# Gold Crest Project: Gesture-Controlled Drone

Joshua Killa

# Contents

# Introduction

The rationale behind this project was to set out and build a physical device involving microcomputers, machine learning and a flying contraption. This combined my interests in engineering and programming, so I narrowed down the project idea to the making of a gesture-controlled drone prototype. I have worked on convolutional neural networks before, but this time I will build my own dataset from scratch and optimise the learning model to recognise gestures to instruct a drone to move around an environment. Machine learning has become a highly applicable skill and I aim to improve my ability in this field, by using computer vision in this project. Through the designing, modelling, and prototyping of a drone system that responds to gestures, I aim to learn more about robotics and the principles of engineering and computing.

# Plan

- [2 hours] Research the parts involved for the gesture recognition device
- [3 hours] Build the gesture recognition device and 3D print necessary parts
- [2 hours] Research and design the gestures that will be recognised by the gesture recognition device to control the drone
- [2 hours] Write iterative photo capturing python script that will save and filter the image files
- [2 hours] Use the python script to capture photos of the gestures to create a training, validation and testing dataset and record the horizontal distance from the camera to the subject and the vertical from the ground to the camera
- Transform the images into tensors and resize them
- [3 hours] Research about the deep learning model I will use and about neural networks
- [6 hours] Make a neural network using a python deep learning library
- [2 hours] Train the neural network model
- Save the model and its hyperparameters (inference data)
- [2 hours] Load it and test it
- [5 hours] Write the live script that takes frames from the camera and displays the prediction in a window
- [2 hours] Research how drones work
- [4 hours] Research parts involved for a python-controlled drone
- [3 hours] Write a spreadsheet of costs and order parts
- [30 min] Order parts
- [3 hours & 30min] Design and 3D print the necessary parts and build the drone
- [6 hours] Assemble the drone
- [2 hours] Setup the controller and drone calibration
- [2 hours] Use the live script to send commands to the drone
- [18 hours] Write a report on the research, design, steps taken and evaluate the final prototype
- [2 hours] Add bibliography and format the document

# Initial Part List:

Gesture recognition device

- Raspberry Pi 4 B (4GB)
- Camera Module B
- Anker PowerCore II 10000mAh (portable charger)
- Anker PowerLine USB 3.0 to USB C cable (3ft)
- The Official Raspberry Pi Camera Guide
- 3D print case
- 5inch LCD 800*480 Resistance screen

Drone

- Electronic speed controllers
- Flight controller
- Sensors
- Motors
- Propellers
- Drone frame
- Radio telemetry kit
- Breadboards
- Jumper cables

# Definitions:

Raspberry Pi: a low-cost, single-board microcomputer the size of a credit card

Microcomputer: A small computer that contains a microprocessor as its central processor

Microprocessor: An integrated circuit that contains all the functions of a central processing unit of a CPU

Convolutional Neural Network (CNN): A deep learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various features in the image and be able to differentiate defined classes of images from one another thus performing well in computer vision problems like image classification and object detection.

Deep learning: A type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher-level features from data.

Machine learning: The use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data.

Hyperparameters: In machine learning, a hyperparameter is a parameter value used to control the learning process (mini-batches, learning rate, layer widths etc)

Virtual environment: A tool used by programmers that helps to keep dependencies required by different projects separate, by creating isolated (Python) virtual environments for them.

Training data: The data you use to train an algorithm or machine learning model to predict the outcome you design your model to predict.

Validation data: Provides an unbiased evaluation of a model fit on the training data set while tuning the model's hyperparameters.

Scoring: Also called prediction and is the process of generating values based on a trained machine learning model, given some new input data.

Colour depth: the number of bits (smallest unit of data storage) used to indicate the colour of a single pixel.

Input channel: The number of channels used for input. Grayscale images have 1 channel and colour images have 3 channels. The number of channels of a tensor changes after each layer of the neural network is applied to it.

Yaw: The twist or oscillation about a vertical axis.

Pitch: The rotation of the aircraft around the lateral axis (side to side). It can be thought of as the “up and down” or “nodding” motion of the aircraft.

Roll: The rotation of a vehicle about the longitudinal axis.

Gesture: a movement of part of the body, especially a hand or the head, to express an idea or meaning.

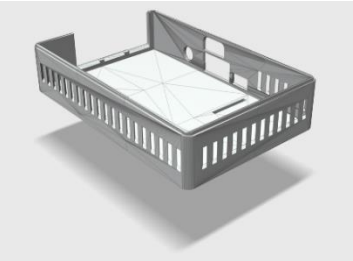
Contrast: Contrast is the degree of difference between two colours or between the lightest lights and darkest darks in an image.

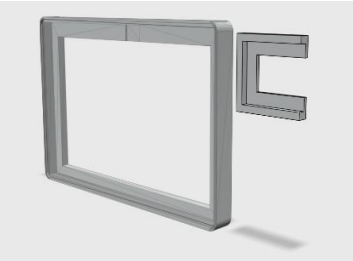
## Gesture Recognition Device Assembly

I measured the dimensions of the Raspberry Pi 4B, LCD touchscreen and PiCamera module and designed a case on Autodesk Fusion 360. This was then printed on one of the DT department’s 3D printers at Hurstpierpoint College.

The case had gaps for air intake, so the microcomputer does not overheat, and it has a slot at the back for the data tape to thread through. The case firmly holds the LCD touchscreen and the PiCamera in place, at the back, angled to record what is directly behind the screen. The case has gaps for where there are ports, allowing the mini-HDMI bridge to connect the screen to the Raspberry Pi 4b, the USB-C cable to connect the power bank to the Raspberry Pi 4B and spaces for the USB-A ports.

## 3D model

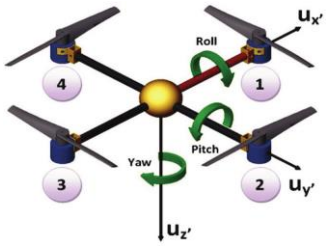


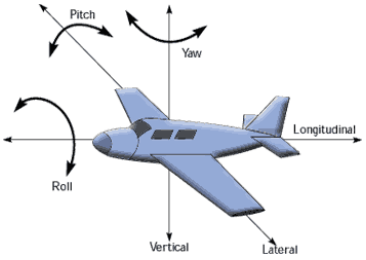


## Pictures of Assembly

# Gesture Research

The controlling of the drone will be based around yaw (left, right), vertical (up, down) and pitch (forward, backwards) movement.



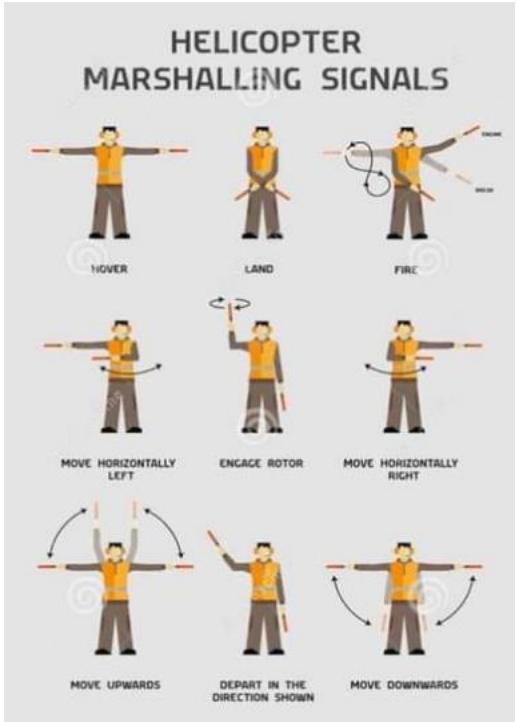

Since gestures are being used to control the movement of the drone, this means that 6 gestures should be defined to enable the drone to move. Since the drone is either moving or not moving it should also know when to hover (remain in a stationary position and stop moving) this can be defined as a 7th gesture or when no gestures can be identified (a null value). This decision will be made during the programming stage.

The gestures must be distinctive and easily identified from each other and if max certainty is below 50% then a null value could be assigned (instructing the drone to hover).

The gestures must be made by the arms and/or hands. Therefore, they can be inspired from a form of aircraft marshalling, a martial art, or a form of sign language.

## Airplane Marshalling


## Helicopter Marshalling




The neural network will need to output many results a second so it will take static feed as an input rather than processing video feed, therefore it will not be able to recognize actions only static forms. Additionally, the images will be black and white so there will be little depth to

them so any appendages that are overlapping the torso will prove impossible for the CNN to recognize.

The signals (gestures) I have created are shown on the next page as *drone marshalling signals*. They do not necessarily always correspond to the marshalling used for helicopters and airplanes as the recognition device will be limited to static inputs and the signals must be very distinct from one another considering that there is no depth perception, since the input images will be represented in a black and white format.

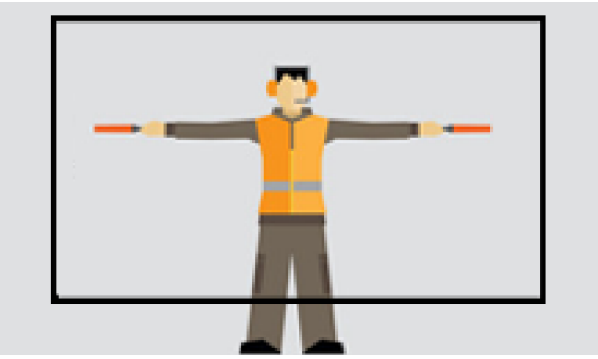
### Drone Marshalling


## Recording the Data

The photos making up the training dataset will be taken on the Raspberry Pi's camera module as it is what will be taking inputs for the CNN once the model is completed and trained.

To make the dataset, 750 photos for each gesture will be taken, the Raspberry Pi has a Python script that will automatically capture 6 images every second once it is run. The 640 X 480 images will be converted into black and white (1 channel), have a Gaussian Blur filter and binary threshold applied, and reduced to 150 X 150 pixels to simplify the image data (array) that will train and be tested on the CNN. The purpose of this is to reduce the input data size so the CNN can be trained quickly and so it can process and output many results a second while running. The images will also be flipped vertically so the subject's left arm will be to the left of the image when displayed.

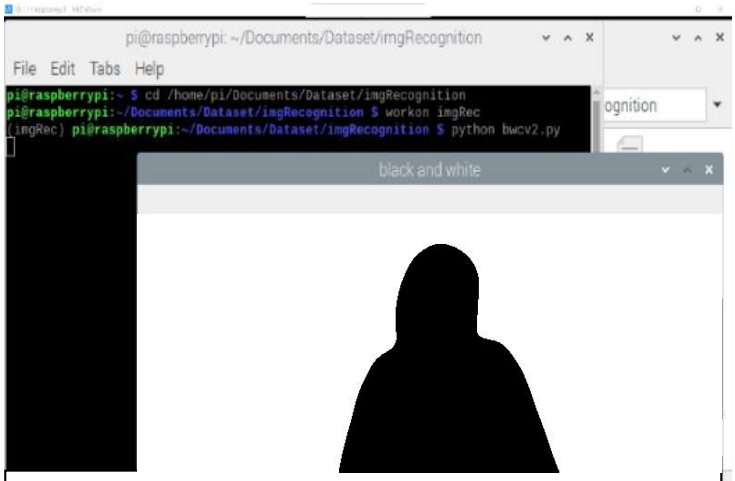
I need to add data variation since standing still will yield 750 identical images for each gesture. To add variation, I will perform subtle movements by acting out the gestures from the aircraft and helicopter marshalling, swaying side to side and by moving backwards and forwards so that every input will be different. I will do this so every input can be represented and to prevent the neural network from being overfitted to my training data.



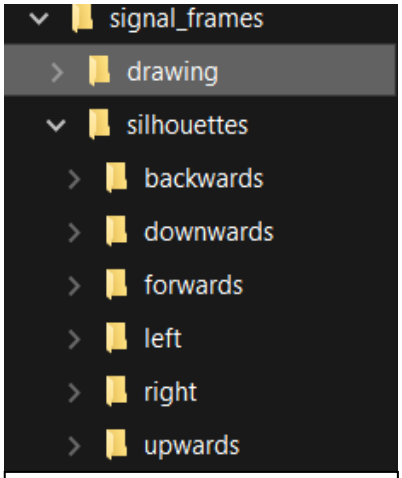
I will stand 3.25 meters from the camera module (1.39m above the floor) so that the width of the images is 25% wider on either side of my wingspan with 25% of the image canvas above

my head and the lowest part that the image captures is my lower torso (illustrated in the picture above). I do not fill the entire canvas because convolutional neural networks identify features better when they are close to the centre as the kernel will pass over central features more times.

My body needs to have high contrast from the surrounding image so when the colour depth is reduced (to black and white) they are the opposite colour to the surrounding image. This can be done by wearing black gloves and a black jumpsuit/hoodie in front of a white backdrop.




Virtual environment activated and python script run (black and white window) on the Raspbian terminal. Pi controlled on laptop via VNC.




Where image silhouettes are saved locally.

The python script is coded on Thonny editor through my computer connected to the raspberry pi via VNC. The script is run on a virtual environment called imgRec which contains the NumPy and OpenCV libraries and it is activated on the terminal. I use OpenCV to convert the live feed of images from the camera into image arrays so that they can be manipulated.

## Gaussian Function

Once I have obtained the image array, the Python script converts the three-channel (colour) image to one channel (grayscale). To get rid of insignificant image features like small shadows in the image, I use the Gaussian blur, which is often used to reduce image noise and detail. This image processing technique applies the Gaussian function (which expresses normal distribution in statistics) to the image array. Below is the equation of the Gaussian function in two dimensions, expressing the transformation of each pixel in the image array.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}}$$

x represents the distance from the origin on the x-axis, y is the distance from the origin on the y-axis, and σ is the standard deviation of the Gaussian distribution. When the Gaussian blur is applied in two dimensions, this function produces a surface whose contours are concentric circles with a Gaussian distribution from the centre pixel. Values from this distribution are used to build a convolution matrix, which is applied to the initial image. Each pixel's new value is set to a weighted average of that pixel's surroundings. The original pixel's value receives the heaviest weight (having the highest

Gaussian value) and surrounding pixels receive smaller weights as their distance to the original pixel increases.

After the blurring technique, I apply a thresholding process to create a binary image from the grayscale image as this will get rid of the details but keep the general shape of the object. This process of frame segmentation effectively separates the object in focus from the background thus generating a silhouette of my body acting out the gestures when applied in my instance.

When the save_images variable is True, the silhouettes (the filtered image arrays are effectively silhouettes of the original image) of the images are saved into folders locally. Each silhouette is named after their assigned signal and is given a number. For example, the 750$^{th}$ (last) image silhouette of the left gesture will be named left_750.

When save_images is False, no image silhouettes are saved locally and this was used when positioning the camera and making sure it was operational. Once the neural network is complete the model will be run under the if condition when save_images is False, and predictions of the detected drone signal will be outputted.

Using a stopwatch, I calculated that I worked out that the python script can save 856 silhouette images in 2min 23 seconds. Therefore 6 are saved in a second and the script would have to run for 125 seconds (2min 5s) to save 750 silhouettes. After the stopwatch reached 2min 5 seconds I reached for the c key on my keyboard which was programmed to end the python script, to stop images from being saved. Then on the python script, I would change the selected_gesture variable from eg forwards to backwards and then run the script to record 750 silhouette images for the backwards class.

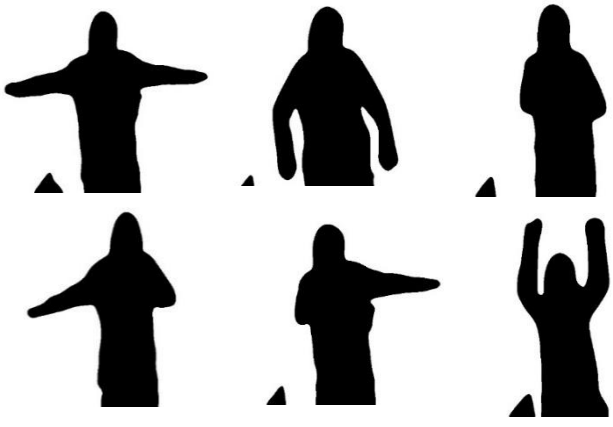
## Data Capture Program

*Below is the Python script that captured the silhouette images (training images) on the gesture recognition device (Raspberry Pi)*

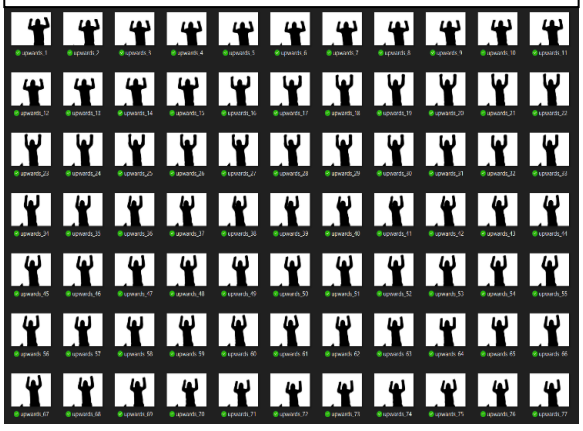
```
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import cv2
import copy
import numpy as np

# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = (640, 480)
camera.rotation = 270
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480))
#camera.contrast = 85
#camera.brightness = 55

#parameters
threshold = 60  # binary threshold
blurValue = 41  # GaussianBlur parameter
bgSubThreshold = 50
save_images = True
selected_gesture = "forwards"
img_counter = 1
# allow the camera to warmup
time.sleep(0.1)

# capture frames from the camera
```

```
for frame in camera.capture_continuous(rawCapture, format="bgr",
use_video_port=True):
    # grab the raw NumPy array representing the image, then initialize the
timestamp
    # and occupied/unoccupied text

    image = frame.array
    image = cv2.bilateralFilter(image, 5, 50, 100)
    image = cv2.flip(image, 1)

    # show the frame
    grayImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(grayImage, (blurValue, blurValue), 0)
    #(thresh, blackAndWhiteImage) = cv2.threshold(grayImage, 480, 640,
cv2.THRESH_BINARY)
    ret, thresh = cv2.threshold(blur, threshold, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

    cv2.imshow('black and white', thresh)

    key = cv2.waitKey(1) & 0xFF

    # clear the stream in preparation for the next frame
    rawCapture.truncate(0)


    # if the `c` key was pressed, break from the loop
    if key == ord("c"):
        break
    else:
        if save_images == True:

            img_name =
f"/home/pi/Documents/Dataset/imgRecognition/signal_frames/silhouettes/{selected_ges
ture} {img_counter}.jpg"
            cv2.imwrite(img_name, thresh)
            print("{} written".format(img_name2))
            img_counter += 1
```

## Captured Data



First 77 training images (silhouettes) of the upwards signal



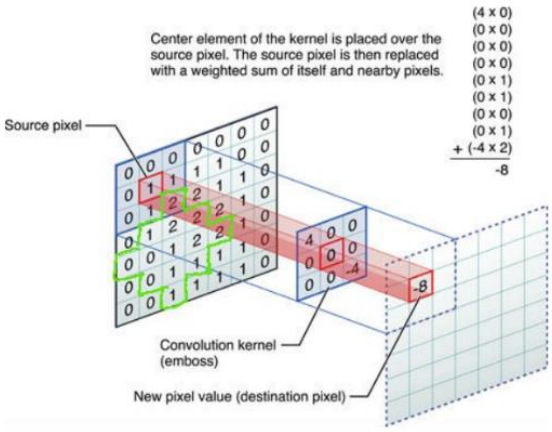
The iterative silhouette-capturing Python Script on the Raspberry Pi successfully captured 750 images per signal class ('backwards', 'downwards', 'forwards', 'left', 'right', 'upwards') and saved them into their respective folders. These were then uploaded onto the cloud (OneDrive) and saved locally on my laptop which had the computational capability to train a neural network.

# How Convolutional Neural Networks do Image Classification

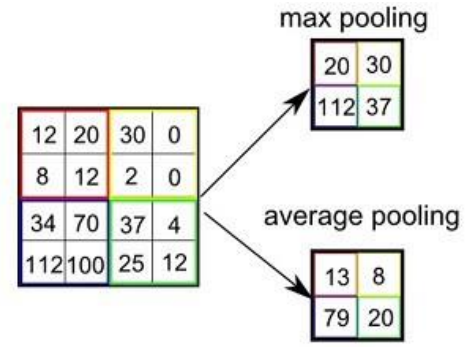
A convolutional neural network (CNN) is constructed with multiple convolution layers, pooling layers, and dense layers.

The idea of the convolution layer is to transform the input image to extract features (eg the position, shape and direction of my arms in the gesture images) to distinguish them correctly. This is done by convolving the image with a kernel. A kernel is specialized to extract certain features. It is possible to apply multiple kernels to a single image to capture multiple features.

Usually, an activation function (eg tanh, relu, or sigmoid) will be applied to the convoluted values to increase the non-linearity.




How kernel is applied to an image to extract features

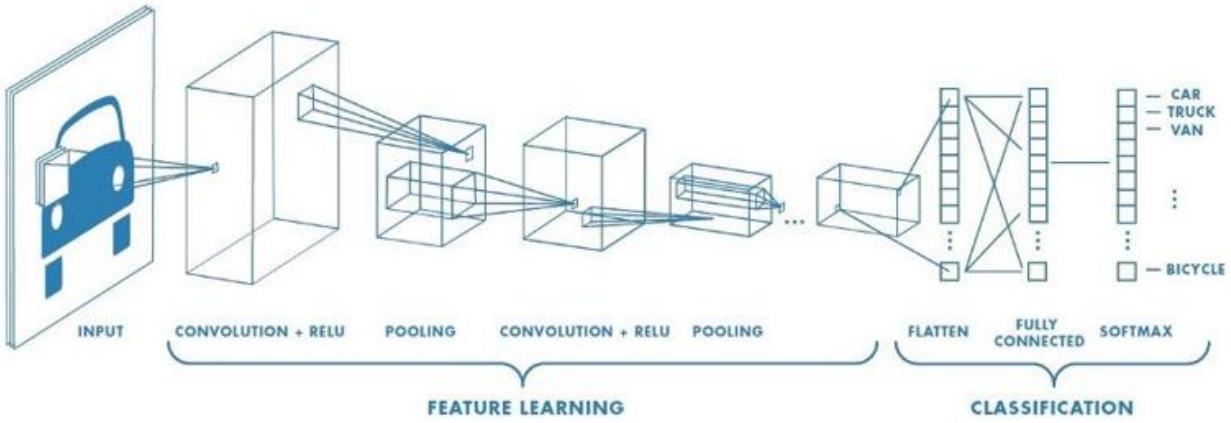



The job of the pooling layer is to reduce the image size resulting in a down-sampled (pooled) feature map. It will only keep the most important features and remove the other area from the image. Moreover, this will reduce the computational cost as well. The most popular pooling strategies are max-pooling and average-pooling.

The max-pooling operation calculates the maximum value for patches of a feature map and is often used for grayscale images.

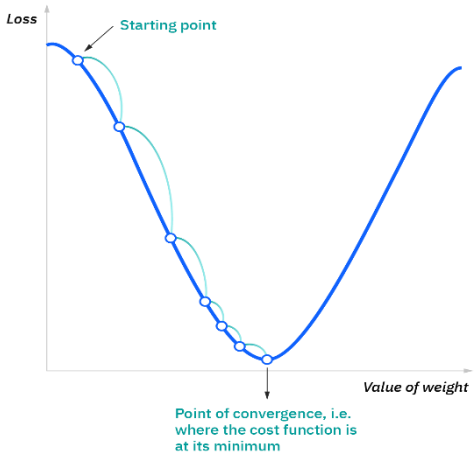
The average pooling operation calculates the average value for patches of a feature map and is often used for colour images. The pooling layer is used after a convolutional layer.

These series of convolution layers and pooling layers will help to identify the features and they will be followed by the dense layers for learning and prediction later.



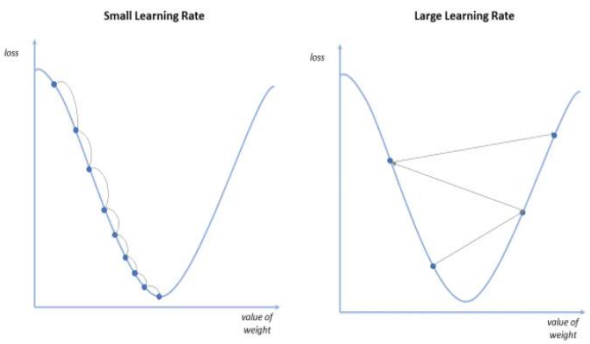
During training, neural networks use gradient descent, an optimization algorithm that helps models learn their training data over time. Batch, stochastic and mini batch are different types of gradient descent.

An oversimplified model of gradient descent can be achieved using a convex function *(shown on the right),* using loss (cost function) on the y axis and value of weight on the x axis.




The starting point is arbitrary and is usually determined by the intial randomly assigned weights. As the accuracy of the learning model increases and it advances from the starting point, we can find the derivative of the slope to observe the steepness.

The cost function within gradient descent acts as a barometer to gauge the model's loss and steepness after each iteration of hyperparameter updates. It measures the average mean squared error between the actual y (output) and the predicted y (output) across an entire training set. The model continuously iterates and generates new parameters moving along the steepest possible descent (negative gradient); however, the steepness gradually reduces over time, and the model eventually reaches a minimum on the curve, a point of convergence. At this point the cost function reaches zero, which could mean that the machine learning model's current state yields the smallest possible error and has become optimized for classifying the data provided by the training dataset, ending the training stage.

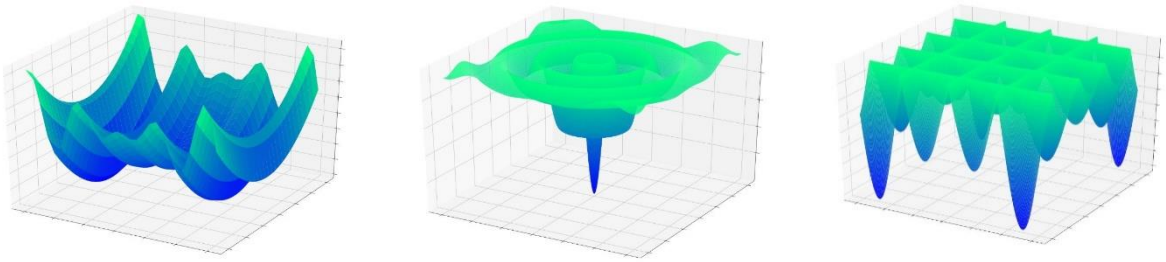



Therefore, the goal of training is to minimise the cost function (error between the predicted and actual value). To do this, the model needs a learning rate and a direction (provided by the cost function). These factors determine the partial derivative calculations for future iterations allowing it to arrive at the local or global minimum (points of convergence).

Learning rate is the size of the steps taken to reach a minimum. This is usually a small value as after each step the model is evaluated and updated based on the cost function. Higher learning rates take larger steps and risk overshooting the minimum. Although smaller learning rates offer greater precision, they take more iterations and time to reach a minimum.

Often throughout training, there are many points of convergence called local minima which will cause the gradient to become zero, however, the lowest point across the entire error function is called the global minimum. If there is only one minimum, no matter where the starting point is you will always reach the global minimum, but this is usually not the case. Therefore, depending on your starting point, you could end up converging towards a local minimum nowhere near the global minimum, which would stall the training process.

*Below are some three-dimensional representations of cost/error functions. Neural networks are not limited to three-dimensional relationships and usually have a dimensionality that exceeds two input variables and one output variable.*

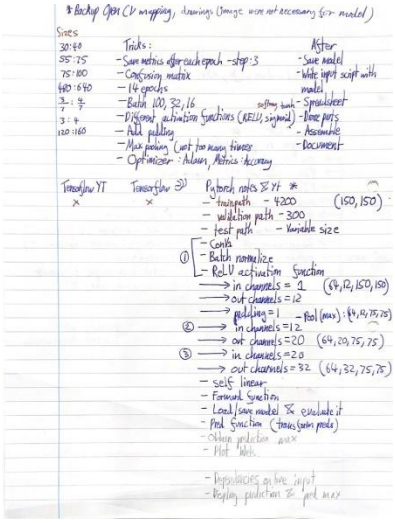


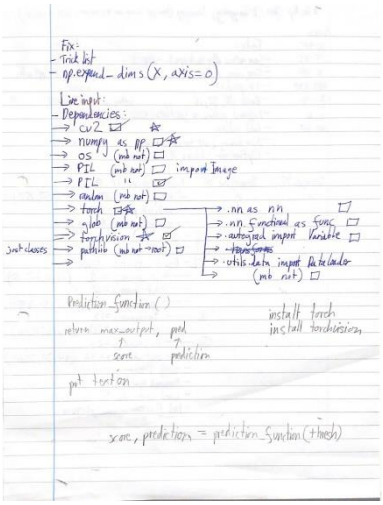
Neural networks have no prior knowledge of the overall error surface associated with its training data and relies on *local* information to reach a global minimum. To avoid converging towards local minima nowhere near the global minimum, gradient descent is implemented so that the model saves its hyperparameters when it reaches a minimum and then continues upwards until it goes down another minimum. This way the global maximum can be found and there are many different models to test on the prediction dataset.

Since the global minimum is the point where the model's hyperparameters provide the lowest point of error they theoretically represent the optimal state of the neural network, where it would be able to classify its training data to the highest possible accuracy. However, high complexity neural networks like convolutional neural networks usually don't benefit from global minima since they are *overfitted* to their training data and would be less effective when processing new inputs, making them obsolete. For this reason, it is important to test the accuracy of different convolutional neural network models on a prediction dataset that includes images that the models were not trained on to conclude whether they have been overfitted to the training data.

# Planning the Neural Network

*On the paper pages below, I planned out how I would go about training and testing the model.*





# Building the Neural Network

I installed the dependencies required for TensorFlow on a virtual environment on Anaconda (a Python distribution platform for scientific computing, that aims to simplify package management and deployment) and then booted JuypterLabs (a web-based development environment for programming in a notebook layout where code is written in blocks) using Anaconda. I passed the PATH (where the training data is stored) and loaded it into the notebook. The images were converted into NumPy arrays and labelled based on which folder they were stored in.

Next, I coded the convolutional neural network model's forward function, optimizer and then the training code block. The training accuracy was very poor, so I removed TensorFlow and installed the PyTorch which would allow me to have more control over the hidden layers.

After I adjusted the code for the PyTorch library, the training images were then resized to 150 X 150, transformed into tensors and trained on the model which yielded a maximum training accuracy of 99.97% and a maximum validation accuracy of 99%.

After that I saved the hyperparameters as best_checkpoint.pth and tested the model on images that had not been used in the validation and test data, which I had drawn in Paint 3D and the model still predicted their image class accurately.

*To see the notebook that trained the CNN see 'Training the Signal Recognition Model' under the Appendix.*

# Implementing the Neural Network

On the Raspberry Pi 4B, I created a Python script and used OpenCV to access the frames in the video capture as an image array. The image array was then filtered using the same functions that I applied to the captured the images for the training data. I copied the model into the Python script and loaded 'best_checkpoint.pth' into the model to obtain the optimal

hyperparameters from the trained model on Jupyter Labs. The filtered image array would then be resized, transformed into a tensor and applied to the model. A popup window would then display the filtered image array, the prediction of the image from the model and the certainty. However, on the Raspberry Pi, PyTorch and OpenCV were running too slowly on the microprocessor so it could not keep making predictions for 32 frames per second making the program unusable.

To get around this, I decided to use my laptop as the gesture recognition device, and I rewrote the code to run on my laptop which has a processor of a much higher clock rate so it would not have this problem. On my laptop, I installed the dependencies (CV2, NumPy, PIL, Torch, TorchVision, Copy and Time) on a virtual environment on an IDE called PyCharm. The only differences were that in this Python script, the frames were accessed from my laptop's webcam, and I had to set the resolution of the webcam to the PiCamera resolution to reuse most of the code.

Although this was not planned, this change was ideal because the laptop could run the model much quicker and make very accurate predictions for 60 frames per second. This way I could reuse the Raspberry Pi 4B for the drone and it would be much easier to get the computer to communicate with the drone's flight controller than getting the Raspberry Pi to communicate with the flight controller.

The purpose of this script is to send movement commands to the drone from my laptop based on the corresponding drone marshalling signal however this has not been implemented into the Python script yet.

*To see the code described, skip to 'Signal Recognition Program' under the Appendix.*

*Screenshots of the popup window created by the code are shown on the next page, displaying the model's predictions of the frames from the laptop's webcam*

black and white

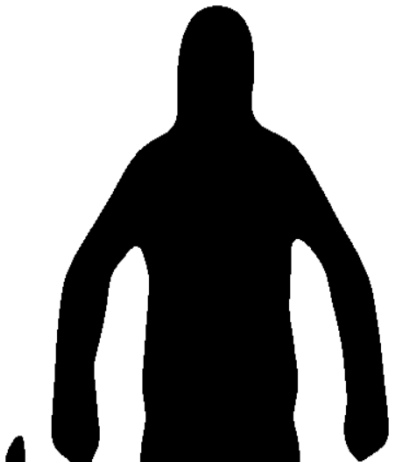
Prediction: left Certainty: 100

black and white

Prediction: right Certainty: 100

black and white

Prediction: downwards Certainty: 100



black and white

Prediction: upwards Certainty: 100

black and white

Prediction: forwards Certainty: 100

black and white

Prediction: backwards Certainty: 100

# How Quadcopters Fly

*The individual components will be explored in the Component Research chapter however, this chapter is a more holistic explanation as to how drones hover and fly.*

Quadcopters are unmanned drones that derive lift from four separate rotors oriented about a vertical axis. It is the propeller direction and the motors' speed and rotation that makes drone flight and manoeuvrability possible.

Generally, drones are controlled by remote control stick movement. The transmitter then sends radio waves to a receiver on the drone that transmits the data to the flight controller. This decodes the movement protocol and sends a signal to the electronic speed controllers which change the voltage across the motors accordingly to make the propellers spin in a particular direction and speed to make the drone move or hover.

The blades are pitched so that when quadcopter propellers spin, they push air down. Since all forces come in pairs, according to Newton's Third Law, for every action force there is an equal and opposite action force. Therefore, when propeller pushes down on the air, the air pushes up on the propeller, and the faster the propellers spin the greater the lift.

In the vertical plane a drone can climb, hover, and descend:

- To hover the net force of the four propellers pushing the drone up must be exactly equal to the weight of the drone.
- To climb, the four propellers need to be spinning at the same speed and faster than they were when hovering so the upwards force from the propellers is greater than the weight of the drone.
- To descend the four propellers, must spin slower than they were when hovering. This way they will produce less upward force than the weight of the drone causing it to descend.

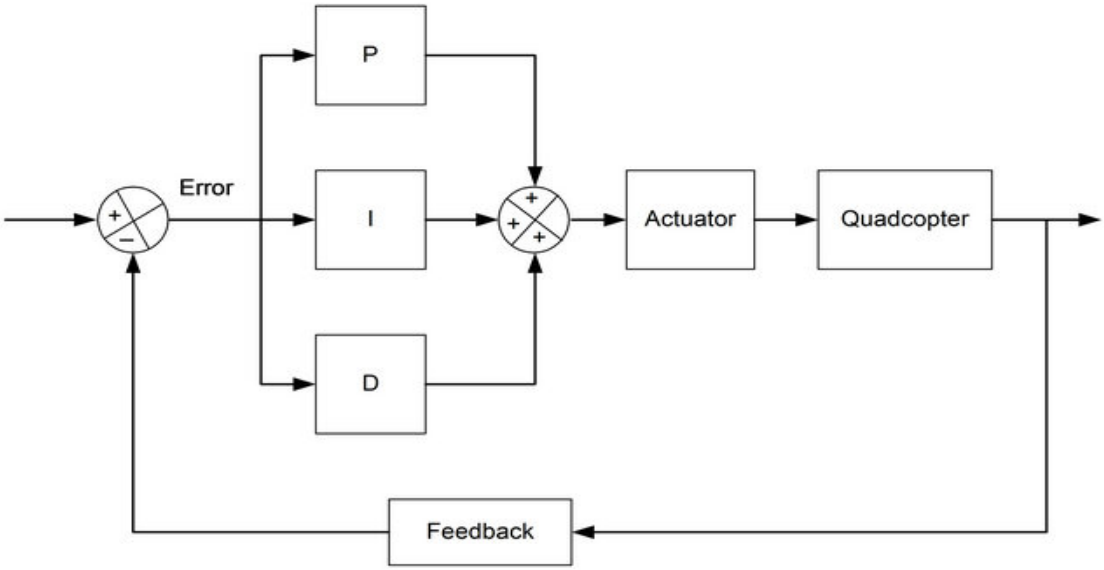

In the above diagram, the four motors are labelled and with two sets of opposite motors configured to rotate in opposite directions, the total angular momentum is zero. Angular momentum is the rotational equivalent of linear momentum that is calculated by multiplying the angular velocity by the momentum of inertia. When a drone is hovering the net angular momentum is zero. For example, the angular momentum of clockwise (CW) motors 1 and 3 could be +4 and the angular momentum of counterclockwise (CCW) motors 2 and 4 would have an angular momentum of -4.

To yaw right, CW motor 1 would have to decrease its speed and therefore its angular momentum relative to the other motors. For example, motor 2 and 4 have an angular momentum of -4, motor 3 has +4 and motor 1 has +2. Therefore, the net angular momentum of the drone is -2, which would cause it to spin right (clockwise). However, in doing this, the overall thrust upwards has decreased which would cause the drone to tip to one corner and start to descend. To do this while keeping the same position in the vertical axis, and without tipping to one side, motors 1 and 2 must decrease in speed and angular momentum and motors 2 and 3 must increase in speed to compensate for the lost upward thrust from motors 1 and 2.

Moving forward, backward, left, and right follows the same principle. This is that the motors on the side of the direction that the drone needs to move in, decrease in speed and the

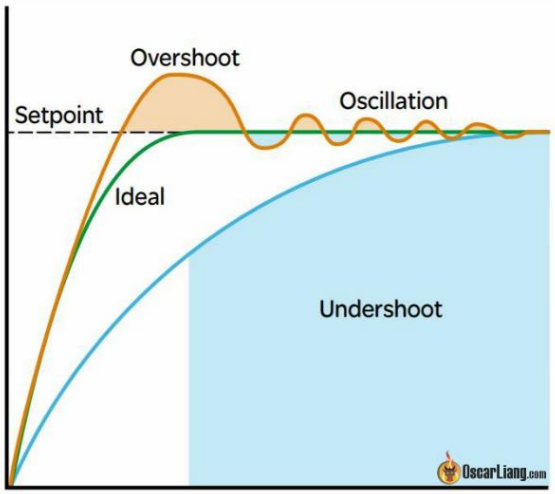
motors at the opposite side increase in speed. For example, to move forward, motor 1 and 2 decrease in speed and motors 3 and 4 will increase in speed so that the drone will pitch forward due to a greater thrust force from the rear motors causing a net thrust force forward. Additionally, it will stay at the same vertical level since the total upwards thrust is equal to the weight and it will not rotate since the net angular momentum is zero. Moving left and right is the same except the drone slightly rolls left or right causing net force in the direction that the drone is tilting towards.

For most remote-control transmitters, moving the right control stick left or right will cause a yawing manoeuvre and moving the right control stick up or down will cause forwards or backwards motion. Additionally, moving the left control stick up or down will increase/decrease your throttle and make your drone climb or descend, moving this control stick left or right will cause the drone to move in the left or right direction.




PID stands for proportional integral derivative and it's part of the flight controller software that reads data from the sensors and calculates how fast the motors should spin to retain the desired rotation of the drone. The purpose of the PID is to correct the *error* between the measured value (from the gyroscope and compass) and the desired 'setpoint.' The error can be minimized by adjusting the control inputs (sent from the transmitter) in every loop, which determines the speed of the motors to effectively correct the drone offset from its desired direction.

There are three overarching modules in a PID controller. The first is 'P' and it is to look at the present error and the further the drone is from the setpoint the harder it pushes to minimise that offset. The second is 'D,' to predict future errors, by looking at how fast the control loop is approaching the setpoint and counteracting 'P' when it gets close to the setpoint to minimise overshoot. The third is 'I' which is the accumulation of past errors, and it looks at forces that happen over time like a wind that is constantly causing the quadcopter to drift from a setpoint, so it increases the motors accordingly to counteract it.



# The Drone System

Since my laptop has become the primary image recognition device, I needed a method of communicating between a drone and my laptop.

There are several methods of achieving this communication that I investigated, and the three main setups I considered was using an ESP device to control the drone over WiFi via the UDP protocol, controlling a prebuilt DJI Tello using the DJI Tello SDK and using a radio telemetry kit to communicate via radio waves.

The ESP 12-F WiFi module was an especially intriguing option however due to its limited range and high latency it posed too many safety hazards. Additionally, the newer, more readily available versions of the ESP2886 were poorly documented for drone implementation and would require voltage regulators to ensure they constantly receive 3.3V.

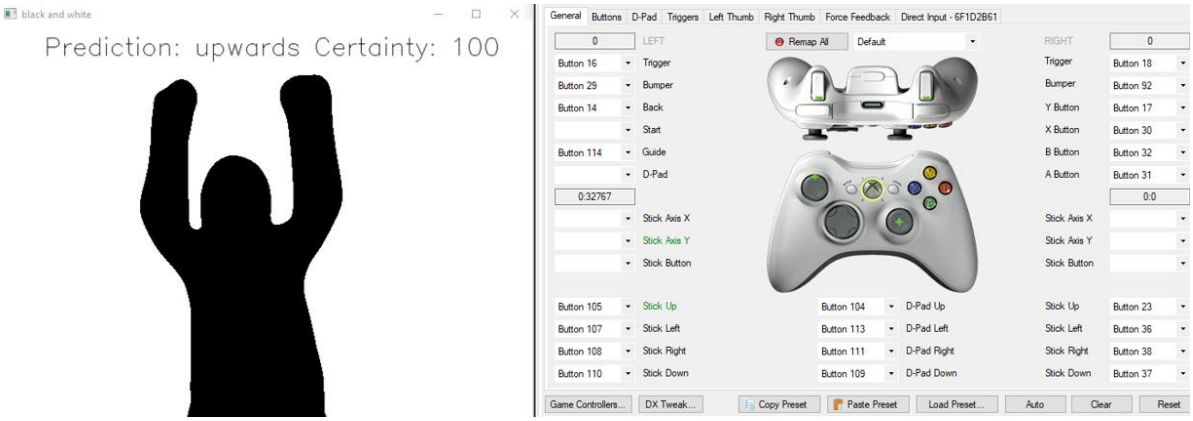
The DJI Tello is the cheapest alternative of the three ideas and is a small drone and with a software development kit (SDK). Although a prebuilt drone would save the time of assembly, it also takes away from the goal of the project, to design and build a drone system, Furthermore the high-level SDK, would also reduce the complexity of the coding the drone system and would oversimplify the project.

The final option that I ended up choosing was using a radio telemetry kit to send MAVLink protocols to the drone's flight controller from my computer using a GCS. While this option was the most expensive alternative of the three, it was the safest as it has the least latency of the three options, and it is unlikely that the drone would fall outside of the 300m+ range. Furthermore, it provided the necessary complexity for this project.

MAVLink is a lightweight messaging protocol for transmitting data between onboard components and communicating between drones and companion computers or ground control stations (GCS) using a radio telemetry kit. It is very efficient and uses just 8 bytes overhead per packet, including start sign, packet authentication, packet drop and corruption detection. Many different programming languages are supported to generate this protocol like Python, Java, JavaScript, Swift, Rust, C/++/# and many more. Additionally, it can be sent from many different types of microcontrollers and operating systems.

Rather than coding the parameters of the MAVLink protocol (that directs the drone to move) directly into my gesture recognition program, I will emulate joystick movements associated with the direction I want to move the drone in. The GCS software will understand these virtual joystick movements and send the corresponding MAVLink protocol to the flight controller using a radio telemetry kit. This protocol will then be sent using 433MHz radio waves and be picked up by the drone's receiver and decoded by the flight controller.

This was implemented in the bwlivepc.py Python script using the xbox360controller Python library to emulate the controller. When I want to end the program, I can stop the script by pressing lowercase c on my keyboard and then manually control the drone back down by using the virtual controller's joysticks on my touchscreen or an actual transmitter.



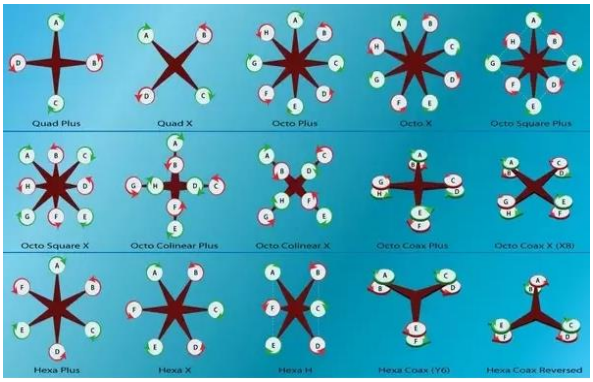

*The program is demonstrated in the screenshot above where I am signaling an upwards movement and the Xbox controller emulator's left joystick is moving upwards.*

# Component Research

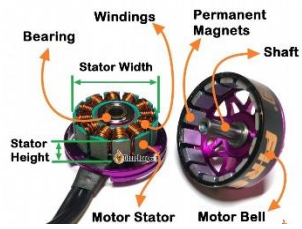
I began my part research with the drone frame as this would determine my approximate take-off weight. From here I could calculate the thrust required to maneuver safely, which motors I would need and in turn which electronic speed controllers and propellers I would need.

I could not purchase a fully integrated board that had the flight controller, power distribution circuit and ESCs embedded into one board as this would go against the aim of the project to build a drone system from scratch. To stick the aims and rationale of the project I had to buy all the necessary parts separately to acquire a fundamental understanding of how drones operate. Since integrated boards are able to fit multiple components on one board, they can be used to build smaller drones, however, to facilitate more parts and the extra weight that comes with that I would need a medium-sized frame. The DJI F450 drone kit is of medium size, weighed at 353g and it can support a takeoff weight between 800g and 1600g which is a suitable range for the parts I would attach to it. Additionally, the bottom PCB of the drone kit was a power distribution/management board (PDB/PMB) that would allow me to solder the battery cable and ESCs directly to it.

The DJI F450 frame has a layout that is described as a Quad X formation meaning that there are four motors to be place on the ends of each of the four arms. Opposing motors spin in the same direction and adjacent motors spin in the opposite direction during flight. This is one of the most common and well-documented drone formations which was another reason why I chose the DJI F450.



To accompany the frame, I had to purchase four brushless DC electric motors to provide thrust. These motors are also known as electronically commutated or synchronous DC motors and do not have carbon brushes like 'brushed motors' which are commonly used for drills and, jackhammers, planers, hedge trimmers and grinders. The carbon brush ensures the optimal transmission of charge to the rotating part, switching the current without a spark. However, they are usually in contact to the slip rings which makes them subject to friction and the switching action of the commutator that constantly makes and breaks the inductive circuit, creates electrical and electromagnetic noise. Additionally, the near-perpetual physical contact with the motor shaft causes the motor to wear out, therefore, reducing its lifespan.




Brushless motors tend to be 10% more efficient and have a longer lifespan than their brushed counterpart since they have no carbon brushes to cause friction and wear out. Instead of the carbon brush, brushless motors have a commutating device featuring an encoder and a drive with a control algorithm allowing the switching of current to be done electronically.

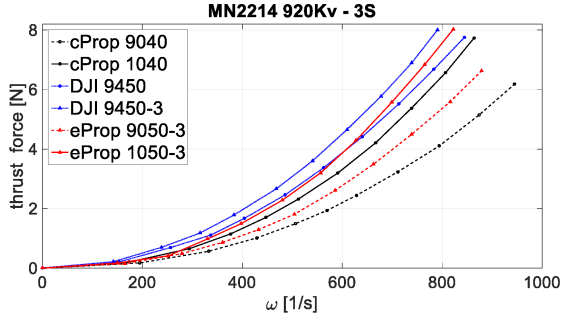
Since the estimated takeoff range of the drone was between 800g and 1600g, the drone had to be capable 1600g thrust at minimum. This is because the drone must have at least a 2:1 thrust to weight ratio to enable it to hover at half throttle and be able to stabilise in winds. 2:1 thrust to weight ratios are typically used for aerial photography purposes and first-person-view (FPV) drones require 4:1 to 7:1 ratios to compete in drone racing and aerobatic flying.

It is likely that I will meet the maximum 1600g takeoff limit and I am not aiming to build an aerobatic drone so I will need to achieve around 3200g of thrust at full throttle. Therefore since I have four motors, at full throttle, each motor will have to produce at least 800g of thrust. Brushless motors are given a Kv rating, indicating the increase in the motor shaft's revolutions per minute (RPM) for every volt supplied. Meaning that a 3-cell lithium-ion battery directly connected to the motor and discharging around 12V would spin the motor shaft 12000 times a minute.

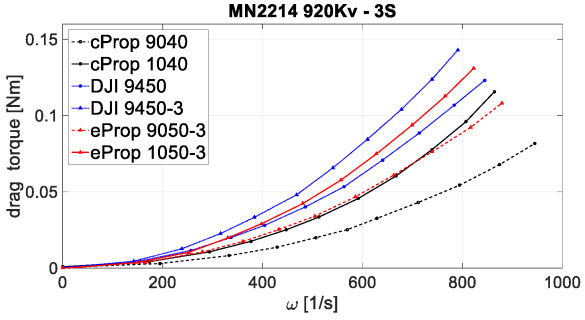
When different sized airscrews/propellers are placed on the motors they spin at different speeds and provide different thrusts. Since there are many other factors dictating the produced thrust, it is very difficult to work out the expected thrust when certain propellers are used, and conditions are met. Therefore, it is easier to experiment with the effects of different propellers on motors of different Kv ratings. The table on the right, provided by Emax indicates the measurements of their different

XA 2212 Brushless motor test record

| Motor type | The voltage (V) | Prop. Size | Current (A) | Thrust (G) | Power (W) | Efficiency (G/W) | RPM |
|---|---|---|---|---|---|---|---|
| XA 2212 820KV | 12 | APC 11*4.7 | 12 | 830 | 144 | 5.8 | 5720 |
| | 8 | APC 11*4.7 | 7.3 | 500 | 58.4 | 8.6 | 4650 |
| XA 2212 980KV | 12 | APC 10*4.7 | 15.1 | 880 | 181.2 | 4.9 | 6960 |
| | 8 | APC 10*4.7 | 9.5 | 550 | 76 | 7.2 | 5470 |
| | 12 | APC 9*6 | 12.3 | 730 | 147.6 | 4.9 | 8220 |
| | 8 | APC 9*6 | 7.1 | 400 | 56.8 | 7.0 | 6090 |
| XA 2212 | 12 | APC 8*4 | 16.4 | 930 | 196.8 | 4.7 | 12020 |
| | 8 | APC 8*4 | 9.1 | 500 | 72.8 | 6.9 | 8900 |



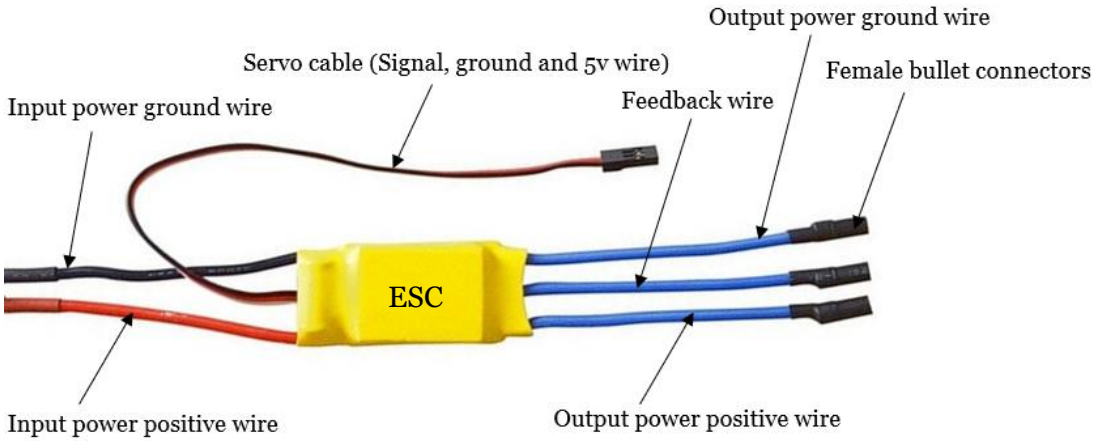

(a)




(b)

motors when different propellors and voltages are applied. I used the table to choose to purchase four 1400kV motors as they produce 940g of thrust when an 8" (8450) propellor is attached and 12V is applied across it. Since there were no 8" propellors readily available, I had to settle for a 10" (1045) pair of clockwise and counterclockwise propellers. As demonstrated in the graph below and partially by the above table, brushless motors produce varying levels of thrust and drag based on the type of propellor. 1400kV motors produce maximum thrust with an 8" propellor (assuming the voltage is constant), therefore using a 10" propellor would increase the drag and reduce thrust, to around 840g. However, propellers are not only optimized for maximum thrust for a particular motor, but different propeller sizes can also optimize efficiency and battery usage time (reduce current drawn from batteries).

According to the Emax table, each 1400kV motor draws 20.6A when an 8" propeller is attached and 12V is applied across. Therefore, it is safe to assume that each motor will draw no more than 30A of current when 12.6V is applied and a 10" propeller is applied. Based on this information I can purchase a 30A electronic speed controller for each of the motors. An electronic speed controller (ESC) regulates the speed of an electric motor by manipulating the voltage applied to it. It does this according to the precise instructions sent to them by the flight controller. The ESCs I purchased can support a continuous max current of 30A and a peak current of 40A, so they should be compatible with the motors at 12V.




The 1400kV demonstrated maximum thrust and current withdrawal at 12V, which is very close to the voltage produced by 3S (3 cell-count) Lipo batteries. These batteries produce 12.6V when fully charged, have a nominal voltage of 11.1V (each cell has a nominal voltage of 3.7V) and a cutoff voltage at 9V (3V for each cell). Since these voltages are very similar to the recorded maximum voltage applied to the 1400kV Emax motor, I purchased a 3S LiPo battery, a LiPo battery charger and an XT60 cable to connect the PDB to the LiPo battery.

During the assembly of the drone, I would require general equipment like zip ties to tie the ESCs to the drone frame's arms and 3M sticky pads to stick different devices and microprocessors to the drone frame. In addition to this equipment, I bought some breadboards and extra female and male jumper cables to connect pins between devices like the ESCs, receiver, flight controller, and for creating my own circuits to set up the receiver and test parts in isolation.

Since I have chosen a radio telemetry kit and joystick emulation as the method of communicating between the image recognition device (my laptop) and the drone's flight controller, I needed to find a ground control station (GCS) software with radio telemetry kit and joystick compatibility. Mission Planner, QGroundControl and PX4 Autopilot are all GCS

software that share these compatibilities. Finally, I had to purchase a flight controller (FC) that supported one of these GCS software.

Flight controllers is effectively the 'brain' or 'control unit' of the aircraft. In the context of drones, they have a range of sensors detecting the movement, position, and rotation of the drone as well as its location and the surrounding windspeeds through the use of a magnetometer (compass), GPS, barometric pressure sensor, accelerometer, gyroscope and a sonar sensor. It uses this sensory data to send information to the ESCs to either stabilize the drone or instruct them to control the motors to move the drone to a certain location or direction.

Most of the flight controllers that support the aforementioned GCS software are clones of the hardware from the APM and Pixhawk series. The last official model of the APM series is the APM 2.8, which was succeeded by Pixhawk and is no longer supported by GCS software. The original Pixhawk (PX) flight controller was released in 2013 and is still supported by the latest GCS software. However, since it was discontinued there are not many left in stock, and it is not readily available in the UK. The Pixhawk hardware and the accompanying GCS software are both open-source projects so as a result, there have been many clones of this flight controller by hobby companies. Here are some of the clones:

- Pixhawk PX4 2.4.8
- RadioLink MiniPix
- Cuav V5 Plus
- Cuav V5 Nano
- Cuav Nora
- Cuav X7/Pro
- Drotek Pixhawk 3
- CubePilot Black
- CubePilot Orange
- CubePilot Purple
- CubePilot Yellow
- CubePilot Green
- Holybro Durandal H7
- Holybro Pix32 v5
- Holybro Pixhawk (PX) 4
- mRo Pixhawk
- mRo Pixracer
- mRo X2.1
- mRo X2.1-777

Although these clones are based on the original Pixhawk FC hardware, many of them have different I/O ports, pins, GCS software and voltage requirements. Therefore, the detailed Pixhawk documentation, explaining how to setup the FC and connect the other components cannot be applied to many of the clones. Most of the clones do have their own documentation and user forums however, the documentation is very limited, and it is difficult to find information applicable to your own specific FC and situation from user forums as there are so many PX clones to choose from.

After researching about each of the different types of PX clones I purchased the Holybro PX4, a Pixhawk clone made by Holybro and PX4, since it was compatible with the radio telemetry kit, I had ordered from Holybro.

To control the drone manually and perform the intial test flights I would need a receiver and transmitter. I purchased the FlySky FS-i6 transmitter as this was the cheapest RC controller that possessed eight native channels (four switches and two joysticks) and was highly regarded by reviewers as a high-quality budget transmitter. To accompany the transmitter, I bought the FlySky FS-iA6B receiver which supports Pulse Position Modulation (PPM) output, a RC protocol I would require for my flight controller.

# Budget Spreadsheet

GOLD CREST

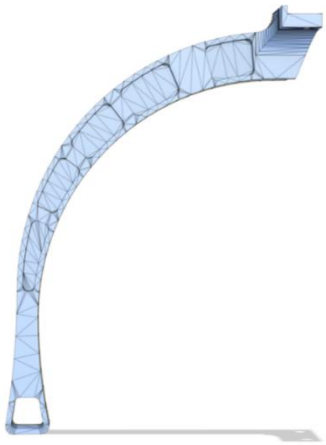
**BUDGET**

| EXPENSE | Cost | NOTES | SELLER | LINK | ordered |
|---|---|---|---|---|---|
| Pixhawk 4 | £100.00 | Flight controller +GPS+PM | 3DXR | https://www.3dxr.co.uk/autopilots-c2/holybro-pixhawk-4-with-plastic-case-gps-and-pm02-p3782 | Yes |
| Drone Frame Kit for DJI F450 | £30.99 | RC Drone Frame Kit | Amazon | https://www.amazon.co.uk/drone-frame-Integrated-Quadcopter- | Yes |
| Zip Ties | £5.80 | 200mm x 2.5mm Black | Amazon | https://www.amazon.co.uk/Cable-2-5mm-Black-Nylon-200mm/dp/B074LV8Y2R/ref=sxin_13_ac_d_rm?ac_md=2-2- | Yes |
| Flysky FS-iA6B | £13.99 | Radio receiver | Amazon | https://www.amazon.co.uk/ICGUANZX-Receiver-Compatible-Transmitter-FS-iA6B/dp/B081CVR7N7/ref=sr_1_1_sspa?crid=2 | Yes |
| FS FlySky i6 2.4G 6ch Transmitter | £54.99 | Controller | Amazon | https://www.amazon.co.uk/FS-Transmitter-Receiver-helicopter-ARRIVAL/dp/B00PF7DQBI | Yes |
| Brushless motors | £76.50 | Spins the propellers | Amazon | https://www.amazon.co.uk/gp/product/B07L1N49N4/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1 | Yes |
| 30A Burshless ESC x4 | £60.00 | Controls voltage to motors | Amazon | https://www.amazon.co.uk/Brushless-Electronic-Controller-Accessories- | Yes |
| Jumper wires kit | £8.99 | Different lengths for breadboard | Amazon | https://www.amazon.co.uk/WANTOUTH-Preformed-Breadboard-Solderless- | Yes |
| Breadboard | £5.29 | Connecting the jumper wires to 3.3V | Amazon | https://www.amazon.co.uk/AZDelivery-102-Breadboard-830-Breadboard-Breadboard- | Yes |
| SiK Telemetry Radio V3 433MHz | £55.95 | Laptop to drone control | Bandgood | https://uk.banggood.com/Holybro-SiK-Telemetry-Radio-V3-100mW-433MHz-or-915MHz-Open-Source-SiK-firmware-Plug-n- | Yes |
| Jumper wires: male to female | £6.99 | male to female, female to female, male to male | Amazon | https://www.amazon.co.uk/MMOBIEL-Multicolored-Breadboard-Compatible- | Yes |
| Li-po battery | £19.99 | 11v Lipo, 2200mAh 3s, powers the drone | Amazon | https://www.amazon.co.uk/gp/product/B08Y8ZGCW1/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1 | Yes |
| Propellers 1045 x4 | £14.04 | Spins for lift and control | Amazon | https://www.amazon.co.uk/KEESIN-Pairs-Propellers-Quadcopter-Multirotor/dp/B01LNE1EJ2 | Yes |
| USB TTL converter | £7.99 | Sends code from laptop to ESP | Amazon | https://www.amazon.co.uk/Yishet-Converter-Compatible-Arduino- | Yes |
| 3M sticky pads | £4.48 | Adhesive pads for flight controller onto frame | Amazon | https://www.amazon.co.uk/Double-Sticky-Adhesive-Mounting-Square/dp/B07C6FTG3W/ref=sr_1_8?crid=1TFR7T88Z68SX& | Yes |
| LiPo battery charger | £16.99 | Charges the battery | Amazon | https://www.amazon.co.uk/Hsizito-Charger-Battery-100-240V-Quadcopter/dp/B07WG25H6K/ref=pd_bxgy_img_1/262- | Yes |
| XT 60 male connector | £8.99 | Connects the frame PCB to the battery | Amazon | https://www.amazon.co.uk/gp/product/B0739MJMRV/ref=ppx_yo_dt_b_asin_title_o01_s00?ie=UTF8&psc=1 | Yes |
| **Total** | **£491.97** | | | | |

After researching what is involved in the making of a drone, I began the budgeting stage. This was one of the most difficult phases of the project and the goal was to find acceptably priced, quality parts that would be compatible with the rest of the drone. Additionally, I had far surpassed the time I had allocated for drone research, so I needed to find readily available parts in the UK. Therefore, I ordered most of the parts from the same reseller (Amazon), which had an assurance that most of the parts would come within a week. Upon receiving all the components, I could then get back on track and build the drone. The total cost for the drone parts was £491.97 which was reimbursed by Hurstpierpoint College.

# 3D Print Landing Legs

To protect the drone frame from harsh landings and crashes during the initial flights due to potential ESC miscalibration, I will 3D print landing legs to absorb the impact and prevent the circuitry on the drone from being in contact with the ground. I found STL files on STLfinder.com for drone landing legs and I chose a model that curves inwards at the top so that the leg can absorb some of the energy from impact with the ground. I then resized the legs on Autodesk Fusion 360 and added screw holes so that the legs can be fitted under the built-in PDB on the DJI F450 frame using the same screws and holes that would attach the drone arms.

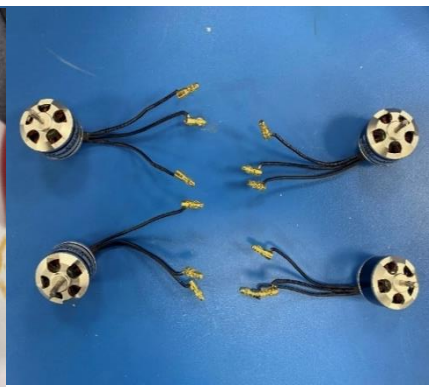


# Building the Drone

## Soldering Bullet Connectors to the Motors

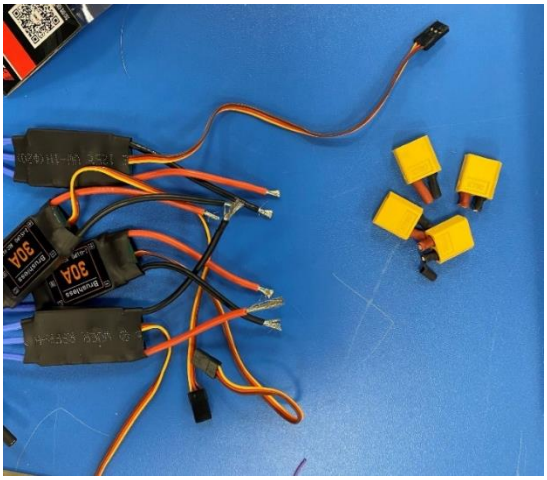
To begin the assembly, I soldered male bullet connectors to the three ends of the brushless motors' wires. This is so that I can connect the motor wires to the female bullet connectors of the ESCs. I could have removed the ESC's female bullet connectors and directly soldered the motors to the ESCs, however if I had done this and the motors were to spin the wrong way, I would have to remove the heat shrink and solder between all the ESC and motor wires and then resolder the positive and negative ESC wires so that they connect to the opposite motor wires. With male bullet connectors I can just unplug the left and right wires (middle wire is feedback and is not involved in flipping the current) from the female bullet connectors of the output negative and positive ESC wires and reattach them to the opposite ESC bullet connectors to change the direction of the current.

*The picture on the right shows the brushless motors with the soldered male bullet connectors.*



## ESC Wire Stripping

Using wire strippers, I removed the end of the ESC's input positive and ground wires that had an XT60 connector attached to them. I did this to expose the wire inside the cable so that it can be soldered to the power distribution board (PDB). Additionally, if I had left the XT60 connector on, I would only be able to connect one ESC to the battery and only one motor would spin.

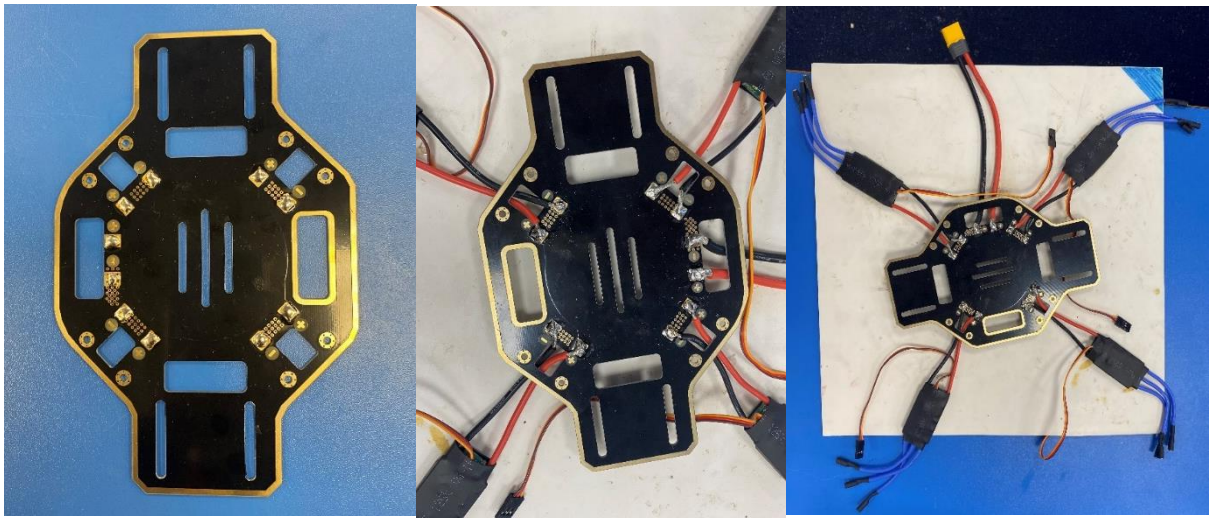



*The picture shows the stripped ends of the ESCs' power cables (red and black) with the XT60 connectors (yellow) cut off to the right of the ESCs.*

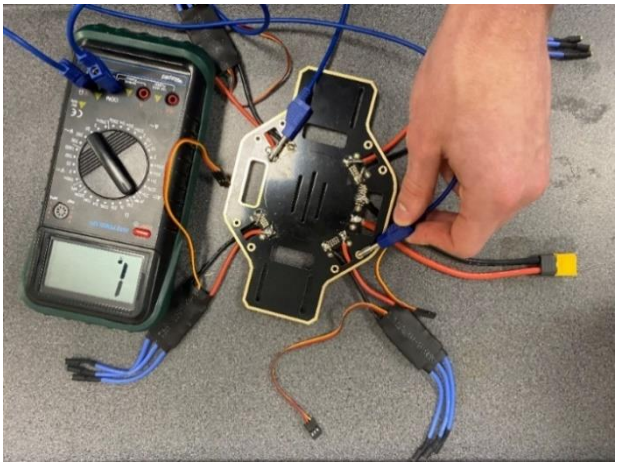
## Soldering the ESCs' input wires and XT60 Cable to the PDB

Solder the ESCs' positive and negative input cables to the respective pads on the power management board integrated onto the frame's bottom plate. To do this, apply solder on each of the pads and then to the ends of the cables. Thread the cables through the gaps on the frame, heat the solder on the wires and pads and then hold the cables onto the pads until the solder solidifies.

*Solder the red wires with positive pads and black wires with negative pads as shown in the pictures below.*



## Checking Circuit Connections



Plug a banana chord to the common negative connection (COM) and another chord to the positive connection on a Multimeter. Then set the Multimeter on ‘continuity test’ (speaker symbol) and place the ends of the chords on different soldered connections on the power management board. It will indicate the circuit is closed if the Multimeter’s speaker beeps when the chords are placed on the solder connections.

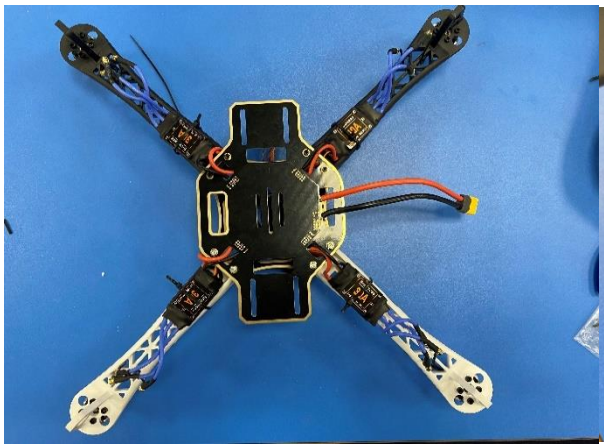
## Attach Drone Arms with Motors to the Frame

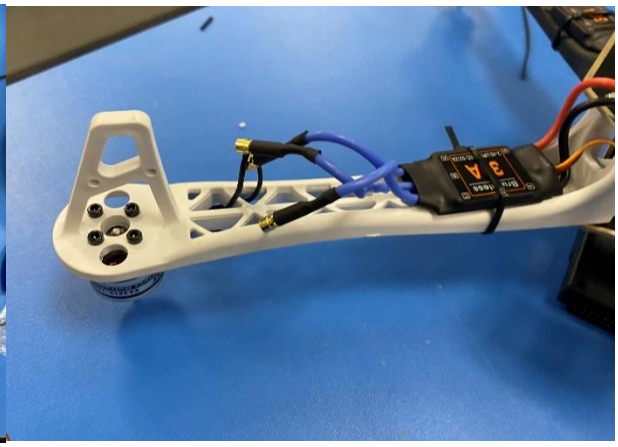
First screw the motors onto the drone arms using 2.5mm screws and thread the wires through the gaps on the drone arms. Then hold an arm in a G clamp and use an Allen key to screw the 2.5mm screws into the frame (bottom plate). Repeat this for each drone arm.

## Connecting the Motors and ESCs

Attach the motor input wire to the ESC output cables using the male/female bullet connectors. Connect the centre motor wire’s male bullet connector to the ESC feedback (centre) cable’s female bullet connector. For opposite drone arms attach the same ESC output cables to the same motor input cables. For adjacent drone arms attach different the outer motor input wires to different ESC output wires. For example, for a pair of opposite drone arms attach the left input motor wires to the left output ESC cable however for the

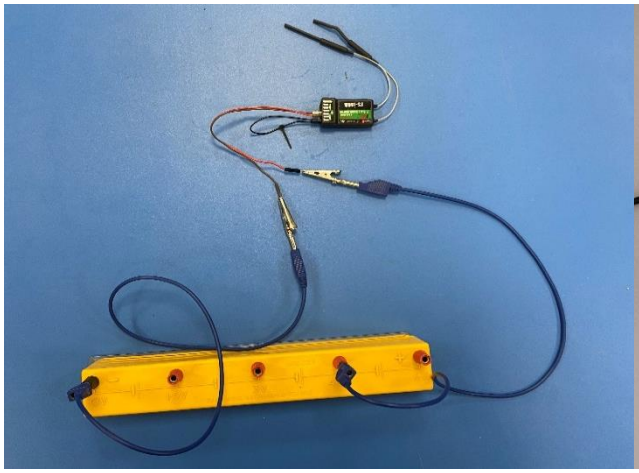
other pair reverse these connections. Following this, use zipties to hold the ESCs and wires in place so that they do not move during flight.





## Setting up the Receiver and Binding to Transmitter

Connect the male-female jumper wires from a 5V supply to the servo pins on the receiver to power it. Then set the receiver in binding mode by placing the binding chord on the B/VCC pins. After the receiver is in binding mode, hold down the transmitter's binding key (bottom left black button) while pushing the ON/OFF switch on to set the transmitter on binding. Once the binding process is complete, go to the transmitter's System age and create an aircraft/glider model named Quad 01. Afterwards go to the Setup page and assign switches C and B to aux channels 5 and 6 respectfully for model Quad 01. Return to the Setup page and scroll to RX Setup where you should enable PPM mode and set channel 13 to a failsafe.



## Attaching the Upper Plate and GPS Mount

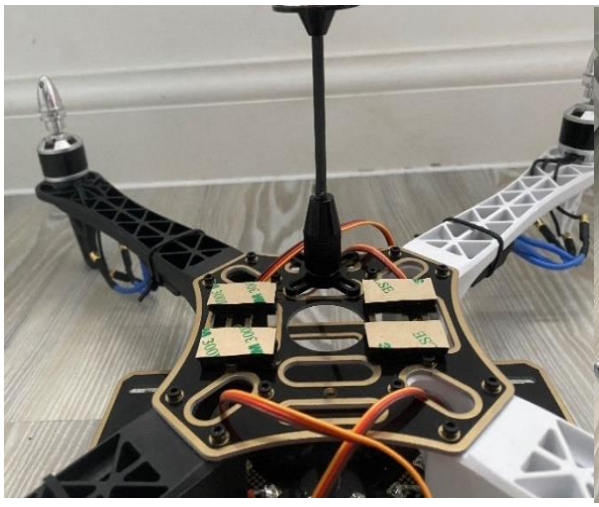
Screw the GPS mount brace into the upper plate using four 2.5mm screws, place the 145mm pole into position on the GPS mount brace and lock into place using a 2.5mm screw to prevent it from sliding out. Slide the pole locking mechanism over the pole and screw it onto the bottom of the GPS mount brace. Following this, place the mount on top of the pole and lock it into place using another screw. Finally stick a double-sided 3M pad onto the top of the GPS mount.

After the GPS mount is securely attached to the upper plate, use sixteen 3mm screws to fix the upper plate onto the drone arms.

## Attaching the Flight Controller

Place the four foam double-sided sticky pads on the upper plate. The foam pads act as vibration dampeners as the flight controller's accelerometer is sensitive to vibrations. On top of the foam pads stick the flight controller (FC) while making sure it is facing forwards (between the white drone arms).

*Flight controller must be forward-facing and centered perfectly as depicted below.*



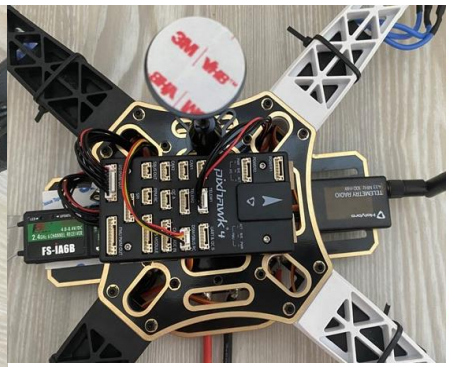
## Add Radio Telemetry kit and Receiver

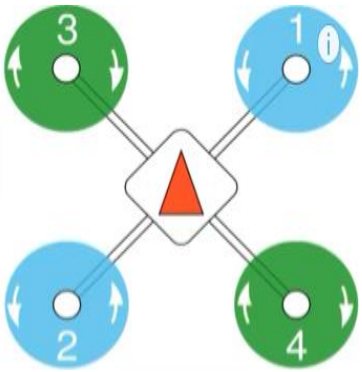
Place a single 15mm 3M sticky pad on either extruding end of the bottom plate. Stick the receiver and one of the Radio Telemetry (RTK) devices onto the pads to secure them. Then screw antenna onto both RTK devices and use a micro-USB to connect the other RTK device to a computer to receive and transmit between the pair.

## Servo Wires to Signal Splitter

Connect the servo wires (5V, ground and signal wires) from each ESC to the corresponding pins on the signal splitter as described by the below right diagram and plug the signal splitter's output connector to the FC (which will send messages via the signal wires to instruct the ESCs on what voltage to apply across the motors).

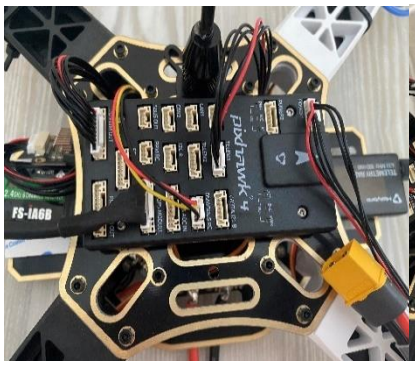
*This is demonstrated by the following diagrams.*

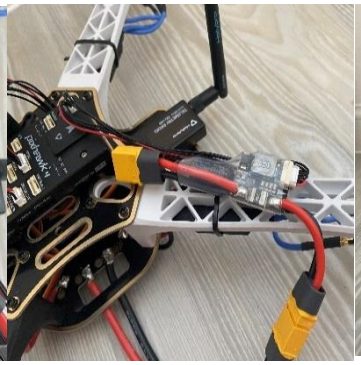



## Connecting Components to the FC

Insert an SD card into the FC and connect the PM02 power cable (attached to the XT60 cable) to the FC's Power 1 plug. Plug the receiver into the FC via the PPM port and plug the attached Radio Telemetry device into the FC via the Telem 1 port. Secure the GPS using the sticky pad situated on the GPS mount and make sure it is forward-facing (in the same direction as the flight controller). Finally connect the GPS to the GPS module port on the FC and plug the XT60 cable into the LiPo battery to start the drone.



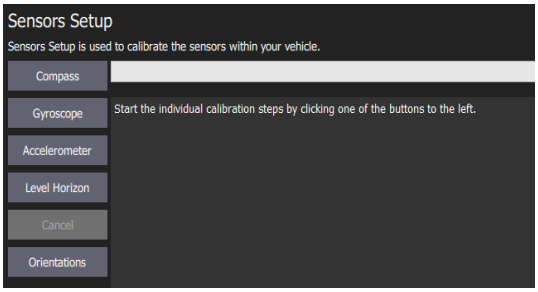



## Strap Battery & Attach Propellers

Thread two cable straps through the bottom plate to secure a LiPo battery in between them. Using the past diagram demonstrating the direction of propeller spin attach the clockwise (CW) propellers to motors 3 & 4 and the counter-clockwise (CCW) propellers to motors 1 & 2 and secure them in place by screwing on the motor caps over the propellers. The propellers which have a raised blade on the left (from a side view) are CW and the propellers with a raised blade on the right are CCW.

## Setting up Drone on GCS

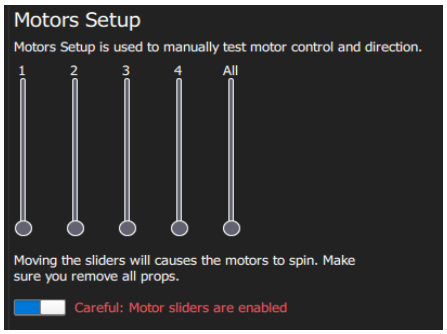
To setup the ground control station and configure the drone I used QGroundControl. After installing the latest firmware into the flight controller (FC) I had to select properties relating to the drone's frame and battery to generate the parameters used to configure the flight controller. Finally, I had to calibrate the FC's sensors by placing the drone in different positions.




Once all the sensors indicated they were ready for flight, I started the transmitter which which was binded to the receiver. This was picked up by the GCS which displayed the corresponding effect each joystick movement would have on the direction of the drone using the sliding bars (Altitude Controls) shown in the screenshot below.

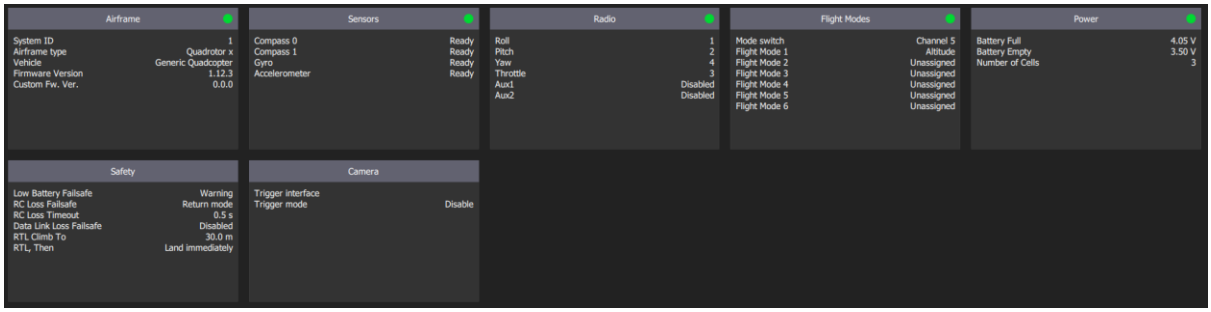

Following this, I enabled the position, stabilise, and hover flight modes using channel 5 (a three-state switch on the transmitter). Additionally, I used channel 6 to enable arming (allows the propellers to spin) and channel 7 as the failsafe which would command the drone to return to its take-off position.

After setting up the transmitter, I connected the battery to the power management board's XT60 connector and tested the motors to make sure they were spinning in the right direction and functioning optimally.

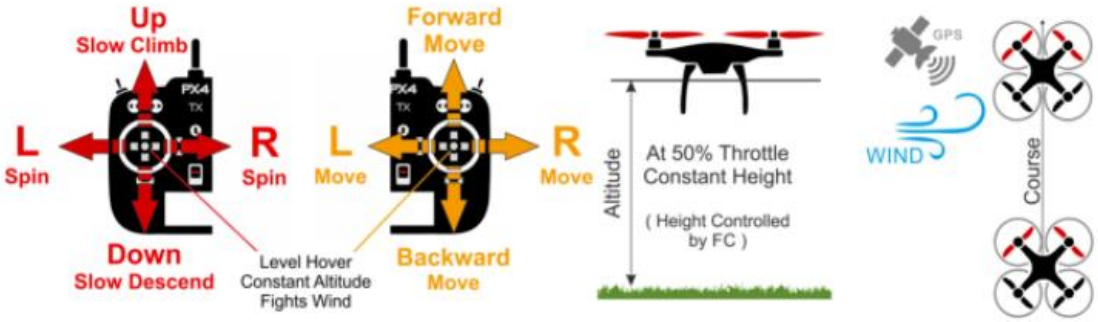



When the basic setup is complete the overview panel should have green indicators at the top right of each section to show that all components are ready for flight.

*This is indicated in the panel below.*




## How to Fly a Drone



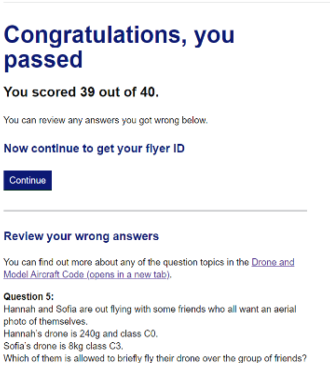

## Safety Measures

While soldering the wires and circuitry involved with the drone, I had to adhere to various protocols to ensure my safety was upheld.

- Hold wires using the helping hand clamp and never touch the heating element to avoid burns.
- Wear eye protection as solder can 'spit.'
- Turn off the soldering iron when not in use.
- Wear gloves when directly handling solder.
- Enable fume extraction and do not breathe in solder flux fumes to avoid throat, eye and lung irritation.
- Work on a fireproof surface to reduce risk of fire.

In order to fly the drone, I had to study and pass an examination on the Drone and Model Aircraft Code to receive my Flyer and Operator ID. After this I printed my Operator ID and stuck it on a drone arm to indicate I was the operator of the drone

*The Flyer ID and examination result are illustrated below.*



**Congratulations, you passed**

**You scored 39 out of 40.**

You can review any answers you got wrong below.

**Now continue to get your flyer ID**

Continue

**Review your wrong answers**

You can find out more about any of the question topics in the Drone and Model Aircraft Code (opens in a new tab).

**Question 5:**
Hannah and Sofia are out flying with some friends who all want an aerial photo of themselves.
Hannah's drone is 240g and class C0.
Sofia's drone is 8kg class C3.
Which of them is allowed to briefly fly their drone over the group of friends?

UNITED KINGDOM

**Drone and model aircraft**

Proof of completion of online training

UK Civil Aviation Authority

**Flyer ID**

**GBR-RP-3PK32JMBZJG2**

**First name:** JOSHUA SHAUN

**Last name:** KILLA

**Expiry date:** 31 March 2027

A1 & A3 OPEN SUB CATEGORY

SCAN ME

register-drones.caa.co.uk

To fly legally, I ensured that there were no local byelaws or airspace restrictions that prevented drone flight in the area. Additionally, I had to make sure I was at least 50m clear of people that were not involved in my activity and 150m away from residential, commercial, and industrial areas. Furthermore, the failsafe I had enabled on the drone further reduced the risk that it could pose and allowed me to fly safely.

# Appendix

## Training the Signal Recognition Model

In [1]:
```
import cv2
import numpy as np
import os
import PIL
from PIL import Image
import random
from tqdm import tqdm
import pickle


import torch
import glob
import torch.nn as nn
from torchvision.transforms import transforms
from torch.utils.data import DataLoader
from torch.optim import Adam
from torch.autograd import Variable
import torchvision
import pathlib

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
```

In [2]:
```
device=torch.device('cpu')
```

```
cpu
```

In [3]:
```
transformer=transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((150,150)),
    transforms.ToTensor(),  #0-255 to 0-1, numpy to tensors
    transforms.Normalize((0.5), (0.5))
])
```

In [4]:
```
train_path='C:/Users/joshu/OneDrive/A levels/Other/Crest/Crest code/imgRecognition/sign
test_path='C:/Users/joshu\OneDrive/A levels/Other/Crest/Crest code/imgRecognition/signa

train_loader=DataLoader(
    torchvision.datasets.ImageFolder(train_path,transform=transformer),
    batch_size=64, shuffle=True
)
test_loader=DataLoader(
    torchvision.datasets.ImageFolder(test_path,transform=transformer),
    batch_size=32, shuffle=True
)
```

In [5]:
```
#categories
root=pathlib.Path(train_path)
classes=sorted([j.name.split('/')[-1] for j in root.iterdir()])
print(classes)
```

```
['backwards', 'downwards', 'forwards', 'left', 'right', 'upwards']
```

In [6]:

```
#CNN Network


class ConvNet(nn.Module):
    def __init__(self,num_classes=6):
        super(ConvNet,self).__init__()

        #Output size after convolution filter
        #((w-f+2P)/s) +1
        #((width-kernel_size+2Padding)/stride) +1

        #Input shape= (64,1,150,150)
        #batch_size = 64
        #channels =1 hopefully not 3 fix?
        #height/width=150
        #stride=1
        #Padding=1
        #kernel_size(f)=3
        #(batch size,channels,height width)

        self.conv1=nn.Conv2d(in_channels=1,out_channels=12,kernel_size=3,stride=1,paddi
        #Shape= (64,12,150,150)
        self.bn1=nn.BatchNorm2d(num_features=12)
        #normalisation
        #Shape= (64,12,150,150)
        self.relu1=nn.ReLU()
        #Shape= (64,12,150,150)

        self.pool=nn.MaxPool2d(kernel_size=2)
        #Reduce the image size be factor 2
        #Shape= (64,12,75,75)


        self.conv2=nn.Conv2d(in_channels=12,out_channels=20,kernel_size=3,stride=1,padd
        #Shape= (64,20,75,75)
        self.relu2=nn.ReLU()
        #Shape= (64,20,75,75)



        self.conv3=nn.Conv2d(in_channels=20,out_channels=32,kernel_size=3,stride=1,padd
        #Shape= (64,32,75,75)
        self.bn3=nn.BatchNorm2d(num_features=32)
        #Shape= (64,32,75,75)
        self.relu3=nn.ReLU()
        #Shape= (64,32,75,75)


        self.fc=nn.Linear(in_features=75 * 75 * 32,out_features=num_classes)

        #Feed forward function

    def forward(self,input):
        output=self.conv1(input)
        output=self.bn1(output)
        output=self.relu1(output)

        output=self.pool(output)

        output=self.conv2(output)
```

```
        output=self.relu2(output)

        output=self.conv3(output)
        output=self.bn3(output)
        output=self.relu3(output)

        #Above output will be in matrix form, with shape (256,32,75,75)

        output=output.view(-1,32*75*75)


        output=self.fc(output)

        return output
```

In [7]:
```
model=ConvNet(num_classes=6).to(device)
```

In [8]:
```
#Optmizer and loss function
optimizer=Adam(model.parameters(),lr=0.001,weight_decay=0.0001)
loss_function=nn.CrossEntropyLoss()
```

In [9]:
```
num_epochs=10
```

In [10]:
```
#calculating the size of training and testing images
train_count=len(glob.glob(train_path+'/**/*.jpg'))
test_count=len(glob.glob(test_path+'/**/*.jpg'))
print(train_count,test_count)
```

```
4200 300
```

In [11]:
```
#Model training and saving best model

best_accuracy=0.0

for epoch in range(num_epochs):

    #Evaluation and training on training dataset
    model.train()
    train_accuracy=0.0
    train_loss=0.0

    for i, (images,labels) in enumerate(train_loader):

        #if torch.cuda.is_available():
        #    images=Variable(images.cuda())
        #    labels=Variable(labels.cuda())

        optimizer.zero_grad()

        outputs=model(images)
        loss=loss_function(outputs,labels)
        loss.backward()
```

```
        optimizer.step()


        train_loss+= loss.cpu().data*images.size(0)
        _,prediction=torch.max(outputs.data,1)

        train_accuracy+=int(torch.sum(prediction==labels.data))

    train_accuracy=train_accuracy/train_count
    train_loss=train_loss/train_count


    # Evaluation on testing dataset
    model.eval()

    test_accuracy=0.0
    for i, (images,labels) in enumerate(test_loader):
        #if torch.cuda.is_available():
        #    images=Variable(images.cuda())
        #    labels=Variable(labels.cuda())

        outputs=model(images)
        _,prediction=torch.max(outputs.data,1)
        test_accuracy+=int(torch.sum(prediction==labels.data))

    test_accuracy=test_accuracy/test_count


    print('Epoch: '+str(epoch)+' Train Loss: '+str(train_loss)+' Train Accuracy: '+str(

    #Save the best model
    if test_accuracy>best_accuracy:
        torch.save(model.state_dict(),'best_checkpoint.pth')
        best_accuracy=test_accuracy
```

```
Epoch: 0 Train Loss: tensor(2.5859) Train Accuracy: 0.8861904761904762 Test Accuracy: 0.
9866666666666667
Epoch: 1 Train Loss: tensor(0.0231) Train Accuracy: 0.9964285714285714 Test Accuracy: 0.
99
Epoch: 2 Train Loss: tensor(0.0024) Train Accuracy: 0.9995238095238095 Test Accuracy: 0.
9866666666666667
Epoch: 3 Train Loss: tensor(0.0267) Train Accuracy: 0.995 Test Accuracy: 0.9933333333333
333
Epoch: 4 Train Loss: tensor(0.0240) Train Accuracy: 0.9973809523809524 Test Accuracy: 0.
99
Epoch: 5 Train Loss: tensor(0.0328) Train Accuracy: 0.9957142857142857 Test Accuracy: 0.
9466666666666667
Epoch: 6 Train Loss: tensor(0.0458) Train Accuracy: 0.9952380952380953 Test Accuracy: 0.
99
Epoch: 7 Train Loss: tensor(0.0082) Train Accuracy: 0.9985714285714286 Test Accuracy: 0.
98
Epoch: 8 Train Loss: tensor(0.0034) Train Accuracy: 0.9992857142857143 Test Accuracy: 0.
99
Epoch: 9 Train Loss: tensor(0.0008) Train Accuracy: 0.9997619047619047 Test Accuracy: 0.
98
```

In [25]:

```
import cv2
import numpy as np
import os
import PIL
from PIL import Image
import random
from tqdm import tqdm
import pickle


import torch
import glob
import torch.nn as nn
import torch.nn.functional as func
from torchvision.transforms import transforms
from torch.utils.data import DataLoader
from torch.optim import Adam
from torch.autograd import Variable
import torchvision
import pathlib
3
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
```

In [26]:

```
train_path='C:/Users/joshu/OneDrive/A levels/Other/Crest/Crest code/imgRecognition/s
pred_path='C:/Users/joshu/OneDrive/A levels/Other/Crest/Crest code/imgRecognition/si
```

In [27]:

```
#categories
root=pathlib.Path(train_path)
classes=sorted([j.name.split('/')[-1] for j in root.iterdir()])
print(classes)
```

```
['backwards', 'downwards', 'forwards', 'left', 'right', 'upwards']
```

In [28]:

```
#CNN Network


class ConvNet(nn.Module):
    def __init__(self,num_classes=6):
        super(ConvNet,self).__init__()

        #Output size after convolution filter
        #((w-f+2P)/s) +1
        #((width-kernel_size+2Padding)/stride) +1

        #Input shape= (64,1,150,150)
        #batch_size = 64
        #channels =1 hopefully not 3 fix?
        #height/width=150
        #stride=1
        #Padding=1
        #kernel_size(f)=3
        #(batch size,channels,height width)

        self.conv1=nn.Conv2d(in_channels=1,out_channels=12,kernel_size=3,stride=1,pa
        #Shape= (64,12,150,150)
        self.bn1=nn.BatchNorm2d(num_features=12)
        #normalisation
```

```
        #Shape= (64,12,150,150)
        self.relu1=nn.ReLU()
        #Shape= (64,12,150,150)

        self.pool=nn.MaxPool2d(kernel_size=2)
        #Reduce the image size be factor 2
        #Shape= (64,12,75,75)


        self.conv2=nn.Conv2d(in_channels=12,out_channels=20,kernel_size=3,stride=1,p
        #Shape= (64,20,75,75)
        self.relu2=nn.ReLU()
        #Shape= (64,20,75,75)



        self.conv3=nn.Conv2d(in_channels=20,out_channels=32,kernel_size=3,stride=1,p
        #Shape= (64,32,75,75)
        self.bn3=nn.BatchNorm2d(num_features=32)
        #Shape= (64,32,75,75)
        self.relu3=nn.ReLU()
        #Shape= (64,32,75,75)


        self.fc=nn.Linear(in_features=75 * 75 * 32,out_features=num_classes)



        #Feed forwad function

    def forward(self,input):
        output=self.conv1(input)
        output=self.bn1(output)
        output=self.relu1(output)

        output=self.pool(output)

        output=self.conv2(output)
        output=self.relu2(output)

        output=self.conv3(output)
        output=self.bn3(output)
        output=self.relu3(output)


            #Above output will be in matrix form, with shape (256,32,75,75)

        output=output.view(-1,32*75*75)


        output=self.fc(output)

        return output
```

In [29]:

```
checkpoint=torch.load('best_checkpoint.pth')
model=ConvNet(num_classes=6)
model.load_state_dict(checkpoint)
model.eval()
```

Out[29]:

```
ConvNet(
```

```
  (conv1): Conv2d(1, 12, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(12, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
  (relu1): ReLU()
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(12, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (relu2): ReLU()
  (conv3): Conv2d(20, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
  (relu3): ReLU()
  (fc): Linear(in_features=180000, out_features=6, bias=True)
)
```

In [30]:

```
#Transforms
transformer=transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((150,150)),
    transforms.ToTensor(),  #0-255 to 0-1, numpy to tensors
    transforms.Normalize((0.5), (0.5))

])
```

In [31]:

```
#prediction function
def prediction(img_path,transformer):

    image=Image.open(img_path)

    image_tensor=transformer(image).float()


    image_tensor=image_tensor.unsqueeze_(0)

    #if torch.cuda.is_available():
    #    image_tensor.cuda()


    output=model(Variable(image_tensor))

    index=output.data.numpy().argmax()

    pred=classes[index]

    return pred
```

In [32]:

```
images_path=glob.glob(pred_path+'/*.jpg')
```

In [33]:

```
pred_dict={}

for i in images_path:
    pred_dict[i[i.rfind('/')+1:]]=prediction(i,transformer)
```

In [34]:

```
pred_dict
```

Out[34]:

```
{'pred_silhouettes\\backwards_1.jpg': 'backwards',
 'pred_silhouettes\\backwards_267.jpg': 'backwards',
 'pred_silhouettes\\backwards_348.jpg': 'backwards',
 'pred_silhouettes\\backwards_42.jpg': 'backwards',
 'pred_silhouettes\\backwards_689.jpg': 'backwards',
```

```
'pred_silhouettes\\backwards_fake_1.jpg': 'backwards',
'pred_silhouettes\\backwards_new_1.jpg': 'backwards',
'pred_silhouettes\\backwards_new_2.jpg': 'backwards',
'pred_silhouettes\\backwards_new_3.jpg': 'backwards',
'pred_silhouettes\\backwards_new_4.jpg': 'backwards',
'pred_silhouettes\\backwards_new_5.jpg': 'backwards',
'pred_silhouettes\\blank_1.jpg': 'forwards',
'pred_silhouettes\\downwards_1.jpg': 'downwards',
'pred_silhouettes\\downwards_14.jpg': 'downwards',
'pred_silhouettes\\downwards_160.jpg': 'downwards',
'pred_silhouettes\\downwards_27.jpg': 'downwards',
'pred_silhouettes\\downwards_3.jpg': 'downwards',
'pred_silhouettes\\downwards_370.jpg': 'downwards',
'pred_silhouettes\\downwards_505.jpg': 'downwards',
'pred_silhouettes\\downwards_667.jpg': 'downwards',
'pred_silhouettes\\downwards_686.jpg': 'downwards',
'pred_silhouettes\\downwards_696.jpg': 'downwards',
'pred_silhouettes\\forwards_250.jpg': 'forwards',
'pred_silhouettes\\forwards_3.jpg': 'forwards',
'pred_silhouettes\\forwards_32.jpg': 'forwards',
'pred_silhouettes\\forwards_326.jpg': 'forwards',
'pred_silhouettes\\forwards_335.jpg': 'forwards',
'pred_silhouettes\\forwards_342.jpg': 'forwards',
'pred_silhouettes\\forwards_38.jpg': 'forwards',
'pred_silhouettes\\forwards_502.jpg': 'forwards',
'pred_silhouettes\\forwards_600.jpg': 'forwards',
'pred_silhouettes\\forwards_700.jpg': 'forwards',
'pred_silhouettes\\forwards_fake_1.jpg': 'forwards',
'pred_silhouettes\\forwards_no_arms_1.jpg': 'forwards',
'pred_silhouettes\\left_198.jpg': 'left',
'pred_silhouettes\\left_215.jpg': 'left',
'pred_silhouettes\\left_226.jpg': 'left',
'pred_silhouettes\\left_25.jpg': 'left',
'pred_silhouettes\\left_26.jpg': 'left',
'pred_silhouettes\\left_3.jpg': 'left',
'pred_silhouettes\\left_474.jpg': 'left',
'pred_silhouettes\\left_680.jpg': 'left',
'pred_silhouettes\\left_700.jpg': 'left',
'pred_silhouettes\\left_fake_1.jpg': 'left',
'pred_silhouettes\\left_fake_2.jpg': 'left',
'pred_silhouettes\\right_199.jpg': 'right',
'pred_silhouettes\\right_204.jpg': 'right',
'pred_silhouettes\\right_220.jpg': 'right',
'pred_silhouettes\\right_240.jpg': 'right',
'pred_silhouettes\\right_26.jpg': 'right',
'pred_silhouettes\\right_37.jpg': 'right',
'pred_silhouettes\\right_420.jpg': 'right',
'pred_silhouettes\\right_423.jpg': 'right',
'pred_silhouettes\\right_518.jpg': 'right',
'pred_silhouettes\\right_590.jpg': 'right',
'pred_silhouettes\\upwards_114.jpg': 'upwards',
'pred_silhouettes\\upwards_127.jpg': 'upwards',
'pred_silhouettes\\upwards_14.jpg': 'upwards',
'pred_silhouettes\\upwards_258.jpg': 'upwards',
'pred_silhouettes\\upwards_37.jpg': 'upwards',
'pred_silhouettes\\upwards_513.jpg': 'upwards',
'pred_silhouettes\\upwards_610.jpg': 'upwards',
'pred_silhouettes\\upwards_614.jpg': 'upwards',
'pred_silhouettes\\upwards_677.jpg': 'upwards'}
```

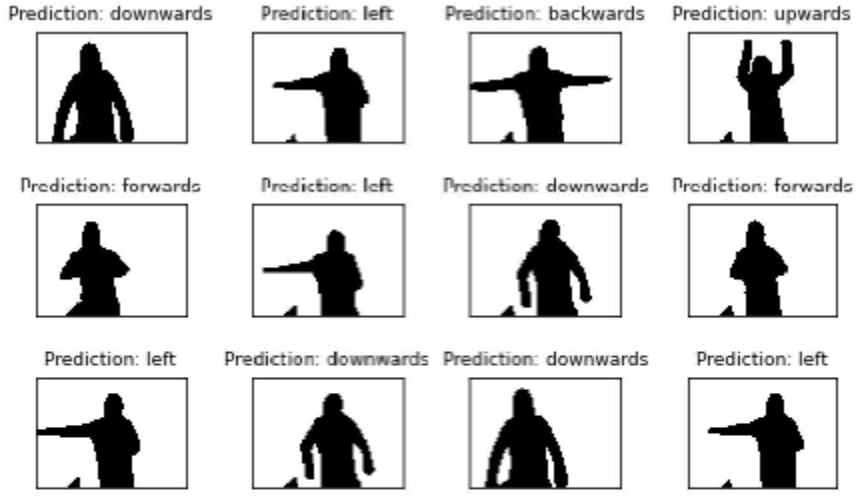
In [35]:

```
DATADIR = "C:/Users/joshu/OneDrive/A levels/Other/Crest/Crest code/imgRecognition/si

#CATEGORIES = ["backwards", "downwards", "forwards", "left", "right", "upwards"]
```

## Applying model to pred data

In [36]:

```
fig = plt.figure()
for i in range(12):
    path = DATADIR
    file = random.choice(os.listdir(path))
    plt.subplot(3,4,i+1)
    plt.tight_layout()
    img_array = cv2.imread(os.path.join(path,file) ,cv2.IMREAD_GRAYSCALE)
    plt.imshow(img_array, cmap='gray', interpolation='none')  # graph it
    prediction_pred = prediction(path+"/"+file,transformer)
    label = file.split('_', 1)[0].replace('.', '')
    #plt.title(("Label: " + label + "    " + "Prediction: " + prediction_pred),c='bla
    plt.title(("Prediction: " + prediction_pred),c='black',fontsize = 9)
    if label != prediction_pred:
        plt.title(("Prediction: " + prediction_pred + " \n Label: " + label),fontsiz
    #plt.show()  # display!

    plt.xticks([])
    plt.yticks([])
#note: these are technically unlabelled but I wrote their made up label in the file
```




In [37]:

```
#prediction function
def prediction(img_path,transformer):

    image=Image.open(img_path)

    image_tensor=transformer(image).float()


    image_tensor=image_tensor.unsqueeze_(0)

    #if torch.cuda.is_available():
    #    image_tensor.cuda()

    input=Variable(image_tensor)


    output=model(input)

    index=output.data.numpy().argmax()

    pred=classes[index]

    output = func.softmax(output,dim = 1)
    max_output,prediction_output = torch.max(output,dim=1)
    max_output = max_output.detach()
    max_output = max_output.numpy()*100
```

```
    max_output = max_output[0]

    return (max_output,pred)
```

In [38]:
```
path = DATADIR
file = 'left_fake_1.jpg'

img_path = path+"/"+file

image=Image.open(img_path)

image_tensor=transformer(image).float()

image_tensor=image_tensor.unsqueeze_(0)

output=model(Variable(image_tensor))
print(output)

index=output.data.numpy().argmax()
print(index)

pred=classes[index]
print(pred)
```

```
tensor([[  8.9370, -20.9948,  -1.1384,  99.1954, -83.8852,   1.9763]],
       grad_fn=<AddmmBackward>)
3
left
```

In [39]:
```
output.data.shape
```

Out[39]:
```
torch.Size([1, 6])
```

In [40]:
```
output.data
```

Out[40]:
```
tensor([[  8.9370, -20.9948,  -1.1384,  99.1954, -83.8852,   1.9763]])
```

In [41]:
```
output = func.softmax(output,dim = 1)
print("Total: ",output.data[:1].sum())
output.data[:1]
```

```
Total:  tensor(1.)
```

Out[41]:
```
tensor([[6.3281e-40, 0.0000e+00, 2.6625e-44, 1.0000e+00, 0.0000e+00, 5.9976e-43]])
```

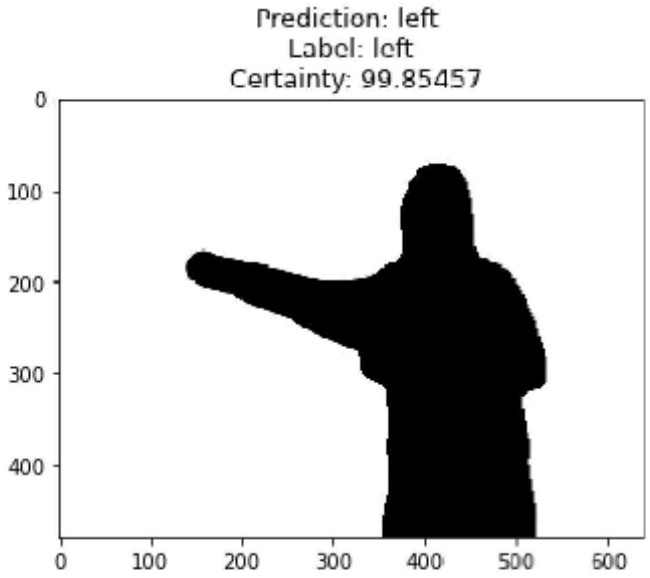
In [42]:
```
max_output,prediction_output = torch.max(output,dim=1)
print(max_output)
max_output = max_output.detach()
print(prediction_output)
print(max_output)
max_output = max_output.numpy()*100
print(max_output)
print(classes[prediction_output])
```

```
tensor([1.], grad_fn=<MaxBackward0>)
tensor([3])
tensor([1.])
[100.]
left
```

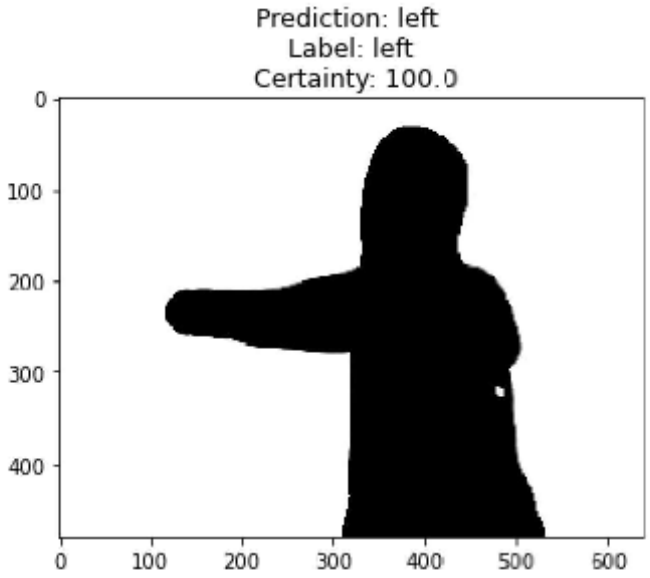
## For testing images at their exact location

In [43]:
```
def testing_img(file):
    path = "C:/Users/joshu/OneDrive/A levels/Other/Crest/Crest code/imgRecognition/s
    #file = random.choice(os.listdir(path))
    img_array = cv2.imread(os.path.join(path,file) ,cv2.IMREAD_GRAYSCALE)
    plt.imshow(img_array, cmap='gray', interpolation='none')
    prediction_pred = prediction(path+"/"+file,transformer)
    certainty = str(prediction_pred[0])
    prediction_pred = prediction_pred[1]
    label = file.split('_', 1)[0].replace('.', '')
    plt.title(("Prediction: " + prediction_pred + " \n Label: " + label + " \n Certa
    if label != prediction_pred:
        plt.title(("Prediction: " + prediction_pred + " \n Label: " + label + " \n C
```

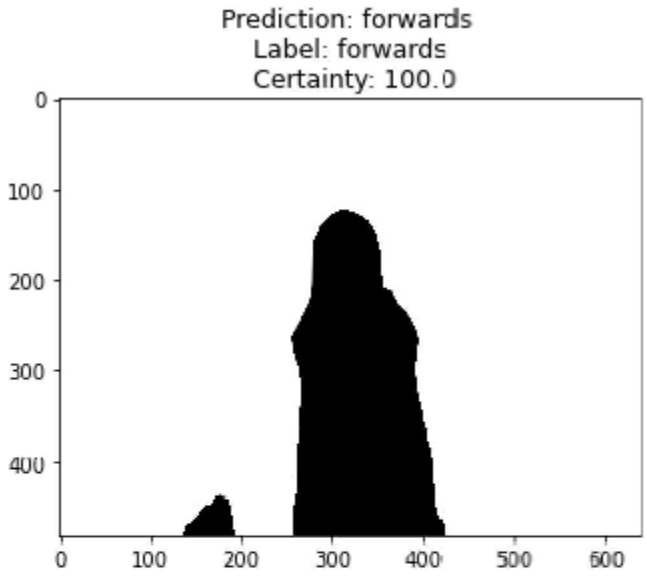
In [44]:
```
#testing model against fake images drawn in 3D paint
testing_img('left_fake_2.jpg')
```



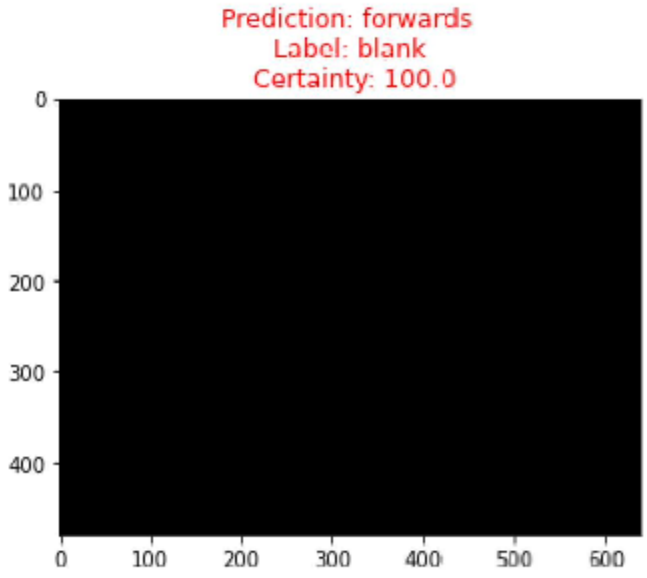

In [45]:
```
testing_img('left_fake_1.jpg')
```



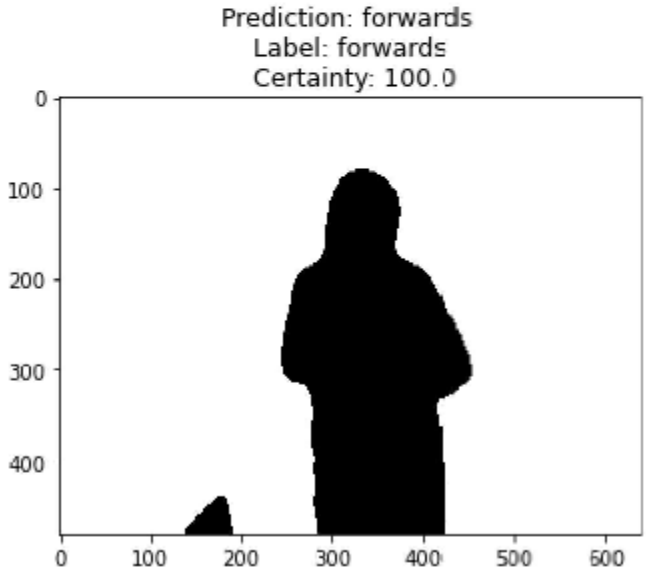

In [46]:
```
testing_img('forwards_no_arms_1.jpg')
```




In [47]:
```
testing_img('blank_1.jpg')
```




In [48]:
```
path = "C:/Users/joshu/OneDrive/A levels/Other/Crest/Crest code/imgRecognition/signa
testing_img(random.choice(os.listdir(path)))
```



## Signal Recognition Program

bwlivepc.py

```
import time
import cv2
import copy
from PIL import Image
import torch
import torch.nn as nn
import torch.nn.functional as func
from torchvision.transforms import transforms
from torch.autograd import Variable

classes = ['backwards', 'downwards', 'forwards', 'left', 'right', 'upwards']


# CNN Network
class ConvNet(nn.Module):
    def __init__(self, num_classes=6):
        super(ConvNet, self).__init__()

        self.conv1 = nn.Conv2d(in_channels=1, out_channels=12, kernel_size=3,
stride=1, padding=1)
        # Shape= (64,12,150,150)
        self.bn1 = nn.BatchNorm2d(num_features=12)
        # normalisation
        # Shape= (64,12,150,150)
        self.relu1 = nn.ReLU()
        # Shape= (64,12,150,150)

        self.pool = nn.MaxPool2d(kernel_size=2)
        # Reduce the image size be factor 2
        # Shape= (64,12,75,75)

        self.conv2 = nn.Conv2d(in_channels=12, out_channels=20, kernel_size=3,
stride=1, padding=1)
        # Shape= (64,20,75,75)
        self.relu2 = nn.ReLU()
        # Shape= (64,20,75,75)

        self.conv3 = nn.Conv2d(in_channels=20, out_channels=32, kernel_size=3,
stride=1, padding=1)
        # Shape= (64,32,75,75)
        self.bn3 = nn.BatchNorm2d(num_features=32)
        # Shape= (64,32,75,75)
        self.relu3 = nn.ReLU()
        # Shape= (64,32,75,75)

        self.fc = nn.Linear(in_features=75 * 75 * 32, out_features=num_classes)

        # Feed forward function

    def forward(self, input):
        output = self.conv1(input)
        output = self.bn1(output)
        output = self.relu1(output)

        output = self.pool(output)

        output = self.conv2(output)
        output = self.relu2(output)

        output = self.conv3(output)
        output = self.bn3(output)
```

```
        output = self.relu3(output)

        # Above output will be in matrix form, with shape (256,32,75,75)

        output = output.view(-1, 32 * 75 * 75)
        output = self.fc(output)

        return output


checkpoint = torch.load('C:/Users/joshu/OneDrive/A levels/Other/Crest/Crest
code/best_checkpoint.pth')
model = ConvNet(num_classes=6)
model.load_state_dict(checkpoint)
model.eval()

prediction = ''
score = 0


# prediction function
def prediction_func(image, transformer):
    image_tensor = transformer(image).float()

    image_tensor = image_tensor.unsqueeze_(0)

    input = Variable(image_tensor)

    output = model(input)

    index = output.data.numpy().argmax()

    pred = classes[index]

    output = func.softmax(output, dim=1)
    max_output, prediction_output = torch.max(output, dim=1)
    max_output = max_output.detach()
    max_output = max_output.numpy() * 100
    max_output = max_output[0]

    return max_output, pred

# Transforms
transformer = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),
    transforms.Resize((150, 150)),
    transforms.ToTensor(),  # 0-255 to 0-1, numpy to tensors
    transforms.Normalize((0.5), (0.5))])

def change_res(width, height):
    cam.set(3, width)
    cam.set(4, height)


cam = cv2.VideoCapture(0)
change_res(640, 480)


# parameters
threshold = 60  # binary threshold
blurValue = 41  # GaussianBlur parameter
bgSubThreshold = 50
# allow the camera to warmup
time.sleep(0.1)


while True:
    # capture frames from the camera
```

```
    ret, frame = cam.read()

    frame = cv2.bilateralFilter(frame, 5, 50, 100)
    frame = cv2.flip(frame, 1)

    grayImage = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(grayImage, (blurValue, blurValue), 0)
    ret, thresh = cv2.threshold(blur, threshold, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

    opencv_image=cv2.cvtColor(thresh, cv2.COLOR_RGB2BGR)
    color_converted = cv2.cvtColor(opencv_image, cv2.COLOR_BGR2RGB)
    pil_image=Image.fromarray(color_converted)
    score, prediction = prediction_func(pil_image,transformer)

    text_thresh = copy.deepcopy(thresh)

    cv2.putText(img = text_thresh,
                text = f"Prediction: {prediction} Certainty: {int(score)}",
                org =  (50, 30), fontFace = cv2.FONT_HERSHEY_SIMPLEX,
                fontScale=1, color = (0, 0, 255),
                thickness=1)

    cv2.imshow('black and white', text_thresh)

    # if the `c` key was pressed, break from the loop
    if cv2.waitKey(1) == ord('c'):
        break


cam.release()
cv2.destroyAllWindows()
```

# Bibliography


Abed, A., & Rahman, S. (2017). Python-based Raspberry Pi for Hand Gesture Recognition. *IJCA*, *173*(4), 18–24.

AI-SPECIALS. (2020, June 30). *Image Classification Using CNN From Scratch In Pytorch-Part 1 Training*. YouTube. https://www.youtube.com/watch?v=9OHlgDjaE2I

Alex. (2018, September 3). *How To Connect Quadcopter Motors And ESC*. DroneTrest Blog. https://blog.dronetrest.com/how-to-connect-motors-and-esc/

Ardupilot. (n.d.-a). *ESP32 WiFi Telemetry — Copter Documentation*. Ardupilot. Retrieved March 26, 2022, from https://ardupilot.org/copter/docs/common-esp32-telemetry.html

Ardupilot. (n.d.-b). *Installing Mission Planner — Mission Planner Documentation*. Ardupilot. Retrieved March 26, 2022, from https://ardupilot.org/planner/docs/mission-planner-installation.html

Ardupilot. (n.d.-c). *Joystick/Gamepad*. Ardupilot. Retrieved March 26, 2022, from https://ardupilot.org/copter/docs/common-joystick.html

Ardupilot. (n.d.-d). *Operation Using Only A Ground Control Station (GCS)*. Ardupilot. Retrieved March 26, 2022, from https://ardupilot.org/copter/docs/common-gcs-only-operation.html

Ardupilot. (n.d.-e). *Pixhawk Wiring Quick Start*. Ardupilot. Retrieved March 26, 2022, from https://ardupilot.org/copter/docs/common-pixhawk-wiring-and-quick-start.html

Ardupilot. (n.d.-f). *Radio Control Systems — Copter Documentation*. Ardupilot. Retrieved March 26, 2022, from https://ardupilot.org/copter/docs/common-rc-systems.html

Ardupilot. (n.d.-g). *Telemetry (landing Page)*. Ardupilot. Retrieved March 26, 2022, from https://ardupilot.org/copter/docs/common-telemetry-landingpage.html

Ardupilot. (n.d.-h). *Vibration Damping*. Ardupilot. Retrieved March 26, 2022, from https://ardupilot.org/copter/docs/common-vibration-damping.html

Asiri, S. (2021, April 22). *Building A Convolutional Neural Network For Image Classification With Tensorflow*. Medium. https://medium.com/@sidathasiri/building-a-convolutional-neural-network-for-image-classification-with-tensorflow-f1f2f56bd83b

*Autopilot Hardware Options — Copter Documentation*. (n.d.). Retrieved March 24, 2022, from https://ardupilot.org/copter/docs/common-autopilots.html

Banggood.com. (n.d.). *Emax XA2212 820KV 980KV 1400KV Brushless Motor For RC Drone FPV Racing*. Www.Banggood.Com. Retrieved March 23, 2022, from https://uk.banggood.com/Emax-XA2212-820KV-980KV-1400KV-Brushless-Motor-for-RC-Drone-FPV-Racing-p-918124.html?utm_source=googleshopping

Battery Equivalents And Replacements. (n.d.). *LiPo 3S Battery 101: All About LiPo 3S Batteries*. Battery Equivalents And Replacements. Retrieved March 26, 2022, from https://www.batteryequivalents.com/lipo-3s-battery-101-all-about-lipo-3s-batteries.html

Blade Helis. (2016, June 20). *Basics Of Drone Flight - Forward Fight*. YouTube. https://www.youtube.com/watch?v=8GROkV4S170

*Breadboard Electronics*. (n.d.). Science Buddies. Retrieved March 24, 2022, from https://www.sciencebuddies.org/science-fair-projects/references/how-to-use-a-breadboard?from=YouTube

*Brushless Vs Brushed Motor: Why You Should Know The Difference - CDZ*. (n.d.). Retrieved March 23, 2022, from https://cordlessdrillzone.com/drill-wars/brushless-vs-brushed-motor/

Choromanska, A., Henaff, M., Mathieu, M., Arous, G., & LeCun, Y. (n.d.). *The Loss Surfaces of Multilayer Networks*. Courant Institute of Mathematical Sciences New York. Retrieved March 26, 2022, from https://arxiv.org/pdf/1412.0233.pdf

Collabnix. (2021, December 5). *How To Control DJI Tello Mini-Drone Using Python – Collabnix*. Collabnix. https://collabnix.com/how-to-control-dji-tello-mini-drone-using-python/

Corrigan, F. (2020, May 6). *How A Quadcopter Works Along With Propellers And Motors - DroneZon*. DroneZon. https://www.dronezon.com/learn-about-drones-quadcopters/how-a-quadcopter-works-with-propellers-and-motors-direction-design-explained/

DJI. (2015, May). *FlameWheel 450 User Manual*. DJI. http://dl.djicdn.com/downloads/flamewheel/en/F450_User_Manual_v2.2_en.pdf

*Drone Motor Calculator*. (2018, August 24). https://www.omnicalculator.com/other/drone-motor

Flight Test Forum. (2017, April 9). *ESC To Flight Controller Question*. FliteTest Forum. https://forum.flitetest.com/index.php?threads/esc-to-flight-controller-question.33789/

*FPV Drone Flight Controller Explained - Oscar Liang*. (n.d.). Oscar Liang. Retrieved March 24, 2022, from https://oscarliang.com/flight-controller-explained/

FPV DroneWorks. (2020, July 15). *Flysky Transmitter*. YouTube. https://www.youtube.com/watch?v=cR1CX8fdgac

GalcoTV. (2015, September 11). *Advantages And Disadvantages Of Brushed And Brushless Motors - A GalcoTV Tech Tip*. YouTube. https://www.youtube.com/watch?v=Y7nQI2xM2as

Google. (n.d.). *Machine Learning Crash Course*. Google Developers. Retrieved March 26, 2022, from https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit

Heintz, B. (2019, December 3). *Brenner Gesture Detection*. Google Docs. https://docs.google.com/presentation/d/1UY3uWE5sUjKRfV7u9DXqY0Cwk6sDNSalZoI2hbSD108/edit

Holybro. (n.d.-a). *PM07 Power Management Board*. Retrieved March 25, 2022, from http://www.holybro.com/manual/PM07-Quick-Start-Guide.pdf

Holybro. (n.d.-b). *SiK Telemetry Radio V3 Quick Start Guide*. Retrieved March 25, 2022, from http://www.holybro.com/manual/Holybro_Sik_Radio_V3_Quick_Start_Guide.pdf

*Holybro X500 + Pixhawk4 Build | PX4 User Guide*. (n.d.). Retrieved March 24, 2022, from https://docs.px4.io/master/en/frames_multicopter/holybro_x500_pixhawk4.html

IBM Cloud Education. (2020, October 27). *Gradient Descent*. IBM. https://www.ibm.com/cloud/learn/gradient-descent

iforce2d. (2018, June 30). *Trying Gamepad/joystick Control In Mission Planner*. YouTube. https://www.youtube.com/watch?v=K61wJ9ps3TY

Intelligent Quads. (2021, September 20). *Drone Programming Libraries 2021*. YouTube. https://www.youtube.com/watch?v=XpkEi7tFZGc

Keim, R. (2020, February 6). *Understanding Local Minima In Neural-Network Training - Technical Articles*. All About Circuits. https://www.allaboutcircuits.com/technical-articles/understanding-local-minima-in-neural-network-training/

Liang, O. (2018, February 8). *Quadcopter PID Explained - Oscar Liang*. Oscar Liang. https://oscarliang.com/quadcopter-pid-explained-tuning/

MATLAB. (2018, October 16). *Drone Simulation And Control, Part 1: Setting Up The Control Problem*. YouTube. https://www.youtube.com/watch?v=hGcGPUqB67Q

MAVLink. (n.d.). *Introduction - MAVLink Developer Guide*. MAVLink. Retrieved March 27, 2022, from https://mavlink.io/en/

*Method For Characterization Of A Multirotor UAV Electric Propulsion System*. (n.d.). MDPI. Retrieved March 24, 2022, from https://www.mdpi.com/2076-3417/10/22/8229/htm

Nast, C. (2019, December 4). *Calculate The Thrust Force On Your Drone!* Wired. https://www.wired.com/story/calculate-thrust-force-on-a-drone/

Painless360. (2015, June 26). *(1/5) PixHawk Video Series - Simple Initial Setup, Config And Calibration*. YouTube. https://www.youtube.com/watch?v=uH2iCRA9G7k

Plitch, A. (2021, October 11). *Which Raspberry Pi Should I Buy?* Tom's Hardware. https://www.tomshardware.com/uk/how-to/raspberry-pi-buying-guide

PX4. (n.d.-a). *Joystick Setup | PX4 User Guide*. PX4. Retrieved March 26, 2022, from https://docs.px4.io/master/en/config/joystick.html

PX4. (n.d.-b). *Radio Control Systems*. PX4. Retrieved March 26, 2022, from https://docs.px4.io/master/en/getting_started/rc_transmitter_receiver.html

*PX4 Quick Start Guide*. (n.d.). Retrieved March 25, 2022, from http://www.holybro.com/manual/Pixhawk4-quickstartguide.pdf

Python Package Index. (n.d.). *Xbox360controller*. Python Package Index. Retrieved March 26, 2022, from https://pypi.org/project/xbox360controller/

QGroundControl. (n.d.). *Joystick · QGroundControl User Guide*. QGroundControl. Retrieved March 26, 2022, from https://docs.qgroundcontrol.com/master/en/SetupView/Joystick.html

STLFinder. (n.d.). *Drone Landing Gear 3d Models*. STLFinder . Retrieved March 24, 2022, from https://www.stlfinder.com/3dmodels/?search=DroneLandingGear

Talking Stuff. (2020, August 11). *Raspberry Pi - 5 Inch Touchscreen Installation*. YouTube. https://www.youtube.com/watch?v=-V4ppZjRVZw

*The History Of Pixhawk*. (n.d.). Retrieved March 24, 2022, from https://auterion.com/company/the-history-of-pixhawk/

UK Civil Aviation Authority. (2022a). *Prepare For The Theory Test*. UK Civil Aviation Authority. https://register-drones.caa.co.uk/individual/prepare-for-the-drone-theory-test

UK Civil Aviation Authority. (2022b). *Registering A Drone or Model Aircraft*. UK Civil Aviation Authority. https://register-drones.caa.co.uk/individual

*What Does A Motor Speed Controller Do? - 4QD - Electric Motor Control*. (n.d.). 4QD - Electric Motor Control. Retrieved March 24, 2022, from https://www.4qd.co.uk/docs/speed-controller-overview/

*Wifi PPM (no App Needed)*. (2017, December 21). Instructables. https://www.instructables.com/Wifi-PPM-no-App-Needed/