Precision Differential Drone Navigation

Yue Su, Paul Malachuk, Dirgh Shah, Arjun Khorana

Abstract

This research lab aimed to use sensitive barometric pressure measurements and accelerometer-based inertial navigation to improve a drone's ability to determine its position and altitude. More precise knowledge of the mid-air location of expensive flying equipment would holistically improve real-life drone applications such as forensic investigation, film production, and navigation in unknown territories. A microcontroller-based drone-mounted instrument package was assembled with a variety of sensors, particularly the Adafruit LSM9DS1 Accelerometer and DPS310 Pressure Sensor, in order to measure the acceleration, pressure, and temperature that the drone experienced while in flight.

Then, the above measurements were processed using mathematical concepts such as Integral Calculus, Fourier Transformations, and the Barometric Formula. This toolkit allowed for calculations of the position and altitude of the drone in different movement conditions. Using Python to implement these methods, graphs that estimated the position and height of the drone for different flight paths were developed. These position and altitude estimates were compared with those on the drone's built-in navigation systems as a baseline. It was found that the barometric altitude measurements were highly accurate to the real-life vertical movement of the drone, but the inertial navigation based on accelerometer data alone was unable to accurately estimate its position. Therefore, one could conclude that barometric pressure measurements with a pressure sensor could help us approximate the position and altitude, while inertial navigation remains a route for further study.

Introduction

The magic and difficulty of modern navigation and positioning systems is the problem of turning easily measurable quantities from sensors and turning them into real location data, as is shown in research from Castro-Toscano et al. (2017). This issue has been found in multiple real-life scenarios that include determining "three-dimensional coordinates of an observed object, stereo vision systems for three-dimensional space location, and global positioning systems for its capability of providing accurate navigation information" (Castro-Toscano et al., 2017). This issue is now investigated by this group, in a particular example of using electronic hardware sensors to create an inertial navigation system that could detect the position and altitude of a drone. The group aspired to study the feasibility of determining the flight path of a drone with the use of a custom-designed suite of airborne instruments. Specifically, the aim was to test methods of improving the ability of various hardware sensors to derive as accurately as possible the location of the drone in space.

Quantities such as position and altitude are vital to determining the device's general location. After further research into the topic, however, the group noticed that while most models of drones could estimate their position and altitude, they could not display their exact location by themselves. Often, one would need to connect their phone so that the drone's remote controller could connect to a navigation system provided by the phone. Ultimately, this turned out to be the case with the drone used in this project, the DJI Mavic 2 Pro, manufactured by SZ DJI Technology Co., Ltd. Upon noting these observations, the group aspired to answer the research question, "Can a combination of sensitive barometric pressure measurements and accelerometer-based inertial navigation improve a drone's ability to determine its position and altitude?"

Hardware

For reference, this section includes the specifications of the equipment used to this end. The following items include the drone used in this lab and the suite of electronics equipment, mostly sensors, that the group selected to attach to it during flight.

1) DJI Mavic 2 Pro:

The DJI Mavic 2 Pro is a drone capable of the following properties shown in Figures #1-#2 and Table #1 below.

Figure #1: DJI Mavic 2 Pro

Max Ascent Speed	5 m/s (S-mode)(not used), 4 m/s (P-mode)
Max Descent Speed	3 m/s (S-mode)(not used), 3 m/s (P-mode)
Max Speed (near sea level, no wind)	72 kph (S-mode)(not used)
Maximum Takeoff Altitude	6000 m
Hovering Accuracy Range	Vertical: ± 0.1 m (when vision positioning is active) ± 0.5 m (with GPS positioning) Horizontal: ± 0.3 m (when vision positioning is active) ± 1.5 m (with GPS positioning)

Table #1: Properties of the DJI Mavic 2 Pro

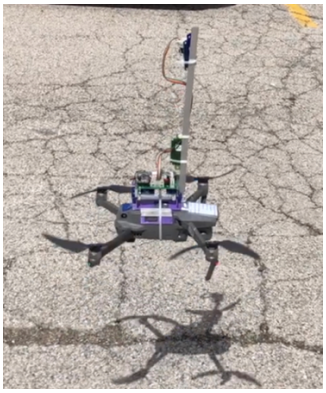

Figure #2: DJI Mavic 2 Pro with all the measuring equipment assembled taking off from the ground

2) Adafruit LSM9DS1 Accelerometer + Gyro + Magnetometer 9-DOF Breakout:
This breakout circuit board is "a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor." (STMicroelectronics, 2022). Hence, the device provides 9 degrees of freedom in the x, y, and z dimensions. Additionally, it supports both I2C protocol, short for *Inter-Integrated Circuit,* and SPI protocol, short for *Serial Peripheral Interface.* Data can be read at a rate up to 400 kHz. Moreover, each sensor supports a broad spectrum of ranges (Fried, 2017a):

1. The accelerometer's scale can be set to "± 2, 4, 8, or 16 g," where g represents the acceleration due to gravity.
2. The gyroscope supports angular frequencies of "± 245, 500, and 2000 degrees per second".
3. The magnetometer contains "± 4, 8, 12, or 16 gauss" full-scale ranges.
4. The LSM9DS1 possesses a gyro angular zero rate of ± 30 degrees per second at its highest sensing range.
5. The accelerometer offset accuracy is ± 90 mg.

For this lab, however, only measurements from the accelerometer were taken in order to derive the position of the drone. Given the time constraints placed on this project, the gyroscope was unable to be used to account for instances where the DJI Mavic 2 Pro was rotated in mid-air.

3) Adafruit DPS310 Precision Barometric Pressure and Altitude Sensor:

This breakout board includes a barometric sensor that allows for a precision of only ±0.002 hectopascals (comparable under normal conditions to ±0.02 meters of altitude) to be measured. Further, the DPS310 is also built with an absolute accuracy of ±1 hectopascal in pressure, comparable to altitudes within an accuracy of 1 meter. Additionally, the sensor can detect temperature with an uncertainty of ±0.5°C in ambient temperatures ranging from -40°C to 85°C (Rembor, 2005). For the purposes of this lab, a level of high precision and accuracy was provided by the DSP310 sensors when collecting our pressure and temperature data measurements with the drone. This way, high precision and accuracy in the drone's altitude were also guaranteed using the Barometric Formula, as described when analyzing the processed data below.

4) Adafruit Ultimate GPS Breakout Board:

Another hardware device that was used in the data collection process was the Adafruit Ultimate GPS breakout board. This breakout board uses the MTK3339 chipset, a GPS (Global Positioning System) module that can "track up to 22 satellites on 66 channels", has an "excellent high-sensitivity receiver" with -165 decibels of tracking, and a "built-in antenna" "up to 10 location updates every second for high speed, high sensitivity logging, or high sensitivity tracking" (Fried, 2012).

5) Adafruit DS3231 Precision RTC Breakout:

The DS3231 is a low-cost I2C real-time clock (RTC) using a crystal oscillator for time tracking.

6) Adafruit Feather M0

The Feather M0 is the microcontroller used in this project. It uses an ATSAMD21G18 ARM Cortex M0 processor, clocked at 48 MHz and at 3.3V logic. Further, 256k GB of flash, 32k GB of RAM storage, and a built-in USB-to-Serial program with debugging capability is included in the Feather M0. Also, a built-in battery is included for charging purposes (Fried, 2015).

7) Adafruit BME680 - Temperature, Humidity, Pressure and Gas Sensor

The Adafruit BME680 sensor measures Temperature, humidity, barometric pressure, and VOC gas. In particular, humidity can be measured with ±3% accuracy, barometric pressure can be measured with an absolute accuracy of ±1 hPa, and temperature can be measured with ±1.0°C accuracy (Fried, 2017b). Originally, the BME680 was considered to measure the altitude using its pressure measurements. However, it was later found that this sensor was not giving output measurements as accurately or as quickly as the DPS310 sensors, so the DPS310 sensors were used for data collection instead and the BME680 was phased out.

8) Micro SD Card Breakout Board

The Micro SD breakout board can read and write data to microSD cards (Fried, 2013). As a result, these files can be stored securely for later use.

Procedure

Instrumentation:

Before collecting data, a way to mount the electronics on the drone safely and securely had to be found. An attachment device was provided that could carry the instrument package without interfering with the drone's functionality. This device is shown in Figure #3 below. Importantly, the structure includes a long pole to create a vertical distance of 27 centimeters between the two DPS310 Precision Barometric Pressure Sensors. This way, the different pressures that the two sensors were both recorded when collecting data. This improves the altitude calculation by avoiding any possible effects from the propellers. The package also includes the Adafruit Feather M0 microcontroller, a compact rechargeable battery to power it, and the other sensors described above: the LSM9DS1 Accelerometer, BME680 sensor and an Adafruit Ultimate GPS. Finally, also mounted towards the bottom of the device is the RTC clock, used in conjunction with the GPS for timekeeping, and the Adafruit MicroSD breakout board to save the data onto an SD card. These breakout boards were connected through a custom printed circuit board pictured in Figure #4 below.

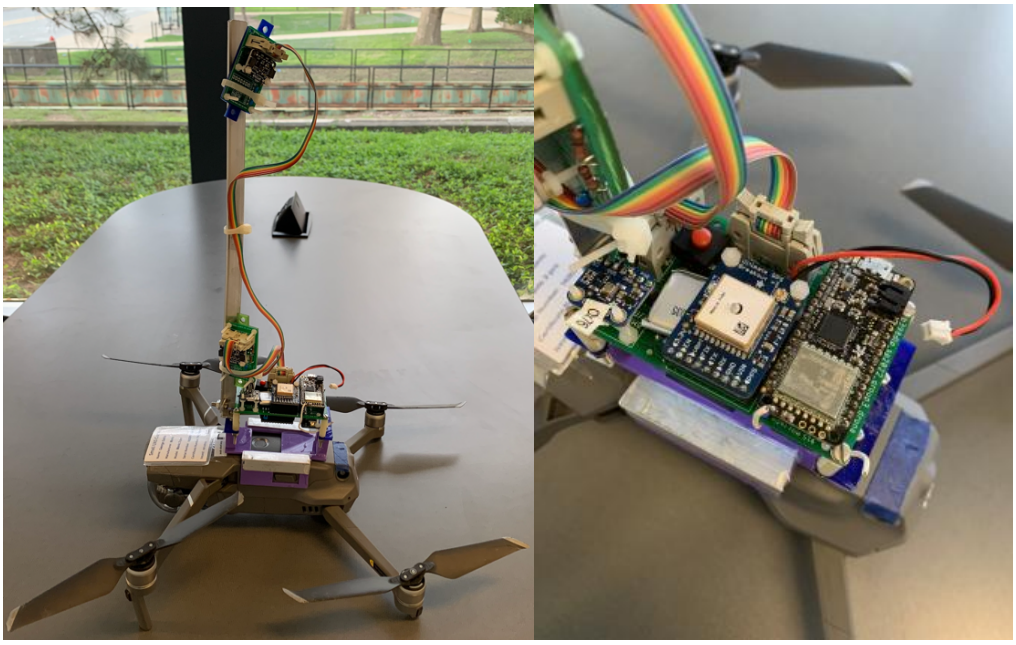

Figure #3: The drone attachment device used to collect data for the drone's position and altitude. This attachment device contains a printed circuit board, the instrument package, and a long pole to separate one DPS310 sensor from any propeller effects.

Figure #4: A close-up of the main circuit board. Visible from right to left are the Feather M0, GPS, Battery, and BME680. Note the rainbow ribbon cable connecting one of the DPS310 sensors mounted on the pole to the main circuit board.

The DAQ code running on the Feather M0 microcontroller collects data from each sensor during flight and writes it comma-separated to a CSV file stored on the SD Card. For each test run, one file is created and stored for analysis. An interval of recording time with the drone motionless on the ground is always stored for use as a reference point in analysis. Since speed and regularity are critical for the accuracy of our measurements, particularly the acceleration data, there is essentially no processing of the data done during flight. Interrupts, a timing method using the CPU clock, were used to keep a steady data writing rate of 1 measurement (1 line in the csv file) every 32 milliseconds. The LSM9DS1 and DPS310s can deliver new measurements at this rate, but the GPS and BME680 update significantly slower. BME680 data was not included in the final DAQ, as the slow read rate was affecting the timing process and its data contribution was redundant due to the two DPS310s. Images of the DAQ code and the resulting CSV file are shown in Figure #5 and Figure #6, respectively.

```
639   daq.print("Note that BME680 and GPS are read less often than DPS310s and LSM9DS1.");
640   daq.println();
641
642   daq.println("millis,BME_T,BME_H,BME_Pa,DPS0_T,DPS0_Pa,DPS1_T,DPS1_Pa,ax,ay,az,GPS_time,GPS_lat,GPS_lon,GPS_alt,GPS_cmnd,GPS_how_many");
643
644   daq.close();
645
646   Serial.println("Just wrote a header line to dfile.");
647
648 }
649
650 ////////////////////////////////////////////////////////////////////////
651
652 void setup() {
653
654   // GG: include a timeout on serial so we don't hang if there's no serial
655   // monitor connection.
656   uint32_t t1234 = millis();
657   Serial.begin(115200);
658   while (!Serial && millis() - t1234 < 5000) {}
659
660   // Serial.begin(115200);
661   // delay(5000);
662   //while(!Serial){}
663
664   Serial.println("Hello! Drone group DAQ starting up.");
665
666   // GG: let's also flash the LED a few times.
667
668   Serial.println("Let's flash the LED a few times...");
669   pinMode(LED_PIN, OUTPUT);
```

Figure #5: An example screenshot of the DAQ code running on the Adafruit Feather M0. Note the header line (642) containing the labels for the columns of comma-separated data.

```
S|40506,Drone csv data file opened is DAQ001.CSV,20:29:01,08/04/2022 UTC,Note that BME680 and GPS are read less often than DPS310s and LSM9DS1.
millis,BME_T,BME_H,BME_Pa,DPS0_T,DPS0_Pa,DPS1_T,DPS1_Pa,ax,ay,az,GPS_time,GPS_lat,GPS_lon,GPS_alt,GPS_cmnd,GPS_how_many
42525,0.00,0.00,0.00,3.56,98219.28,10.08,98213.98,0.016,-0.028,-1.014,0,0,0,0,no_new_GPS_yet,0
42543,0.00,0.00,0.00,3.56,98219.28,10.08,98213.98,0.015,-0.028,-1.014,0,0,0,0,no_new_GPS_yet,0
42558,0.00,0.00,0.00,3.56,98219.28,10.09,98214.04,0.016,-0.027,-1.013,0,0,0,0,no_new_GPS_yet,0
42572,0.00,0.00,0.00,3.56,98219.28,10.09,98214.04,0.015,-0.026,-1.012,0,0,0,0,no_new_GPS_yet,0
42585,0.00,0.00,0.00,3.56,98219.03,10.09,98214.04,0.018,-0.026,-1.012,0,0,0,0,no_new_GPS_yet,0
42598,0.00,0.00,0.00,3.56,98219.03,10.09,98214.04,0.018,-0.025,-1.011,0,0,0,0,no_new_GPS_yet,0
42611,0.00,0.00,0.00,3.56,98219.03,10.09,98214.04,0.015,-0.027,-1.010,0,0,0,0,no_new_GPS_yet,0
42624,0.00,0.00,0.00,3.56,98219.03,10.09,98214.04,0.015,-0.025,-1.009,0,0,0,0,no_new_GPS_yet,0
42641,0.00,0.00,0.00,3.56,98219.03,10.09,98214.04,0.015,-0.026,-1.010,0,0,0,0,no_new_GPS_yet,0
42654,0.00,0.00,0.00,3.56,98219.03,10.09,98215.29,0.016,-0.026,-1.010,0,0,0,0,no_new_GPS_yet,0
42667,0.00,0.00,0.00,3.56,98219.03,10.09,98215.29,0.016,-0.024,-1.009,0,0,0,0,no_new_GPS_yet,0
42680,0.00,0.00,0.00,3.56,98219.03,10.09,98215.29,0.015,-0.025,-1.009,0,0,0,0,no_new_GPS_yet,0
42695,0.00,0.00,0.00,3.56,98218.99,10.09,98215.29,0.016,-0.024,-1.010,0,0,0,0,no_new_GPS_yet,0
42711,0.00,0.00,0.00,3.56,98218.99,10.09,98215.29,0.016,-0.026,-1.011,0,0,0,0,no_new_GPS_yet,0
42725,0.00,0.00,0.00,3.56,98218.99,10.09,98215.29,0.013,-0.027,-1.010,0,0,0,0,no_new_GPS_yet,0
42740,0.00,0.00,0.00,3.56,98218.99,10.09,98215.29,0.015,-0.027,-1.011,0,0,0,0,no_new_GPS_yet,0
42753,0.00,0.00,0.00,3.56,98218.99,10.10,98215.45,0.014,-0.027,-1.012,0,0,0,0,no_new_GPS_yet,0
42766,0.00,0.00,0.00,3.56,98218.99,10.10,98215.45,0.016,-0.027,-1.012,0,0,0,0,no_new_GPS_yet,0
42781,0.00,0.00,0.00,3.56,98218.99,10.10,98215.45,0.016,-0.027,-1.013,0,0,0,0,no_new_GPS_yet,0
42795,0.00,0.00,0.00,3.56,98219.20,10.10,98215.45,0.015,-0.026,-1.014,0,0,0,0,no_new_GPS_yet,0
42808,0.00,0.00,0.00,3.56,98219.20,10.10,98215.45,0.015,-0.028,-1.013,0,0,0,0,no_new_GPS_yet,0
42821,0.00,0.00,0.00,3.56,98219.20,10.10,98215.45,0.016,-0.027,-1.013,0,0,0,0,no_new_GPS_yet,0
```

```
S|40506,Drone csv data file opened is DAQ001.CSV,20:29:01,08/04/2022 UTC,Note that BME680 and GPS are read less often than DPS310s and LSM9DS1.
millis,BME_T,BME_H,BME_Pa,DPS0_T,DPS0_Pa,DPS1_T,DPS1_Pa,ax,ay,az,GPS_time,GPS_lat,GPS_lon,GPS_alt,GPS_cmnd,GPS_how_many
41048,0.00,0.00,0.00,7.96,98215.12,16.19,98211.47,0.022,-0.049,-1.003,202525.000,4006.6669,08813.3550,224.4,GPGGA,3
```

Figure #6: An example screenshot of the CSV file stored on the SD Card after a test run, as well as a close-up of the header line labeling each column and a line including GPS data. The data from this file is collected from the SD Card and processed with Python. At the beginning of each column is the number of milliseconds since data collection began, which is used as the time marker in later analysis.

Once the data is collected, it can be viewed and processed. A python program is used to read the CSV file and store data into arrays which can be processed further. The code snippets in Figure #7 below shows how the data was read from the CSV file into arrays:

```
10
11  def dataloading(data,num):
12      arr = []
13      for i in range(len(data)):
14          if (num == 0 ):
15              arr.append(float(data[i][num])/1000)
16          elif (num == 4 or num == 6 ):
17              arr.append(float(data[i][num]))
18          elif (num == 5 or num == 7 ):
19              arr.append(float(data[i][num])/1000)
20          elif (num == 8 or num == 9 or num == 10):
21              arr.append(float(data[i][num])*-9.81)
22          elif (num == 12 or num == 13):
23              deg = float(data[i][num][:2])+(float(data[i][num][2:9])/60)
24              arr.append(deg*(np.pi/180))
25          else:
26              arr.append(float(data[i][num]))
27      return arr
28
```

```
38
39  file = open("/Users/dirgh/Desktop/Phys 398/dirghtest.csv","r")
40  raw_data = []
41  for lines in file:
42      raw_data.append(lines.split(','))
43
44  raw_data = raw_data[2:]
45
46  time = dataloading(raw_data,0)
47
48  acc_x = dataloading(raw_data,8)
49  a_x = []
50  for i in range(len(acc_x)):
51      a_x.append(acc_x[i] - np.mean(acc_x[:100]))
52
53  acc_y = dataloading(raw_data,9)
54  a_y = []
55  for j in range(len(acc_y)):
56      a_y.append(acc_y[j] - np.mean(acc_y[:100]))
57
58  acc_z = dataloading(raw_data,10)
59  a_z = []
60  for k in range(len(acc_z)):
61      a_z.append(acc_z[k] - np.mean(acc_z[:100]))
62
63  dps_temp_1 = dataloading(raw_data,4)
64  dps_temp_2 = dataloading(raw_data,6)
65
66  dps_pres_1 = dataloading(raw_data,5)
67  dps_pres_2 = dataloading(raw_data,7)
68
```

Figure #7: The code snippets that represent the conversion from data read in the CSV file into arrays.

In Figure #7 above, the function in lines 11 to 27 appends data into arrays alongside processing the data depending on which data is being read. For example, first the CSV file is read and stored in the variable raw_data as seen in the code lines 39 to 42. Line 44 of the code removes the column heading stored in the raw data. Then in line 48, the acceleration in the x axis is being stored in an array called acc_x. When the dataloading function is called, it recognizes that the acceleration values are read by the associative column number and multiplies each value by -9.81 m/s^2 to convert the raw values (which were read in terms of g, earths' gravitational acceleration, by the LSM9DS1) to get acceleration values in terms of meters per second. In lines 49 to 51, each acceleration value is subtracted from the average value of the acceleration of the drone during the stationary calibration period, because the sensor read a constant acceleration value even when the drone wasn't moving. Similarly, all the other values from the CSV file were read into their respective arrays, which can be easily plotted to view the raw data.

Flight Plan:

To assess the viability of the sensors and methods used, a flight plan was devised to systematically gather data relevant to the instruments and analysis tools. Important to this is having reference sources of position measurement, such as the drone's built-in navigation software, GPS data, and real-world estimates of distance traveled. These tests and their purposes are as follows:

Stationary Test:

The drone sits motionless on the ground indoors. Data is taken for 300 seconds to find the baseline acceleration noise and pressure/temperature fluctuations.

Propeller Test:

The drone is still motionless on the ground indoors, but with propellers on to isolate effects of motor vibration on acceleration data and propeller airflow on pressure data. After roughly 80 seconds for calibration, the propellers turn on for 100 seconds, then off again. The drone then rests for another 80 seconds.

Combined Vertical and Horizontal Test (Full Flight Test):

A vertical and horizontal flight test to demonstrate all instruments and analysis methods. The object of this flight was to gather pressure data suitable for altitude estimation as well as acceleration data suitable for integration. This means a vertical portion with clearly defined altitudes and a horizontal portion with clearly defined distances, all recorded directly from the Mavic 2 Pro's controller. The flight description is as follows:

> After a 100 second reference period of no movement, the drone ascended to 4, 13, then 33 meters directly upwards, waiting for 10-20 seconds at each altitude, then descended at a constant rate and landed at an overall time of 210 seconds. After 20 seconds motionless on the ground, it took off again and moved 24 meters in the "-x" direction (based on the orientation of the LSM9DS1) at a height of 10 meters. It then landed at this position at an overall time of 270 seconds. The final displacement of the drone should therefore be 24 meters in the -x direction and no motion in the y direction (where z is the vertical axis).

Results (Data Collection)

Before doing any processing on the data to get position values, it is useful to examine the plots of raw data directly to better understand the data being collected, find possible sources of error, and evaluate the effectiveness of the procedure.

Raw Data from Full Flight Test:

The Full Flight Test, while the most complex test, completely demonstrates the function of our instrumentation and is what most of the analysis is performed on. Shown in Figures #8-#11 below, there are graphs of the raw data, plotted from the DAQ for this test, a combined vertical and horizontal drone flight that ran on April 19th, 2022.

The DPS310 sensors provide a remarkably accurate illustration of the vertical component of the motion described for the test. The raw pressure data recorded from DPS1 (black, mounted close to the body of the drone) and DPS2 (red, mounted at the end of the pole) is shown in Figure #8 below. Note the "plateaus" where the drone did not move vertically. At some plateaus, however, there are still slight disturbances in pressure. Speculating, a reason for this is Bernoulli's Principle, where strong winds flowing over the DPS310 would cause a local decrease in pressure. This can be seen in the middle of the plateau around t=260 seconds. At that time, the drone is moving horizontally at speed and there is a pronounced drop in pressure that may be due to this effect, as the drone did not actually change its height during this part of the motion.

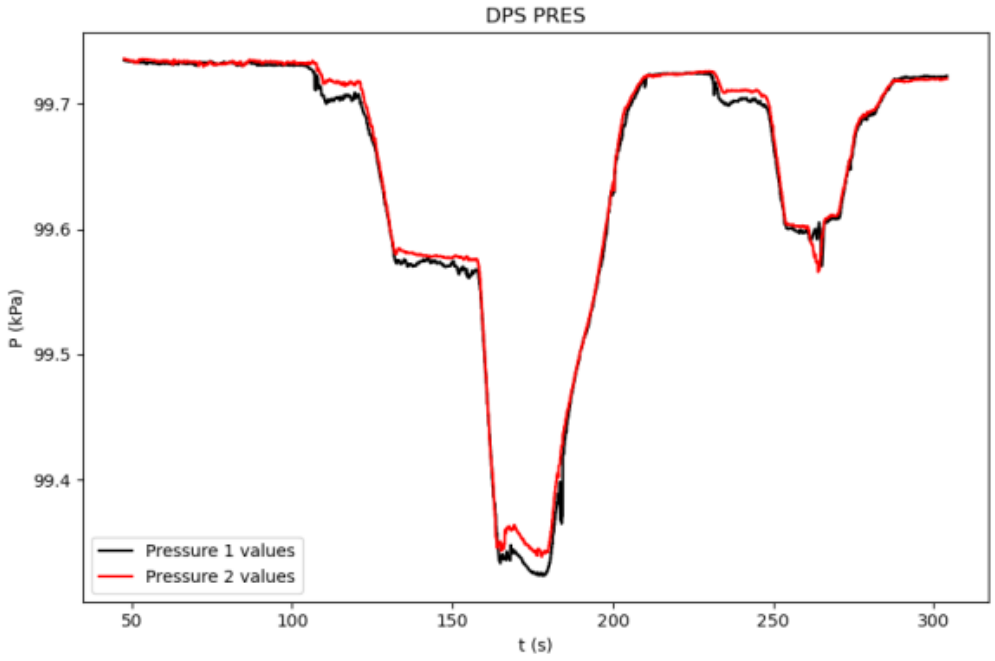

Figure #8: The raw pressure data recorded by both DPS310 Pressure sensors. DPS1, by the body of the drone, is in black, while DPS2, on the tip of the stick, is in red. Besides being an accurate map of the general timing and vertical motion of the flight, the spikes during high winds at t=170s and fast horizontal motion at t=260s could indicate the presence of Bernoulli's Principle.

This spike calls for calculation. For both DPS310s, the spike has a height of 38 Pa. The disturbance (and horizontal motion of the drone) occurs over a time of 5 seconds, and it is known from the drone controller that the drone travelled 24 meters. Bernoulli's Principle states that $\Delta P = \frac{1}{2} p \Delta v^2$, plugging in the pressure difference of 38 Pa and an air density of roughly 1.225kg/m^3, then the velocity would have to be 7.87m/s for the pressure difference to be as it appears in the graph. While this is faster than the average speed of 4m/s calculated from position and timing alone, the drone moved very quickly, at a non-constant rate (its max speed is 20m/s!), in a windy environment, so it would not be unreasonable if the total airflow over the DPS310s reached this speed.

The difference in pressure reading between the two DPS310s also increased whenever the drone was hovering and not moving vertically, as seen below in Figure #9. Likely due to the increase in wind at higher altitudes, it appears that the difference between the pressure readings correlates with the height of the drone.

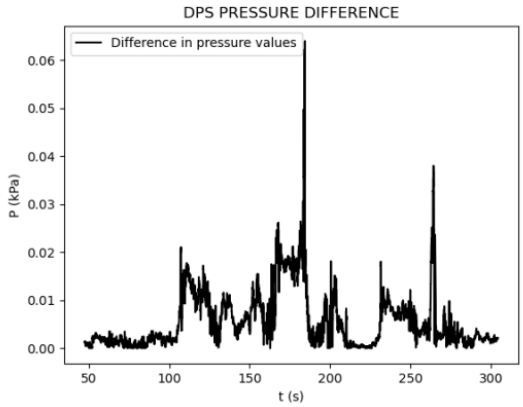

Figure #9: The absolute value of the difference between the two DPS310s. It was observed that the difference increases when the propellers turn on and as the drone takes to the windy sky.

The temperature recorded by both DPS310s is then shown in Figure #10 below. Interestingly, the temperature readings become more similar at higher altitudes, and there is a significant separation at the beginning of the data, even as the drone sits motionless on the ground.

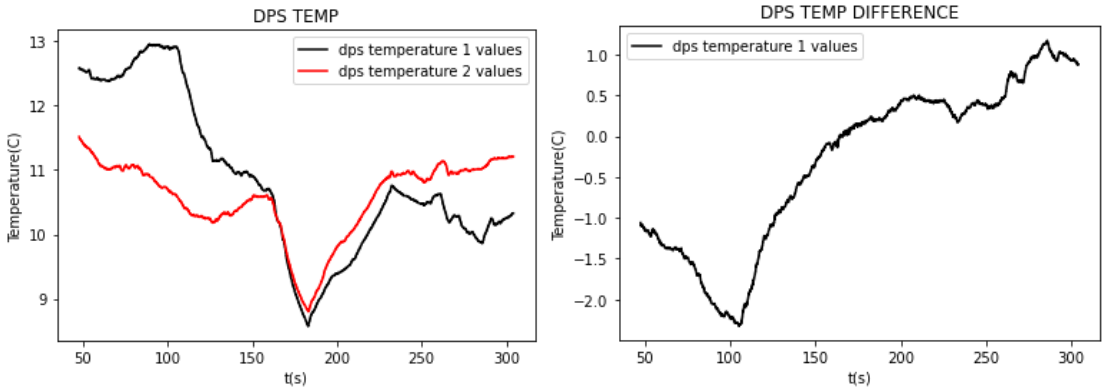

Figure #10: The temperature recorded by both DPS310s, the body-mounted DPS1 in black and the pole-mounted DPS2 in red.

The raw acceleration graphs from the DAQ show the degree that the vibration of the drone affects the raw acceleration data. Shown in Figure #11 below are the three axes (x, y, z) that the accelerometer measures independently. Recall that after 100 seconds, the propellers turn on and the drone begins its flight. Immediately, the acceleration data on all three axes is overcome with the effects of the resulting vibration. Little information can be immediately gleaned from this data, necessitating the analysis described below.

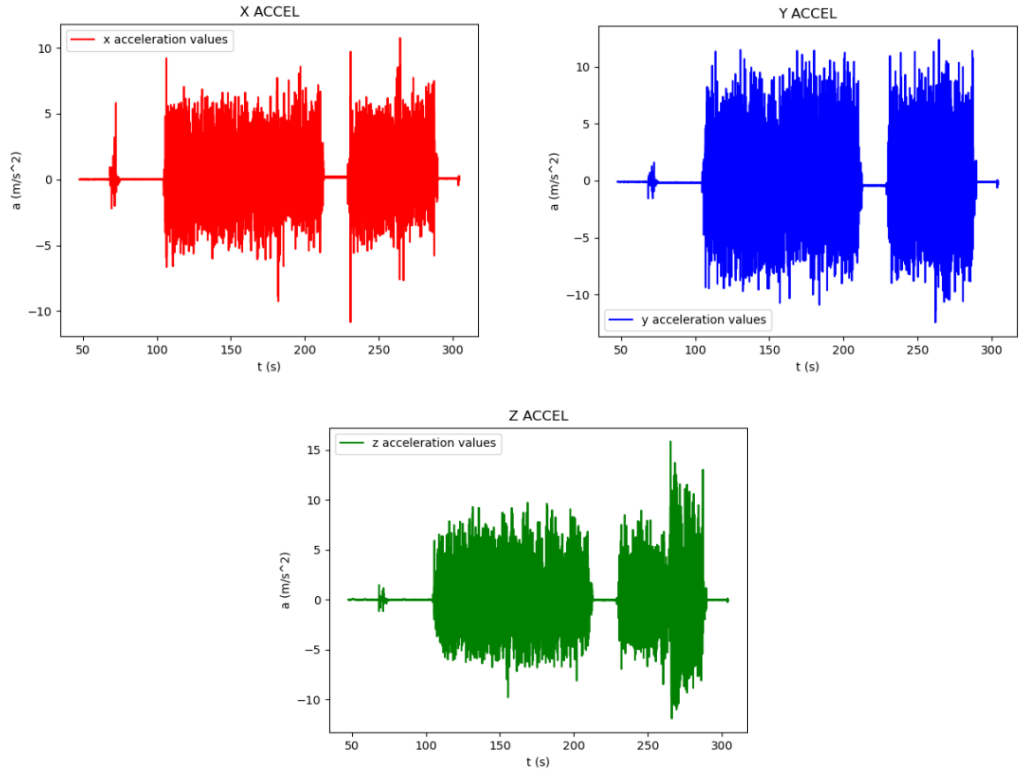

Figure #11: The raw acceleration data from the LSM9DS1 for each axis. The main feature of note is the intensity of vibration experienced by the drone.in flight. All data have been calibrated from the initial reference rest period, therefore the Z direction's 9.86m/s^2 acceleration due to gravity has been offset.

Raw Data from Propeller Test:

Now that the overview of the full flight test is complete and the expected data is clear, the other tests can be used as baselines for the sensor behavior that may explain anomalies, inconsistencies, and errors in the data. The Propeller Test, where the drone does not take flight, but the propellers are turned on between t=80s and t=170s, is next, as shown in Figure #12 below. The raw data from the DPS310s shows small fluctuations, compared to the large changes and plateaus visible in the full flight test above. The total change in pressure was negligible, only 19 Pa, but exhibits a curious downward trend, seemingly independent of the point where the propellers turn on.

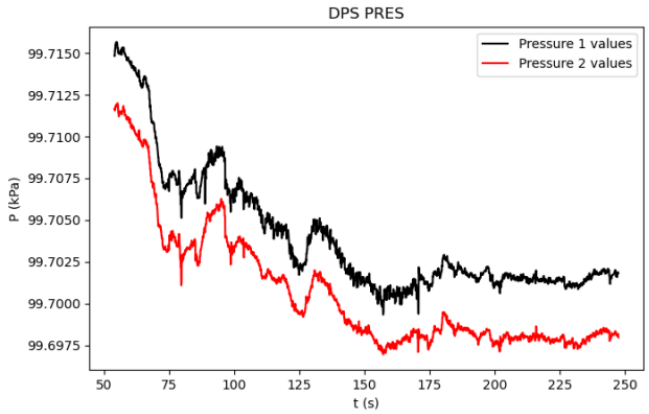

Figure #12: The raw pressure data from the two DPS sensors. DPS1 (in black) is near the body and DPS2 (in red) is on the pole.

More illuminating is the difference between the pressure values, as shown in Figure #13 below, where the effects of the propellers are clearly visible. This graph shows the absolute value of the pressure difference between the two DPS310s, which interestingly decreases when the propellers are on. Speculatively, this could be again due to the Bernoulli Principle, where air movement from the propellers causes DPS1's local pressure to decrease, while DPS2, at the top of its protective pole, feels no effects and keeps the same pressure. This could explain why the pressure difference would become smaller rather than larger. Unfortunately, this cannot be calculated like the spikes in the full flight test, as the speed of the air created by the propellers alone cannot be easily found. Also included in Figure #14 below is the temperature experienced by the DPS310 sensors. There is little of note, but the temperature of DPS1, by the body of the drone, does heat up ever so slightly once the propellers turn on. The drone body heats up slightly to the touch when it is active, so this may be detected by DPS1.

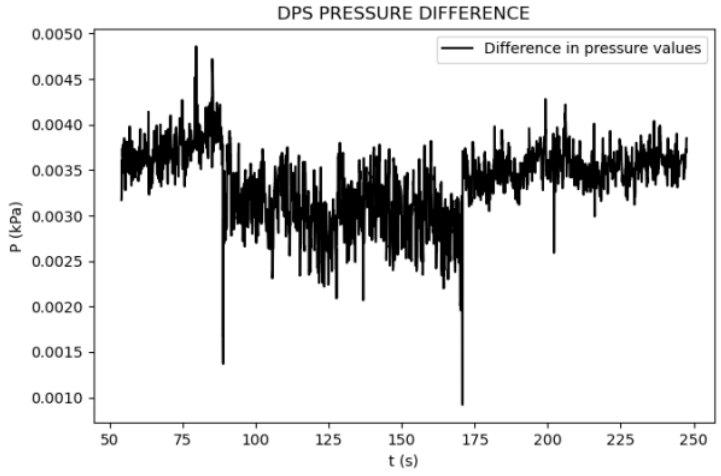

Figure #13: The difference between the two pressure sensors during the propeller test. The effect of the propellers visibly decreases the pressure difference between t=80s and t=170s.

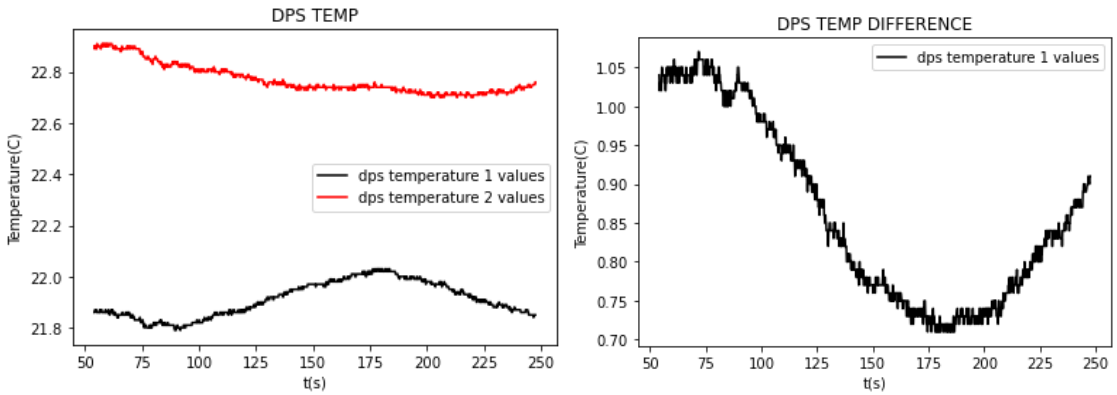

Figure #14: The temperature recorded by both DPS310s and the difference in the values, both of which slow and small variation.

The raw acceleration data, as shown in Figure #15 below, is similar to that of the combined test. The LSM9DS1 experiences periods of extremely noisy data due to the vibration when the motors are on. The degree of vibration experienced is more pronounced in the x and y directions because the propellers spin in that plane. There is a surprisingly high spike in acceleration in the x-axis just as the drone turns on.

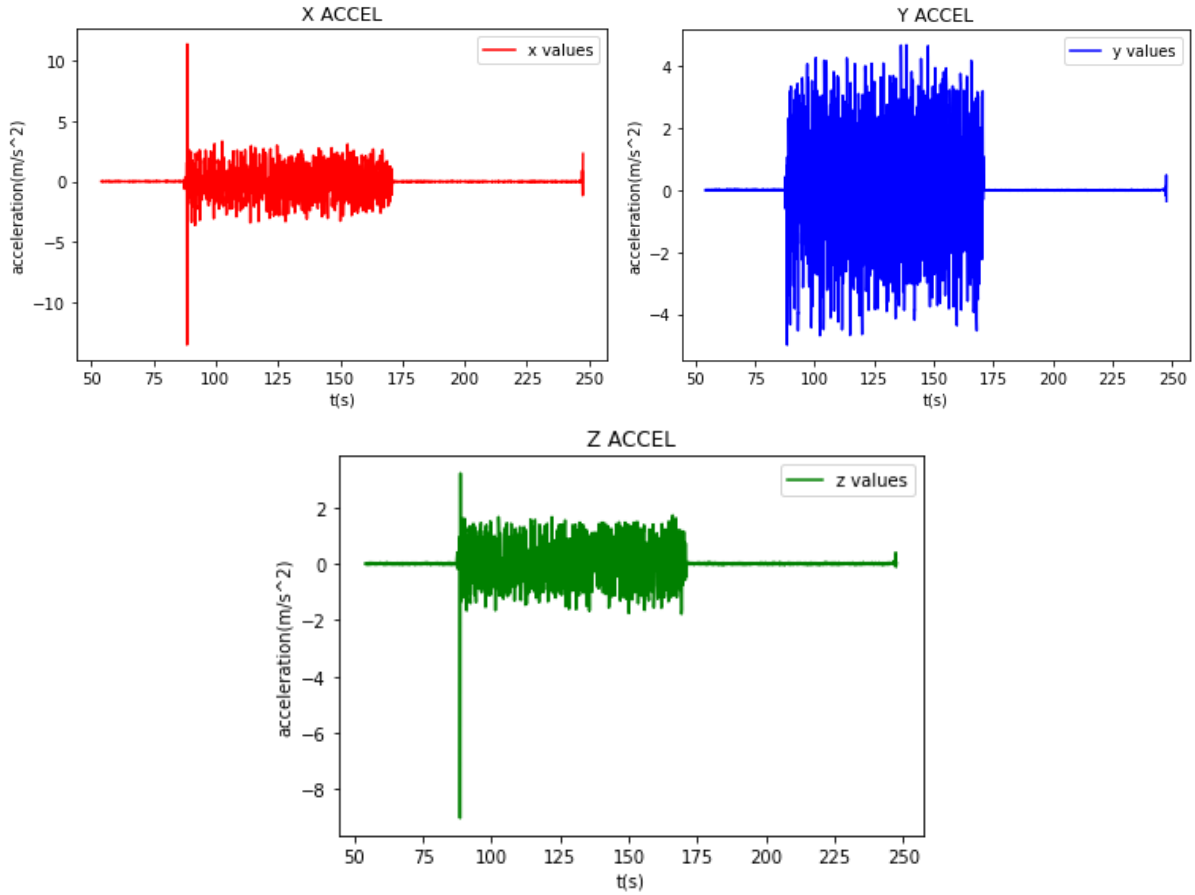

Figure #15: The acceleration in all three axes during the Propeller Test.

Raw Data from Stationary Test:

The stationary test, 400 seconds of a stationary drone, serves as a reference for the other tests. However, interesting patterns emerge, particularly in the overall pressure, as shown in Figure #16 below. There is a roughly 10 Pa decrease in pressure over the duration of the test, experienced exactly the same (save for the offset due to their height difference) by both pressure sensors. This is similar to the decrease observed in the Propeller Test. Since the motion is identical, the pressure reported must be the actual pressure of the room and so two sensors experiencing the same noise is unlikely. The level of disagreement is, at most, 4.4 Pa between the two DPS310s, shown in Figure #17 below. This is a testament to the precision of the DPS310, and an example of how using two of them allows for cross-calibration.

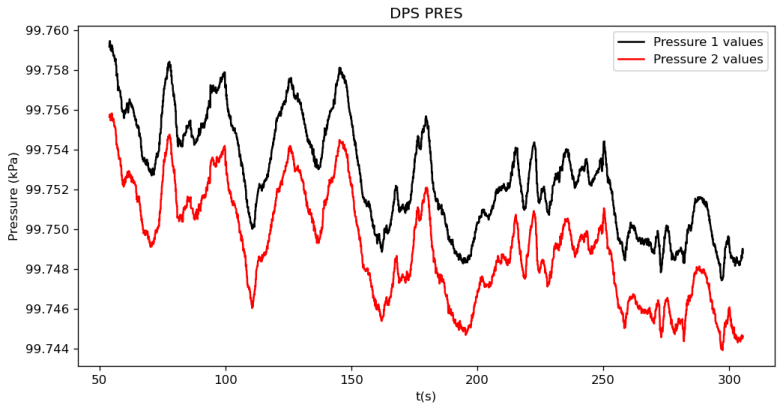

Figure #16: The difference between the two pressure sensors during the Stationary Test.

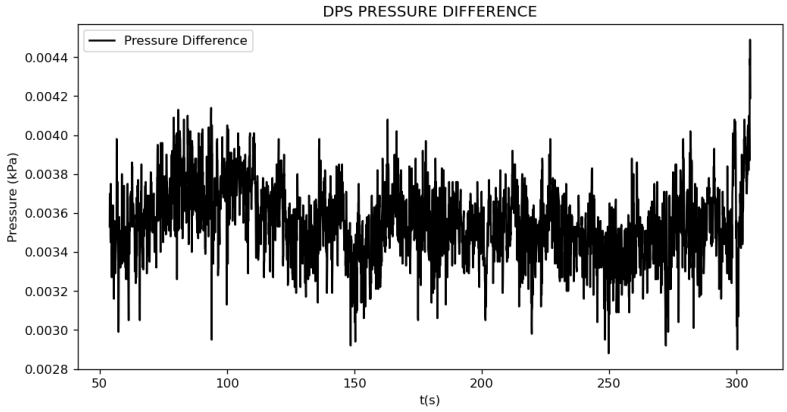

Figure #17: The corresponding difference in pressures from the Stationary Test.

The downward trend in the DPS310 temperature readings, shown in Figure #18 below, is very similar to the one seen in the Propeller Test. DPS2 has "cooled down" in both the calibration tests, which suggests a phenomenon, especially if the heating up of DPS1 in the Propeller Test is indeed due to the warmth of the drone body. This could be the sensors needing time from start-up to calibrate to their surroundings, which has unfortunately not been accounted for in this lab. Otherwise, the acceleration graphs shown in Figure #19 below are unremarkable, showing only small variations over time due to the noise of the LSM9DS1 and the environmental vibrations in the room.

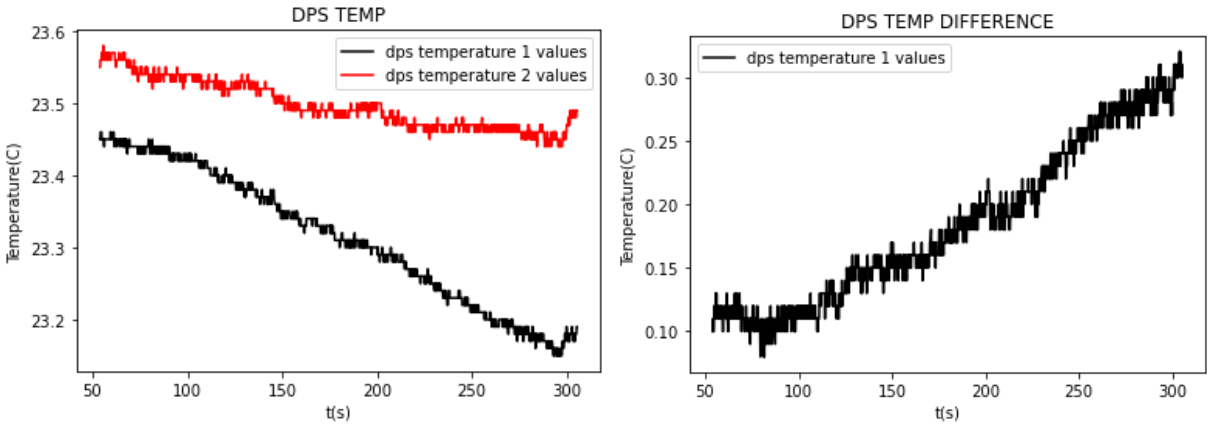

Figure #18: The temperature recorded by both DPS310s and the difference in the values, both of which show slow and small variation.

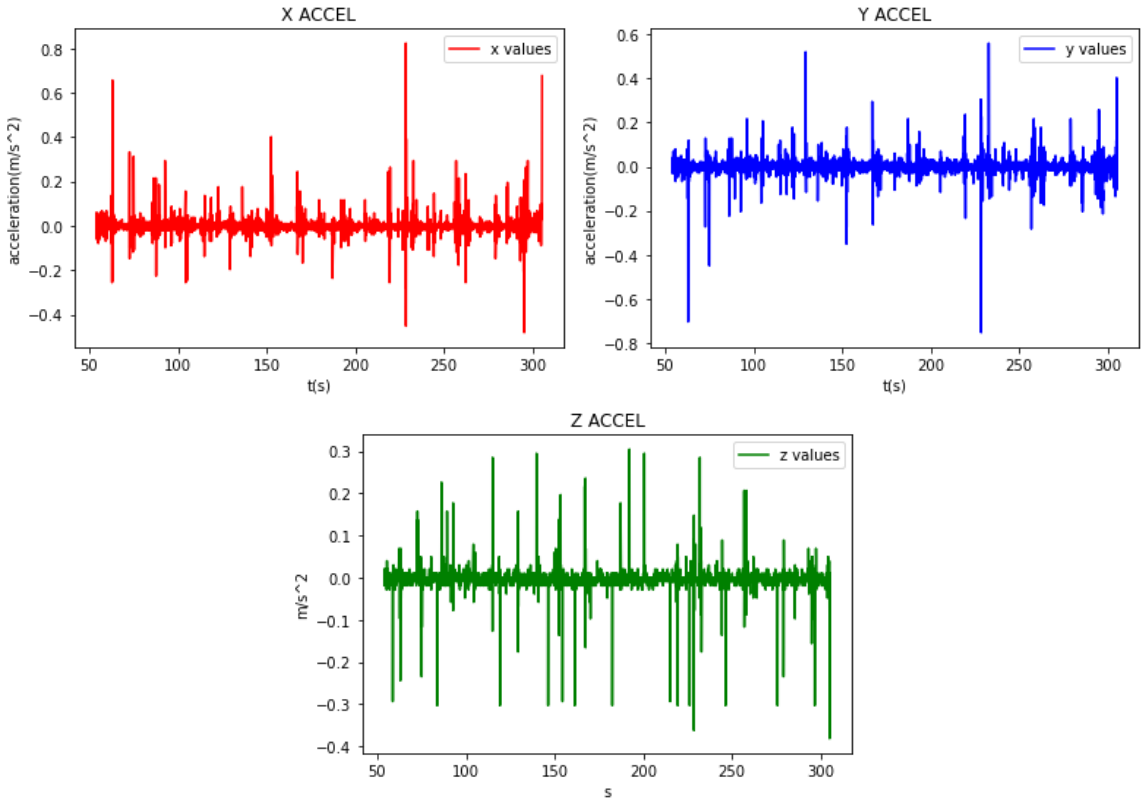

Figure #19: The acceleration in all three axes during the Stationary Test.

Data Processing

Drone Position Calculation using Inertial Navigation

Method #1: Numerical Integration (Trapezoidal Rule)

Inertial navigation is a system where the position of an object without external references is calculated by motion sensors within the object itself, such as the accelerometer in the instrument package mounted to the drone.

From Equation (1) below, we know that the acceleration is the second derivative of displacement with respect to time:

$$\frac{d^2s(t)}{dt^2} = a(t) \qquad (1)$$

Here, the displacement of the drone at time t is represented by s(t), and the acceleration of the object at time t is represented by a(t). The acceleration of the drone is measured by the sensor on the drone every 32 milliseconds. The acceleration values must be integrated twice for the displacement to be found. When acceleration is integrated once, velocity is found, and when velocity is integrated, displacement is found at time t, as seen in Equations (2) and (3) below:

$$v = v_0 + at \qquad (2)$$
$$x = x_0 + vt \qquad (3)$$

For the cumulative double integral to be computed, numerical integration was first attempted using the Trapezoidal Rule. This rule is a method of integration where the area under a curve is calculated by dividing the total area into smaller trapezoids. More specifically, the area enclosed by each trapezoid under the graph is summated into a final integration. Eventually, the approximation of the total area enclosed by the curve is shown in Figure #20 below.

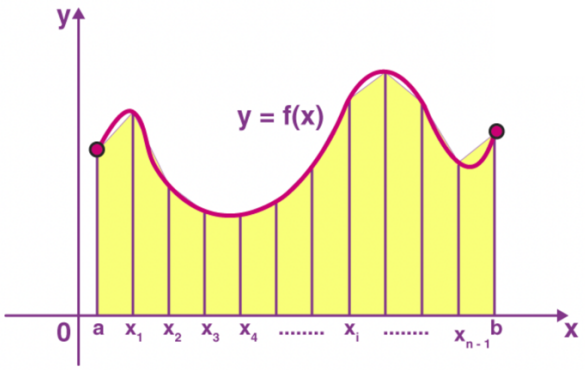

Figure #20: A graphical representation of the Trapezoid Rule.

For the Trapezoidal Rule to be better understood, let f(x) be a continuous function, as seen above in Figure #, on the interval [a, b]. The definite integral of f(x) is, thus, the sum of the areas of all the trapezoids. Subsequently, the integral shown in Equation (4) below is formed.

$$\int_a^b f(x)dx = \sum Area\ of\ Trapezoids \qquad (4)$$

Additionally, the area of a trapezoid is shown in Equation (5) below. Here, the lengths of the trapezoid bases are represented by variables a and b. The height of the trapezoid is further represented by the variable h.

$$A = \frac{a+b}{2}h \qquad (5)$$

Similarly, the height of each trapezoid in Equation (5) is equal to the length of the data intervals on the x-axis, and the base is equal to the f(x) values. So, the summation of all areas is shown in Equation (6) below, where the length of the intervals is equal, represented by the variable Δx:

$$\frac{\Delta x \times (f(x_0) + f(x_1))}{2} + \frac{\Delta x \times (f(x_1) + f(x_2))}{2} + \cdots + \frac{\Delta x \times \big(f(x_{n-1}) + f(x_n)\big)}{2}$$
$$= \frac{\Delta x}{2}(f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)) \qquad (6)$$

Furthermore, the displacement being the second integral of acceleration, the Trapezoidal Rule had to be carried out twice. The velocity of the drone was found from the first integration, while the displacement of the drone was found from the second integration. This result was achieved with the help of the Trapezoidal Integration Method module in the Python library "scipy.integrate" as shown in the code snippet from Figure #21 below:

```
267
268     v_x = integrate.cumtrapz(a_x, time, initial=0)
269
270     d_x = integrate.cumtrapz(v_x, time, initial=0)
271
272     plt.plot(time, v_x, 'b-', label = "Velocity travelled in x axis")
273
274     plt.title("X VELCOITY")
275     plt.ylabel('v_x (m/s)')
276     plt.xlabel('t (s)')
277     plt.legend()
278     plt.show()
279
280     plt.plot(time, d_x, 'g-', label = "Displacement travelled in x axis")
281     plt.title("X DISPLACEMENT")
282     plt.ylabel('s_x (m)')
283     plt.xlabel('t (s)')
284     plt.legend()
285     plt.show()
286
```

Figure #21: A code snippet of the Trapezoidal Integration Method in Python

In line 268, the acceleration values stored in the variable *a_x* are being integrated once using the Trapezoidal Rule, where the time interval is represented by the variable Δx. Then, in line 270, the array of velocity values was integrated once more to give an array of displacement values. This procedure was done with acceleration values from the x, y, and z-axis, and the resulting graphs can be seen in Figures #22, #23, and #24 below:

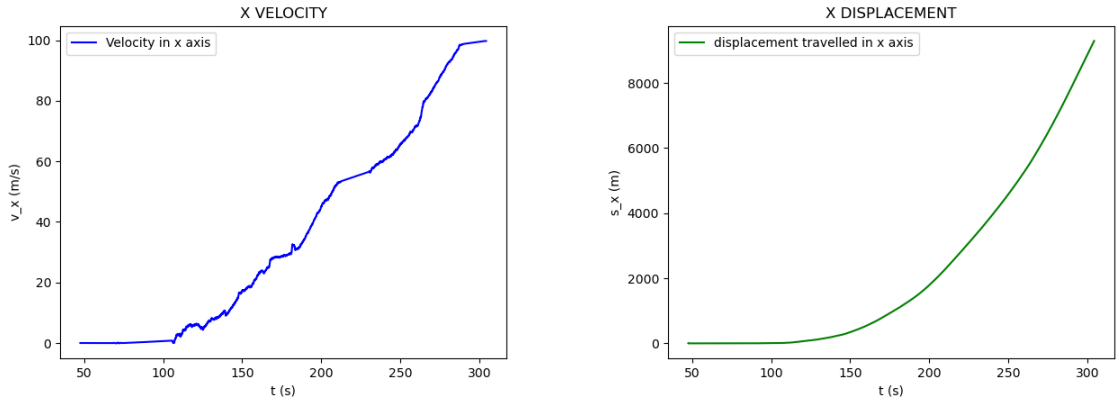

Figure #22: The velocity and displacement of the drone graphed in the x-direction with the use of numerical integration

As described in the full flight test, the drone was stationary from t = 0 to t = 100, accurately displayed in the velocity and displacement graphs. Then takeoff occurred, and displacement by the drone was found in the z-direction. After that, the drone landed at t = 210. From these observations, it is shown that if the drone only moves in the z-direction, acceleration values continue to be registered in the x and y-directions as well. Upon further discussion, it was found that a significant reason for the acceleration values was surrounding factors such as a windy environment and the jerky motion of the drone.

Further, the windy environment and the jerky motion of the drone resulted in the velocity in x-direction constantly increasing until t = 300. It is critical to note that this measurement had been conducted despite the drone being placed on the ground between t = 210 and t = 230, and t = 270 onwards. Thus, acceleration and velocity in the x-direction had occurred, as shown in Figure #22, despite different events occurring in reality. Consequently, it meant that there was exponential growth shown on the displacement graph in Figure #22, such that a distance of 9000 meters in the x-direction was eventually reached. This was because a constant velocity of the drone was assumed by the DAQ program throughout the data collection process. Since the actual displacement in the x-direction was 24 meters, the corresponding percentage error calculated turned out to be the value shown in Equation (7) below.

$$\frac{9000-24}{24} \times 100 = 37400\% \qquad (7)$$

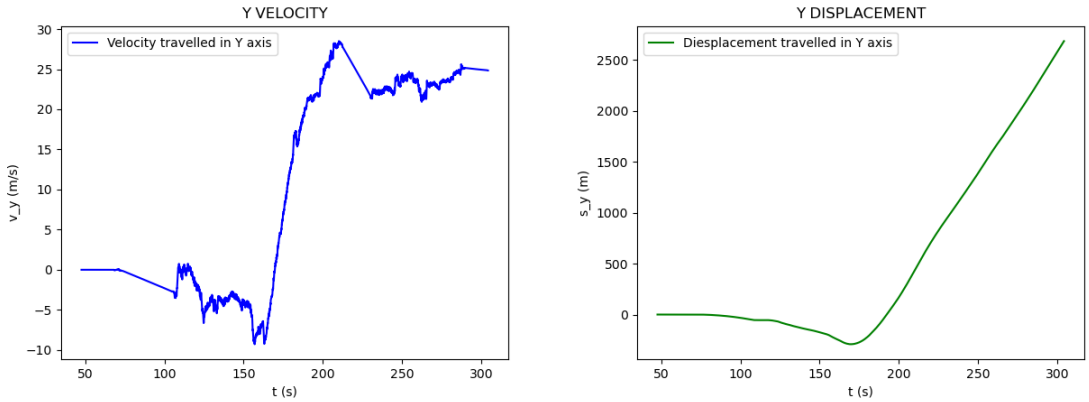

Figure #23: The velocity and displacement of the drone graphed in the y-direction with the use of numerical integration

Further, it can be seen in the graphs shown in Figure #23 above that the windy environment and the shaky motion of the drone resulted in the acceleration in the y-direction being indirectly recorded by the LSM9DS1 circuit board. Subsequently, peaks in the graph for the velocity in the y-direction were created, and the total displacement found at the end of the flight test was 2600 meters. This had occurred despite the drone not moving in the y-direction during the flight test.

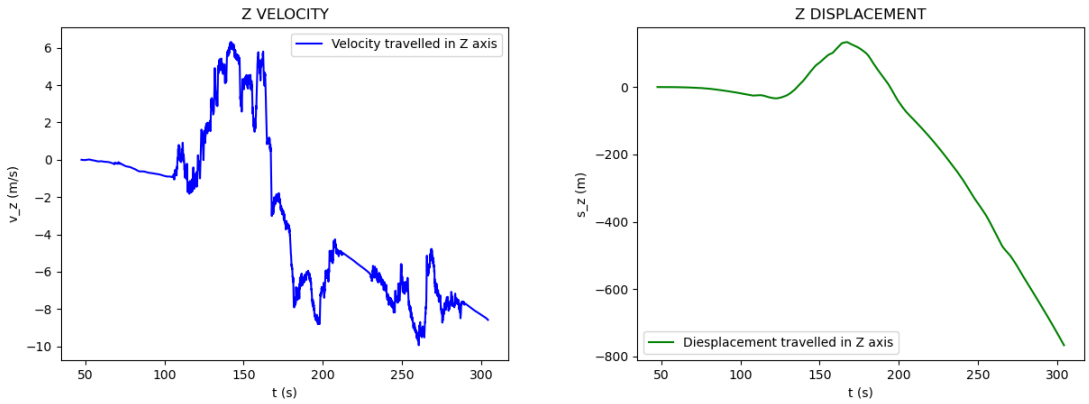

Figure #24: The velocity and displacement of the drone graphed in the x-direction with the use of numerical integration

Finally, it is observed in Figure #24 that values of 0 m/s and 0 meters are found for the velocity and displacement in the z-direction, respectively, between t = 0 and t = 100. This is closely correspondent to a stationary drone, as described in the test flight path. However, an altitude of 33 meters was reached at t = 160, accurately shown by an increase in velocity and displacement in Figure #24. Additionally, the descent of the drone from t = 160 before landing at t = 210 can also be seen in Figure #24 due to the increase in velocity in the negative z-direction and the decrease in displacement. However, the drone was stationary on ground level between t = 210 and t = 230, but a straight line at v_z = 0 was not registered due to the impact that the drone experienced upon landing. After that, the following ascent and descent at t = 230 and t = 270, respectively, can be seen. This is because a corresponding increase and decrease in velocity in

the z-direction can be seen in Figure #24 above. However, a low value is seen when calculating the difference in time between the first descent at t = 210 and the last descent at t = 270. Consequently, the motion between the low differences in time is neglected upon the final integration for the z-direction displacement to be found. Instead, it is seen as a continuous decrease in position. The max height calculated in this plot was 133.45 meters and so the percentage error produced was 304.39%, as shown in Equation (8) below:

$$\frac{133.45-33}{33} \times 100 = 304.39\% \qquad (8)$$

Hence, it can be observed that spikey acceleration values were recorded by the accelerometer in the LSM9DS1 circuit board. Ultimately, as shown from the above graphs in Figures #22 to #24, the resulting position of the drone obtained was not representative of the actual test flight path, especially as the error found for the displacement in the x-direction and z-direction increased to a value of 37400% and 304.39%, respectively. Therefore, it was discovered that for this lab, there was a high level of difficulty for numerical integration to be carried out when finding the position of the drone.

Method #2: Fourier Transformations

As a result of large imprecision and inaccuracy in the processed data from numerical integration through the Trapezoidal Rule, Fourier Transformations were then used to calculate the approximate position of the drone.

When the functions for the acceleration, $a(t)$, and displacement, $s(t)$, are integrated, the formulas for Fourier Transformations are found, shown in Equation (9) below. It is important to note that the angular frequency is represented by the variable w and the displacement in terms of the angular frequency is represented by the function $\tilde{s}(w)$. Thus, the acceleration in terms of the angular frequency is represented by the function $\tilde{a}(w)$.

$$\tilde{s}(w) = \int_{-\infty}^{\infty} s(t)e^{iwt}dt \qquad \tilde{a}(w) = \int_{-\infty}^{\infty} a(t)e^{iwt}dt \qquad (9)$$

The equations above can then be converted into Equation #10 below when $s(t)$ and $a(t)$ are isolated. These formulas are also known as the Inverse Fourier Transformation.

$$s(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{s}(w)e^{iwt}dw \qquad a(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{a}(w)e^{iwt}dw \qquad (10)$$

After finding the Inverse Fourier Transformation equations above, they are inputted into the differential equation, $\frac{d^2s(t)}{dt^2} = a(t)$, which is equivalent to Equation (11) below.

$$\frac{d^2}{dt^2}\left(\frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{s}(w)e^{iwt}dw\right) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{a}(w)e^{iwt}dw$$

$$\frac{1}{2\pi}\int_{-\infty}^{\infty} \frac{d^2}{dt^2}\left(\tilde{s}(w)e^{iwt}dw\right) = \frac{1}{2\pi}\int_{-\infty}^{\infty} \tilde{a}(w)e^{iwt}dw \qquad (11)$$

Given that the values inside the integrals are equal, Equation (11) can be further reduced to:

$$\frac{d^2}{dt^2}(\tilde{s}(w)e^{iwt}dw) = \tilde{a}(w)e^{iwt}dw$$

$$i^2w^2\tilde{s}(w)e^{iwt} = \tilde{a}(w)e^{iwt}$$

$$\tilde{s}(w) = \frac{\tilde{a}(w)}{i^2w^2} = \frac{-\tilde{a}(w)}{w^2}$$

Therefore, using the Fourier Transformation module in Python's SciPy library, the values for $\tilde{a}(w)$ and w can be calculated. Further, the values for $\tilde{s}(w)$ can be calculated upon dividing $-\tilde{a}(w)$ by w^2. From there, the array of values for s(t) can be derived using the Inverse Fourier Transformation function in Python, the code for which is shown in Figure #25 below.

```
240
241     afft = fft(a_z)
242     af = fftfreq(len(afft))
243
244     xfft = []
245
246     for i in range(len(afft)):
247         if (af[i] == 0):
248             xfft.append(0)
249         else:
250             xfft.append(-afft[i]/((2*np.pi*af[i])**2))
251
252
253     xdisp = ifft(xfft)
254
255
256     plt.plot(time, xdisp, 'g-', label = "displacement travelled in z axis")
257
258     plt.title("Z DISPLACEMENT")
259     plt.ylabel('s_z (m)')
260     plt.xlabel('t (s)')
261     plt.legend()
262     plt.show()
263
```

Figure #25: A Python code snippet where the Inverse Fourier Transformation function is portrayed.

In lines 241 and 242 from Figure #25 above, the $\tilde{a}(w)$ and frequency (f) values are calculated, respectively. Given that w is found by multiplying f by 2π, each frequency value is multiplied by 2π to obtain values for w in line 250 of Figure #_. Then, the for loop in lines 246 to 250 is used to divide each value of $\tilde{a}(w)$ by $-w^2$ in order $\tilde{s}(w)$ for to be found. However, if the value of w^2 is equal to zero, then the value of $\tilde{s}(w)$ returned by the for loop is equal to zero. Moreover, in line 253, the function s(t) is obtained by conducting an Inverse Fourier Transformation on the values for $\tilde{s}(w)$.

As a result, the displacement of the drone in the x, y, and z-direction is calculated with the use of the code in Figure #25 above. The resulting graphs for these measurements are shown in Figures #26 and #27 below.

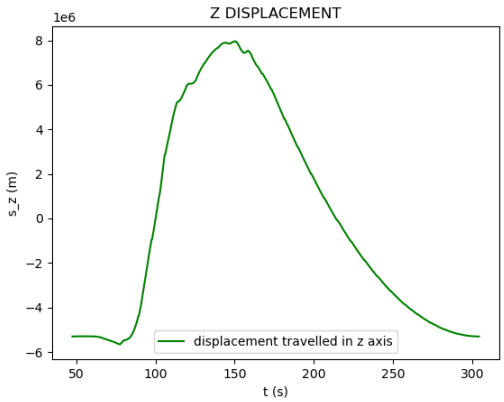

Figure #26: A graph where the displacement of the drone in the z-direction is portrayed after Fourier Transformations are conducted on the data.

As shown in Figure #26, a range of -6×10^6 meters to 8×10^6 meters arose when the measured values of the z-direction displacement were plotted. Further, it is seen that takeoff of the drone occurred when $s\_z < 0$ as $t = 100$, indicating that the test flight path of the drone was not represented accurately. However, a better representation of the test flight path in the z-direction is shown from the general shape of the above graph. This is because little to no change in the displacement in the z-direction is seen between $t = 0$ and $t = 100$, showing that the drone is stationary during this time period. Then, the peak of the graph was reached at $t = 160$, which is representative of the maximum height, 33 meters, being reached by the drone at a similar time. But the descent of the drone at $t = 210$ and the second takeoff of the drone at $t = 230$ cannot be seen from the displacement in the z-direction in Figure #26. This was because the frequencies produced by the quick landing and subsequent takeoff had not appeared for a significant amount of time. Thus, the Inverse Fourier Transform was calculated such that drone went through a constant descent from $t = 210$ to $t = 230$ when in reality, that was not the case.

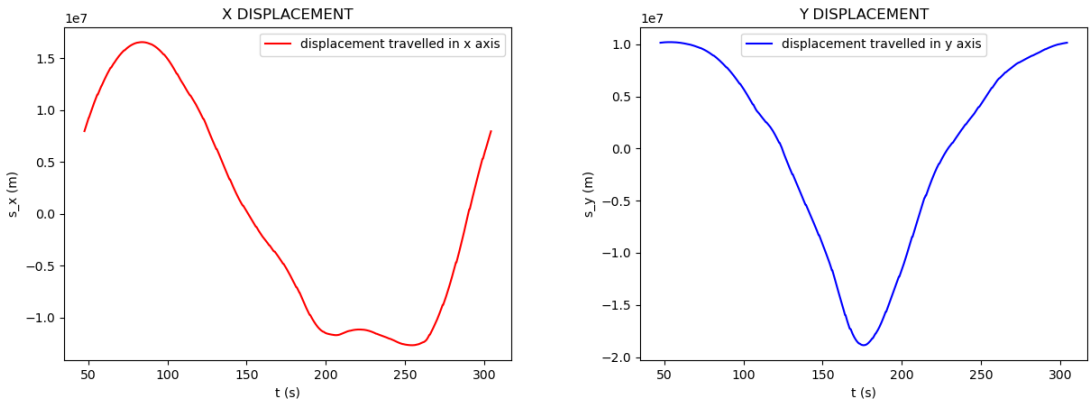

Figure #27: Graphs where the displacement of the drone in the x-direction and the y-direction are portrayed after Fourier Transformations are conducted on the data.

Unlike the displacement graphs in the z-direction, however, the test flight path is not portrayed by any feature of the displacement graphs for the x and y-direction in Figure #27 above. Firstly, there should have been no change in displacement in the y-direction. Secondly, a displacement of roughly 2 meters was shown in the x-direction from t = 230 onwards when in reality, a displacement of 24 meters should have been shown. The errors found in Figure #27 could be attributed to the effect of the shaky motion of the drone on the acceleration values recorded by the accelerometer in the LSM9DS1 breakout board.

On the one hand, it is important to note that the mathematical equations for the Fourier Transformation used by the group to calculate the displacement of the drone was taken from an integral with bounds from $-\infty$ to $+\infty$. On the other hand, a discrete Fourier Transformation is carried out by the relevant libraries on Python. A convolution of the signal produced by the LSM9DS1 breakout board and the time interval is taken to create a window function as shown in the Fourier Transformation from Equation (12) below. The window function is a function assumed by Python based on the number of data points from the LSM9DS1 and the DAQ code.

$$a_{dft}(w) = W(w)\tilde{a}(w) \qquad (12)$$

Here, the acceleration in terms of the angular frequency after a discrete Fourier Transformation is represented by the function $a_{dft}(w)$ and the window function is represented by the function $W(w)$. Consequently, the acceleration in terms of angular frequency after a regular Fourier Transformation is represented by the function $\tilde{a}(w)$.

Due to this window function, the values at the start and end time become undefined affecting the shape of the graph and which might also affect the magnitudes of values of the graph and hence produce a graph which does not accurately represent the actual flight path.

Drone Altitude Calculation Using the Barometric Formula

The drone's displacement in the z-axis, otherwise known as the altitude, can be accurately calculated using the Barometric Formula, shown in Equation (13) below. This formula takes the pressure and temperature of the surrounding environment into account when calculating the altitude. When viewing Equation (13), it is essential to note that the recorded atmospheric pressure by the DPS310 sensors is represented by the variable P_h. Additionally, the atmospheric pressure at sea level, 101.325 kPa, is represented by the variable P_0 (Encyclopedia Britannica, 2021). Further, the mass of the surrounding air is represented by the variable m, the acceleration due to gravity, 9.81 m/s^2, is represented by the variable g, and Boltzmann's constant, 1.38e23 J/K, is represented by the variable k. Ultimately, this means that the temperature of the surrounding air must be represented by the variable T, and the altitude is represented by the variable h (Nave, 2011).

$$\frac{P_h}{P_0} = e^{\frac{-mgh}{kT}} \qquad (13)$$

Upon further observation of Equation (13), a clear relationship between the recorded pressure and temperature by the DPS310 sensors can be seen. Ultimately, when the recorded temperature is increased over time, an exponential increase in the recorded pressure is found. As a result, when the recorded temperature is decreased over time, an exponential decrease in the recorded pressure is found. Moreover, Equation (13) could be further simplified by using the equation shown in Equation (14) below, where the air density at sea level, 1.225 kg/m^3, is represented by the variable ρ_0 (Nave, 2011).

$$\rho_0 = \frac{mP_0}{kT} \rightarrow \frac{\rho_0}{P_0} = \frac{m}{kT} \qquad (14)$$

Using Equation (13) and Equation (14), the Barometric Formula can be derived as the equation shown in Equation (15) below.

$$\frac{P_h}{P_0} = e^{\frac{-\rho_0 g h}{P_0}} \qquad (15)$$

Equation (15) can be altered further to make the altitude variable, h, the subject of the equation. Hence, the Barometric Formula is the equation shown in Equation (16) below (Nave, 2011).

$$h = \frac{-P_0 \times \ln\left(\frac{P_h}{P_0}\right)}{\rho_0 g} \qquad (16)$$

Importantly, when collecting data, it was found that P_0 was not entirely accurate in Equation (16) as it assumes sea level at 0 meters. However, our data collection process was conducted at a position above sea level, so Figure #_ had to be modified by subtracting the ground level of our location, written as the offset C. This way, the ground level of our location was now considered 0 meters by the DAQ code when the drone was stationary on the ground. Ultimately, it meant that the equation used to find the altitude of the drone from the data collected by the DPS310 sensors is shown in Equation (17) below.

$$h = \frac{-P_0 \times \ln\left(\frac{P_h}{P_0}\right)}{\rho_0 g} + C \qquad (17)$$

Using Python, the array of recorded pressure values was converted into altitude values, and the offset for both DPS310 sensors is calculated in lines 78 and 79. Additionally, the Python-based function that inputs the recorded pressure values and the offset to calculate the altitude can be seen in lines 31 to 37. All of this information is shown in Figure #28 below.

```
30
31  def alt_calc(data,offset,p_0=101325,g = 9.81,rho_0 = 1.225):
32      arr2 = []
33      k1 = -p_0/(g*rho_0)
34      k2 = offset
35      for i in range(len(data)):
36          arr2.append((k1*np.log(data[i]*1000/p_0))+offset)
37      return arr2
```

```
77
78  dps1_offset = (101325/(9.81*1.225))*np.log(np.mean(dps_pres_1[:20])*1000/101325)
79  dps2_offset = (101325/(9.81*1.225))*np.log(np.mean(dps_pres_2[:20])*1000/101325)
80
81  alt1 = alt_calc(dps_pres_1,dps1_offset)
82  alt2 = alt_calc(dps_pres_2,dps2_offset)
```

Figure #28: A Python code snippet where the array of recorded pressure values is converted into altitude values using the calculated offset.

The obtained altitude values were then graphed using the "Matplotlib" library in Python using the code above. The graph, however, can be seen in Figure #29 below.

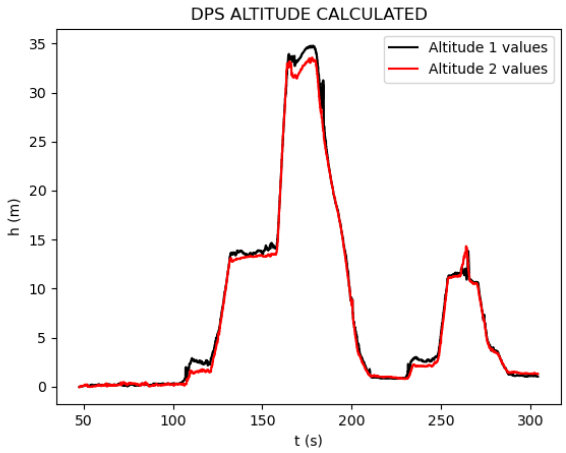

Figure #29: A graph where the altitude values recorded by two DPS310 sensors are compared

From Figure #29, one can observe that the flight path of the Full Flight Test is accurately depicted by the calculated altitude values. The corresponding plot of the difference between the two altitude values from the two different DPS310s shows the level of disagreement. The large spikes in the altitude difference, like in the raw pressure data, correspond to when the drone was in windy conditions or moving quickly, as explained through Bernoulli's Principle. Additionally, as shown in the Propeller Test, the DPS310 mounted on the body of the drone will report a lower pressure (and therefore a higher altitude) than the DPS310 mounted at the top of the pole. As such, the approximate difference in altitude between the altitudes recorded by both DPS310 sensors are shown in Figure #30 below.

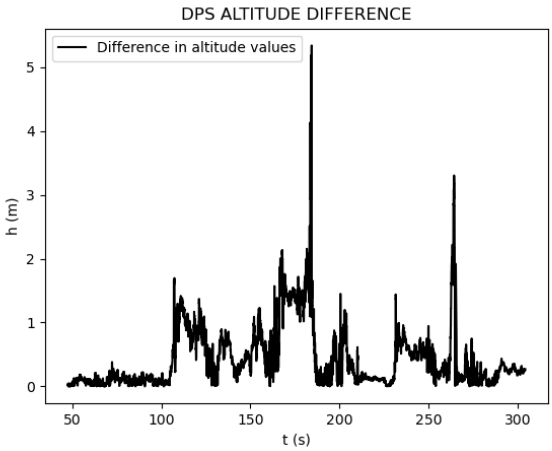

Figure #30: A graph where the difference in altitude values recorded by the DPS310 sensors are plotted

For each time window where the drone hovered at a particular height, the average altitude was calculated between the measurements of both DPS310s. The difference between the average altitude calculated by the pressure method and the actual altitude measured by the drone controller is shown in Table #2 below.

DPS1 (m)	DPS2 (m)	Average (m)	Actual (m)	Separation/Error
13.85	13.32	13.58	13	+0.58m/4.462%
33.90	32.58	33.24	33	+0.24m/0.727%
11.34	11.5	11.42	10	+0.42m/3.818%

Table #2: A table where the difference between the average altitude calculated between both DPS310 sensors and the actual altitude measured by the remote controller of the drone.

As a result, it can be seen that an average error of 3.67% is found, as expected due to the precision of the DPS310. Considering that the mechanism that the drone controller uses to report the data is currently unknown, and its precision only goes to 2 digits, it could even be the case that the pressure-altitude calculation method is closer to the real-life value.

Discussion

In our experiment, we used 3 different methods of evaluation to estimate the drone position: trapezoidal integration, Fourier transformation and barometric formula method. Out of all the methods used, it was found that estimating the altitude of the drone with DPS310 pressure data was more effective than the inertial navigation system. The average percentage error in altitude we calculated using the pressure values from the DPS310 was 3.67%, while the difference in position using the trapezoidal method of integration and Fourier transformation were incredibly huge, for example, the percentage error in the x displacement calculated using the trapezoidal method was 37400% and the distance values obtained from the Fourier transformation were in the magnitudes of 106 which is an even bigger error.

In this experiment, it was observed that acceleration data is much more unreliable than pressure. This, we think, is mainly because of the jerky motion of the drone and the wind which produces sharp spikes in acceleration data on all axis and these small spikes in acceleration accumulate exponentially because of the integral and give a big displacement value which is not close to the real displacement value. The pressure values aren't affected a lot due to the jerky motion of the drone and the change in surrounding pressure due to the propellors of the drone, but the only drawback is that one can only obtain altitude values (position in the z-axis).

Conclusion and Future Studies

In this lab, exploratory studies were conducted on a set of methods to derive the position and altitude of a drone in flight using only an onboard sensor package. A suite of inexpensive instrumentation controlled by a programmable microcontroller was flown onboard a drone on a series of systematic tests, the data for which was collected and analyzed after the fact. Altitude navigation through the Barometric Formula proved to be highly accurate due to the precision of the DPS310 sensors, but the integration of acceleration data failed to deliver reasonable position estimates. The nature of double integration over a set of discreet, error-prone, and erratic data points, whether by Trapezoid Rule or Fourier Transform, led to runaway velocities and unrecognizable positions. However, the process of inertial navigation, sometimes known as dead reckoning, has been successfully implemented in many contexts and there are plenty of opportunities for improvement in future studies.

For example, more focus would be placed on determining the position and altitude of the drone when in rotation and when making a turn. In a bank or turn, a change is made to the direction the LSM9DS1 is pointing, which means the acceleration data would no longer correspond directly to the cardinal axes the way this paper assumes it to be. The acceleration due to gravity would also need to be factored out in a significantly more complicated manner than just offsetting the z-axis like was done here. Rotation matrices and separate reference frames would need to be introduced when the calculations for position are derived. The applications of these concepts would be similar to those in studies such as Castro-Toscano et al. (2017) and Petritoli and Leccese (2021) and could significantly improve navigational accuracy for an object moving dynamically like the drone in this lab. Use of the gyroscope and magnetometer features of the LSM9DS1 would be essential.

Many small improvements to the current experimental setup and procedure could also be made. The LSM9DS1 breakout board could be wired to the microcontroller through an SPI protocol instead of the current I2C protocol. An SPI connection could allow the incoming acceleration data from the LSM9DS1 to be read significantly faster by the microcontroller. Hence, a higher number of data points could be used for more accurate data observations. Given enough data points, even the spasmatic vibrations caused by the drone in flight could be read with accuracy, leading to cleaner integrations for velocity and position. Additionally, more sophisticated use of the Fourier Transform methodology could be used to find and cut out unwanted frequencies of motion, such as the spin of the propellers or artefacts from the sampling rate of the testing apparatus. Some of these were tested by this group, but the results were incomplete and need more refinement of the methods to produce reliable data.

The applications of this technology are numerous, and likely are already in use, but applying the principles of classical mechanics in such a straightforward yet surprisingly complex way is still a fascinating exercise. Beyond the scope of this project, inertial navigation is still an incomplete problem. The “platonic ideal” of the method – reliable position data from only acceleration data, over a long period of time, with no external influence – is currently still unachievable. This project reveals on a small scale just how quickly the tiniest imperfections in instrument or algorithm can quickly send the position estimate racing away under integration, but it all begs the unanswerable question: How good, how precise, how minute, would the equipment - the accelerometer and the integration algorithm - really have to be to achieve true long-term inertial navigation? Under the principles of classical mechanics, a black box could be built, or at least imagined, that, without any knowledge but its own acceleration (and, of course, the initial conditions), be toured all around the world, the universe, and be able to say exactly how far it went and the exact path it took to get there.

References

Works Cited:

1. Castro-Toscano, M., Rodríguez-Quiñonez, J., Hernández-Balbuena, D., Lindner, L., Sergiyenko, O., Rivas-Lopez, M., & Flores-Fuentes, W. (2017, June 1). *A Methodological use of Inertial Navigation Systems for Strapdown Navigation task.* IEEE Conference Publication | IEEE Xplore. Retrieved April 22, 2022, from *https://ieeexplore.ieee.org/document/8001484/;jsessionid=9sNTJq2EUkh-eg7HoxbwgE3H-8upN1eAgpJNbkXXREDOiMMRgW3C!1032670278?tp=&arnumber=8001484&tag=1*
2. *Mavic 2 - DJI*. (2022). DJI. Retrieved April 15, 2022, from https://www.dji.com/mavic-2
3. Nave, R. (2011). *The Barometric Formula*. Georgia State University. Retrieved April 15, 2022, from http://hyperphysics.phy-astr.gsu.edu/hbase/Kinetic/barfor.html#c2
4. Britannica, T. Editors of Encyclopaedia (2021, January 11). *atmospheric pressure*. *Encyclopedia Britannica*. https://www.britannica.com/science/atmospheric-pressure
5. Sharma, M. (2020, December 13). *Trapezoidal Rule - C Program*. BragitOff.Com. Retrieved April 15, 2022, from https://www.bragitoff.com/2017/08/trapezoidal-rule-c-program/
6. Fried, L. (2017a, February 1). *Adafruit LSM9DS1 Accelerometer + Gyro + Magnetometer 9-DOF Breakout*. Adafruit Learning System. Retrieved April 15, 2022, from https://learn.adafruit.com/adafruit-lsm9ds1-accelerometer-plus-gyro-plus-magnetometer-9-dof-breakout
7. Rembor, K. (2005). *Overview | Adafruit DPS310 Precision Barometric Pressure and Altitude Sensor | Adafruit Learning System*. Adafruit Learning System. Retrieved April 1, 2022, from https://learn.adafruit.com/adafruit-dps310-precision-barometric-pressure-sensor?view=all
8. Fried, L. (2012, August 23). *Adafruit Ultimate GPS*. Adafruit Learning System. Retrieved April 22, 2022, from https://learn.adafruit.com/adafruit-ultimate-gps/overview
9. Petritoli, E., & Leccese, F. (2021, June 23). *Navigation Equations, Uncertainty and Error Budget in Inertial Navigation Systems*. IEEE Conference Publication | IEEE Xplore. Retrieved April 25, 2022, from https://ieeexplore.ieee.org/abstract/document/9511784
10. Fried, L. (2016, February 3). *Adafruit DS3231 Precision RTC Breakout*. Adafruit Learning System. Retrieved April 29, 2022, from https://learn.adafruit.com/adafruit-ds3231-precision-rtc-breakout
11. Fried, L. (2015, November 27). *Adafruit Feather M0 Basic Proto*. Adafruit Learning System. Retrieved April 29, 2022, from https://learn.adafruit.com/adafruit-feather-m0-basic-proto/overview
12. Fried, L. (2017b, November 8). *Adafruit BME680*. Adafruit Learning System. Retrieved April 29, 2022, from https://learn.adafruit.com/adafruit-bme680-humidity-temperature-barometic-pressure-voc-gas
13. Fried, L. (2013, July 31). *Micro SD Card Breakout Board Tutorial*. Adafruit Learning System. Retrieved April 29, 2022, from https://learn.adafruit.com/adafruit-micro-sd-breakout-board-card-tutorial

Acknowledgments

The entire group for this lab would like to thank Professor George Gollin and Shubhang Goswami for their insightful input in helping us complete this lab in a limited time. Whenever we needed help with many challenging concepts and methods to find the desired results, the two people mentioned above were the essential people to go to for help. It did not matter if the problem or question was hardware-related or software-related. Still, Gollin and Goswami provided precious assistance that put us in the right direction to answering our hypothesis for the lab. Also, both of them were able to keep the mood and the environment pretty light, so no one felt any pressure in trying to achieve and complete their assignment. In particular, however, we would like to thank Professor Gollin for taking the time to create the attachment device for us for the data collection of the drone's acceleration in the easiest way possible.