Team Gmork

Spring 2023

Comparing Carbon Dioxide Concentrations Pre, During, and Post Pandemic Years

&

Pre-Engineering Data for the Application of Mini Vacuum Pumps in Near Vacuum Environments

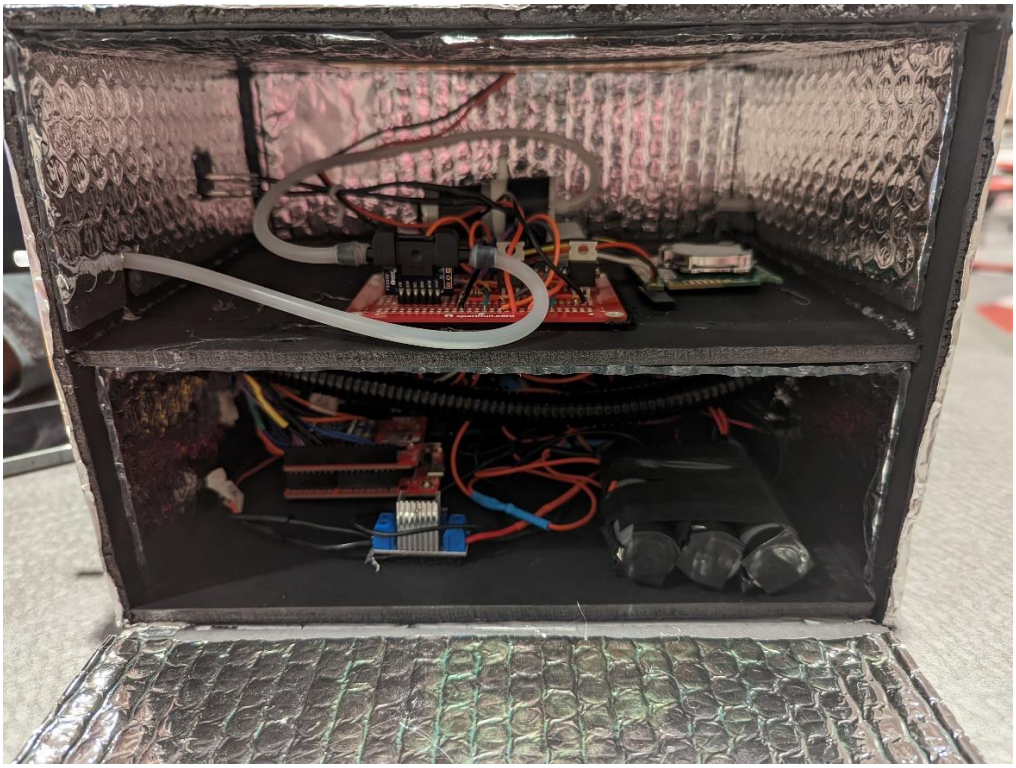
~Linda Karas & Adam Cobb~

Front Range Community College, Boulder County Campus
2190 Miller Drive, Longmont, CO 80501

lkaras@student.cccs.edu & acobb25@student.cccs.edu
Faculty Advisor: Anthony Riley - Anthony.Smith@frontrange.edu

Table of Contents

Figure 1 (Inside of DemoSat)



1.0 Introduction

The objective of our primary experiment is to collect CO2 data. With the information we obtain, our goal is to analyze the CO2 data received from our flight and then to compare our CO2 data to prior DemoSat collected CO2 data. Specifically, we were hoping to target the year before, during, and post pandemic (2019, 2020, and 2021) with the use of prior DemoSat teams that selected CO2 as the focus of their experiments. Unfortunately, the CO2 data from the 2020 team was absent from their final report. So, we will be focusing on the comparison of our collected CO2 data to the years pre and post covid years, 2019 and 2021 respectively. Our overall hope is to show that CO2 in the atmosphere has decreased from 2019, but to see stability would be more preferred over than increase if a decrease in data is not present.

In addition to the CO2 experiment, we also performed a secondary pre-engineering experiment. This experiment tested a mini vacuum pump and this was done to determine whether a vacuum pump would successfully function at near vacuum altitudes. Initially, we looked at measuring the microplastics in the atmosphere, however, the equipment and any research data was out of scope or not available. One of the biggest concerns we had was whether or not the mini vacuum pump would even function in the vacuum environment at the elevation the balloon takes the payloads, or if it would function for the duration of the flight. Being able to add this secondary experiment may not have fully benefited the CO2 experiment, but after testing and proving that the mini vacuum pump could function in the atmosphere environment we were going to be flying in, we were able to tie the mini vacuum pump in as a way to create a direct inflow of air to the CO2 sensor and as a real-time experiment for viable data for future teams should they want to use this component in their experiments.

2.0 Build Materials

- K30 FR Fast Response 10,000ppm CO2 Sensor
- Air Pump and Vacuum DC Motor 2.5[lpm]
- MCP9808 Interior Temperature Sensors x2
- Adafruit BMP390 Precision Barometric Pressure and Altimeter Sensor
- MPRLS Ported Pressure Sensor
- Silicone Tubing for Air Pump and Valves (3mm)
- Lexar 32 GB Micro SD Card
- Micro SD Shield
- SparkFun RedBoard Plus
- Male Pin Header Connectors
- SparkFun 5x15cm Heating Pad
- Adafruit 5x10cm Heating Pad
- FS2012-1100-NG Flow Volume Sensor
- LG M50LT 21700 5000mAh 14.4A Lithium Batteries x3
- N-Channel Power MOSFET x2
- 5A Constant Voltage and Constant Current Step-Down Power Module
- Step-Down Circuit Board Module x2
- Aluminum foil tape
- Reflective Insulation
- Foamboard
- 21700 Battery Charger
- SparkFun Solder-able Breadboard
- Stackable Shield Header

3.0 Purpose

3.1 Primary Experiment (CO2)

The CO2 Sensor: At the heart of this experiment was the K30 FR Fast Response 10,000ppm CO2 Sensor. This particular sensor was selected based on a couple key factors and their parameters: The CO2 spectrum this sensor is capable of detecting is 0-10,000 ppm with an accuracy of +/- 30 ppm. This was optimal due to the temperature of altitude range it functions in. Since the payload could potentially reach over 100,000 feet in altitude and the temperature could hit an extreme of -30 degrees Celsius, these key factors were very important to the decision of using this particular CO2 sensor as they were a high potential for mission failure if either did not function or work in the parameters required. The sensor could also record more data at a time at 0.5 seconds per data point.

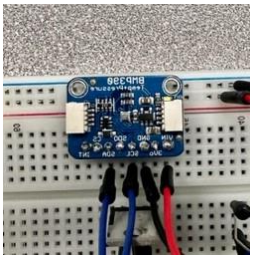
(Note: This sensor has a potential life expectancy of up to 15 years. If a future team is interested using this sensor, and would like to save ~$120.00, please contact Linda Karas for pick up-no shipping).
Data Sheet: S8 prod spec_ ny_1 (shopify.com)

3.2 Pre-Engineering Experiment (Mini Vacuum Pump)

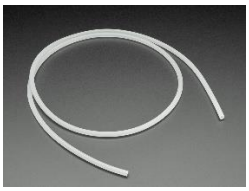
Mini Vacuum Pump: As a secondary pre-engineering experiment, we also tested the Air Pump and Vacuum DC Motor. Initially, for our team experiment, we were going to test for microplastics in the atmosphere, but there was no relatable prior experimental data for us to look to. In addition, we were unsure if we would even be able to test the mini vacuum pump prior to launch had we found any prior (conclusive) flight data. That was too big of a risk to take given the value that prior data was in determining the fate of the entire experiment. As a result, a compromise was made, and we decided to provide pre-engineering data on this pump and include it as a part of our payload for our primary experiment. We were able to test both on ground, and then, successfully as a part of the payload.

4.0 Supporting Sensors and Parts

Temperature Sensors: The MCP9808 Digital Temperature Sensor is the digital sensor that was used to read the internal temperatures of the payload. There were two in the payload, one in the top level, and one in the bottom level. These talked directly with the heaters to turn them on and off to maintain a temperature above 11 degrees Celsius.

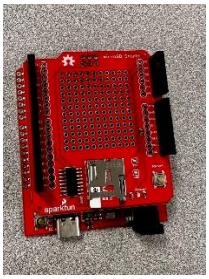


Altimeter & Pressure Sensor: The Adafruit BMP390 Precision Barometric Pressure and Altimeter Sensor was used to record the barometric pressure and the altitude of the payload as it ascended and descended. Recording the pressure was data to be used along with the vacuum and the altimeter is important so we can document the maximum altitude of the payload, as well as correlate the CO2 data to the altitude. It also had another temperature sensor built into it. The code only needed the local pressure inputted before flight to have an accurate reading.



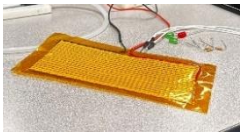
Silicone Tubing: Silicone Tubing for Air Pumps and Valves (3mm) was used to connect the vacuum pump was connected to the CO2 sensor on one end. The other end was used as suction intake to pass air directly over the CO2 sensor.

Micro SD Card: The Lexar 32 GB Micro SD Card was used to collect and store the data from all sensors on board the payload.



Arduino Board: Micro SD Shield seated atop the SparkFun RedBoard Plus with additional Male Pin Header Connectors. This combination of parts was the main computer component for the experiment. All code, sensors, the micro-SD card, heater, and mini vacuum pump were integrated into this stacked Arduino. We were able to solder the bottom level temperature sensors and ported pressure sensor onto the prototype board of the SD Shield to save room and space.

Lithium Batteries: LG M50LT 21700 5000mAh 14.4A. The payload consisted of 3 of these batteries combined into a series. Their nominal voltage per battery was 3.6v. And the maximum voltage per battery was 4.2v. Discharge cut-off voltage was 2.5v per battery. Individual capacity was rated at 5000mAh. However, during charging, a measured 4500mAh was consistently being recorded. The combined maximum voltage for flight was 12.6v. The combined nominal voltage for flight was 10.8v. The combined capacity for the flight was 5000mAh. A combined discharge cut-off voltage of 7.5v would be expected. Approximate dimensions are: 21.10mm x 70.20mm. Approximate individual weight: 67.8g. These batteries are also rated to 14.4 amps of maximum continuous discharge.
Data Sheet: PRODUCT SPECIFICATION (shopify.com)



Heaters: The SparkFun 5x15cm Heating Pad and Adafruit 5x10cm Heating Pad were used to provide additional heat. Though the sensors we used in this experiment are performance rated for individual maximum low temperatures that were in range of the flight, including a heating pad

provided a fail-safe in the event a sensor failed due to temperature. The heating pad provided an (approximate) additional 65 degrees Celsius to the internal compartment of the payload.

Flow Volume Sensor: The FS2012-1100-NG Flow Volume Sensor is the intermediary component between the mini vacuum pump and the CO2 sensor. This sensor registered the air volume that was passing through the tubing.
Data Sheet: FS2012 Datasheet (renesas.com)

Power Supply Regulator: UMLIFE 12PCS LM317 DC-DC Converter Buck Step-Down Circuit Board Module 4.2-40V to 1.2-37V Linear Regulator Adjustable Voltage Regulator Power Supply. Two of these were used to convert to the required voltage. 11v for the pump, and 7v for the CO2 sensor.

Power Step-Down Module: DUTTY 5A Constant Voltage and Constant Current Step-Down Power Module with Voltage and Current Power Display. We used one to power the heaters as the heaters drew about 800mAh at 8v each.

SparkFun Solder-able Breadboard: We used this BreadBoard to run the flow sensor, two MOSFETs, BMP390, one MCP9808 and the heaters.

5.0 Budget

5.1 Build Cost

Item	Cost
CO2 K30 FR Sensor	$120
BMP390 temp and altitude sensor	$14.99

MCP9808 Temp sensor 1&2 x2	$17.28
MPRLS Ported Pressure Sensor	$14.95
Mini Vacuum Pump	$7.50
Silicone tubing	$2.50
Flow Volume Sensor	$0
Micro SD Shield	$18.50
32GB Micro Sd card	$5.33
Male Pin Header Connector	$10.99
SparkFun RedBoard Plus	$21.50
Heating Pad 1 5x15cm	$5.50
Heating Pad 2 - 10cm x 5cm	$10.49
SparkFun Solder-able Breadboard	$5.50
Batteries – LG M50LT 21700 5000mAh 14.4A Battery x3	$23.07
5A Constant Voltage and Constant Current Step-Down Power Module	$11.99
N-Channel Power MOSFET x2	$1.60
Step-Down Circuit Board Module	$2.14
Stackable Shield Header	$15.99
Aluminum foil tape	$4.97
Insulation	$9.97
Foamboard	$9.99
21700 battery charger	$32.99
Build Total	**$367.74**

5.2 Extra Supplies

Extra Supplies	**Cost**
Miniature 5V Cooling Fan x2	$5.90
SparkFun Digital Temperature Sensor Breakout - TMP102	$5.50
SparkFun Solder-able Breadboard	$5.50
Micro SD card 32GB x2	$10.66
4 Units Pack 9 Volt	$9.98
T-Connector For Silicone Tubing	$4.50
6V Air Valve	$5.90
Vacuum pump	$7.50
21700 Extra batteries x7	$53.82
MOSFET extra x8	$6.39
Extra Supplies Cost Total	**$115.65**

5.3 Grand Total Cost

Grand total	**$483.39**

5.4 Total Build Weight

Device	**Mass[g]**
Arduino and microSD shield	35
Perma-proto ½ breadboard	12
Mini Pump [2.5 Ipm]	61
Flow Volume and 1m silicone tubing	22
BMP390 Temp+Pressure	3
5x10 heating pad	3
5x15 heating pad	8
5A Power supply LED display	31
1.5A Power supply – Pump	10
1.5A Power Supply – CO2	10
3.6v battery x3	204
CO2 Sensor	15
Pressure sensor	3
Temp and temp 2	8
Case and build material	435
Total Mass[g]	**860[g]**

6.0 Launch and Recovery

The morning of April 1st, 2023 was a very cold morning. With temperatures in the 20-30s Degrees Fahrenheit. Our payload came in at 860[g]. 60[g] over the weight limit but we were still allowed to fly. We were scheduled to launch on the second balloon which was the 1500[g] balloon. Our payload was put on the bottom of the string because we had the heaviest payload.

Before flight, we had to program the Arduino one last time to record the local air pressure which was 30.08inHg for accurate altitude measurements. This pressure was coded into our Arduino and uploaded one last time. Along with some last few checks to make sure everything was turning on and working. The payload door was closed and taped up one last time.

Our payload had two on/off switches. One for all the sensors and Arduino. The other for the pump. This was so we could save some battery life by only turning on the heaters and sensors to initiate the heating element and prepare the conditions inside for the flight. As we walked outside with our payload in hand 15 minutes before launch, we turned on the heaters. To do this we flipped the switch on and off three times to ensure everything turned on appropriately as we didn't have any indicator lights on the outside to know if we were gathering data or not.

In 1/100 chance the CO2 sensor would not turn on and start recording data. So, by turning the payload on and off three times this decreased the odds that our payload would not be recording data during flight.

Standing outside and waiting for the liftoff moment a countdown initiated. Quickly we flipped the pump switch to the on position and put tape over the switch to keep it in the on position. Three seconds later the balloon lifted off with our payload.

The balloon unfortunately only managed to get to 63000 feet before it popped. We were expecting at least 10000 feet. The Payload landed in a field where a rancher retrieved out payloads for us and dropped them off next to a road for us to retrieve. Upon walking up to the payload, the sweet sound of the pump still running was music to our ears as at least the payload survived the drop! Opening the payload in the car and loading the MicroSD card into a computer we found that we collected all data and there were zero interruptions.

Figure 2 (Payload still on flight string after landing)

7.0 Results

7.1 CO2 Results

After reviewing the data provided by prior DemoSat teams for the timeframe we are interested in comparing our CO2 data to, it's been determined that the results and data those teams collected were either not sufficiently documented or, unfortunately, not included at all. As a result, the CO2 data that we can and will compare ours to is that from the Mauna Loa Observatory in Hawaii partnered with Scripps in San Diego, CA. Moving forward, the data we will be comparing to will be from the historical data for the averaged atmospheric CO2 levels from the years 2019, 2020, and 2021.
https://scrippsco2.ucsd.edu/data/atmospheric_co2/mlo.html

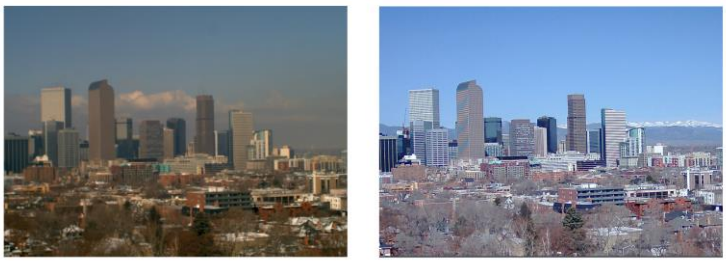
Our CO2 experiment payload was included in the DemoSat launch that was held on April 1, 2023 at Deer Trail Elementary in Deer Trail, CO. The elevation of Deer Trail is 5,190 feet, and the balloon our payload was attached to burst at 63,000 feet. The final weight of our payload was 860g. All systems worked without incident to 52,000 feet. At that altitude, the altimeter sensor failed, but the data received up until the altitude was collected with minimal errors reported.
The CO2 levels that were collected from our experiment and the comparison data are as follows:

Comparison Data-Mauna Loa Observatory				Team Gmork
2019	2020	2021		2023
Avg: 411 ppm	Avg. 414 ppm	Avg. 416 ppm		Avg. 413 ppm

This data does not support any significant downturn in the CO2 levels in our atmosphere. However, the average is steady and there is no major negative decline in our CO2 air quality.

The Colorado Health Institute did put out a photo comparison of the air quality after one month of the initial lock-down Denver went through. It speaks volumes:

Figure 1. Images From Colorado Department of Public Health and Environment Streaming Webcam at 10 a.m. March 2, 2020 (Pre-Stay-at-Home) and at 10 a.m. April 6, 2020 (During Stay-at-Home)[1]



Source:
https://www.coloradohealthinstitute.org/sites/default/files/file_attachments/Catalyst%20or%20Challenge_0.pdf

7.2 Vacuum Pump Results

The data that we got back from our flow sensor was amazing. It showed an exponential curve downward from 4.9 liters per minute, all the way down to 0.194[lpm] at 63000 feet as seen in **Table 17 Flow vs. Altitude (Ascent & Descent)** and **Table 18 Flow vs. PSI (Ascent & Descent)**. This was some remarkable and breakthrough data as this directly proved the usage of a vacuum pump in high altitude environments.

The data that is available online for the performance of a vacuum pump in near vacuum environments is limited to almost no existence. We have not found any other reports that directly reported on the

performance of a vacuum pump in near vacuum environments to the accuracy that we recorded with our calibrated flow sensor. We were over-volting our pump that was rated for a 5 volt nominal voltage all the way up to 11 volts to achieve a higher liters per minute value at ground level. The pump was only supposed to be rated for 2.5[lpm] at 5 volts of power. We managed to get it up to 4.9[lpm] safely for the flight without overheating. Also, noting that the pump is also not rated to run continuously, but rather, 10 seconds on and off 5 seconds in cycles. It ran continuously the entire flight and recorded the flow rate every 0.5 seconds. 9,283 data points per sensor were recorded.

Unfortunately, our altitude sensor stopped working at 52,000 feet. So, rather than 10,000 feet of missed data, we included the results of comparing the vacuum pump flow vs. atmospheric PSI - which recorded data all the way up to 63,000 feet. Regarding the use for this vacuum pump for the potential to use it to collect atmospheric microplastics, a slightly larger pump is recommended if the weight limits allowed it. Our vacuum pump came in at 68 grams, while a bigger upgraded pump could potentially pump at 12[lpm] and weigh about 200 grams. Comparing our Pump data to Table 14 Flow vs. PSI (Ascent), you can clearly see a linear downward line as the atmospheric pressure decreases. A linear trendline from the **Table 20 Trendline for Flow vs. PSI (Ascent)** gives an equation of **Y=0.425*x + -0.247.** A calculated 0[lpm] flow rate would predict the vacuum pump to reach at least 73,000 feet or 0.58PSI before stalling. Although we would expect the flow rate curve to become exponential around the stall period of 73,000 feet and possibly be able to still pull in airflow at 100000 feet. Using an altitude conversion chart helps in getting this data included here, **Table 19 Altitude Conversion**.

The ability to collect and analyze microplastics using a non-invasive method, such as a vacuum pump, is possible and feasible. Using a flow volume meter to measure the airflow going in would definitely be required for this kind of experiment since you would want to measure the total volume coming into your system and running through your particulate filter so you could measure at different altitudes with different filters in your system and keep the total volume of air collected onto the filter the same. This could easily be done with one pump and multiple chambers with valves to switch the airflow to each chamber as altitude increases. Using an integral you could calculate the volume and switch from chamber to chamber as you meet the volume of air collected onto each filter and recording the slice of air/altitude that the filter collected. Bringing the filters into a spectrometer lab on ground level with minimum data interference, you could safely analyze the filters for the different types of plastics on them. The recommended filters to use would be an aluminum oxide filter.

8.0 Conclusions and Lessons Learned

From the CO2 data that we collected and the comparison sets from the Mauna Loa Observatory, it is clear that the pandemic did not curb human behavior and the dependency we have on vehicles and other CO2 producing needs. However, the CO2 averages appear to be at a consistent and steady level from 2019, 2020, 2021, and 2023, so not all is lost. In order to relaunch the payload all we had to do was recharge the batteries and replace the altitude sensor and hope the balloon makes it to 100000 feet this time. The more pump data and CO2 data would be great at higher altitudes.

9.0 Message to Next Year

Decide on a team project much sooner rather than later. Seriously. In doing this, you are allowing the team more time work on the project, you are also giving your team more time to pivot should the original idea not work out. This also gives time to make adjustments should teammates not stay onboard for the duration of the project. Communicate and work with your teammates. Share responsibilities and duties. Ask for help when needed. Appreciate your mentors and the advice, knowledge, and experience they have to offer. Realize that whatever experiment the team decides on, it will always take longer than you believe it will. Time will go by faster than you think-do not procrastinate.

Start working on the team project right away during winter break. Keep in touch with your teammates as much as possible so none get lost along the way. We ended up losing three of our teammates in January and we managed to complete the project with lots of stress along the way. You want five team members to help you complete this project and expect lots of work if you come down to less than five. If you can, before winter break starts, come up with an entire schedule of your project with dates and deadlines.

7.11 CO2 Graphed Results

Table 1 CO2 vs. Altitude (Ascent)

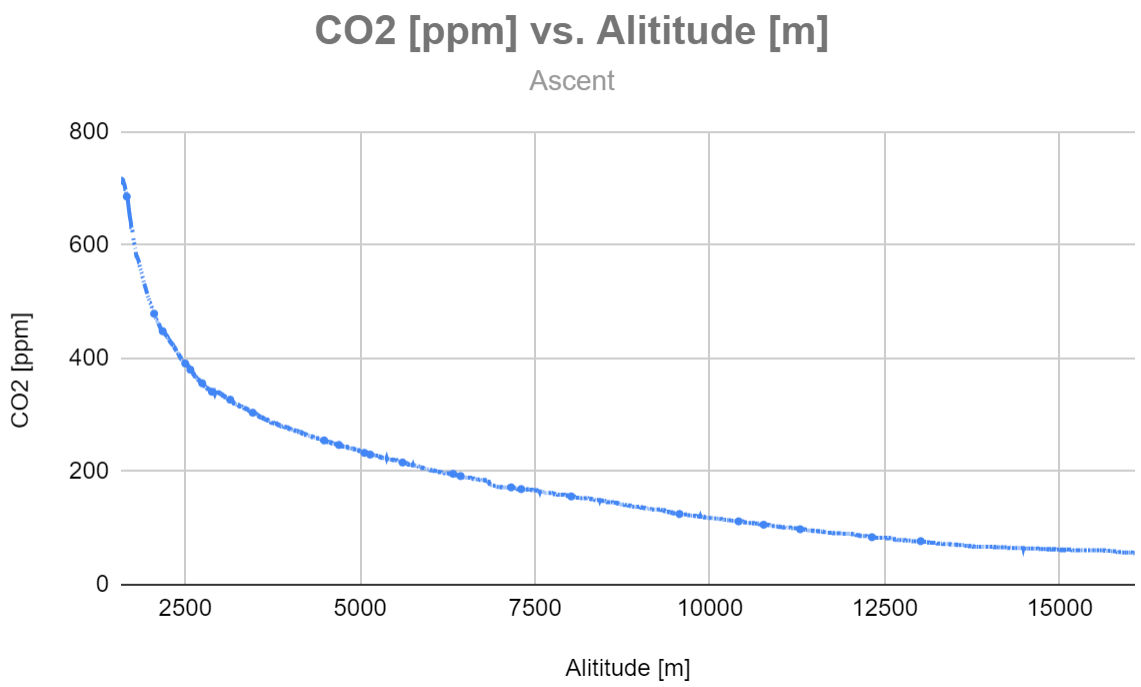


Table 2 CO2 vs. PSI (Ascent)

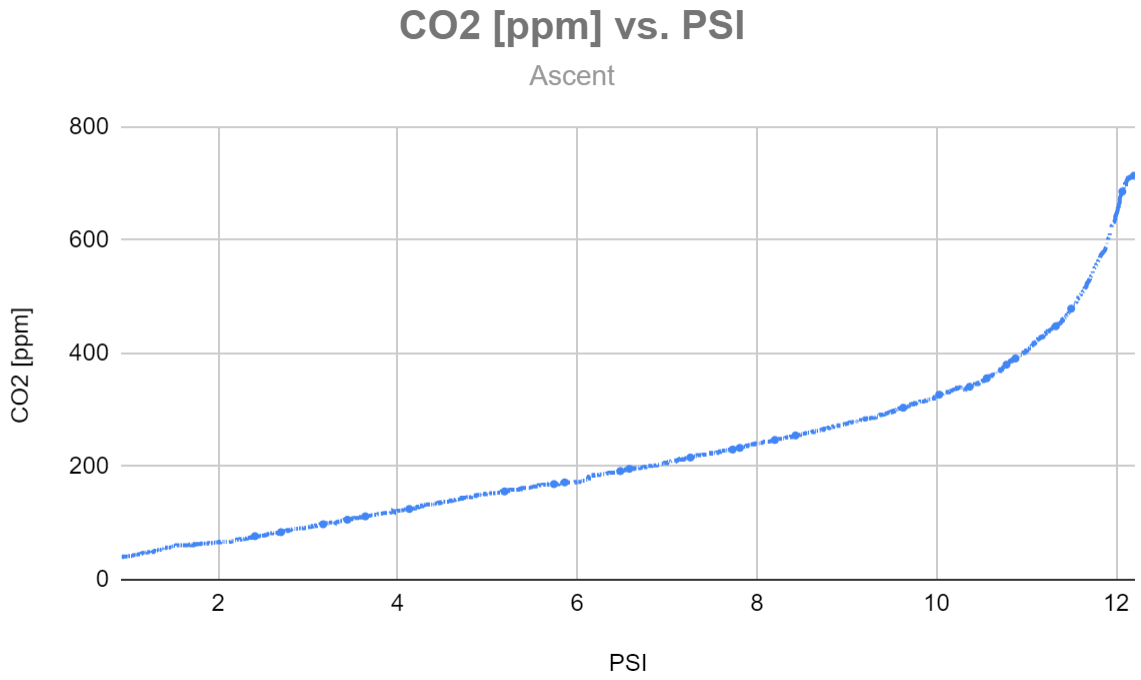


Table 3 CO2 vs Altitude (Descent)

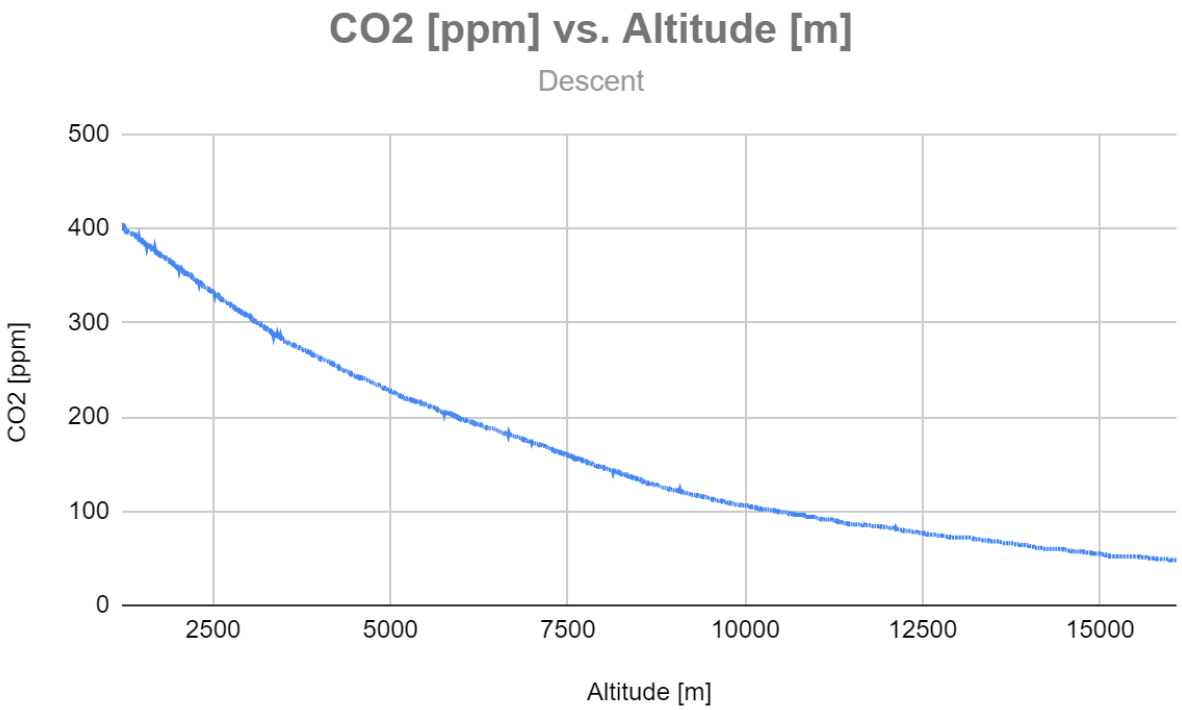


Table 4 CO2 vs. PSI (Descent)

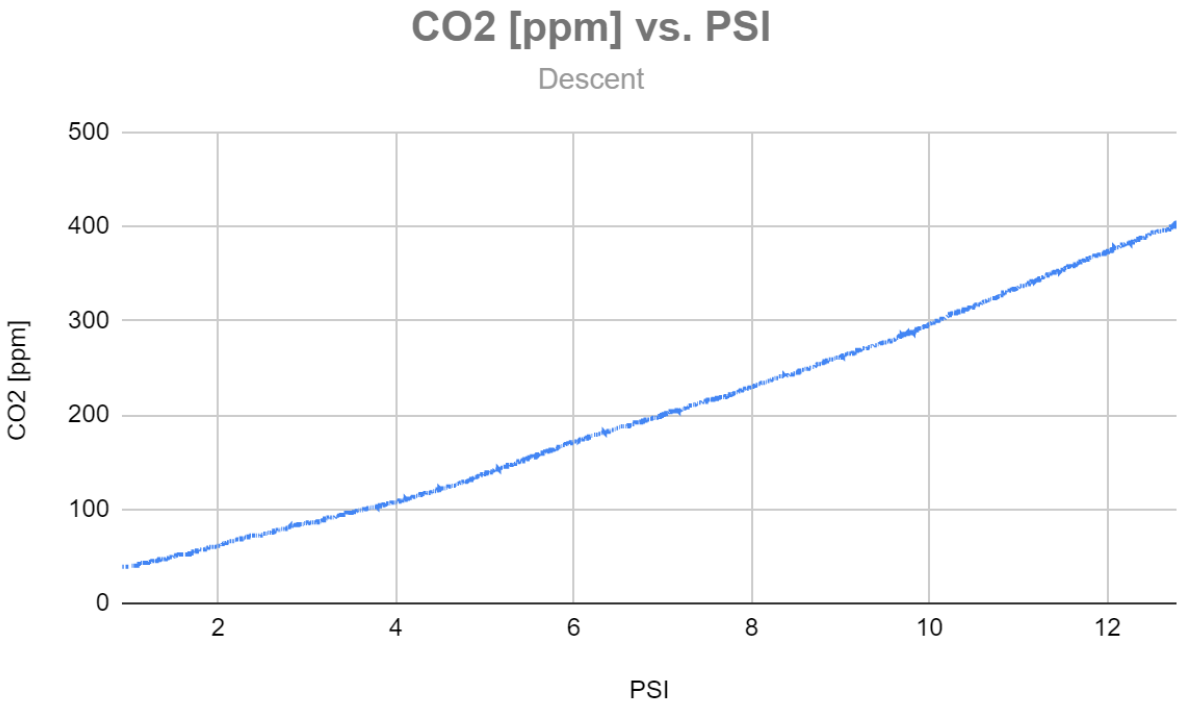


Table 5 CO2 vs. Altitude (Ascent & Descent)

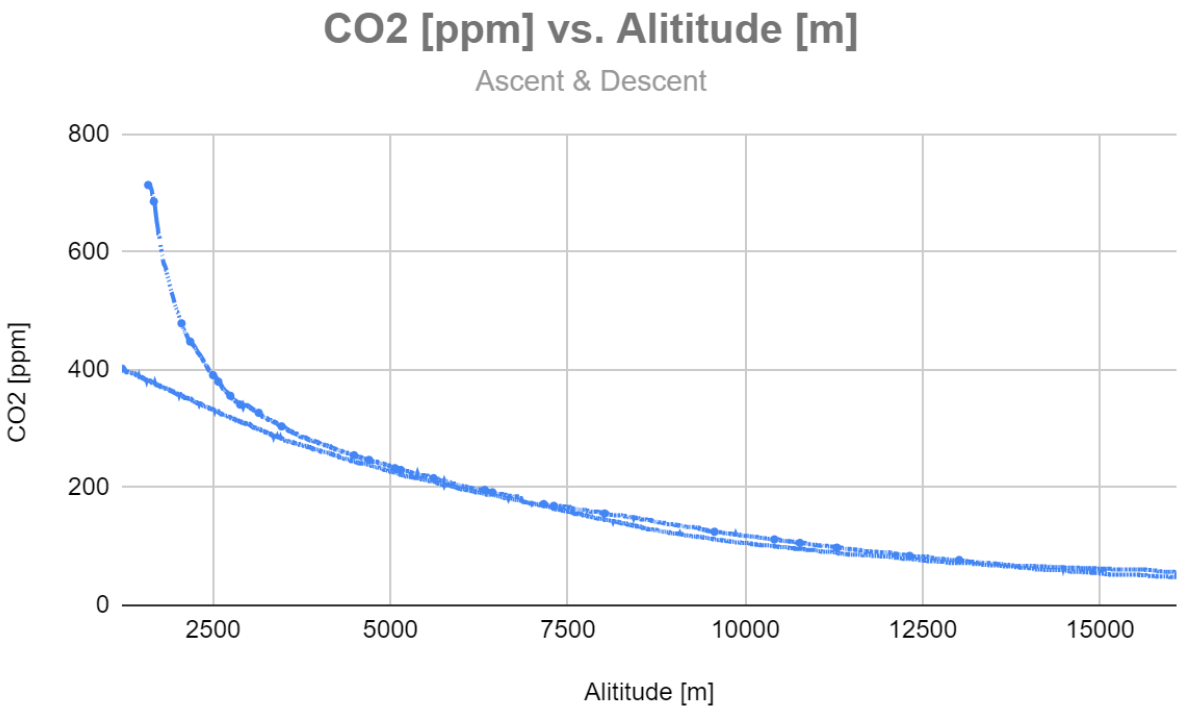
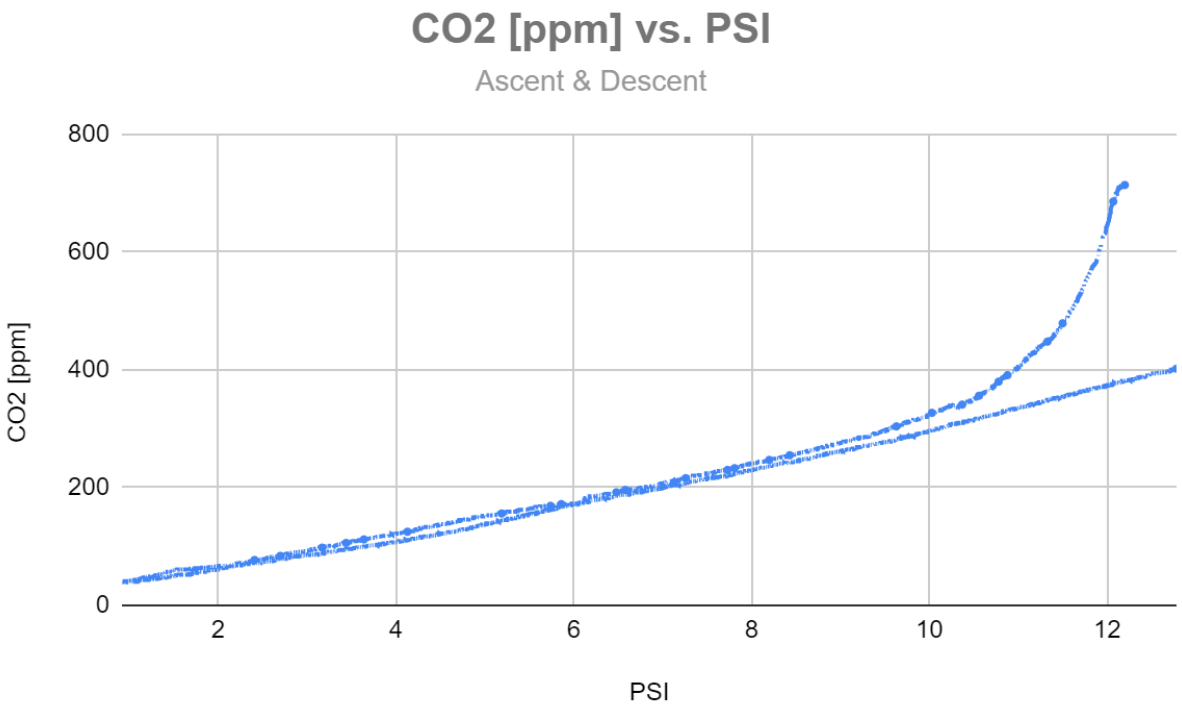


Table 6 CO2 vs PSI (Ascent & Descent)



7.12 CO2 Corrected for Altitude Graphed Results

Table 7 Corrected CO2 vs. Altitude (Ascent)

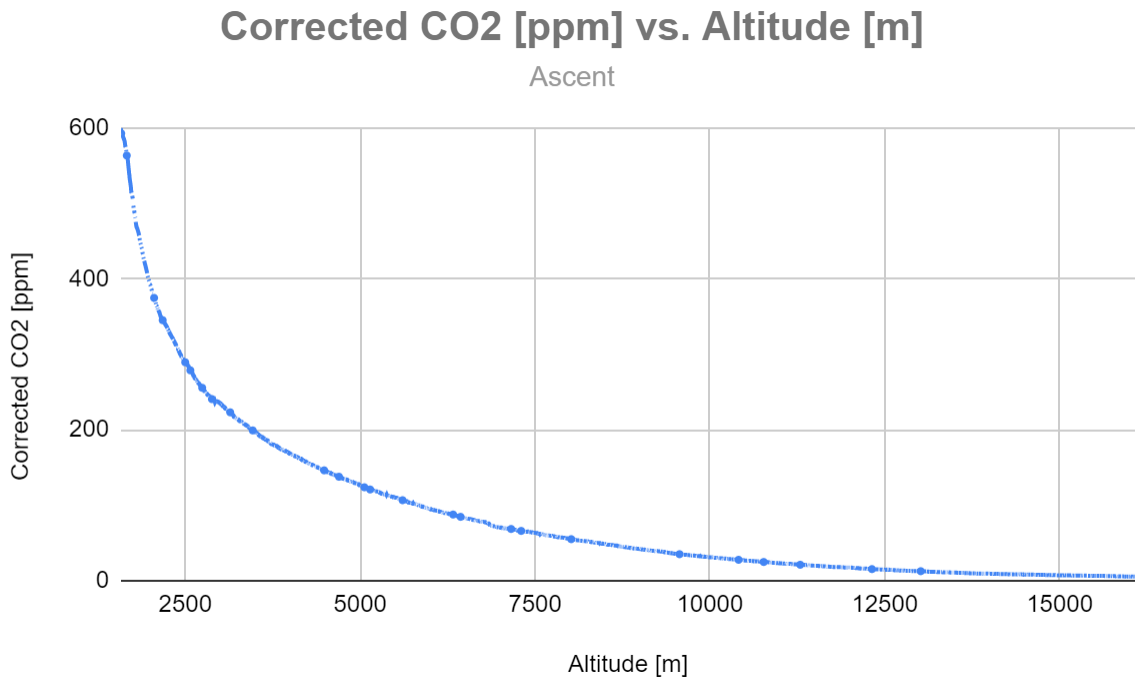


Table 8 Corrected CO2 vs. PSI (Ascent)

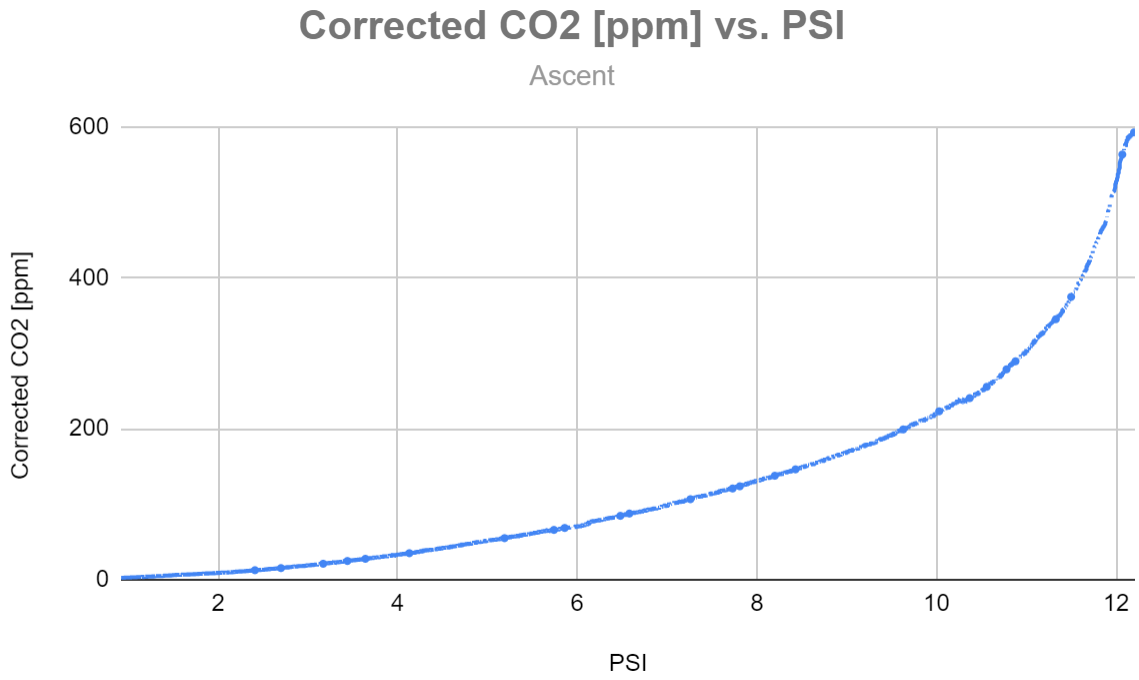


Table 9 Corrected CO2 vs. Altitude (Descent)

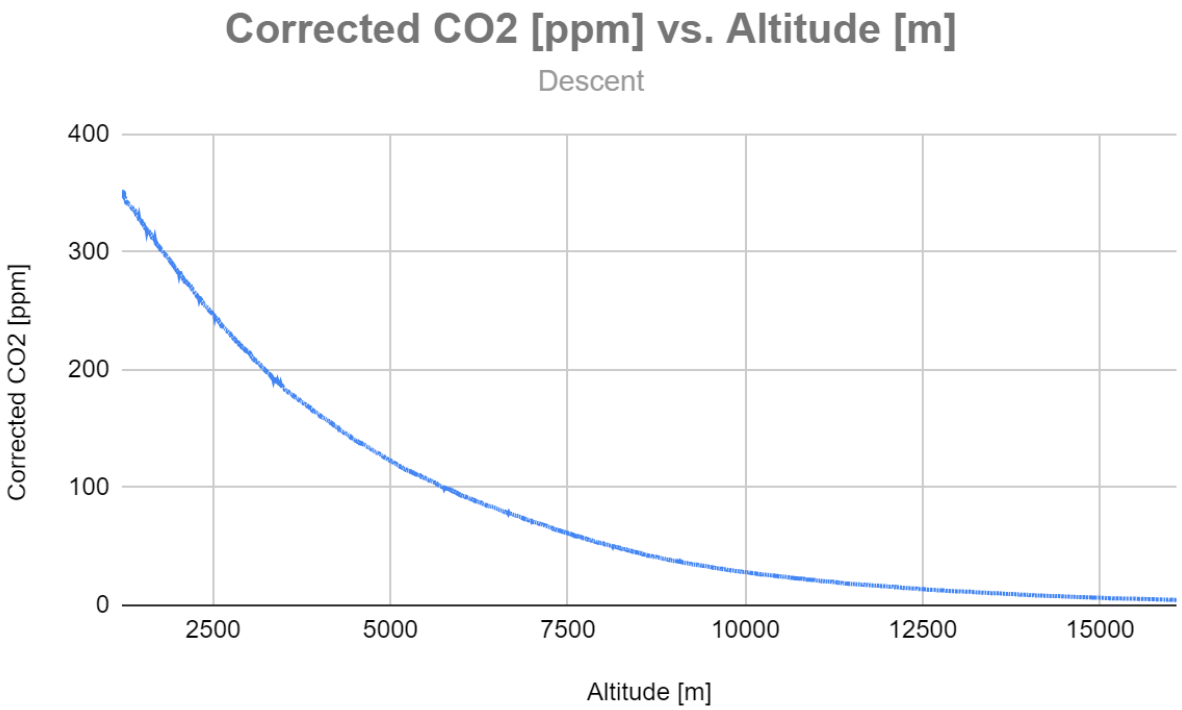


Table 10 Corrected CO2 vs. PSI (Descent)

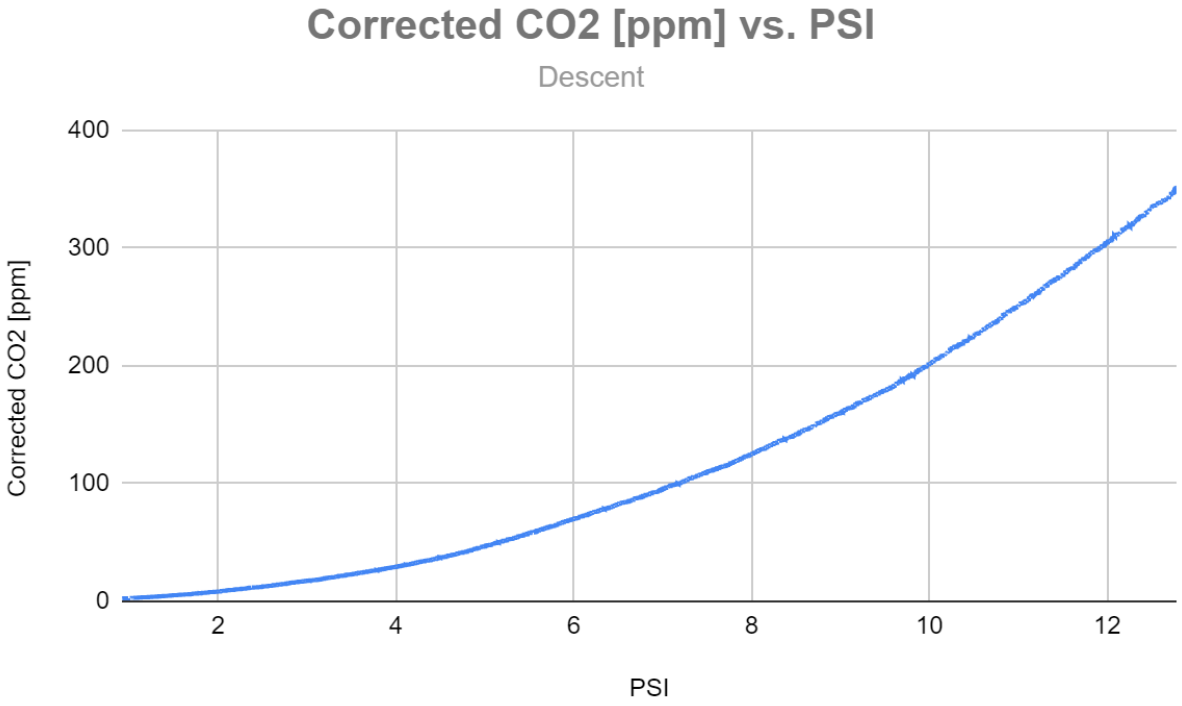


Table 11 Corrected CO2 vs. Altitude (Ascent & Descent)

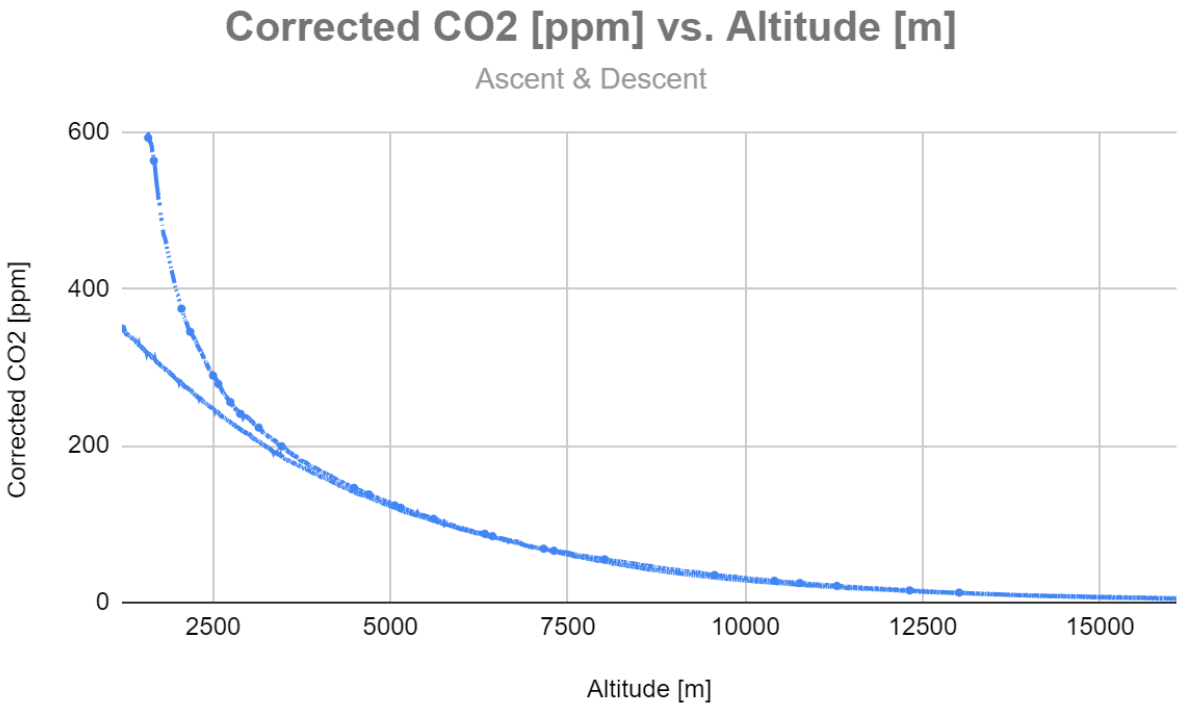
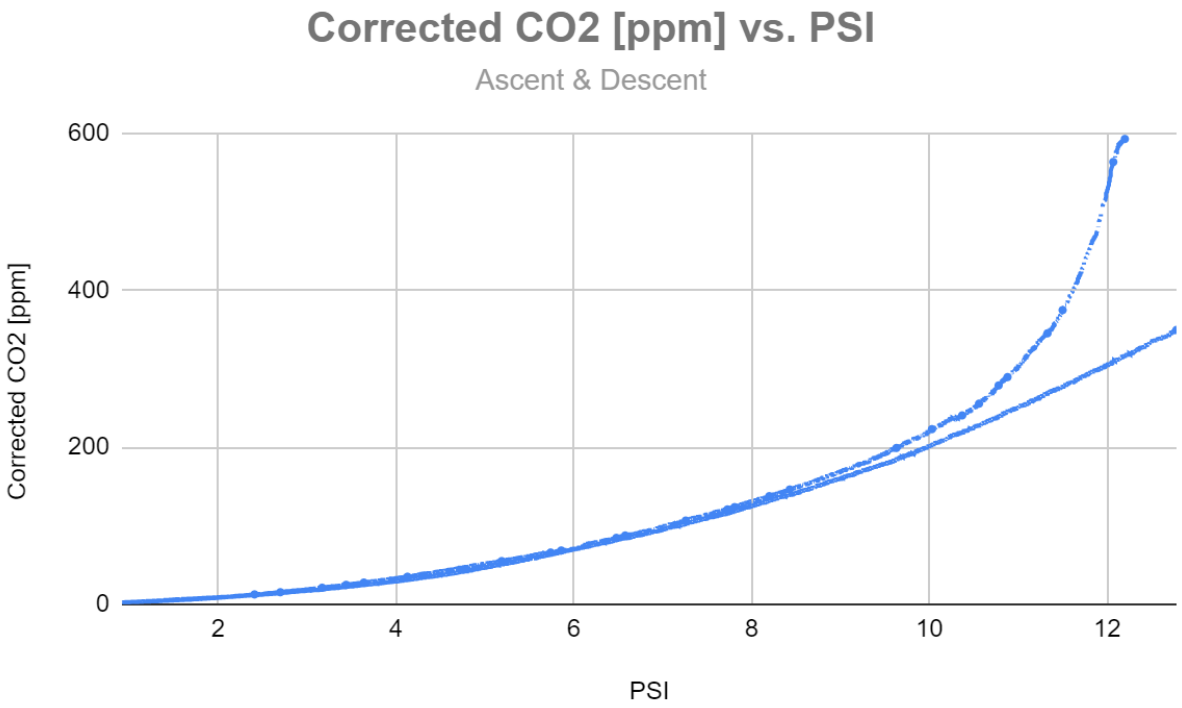


Table 12 Corrected CO2 vs. PSI (Ascent & Descent)



7.21 Vacuum Pump Graphed Results

Table 13 Flow vs. Altitude (Ascent)

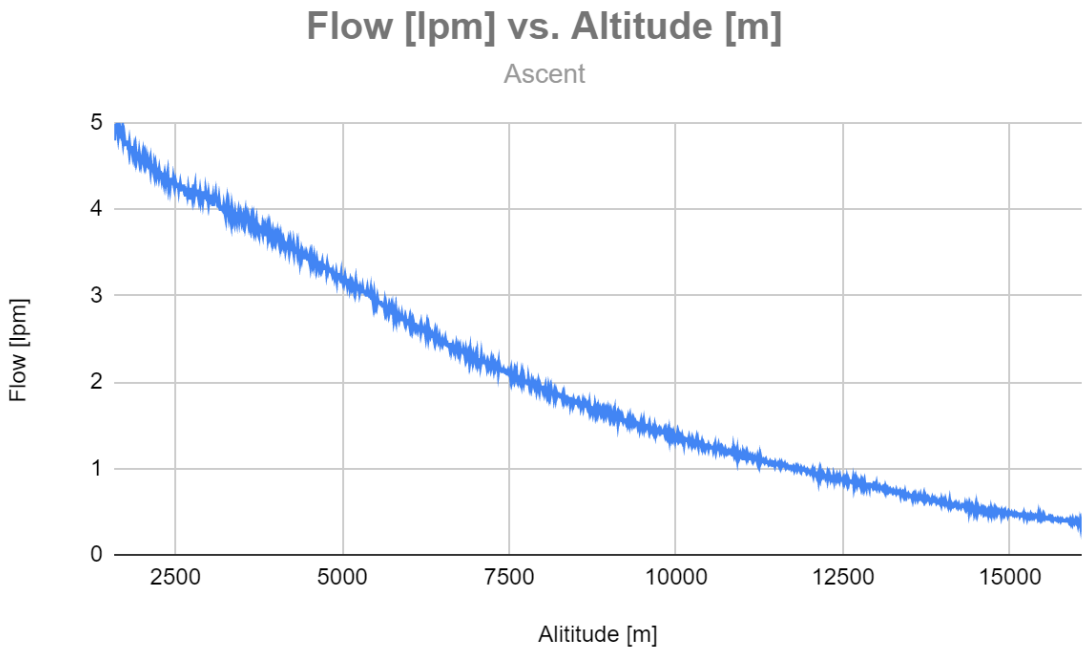


Table 14 Flow vs. PSI (Ascent)

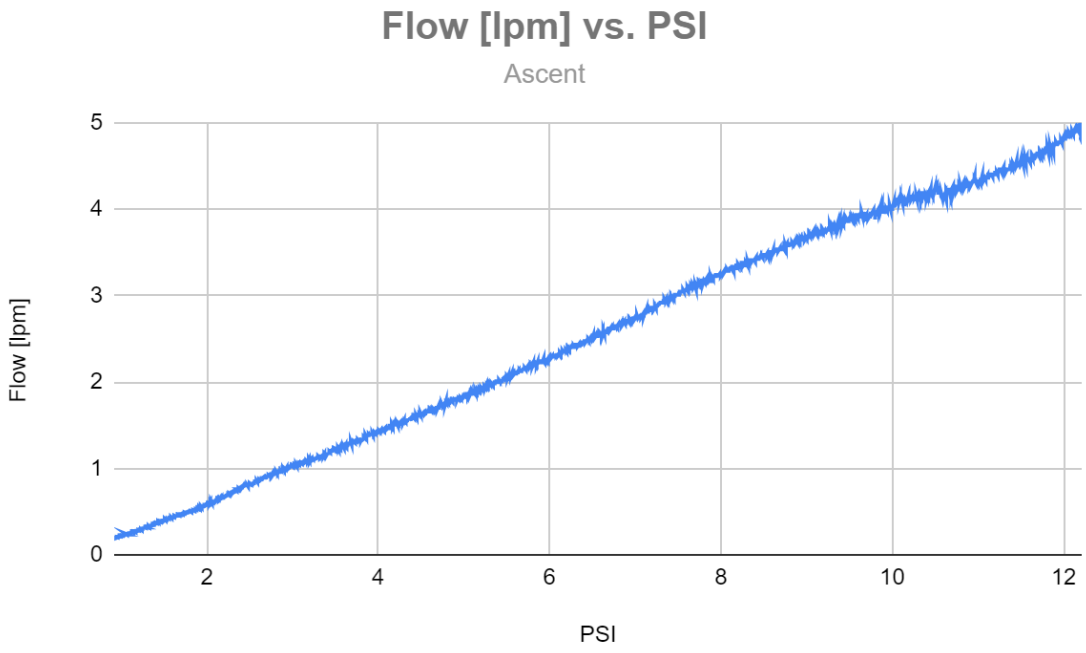


Table 15 Flow vs. Altitude (Descent)

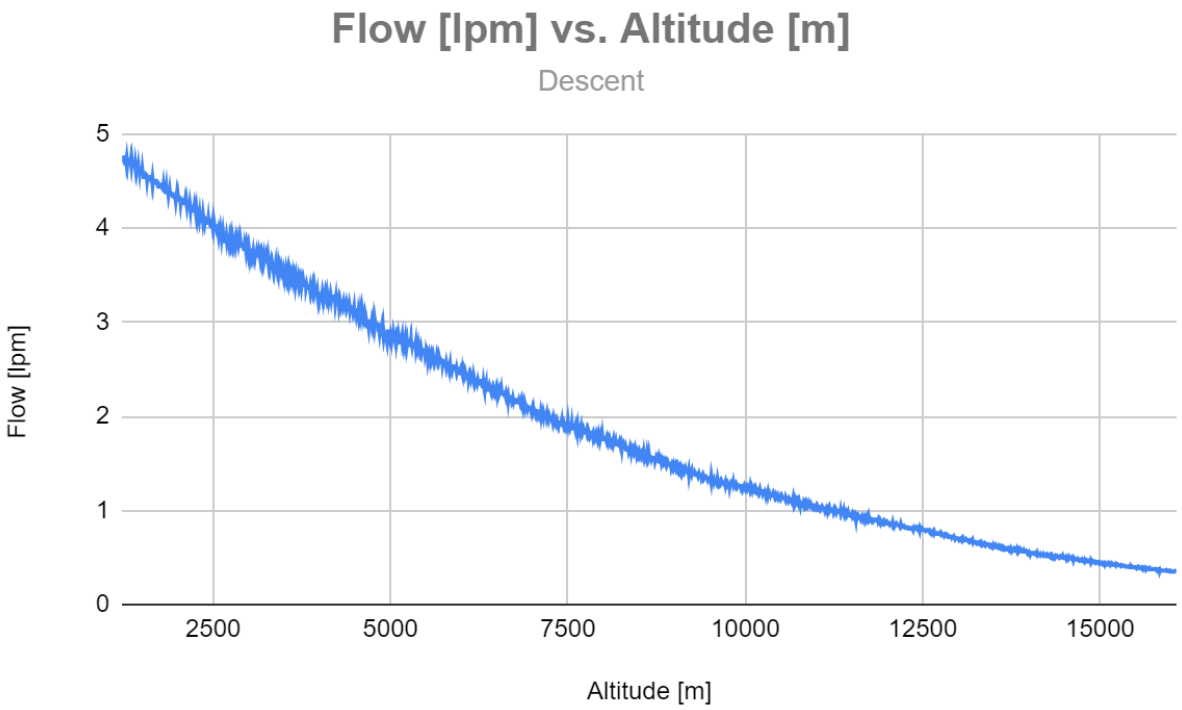


Table 16 Flow vs. PSI (Descent)

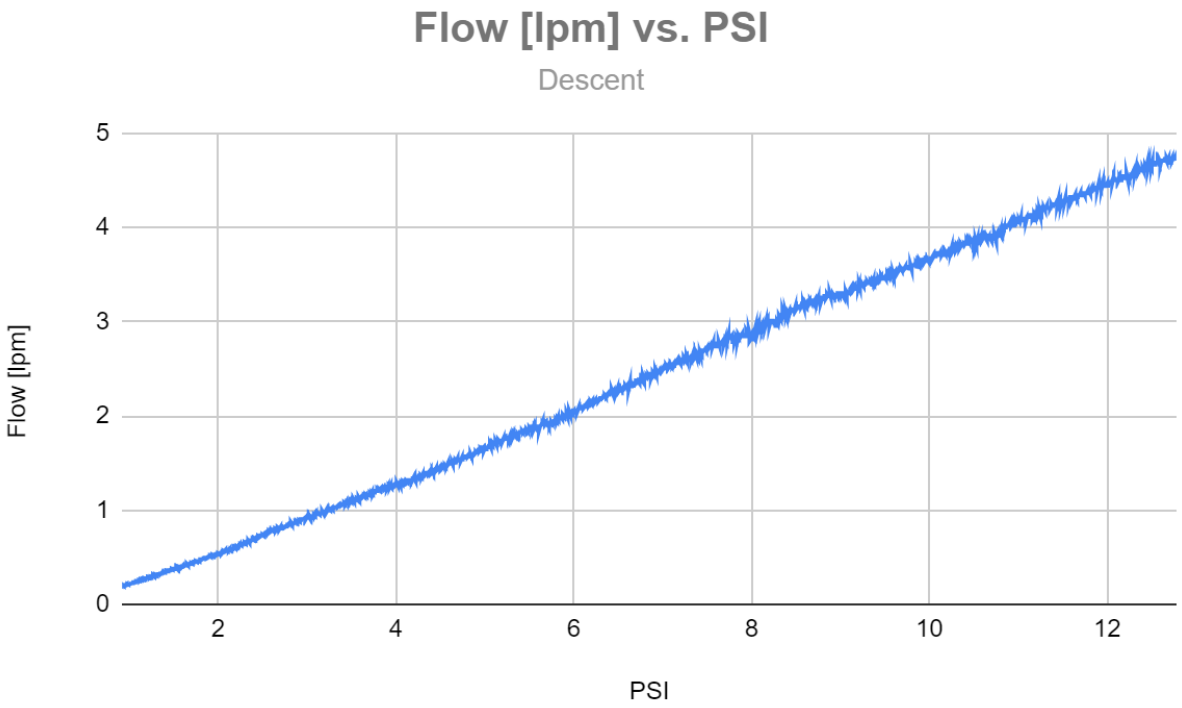


Table 17 Flow vs. Altitude (Ascent & Descent)

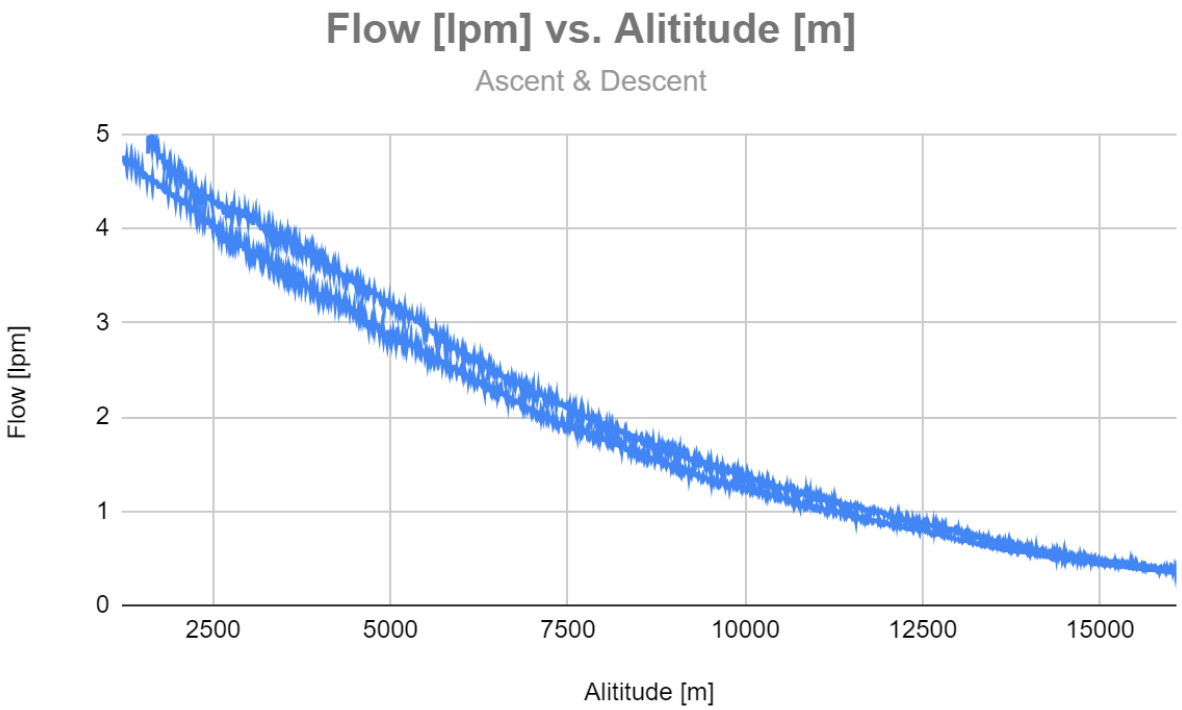


Table 18 Flow vs. PSI (Ascent & Descent)

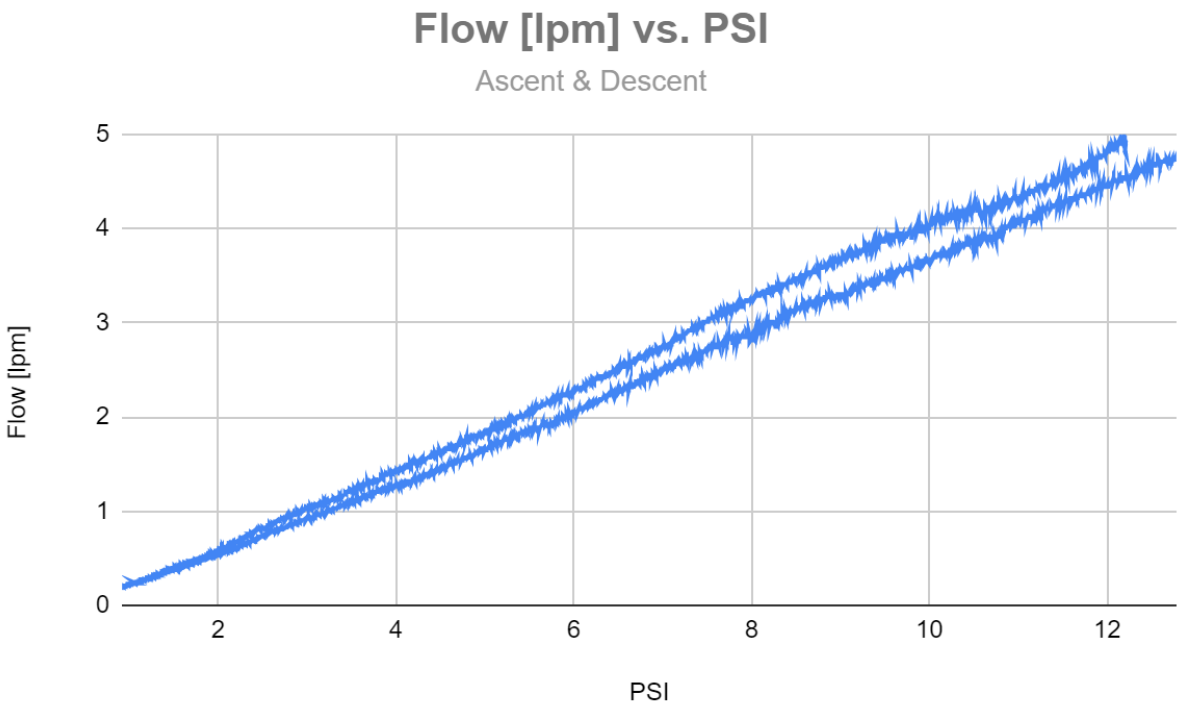
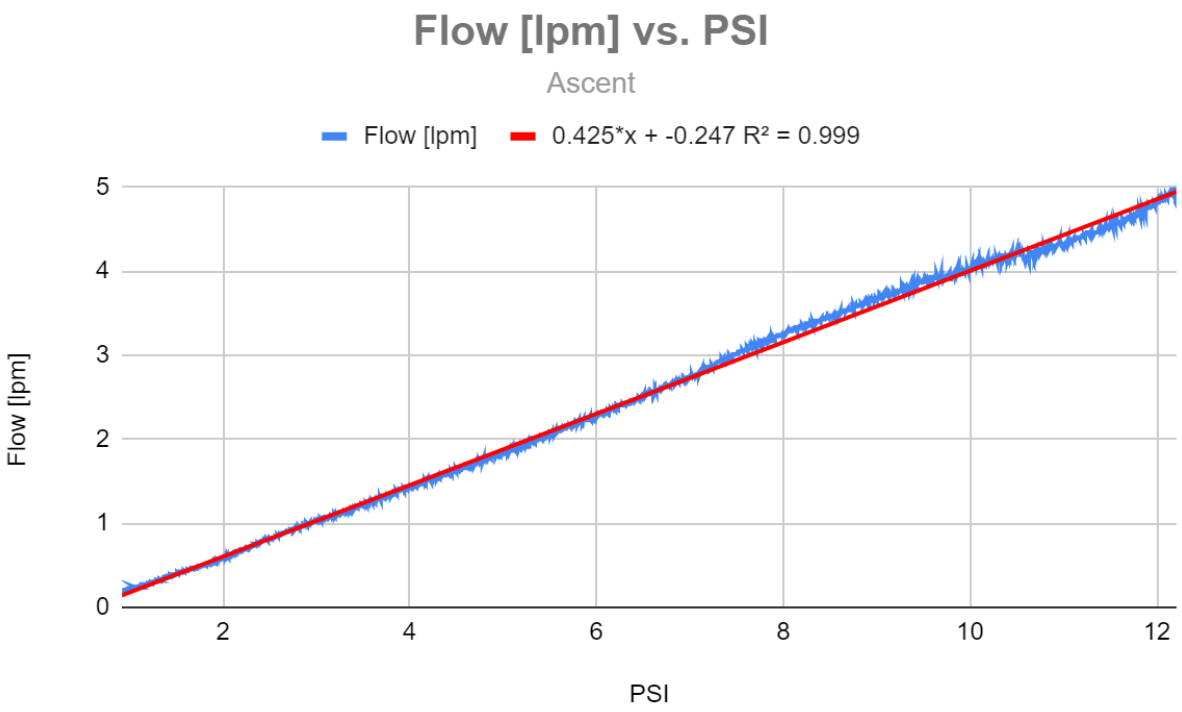


Table 19 Altitude Conversion Ft	m	mbar	psi	mmHg	inHg
0	0	1013.25	14.696	760	29.921
50	15.24	1011.42	14.669	758.63	29.867
100	30.48	1009.59	14.643	757.26	29.813
200	60.96	1005.95	14.59	754.52	29.706
300	91.44	1002.31	14.537	751.8	29.598
400	121.92	998.689	14.485	749.08	29.491
500	152.4	995.075	14.432	746.37	29.385
600	182.88	991.472	14.38	743.67	29.278
700	213.36	987.88	14.328	740.97	29.172
800	243.84	984.298	14.276	738.28	29.066
900	274.32	980.727	14.224	735.61	28.961
1000	304.8	977.166	14.173	732.93	28.856
2000	609.6	942.129	13.664	706.65	27.821
3000	914.4	908.117	13.171	681.14	26.817
4000	1219.2	875.105	12.692	656.38	25.842
5000	1524	843.073	12.228	632.36	24.896
6000	1828.8	811.996	11.777	609.05	23.978
7000	2133.6	781.854	11.34	586.44	23.088
8000	2438.4	752.624	10.916	564.51	22.225
9000	2743.2	724.285	10.505	543.26	21.388
10000	3048	696.817	10.106	522.66	20.577
15000	4572	571.82	8.2935	428.9	16.886
20000	6096	465.633	6.7534	349.25	13.75
25000	7620	376.009	5.4536	282.03	11.104
30000	9144	300.896	4.3641	225.69	8.8855
35000	10668	238.423	3.458	178.83	7.0406
40000	12192	187.54	2.72	140.67	5.5381
45000	13716	147.48	2.139	110.62	4.355
50000	15240	115.97	1.6821	86.987	3.4247
55000	16764	91.199	1.3227	68.405	2.6931
60000	18288	71.717	1.0402	53.792	2.1178
65000	19812	56.397	0.818	42.301	1.6654
70000	21336	44.377	0.6436	33.286	1.3105
75000	22860	34.978	0.5073	26.236	1.0329
80000	24384	27.615	0.4005	20.713	0.8155
85000	25908	21.837	0.3167	16.379	0.6448
90000	27432	17.296	0.2509	12.973	0.5107

95000	28956	13.721	0.199	10.291	0.4052
100000	30480	10.902	0.1581	8.1769	0.3219

Table 20 Trendline for Flow vs. PSI (Ascent)



Not part of the official report, but code for any future use

11.0 Flight Code

Main Code:

```
//Adam Cobb & Linda Karas DEMOSAT MASTER CODE 4
//April 1st, 2023

//Heater Control 1&2 initalize code
  int fetPin1 = 2;
  int fetPin2 = 3;
  bool heater1 = false;
  bool heater2 = false;
//-------------------------------------------------------------------
//BMP3XX temperature & pressure sensor initalize code

  #include <Wire.h>
```

```
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include "Adafruit_BMP3XX.h"
#include <math.h>
#include "FS.hpp"


#define SEALEVELPRESSURE_HPA ((30.08)*33.864)  //Set by looking at the local weather station
//inHg measurement and *33.864 will convert to Millibars here for calibration of altitude measurement.

Adafruit_BMP3XX bmp;
float Feet_calculation;


//-------------------------------------------------------------------------------
//MCP9808 temperature1&2 sensor initalize code
 //#include <Wire.h>
#include "Adafruit_MCP9808.h"

 // Create the MCP9808 temperature sensor objects with different I2C addresses
 Adafruit_MCP9808 tempsensor1;
 Adafruit_MCP9808 tempsensor2;

//-------------------------------------------------------------------------------
//MPRLS Exterior Pressure sensor initalize code
 //#include <Wire.h>
#include "Adafruit_MPRLS.h"

 // You dont *need* a reset and EOC pin for most uses, so we set to -1 and don't connect
#define RESET_PIN  -1  // set to any GPIO pin # to hard-reset on begin()
#define EOC_PIN    -1  // set to any GPIO pin to read end-of-conversion by pin
 Adafruit_MPRLS mpr = Adafruit_MPRLS(RESET_PIN, EOC_PIN);
//-------------------------------------------------------------------------------
//Flow Volume Sensor initalize code
FS flowSensor;
//-------------------------------------------------------------------------------
//CO2 Sensor intitalize code

 //Don't use pins 8,10,11,12,13 those are used for the SD card reader
#include "SoftwareSerial.h"

 SoftwareSerial K_30_Serial(6,7);  //Sets up a virtual serial port
                                   //Using pin 12 for Rx and pin 13 for Tx
```

```
  byte readCO2[] = {0xFE, 0X44, 0X00, 0X08, 0X02, 0X9F, 0X25};  //Command packet to read Co2 (see app note)
  byte response[] = {0,0,0,0,0,0,0};  //create an array to store the response

  //multiplier for value. default is 1. set to 3 for K-30 3% and 10 for K-33 ICB
  int valMultiplier = 1;
//--------------------------------------------------------------------------------
//SD Card initalize code
  #include <SD.h>

  File DataFile;
  const int SDPin = 8;  // pin 8 for spark fun sd card module
//--------------------------------------------------------------------------------
//Timer initalize code
  unsigned long startTime = 0; // Initialize start time to 0
  unsigned long elapsedTime = 0; // Initialize elapsed time to 0
//--------------------------------------------------------------------------------



void setup() {
  delay(250);//Give time for everything to turn on
  //SD Card setup code
    SD.begin(SDPin); //connect to sd card
    delay(10);//delay for SD card to turn on
    DataFile = SD.open(F("FLIGHT.txt"), FILE_WRITE); // create a file object named "DataFile" and use it to open the existing file named
"TestData.txt." on the Sd card

    DataFile.println(F("\n\nBegin of data collection\n\n"));

    DataFile.close();   // Close the file to save the data
  //--------------------------------------------------------------------------------
  //Heater Control 1&2 setup code
    Wire.begin();
    pinMode(fetPin1, OUTPUT);
    pinMode(fetPin2, OUTPUT);
  //--------------------------------------------------------------------------------
  //BMP3XX temperature & pressure sensor setup code
    //Serial.begin(9600);
    while (!Serial);
    //Serial.println("Adafruit BMP388 / BMP390 test");


    if (!bmp.begin_I2C()) {   // hardware I2C mode, can pass in address & alt Wire
    //if (! bmp.begin_SPI(BMP_CS)) {  // hardware SPI mode
    //if (! bmp.begin_SPI(BMP_CS, BMP_SCK, BMP_MISO, BMP_MOSI)) {  // software SPI mode
```

```
    //Serial.println("Could not find a valid BMP3 sensor, check wiring!");
    while (1);
  }

  // Set up oversampling and filter initialization
  bmp.setTemperatureOversampling(BMP3_OVERSAMPLING_8X);
  bmp.setPressureOversampling(BMP3_OVERSAMPLING_4X);
  bmp.setIIRFilterCoeff(BMP3_IIR_FILTER_COEFF_3);
  bmp.setOutputDataRate(BMP3_ODR_50_HZ);


//--------------------------------------------------------------------------
//MCP9808 temperature sensor1&2 setup code
  //Serial.begin(9600);
  //while (!Serial); //waits for serial terminal to be open, necessary in newer arduino boards.
  //Serial.println("MCP9808 demo");

  // Make sure the first sensor is found
  if (!tempsensor1.begin(0x19)) {
    //Serial.println("Couldn't find MCP9808 #1! Check your connections and verify the address is correct.");
    while (1);
  }
  //Serial.println("Found MCP9808 #1!");

  // Make sure the second sensor is found
  if (!tempsensor2.begin(0x1A)) {
    //Serial.println("Couldn't find MCP9808 #2! Check your connections and verify the address is correct.");
    while (1);
  }
  //Serial.println("Found MCP9808 #2!");

  // Set the resolution mode for each sensor
  tempsensor1.setResolution(3);
  tempsensor2.setResolution(3);
//--------------------------------------------------------------------------------------------------------------
//MPRLS Pressure sensor setup code
  //Serial.begin(9600);
  //Serial.println("MPRLS Simple Test");
  if (! mpr.begin(0x18)) {
    //Serial.println("Failed to communicate with MPRLS sensor, check wiring?");
    while (1);
  }
  //Serial.println("Found MPRLS sensor");
    //--------------------------------------------------------------------------------------------------------------
//Flow Volume Sensor setup code
```

```
  // initialize serial communications at 9600 bps:
  //Serial.begin(9600);
  // set internal analog reference to 1.1V
  analogReference(INTERNAL);
 //-------------------------------------------------------------------------------------------------------
 //CO2 Sensor setup code
  // put your setup code here, to run once:
  Serial.begin(9600);        //Opens the main serial port to communicate with the computer
  K_30_Serial.begin(9600);    //Opens the virtual serial port with a baud of 9600
 //-------------------------------------------------------------------------------------------------------
}
```

```
void loop() {
 //SD Card loop code
  DataFile = SD.open(F("FLIGHT.txt"), FILE_WRITE); // create a file object named "DataFile" and use it to
  //open the existing file named "TestData.txt." on the Sd card
 //-----------------------------------------------------------------
 //Timer loop code
  if (startTime == 0) {
  startTime = millis(); // Record the start time if it hasn't been recorded yet
  }

  elapsedTime = ((millis()) - (startTime)); // Calculate the elapsed time

  // Output the elapsed time in milliseconds to the serial monitor
  DataFile.print(F("Elapsed time (ms): "));
  DataFile.println(elapsedTime);
 //-----------------------------------------------------------------
 //BMP3XX temperature & pressure sensor loop code
  if (! bmp.performReading()) {
    //Serial.println("Failed to perform reading :(");
    DataFile.println(F("Failed to perform reading :("));
    return;
  }
  //Serial.print("BMP390 Temp = ");
  //Serial.print(bmp.temperature);
  //Serial.println(" *C");

  DataFile.print(F("BMP390 Temp = "));
  DataFile.print(bmp.temperature);
```

```
DataFile.println(F(" *C"));

//Serial.print("BMP390 Pressure = ");
//Serial.print(bmp.pressure / 100.0);
//Serial.println(" hPa");

    DataFile.print(F("BMP390 Pressure = "));
DataFile.print(bmp.pressure / 100.0);
DataFile.println(F(" hPa"));

//Serial.print("BMP390 Approx. Altitude = ");
//Serial.print(bmp.readAltitude(SEALEVELPRESSURE_HPA));
//Serial.print(" m and ");
Feet_calculation = ((bmp.readAltitude(SEALEVELPRESSURE_HPA)) * 3.28084);
//Serial.print(Feet_calculation);
//Serial.println(" ft");

DataFile.print(F("BMP390 Approx. Altitude = "));
DataFile.print(bmp.readAltitude(SEALEVELPRESSURE_HPA));
DataFile.print(F(" m and "));
DataFile.print(Feet_calculation);
DataFile.println(F(" ft"));

//-------------------------------------------------------------------------------------------
//MCP9808 temperature 1&2 sensor loop code
////Serial.println("Reading temperatures...");

// Read and print the temperature from the first sensor
float c1 = tempsensor1.readTempC();
float f1 = tempsensor1.readTempF();
Serial.print("MCP9808 Sensor #1 temp: ");
Serial.print(c1, 4);
Serial.print(" *C and ");
Serial.print(f1, 4);
Serial.println(" *F.");

DataFile.print(F("MCP9808 Sensor #1 temp: "));
DataFile.print(c1, 4);
DataFile.print(F(" *C and "));
DataFile.print(f1, 4);
DataFile.println(F(" *F."));

// Read and print the temperature from the second sensor
float c2 = tempsensor2.readTempC();
float f2 = tempsensor2.readTempF();
```

```
  // Serial.print("MCP9808 Sensor #2 temp: ");
  // Serial.print(c2, 4);
  // Serial.print(" *C and ");
  // Serial.print(f2, 4);
  // Serial.println(" *F.");

  DataFile.print(F("MCP9808 Sensor #2 temp: "));
  DataFile.print(c2, 4);
  DataFile.print(F(" *C and "));
  DataFile.print(f2, 4);
  DataFile.println(F(" *F."));
//-----------------------------------------------------------------------------------------------
//MPRLS Exterior Pressure sensor loop code
  float pressure_hPa = mpr.readPressure();
  //Serial.print("MPRLS Pressure (hPa): "); //Serial.println(pressure_hPa);
  //Serial.print("MPRLS Pressure (PSI): "); //Serial.println(pressure_hPa / 68.947572932);

  DataFile.print(F("MPRLS Pressure (hPa): ")); DataFile.println(pressure_hPa);
  DataFile.print(F("MPRLS Pressure (PSI): ")); DataFile.println(pressure_hPa / 68.947572932);
//-----------------------------------------------------------------------------------------------
//Flow Volume sensor loop code
  // initialize average sensor value

  float sensorAverage = flowSensor.getFlow();
  //print the results to the serial monitor:
  // Serial.print(F("FS1012 flow sensor = " ));
  // Serial.println(sensorAverage);

  DataFile.print(F("FS1012 flow sensor = ") );
  DataFile.print(sensorAverage,4); DataFile.println(F(" LPM"));


  // wait to ensure at least 100ms between each serial print
  //delay(serialRateOutput - (sampleAverage * analogSampleDelay));
//-----------------------------------------------------------------------------------------------
//CO2 Sensor loop code
  sendRequest(readCO2);
  unsigned long valCO2 = getValue(response);
  // Serial.print("Co2 ppm = ");
  // Serial.println(valCO2);
  // Serial.println("");
  DataFile.print(F("Co2 ppm = "));
  DataFile.println(valCO2);
  // DataFile.print("Co2 ppm = ");
  // do
```

```
  // {
  //   sendRequest(readCO2);
  //   valCO2 = getValue(response);
  // } while(valCO2 > 10000);

  // if(valCO2>10000){
  //   sendRequest(readCO2);
  //   unsigned long valCO2 = getValue(response);
  // }
  // else{
  //   Serial.println(valCO2);
  //Serial.println(valCO2);
  //Serial.println("");


  /***************************heaters************************/
  DataFile.print(F("heater1 = ")); DataFile.println(heater1);
  DataFile.print(F("heater2 = ")); DataFile.println(heater2);

  DataFile.println("");
//-------------------------------------------------------------------------------------------------
//Heater Control 1&2 loop code

  if (c1 > 16) {
  digitalWrite(fetPin1, LOW); // 0% heater off
  heater1 = false;
  }
  else if (c1 < 11) {
  digitalWrite(fetPin1, HIGH); // heater on at 100%
  heater1 = true;
  }
  if (c2 > 16) {            //59 *F
  digitalWrite(fetPin2, LOW); // 0% heater off
  heater2=false;
  }
  else if (c2 < 11) {       //50 *C
  digitalWrite(fetPin2, HIGH); // heater on at 100%
  heater2=true;
  }
  // else {
  // analogWrite(fetPin, 179); // 70% duty cycle on heater
  // delay(10000); }
  // delay(1000);}
//-------------------------------------------------------------------------------------------------
//SD Card close and save loop code
```

```
   DataFile.close(); // Close the file to save the data
  //-----------------------------------------------------------------------------------------------
  //Total Delay time loop code
   delay(311); //total loop delay
  //-----------------------------------------------------------------------------------------------
}



void sendRequest(byte packet[])
{
  while(!K_30_Serial.available())  //keep sending request until we start to get a response
  {
   K_30_Serial.write(readCO2,7);
   delay(50);
  }

  int timeout=0;  //set a timeoute counter
  while(K_30_Serial.available() < 7 ) //Wait to get a 7 byte response
  {
   timeout++;
   if(timeout > 10)    //if it takes to long there was probably an error
     {
       while(K_30_Serial.available())  //flush whatever we have
         K_30_Serial.read();

         break;                     //exit and try again
     }
     delay(50);
  }

  for (int i=0; i < 7; i++)
  {
   response[i] = K_30_Serial.read();
  }
}

unsigned long getValue(byte packet[])
{
   int high = packet[3];                    //high byte for value is 4th byte in packet in the packet
   int low = packet[4];                     //low byte for value is 5th byte in the packet


   unsigned long val = high*256 + low;              //Combine high byte and low byte with this formula to get value
   return val* valMultiplier;
```

```
}
```

FS.cpp code:

```
#include "FS.hpp"

FS::FS(){
  this->address=0x07;//address for the sensor
}



int FS::getAddress(){

  return this->address;
}


float FS::getFlow(){


  Wire.requestFrom(this->address,2);
  delay(1);
  int MSB = Wire.read();
  int LSB = Wire.read();

  float LPM = ((MSB<<8)+LSB)/1000.0;

  return LPM;

}
```

FS.hpp code:

```
#pragma once
#include "Arduino.h"
#include <Wire.h>

class FS{

  private:
    int address;

  public:
```

```
  int getAddress();//return the i2c address the sensor is using.

  FS();//constructor

  float getFlow();//get the flow rate of the sensor

};
```
