UNMANNED AERIAL VEHICLES

SELIN KAYAY
CANDIDATE NO: 5034
CENTER NO: 11001

Contents

Analysis

0.1 Background

Quadcopters are often limited in terms of control and ability to adjust to a dynamic environment. Humans have sensory organs covered with receptors allowing detection of these changes to which we can process and decide on how to react to the stimulus. This allows us to survive in the situation of a threat and we can continue learning from these experiences with our ability to store past sensory experiences and process them with the help of what we call neural networks which is briefly our CNS (CNS - central nervous system).

ANN - Artificial Neural Networks are imitations of these biological neural networks that can be used in deep learning of the program which gives way to an artificially intelligent program once trained sufficiently. Implementing such features when programming a UAV could give way to an intelligent machine which is aware of its surroundings and able to make decisions based upon this.

Most simple drones possess the ability to follow instructions through a remote where we input directions for them to then carry out. This is useful in applications such as delivery, search and rescue operations, safety and surveillance etc. However, these drones are under heavy limitations when it comes to the fact that weather is variable, especially since climate change has made daily conditions more and more unpredictable, and the environment is unconditionally dynamic regardless of already mapped out terrain by satellites. Moving objects and resistance factors, most commonly air resistance, cause a lot of cases where the drones are lost, damaged or terminated due to the crashing of the software as the parameters within which the drone operates are exceeded.

I was made aware of these limitations when I signed up for STEM club at the start of year 12 where I was assigned to setup a quadcopter and supervise younger kids as they operated it. The quadcopter had very sensitive controls and crashed often due to a lack of calculation and awareness of environment. The controls were inputted entirely by the user which caused a crash very often as the users attempted multiple inputs at once such as a backflip as you move downwards.

In this project I will explore intelligent UAVs in terms of their operation and possible features such as autonomous travel. To take it further I will explore neural networks and object detection algorithms.

0.2 Scope

The goal of the project is to produce a simple drone with manual controls then take it a step further with the ability to detect objects and return information about them – such as what they are and where they are. The drone built will be a 250 quadcopter and the tech specs will be explored further on. The object detection and data processing features will be running on a Raspberry Pi model 3 B+.

The figure below shows the main ideas about the system:

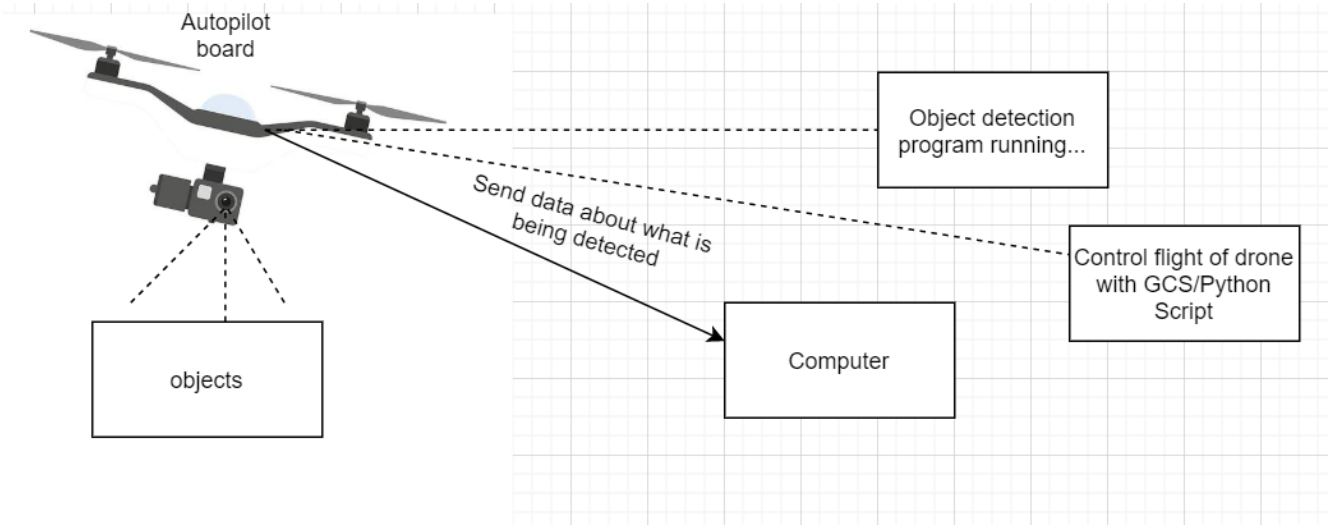


Figure 1

Figure 1 showcases the 3 simple steps provided by Nanonets, which is an API that I will further research in order to produce my objectives (I will not be using Nanonets in my project).
Considerations for further research:

1. For further insight, I must know how other autonomous drones work and how an object detection program will be produced.
2. Before autonomy, a prototype creation of a manually controlled quadcopter must be produced. This is very simple with the use of software such as LibrePilot, compatible with the flight controller managing the flight.

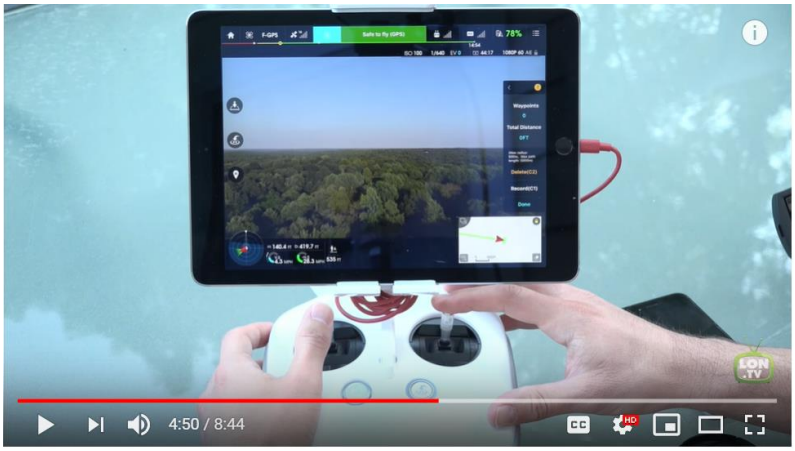
0.3 An Investigation into Autonomous Flight

What's on the market already?

DJI Phantom Series

- Obstacle Avoidance
- Autonomous Flight

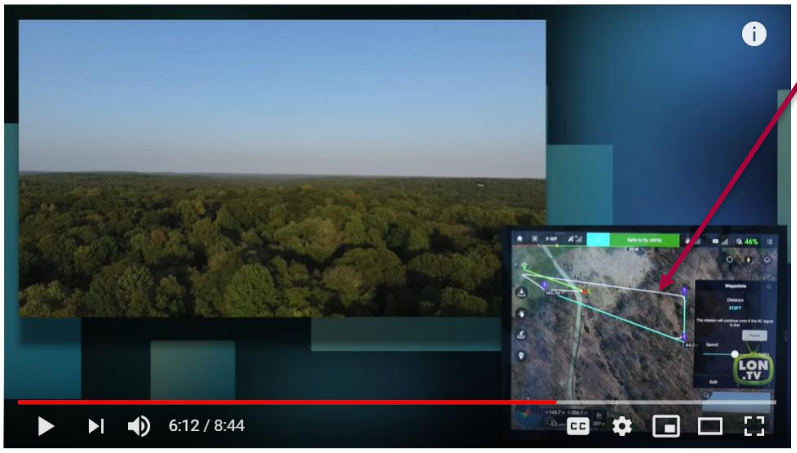
DJI are one of the most respectable drone producers – their Phantom line of quadcopters are well known to be utilized for quality aerial photography.



DJI Phantom 3 Drone - New Autonomous / Intelligent Flight ! - Waypoints , Follow Me , and Circular

Figure 2

Figure 2 showcases the autonomous flight feature of a Phantom 3 Pro quadcopter. The GUI provides the user the real time camera footage recorded by the quadcopter as well as a GPS map in the corner.
The user can then set waypoints for the drone to head towards autonomously.



Here the waypoints are set, and the quadcopter simply travels there and back as planned without the user attending.

Considerations to take during flight:

- There should be a return to base function in case of battery running out
- Waypoints set should be realistic with the current battery and usual speed of the craft.

What can I take form the DJI Phantom series?

The scope of the project is mainly detecting objects with a Pi camera which will ultimately allow the drone to act upon what it detects. A further step to autonomy that I envision with further research is the addition of a GPS module; ultimately setting waypoints with GPS coordinates as above to leave the travelling and avoidance all to the system.

Object Detection

The theory behind Image Classification is usually visualized with CNNs (Convolutional Neural Networks).

A traditional neural network receives a series of inputs, takes the weights of each, multiplies them together.
The sum of the inputs x their weights are multiplied by an activation function.
The processed data is passed through a series of layers called hidden layers which usually leads to an output, effectively classifying the inputs.

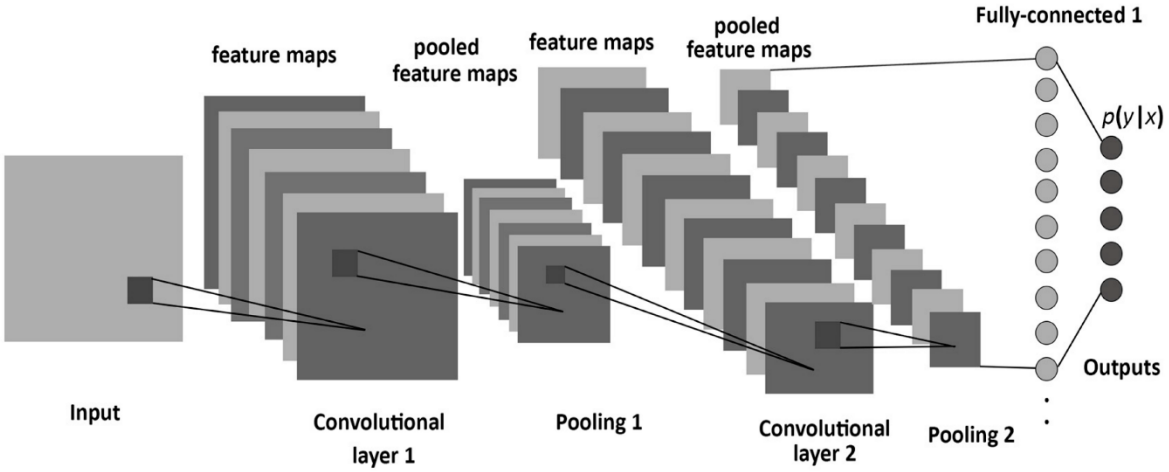


Figure 3

What happens in the hidden layer of an artificial neural network is unknown and simply out of scope just as in biological neural networks.

OpenCV is an open source platform for machine learning applications.
Where OpenCV comes into this picture is that the input, the image, will be passed into a CNN model as a matrix, called a blob, which is calculated as described above.

Example

Figure 4

Here on Figure 4, I have used an API called Nanonets to provide an example as to what I would like to produce.
How this model trains is as follows:

1. Input >= 50 images of an object you would like the model to detect.
2. Label the object with a box manually in each image fed into the model
3. When given a different image containing the object trained to detect, the model will draw a box with a level of confidence/accuracy around that object.

For my model in Figure 4 I trained to detect houses. The model gives a response of the dimensions of a typical house, with knowledge based on the previously labeled inputs, and makes the prediction that in the unseen image the house must have similar dimensions.

More data will consequently increase accuracy.

Hence for my project, the more the quadcopter travels, the more data it collects for training hence the more intelligent it becomes with the experience.

0.4 Discussion/Interview with a potential user

Question	Answer
What problem could I solve with a drone that 'sees'?	A drone, with a bird view camera, can detect and report about objects in an area we send it off on a mission to. This could be applied where humans couldn't be present to see. (e.g. steep mountains, natural disaster sites to name some)
Should I do onboard processing of the object detection program (on the RPi) or stream real time to my laptop (from the drone) where the processing can also occur?	Building OpenCV on the RPi can be challenging and time consuming but you will also need a TCP connection if you decided to stream real-time video otherwise. I recommend on-board processing since streaming and making decisions should be as quick as possible with little to no latency.
Wouldn't it be dangerous to fly a drone autonomously, without intervention?	Yes, you will need to tune the drone manually before flying it with a script. Make sure to read up on flight modes and pick the right one for your testing.

Conclusion of the Research

The scope will be the creation of a quadcopter from scratch and connecting a Raspberry Pi to the flight controller in order to execute telemetry and the real time object detection ability with the help of a Pi camera.

From the discussion I have an idea of what I should achieve and how I should test it. I am planning on building the prototype – flying to autotune it - followed by compiling OpenCV and building my code on the Pi.

Vaguely, the steps/objectives will be as followed:

1. Creation of the quadcopter
2. Telemetry with a Raspberry Pi
3. Object Detection with OpenCV on a Raspberry Pi 3 B+

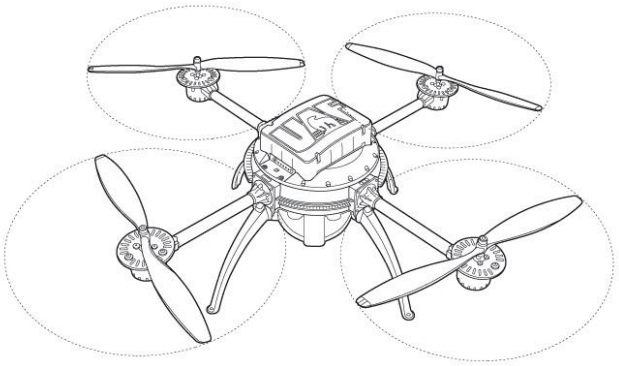
0.5 Objectives

1. **Creation of the quadcopter (DESIGN for prototyping)**
    a. Overview of quadcopter aerodynamics
        i. Derive Formulas for Flight Time Calculations and Average Amp Draw.
    b. Batteries and Power Systems
        i. Calculate maximum current drawn from the battery used.
        ii. Deduce a connection diagram of the PDB to the battery and the ESCs.
    c. Brushless DC Motors and ESCs
        i. Assess what motor rpm would be appropriate for the prototype.
        ii. Learn the mechanism that an ESC supplies to the system
        iii. Calculate amperage of ESC required, compatible with the assessed appropriate motor
    d. Drone Control Systems (flight controllers)
        i. Discuss stabilization hardware purposes to deduce which control system would be appropriate to use for my prototype.
    e. Propeller type and appropriate distribution
        i. Calculate what dimensions would be appropriate for the prototype; Learn dimension specifications of propellers.
    f. Component Placement and Circuit Connections
        i. Produce connection diagrams of the entire system including the motors and CC3D
            1. Document progress on:
                a. Assembly of the frame
                b. Motor Testing (rotation direction)
                c. ESC and PDB connections
                d. Mounting the Flight Controller
                e. Ground Control Software Configuration
                f. Receiver and Transmitter Telemetry
                g. Completed Prototype
2. **Manual Fly Testing (Testing for manual controls)**
    a. Describe all problems that arose during the first flight.
        i. Describe the steps to solving the problems in order to take off the drone.
    b. Record a Full Flight Video where the drone is required to simply take off from ground.
3. **Building on Prototype (Hardware and Software Changes)**
    a. Get a compatible autopilot board to communicate with the RPi (Navio 2).
    b. SSH into the RPi to setup the flight controller.
    c. Enable Ardupilot to run on the board on boot.
    d. Connect the drone (Navio 2) to the GCS (Mission Planner).
    e. Connect the drone to a Python script.
    f. Get DroneKit and find useful functions to return Altitude data to the GCS.
    g. Connect GPS to the drone for different flight modes.
    h. Write the Python script to control the drone.
        i. Connect the script to the drone.
        ii. Safety: Do pre-arm checks before taking off.
        iii. Start producing and processing altitude readings and comparing to the set target altitude. (The data produced during flight are stored in the form of DataFlash Logs
        iv. When the current altitude reaches the target hover then land.

4. **Object Detection with Open CV (DESIGN for steps to basic Autonomy)**
    a. Explanation Of The Program
        i. Explain what the dnn module will be used for.
        ii. Build an environment for the program with all the packages in path.
        iii. Parse arguments through the command line for a user-friendly interface (Argparse)
        iv. Plan and document on the steps to build the detection network architecture.
        v. Pass real time data into the network for detections.
        vi. Pass the blob into the CNN.
        vii. Get the confidence scores of the detections.
        viii. Draw the bounding boxes around the detections with confidence labels.
    b. Write the object detection program and run it on the Raspberry Pi.
5. **Basic Autonomous Flight Testing (TESTING for basic autonomous flight)**
    a. Test the object detection program running on the Raspberry Pi:
        i. Show a certain object in the dataset and record terminal output of what the object seen is.
    b. Test the Drone Kit Python Script flying the drone.
        i. Get a GPS Fix by taking the drone outdoors and arm the motors through the Python Script.
        ii. Get altitude readings from the GPS/GNSS receiver for initial position. Input a target altitude, x, to reach and get continuous readings, y, of altitude, during flight, which should be compared to the target altitude. If y >= x at any point, then hover in current altitude and land shortly after.
    c. Complete a described mission successfully.

Analysis - (Prototype Planning)

1 Component Research



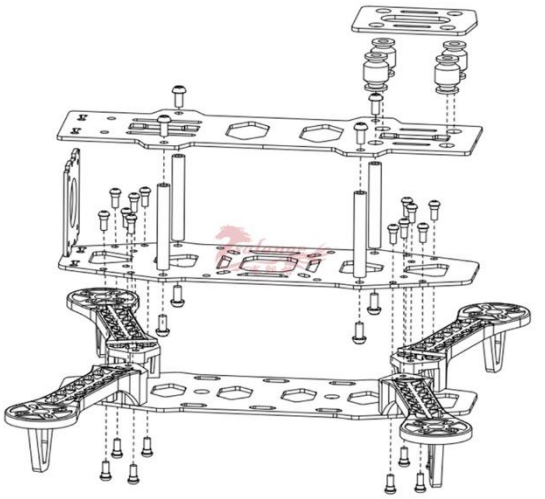
The prototype of the quadcopter will be made just to test for manual controls and get familiar with the flight controller as well as motors.

The prototype will be basic, and I will be using a radio transmitter and receiver initially for the manual control testing section – autonomous flight will consequently be the final section.

Frame used: QAV 250, carbon fiber frame

Carbon fiber is significantly lighter than aluminum, steel, or titanium. This lower density also means carbon frames do a better job of absorbing vibration.

The prototype which look as such:



This is the model of the frame of the quadcopter – QAV 250 is a very common FPV racing drone with a small frame and high kV rated motors.

Limitations:

- Drains battery quickly
- Small space for additions

Advantages:

- Quicker than most drones
- Easy to build and manage
- Inexpensive
- Supports FPV camera setup

1.1 Batteries And Power Systems

Lithium-polymer batteries have the highest power : weight ratio, hence are used often to power UAVs. Li-Po batteries must be taken care of to not be discharged fully or damaged during use. Each cell must be made sure it stays at least 3V (no lower than 3.3V) before being recharged.
Batteries must be chosen taking size of the multirotor and ideal flight time into consideration. Flight Time and batteries were discussed previously for Quadcopter Aerodynamics.

Battery used: 1300Mah 11.1V 85C 3S LiPo
A professional battery balance charger and XT-60 type plugs are required for the build.

The maximum current drawn from the battery is calculated based on the c rating and discharge.

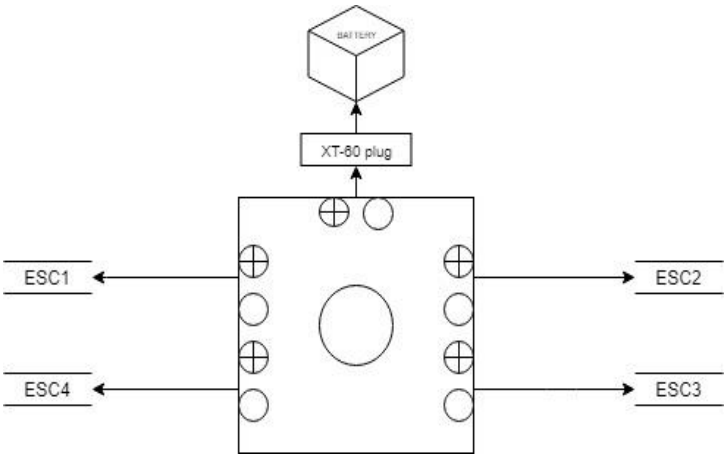
$$I = C * Ah$$
$$85 * 1.3 = 110.5A$$

Knowing the ESCs draw 12A max to all 4 motors. 48A is the margin of current drawn at a time – This is enough for the battery I've chosen.

The current is supplied form the battery to a Power Distribution Board (PDB), which distributes it to all 4 connected ESCs.

Using LiPo batteries are the most advantageous for most multirotor builds as they have higher capacities and discharge rates hence, they hold more power and allow faster power transfer.
However, LiPo batteries d have specific handling instructions for discharge and recharging and have a risk of catching on fire if punctured and exposed to air.

Here is the connection diagram for the PDB:



The plus sign is the live wire and empty is ground.

1.2 Brushless DC Motors and ESCs

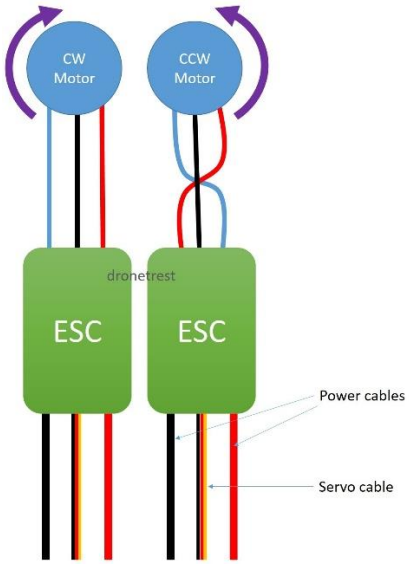
Motor type must be selected depending on the size of the propellers or the drone. Smaller propellers are powered by motors with higher rpm due to less air resistance being experienced overall. A motor of 2300-2400Kv means the motor has approximately 2350 rpm of speed for each volt supplied. This suits my prototype, which is a 250 build – commonly known as an FPV racing drone.

Knowing the Kv rating of a motor will help you determine how fast that motor will rotate when a given voltage is applied to it. Kv allows you to get a handle on the torque that can be expected from a motor.
A high Kv motor has fewer winds of thicker wire that carry more amps at fewer volts and spin a smaller prop at high revolutions.
For a small multirotor you would use smaller props hence high Kv ratings to obtain higher rpm and lower torque.

Motors used: Brushless DC motors 2400Kv (size: 1804, input: 2S-3S)
ESCs Used: 12A brushless with BEC (2S-3S)

2 will motors will spin anti-clockwise and 2 clockwise, depending on how you will connect it to the 3-phase ESC output.

An Electronic Speed Controller (ESC) draws current to drive the motor based on output from the Flight Controller.



The red and black power cables are soldered onto the PDB for max 12A each – hence a total of 48A is drawn out of the battery, just for the motors, maximum.

The servo cable, also known as the BEC (Battery Eliminating Circuit) is connected directly to the Flight Controller from which the current drawn into the ESC and rpm of the motors are controlled with inputs received with a receiver connected to one of its 3 ports.

Direction of the motors are simply switched by switching the outer two cables on the 3-phase output side – usually labeled “motor.”

Amperage of the ESCs vs the motor:

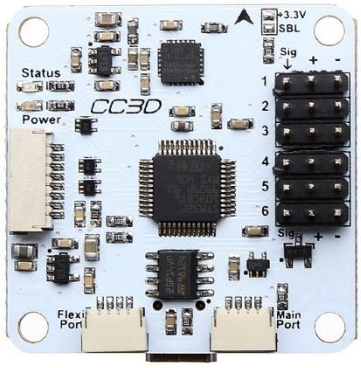
The selection of ESC is supposed to be such that the ESC rating is 1.2 to 1.5 times of the maximum rating of the motor. The 1804 2400Kv motor at full load demands between 6A - 8A current. The ESCs I used are 12A.

$$\frac{ESC\ rating}{Max\ motor\ rating} = \frac{12}{8} = 1.5$$

1.3 Drone Control Systems

Flight controllers are small circuit boards with the main function of providing control for the ESC helping them in directing the RPM of each motor in response to input.

Flight controller used: CC3D



The main function of the Flight Controller is to provide control for the Electronic Speed Controller (ESC) to direct the rpm of motors based on inputs from the transmitter.

The controller is a stabilization hardware with built in gyro and accelerometer sensors, ultimately configured by ground control software, firmware such as LibrePilot.

The CC3D has 6 servo ports, a receiver input port and 2 extension ports (flexi and main).
The servo ports are connected to the ESCs for control of the motors directly by the controller. The extension and receiver ports can be used to telemetry, protocols such as SBUS, IC2 etc. and GPS modules.

1.4 Raspberry Pi Model 3 B+

Raspberry Pi is a micro-computer for data processing with GPIO pins and high processing capability.
The GPIO pins can be used for reading digital logic signals or for outputting digital logic levels.

1.4GHz 64-bit quad-core processor and microSD slot makes this model the most suitable for my object detection plan.

The 16GB microSD card stores the OS of the Pi (Raspbian) and other important data like IP addresses etc.

I will use the Pi for PWM channeling, Telemetry and communication with the CC3D. The further step is an object detection program with RPi camera for real time input.

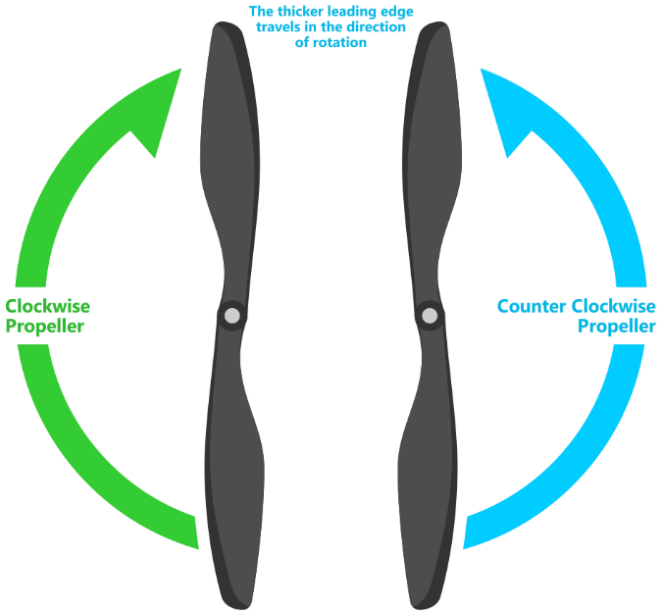
The GPIO pins will be used for PWM channeling and communication with the CC3D for telemetry. The further step is running an object detection program on it with real time Raspberry Pi camera input.

← Pi camera connected to the RPi 3B+.

1.5 Propeller type and appropriate distribution

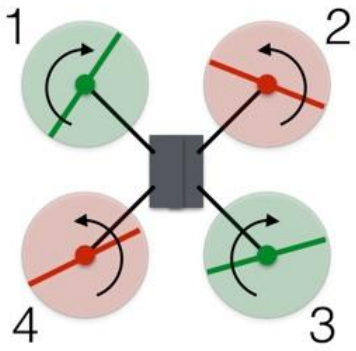
The propellers will be selected based on the size of the drone and motor speed. Mine is a 250mm frame and motor speed is 2400Kv hence the propeller size 5030; diameter and pitch, respectively is perfect. Larger diameter means it can push more air down.
Carbon fiber propellers are of higher quality and stiffer in resisting air. The props must be secured in place tightly to avoid unscrewing on its own during throttle.

Propellers used: 5030 carbon fiber self-locking props.
Propeller direction must be considered during installation:



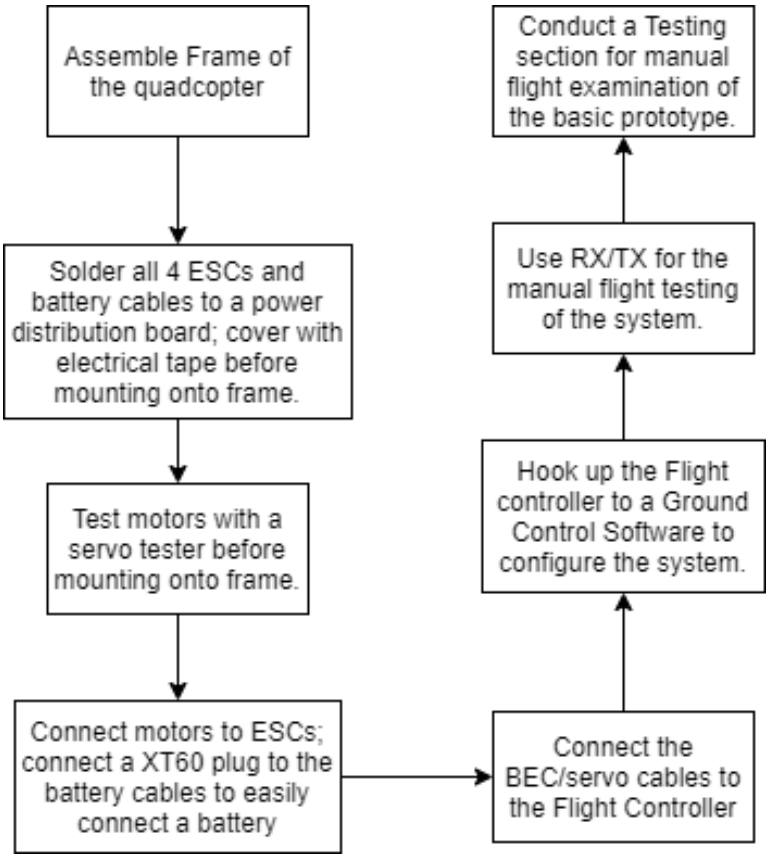
The leading edge must travel in direction of spin to push air down successfully.

The motor direction alternates from CW to CCW across the copter:



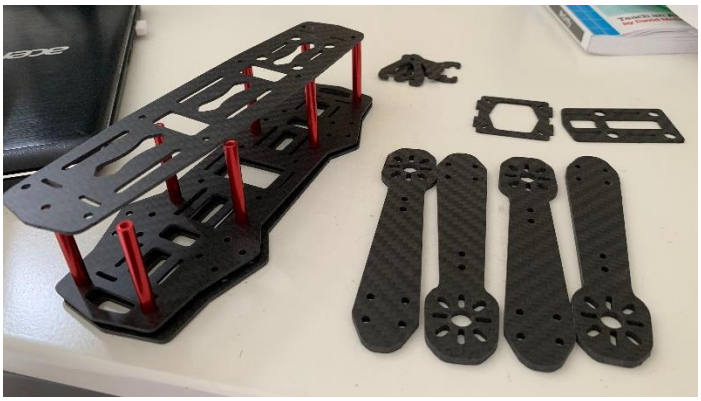
Design (Prototype)

1 System Design



2 Component Placement and Circuit Connections – Prototyping Process

1.6.1 Frame Assembly



These are all carbon fiber parts, as discussed are sturdy, absorb vibrations and are the best appropriate density, material-wise.

The PDB could be placed in between the two bottom plates to save space, insulated with electrical tape. This frame is 250mm from one arm to the other, diagonally. It is a Quadcopter X, also known as a QAV 250.

1.6.2 Motor Testing

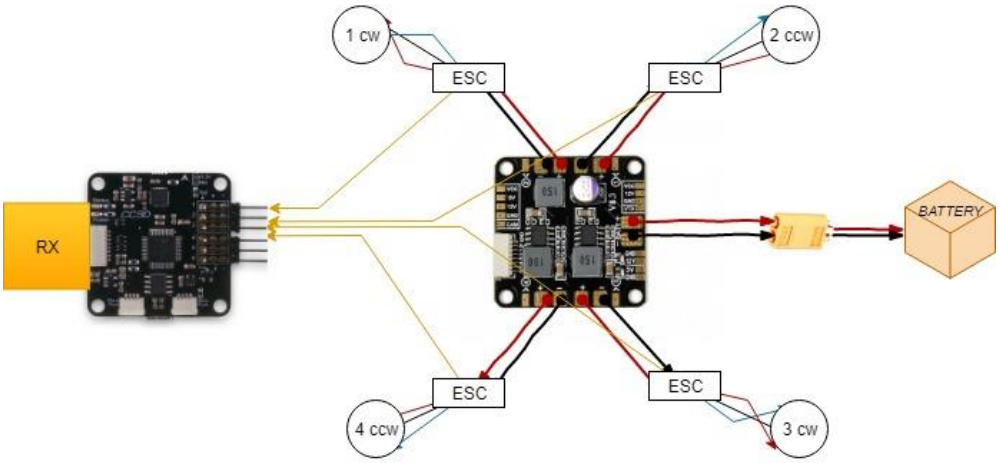
I had to test all 4 motors individually to label the rotation directions on before mounting them onto the frame.

I used one of these servo testers to spin the motors one by one and label CW/CCW accordingly.

The three phased wires that connect the CW motors to the ESCs had to swapped as discussed on section 1.3.

1. The motors are connected to the ESCs, temporarily matching all three wires in parallel.
2. Power the ESC through the PDB; For this you must solder the XT-60 plug to plug in the battery.
3. Connect the servo cable of the ESC to the servo tester output.
4. Observe direction of rotation and label the motor accordingly
5. Swap the outer two of the three phased wires connected to the motor to have a pattern of CW,CCW,CW,CCW (1 to 4)

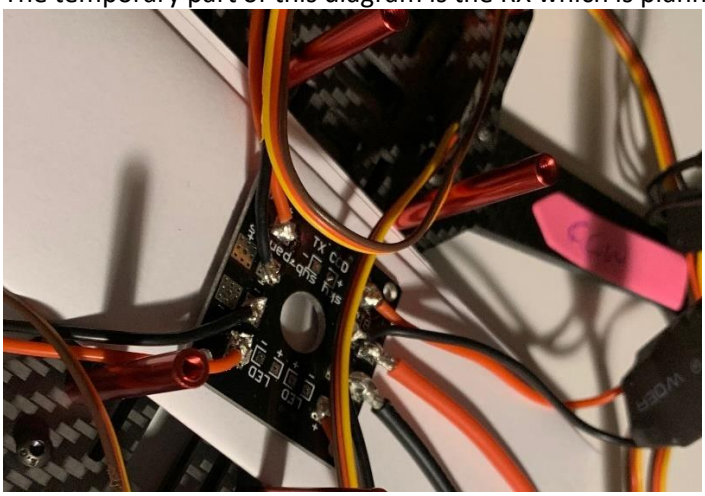
1.6.2 ESC and PDB connection



The final connection looks as the figure above. The ESCs are connected to the servo port of the CC3D and the system is powered with a battery capable of supplying max 110.5A of which 48A is drawn for the motors.

The flight controller is connected to a receiver in which it receives inputs from to direct the entire system.

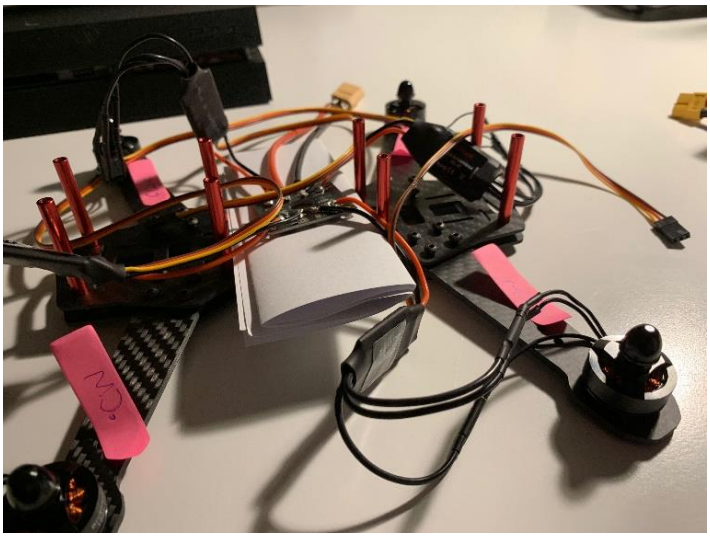
The temporary part of this diagram is the RX which is planned to be replaced with the Raspberry Pi.



I planned to place the PDB in between the two bottom plates that sandwich the arms of the quad.

I taped the entire board with electrical tape in attempt to insulate it before that.

Since the frame is carbon fiber, which conducts electricity, the circuit could theoretically short if the solder touches the frame.

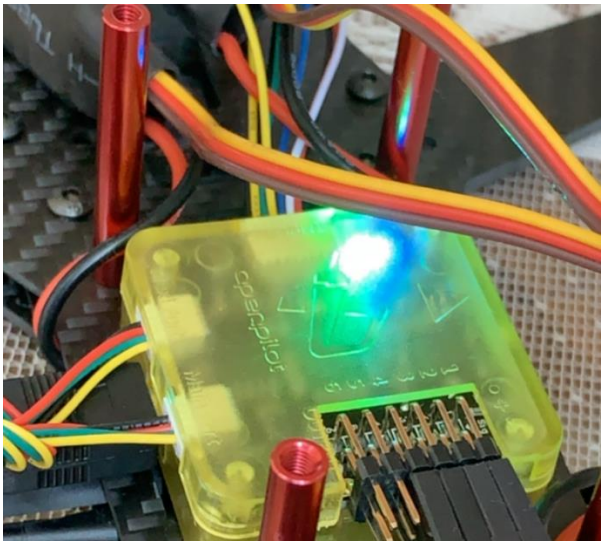


Here, the PDB has the battery power cables and all 4 ESC power cables soldered on.

The motors were all tested hence I mounted them onto the frame at this stage.

Now, I can mount the flight controller on and connect the servo cables of all 4 ESCs.

1.6.3 Mounting the Flight Controller



I placed the flight controller right in the middle of the bottom frame with double sided tape.

The controller, as discussed, is a stabilization hardware with gyro and accelerometer functions built in. This means, when configured on the correct flight mode, it will level the quadcopter during throttle.

The BEC/servo wires are connected to the controller for output to be sent directly to the ESCs which controls motor speeds.

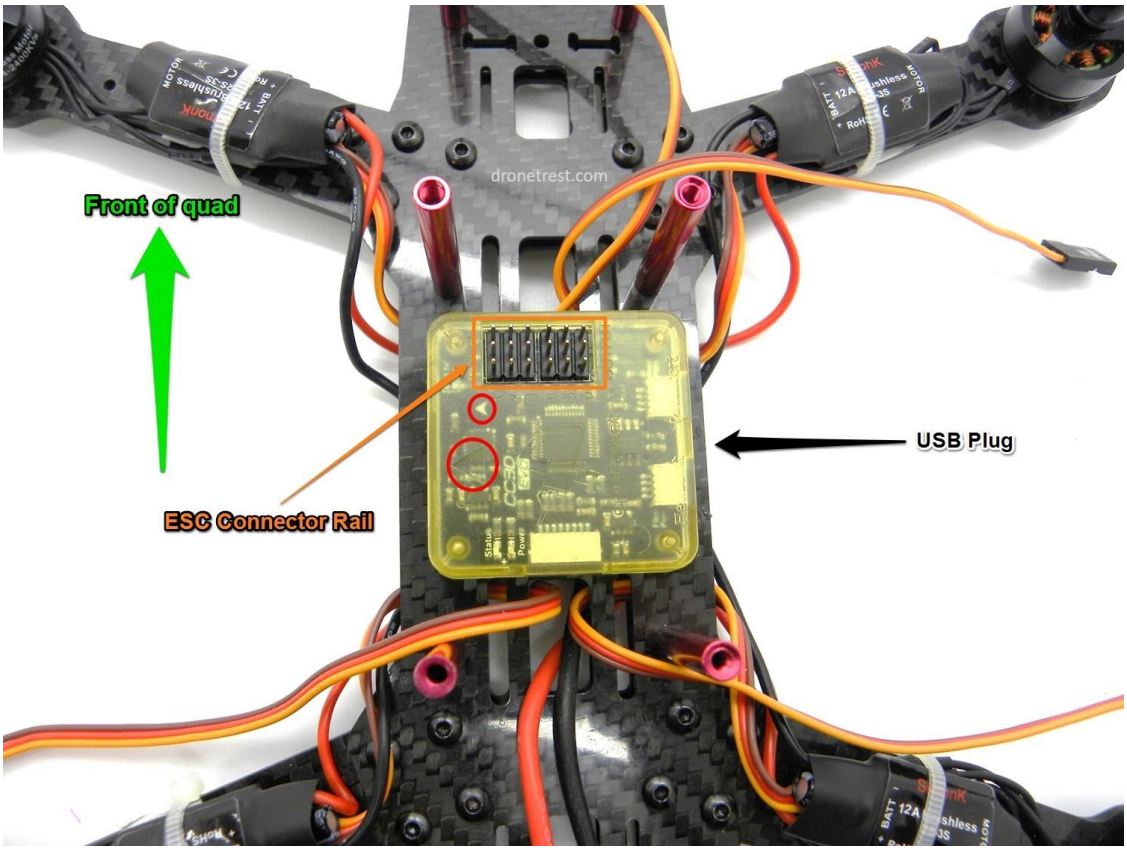
For clarification,

The order in which the BEC is plugged into the flight controller servo port is important.

And the BEC connected to the specific motor on the frame must be labeled clearly 1 to 4 since this must be specified on the ground control software.

The firmware updated on the flight controller must know which motor is which .

Here's a similar image of the final product after mounting;



1.6.4 Ground Control Software Configuration

After mounting the CC3D (flight controller), the next step was to configure everything connected and add the receiver to the system.

Here, I used a Ground Control Software, compatible with the CC3D – LibrePilot. This software is a relatively new one with an improved GUI. Configuration is done with a wizard.

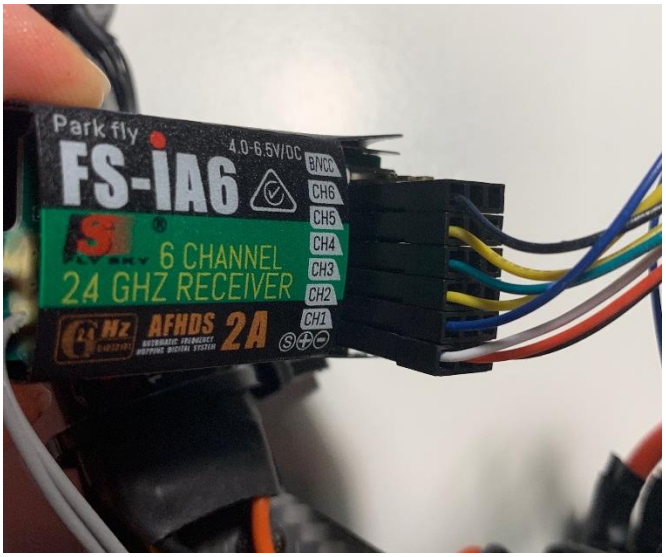
- Connected the CC3D to my laptop to run the vehicle setup wizard.

The wizard calculates the leveling functions of the CC3D mounted onto the quad, the ESCs are calibrated with the connection of the battery, the motors may be test spun and set a minimum rpm manually.

This is all in detail in the TESTING section.

Now I must configure the transmitter and bind it to the receiver to be connected to the CC3D. The prototype is then completed.

1.6.5 Receiver and Transmitter



I used a FS-IA6 receiver for the Manual Flight-Testing section.

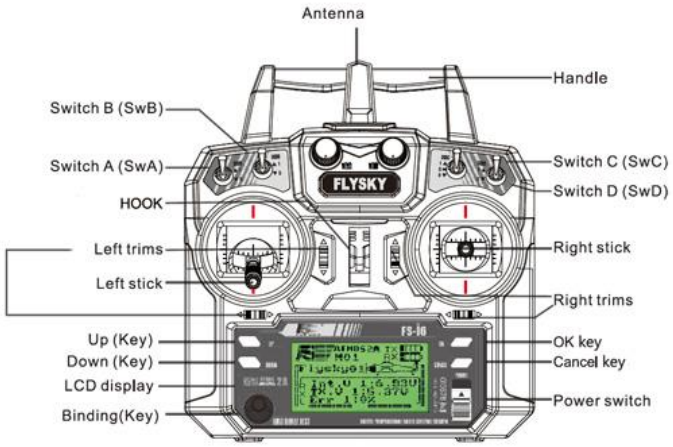
The receiver has 6 channels available. The CC3D supports these many inputs. I plugged in all the signal cables one by one in parallel to the channels as shown; the channels can be configured with a wizard provided by LibrePilot.

Here, after configuration of the transmitter with the wizard, Channel values are shown as commands are sent from the transmitter which the receiver picks up.

Input Channel Configuration

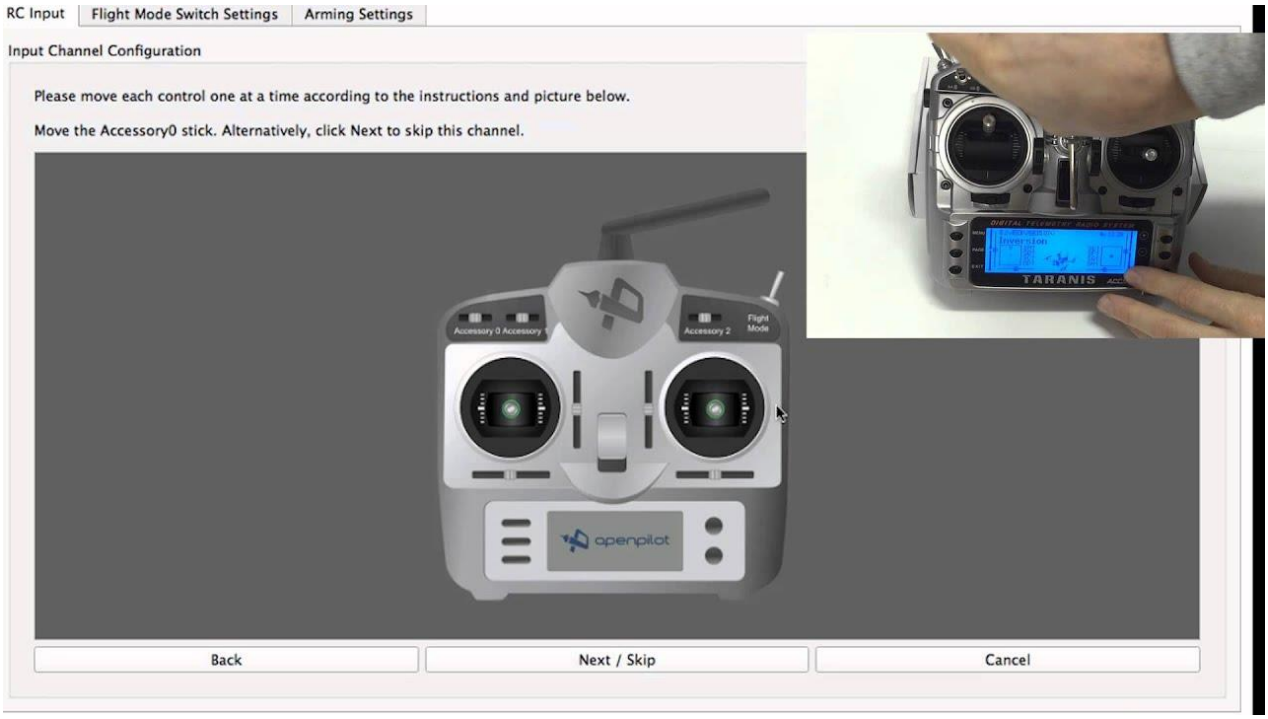
Function	Type	Number	Min	Channel Value	Neutral	Max
Throttle	PWM	Chan 3	1001	1994	1040	1994
Roll	PWM	Chan 1	1001	1993	1492	1993
Pitch	PWM	Chan 2	1973	1973	1499	999
Yaw	PWM	Chan 4	1000	1997	1500	1997
FlightMode	PWM	Chan 5	999	1999	1499	1999

I have noticed different ways of channeling. These were mainly PWM, PPM one shot and other protocols such as Spektrum Satellite.
I used PWM channeling, which corresponds to every signal wire being one channel. I will discuss PWM (Pulse Width Modulation) in more depth in the Telemetry with Raspberry Pi section.
E.g. wire 3 was channel 3 which corresponds to Throttle.

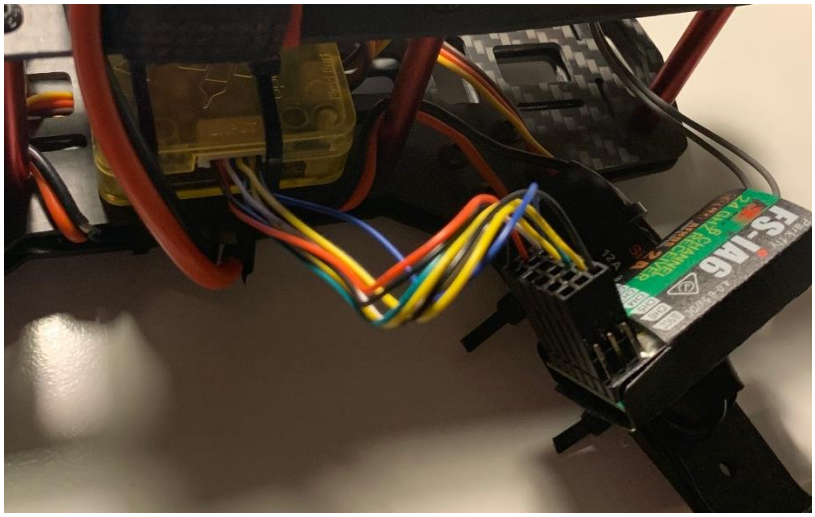


The transmitter used must be compatible with the receiver. I used an FS-I6, which was compatible.

The transmitter has different modes to be used in – most common one is Acro 2. This is when throttle and pitch is on the left while yaw and roll is on the right-hand side of the gimbals.



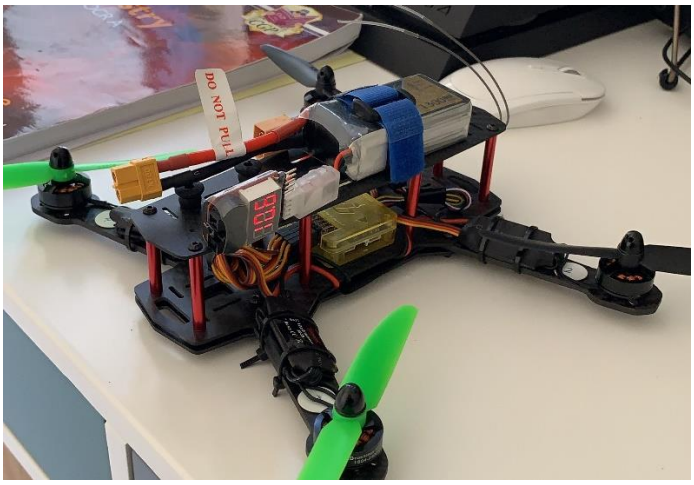
As shown above, LibrePilot configures the input channel – sending commands to the receiver which inputs them into the CC3D. The channel values and corresponding movements are all uploaded on the flight controller.



This is how the connection looks. This is a temporary solution to test manual controls of the quadcopter and make sure the CC3D levels the system safely during throttle.

1.6.6 Completed Prototype

After configuration and some cleaning up of component placements, the prototype is complete.


DO NOT PUL

Testing (Manual Flight Controls of the Prototype)

2 Manual Fly Testing

2.1 First Flight Problems

At first flight two problems, intertwined with each other, arose.

2.6.2 Problem #1

As the quadcopter arms the propellers will unscrew themselves and come off during rotation. Not only is this very unsafe it also completely blows off the flight.

Here's how it looks: https://www.youtube.com/watch?v=UYAjVwR9AjI&feature=youtu.be

Solution

I immediately ordered what I found as "self-locking props." These lock into place and have adaptors to be mounted onto the motor screws.

However, also with this problem came a much larger one.

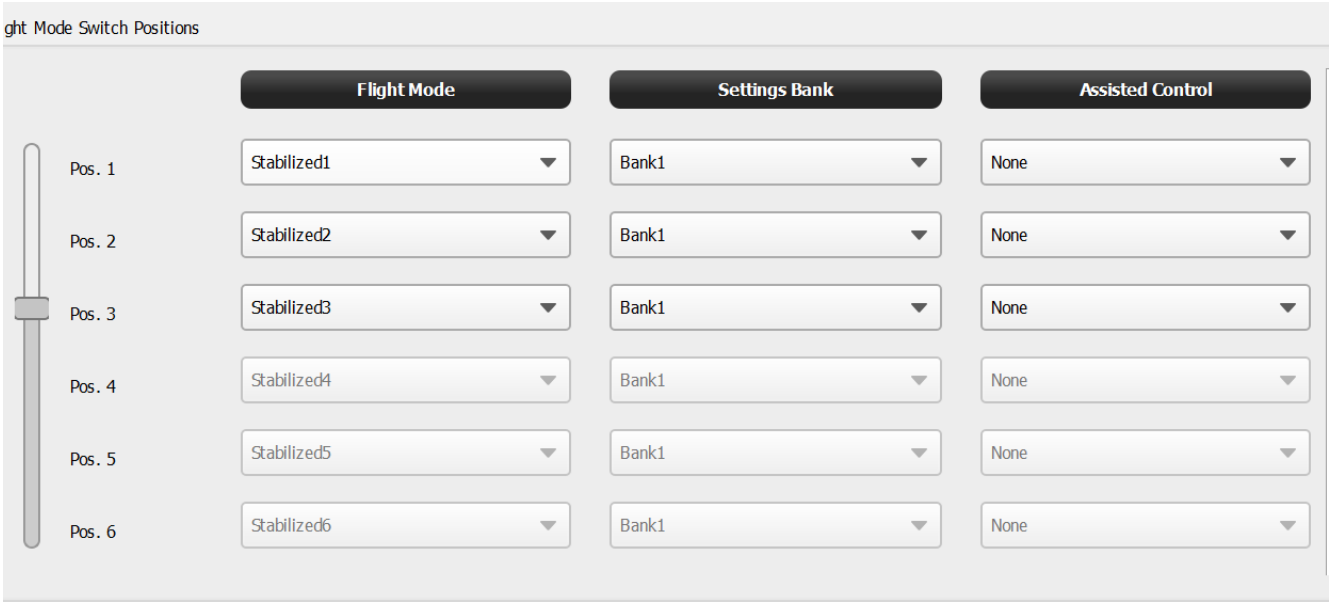
2.6.3 Problem #2

As soon as the quadcopter receives the throttle command it does an instant flip. Along with Problem #1 this put the project on a temporary hold.

Here's how that looks: https://www.youtube.com/watch?v=8NIWt7a0w9g&feature=youtu.be

Attempt to Solve #1

I figured it could be a leveling/attitude problem (Flight Modes) hence I checked the configuration tabs and manually altered some changes to see if anything fixes.



I reduced Flight modes down to 3 to reduce complications of control.

	Roll	Pitch	Yaw	Thrust
Stabilized 1	Attitude	Attitude	Acro+	CruiseControl
Stabilized 2	Attitude	Attitude	AxisLock	Manual
Stabilized 3	Attitude	Attitude	AxisLock	Manual
Stabilized 4	Attitude	Attitude	Acro+	CruiseControl
Stabilized 5	Attitude	Attitude	Rate	CruiseControl
Stabilized 6	Attitude	Attitude	AxisLock	CruiseControl

Some research into Flight Modes:

Attitude: if you let go of manual control and return the stick to a neutral position the multirotor would level and drift to a halt in hover.

Acro+: this mode is based on rate and it loops the featured stick based on gyro suppression.

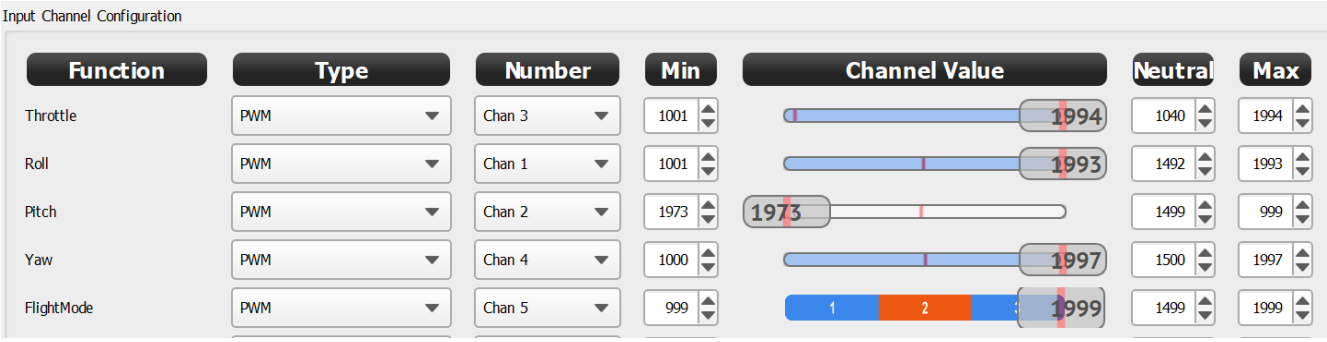
Rate: If you let go and return to a neutral position the multirotor keeps going in the same direction hence the pilot must return the craft to level. This is much more manual.

Other modes aren't very important to know in this case since I know that I must be on Attitude mode for Roll and Pitch. I set all 3 flight modes to Attitude.

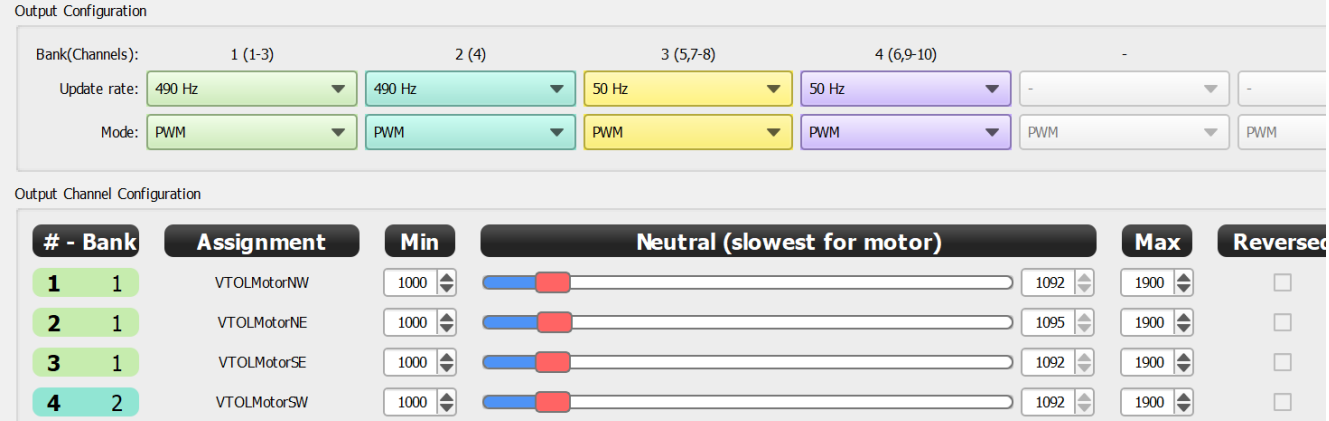
None of this fixed the problem.

Attempt to solve #2

I reconfigured the entire quadcopter system on LibrePilot.

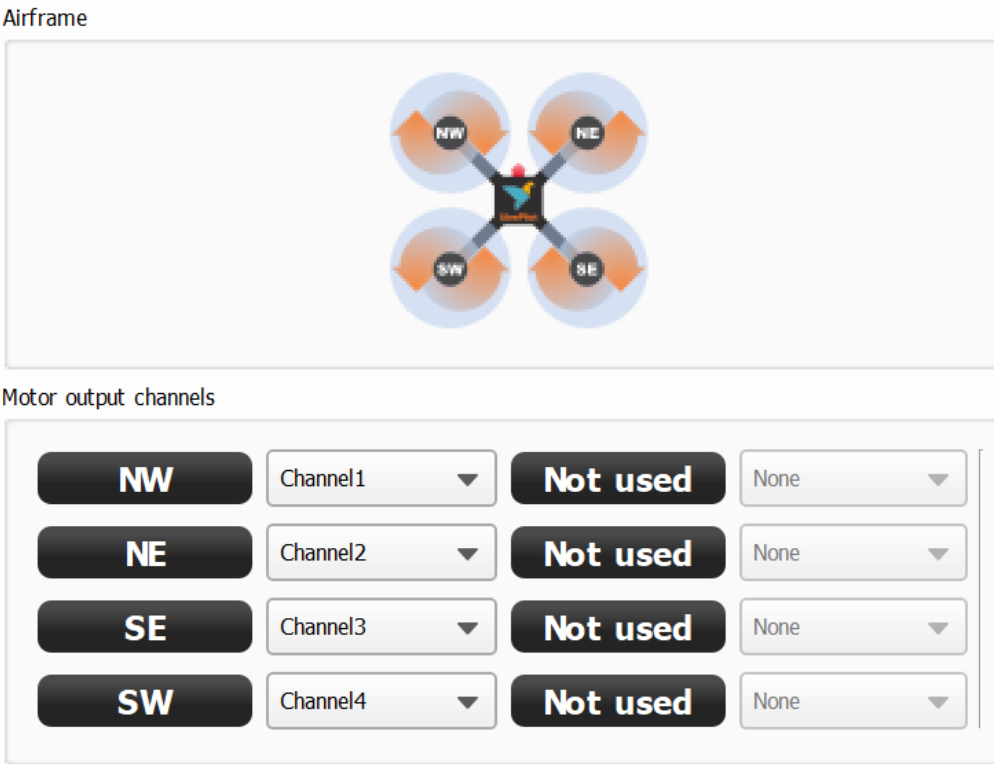


Here I reconfigured the transmitter and made sure to check for any reverse channels which in fact mirrors the command outputted. This as a result would do the opposite of what is being commanded.

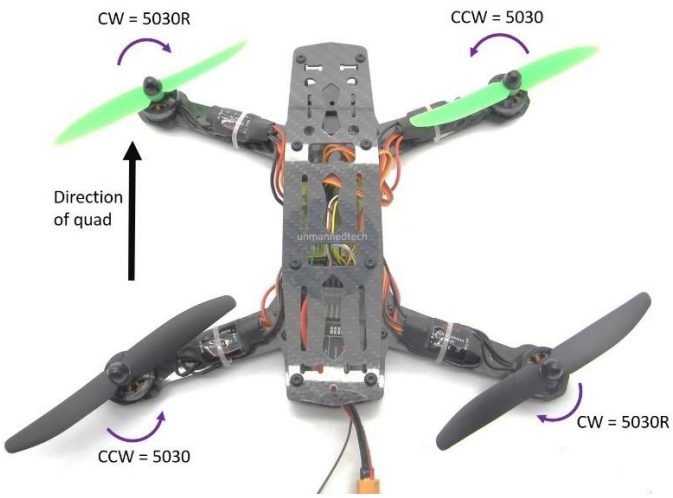


I manually entered new maximum rpm for each motor and tested if they spin quick enough to resist air. I then figured if it could flip it was surely resisting air just fine.

I decided to check each motor again to make sure everything was spinning the correct way hence aerodynamics wasn't the issue.



LibrePilot already showed me what direction each motor should be rotating. I also made sure to assign the right channel to each one in the servo ports of the CC3D.



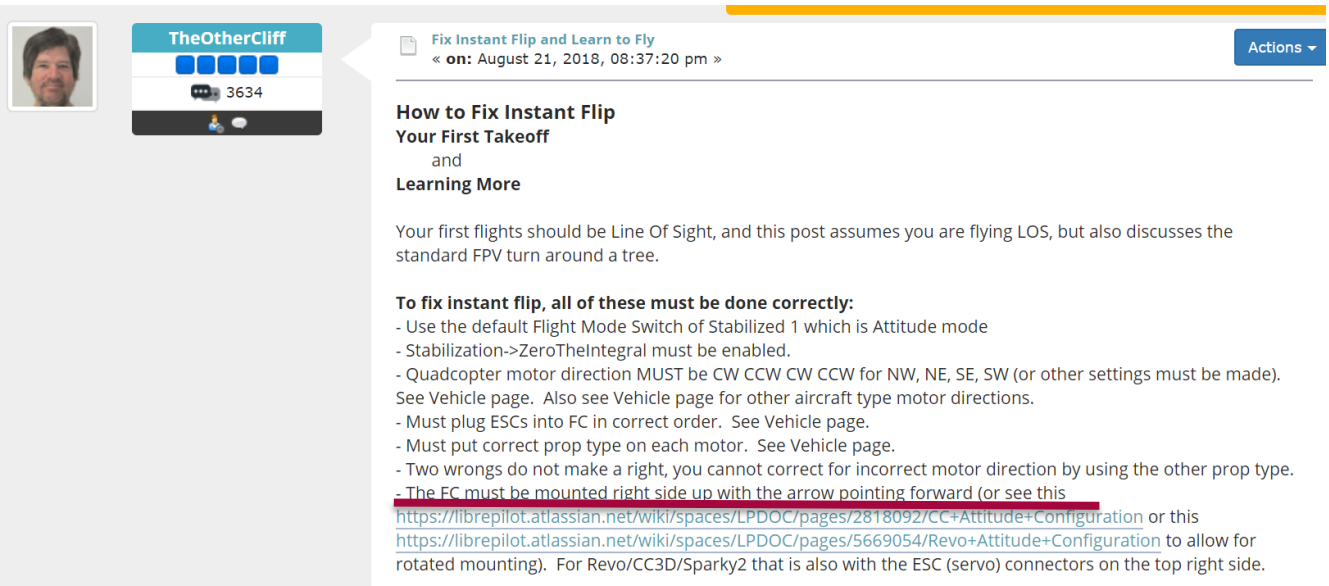
Here's how the props should be installed specific to the motor rotation direction.

I took all this into consideration.

Problem #2 still existed.

Solution

I looked for some help on the LibrePilot forum.



TheOtherCliff

3634

Fix Instant Flip and Learn to Fly
« **on:** August 21, 2018, 08:37:20 pm »

Actions

How to Fix Instant Flip
Your First Takeoff
and
Learning More

Your first flights should be Line Of Sight, and this post assumes you are flying LOS, but also discusses the standard FPV turn around a tree.

To fix instant flip, all of these must be done correctly:
- Use the default Flight Mode Switch of Stabilized 1 which is Attitude mode
- Stabilization->ZeroTheIntegral must be enabled.
- Quadcopter motor direction MUST be CW CCW CW CCW for NW, NE, SE, SW (or other settings must be made). See Vehicle page. Also see Vehicle page for other aircraft type motor directions.
- Must plug ESCs into FC in correct order. See Vehicle page.
- Must put correct prop type on each motor. See Vehicle page.
- Two wrongs do not make a right, you cannot correct for incorrect motor direction by using the other prop type.
- The FC must be mounted right side up with the arrow pointing forward (or see this https://librepilot.atlassian.net/wiki/spaces/LPDOC/pages/2818092/CC+Attitude+Configuration or this https://librepilot.atlassian.net/wiki/spaces/LPDOC/pages/5669054/Revo+Attitude+Configuration to allow for rotated mounting). For Revo/CC3D/Sparky2 that is also with the ESC (servo) connectors on the top right side.

Having done everything said but the last bullet point. I spotted that the Flight Controller just wasn't mounted the right side up (the arrow pointing the nose of the craft).

I took everything out and remounted to give this a go. The flight went smooth as a result.

Here's the solution flight: https://www.youtube.com/watch?v=AgnUQG0w1WU&feature=youtu.be

2.2 Further Testing

Flight outside to tune the drone:

Flight #1: https://www.youtube.com/watch?v=UJHHGZ_vGXM

Flight#2: https://www.youtube.com/watch?v=I35LA5V_pXU

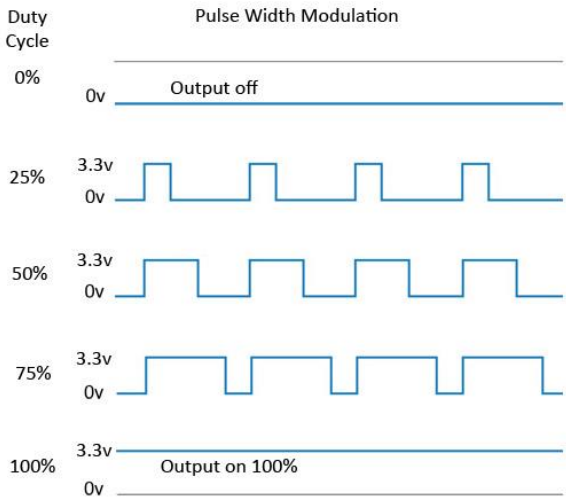
Implementation (Steps To Autonomy)

3 Telemetry with Raspberry Pi

3.1PWM Theory

Pulse Width Modulations (PWM) is a method of getting analog signals from digital means. This creates discrete chopped up parts of the electrical signal which reduces the average power delivered. This method is commonly used to control DC motors.
Two main components of PWM defines its behavior: *a duty cycle and a frequency.*

Duty Cycles: amount of time that the signal is on (a high state) as a percentage overall the time it takes to complete one cycle.
Frequency: How fast the PWM completes one cycle.



Digital output is either on or off; it cannot vary between on and off as analog output could. If an LED or motor is connected to a normal digital output, it will only operate at full on or full off states.

With PWM varying levels of output to an electrical device is simulated. At any given time, the digital output will still be on or off, but the output is varied with widths which controls effective output.

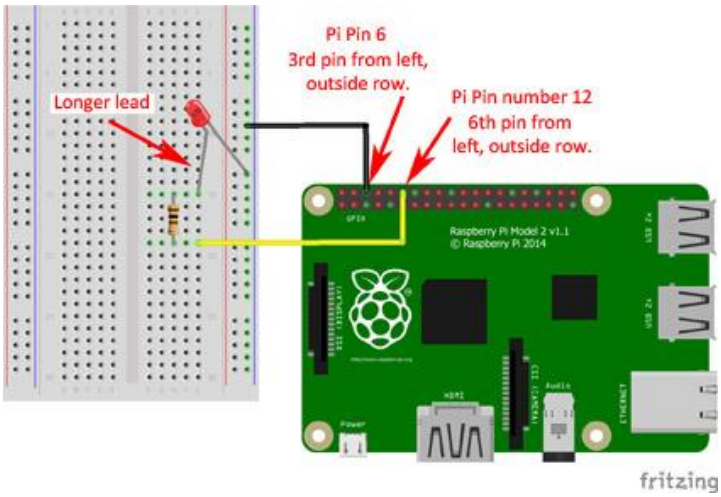
e.g. if the on cycle is shorter than the off cycle, the overall duty cycle is lowered; vice versa.
Thus, if the duty cycle is 25%, this means 25% on and 75% off cycles. Duty cycles are varied between 0 – 100%

Therefore, frequency determines how fast the PWM switches between high and low states.
Fast enough rates of alternating between on and off states, with a certain duty cycle, the output will behave as an electrical signal (voltage analog signal) as it provides power to the motors.

3.2PWM output from GPIO Pins on the RPi; LED circuit to indicate low and high states.

Since an LED is either on or off with any digital output and cannot be varied like analog outputs. I can program the GPIO pins on the RPi to produce PWM and check it works with an LED as the duty cycles vary from 0 to a 100%

Connection:



I used the GPIO18 pin (pin 12) and a ground pin (pin 6) from the Pi.

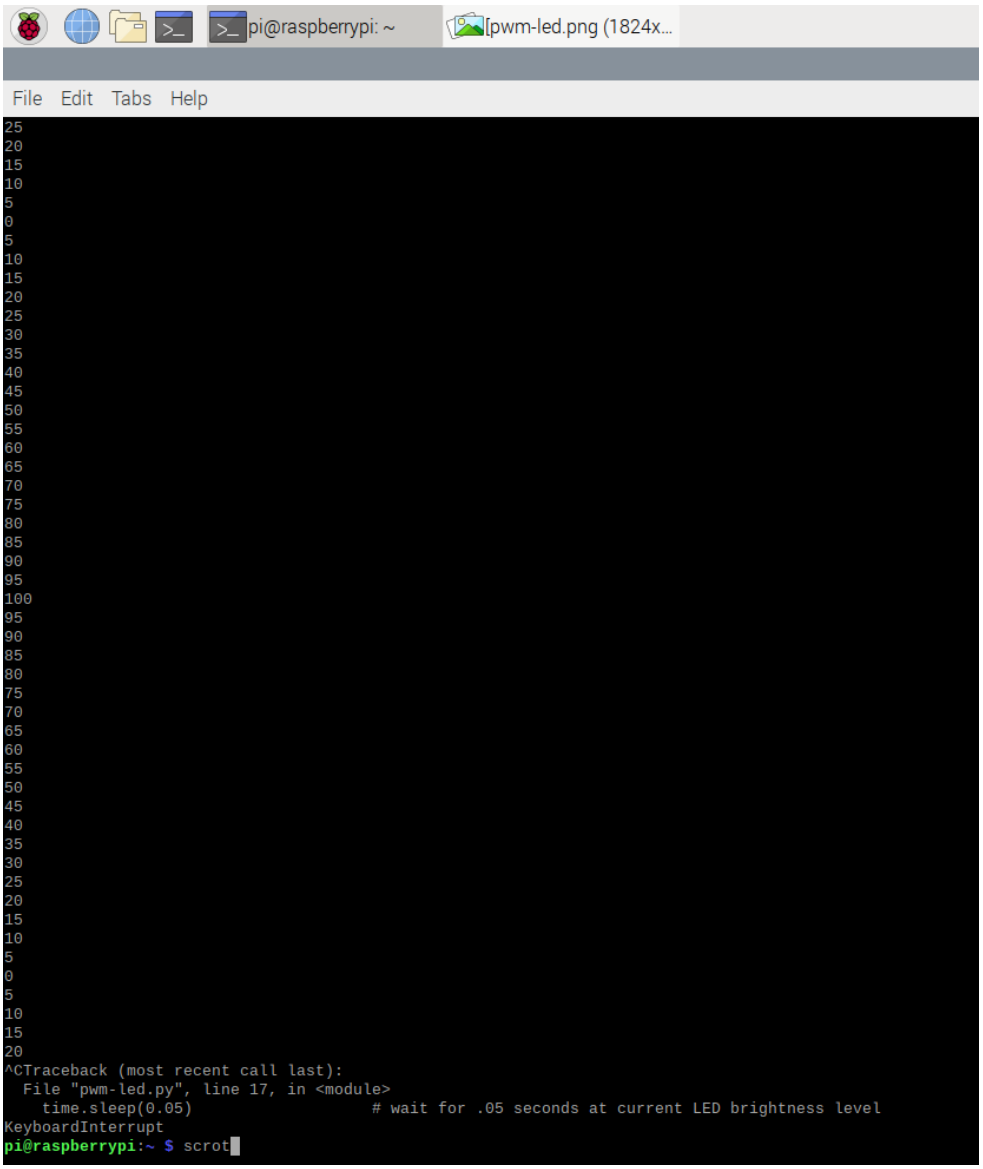
The anode connects to a 220Ω resistor and the cathode connects to the ground while the other end of the resistor will connect to the GPIO18 pin on the Pi (pin 12).
Lastly connect any pinon ground of the breadboard to the Pi pin 6 (GND).

This completes the circuit.

Code:

```
import RPi.GPIO as IO
import time

IO.setmode(IO.BOARD)     # set to pin numbering by choosing board
IO.setup(12, IO.OUT)     # set GPIO 12 as output
pwm = IO.PWM(12, 100)    # set PWM on pin 12 to 100Hz frequency

duty_cycle = 0                      # set duty cycle to 0%
pwm.start(duty_cycle)               # start PWM with duty cycle 0%

try:
  while True:                       # this loops the program until
KeyboardInterrupt (Ctrl + C)
    for duty_cycle in range(0, 101, 5):    # loop from 0 to a 100 with step 5
      pwm.ChangeDutyCycle(duty_cycle)
      time.sleep(0.05)              # this sleeps 0.05s at the current duty_cycle,
provides specific LED brightness to indicate
      print(duty_cycle)
    for duty_cycle in range(95, 0, -5):    # loops 95 to 5, step -5, hence is just
a loop down to decrease duty cycle, LED brightness decreases here
      pwm.ChangeDutyCycle(duty_cycle)
      time.sleep(0.05)
      print(duty_cycle)
except KeyboardInterrupt:
pwm.stop()
IO.cleanup()                        # cleans up and resets GPIO ports
```

Screenshots of Pi:

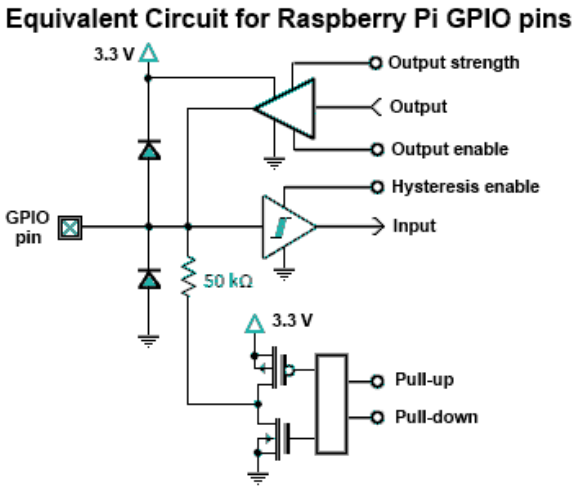


The screenshot shows the loop with steps of 5 as the duty cycle increases and decreases – in result varying the brightness of the LED in the circuit.

The outcome:
https://www.youtube.com/watch?v=Mug2mcwXhlY&feature=youtu.be

This demonstration is a look at how the motors spin with the use of digital signal modulation (PWM to be exact).

The problem facing this project with communication between the Raspberry Pi and the CC3D stability hardware is that there is a voltage level difference between the GPIO pins of the RPi and the CC3D.



The Raspberry Pi UARTs

The SoCs used on the Raspberry Pis have two built-in UARTs, a PL011 and a mini UART. They are implemented using different hardware blocks, so they have slightly different characteristics. However, both are 3.3V devices, which means extra care must be taken when connecting up to an RS232 or other system that utilises different voltage levels. An adapter must be used to convert the voltage levels between the two protocols. Alternatively, 3.3V USB UART adapters can be purchased for very low prices.

Figure 5- RPi UART info (https://www.raspberrypi.org/documentation/configuration/uart.md)

The output drives from 0 to 3.3V. Several of these pins are wired up in parallel and there is simply a difference in voltage levels at which both the ports operate. The CC3D works at 3.3V and tolerates 5V while the GPIO pins work at 3.3V with no tolerance to 5V.

In theory, 5V based logic would work. Hence, to keep this project going as is I must build a voltage level converter out of transistors or DC-DC converter with 5V output. This converter will allow me to communicate the Raspberry Pi with the CC3D and eliminate the problem of difference in voltage levels.

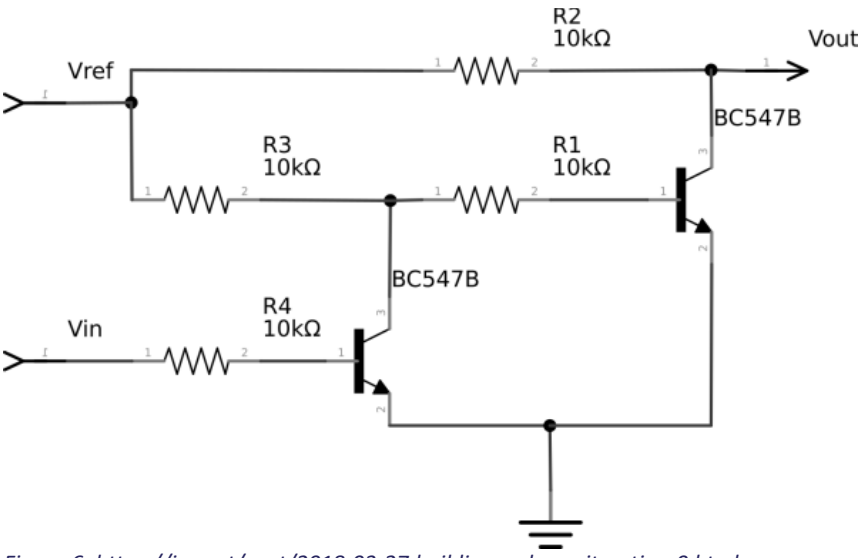


Figure 6- https://jany.st/post/2018-03-27-building-a-drone-iteration-0.html

Here's a circuit of what needs to be done.

4 Voltage Dividers

Voltage dividers are circuits specially built to turn a large voltage into a smaller one.
Resistors could be wired in series to input the voltage into and hence output a fraction of the voltage inputted.

The circuit:
Simply the idea is to apply an input voltage into a series of resistors to cut it down; the resistors will all share the same current, but the voltage will divide among the two, depending on their resistance value.
As all linear resistors follow Ohm's Law, I could deduce the equation.

4.1 Deriving the Voltage Division Equation

The equation:

Knowing Ohm's Law is $V = IR$
And that, $Vin = V1 + V2$; where *Vin* is the voltage input.
We can say that; $Vin = IR2 + IR2$
So, we can rearrange into: $I(R1 + R2) = Vin$
Then,

$$\frac{Vin}{R1 + R2} = I$$

We also know that, $Vout = IR2$, since the voltage out is the second one; V2.
And now we substitute that back into the previous equation by rearranging to make *I* the subject:

$$I = \frac{Vout}{R2}$$

$$\frac{Vin}{R1 + R2} = \frac{Vout}{R2}$$

Final formula, for Vout is:

$$V_{out} = \frac{V_{in} \cdot R2}{R1 + R2}$$

Therefore, Vout is directly proportional to Vin and the ratio of R1 and R2. There will be a resistor closer to the input (Vin) voltage and one closer to ground (Vout). Vout is the divided voltage.

Due to the difference in voltage levels between the operating ports, I will have to use a bidirectional voltage leveler for the serial communication between the IO ports.

If Vin is 5V and Vout is 3.3V with R2 being a 2K Ohms we can work out what R1 should be.

$$3.3 = \frac{5 \cdot 2000}{R1 + 2000}$$

Therefore, 3.3R1 = 10000 – 6600
So, R1 = 1030.30 (recurring) Ohms.
In conclusion R1 could be approximately a 1K ohms resistor for a Vout of 3.3V from 5V.

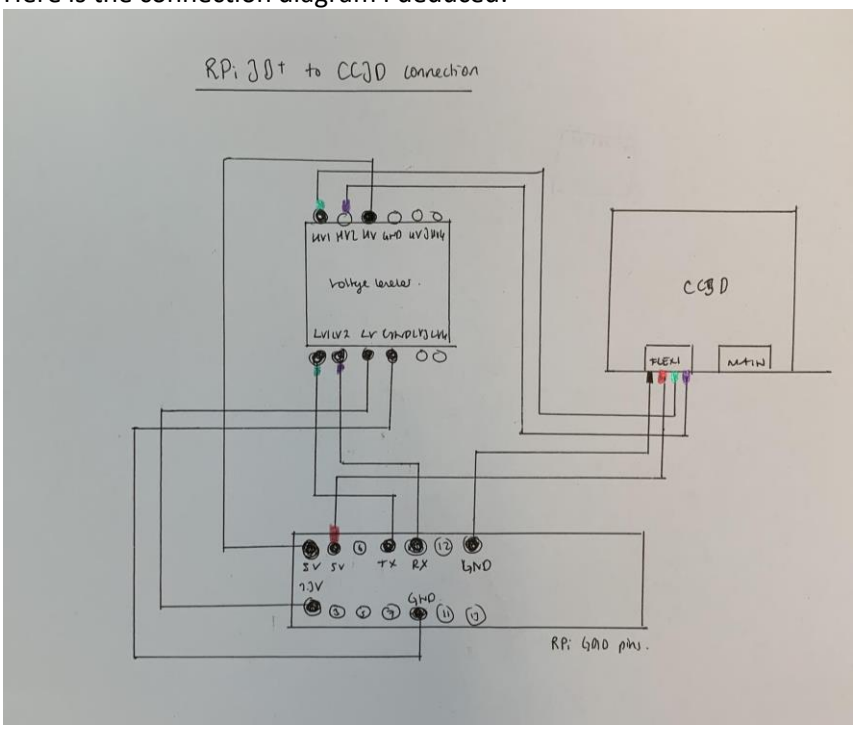
5 Communication of the Flight Controller with the Raspberry Pi

5.1 Bidirectional Voltage Leveler + Connection of the FC with the RPi

I need to now communicate the two main boards of this system; the flight controller which stabilizes the drone and spins the motors through its servo port and the RPi which is what effectively makes the drone autonomous, giving it object detection and obstacle avoidance abilities.

Since there is a voltage difference between the GPIO pins of the RPi and the input port of the CC3D I need to use a bidirectional voltage leveler to make the boards compatible for back and forth communication. This is neater and easier to work with than a bunch of parallel resistors on a prototype board stuck to the roof of the drone. Essentially, the step down/up is done as I discovered in the above sections.

Here is the connection diagram I deduced:



The RPi provides the LV (low voltage) at 3.3V while the CC3D provides the HV at 5V.

The leveler has 4 channels of which I used two to connect the UART (rx/tx) pins of the RPi to the corresponding rx/tx channels of the flexi input port that the CC3D has. This is the main connection from which I can achieve serial communication between the boards.

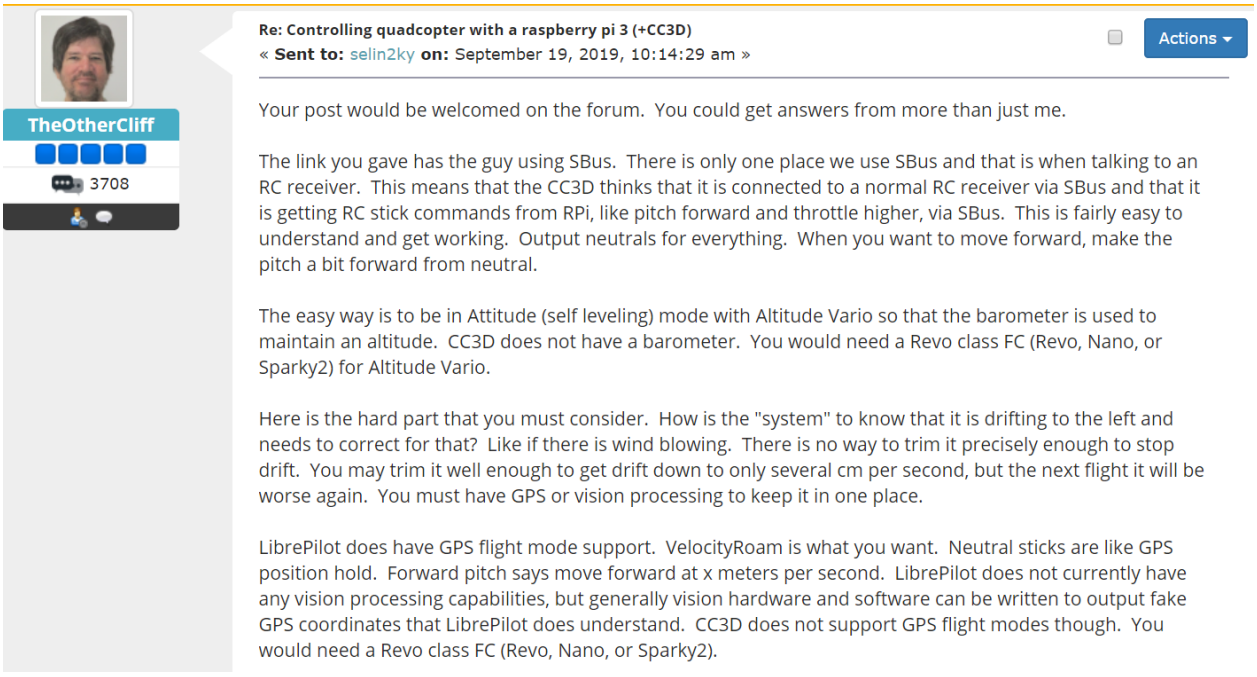
This solution overall, has defeated the voltage difference problem of the boards with low costs.

Link to the connection: https://www.youtube.com/watch?v=__dwM0xgLmM

Although the boards are now directly connected and they power up, **they need to also communicate and spin the motors to have an ultimate success.**
I'm planning on asking the LibrePilot forum for some guidance.

5.2 Communicating the RPi serially with S.BUS

The use of PWM previously brought success to manual control of the drone. However, now that I'm diving into autonomy, connecting a companion computer – the companion computer running the object detection network must be able to give feedback to the system during flight. In simpler terms, the RPi, must be able to tell the motors to act upon the situation encountered. With some feedback from the forum I can keep the project going:



Re: Controlling quadcopter with a raspberry pi 3 (+CC3D)
« **Sent to:** selin2ky **on:** September 19, 2019, 10:14:29 am »

Actions

TheOtherCliff

3708

Your post would be welcomed on the forum. You could get answers from more than just me.

The link you gave has the guy using SBus. There is only one place we use SBus and that is when talking to an RC receiver. This means that the CC3D thinks that it is connected to a normal RC receiver via SBus and that it is getting RC stick commands from RPi, like pitch forward and throttle higher, via SBus. This is fairly easy to understand and get working. Output neutrals for everything. When you want to move forward, make the pitch a bit forward from neutral.

The easy way is to be in Attitude (self leveling) mode with Altitude Vario so that the barometer is used to maintain an altitude. CC3D does not have a barometer. You would need a Revo class FC (Revo, Nano, or Sparky2) for Altitude Vario.

Here is the hard part that you must consider. How is the "system" to know that it is drifting to the left and needs to correct for that? Like if there is wind blowing. There is no way to trim it precisely enough to stop drift. You may trim it well enough to get drift down to only several cm per second, but the next flight it will be worse again. You must have GPS or vision processing to keep it in one place.

LibrePilot does have GPS flight mode support. VelocityRoam is what you want. Neutral sticks are like GPS position hold. Forward pitch says move forward at x meters per second. LibrePilot does not currently have any vision processing capabilities, but generally vision hardware and software can be written to output fake GPS coordinates that LibrePilot does understand. CC3D does not support GPS flight modes though. You would need a Revo class FC (Revo, Nano, or Sparky2).

Figure 7 - LibrePilot Forum help to communicate the RPi with the CC3D via S.BUS

The S.BUS Protocol is a serial protocol that was developed by Futaba for hobby remote control applications. It is derived from the RS232 protocol, but the voltage levels are inverted. The protocol provides 16 channels of 11 bits each, two digital channels, and two flags for "frame lost" and "failsafe".

The quirk of S.BUS is that The voltage levels are inverted. So, while a 0 with a normal serial port is encoded with a low voltage, it is encoded with a high voltage with SBUS. This protocol is just transmitting some data over UART.

The SBUS bridge uses one serial port to receive messages from a receiver on the RX line and send commands to a flight controller on the TX line. It can be commanded by Control Command ROS messages for autonomous quadrotor flight. The CC3D and the Raspberry Pi can talk telemetry over such simulated serial port using the UAVTalk protocol. The LibrePilot source code provides python bindings for that. Additionally, it continuously checks the incoming messages from the RC transmitter and checks whether an arm switch is on. If the arm switch is on, it directly forwards the received RC commands to the flight controller ignoring incoming commands for autonomous flight. This enables flying a quadrotor purely manually or taking over by a remote control at any time during autonomous flight for safety.

Here is the supposed connection:

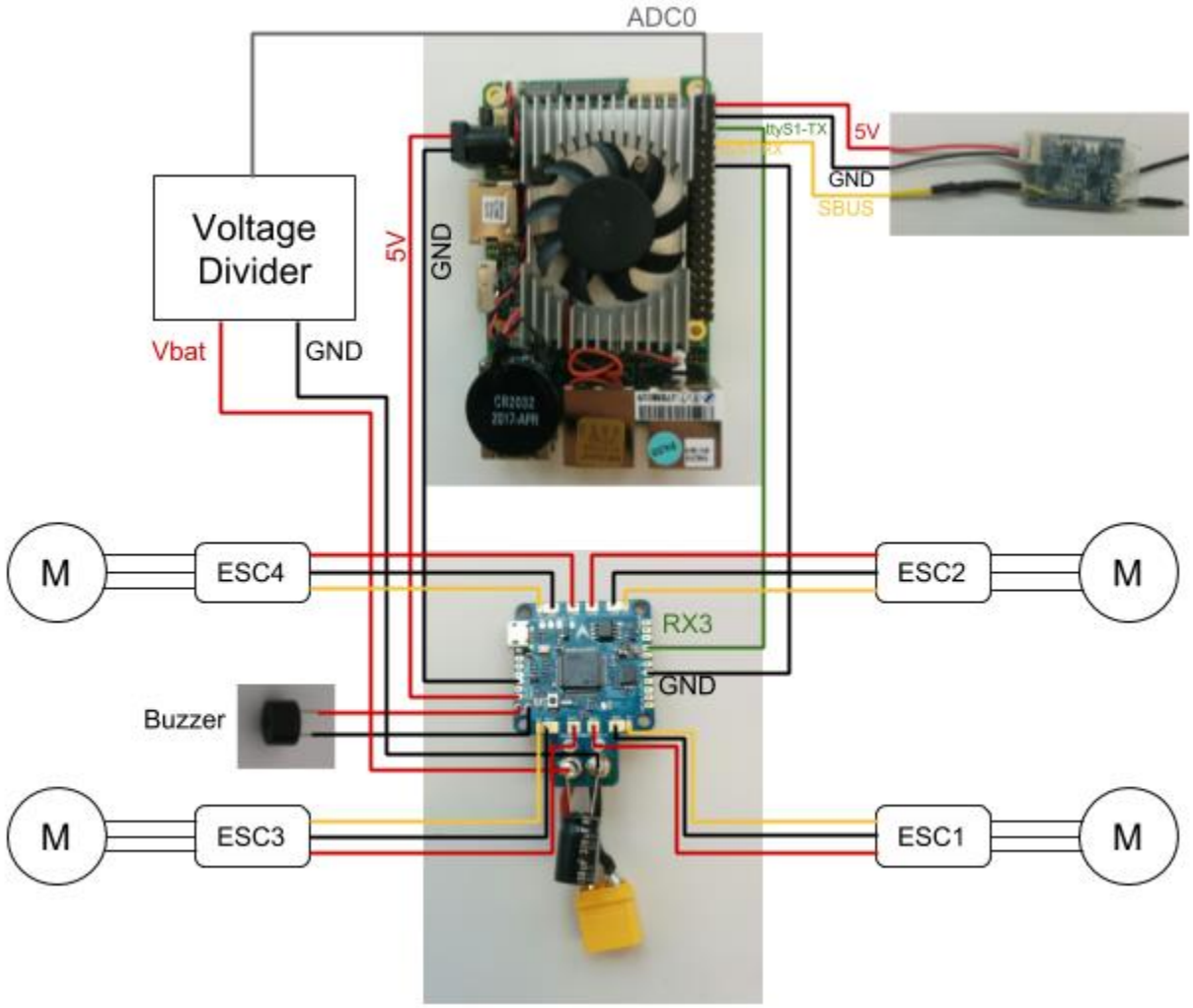


Figure 8 – RPI AND PDP S.BUS connection diagram (https://github.com/uzh-rpg/rpg_quadrotor_control/wiki/SBUS-Wiring)

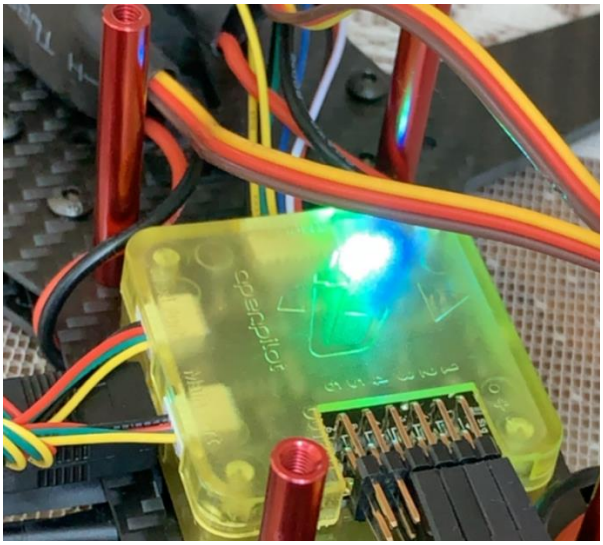
6 Highlight of Major Problem at this stage

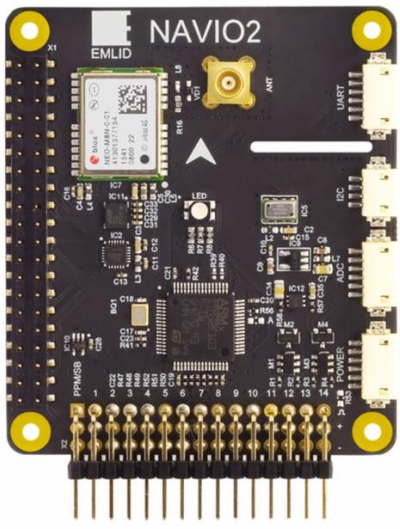
I have quickly realized the MAVLink protocol will not communicate with my current autopilot board – CC3D easily.

- I have spent approximately 1 month trying to connect the autopilot to the Raspberry Pi and have now concluded that even if I did connect the two the product still wouldn't be satisfactory. I could possibly use the serial port and send commands using Python, however, CC3D doesn't have GPS and I have been advised that I get a board that supports GPS because without it the drone will always drift even without wind.

The circuity above and the numerous forum posts were to make this connection. **At this stage, I will order a new autopilot board** – which I will investigate into much more properly to find the right one for me.

Here is the problem, left, and the solution, right:





I have investigated protocols to communicate the boards with via the Ground Control Station. I have found – MAVLink, a communication protocol for small unmanned aerial vehicles – which I will get into in the next section now.

Design (Working around the Problem)

Building a drone overlook:

1. Hardware

The already built prototype in the first design section describes the fundamental hardware of the system. My goal is to be able to communicate this simple prototype drone with a companion computer, which is the Raspberry Pi, that can process inputs such as objects in its field of view.

The hardware additions to the prototype to achieve this will briefly be:

Hardware	**Addition to system**
A flight controller board that is compatible with the companion computer (Raspberry Pi) – Navio2 flight controller board.	Process inputs such as objects in field of view of the system camera. (picamera in this case)
Power module	Power the companion computer and the flight controller.
GPS/GNSS receiver	Collect input data such as current location and altitude readings.

2. Software

- **Control**

In the context of my project, control is all about the movement of the drone. For this I will be making use of a library called DroneKit.
I need to connect a Python script to the drone and run a take off function using DroneKit to give the drone the ability to take off on its own upon booting the flight controller.

- **Perception**

Perception is briefly awareness of surroundings. In this project, I will give the drone eyes only. Although this gives it weaknesses in other areas such as depth perception – eyes are the best place to start in order to give this drone the basic autonomy as described in the scope of the project.
Neural networks and image processing fall under the category of computer vision. This will require complex matrix operations and an idea as to how a neural network processes inputs. Hence, the point is – giving the drone eyes, on its own, is a complex task that I would like to tackle for a complete project.

The one program hypothesis:

The Future of Robotics and Artificial Intelligence (Andrew Ng, Stanford University, STAN 2011)

In this video, Andrew emphasizes how a human brain carries out all its functions and processes all inputs as a single unit. He hypothesizes that a robot should be able to perceive and control, accordingly, passing all its inputs through a single program to function.

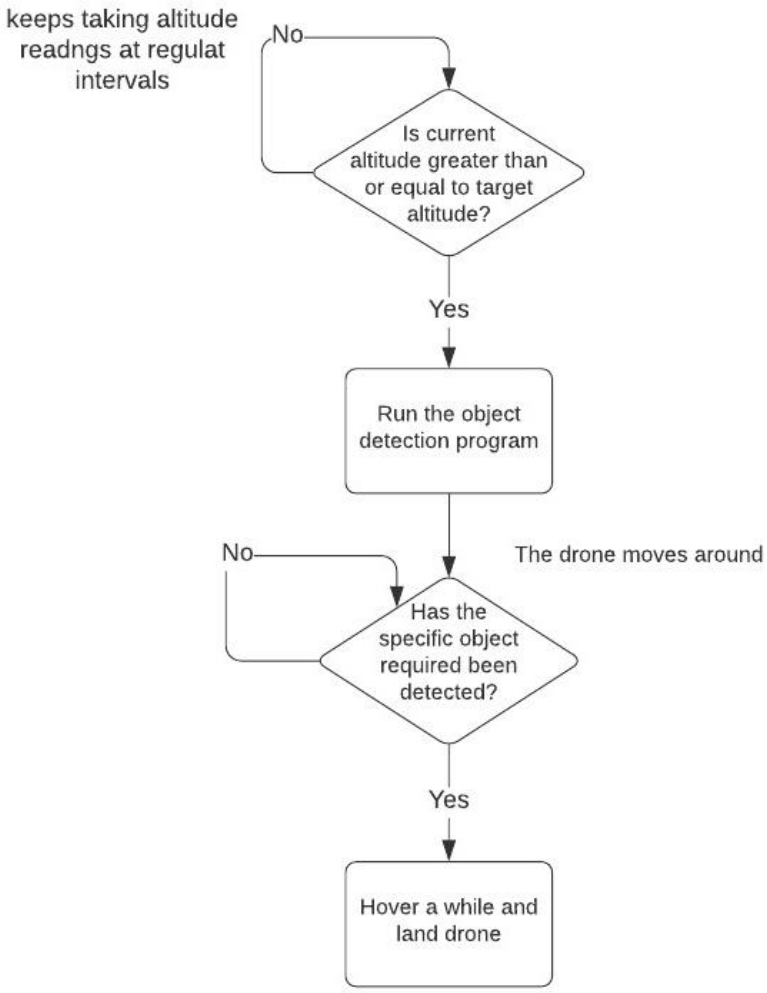
Taking such a concept into account, I would like my Python code to be a single program with multiple functions. What the drone perceives through its camera and GPS will be inputted and processed in this script and will output as the control/behavior of the drone.

Pseudocode could look something like this – assuming the takeoff, object_detected and land functions have been written within the same script.

```
takeoff(target_altitude)
IF object_detected() == True:
        OUTPUT "[Object] Detected"
        time.sleep(hover_time)
        land()
END IF
```

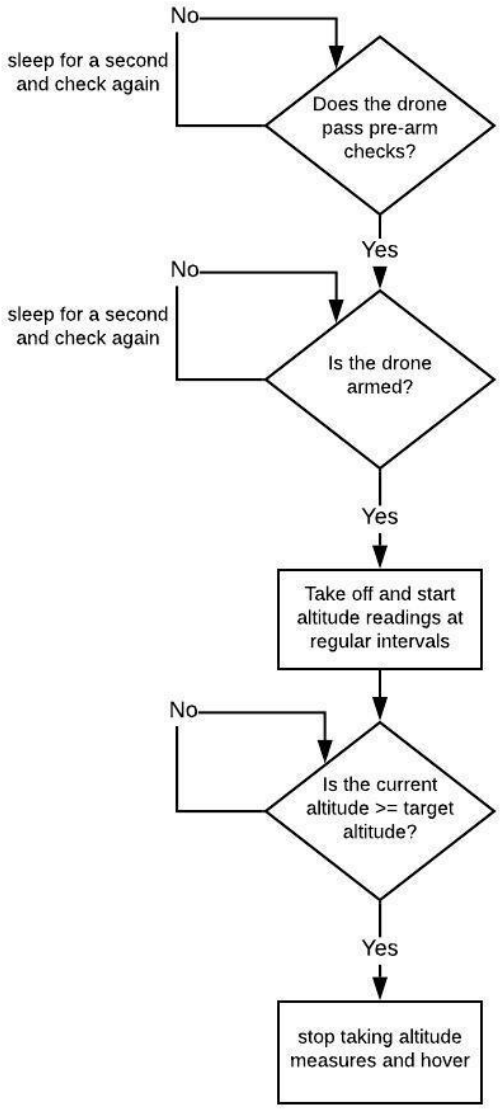
The simplicity of this program which is for a drone that flies and lands if it sees an object is what makes Andrew's hypothesis very commendable.

Here is my design for that single program:

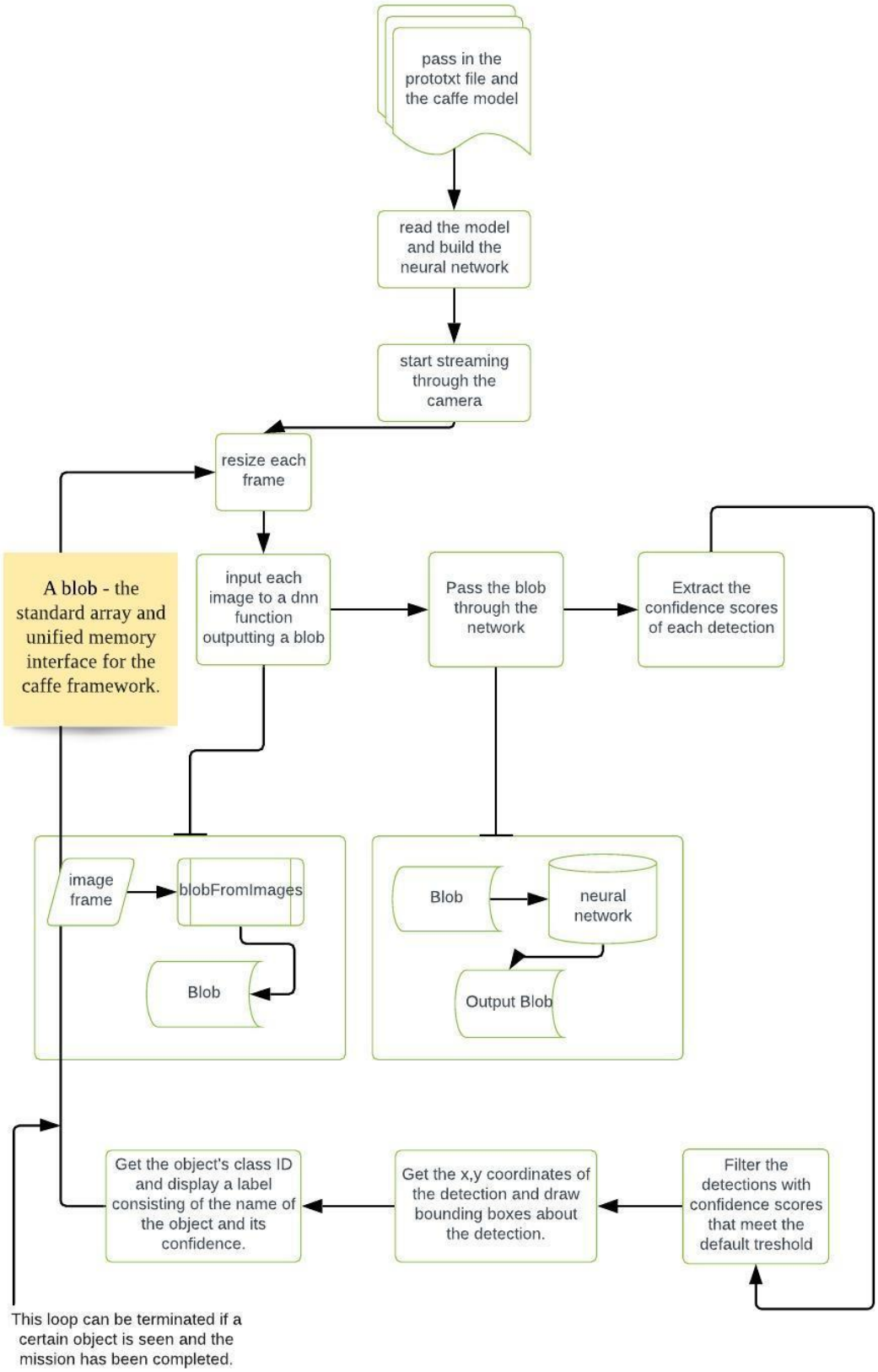


The designs for takeoff and object detection:

Takeoff:


No
sleep for a second and check again
Does the drone pass pre-arm checks?
Yes
No
sleep for a second and check again
Is the drone armed?
Yes
Take off and start altitude readings at regular intervals
No
Is the current altitude >= target altitude?
Yes
stop taking altitude measures and hover

Object Detection:



7 Hardware Changes

7.1 (Fixing the Prototype – New Build)

Communicating the RPi with the FC was much more complex than I figured due to incompatibilities of the FC with any form of serial communication protocol. I've had a look at other various flight controller boards and have concluded that what makes this project complex is the simplicity of the CC3D.

The CC3D doesn't support GPS and neither can it talk with the MAVLink protocol. This really limits the project unless I change the main board or completely discard the autopiloting abilities of the board to make the RPi the brains of the drone. This isn't very ideal as integrating the sensors and trying to make the system stable with additional parts to a RPi has been deemed not very resourceful by previous attempters of such a project.

The better alternative seems to be: An Emlid Navio2 (RPi hat)

Use of a Navio 2, a mission planner such as ArduPilot and dronekit, a library to control the drone autonomously, I could create a larger research drone where the Navio 2 and a companion computer, RPi, can communicate back and forth with MAVLink running on the RPi – controlled via SSH in initials of this project.

The companion computer's presence in the system allows complex computations during flight for a better autonomous system where the use of hardware isn't insanely complex due to all the incompatibilities, I faced with the low cost CC3D.

The Navio 2 seems to be easier to setup and less pricy since it is compatible with everything I have currently in the system.

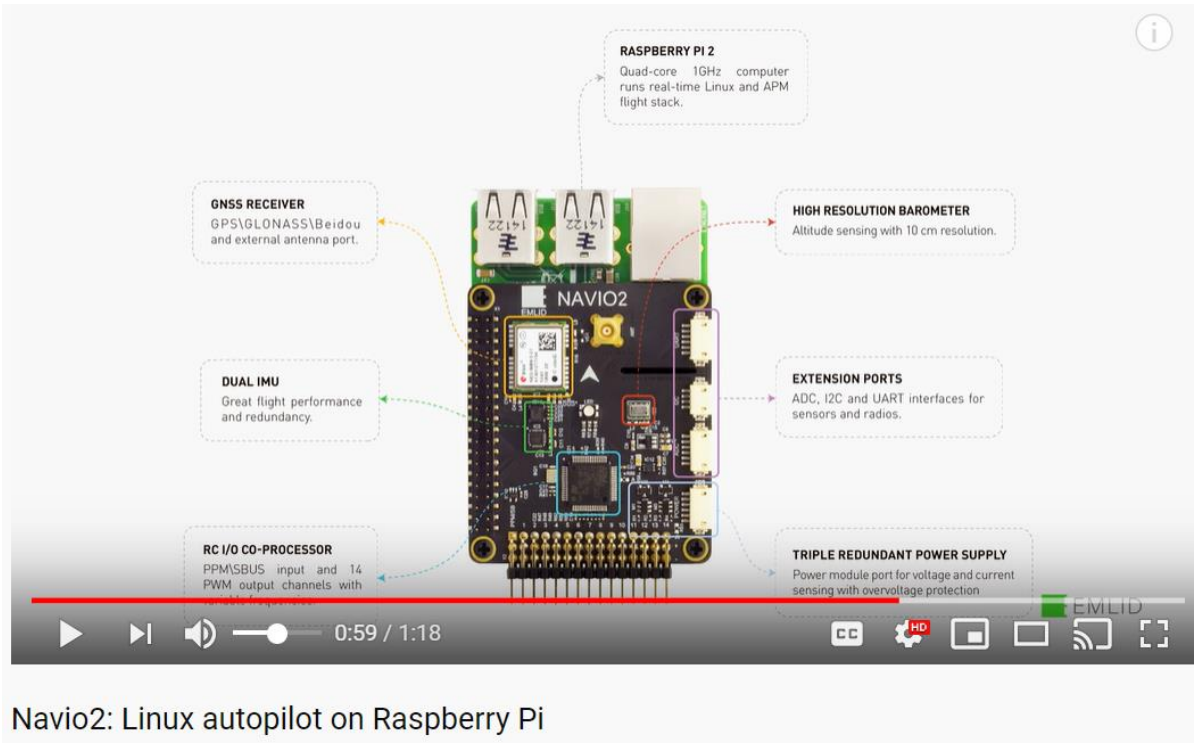


Figure 9 - Research into the Navio 2 capabilities and specs

Essentially, it is a hat for the companion computer - RPi most commonly. It has lots of built in features that no longer requires me to think about adding on every single feature and trying to make it work with a low-level autopilot like the CC3D. Some features that would make this board perfect for my project:

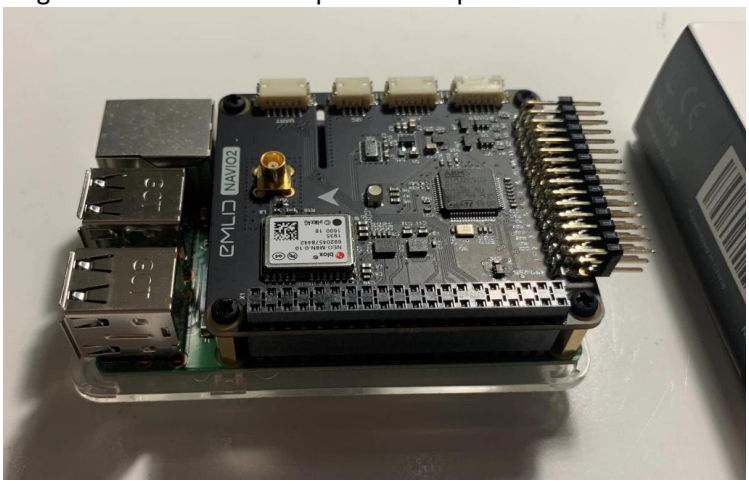
- Dual Inertial Measurement Unit (IMU)
- Barometers (accurate to 10cm of resolution)
- GNSS receiver tracks satellites from all over the world.
- Magnetometer determines the direction of heading
- Extension ports allocate for additional sensors and radio telemetry.
- PPM or SBUS inputs from receiver, and outputs 14 PWM channels for motors and servos at the servo rails .

7.1.1 'Reconfigurations' of the system:

I am going to remove the CC3D from the previous 250 build (prototype) and add these components:

- An Emlid Navio2 autopilot board
- A Power Module
- A PPM supporting receiver (either purchase a new one or use an encoder)

Mounting the board on the companion computer:



Below are the power module and the PPM supporting reciever I picked, also compatible with my current transmitter:

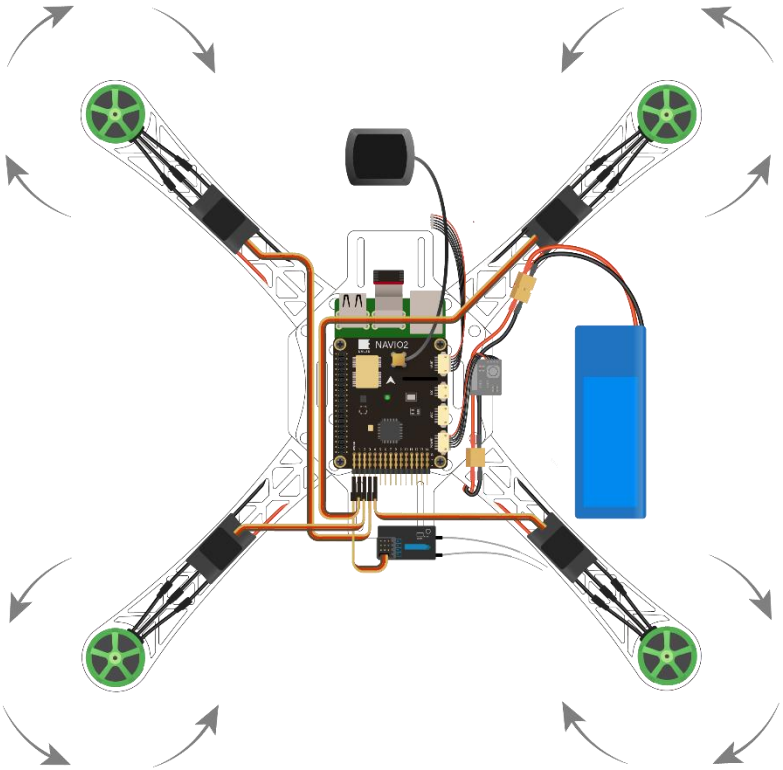


Figure 10 - The new setup summarized.

7.2 Software Changes (DroneKit)

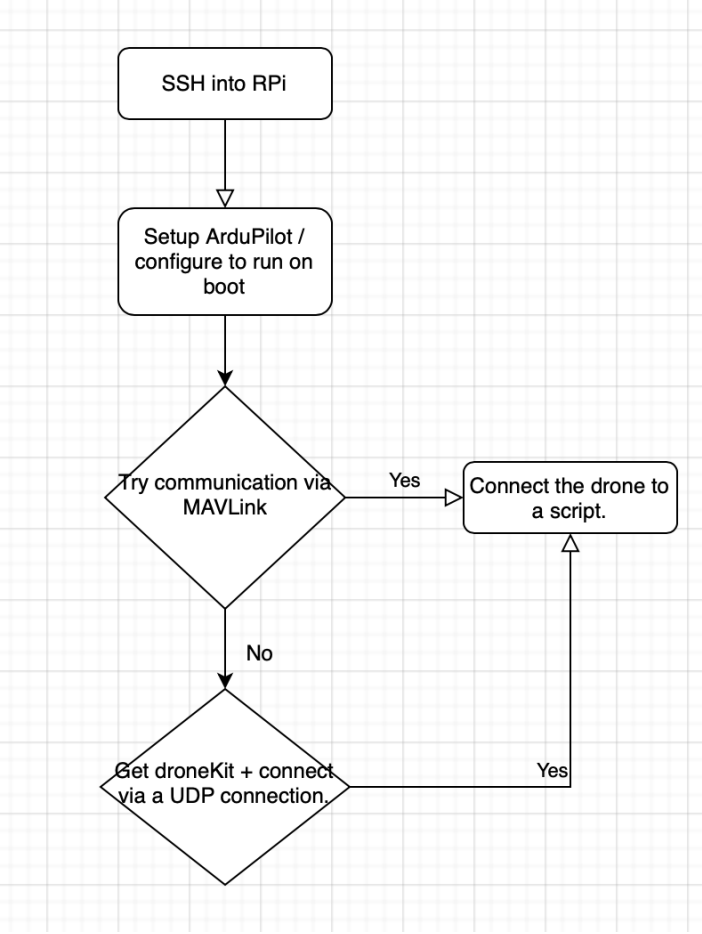
7.2.1 SSH into Pi and setup the RPi software

Emlid (company producing the autopilot board) has their own Rasbian Image . Things to note:

- The Raspberry Pi will be used headless - There is a console prompt – making access to the drone easy.
- The use of an 8-16GB SD card is recommended but larger sizes 32GB+ (being exFAT file systems) is usually preferred since the companion computer is usually desired to do more computation rather than just being able to fly the drone manually.

Extra Note: The Raspberry Pi bootloader cannot read exFAT filesystems, only reads FAT32, therefore the SD card will need to be formatted to FAT32 if a default exFAT is being used.

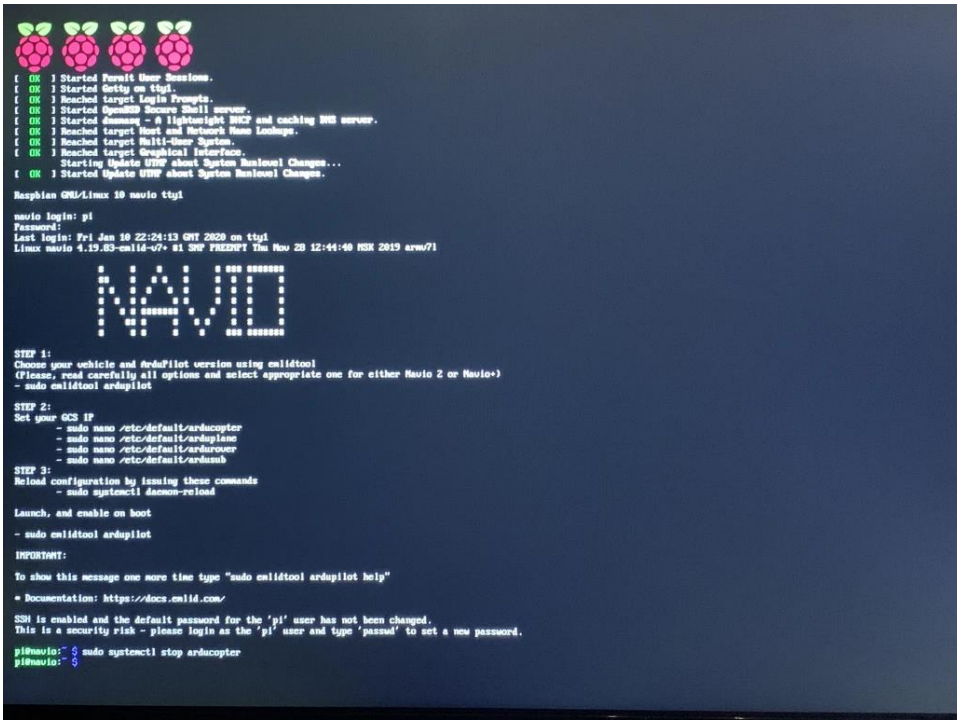
Here's a brief diagram of what's going on:



To make an extended capacity SD card readable by the RPi bootloader – the file system must be formatted to FAT32. The OS image can be burned afterwards.

Using an SSH client – broke into the RPi with the distributed Raspbian image of the flight controller.

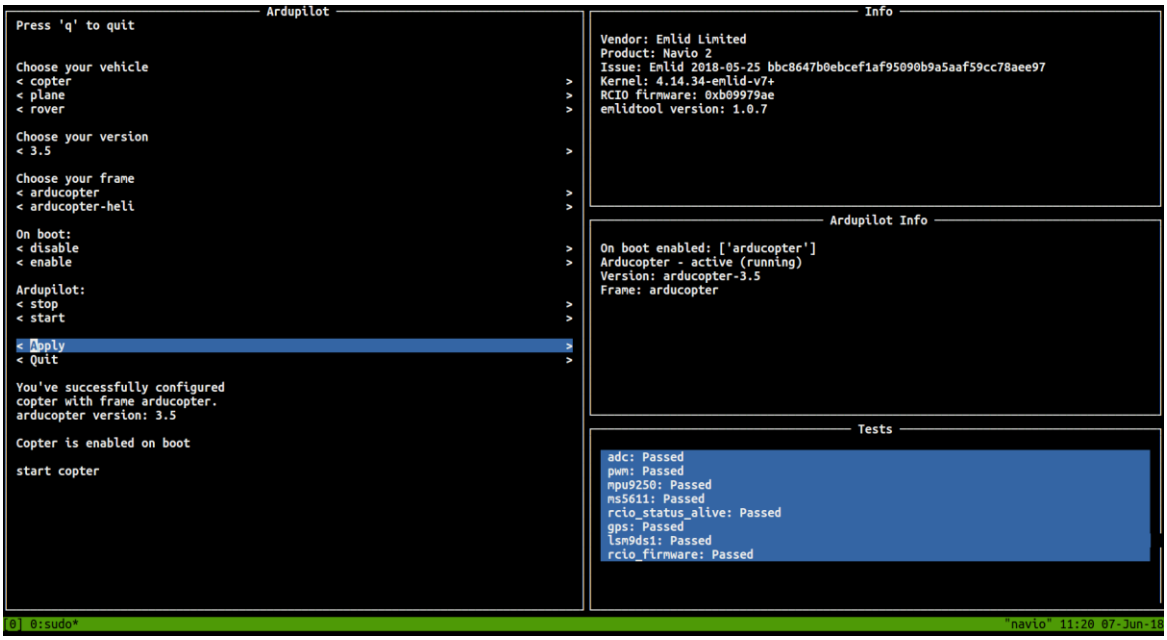
Here is the menu once I SSH into it with the image burned:



The configs are done here to run Ardupilot on the Navio - which is an open-source autopilot system for multicopters, helicopters, and other rotor vehicles.

Here is the simple interface:

Here I just enable the drone to run ArduCopter on boot.



This makes sure that when the battery is connected, the autopilot boots and ArduPilot will start up. This will allow me to connect to the drone from the GCS and receive logs as well as monitor the drone.

7.2.3 MavProxy vs DroneKit

As mentioned before – I was planning on using the MAVLink protocol to communicate with the drone through the Ground Control Station. For this I have found MAVProxy – which is a command-line based "developer" ground station software that complements the GCS used, such as Mission Planner. MAVProxy is commonly used by developers for testing new builds.

MavProxy was a bit complicated to setup but easy to use since it was simply using simple built in commands to arm and fly the drone through the MavLink protocol. I decided instead of MavProxy I could setup an environment where I connect a python script to the Pi and write my own code to control the drone.

For this there is DroneKit, another open source API, which I saw on the Emlid website. I will use DroneKit as it seemed a bit more suited for what I'm doing. Its API communicates with vehicles over MAVLink. It provides programmatic access to a connected vehicle's telemetry, state and parameter information, and enables both mission management and direct control over vehicle movement and operations. To use dronekit I must establish a connection between the drone and my laptop, where the GCS runs.

Note: Where I attempted to setup MavProxy though – I encountered an error that helped me along the way. ** I have more about what I did and why it didn't work in the Appendix.

```
pi@navio:~ $  mavproxy.py --master=/dev/serial0 --baudrate 921600 --aircraft My
opter
Connect /dev/serial0 source_system=255
no script MyCopter/mavinit.scr
Log Directory: MyCopter/logs/2020-02-20/flight1
Telemetry log: MyCopter/logs/2020-02-20/flight1/flight.tlog
Waiting for heartbeat from /dev/serial0
 MAV> link 1 down
```

Here I attempted to connect to the drone through MavProxy. The prompt throws 'link 1 down' which suggests that the link to the drone wasn't working/inactive – which was the Serial0 port.

Instead of the serial port I figured – I could also establish a connection through the internet (using TCP/UDP).

Implementation (Solution to the Problem)

8 Connecting the RPi to the GCS through a UDP connection (SOLVED)

8.1 The Connection

1. Point ArduPilot to Localhost

I must alter the settings for ArduPilot and set where it should get start up on. If I add an option to the file called “TELEM3” and point that to localhost, through port 14550 then add it to OPTS – then localhost will be listened on. Then I can ask for my vehicle to connect to 0.0.0.0:14550 which means - all IP addresses on the local machine. The localhost/loopback address is used to establish an IP connection to my own machine.

Port(s)	Protocol	Service	Details	Source
14550	udp	applications	MAVLink Ground Station Port	*SG*

Briefly, I will setup a UDP connection to send datagrams to the vehicle.

```
pi@navio:~ $ sudo nano /etc/default/arducopter
```

```
# Default settings for ArduPilot for Linux.
# The file is sourced by systemd from arducopter.service

TELEM1="-A /dev/ttyAMA0"
TELEM2="-C tcp:0.0.0.0:5760"
TELEM3="-D udp:127.0.0.1:14550"
# Options to pass to ArduPilot
ARDUPILOT_OPTS="$TELEM1 $TELEM2 $TELEM3"

                        #    # ######  #       #####
                        #    # #       #       #    #
                        ###### #####   #       #    #
                        #    # #       #       #####
                        #    # #       #       #
                        #    # ######  ######  #

# -A is a console switch (usually this is a Wi-Fi link)

# -C is a telemetry switch
# Usually this is either /dev/ttyAMA0 - UART connector on your Navio
# or /dev/ttyUSB0 if you're using a serial to USB convertor

# -B or -E is used to specify non default GPS

# Type "emlidtool ardupilot" for further help
```

I also set TELEM2 to 0.0.0.0:5760 which is the port for a TCP connection to the GCS.

Mapping between switches and serial ports (TCP or UDP can be used instead of serial ports):

- -A - serial 0 (always console; default baud rate 115200)
- -C - serial 1 (normally telemetry 1; default baud rate 57600)

  3DR Radios are configured for 57600 by default, so the simplest way to connect over them is to run with -C option.
- -D - serial 2 (normally telemetry 2; default baud rate 57600)
- -B - serial 3 (normally 1st GPS; default baud rate 38400)
- -E - serial 4 (normally 2st GPS; default baud rate 38400)
- -F - serial 5

A baud rate of 57600 is sufficient for a UDP connection – typed in TELEM3 and the loopback address to send the traffic to through port 14550.
Now I have a telemetry port listening on localhost and a vehicle looking for all IPs on the local machine.

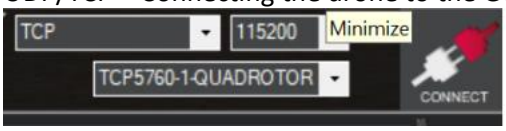
3. Setup ArduPilot Parameters

Setting up which protocol to use – this is what ArduPilot is instructed to use on boot of the Navio2 - MavLink.

SERIAL0_PROTOCOL	1		1:MAVlink1 2:MAVLink2	Control what protocol to use on the console.
SERIAL1_PROTOCOL	1		-1:None 1:MAVLink1 2:MAVLink2 3:Frsky D 4:Frsky SPort 5:GPS 7:Alexmos Gimbal Serial 8:SToRM32 Gimbal Serial 9:Rangefinder 10:FrSky SPort Passthrough (OpenTX) 11:Lidar360 13:Beacon 14:Volz servo out 15:SBus servo out 16:ESC Telemetry 17:Devo Telemetry 18:OpticalFlow 19:RobotisServo 20:NMEA Output 21:WindVane 22:SLCAN 23:RCIN 24:MegaSquirt EFI 25:LTM	Control what protocol to use on the Telem1 port. Note that the Frsky options require external converter hardware. See the wiki for details.
SERIAL2_PROTOCOL	2		-1:None 1:MAVLink1 2:MAVLink2 3:Frsky D 4:Frsky SPort 5:GPS 7:Alexmos Gimbal Serial 8:SToRM32 Gimbal Serial 9:Rangefinder 10:FrSky SPort Passthrough (OpenTX) 11:Lidar360 13:Beacon 14:Volz servo out 15:SBus servo out 16:ESC Telemetry 17:Devo Telemetry 18:OpticalFlow 19:RobotisServo 20:NMEA Output 21:WindVane 22:SLCAN 23:RCIN 24:MegaSquirt EFI 25:LTM	Control what protocol to use on the Telem2 port. Note that the Frsky options require external converter hardware. See the wiki for details.

4. Connect to the GCS

UDP/TCP – Connecting the drone to the GCS (Mission Planner).



The ground control station is essentially the interface displaying information about the flight.

5. Making the connection:

```
from dronekit import connect

vehicle = connect('0.0.0.0:14550', wait_ready=True)

print("Mode: %s" % vehicle.mode.name)
```

```
pi@navio:~ $ python3 connect.py
Mode: STABILIZE
pi@navio:~ $
```

It successfully connects. Now I can write the script.

8.2 The DroneKit Code (Control the drone through a Python Script)

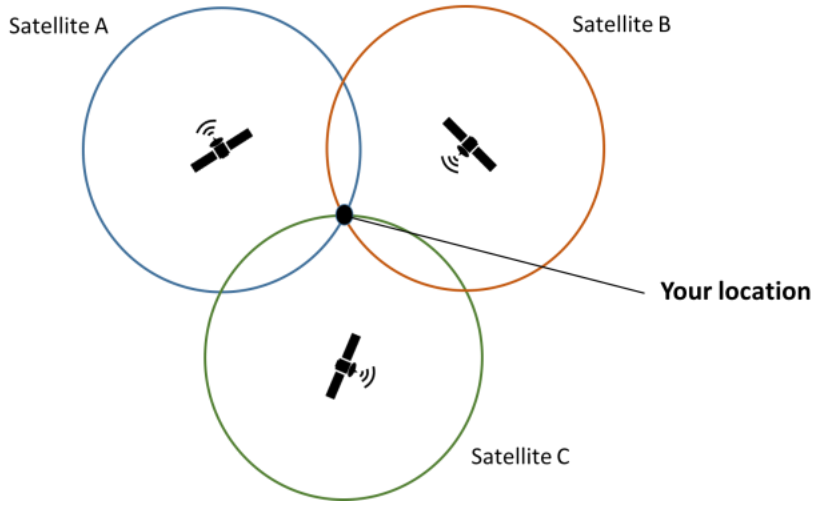
I got dronekit in the same path as OpenCV on the RPi. Now I can write the script which will fly the drone autonomously.

To check the current altitude of the drone I am able to use the (self.vehicle.location.global_relative_frame.alt) function from the dronekit library – it tells me if the drone is at the target yet by reading altitude in regular intervals.

Define a vehicle, drone in this case – and pass it to the location function to get altitude readings during flight:

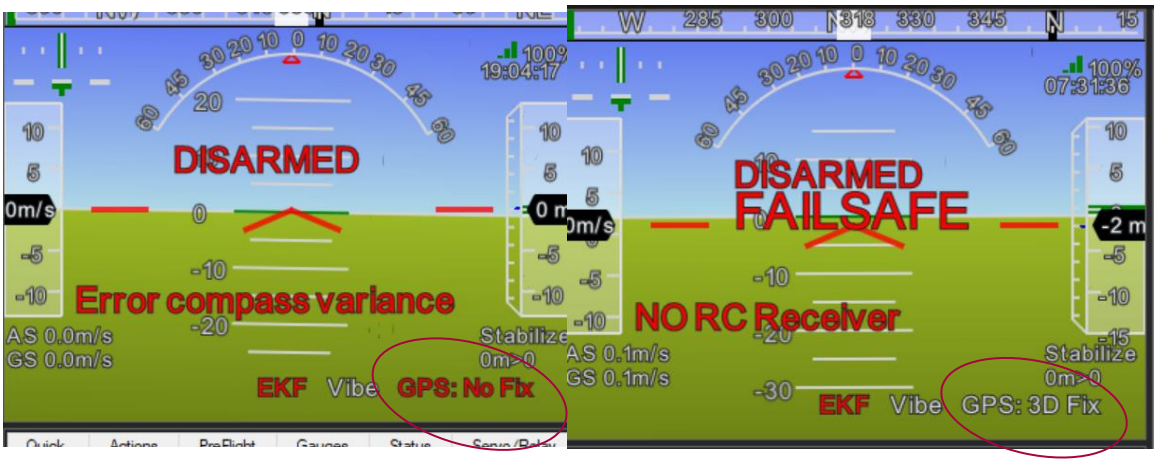
```
vehicle = connect('0.0.0.0:14550', wait_ready=True)

print(" Altitude: ", vehicle.location.global_relative_frame.alt)
```

For the script to run I must pass all pre-arm checks and have a GPS fix. I can't get a GPS fix indoors hence must go out to test this script, in the Testing section.



Trilateration is the name of the finding process. Three of the closest satellites to my location approximate my current position by calculating exactly where they intersect.
Of course, this doesn't work indoors.

The GCS throws errors for when the drone isn't ready to go – this time it was helpful to know the script didn't run due to hardware issues:



These mean I need to get GPS Fix to run this function:

The takeoff function:

```
def takeoff_drone(aTargetAltitude):
    print("Basic pre-arm checks")
    # Don't arm until drone is checked
    while not vehicle.is_armable:
        print(" Waiting for drone to get ready...")
        time.sleep(1)

    print("Arming motors")
    vehicle.mode = VehicleMode("LOITER")
    vehicle.armed = True

    while not vehicle.armed:
        print(" Drone is arming...")
        time.sleep(1)

    print("Taking off!")
    vehicle.simple_takeoff(aTargetAltitude)  # Take off to target altitude

    # Check that vehicle has reached takeoff altitude
    while True:
        print(" Altitude: ", vehicle.location.global_relative_frame.alt)
        # Break and return from function just below target altitude.
        if vehicle.location.global_relative_frame.alt >= aTargetAltitude * 0.95
            print("Target Reached!!")
            break
        time.sleep(1)
```

Loiter Mode automatically attempts to maintain the current location, heading and altitude.

Take an altitude reading and compare to target altitude – if not >= target - take another reading a second later.

Altitude readings are recorded – once the aTargetAltitude is passed, the drone will set to LAND mode.

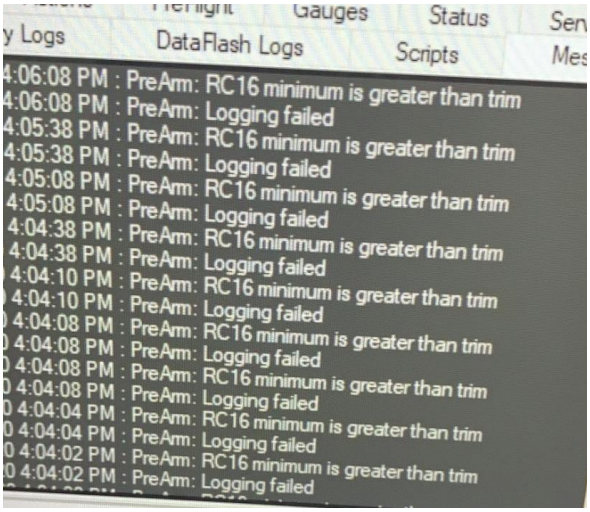
Testing (Autonomous Flight Testing)

9 Test run for RPi and flight controller communication

9.1 Manual Testing (Problem Analysis)

A couple of essential errors to mention:

1. **RC minimum is greater than trim error:**



Errors are monitored from the GCS. This one will stop the drone from ever arming as it won't pass the pre arm checks and neither will it accept proper transmitter input.

This took me a while to fix – eventually found the solution in the parameter list of the GCS.

RC1_DZ	20	PWM	0 200	PWM dead zone in microseconds around trim or bottom
RC1_TRIM	1554	PWM	800 2200	RC trim (neutral) PWM pulse width in microseconds. Typically 1000 is lower limit, 1500 is neutral and 2000 is upper limit.
RC10_DZ	0	PWM	0 200	PWM dead zone in microseconds around trim or bottom
RC10_TRIM	1550	PWM	800 2200	RC trim (neutral) PWM pulse width in microseconds. Typically 1000 is lower limit, 1500 is neutral and 2000 is upper limit.
RC11_DZ	0	PWM	0 200	PWM dead zone in microseconds around trim or bottom
RC11_TRIM	1550	PWM	800 2200	RC trim (neutral) PWM pulse width in microseconds. Typically 1000 is lower limit, 1500 is neutral and 2000 is upper limit.
RC12_DZ	0	PWM	0 200	PWM dead zone in microseconds around trim or bottom
RC12_TRIM	1550	PWM	800 2200	RC trim (neutral) PWM pulse width in microseconds. Typically 1000 is lower limit, 1500 is neutral and 2000 is upper limit.
RC13_DZ	0	PWM	0 200	PWM dead zone in microseconds around trim or bottom
RC13_TRIM	1550	PWM	800 2200	RC trim (neutral) PWM pulse width in microseconds. Typically 1000 is lower limit, 1500 is neutral and 2000 is upper limit.
RC14_DZ	0	PWM	0 200	PWM dead zone in microseconds around trim or bottom
RC14_TRIM	1550	PWM	800 2200	RC trim (neutral) PWM pulse width in microseconds. Typically 1000 is lower limit, 1500 is neutral and 2000 is upper limit.
RC15_DZ	0	PWM	0 200	PWM dead zone in microseconds around trim or bottom
RC15_TRIM	1550	PWM	800 2200	RC trim (neutral) PWM pulse width in microseconds. Typically 1000 is lower limit, 1500 is neutral and 2000 is upper limit.
RC16_DZ	0	PWM	0 200	PWM dead zone in microseconds around trim or bottom
RC16_TRIM	1550	PWM	800 2200	RC trim (neutral) PWM pulse width in microseconds. Typically 1000 is lower limit, 1500 is neutral and 2000 is upper limit.
RC2_DZ	20	PWM	0 200	PWM dead zone in microseconds around trim or bottom
RC2_TRIM	1558	PWM	800 2200	RC trim (neutral) PWM pulse width in microseconds. Typically 1000 is lower limit, 1500 is neutral and 2000 is upper limit.
RC3_DZ	30	PWM	0 200	PWM dead zone in microseconds around trim or bottom
RC3_TRIM	1500	PWM	800 2200	RC trim (neutral) PWM pulse width in microseconds. Typically 1000 is lower limit, 1500 is neutral and 2000 is upper limit.
RC4_DZ	20	PWM	0 200	PWM dead zone in microseconds around trim or bottom

I had to set each RC trim equal to or greater than the minimum (usually 1000).

2. ESC Calibration Issue

This problem took me very long to understand and fix. I asked the forum for this.

sealion420 selin2ky@gmail.com 1

Hello - Can someone please help me?

I am building a drone with a Navio 2 and RPi 3B+. My issue is ESC calibration. I've looked at all the posts and I still am not able to tackle it and get the motors to spin.

RX: FS-iA6B
ESCs: SimonK 12A brushless
Motors: 1804-2400Kv brushless DC motors
Using MissionPlanner AP.

When I plug in the battery the light is green for a while, then flashes red and blue then keeps flashing green afterwards continuously (2 beats at a time). I changed the parameter for ESC_CALIBRATION to 2 then also tried 3 - it flashes the same... Am I not able to enter calibration mode - what is wrong??

The ESCs all beep in unison when I connect the battery does that mean it's already calibrated?? In that case why won't the motors spin although I arm them (5 second right yaw)

I've seen in most posts that they calibrated the ESCs indivually with their recievers and it worked afterwards. How can I calibrate individually - what does this mean?

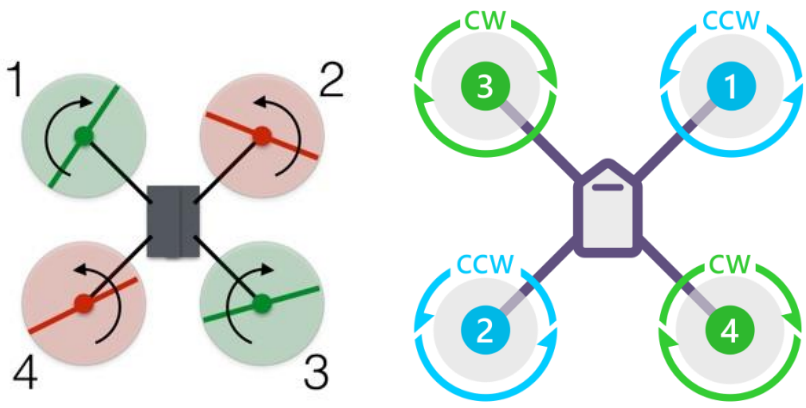
After some answers on how to do individual calibration – I carried it out and the motors started spinning + the board recognized the transmitter input.
ESC calibration issue: https://www.youtube.com/watch?v=BXvvuF2dHNk

3. The drone doesn't take off (again):

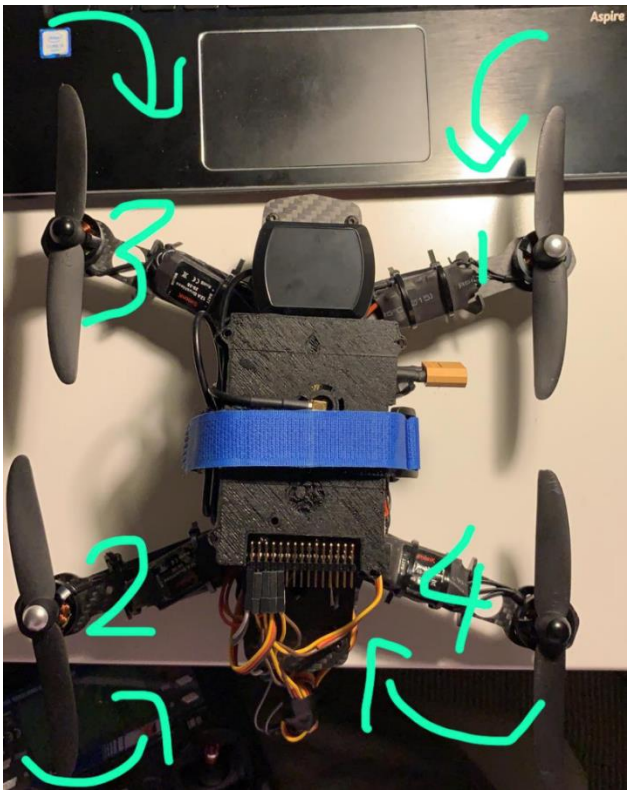
Not taking off: https://www.youtube.com/watch?v=5WKIt3HEwF4

Since I had changed the autopilot boards the configurations have also changed. I wasn't aware that the firmware would differ in motor spin direction and the position + numbering of them.
Hence, I was pushing air upwards instead of down in one side of the drone – hence it did a flip motion on the ground.

Moral of the story is as the firmware changed; system requirements also have changed (LibrePilot, ArduPilot):



I switched up the servo cables at the servo rack of the Navio and recalibrated everything just in case. Ended up with:



This setup worked perfectly, and everything is now smooth.

ARMING_CHECK	12542		0:None 1:All 2:Barometer 4:Compass 8:GPS Lock 16:INS(INertial Sensors - accels & gyros) 32:Parameters(unused) 64:RC Channels 128:Board voltage 256:Battery Level 1024:LoggingAvailable 2048:Hardware safety switch 4096:GPS configuration 8192:System 16384:Mission 32768:RangeFinder	Checks prior to arming motor. This is a bitmask of checks that will be performed before allowing arming. The default is no checks, allowing arming at any time. You can select whatever checks you prefer by adding together the values of each check type to set this parameter. For example, to only allow arming when you have GPS lock and no RC failsafe you would set ARMING_CHECK to 72. For most users it is recommended that you set this to 1 to enable all checks.
ARMING_RUDDER	2		0:Disabled 1:ArmingOnly 2:ArmOrDisarm	Allow arm/disarm by rudder input. When enabled arming can be done with right rudder, disarming with left rudder. Rudder arming only works in manual throttle modes with throttle at zero +- deadzone (RCx_DZ)

Lastly, I configured which checks ArduPilot should make pre-arm and the method of arming the drone (yaw right).

Good to go MissionPlanner screen:

9.2 Testing the DroneKit script to arm the motors

```
pi@navio:~ $ python takeoff.py
Connecting to vehicle on: None
Basic pre-arm checks
Arming motors
Taking off!
 Altitude:  0.162
 Altitude:  0.348
 Altitude:  0.413
 Altitude:  0.785
 Altitude:  1.182
 Altitude:  1.136
 Altitude:  1.013
 Altitude:  0.867
 Altitude:  1.054
 Altitude:  1.272
 Altitude:  1.24
 Altitude:  1.208
 Altitude:  1.024
 Altitude:  0.689
 Altitude:  0.458
 Altitude:  0.342
 Altitude:  0.465
 Altitude:  0.701
 Altitude:  0.82
 Altitude:  0.88
```

Arms the motors through the dronekit script: https://www.youtube.com/watch?v=sdwlUTbwriY

9.3 First Navio2 Flight (Manual)

First try flight with the Navio 2 setup:
https://www.youtube.com/watch?v=bC9M65_Cou8

Implementation (Further Steps to Advanced Autonomy)

10 Real Time Object Detection with Open CV 4

10.1 Explanation of the program

10.1.1 DNN module

The *dnn module* is the deep neural network module. With this module I will simply load a pre trained neural network with which I will categorize objects in a real time video using the module's class labels. The objects will be categorized with bounding boxes. These boxes basically are the classifiers of what the network thinks the object is and where it is in the real time stream.
I will simply use my laptop camera for the first version of this before I get everything running on the Pi with the Pi camera.

As discussed on page 6, here's the result I'm looking for, but real time:

Figure 11

Bounding boxes and the confidence level as to what's labelled being correct is what I am looking to add to the program.

10.1.2 Other Libraries and files needed

Libraries
Having researched this project for a while now, first thing I know I need is cv2, imutils, imutils.video, argparse and numpy.

- Cv2 is the opencv library through which I will use cv2.dnn() often for the deep neural network module
- Imutils is an image processing package that can resize, translate and work on visual inputs etc.
- Imutils.video is specifically an extension module to work with real time streams. The class VideoStreams in fact supports a webcam and a picamera. Starting off with the webcam currently, picamera will come later.
- Agparse is for user friendly command line interfaces performing argument parsing, I will discuss this further.
- Numpy is used for multidimensional arrays

The program is mainly just the use of these libraries to pass an input of the video stream through a neural network, provided by the cv2.dnn() module.

Files
For a CNN to process the input stream, I need to build the actual network. I will do this through passing these two files as arguments at runtime.

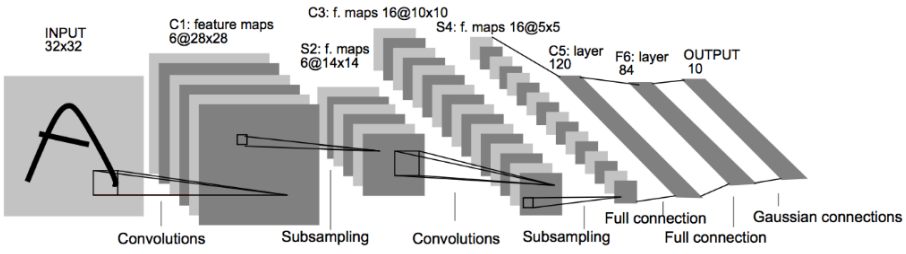
- .prototxt file
- .caffemodel file

The prototxt file is the MobileNets SSD detection network architecture while the caffemodel is simply data about the weights of the inputs for the 20 classes that the Caffe framework can detect.

10.1.4 Caffe Model

Caffe is a deep learning framework. The developers refer to a deep learning neural network as a 'brewer' and suggest that they 'brew for expression, speed, modularity for flexibility, openness for reproducibility' etc. This list has caught my eye as it follows exactly pros of the OOP paradigm.

The architecture I'm using is a convolutional neural network, the framework looks as such:



CNN called LeNet by Yann LeCun (1998)

Figure 12

The entire architecure is an implementation of the MobileNets SSD detection network. I've looked through the file which seemed to consist of 17 layers and 21 classes, one of them being a background class. Therefore I know this network has the weights of 20 different object class inputs and can output them/detect them if inputted into the network.

I will use the Caffe deep learning framework for my program which is a version of TensorFlow's implementation of MobileNets. The framework was trained on the COCO dataset hence can label up to 2 classes, including:
Airplanes, bicycles, birds, boats, bottles, buses, cars, cats, chairs, cows, dining tables, dogs, horses, motorbikes, people, potted plants, sheep, sofas, trains, and tv monitors.
As suggested on its website.

10.1.5 Building the detection network

10.1.5.1 Argument Parsing through the command line

I will import argparse to let the user of the program provide values for variables during runtime. This is important to produce a robust system where the user can input videos into the program which need not be pre-determined.
This communication is on a command level and gives flexibility to the user as the program is ran.

The argparse module makes it easy to write user-friendly command-line interfaces.

```
Parser = argparse.ArgumentParser()

ArgumentParser.add_argument(name or flags...[, action][, nargs][, const][,
default][, type][, choices][, required][, help]…)
```

For my implementation I needed the arguments to be files that build the CNN and pass the weights of the inputs that I give it with the stream.

```
Parser = argparse.ArgumentParser()
Parser.add_argument("--prototxt", required=True)
Parser.add_argument("--model", required=True)
Parser.add_argument("--confidence", type=float, default=0.2)
args = vars(Parser.parse_args())
```

```
C:\DS\Anaconda3\envs\opencv_this\python.exe C:/Users/Acer/PycharmProjects/object_detection/real-time-object-detection/object_detection_dnn.py
usage: object_detection_dnn.py [-h] --prototxt PROTOTXT --model MODEL
                               [--confidence CONFIDENCE]
object_detection_dnn.py: error: the following arguments are required: --prototxt, --model

Process finished with exit code 2
```

When I run the code that constructs the parser and passes in the arguments I get this input at terminal. This is because, as I said, I must pass in the values for the arguments "prototxt" and "model" as I run the program, which happen to be the prototxt and caffemodel files that build the neural network.

As I add arguments to the arguments parser with the Parser.add_argument() method I have specifically stated that values are required to be passed in as the user runs the program with Required = True.

This does give the user the flexibility to run their own models and add their own input weights. Since my project isn't to serve this purpose I can just pass in the proto file and the model directly into the network, CNN, to avoid having to command the path to the files each time the program is ran. This is also easier for me as I develop the system.

```
Parser = argparse.ArgumentParser()
Parser.add_argument("--prototxt", required=False)
```

```
Parser.add_argument("--model", required=False
Parser.add_argument("--confidence", type=float, default=0.2)  # default confidence
is 20%
args = vars(Parser.parse_args())
```

Uses the .parse_args() method. This will inspect the command line, convert each argument to the appropriate type and then invoke the appropriate action. The variables are stores in args[] which can now allow me to access the argument throughout the program as args[some_argument].

10.1.5.2 cv2.dnn.readNetFromCaffe

Open CV has a Net class from which I will use the readNetFromCaffe module which reads a network model stored in Caffe model in memory.

Open CV docs:

```
Retval = cv.dnn.readNetFromCaffe(prototxt,caffeModel)
```

The files I touched on in 6.1.2 are what will be passed here.

- Prototxt is the MobileNets SSD detection network architecture
- Caffemodel is data about the weights of the inputs for the 20 classes

So;

```
CNN = cv2.dnn.readNetFromCaffe("MobileNetSSD_deploy.prototxt.txt",
"MobileNetSSD_deploy.caffemodel")
```

The model read, Caffe Model:

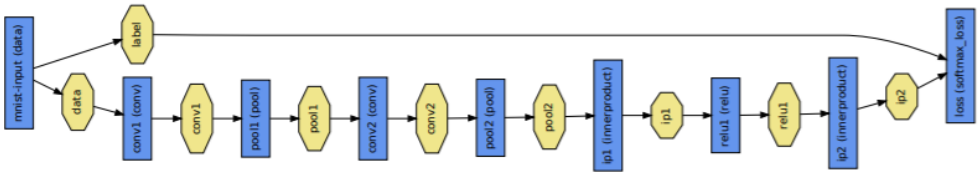


Figure 1: An MNIST digit classification example of a Caffe network, where blue boxes represent layers and yellow octagons represent data blobs produced by or fed into the layers.

This is the architecture that the prototxt file contains in a flowchart form. The data, as a blob, is inputted to pass through two convolutional layers, followed by a pooling layer.
A very vague explanation of the process would be: The blob inputted into the network is passed through the convolution layers where the processing occurs. The detection produced is the data multiplied by its weight and an activation function. The data goes through multiple layers before an output in produced.

Slides from Caffe developers suggest that data flows through the created net as blobs which happen to be multidimensional arrays.

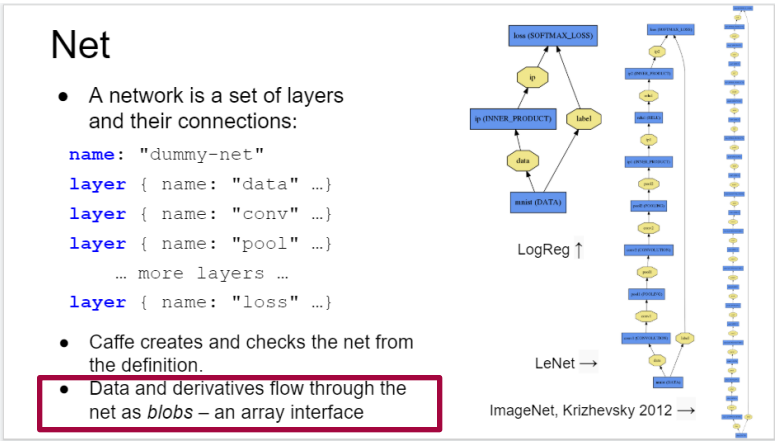


Figure 13 – Caffe slides, explaining the network structure

Browsing through the Caffe Framework meeting slides and website it's clear how data is inputted into the detection network.

Blobs:

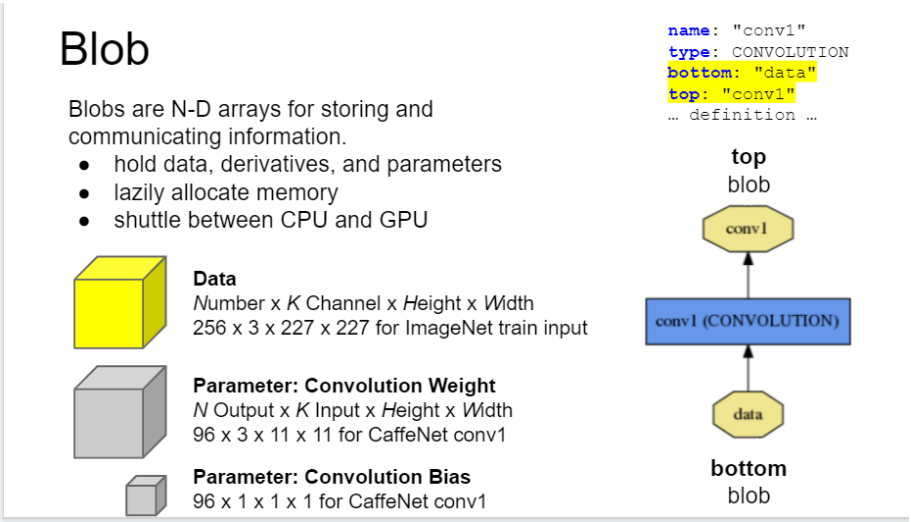


Figure 14 - Caffe slides - Blobs

The CNN, convolutional network takes inputs as blobs, which are multidimensional arrays for storing communication information.
The data I want to be inputting is the real time camera footage through my laptop camera. Therefore, I know I can make use of the imutils functions which provide a VideoStream class that supports both webcams and the picamera for image processing.

I will, in order, stream through my camera, take frames every 2 seconds with the video.FPS() class which basically sets a timer to take the frames, and resize them before I store them in a multi-dimensional array - convert the data there into a blob to feed into the detection network.

```
# Stream and start FPS counter

vs = VideoStream(usePiCamera=True).start()
print("Webcam is turning on...")
time.sleep(2.0)
Frames_Per_Second = FPS().start()

# loop through Frames read from the stream and resize them, all equal length

while True:
    Frame = Video_Stream.read()
    Frame = imutils.resize(Frame, width=400)
```

Now I need to convert the Frame contents into a blob.

10.1.5.3 cv2.dnn.blobFromImage()

I know that a blob is a collection of images with the **same dimensions**, which is resized before input, and same depth, that are all processed in the same manner. It is effectively a 4-dimensional array as Blob(image, scale, size, mean subtraction)

In form from Open CV docs:

```
blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size, mean, swapRB=True)
```

Here are the parameters, taken from the Open CV docs. The parameters required reflect the properties behind how the neural network in fact works.

- I need to feed the frames as the collection of images into 'image' and resize them into the same dimensions. Typically, (299,299)+
- The scale factor is to scale the images after mean subtraction. I'm planning to go with the default and see if this causes any issues.
- Size is again the spatial size the CNN expects, this is just the same as the frame sizes.
- The mean is the mean of the subtraction values.
- SwapRB seems to be color related, swapping the R and B in RGB. I will leave this default as well.

This function will return a blob, as an input image.

```
# Create 4-dimensional blob from image:
blob = cv2.dnn.blobFromImage(image, 1.0, (300, 300), [104., 117., 123.],
False, False)
```

In this case, this means we want to run the model on BGR images resized to *300 x 300*, applying a mean subtraction of `(104, 117, 123)` values for the blue, green, and red channels, respectively. This can be summarized in the following table:

Model	Scale	Size WxH	Mean subtraction	Channels order
OpenCV face detector	1.0	*300 x 300*	104, 177, 123	BGR

At this point, we can set the blob as input and obtain the detections as follows:

```
# Set the blob as input and obtain the detections:
net.setInput(blob)
detections = net.forward()
```

Figure 15 - Open CV4 online materials for the dnn module

Using the default scale factor, the program threw no exceptions but the camera would turn on and then off immediately. With trial and error I just decreased the scaling - thinking maybe it was too much for the network to process.

Therefore, scale is now 0.005 – which runs smoothly.
Mean subtraction is just 123 – taken from the docs.

```
# Making use of the dnn module of Open CV to get blobs from the frames

Main_Blob = cv2.dnn.blobFromImages(cv2.resize(Frame, (300, 300)), 0.005, (300,
300), 123)
```

Here's a flowchart of the system so far.

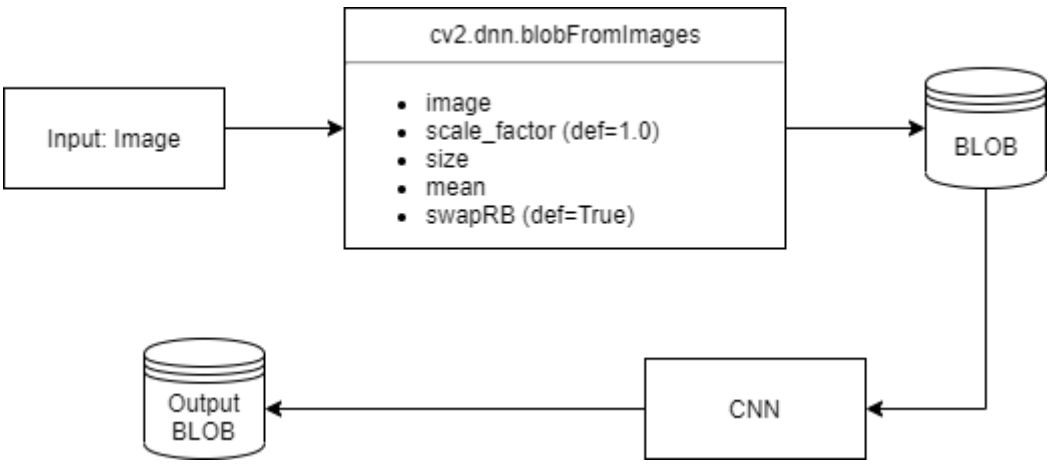


Figure 16 – How frame inputs are converted into a blob to be passed through the CNN, producing detections.

10.1.5.4 Passing the blob into the CNN

Again, from the cv2.dnn.Net() class I can use two methods to pass in the blob and forward the output from the network.

The new input value for the network is set with:

```
cv.dnn_Net.setInput(blob_name)
```

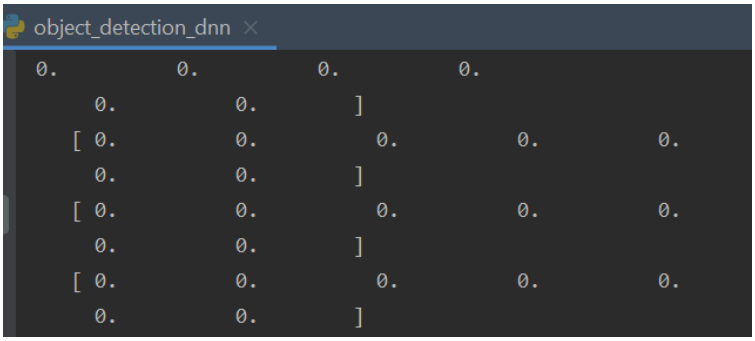
The output blobs are forwarded with:

```
outputBlobs = cv.dnn_Net.forward()
```

So;

```
CNN.setInput(Main_Blob)
    Detection = CNN.forward()
    print(Detection)  # To see the output blob
```

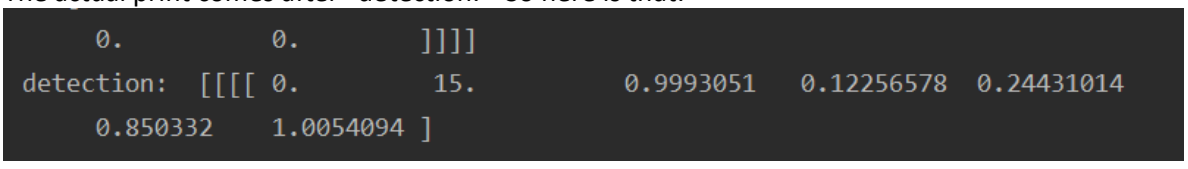
Here's the Detection, output blob, printed at Terminal.

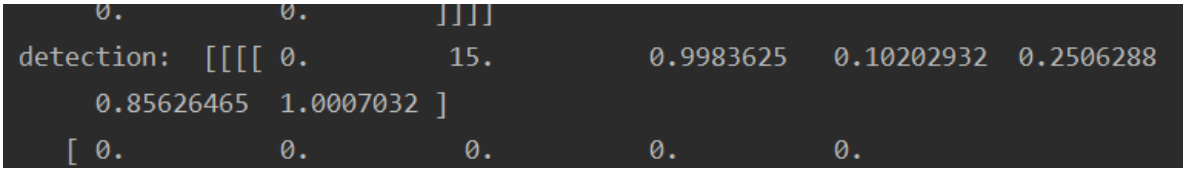


This, to me, seems like a 4-dimensional array with no values – suggesting no detections being made from the frames inputted into the network. So I figured to add a label to see what happens when I actually put an object in the camera that is a part of the 20 classes that the Caffe model contains.

```
print("detection: ", Detection)
```

The actual print comes after "detection: " So here is that:





I checked GitHub for opencv/samples/dnn and there is an explanation of what the output blob in fact holds. So now I know that 0 is the batchID, 15 is the classID for the detection of a person indeed and the confidence score is 0.99… rounded to a 100%

The left, top, right, bottom values are to coordinates of where the detection lies within the frames passed into the CNN as a blob. I will use the output blob to get the coordinates of the bounding box to draw around the detection, the classID name and confidence scores to label the box on the frame.

Briefly, the Output blobs are 1 x 1 x N x 7 and N x C → I can use this for the next section where I get boxes and confidence scores.

```
elif lastLayer.type == 'DetectionOutput':
    # Network produces output blob with a shape 1x1xNx7 where N is a number of
    # detections and an every detection is a vector of values
    # [batchId, classId, confidence, left, top, right, bottom]
```

```
elif lastLayer.type == 'Region':
    # Network produces output blob with a shape NxC where N is a number of
    # detected objects and C is a number of classes + 4 where the first 4
    # numbers are [center_x, center_y, width, height]
```

Where N is a vector of 7 values

	0	1	2	3	4	5	6
1	Batch ID	Class ID	Confidence_score	left	top	right	bottom

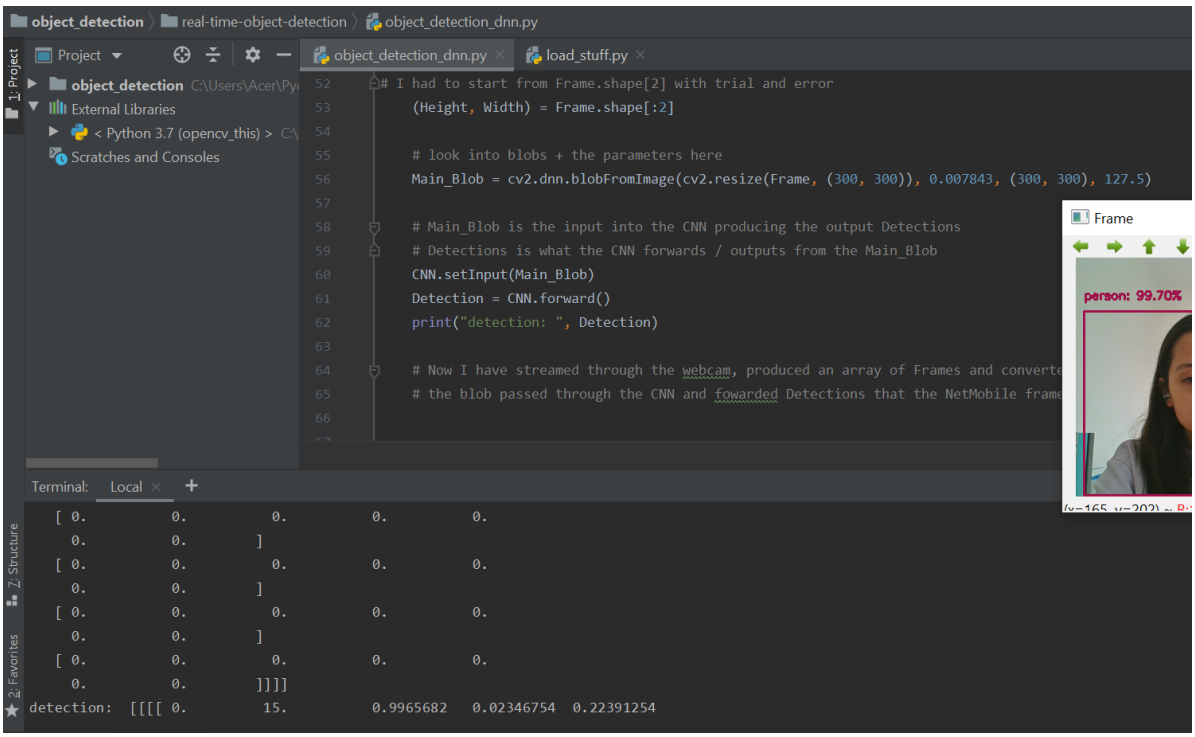


Figure 17 - Print screen of Detections[] in the terminal

10.1.5.5 Measuring accuracy of the detections

I know from observing the output blob in 6.1.8 that the third column gives the confidence scores of the detection. I must loop through the array to get to Detection[2] which is column 3, hence confidence_score.

So, `Detection[0, 0, i, 2]`

I can use numpy to look through the array with numpy.arange()

```
numpy.arange([start], stop, [step ], dtype=None)
```

So:

```
    for i in numpy.arange(0, Detection.shape[2]):
        confidence_score = Detection[0, 0, i, 2]
```

I used the .shape() property to get the current shape of the array, which is used to get to the exact column 3 where the scores are.

10.1.5.6 Marking the detection with bounding boxes

First, I have mentioned the product I wanted as a result of this program which included bounding boxes and a confidence score of the detection around the object detected. These are to let the user know what is detected and how certain the CNN is of the output blob.

I must make sure now that detections will only be made if the confidence score that the output blob gave is greater than the default argument confidence. This is to filter out weak detections.

```
        if confidence_score > args["confidence"]:
```

To create a label for the detection I must get the values of the class label. Also get the x,y values of the box to be drawn around the object detected.
I can get to the number of detections in the third column of the output blob from which I want column number one containing the classID of that Detection.

```
Classes = ["background", "aeroplane", "bicycle", "bird", "boat", "bottle", "bus", "car",
"cat", "chair", "cow", "dining_table", "dog", "horse", "motorbike", "person",
"potted_plant", "sheep", "sofa", "train", "tv_monitor"]
```

```
            index = int(Detection[0, 0, i, 1])
```

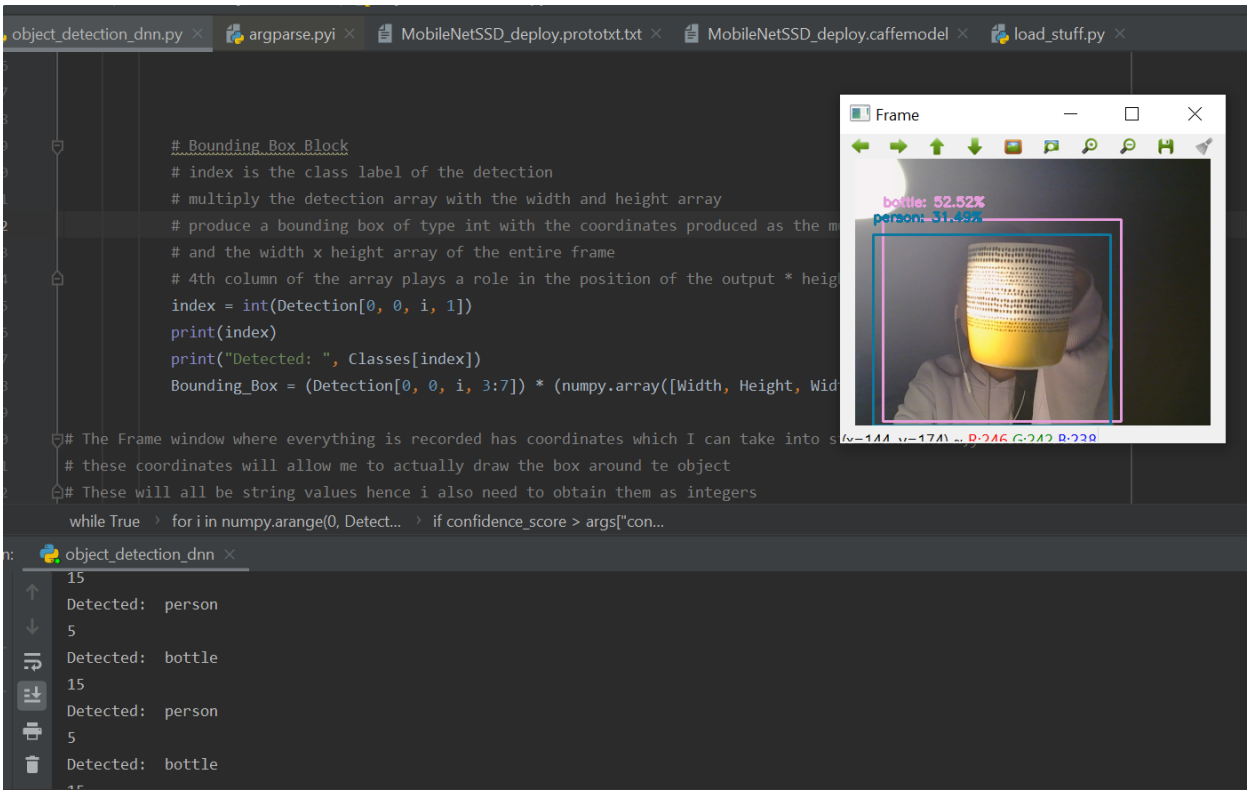


Figure 18 - print screen of the terminal and the Frame in action

Here, I can see the index being taken and passed into Classes[] returns the class in the array Classes that has been detected.
Since the input is given with a FPS counter – the input is simply frames per second and I get a continuous loop of the same detection in that frame with 2 seconds delay.

The bounding box coordinates are created with matrix operations. And I can extract the coordinates of the bounding box in a 4-dimensional array as type integer to use them to draw a rectangle around the detection .

```
            Bounding_Box = (Detection[0, 0, i, 3:7]) * (numpy.array([Width,
Height, Width, Height]))

            (startX, startY, endX, endY) = Bounding_Box.astype("int")
```

The label has to be the class and the confidence score concatenated.
Therefore, I wasn't able to just run:

```
label = (Classes[index]) + (confidence_score * 100)  # didn't work
```

Throws this, I have to format from double to string to have an actual label for the bounding box.

```
    0.0000000e+00 0.0000000e+00 0.0000000e+00]]]]
Traceback (most recent call last):
  File "C:/Users/Acer/PycharmProjects/object_detection/real-time-object-detection/object_detection_dnn.py", line 106, in <module>
    cv2.putText(Frame, label, (startX, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5, Colors[index], 2)
SystemError: <built-in function putText> returned NULL without setting an error

Process finished with exit code 1
```

```
label = "{}: {:.2f}%".format(Classes[index], confidence_score * 100)
```

Creates:

Now I just need to draw the rectangle with the Open CV library .rectangle() method and pass in the coordinates I extracted with the matrix operation involving the detection coordinates and the height and width of the entire frame inputted into the blob.

The draw function in cv2 for rectangles:

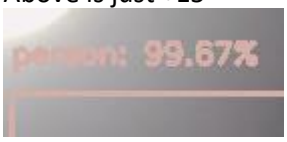
```
cv.Rectangle(img, pt1, pt2, color, thickness=1, lineType=8, shift=0)
```

I just passed in the Frame, the coordinates of the bounding box, a random color from Colors[] and a thickness of 2 to make is nice and visible.

```
cv2.rectangle(Frame, (startX, startY), (endX, endY), Colors[index], 2)
```

Last thing - just display the label next to the bounding box, where the x coordinate is the same as the bounding box so that the label follows the box directly as the object or the camera moves. But the y coordinate can be calculated where the label could be above or below the bounding box so that the label doesn't leave the frame with the box if the object moves towards an extreme end.
So;

Above is just +15



Below is -15 the starting y coordinate which is the y coordinate of the box.


```
y = startY - 15
            if startY - 15 > 15:
                y = startY - 15
            else:
                y = startY + 15
            cv2.putText(Frame, label, (startX, y), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
Colors[index], 2)
```

10.2 How would a user work with this system?

10.2.1 How does training happen? (the neural network process)

A neuron is a mathematical function that takes several inputs and outputs a value. If the function is a linear combination function, then, the output would be the sum of each input value multiplied by its weight.
E.g.

$$f(x_1, x_2) = w_1 x_1 + w_2 x_2$$

A neural network is also a mathematical function. It is defined by a bunch of neurons connected to each other. This means that the output from one neuron is used as an input to other neurons in the network.
The output value of passing some inputs through a neural network will be the sum of each input value multiplied by its weight multiplied by an activation function such as a 'Sigmoid Activation'. The output can be a value in between 0 and 1.

There are of course multiple layers to this and much more to go into such as Loss Functions, Gradient Descent, etc. However, it is sufficient for a user to know that to detect objects within images we need ,for each dataset, to find a function, which takes all the numbers from an image and outputs a probability of there being a specific object somewhere within the image.

When it comes to training, there are different approaches a neural network will 'learn':

- Supervised - It is possible to compare the network's calculated values for the output nodes to "correct" values and calculate an error term for each node. These error terms are then used to adjust the weights in the hidden layers so that, hopefully, the next time around the output values will be closer to the "correct" values. This is telling the network what is right and spoon feeding it correct answers until it can independently come to that answer on its own.
- Iterative - During the training of a network the same set of data is processed many times as the connection weights are continually refined. During this learning phase, the network learns by adjusting the weights to be able to predict the correct class label of input samples.

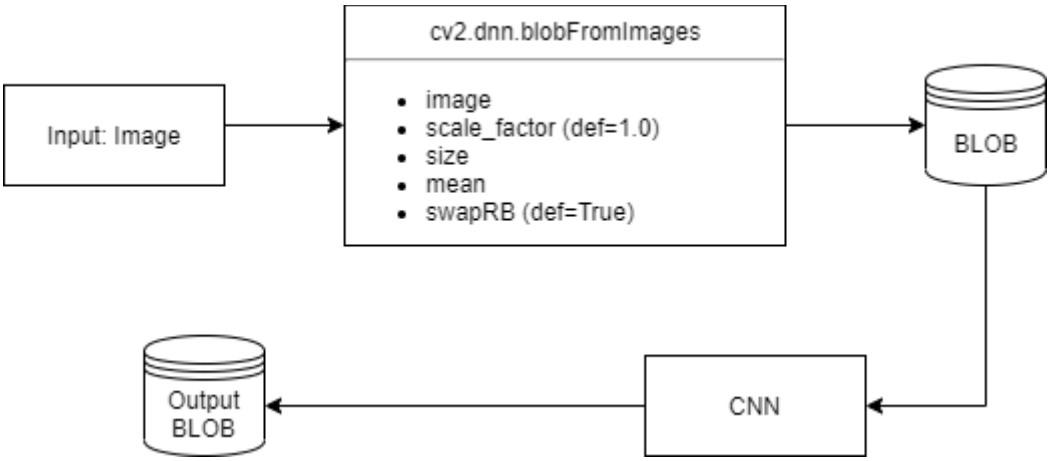
There is also Feedforward, Back-Propagation but since my model was pre-trained in a supervised and iterative manner, I would only explain the two techniques to the user now.

10.2.2 Data storage within the Framework

The Caffe framework stores and communicates data using blobs. Mathematically, a blob is an N-dimensional array stored in a C-contiguous fashion. Blobs provide a unified memory interface holding data, they are wrappers over the actual data being processed and passed along by the framework.

This means that trailing dimensions are stored first. That is if you have c by h by w 3d blob, in memory rows will be saved one after the other, and after completing all the rows of the first channel, only then the rows of the next channel are written.

So, what happens to blobs that went through the network?



The output blob looks like this if something is detected:

```
     0.          0.        ]]]]
detection:  [[[[ 0.         15.          0.9993051   0.12256578  0.24431014
     0.850332    1.0054094 ]
```

It is a vector of N x C where N is a vector of the 7 values represented in the print screen above.
As said before – N is:

	0	1	2	3	4	5	6
N	Batch ID	Class ID	Confidence_score	left	top	right	bottom

Memory on the host and device is allocated on demand (lazily) for efficient memory usage.
Data outputted as a result can be accessed through the blobs which are basically containers of data used across the object detection program at runtime.

10.2.3 Can a user add more objects to the database?

The network used is an implementation of the MobileNets SSD detection network. This network has the weights of 20 different object class inputs and can output them/detect them if inputted into the network.

To detect other objects a user would need **to create their own prototxt file containing the inputs and training weights of each object.** Such data is available online – websites such as ImageNet. As explained, *to know that to detect objects within images we need, for each dataset, to find a function, which takes all the numbers from an image and outputs a probability of there being a specific object somewhere within the image.*

10.3 Onboard Processing with the Raspberry Pi

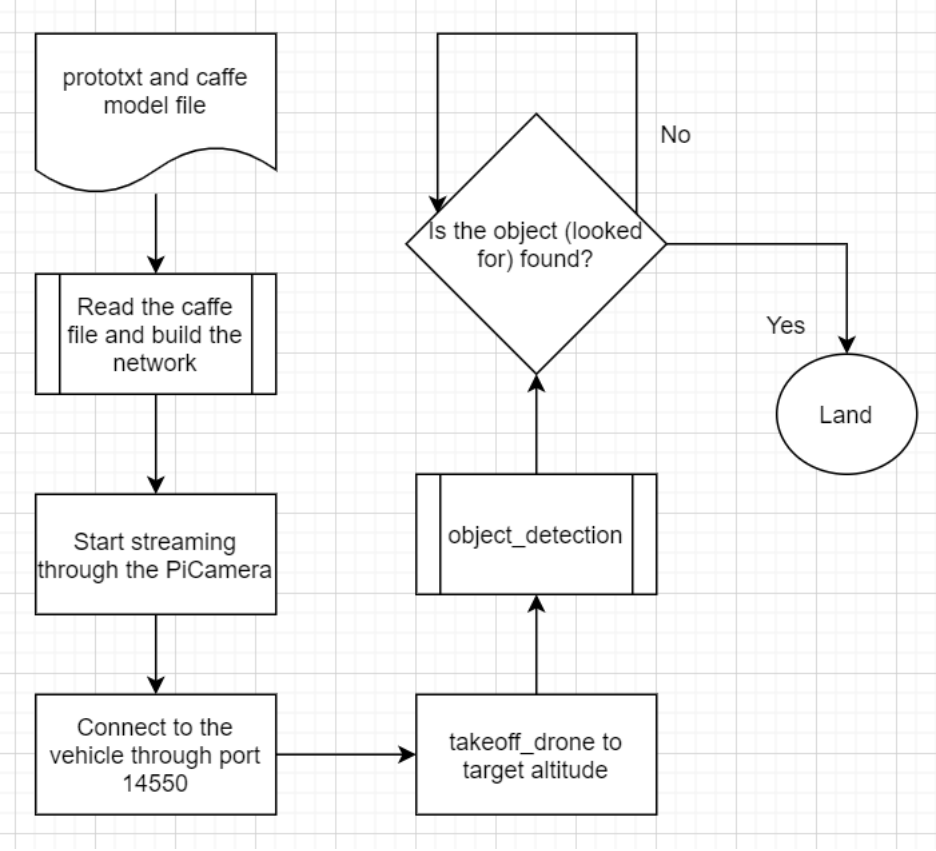
There could be two robust ways I thought of to pass the footage taken on the picamera to the neural network. That is either onboard processing or possibly passing the footage taken on the picamera to my PC through some pipeline-based framework such as gstreamer and doing the processing on my PC.

Using gstreamer data loss could be an issue and I have never used it before hence couldn't picture how to contribute it to my code. Hence, I decided to process locally on the raspberry pi.

Steps were to:

1. Get OpenCV on the Raspberry Pi.
2. Get the Raspbian GUI on the Emlid image to see if the picamera streams.
3. FTP the python file (my main solution code) from my PC, where I developed it, to the Raspberry Pi.
4. Test Run on the Pi.

10.4 Full Solution (Object Detection + Dronekit)



```
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import cv2
from dronekit import connect, VehicleMode, LocationGlobalRelative
import time

# arguments for user friendliness
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
                help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
                help="path to Caffe pre-trained model")
ap.add_argument("-c", "--confidence", type=float, default=0.25,
                help="minimum probability to filter weak detections")
args = vars(ap.parse_args())


Classes = ["background", "aeroplane", "bicycle", "bird", "boat",
           "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
           "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
           "sofa", "train", "tvmonitor"]

# draw samples from a uniform distribution of colors to set to each class's bounding box
# numpy.random.uniform(low, high, size)
Colors = np.random.uniform(0, 255, size=(len(Classes), 3))

# load prototxt and the caffe model
print("[INFO] loading model...")

# Now I can build the CNN (network) with the files prototxt and caffemodel in path
# using the cv2.dnn() module I can use the method where I read from Caffe
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

#start up the pi camera stream and read frames ps
print("[INFO] starting video stream...")
vs = VideoStream(usePiCamera=True).start()
time.sleep(2.0)
fps = FPS().start()

# connect to the drone through a udp connection (port 14550)
print("Connecting to vehicle...")
vehicle = connect('0.0.0.0:14550', wait_ready=True)


# Function to arm and then takeoff to a user specified altitude
def takeoff_drone(aTargetAltitude):
    print("Doing Pre-Arm checks...")
    # Don't let the user try to arm until autopilot is ready
    while not vehicle.is_armable:
        print(" Waiting for drone to get ready...")
        time.sleep(1)

    print("Arming motors...")
    # Copter should arm in GUIDED mode
```

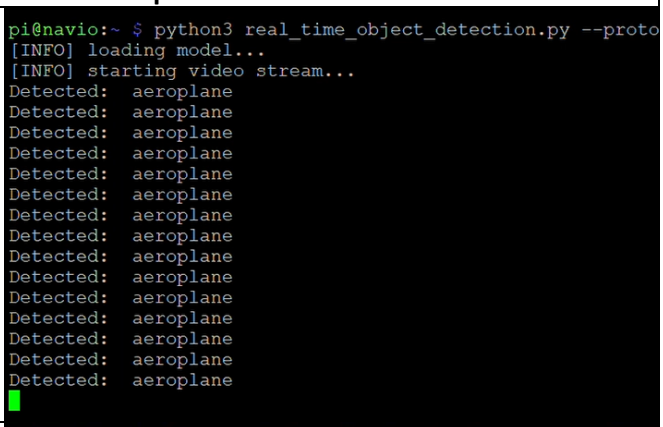
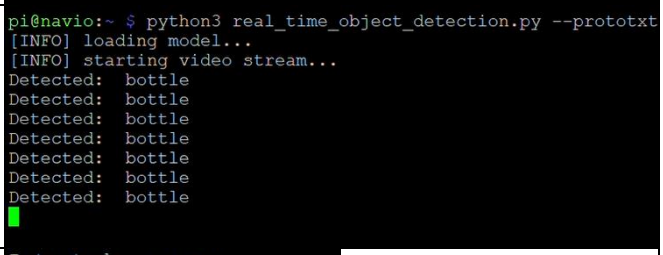
```
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    while not vehicle.armed:
        print(" Drone is arming...")
        time.sleep(1)

    # Take off to target altitude
    print("Taking off!")
    vehicle.simple_takeoff(aTargetAltitude)

    # Check that vehicle has reached takeoff altitude
    while True:
        print(" Altitude: ", vehicle.location.global_relative_frame.alt)
        # Break from function if current altitude is >= target altitude
        if vehicle.location.global_relative_frame.alt >= aTargetAltitude * 0$
            print("Target Reached!!")
            break
        time.sleep(1)


def object_detection():
    while True:
        # main loop
        # read frames and resize each to the exact same dimensions
        frame = vs.read()
        frame = imutils.resize(frame, width=400)

        # Slice the dimensions of the frame to get height and width
        # produce a blob from the input frames
        (h, w) = frame.shape[:2]
        blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
                                     0.007843, (300, 300), 127.5)

        # blob is the input into the CNN producing the output detections
        # detections is what the CNN forwards / outputs from the blob
        net.setInput(blob)
        detections = net.forward()

        # looping through the detections again with the .shape() attribute to return the
dimensions
        for i in np.arange(0, detections.shape[2]):

            # get confidence of detection from the array
            confidence = detections[0, 0, i, 2]


            # if confidence of the detection is greater than the default confidence
argument then run
            # have a 25% threshold for detections
            if confidence > args["confidence"]:

                # idx is the class label of the detection
                # multiply the detection array with the width and height array
                # produce a bounding box of type int with the coordinates produced as the
multiple of the detection
                idx = int(detections[0, 0, i, 1])
                print("Detected: ", Classes[idx])
```

```
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])


                # These will all be string values hence get them as integers
                (startX, startY, endX, endY) = box.astype("int")

                # draw the prediction on the frame
                label = "{}: {:.2f}%".format(Classes[idx],
                                             confidence * 100)

                # draw the box around the object
                cv2.rectangle(frame, (startX, startY), (endX, endY),
                              Colors[idx], 2)
                y = startY - 15 if startY - 15 > 15 else startY + 15
                cv2.putText(frame, label, (startX, y),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, Colors[idx], 2)
                # If a certain object is seen stop flying...
                if idx == 1:
                    print("Detected: ", Classes[idx])
                    break



        # if the q is pressed break loop
        key = cv2.waitKey(1) & 0xFF
        if key == ord("q"):
            break

        # update the FPS counter
        fps.update()


        # cleanup
        cv2.destroyAllWindows()
        vs.stop()
    return True


# Take off to 5m and land if object 8 is detected.
takeoff_drone(5)
if object_detection():
    print("Take off complete")

    # Hover for 10 seconds
    time.sleep(10)

    print("Now let's land")
    vehicle.mode = VehicleMode("LAND")

    # Close vehicle object
    vehicle.close()
```

Final Testing

11 Test the Object Detection program on the drone

Here I will test whether the PiCamera on the drone processes frames and detects objects properly.

Object	Video	Console Output
Aeroplane	https://www.youtube.com/watch?v=49EQ1J-mv7A	
Bottle	https://www.youtube.com/watch?v=WeXWb8N7WTw	
Person	In link above: recorded after bottle.	

1. **Airplane Detection:**


```
pi@navio:~ $ python3 real_time_object_detection.py --prototxt MobileNetSSD
[INFO] loading model...
[INFO] starting video stream...
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
Detected:  aeroplane
```

** The rest are in video form.

12 Final test (Mission)

As mentioned during the Analysis and Interview section a drone that sees can be used as a pair of eyes for places humans can't remain at.

I am not able to test at such a place but I can test the main functions I programmed the drone to carry out.

I will fly the drone to a certain altitude and get it to report what it sees to me through the Terminal.

13 End-User Testing and Feedback

Evaluation

How well does the solution meet the requirements of my scope and objectives?

My scope briefly, was a 'seeing' autonomous drone. The way I thought of it and structured my solution consisted of letting a Python script control the drone instead of me and getting it to perform a task for me such as detect the objects surrounding it and letting me know about it then doing something else – like landing if you see a person ahead.

I believe I have met these requirements in the sense that the drone does what I wanted it to do.

In terms of my objectives:

My first objective was the creation of a prototype quadcopter. I was successful in the creation, however, not successful on the long-run with my component choices although my scope and objective were clearly set out to be 'running an object detection program on the quadcopter - during flight.' My component choice that ended up creating a problem was the flight controller. However, this was meant to be a prototype to get a start on the project. I managed to produce a vehicle that did what it was supposed to do – fly.

My second objective was a representation of the success of my first objective – I had to fly the prototype manually. For this, I was communicating with the drone through a transmitter using RX/TX protocols (PWM). During this objective I came across the first couple of issues out of the million throughout this project. I fixed each one by one and developed skills in communicating my issues through a community forum of developers working on projects in the same space as mine. I was successful, in the end, and achieved this objective with a recording of the drone's brief first few flights.

My third objective was a fix of the problems the first objective's slip up caused – a flight controller with limited functions. I decided to make a new design and implementation section after I documented the problem my flight controller caused my covering of the entire scope that I had set myself. I introduced a new flight controller that would fix this problem then I designed my solution to the idea of arming the drone and taking off through a python script, without human intervention. I also introduced new software (library) for this solution and carefully tested the finished code individually before moving onto adding the main feature – which was a means of perception for the drone in the form of 'sight'.

My fourth objective was building a neural network and running an object detection program on board (on the Raspberry Pi). I added another Design & Implementation section for this where I investigated the Caffe framework and found a suitable trained dataset instead of building the network and training the images myself. The pre-trained Caffe model was already robust and had a pretty good accuracy score. I also familiarized myself with the dnn module in OpenCV which would help me develop my code using the Caffe framework. The creator of the framework had provided a good amount of documentation and there happened to be many Dummies books available on these topics. I clearly documented how the framework stores data and creates detections from a passed in frame/image. I provided many comments throughout my code and made it followable. I tested the finished solution individually before producing a single system where I would run this program on the Raspberry Pi directly. On-board processing was tricky mainly due to the environment not being easy to setup (OpenCV) and a lack of experience with the Linux command line. The SD card was full on my first try of setting up the environment and stopped compiling the OpenCV build halfway. I changed SD

cards quite a lot and encountered countless problems when building OpenCV on my Raspberry Pi – also the fact that the Emlid Raspbian image (image provided by the company that produced the new flight controller I purchased for objective 3) didn't provide a GUI wasn't helpful in the start either. After I built the environment to run my object detection program in it was easy to test the code finish my objective successfully. Overall, the object detection solution is contained as a function that returns True if a specific object is detected.

This helped me reach my objective of producing a single system that flies and detects objects at the same time since I was able to take the 'one program hypothesis' approach and write a short selection statement that cleared the entire scope seamlessly. Since I contained takeoff and object detection as functions, I was able to produce a logic where the drone would fly to an altitude and land if it saw a certain object.

My fifth objective was testing the success of my scope – 'autonomous flight'. In this section I individually tested the takeoff function where I armed the motors through my solution for objective 3 and then proceeded to select a test where I armed the motors and placed an object in front of the camera to see if the motors stopped, without a target altitude – this test solely depended on testing whether the drone was able to see and act upon it.

Peer Review (Drone community user feedback)

'I like the 'one program hypothesis' and I like that you were trying to create a simple solution to a complex task. I think you reached your scope successfully and documented the problems you encountered properly, which I think is important for development. I do think you can expand on the scope in the future and add more perception features such as depth perception so that the drone is able to tell where the object is relative to its position as well as what the object is. I do think adding altitude readings and precise GPS locations covers that perception partially though. Overall, I liked your approach to the scope you set initially though.'

I agree with the comments made here since I did in fact talk about control and perception for objective 3 where I disclosed the development of software for such robotics systems. I am looking forward to add such features in the future when I expand on this project.

How can I improve the solution?

Here are some evaluations of what I could have improved:

- UDP over TCP

For the connection of the drone to the Python Script I used a UDP connection through port 14550 (specified as the MavLink GC Port)

Port(s)	Protocol	Service	Details	Source
14550	udp	applications	MAVLink Ground Station Port	*SG*

UDP is a connection-less protocol which means that it doesn't establish a connection before sending data. UDP doesn't have error-correcting, unlike TCP, hence acknowledgements/handshakes aren't waited for – if the recipient misses a packet the sender won't resend. The overall advantage of UDP, however, is that the sacrifice made for reliability is all weighed down to speed. Since acknowledgement isn't waited for devices communicate more quickly, which is important for the drone.

In fact, the connection of the drone to the GCS is with TCP – this still has a limitation on its own as a WAP usually has a radius coverage of 45m indoors and up to 90m outdoors.

- Frames per second with the Pi Camera/ low resolution

Overall, the resolution of the Pi Camera is just about adequate. Its sensor has a resolution of 5 megapixel, and supports 1080p @ 30fps, 720p @ 60fps and 640x480p 60/90 video recording. This is fine for my project however I am aware that it should be much quicker and demanding for real life applications. The drone must be able to process and decide quickly during flight to serve its purpose properly.

The Raspberry Pi Camera Module uses the onboard GPU and its memory. The recommended minimum is 128MB to the GPU's allotment. I changed the memory allotment to 256MB for the object detection feature.

- Things that can be done to speed up processing of the RPi:

1. **Use a smaller dataset**

Yolo-LITE/Tiny-YOLOV2: a real-time object detection model developed to run on portable devices such as a laptop or cellphone lacking a Graphics Processing Unit (GPU).

Based on the original object detection algorithm YOLOV2, YOLO- LITE was designed to create a smaller, faster, and more efficient model increasing the accessibility of real-time object detection to a variety of devices.

Model	mAP	FPS
Tiny-YOLOV2	**23.7%**	2.4
SSD Mobilenet V1	21%	5.8
YOLO-LITE	12.26%	**21**

2. **Jetson Nano Processing**

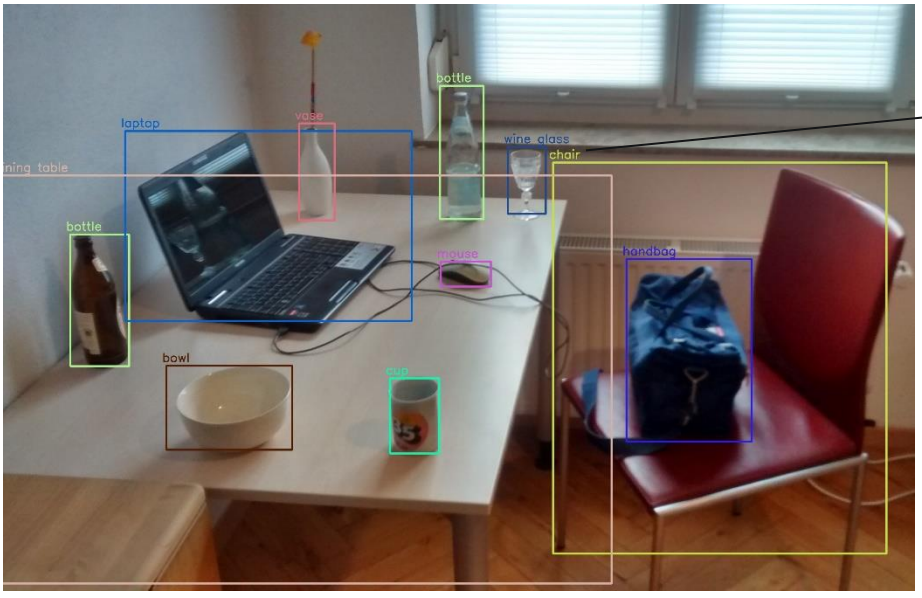
Jetson NANO is a board with a quad-core 64-bit ARM CPU and a 128-core integrated NVIDIA GPU. It also includes 4GB LPDDR4 memory. For even a larger scale processing of maybe enormous datasets as well as quick processing onboard this board is very suitable.

The newly released JetPack 4.2 SDK provides a complete desktop Linux environment for Jetson Nano based on Ubuntu 18.04 with accelerated graphics, support for NVIDIA CUDA Toolkit 10.0, and libraries such as cuDNN 7.3 and TensorRT 5.The SDK also includes the ability to natively install popular open source Machine Learning (ML) frameworks such as TensorFlow, PyTorch, Caffe, Keras, and MXNet, along with frameworks for computer vision and robotics development like OpenCV and ROS.

An overall idea that could take this project to a more user-friendly product:

I could provide a user interface that integrated all the functionality into a single app.
I am familiar with such applications from the Robotics club at my college where we used the 'Parrot' app providing full control of the vehicle through an interface looking as such:

My implementation of this could be a similar interface with the object detection occurring at the same time. With the use of a 4-axis proximity sensor the distance of the detected object relative to the drone could be displayed on the bounding box label. E.g.:

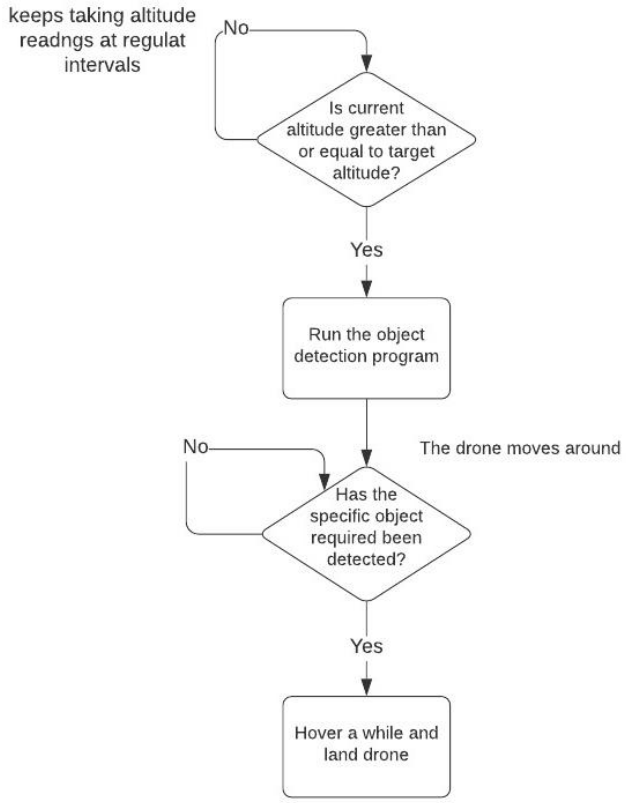


chair
97% (confidence)
1.5m (distance relative proximity sensor)

Highlights of the solution

The solution to this system consists of the put together hardware and the software that runs on it to control and provide perception to the system to ultimately deduce suitable decisions upon the percieved data collected from the drone's current environment.

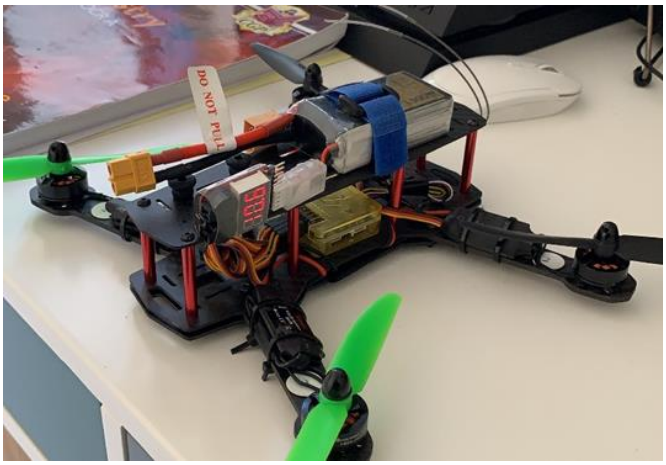
The main, brief flowchart of the solution (repeated here):



To produce this simple program:

HARDWARE:

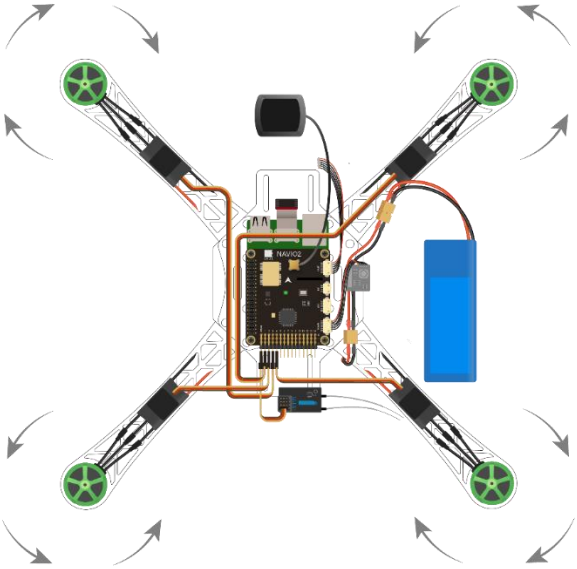
Prototype:

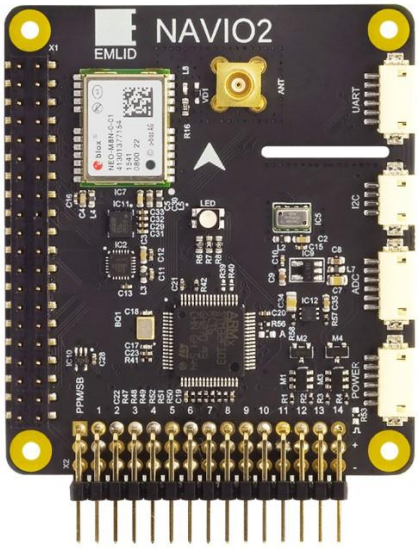


The CC3D flight controller was my initial control system in the first design of the quadcopter. Consists of a 32-bit microcontroller and a 3-Axis Gyrometer + Accelerometer (MPU6000).
The main issues are:

- Operates at a different voltage level than the RPi GPIO pins, which can be fixed by making a voltage dividor circuit.
- The Raspberry pi 3B+ only has 2 UART pins + not sure if the CC3D is compatible to communicate with the Raspberry Pi.
- If I ended up communicating them – I couldn't use a GPS/GNSS receiver and neither does it have a barometer hence I cannot take altitude readings hence the drone is likely to experience noticable drift during flight which makes it very unstable and dangerous.

Final Build:





What this final build fixed and provided:

- Direct ‘hat’ that sits on top of the Raspberry Pi with no voltage level difference between the boards’ IO pins.
- Has a barometer that can sense altitude with a 10cm resolution.
- Has a dual IMU unit (Gyro+Acclerometer)

The final build is a drone ready to run various programs on a powerful companion computer. The final product was hard to reach but it is exactly what this project needed.

Candidate No: 5034 | Center No: 11001

SOFTWARE:

- **Advanced matrix operations**

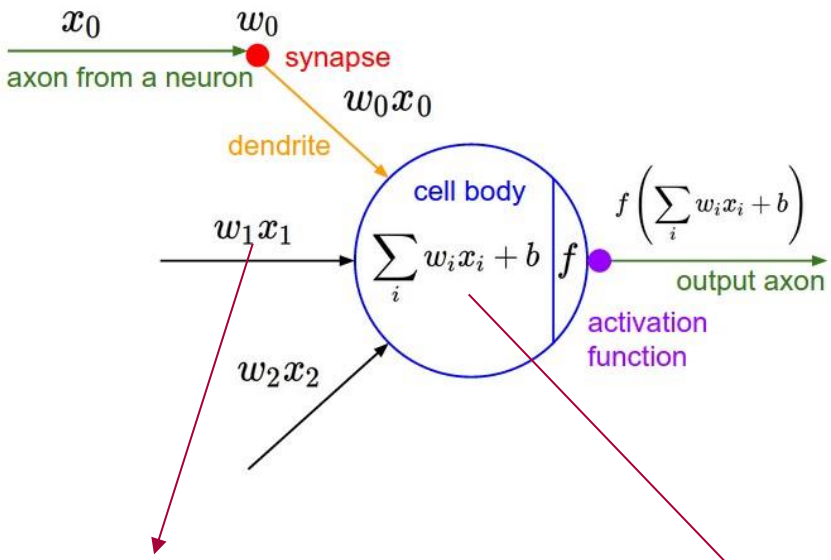
In the Caffe framework I worked with blobs (N-dimensional array stored in a C-contiguous fashion).
Passed in a 4 dimensional array into the network and the detections outputs a 4 dimensional array such that:
Detection[0, 0, i, x]
Where i is a 7 dimensional array accessed though x (0-6):

```
     0.          0.        ]]]]
detection:  [[[[ 0.         15.         0.9993051   0.12256578  0.24431014
     0.850332    1.0054094 ]
```

What happens is:

$$f(x_1, x_2) = w_1 x_1 + w_2 x_2$$

The inputs are multiplied by their weights and summed. The sum of the products of an input and its weight is multiplied by an activation function.



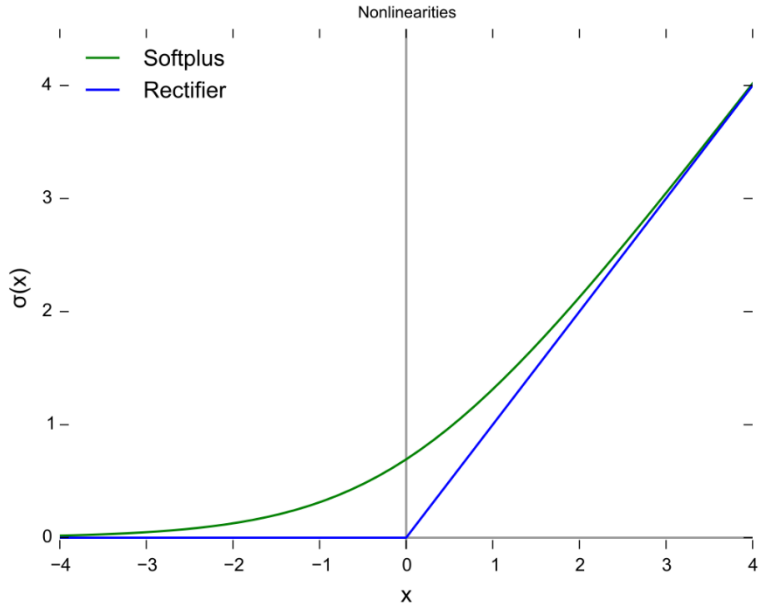
CONVOLUTION:

The kernel – consists of the weights of the inputs.

Output: Weighted sum

The blue matrix of the inputs are multiplied by the 'kernel' consisting of the weights of the inputs – the weighted sums are the output value.

The activation function, commonly, for the Caffe Framework is ReLU - **rectified linear unit.**



The softplus is :

$$f(x) = \ln(1 + e^x),$$

Which is an approximation to the recrifier:

$$f(x) = x^+ = \max(0, x),$$

Where x is the input to the neuron.

This means that the rectified linear activation function will return 0 if the input is 0 or less, and returns the input itself if the input is greater than 0.

Could also be briefed as:

```
if input > 0:
      return input
else:
      return 0
```

The output that comes out of the network is the detection.

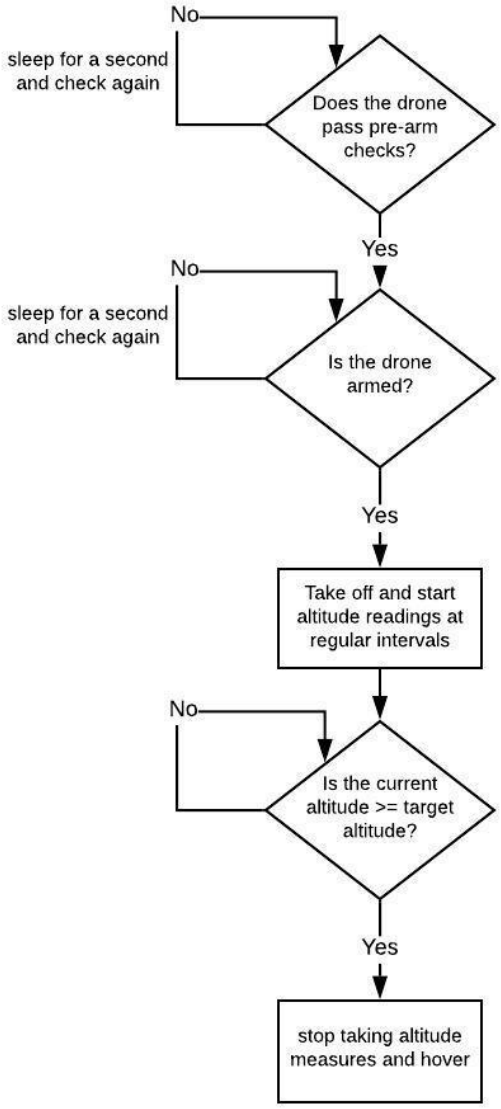
```
net.setInput(blob)
detections = net.forward()
```

- Loosely coupled modules (subroutines) – module code interacts with other parts of the program through its interface only.
- Cohesive modules (subroutines) – module code does just one thing

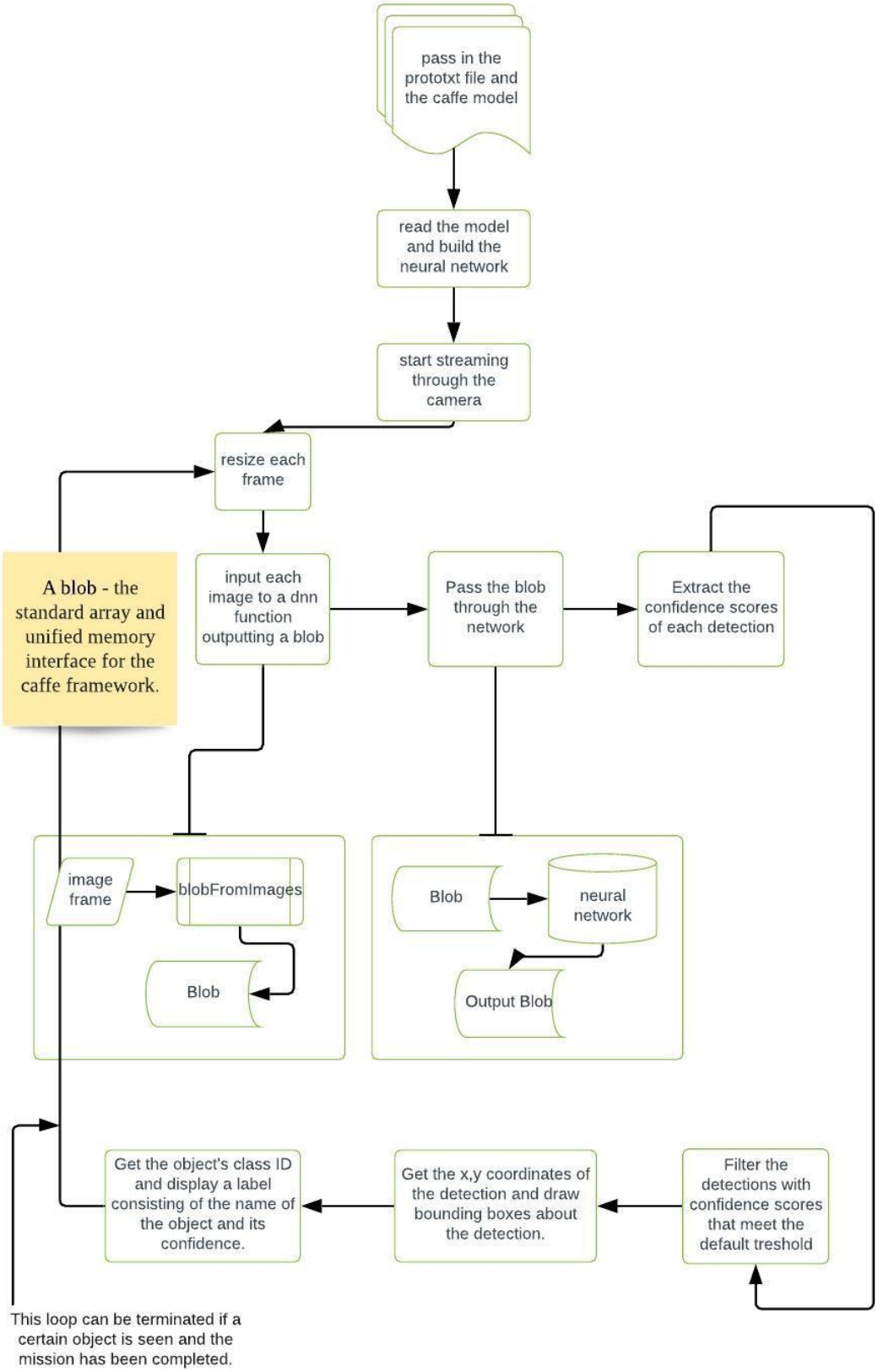
I have two separate functions doing a single thing – hence producing a simple 5 line program:

```
takeoff_drone(5)
if object_detection():
    time.sleep(10)
    vehicle.mode = VehicleMode("LAND")
    vehicle.close()
```

Takeoff:



Object Detection:



I created a website with posts about the entire progress – decided to also sort of make a business out of my hobby if I can. Please check it out – it does have content that will help understand the process such as building the environment to develop my object detection code for on board processing on the Raspberry Pi and building drones etc…

Link: http://dronebuilderexperts.com/