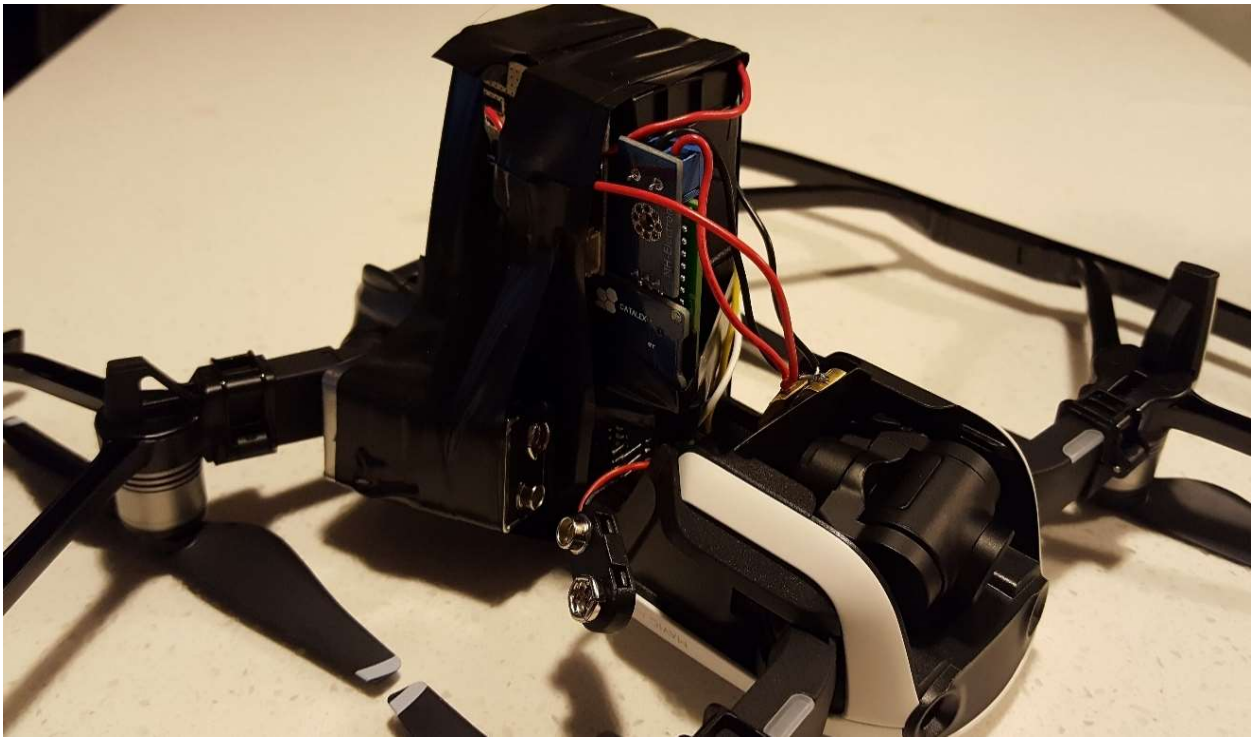
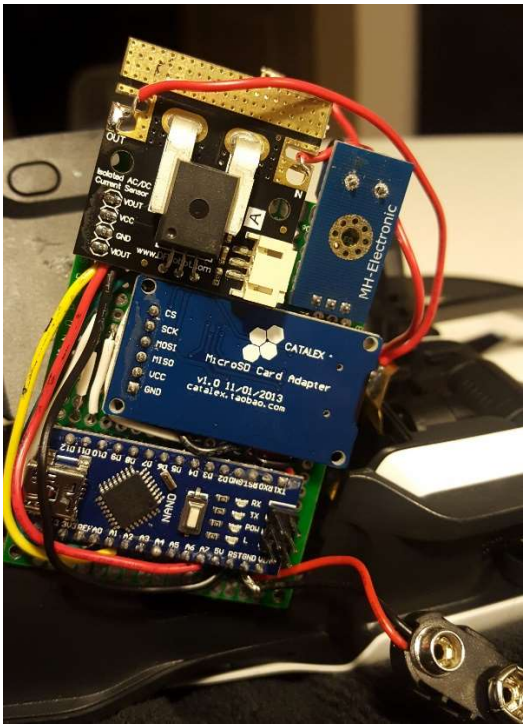
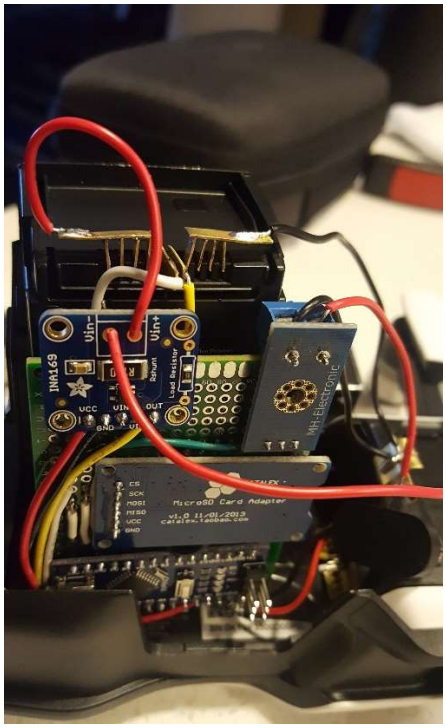
John Pace
Student ID: 105764757
CSCI 4739 - Senior Design II

The focus of this project has been to analyze, characterize, model, and predict the behavior of drones using externally collected data in conjunction with onboard sensor readings logged during flight. By analyzing certain characteristics of battery current draw and voltage readings, specific drone behaviors can be determined or predicted, which enables us to model location, improve performance, or detect abnormalities from external battery readings. The motivation for this stems from the use of drones in establishing temporary mobile networks. These networks require drones to be in specific locations to function, however in military or disaster scenarios traditional location services may not be available. The proposed solution for this is to explore if a drone's position can be estimated solely through information collected from its battery. This project extends a previous study that focused on collection and analysis of battery metrics from drones.

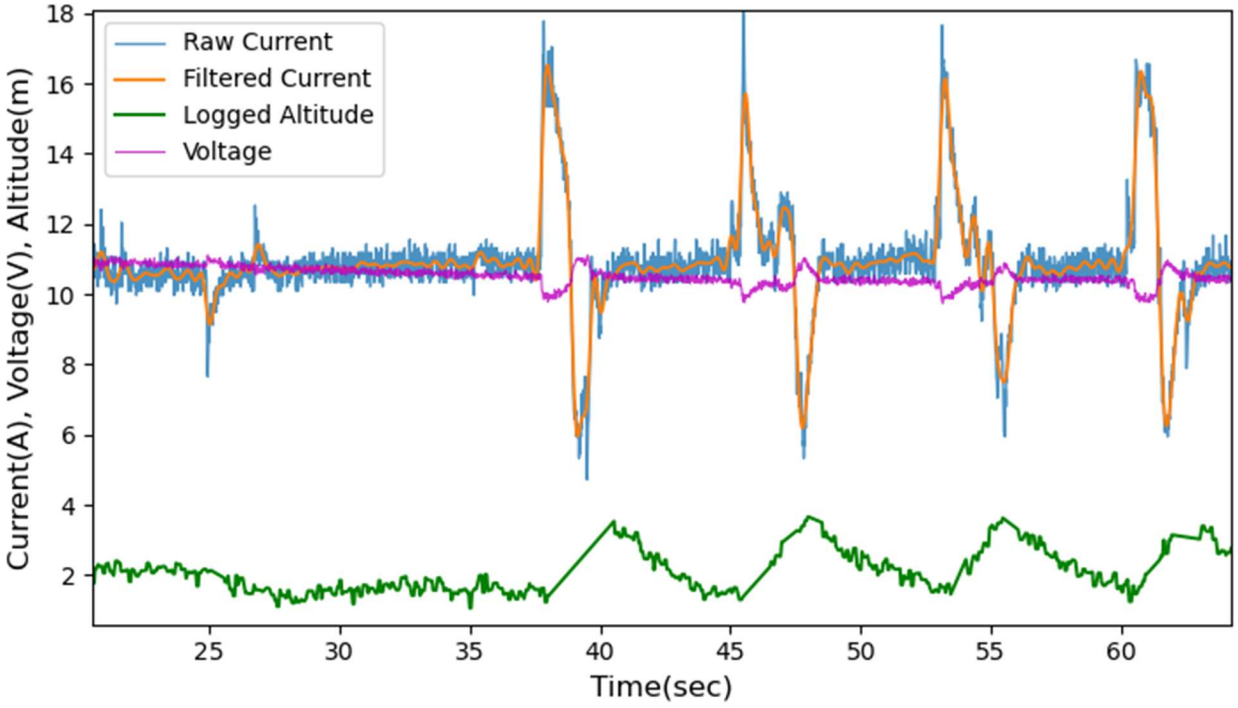
The drone used in this project is the DJI Mavic Air, a mid-range portable consumer drone which was chosen due to its real-world popularity and available SDK. A variety of onboard sensor readings are available from this drone, including individual motor rpm, current draw, acceleration, velocity, gyroscope readings, and many more. These readings are stored locally on the drone during flight and can later be downloaded and decrypted as a .csv file. Throughout this project these onboard readings will be cross-referenced and analyzed in conjunction with battery metrics logged externally by a microcontroller-based system. This Arduino-based external microcontroller taps into the existing battery pinout of the drone to independently measure the voltage and current draw of the drone. It then logs these readings and transmits them to a program that handles live predictions.



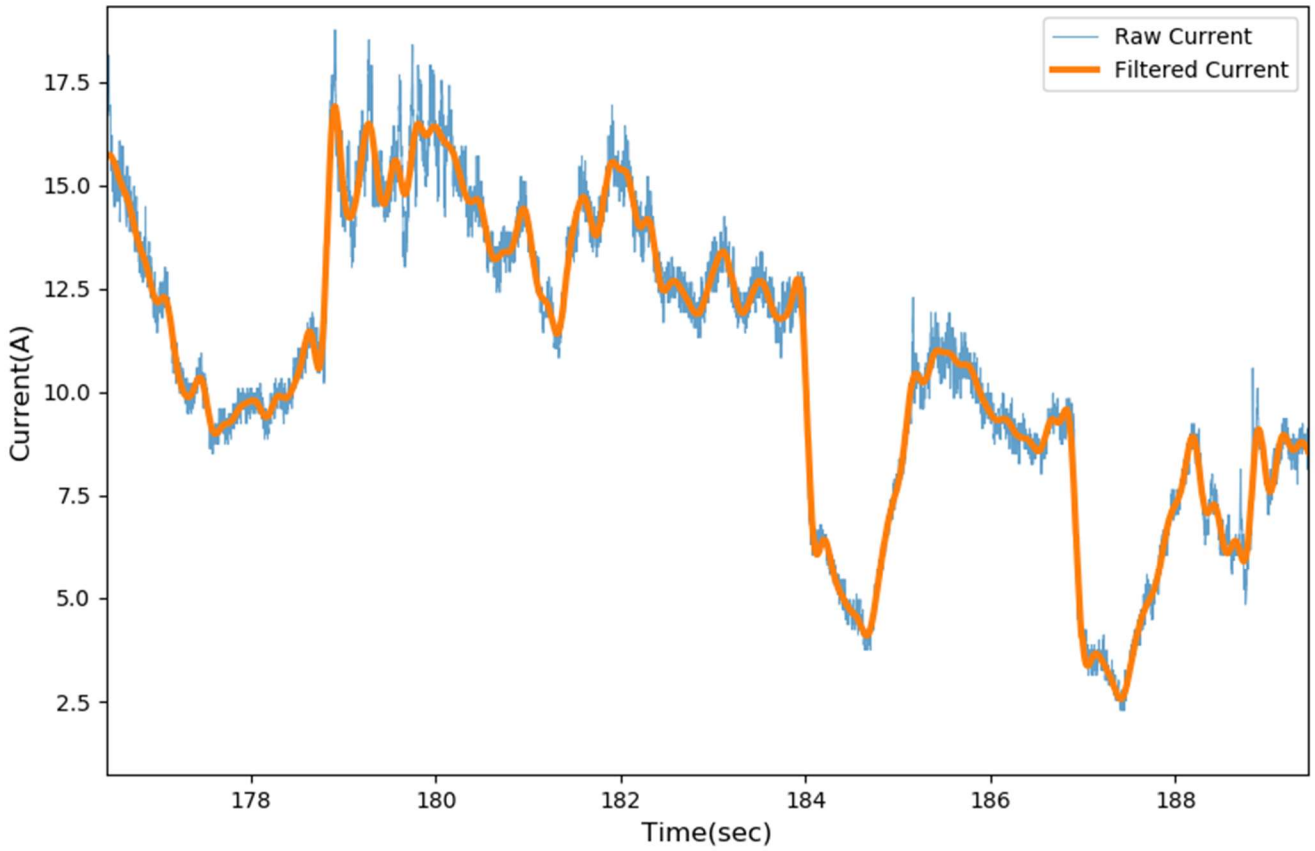




This system is required both to validate internal readings and to provide a higher granularity of data than the onboard flight logs. The onboard current readings are taken at a rate of 10hz, while the external device captures current readings at up to 400hz. This data frequency is required for proper detection of the nuances within the current trace that define certain maneuvers during flight. This external system accurately measures and logs cumulative current and voltage of the drone during flight and has been cross referenced with the drones own internal flight logs for accuracy. A current and voltage trace measured by this system with the drone's measured altitude is shown below:

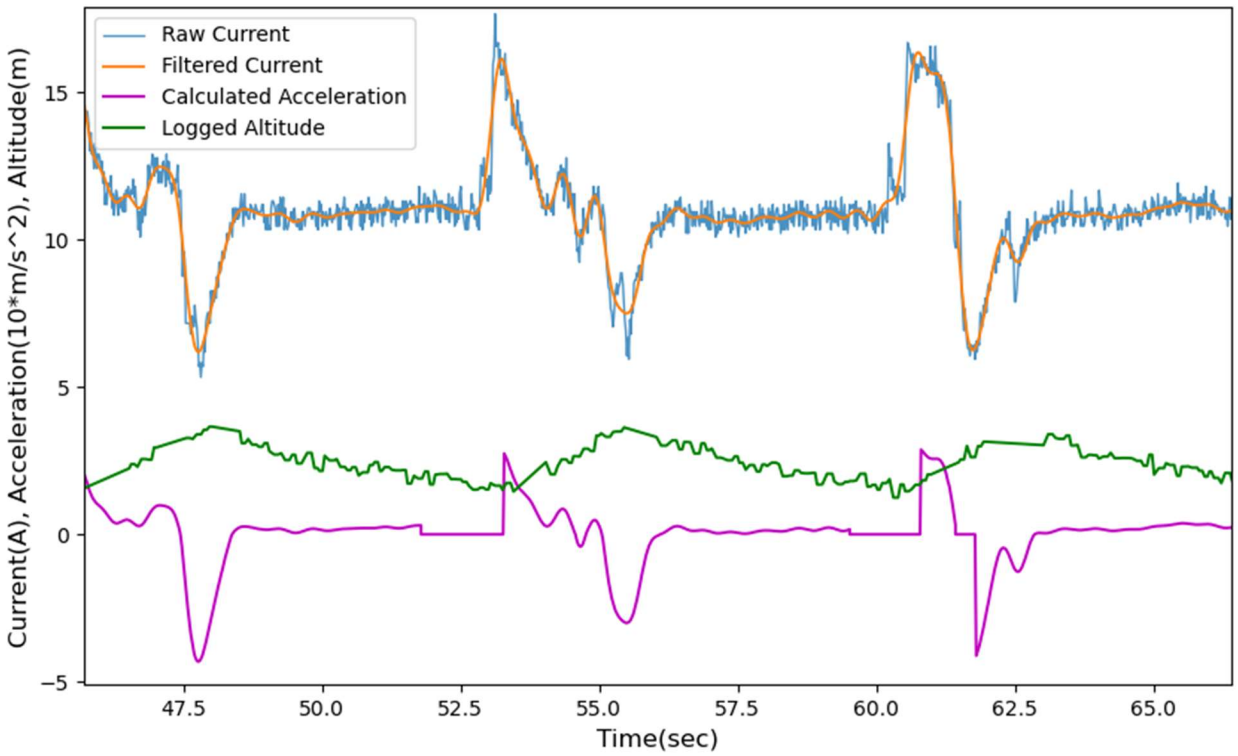


One brief note on this data is the amount of noise within the raw current trace. Because classification of what maneuver or 'state' the drone is in is done based on peaks and craters in the current trace, I applied a Butterworth lowpass filter to the raw current data. This gave a smoother fitted line to the raw current data, and it is this filtered current trace that is used for defining states and calculating acceleration. An example of the filtered line compared to the raw current trace:



Analysis of this filtered data allowed me to determine characteristics and traits of the battery current trace that are specific to certain flight patterns. Using these unique characteristics, algorithms were developed to detect what flight state the drone is in. Further analysis of filtered the battery current trace in conjunction with onboard sensor data allowed a relationship to be found mapping cumulative battery current to vertical drone acceleration. Using the detected flight state and calculated acceleration of the drone an estimation of the drone's altitude can then be made.

There are four primary flight states to be considered in estimating altitude: takeoff, hovering, ascending, and descending. After characterizing the unique current traits of each flight state, a program was developed to classify these states based on the externally collected battery data. This program uses a moving time window that holds a feature set over the previous two seconds of received battery metrics. This feature vector stores all voltage and current readings within the time period, all peaks and craters in the current trace, the current drone flight state, timestamps of the previous drone flight state, along with calculated acceleration, velocity, and altitude values. Peaks and craters in the current trace refer to relative maximums and minimums in the wave pattern when viewing a plot of the current trace, or when the derivative is zero.



The majority of the work done by this program is performed in a large while loop that constantly receives battery readings transmitted from the microcontroller and does the necessary calculations. After establishing a Bluetooth connection and setting up all necessary data structures, readings are received, decoded, and converted from raw analog values to the appropriate current and voltage values with the following code:

```
while True:
    data = sock.recv(1024)
    string += str(data.decode('utf-8'))
    data_end = string.find('\n')
    if data_end != -1:
        string = string[:data_end]
        vals = string.split(", ")
        if len(vals) > 2:
            if(vals[0] is not ''):
                time = ((float(vals[0])) / 1000)
            else:
                time = prev_time+0.03

            voltage =  (int(vals[1]) * 5 / 1024) / (7500 / (37500))
            current = (((int(vals[2])) - 510) * 5 / 1024 / 0.04 - 0.04)

            if len(buffer) > 68:
                total = total - buffer[0][1]
                buffer = buffer[1:]
            buffer = np.vstack((buffer, np.array([time, current])))

            total = total + current
            avg = total / len(buffer)
```

The remainder of this loop focuses on state detection, acceleration mapping, and altitude estimation. The takeoff sequence is defined by a time period in which the earliest current peak is between 40% and 60% of the expected current average of the hovering state based on voltage, and the following peak is between 90% and 140% of this average. The time difference between these two peaks must be between 2.5 and 3.2 seconds. Having the correct peak pattern is not enough to fully determine the current pattern of a takeoff sequence, so the current craters over this same time period are also checked to ensure they too are within specific current ranges and within the correct time window based on the timestamp of their preceding peaks. This automated procedure is hardcoded in the drone's internal software, making it very specific and without much variance, so detection of takeoff is simply a lengthy series of if statements.

```
# takeoff sequence detection
if 4 < avg < 8 and state == 0:
    if (1 < len(peaks) < 4) and (2 < len(craters) < 5):
        if (4 < peaks[0][1] < 6) and (10 < peaks[1][1] < 14):
            if (2.5 < peaks[1][0] - peaks[0][0] < 3.2):  # peaks pass
                if (1.2 < craters[0][1] < 2) and (1.2 < craters[1][1] < 2.2):
                    if (1 < craters[1][0] - craters[0][0] < 1.8):
                        if (0.1 < craters[2][0] - peaks[1][0] < .5):
                            state = 2  # drone in takeoff.
                            takeoffStart = craters[0][0]
                            takeoffEnd = craters[2][0]
                            peaks = [[0, 0]]
                            craters = [[0, 0]]
                            maxPeak = 0
                            minCrater = 0
                            del buffer
                            buffer = np.zeros((1, 2), dtype=int)
                            buffer[0] = np.array([time, current])
                            total = total + current
                            avg = total / len(buffer)
                            alt.append([time, 2])
```

More interesting are the hovering, ascending, and descending states. The hovering flight state is defined as a time period in which the maximum current peak and the minimum current crater are within 20% of the predicted idle current based on voltage, and the drone remains classified as in this state so long as these conditions are valid. A 20% margin in current is required as the current values during idle hovering of the drone may fluctuate by up to 14% of the idle current every hundredth of a second.

Because we are only considering vertical positioning, once a period of hovering has ended the drone is known to be either ascending to descending. Ascensions begin with a sharp spike in current, as much as 190% of the idle average or 19A, in order to accelerate the drone upward and remain at an elevated level while the drone is still climbing. Before the highest altitude of the climb is reached however, the current has a sharp drop to as low as 50% of the idle average. This is due to the velocity of the drone continuing to carry it upward even after removing all throttle input. The highest point of the climb can then be found by the lowest crater in the current trace following an ascension, and it is at this point, when the drone begins moving downward, that all motors must increase in rpm to compensate for this change in direction and control the descent.

The current pattern for descending is much more subtle than ascension. Descent is characterized as beginning at the lowest crater following a decrease in current greater than 20% the current average, followed by an average current level between 60% and 95% of the anticipated average. This period ends when a small peak above the anticipated average current is seen and the average current level returns to normal.

During periods of ascending and descending, current can be mapped to a preliminary acceleration by subtracting a specified offset from the current value and dividing by the average current over the time period. This preliminary acceleration is then scaled, relative to the adjusted current reading, to a final acceleration value.

After calculating an acceleration value based on the current feature set, the drone's altitude is estimated by using this acceleration along with the sample frequency of battery readings to determine velocity and then position.

Inside the main loop, the following code block does all of the state detection, acceleration mapping, and altitude prediction for hovering, ascending, and descending:

```
# Current mapping for ascending/descending
if state == 3:
    accelMapping = ((current - .505) / 10)
    if (accelMapping >= 1):
        accelMapping = (accelMapping * (((1 + (1 / accelMapping)) / 2)))
    else:
        accelMapping = accelMapping / 1.1
    accelMapping = accelMapping - 1

    accel.append([time, (accelMapping)])
    if not descending and minCrater < 8:
        descending = True
        ascending = False
        print("Descending at ", min(craters, key=lambda x: x[1]))

    timePeriod = time - prev_time
    veloc.append([time, ((accel[len(accel) - 1][1] * timePeriod) + veloc[len(veloc) -
1][1])])
    alti = (alt[len(alt) - 1][1] + ((veloc[len(veloc) - 1][1]) * timePeriod) + 0.5 *
(
                accel[len(accel) - 1][1] * (timePeriod) ** 2))
    if alti <= 1:
        alt.append([time, 1])
    elif alti >=3:
        alt.append([time, 3])
    else:
        alt.append([time, alti])
    if 10.2 < avg < 11.2 and (maxPeak - minCrater < 2):
        print("-----------------CLIMB ENDED: {}".format(craters[len(craters) -
1][0]))
        ascending = False
        descending = False
        veloc.append([time, 0])
        alt.append([time, alt[len(alt) - 1][1]])
```

```
if 10.2 < avg < 11.2 and state != 2:
    if (maxPeak - minCrater < 2):
        print("-----------------HOVER START: {}".format(
            max(craters[len(craters) - 1][0], peaks[len(peaks) - 1][0])))
        if state == 0:
            alt.append([time, 2])
        else:
            alt.append([time, alt[len(alt) - 1][1]])
        veloc.append([time, 0])
        accel.append([time[i], 0])
        state = 2

elif state == 2:
    veloc.append([time, 0])
    accel.append([time[i], 0])
    alt.append([time, alt[len(alt) - 1][1]])
    if (maxPeak - minCrater > 2 and minCrater != 0):
        # print("-----------------HOVER END: {}".format(time[i]))
        print("-----------------HOVER END: {}".format(craters[len(craters) - 1][0]))
        climbStart = time
        print("-----------------CLIMB START: {}".format(climbStart))
        ascending = True
        state = 3
```

Most importantly throughout this main loop is constant maintenance of the time buffer along with updated tracking of all peaks and craters within the current trace for the current time period. This is done with the following:

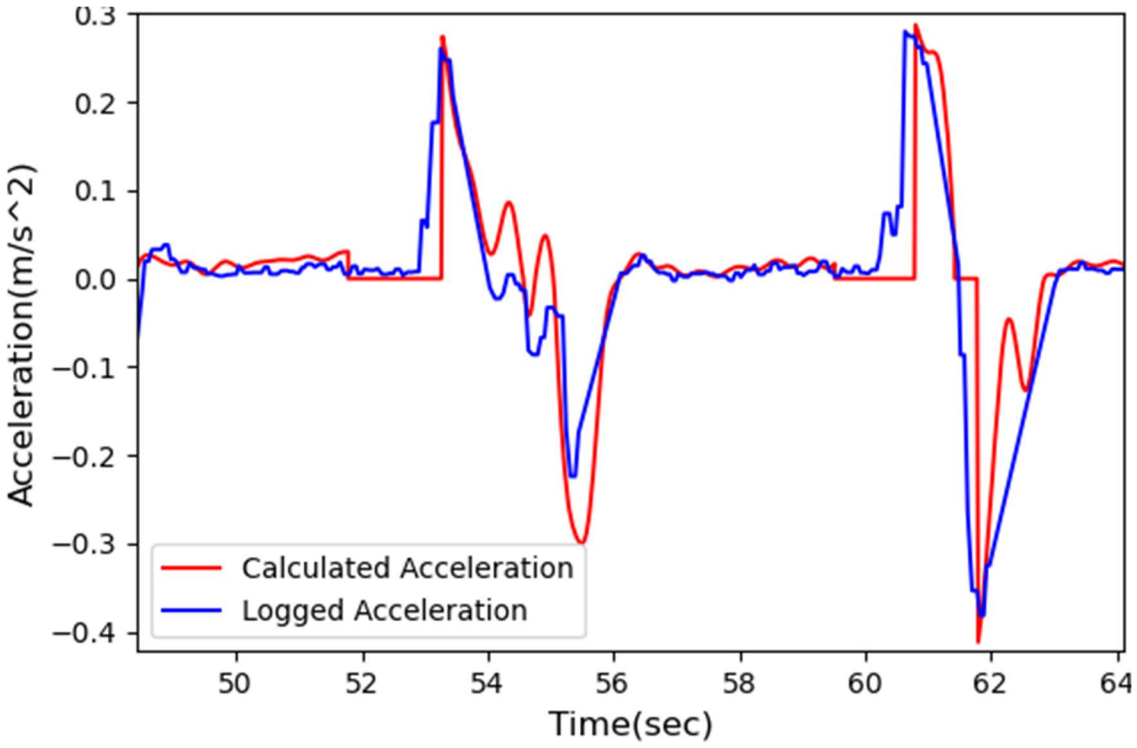
```
# remove peaks/craters outside of time window
if ((time - float(peaks[0][0])) > 2):
    if len(peaks) > 1:
        # peaks = peaks.remove(0)
        del peaks[0]
        maxPeak = max(peaks, key=lambda x: x[1])[1]
if ((time - craters[0][0]) > 2):
    if len(craters) > 1:
        del craters[0]
        minCrater = min(craters, key=lambda x: x[1])[1]

# detect peaks/craters
if rising:
    if prev_current < current:
        rising = True
    elif abs(craters[len(craters) - 1][1] - prev_current) > 0.4:
        peakFound = True
        peaks.append([prev_time, prev_current])
        maxPeak = max(peaks, key=lambda x: x[1])[1]
        rising = False
else:
    if prev_current > current:
        rising = False
    elif abs(peaks[len(peaks) - 1][1] - prev_current) > 0.4:
        peakFound = False
        craters.append([prev_time, prev_current])
```

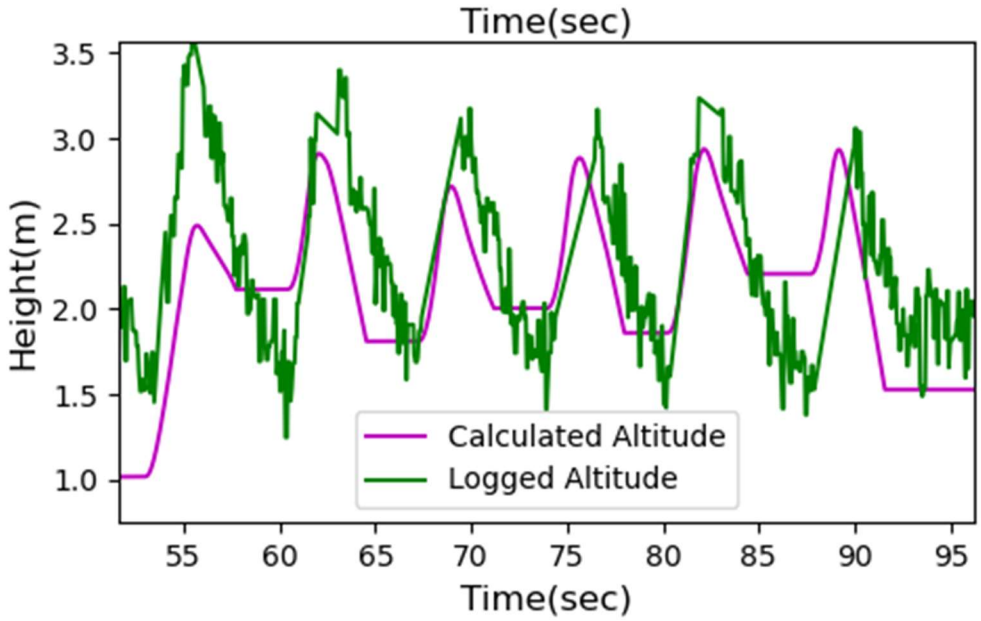
```
        minCrater = min(craters, key=lambda x: x[1])[1]
        rising = True


if prev_current <= current and not peakFound:
    possibleMax = current
elif prev_current >= current and peakFound:
    possibleMax = current
```

It is hard to determine exactly how accurate the results of this are, as the only data available to reference these predictions currently are the sensor logs taken from the drone itself, which are not perfectly accurate. Regardless, the acceleration values measured by the drone's own sensors and those calculated from my program using the externally collected battery metrics are in line with each other, following the same trends and magnitudes.



More difficult to determine is the accuracy of the altitude predicted from this calculated acceleration. Again, both the altitude predicted from my program using battery metrics and the altitude as measured from the drone itself follow similar patterns, however the drone's own internal altitude measurement is not perfectly accurate. The graph below plots these two altitudes for a series of repetitive, consistent, and smooth maneuvers of ascending and descending, however the drone's internal readings show a large amount of noise and fluctuation that did not exist in the original flight. The true altitude is likely somewhere between these two plots.



Currently my program is able to accurately classify the drone state as being in takeoff, hovering, ascending, and descending based on battery metrics. Using externally measured battery current alone, the program is also able to calculate both acceleration and altitude values accurate to the internal sensor readings taken from the drone's flight logs. This is all done in real time while the drone is in flight, using transmitted battery readings from the microcontroller attached to the drone. The predicted results for altitude are not yet accurate enough for real-world applications, but they do follow the drone's flight pattern and demonstrate enough accuracy to justify further work on this study. Moving forward the largest hurdle to overcome would be accounting for large current fluctuations due to wind and other external factors, along with expanding this methodology to define horizontal position as well by examining the current trace of each individual motor.

Hardware Used:

Arduino Nano

Arduino voltage sensor

Micro sd card module

Bluetooth module

32gb micro sd card

INA169 current sensor

50A SEN0098 current sensor

DJI Mavic Air drone

Software used:

C++ on Arduino for the external logs.

DatCon Extract DJI to extract onboard drone logs into a .csv file.

Python with matplotlib, numpy, pandas, and scipy libraries for data analysis.

TKinter for real-time GUI.