Quadcopter Delivery Verification

ECE 445 SP2016

TA: Luke Wendt
Team Members: Sachin Weerasooriya - Electrical Engineering
Raymond Hoagland - Computer Engineering
Rahul Joshi - Computer Engineering

Final Report for ECE 445, Senior Design, Spring 2016
May 4, 2016
Project No. 22

Abstract

Quadcopter Delivery Verification is a system, designed to be used in conjunction with current drone delivery models. This system uses a combination of landing pad encryption and threat detection schemes to provide additional robustness and security for the drone delivery process while remaining relatively inexpensive to implement. The landing pad is encrypted to provide authentication for the intended recipient, while a messaging server and a threat detection module provide a two-pronged approach to keeping packages safe from potential harm.

Table of Contents

1.0 Introduction

1.1 Statement of Purpose:

Traditionally upon ordering, recipients may choose to either have packages dropped off at a doorstep or request signature confirmation upon delivery. Services like Amazon PrimeAir look to speed up delivery times through the use of drones in the shipping process. Their current delivery model involves first attaching a package to the drone, taking off, and converting to a plane once a previously set altitude is reached. The drone then flies to the vicinity of the landing pad and converts back to a drone in preparation for landing on a marker, laid out by the recipient. The drone then deposits the package and returns to the factory for its next shipment. We believe this delivery model inherently contains a major flaw; in the event that something valuable is shipped, it is desirable to have a mechanism for confirming that someone is available to pickup the package immediately. Our project aims to solve this by creating a procedure that will allow a drone to wait for recipient verification before the package is delivered.

1.2 Objectives

1.2.1 Goals:

The goals of this project are threefold: design an embedded system on a drone which provides a mechanism for authenticated delivery, determine the intended landing location with the assumption that the drone is already in the proximity of the user's location, and have the drone wait at this location while avoiding potential threats to the package until user confirmation is received or a timeout period is exceeded.

1.2.2 Benefits:

This system aims to bolster the speed and efficiency of drone package delivery while maintaining the high standard of security allowed by standard delivery procedures requiring recipient signatures.

1.3 Usefulness of Project

Quadcopter Delivery Verification's intended use case is in tandem with an existing drone delivery model that leverages global position system (GPS) coordinates to maneuver the drone to the vicinity of the recipient's location. At this stage in the delivery process, Quadcopter Delivery Verification will become active, helping to guide the drone to the user-specific landing pad and successfully deliver the package. This will allow drones to deliver to more expensive and delicate items, which require a more secure delivery method.

1.4 Block Diagram

Figure 1 depicts an overview of the system and the interactions between the modules of this project.

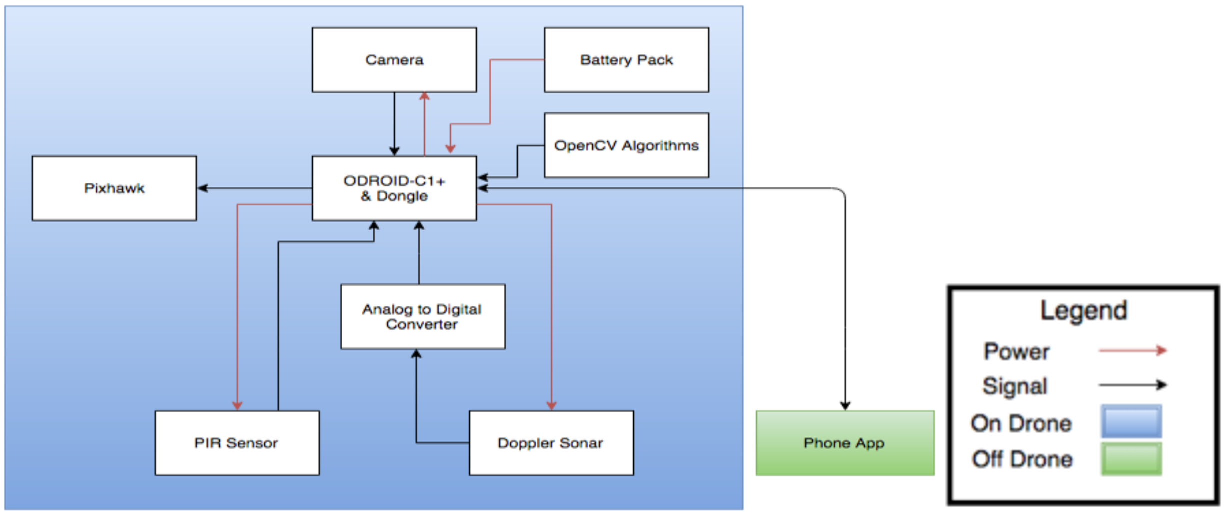


Figure 1: Block Diagram

2.0 Design

2.1 Block Descriptions

2.1.1 3DR Iris+ Drone:

This is the main component of the project which all other components are mounted to. The project leverages these components in addition to the onboard Pixhawk microcontroller to navigate the drone to the designated landing pad and deliver the package. When purchasing a drone, we had two major stipulations. The drone needed to use the Pixhawk microcontroller for flight control and have a large enough payload to carry our additional components. As the Pixhawk is relatively new, we decided to use the Iris+, which used the Pixhawk and is built by the same company that manufactured the Pixhawk controller itself.

2.1.2 Logitech C920 Camera:

The camera captures image frames, which are analyzed in OpenCV to inform decisions about the drones actions. Research showed that the majority of recent drone projects employed this camera due to its rich feature support in OpenCV and relatively inexpensive price. Other cameras supporting the necessary options for OpenCV; however, could easily be incorporated without changing the overall design of the system.

In order to capture adequate images in a variety of lighting conditions, we experimented with different values for the camera's brightness and exposure. For the purposes of our project, we consider standard lighting to be three fluorescent lights at approximately nine feet off the ground shining on our test-landing pad. In this lighting, we found that an exposure rate of 60% and brightness of 20% provided image frames with enough detail to be successfully decoded while avoiding oversaturation. Due to the potential for wing to cause the drone to slightly drift while hovering above the landing pad, we performed a

Gaussian Blur on the captured image, which allowed us to sharpen the image and remove small amounts of blur caused by the camera movement. We experimentally determined that a two second delay between image frames was ideal for our project, as per our requirements, each captured frame takes a maximum of four seconds to process using our decoding algorithm. This delay thus helps to reduce the instances where we process the same frame more than once. Additionally, because we expect the drone to be drifting quite slowly, this sampling rate is high enough to provide several frames with which we can attempt to successfully decode the landing pad. Finally, to provide a standard image size for all of the processing algorithms and remove any negative scaling effects, we set the camera to capture frames at 512x512 pixels(px).

2.1.3 Odroid C1+ Single Board Computer (SBC):

This is used as the main processing unit for the software portion of this project. The C1+ runs a Extensible Messaging and Presence Protocol (XMPP) server that sends a notification through Google Cloud Messaging (GCM) to the recipient's Android phone app to indicate the drone's arrival. The server also receives confirmation when the recipient is ready to pick up the package, prompting the drone to release the package and depart. Additionally the C1+ controls a sub-process to periodically capture image frames, a sub-process that analyzes the captured frames in OpenCV, and a MAVProxy shell used for guiding the drone. Finally, the C1+ runs a sub-process which analyzes the output of the threat detection system through the use of General-Purpose Input/Output (GPIO) pins and the onboard Analog Digital Converters (ADC). The C1+'s increased processing power, onboard ADC's, and compact design made it an immediate choice for this project; however, other similar SBCs could be used along with offboard ADCs, with the tradeoff being necessary integration of the two.

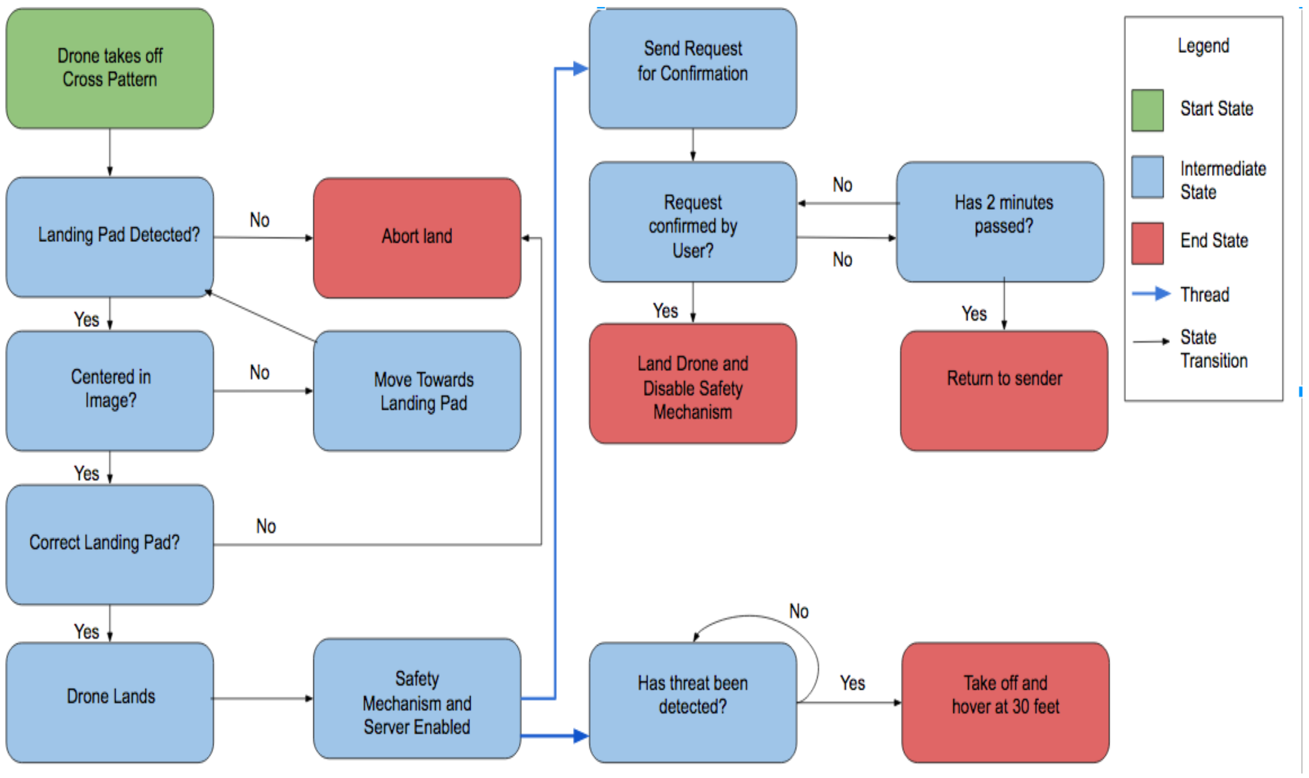


Figure 2: ODroid Logic FlowChart

2.1.4 OpenCV Algorithms:
The OpenCV algorithms are in charge of processing images to first center the drone above the landing pad and then crop out, normalize, and decode the landing pad.

We use base four to represent the result of a SHA-1 encoding performed on the user's unique identifier such as an email address when encoding the landing pad. Alternative approaches include different encoding schemes such as SHA-3 which is newer and thus less widely supported and binary encoding which has higher information density and thus may require further scaling of the landing pad in order to meet the same height requirements for decoding.

In order to center the landing pad in the capture image frame, we make use of the purple corners and center of the landing pad by applying Hue-Saturation-Value (HSV) filtering to find all purple pixels in the captured image and calculate the average x and y coordinates. We use an experimentally chosen 200x200 px dead zone in the center of our image (512x512 px) to represent the acceptable centered position.

Once the landing pad is in the center of the captured image, we crop it out so that only the landing pad remains. To do this, we find the outer white and black borders which identify the edges of the landing pad. These borders are represented by a pair of contours, or regions which consist of the same color, that have an area ratio between .85 and 2.0, where we expect the true ratio to be 1.2. We then crop around these contours, leaving the landing pad bordered by a single black rectangle. To make sure the outer black border is recognized as a contour, we create an outer 5x5 px white border surrounding it. We then rotate the contour and landing pad combination in increments of 3° using a rotation matrix (Figure 3), while recording the minimum ratio of areas between the contour itself and a bounding rectangle, which contains all points of the contour. If the minimum ratio falls below 1.2, where the ratio for a perfect 90° alignment is exactly 1, we finish this process by performing HSV filtering to locate the top left, top right, bottom left, and bottom right purple coordinates in the image and resizing the image to our standard size.

$$M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

Figure 3: Rotation Matrix [5]

Now that the landing pad has been isolated, we apply normalization to the image to account for any non-ideal lighting conditions. We use a combination of Contrast Limited Adaptive Histogram Equalization (CLAHE) and Red Green Blue (RGB) normalization techniques. For each pixel in the image, RGB normalization computes the average of the three color channels to more evenly distribute any saturation in the image (Figure 4). Following normalization, we decode the remaining image (Figures 5 and 6).

$$r = \frac{R}{R+G+B}$$
$$g = \frac{G}{R+G+B}$$
$$b = \frac{B}{R+G+B}$$
$$r+g+b=1$$

Figure 4: RGB Normalization [11]

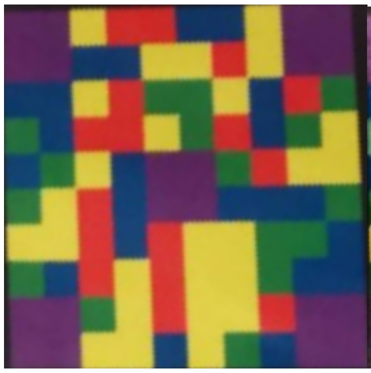


Figure 5: Original Image

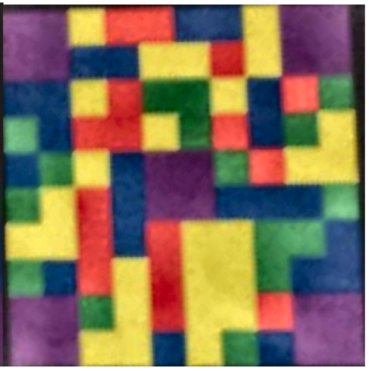


Figure 6: Normalized Image

When decoding the image we use HSV filtering to sequentially look for each of the encoding colors (Figure 7). For each color, we examine matching pixel locations in the image and map them to corresponding encoded blocks in the original image. To remain tolerant to minor rotation inaccuracies or other complications, we assign ties to the expected color. If two pixels within the same encoded block representing differing colors are examined, one of which corresponds to the block's color in the original image (Figure 5) and one which does not, the result is recorded as the expected color (Figure 8). During this phase, we consider all four possible 90° rotations to determine correct alignment. We then compare each of the four final results with the expected result using direct string similarity and record the best result. If the highest result is above the threshold of 0.80, we consider the landing pad to be successfully verified, and the drone begins landing.

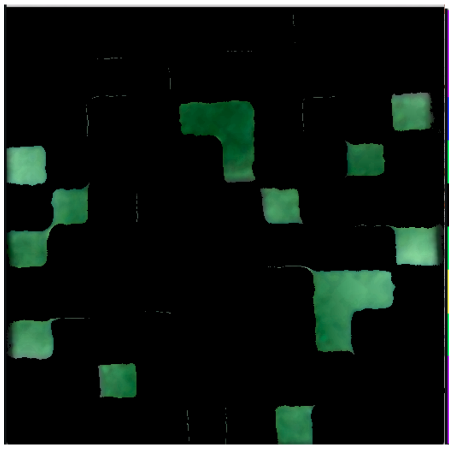


Figure 7: HSV Filtered Image

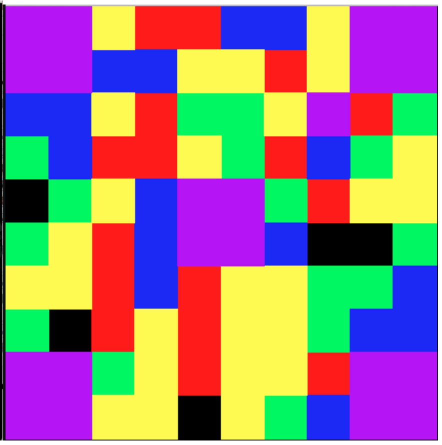


Figure 8: Final Reconstructed Image

2.1.5 Google Cloud Messaging (GCM) Server:
The GCM server is in charge of initiating a request for confirmation and waiting for the recipient's response to indicate that the drone should deliver the package (Figure 9). Alternatives to GCM exist, but its' ease of use and wide support made it an ideal candidate for this part of our project.

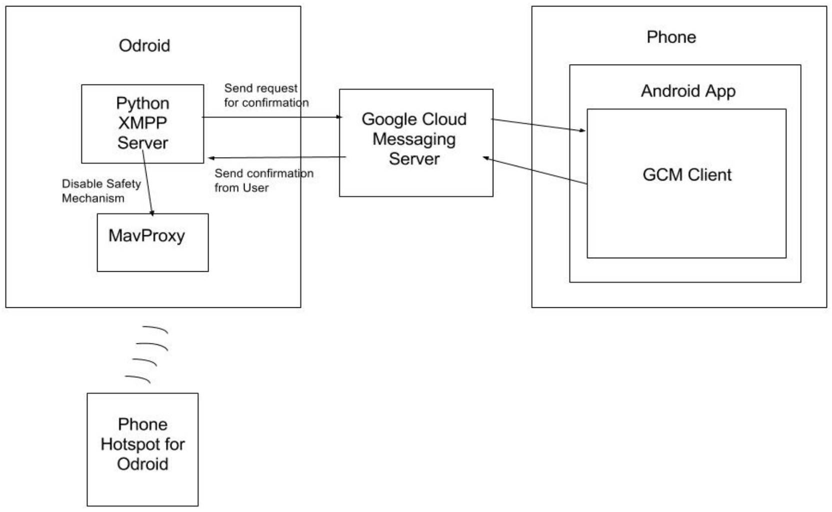


Figure 9: GCM Server Overview

2.1.6 Android Lollipop Phone App:
The phone app runs on the recipient's smartphone and receives the request for confirmation notification when the drone has successfully decoded the landing pad (Figure 10). When this notification is received, the confirm button becomes enabled, allowing the user to approve the delivery (Figure 11). At the time of the project's development, Android was easier to quickly develop on and was suitable for this project, however the app could be extended to iOS without changing the overall design.

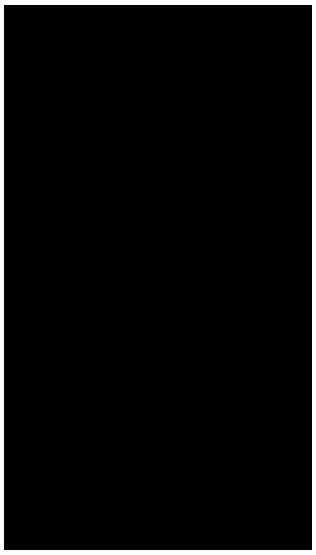


Figure 10: App (Disabled)

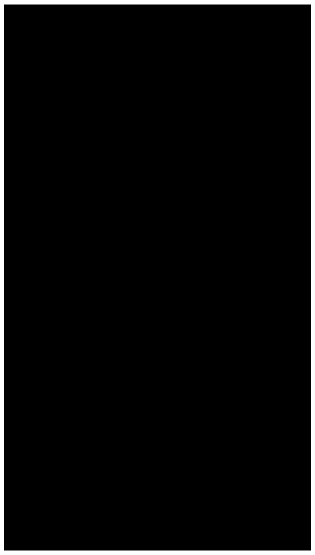


Figure 11: App (Enabled)

2.1.7 Passive Infrared Sensor (PIR) :
The PIR module is a device that looks for a change in the ambient infrared in the environment in order to detect movement. Figure 12 gives a visual representation of how the device works.

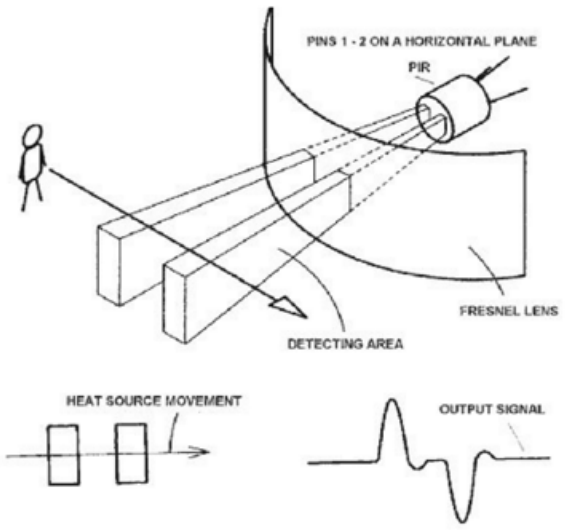


Figure 12: PIR Module Operation

Two strips of infrared (IR) sensitive material work in unison to form a differential. In a static environment, both strips should see the same amount of IR, resulting in a low voltage. When a warm body, such as a human, enters the field of view, initially only one strip will detect the IR which will cause a positive spike. Eventually, both strips should see the warm body, and finally only the other strip will see the signal, resulting in a negative spike. This device outputs a digital low when the differential is close to equal and a high value once this differential breaks a certain value.

As a result, we can use the output signal immediately without any amplification or filtering. The field of view of each sensor is approximately 90^o. We placed six sensors with even spacing around the drone so that no dead spots exist. If any of the PIR sensors are observed as digital high, we check the output from the Doppler on the same side of the drone. If both values are digital high within one second of one another a threat is detected. A basic schematic showing single connected PIR sensor is shown in Figure 13.

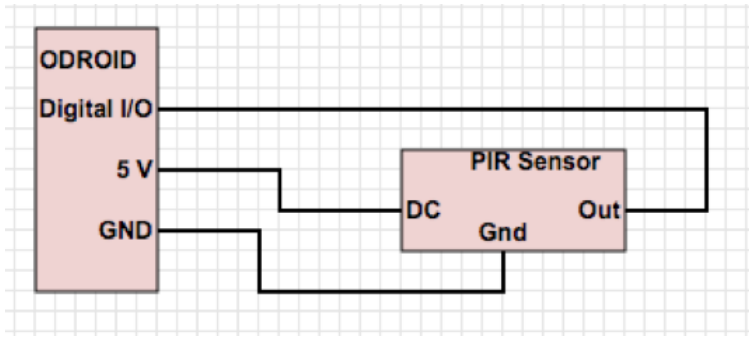


Figure 13: PIR Sensor Schematic

We use the source voltage and ground from the specially assigned 5V and GND pins on the ODroid C1+, which will be onboard the drone and is in charge of controlling the drone. We then feed the sensor output into one of the GPIO pins on the ODroid and compare this output using C to the output of the corresponding Doppler in order to determine whether there is a threat or not.

2.1.8 Microwave Motion Sensor:
The microwave motion sensor emits a high frequency signal at 10.525 GHz. Anytime the signal hits a moving object, the reflected signal is frequency shifted by a value that is linearly proportional to the speed of the object. This relationship can be seen below in Equation 1**.**

$$F_d = 2V\frac{F_t}{c}cos(\theta)$$

Equation 1: Doppler Shift Frequency

Note that in this equation F_d represents the shifted Doppler frequency, V represents the speed the object is travelling at, F_t represents the transmit frequency (10.525 GHz), c represents the speed of light (3.0 x 10^8 m/s) and theta represents the angle between the moving target and axis of the module. It should also be noted that the amplitude of this shifted signal is in the microvolt range so amplification is vital. As the doppler has a wider field of view and longer sensing range than the PIR sensor, only two sensors should be necessary to cover the entire 360^o.

Using Equation 1, we can determine what Doppler shifts to expect. As it is conceivable that a thief may try to approach the drone very slowly to fool the system, we can assume a situation where V is very small to be an important test case. If we take V to be .2 m/s and theta to be 45 degrees, we can see in Equation 2 that the corresponding shift is 4.8 Hz.

$$F_d = 2 * .2(m/s)) \frac{10.525GHz}{3 * 10^8 m/s} cos(45) = 4.8Hz$$

Equation 2: Doppler Shift Frequency Calculation 1

As a result, for this project we want to block frequencies under this threshold so as to avoid amplifying DC signals. The app note for our chosen Doppler, the HB100 [25], uses a first order resistor-capacitor circuit (RC filter) with a capacitor of 4.7uF and a 10K ohm resistor. We can see from Equation 3 below that the cutoff frequency for this low pass filter is 3.39 Hz.

$$F_{cutoff} = \frac{1}{2 * \pi * R * C} = \frac{1}{2 * \pi * 4.7 * 10^{-6} * 10^4} = 3.39Hz$$

Equation 3: Cutoff Frequency Calculation 1

This result is close to our desired cutoff in Equation 2, so we did not modify the resistor and capacitor values.

Our next step was to determine the appropriate gain and filter for the Doppler. In order to establish an upper band limit of frequencies to operate with, we again reference Equation 1**.** Because we expect our primary threats to consist of humans and small pets, we can infer that we can establish a maximum speed that our design can deal with. By assuming that the maximum speed a human can move at is 10 m/s, we can see in Equation 4 that our upper cutoff frequency should be 701Hz.

$$F_d = 2 * 10(m/s) * \frac{10.525GHz}{3 * 10^8(m/s)} cos(0) = 701Hz$$

Equation 4: Doppler Shift Frequency Calculation 1

We then chose R and C values to achieve this cutoff. In order to keep this project inexpensive, we aimed to maximize our resistance value while minimizing our capacitance value. The App sheet used a 2.2 pF capacitor along with a 1M ohm resistor. Keeping the 1M ohm resistor, we can see that a 225 pF capacitor is ideal for our needs, as shown in Equation 5.

$$F_{cutoff} = \frac{1}{2 * \pi * R * C} = \frac{1}{2 * \pi * 225 * 10^{-12} * 10^6} = 707.36Hz$$

Equation 5: Cutoff Frequency Calculation 2

The app note uses a gain that aims to accurately detect human movement. Because we did not change the resistor in our RC filter, the gain after exiting the two-stage opamp is maintained. As a result, we carried over the rest of the circuit unchanged from the app sheet. The app sheet schematic is shown below in Figure 14.

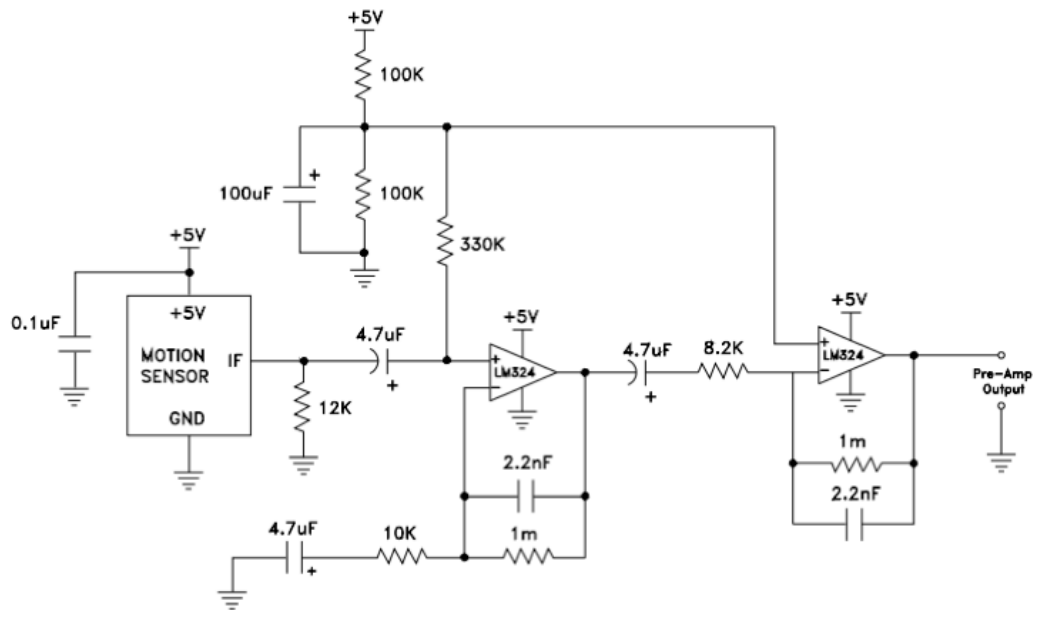


Figure 14: HB100 Pre-Amplifier Circuit

Figure 15 shows the modified circuit design used in our project. Our drone uses two dopplers, which means two copies of the same amplifier circuit are needed, but the second copy is not shown for brevity.

Finally, we send the amplified output signals to a 30-pin header. The output from the second stage of the amplifier is sent through a voltage divider to ensure that the output never exceeds 1.8V, the maximum permissible input value allowed by the ADCs on the ODroid. We then take the amplified signal from the PCB and deliver it to one of the two ADCs found on the ODroid. This signal once digitized undergoes a similar process as the PIR sensor output. However, in this case more logic must be implemented in software because the output signal is not restricted to a digital high or low. Depending on which side of a threshold the output voltage falls, we determine whether the signal should be treated as digital high or low. We explain how these thresholds are established in Section 2.2.10.

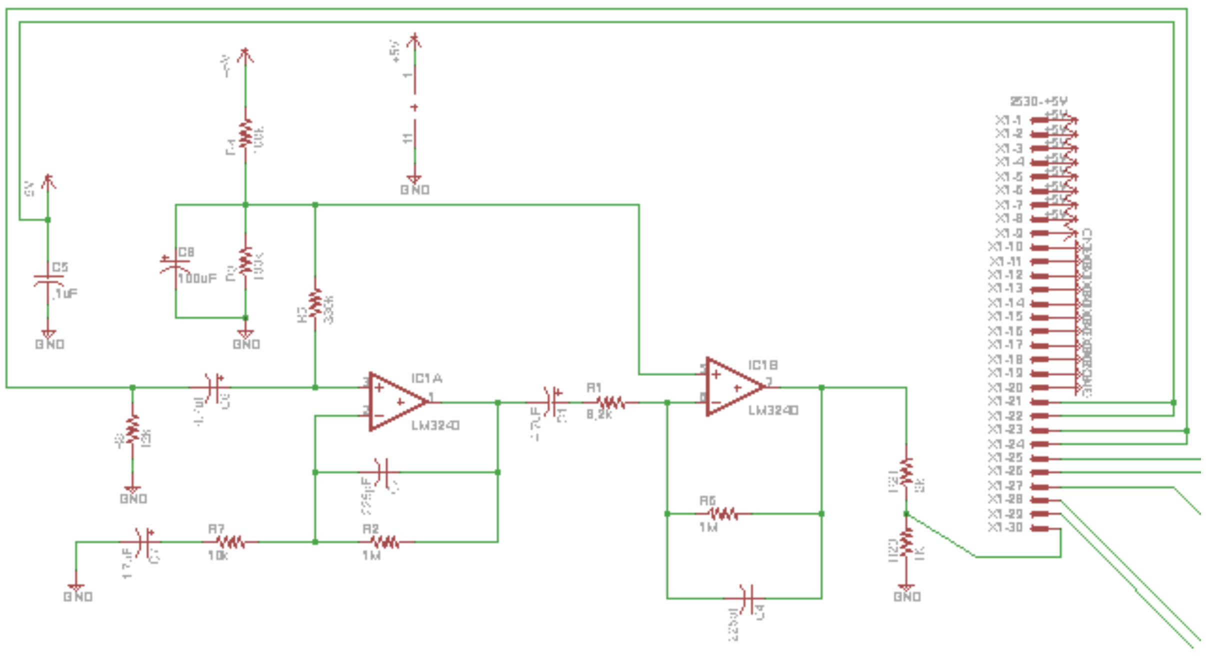
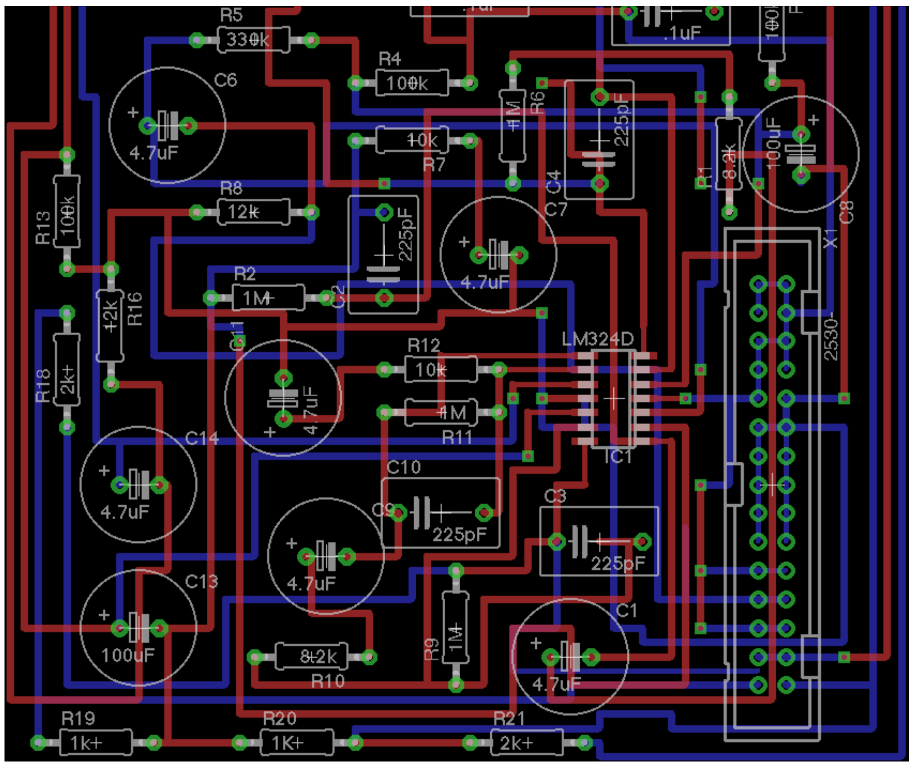


Figure 15: Designed Pre-Amplifier Circuit

2.1.9 PCB/Perfboard Design

Despite going through two revisions for the PCB, due to unidentified complications, we completed the circuit on perfboard for demo purposes. Both the PCB and perfboard are shown below (Figure 16).



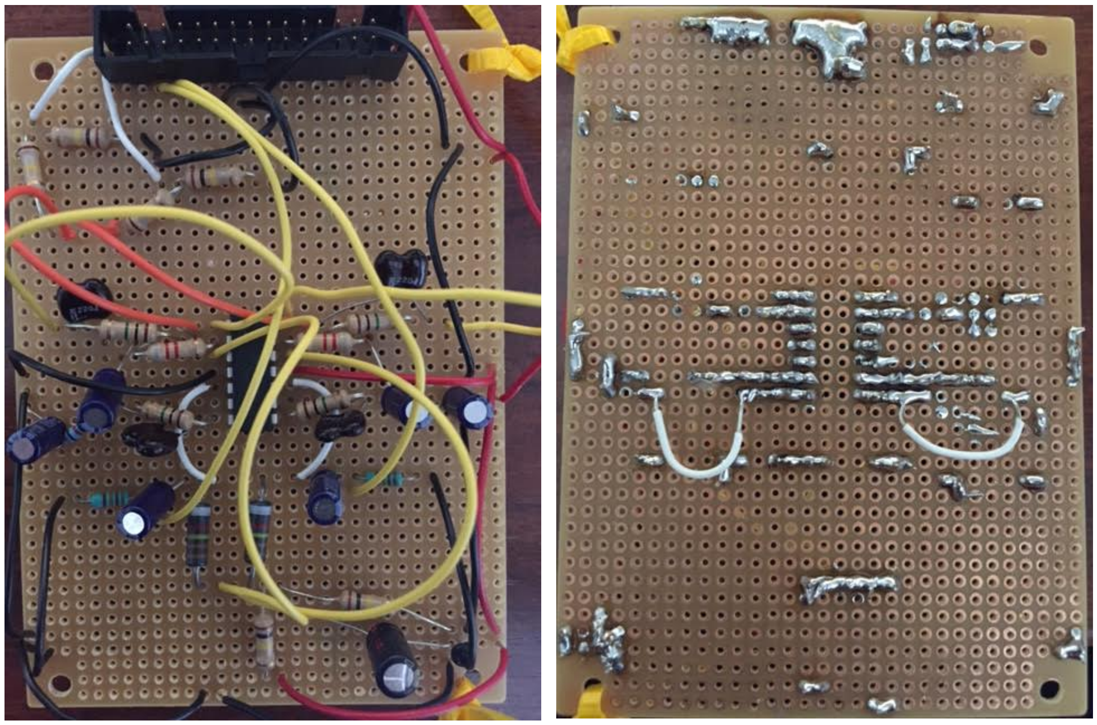


Figure 16: PCB Layout/ Perfboard

2.1.10 Threat Detection System

As we have two methods of detecting motion, we combine them to produce a more accurate output with fewer false positives and false negatives. We accomplish this by placing sensors around the drone; in particular, six PIR sensors and two dopplers. We positioned a PIR sensor on the front and back of the drone along with two more each on the left and right side of the drone. In addition, there is a single doppler each on the left and right side.

We then designed a script in order to monitor the output of both types of sensors. For the purposes of the demo, we only fitted and monitored one side of the drone with sensors in order to reduce noise from other groups or other sources in the lab. We monitored both PIR sensors and the doppler on the side of the drone. We defined a threat as a situation in which both sensors output a digital high within the same second of each other. We used an infinite loop to monitor the GPIO pins which handled the PIR sensor inputs and monitored the doppler output; if the doppler output exceeded 1.6V, it was considered "high".

3.0 Verification

3.1 Testing Procedure

When ordering the drone for our project we encountered significant delays and thus chose to verify our components in a modular fashion. This assisted us in two major ways, first it allowed us to maximize the number of points earned during the demo and second it facilitated an easier method for testing our design both prior to and following the integration of the various modules. A more detailed description of the requirements and verifications used for this project is located in Appendix A.

3.2 Power Module

The key requirement of the power module was that the device supply the ODroid with the 5V, 2A biasing necessary for the device to remain powered for at least the length of the drone's maximum battery lifetime of twenty-five minutes. For validation, we powered the ODroid with all peripheral sensors attached via battery pack and stopped once the test reached the one hour mark as it became clear that the supply was more than sufficient for our purposes.

3.3 PIR Sensor

Because the PIR sensor didn't require any pre-amplification or filtering, the testing procedure for the device was fairly straightforward. We connected the Vin and ground (GND) pins to the 5V and GND pins respectively on an Arduino Uno board. I then passed the digital output through a grounded resistor in series with an light-emitting diode (LED). We then approached the sensor to test the claimed seven meter range.

We immediately noticed an issue wherein the sensor would output a high voltage and stay high even without any movement. In order to rectify this, we adjusted one of the device knobs which tuned how long the sensor output remained digital high after seeing motion. After reducing the sensor delay, we observed the expected behavior. We then tested the range of the sensor by first evaluating at which distance the sensor could first detect our movement. At 20 feet away, the sensor immediately detected our movement when coming into the sensor's field of view, which we observed to be approximately 90 degrees, as claimed.

3.4 Microwave Motion Sensor

To verify this module, we first assembled the amplifier circuit on a breadboard and wired the doppler to the amplifier circuit. We then observed the output signal to determine whether it had a constant output signal along with oscillations as motion is detected. Once we confirmed that the circuit worked, we proceeded to assemble the PCB. We then attached the Doppler to the PCB to ensure the output emulated the breadboard circuit's results.

After first connecting the doppler to the breadboarded circuit, we observed a steady 2.5V output signal as expected but no oscillations in response to movement. Because the active filtering circuit was largely borrowed from the HB100 application sheet, we surmised that it was likely that the circuit itself was correct and that the error was a minor wiring issue or a faulty chip. We thus took the following steps to debug.

We first removed the LM324 op amp from the circuit and put it on its own breadboard. At this point, we connected the op amp as a voltage buffer in which the output is fed back into the negative terminal and 2.5V is fed into the positive terminal of the op amp. We observed the output voltage to be 2.5V as expected. We repeated this process for the other op amp on the chip, which led to the same result. We therefore ruled out the possibility of a faulty LM324 chip.

Our next step was to check that the expected gain from each opamp was achieved. We accomplished this by removing the capacitors on the feedback loop of each op amp in order to remove the filtering. A 1mV input was then fed into each stage individually and the expected output was observed from both cases. From [22], we see that the gain for the first stage was expected to be 101 and the gain for the second stage 122. Once we verified that this gain was correctly output, the capacitors were then re-connected and the input to the op amp connected to a signal generator. At this point, we generated input signals at various frequencies inside and outside the pass-band filter in order to determine whether the signal was attenuated as expected. Figures 17, 18, and 19 show the output of these input signals with various frequencies.

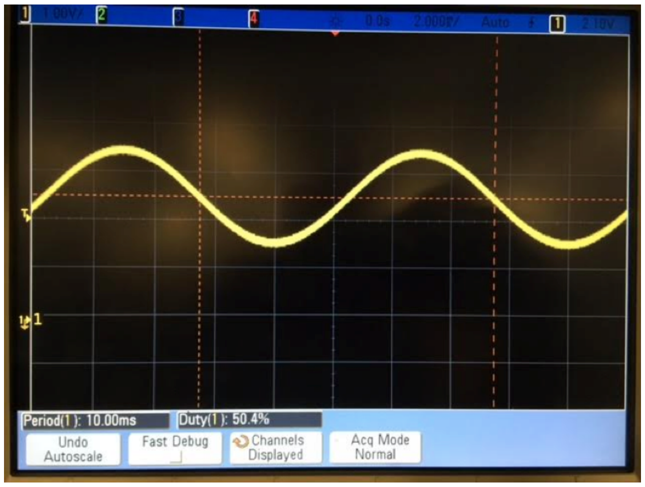


Figure 17: Output after a sinusoidal input of freq = 300 Hz (inside passband) with a peak voltage of 10 mV

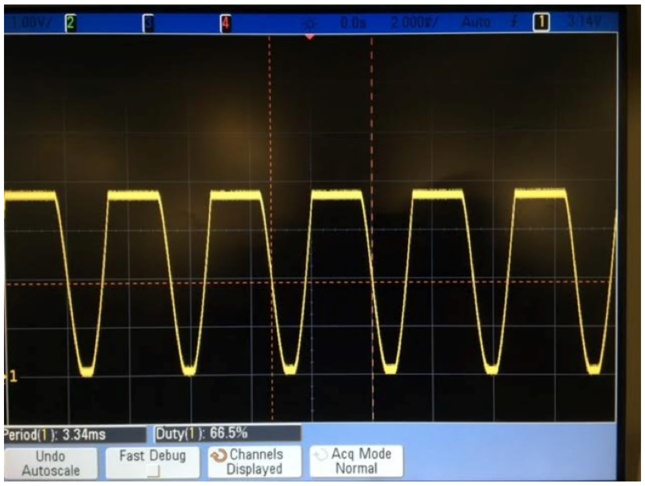


Figure 18: Output after a sinusoidal input of freq = 300 Hz with a peak voltage of 50 mV

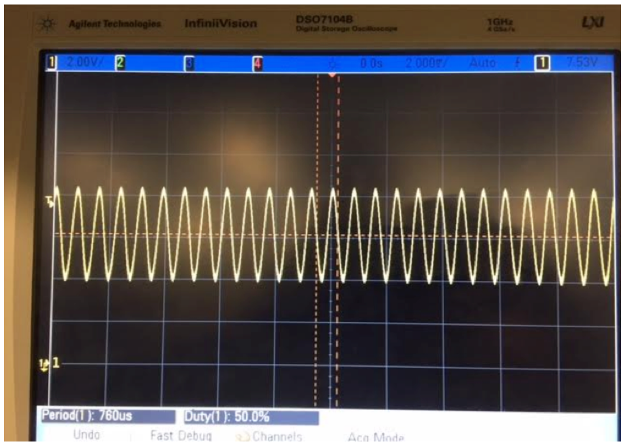


Figure 19: Output after a sinusoidal input of f = 1300 Hz (outisde of passband) with peak voltage of 50 mV

We observed that these plots were all correct which ruled out many potential errors. At this point, we re-connected the two stages and re-connected the doppler to the circuit. This time, the output responded to motion as anticipated. This was somewhat strange as we did not modify the actual circuit design. As a result, we believe the circuit likely had a loose or misplaced connection which was rectified during the testing of each stage.

After verifying the circuit functioned correctly, our next step was to verify that the range of the device was as expected. To test this, a person stood over 20 feet away from the sensor and began walking towards it. We then monitored the signal output on the oscilloscope to see the response. The signal was observed to attenuate when a walking target was within 15 feet of the target. We repeated this at various angles ranging from -90 to 90 degrees to the device and the sensor sensitivity remained relatively unchanged. We then approached the sensor at sprinting speed (10mph) to see if the frequency of the output signal was higher than the frequency of the output signal at walking speed (3mph). Figures 20 and 21 demonstrate the doppler output for both of these cases.

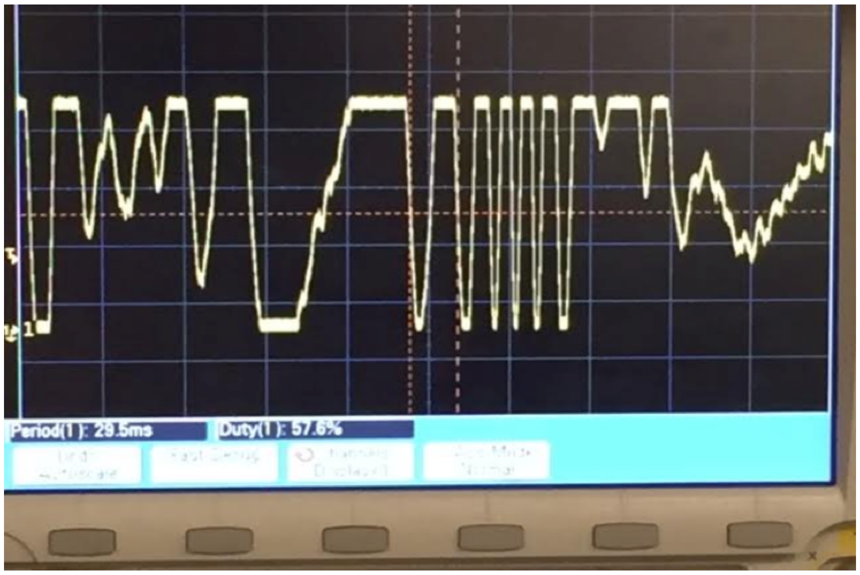


Figure 20: Approaching Sensor at walking speed

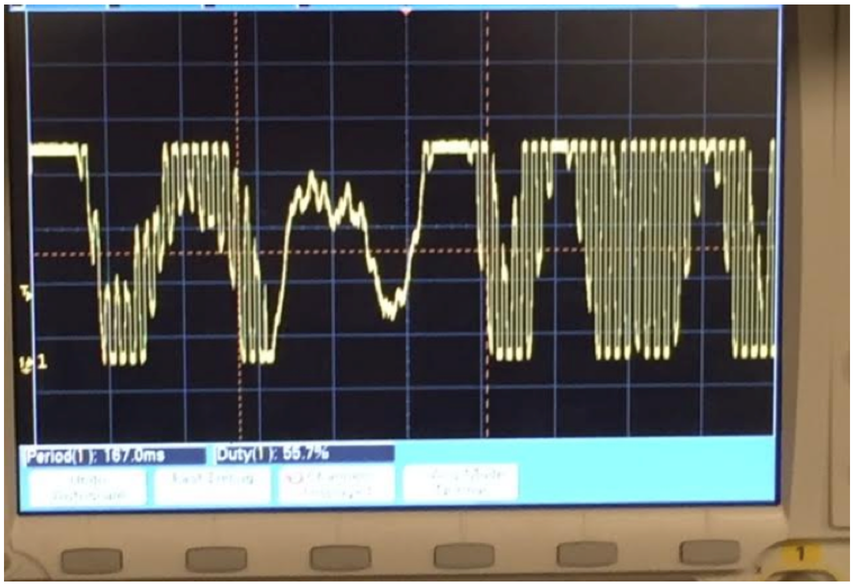


Figure 21: Approaching sensor by sprinting

3.5 Analog to Digital Converter

We used the ADCs in our project to digitize the analog output from the doppler amplifier circuit. Because we know what the doppler output looks like, the key step was to validate that the oscilloscope data matched the data recorded by the ADC. We accomplished this by connecting the amplified ADC output to both an oscilloscope and the ADC. We then stored, exported to Excel, and plotted the first 10,000 samples from the ADC. We compared the outputs of both to ensure a match (Figure 22).

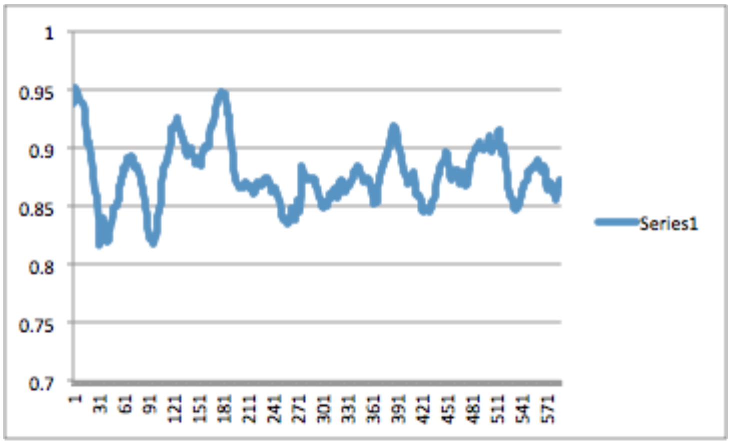
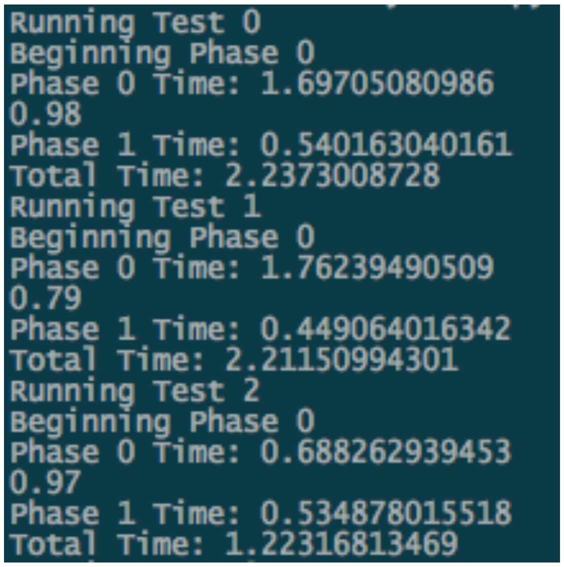


Figure 22: ADC Plot

3.6 OpenCV

The first requirement for the OpenCV algorithms specify that the decoding steps should complete with a minimum speed of .25fps, or that each frame must be completely processed within 4 seconds. As observed in our tests, the processing times ranged between 1.098s and 2.237s, well below our threshold of 4 seconds (Figure 23). Additionally, we found that the accuracy at which the landing pad could be decoded ranged from 65% to 98% for our test images, which were captured in our standard lighting conditions.



```
Running Test 0
Beginning Phase 0
Phase 0 Time: 1.69705080986
0.98
Phase 1 Time: 0.540163040161
Total Time: 2.2373008728
Running Test 1
Beginning Phase 0
Phase 0 Time: 1.76239490509
0.79
Phase 1 Time: 0.449064016342
Total Time: 2.21150994301
Running Test 2
Beginning Phase 0
Phase 0 Time: 0.688262939453
0.97
Phase 1 Time: 0.534878015518
Total Time: 1.22316813469
```

Figure 23: Example Decoding Times/Accuracy

The second verification for the OpenCV algorithms was to ensure that height calculations were 85% (+/- 5%) accurate. We observed that in our standard lighting conditions, the average accuracy was 97.35% which was more than enough to meet our requirements (Figures 24 and 25).

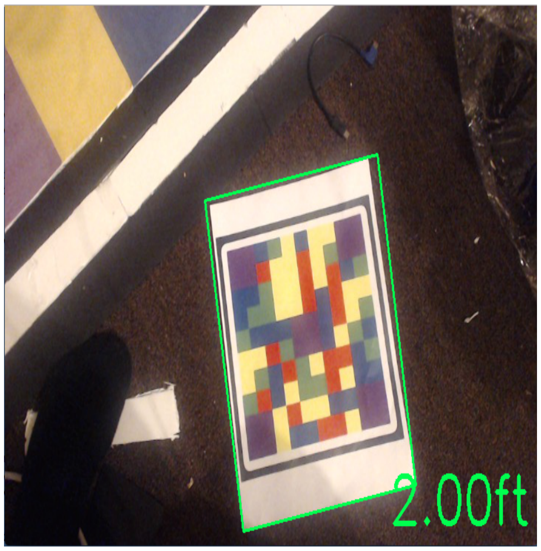


Figure 24: Known height

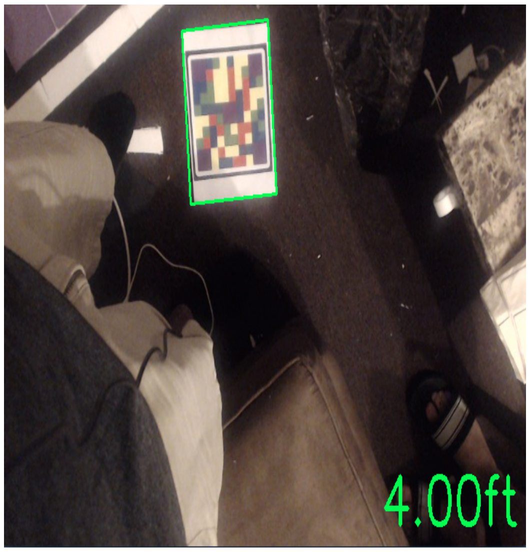


Figure 25: Estimated height (100% accuracy)

3.7 MAVProxy

We were able to verify that we could control the drone through MAVProxy using simulations (Figure 26). Unfortunately, we were unable to verify that we could land the drone at the center of a landing pad due to complications with the DF-13 connector. The lack of this connector prevented us from testing our ODroid with the drone and thus from testing the MAVProxy code with the OpenCV algorithms.

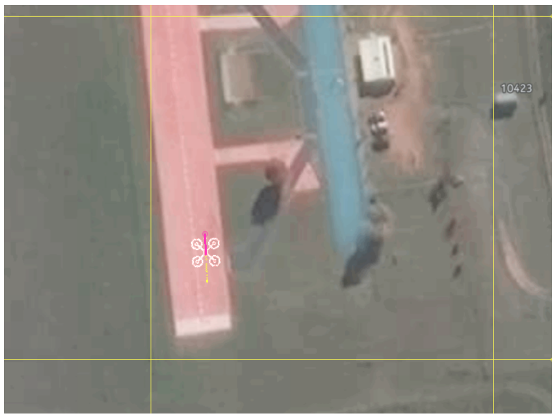


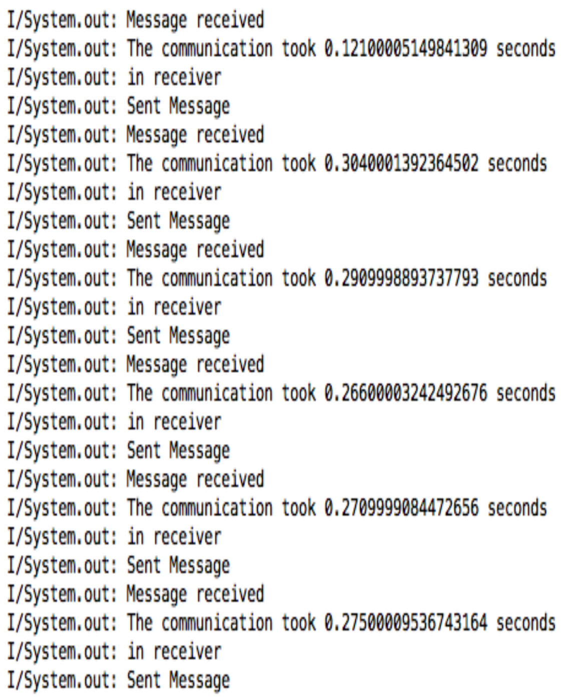
Figure 26: MAVProxy SITL Simulation

3.8 Android Phone App

The main requirements for the phone app were that it work on a smartphone running Android Lollipop and that messages sent between the app and the server take less than 1 minute.

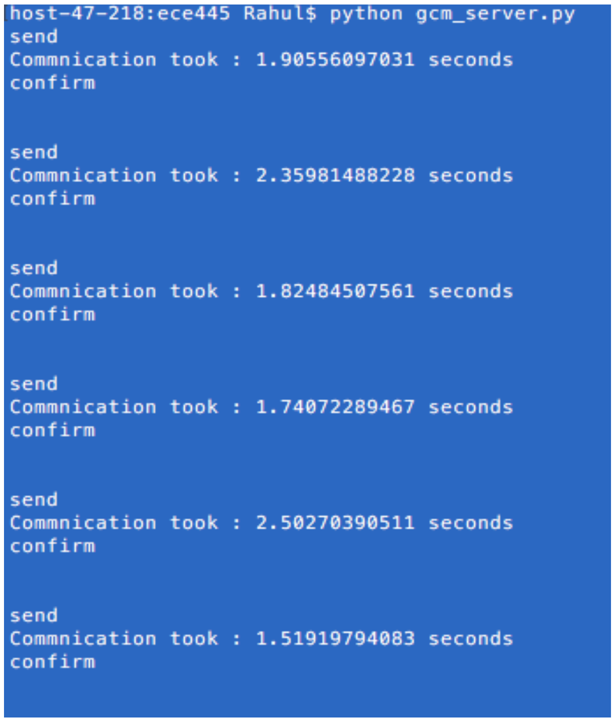
The app worked successfully on both a Genymotion simulation and a LG G3 running Android Lollipop, which can be seen above. (Figures 10 and 11).

We observed that both the request for confirmation and confirmation messages between the ODroid server and the app took significantly less than one minute to complete, within the range of 0.121 and .304 seconds from the ODroid to the app, and between 1.519 and 2.503 seconds in the opposite direction (Figures 27 and 28).



```
I/System.out: Message received
I/System.out: The communication took 0.12100005149841309 seconds
I/System.out: in receiver
I/System.out: Sent Message
I/System.out: Message received
I/System.out: The communication took 0.3040001392364502 seconds
I/System.out: in receiver
I/System.out: Sent Message
I/System.out: Message received
I/System.out: The communication took 0.2909998893737793 seconds
I/System.out: in receiver
I/System.out: Sent Message
I/System.out: Message received
I/System.out: The communication took 0.26600003242492676 seconds
I/System.out: in receiver
I/System.out: Sent Message
I/System.out: Message received
I/System.out: The communication took 0.2709999084472656 seconds
I/System.out: in receiver
I/System.out: Sent Message
I/System.out: Message received
I/System.out: The communication took 0.27500009536743164 seconds
I/System.out: in receiver
I/System.out: Sent Message
```

Figure 27: ODroid to App Communication times



```
host-47-218:ece445 Rahul$ python gcm_server.py
send
Commnication took : 1.90556097031 seconds
confirm


send
Commnication took : 2.35981488228 seconds
confirm


send
Commnication took : 1.82484507561 seconds
confirm


send
Commnication took : 1.74072289467 seconds
confirm


send
Commnication took : 2.50270390511 seconds
confirm


send
Commnication took : 1.51919794083 seconds
confirm
```

Figure 28: App to Odroid Communication times

3.9 PCB/Perfboard

The verification process for the PCB design was a time-consuming and unsuccessful process. The first PCB design contained several problems that we discovered during the verification process. For starters, some traces on both the top and bottom layer led to the header pin, making it nearly impossible to solder each pin on the component properly. We left the shunt capacitor between the 5V biasing voltage for the Doppler floating as opposed to connected to the header pins, which the Doppler would be connected to. The first revision also lacked the voltage divider at the output required to step down the 5V output to 1.8V. These corrections were remedied in the second revision, but we were still unable to get the second PCB working as intended.

Once the second board was soldered, our first step was to connect the board to a 5V source and ground in order to verify any short circuits were present in the circuit. We initially connected the circuit to a power supply and as soon as both terminals were connected to the header pins of the circuit, the current draw from the voltage source spiked to nearly 1A, which was clearly incorrect as the verified breadboard circuit only drew 38mA. As a result, we determined that some of the components were soldered poorly, causing a short circuit. Using the connection tool on a multimeter, we were able to find all the sources at which the solder undesirably connected signals.

Once we found and fixed all of these short circuits, the PCB was re-connected to a power supply and this time drew the expected 38mA. The Doppler was then connected to the PCB but the output did not match the output from the breadboard circuit so we surmised that further errors must still exist. After probing voltages at various parts in the circuit and comparing it to the probed voltage on the breadboard circuit, we recognized that the voltage on the ground pin of the op amp was at 0V on the breadboard as expected but 4.4V on the PCB. We later determined that this occurred because the IC was mounted upside down; as a result, 5V from the supply was connected to the ground pin of the IC and vice versa. When we desoldered and placed the IC correctly, the probed voltage at the ground pin returned to 0V as expected.

At this point, we re-connected the Doppler but again the output was incorrect. This time however, there was a DC offset, a desired behavior. We observed this offset to be 3.3V; however, as opposed to the expected .9V with the step down voltage divider in place. Despite several hours of debugging, we were never able to determine the cause of the problem. As a result, we were forced to move our design to a perfboard for demonstration purposes. We soldered the components onto the perfboard and when the Doppler was connected, the output matched the output from the breadboard circuit with no need for additional debugging.

3.10 Threat Detection Script

Because we knew both sensors worked correctly on their own, our validation for the threat detection script remained fairly short and simple. In order to test that both inputs were being analyzed, we disconnected the Doppler first, followed by both PIR sensors. During each of these tests, we wanted to ensure that a threat was never detected because although the sensor that remained connected would invariably go high if there was motion, a threat should never be detected because one type of sensor was disconnected. This test worked. When both sensors were connected, the logic also worked as desired.

4.0 Costs

4.1 Parts

Item	Cost
3DR Iris+	$599
Logitech C920	$70
PIR Motion Sensor (HC-SR501) (4)	$8.49 ($33.96)
Force Sensor (FSR_0.5_DE2) (4)	$6.26 ($25.04)
Minzi Battery Pack- 4600 mAh - 5V/2A	$19.99
Odroid C1+	$50
WiFi Dongle	$0 - already own
100K Resistor (8)	$0 - already own
1M Resistor (8)	$0 - already own
8.2k Resistor (4)	$0 - already own
12k Resistor (4)	$0 - already own
10k Resistor (4)	$0 - already own
.1 uF Capacitor (4)	$0 - already own
100 uF Capacitor (8)	$0 - already own
225 pF Capacitor (12)	$0 - already own
LEDs (3)	$0 - already own
LM324 Opamp (3)	$1.63

30 Pin Header	$6.80
Zip Ties (100)	$7.99
Total	$778.00

Table 1: Parts Costs

4.2 Labor

Name	Hourly Rate	Total hours invested	Total
Raymond Hoagland	$50	200	$25000
Rahul Joshi	$50	200	$25000
Sachin Weerasooriya	$50	200	$25000
Total			$75000

Table 2: Labor Costs

4.3 Total

Section	**Total**
Labor	$75000.00
Parts x 2	$1556.00
Total	$76556.00

Table 3: Total Costs

5.0 Conclusion

5.1 Accomplishments

We achieved significant results in three main areas of the project: the OpenCV algorithms, threat detection system, and user application and server.

In standard lighting conditions, we were able to decode test landing pads with 75% accuracy. In brighter lighting conditions such as the outside environment, we originally saw very low accuracy similar to that of an incorrect landing pad in the 30%-40% range, as the bright light saturated the channels of the image, making it much harder to accurately apply HSV filtering. The addition of normalization techniques helped to increase this accuracy, but we needed to adjust the expected colors to account for the lighting saturation. Additionally, we were able to correctly determine in which direction to move the drone to center the landing pad in the capture image frame with 65% accuracy.

For the threat detection system, we were able to develop a script that accurately determines if there is a threat or not. If the PIR sensor output a digital high and the Doppler was determined to output high as defined in the design stage within the same second of one another, a threat was detected. We were able to successfully detect human threats approaching the drone before the threat was within 15 feet of the drone as we originally intended. We also observed very few false positives and false negatives with this module. This however was accomplished in controlled, low noise environments. Going forward, we would like to test the system outdoors in noisier environments to ensure that the system is robust enough for real-world applications.

For the Android app, the Python XMPP server on the ODroid was able to successfully communicate with the Android app through the GCM client within one minute as per our requirements.

5.2 Uncertainties

Although we were able to achieve results in the three main areas of this project, there were some uncertainties due to delays and complications with receiving the drone. The first major uncertainty was that we were unable to verify that the MavProxy controls worked on the drone because the ODroid could not be correctly plugged into the drone using the obscure DF-13 connector. The second major uncertainty was our inability to verify that all of the modules could work together as a completely integrated system due to these same delays.

5.3 Ethical Considerations

The project presented in this design review will be implemented with guidance and recognition of the IEEE code of ethics.

1. To accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment

The goal of this project is to use a drone to locate a landing pad encoded using a unique identifier for a customer and safely deliver a package using this drone. When working on this project we must focus on safety of the customer and others around the drone. As we develop our system for the drone, anyone who may be affected will be apprised of any safety concerns as they arise.

2. To avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist

Due to the wholly academic nature of this project and the lack of sponsors or other motives, there is little concern of conflicts of interest. That being said, if any such conflicts are encountered, all parties affected by this project will be informed of these developments immediately.

3. To be honest and realistic in stating claims or estimates based on available data

The data that is collected during the course of this project will not be falsified in any manner and shall be analyzed in strict accordance to the requirement and verification table provided (Appendix A).

4. To reject bribery in all its forms
As stated above, the academic nature of this project makes it unlikely that we will receive bribes, however, all potential briberies will be summarily rejected.

5. To improve the understanding of technology; its appropriate application, and potential consequences
The ethical and safety measures outlined in this paper will guide us in understanding the proper applications and consequences of the use of the technologies employed in the project.

6. To maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations
This team has a diverse set of skills and each member will contribute to the project based on his or her experience. The two general areas which will be focused on during this project are software development and circuit design.

7. To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to credit properly the contributions of others
The advice from professors and TA's will be taken into full account during the extent of this project. The sources for each and every idea that is not wholly our own employed in the project will be properly cited.

8. To treat fairly all persons and to not engage in acts of discrimination based on race, religion, gender, disability, age, national origin, sexual orientation, gender identity, or gender expression
All persons within the team will conduct themselves in a proper manner, both when interacting with other team members and with other individuals.

9. To avoid injuring others, their property, reputation, or employment by false or malicious action
The team will take all necessary precautions by following the safety measures outlined below when working on project testing and will not engage in any malicious actions towards others.

10. To assist colleagues and co-workers in their professional development and to support them in following this code of ethics
This ethical statement is to help this team follow the IEEE code of ethics and also to help other teams also uphold such a code.

5.4 Delays and Complications
The project was encumbered by several delays and complications that led to our inability to fully complete certain aspects of the project.

The first delay was the delivery of the drone itself. We initially ordered a drone called FlyTrex Sky in early February, but the company repeatedly delayed the delivery date until the week of March 28th, at which point the company informed us of a major bug in the design which prevented them from being ready to delivery the drone within the semester. At this point, we had to order a different drone that still fulfilled the desired requirements of utilizing a Pixhawk controller and enough weight capacity to carry the additional components used in our project. Due to this delay, we did not receive a working drone until three weeks before the demo date.

We encountered a second complication when attempting to connect the ODroid to the Pixhawk microcontroller. The Pixhawk uses the DF-13 connector. After purchasing the recommended cable, we recognized that this cable required the use of a crimping tool so as to secure the cable. As an alternative, we ordered another DF-13 cable, which was only available from Europe and Asia. Upon receiving this second cable, we noticed that the cable contained ridges preventing it from fitting in the Pixhawk's port. This once again hindered our attempt to fully integrate our modules and test the project as a complete system.

The third complication we faced was the sparsity of documentation available for MAVProxy. As MAVProxy is a new project based on the MAVLink protocol, the documentation is incomplete and provides few examples with which to begin developing code, relying on the assumption that the programmer is already familiar with the MAVLink protocol. This limited information forced us to rely on experimentation to determine what commands to use, such as for determining the current GPS location of the drone.

The fourth complication was the lack of support for the Software in the Loop (SITL) simulation software on the ODroid due to the system architecture of the ODroid. The developers specifically mentioned in forum posts that they did not plan on support for this architecture. Due to this limitation, we could only verify that the simulation worked on a laptop and not the ODroid itself, limiting the amount of testing we could do on the integration of our threat detection system and OpenCV algorithms.

The final complication was glare on the landing pad. The presence of glare in the captured image occasionally changed the color of the black border on the pad so much that it became impossible to detect the desired contour pattern, as contour detection relies on a region which shares the same color. Without recognizing the location of the nested contour, we were unable to crop the image and therefore unable to accurately decode the image to verify its authenticity.

5.5 Further Work

In the future, there are many areas in the project we would like to explore and potentially improve. First of all, we would like to get a working DF-13 connector and thus be able to test and debug our system as a unified product and ensure it can fly as specified.

Secondly, we would like to explore the potential advantages of converting the landing pad to a base-2 color scheme and using a different corner detection algorithm, perhaps like that used to detect the corners of QR codes. Based on feedback received during the demo and presentation, we believe using a base-2 encryption model in conjunction with techniques such as autocorrelation could help improve our overall accuracy when attempting to decode the landing pad.

Thirdly, we believe implementing additional redundancy measures in our landing pad design could help increase our overall accuracy even further by providing a method for error correction, while simultaneously reducing the probability of false detections that could lead to safety concerns.

Fourthly, we believe that either reducing the effects of glare through further use of thresholding or compensation using techniques like Hough lines could help improve the accuracy of our landing pad decryption algorithms by ensuring the contours bordering the landing pad can be easily identified.

Lastly, we believe there is significant room for improvement in ensuring the robustness of our system in terms of health and safety concerns. Supplementing our current computer vision algorithms with additional methods of verifying that the landing pad truly exists to reduce the chance of false positives during decryption could prove a large step in this direction. Implementing additional algorithms to ensure that people or other obstacles are not present in the landing area as the drone begins to land could further improve the safety of our device.

6.0 References

[1]"Introducing Copter." *Copter*. Ardupilot, n.d. Web. 29 Feb. 2016. <http://copter.ardupilot.com/wiki/introduction/>.

[2]"Getting Started with Images¶." *OpenCV 3.0.0-dev Documentation*. Opencv Dev Team, 10 Nov. 2014. Web. 29 Feb. 2016. <http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html>.

[3]"How to Create and Decode a QR Code in Python Using Qrtools."*Cerebro.toBlog()*. Ralgozino, 13 June 2011. Web. 29 Feb. 2016. <https://ralgozino.wordpress.com/2011/06/13/how-to-create-and-decode-a-qr-code-in-python-using-qrtools/>.

[4]Rosebrock, Adrian. "Find Distance from Camera to Object Using Python and OpenCV." *PyImageSearch*. PyImageSearch, 19 Jan. 2015. Web. 29 Feb. 2016. <http://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/>.

[5]"OpenCV: Geometric Transformations of Images." *OpenCV 3.0.0-dev Documentation*. N.p., n.d. Web. 29 Feb. 2016. <http://docs.opencv.org/3.1.0/da/d6e/tutorial_py_geometric_transformations.html#gsc.tab=0>.

[6]"Finding Contours in Your Image¶." *OpenCV 2.4.12.0 Documentation*. N.p., n.d. Web. 29 Feb. 2016. <http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html>.

[7]"MAVProxy¶." *MAVProxy — MAVProxy 1.4.38 Documentation*. N.p., n.d. Web. 29 Feb. 2016. <http://dronecode.github.io/MAVProxy/html/index.html>.

[8]"Communicating with Odroid via MAVLink." Ardupilot, n.d. Web. 29 Feb. 2016. <http://dev.ardupilot.com/wiki/odroid-via-mavlink/>.

[9]"DroneKit Tutorial." Ardupilot, n.d. Web. 29 Feb. 2016. <http://dev.ardupilot.com/wiki/droneapi-tutorial/>.

[10]Mackay, Randy. "Ardupilot-balloon-finder." *GitHub*. N.p., n.d. Web. 29 Feb. 2016. <https://github.com/rmackay9/ardupilot-balloon-finder>.

[11]"Rg Chromaticity." *Wikipedia*. Wikimedia Foundation, n.d. Web. 15 Apr. 2016. <https://en.wikipedia.org/wiki/Rg_chromaticity>.

[12]"Adaptive Histogram Equalization." *Wikipedia*. Wikimedia Foundation, n.d. Web. 03 May 2016. <https://en.wikipedia.org/wiki/Adaptive_histogram_equalization>.

[13]"OpenCV: Histograms - 2: Histogram Equalization." *OpenCV: Histograms - 2: Histogram Equalization*. N.p., n.d. Web. 03 May 2016.

<http://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html>.

[14]"How to Sharpen an Image in OpenCV?" - *Stack Overflow*. N.p., n.d. Web. 15 Apr. 2016. <http://stackoverflow.com/questions/4993082/how-to-sharpen-an-image-in-opencv>.

[15]"Python Gcm." *Gist*. N.p., n.d. Web. 03 May 2016. <https://gist.github.com/vietkute02/9680901>.

[16] "Google Cloud Messaging." *GitHub*. N.p., n.d. Web. 03 May 2016. <https://github.com/codepath/android_guides/wiki/Google-Cloud-Messaging>.

[17] "Simple Downstream Messaging." *Google Developers*. N.p., n.d. Web. 03 May 2016. <https://developers.google.com/cloud-messaging/downstream#sending_topic_messages_from_the_server>.

[18]"Send Upstream Messages." *Google Developers*. N.p., n.d. Web. 03 May 2016. <https://developers.google.com/cloud-messaging/upstream>.

[19]"Simple Illumination Correction in Images OpenCV C++." - *Stack Overflow*. N.p., n.d. Web. 03 May 2016. <http://stackoverflow.com/questions/24341114/simple-illumination-correction-in-images-opencv-c>.

[20]"Changing the Contrast and Brightness of an Image!¶." *Changing the Contrast and Brightness of an Image! — OpenCV 2.4.13.0 Documentation*. N.p., n.d. Web. 03 May 2016. <http://docs.opencv.org/2.4/doc/tutorials/core/basic_linear_transform/basic_linear_transform.html>.

[21]"Non-blocking Read on a Subprocess.PIPE in Python." *Io*. N.p., n.d. Web. 15 Apr. 2016. <http://stackoverflow.com/questions/375427/non-blocking-read-on-a-subprocess-pipe-in-python>.

[22]"Limpkin's Blog." *Making the Electronics for a $7 USD Doppler Motion Sensor* -. N.p., n.d. Web. 04 May 2016. <http://www.limpkin.fr/index.php?post/2013/08/09/Making-the-electronics-for-a-%247-USD-doppler-motion-sensor>.

[23]"PIR Motion Sensor." *How PIRs Work*. N.p., n.d. Web. 04 May 2016. <https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/how-pirs-work>.

[24]"10.525GHz Microwave Motion Sensor Module." (n.d.): n. pag. Web. <http://www.limpkin.fr/public/HB100/HB100_Microwave_Sensor_Module_Datasheet.pdf>.

[25]Module, X-Band Microwave Motion Sensor. *MSAN-001* (n.d.): n. pag. Web. <http://www.limpkin.fr/public/HB100/HB100_Microwave_Sensor_Application_Note.pdf>.

[26]"Single-Supply Op Amp Selection Guide." *Op Amps for Everyone* (2003): n. pag. Web. <http://www.ti.com/lit/an/sloa030a/sloa030a.pdf>.

Appendix A: Requirements and Verifications Table

Table 4: R/V Table

Requirements	**Verification**	**Points**
Passive Infrared (PIR) Sensor 1. Must output high of 3.3 V (+/- .3V) when motion is detected. Must output low of 0.0 V (+/-.3 V). 2. Must be able to detect an object approaching the drone within 20 ft(+/- 5 ft)	**1. Verification process for item 1** a. Connect PIR sensor power to 5V from arduino board. Note that you must allow the necessary 1 minute setup time before trying to detect motion b. Connect ground and output signal of PIR sensor to digital multimeter c. Ensure that the output voltage when no motion is detected is between -.3V and .3V d. Wave hand over top of lens on PIR sensor and ensure that the output voltage is between 3V and 3.6V **2. Verification process for item 2** a. Connect PIR sensor power and GND to 5V and GND on arduino board b. Connect PIR sensor output in series to a 330 Ohm resistor, LED, and ground said loop c. Have one person stand 50 feet from drone and begin walking to drone d. Probed output voltage using multimeter from sensor during this period should be at 0.0(+/-.3)V e. Once person approaching drone is within 20 ft , probed output voltage using multimeter from sensor should be 3.3(+/-.3V)	**2.5**
Doppler Radar Pre-Amplifier Circuit 1. The two stage amplifier must have a gain of 12317 +/- 300. The first stage must have a gain of 101 +/- 10. The second stage must have a gain of	**1. Verification process for item 1** a. Build the two stage amplifier according to figure 12 in Sachin's lab notebook on a breadboard. Figure attached at the end of the RV table as well. b. Disconnect the connection shown in	

122 +/- 10. 2. Must have a passband from 2.82 Hz (+/- 1.82Hz) to 701 Hz +/- 100 Hz. This corresponds to outputting a high for objects moving in the sensor range at speeds between the respective lower bound and upper bound ranges of .032 to.148 mph and 19.19 to 25.5 mph	the schematic that connects the output of the first opamp to the lowpass filter that leads into the positive terminal of the second opamp c. Test the gain of the first OpAmp by connecting the output of the stage to an oscilloscope d. Using a waveform generator, input a signal with a frequency of 300 Hz (+/- 5Hz) (well within the passband) and an amplitude of 5 mV (+/-.5mV) e. Verify that the output voltage has an amplitude of .5V (+/- .1V) f. Test the gain of the second OpAmp by connecting the output of it to an oscilloscope g. Input a signal with a frequency of 300 Hz (well within the passband) and an amplitude of 10mV h. Verify that the output voltage has a voltage peak to peak of 2.4 V (+/- .4V) **2. Verification process for item 2** a. Now input a signal with a frequency of 1300 Hz and an amplitude of 5mV to the first stage of the the opamp b. Verify that the output voltage now has an amplitude of half the voltage at 300 Hz (+/-.2V) c. Now input a signal with a frequency of 1300 Hz and an amplitude of 10 mV to the 2nd stage of the the opamp d. Verify that the output voltage now has a peak to peak voltage of half the voltage at 300 hz (+/- .2 V)	**10**
Doppler Radar Sensor 1. Must see an output waveform constant output signal with a DC	**1. Verification process for item 1** a. Connect the Doppler sensor to the pre-amplifier circuit as specified by the attached schematic. b. Connect the output of the	

offset of 2.5V (+/- .5V) when there is no motion 2. Must see an output waveform with a peak to peak voltage of 4V (+/- .5V) when motion is detected 3. Must output high of 3.3 V (+/- .3V) when motion of something deemed a threat is detected. A threat is anything with a peak to peak voltage greater than 2 volts peak to peak (+/- .5 volts). Must output low of 0.0 V (+/-.3 V) during other cases.	preamplifier to an oscilloscope c. Have one person stand 40 feet from the sensor and stand completely still d. Output signal on the oscilloscope should be a constant output signal with a DC offset between 2 and 3 volts **2. Verification process for item 2** a. Subject then begins walking towards the drone and should notice an increase in a sinusoidal like output signal with a peak to peak voltage no greater than 4V. This output will increase as the subject gets closer to the sensor until it saturates. b. Once person is within 15ft (+/-5ft) sensor output should be saturated **3. Verification process for item 3** a. Now connect the output of the sensor to one of the ADC pins on the ODROID b. Run the provided testscript which will analyze the voltage level of the input signal c. If the input signal is greater than 2 volts peak to peak (+/- .5 Vs), output a high on one of the GPIO pins. Else, output a low on the same GPIO pin. d. Another person will probe this voltage and see either 3.3V (+/-.3V) or 0.0V (+/- .3V) depending on the decision made in part c	**10**
System Threat Detection 1. Sensors must both detect high within the same 1.5 seconds	*Note that it must first be shown that both sensors operate correctly on their own first for this verification to work properly **1. Verification process for item 1** a. Connect PIR sensor to 5V power and GND b. Connect output of PIR sensor to a 330 ohm resistor and LED c. Connect the doppler radar as described in the previous module d. Take the output of doppler sensor	**2.5**

	after the digital decision making and feed it through another 330 ohm resistor in series with an LED e. Position both sensors next to each other, facing in the same direction and have a person (1) start still 40 feet away f. Have another person (2) behind sensor with a stop watch g. Have person (1) begin walking toward the drone h. Have person (2) start stop watch when either LED is lit i. Ensure that both LEDs are high within 1.5 seconds	

5.1.2 OpenCV Algorithms

Requirements	Verification	Points
Total .25fps processing speed	1. Python testbench wrapper using time module to measure time elapsed between image frame capture and output from Border decryption code 2. Ensure total elapsed time less than 4 seconds per input image maximum. 3. Testbench also used for color decryption verification.	**7.5**
Distance calculation using focal length with accuracy of 85(+/-5)% at 10ft	1. Bring drone to height of 10 feet using given drone software 2. Run distance algorithm to get estimated height (given initial data point to find camera focal length) 3. Calculate the percent difference of the two heights to see if only 15%(+/-5)% different, providing enough room for the drone to slow down enough to land.	**2.5**
Color decryption of border works at 10 ft off the ground	1. Use random collection test set of images at varying heights surrounded	

with 80(+/-5)% accuracy.	by different backgrounds with different encoded strings and different angles of rotation with testbench for timing verification. 2. Determine success rate by determining average success trials: # of successes / # of trials 3. Some test images from the collection must include more than one landing pad, only one of which is correct.	**5**

5.1.3 MAVProxy Algorithms

Requirements	Verification	Points
Ensure that the control algorithm gets the drone to land at the center of the landing pad +/- 2.5 ft from the middle.	1. Modify initial takeoff software procedure so that drone flies at a lower height of 5 feet off the ground as opposed to standard operating 30 feet to facilitate easy processing of the landing pad border 2. Run the main script without landing pad code verification to determine where the drone reaches in relation to the center of the landing pad.	**5**

5.1.4 Phone App

Requirements	Verification	Points
User confirmation sent and received within 1 minute(+-1 minute) from GCM client to Odroid XMPP server and vice-versa.	1. Send packet from GCM client on app with original UTC timestamp, compare with utc time when received on Odroid C1+ XMPP server. Time difference should be negligible, timestamps must differ by no more than 0-2 minutes.	**3**

	2. Send packet from Odroid server with utc timestamp , compare with UTC timestamp on GCM client on mobile app when received. Time difference between internal clocks should be negligible.	
Works on Android Lollipop devices.	1. Use simulated phone in Android Studio to run application on. Simulated phone should be able to run as much of the app as possible without network connection as possible without error.	**2**

Appendix B: Source Code

stable_image.cpp

```
#include <opencv2/highgui/highgui.hpp>
#include "opencv2/opencv.hpp"
#include <unistd.h>

#define DEBUG true

using namespace cv;

/* copy over frame1 to frame2 and re-populate (no longer necessary) */
void capture_img(VideoCapture cam, Mat *frame1, Mat *frame2, int firstFrame) {
    *frame2 = (*frame1).clone();
    cam >> *frame1;
}

int main(int argc, char *argv[]) {
    if ((!DEBUG) && (argc < 2)) {
        std::cout << "invalid args" << std::endl;
    }

    /* ODROID should only have one camera, ignore laptop webcam */
    VideoCapture cam;
    if (DEBUG) {
        cam = VideoCapture(1);
    }
    else {
        cam = VideoCapture(0);
    }

    /* make sure camera correctly opens */
    if(!cam.isOpened()) {
        std::cout << "Unable to open camera." << std::endl;
    }

    /* camera settings  (PNG) (512 x 512 image) (50% exposure) (20% brightness) */
    cam.set(CV_CAP_PROP_FOURCC,CV_FOURCC('M','P','N','G'));
    cam.set(CV_CAP_PROP_FRAME_WIDTH, 512);
    cam.set(CV_CAP_PROP_FRAME_HEIGHT, 512);
    cam.set(CV_CAP_PROP_EXPOSURE, 50);
    cam.set(CV_CAP_PROP_BRIGHTNESS, 20);

    Mat image1;
    Mat image2;
    int firstFrame = 0;
    int frameIdx = 0;
    /* start processing frames */
    while (1) {
        capture_img(cam, &image1, &image2, firstFrame);

        /* ignore first frame (no longer needed) */
        if (firstFrame == 0) {
            firstFrame = 1;
        }
        else {
            Mat blurred;

            /* borrowed from http://stackoverflow.com/questions/4993082/how-to-sharpen-an-image-in-opencv
[14] */
            /* apply blurring to image capture to account for small movements */
            double sigma = 5, threshold = 5, amount = 1;
            GaussianBlur(image2, blurred, Size(), sigma, sigma);
            Mat lowContrastMask = abs(image2 - blurred) < threshold;
            Mat sharpened = image2*(1+amount) + blurred*(-amount);
            image2.copyTo(sharpened, lowContrastMask);

            if (argc < 2) {
                imwrite("tpp"+std::to_string(frameIdx)+".png", sharpened);
            }
            else {
                imwrite(argv[1], sharpened);
            }
            frameIdx += 1;
        }
        sleep(2);
        /* small sleep to allow for scene to change */
        //usleep(500);
    }
    return 0;
}
```

normalize_image.cpp

```
#include <algorithm>
#include <vector>

#include <opencv2/core/core.hpp>
#include "opencv2/opencv.hpp"

#include "gen_border.hpp"
```

```
#define IMG_HEIGHT 512
#define IMG_WIDTH 512
#define ROW 10
#define COL 10
#define PURPLE 4

using namespace cv;

/* averages RGB channels for normalization */
void rgbNormalize(Mat *bgr_image) {
    for (int i = 0; i < bgr_image->rows; i++) {
       for (int k = 0; k < bgr_image->cols; k++) {
           float reVal = (float)bgr_image->at<cv::Vec3b>(i,k)[2];
           float blVal = (float)bgr_image->at<cv::Vec3b>(i,k)[1];
           float grVal = (float)bgr_image->at<cv::Vec3b>(i,k)[0];
           float sum = reVal + blVal + grVal;

           bgr_image->at<cv::Vec3b>(i,k)[2] = (int)((reVal/sum)*255.0);
           bgr_image->at<cv::Vec3b>(i,k)[1] = (int)((blVal/sum)*255.0);
           bgr_image->at<cv::Vec3b>(i,k)[0] = (int)((grVal/sum)*255.0);
       }
    }
}

/* borrowed from http://stackoverflow.com/questions/24341114/simple-illumination-correction-in-images-
opencv-c [19] */
/* https://en.wikipedia.org/wiki/Adaptive_histogram_equalization#Contrast_Limited_AHE [12] */
/* convert image to lab space and evaluate CLAHE to spread colors over entire color spectrum */
void claheNormalize(Mat *bgr_image) {
    Mat lab_image;
    cvtColor(*bgr_image, lab_image, CV_BGR2Lab);

    /* split original image into 3 color planes */
    std::vector<Mat> lab_planes(3);
    split(lab_image, lab_planes);

    /* create histogram CLAHE to normalize */
    Ptr<CLAHE> clahe = createCLAHE();
    clahe->setClipLimit(4);
    Mat dst;
    clahe->apply(lab_planes[0], dst);

    /* merge planes and convert back to normal bgr */
    dst.copyTo(lab_planes[0]);
    merge(lab_planes, lab_image);
```

```
   Mat image_clahe;
   cvtColor(lab_image, *bgr_image, CV_Lab2BGR);
}

/* leverage known color to scale pixels back to original color */
void referenceNormalize(Mat *bgr_image) {
    unsigned int rul = 0, gul = 0, bul = 0;
    unsigned int rur = 0, gur = 0, bur = 0;
    unsigned int rll = 0, gll = 0, bll = 0;
    unsigned int rlr = 0, glr = 0, blr = 0;
    float avg_red = 0.0, avg_blue = 0.0, avg_green = 0.0;
    float num_pixels = ((2.0*(IMG_HEIGHT/ROW))*(2.0*(IMG_WIDTH/COL)));
    float factor_r = 0.0, factor_b = 0.0, factor_g = 0.0;

    /* upper left */
    for (int i = 0; i < (IMG_HEIGHT/ROW)*2; i++) {
       for (int k = 0; k < (IMG_WIDTH/COL)*2; k++) {
           bul += bgr_image->at<cv::Vec3b>(i,k)[0];
           gul += bgr_image->at<cv::Vec3b>(i,k)[1];
           rul += bgr_image->at<cv::Vec3b>(i,k)[2];
       }
    }

    /* upper right */
    for (int i = 0; i < (IMG_HEIGHT/ROW)*2; i++) {
        for (int k = (IMG_WIDTH-(2*(IMG_WIDTH/COL))); k < IMG_WIDTH; k++) {
            bur += bgr_image->at<cv::Vec3b>(i,k)[0];
            gur += bgr_image->at<cv::Vec3b>(i,k)[1];
            rur += bgr_image->at<cv::Vec3b>(i,k)[2];
        }
    }

    /* lower left */
    for (int i = (IMG_HEIGHT-(2*(IMG_HEIGHT/ROW))); i < IMG_HEIGHT; i++) {
        for (int k = 0; k < (IMG_WIDTH/COL)*2; k++) {
            bll += bgr_image->at<cv::Vec3b>(i,k)[0];
            gll += bgr_image->at<cv::Vec3b>(i,k)[1];
            rll += bgr_image->at<cv::Vec3b>(i,k)[2];
        }
    }

    /* lower right */
    for (int i = (IMG_HEIGHT-(2*(IMG_HEIGHT/ROW))); i < IMG_HEIGHT; i++) {
        for (int k = (IMG_WIDTH-(2*(IMG_WIDTH/COL))); k < IMG_WIDTH; k++) {
            blr += bgr_image->at<cv::Vec3b>(i,k)[0];
            glr += bgr_image->at<cv::Vec3b>(i,k)[1];
            rlr += bgr_image->at<cv::Vec3b>(i,k)[2];
        }
    }

    /* average each corner */
    rul /= num_pixels;
    gul /= num_pixels;
    bul /= num_pixels;

    rur /= num_pixels;
    gur /= num_pixels;
    bur /= num_pixels;

    rll /= num_pixels;
    gll /= num_pixels;
    bll /= num_pixels;

    rlr /= num_pixels;
    glr /= num_pixels;
    blr /= num_pixels;

    /* average over four corners */
    avg_red = (rul+rur+rll+rlr)/4.0;
    avg_blue = (bul+bur+bll+blr)/4.0;
    avg_green = (gul+gur+gll+glr)/4.0;

    /* scaling factor to apply to all pizels in image */
    factor_b = (COLOR_TABLE[PURPLE][0])/avg_blue;
    factor_g = (COLOR_TABLE[PURPLE][1])/avg_green;
    factor_r = (COLOR_TABLE[PURPLE][2])/avg_red;

    /* scale each pixel */
    for (int i = 0; i < bgr_image->rows; i++) {
        for (int k = 0; k < bgr_image->cols; k++) {
            bgr_image->at<cv::Vec3b>(k,i)[0] = std::min(255, int(bgr_image-
>at<cv::Vec3b>(k,i)[0]*factor_b));
            bgr_image->at<cv::Vec3b>(k,i)[1] = std::min(255, int(bgr_image-
>at<cv::Vec3b>(k,i)[1]*factor_g));
            bgr_image->at<cv::Vec3b>(k,i)[2] = std::min(255, int(bgr_image-
>at<cv::Vec3b>(k,i)[2]*factor_r));
        }
    }
}

/* based on
```

```
http://docs.opencv.org/2.4/doc/tutorials/core/basic_linear_transform/basic_linear_transform.html [20] */
/* brighten/darken image using saturation */
void brighten(Mat image, Mat *bgr_image, int beta=-50) {
    double alpha = 1.0;

    /* saturate each pixel to brighten/darken image */
    for(int y = 0; y < image.rows; y++ ){
        for(int x = 0; x < image.cols; x++ ) {
            /* apply to each color channel */
            for(int c = 0; c < 3; c++ ) {
                bgr_image->at<Vec3b>(y,x)[c] =
                saturate_cast<uchar>(alpha*(image.at<Vec3b>(y,x)[c] ) + beta);
            }
        }
    }
}

int main(int argc, char** argv)
{
    if (argc < 3) {
        std::cout << "invalid args" << std::endl;
    }

    else {
        Mat image;
        Mat bgr_image;

        int idx = 0;

        /* darken the image and then rgb normalize (last iter will brighten) */
        while (idx < 2) {
            image = imread( argv[1]);
            bgr_image = Mat::zeros( image.size(), image.type() );
            if (idx < 1) {
                brighten(image, &bgr_image, 0);
                rgbNormalize(&bgr_image);
            }
            else {
                /* clahe normalize and then final rgb */
                brighten(image, &bgr_image, 0);
                claheNormalize(&bgr_image);
                //rgbNormalize(&bgr_image);
            }
            imwrite(argv[2], bgr_image);
            idx += 1;
        }
        std::cout << "success" << std::endl;
    }
    return 0;
}
```

gen_border.cpp

```
#include <iostream>
#include <math.h>

#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>

#include <openssl/sha.h>

#include "gen_border.hpp"

/* defines for image handling */
#define IMG_HEIGHT 512
#define IMG_WIDTH 512
#define OFFSET_X 20
#define OFFSET_Y 20
#define ROW 10
#define COL 10

using namespace cv;

/* consts for border generation */
const short PADDING_BITS[] = {0, 1, 8, 9, 10, 11, 18, 19, 44, 45, 54, 55, 80, 81, 88, 89, 90, 91, 98, 99};
const short PAD_LEN_MID = 4;
const short PAD_LEN_END = 16;

/* encode blocks back to corresponding color */
char toHex(int c) {
    if (c < 10) {
        return '0'+c;
    }
    else {
        return 'A'+(c-10);
    }
}

/* percentage of matching characters (only equal length strings) */
float similarity(string s1, string s2) {
    int special_idx = 0;
    int total_wrong = 0;
    for (int i = 0; i < s1.length(); i++) {
        if (i == PADDING_BITS[special_idx]) {
            special_idx += 1;
        }
        else {
            if (s1[i] != s2[i]) {
                total_wrong += 1;
            }
        }
    }
    return 1-(float)total_wrong/(s1.length());
}

string generateBorder(const unsigned char *str, string output_file, bool create_img = false) {
    /* quad digits for now*/
    unsigned char hash[20];
    char quad_digs[ROW*COL+1];
    int special_idx = 0, idx = 0, hash_idx = 0, write_idx = 0, h1 = -1, h2 = -1, l1 = -1, l2 = -1;
    /* create SHA1 encoded string for security */
    SHA1(str, sizeof(str) - 1, hash);
    while (idx < ROW*COL) {
        /* ignore padding blocks */
        if (idx == PADDING_BITS[special_idx]) {
            special_idx += 1;
            quad_digs[idx] = '4';
        }
        else {
            /* take a hex digit and break into base 4 (4 primary colors) */
            if (h1 == -1) {
                int val = hash[hash_idx];
                h1 = val % 4;
                val /= 4;
                h2 = val % 4;
                val /= 4;
                l1  = val % 4;
                val /= 4;
                l2 = val % 4;
                val /= 4;
                hash_idx += 1;
            }
            if (l2 != -1) {
                quad_digs[idx] = l2+'0';
                l2 = -1;
            }
            else if (l1 != -1) {
                quad_digs[idx] = l1+'0';
                l1 = -1;
            }
            else if (h2 != -1) {
                quad_digs[idx] = h2+'0';
                h2 = -1;
            }
            else if (h1 != -1) {
                quad_digs[idx] = h1+'0';
                h1 = -1;
            }
        }
        idx += 1;
    }
    /* C-style string */
    quad_digs[100] = '\0';

    /* create the black image with white border */
    if (create_img) {
        Mat img(IMG_HEIGHT+OFFSET_Y*12, IMG_WIDTH+OFFSET_X*12, CV_8UC3, Scalar(0,0,0));
        rectangle(img, Point(0+OFFSET_Y*2, 0+OFFSET_X*2), Point(IMG_HEIGHT+OFFSET_Y*10,
IMG_WIDTH+OFFSET_X*10), Scalar(255,255,255), OFFSET_Y*2);
        rectangle(img, Point(0+OFFSET_Y*4, 0+OFFSET_X*4), Point(IMG_HEIGHT+OFFSET_Y*8,
IMG_WIDTH+OFFSET_X*8), Scalar(0,0,0), OFFSET_Y*2);
        /* color in the blocks */
        int spec_idx = 0, hex_idx = 0, block_idx = 0, digit;
        for (int k = 0; k < ROW; k++) {
            for (int i = 0; i < COL; i++) {
                digit = quad_digs[hex_idx]-'0';
                Scalar color = COLOR_TABLE[digit];
                hex_idx += 1;
                block_idx += 1;
                rectangle(img, Point(i*(IMG_HEIGHT/ROW)+OFFSET_Y*6, k*(IMG_WIDTH/COL)+OFFSET_X*6),
Point((i+1)*(IMG_HEIGHT/ROW)+OFFSET_Y*6, (k+1)*(IMG_WIDTH/COL)+OFFSET_X*6), color, -1);
            }
        }

        /* save template */
        imwrite(output_file, img);
    }
    return quad_digs;
}

void decodeBorder(string input_file, string toFind) {
    /* load the image */
    Mat img = imread(input_file, CV_LOAD_IMAGE_COLOR);
    resize(img, img, Size(IMG_WIDTH, IMG_HEIGHT));
    /* result strings */
    char default_z[101];
    for (int i = 0; i < ROW*COL; i++) {
        default_z[i] = 'Z';
    }
    default_z[100] = '\0';
    string results[] = {default_z, default_z, default_z, default_z};

    /* threshold the values */
    Mat hsv_color;
    hsv_color.create(1,1,CV_8UC(3));
    Mat hsv_img(IMG_HEIGHT, IMG_WIDTH, CV_LOAD_IMAGE_COLOR, Scalar(0,0,0));
    Mat mask(IMG_HEIGHT, IMG_WIDTH, CV_8UC3, Scalar(0,0,0));
    Mat or_res(IMG_HEIGHT+OFFSET_Y, IMG_WIDTH+OFFSET_X, CV_8UC3, Scalar(0,0,0));

    for (int bgr_idx = 0; bgr_idx < COLOR_TABLE_LEN; bgr_idx++) {
        /* convert img and color to hsv, get inRange and bitwise AND with mask*/
        cvtColor(img, hsv_img, CV_BGR2HSV);
        Scalar bgr = COLOR_TABLE[bgr_idx];
        Mat color(1, 1, CV_8UC(3), bgr);
        cvtColor(color, hsv_color, CV_BGR2HSV);
        inRange(hsv_img, Scalar(hsv_color.at<Vec3b>(0,0)[0]-10,0,0),
Scalar(hsv_color.at<Vec3b>(0,0)[0]+10,255,255), mask);
        Mat res;
        bitwise_and(img,img,res, mask=mask);

        /* decode blocks from information */
        vector<int> l_blocks[4];
        for (int i = 0; i < res.rows; i++) {
            for (int k = 0; k < res.cols; k++) {
                /* check if value > 0 */
                int b = (int)res.at<Vec3b>(i, k)[0];
                int g = (int)res.at<Vec3b>(i, k)[1];
                int r = (int)res.at<Vec3b>(i, k)[2];
                if ((b > 0) || (g > 0) || (r > 0)) {
                    int n_row = floor(i*(float)ROW/IMG_HEIGHT);
                    int n_col = floor(k*(float)COL/IMG_WIDTH);
                    int blocks[4] = {int(n_row*ROW+n_col), (ROW*COL-1)-int(n_col*COL+(ROW-1-n_row)),
(ROW*COL-1)- int(n_row*ROW+n_col), int(n_col*COL+(ROW-1-n_row))};

                    /* check if block already used for this rotation angle */
                    for (int b_idx = 0; b_idx < 4; b_idx++) {
                        bool flag = false;
                        for (int lb_idx = 0; lb_idx < l_blocks[b_idx].size(); lb_idx++) {
                            if (l_blocks[b_idx][lb_idx] == blocks[b_idx]) {
                                flag = true;
                                break;
                            }
                        }
                        if (flag == true) {
                            continue;
                        }

                        /* determine boundaries */
                        float b_x_left_boundary = n_col*((float)IMG_WIDTH/COL);
                        float b_x_right_boundary = ((n_col+1)*((float)IMG_WIDTH/COL));
                        float b_x_left_mid = b_x_left_boundary + ((float)IMG_HEIGHT/ROW*.4);
                        float b_x_right_mid = b_x_left_boundary + ((float)IMG_HEIGHT/ROW*.6);

                        float b_y_top_boundary = n_row*((float)IMG_HEIGHT/ROW);
                        float b_y_bot_boundary = ((n_row+1)*((float)IMG_HEIGHT/ROW));
                        float b_y_top_mid = b_y_top_boundary + ((float)IMG_WIDTH/COL*.4);
                        float b_y_bot_mid = b_y_top_boundary + ((float)IMG_WIDTH/COL*.6);

                        /* if in middle of boundary, check if right color or first guess */
                        if ((k >= b_x_left_mid) && (k <= b_x_right_mid) && (i >= b_y_top_mid) && (i <=
b_y_bot_mid)) {
                            l_blocks[b_idx].push_back(blocks[b_idx]);
                            bool block_set_flag = false;
                            if (results[b_idx][blocks[b_idx]] == 'Z') {
                                block_set_flag = true;
                            }
                            else if (toFind[b_idx] == toHex(bgr_idx)) {
                                block_set_flag = true;
                            }
                            if (block_set_flag) {
                                rectangle(or_res, Point(int(b_x_left_boundary), int(b_y_top_boundary)),
Point(int(b_x_right_boundary), int(b_y_bot_boundary)), COLOR_TABLE[bgr_idx], -1);
                                results[b_idx][blocks[b_idx]] = toHex(bgr_idx);
                            }
                        }
                    }
                }
            }
        }
    }

    /* determine best rotation string */
    float dist_score = 0.0;
    int idx = -1;
    for (int i = 0; i < 4; i++) {
        float new_score = similarity(results[i], toFind);
        if (new_score > dist_score) {
            dist_score = new_score;
            idx = i;
        }
    }
    std::cout << "result " << dist_score << std::endl;
}

int main(int argc, char* argv[]) {
    if (argc < 6) {
        std::cout << "invalid args" << std::endl;
    }
    else {
        string should_encode = argv[1];
        string should_decode = argv[2];
        string em_addr_str = argv[3];
        const unsigned char *email_address = new unsigned char[em_addr_str.length()+1];
        strcpy((char *)email_address,em_addr_str.c_str());
        string encode_filename = argv[4];
        string decode_filename = argv[5];
        if (should_encode == "Y") {
            generateBorder(email_address, encode_filename, true);
        }
        if (should_decode == "Y") {
            string encoded_hash = generateBorder(email_address, encode_filename, false);
            decodeBorder(decode_filename, encoded_hash);
        }
    }
    return 0;
}
```

gen_border.hpp

```
#ifndef BORDER_H
#define BORDER_H

/* perfect lighting */
const cv::Scalar COLOR_TABLE[] = {cv::Scalar(0,34,255), cv::Scalar(64,247,2), cv::Scalar(255,30,0),
cv::Scalar(3, 251, 255), cv::Scalar(255,0,174)};

/* saturated lighting */
//const cv::Scalar COLOR_TABLE[] = {cv::Scalar(104, 0, 150), cv::Scalar(105,102,47),
cv::Scalar(147,72,35), cv::Scalar(62, 95, 96), cv::Scalar(255,0,174)};

const short COLOR_TABLE_LEN = 5;
```

```
#endif
```

gen_template.cpp

```
#include <iostream>
#include <math.h>
#include <limits>
#include <sstream>

#include <opencv2/highgui/highgui.hpp>
#include "opencv2/opencv.hpp"

#include "gen_border.hpp"

#define IMG_HEIGHT 512
#define IMG_WIDTH 512
#define COLOR_TABLE_LEN 5

using namespace cv;

const float pi = (float)(22/7);


/* borrowed from http://docs.opencv.org/3.1.0/da/d6e/tutorial_py_geometric_transformations.html#gsc.tab=0
[5] */
Mat sub_img(Mat img, Point2f center, float theta,int width, int height){
    Mat out;
    Point pt(width/2, height/2);

    /* rotates image by theta(radians) */
    Mat M = getRotationMatrix2D(pt,theta*180/pi,1);
    Size s_1 (height,width);
    warpAffine(img,out,M,s_1);

    return out;
}

void white_out(Mat img,Point top_l,Point bottom_r, vector<Point> contour){
    /* white out unwanted pixels in bounded rectangle but outside of contour */
    for(int x = top_l.x; x< bottom_r.x; x++){
        for(int y = top_l.y; y<bottom_r.y; y++){
            Point test (x,y);
            double dist = pointPolygonTest(contour,test,true);
            if(dist < 1) {
                img.at<Vec3b>(y,x) = Vec3b(255,255,255);
            }
        }
    }
}

void find_matching_contour(Mat im, vector< vector<Point> >* contours, int* best_cnt_idx )
{
    Mat imgray;
    Mat thresh;
    vector<Vec4i> hierarchy;
    double parent_area, child_area;
    float ratio, min_ratio = -1;

    /* convert to grayscale, find contours */
    cvtColor(im,imgray,COLOR_BGR2GRAY);
    threshold(imgray,thresh,127,255,0);
    findContours(thresh,*contours,hierarchy,CV_RETR_TREE,CV_CHAIN_APPROX_SIMPLE);

    *(best_cnt_idx) = -1;
    for(int x = 0; x<hierarchy.size(); x++){
        if((int)hierarchy[x][3] == -1) {
            continue;
        }

        parent_area = contourArea((*contours)[hierarchy[x][3]]);
        child_area = contourArea((*contours)[x]);

        /* child insignificant */
        if(child_area<200) {
            continue;
        }

        /* desired ratio range between white outline and color border */
        ratio = (float)parent_area/child_area;
        if(ratio > .85 && ratio < 2.0) {
            //if (ratio > min_ratio) {
                *(best_cnt_idx) = x;
                min_ratio = ratio;
                break;
            //}
        }
    }

}

void find_contour_rect(vector< vector<Point> > contours, int best_cnt_idx, Point* fst, Point* snd){
    int fst_x = std::numeric_limits<int>::max();
    int fst_y = std::numeric_limits<int>::max();
    int snd_x = 0;
    int snd_y = 0;

    /* setup coordinates to crop (top left, bottom right) */
    for (int x = 0; x < contours[best_cnt_idx].size(); x++){
        if(contours[best_cnt_idx][x].x < fst_x)
            fst_x = contours[best_cnt_idx][x].x;
        if(contours[best_cnt_idx][x].x > snd_x)
            snd_x = contours[best_cnt_idx][x].x;
        if(contours[best_cnt_idx][x].y < fst_y)
            fst_y = contours[best_cnt_idx][x].y;
        if(contours[best_cnt_idx][x].y > snd_y)
            snd_y = contours[best_cnt_idx][x].y;
    }

    fst->x = fst_x;
    fst->y = fst_y;
    snd->x = snd_x;
    snd->y = snd_y;
}

void find_contour_rotated(Mat im, vector< vector<Point> >* contours, int* child, int* bst_idx){
    Mat imgray;
    Mat thresh;
    vector<Vec4i> hierarchy;
    int bst_area = -1;
    *bst_idx = -1;

    /* convert to grayscale and find contours */
    cvtColor(im,imgray,COLOR_BGR2GRAY);
    threshold(imgray,thresh,127,255,0);
    findContours(thresh,*contours,hierarchy,CV_RETR_TREE,CV_CHAIN_APPROX_SIMPLE);

    /* find largest contour area */
    for(int x = 0 ; x<contours->size(); x++){
        double area = contourArea((*contours)[x]);
        if(area > bst_area){
            bst_area = area;
            *bst_idx = x;
        }
    }

    *child = hierarchy[*bst_idx][2];
}
```

```
Mat recrop(Mat img, int lower_bound, int upper_bound) {
    Point fst(IMG_WIDTH*2, IMG_HEIGHT*2);
    Point snd(-1, -1);

    Mat hsv_color;
    hsv_color.create(1,1,CV_8UC(3));
    Mat hsv_img(IMG_HEIGHT, IMG_WIDTH, CV_LOAD_IMAGE_COLOR, Scalar(0,0,0));
    Mat mask(IMG_HEIGHT, IMG_WIDTH, CV_8UC3, Scalar(0,0,0));

    /* find purple borders */
    for (int i = COLOR_TABLE_LEN-1; i < COLOR_TABLE_LEN; i++) {
        cvtColor(img, hsv_img, CV_BGR2HSV);
        Scalar bgr = COLOR_TABLE[i];
        Mat color(1, 1, CV_8UC(3), bgr);
        cvtColor(color, hsv_color, CV_BGR2HSV);
        inRange(hsv_img, Scalar(hsv_color.at<Vec3b>(0,0)[0]-10,lower_bound,lower_bound),
Scalar(hsv_color.at<Vec3b>(0,0)[0]+10,upper_bound,upper_bound), mask);
        Mat res;
        bitwise_and(img,img,res, mask=mask);

        for (int i = 0; i < res.rows; i++) {
            for (int k = 0; k < res.cols; k++) {
                int b = (int)res.at<Vec3b>(i, k)[0];
                int g = (int)res.at<Vec3b>(i, k)[1];
                int r = (int)res.at<Vec3b>(i, k)[2];
                if ((b > 0) || (g > 0) || (r > 0)) {
                    if (k < fst.x) {
                        fst.x = k;
                    }
                    if (k > snd.x) {
                        snd.x = k;
                    }
                    if (i < fst.y) {
                        fst.y = i;
                    }
                    if (i > snd.y) {
                        snd.y = i;
                    }
                }
            }
        }
    }

    /* crop around purple borders and resize again --> img should be rotated already so should be okay */
    Mat crop_img  = img(Rect(fst.x, fst.y,snd.x-fst.x,snd.y-fst.y));
    resize(crop_img, crop_img, Size(IMG_HEIGHT, IMG_WIDTH));
    return crop_img;
}

int process(string input_filename, string output_filename, int lower_bound, int upper_bound){
    Mat img = imread(input_filename, CV_LOAD_IMAGE_COLOR);
    try {
        resize(img, img, Size(IMG_HEIGHT*1.2, IMG_WIDTH*1.2));
    }
    catch (cv::Exception) {
        std::cout << "file being modified..failed" << std::endl;
        return -1;
    }
    vector< vector<Point> > contours;
    int best_cnt_idx, mult = 0, divis = 30, best_x, best_y, best_w, best_h;
    float min_ratio = std::numeric_limits<float>::max();
    Point fst;
    Point snd;
    vector<int> bst_dims;
    Mat bst_img;

    /* find the contour we want to crop */
    find_matching_contour(img, &contours, &best_cnt_idx);
    if(best_cnt_idx == -1) {
        std::cout << "missing contour.." << std::endl;
        return -1;
    }

    /* find bounding rect coords and whiteout extra pixels */
    find_contour_rect(contours,best_cnt_idx, &fst, &snd);
    Point new_fst(std::max(0, fst.x-5),std::max(0, fst.y-5));
    Point new_snd(std::min(snd.x+5, IMG_WIDTH),std::min(snd.y+5, IMG_HEIGHT));

    white_out(img,new_fst,new_snd,contours[best_cnt_idx]);

    /* crop out the desirect rect and resize up */
    Mat crop_img = img(Rect(new_fst.x, new_fst.y,new_snd.x-new_fst.x,new_snd.y-new_fst.y));
    resize(crop_img, crop_img, Size(IMG_HEIGHT, IMG_WIDTH));

    /* try different rotations in 5 degree increments */
    Point center(IMG_HEIGHT/2,IMG_WIDTH/2);
    while(mult <= 15) {
        Mat rotated = sub_img(crop_img,center,pi/divis*mult,512,512);

        vector< vector<Point> > contours;
        int child;
        int bst_idx;
        find_contour_rotated(rotated, &contours, &child, &bst_idx);
        if (child == -1) {
            mult += 1;
            continue;
        }
        Rect curr_rect = boundingRect(contours[child]);

        double contour_area = contourArea(contours[child]);
        int x = curr_rect.x;
        int y = curr_rect.y;
        int w = curr_rect.width;
        int h = curr_rect.height;
        double bounded_area = w*h;
        double ratio = bounded_area/contour_area;

        /* ratio between bounding rect and contour --> MIN for 90 degree angles */
        if(ratio < min_ratio){
            min_ratio = ratio;
            best_x = x;
            best_y = y;
            best_w = w;
            best_h = h;
            bst_img = rotated;
        }
        mult++;
    }

    /* if able to rotate within reasonable bounds, resize and finish */
    if(min_ratio < 1.2){
        Mat im = bst_img(Rect(best_x, best_y, best_w, best_h));
        resize(im, im, Size(IMG_HEIGHT, IMG_WIDTH));
        Mat crop_final = recrop(im, lower_bound, upper_bound);
        imwrite(output_filename, crop_final);
    }

    else {
        std::cout << "rotate failed" << std::endl;
        return -1;
    }
    return 0;
}

int main(int argc, char* argv[]){
    if (argc < 5) {
        std::cout << "invalid args" << std::endl;
    }
    else {
        string input_filename = argv[1];
        string output_filename = argv[2];

        std::istringstream s3(argv[3]);
        int lower_bound, upper_bound;
        if (!(s3 >> lower_bound)) {
            std::cout << "invalid args" << std::endl;
        }
        std::istringstream s4(argv[4]);
        if (!(s4 >> upper_bound)) {
            std::cout << "invalid args" << std::endl;
        }

        if (process(input_filename, output_filename, lower_bound, upper_bound) == 0) {
            std::cout << "completed" << std::endl;
        }
    }
    return 0;
}
```

locate_border.cpp

```
#include <iostream>
#include <math.h>

#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>

#include <openssl/sha.h>

#include "gen_border.hpp"

/* defined for image handling */
#define IMG_HEIGHT 512
#define IMG_WIDTH 512

using namespace cv;

int outlineBorder(string input_file) {
    Mat img;
    try {
        img = imread(input_file, CV_LOAD_IMAGE_COLOR);
        resize(img, img, Size(IMG_WIDTH, IMG_HEIGHT));
    }
    catch (cv::Exception) {
        std::cout << "failed" << std::endl;
        return -1;
    }

    Mat hsv_color;
    hsv_color.create(1,1,CV_8UC(3));
    Mat hsv_img(IMG_HEIGHT, IMG_WIDTH, CV_LOAD_IMAGE_COLOR, Scalar(0,0,0));
    Mat mask(IMG_HEIGHT, IMG_WIDTH, CV_8UC3, Scalar(0,0,0));
    Mat or_res(IMG_HEIGHT, IMG_WIDTH, CV_8UC3, Scalar(0,0,0));

    unsigned long long num_pixels = 0;
    unsigned long long x_val = 0;
    unsigned long long y_val = 0;

    Point avg_ll = Point(-1, -1), avg_lr = Point(-1, -1), avg_ul = Point(-1, -1), avg_ur = Point(-1,
-1);
    for (int bgr_idx = 4; bgr_idx < COLOR_TABLE_LEN; bgr_idx++) {
        /* convert img and color to hsv, get inRange and bitwise AND with mask*/
        cvtColor(img, hsv_img, CV_BGR2HSV);
        Scalar bgr = COLOR_TABLE[bgr_idx];
        Mat color(1, 1, CV_8UC(3), bgr);
        cvtColor(color, hsv_color, CV_BGR2HSV);
        inRange(hsv_img, Scalar(hsv_color.at<Vec3b>(0,0)[0]-10,90,90),
Scalar(hsv_color.at<Vec3b>(0,0)[0]+10,255,255), mask);
        Mat res;
        bitwise_and(img,img,res,mask=mask);
        bitwise_or(res, or_res, or_res);

        for (int i = 0; i < res.rows; i++) {
            for (int k = 0; k < res.cols; k++) {
                int b = (int)res.at<Vec3b>(i, k)[0];
                int g = (int)res.at<Vec3b>(i, k)[1];
                int r = (int)res.at<Vec3b>(i, k)[2];
                if ((b > 0) || (g > 0) || (r > 0)) {
                    x_val += k;
                    y_val += i;
                    num_pixels += 1;
                }
            }
        }
    }
    Point center = Point(-1, -1);
    center.x = x_val/((float)num_pixels);
    center.y = y_val/((float)num_pixels);
```

```
        std::cout << center.x << " " << center.y << std::endl;

        if ((center.x >= 0) && (center.x <=156)) {
            std::cout << "LEFT" << std::endl;
        }
        else if ((center.x >= 356) && (center.x <= 512)){
            std::cout << "RIGHT" << std::endl;
        }
        else if ((center.y >= 0) && (center.y <= 156)) {
            std::cout << "UP" << std::endl;
        }
        else if ((center.y >= 356) && (center.y <= 512)) {
            std::cout << "DOWN" << std::endl;
        }
        else {
            std::cout << "CENTER" << std::endl;
        }
        return 0;
    }

    int main(int argc, char *argv[]) {
        if (argc < 2) {
            std::cout << "invalid args." << std::endl;
        }
        else {
            string input_file = argv[1];
            outlineBorder(input_file);
        }
        return 0;
    }
```

Makefile

```
CC = g++
CRYPTO = -lcrypto
FLAGS = -ggdb -std=c++11

all: template border capture locate stabilize normalize

template: gen_template.cpp
    $(CC) $(FLAGS) `pkg-config --cflags opencv` -o template gen_template.cpp `pkg-config --libs opencv` 
$(CRYPTO)

border: gen_border.cpp gen_border.hpp
    $(CC) $(FLAGS) `pkg-config --cflags opencv` -o border gen_border.cpp `pkg-config --libs opencv`
```

```
$(CRYPTO)

capture: capture_image.cpp
    $(CC) $(FLAGS) `pkg-config --cflags opencv` -o capture capture_image.cpp `pkg-config --libs opencv`
$(CRYPTO)

locate: locate_border.cpp
    $(CC) $(FLAGS) `pkg-config --cflags opencv` -o locate locate_border.cpp `pkg-config --libs opencv`
$(CRYPTO)

stabilize: stable_image.cpp
    $(CC) $(FLAGS) `pkg-config --cflags opencv` -o stabilize stable_image.cpp `pkg-config --libs opencv`
$(CRYPTO)

normalize: normalize_image.cpp
    $(CC) $(FLAGS) `pkg-config --cflags opencv` -o normalize normalize_image.cpp `pkg-config --libs
opencv` $(CRYPTO)
```

devel2.py

```
import subprocess
import math
import time
import select
import json
import re
import sys
import os
from threading  import Thread
from Queue import Queue, Empty

# CONSTANTS
HOLD_TIME = .1499999999999999994
SLEEP_TIME = .5
M_FT_CONV = 3.28084
FLYING_ALTITUDE = 30 #ft
FENCE_OUTPUT_FILE = "./fence.txt"
LAT_LON_DIV = 10000000.0
LATITUDE_CONV = 10000.0/90.0*3280.4
LONGITUDE_CONV = 364537.402
EMAIL_ADDRESS = None
THRESHOLD = .75
UPPER = 255
OPTIONS = [65, 80]

# Processes/Pipes
```

```
proc_proxy = None
proc_cv = None
proc_ci = None
proc_GPIO = None
proc_APP = None
pipes_proxy = None
pipes_cv = None
pipes_ci = None
pipes_GPIO = None
pipes_APP = None
t_proxy = None
t_ci = None
t_cv = None
t_GPIO = None
t_APP = None
q_proxy = None
q_ci = None
q_cv = None
q_GPIO = None
q_APP = None

# Other variables
DEBUG = True
is_confirmed = False
gpio_high = False
should_terminate = False

# Queueing
def enqueue_output(out, queue):
    for line in iter(out.readline, b''):
        queue.put(line)
    out.close()

def setup():
    global proc_proxy, proc_cv, proc_ci, proc_GPIO, proc_APP
    global pipes_proxy, pipes_cv, pipes_ci, pipes_GPIO, pipes_APP
    global DEBUG
    global t_proxy, t_cv, t_ci, t_GPIO, t_APP
    global  q_proxy, q_cv, q_ci, q_GPIO, q_APP

    # for now assuming SITL device
    if DEBUG:
        '''
        dronekit-sitl copter
        '''
        proc_proxy = subprocess.Popen(["exec mavproxy.py --master=tcp:127.0.0.1:5760 --map"],
stdout=subprocess.PIPE, stdin=subprocess.PIPE, shell=True)
    else:
        proc_proxy = subprocess.Popen(["exec mavproxy.py --map"], stdout=subprocess.PIPE,
stdin=subprocess.PIPE, shell=True)
    pipes_proxy = [proc_proxy.stdin, proc_proxy.stdout]
    # queueing idea borrowed from http://stackoverflow.com/questions/375427/non-blocking-read-on-a-
subprocess-pipe-in-python [21]
    q_proxy = Queue()
    t_proxy = Thread(target=enqueue_output, args=(pipes_proxy[1], q_proxy))
    t_proxy.daemon = True
    t_proxy.start()

    proc_cv = subprocess.Popen(["exec bash"], stdout=subprocess.PIPE, stdin=subprocess.PIPE, shell=True)
    pipes_cv = [proc_cv.stdin, proc_cv.stdout]
    q_cv = Queue()
    t_cv = Thread(target=enqueue_output, args=(pipes_cv[1], q_cv))
    t_cv.daemon = True
    t_cv.start()

    proc_ci = subprocess.Popen(["exec ../stabilize ./tmp_capture.png"], stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
    pipes_ci = [proc_ci.stdin, proc_ci.stdout]
    q_ci= Queue()
    t_ci = Thread(target=enqueue_output, args=(pipes_ci[1], q_ci))
    t_ci.daemon = True
    t_ci.start()

    proc_GPIO = subprocess.Popen(["exec bash"], stdin=subprocess.PIPE, stdout=subprocess.PIPE, shell=True)
    pipes_GPIO = [proc_GPIO.stdin, proc_GPIO.stdout]
    q_GPIO = Queue()
    t_GPIO = Thread(target=enqueue_output, args=(pipes_GPIO[1], q_GPIO))
    t_GPIO.daemon = True
    t_GPIO.start()

    #if DEBUG:
    #    proc_APP = subprocess.Popen(["exec bash"], stdin=subprocess.PIPE, stdout=subprocess.PIPE,
shell=True)
    #else:
    proc_APP = subprocess.Popen(["exec python ../gcm_server.py"], stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)
    pipes_APP = [proc_APP.stdin, proc_APP.stdout]
    q_APP = Queue()
    t_APP = Thread(target=enqueue_output, args=(pipes_APP[1], q_APP))
    t_APP.daemon = True
    t_APP.start()
    return True

# close pipes and kill processes
def teardown():
    global proc_proxy, proc_cv, proc_ci, proc_GPIO, proc_APP
    global pipes_proxy, pipes_cv, pipes_ci, pipes_GPIO, pipes_APP
    global t_proxy, t_cv, t_ci, t_GPIO, t_APP

    def closeup(p, pipes, t):
        for pipe in pipes:
            try:
                pipe.close()
            except:
                pass
        p.terminate()
        try:
            p.kill()
        except Error:
            pass

    closeup(proc_proxy, pipes_proxy, t_proxy)
    closeup(proc_cv, pipes_cv, t_cv)
    closeup(proc_ci, pipes_ci, t_ci)
    closeup(proc_GPIO, pipes_GPIO, t_GPIO)
    closeup(proc_APP, pipes_APP, t_APP)

    return True

# process image (remove borders, decode, or look for direction)
def scan_image(mode="template", args=None):
    global pipes_cv

    if DEBUG:
        return None
    else:
        if mode == "template":
            pipes_cv[0].write("../template ./tmp_capture{0}.png {1} {2}\n").format(args["idx"],
argx["lower"], args["upper"])
            output = collect_output("opencv", keys=["missing", "complete", "error", "invalid"],
debug=False)
            return output
        elif mode == "decode":
            pipes_cv[0].write("../border N Y {0} {1} {2}\n".format(EMAIL_ADDRESS, "./tmp_capture{0}.png",
"./tmp_solution.png").format(args["idx"]))
            output = collect_output("opencv", keys=["invalid", "result"], debug=False)
            return output
        elif mode == "locate":
            pipes_cv[0].write("../locate ./border.png")
            output = collect_output("opencv", keys=["invalid", "LEFT", "RIGHT", "UP", "DOWN", "CENTER"],
debug=False)
            return output

def collect_output(proc, attempts=30, keys=["GPS_RAW_INT"], flag="or", debug=False):
    # gather output for timeout seconds, examine
    q = None
    if ("mav" in proc) or ("proxy" in proc):
        global q_proxy
        q = q_proxy
    elif "opencv" in proc:
        global q_cv
        q = q_cv
    elif "capture" in proc:
        global q_ci
        q = q_ci
    elif "GPIO" in proc:
        global q_GPIO
        q = q_GPIO
    elif "APP" in proc:
        global q_APP
        q = q_APP

    used_attempts = 0
    while (used_attempts < attempts):
        try:
            line = q.get_nowait()
            print 'Discovered: '+line
            found = False
            if flag == "and":
                found = True
                for key in keys:
                    if not (key in line):
                        found = False
                        break
            else:
                found = False
                for key in keys:
                    if key in line:
                        found = True
                        break
            if found:
                return line
        except:
            pass
        used_attempts += 1
    return None

# query for gps status on drone, return information decoded
def get_curr_latlon():
    global pipes_proxy

    output = None
    while output is None:
        pipes_proxy[0].write("status GPS_RAW_INT\n")
        output = collect_output("proxy")

        if output is None:
            print "unable to retrieve gps corodinates..try again?"
        else:
            gps_info = output.split("GPS_RAW_INT")[1]
            gps_info_str = json.dumps(gps_info)
            gps_info_arr = (filter(None, re.split("[{, \!?:}\"\n]+", gps_info_str)))[:-1]
            gps_info_dict = {}
            for i in xrange(0, len(gps_info_arr), 2):
                gps_info_dict[gps_info_arr[i]] = gps_info_arr[i+1]

    curr_lat, curr_lon = int(gps_info_dict['lat'])/LAT_LON_DIV, int(gps_info_dict['lon'])/LAT_LON_DIV
    return (curr_lat, curr_lon)

# issue GPS location for drone to travel to
def go_to(lat, lon, fn=None, fn_args = None, ret=False, curr_lat=None, curr_lon=None):
    global pipes_proxy

    pipes_proxy[0].write("guided {0} {1} {2}\n".format(lat, lon,
int(math.ceil(FLYING_ALTITUDE/M_FT_CONV))))

    output = None
    if not (fn is None):
        if not (fn_args is None):
            output = fn(fn_args)
        else:
            output = fn()

    if ret and (not (curr_lat is None) and (not curr_lon is None)):
        time.sleep(5)
        pipes_proxy[0].write("guided {0} {1} {2}\n".format(curr_lat, curr_lon,
int(math.ceil(FLYING_ALTITUDE/M_FT_CONV))))

    return output
```

```
# set the drone in guided and arm
def arm_throttle():
    global pipes_proxy

    pipes_proxy[0].write("mode guided\n")
    pipes_proxy[0].write("arm throttle\n")

# takeoff to flying altitude (beginning/threat avoidance)
def takeoff(arm=True):
    global pipes_proxy

    # optionally arm the throttle
    if arm:
        arm_throttle()
        time.sleep(5)

    pipes_proxy[0].write("takeoff {0}\n".format(FLYING_ALTITUDE/M_FT_CONV))

    print "completed takeoff commands"
    time.sleep(30)

# set the drone into landing mode (not RTL)
def land():
    global pipes_proxy

    pipes_proxy[0].write("mode land\n")
    time.sleep(10)

    print "completed landing commands"

# look for the pad's direction
def initial_search():
    global pipes_proxy

    pipes_proxy[0].write("mode guided\n")

    # establish coordinates to evaluate
    curr_lat, curr_lon = get_curr_latlon()
    lat_u = curr_lat+(50/(LATITUDE_CONV))
    lat_d = curr_lat-(50/(LATITUDE_CONV))
    lon_r = curr_lon+(50/(LONGITUDE_CONV))
    lon_l = curr_lon-(50/(LONGITUDE_CONV))
    best_dir = None

    # chceck if the direction is useful
    def move_helper(direction):
        best_dir = None
        for i in xrange(50):
            output = scan_image(mode="locate")
            if not (output is None):
                best_dir = direction
            time.sleep(0.05)
        return best_dir

    # try all four directions in cross
    output_u = go_to(lat_u, curr_lon, move_helper, "up", True, curr_lat, curr_lon)
    time.sleep(10)
    output_d = go_to(lat_d, curr_lon, move_helper, "down", True, curr_lat, curr_lon)
    time.sleep(10)
    output_r = go_to(curr_lat, lon_r, move_helper, "right", True, curr_lat, curr_lon)
    time.sleep(10)
    output_l = go_to(curr_lat, lon_l, move_helper, "left", True, curr_lat, curr_lon)
    time.sleep(10)

    if not (output_u is None):
        best_dir = output_u
    elif not (output_d is None):
        best_dir = output_d
    elif not (output_r is None):
        best_dir = output_r
    elif not (output_l is None):
        best_dir = output_l

    print "drone initial flight path completed ", best_dir
    return best_dir

def move_towards_landing(direction):
    global pipes_proxy, pipes_cv

    # establish directional lat/lon
    curr_lat, curr_lon = get_curr_latlon()
    lat_u = curr_lat+(20/(LATITUDE_CONV))
    lat_d = curr_lat-(20/(LATITUDE_CONV))
    lon_r = curr_lon+(20/(LONGITUDE_CONV))
    lon_l = curr_lon-(20/(LONGITUDE_CONV))

    # repositino the drone
    if (direction == "LEFT"):
        output = go_to(curr_lat, lon_l)
        time.sleep(10)
    elif (direction == "RIGHT"):
        output = go_to(curr_lat, lon_r)
        time.sleep(10)
    elif (direction == "UP"):
        output = go_to(lat_u, curr_lon)
        time.sleep(10)
    elif (direction == "DOWN"):
        output = go_to(lat_d, curr_lon)
        time.sleep(10)
    elif (direction == "CENTER"):
        print "already in the center of the pad"
        return 1

    print "moved towards landing location ", direction
    return 0

# GPIG pin handler, will just check if a 1 is output indicating threat detected (Thread)
def check_GPIO():
    global pipes_GPIO, gpio_high

    while (1):
        value = collect_output("GPIO", keys=["1"])
        if not (value is None):
            gpio_high = True
            should_terminate = True
        if should_terminate:
            break

# App handler, tell app to send request, break upon confirmation (Thread)
def check_APP():
    global pipes_APP, is_confirmed, should_terminate

    pipes_APP[0].write("send\n")
    while (1):
        output = collect_output("APP", keys=["confirm"])
        if not (output is None):
            print 'done'
            is_confirmed = True
            should_terminate = True
        if should_terminate:
            break

if __name__ == "__main__":
    with open("../email.txt") as email_file:
        for line in email_file:
            EMAIL_ADDRESS = line
            break
```

```
setup()
time.sleep(20)
takeoff()
best_dir = initial_search()
if best_dir is None:
    pass
else:
    output = move_to_landing(best_dir)
    while (output != 1):
        best_dir = scan_image(mode="locate")
        output = move_to_landing(best_dir)
    # try decoding
    is_landing = False
    output = scan_image(mode="template", args={"idx": 0, "lower": OPTIONS[0], "upper": UPPER})
    if ("complete" in output):
        result =  scan_image(mode="decode", args={"idx": 0}).split("result")[1]
        if int(result) > THRESHOLD:
            is_landing = True
            land()
    if not is_landing:
        output = scan_image(mode="template", args={"idx": 1, "lower": OPTIONS[1], "upper": UPPER})
        if ("complete" in output):
            result =  scan_image(mode="decode", args={"idx": 1}).split("result")[1]
            if int(result) > THRESHOLD:
                is_landing = True
                land()
    if not is_landing:
        pass
        # TODO abort msg
land()

# start the GPIO/APP threads
#thread_app = Thread(target=check_GPIO, args=[])
#thread_app.daemon = True
#thread_msg = Thread(target=check_APP, args=[])
#thread_msg.daemon = True
check_APP()
gpio_high = False
is_confirmed = False

print 'reaching stopping point'

last_checked = time.time()
while(1):
    # keep throttle armed in case we need to take off quickly
```

```
        if (abs(time.time() - last_checked)) > 4.0:
            arm_throttle()
        if (gpio_high):
            takeoff(arm=False)
        elif (is_confirmed):
            land()
            # TODO shine_gpio led?
    print 'tearing down now'
    teardown()
```

verification_testbench.py

```
import subprocess
import os
import sys
import time
from threading  import Thread
from Queue import Queue, Empty

UPPER = 255
OPTIONS = [50, 65, 80]

proc_cv = None
pipes_cv = None
t_cv = None
q_cv = None

# Queueing
def enqueue_output(out, queue):
    for line in iter(out.readline, b''):
        queue.put(line)
    out.close()

# create the opencv shell to issue commands to
def setup():
    global proc_cv, pipes_cv, t_cv, q_cv

    proc_cv = subprocess.Popen(['bash'], stdout=subprocess.PIPE, stdin=subprocess.PIPE)
    pipes_cv = [proc_cv.stdin, proc_cv.stdout]

    q_cv = Queue()
    t_cv = Thread(target=enqueue_output, args=(pipes_cv[1], q_cv))
    t_cv.daemon = True
    t_cv.start()
    return True
```

```
# close opencv pipes and exit the shell
def teardown():
    global proc_cv, pipes_cv, t_cv, q_cv

    for pipe in pipes_cv:
        try:
            pipe.close()
        except:
            pass

    proc_cv.kill()
    return True

# look through the output (non-blocking using queue)
def collect_output(attempts=30, keys=[], flag="or", debug=False):
    global q_cv

    q = q_cv
    used_attempts = 0
    while (used_attempts < attempts):
        try:
            line = q.get_nowait()
            if debug:
                print 'Discovered: '+line
            found = False
            if flag == "and":
                found = True
                for key in keys:
                    if not (key in line):
                        found = False
                        break
            else:
                found = False
                for key in keys:
                    if key in line:
                        found = True
                        break
            if found:
                return line
        except Exception as e:
            pass
    return None

if __name__ == "__main__":
    setup()
    testfile = "./tests.txt"
    # if testfile specified, override default
    if (len(sys.argv) > 1):
        testfile = sys.argv[1]

    tmp_dest = "./tmp_out"

    with open(testfile, 'r') as input_file:
        successes = 0
        failures = 0
        threshold = -1.0
        firstLine = True
        test_idx = 0
        for line in input_file:
            # read in threshold for success values
            if firstLine:
                threshold = float(line)
                firstLine = False
            else:
                # run test case
                print "Running Test {0}".format(test_idx)
                (input_image, email_addr, should_normalize) = line.replace("\n", "").split("|")
                input_image = input_image.lstrip().rstrip()
                email_addr = email_addr.lstrip().rstrip()
                should_normalize = int(should_normalize)
                print "Beginning Phase 0"
                # start clock
                t0 = time.time()
                failed = [True for x in xrange(len(OPTIONS))]
                highest_acc = -1

                # try removing borders and rotating (multiple options for threshold for outliers)
                for opt_idx in xrange(len(OPTIONS)):
                    pipes_cv[0].write("./template {0} {1} {2} {3}\n".format(input_image,
tmp_dest+str(opt_idx)+".png", OPTIONS[opt_idx], UPPER))
                    phase0_out = collect_output(attempts=100, keys=["missing", "invalid", "completed",
"failed"], flag="or", debug=False)
                    if (phase0_out is None) or ("missing" in phase0_out) or ("invalid" in phase0_out) or
("failed" in phase0_out):
                        continue
                    else:
                        failed[opt_idx] = False
                        if should_normalize:
                            pipes_cv[0].write("./normalize {0} {1}\n".format(tmp_dest+str(opt_idx)+".png",
tmp_dest+str(opt_idx)+".png"))
                # stop clock
                print "Phase 0 Time: {0}".format(time.time()-t0)
                # start secondary clock
                t1 = time.time()
                for opt_idx in xrange(len(OPTIONS)):
                    if (failed[opt_idx] is False):
                        # try decoding
                        pipes_cv[0].write("./border N Y {0} {1} {2}\n".format(email_addr, input_image,
tmp_dest+str(opt_idx)+".png"))
                        phase1_out = collect_output(["invalid", "result"], "or")
                        # accuracy of removing borders with this threshold
                        if "invalid" in phase1_out:
                            continue
                        result = phase1_out.split("result")[1]
                        os.remove(tmp_dest+str(opt_idx)+".png")
                        observed = float(result)

                        if (observed > highest_acc):
                            highest_acc = observed
                print highest_acc
                # stop clock
                print "Phase 1 Time: {0}".format(time.time()-t1)
                print "Total Time: {0}".format(time.time()-t0)

                # if able to decode with high enough accruacy this is success
                if (highest_acc > threshold):
                    successes += 1
                else:
                    failures += 1
                test_idx += 1
        print "Correct: {0}, Failures: {1}, Accuracy: {2}".format(successes, failures,
(float(successes)/(successes + failures)))

    teardown()
```

live_verification_locate.py

```
import subprocess
import sys
from threading  import Thread
from Queue import Queue, Empty
from time import sleep
import cv2

proc_cv = None
```

```
proc_ci = None
pipes_cv = None
pipes_ci = None
t_cv = None
q_cv = None

# Queueing
def enqueue_output(out, queue):
    for line in iter(out.readline, b''):
        queue.put(line)
    out.close()

# create the opencv shell to issue commands to
def setup():
    global proc_cv, proc_ci
    global pipes_cv, proc_ci
    global t_cv, q_cv

    proc_cv = subprocess.Popen(['bash'], stdout=subprocess.PIPE, stdin=subprocess.PIPE)
    pipes_cv = [proc_cv.stdin, proc_cv.stdout]

    proc_ci = subprocess.Popen(["exec ./stabilize ./test_capture.png"], stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)

    q_cv = Queue()
    t_cv = Thread(target=enqueue_output, args=(pipes_cv[1], q_cv))
    t_cv.daemon = True
    t_cv.start()
    return True

# close opencv pipes and exit the shell
def teardown():
    global proc_cv, proc_camera
    global pipes_cv, pipes_camera
    global t_cv, q_cv

    for pipe in pipes_cv:
        try:
            pipe.close()
        except:
            pass
    for pipe in pipes_camera:
        try:
            pipe.close()
        except:
            pass
    proc_cv.kill()
    proc_camera.kill()
    return True

# look through the output (non-blocking using queue)
def collect_output(attempts=30, keys=[], flag="or", debug=False):
    global q_cv

    q = q_cv
    used_attempts = 0
    while (used_attempts < attempts):
        try:
            line = q.get_nowait()
            if debug:
                print 'Discovered: '+line
            found = False
            if flag == "and":
                found = True
                for key in keys:
                    if not (key in line):
                        found = False
                        break
            else:
                found = False
                for key in keys:
                    if key in line:
                        found = True
                        break
            if found:
                return line
        except Exception as e:
            pass
    return None

if __name__ == "__main__":
    setup()
    input_image = "./test_capture.png"
    test_idx = 0

    sleep(2)
    while(1):
        # run test case
        print "Running Test {0}".format(test_idx)
        print "Beginning Phase 0"
```

```
            # try removing borders and rotating (multiple options for threshold for outliers)
            pipes_cv[0].write("./locate {0}\n".format(input_image))
            inp = cv2.imread(input_image)
            phase0_out = collect_output(attempts=100, keys=["invalid", "LEFT", "RIGHT", "UP", "DOWN", 
    "CENTER", "failed"], flag="or", debug=False)
            if (phase0_out is None) or ("invalid" in phase0_out) or ("failed" in phase0_out):
                continue
            else:
                print phase0_out
                try:
                    cv2.imshow("Captured Image", inp)
                    cv2.waitKey(0)
                    cv2.destroyAllWindows()
                except:
                    print "skipping"

            sleep(2)

            test_idx += 1
        teardown()
```

live_verification_decode.py

```
import subprocess
import os
import sys
from threading  import Thread
from Queue import Queue, Empty
from time import sleep
import numpy as np
import cv2
import time

UPPER = 255
OPTIONS = [50, 65] #80]

proc_cv = None
proc_ci = None
pipes_cv = None
pipes_ci = None
t_cv = None
q_cv = None
```

```
# Queueing
def enqueue_output(out, queue):
    for line in iter(out.readline, b''):
        queue.put(line)
    out.close()

# create the opencv shell to issue commands to
def setup():
    global proc_cv, proc_ci
    global pipes_cv, proc_ci
    global t_cv, q_cv

    proc_cv = subprocess.Popen(['bash'], stdout=subprocess.PIPE, stdin=subprocess.PIPE)
    pipes_cv = [proc_cv.stdin, proc_cv.stdout]

    proc_ci = subprocess.Popen(["exec ./stabilize ./test_capture.png"], stdin=subprocess.PIPE,
stdout=subprocess.PIPE, shell=True)

    q_cv = Queue()
    t_cv = Thread(target=enqueue_output, args=(pipes_cv[1], q_cv))
    t_cv.daemon = True
    t_cv.start()
    return True

# close opencv pipes and exit the shell
def teardown():
    global proc_cv, proc_camera
    global pipes_cv, pipes_camera
    global t_cv, q_cv

    for pipe in pipes_cv:
        try:
            pipe.close()
        except:
            pass
    for pipe in pipes_camera:
        try:
            pipe.close()
        except:
            pass

    proc_cv.kill()
    proc_camera.kill()
    return True

# look through the output (non-blocking using queue)
```

```
def collect_output(attempts=30, keys=[], flag="or", debug=False):
    global q_cv

    q = q_cv
    used_attempts = 0
    while (used_attempts < attempts):
        try:
            line = q.get_nowait()
            if debug:
                print 'Discovered: '+line
            found = False
            if flag == "and":
                found = True
                for key in keys:
                    if not (key in line):
                        found = False
                        break
            else:
                found = False
                for key in keys:
                    if key in line:
                        found = True
                        break
            if found:
                return line
        except Exception as e:
            pass
    return None

if __name__ == "__main__":
    setup()
    threshold = .60
    email_addr = "hoaglan2@illinois.edu"
    tmp_dest1 = "./tmp_out1-"
    tmp_dest2 = "./tmp_out2-"
    input_image = "./test_capture.png"
    successes = 0
    failures = 0
    test_idx = 0

    if len(sys.argv) < 3:
        print "using default success threshold of .60 and email of hoaglan2@illinois.edu"

    else:
        threshold = sys.argv[1]
        email_addr = sys.argv[2]
    sleep(2)
    while(1):
        # run test case
        print "Running Test {0}".format(test_idx)
        print "Beginning Phase 0"
        # start clock
        start_time = time.time()
        failed = [True for x in xrange(len(OPTIONS))]
        highest_acc = -1

        # try removing borders and rotating (multiple options for threshold for outliers)
        for opt_idx in xrange(len(OPTIONS)):
            pipes_cv[0].write("./template {0} {1} {2} {3}\n".format(input_image,
tmp_dest1+str(opt_idx)+".png", OPTIONS[opt_idx], UPPER))
            phase0_out = collect_output(attempts=100, keys=["missing", "invalid", "completed", "failed"],
flag="or", debug=False)
            if (phase0_out is None) or ("missing" in phase0_out) or ("invalid" in phase0_out) or ("failed"
in phase0_out):
                continue
            else:
                failed[opt_idx] = False
                # perform normalization
                pipes_cv[0].write("./normalize {0} {1}\n".format(tmp_dest1+str(opt_idx)+".png",
tmp_dest2+str(opt_idx)+".png"))
                phase1_out = collect_output(attempts=100, keys=["invalid", "success"], flag="or",
debug=False)
        try:
            original = cv2.imread(input_image)
        except:
            continue
        end_time = time.time()
        print "Time to complete {0}: {1} seconds".format(test_idx, (end_time - start_time))
        cropped = []
        normalized = []
        for opt_idx in xrange(len(OPTIONS)):
            if (failed[opt_idx] is False):
                try:
                    n_cropped = cv2.imread(tmp_dest1+str(opt_idx)+".png")
                    n_normalized = cv2.imread(tmp_dest2+str(opt_idx)+".png")
                    n_cropped = cv2.resize(n_cropped, (200, 200))
                    n_normalized = cv2.resize(n_normalized, (200, 200))
                    if n_normalized is None:
                        print tmp_dest2+str(opt_idx)+".png"
                    cropped.append(n_cropped)
                    normalized.append(n_normalized)
                except:
                    pass
        if len(cropped) > 1:
            cropped_display = np.hstack(tuple(cropped))
            print 'klo'
            normalized_display = np.hstack(tuple(normalized))
            cv2.imshow("Original Image", original)
            cv2.imshow("Cropped Images", cropped_display)
            cv2.imshow("Normalized Images", normalized_display)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
        else:
            sleep(2)

        for opt_idx in xrange(len(OPTIONS)):
            if (failed[opt_idx] is False):
                # try decoding
                pipes_cv[0].write("./border N Y {0} {1} {2}\n".format(email_addr, input_image,
tmp_dest2+str(opt_idx)+".png"))
                phase1_out = collect_output(["invalid", "result"], "or")
                # accuracy of removing borders with this threshold
                if "invalid" in phase1_out:
                    continue
                os.remove(tmp_dest1+str(opt_idx)+".png")
                os.remove(tmp_dest2+str(opt_idx)+".png")
                result = phase1_out.split("result")[1]
                observed = float(result)

                if (observed > highest_acc):
                    highest_acc = observed
        print highest_acc

        # if able to decode with high enough accruacy this is success
        if (highest_acc > threshold):
            successes += 1
        else:
            failures += 1
        test_idx += 1
    print "Correct: {0}, Failures: {1}, Accuracy: {2}".format(successes, failures,
(float(successes)/(successes + failures)))

    teardown()
```

Gcm_server.py

```
#Source code taken and modified from [15] <https://gist.github.com/vietkute02/9680901>
#!/usr/bin/python
import sys, json, xmpp, random, string
from subprocess import PIPE, Popen
from threading  import Thread
from Queue import Queue, Empty
import datetime

SERVER = 'gcm.googleapis.com'
PORT = 5235
USERNAME = "632877030162"
PASSWORD = "AIzaSyBnar8mPqv6CmZ8gqOQBtUOOyq28nw2cwY"
REGISTRATION_ID = "d3Z--G_Ilq0:APA91bE5qwn034sII-Sm5IkZdsUWfNfBVQVjbcNKDDGcicjI9G7SCWmeuri7o_D99NlymrbLN-
JlbyKcSrUfR2iW_-1jShrIILnX_2e0QBsrgBPHb0__pALCc85oHhX82U7iuUv0kDKi"
#REGISTRATION_ID =
"fkDwv4iOT74:APA91bFSC8ANDGgnimDl6DpWXV4og8cXl54Oj3xluTPmgs8dOI43iU7sCwEMmOljNqoILsHmD3VsZ3WGd-iuTa0H-
ab0oWZOJTbqztysUzURwvpCdM02ww7BdcWqsS4fHhPAPG_W9leB"
ON_POSIX = 'posix' in sys.builtin_module_names

unacked_messages_quota = 100
send_queue = []

#Determine current UTC time
def UtcNow():
    now = datetime.datetime.utcnow()
    return (now - datetime.datetime(1970, 1, 1)).total_seconds()

# Return a random alphanumerical id
def random_id():
  rid = ''
  for x in range(8): rid += random.choice(string.ascii_letters + string.digits)
  return rid

#Determine communication time if received message and print "confirm" message
def message_callback(session, message):
  server_time = UtcNow()
  gcm = message.getTags('gcm')
  if gcm:
    gcm_json = gcm[0].getData()
    msg = json.loads(gcm_json)
    if("message_type" not in msg):
      app_time = int(msg["data"]["time_test"])
      print("Commnication took : " + str((server_time-app_time)) + " seconds")
      print("confirm")
      print('\n')
```

```
#Package data into json to send to app
def send(json_dict):
  template = ("<message><gcm xmlns='google:mobile:data'>{1}</gcm></message>")
  client.send(xmpp.protocol.Message(
      node=template.format(client.Bind.bound[0], json.dumps(json_dict))))

#Process messages in queue
def flush_queued_messages():
  global unacked_messages_quota
  while len(send_queue) and unacked_messages_quota > 0:
    send(send_queue.pop(0))
    unacked_messages_quota -= 1

#client = xmpp.Client('gcm.googleapis.com', debug=['socket'])

#Set up xmpp server
client = xmpp.Client('gcm.googleapis.com', debug=[])
client.connect(server=(SERVER,PORT), secure=1, use_srv=False)
auth = client.auth(USERNAME, PASSWORD)
if not auth:
  print 'Authentication failed!'
  sys.exit(1)

#Map server to message call back function
client.RegisterHandler('message', message_callback)

#Enqueue sends to queue
def enqueue_output(out, queue):
      for line in iter(out.readline, b''):
          queue.put(line)
      out.close()

q = Queue()
t = Thread(target=enqueue_output, args=(sys.stdin, q))
t.daemon = True # thread dies with the program
t.start()


#Contiously runs server, sending messages when "send" command is inputed by user
while True:
  client.Process(1)
  flush_queued_messages()

  # read line without blocking
  try:  line = q.get_nowait() # or q.get(timeout=.1)
  except Empty:
      continue

  if("send" in line):
    send_queue.append({'to': REGISTRATION_ID,
                   'message_id': 'reg_id',
                   'data': {'message_destination': 'RegId',
                            'message_id': random_id(),
                            'time_test': str(UtcNow())}})
```

gpio.c

```
#include <stdio.h>


#include <stdlib.h>


#include <stdint.h>


#include <pthread.h>


#include <unistd.h>


#include <string.h>


#include <time.h>


#include <wiringPi.h>


#include <wiringPiI2C.h>


#include <wiringSerial.h>


#include <lcd.h>


#include <iostream>


#include <thread>
```

```
void check(const int doppler, const int pir1, const int pir2);

bool analogCheck(const int doppler);

// Pin number declarations for ODROID-C1+

const int pirLeftFront = 2;

const int pirLeftBack = 3;

const int pirRightFront = 4;

const int pirRightBack = 5;

const int outputVar = 27;



#define PORT_ADC1 1

#define PORT_ADC0 0



bool outputVal = false;



int main(void)

{

        wiringPiSetup();
```

```
pinMode(pirLeftFront, INPUT);

pinMode(pirLeftBack, INPUT);

pinMode(pirRightFront, INPUT);

pinMode(pirRightBack, INPUT);

pinMode(outputVar, OUTPUT);



digitalWrite(outputVar, LOW);

digitalWrite(pirLeftFront, LOW);

digitalWrite(pirLeftBack, LOW);

digitalWrite(pirRightFront, LOW);

digitalWrite(pirRightBack, LOW);




bool pLF, pLB, pRF, pRB, dL, dR;

while(1)

{

        pLF = digitalRead(pirLeftFront);

        pLB = digitalRead(pirLeftBack);
```

```
                pRF = digitalRead(pirRightFront);

                pRB = digitalRead(pirRightBack);

                dL = analogCheck(PORT_ADC1);

                dR = analogCheck(PORT_ADC0);

                std::cout << dR << std::endl;

                std::cout << pLF << " " << pLB << " " << pRF << " " << pRB << " " << dL << " " << dR <<
'\n';

        //      if(pLF == true || pLB == true  || dL == true)

        //      {

        //              std::cout << "potential threat, left" << '\n';

        //              check(PORT_ADC1, pirLeftFront, pirLeftBack);

        //      }

                if(pRF == true || pRB == true  || dR == true)

                {

                        std::cout << "potential threat, right" << '\n';

                        check(PORT_ADC0, pirRightFront,pirRightBack);

                }

                if(outputVal == true)
                {
                        std::cout << "entering infinite loop" << std::endl;
                        while(1){
                                std::cout << "in the loop" << std::endl;
                                digitalWrite(outputVar, HIGH);
                                delay(500);
                                digitalWrite(outputVar, LOW);
                                delay(500);
                                digitalWrite(outputVar, HIGH);
                                delay(500);
                                digitalWrite(outputVar, LOW);
                                delay(500);
                                digitalWrite(outputVar, HIGH);
                                digitalWrite(outputVar, LOW);
                        }//break;
                }
        }
}
```

```
void check(const int doppler, const int pir1, const int pir2)

{

        bool loopDecide;

        if(analogCheck(doppler) == true)

                loopDecide = false;

        else

                loopDecide = true;



        time_t start, end;

        time(&start);

        end = start;

        if(loopDecide == false)

        {

                while(difftime(end, start) < 2)

                {

                        if(digitalRead(pir1) == HIGH || digitalRead(pir2) == HIGH)

                        {
```

```
                        outputVal = true;

                        break;

                }

                time(&end);

        }

}

else

{

        while(difftime(end,start) < 2)

        {

                if(analogRead(doppler) == true)

                {

                        outputVal = true;

                        break;

                }

                time(&end);

        }

}
```

```
}



bool analogCheck(const int doppler)

{

        int adcVal;

        adcVal = analogRead(doppler);

        //printf("%d\n", adcVal);

        std::cout << adcVal*1.8/1024.0 << '\n';

        if(adcVal*1.8/1024.0 > 1.6){return true;}

        return false;

}
```