AeriaL Instrumented ExploratioN Systems (A.L.I.E.N.S.)
Final Paper

Mahderekal Regassa, Sydney Nguyen, Emma Slibeck, Katie Shaw, Zoë Street, Michelle Tu

ASTR 202: Hands-on Planetary Exploration

Dr. Wesley Watters

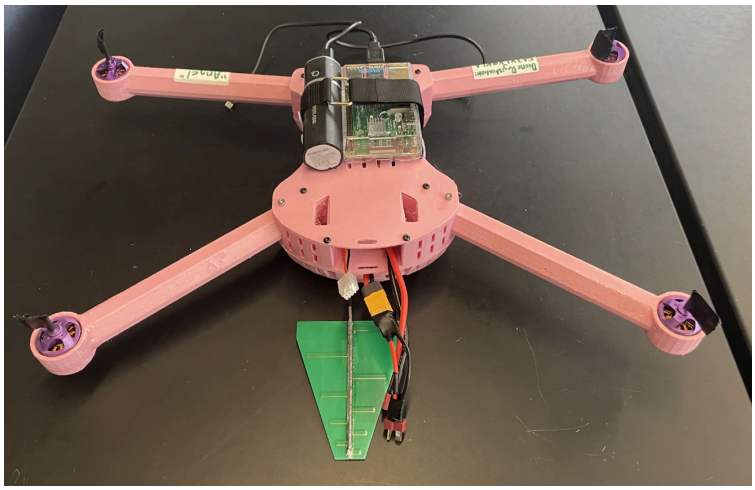
May 11, 2023

Table of Contents

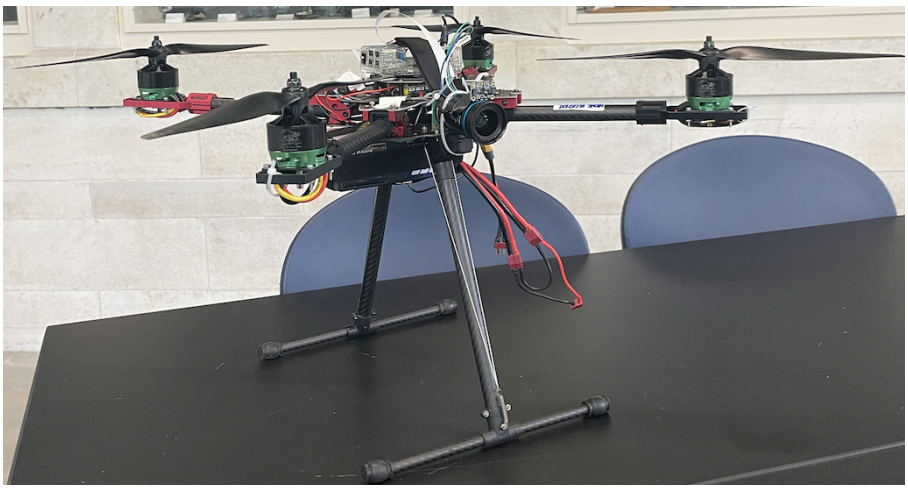
Page

Abstract

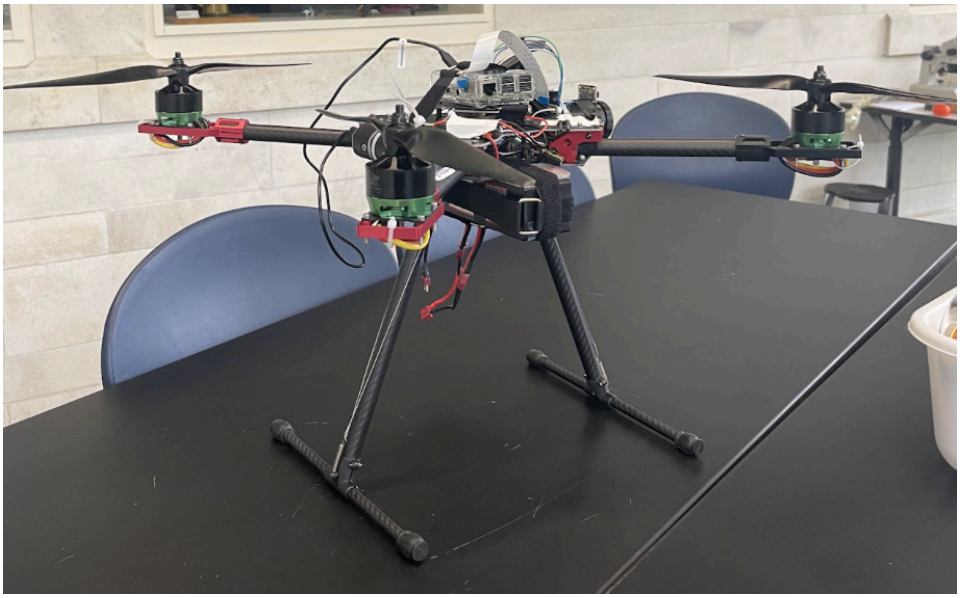
ALIENS is a project to design, build, and deploy 2 drones. The data collected by the sensor payload of these two drones models a UAP encounter. We assembled the first drone, named “Stitch”, using a pre-existing frame, and we 3D printed the frame of the second drone, named “Angel”. We chose the components for both of these drones, including flight controllers, motors, propellers, batteries, receivers, and antennae in accordance with the specifications of these frames. The sensor payload of each drone includes an HQV, FPV, IMU, and GPS. Stitch additionally carries a radio receiver and antenna. We used HQVs to capture images to be used in photogrammetry. The IMUs and GPSs were used to determine the HQVs’ locations, directions, and orientations to assist with photogrammetry. We used the FPVs to capture a live video feed of the drones’ deployment and to assist with positioning of the HQVs. We used the radio receiver and antenna because genuine UAP encounters have recorded anomalous radio frequency signals. Our drones used a model UAP to investigate, which consists of a collection of instruments to measure. One of these instruments is a radio module that produces radio signals, and the other is a balloon, which we used for photogrammetry calculations.



a) Angel



b) Stitch front view



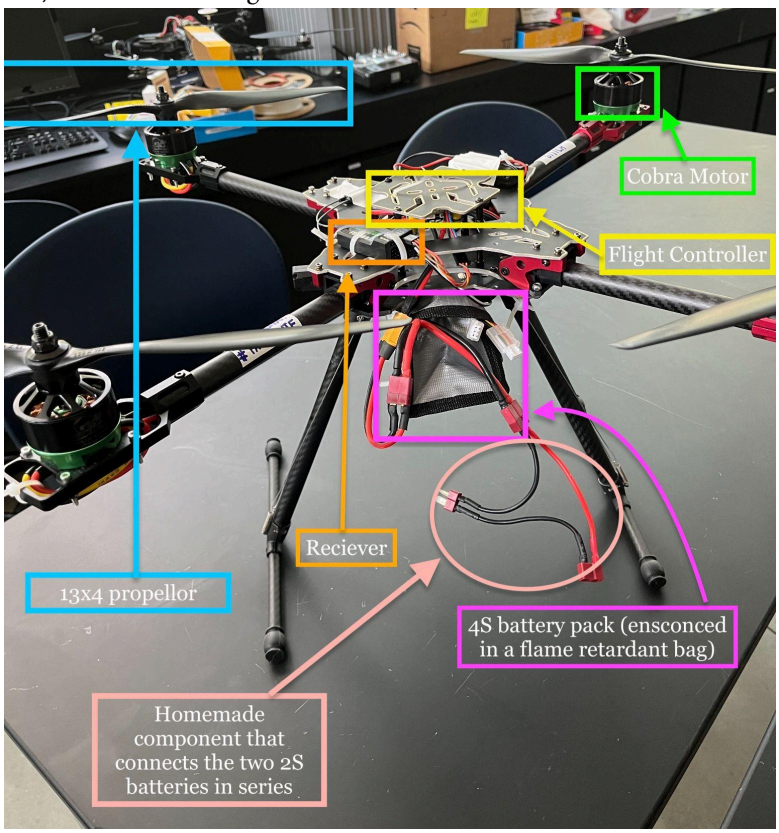
c) Stitch back view

Fig. 1.2 The two drones: Stitch and Angel

Vehicle Description

"Stitch" is a quadcopter drone built from a HMF U580 Pro Carbon Fiber Umbrella Folding Quadcopter Kit, which has a diameter of approximately 24 inches from the end of one motor arm to another (see Fig 1.3 for reference). Stitch's lyft is powered by four Cobra C3510/20 brushless motors, each with a KV value of 1000 and powered by a 4S 5200mAh LiPo battery pack composed of 2 2S batteries. This drone employed the SpeedyBee F7 V3 BL32 50A 30x30 Stack as the flight controller, and this component also contained the power distribution board (PDB), the 4 electronic speed controllers (ESCs), and various ports including UART ports and a port for the receiver. For this drone, we used a FS-iA6b receiver that paired with our Flysky Fs-i6X 6-10 2.4Ghz AFHDS RC Transmitter. Stitch used 13x4 in [diameter x pitch] propellers of two different forms, 2 right-hand propellers for motors that spin in a clockwise rotation and two left-hand propellers for motors that spin in a counterclockwise rotation. In order to generate lyft, the motors on the same diagonal must spin in one direction while motors on the other diagonal must spin in the other direction. Stitch also had retractable landing gear, which can be moved from 'landing' position where the feet are below the body of the drone to 'retracted' position where the legs extend horizontally where the feet extend to the side of the drone. Stitch's sensor payload consists of a GPS system, first-person camera for remote observation, IMU for flight monitoring, photogrammetry for image modeling, and a radio spectrum analyzer to detect external radio signals. The FPV system is an independent system on the drone powered by a separate 7.4 V battery pack, while the GPS, IMU, and photogrammetry utilize a Raspberry Pi powered by a portable battery to control operations. In Fig. 1.3 b) below, one can see the sensors attached loosely to Stitch. The sensors were not attached securely to Stitch in a manner that maintained Stitch's previously balanced center of mass because a sensor flight with Stitch was descoped, and the sensors were tested on Stitch by raising Stitch manually and walking around.

a) Annotated image of Stitch before sensors were mounted



b) HQV, IMU, FPV Cam, Radio Spectrum Analyser, portable battery, and Rasberry Pi mounted on Stitch

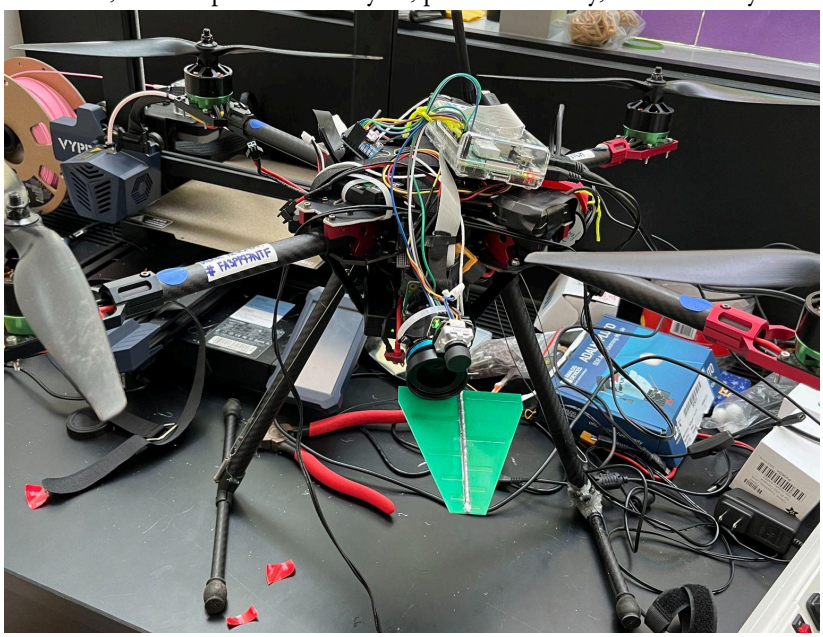
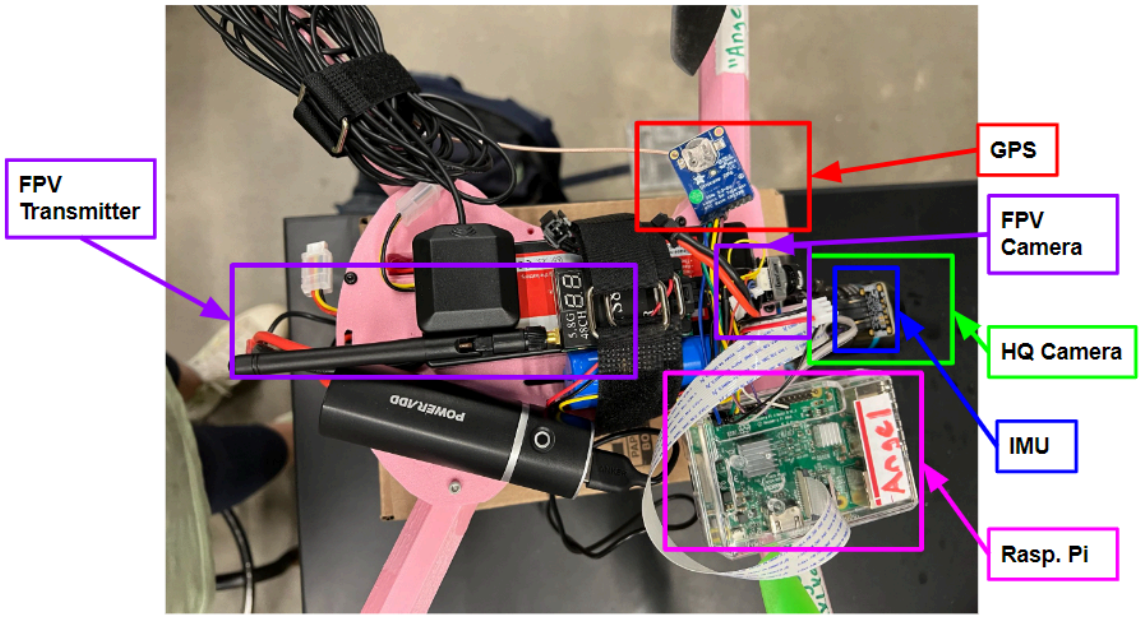
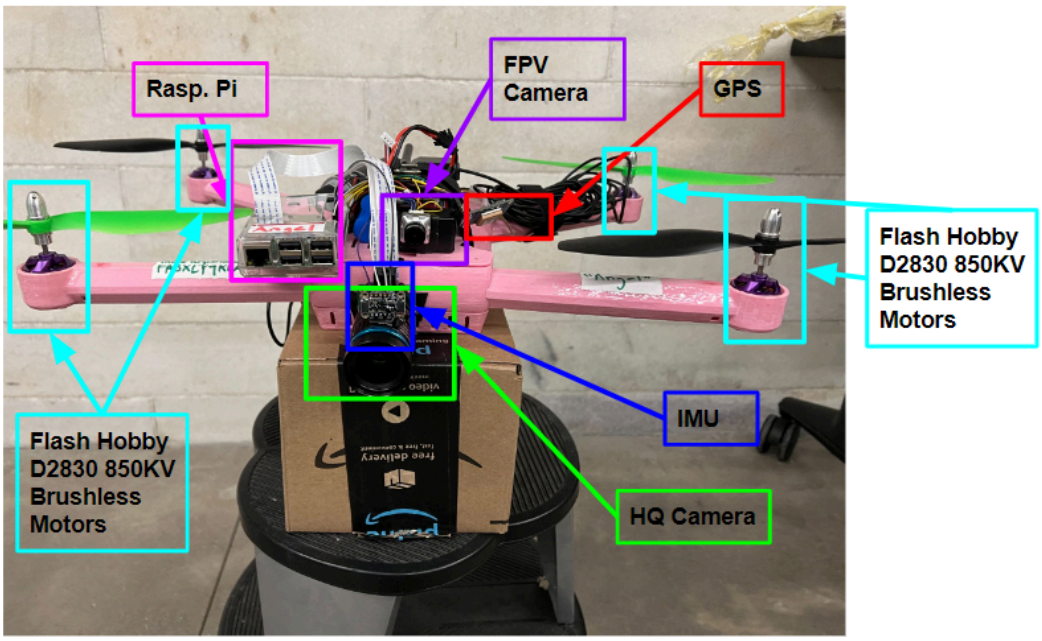


Fig. 1.3 Stitch without sensors (top image) and with sensors mounted (bottom image)

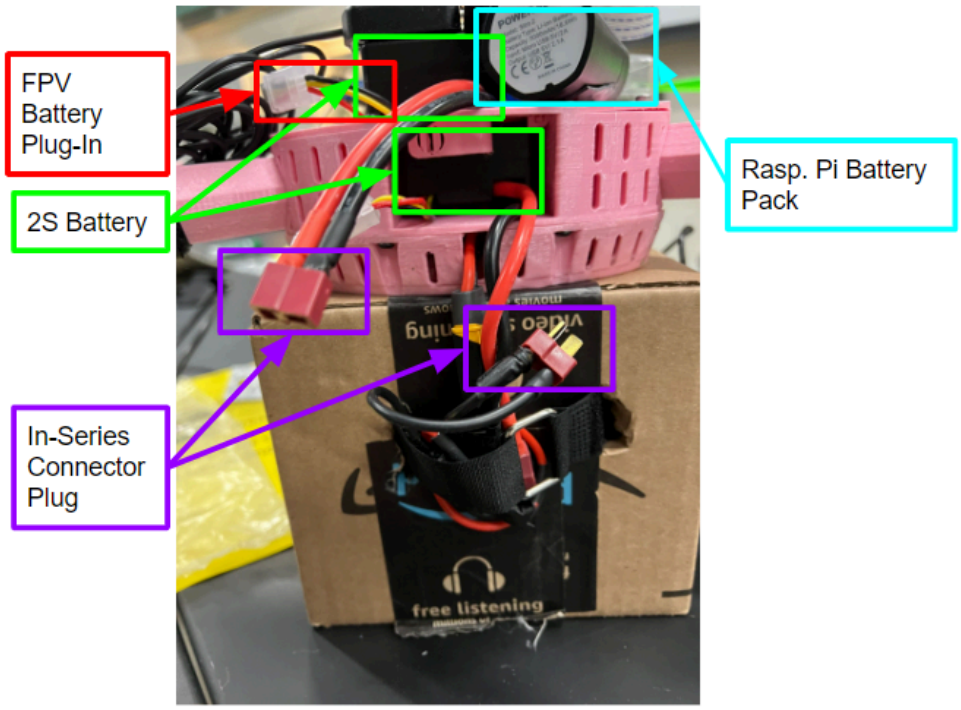
"**Angel**" is a quadcopter drone built from a 3D printed frame using PETG filament. The frame consists of 4 arms, a top plate, frame body, and sensor cage reinforced by an epoxy coating to prevent breakage. "Angel" was not designed with any landing gear and in its final testing iteration was secured to a small cardboard box to provide lift off the ground. Further designs that were not realized in the given timeframe include four 3D printed legs that "hook" onto the arms. "Angel" is a smaller drone than "Stitch" measuring only 20 inches across with four 6.5 inch arms. "Angel" is controlled by a SpeedyBee F7 V3 BL32 50A 30x30 Stack Flight Controller and ESC connected to four Flash Hobby D2830 850KV Brushless Motors (1300 KV value) which are powered by two 2S batteries connected in series for a 4S battery pack equivalence. "Angel's" lift was provided by four 10 x 4.5 propellers which are slightly larger than the recommended and may have led to inflight issues as the propellers collided during flight. The propellers are arranged on the drone such that the propellers opposite each other spin in the same direction while neighboring propellers spin in opposite directions. At the conclusion of this project Angel was not connected to a receiver and RC transmitter drastically limiting its flight capabilities, as without these controls the drone is unable to automatically stabilize when in the air. "Angel's" sensor payload consists of a GPS system, first-person viewpoint system for remote observation, IMU for flight monitoring, and photogrammetry for image modeling. The FPV system is an independent system on the drone powered by a separate 7.4 V battery pack, but the GPS, IMU, and photogrammetry utilize a Raspberry Pi with an external battery pack to control operations. The images below document the final testing form of "Angel" with labeled parts as visible. The IMU and ESC stack is encased inside the drone frame and not visible from the selected images.



a) Top view of Angel



b) Front view of Angel



c) Back view of Angel

Fig. 1.4 Angel from different views with sensors mounted

Description of Sensor Measurements

Sensor Payload

We used a pair of Adafruit LSM6DSOX + LIS3MDL IMUs on our drones. These IMUs are 9 degrees of freedom sensors, combining accelerometers, gyroscopes, and magnetometers. We wired each IMU to a Raspberry Pi, which included scripts for data collection and automation. We oriented our IMUs such that, from the point of view of the cameras, the x-axis is left and right, the y axis is forward and back, and the z axis is up and down. Radio spectrum analysis was done with the HackRF, with a directional log periodic antenna with a range of 1350-9500 MHz. GPS data was recorded with an Adafruit Ultimate GPS with a 5 Meter SMA external antenna.

Measurements and Sources of Uncertainty

1. Radio spectrum

For the radio spectrum analyzer, we conducted a spectrum sweep using the HackRF from 2000-2500 MHz with the goal of picking up a "UAP transmission" at 2300 MHz. The frequency was chosen from within the receiving/transmission ranges of the Adalm Pluto (325-3800 MHz), the HackRF (1 MHz-6GHz), and the HackRF's directional antenna (1350 - 9500 MHz). 2300 MHz was one of the only frequencies that it was legal to transmit on under Wes's HAM radio license that also fell within our transmission/receiving ranges. The HackRF performed around 15 sweeps per second for 6 minutes.

Radio spectrum analysis was a portion of the data collection that had fewer sources of uncertainty overall, as the radio transmission was also programmed by us. However, there was one phenomenon that was unexplained - on the heatmap generated from spectrum sweep data, the expected signal at 2300 MHz was accompanied by bands of fainter signals at 2280, 2290, 2310, and 2320 MHz that followed the pattern of our generated signal at 2300 MHz. For example, in the morse code heatmap, the faint signal bands follow the morse code pattern as well.

2. GPS

We got a GPS recording every second. All GPS technology has about a 3m accuracy.

3. Photogrammetry

We used two Raspberry Pi High Quality cameras with 16mm lenses for the purpose of photogrammetry. For template matching and generation of the disparity map using stereo photogrammetry, we used two images of the mock UAP taken simultaneously, from different angles. Generation of a disparity map uses the principle that in stereo images, the amount that one feature shifts from image to image is inversely proportional to the feature's real-world distance from the camera. The template matching step involves converting the images to grayscale, splitting each image into overlapping square tiles, and comparing each tile of one image to each tile of the other to find correspondences. There are multiple parameters that can be changed for template matching: 1) the calculation of tile similarity - I used sum of absolute differences, which performed very similarly to sum of squared differences; 2) the window size - I opted for 100 pixels with images of around 3000 pixels wide, to balance between efficiency and performance; 3) the search method for similar tiles - I used either an exhaustive search, where each tile was compared against every tile of the second image, or a horizontal search, where each tile was compared against every tile in its same row in the second image. The "best" (closest to

ground truth) results were obtained with sum of absolute differences, 100 pixel window size, and horizontal search. We ran template matching with the script template_matching.py which stores template matching results in a json file of the form {tile 1 index: image 2 match, tile 2 index: image 2 match, ...} where tile 1 index is the index of some tile in image 1, and image 2 match is the index of its corresponding tile in image 2.

The disparity map is generated from this json file of template matching results. It calculates the euclidean distance between each tile and its match, and stores the results in a 2D list that can be converted to a heatmap that shows the distance from each tile to the camera. Disparity mapping is done with the script disparity-map.py.

There is some noise and uncertainty inherent to the process of stereo photogrammetry - with only two images, there is potential for confusion depending on the patterns in the images. For example, when taking test images in front of blank walls, the heatmap of the wall appears very noisy. This is because it is such a uniform surface - all of the tiles of the wall are very similar, so wall tiles are matched to each other arbitrarily, and some can match to nearby tiles while some are matched to farther ones. One possible solution for this is to establish a similarity threshold, and start matching tiles at their current location. For instance, a blank wall tile at index 4 could start searching for a match in image 2's index 4, match with that tile, and stay matched with that tile for the remainder of the search. I am in the process of testing different thresholds for the best heatmap. Another possible solution is to establish a threshold for "blankness" of a tile, and to default to self-matching with tiles considered blank.

4. IMU

The IMUs' data is output in 3 directions for each type of measurement, for a total of 9 outputs for each sampling. Acceleration is given in meters per second squared, rotation in radians per second, and magnetic field strength in microteslas. It samples twice a second. We determined the error in the measurement for each component sensor by leaving the IMU still while collecting data for 5 minutes. We then found the standard deviation and divided it by the square root of the sample size. While the IMU was at rest, it did not rotate, so all gyroscope outputs were 0. In this test, the sample size was 558, and the errors we calculated are as follows:

Error	x-axis	y-axis	z-axis
Acceleration	0.0002972	0.00029997	0.00024828
Magnetic field	0.01485799	0.01487815	0.0200634

Further sources of uncertainty come from orienting the downward-facing axis to gravity, and aligning the axes with magnetic North. We programmed the IMUs to collect data via a Python script. We modified a script originally obtained from the Adafruit website for ALIENS' purposes.

Results and Interpretations

1. Radio spectrum

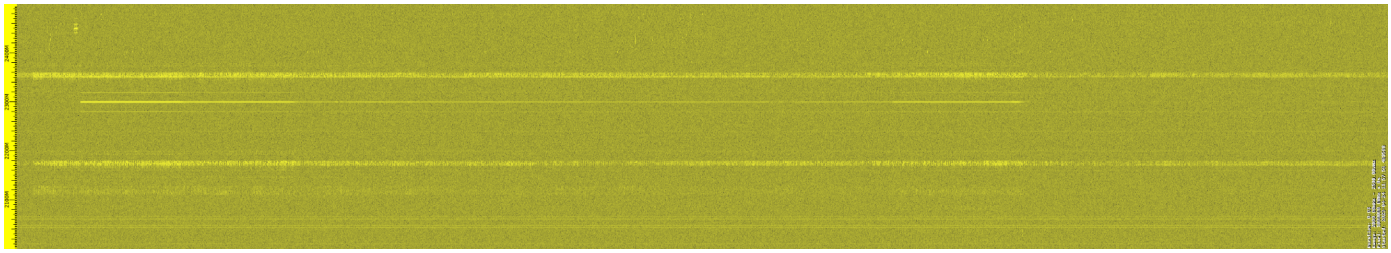


Fig. 2.1 Outdoor test of HackRF One and Adalm Pluto

This waterfall plot (rotated 90 degrees for readability, so time is on the x-axis and frequency, from 2000-2500 MHz, is on the y-axis) shows data from an outdoor test of the HackRF and Adalm Pluto. For this test, I pointed the directional antenna directly at the Adalm Pluto antenna, then pointed 45, 90, 135, 180, 225, 270, and 315 degrees away from the antenna for 15 seconds each. The most intense, brightest line at 2300 MHz on the plot is from the antenna pointed directly at the transmitter, and the signal becomes fainter as the antenna is pointed more away from the transmitter.

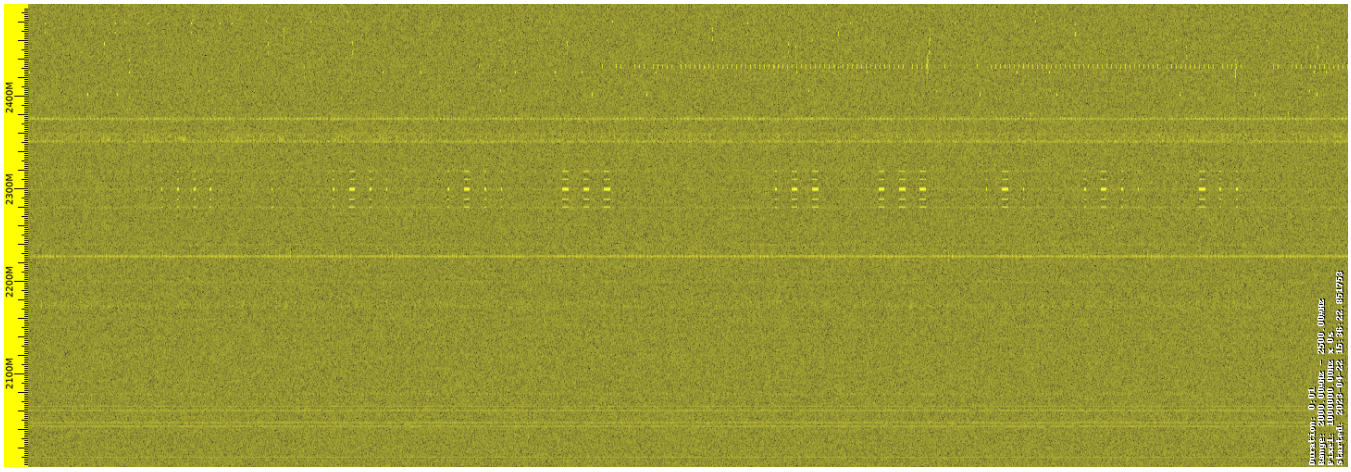


Fig. 2.2 Morse code script test

This plot shows the morse code script in action, as well as the faint presence of "mirror" signals that appear at frequencies neighboring 2300 MHz. The morse code here reads "hello world".

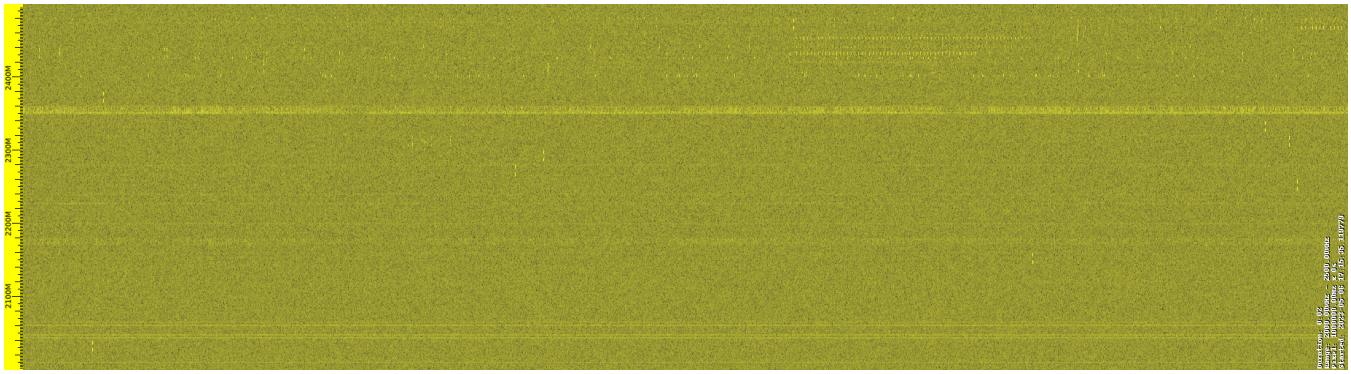


Fig. 2.3 Final “flight”

This is the plot from our 6 minute final outdoor test, with the HackRF mounted on Stitch and carried around. Even though the Adalm Pluto was transmitting a morse code massage, it does not show up on the plot - this could be because it was too intermittent of a signal to pick up while moving around, or because the distance from the transmitter to the receiver was too great.

2. FPV images (Emma)

The remote observation systems on the drones functioned as designed, but the transmitter signal seemed insufficient for communication in a field with obstructions. The FPV system needed a line of sight signal in order to connect to the monitoring station without experiencing serious static and interference. While test flights were unsuccessful, footage of the FPV during these flights was captured and can be found here. Further developments in this system would have included realtime automatic saving of this footage into a secure harddrive.

3. GPS

In our final test, we were able to get a fix and capture one data point for both Stitch and Angel.

date, gps_time_utc, system_time_utc, latitude, longitude

Stitch: 2022-05-03, 22:12:48, -2:12:48, 42.295591, -71.302351

Angel: 2022-05-03, 22:12:51, -2:12:51, 42.294590, -71.302580

4. Raspberry Pi HQ Camera Images

Stitch:

Angel:

These images were taken at a distance of about 5 meters from the balloon while walking and holding the drones.

5. Photogrammetry disparity maps

To calibrate the optimal thresholds and window sizes, these images were used:

For the following heatmaps, darker pixels correlate to tiles that move less from image to image (sections of the image farther away from the camera, and lighter pixels correlate to tiles that move more from image to image (sections closer to the camera).

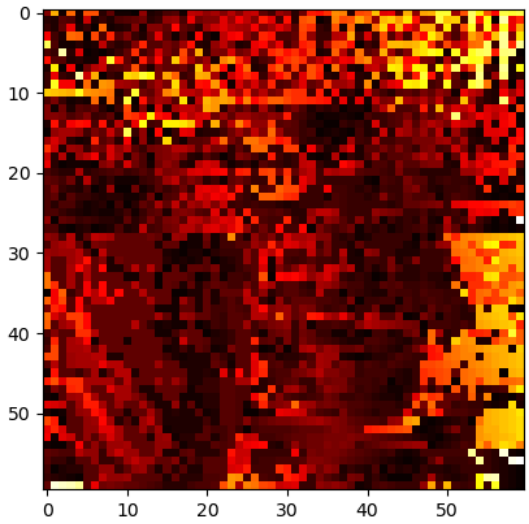


Fig. 2.4 Disparity heatmap with window size 100 px, complete search

This heatmap was generated with a window size of 100 using a complete search (each tile of image 1 is compared against every single tile of image 2). The ridges of the vase can be seen, as well as the plant's outline against the wall - however, the blank wall is very noisy because of the issues with template matching blank spaces as discussed earlier.

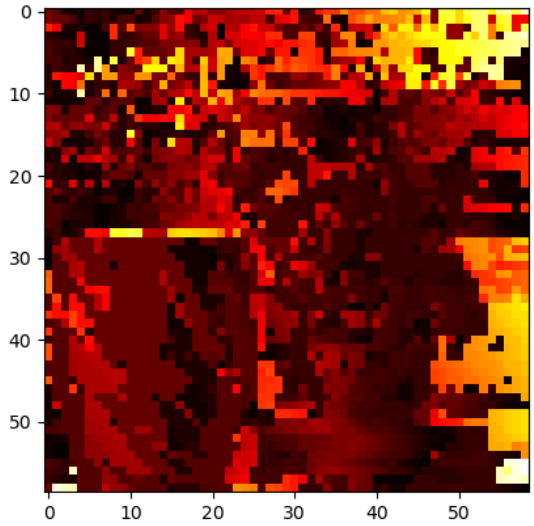
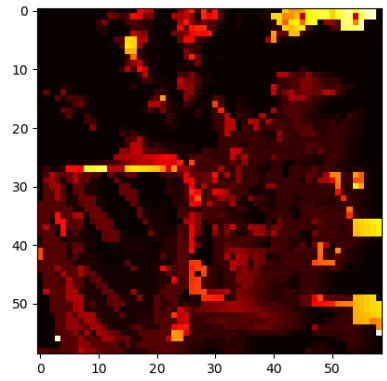
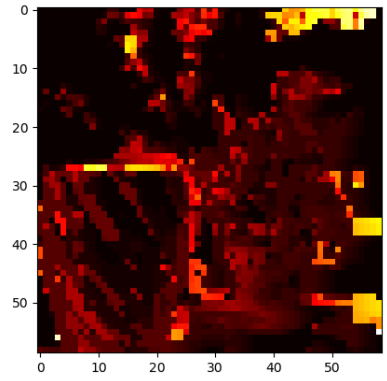
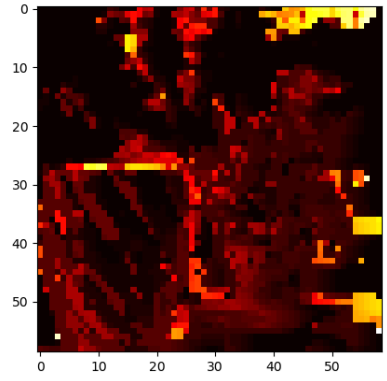


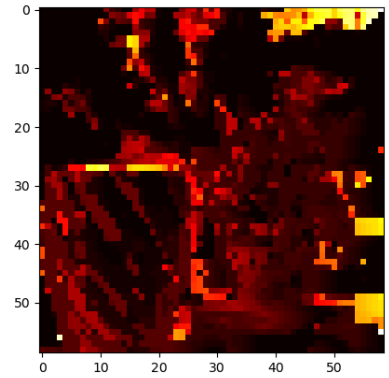
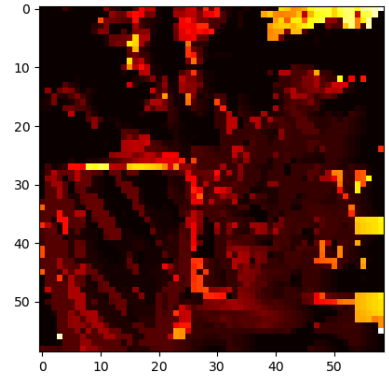
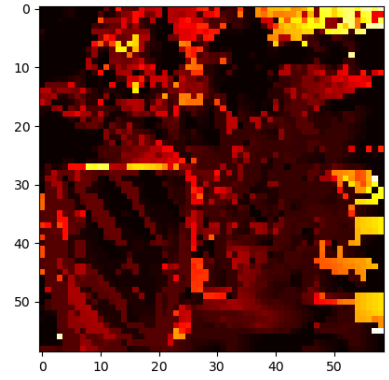
Fig. 2.5 Disparity heatmap with window size 100px, horizontal search

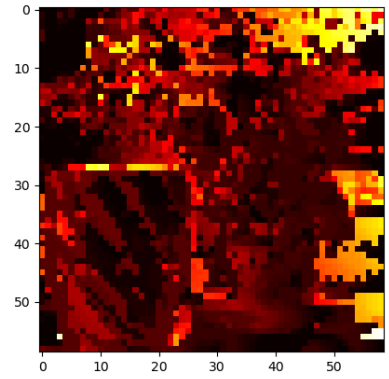
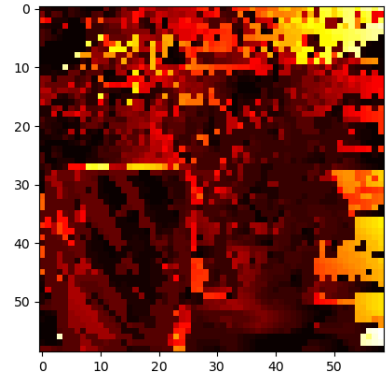
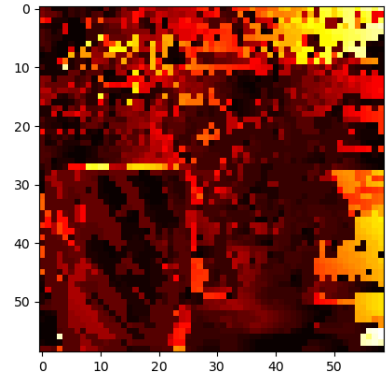
This heatmap was generated with a window size of 100 and horizontal search (every tile of image 1 is compared against its corresponding row in image 2) - it is smoother than the complete search, but still suffers from noise.

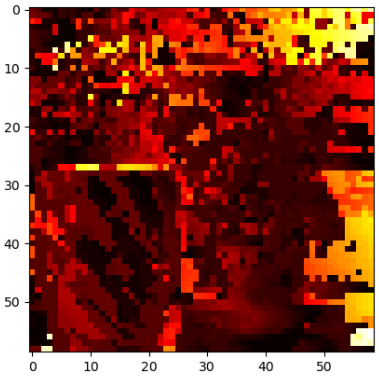
Next, I tested with a horizontal search policy that starts the search for each tile at its respective location before moving down the row, and immediately returns a tile as a match if it falls above a certain threshold in

hopes that the issue of noisy blank surfaces would be addressed. The following tests have different thresholds, and were generated with a window size of 100 and the colorscheme “hot”.

Threshold	Heatmap
9.65e-06	
9.7e-06	
9.8e-06	

9.9e-06	
1e-05	
1.1e-05	

1.2e-05	
1.3e-05	
1.4e-05	

1.5e-05	

Even very small threshold changes have a large effect on the noisiness of the background.

6. IMU

We conducted a test of the IMU mounted on Stitch on 29 April 2023. Because Stitch did not fly at the time of this test, a member of the team walked around carrying Stitch as the sensors collected data. This test lasted 36 seconds and gathered 72 data samples, given that data is sampled every half-second. The following IMU data, therefore, represents their path of travel. We determined the minimum, maximum, average, and standard deviation for each sensor along each axis, which are as follows:

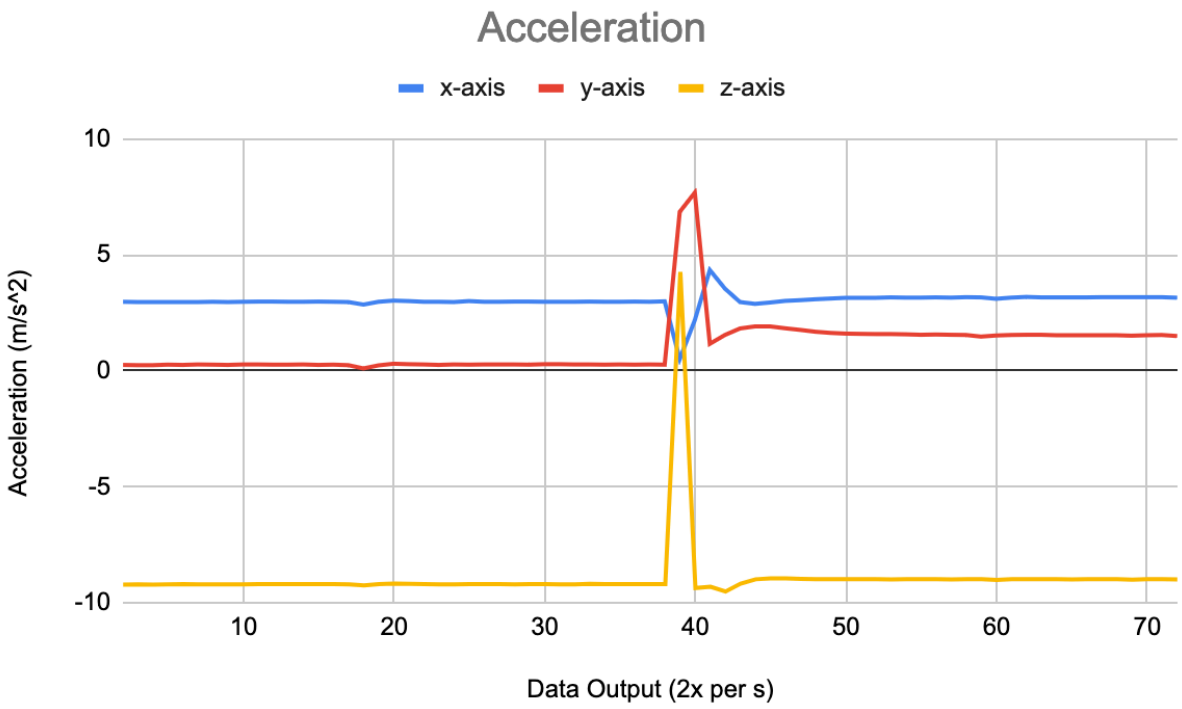
Acceleration	x-axis	y-axis	z-axis
Minimum	0.48	0.08	-9.52
Maximum	4.32	7.66	4.26
Average	2.998169014084507 6	1.051690140845070 5	-8.93169014084507 2
Standard Deviation	0.369317260685001 04	1.245657228998538 8	1.581342793561176 8

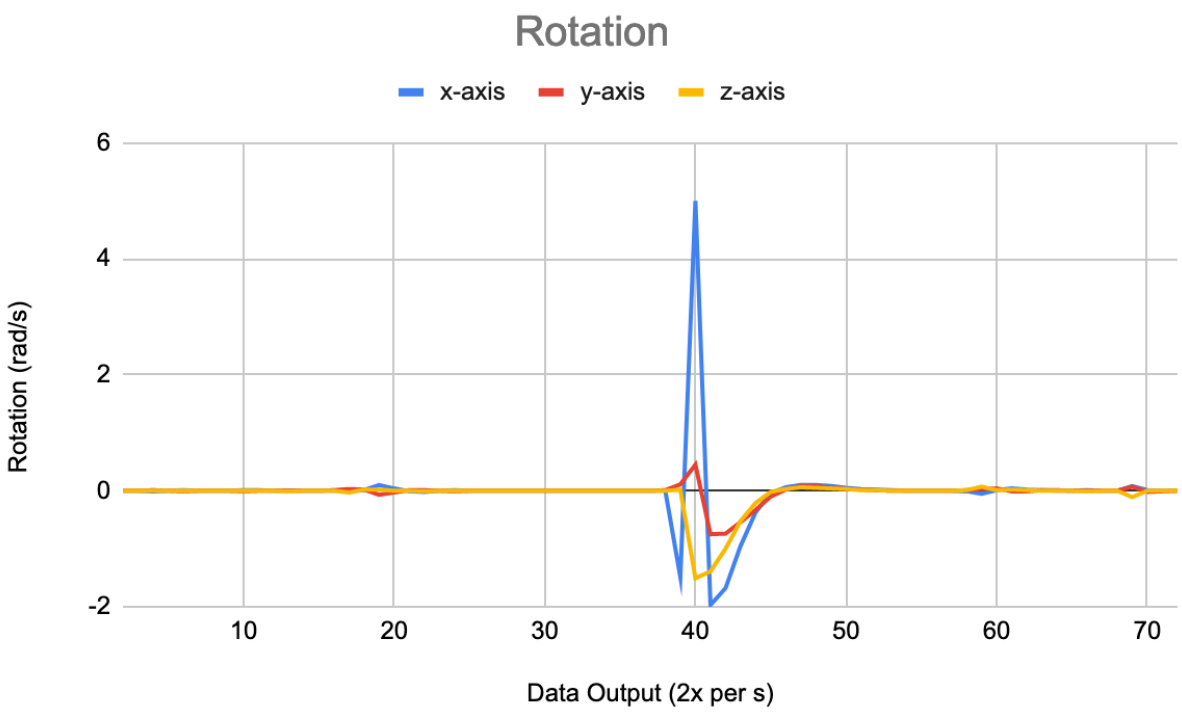
Rotation	x-axis	y-axis	z-axis

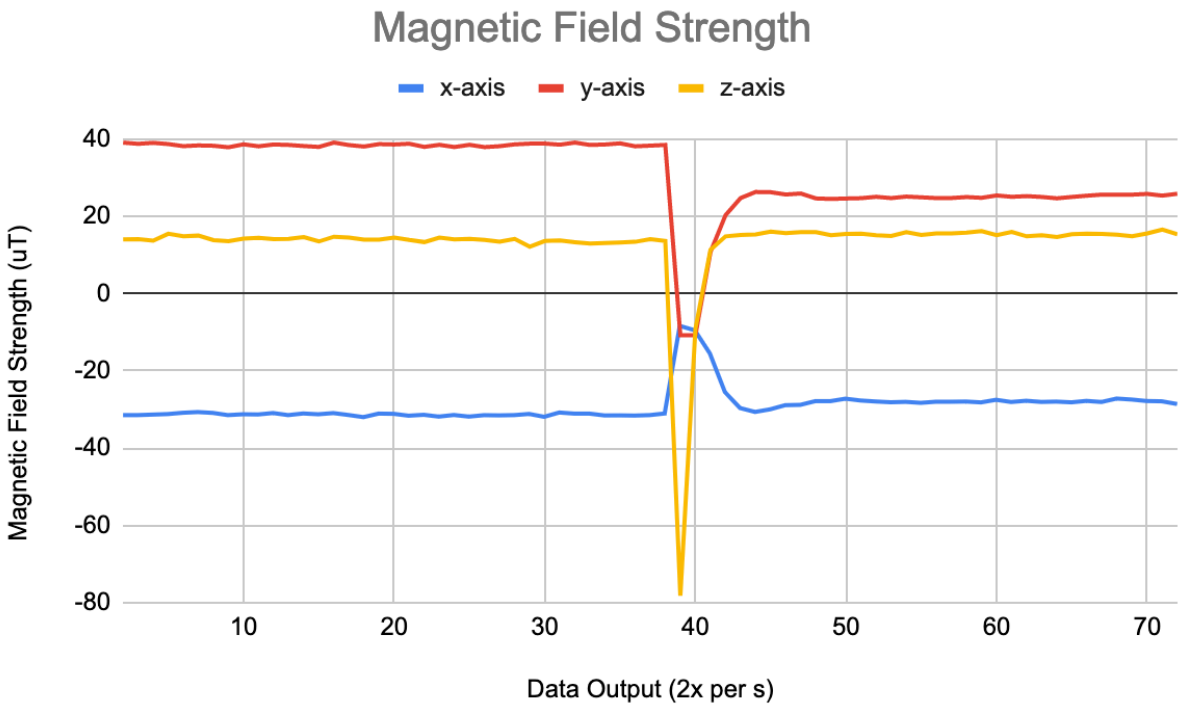
Minimum	-1.97	-0.75	-1.51
Maximum	5.0	0.45	0.07
Average	-0.01098591549295 7752	-0.02140845070422 535	-0.06211267605633 803
Standard Deviation	0.702248210462227 6	0.156369873675763 8	0.272778688053555 65

Magnetic Field	x-axis	y-axis	z-axis
Minimum	-32.13	-10.8	-78.18
Maximum	-8.52	39.05	16.56
Average	-29.1494366197183 12	30.60647887323943	12.93633802816901 5
Standard Deviation	4.170649926493859	9.968158192089017	11.28734443243587 6

We also plotted the values of the IMU data collected during this same test of sensors. All 9 data outputs show an abrupt change in the data from samples 38 to 40, or from 19 to 20 seconds into the test. During this change, acceleration along the x-axis decreased while acceleration along the y and z axes increased. Rotation along the x-axis increased, rotation along the y-axis increased slightly followed by a small decrease, and rotation decreased along the z-axis. The magnetic field strength along the x-axis increased, while the strength decreased along the y and z axes. For acceleration, rotation, and magnetic field strength, the behavior along the x-axis– increase or decrease– did not match that of the y and z axes. The way data changed abruptly for all 9 data outputs at these samples suggests a sharp change in Stitch's path of travel, such as an accidental drop by the team member carrying it. However, the team member's path was smooth and continuous, with the exception of a few pauses in movement. The reason, then, for this anomalous behavior could be due to a fault in the sensor but is otherwise undetermined.


Acceleration
x-axis
y-axis
z-axis
10
5
0
-5
-10
Acceleration (m/s^2)
10
20
30
40
50
60
70
Data Output (2x per s)


Rotation
x-axis
y-axis
z-axis
6
4
2
0
-2
Rotation (rad/s)
10
20
30
40
50
60
70
Data Output (2x per s)



Discussion

1. Stitch Construction Discussion

The group would classify Stitch as a work-in-progress: Though as a team, we had successful achievements with achieving lyft and a balanced takeoff, we would have liked a few more weeks to make Stitch truly flight-worthy. Currently, a few seconds after liftoff, the drone starts to drift and shake uncontrollably. When that happens we kill the switch or the failsafe kicks in, both resulting in a crash landing (see videos linked in appendix or in our respective journals). From our correspondence with experts, we have realized that the drift is expected. However, after numerous weeks of troubleshooting for the issue of stability, we have found the root cause to be in the Proportional-Integral-Derivative controller. This is an algorithm inside the flight controller that calculates and instructs motors to run at certain speeds based on sensor readings and controller stick commands. The PID controller attempts to correct for the difference between the gyroscope's real-time measurements and the ideal orientation as programmed by the flight controller, thus the choice of PID gains has a huge impact on the stability of the drone in flight. Due to lack of time and enough knowledge on PID controllers, our team was not able to tune the PID gains to match the specifications of Stitch and achieve a stable flight in time.

In addition to this, our test flights were affected by the levelness of the test site but also the sturdiness and stability of the landing gear and drone frame. The crashlands during our test flights severely damaged different components of the drone which required us to allocate time in fixing for more tests. Due to these reasons, we weren't able to pursue more promising methods of PID tuning such as using the flight controller's blackbox readings to estimate PID gains appropriate for Stitch. Tuning with this method would accomplish the next check in the process of readying Stitch for a proper flight, and the tuning would then need to be done again

as sensors are added. Thus, perhaps this state would be a good mark from which a future group could resume working on Stitch.

2. Angel construction discussion

While "Angel" never made it above a foot off the ground, its testing protocol produced significant wear and tear on the printed frame. As evident below the repeated crashing of the frame into the ground and stress of takeoff on the body resulted in fissures forming along the outer edges and even sections of the arms caving in. While the lack of landing gear on 'Angel" may have impacted this given the repeated flipping and crashing landing gear would not have been sufficient in protecting the body alone. The "flip and crash" of "Angel" is notable for two reasons. The first being as described in the discussion of "Stitch" when the flight controller senses the drone destabilize in air (wobbling too much or reaching a flip critical point) it will automatically shut down to prevent an accelerated crash into the surrounding area. The second being that at the time of testing the only control on the motors was the throttle test using the Speedy Bee app as a controller. This test did not involve any self-stabilization on behalf of the drone, limiting its flight capabilities. As discussed with "Stitch" the team experienced significant issues with the PID tuning which would likely have been encountered in Angel as well.

The initial wiring and testing of programs that run the IMUs went smoothly. Mounting Stitch's IMU on the drone was successful as well. The small size and low weight of the sensor provided us with few constraints on how to attach it. As a result, we were able to attach it directly to the HQV. This minimized any offset between the HQV's position and the IMU's position so that data collected by the IMU reflected that of the camera's movement as closely as possible.

Processing the raw IMU data, however, presented a great challenge. To reiterate, the purpose of the IMU is to find the HQV's position and orientation for use with photogrammetry. Knowing this would enable us to calculate the distance to the model UAP, and in the event of a genuine UAP encounter, displacement would be critical scientific information to gather. Investigating how to process the raw IMU data revealed that this is done using complex mathematics, which in turn requires complex programs. With numerous difficulties encountered during the ALIENS project, we were not able to calculate this position and orientation data. Given more time, processing this IMU data is a clear next step and something that ALIENS' IMU expert is interested in applying to other scientific projects.

Conclusions

For the success of the flight of the drones, please see the videos linked in the appendix. In some ways the drone engineering sub-team did not achieve their original goals of a controlled flight with sensors, but on a relative aspect they believe they made worthwhile progress with Stitch over the semester. The drone engineering sub-team for the alternative, 3D printed drone were able to successfully construct a quadcopter from a 3D model, with operational motors and sensor payload. While the flight objectives were not achieved due to time and technology constraints the team believes Angel to be a theoretical success despite lack of complete practical testing as all individual system parts were working as desired. As for the sensor payload, each sensor was working individually as expected and each Pi was proven to work headlessly when mounted to the drone and connected to a Pi battery. The template matching/disparity mapping part of photogrammetry is in progress, and we expect to continue testing the algorithm to find optimal parameters.

Appendix

1. The following is a Python script used to collect raw IMU data. It was adapted from code written by Adafruit:

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries

# SPDX-License-Identifier: MIT


import time

import board

from adafruit_lsm6ds.lsm6dsox import LSM6DSOX as LSM6DS

from datetime import datetime

from adafruit_lis3mdl import LIS3MDL


i2c = board.I2C()  # uses board.SCL and board.SDA

#i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT
connector on a microcontroller

accel_gyro = LSM6DS(i2c)

mag = LIS3MDL(i2c)

timestamp = datetime.now()


while True:

    acceleration = accel_gyro.acceleration

    gyro = accel_gyro.gyro

    magnetic = mag.magnetic

    print(timestamp)

    print("Acceleration: X:{0:7.2f}, Y:{1:7.2f}, Z:{2:7.2f}
m/s^2".format(*acceleration))

    print("Gyro          X:{0:7.2f}, Y:{1:7.2f}, Z:{2:7.2f}
rad/s".format(*gyro))

    print("Magnetic      X:{0:7.2f}, Y:{1:7.2f}, Z:{2:7.2f}
uT".format(*magnetic))

    print("")

    time.sleep(0.5)
```

2. The following is a Python script used to perform statistical calculations on raw IMU data.

```
#code written by Wes for ASTR202 sensor lab
#modified by Zoë

import csv
import numpy as np
from matplotlib import pyplot as plt; plt.ion()

#this file should be 3 columns (x,y,z) of straight-up numbers and
nothing else for only 1 measurement type at a time
#so only test acceleration, run it, then stop and switch out the
file name
filename = 'accel-stats.csv'

# Create 3 empty lists (in which to store the data)
columnX = []
columnY = []
columnZ = []

# Load data from the csv columns:
with open(filename) as csvDataFile:
      csvReader = csv.reader(csvDataFile)
      for row in csvReader:
      columnX.append(row[0])
      columnY.append(row[1])
      columnZ.append(row[2])

# Print the data from the two columns (uncomment for debugging)
#print(columnX)
#print(columnY)
```

```
#print(columnZ)


# Convert the data from the 3 columns into arrays:
# (so that we can plot them!) This also strips off the column
names.
columnX_arr = np.array(columnX[1:],dtype=np.double)
columnY_arr = np.array(columnY[1:],dtype=np.double)
columnZ_arr = np.array(columnZ[1:],dtype=np.double)


# Now plot column B vs column A:
#plt.plot(columnA_arr, columnB_arr,'o')
#plt.xlabel('A') # Label the x axis
#plt.ylabel('B') # Label the y axis
#plt.savefig('graph.png',dpi=150) # Save the plot.


# Now calculate the averages and standard deviations
#print('x average:',columnX_arr.mean())
#print('x standard deviation:',columnX_arr.std())


#print('y average:',columnY_arr.mean())
#print('y standard deviation:',columnY_arr.std())


#print('z average:',columnZ_arr.mean())
#print('z standard deviation:',columnZ_arr.std())


print('x min:',min(columnX_arr))
print('x max:',max(columnX_arr))


print('y min:',min(columnY_arr))
print('y max:',max(columnY_arr))
```

```
print('z min:',min(columnZ_arr))
print('z max:',max(columnZ_arr))
```

3. This script translates a message to morse code and transmits it at 2300 MHz using the Adalm Pluto, adapted from the testing script on the Adalm Pluto website.

```
import numpy as np
import adi
import time
import sys

sample_rate = 1e6 # Hz
center_freq = 2300e6 # Hz

sdr = adi.Pluto("ip:192.169.2.1")
sdr.sample_rate = int(sample_rate)

# filter cutoff, just set it to the same as sample rate:

sdr.tx_rf_bandwidth = int(sample_rate)
sdr.tx_lo = int(center_freq)

# Increase to increase tx power, valid range is -90 to 0 dB

sdr.tx_hardwaregain_chan0 = -10

N = 10000 # number of samples to transmit at once
t = np.arange(N)/sample_rate

# Simulate a sinusoid of 100 kHz, so it should show up at 915.1 MHz
# at the receiver

samples = 0.5*np.exp(2.0j*np.pi*100e3*t)
```

```
# The PlutoSDR expects samples to be between -2^14 and +2^14, not -1
and +1 like
# some SDRs

samples *= 2**14



def transmitDot():
    for x in range(10):
        sdr.tx(samples)
    time.sleep(1)

def transmitDash():
    for x in range(40):
        sdr.tx(samples)
    time.sleep(1)

def translate(message):
    morseDict = {'a':'.-',
                 'b':'-...',
                 'c':'-.-.',
                 'd':'-..',
                 'e':'.',
                 'f':'..-.',
                 'g':'--.',
                 'h':'....',
                 'i':'..',
                 'j':'.---',
                 'k':'-.-',
                 'l':'.-..',
```

```
            'm':'--',
            'n':'-.',
            'o':'---',
            'p':'.--.',
            'q':'--.-',
            'r':'.-.',
            's':'...',
            't':'-',
            'u':'..-',
            'v':'...-',
            'w':'.--',
            'x':'-..-',
            'y':'-.--',
            'z':'--..'}
for c in message:
    if c == ' ':
        time.sleep(7)
    elif c in morseDict:
        print('sending', c)
        morse = morseDict[c]
        for d in morse:
            if d == '.':
                transmitDot()
                print('sending dot')
            else:
                print('sending dash')
                transmitDash()
        time.sleep(3)
    else:
        print('invalid character')
```

```
def main():
    message = sys.argv[1]
    translate(message)


main()
```

- Code for photogrammetry: https://github.com/michelletu107/astr202-photogrammetry
- Stitch test flight videos: https://drive.google.com/drive/folders/1TSFpy12emX5R73Ret9BPn6GlOP0xiJQs?usp=share_link

Bibliography

- IMUs purchased: https://www.adafruit.com/product/4517
- Guide to using the Adafruit LSM6DSOX + LIS3MDL IMU, in addition to code used to collect data: https://learn.adafruit.com/st-9-dof-combo/python-circuitpython
- Photogrammetry sources: https://www.cs.cmu.edu/~16385/s17/Slides/13.2_Stereo_Matching.pdf

  https://www.baeldung.com/cs/disparity-map-stereo-vision