The Design and Implementation of an Autonomous Surveillance and Security Drone

Nikki Fayra
nikki@fayra.com
https://nikkifayra.com

University of Illinois at Urbana-Champaign
Department of Computer Science - Senior Thesis [*]

May 28, 2016 – Version 1.3

Abstract

Autonomous surveillance drones, also known as unmanned aerial vehicles, would improve security for individuals, businesses, and government agencies. Drones present an affordable, effective method to automatically surveil a large open area to identify unauthorized personnel at close range. This paper fully describes the design and implementation of an autonomous unmanned aerial vehicle which can patrol an area and identify humans. Potential future improvements of my aircraft and some alternative uses of unmanned aerial surveillance, other than surveilling an area of private property, are also discussed.

[*]A senior thesis submitted to the Computer Science Department of University of Illinois at Urbana-Champaign in partial fulfillment of the requirements for the degree of Bachelor of Science.

I would like to thank my thesis advisor, Ryan Matthew Cunningham, for his guidance and support of this project.

Contents

1. Introduction

The main goal of this project was to design and implement an autonomous aerial drone which could autonomously patrol an outdoor environment and identify unauthorized personnel through image processing. The target audience would be individuals, businesses, and government entities.
Individuals could have a personal drone to conduct surveillance of their private property. They could also have a personal drone follow them overhead to surveil their surroundings if not on their property, such as a public park.
Businesses and government entities could have a fleet of drones patrolling an area of their property, such as vacant space surrounding research labs, production facilities, and military installations.
In this paper, I fully describe the design and implementation of an autonomous drone which can patrol an area and identify humans.

2. Drone Design and Construction

I initially wanted to design and implement an autonomous surveillance and security drone built solely with the Raspberry Pi[1] as the controller. After realizing that controlling the motors and doing image processing would be very difficult without using a real time operating system[2], I decided to additionally use the ArduPilot Mega (APM)[3] flight controller which would be responsible for controlling the motors. I also used additional components such as a camera and a speaker. The full parts list can be seen in Appendix A.

2.1. Raspberry Pi

I chose to use the Raspberry Pi instead of another similar small computer since it has the largest community surrounding it. I initially started with the Raspberry Pi 2 Model B, but later also developed the drone with the Raspberry Pi 3 Model B. The Raspberry Pi is responsible for the high level drone control code which interfaces with the ArduPilot Mega, and it is also responsible for capturing images from the camera.

2.2. ArduPilot Mega

The ArduPilot Mega contains autopilot firmware and an Inertial Measurement Unit containing a combination of sensors such as an accelerometer and a barometric pressure sensor[1] [4]. An additional external compass/GPS unit is also used.

2.3. Drone Registration

In the United States, unmanned aircraft weighing more than 250 grams must be registered with the Federal Aviation Administration.[5] If a drone weighs more than 25 kilograms, then it needs to be registered as a traditional aircraft. If it weighs more than 250 grams but less than 25 kilograms it can be registered online as a Small Unmanned Aircraft. There is a $5 fee, and the registration identifier must be placed on the drone. I received the registration identifier, `FA3APMN3FT`, which I secured on the front of the camera and on the Raspberry Pi case.

[1]The barometric pressure sensor follows air pressure, so the altitude is not exact.

2.4. Drone Assembly

I began with assembling the DJI Flamewheel F450 ARF Kit one arm at a time to the lower power distribution board. I used the F450 User Manual v2.2[7] and the DJI F330 FlameWheel QuadCopter Assembly guide[8] as references when assembling the drone. While assembling each arm I also fastened a leg, electronic speed controller, and a motor as shown in figure 1, 2, 3, and 4. The electronic speed controllers (ESCs) were soldered to the power distribution board. The motors were connected to the ESCs via the motors' bullet connectors and the servo connections were temporarily attached to the landing gear.

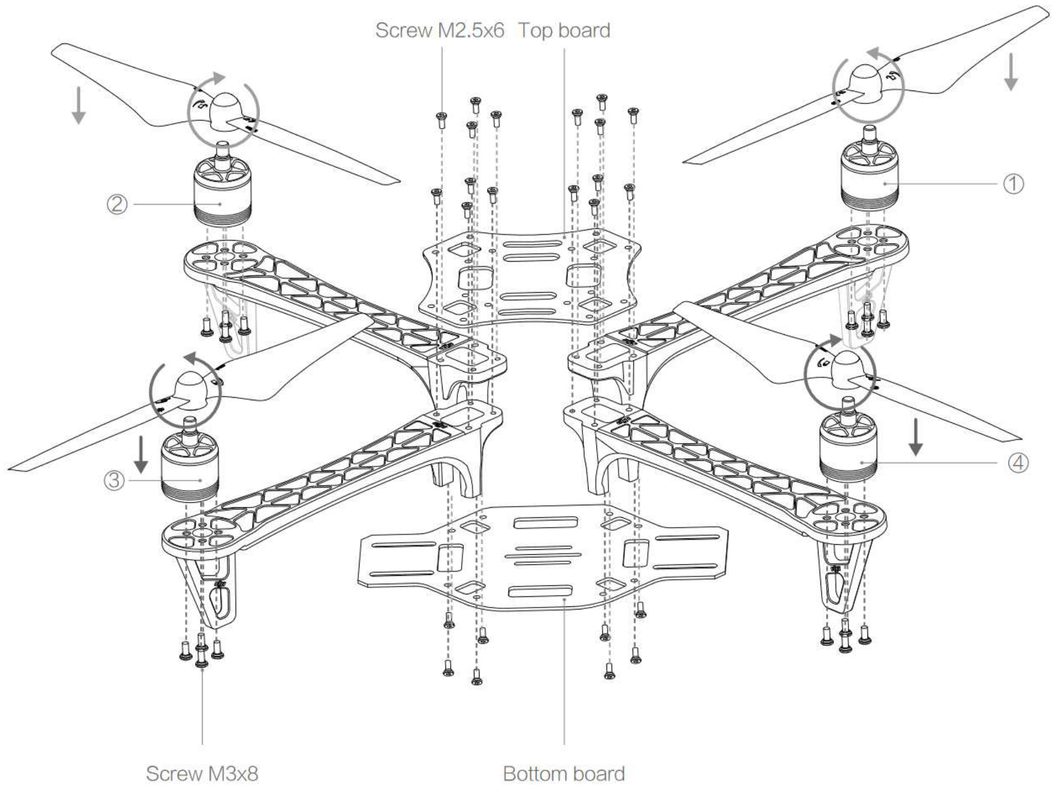


Figure 1: Flamewheel F450 assembly diagram[7]

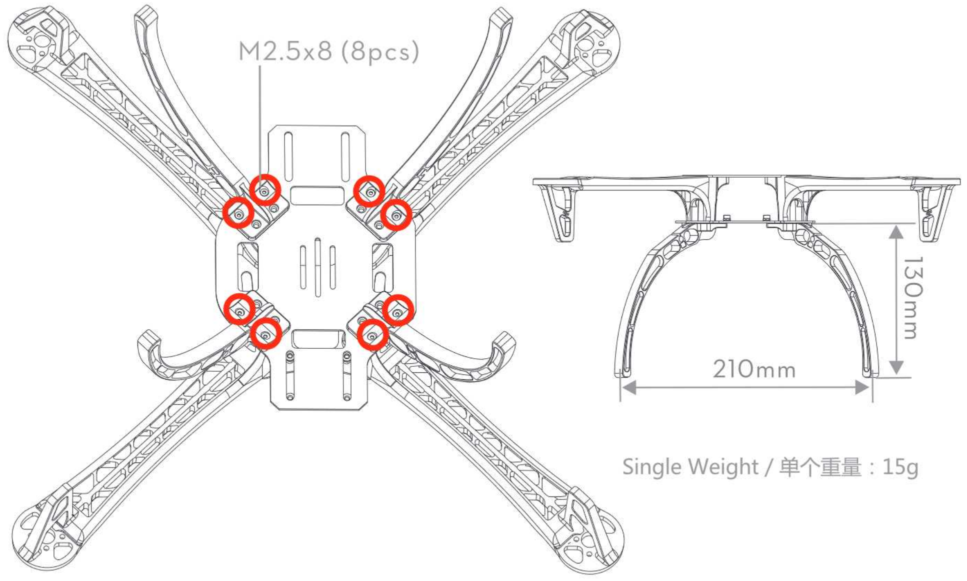


Figure 2: DJI Flamewheel F450 landing gear assembly diagram[9]

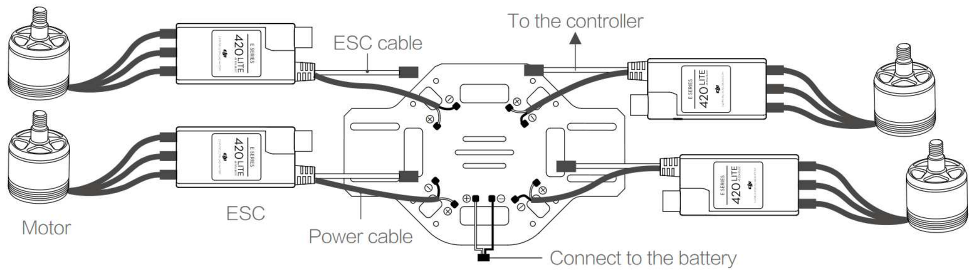


Figure 3: DJI Flamewheel F450 electronic speed controller and motor assembly diagram[7]

After most of the frame was assembled, I determined placement for the ArduMega Pilot (APM), GPS/compass module, Raspberry Pi, and battery. The APM was placed on the top of the lower board. I installed the GPS/compass module onto the bottom of the upper board using the supplied adhesive pad. The Raspberry Pi was fastened to the top of the upper board using Velcro strips and zip-ties.

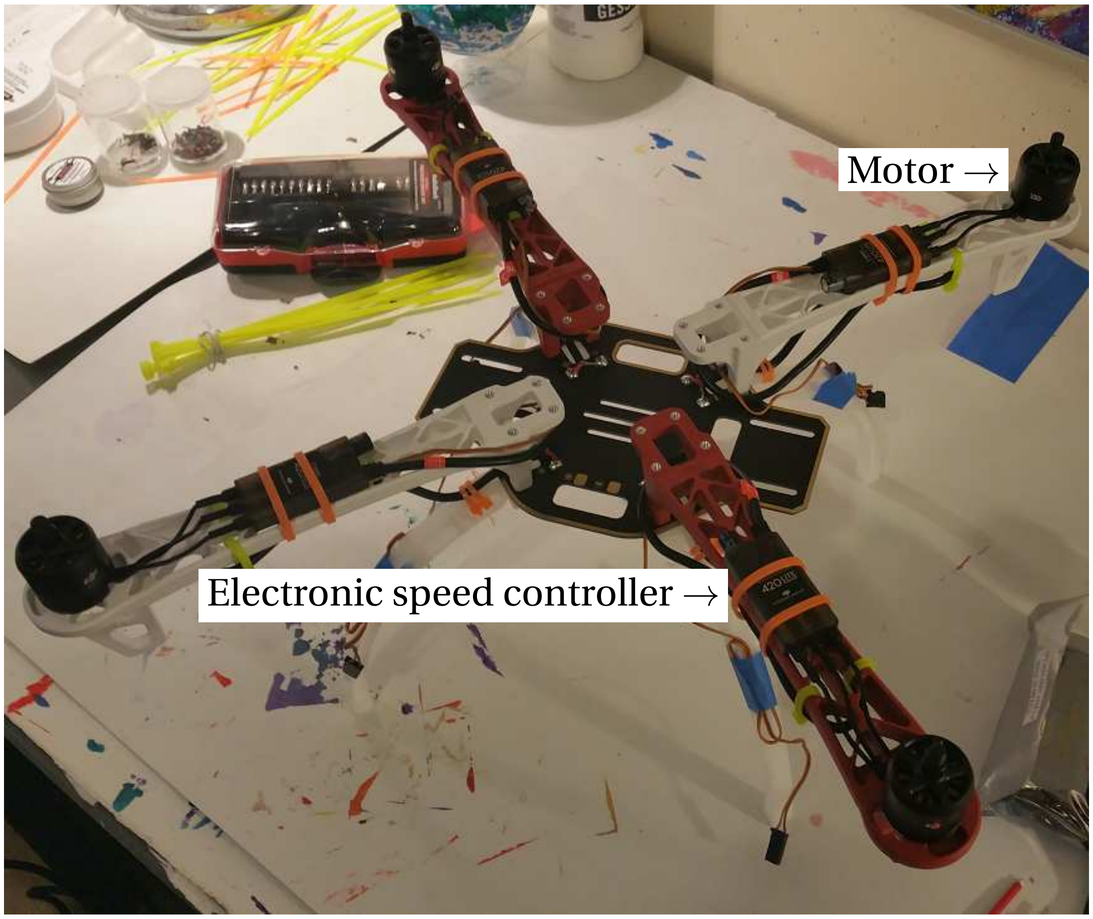


Figure 4: Partial assembly of arms, electronic speed controllers, motors, and landing gear

I connected the three male to male servo connectors between the radio receiver and the APM's input channels 1-6 and 8. Channel 8 supplies the receiver with power. The servo connectors from the ESCs were connected to the APM's output channels 1-4 as shown in figure 5. The battery input and UBEC step down converter power wires were soldered to the lower power distribution board. The APM power module was fastened to one of the legs.

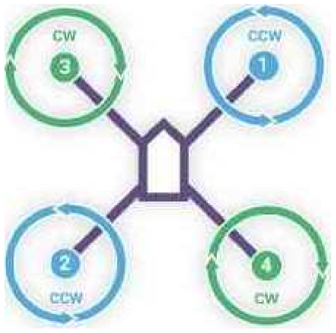


Figure 5: Motor channel layout for Quad X configuration[13]

The battery was fastened below the lower board using a supplied Velcro harness and zip-ties. The female XT60 connector was soldered to the input power wires. The input power wires were connected to the APM power module pass-through, and the battery was connected to the APM power module. See figure 6, 7, and 8.

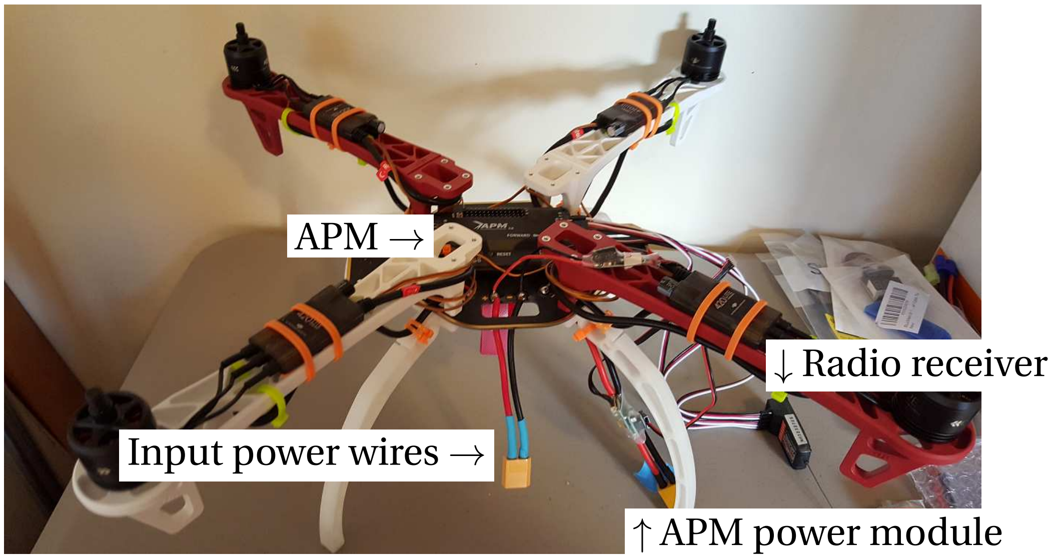


Figure 6: Partial assembly including the APM, radio receiver, power wires, UBEC step down converter, and APM power module

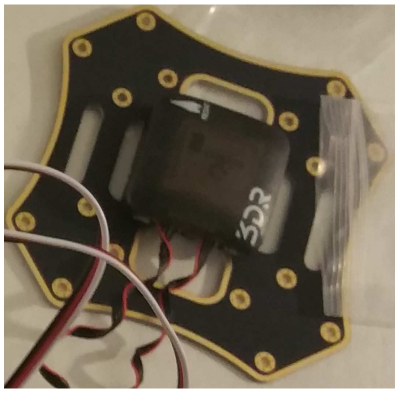


Figure 7: GPS/compass module placement

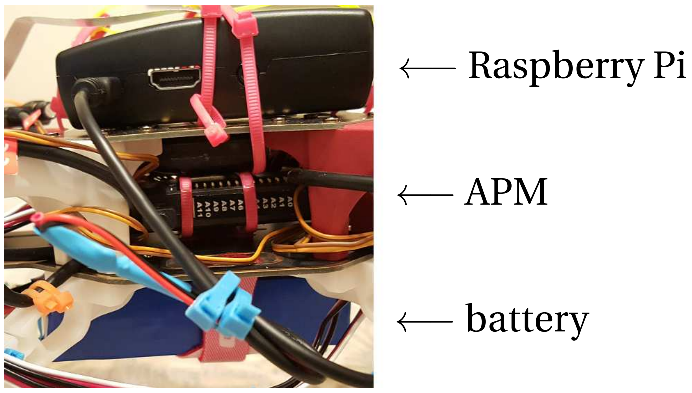


Figure 8: Raspberry Pi, APM, and battery placement

The power wires from the UBEC for the Raspberry Pi were connected to a female to male servo wire. The male end of the servo wire was cut, and soldered to the power and ground connections of the micro-USB end of a micro-USB cable. This allowed the Raspberry Pi to be powered via its micro-USB port instead of the power GPIO pin. Powering it via the GPIO pins does not protect against incorrect voltage or current spikes, which could permanently damage the board[10]. Current spikes and incorrect voltage can occur on the drone due to the motors requiring more power or the battery being low, so I decided that powering via USB was the safest option.
The micro-USB cable connecting the Raspberry Pi to the ArduMega Pilot (APM) for communication was stripped, and the power wire was cut. This was to prevent the APM from receiving power from the Raspberry Pi, since the APM is powered from the power module. Supplying power over USB and the power module at the same time could damage the components[11]. See figure 9.

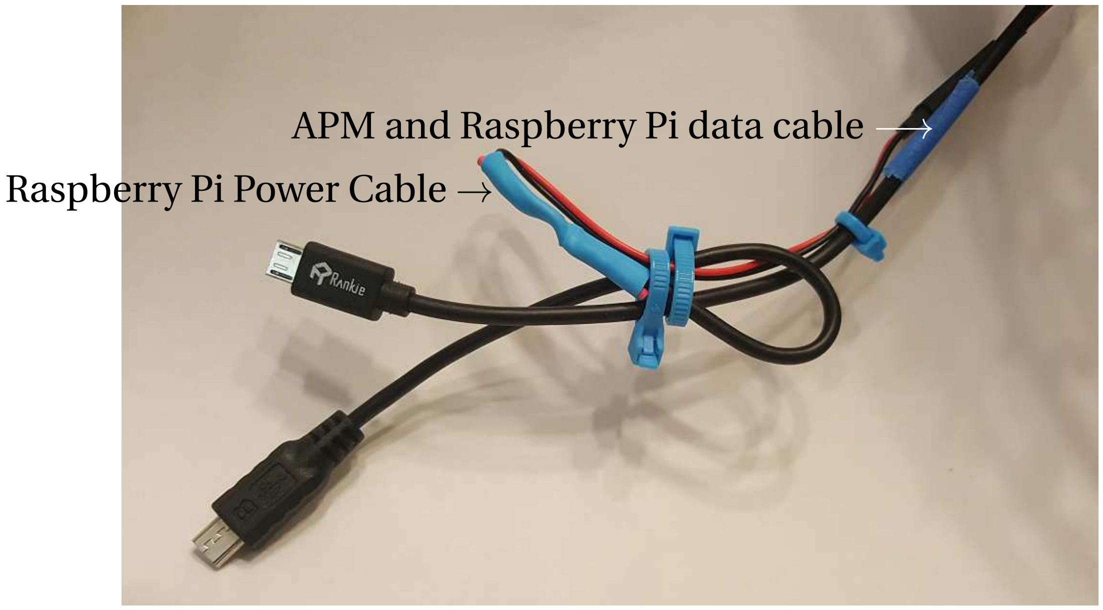

APM and Raspberry Pi data cable $\longrightarrow$
Raspberry Pi Power Cable $\rightarrow$

Figure 9: USB cable wiring

The speaker was mounted to the back of the lower board using Velcro and a zip-tie. The camera was mounted to the front of the lower board, and the USB WiFi dongle was plugged in to the Raspberry Pi. See figure 10, and 11. The remaining connections were made and propellers were fastened as shown in figure 12.

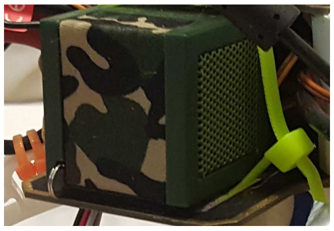


Figure 10: Speaker

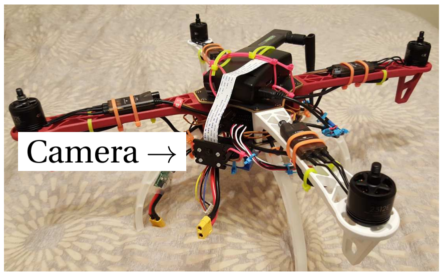


Figure 11: Completed initial drone build without propellers

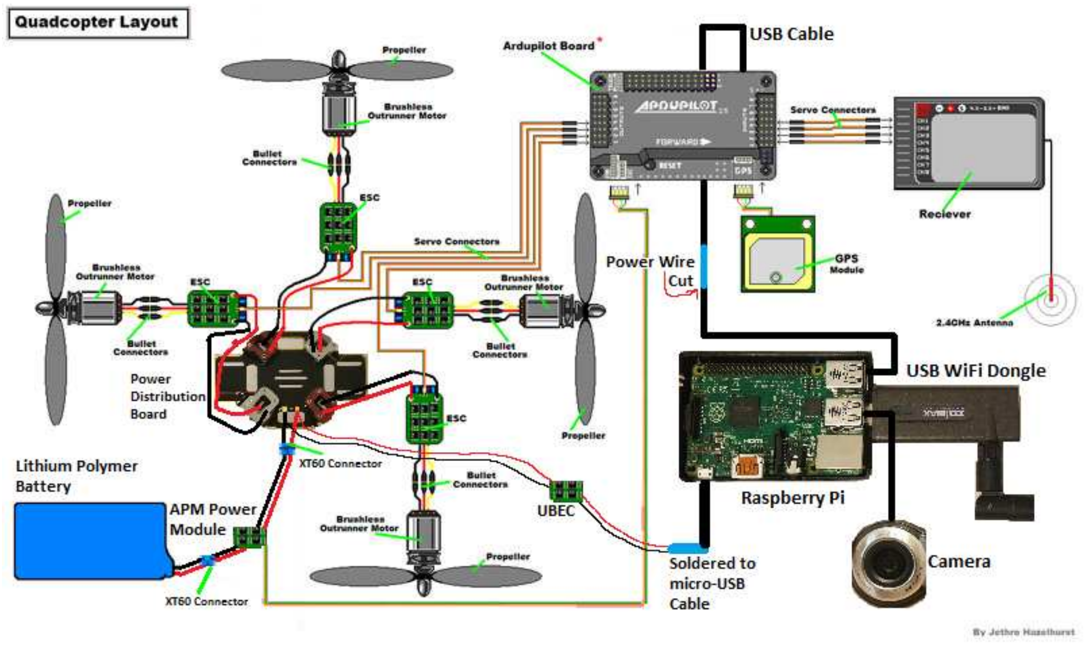


Figure 12: Full Drone Layout, modified image by Jethro Hazelhurst[12]

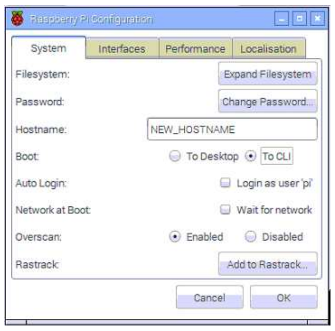
3. Drone Configuration

3.1. Raspberry Pi Configuration

Raspbian GNU/Linux 8 (Jessie)[15], a Debian derivative, was installed as the Raspberry Pi's operating system. Specifically, I installed the 2015-09-28 build[16] on the Raspberry Pi 2 Model B, and the 2016-02-26 build[17] on the Raspberry Pi 3 Model B. I have kept the systems up-to-date, so the most recent version of the operating system should work without problems.

3.1.1. Operating System Installation and Configuration

Win32 Disk Imager[18] was used to write the Raspbian image to the micro-SD card. Similar graphical and command line utilities are available for GNU/Linux and Mac OS X operating systems.[19] Upon first booting the Raspberry Pi, I connected it to my local network via Ethernet and logged in via SSH. I changed the password for the default user, "pi", from "raspberry" to a stronger password with the `passwd` utility, expanded the



Filesystem:
Password:
Hostname: NEW_HOSTNAME
Boot: To Desktop [✓ To CLI]
Auto Login: ☐ Login as user 'pi'
Network at Boot: ☐ Wait for network
Overscan: [✓ Enabled] Disabled
Rastrack:

Figure 13: Raspberry Pi Configuration

filesystem to the entire micro-SD card with `raspi-config`[28], and updated packages with `sudo apt-get update` and `sudo apt-get upgrade`.

I installed `tightvncserver` with `apt-get` to access the X desktop remotely via VNC when needed. I launched a `vncserver` instance, connected with the TightVNC desktop client[90], and went into the Raspberry Pi Configuration[2] to boot to CLI instead of the X desktop, change hostname, disable

[2]raspi-config can also be used for these configuration options.

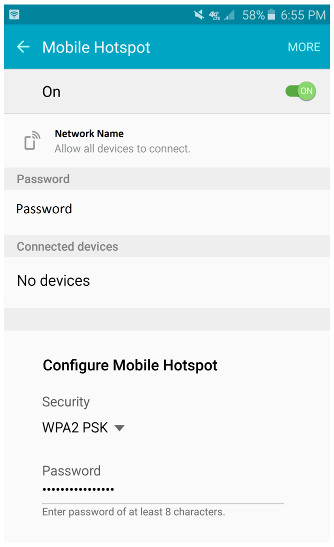
auto-logins, and enable the camera module. [3]

3.1.2. Network Connectivity and Configuration

In order for the drone to have local network connectivity and Internet connectivity, I used my Samsung Galaxy Note 5's mobile WiFi hotspot which uses WPA2-PSK with CCMP (AES-based) encryption.[20][21]

This communication method is secure as long as:

- The pre-shared key is kept secret.[22]
- The pre-shared key has sufficient entropy.



58% 6:55 PM

Mobile Hotspot MORE

On ON

Network Name
Allow all devices to connect.

Password
Password

Connected devices
No devices

Configure Mobile Hotspot

Security
WPA2 PSK

Password
••••••••••••••••
Enter password of at least 8 characters.

Figure 14: Samsung Galaxy Note 5 Mobile Hotspot

It is important to note that the pre-shared key needs to have sufficient entropy, otherwise the encrypted key could be obtained by an attacker from snooping on network traffic and it could be cracked offline.[23][24] I used *How Secure Is My Password?*[25] to measure the approximate strength of a password equivalent to the one I used and it reports, ”It would take a computer about 131 Billion Years to crack your password.”

There are other security concerns relating to WPA2 such as attacks on Wi-Fi Protected Setup, Microsoft's MS-CHAPv2 protocol, and a vulnerability abusing the shared Group Temporal Key which assumes the attacker is already authenticated.[20] These concerns do not affect my implementation and use.

WPA2 Enterprise solves the security issues discussed relating to authentica-

[3]Enabling the camera module is only needed for the Raspberry Pi camera module, not a USB webcam.

tion since it no longer depends on a pre-shared key.[22][27][26] It is easy to infer how my implementation could be extended to use a WPA2 Enterprise network with multiple access points. This would allow greater connectivity range, better security, and would be a good choice for connecting surveillance drones to a WiFi network on business and government property.
In order to connect to the WiFi access point with the Raspberry Pi I edited `/etc/wpa_supplicant/wpa_supplicant.conf` to the following configuration to connect to my WPA2-PSK network[42][43]:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
network={
        ssid="SSID_Goes_Here"
        psk="Pre-shared_key_goes_here"
        proto=WPA2
        key_mgmt=WPA-PSK
        pairwise=CCMP
}
```

These lines in `/etc/network/interfaces` were also commented out, so the system would only use the wlan0 interface.

```
#allow-hotplug wlan1
#iface wlan1 inet manual
#    wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
```

3.1.3. WiFi Connection Analysis

For the Raspberry Pi 3 Model B, I disabled the onboard WiFi card as the performance was not good in flight due the lack of an external antenna. Instead, the USB WiFi dongle was used.

It is possible to solder an external antenna to the board, but it would violate Federal Communications Commission regulations[45]. The board contains pads where U.FL connector for an external antenna could be soldered. See figure 15.

To disable the driver for the onboard WiFi card, and also the onboard Bluetooth card, `/etc/modprobe.d/raspi-blacklist.conf` was edited to the following:[44]

```
blacklist brcmfmac
blacklist brcmutil
blacklist btbcm
blacklist hci_uart
```

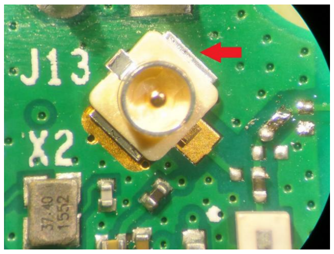


Figure 15: U.FL connector overlaid on the provided pads[45]

I initially tested connection speed indoors by setting up a NGINX HTTP server[89] on the Raspberry Pi. I downloaded a 100 megabyte (100MB) file from a distance of 0.5 meters, and 10 meters with walls as obstacles and measured the following approximate average speeds: [4]

Configuration	**Speed (0.5m)**	**Speed (10m)**
Raspberry Pi 2 Model B With USB WiFi Dongle	1.8 MB/s	700 KB/s
Raspberry Pi 3 Model B With onboard WiFi	5 MB/s	1.2 MB/s
Raspberry Pi 3 Model B With USB WiFi Dongle	530 KB/s	400 KB/s

As shown, the Raspberry Pi 2 Model B performs significantly better than the Raspberry Pi 3 Model B using the same USB WiFi dongle[5], most likely due to additional power draw requirements on the Raspberry Pi 3 Model B, so it is unable to supply enough power to the USB WiFi dongle.[46]

[4] I also tested connecting the Raspberry Pi systems to a home router instead of the phone hotspot. The speeds were the same, showing that the phone is not a bottleneck.

[5] Both Raspberry Pi systems use the same model SD card, thus read speed is not an issue.

The connection using the USB dongle during flight is slower than the (10m) results above, likely due to interference from motors and radio transmitter, additional power draw, and greater distance. A connection can be maintained at approximately 45 meters with direct line of sight.

3.2. APM Planner and Mission Planner

The ArdruPilot Mega (APM) uses the MAVLink (Micro Air Vehicle Link) [31] [32] protocol over serial to communicate with a companion computer or ground station. In order to configure and arm the ArdruPilot Mega for flight, software supporting the MAVLink protocol and APM configuration is needed. APM Planner[29] and Mission Planner[30] are two available programs which communicate with the APM over MAVLink and support configuration and installing firmware. I installed Mission Planner on a Windows computer, and compiled APM Planner on the Raspberry Pi. The steps needed to compile APM Planner along with installing its dependencies are listed below.
[29]

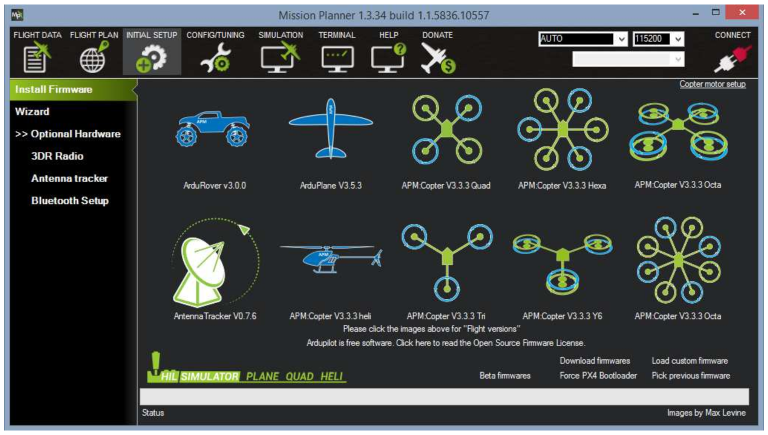
```
git clone https://github.com/ArduPilot/apm_planner
cd apm_planner
sudo apt-get install qt5-qmake qt5-default qtscript5-dev  \
    libqt5webkit5-dev libqt5serialport5-dev libqt5svg5-dev  \
    qml-module-qtquick2 libsdl1.2-dev libsndfile-dev \
    flite1-dev libssl-dev libudev-dev libsdl2-dev
qmake qgroundcontrol.pro
make
```

Running APM Planner:

```
./release/apm_planner
```

3.3. ArduPilot Mega Configuration

The first step was to install firmware via Mission Planner. The APM was connected to Mission Planner over a USB cable at a baud rate of 115200. APMCopter firmware v3.2.1 is the latest supported firmware for the APM, since later versions are too large to fit onto the board.[33] In Mission Planner, under `Initial Setup` → `Install Firmware`, the APMCopter Quad firmware was selected. This will automatically install the latest supported firmware for the device.[34] See figure 16.



Please click the images above for "Flight versions"

Ardupilot is free software. Click here to read the Open Source Firmware License.

Figure 16: Mission Planner firmware installation screen

Next, the hardware was configured.[35] Under `Initial Setup` → `Mandatory Hardware` → `Frame Type`, The 'X' frame type was selected. Under `Initial Setup` → `Mandatory Hardware` → `Radio Calibration`, the radio was calibrated by moving each stick and trim tab on the radio transmitter through its full range. This menu is similar to the APM Planner menu which can be seen in figure 19.

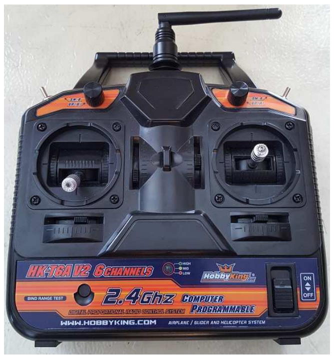
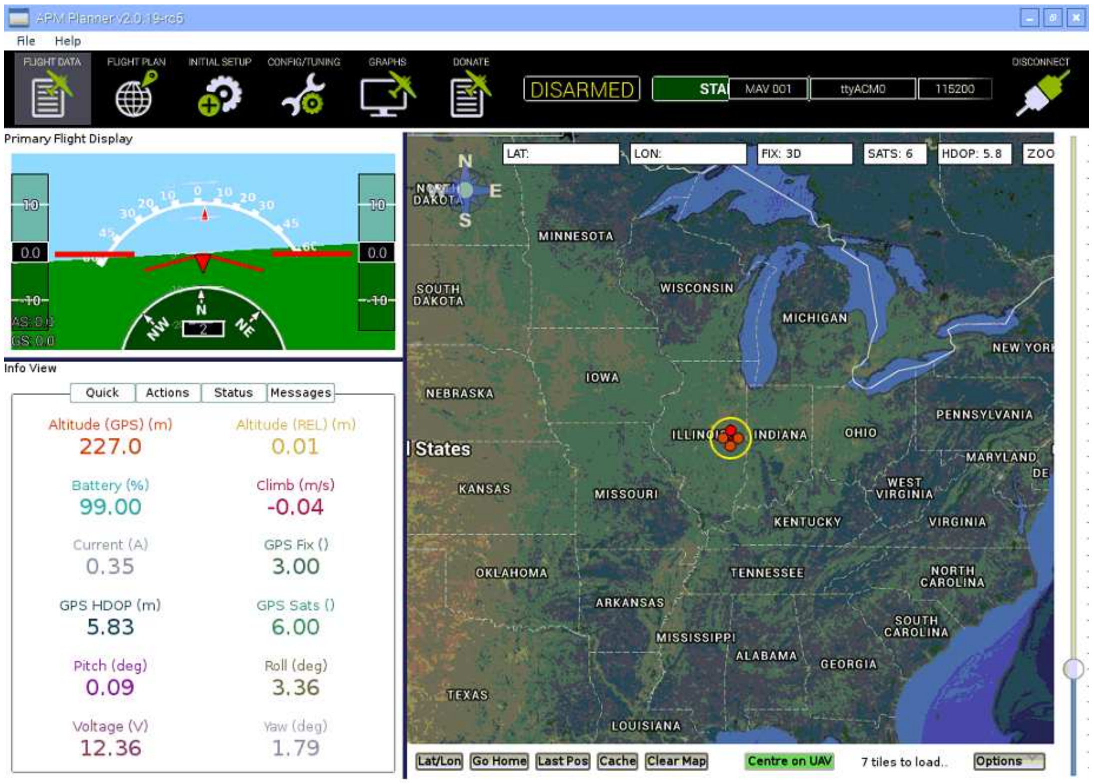


Figure 17: Hobby King 2.4Ghz 6Ch Transmitter

After the radio was calibrated, the drone was disconnected from Mission Planner and the electronic speed controllers (ESCs) were calibrated using the ”All at once” method.[36] The steps to this method are the following:

1. Turn on the radio transmitter with maximum throttle.
2. Connect the battery to the drone.
3. Ensure the APM’s LED lights are forming a cyclical pattern.
4. Disconnect and reconnect the battery.
5. The APM is now in ESC calibration mode. The red and blue LEDs should be blinking alternatively.
6. Wait for the ESCs to emit a musical tone, in this case it is three beeps for the 3 cell battery. Two additional beeps indicate that the maximum throttle has been captured.
7. Set the transmitter’s throttle to the minimum position.
8. The ESCs should now emit a long tone indicating that calibration is complete. The battery can now be disconnected.

Next, the compass and accelerometer were calibrated using APM Planner on the Raspberry Pi. This was done outdoors so that indoor electronics would not interfere. I used an Android SSH app[72] to connect to the Raspberry Pi, then launched a `vncserver` instance and accessed it via an Android VNC client app.[73] APM Planner was launched and connected to the APM via USB on the device /dev/ttyACM0 at a baud rate of 115200. See figure 18.
The compass calibration is accessed under `Initial Setup` → `Mandatory Setup` → `Compass`. This involved rotating the drone on all three axes for a set time frame, so that each side of the vehicle (front, back, left, right, top and bottom) pointed towards the ground.[37] The magnetic declination (angle between magnetic north and true north)[38], is determined automatically from GPS.[39]
The accelerometer calibration is accessed under `Initial Setup` → `Mandatory Setup` → `Accel Calibration`. This involved placing each of the vehicle’s sides level on the ground.[40]



MAV 001 ttyACM0 115200

LAT: LON: FIX: 3D SATS: 6 HDOP: 5.8

Altitude (GPS) (m) 227.0	Altitude (REL) (m) 0.01
Battery (%) 99.00	Climb (m/s) -0.04
Current (A) 0.35	GPS Fix () 3.00
GPS HDOP (m) 5.83	GPS Sats () 6.00
Pitch (deg) 0.09	Roll (deg) 3.36
Voltage (V) 12.36	Yaw (deg) 1.79

7 tiles to load..

Figure 18: APM Planner

Finally, compass-motor calibration was performed to account for magnetic interference from the motors. I used Mission Planner indoors for this step since it was not working with APM Planner. This is accessed under `Initial Setup` → `Optional Hardware` → `Compass/Motor Calibration`. The calibration involves running the motors from no throttle to full throttle, then quickly bringing the throttle back to zero. A graph displaying the percentage of interference by throttle level is shown. According to the ArduPilot documentation, the recommended interference is below 30% and 31% - 60% is considered a grey area. My interference was in this grey area range.[41]

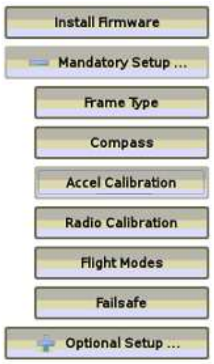


Figure 19: APM Planner Initial Setup Menu

4. Initial Flight Testing

My initial flight tests consisted of putting APM in stabilize mode and arming it with APM Planner via VNC. Stabilize mode allows the vehicle to be flown manually with the radio transmitter, but self-levels the roll and pitch axes.[14] My secondary tests involved using the GPS-assisted loiter mode, which holds a GPS position but allows for manual control of altitude and position via the radio transmitter.[61]

4.1. Battery Charging

I used 2200mAh (milliamperehours) three cell 40C 11.1v lithium polymer batteries to power the drone. The C rating, in this case 40C, indicates the continuous current draw in amps that the cell supports.[51] The C rating is multiplied by the cell capacity in mAh to obtain the continuous current in mA (milliamps).

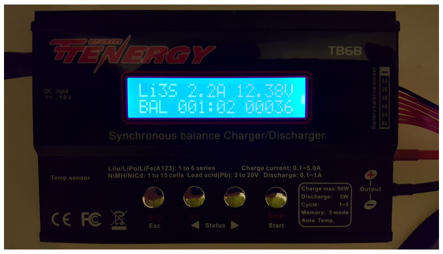


Figure 20: Tenergy TB6-B Balance Charger shown charging a battery.

The batteries were charged in balance mode using a Tenergy TB6-B Balance Charger at 1C, or 2.2amps, to an end voltage of 12.6v, or 4.2v per cell.[49][52] The batteries were charged in a fireproof bag in case of an accidental fire. See figure 20.

If the battery is below 3v per cell before charging, special care needs to be taken when recharging the battery. Below this voltage, the battery could be permanently damaged due to its internal resistance to charging increasing.[50] The battery needs to be slowly charged at a rate of $\frac{C}{20}$ through $\frac{C}{10}$ until it is above 3v per cell.

4.2. Incorrect Motor Configuration

Initially the drone would immediately flip over upon takeoff. I was not sure if this was due to incorrect configuration as described in subsection 3.2. After re-calibration and troubleshooting, I realized that one of the motors was rotating in the incorrect direction. When initially building the drone, what I decided on as the ”front” orientation was different from the finished product due to the mounting placement for the camera, and the motors were in incorrect positions according to their labels. See figures 5 and 21.

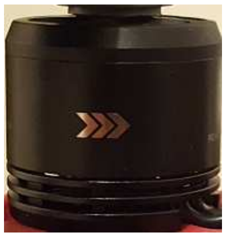


Figure 21: Motor with direction label shown

After swapping the motors' locations by 90 degrees, the problem persisted. The labels on the motors do not matter since they can rotate in either direction. To fix the problem, two of the bullet connectors between the motor and the Electronic Speed Controller needed to be swapped.[13][14]

4.3. Camera GPS Interference

After the motor configuration was fixed, the drone flew correctly in stabilize mode. The drone also flew without issue in the GPS-assisted loiter mode.
I wrote the following simple bash script to capture an image every two seconds using the `raspistill` Raspberry Pi camera module utility.[88] The script also places the images into a directory for the NGINX web server, and updates an HTML file with the new file.

```
#!/bin/bash

while sleep 2
do
DATE=$(date +"%Y-%m-%d_%H%M%S")
```

```
raspistill --nopreview --timeout 1 -vf -hf -o /var/www/html/camera/$DATE.jpg
list_dir=‘ls -t /var/www/html/camera/‘
echo "<html><body>" > /var/www/html/camera/index.html
for i in $list_dir
do
echo "<a href=\"$i\">$i</a></br>" >> /var/www/html/camera/index.html
done

echo "</body></html>" >> /var/www/html/camera/index.html
done
```

When attempting to fly the drone in the GPS-assisted loiter mode with the camera script active, the drone was not stable when the camera was in use and crashed. After some research to determine if this was a power problem or interference problem, I determined that when the camera was in use it was interfering with the GPS.
I attempted to fix this by creating a Faraday cage around the camera, while monitoring the GPS fix within APM Planner. Wrapping aluminum-foil around the camera's ribbon cable and completely around the camera and its board solved

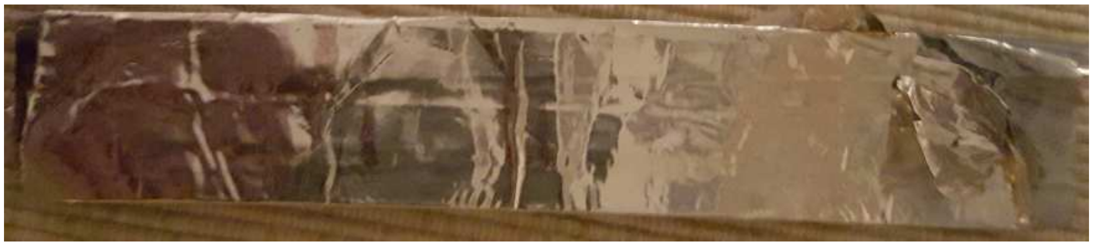


Figure 22: Foil wrapping for camera module ribbon cable

the issue, but when enough room was made for the camera module, the problem persisted. See figure 22.

I replaced the Raspberry Pi camera module with a USB webcam which did not interfere with the GPS. The line in the above bash script using `raspistill` was modified to the following, using the `fswebcam` program obtained with `apt-get`.

```
fswebcam --resolution 1600x1200 /var/www/html/camera/$DATE.jpg
```

4.4. Flight Time

Using the 2200mAh 11.1v three cell batteries, the drone can fly for approximately 15 minutes before the battery's voltage drops below 10v, at which point the drone should be landed. Time varies depending on usage and wind conditions. When the battery drops below around 9.5v, the drone will not have enough power to continue operating safely. I had one instance during flight testing when the battery's voltage was too low, and the drone crash landed from 20 meters altitude.[6] The propellers were still operating, but could not land the drone slow enough. Due to this crash, and a few other accidents when initially testing the drone, a few legs and propellers were damaged and needed to be replaced.

5. Control Program and Server

I wrote the code to control the drone with Python 2.7 the using DroneKit-Python[48] library. This library interfaces with the APM via MAVLink. I used various other libraries for capturing images, processing images, communication, and other features. The other libraries are described in later sections. The code has various configuration options which will be described in their relevant sections.

[6]The voltage of the battery before recharging was 2.7v per cell, and needed to be recharged carefully as described in subsection 4.2.

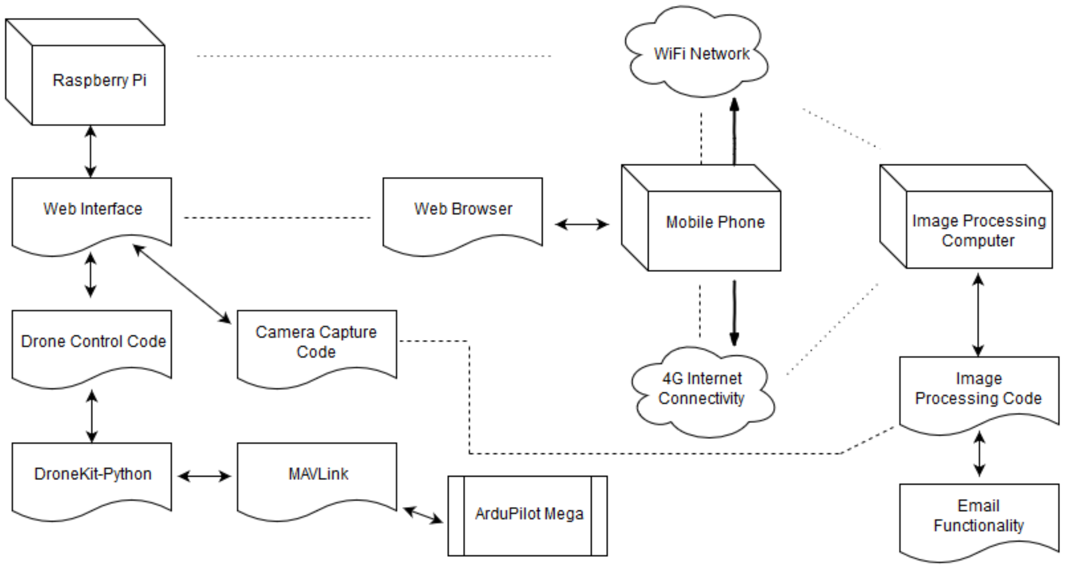


Figure 23: Software Architecture overview, the image processing code is shown in section 6. and section 7.

5.1. Dependencies

The dependencies needed for the code segments in this section of the document can be installed on the Raspbian with the following commands[7]:

```
sudo apt-get install python-dev
sudo pip install dronekit
sudo apt-get install python-opencv
sudo apt-get install espeak
```

`python-dev` installs the python development headers which are needed when DroneKit-Python is installed. The `pymavlink`[91] package is installed along with DroneKit-Python, which is a MAVLink communication library for python. The OpenCV 2 Python library[92] is used for capturing images, and will be

[7]Python import statements are not shown in the code segments for the sake of simplicity.

used for image processing in section 6. eSpeak[77], along with Omxplayer[78] which is pre-installed on Raspbian, are used for text to speech audio output. I also used Ivmech's PID Controller for heading control. It is a simple implementation of a Proportional-Integral-Derivative controller in Python. I made a slight modification to it in order for it to be able to be used as a Python module.[56]

5.2. Capturing Images

After determining I was going to use python for DroneKit-Python, I started working on the image capturing aspect first. I began my codebase by modifying Igor Maculan's Simple Python Motion Jpeg Server[54] into a multi-threaded[8] HTTP server,[55] so that multiple requests could be handled simultaneously.

5.2.1. Capture Thread

I defined a thread to capture the images from the USB webcam using OpenCV, save the images to disk, and store the last image with its length in a global variable.

```
class CaptureThread(Thread):
    def __init__(self):
        ''' Constructor. '''
        Thread.__init__(self)

    def run(self):
        while True:
            try:
                rc,img = capture.read()
```

[8]I am using Python's higher level threading interface, which only allows one thread to execute python code at a time due to the global interpreter lock, but this is still an appropriate model to run multiple tasks simultaneously via context switching.[59]

```
                if not rc:
                    continue
                imgRGB=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
                jpg = Image.fromarray(imgRGB)
                filename = #this is constructed from a combination of
                    coordinates and current time which will be shown in the
                    paper later
                tmpFile = open("images/" + filename, "w")
                jpg.save(tmpFile,'JPEG')
                currentJpg = jpg
                currentJpgLength = tmpFile.tell()
                tmpFile.close()
                time.sleep(1.0/captureRate) #sleep for 1/captureRate, where
                    captureRate is the configuration option defining the
                    number of frames per second to capture.
            except KeyboardInterrupt:
                break
```

The camera is initialized and the capture thread is started[9] with the following if the configuration option `USE_CAMERA` is set to `True`:

```
if USE_CAMERA:
    capture = cv2.VideoCapture(0)
    capture.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, 640)
    capture.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, 360)
    capture.set(cv2.cv.CV_CAP_PROP_SATURATION,0.2)
    captureThreadObj = CaptureThread()
    captureThreadObj.setName('Capture Thread')
    captureThreadObj.start()
```

[9]Threads for other tasks will be shown in later sections. They are started in a similar manner, so their start code has been omitted for the sake of simplicity.

5.2.2. Multithreaded HTTP Server and Image Stream

The `ThreadedHTTPServer` class was created, which allows HTTP requests to be handled in separate threads.

```
class ThreadedHTTPServer(ThreadingMixIn, HTTPServer):
    """Handle requests in a separate thread."""
    bind_and_activate = False # Prevents the class constructor from
        attempting to bind and activate the server
    allow_reuse_address = True #Allows the server's address and port to be
        reused if it was improperly terminated
```

The RequestHandler class is the main HTTP server:

```
class RequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        # self.path is a string containing the HTTP GET request, and this
            function uses pattern matching to handle the requests
        if self.path == '/cam.mjpg' and USE_CAMERA and USE_MJPEG_STREAM:
            # If the camera stream is requested, the camera is enabled with
                USE_CAMERA, and the stream is enabled with USE_MJPEG_STREAM,
                then the request is handled
            # Send the HTTP response headers
            self.send_response(200)
            self.send_header('Content-type','multipart/x-mixed-replace;
                boundary=--jpgboundary')
            self.send_header('Access-Control-Allow-originalin', '*')
            self.end_headers()
            # Loop infinitely and send the last captured frame
            while True:
                try:
                    if currentJpg is not None:
                        self.wfile.write("--jpgboundary")
                        self.send_header('Content-type','image/jpeg')
                        self.send_header('Content-length',
                            str(currentJpgLength))
                        self.send_header('Access-Control-Allow-originalin', '*')
                        self.end_headers()
                        currentJpg.save(self.wfile,'JPEG')
                    time.sleep(1.0/viewRate) #sleep for 1/viewRate, where
                        viewRate is the configuration option defining the
                        number of frames per second to stream.
                except KeyboardInterrupt:
                    break
            return
        elif self.path == '/something/else':
                # Additional paths will be processed here, such as
                    /action/arm, which arms the drone
        elif self.path.startswith('/another/path'):
                # Some paths are handled by pattern matching the prefix, since
                    they contain additional information at the end. Such as
                    /set/destqueue/, which sets a way-point and contains
                    latitude and longitude after the prefix.
    def log_message(self, format, *args):
        # This function is defined with no body to prevent the server from
            logging activity in console output
        return
```

The server is then started with:

```
server = ThreadedHTTPServer(('',8080),RequestHandler)
```

5.3. Web Control Interface

I created a web interface to send commands to the drone, display the camera stream, and view status information from the drone. The main file for this web interface is a HTML file.

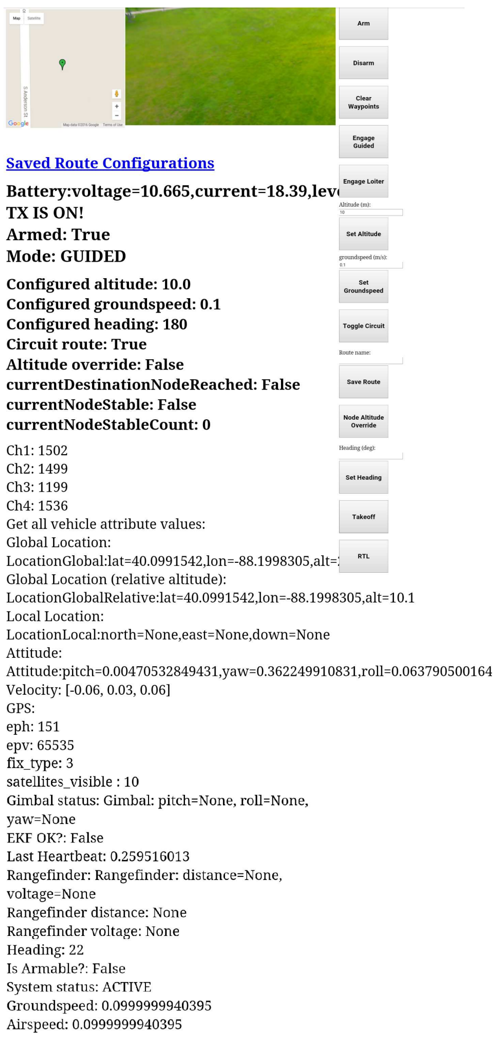
I created a REST-like API[a][86] using HTTP GET request paths. For example, the request ”/action/arm” arms the drone.

This JavaScript function sends an asynchronous HTTP GET request:[57]

```javascript
function httpGetAsync(url) {
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open("GET", url, true);
    xmlHttp.send(null);
}
```

As seen in figure 24, the web interface contains a Google map, camera stream, live status information, text fields, and buttons.

[a]It is not strictly a REST API, since it only uses GET requests. A proper REST API would be an improvement.



Saved Route Configurations

Battery:voltage=10.665,current=18.39,lev
TX IS ON!
Armed: True
Mode: GUIDED

Configured altitude: 10.0
Configured groundspeed: 0.1
Configured heading: 180
Circuit route: True
Altitude override: False
currentDestinationNodeReached: False
currentNodeStable: False
currentNodeStableCount: 0

```
Ch1: 1502
Ch2: 1499
Ch3: 1199
Ch4: 1536
Get all vehicle attribute values:
Global Location:
LocationGlobal:lat=40.0991542,lon=-88.1998305,alt=
Global Location (relative altitude):
LocationGlobalRelative:lat=40.0991542,lon=-88.1998305,alt=10.1
Local Location:
LocationLocal:north=None,east=None,down=None
Attitude:
Attitude:pitch=0.00470532849431,yaw=0.362249910831,roll=0.063790500164
Velocity: [-0.06, 0.03, 0.06]
GPS:
eph: 151
epv: 65535
fix_type: 3
satellites_visible : 10
Gimbal status: Gimbal: pitch=None, roll=None,
yaw=None
EKF OK?: False
Last Heartbeat: 0.259516013
Rangefinder: Rangefinder: distance=None,
voltage=None
Rangefinder distance: None
Rangefinder voltage: None
Heading: 22
Is Armable?: False
System status: ACTIVE
Groundspeed: 0.0999999940395
Airspeed: 0.0999999940395
```

Figure 24: Web control interface

The text fields and buttons use the `httpGetAsync(url)` JavaScript function in order to send information to the drone. This example below arms the drone[10]:

```
<button onclick="httpGetAsync('/action/arm')" style="width: 150px;height:
    100px;position: absolute; LEFT:1020px; TOP:0px;"><h2>Arm</h2></button>
```

The live drone status information is stored in a `div`:

```
<div id="status">STATUS LOADING</div>
```

This is updated with the following JavaScript:

```
function updateStatus(){
    var xmlHttp = new XMLHttpRequest({
        mozSystem: true
    });

    xmlHttp.open("GET", "/status.html", false);
    xmlHttp.send(null);
    var res = xmlHttp.responseText;
    document.getElementById("status").innerHTML = res;
}
```

5.3.1. Google Maps API

The Google Maps JavaScript API[58] was used to display the current location of the drone, shown as a green marker in figure 24 along with way-points. The map is initialized with the following JavaScript code:

```
var map; // The map
var dronelatLng; // Current drone latitude and longitude
var marker = null; // Drone position marker
var markers = []; // Waypoint markers
```

[10]See subsubsection 5.5.1. for the Python web server side of this code

```
var lastMarkerRetval = ""; // Stores the previous way-point markers

// Initializes the map
function initMap() {
    getLatLon(); // Get the current latitude and longitude of the drone
    // Parser function parses the latitude and longitude string into a Google
        Maps latitude/longitude object
    var latlng = getLatLngFromString(dronelatLng);
    // Initialize the map
    map = new google.maps.Map(document.getElementById('map'), {
        center: latlng,
        zoom: 19
    });
    // Add a listener for a tap on the map, which will add a new way-point
    map.addListener('click', function(e) {
        placeMarkerAndPanTo(e.latLng, map);
    });
}

// Gets the current latitude and longitude of the drone
function getLatLon() {
    var xmlHttp = new XMLHttpRequest({
        mozSystem: true
    });
    xmlHttp.open("GET", "/get/latlon", false);
    xmlHttp.send(null);
    dronelatLng = xmlHttp.responseText;
}

// When the map is tapped, a request is sent containing the new way-point
function placeMarkerAndPanTo(latLng, map) {
    var xmlHttp = new XMLHttpRequest({
        mozSystem: true
    });
    xmlHttp.open("GET", "/set/destqueue/" + latLng.toString(), false);
    xmlHttp.send(null);
}
```

The map object is inserted into the HTML file using the following, where `$GOOGLEMAPSAPIKEY` is the Google Maps API key obtained from a Google Developer account.[58]

```
<script src="https://maps.googleapis.com/maps/api/js?
key=$GOOGLEMAPSAPIKEY&callback=initMap&sensor=false" async defer></script>
```

This JavaScript updates the map and the drone's status HTML on a set interval, in this case every 500ms:

```
function updateFunction() {
    getLatLon(), placeMarker(), placeDestMarkers(), updateStatus()
}
setInterval(updateFunction, 500);
```

`getLatLon()` is shown above, and gets the current latitude and longitude of the drone. `updateStatus()` is also shown above, and updates the drone status HTML.

`placeMarker()`, shown below, adds a green marker icon[74] to the map with the current position of the drone. `placeDestMarkers()`, shown below, gets the current way-point queue from the drone, clears the current way-point markers, and adds the current way-points to the map.

```
function placeMarker() {
    var latlng = getLatLngFromString(dronelatLng);
    if(marker == null){
        marker = new google.maps.Marker({
            position: latlng,
            map: map,
            icon: 'green_marker.png'
        });
    } else {
        marker.setPosition(latlng);
    };
}

function placeDestMarkers() {
    var xmlHttp = new XMLHttpRequest({
        mozSystem: true
    });
    xmlHttp.open("GET", "/get/destqueue", false);
    xmlHttp.send(null);
    var destList = xmlHttp.responseText;
    // Ensures that the new markers are not identical to the existing ones
    if(destList != lastMarkerRetval){
        lastMarkerRetval = destList;
        destList = destList.replace(/[({})]/g, '');
        destList = destList.split(',');
        deleteMarkers(); // Helper function to clear the current markers
        var j = 0;
        for (var i = 0; i < destList.length; i+=2){
            addMarker(new google.maps.LatLng(parseFloat(destList[i]),
                parseFloat(destList[i+1])), j++);
        }
    }
}
```

5.4. Start Script

I used a bash script to start my code while developing. Errors in the code could cause the program to hang, or the APM to refuse new connections. Python instances are killed before the code starts, and after it exits. `usbreset`[62] is a C program which resets the APM's USB device.

```
pkill -9 python # Kill leftover python instances
```

```
sudo ./usbreset /dev/bus/usb/001/004 # Reset usb device
python server.py # Start the server, control, and image capture code
pkill -9 python # Kill leftover python instance
```

The APM's USB device was found with `lsusb`.

```
 $ lsusb
Bus 001 Device 005: ID 046d:0809 Logitech, Inc. Webcam Pro 9000
Bus 001 Device 004: ID 2341:0010 Arduino SA Mega 2560 (CDC ACM)
Bus 001 Device 003: ID 0424:ec00 Standard Microsystems Corp. SMSC9512/9514 Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Standard Microsystems Corp.
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

5.4.1. USB Reset

The following is the code for `usbreset`[62]:

```c
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/ioctl.h>
#include <linux/usbdevice_fs.h>

int main(int argc, char **argv)
{
    const char *filename;
    int fd;
    int rc;

    if (argc != 2) {
        fprintf(stderr, "Usage: usbreset device-filename\n");
        return 1;
    }
    filename = argv[1];
    fd = open(filename, O_WRONLY);
    if (fd < 0) {
        perror("Error opening output file");
        return 1;
    }

    printf("Resetting USB device %s\n", filename);
    rc = ioctl(fd, USBDEVFS_RESET, 0);
    if (rc < 0) {
        perror("Error in ioctl");
        return 1;
    }
    printf("Reset successful\n");

    close(fd);
    return 0;
}
```

This is compiled with `gcc usbreset.c -o usbreset`.

5.5. Controlling the Drone

I initially based my code off of `DroneKit-Python's` Simple Go To example.[53] Using argparse, the APM's USB device is specified, then the connection is established if the configuration variable `CONNECT_VEHICLE` is `True`.

```
parser = argparse.ArgumentParser()
parser.add_argument('--connect', default='/dev/ttyACM0',
                    help="vehicle connection target. Default '/dev/ttyACM0'")
args = parser.parse_args()

if CONNECT_VEHICLE:
    vehicle = connect(args.connect, wait_ready=True)

    # Get Vehicle Home location - will be `None` until first set by autopilot
```

```
while not vehicle.home_location:
    cmds = vehicle.commands
    cmds.download()
    cmds.wait_ready()
    if not vehicle.home_location:
        print " Waiting for home location ..."
# We have a home location, so print it!
print "\n Home location: %s" % vehicle.home_location

# Set vehicle home_location, mode, and armed attributes (the only
    settable attributes)
# Home location must be within 50km of EKF home location (or setting will
    fail silently)
my_location_alt=vehicle.location.global_frame
my_location_alt.alt=222.0 # Altitude of Urbana, IL
vehicle.home_location=my_location_alt

# Set the vehicle's mode to stablize initially
vehicle.mode = VehicleMode("STABILIZE")
while not vehicle.mode.name=='STABILIZE': #Wait until mode has changed
    print " Waiting for mode change ..."
    time.sleep(1)
```

5.5.1. Arming

When the APM is armed from the web interface, the following code on the web server is executed to arm the APM[11].

```
elif self.path == '/action/arm':
            self.send_response(200)
            self.send_header('Content-type','text/html')
            self.send_header('Access-Control-Allow-originalin', '*')
```

[11]There are many similar segments of code, where the web server handles an HTTP requests to change state, send data, or update data. Most of these are not included in this document for the sake of simplicity.

```
        self.end_headers()

        if CONNECT_VEHICLE:
            armingInProgress = True # This variable is displayed in the
                status HTML
            vehicle.armed = True
            while not vehicle.armed:
                print " Waiting for arming..."
                time.sleep(1)
            armingInProgress = False

        return
```

There is a suite of pre-arm safety checks which will prevent the APM from arming if the required conditions are not met.[93] These include but are not limited to: bad GPS fix, bad readings from the barometer and accelerometer which usually go away by having the drone sit for a short time, and other hardware failures.
The parameters of the pre-arm safety check can be modified, this was necessary when testing functionality indoors, the following is an example:[12]

```
vehicle.parameters['GPS_HDOP_GOOD'] = 300.0 # Sets the minimum GPS_HDOP
    which is required to arm, default is 230
vehicle.parameters['ARMING_CHECK'] = 0 # Disables the pre-arming checks
    all-together, setting to 1 enables the checks
```

After landing, the APM will automatically disarm after a short time.

[12]HDOP is a measure of GPS precision.[75]

5.5.2. Takeoff

There are two methods for takeoff after arming the APM:

1. Keeping the APM in stabilize mode, or setting it to loiter mode, then manually taking off using the radio transmitter.

2. Putting the APM into guided mode, then using `vehicle.simple_takeoff(configAlt)`, where `configAlt` is the globally configured relative altitude[13]. The autopilot will then takeoff and hold above its start position.[14]

5.5.3. Autonomous GPS Guided Flight

As seen in subsection 5.3., the web interface handles setting way-points. Way-points are stored in a queue, and when the option `configCircuit` is `True` nodes will be added back to the queue when they are flown to. This allows the drone to autonomously fly in a continuous circuit over a pre-defined area. See figure 26 and 25. Way-points are stored in a `DestNode` class containing the latitude, longitude, altitude, and heading of each way-point. When the node is reached, the drone will orient to the specified heading as seen in subsubsection 5.5.5. The drone will approximately maintain this set heading when flying to the next node.

[13]Relative altitude is defined in subsubsection 5.5.3.

[14]The EMERGENCY_LANDING_ENABLED option described in subsubsection 5.5.6. needs to be set to False for this to work, otherwise the emergency landing code will place the APM into land mode. To work around this, the APM needs to be placed in loiter mode and the radio transmitter turned on to minimum throttle first.

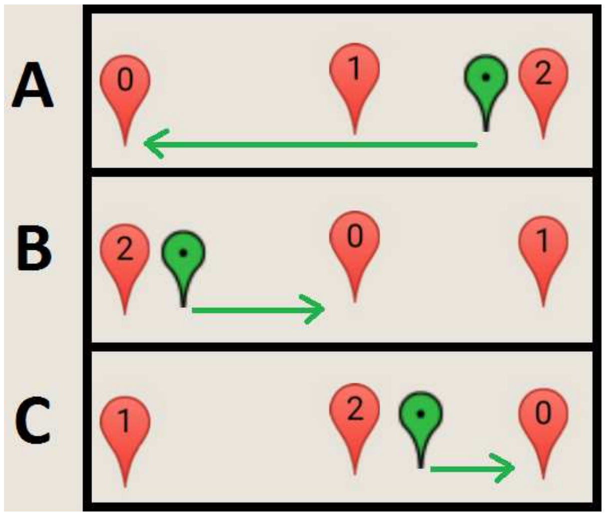


Figure 25: Example with three states of the queue for a three node circuit.

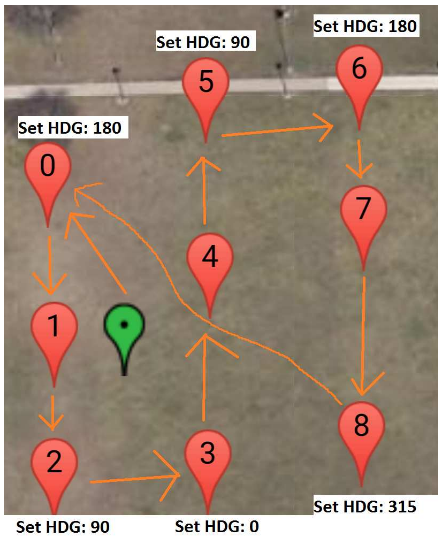


Figure 26: Example circuit route with nine nodes

The following is the code for the `DestNode` class:

```
class DestNode:
    def __init__(self, lat, lng, alt, hdg):
        self.lat = lat
        self.lng = lng
        self.alt = alt
        self.hdg = hdg
```

When the map is tapped on the web interface, the HTTP server handles the request "/set/destqueue" below. A new `DestNode` is created and placed into a queue. This is reflected on the web interface.

```
elif self.path.startswith('/set/destqueue'):
          latLngString = self.path.split('/')[-1]
          latLngString = latLngString.replace("%20", " ")
          latLngTuple = tuple(float(x) for x in
              latLngString.strip('()').split(','))
          lat = latLngTuple[0]
          lng = latLngTuple[1]
```

```
            node = DestNode(lat, lng, configAlt, configHeading)
            destQueue.put(node)

            self.send_response(200)
            self.send_header('Content-type','text/html')
            self.send_header('Access-Control-Allow-originalin', '*')
            self.end_headers()
            self.wfile.write('True')
            return
```

When the drone is placed into guided mode via the web interface, it begins autonomous flight.

```
elif self.path == '/action/guided':
            self.send_response(200)
            self.send_header('Content-type','text/html')
            self.send_header('Access-Control-Allow-originalin', '*')
            self.end_headers()
            if CONNECT_VEHICLE:
                guidedNextNode()

            return
```

This function handles engaging the autopilot into guided mode, and starts flight to the next node. If the setting, `configCircuit` is `True`, then the node is added back to the end of the queue. `LocationGlobalRelative` uses the relative altitude, not absolute. For example, since Urbana, IL's altitude is 222m, if the node's altitude is 10m, then the drone will fly to an altitude of 232m.

```
def guidedNextNode():
    if destQueue.empty():
        return

    # Place the APM into the guided flight mode.
    modeChangeInProgress = True
    vehicle.mode = VehicleMode("GUIDED")
    while not vehicle.mode.name=='GUIDED': #Wait until mode has changed
        print " Waiting for mode change ..."
        time.sleep(1)

    modeChangeInProgress = False
    node = destQueue.get()
    # If the configuration variable, configCircuit is True, then the node is
        placed back onto the end of the queue.
    if configCircuit:
        destQueue.put(node)
    currentDestinationNode = node
    configHeading = node.hdg # Set the global heading to the node's heading
    currentDestinationNodeReached = False
    # If the configuration variable, configAltitudeOverride is True, then the
        global altitude variable is used instead of the node's altitude.
    if configAltitudeOverride:
        location = LocationGlobalRelative(node.lat, node.lng, configAlt)
    else:
        location = LocationGlobalRelative(node.lat, node.lng, node.alt)

    # The autopilot begins to fly to the node at the configured groundspeed
    vehicle.simple_goto(location, groundspeed=configGroundspeed)
```

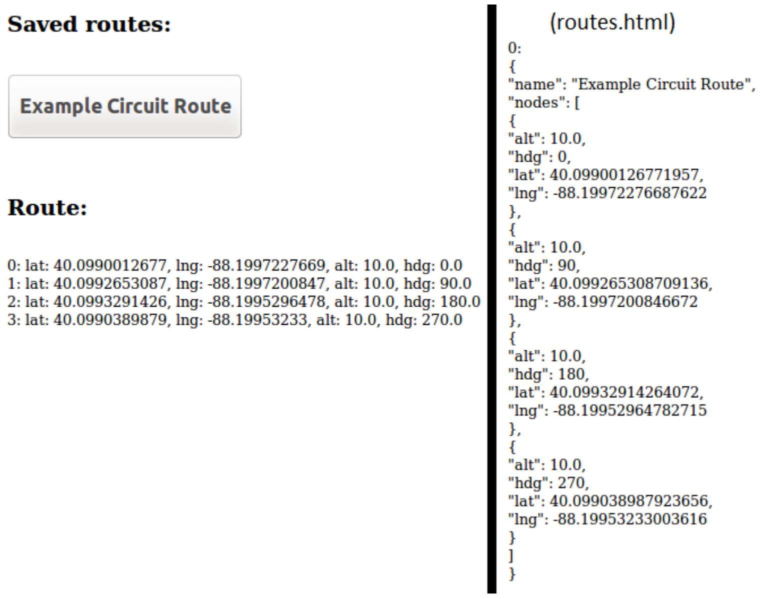
Routes can also be saved and loaded to/from the JSON format. The code below saves the current route to a JSON file.

```
elif self.path.startswith('/set/route'):
            name = self.path.split('/')[-1]
            name = name.replace("%20", " ") # Un-escape spaces
            data = {'name': name, 'nodes': []} # begin building the JSON data

            for node in list(destQueue.queue):
                new_json_node = {'lat': node.lat, 'lng': node.lng, 'alt':
                    node.alt, 'hdg': node.hdg}
                data['nodes'].append(new_json_node)

            file = open(json_routes_prefix + str(json_routes_index) +
                json_routes_suffix, "w")
            file.write(json.dumps(data, sort_keys=True, indent=4,
                separators=(',', ': ')))
            file.close()
            json_routes_names_list.append(name)
            json_routes_index=json_routes_index+1

            self.send_response(200)
            self.send_header('Content-type','text/html')
            self.send_header('Access-Control-Allow-originalin', '*')
            self.end_headers()
            self.wfile.write('True')
            return
```

These are stored in the `routes` directory with the format `routeN.json`, where N is an integer, as defined by the below code:

```
json_routes_prefix = "routes/route"
json_routes_suffix = ".json"
json_routes_index = 0
json_routes_names_list = []
# This code loads the route names into a list for display.
while os.path.isfile(json_routes_prefix + str(json_routes_index) +
    json_routes_suffix):
    json_data = open(json_routes_prefix + str(json_routes_index) +
        json_routes_suffix)
    data = json.load(json_data)
    json_routes_names_list.append(data['name'])
    json_routes_index = json_routes_index+1
```

These routes are displayed on the web interface as seen in figure 27. Below the status information in figure 24, **Saved routes:** is displayed with buttons for each route. Clicking this button will load the route and it will display on the map and in the **Route:** section. Clicking the **Saved Route Configurations** hyperlink as shown in figure 24, will display the `routes.html` file generated by the web server, showing the raw JSON for each route.



```
0: lat: 40.0990012677, lng: -88.1997227669, alt: 10.0, hdg: 0.0
1: lat: 40.0992653087, lng: -88.1997200847, alt: 10.0, hdg: 90.0
2: lat: 40.0993291426, lng: -88.1995296478, alt: 10.0, hdg: 180.0
3: lat: 40.0990389879, lng: -88.19953233, alt: 10.0, hdg: 270.0
```

```
0:
{
"name": "Example Circuit Route",
"nodes": [
{
"alt": 10.0,
"hdg": 0,
"lat": 40.09900126771957,
"lng": -88.19972276687622
},
{
"alt": 10.0,
"hdg": 90,
"lat": 40.099265308709136,
"lng": -88.1997200846672
},
{
"alt": 10.0,
"hdg": 180,
"lat": 40.09932914264072,
"lng": -88.19952964782715
},
{
"alt": 10.0,
"hdg": 270,
"lat": 40.099038987923656,
"lng": -88.19953233003616
}
]
}
```

Figure 27: Route buttons, current route, and routes.html

5.5.4. Reaching the Node

The `RouteThread` class handles reaching the node, then moving onto the next node. The `isclose` helper function was back-ported from Python 3.[79]

```python
class RouteThread(Thread):
    def __init__(self):
        ''' Constructor. '''
        Thread.__init__(self)

    def run(self):


        while True:
            if currentDestinationNodeReached:
                # Node is reached.
                # Heading control is now handled here, and timeAtNode is
                    updated to the timestamp when the drone was stabilized at
                    the node.

                # If the configuration variable, CONFIG_TIME_AT_NODE is not 0,
                    the drone wait at this node for CONFIG_TIME_AT_NODE
                    seconds, then the drone moves onto the next node.
                if CONFIG_TIME_AT_NODE != 0 and timeAtNode != 0 and
                    int(time.time()) > timeAtNode + CONFIG_TIME_AT_NODE:
                    guidedNextNode()

            elif currentDestinationNode is not None:
                # Node is not reached, and a destination is set.
                if configAltitudeOverride:
                    currentAlt = configAlt
                else:
                    currentAlt = currentDestinationNode.alt
                # This function isclose, is used to check if the drone is
                    approximately at the node
                if isclose(currentDestinationNode.lat,
                    vehicle.location.global_relative_frame.lat,
                    abs_tol=0.00001) and isclose(currentDestinationNode.lng,
                    vehicle.location.global_relative_frame.lon,
                    abs_tol=0.00001) and isclose(currentAlt,
                    vehicle.location.global_relative_frame.alt, abs_tol=0.5):
                    # If the isclose checks pass, the node was reached
                    currentDestinationNodeReached = True
                    # Start heading control here

            time.sleep(0.1)

def isclose(a, b, rel_tol=1e-09, abs_tol=0.0):
    return abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)
```

5.5.5. Proportional-Integral-Derivative Heading Controller

Ivmech's PID controller was used to align the drone on the correct heading when the node was reached. The pulse width modulation of the radio transmitter's yaw channel is overridden as part of the process.[83] A PID controller is a control loop feedback system, where proportional control accounts for present values of the error, integral control accounts for accumulated values of the error, and derivative control accounts for possible future values of the error.[76]

The `# Start heading control here` section in the previous code contains the following. This is where the PID controller is initialized. The coefficients $p = 0.2$, $i = 0.2$, and $d = 0$ were obtained experimentally using the drone simulator described in subsubsection 6.2.2. I am only using proportional and integral control. The integral control is needed to account for when the drone is close enough to the correct heading, but error accumulates, it gets slightly nudged to the correct heading.

```
currentNodeStable = False
currentNodeStableCount = 0
pid = ivPID.PID.PID(0.2, 0.2, 0)
pid.SetPoint=0.0
pid.setSampleTime(0.1)
```

The `# Heading control is now handled here` section in the previous code contains:

```
heading = vehicle.heading
# Compute the difference in degrees between the current heading and desired
    heading for both directions: if the drone was to yaw right or left
rightDiff = positiveDiff(configHeading - heading)
leftDiff = positiveDiff(heading - configHeading)
if isclose(heading, configHeading, abs_tol=5):
    # If the heading is within 5 degrees of the desired heading, we consider
```

```
            it stable.
        if currentNodeStableCount < 30:
            # Increment a counter, we want the drone to be holding this heading
                for 30 iterations before we consider it stable.
            currentNodeStableCount = currentNodeStableCount+1
        elif currentNodeStable is False:
            # If the drone has been stable for 30 iterations, we consider it
                stable.
            currentNodeStable = True
            timeAtNode = int(time.time()) #start unix timestamp
            currentNodeHeading = configHeading
    else:
            # If the heading is not within 5 degrees, the drone is not stable.
        currentNodeStableCount = 0
        currentNodeStable = False

    # Now the PID controller is updated with the current heading difference,
        using either the right or left difference, whichever is less.
    if rightDiff < leftDiff:
        pid.update(-rightDiff)
    else:
        pid.update(leftDiff)

    output = pid.output
    # Now override the yaw channel's pulse width modulation with scaled down
        output from the PID controller, multiplied by the stable heading PWM,
        and added to the stable heading PWM.
    overridePWM = STABLE_YAW + (output*0.01)*STABLE_YAW
    # If this value is above the maximum desired rate of turn for left or right
        yaw, it is set to the maximum desired rate.
    if overridePWM < LEFT_YAW:
        overridePWM = LEFT_YAW
    elif overridePWM > RIGHT_YAW:
        overridePWM = RIGHT_YAW
```

```
vehicle.channels.overrides['4'] = overridePWM
```

The following constants are used above, which were determined experimentally:

```
STABLE_YAW = 1525 # The PWM which holds the drone in a stable position
LEFT_YAW = 1300 # The PWM causing the maximum desired left yaw rate of turn
RIGHT_YAW = 1750 # The PWM causing the maximum desired right yaw rate of turn
```

5.5.6. Landing

There are three methods to land the drone normally:

1. Putting the APM into loiter mode via the web interface, then manually landing using the radio transmitter.
2. If `EMERGENCY_LANDING_ENABLED` is `True`, turn off the radio transmitter while not in loiter, stabilize, or land mode. This will enable the emergency landing protocol described below which puts the APM into land mode.
3. Putting the APM into RTL (return-to-land) mode via the web interface, and having the autopilot land itself at the starting location.

The second method operates using the `EmergencyLandingThread`:

```
class EmergencyLandingThread(Thread):
    def __init__(self):
        ''' Constructor. '''
        Thread.__init__(self)

    def run(self):
        while True:
                # vehicle.channels['3'] <= 900 detects that the radio
                    transmitter was turned off, this channel will not go below
                    this value unless the transmitter is off.
```

```
            if vehicle.channels['3'] <= 900 and vehicle.mode.name != "LAND"
                and vehicle.mode.name != "LOITER" and vehicle.mode.name !=
                "STABILIZE":
                vehicle.channels.overrides['4'] = None #Removes the yaw
                    override
                # Clear the destination nodes so that the drone will not
                    resume guided flight
                currentDestinationNode = None
                currentDestinationNodeReached = False
                # Put the APM into land mode.
                vehicle.mode = VehicleMode("LAND")
                time.sleep(3)
            time.sleep(0.1)
```

A fallback option, if `LAND_WHEN_BATTERY_LOW` is set to `True`, is to have the drone automatically switch to RTL mode when the battery is below a minimum battery voltage defined with `MIN_BATTERY_VOLTAGE`. This is done via the `SystemMonitorThread`:

```
class SystemMonitorThread(Thread):
    def __init__(self):
        ''' Constructor. '''
        Thread.__init__(self)

    def run(self):
        while True:
            if vehicle.battery.voltage < MIN_BATTERY_VOLTAGE:
                if LAND_WHEN_BATTERY_LOW and not vehicle.mode.name=='RTL':
                    modeChangeInProgress = True
                    vehicle.mode = VehicleMode("RTL")
                    while not vehicle.mode.name=='RTL': #Wait until mode has
                        changed
                        time.sleep(1)

            time.sleep(5)
```

5.5.7. Testing with drone simulator

The drone simulator, `dronekit-sitl`[60] was used on an up-to-date Ubuntu 15.10 system while developing and testing aspects of the drone control functionality. It is installed with:

```
sudo apt-get install python-pip python-dev
pip install dronekit-sitl -UI
```

Running `dronekit-sitl copter` starts a copter simulator which listens on the TCP port 5760. As seen in subsection 5.5., connection is established in my code with `--connect tcp:IPADDRESS:5760` passed in as a parameter.

5.6. Audio Playback

Audio playback via the `speak(str)` function below, uses text to speech to acknowledge various commands and states[15] such as ”Drone Armed”. It uses `espeak` to create a WAV audio file which is then played back with `omxplayer` if the configuration variable `USE_AUDIO` is set to `True`.

```
def speak(s):
    if USE_AUDIO:
        Popen(["/usr/bin/espeak", "\"" + s + "\"", "-ven-us+f4", "-s200",
            "-w", "tmp.wav"], close_fds=True)
        Popen(["/usr/bin/omxplayer", "tmp.wav", "--vol", "1000"],
            close_fds=True)
```

5.7. Logging

Logging is enabled when the configuration variable `LOGGING_ENABLED` is set to `True`. The log file is opened with the current time-stamp.

[15]speak was omitted in the code snippets shown in order to keep them shorter.

```
if LOGGING_ENABLED:
    now = time.strftime("%c")
    flightLogFile = open("logs/flight-" + now.replace(" ", "-").replace("--",
        "-") + ".log", "a")
```

Then the `writeToLog(str)` function is used in various areas of the code[16] in order to log the drones activity during flight. These logs were used for debugging by analyzing them after flight.

```
def writeToLog(str):
    if LOGGING_ENABLED:
        flightLogFile.write(str)
```

5.8. Shutting down

The code is shutdown when it receives the SIGINT signal[17], which invokes the following handler function:

```
def handler(signum, frame):
    if CONNECT_VEHICLE:
        vehicle.close() # Close vehicle object
    if USE_CAMERA:
        capture.release() # Close camera
    server.socket.close() # Close web server
    # Stop running threads
    for thr in threading.enumerate():
        while thr.isAlive():
            try:
                thr._Thread__stop()
            except:
                print "Additional thread could not be terminated"
        if LOGGING_ENABLED:
```

[16]writeToLog was omitted in the code snippets shown in order to keep them shorter.
[17]SIGINT is created by the key combination control+C in the running terminal.

```
        flightLogFile.close() # Close log file
    sys.exit()
```

The SIGINT handler is setup with:

```
signal.signal(signal.SIGINT, handler)
```

6. Image Processing

For human detection, I began researching how to use OpenCV's object detection capabilities since I was already using OpenCV to capture images. I found an article, *Adrian Rosebrock's: Pedestrian Detection OpenCV*[64], by the author of imutils[84], with an example of pedestrian detection using OpenCV.

I used OpenCV's object detection, which uses a Histogram of Oriented Gradients[69] method in conjunction with a Linear Support Vector Machine.[70][71]

6.1. Dependencies

The dependencies for the Python image processing code can be installed with the following:

```
sudo apt-get install python-numpy
sudo pip install imutils
sudo pip install cv2
```

6.2. Initial Attempt

I initially attempted to implement an algorithm to identify humans in overhead images in an open field with the drone at an altitude of 30 meters and with the camera at an angle of approximately 45 degrees downwards.

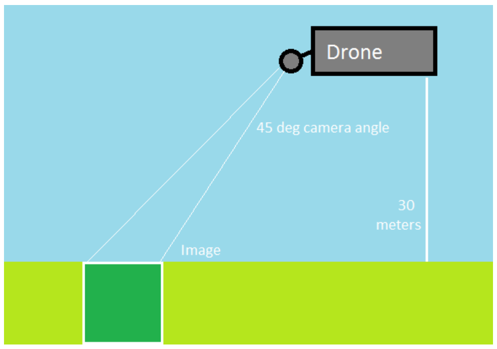


Figure 28: Illustration of drone at 30m, with camera at a 45 degree angle

6.2.1. Capturing Test Images

I initially collected a test image set of 5212 images. 2528 of the images are positive, where there is a human in them. 2684 are negative images, where there is not a human. These images were collected by having the drone fly over routes, and moving myself into and out of the camera's view.

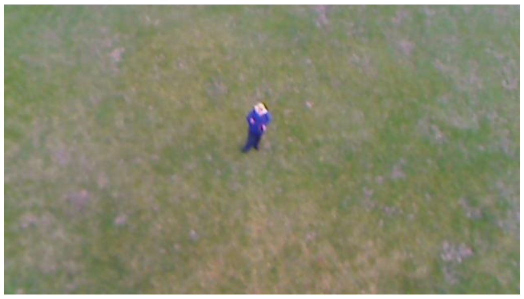


Figure 29: Example positive image

Figure 30: Example negative image

6.2.2. Algorithm

Adrian Rosebrock's: Pedestrian Detection OpenCV [64], uses OpenCV's `DefaultPeopleDetector`. The example expects close-range, vertically and

horizontally level images, and did not function on my test images. I modified his example for my image processing code.
I used `trainHOG`,[63], which uses the svmlight engine[85], to train a new HOG/SVM with my test images. `trainHOG` is compiled with the following on the Ubuntu 15.10 system mentioned in :

```
sudo apt-get install git pkg-config libopencv-dev
git clone https://github.com/DaHoC/trainHOG
cd trainHOG/
cd svmlight
wget http://download.joachims.org/svm_light/current/svm_light.tar.gz
tar -xf svm_light.tar.gz
make
cd ..
g++ ‘pkg-config --cflags opencv‘ -c -g -MMD -MP -MF main.o.d -o main.o main.cpp
gcc -c -g ‘pkg-config --cflags opencv‘ -MMD -MP -MF svmlight/svm_learn.o.d \
    -o svmlight/svm_learn.o svmlight/svm_learn.c
gcc -c -g ‘pkg-config --cflags opencv‘ -MMD -MP -MF svmlight/svm_hideo.o.d \
    -o svmlight/svm_hideo.o svmlight/svm_hideo.c
gcc -c -g ‘pkg-config --cflags opencv‘ -MMD -MP -MF svmlight/svm_common.o.d \
    -o svmlight/svm_common.o svmlight/svm_common.c
g++ ‘pkg-config --cflags opencv‘ -o trainhog main.o svmlight/svm_learn.o \
    svmlight/svm_hideo.o svmlight/svm_common.o ‘pkg-config --libs opencv‘
```

I split the test set approximately in half into a training set and validation set. `trainHOG` expects the image to be of a resolution 64x128, so I needed to resize the images using OpenCV. Since my images were horizontal instead of vertical, I rotated them clockwise by 90 degrees.
Below is Python code to rotate and resize the images:

```
filePaths = list(paths.list_images("images"))
for filePath in filePaths:
    filename = filePath[filePath.rfind("/") + 1:]
    image = cv2.imread(filePath)
    # Rotate with PIL
    pil_image = Image.fromarray(image)
    pil_image = pil_image.rotate(270, expand=True)
    # Resize with OpenCV
    image = np.array(pil_image)
    image = cv2.resize(image, (64,128))
    cv2.imwrite("imagesout/" + filename, image)
```

After re-sizing, the images were moved into 'pos' and 'neg' directories, separated by positive and negative images respectively, in `trainHOG`'s working directory. Then the machine is trained by running `./trainhog`.
After training, a file, `genfiles/cvHOGClassifier.yaml` is created containing the SVMDetector vector needed by OpenCV. I then wrote the image processing code using this vector:

```
hog = cv2.HOGDescriptor() # initialize the HOG descriptor
hog.setSVMDetector(np.array([ ... The SVMDetector vector goes here ...]))
filePaths = list(paths.list_images("images"))
positives = 0 # Number of positive detections
negatives = 0 # Number of negatives
maxWeight = 0 # Max weight seen
lapse = [] # An array keeping track of the number of lapses between positives
lapseCount = 0 # Keep track of the current running lapse count
for filePath in filePaths:
    filename = filePath[filePath.rfind("/") + 1:]
    image = cv2.imread(filePath)
    original = image.copy() # Save the original image for output
    # Rotate with PIL
    pil_image = Image.fromarray(image)
    pil_image = pil_image.rotate(270, expand=True)
    # Resize with OpenCV
    image = np.array(pil_image)
    image = cv2.resize(image, (64,128))
    # Detection
    (rects, ((weight))) = hog.detect(image)
    # Keep track of maximum weight seen
    if len(weight) is 1 and weight > maxWeight:
        maxWeight = weight

    if len(rects) > 0 and len(weight) is 1 and weight > MINWEIGHT:
        # MINWEIGHT will be the minimum weight of the detection we want in
            order to consider this a positive.
        # Positive
        lapse.append(lapseCount)
        lapseCount = 0
        positives = positives + 1
        cv2.imwrite("imagesout/pos/" + filename, original)
    else:
        # Negative
        lapseCount = lapseCount + 1
        negatives = negatives + 1
        cv2.imwrite("imagesout/neg/" + filename, original)

# When the algorithm is done, print the results:
print "POSITIVES: " + str(positives)
print "NEGATIVES: " + str(negatives)
print "MAXWEIGHT: " + str(maxWeight)
print "LAPSES: " + str(lapse)
```

6.2.3. Results from Test Data

By running this with different values of `MINWEIGHT`, I was able to find a good value which would cause the training set to have a very small number of false positives, but approximately a 10% true positive detection rate. I determined that this is a good enough detection rate since the camera will be capturing at at 12-24 frames per second, so if a person is in view for one second then

they should be detected.
However, when running against the validation set the results were not good. There were too many false positives .
I retrained the machine with the entire test set, and it functioned with the following results.
Output from `trainHOG` when generating the vector:

```
Testing training phase using training set as test set
(just to check if training is ok - no detection quality conclusion with this!)
Results:
True Positives: 2184
True Negatives: 2387
False Positives: 297
False Negatives: 344
Testing custom detection using camera
```

Output from my Python code:

```
POSITIVES: 373
NEGATIVES: 4839
MAXWEIGHT: [[ 1.63662334]]
LAPSES: [410, 28, 561, 2, 32, 128, 5, 15, 9, 0, 4, 0, 0, 96, 2, 30, 33, 55,
50, 4, 1, 2, 1, 5, 1, 0, 3, 48, 1, 10, 1, 2, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 2, 0, 1,
2, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 7, 0, 1, 6, 2, 0, 1, 0, 2,
0, 0, 3, 3, 0, 5, 9, 0, 1, 0, 3, 5, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 185, 0, 2, 2, 1, 1, 23, 36, 3,
0, 5, 1, 0, 1, 6, 1, 1, 1, 7, 3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 5, 14, 7, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
5, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 3, 318, 6, 3, 78, 410, 3, 78, 0, 2, 0, 0, 0, 2, 1, 2, 2, 0, 0, 0, 2, 4,
1, 32, 18, 0, 1, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0, 0,
8, 2, 0, 2, 3, 0, 1, 0, 2, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1,
20, 0, 0, 1, 3, 2, 2, 0, 1, 1, 8, 1, 0, 0, 0, 1, 0, 1, 0, 0, 2, 0, 1, 0, 0,
0, 0, 1, 2, 0, 0, 0, 0, 0, 1, 17, 9, 7, 3, 13, 17, 6, 6, 9, 0, 6, 0, 1, 818,
9, 135, 25, 244, 0, 276, 103, 6, 29, 59, 1]
```

Again there were a very small number of false positives, and about a 14% detection rate. However, I was concerned about the lapse length between correct detentions. The mean lapse was 12.8097, but the largest lapse was 818 images. This would be a lapse of 68.1677 seconds at 12fps, or 34.0833 at 24fps. I was also concerned about over-fitting to these specific images since the validation set preformed poorly initially.
The performance speed for this algorithm was fast enough so it could run on the Raspberry Pi.

6.2.4. Results from Live Data

When testing the algorithm live, the algorithm was barely functional with a multitude of false positives and false negatives. I determined that I needed a better algorithm.

6.3. Final Algorithm

I decided that using OpenCV's `DefaultPeopleDetector` was the best course of action at this point. I collected a test set with the drone at an altitude of 10 meters and a camera angle of approximately 15 degrees. This algorithm would not depend on the surrounding area to be an open field, but would function better if it was. Then I modified my previous code to use the built-in person detector.

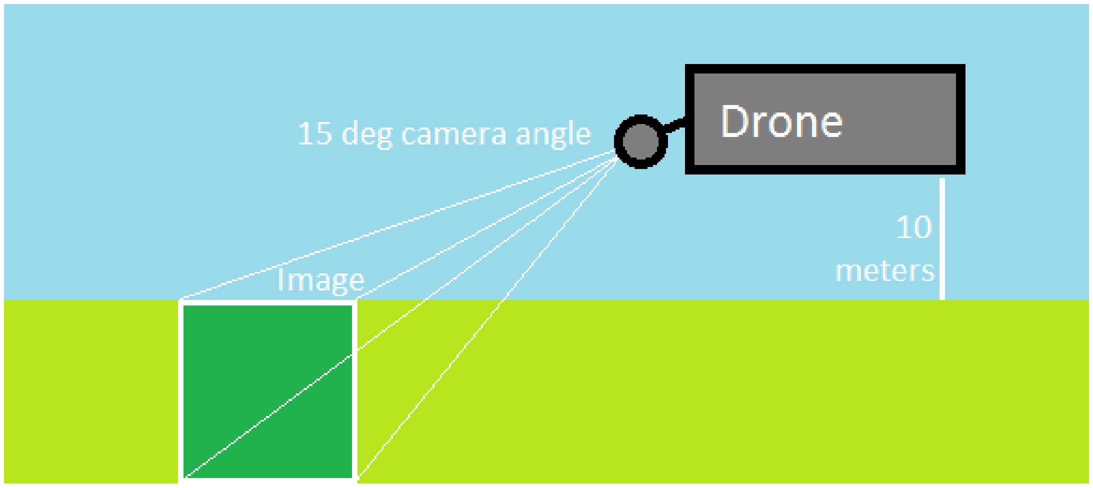


Figure 31: Illustration of drone at 10m, with camera at a 15 degree angle

I collected a test image set of 2799 positive images, some with obstacles in the way, such as a tree, car, and park bench.

6.3.1. Algorithm

I modified my previous code to the following:

```
hog = cv2.HOGDescriptor() # initialize the HOG descriptor
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
filePaths = list(paths.list_images("images"))
positives = 0 # Number of positive detections
negatives = 0 # Number of negatives
for filePath in filePaths:
    filename = filePath[filePath.rfind("/") + 1:]
    image = cv2.imread(filePath)

    # Initial detection, this is faster and detects people at a far distance
        better
    (rects, weights) = hog.detectMultiScale(image, winStride=(2, 2),
        padding=(8, 8), scale=4.2)
    # If no people were detected, we run a slower, close-range detection. The
        image is resized for better results.
    if len(rects) == 0:
        image = imutils.resize(image, width=min(400, image.shape[1]))
        (rects, weights) = hog.detectMultiScale(image, winStride=(4, 4),
            padding=(8, 8), scale=1.05)

     # draw rectangles around detected people
    for (x, y, w, h) in rects:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)

    if len(rects) > 0:
        # Positive detection
        positives = positives + 1
        cv2.imwrite("imagesout/pos/" + filename, image)
    else:
        # Negative
        negatives = negatives + 1
        cv2.imwrite("imagesout/neg/" + filename, image)

# When the algorithm is done, print the results:
print "POSITIVES: " + str(positives)
print "NEGATIVES: " + str(negatives)
```

I used *Adrian Rosebrock's: HOG detectMultiScale parameters explained*[65] article in order to figure out the best parameters to use with `detectMultiScale` in the above code.

I attempted values for `winStride` between `(1,1)` and `(8,8)`. `(8,8)` was too inaccurate for resized images, `(1,1)` was too slow, and `(4,4)` was fine for resized images, but was too inaccurate for people further away. `(2,2)` was fast enough with enough accuracy for a further distance, so it is used first. The `scale` is enlarged to 4.2 to increase processing time. 4.2 was four seconds faster compared to 2.1. `scale` values larger than 4.2 did not help increase processing time by a significant amount. The original values from *Adrian Rosebrock's: Pedestrian Detection OpenCV* [64] article were used for the closer range detection.

6.3.2. Results

The algorithm functioned well, making 1819 positive detections, and 980 negatives. Most of the positive detections were correct, however some incorrectly identified a tree or park bench as being a person, usually when the image was blurred due to drone movement. The algorithm usually decides that an image is negative if a person is only partially in the frame, such as half of the body or only the torso.

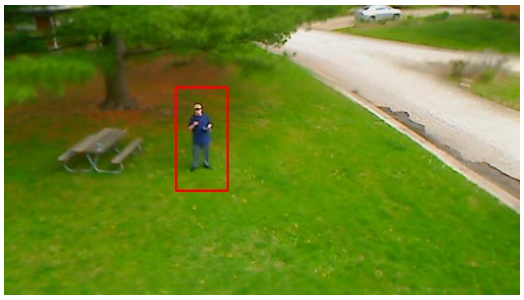


Figure 32: Example positive image

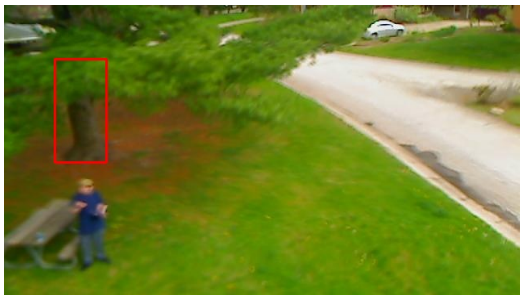


Figure 33: Example false positive image

7. Live Image Processing Implementation

Since the Raspberry Pi could not process images fast enough, I needed to send the images to another, more powerful, machine as a dedicated image processing computer. I used an ASUS G750JX-DB71 laptop with Ubuntu 14.04.4 LTS installed. The ASUS laptop can process approximately 3 images per second.

7.1. Dependencies

The additional dependencies needed for transferring the images from the Raspberry Pi to the image processing computer can be installed on Raspbian with the following commands:

```
sudo apt-get install libffi-dev
sudo pip install cryptography
sudo pip install pysftp
```

The pysftp[80] library is used in order to send images over SFTP (SSH File Transfer Protocol) to the dedicated image processing computer. This library depends on libffi[81], and the cryptography python library.[82]

7.2. Receiving Images from the Drone

The image processing computer had the user 'pi' added to it. The `CaptureThread` in the Raspberry Pi drone code had the following added to it, so that if the option `CONFIG_SEND_IMAGES` was set to `True`, then images would be sent to the image processing machine.

```
if CONFIG_SEND_IMAGES:
    sftpSendCounter = 0
    print "SENDING IMAGE."
    try:
        with pysftp.Connection('IP_ADDRESS', username='pi',
            password='PASSWORD_GOES_HERE') as sftp:
            with sftp.cd('image-processing/images'): # temporarily chdir
                sftp.put("images/" + filename) # upload file
    except:
        print "Failed to connect to SFTP host."
```

In order to process the images when they were uploaded to the computer, and display them, the `image = cv2.imread(filePath)` line in the previous image processing code from subsection 6.3. was changed to the following:

```
# Ensures the file is not empty when it is being processed
if os.stat("images" + "/" + filename).st_size > 0:
    image = cv2.imread("images" + "/" + filename)
    # Displays the last image
    cv2.imshow('Drone Image Feed', image)
    cv2.waitKey(1)
```

Additionally, the for loop in the code was changed to have a while loop surrounding it which checks for new files uploaded to the machine:

```
# Empty set initially
imagePaths = set([])
# Launch the image display window
cv2.namedWindow('Drone Image Feed', cv2.WINDOW_AUTOSIZE)
while True:
    imagePathsOld = imagePaths
    imagePaths = set(os.listdir("images"))
    # Get the new files, difference between current and new
    currentFiles = list(imagePaths - imagePathsOld)
    os.chdir("images")
    # Sort by time so they are processed in correct order
    currentFiles.sort(key=os.path.getmtime)
    os.chdir("..")
    for filename in currentFiles:
        # Do the image processing here
```

7.3. Email Functionality

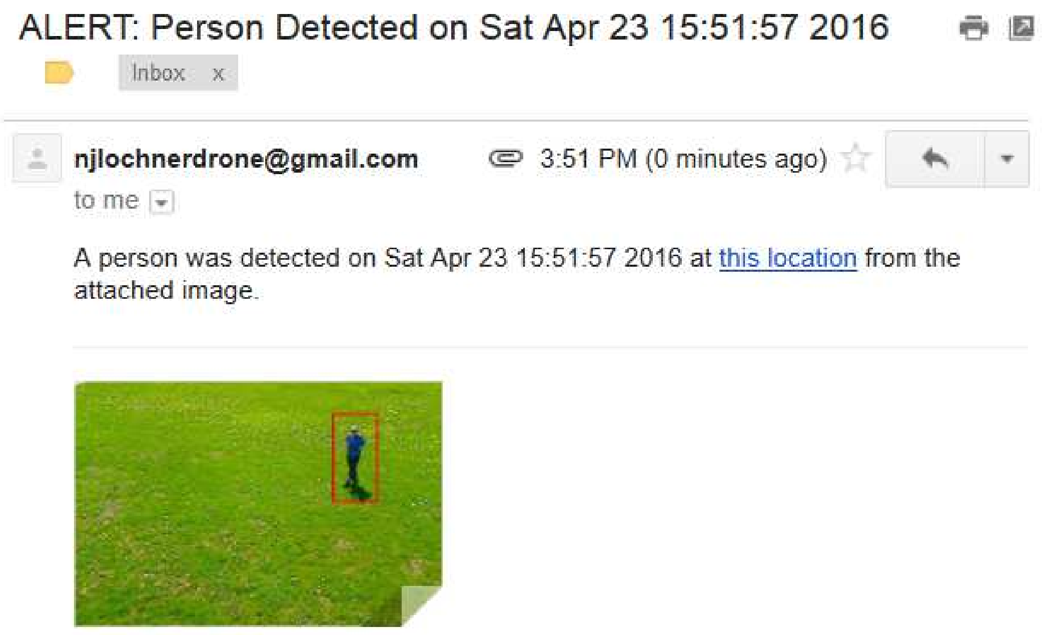
A Python script was created in order to send an email when a person is detected by the image processing computer. Similarly to the image processing script, this script checks for new files in the ”imagesout/pos” directory. When a new image is detected, it sends an HTML email via GMail with the image, location, and current time.

This function sends the email:

```
def send_email(user, pwd, recipient, subject, img, lat, lng, timestamp):
    gmail_user = user
    gmail_pwd = pwd
    TO = recipient if type(recipient) is list else [recipient]
    msg = MIMEMultipart()
    msg['From'] = user
    msg['To'] = ",".join(TO)
    msg['Subject'] = subject
    msg.preamble = subject

    html = """\
    <html>
      <head></head>
      <body>
        A person was detected on """ + timestamp + """ at <a
            href="https://www.google.com/maps/?q=""" + lat + "," + lng +
            """">this location</a> from the attached image.
      </body>
    </html>
    """
    part = MIMEText(html, 'html')
    msg.attach(part)
    msg.attach(img)
    try:
        server = smtplib.SMTP("smtp.gmail.com", 587)
        server.ehlo()
        server.starttls()
        server.login(gmail_user, gmail_pwd)
        server.sendmail(FROM, TO, msg.as_string())
        server.close()
    except:
        print "failed to send mail"
```

When a new image is detected, the image is read and the function is called. The latitude and longitude are parsed from the filename.

```
image = cv2.imread(args["images"] + "/" + filename)
parts = filename.split("_")
lat = parts[0]
lng = parts[1]

timestamp = time.strftime("%c")
fp = open(args["images"] + "/" + filename, 'rb')
img = MIMEImage(fp.read())
fp.close()
send_email("someone@gmail.com", "PASSWORD_GOES_HERE", [ ... List of email
    address to send to ...], "ALERT: Person Detected on " + timestamp, img,
    lat, lng, timestamp)
```



ALERT: Person Detected on Sat Apr 23 15:51:57 2016

Inbox x

njlochnerdrone@gmail.com 3:51 PM (0 minutes ago)

to me

A person was detected on Sat Apr 23 15:51:57 2016 at this location from the attached image.

Figure 34: An email sent from the image processing code

The hyperlink in the email shown in figure 34 goes to the URL, https://www.google.com/maps/?q=40.098989,-88.1994112. See figure 35.

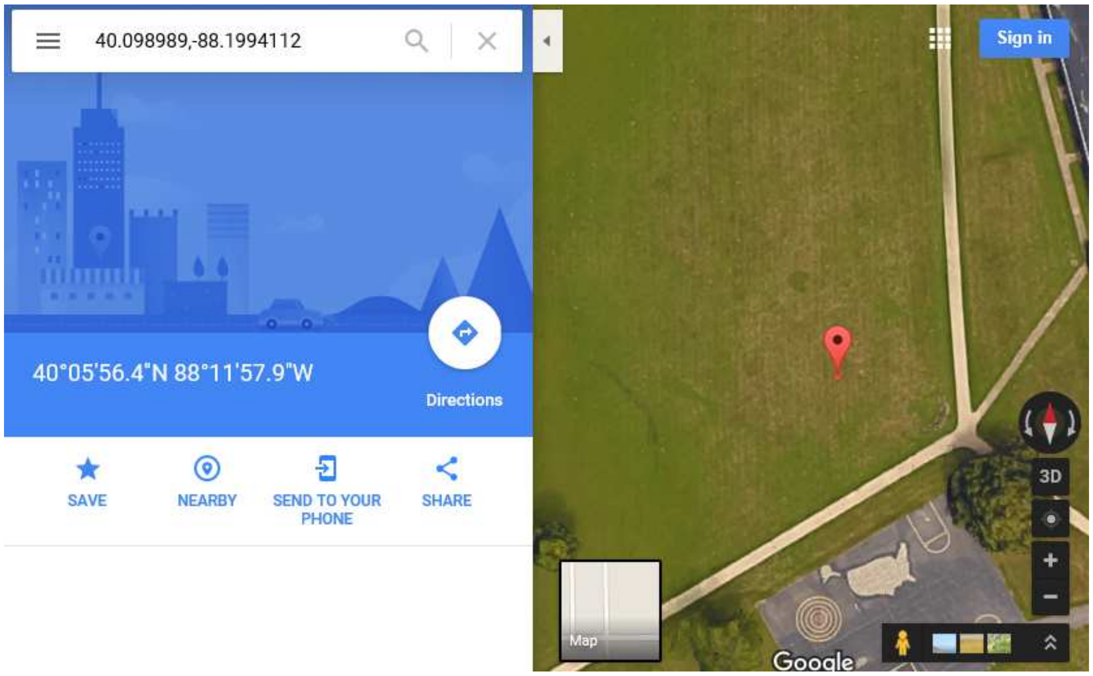


Figure 35: The Google Maps link contained in the email

7.4. Live Results

The live results of the final image processing algorithm were good. Out of 600 images taken, 199 were correct positives. There were no false positives, as it was tested over an open field, and the false negative rate was approximately 50%, usually when the person was out of the frame or if the image was blurred. A 50% correct positive rate is much better than the initial algorithm's results. On average, one out of every two frames containing a person would correctly identify them. At a capture rate between 12-24 frames per second, a person should be identified correctly if they appear within the camera's view.

8. Conclusion

In conclusion, I have shown that autonomous drones are a completely viable option for the automatic surveillance of open areas of property. Individuals, businesses, and government entities could use them to improve security. The implementation I have described in this paper allows a drone to takeoff, patrol a pre-defined area, and return to its takeoff location fully autonomously.

However, this implementation is not completely automatic since the flight time is approximately 15 minutes. Someone will need to be handling the replacement and charging of batteries. An implementation with a robotic charging bay could solve this issue. This would have the drone takeoff from a charging bay, where the charging bay would automatically replace and recharge batteries, with robotic arms for instance. Inductive charging[66] (wireless charging), could be another solution.

There is also the concern about the drone having only a 50% rate of correct human detection. I do not believe this is a large concern, as the drone can travel much faster than a human can run. This could be alleviated by having a fleet of drones patrol in a pattern, possibly communicating with each other so that they are synchronized with their positions.

Due to the issues mentioned with incorrect identifications when the image is blurred a better camera could help alleviate this issue. Having the drone rotate 360 degrees when reaching a point could help alleviate the issue of people being partially out of the frame not being recognized. A stereo camera with two lenses in order to perceive depth, or a better algorithm could also aid in recognition. An example of this is DJI's new drone.[67] An additional future improvement could be to use a microphone on the drone, but this would be a challenge to filter out the propeller noise.

Autonomous surveillance drones could also be used for uses other than surveilling private property. The drone could receive GPS information from the mobile phone via the web interface and follow the owner capturing images for additional security as they are walking in a public area, such as a park. Police can use drones to aid in chases of suspects, or surveil an area for suspicious activity. Drones can also be used to patrol over roads and highways, identifying the speed of cars.

However, uses of drones come with safety hazards. A malfunction could cause the drone to crash and damage property, injure, or even kill a person. In the United States, there are also many regulations towards operating drones[68]. For individual use, the Federal Aviation Administration (FAA) requires that the drone is kept in line of sight of the operator, so legally you can not have a drone autonomously patrolling the area around your house. For business purposes, the drone and its use needs to be approved by the FAA.

A Appendix: Parts and Components List

This appendix contains the full parts and components list and their descriptions.

Part Name	**Description**
Raspberry Pi (Pi 2 Model B or Pi 3 Model B)	Drone companion computer
Raspberry Pi Case	
32GB Micro SD Card	Storage for Raspberry Pi
APM 2.6+ Assembled (ArduPilot Mega)	Autopilot board based on Arduino
3DR uBlox GPS with Compass Kit	GPS and compass, connects to the APM
Edimax EW-7318USg USB WiFi Dongle	USB WiFi Dongle for Raspberry Pi
Logitech Tessar 2.0/3.7 USB Webcam	Camera for the Raspberry Pi
Hobby King 2.4Ghz 6Ch Transmitter and Receiver V2 (Mode 2)	Radio transmitter and receiver used to manually pilot the drone
DJI Flamewheel F450 ARF Kit	Kit including the drone's body, arms, power distribution board, DJI E-SERIES 420 LITE electronic speed controllers, DJI 2312E motors, and 9 inch propellers
Two DJI Flame Wheel 450/550 Landing Gear Sets	Landing gear with spares in case of crash damage
Spare DJI Phantom 2 Vision Self-Tightening 9 Inch Propellers	Spare propellers in case of crash damage
Boom Cube Speaker	Speaker for Raspberry Pi Audio, has built-in battery charged by mini USB

APM Power Module with XT60 Connectors Kit	Pass-through XT60 cable with power module board. The power module supplies 2.25A at 5.37V and provides voltage monitoring. Includes connection cable to APM.[6]
Two Zippy Flightmax 2200mAh 3S1P 40C Batteries	11.1V Lithium polymer batteries with XT60 connectors
Female XT60 Connector	Used to solder to battery connector cables to the power distribution board
UBEC DC/DC Step-Down (Buck) Converter - 5V @ 3A output	Step down converter to power the Raspberry Pi
Two USB to Micro-USB cables	Used for connecting the Raspberry Pi to the APM, and for connecting the power cable from the UBEC to the Raspberry Pi
Three male to male servo cables	Connects the radio receiver to the APM.
One male to female servo cable	Connects the power cable from the UBEC to the Raspberry Pi
Plastic Zip-ties	Zip-ties used for securing components
VELCRO Brand - Industrial Strength - 2 Inch Strips	Velcro strips used for securing components
Soldering iron, heat gun, solder, wire stripper	Used when soldering wires to connections on the power distribution board and soldering wires together
Heat shrink tubing and electrical tape	Used to cover cables after soldering
SMAKN 1-8S Battery Checker	Used to check battery voltage before and after flight
Tenergy TB6-B Balance Charger	Lithium polymer battery charger
XT60 Charge Cable Banana Plug	Charging cable for batteries
BW Fireproof Lipo Battery Safe Bag	Contains the batteries while charging in case of a fire

References

[1] Raspberry Pi
https://en.wikipedia.org/wiki/Raspberry_Pi

[2] Wikipedia: Real-time operating system
https://en.wikipedia.org/wiki/Real-time_operating_system

[3] Wikipedia: Ardupilot
https://en.wikipedia.org/wiki/Ardupilot

[4] ArduPilot Documentation: Altitude Hold Mode
http://ardupilot.org/copter/docs/altholdmode.html

[5] Federal Aviation Administration: Aircraft Registry – Aircraft Registration: Unmanned Aircraft (UA)
https://www.faa.gov/licenses_certificates/aircraft_certification/aircraft_registry/UA/

[6] ArduPilot Documentation: Powering the APM2 – Copter Documentation
http://ardupilot.org/copter/docs/common-powering-the-apm2.html

[7] DJI: F450 User Manual v2.2
http://www.dji.com/product/flame-wheel-arf/download
http://dl.djicdn.com/downloads/flamewheel/en/F450_User_Manual_v2.2_en.pdf

[8] ArduPilot Documentation: DJI F330 FlameWheel QuadCopter Assembly – Copter Documentation
http://ardupilot.org/copter/docs/dji-f330-flamewheel.html

[9] DJI: Landing Gear for Flame Wheel
http://www.dji.com/product/flame-wheel-arf/download
http://dl.djicdn.com/downloads/flamewheel/Landing_Gear_for_Flame_Wheel.pdf

[10] ModMyPi: How do I power my Raspberry Pi
http://www.modmypi.com/blog/how-do-i-power-my-raspberry-pi

[11] DIY Drones: Connect USB and power module at the same time?
http://diydrones.com/forum/topics/connect-usb-and-power-module-at-the-same-time

[12] Jethro Hazelhurst: Typical Quadcopter Layout
http://www.arducopter.co.uk/all-arducopter-guides/2connectingeverything-for-arducopter

[13] ArduPilot Documentation: Connect ESCs and Motors – Copter Documentation
http://ardupilot.org/copter/docs/connect-escs-and-motors.html

[14] ArduPilot Documentation: Stabilize Mode
http://ardupilot.org/copter/docs/stabilize-mode.html

[15] Raspberry Pi Foundation: Download Raspbian for Raspberry Pi
https://www.raspberrypi.org/downloads/raspbian/

[16] Raspberry Pi Foundation: Index of /raspbian/images/raspbian-2015-09-28
https://downloads.raspberrypi.org/raspbian/images/raspbian-2015-09-28/

[17] Raspberry Pi Foundation: Index of /raspbian/images/raspbian-2016-02-29
https://downloads.raspberrypi.org/raspbian/images/raspbian-2016-02-29/

[18] SourceForge: Win32 Disk Imager
https://sourceforge.net/projects/win32diskimager/

[19] Raspberry Pi Documentation: Installing operating system images
https://www.raspberrypi.org/documentation/installation/installing-images/README.md

[20] Wikipedia: Wi-Fi Protected Access
https://en.wikipedia.org/wiki/Wi-Fi_Protected_Access

[21] Wikipedia: CCMP
https://en.wikipedia.org/wiki/CCMP

[22] How-To Geek: Warning: Encrypted WPA2 Wi-Fi Networks Are Still Vulnerable to Snooping
http://www.howtogeek.com/204335/warning-encrypted-wpa2-wi-fi-networks-are-still-vulnerable-to-snooping/

[23] Null Byte: How to Hack Wi-Fi: Cracking WPA2-PSK Passwords Using Aircrack-Ng
http://null-byte.wonderhowto.com/how-to/hack-wi-fi-cracking-wpa2-psk-passwords-using-aircrack-ng-0148366/

[24] Rootsh3ll Wi-Fi Security and Pentesting Series (RWSPS): Cracking WPA2-PSK with Aircrack-ng [ch3pt4]
http://www.rootsh3ll.com/2015/09/rwsps-wpa2-cracking-aircrack-ng-dictionary-attack-ch3pt4/

[25] How Secure Is My Password?
https://howsecureismypassword.net/

[26] Information Security Stack Exchange: Why is WPA Enterprise more secure than WPA2?
http://security.stackexchange.com/questions/35780/why-is-wpa-enterprise-more-secure-than-wpa2

[27] Eric Geier: Deploying WPA2-Enterprise Wi-Fi Security in Small Businesses
http://www.windowsnetworking.com/articles-tutorials/wireless-networking/Deploying-WPA2-Enterprise-Wi-Fi-Security-Small-Businesses.html

[28] Raspberry Pi Documentation: raspi-config
https://www.raspberrypi.org/documentation/configuration/raspi-config.md

[29] GitHub: ArduPilot/apm_planner: APM Planner Ground Control Station (Qt)
https://github.com/ArduPilot/apm_planner

[30] ArduPilot Documentation: Installing Mission Planner
http://ardupilot.org/copter/docs/common-install-mission-planner.html

[31] Wikipedia: MAVLink
https://en.wikipedia.org/wiki/MAVLink

[32] ArduPilot Documentation: MAVLink Commands
http://ardupilot.org/dev/docs/mavlink-commands.html

[33] ArduPilot Documentation: APM 2.5 and 2.6 Overview
http://ardupilot.org/copter/docs/common-apm25-and-26-overview.html

[34] ArduPilot Documentation: Loading Firmware onto Pixhawk/APM2.x/PX4
http://ardupilot.org/copter/docs/common-loading-firmware-onto-pixhawk.html

[35] ArduPilot Documentation: Mandatory Hardware Configuration
http://ardupilot.org/copter/docs/configuring-hardware.html

[36] ArduPilot Documentation: Electronic Speed Controller (ESC) Calibration
http://ardupilot.org/copter/docs/esc-calibration.html

[37] ArduPilot Documentation: Compass Calibration in Mission Planner
http://ardupilot.org/copter/docs/common-compass-calibration-in-mission-planner.html

[38] Wikipedia: Magnetic declination
https://en.wikipedia.org/wiki/Magnetic_declination

[39] DIY Drones: NEW FEATURE: Automatic Compass Declination
http://diydrones.com/profiles/blogs/new-feature-automatic-compass-declination

[40] ArduPilot Documentation: Accelerometer Calibration in Mission Planner
http://ardupilot.org/copter/docs/common-accelerometer-calibration.html

[41] ArduPilot Documentation: Advanced Compass Setup
ardupilot.org/copter/docs/common-compass-setup-advanced.html

[42] We Work We Play: Automatically connect a Raspberry Pi to a Wifi network
http://weworkweplay.com/play/automatically-connect-a-raspberry-pi-to-a-wifi-network/

[43] Polytechnic University of Catalonia: Example wpa_supplicant configuration file
http://www.cs.upc.edu/lclsi/Manuales/wireless/files/wpa_supplicant.conf

[44] Raspberry Pi Forums: How to disable the Pi3's WLAN & Bluetooth?
https://www.raspberrypi.org/forums/viewtopic.php?f=63&t=138610

[45] Dorkbot PDX: External antenna modifications for the Raspberry Pi 3
http://www.dorkbotpdx.org/blog/wramsdell/external_antenna_modifications_for_the_raspberry_pi_3

[46] The MagPi Magazine: Raspberry Pi 3 is out now! Specs, benchmarks & more
https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/

[47] ArduPilot Documentation: Flight Modes
http://ardupilot.org/copter/docs/flight-modes.html

http://diydrones.com/forum/topics/copter-propellers-rotation-incorrect

[48] GitHub: dronekit/dronekit-python: DroneKit-Python library for communicating with Drones via MAVLink.
https://github.com/dronekit/dronekit-python

[49] IndoorFPV.com: LiPo Batteries- Key Points To Remember
http://indoorfpv.com/content/lipo-batteries-key-points-remember

[50] Gabriel Staples: Restoring/Recharging Over-discharged LiPo (Lithium Polymer) Batteries!
http://www.electricrcaircraftguy.com/2014/10/restoring-over-discharged-LiPos.html

[51] Revolectrix: Battery C Rating
http://www.revolectrix.com/tech_data/lipoCalc/Battery_C_Rating.htm

[52] All-Battery.com: Tutorial: Tenergy TB6B Balance Charger for NiMH/LiPO/LiFe Battery Packs
https://www.youtube.com/watch?v=5AIAomVTKHw

[53] DroneKit-Python: Example: Simple Go To (Copter)
http://python.dronekit.io/examples/simple_goto.html

[54] Igor Maculan: Simple Python Motion Jpeg (mjpeg server) from webcam. Using: OpenCV,BaseHTTPServer
https://gist.github.com/n3wtron/4624820

[55] Stack Overflow: Multithreaded web server in python
http://stackoverflow.com/a/14089457

[56] GitHub: nikkifayra/ivPID forked from ivmech/ivPID: Python PID Controller
https://github.com/nikkifayra/ivPID

[57] Stack Overflow: HTTP GET request in JavaScript?
http://stackoverflow.com/a/4033310

[58] Google Developers: Google Maps JavaScript API
https://developers.google.com/maps/documentation/javascript/

[59] Python 2.7.11 Documentation: threading Higher-level threading interface
https://docs.python.org/2/library/threading.html

[60] DroneKit Documentation: Setting up a Simulated Vehicle (SITL)
http://python.dronekit.io/develop/sitl_setup.html

[61] ArduPilot Documentation: Loiter Mode
http://ardupilot.org/copter/docs/loiter-mode.html

[62] Stack Exchange, Ask Ubuntu: How do you reset a USB device from the command line?
http://askubuntu.com/a/661

[63] GitHub: DaHoC/trainHOG: Example program showing how to train your custom HOG detector using openCV
https://github.com/DaHoC/trainHOG

[64] Adrian Rosebrock: Pedestrian Detection OpenCV (Adrian informed me via email that the sample is under the MIT License.)
http://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/

[65] Adrian Rosebrock: HOG detectMultiScale parameters explained
http://www.pyimagesearch.com/2015/11/16/hog-detectmultiscale-parameters-explained/

[66] Wikipedia: Inductive charging
https://en.wikipedia.org/wiki/Inductive_charging

[67] The Verge: The revolutionary chipmaker behind Googles project Tango is now powering DJIs autonomous drone
http://www.theverge.com/2016/3/16/11242578/movidius-myriad-2-chip-computer-vision-dji-phantom-4

[68] Federal Aviation Administration: Unmanned Aircraft Systems (UAS) Frequently Asked Questions
https://www.faa.gov/uas/faq/

[69] Wikipedia: Histogram of oriented gradients
https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

[70] Wikipedia: Support vector machine
https://en.wikipedia.org/wiki/Support_vector_machine

[71] Adrian Rosebrock: Histogram of Oriented Gradients and Object Detection
http://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/

[72] Google Play: ConnectBot
https://play.google.com/store/apps/details?id=org.connectbot&hl=en

[73] Google Play: VNC Viewer
http://www.pyimagesearch.com/2014/11/10/histogram-oriented-gradients-object-detection/

[74] Stack Overflow: Google Maps API 3 - Custom marker color for default (dot) marker
http://stackoverflow.com/questions/7095574/google-maps-api-3-custom-marker-color-for-default-dot-marker

[75] Wikipedia: Dilution of precision (GPS)
https://en.wikipedia.org/wiki/Dilution_of_precision_%28GPS%29

[76] Wikipedia: PID controller
https://en.wikipedia.org/wiki/PID_controller

[77] eSpeak: Speech Synthesiser
http://espeak.sourceforge.net/

[78] eLinux.org: Omxplayer
http://elinux.org/Omxplayer

[79] Stack Overflow: What is the best way to compare floats for almost-equality in Python?
http://stackoverflow.com/a/33024979

[80] Python Package Index: pysftp 0.2.8
https://pypi.python.org/pypi/pysftp

[81] Wikipedia: libffi
https://en.wikipedia.org/wiki/Libffi

[82] Python Package Index: cryptography 1.3.2
https://pypi.python.org/pypi/cryptography

[83] DroneKit Documentation: Example: Channels and Channel Overrides
http://python.dronekit.io/examples/channel_overrides.html

[84] GitHub: jrosebr1/imutils
https://github.com/jrosebr1/imutils

[85] SVMlight: Support Vector Machine
http://svmlight.joachims.org/

[86] Wikipedia: Representational state transfer
https://en.wikipedia.org/wiki/Representational_state_transfer

[87] Raspberry Pi Documentation: Using a standard USB webcam
https://www.raspberrypi.org/documentation/usage/webcams/

[88] Raspberry Pi Documentation: raspistill
https://www.raspberrypi.org/documentation/usage/camera/raspicam/raspistill.md

[89] NGINX
https://www.nginx.com/resources/wiki/

[90] TightVNC: VNC-Compatible Free Remote Control / Remote Desktop Software
http://www.tightvnc.com/

[91] GitHub: mavlink/pymavlink/
https://github.com/mavlink/mavlink/tree/master/pymavlink

[92] OpenCV
http://opencv.org/

[93] Pre-Arm Safety Check
http://ardupilot.org/copter/docs/prearm_safety_check.html

[94] Stack Overflow: Is there a chart of which OSS License is compatible with which?
http://stackoverflow.com/questions/1978511/is-there-a-chart-of-which-oss-license-is-compatible-with-which

[95] Free Software Foundation: GNU General Public License Version 3
http://www.gnu.org/licenses/gpl-3.0.en.html

[96] Association for Computing Machinery: Guidance for Authors on Fair Use
http://www.acm.org/publications/guidance-for-authors-on-fair-use

Copyright and Fair Use Notice, Information, and Disclaimers