Each	Qt.	Cost
14" Cable Ties	$3.16	1	$3.16
4" Cable Ties	$4.04	1	$4.04
Shell	$4.00	1	$4.00
Paracord	$9.99	1	$9.99
Kevlar Tether (50 lbs, 200 ft)	$11.99	1	$11.99
Spring Scale	$10.11	1	$10.11
		Total Cost	$43.29

Table 5.3.5 - Payload Build Specifications and Cost

Helium Connection

	Cost Each	Qt.	Cost
1/2" Quick Connect Ball Valve	$9.99	1	$9.99
1/2" Quick Connect to 1/2" NPT Adapter	$9.98	1	$9.98
3"x2" Coupling PVC	$9.19	1	$9.19
2"x1/2" Reducer Bushing PVC	$5.27	1	$5.27
Vinyl tube 5/16" ID 1/2" OD (per ft)	$0.52	0.5	$0.26
PVC Cement 4oz	$4.92	1	$4.92
		Total Cost	$39.61

Table 5.3.6 - Helium Connection Build Specifications and Cost

Extra Unused Material

	Cost

Tubing we didn't use	$21.23
Transceiver (unused)	$17.99
Extra 9V Battery	$3.51
Total	$42.73

Table 5.3.7 - Extra Unused Material Build Specifications and Cost