**Week:** 16

**Date:** 12/12/2022  
**Total hours:** 30+

**Key Points**

I’m not sure it was clear in the last progress report, but last week was the drone’s first flight, and subsequent tuning of the IMU signal processing chain and the PID loops.

Most of this involved setting the correct gains on the accelerometer noise estimation, changing the decimation factor and low pass filter bandwidth for the motor output, and drastically increasing derivative feedback in the PID loops.

After flight tests showed the drone to be very controllable and quite stable, I moved on to implementing auto-throttle, where the drone will maintain a precise height off the ground using LiDAR, without input from the pilot. This took a couple days due to the difficulty and danger of testing the system, as well as improper tuning of the PID loop. After figuring out some filtering strategies to handle the low update rate of the LiDAR data impacting the derivative feedback to the PID loop, and some additional PID tuning, the PID control loop now easily and quickly locks onto the height desired after a switch is thrown on the remote.

In terms of autonomous functionality, I mounted the PI Zero onto the drone, created the connector solution from the Pi to the flight controller, and assisted my team members who were responsible for the autonomous functionality in building the LiDAR wire harness, soldering the harness to the Pi, testing it, and configuring the Pi for use. I also implemented an automatic landing system to land the Pi once the target object is found, however this we are unable to test with the propellers on at the moment.

In totality, I am very happy with the performance of the flight controller on the drone, although I would really like to spend more time tuning the signal chain and utilize the second IMU to really dial in the stability and accuracy of the attitude estimation algorithm.

I am not so happy with the navigation system, as it was implemented too late to be properly tested, integrated, and debugged, although more progress was made on it than I feared. Altogether, it demonstrates that the system foundation works very well as a whole, and is a great starting point for more interesting development using the STM32 flight controller + Pi nav computer system. I am excited to try implementing the system on a smaller drone platform, as well as trying to put a GPS and additional electronics on the larger drone, as it has so much thrust to spare. I also want to analyze the flight data from the drone and see if any additional kinematic information can be estimated using the IMU to assist with flight, such as tracking velocity, enhancing hovering ability, etc.

**Auto-throttle**

In order to implement auto-throttle, or the ability to maintain a dictated height off the ground autonomously, I needed to integrate Santiago’s I2C Lidar code, mount the LiDAR, write the control loop, and debug any issues I came across.

First, I and Santiago integrated the LiDAR code that he’d written and merged with my flight controller code, and verified that the system still booted and ran properly with the I2C enabled, but without making any calls to the LiDAR driver.

Next I installed the LiDAR, which entailed soldering the LiDAR connector on the PCB, mounting the LiDAR, and routing cable to it in such a way that the LiDAR would be removeable, but that the cables couldn’t be sliced by the motors. I also had to create a connector on the flight controller so that the entire cable could be disconnected if necessary.

A picture containing electronics, circuit

Description automatically generated

Fig. 16- . New I2C connector implemented

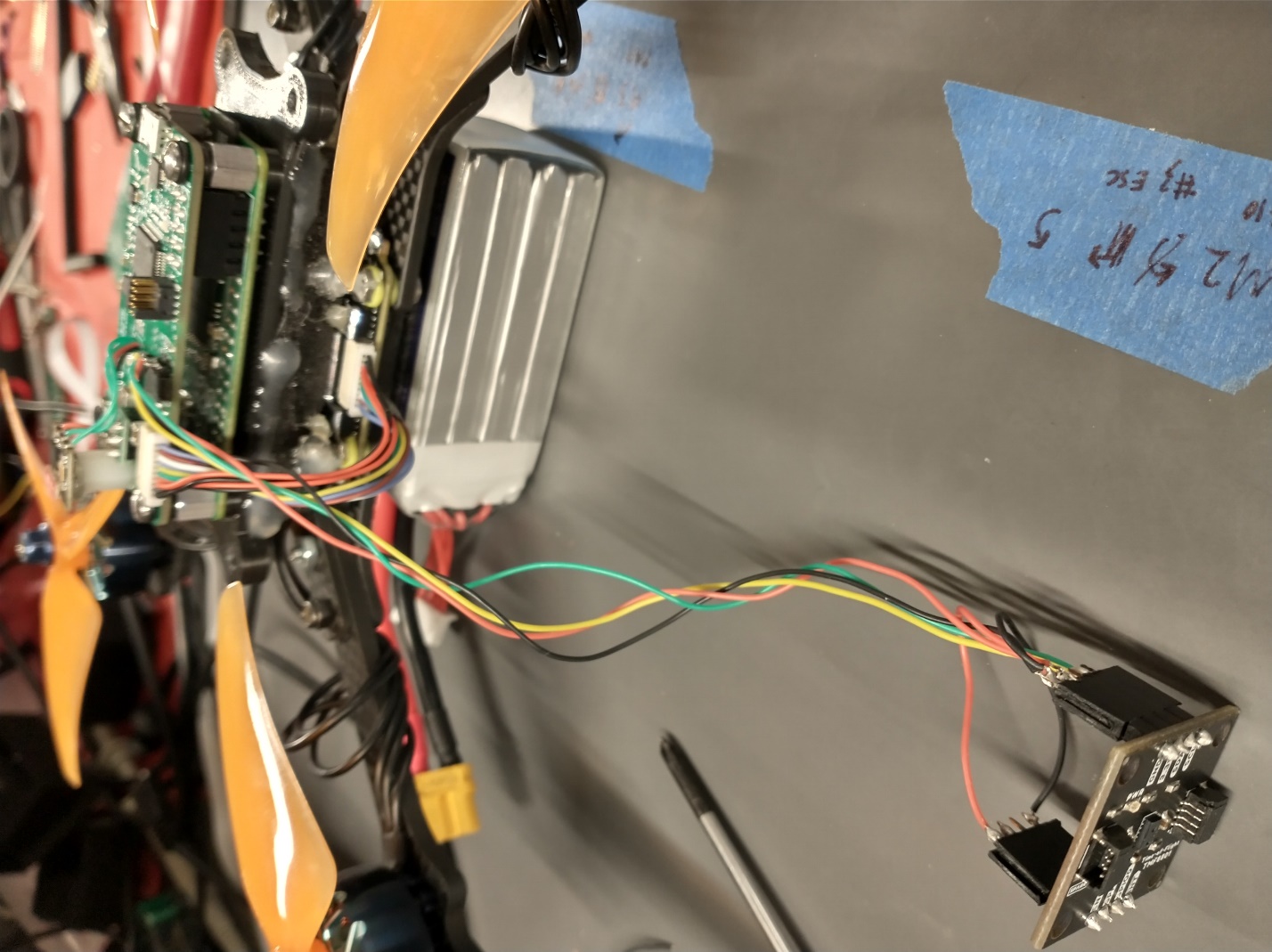


Fig. 16- 2. LiDAR wiring to flight controller

After completing the wiring to the flight controller, I mounted the LiDAR on one of the outer arms of the drone, with a piece of foam in between. I wrapped the communication cabling around the arm to take up all the slack and prevent it from bouncing up into the rotors.

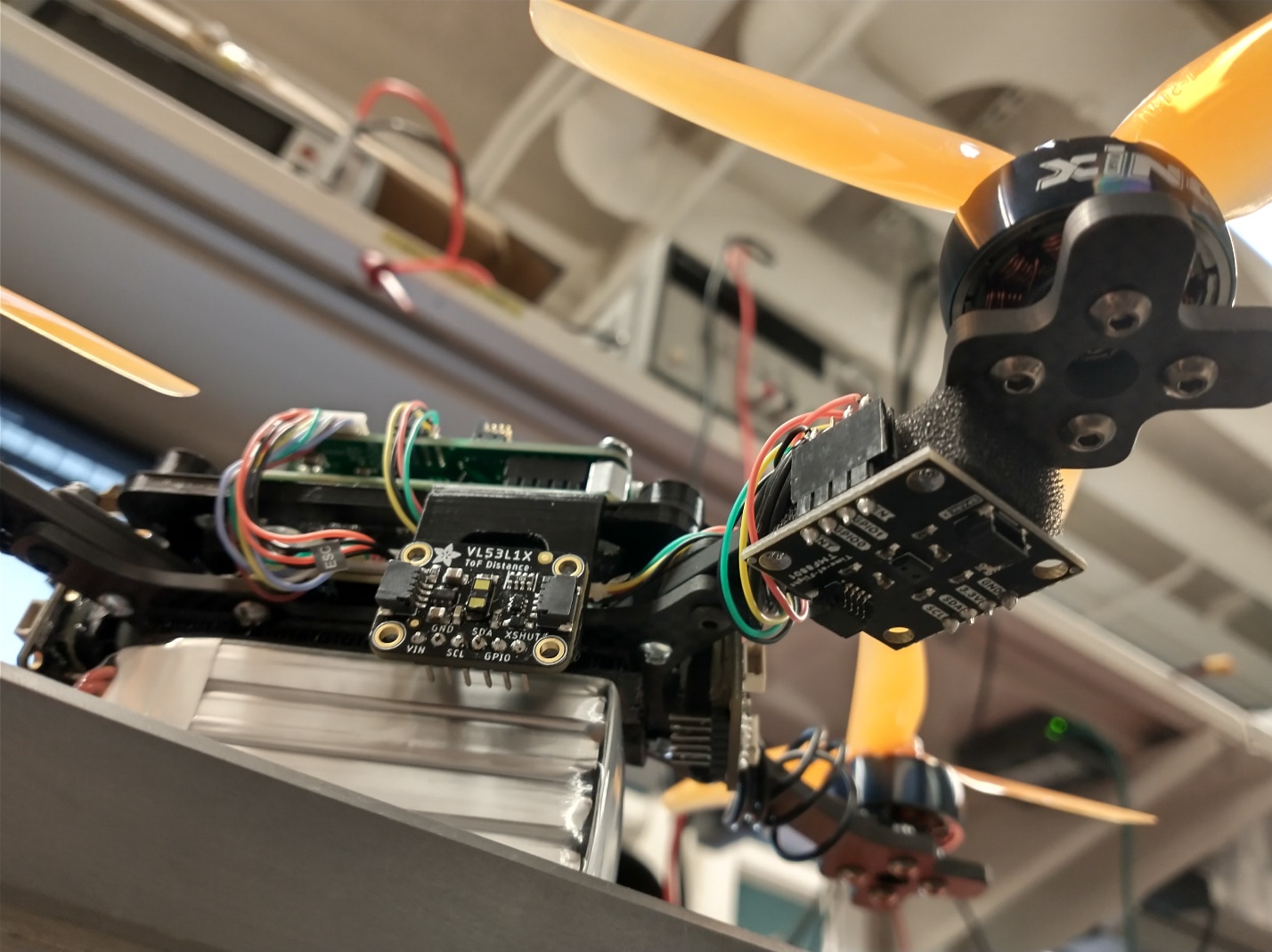


Fig. 16- 3. Downward facing LiDAR mounted and ready

I then tested the newly mounted LiDAR running in the main loop, which successfully showed the LiDAR data streaming in. Interestingly, if the sensor detects nothing, it returns a distance of zero, rather than its max distance of 600mm. To make the sensor work with my coming control loop, I added some quick logic to change this 0 distance to 600mm.

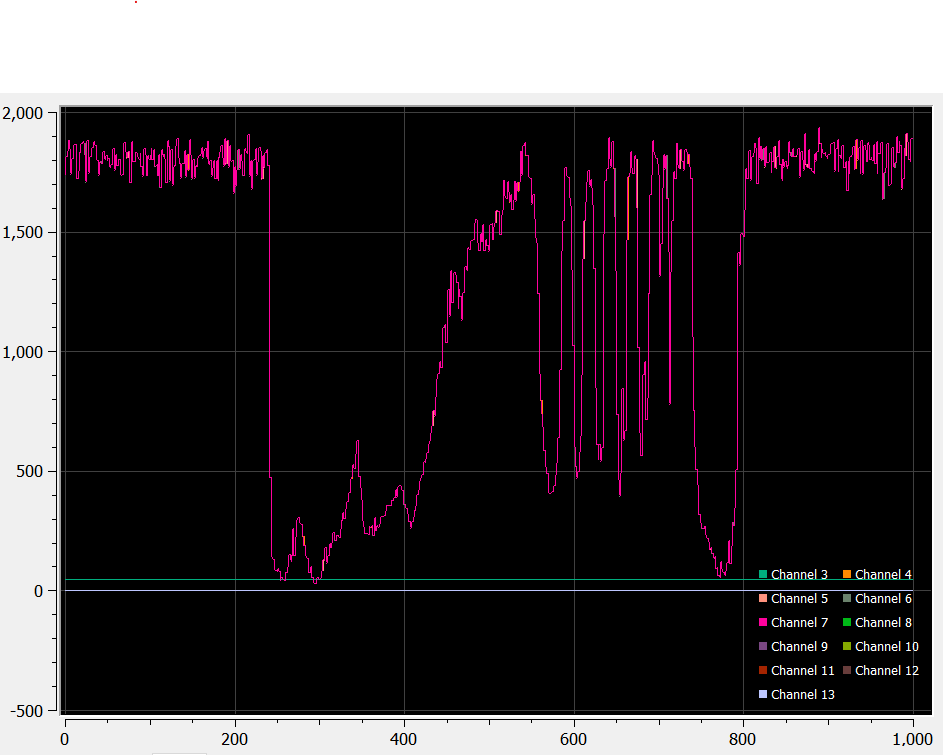


Fig. 16- 4. LiDAR data from moving hand up and down over sensor.

After testing what would happen if the I2C cable came loose, or the communication with the LiDAR hung during the execution of the main loop, it became clear that any failure would result in the main loop hanging, and loss of control of the drone.

To combat this in the short time available, instead of implementing the I2C distance monitoring in another thread, I implemented a watchdog in the Systick timing interrupt.

This watchdog watches to see if a flag gets reset, and if the flag is not reset for 50 executions of the Systick interrupt (a 1ms interrupt), so 50ms, the motor PWM values are set to zero which will immediately cut the motors. This means the drone will basically fall out of the air, but this is significantly better than it continuing to fly off in a random direction or tumble uncontrollably.

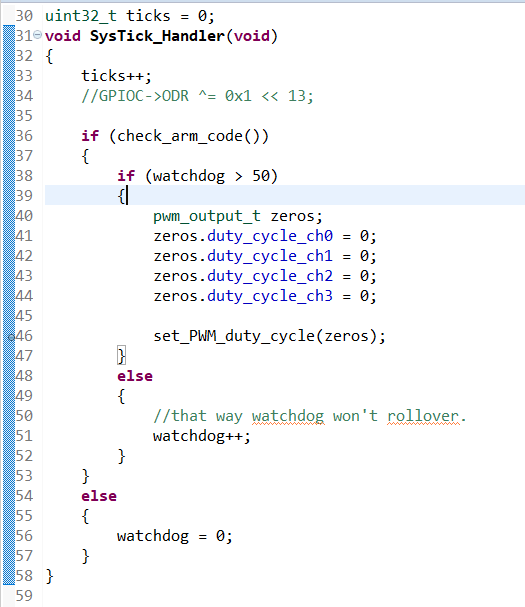


Fig. 16- 5. Watchdog logic

Once I was satisfied with the safety of the code, I had to implement the control loop. I used another PID loop using my same PID loop library code. After spending some time setting bounds on the PID output and attempting to fly it using only proportional feedback (P) in the control loop, I realized that it desperately needed to use derivative feedback to avoid either not being able to control the height of the drone or entering an unstable oscillation, due to the physical transfer function of the drone being a falling mass.

After adding derivative feedback to the PID loop tune, I noticed that nothing was changing the motor output plot. After some thinking and debugging, I quickly realized that this was due to the sudden change and (relative to the main flight loop) slow update rate of the LiDAR data. Instead of smoothly transitioning from one distance reading to the next, it would jump after an update, making the transition instantly. This resulted in 1 sample of the derivative feedback loop reacting to the change, but this 1 outlier would have no effect on the motor output, as the next motor output sample generated would go right back to the previous value before the value changed.

To combat this, I first implemented a FIR filter to induce a smooth change in the data, however this proved computationally intensive, and had too low of cutoff frequency. So instead, I utilize a single pole IIR filter to efficiently allow the newly measured value to be smoothly approached by the computed filter output, without sudden jumps, but also with minimal delay as opposed to delay induced by causality of FIR filters.

Graphical user interface, text, application

Description automatically generated

Fig. 16- 6. IIR filtering of distance measurement

After implementing this filter, the data smoothly changed between updates of the data, with the settling time of the filter approximately equal to the period of the LiDAR update routine.

With that issue fixed, implementing derivative feedback gave a noticeable impact on the motor outputs in the data plot. With this boosting my confidence, I tried another flight test, which yielded a slightly janky but working auto-throttle test! I retuned the PID loop accordingly, and the auto-throttle now works excellently, quickly reacting to any change in LiDAR reading and swiftly locking on to the targe height.

With this implemented, I can now fly the drone with one hand, only steering it laterally. No throttle necessary!

**Pi assembly**

For the autonomous section of the drone, I needed to mount the Pi Zero along with the flight controller. They are designed with the same form factor intentionally, so that the zero can plug directly into the flight controller to communicate with it and send guidance commands.

A picture containing electronics

Description automatically generated

Fig. 16- . Demo of mounting of Pi to flight controller blank board

Aside from the positioning of the boards, I had to figure out the connector interface from Pi to PCB. I ended up choosing a shorter than normal version of 0.1” female headers for the flight controller, plugging into slightly trimmed male headers on the Pi.

A picture containing text

Description automatically generated

Fig. 16- . Figuring out header sizing and flight controller mount.

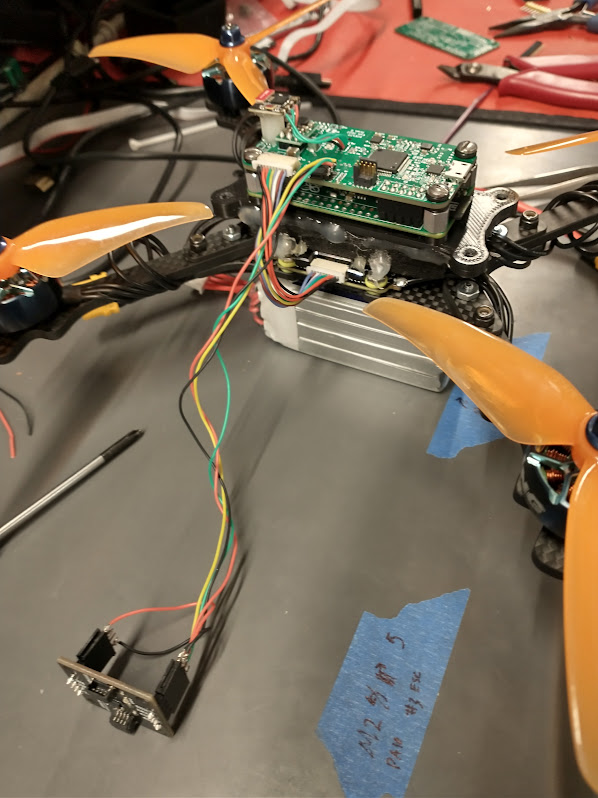


Fig. 16- . (Kinda grainy) Completed mounting interface.

With the Pi now receiving power and its UART connection through the above connector, the team set up SSH login through the USB port and tested UART communication with the flight controller using the existing code demo-ed using the Pi 3.

This ended being a massive pain due to some networking issues and a kernel setting that Owen fixed.

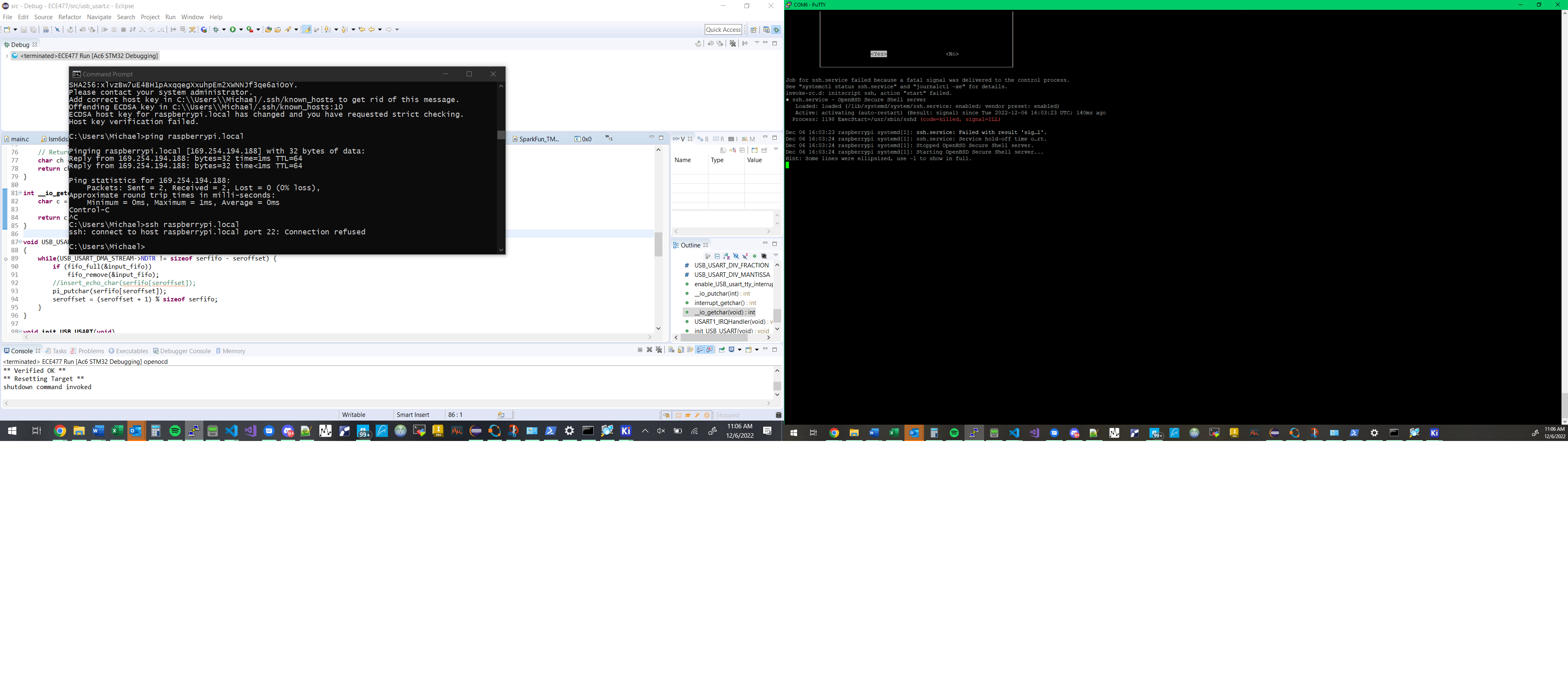


Fig. 16- . Not great....

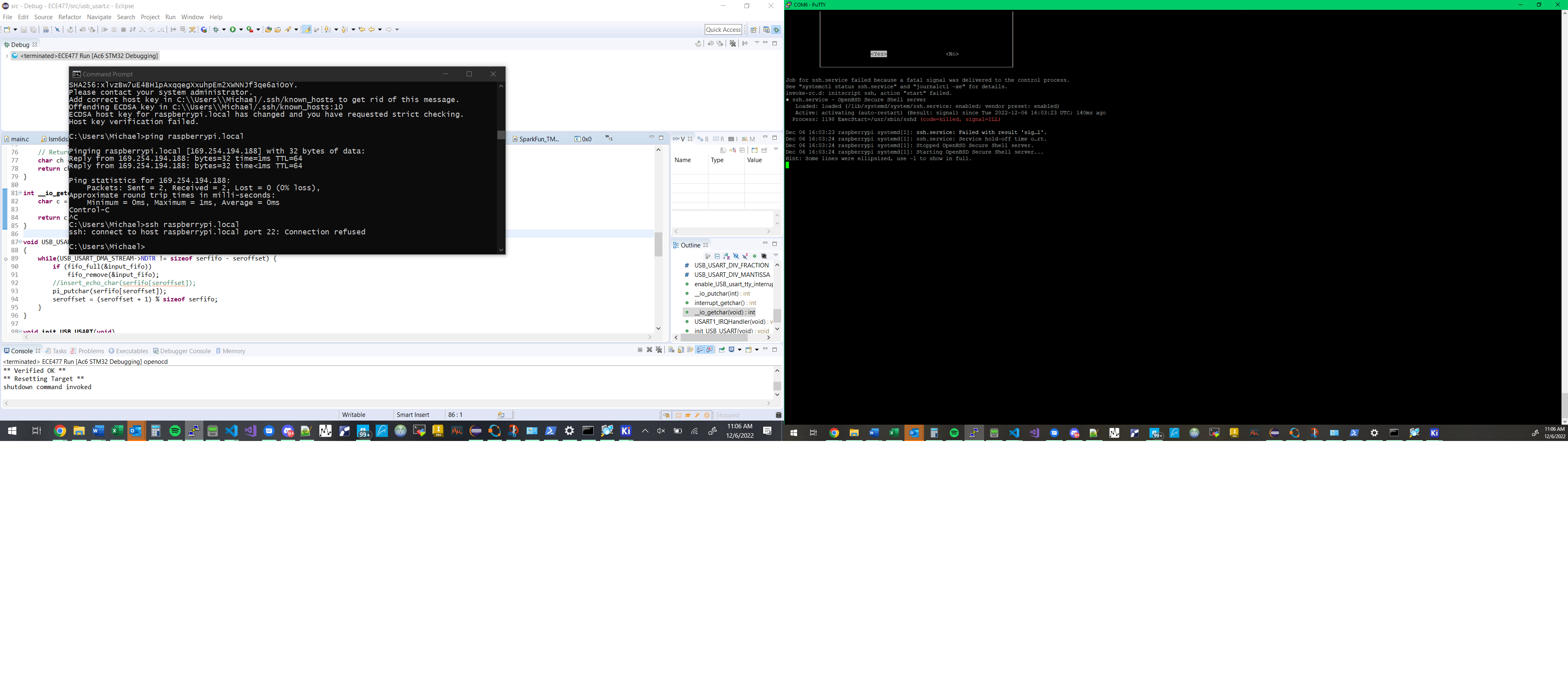


Fig. 16- . Also bad... SSH service won't start.

However, at long last, we were able to plug in a USB cable to power the entire flight control & navigation assembly. Then after booting, we could log in to the pi through SSH.

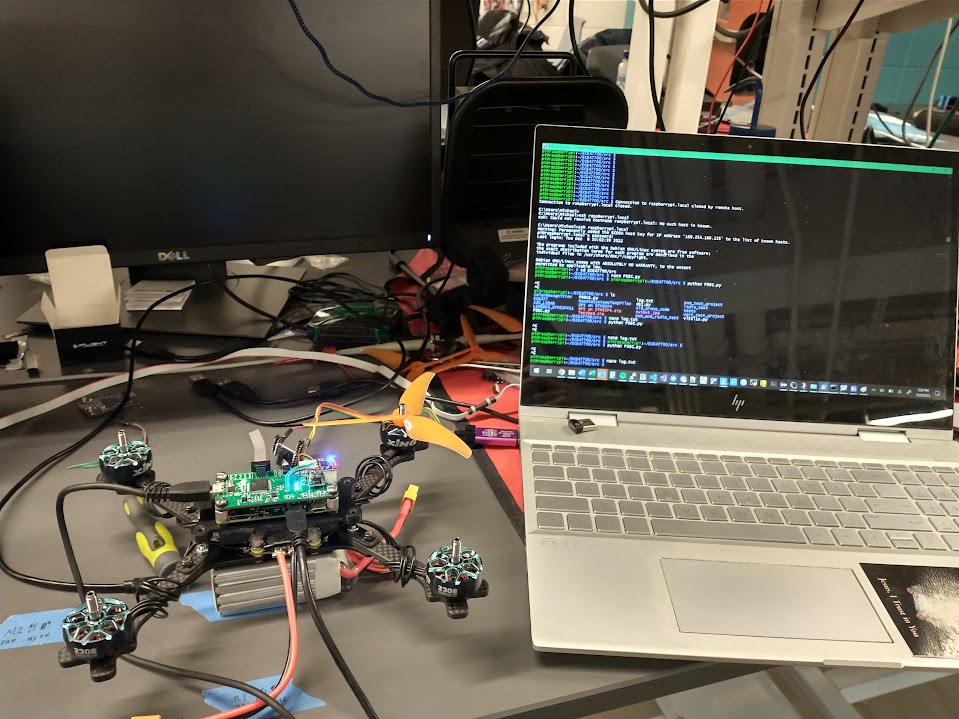
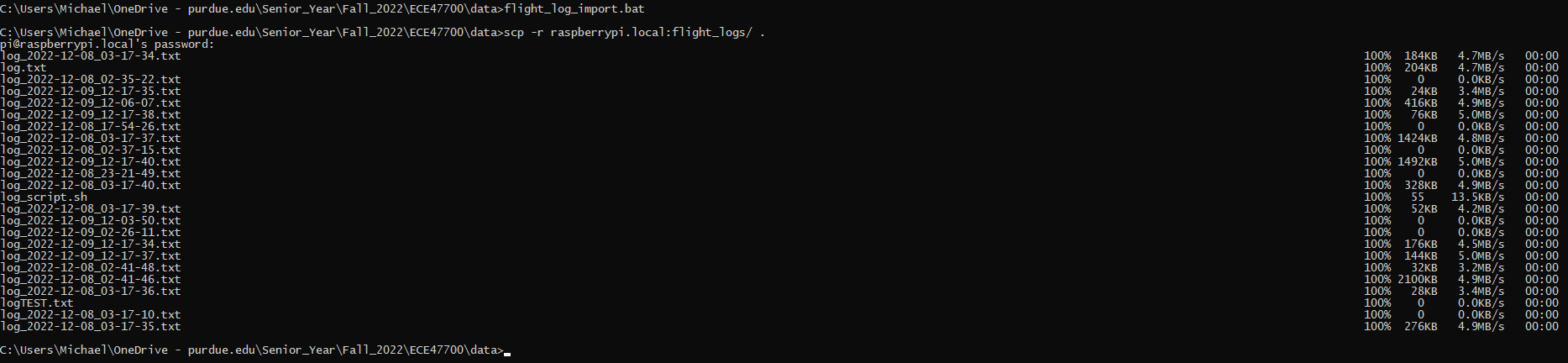


Fig. 16- . SSH’ed into the Pi Zero

**Flight logs**

With the Pi Zero communicating, the first thing implemented was logging all transmitted data from the Pi Zero to a timestamped log file, which Owen got working quite nicely. I then wrote some simple scripts to rename them for use on a Windows system, and to SCP them off the Pi from the root directory so I could pull them into MATLAB and run simulations and analysis on them.

Text

Description automatically generated with medium confidence

Fig. 16- . SCP copy of log files over to Windows from Pi Zero

Chart

Description automatically generated

Fig. 16- . Sample MATLAB plot of gyro data logged by Pi from flight controller.

**Camera & Autonomous mode**

Lastly, I tested the autonomous scripts Owen developed to test their integration with the flight controller, and with the slightly smaller camera used on the Pi Zero.

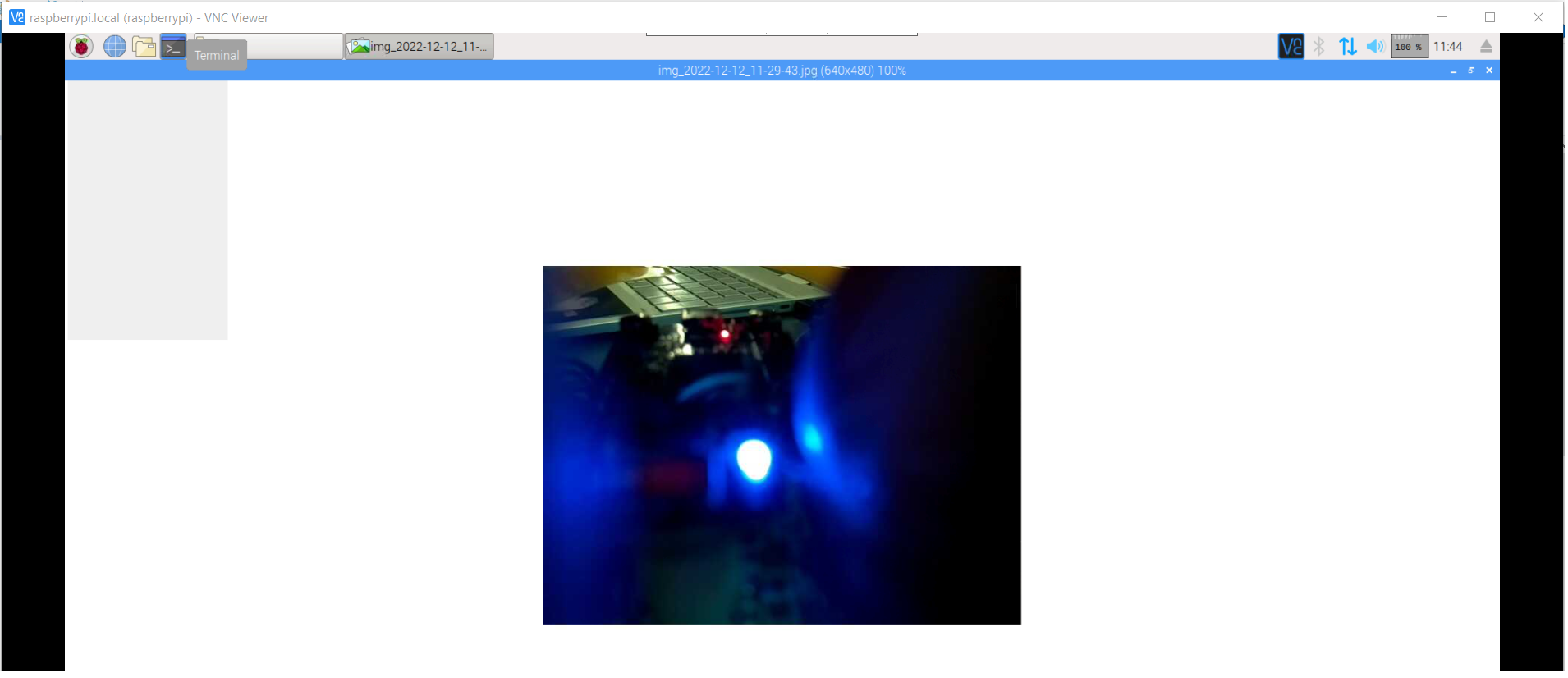
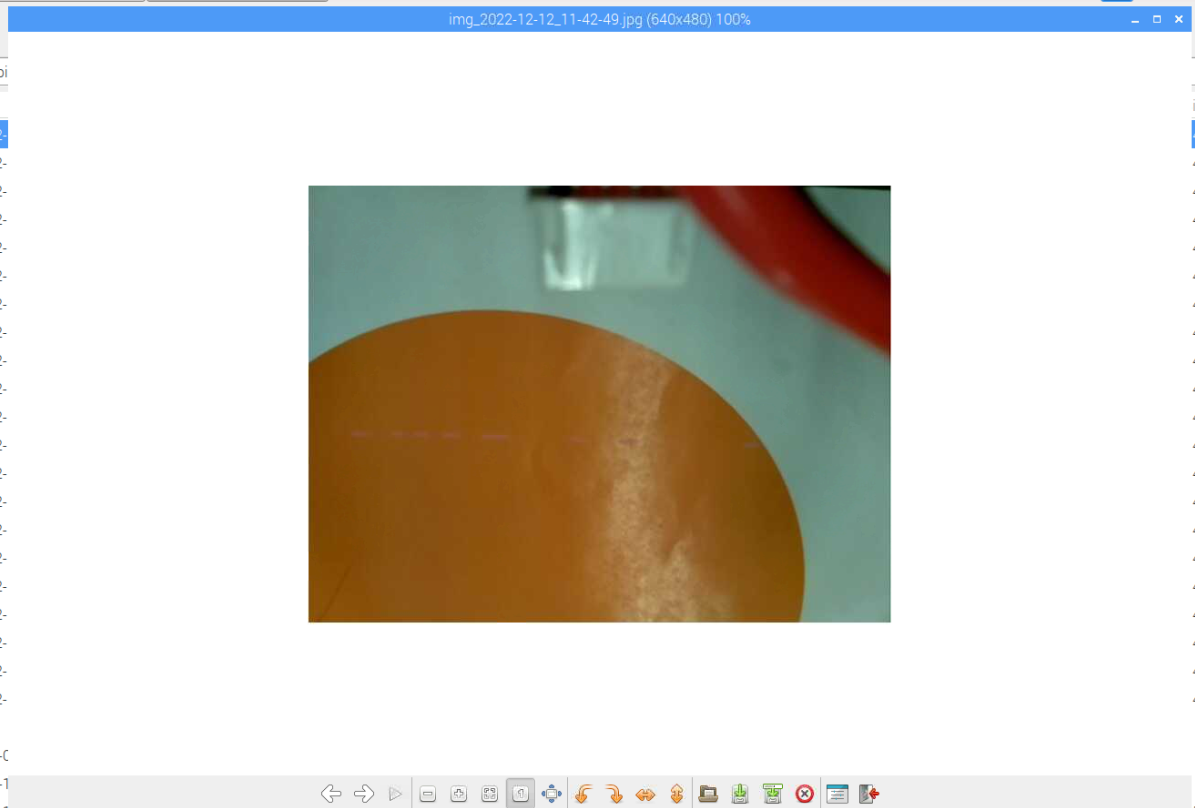


Fig. 16- . Photos logged by Pi Zero viewed through remote login via VNC

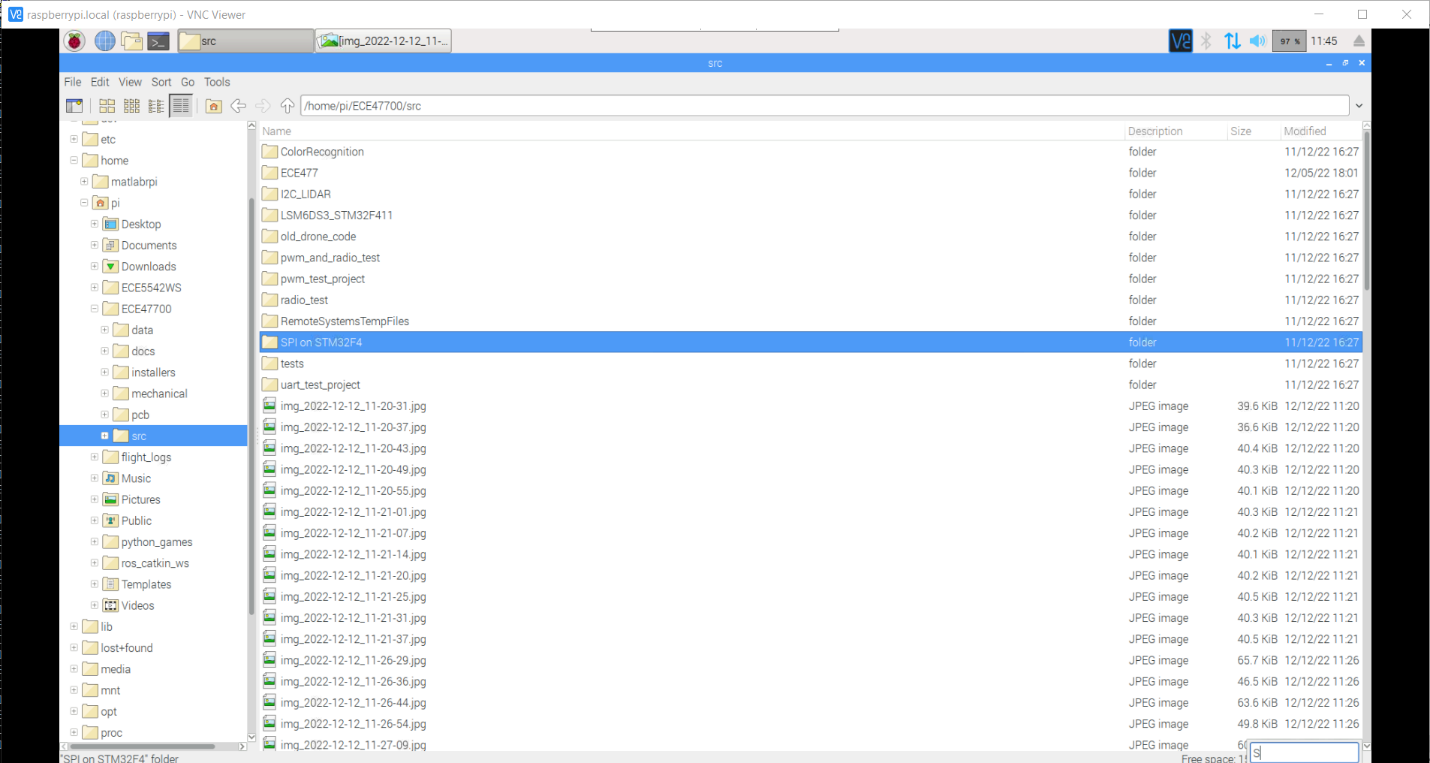


Fig. 16- . VNC Viewer showing file explorer with lots of photos.

With the camera working nicely, I tested the steering commands coming from the Pi, ensuring that they were being properly parsed and, when in fully autonomous mode, passed along to the control loops as if they were normal radio control stick inputs.

After editing my control loop code to conditionally take these values as the inputs instead of the control stick values, I both got the commanded steering values from the Pi to come through, but also to clearly control the motors when in auto mode.

Graphical user interface

Description automatically generated

Fig. 16- . Motor values show steering commands resulting in control loop attempting to drive drone to commanded attitude.

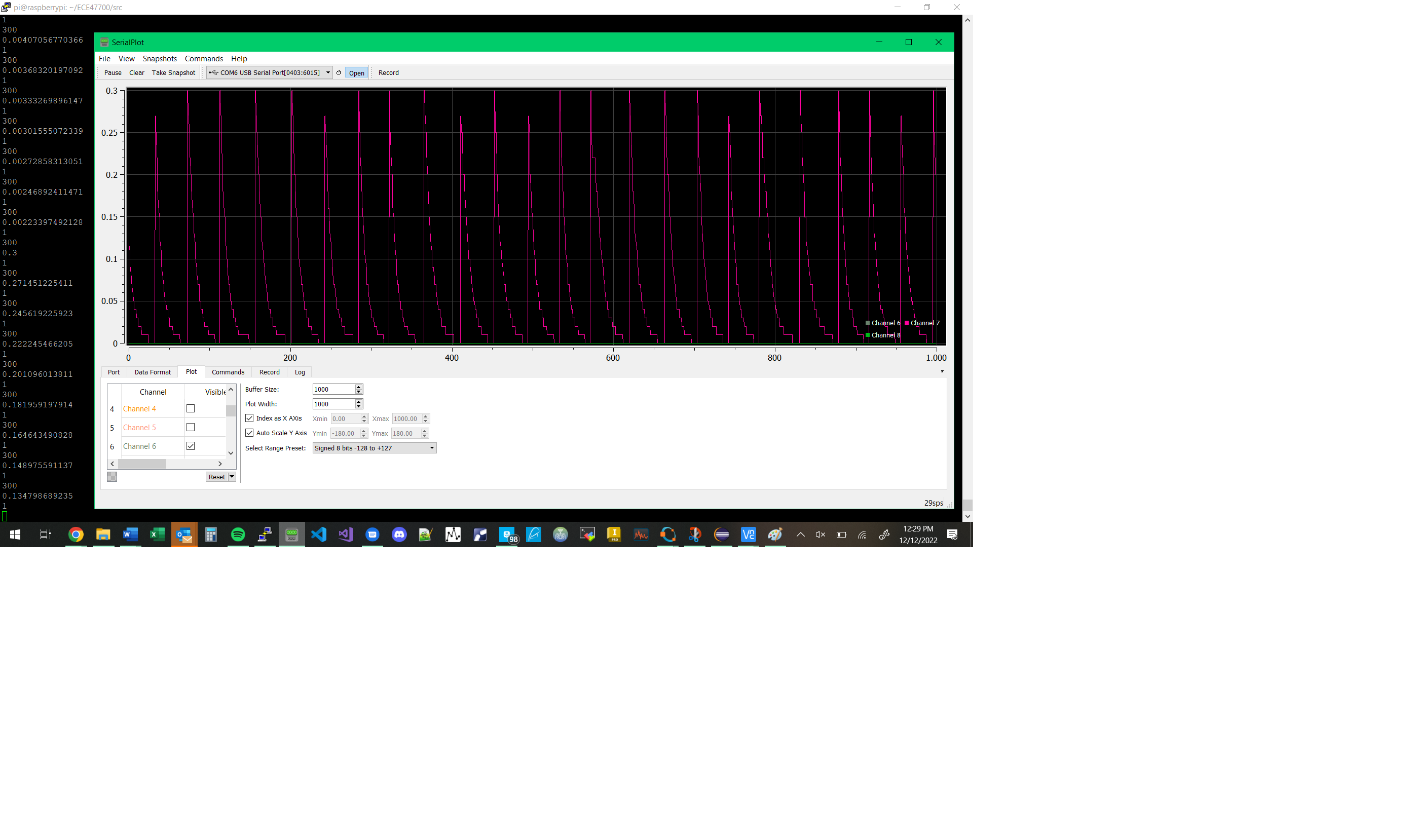


Fig. 16- . Raw commanded attitude data from Pi Zero. Pi sends pulses of positive steering commands to nudge drone forward.

Lastly, I implemented a simple landing sequence to land the drone upon it finding the visual target. This meant that, while in auto mode, if a ‘found object’ command was sent, the drone would immediately begin to descend and land by reducing its target height above ground, and its throttle center point that the control loop superimposes its feedback upon.

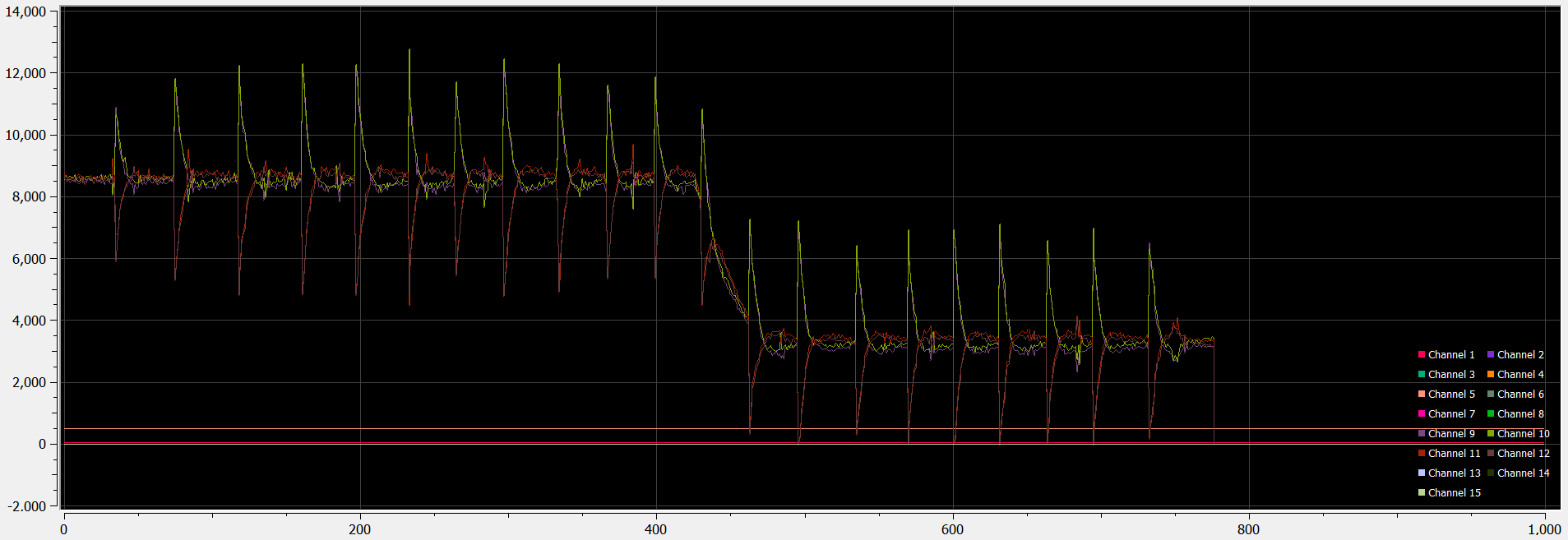


Fig. 16- . Landing resulting in exponential decay of target elevation and motor throttle