Major Project Document_Group1

by JanakiRam Basava

Submission date: 21-Jun-2022 11:17AM (UTC-0400)
Submission ID: 1858548452
File name: Major_Project_Document.docx (20.8M)
Word count: 18895
Character count: 101977

THE FLYING PET

Automated Drone system using Computer Vision

Abstract

The recent techno-trend is frequently including drones one way or the other. These have attracted increasing attention due to usage in communications, photography, agriculture, surveillance and numerous public services and many more vast ranges of applications. But still, there had been a minimal effort of including these automated drones in commercial or general usage. These applications range from greater yet simpler applications such as grocery shopping, pet care taking, and babysitting to sophisticated applications such as surveillance and security.

This paper is a direct implication to answering these commercial and general requirements with the help of automated drones. We aim to propose a collaboration between a drone and machine learning so that the drones are automated to move on their own. The IoT device in the drone is loaded with a machine learning algorithm that can identify and distinguish between faces and track them and also drone mechanics to make drones move accordingly. Built-in with the power of object recognition and automatic navigation, the drone can follow a particular, distinguishable object, either as a part of surveillance or assistance. With photography and videography becoming an integral part of everyone's lives, this drone can help film sports videos and professional cinematography.

Keywords: Surveillance, security, Drone, Machine Learning, recognition, automation, djitellopy, face tracking, web page, rc controls, pet tracking.

INDEX

Contents	Page No.
Chapter 1: INTRODUCTION	01
1.1 About Drone	01
1.1.1 Types of drones	01
1.1.1.1 Single Rotor Helicopters	01
1.1.1.2Multiple Rotor Drones	01
1.1.1.3 Fixed Wing Drones	01
1.1.2 Uses of drones	02
1.1.2.1 Military Purposes	02
1.1.2.2 Delivery systems	02
1.1.2.3 Rescue systems	02
1.1.2.4 Agriculture sector	03
1.1.2.5 Space exploration	03
1.1.2.6 Wild Conservation	03
1.1.2.7 Historical Conservation	03
1.1.2.8 Medicine	04
1.1.2.9 Still Photography	04
1.2 Motivation	04
1.2.1 Commercial Usage	05
1.2.2 Surveillance	05
1.3 Problem Statement	06
1.3.1 Use Case Focused	06
1.4 Scope	06
1.5 Objective	06

Chapter 2: Literature Survey 07
2.1 Detecting Object 07
2.2 Object Tracking 10
2.3 Applications 12
2.4 Course resources 18
2.5 Existing System 18
Chapter 3: Proposed System 19
Chapter 4: Software Design 21
4.1 Use-case Diagram 21
4.1.1 System 22
4.2 Sequence diagram 23
4.2.1 Face recognition 25
4.2.2 Processor 26
4.2.3 Drone communication 27
4.3 Class diagram 28
4.3.1 System 28
4.3.2 Drone 29
4.4 Activity diagram 30
4.4.1 Face recognition model 30
4.4.2 System 31
CHAPTER 5: Technologies & Requirements 33
5.1 Drone Dynamics 33
5.1.1 Drone Components 34
5.1.1.1 Frame 34
5.1.1.2 Electronic Speed Controllers & Motors 34
5.1.1.3 Propellers 34
5.1.1.4 Flight Controller 35
5.1.1.5 Inertial Measurement Unit 35
5.1.1.6 Drone Camera 36
5.1.1.7 Drone Battery 36

5.1.2 Quadcopter- drone Physics 36
5.1.2.1 Quadcopter Frame Configuration 36
5.1.2.2 Basic functional movements of a drone 37
5.1.2.3 Translational movements of a drone 38
5.1.2.4 Requirement for rotational motion 38
5.1.2.5 Rotational Motion of a Drone 39
5.1.3 Quadcopter-drone Flight Coding 43
5.2 The Prototype – Ryze Tello Drone 44
5.2.1 Prototype Drone Components 44
5.2.2 Prototype Drone Connections 44
5.2.3 Prototype Drone Dynamics- Movements 45
5.3 Software Technologies 46
5.3.1 python 46
5.3.2 OpenCV 47
5.3.3 Flask 47
5.4 Functional Requirements 48
5.5 Non-Functional Requirements 49
5.6 Software Requirements 49
5.7 Hardware Requirements 49
Chapter 6: Coding 51
6.1 The MVC Architecture 51
6.2 Face Recognition 53
6.2.1 Training the face to our model 54
6.2.2 face_rec_util.py 55
6.3 Drone Functionalities 57
6.3.1 Face-Tracking Using Drone 57
6.3.2 drone_manager.py 59
6.4 Web Application 63
6.4.1 config.py 63
6.4.2 layout.html 64

6.4.3 index.html 65

6.4.4 controller.html 66

6.4.5 server.py 71

6.4.6 base.py 74

Chapter 7: Testing and Results 76

7.1 Drone's functionality with respect to facial tracking 76

7.1.1 A basic code to fly the drone 76

7.1.1.1 Result: basic code to fly the drone 76

7.1.2 Drone Calibration 77

7.1.2.1 Result: Drone Calibration 78

7.1.3 Understanding GPU Performance 78

7.1.3.1 Result: Understanding GPU Performance 78

7.1.4 OpenCV file issues faced while face tracking 79

7.1.4.1. Result: OpenCV files issues faced while face tracking 79

7.1.5 Setting PID values 80

7.1.5.1 Result: Setting PID values 81

7.1.6 Setting FB values 81

7.1.6.1 Result: Setting FB values 83

7.1.7 IMU -lighting conditions 83

7.1.7.1 Result: Testing conditions 83

7.1.7.2 Result: IMU -lighting conditions 84

7.1.8 Face Recognition & tracking: Confidence 85

7.1.8.1 Result: Face recognition and tracking: Confidence 85

7.2 Website 87

7.2.1 UI 88

7.2.1.1 Result: Home page or index page 88

7.2.1.2 Result: Battery alert 88

7.2.1.3 Result: Controller page 89

7.2.2 Controlling drone 89

7.2.2.1 Result: Drone moving down with down 89

arrow

7.2.2.2 Result: Drone towards its right with rotate option 90

7.3 Face tracking through the website 90

7.3.1 Commands log, stream and RC controls 91

7.3.1.1 Result: Commands log, stream and RC controls 91

7.3.2 Final test 92

7.3.2.1 Result: Final test 92

CHAPTER 8: Conclusion 94

8.1 Conclusion 94

8.2 Future Scope 95

8.2.1 Improvement of Web Application 95

8.2.2 Migration to mobile Application 95

8.2.3 Code and Technical changes 95

8.3 RoadMap: Motivation to Future Scope 97

Bibliography 99

LIST OF FIGURES & TABLES

Contents	Page No.
Fig 2.1: Some images of object detection in images as examples	11
Fig 2.2: Before and after detection of people	13
Fig 2.3: Positive and negative frame snapshots	15
Fig 2.4: Showing different perspectives of observation of drone	16
Fig 3.1: Proposed Overview	19
Fig 3.2: Interface overview	20
Fig 3.3: Architecture Diagram	20
Fig 4.1: Use case diagram of the entire system	22
Fig 4.2: Sequence diagram of face recognition	25
Fig 4.3: Sequence Diagram of the processor	26
Fig 4.4: Sequence Diagram of Tello drone communication	27
Fig 4.5: Class diagram of the entire system	28
Fig 4.6: Class diagram of drone initialization	29
Fig 4.7: Activity diagram of Face recognition model	30
Fig 4.8: Activity diagram of the entire system	31
Fig 5.1: ESCs and motors in a drone	34
Fig 5.2: Airfoil shape propeller	35
Fig 5.3: quadcopter + vs X	37
Fig 5.4: thrust control	38
Fig 5.5: Forces behind Rotational motion	39
Fig 5.6: Roll motion	40
Fig 5.7: Roll Control	40
Fig 5.8: Pitch motion	41
Fig 5.9: Pitch Control	41
Fig 5.10: Yaw motion	42

Fig 5.11: Yaw Control 42

Fig 5.12: PID controller gate 43

Fig 5.13: Tello Drone Components 44

Fig 5.14: Tello communication flow 45

Fig 5.15: Internal code for ‘move’ API 46

Fig 6.1: MVC model Architecture 51

Fig 6.2: Directory structure using MVC 52

Fig 6.3: Trained face 54

Fig 6.4: RC controls graph 58

Fig 7.1: Shows the result of flying the drone in one environmental condition. 77

Fig 7.2: Trained image in HEIC format 79

Fig 7.3: PID controller circuit equation 80

Fig 7.4: Drone moving too close 82

Fig 7.5: Drone moving too far 82

Fig 7.6: Naveen training image 83

Fig 7.7: Facility used in drone testing 84

Fig 7.8: Detection failure due to low light 84

Fig 7.9: Facility with windows covered 85

Fig 7.10: Raja training image 85

Fig 7.11: System detecting target face 86

Fig 7.12: TPP of drone detecting target face 87

Fig 7.13: index page 88

Fig 7.14: battery alert 88

Fig 7.15: Controller page 89

Fig 7.16: Drone obeying ‘Down’ command 90

Fig 7.17: Drone obeying ‘Right’ command 90

Fig 7.18: Shows command logs received at the server-side 91

Fig 7.19: Shows the streaming facility on the website. 91
Fig 7.20: Shows the RC control's command logs received at the server-side 92
Fig 7.21: Screenshot from a screen-recording of the website and server-side put side to side 93
Fig 7.22: The snap from the third-person point of view 93
Fig 8.1: RPA 97
Fig 8.2: CA 97

Chapter 1

INTRODUCTION

1.1 About drones

Unmanned Aerial Vehicles (UAVs) also known as drones – first designed as military weapons are now equipped with high-resolution cameras and industry-leading sensors and have been quickly adopted in various applications like surveillance, traffic and human control, safety of citizens, commercial use, and more.

1.1.1 Types of drones

Drones are categorized in the following way based on their functionalities and hardware components.

1.1.1.1 Single Rotor Helicopters

These are tiny helicopter like models which are run or powered by either electricity or gas. As it run on gas and with only one blade, improved stability is observed in these types of drones thus making the drones travel longer distances. UAVs are commonly used to transport heavy things, such as LIDAR equipment.

1.1.1.2 Multiple Rotor Drones

Drones with several rotors are often the tiniest on the market and are also light-weight. In terms of altitude, velocity, displacement, they are limited. These drones often stay in flight for about 20-30 min while even carrying a camera along with it.

1.1.1.3 Fixed Wing Drones

These drones are regular aero planes, but these don't have rotors, instead have wings that produce lift. These drones are often fueled by gasoline rather than electricity, allowing for up to 16 hours of flight time. These drones must take off and land on runways in the same way that planes do because to their size and structure.

1.1.2 Uses of drones

1.1.2.1 Military Purposes

Drones are most commonly used in the military application. In the 1940s, crude drones are used by the British and American military to spy on their enemies and Axis countries. Drones now include functionalities such as identifying laser ranges, executing airstrikes, infrared imaging, making them far more advanced than previous UAVs. The most popular aircraft used in the military currently is the MQ-9 Reaper. The drone is around 36 feet long and equipped with various weapons and abilities to perform reasoning on the collected data. It can fly unobserved up to 50,000 feet in the air.

1.1.2.2 Delivery systems

Self-contained autonomous UAVs that bring all types of products to your door are commonly known as delivery drones. “Last mile” is also another name for these drones as they make deliveries from local businesses or warehouses to the destination without any human inputs, so these are known as automated. Merchants and grocery shops all over the country are using drones as an effective alternative to vehicles and delivery personnel. These drones can deliver things up to 55 pounds without requiring you to leave your home. A number of prominent corporations are experimenting with delivery drones in a variety of ways.

1.1.2.3 Rescue systems

To deploy humans to rescue area is not at all safe all the time; even a small mistake can lead to disaster in that kind of situation because of severity and scope of the tragedy. Drones can help in this situation. An AUV (autonomous underwater vehicle) is sent into the ocean to rescue capsized boats and even drowning individuals by assisting them. If the drone is large enough, they can even help in rescue. Drones are used in snowy places to locate people who are stuck. The K-MAX is a helicopter manufactured by Kaman, who builds aircrafts. It does not require a pilot and is efficient enough to move cargo weighing a minimum of 6,000 pounds. The K-MAX has already assisted in the fight against fires in China and Australia.

1.1.2.4 Agriculture sector

In the agriculture industry, drones have been proven to be beneficial at all stages of the process, offering farmers many alternatives to improve productivity by enhancing the process and lowering the physical strain on farmers and thereby, improving crop yield. The use of UAVs makes field surveys, planting over fields, following animals, and predicting crop output easier, saving time for farm employees.

1.1.2.5 Space exploration

Unmanned aircraft designed for space flight has been tested by NASA and the Air Force of the U.S. The X-37B unmanned aerial aircraft, which resembles a miniature space shuttle, is classified by the Air Force. For more than 719 days (last two years), it has been discreetly orbiting the world, breaking the track-record for the most extensive mission by a pilotless aircraft. The main intent behind X-37B, according to the Air Force, is "reusable spacecraft technologies for America's future in space" as well as "conducting experiments that can be returned to and studied on Earth."

1.1.2.6 Wildlife Conservation

Drones are a more efficient and cost-effective technique to save animals. Tracking animal numbers with humans on ground is nearly impossible. Drones became an eye in the sky. Wildlife scientists can use drones to track a wandering heard of animals, from Borneo's orangutans to the Great Plains' bison, to learn more about the species' well-being and their habitats. In the regions of Africa and Asia, where poaching is prevalent, drones are useful in the battle against the illegal activity. Thus, drones are helpful for conservation.

They are even employed to assist with forestry operations all around the world. These drones fly over fire-ravaged forest floors, dropping seed containers carrying seeds, fertilizers, and nutrients that will help trees regenerate. Since the 1990s, 300 million acres of land has been deforested. Seed-planting drone technology can complete a reforestation project that would take humans 300 years to complete.

1.1.2.7 Historical Conservation

The use of unmanned aerial vehicles (UAVs) is becoming increasingly important in historical preservation initiatives. The 3D reconstructions of historical locations around Europe, including Chernobyl, Ephesus, Turkey, and Jewish graves are being mapped out by the use of drones. Historical preservationists can use 3D imaging to rebuild vanished locations while looking for hints about culture and architecture from this vantage point.

1.1.2.8 Medicine

UAVs are increasingly used in transporting medical supplies and merchandise to off-the-grid Alaskan settlements. Alaskans are using drones to bring life-saving medical supplies instead of depending on their traditional methods like snowmobiles, dog sledges, or rescue vehicles that can't locomote in frosty regions.

Transplant recipients are provided with donated organs with the help of drones. Using a specially developed drone, a kidney was recently transferred from a hospital to another in less time than it is done using human resources and their methods. This might assist to speed up the often excruciatingly slow rate at which donations are distributed. Dispatchment of organs is frequently carried out by aircrafts - private or commercial. Four in hundred organ deliveries are delayed by at least two hours due to lapses and errors in judgement. In such situations, drones can make organ transplants comparatively more secure and safe by saving a great deal of time.

1.1.2.9 Still Photography

Photographers who shoot large-scale aerial photographs are widely benefited from the use of drones. Have you ever wanted to have a bird's point-of-view of your city, location, or a beach? Drones, which are developed with high resolution cameras and which are photography-specific, can give one a new perspective on the places or objects and many more.

1.2 Motivation

Drones have been adapted into various disjoint fields of transportation, photography and videography. Adding a few more features like camera visualization, microphone and intelligence in decision making, these drones can be taken to much more heights in applications of surveillance and commercialization. The domestication of drones is the next-in-line step (which happens with every technology: fax to smartphones). With the help of cognitive automation, we can make drones and systems understand, think, behave, and even solve stuff like humans. Cognitive automation will use quantitative metrics in human judgement. It also augments the human intelligence which helps the drone to assess the environment or needs multiple simulations to understand the surroundings for developing the near-autonomous drones which not only do as it is trained but can think and adapt to the situation and surroundings and perform the tasks.

1.2.1 Commercial Usage

Automated drones play a significant role in commercial industries and their usages. The four major fields where these drones can be used are farming, entertainment, delivery and logistics as well as filmmaking and photography. Because of its diversity, the commercial drone sector has attracted corporations of various types to invest heavily in it. The utilization of the use of commercial drones has risen dramatically in recent years in some European and Asian countries, such as the United Kingdom and Japan. The Federal Aviation Administration (FAA) in the United States, on the other hand, has yet to create regulations for commercial drone operations (Ballve,2015).

1.2.2 Surveillance

Surveillance systems based on drone deployment have received a lot of interest since the introduction of aerial drones. Aerial drone-based surveillance systems are more adaptable to demand due to their better controllable mobility. Drone surveillance can be viewed as a justifiable, impartial activity that serves the interests of all parties involved, or as an oppressive technology that serves the interests of a few at the expense of others. A well-functioning public safety system requires constant monitoring and observation and this cannot be provided by the use of a static camera

or sensor as these cannot travel along with the subject to all the places, but the drone which is automated can travel and follow the subject or the target if trained using machine learning algorithms or cognitive deep learning. For example, hundreds of thousands of people may attend a festival or gathering. Because of the low frequency of events, constructing a static surveillance system is inefficient. As a result, aerial drone networks become a viable option.

1.3 Problem statement

This thought of extending drone application to surveillance and monitoring made us motivated and passionate about this project to introduce "domestic drones" which could be used for everyday utilities and monitoring systems.

1.3.1 Use Case focused

A drone that follows an individual just like a pet. In order to follow a target and only the targeted person, the drone uses facial recognition and tracks his/her/their movements and follows them.

1.4 Scope

The scope of this project is limited. The drone will have an autonomous flying control but the control is only with the help of facial recognition and tracking. The controls of the drone can also be manually overridden. We have only considered letting the drone track only a single person now despite them being in a group. No tracking of two different people or selective tracking of people is implemented yet. The solution also is generalised in order to broaden the scope of the project's uses i.e. the above use-case can solve many problems or ease many tasks at once which would be discussed in the later chapters.

1.5 Objectives

- A drone that tracks an individual through his facial recognition and follows the person.
- An application to give the user control over the drone.

Chapter 2

Literature Survey & Existing System

2.1 Detecting object

2.1.1 "*You only look once: Unified, real-time object detection, 2016*"

In prior work, broad classifiers were re-purposed for the specific aim of object detection. Rather, J. Redmon et al. [1] regard object detection to be a sort of regression issue with the help of associated probabilities of classes as well as spatially separated bounding boxes.

To predict the probabilities of various classes as well as bounding boxes YOLO generally uses a single neural network. YOLO improves its detection performance by training on the entire image.

The fast version of YOLO runs at over 150 frames per second. This could imply that the system can process and analyse video streams in real time, with a latency of as little as 25 milliseconds. Other real-time systems achieve less than half the mean average precision of YOLO. YOLO makes half the number of backdrop mistakes as Fast R-CNN.

The image given as input is split up into a SXS grid by the technology. If the item's centroid falls within a grid cell, that cell will identify the article. On the ImageNet 2012 validation set, the accuracy was 88 percent. The main flaw in this paper is that YOLO establishes strong spatial constraints on bounding box predictions because only two boxes and a class can be predicted from each grid cell. Small items that emerge in groups, such as a flock of birds, are difficult for the model to handle. They gain a better understanding of yolo, and the author of this research attempted to demonstrate that YOLO is superior to Fast R-CNN in terms of background errors.

2.1.2 "*Adapting YOLO Network for Ball and Player Detection, 2016*"

M. Buric et.al. [2] presents research on the task of recognizing players in real-world handball photos using sports balls. In the machine learning application, there are a variety of detection models.

Their reference model, identified by the letter Y, is a pre-trained YOLOv2 model with 608 608 input picture size and weights pre-trained on the COCO dataset. Other COCO dataset classes, such as human and sports ball, are included in the pre-trained model.

The model (Y+) is a fine-tuned version of the pre-trained model Y, which uses 394 images from the custom data set. Because the model's training was determined to be unstable, only a small number of epochs were used. It can be argued that existing little items become too small when scaled, resulting in a mistake, while using the same data set, which consisted of big photographs with small annotated objects, to train future models with higher input resolution without difficulty.

The third model (YB) was trained only for the ball class using transfer learning and both public custom parts of the dataset, which totaled 1837 images. In this and later editions, the input image resolution was increased from 608 × 608 in the initial model to 720 x 720.

The input image size was increased, and the YB model was only trained and evaluated for the sports ball class. The model performed better on distant objects, as expected, however the public part of the dataset's recognition of ball objects was severely deteriorated, with several false-positive ball object detections on human figures.

The YPB model fared much better than the YB+ model in both human and ball classes. The YPB+ and YPBF models' detection on the public dataset did not decline, allowing them to be used in sports other than handball with greater ease.

2.1.3 "*Drone-based object counting by spatially regularized regional proposal network, 2017*"

In [3], The authors of this study attempted to solve the item counting problem in drone-based videos. To change or convert the multi-dimensional picture space to a non-negative integer the algorithm that is used is Object counting algorithms that are currently learning a regression model. It's been difficult to implement a counting automation mechanism for deep-learning models. The object counting system is a fully convolutional deep learning neural network that takes any image as input and generates object recommendations that are most likely to include the instance. They observed that there are some specific patterns (layout patterns) that may be used to forecast objects that would move in the same direction and adjacently for a set of object instances.

The score for Spatial Patterns is A collection's bulk of objects follow a similar pattern. In a parking lot, cars will line up in one way, and ships will regularly hug the coastline. The model is implemented using Caffe. They constructed all of the baseline methods based on the findings/study of the VGG-16 networks, which contain 3 fully-connected layers and 13 convolutional layers, to ensure equality in evaluating the efficacy of various standard approaches. All of the baseline layer settings and the proposed model are optimized using the CARPK dataset or the PUCPR+ dataset, utilizing weights pre-trained on ImageNet 2012.

2.1.4 "*Faster R-CNN: Towards real-time object detection with region proposal networks, 2015*"

[4] discusses the usage of a faster R-CNN detection algorithm. Region proposal algorithms are for Object detection to hypothesize object placement. More improvements include SPPnet and Fast R-CNN. As a result, the detection networks' runtime is lowered. With the detection network in this research, the authors create an RPN that stores the full picture convolutional functions.

In, RPN input is an image and output is a series of rectangular item recommendations, each with a score for objectness. Using RPN+VGG, Fast R-CNN result for unshared features is 68.5 percent, a little higher than the SS baseline. RPN+VGG is marginally inflated than the SS baseline. A unified, deep learning based totally item identification device can now run at five-17 frames per second using this technology.

2.1.5 "*SSD: Single shot multi-box detector, 2016*"

W. Liu et.al. through [5], They went over the SSD in detail. To recognize items in a photograph, this technique employs a single deep neural community. The accuracy improves as the image clarity improves. If the image is 512x512 pixels, for example, the accuracy is 76.9%. This approach only works at a speed of 7 frames per second. The Single Shot Detector (SSD) is a faster version of the previous cutting-edge technology (YOLO). A feed-forward convolutional network is used in the SSD method.

PASCAL VOC has risen to 74.3 percent mAP for SSD, up from 63.4 percent mAP for YOLO. In high-traffic scenarios, such as sports, SSD may recognize people.

2.1.6 "*Focal loss for dense object detection, 2017*"

In [6], Object detectors are described by T.Y. Lin et al. The R-CNN-popularized two-stage approach has produced the highest accuracy object detectors to date. One-stage detectors are unable to outperform top-performing detectors due to a significant class imbalance. The fundamental reason is determined to be the large foreground-background class imbalance that occurs during intensive detector training. The image focus loss is meant to handle the single-stage object recognition scenario in which the foreground and background classes are severely uneven during the training. The RetinaNet De-tector network is made up of a foundation network and two job-determined subnetworks.

2.2 Object Tracking

2.2.1 "*The unmanned aerial vehicle benchmark: Object detection and tracking, 2018*"

In [7], The study's goal is to create a new UAV benchmark that includes a new level difficulty and focuses on complex circumstances. The 14 different sorts of features include atmospheric state, flying height, camera view, occlusion, and vehicle type. Then, for each task, the most up-to-date state-of-the-art algorithms are used to conduct a complete quantitative analysis.

The three key parameters that effect the MOT mission are weather conditions, flying altitude, and camera view. When shooting video, the time of day, night, fog, and other meteorological variables all play a role in defining the lighting. There are two types of object detection frameworks.

The SOT is in control by deep learning-based techniques and correlation filters. Trackers are divided into three categories:

i. Correlation filter (CF) trackers with handcrafted features.
ii. CF trackers with deep features.
iii. Deep trackers.

2.2.2 "*Vision meets drones: A challenge, 2018*"

In [8], The study presents a high-scale visually perceptible object detection and tracking benchmark with the goal of improving drone systems' visual comprehension tasks.

In the VisDrone2018 benchmark, four tasks are benchmarked and evaluated: Tracking for single objects and multiple objects, detecting objects in videos and photos.
Bounding boxes for a range of various items are uniquely marked in each image. Two types of relevant annotations that have been offered to us are the occlusion ratio and the truncation ratio.

All of the following functions are considered exceptionally difficult in the suggested dataset because of characteristics such as large scale, occlusion, position volatility, and quick motion.

Fig 2.1: Some images of object detection in images as examples

2.2.3 "*Convolutional pose machines, 2016*"

In [9], Pose Machines developed an approach to learning implicitly rich spatial representations using sequential prediction, which is described in the article. A predictor has been trained to generate predictions for each image in this convolutional pose machine. As input, the previous stage's image characteristics and belief maps are utilized. The idea maps allow the CPM to develop spatial models of part interactions which are rich image-dependent by providing an expressive non-parametric encoding of spatial location uncertainty for each part in the next stage.

Learning in CPMs, key-point localization based on local picture evidence, sequential prediction based on learned spatial context characteristics, and key-point localization based on local picture evidence are some of them. CPMs provide a complete architecture for dealing with the majority of structured prediction problems in CV without the need for graphical-model inference (computer vision). Because the posture refines uncertainty-preserving beliefs between stages by strengthening communication, the authors demonstrated that a sequential architecture is made up of implicit convolutional networks and is also capable of learning spatial models.

2.3 Applications

After looking at some standards and implementations in object detection as well as object tracking, let now move on to some of the applications of drone technology in other fields. The main intention of this project is to utilize the drone technology for commercial and general purposes and the core of the application is, drone following a particular person by recognizing him by face.

2.3.1 "*Unsupervised human detection with an embedded vision system on a fully autonomous UAV for search and rescue operations, 2019*"

In [10], It discusses the use of unmanned aerial vehicle (UAV)/drone technology in SAR missions. Search and rescue missions are one of the most difficult applications for such airborne robots because crises can occur at any moment and in any location, even if by accident.

The majority of the training dataset used is images from movies filmed by the relevant authors in order to provide a sufficient size dataset for the required model. In order to create their own unique training dataset, the authors tagged all of the swimmer photographs. As a result, bounding boxes in each of these photographs were created using an open-source labelling program.

This project involved incorporating machine learning methods and models into a completely autonomous UAV/drone that are used for rescue in detection of individuals who are in danger in the disaster in real life maritime. The image processing hardware, which is attached to the drone and uses the Nvidia Jetson TX1, can process video in real time, removing the requirement for video transmission for processing in a station that is located on ground and return to the UAV with the results, which might create considerable processing time delays The planned rescue system, on the other hand, have been established for swimmer identification in open waters; nevertheless, it is difficult to detect people in places like wetlands, and it does not provide emergency assistance for those engaged in sports activities.

Fig 2.2: Before and after detection of people in open waters left and right image respectively.

2.3.2 "*A convolutional neural network approach for assisting avalanche search and rescue operations with UAV imagery, 2017*"

In [11], the search and rescue operations are narrowed down to avalanche-related missions. Avalanches, which are large masses of snow, ice, and boulders sliding swiftly down a mountainside, kill around 150 people every year throughout the world. Avalanche victims, for example, could have collided with barriers and been injured while being transported by snowslides or falling over a cliff on the avalanche's path. When an avalanche comes to a halt, it settles like a hefty rock, making it nearly impossible for anyone trapped beneath it to move. The depth to which patients are buried, the amount of clear airway available, and the degree of physical injuries all have an impact on their chances of survival.

The drone selects fields of concern from the frames in the video using image segmentation, then employs a method established on Convolutional Neural Networks to represent the image and train an SVM classifier to recognize objects. In addition, the study describes a Hidden Markov Models in which there is a post-processing method drawn in it that improves the classifier's judgement by using the association between consecutive frames in the video.

The effect prediction performance of the post-processing technique can be above or below the threshold. These changes - positive or negative - are most common between consecutive frames in which the classifier's prediction transitions from one state to another. Finally, utilizing drones outfitted with cameras, the team has developed a mechanism to support avalanche SAR operations.

The Unmanned Aerial Vehicles are used to collect Electronic Health Record images of the avalanche debris and to pick areas of concern inside the image. The obtained image is then handled by a system that includes a technique involving pre-processing, an appropriate image descriptor is drawn out using a Convolutional Neural Network which is pre-trained. For object detection, a trained linear Support Vector Machine classifier was used, as well as a Hidden Markov M-based post-processing approach.

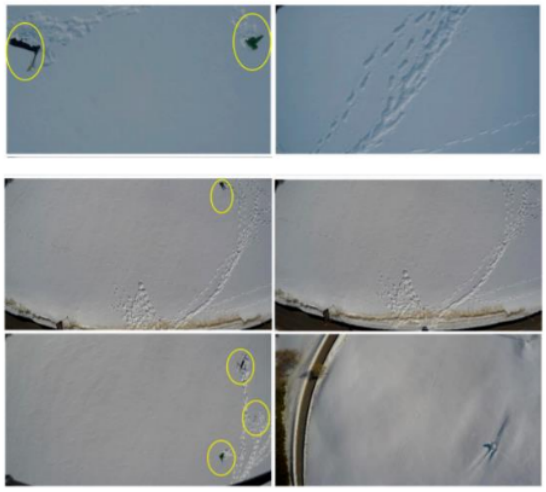


Fig 2.3: Positive and negative frame snapshots taken from the second dataset which are left and right respectively. Yellow circles represent the objects of interest (skis, jacket to simulate bottom half-buried victim).

2.3.3 "*Towards using Unmanned Aerial Vehicles (UAVs) in Wilderness Search and Rescue, 2016*"

In [12], video mosaicking one of the top issues identified was the need to improve the clarity and usability of the drone's search video. To provide more chance and stability in observation, the scientists developed a temporally local mosaic. According to Interaction Studies 10 (3) 2009, the term mosaic is used in CV (computer vision) to describe the effect of "stitching" UAVs in WiSAR 14 integrate many frames of video from a mobile camera observing a static environment to form a single larger-area image.

A user interface that can be described as "augmented virtuality" has been developed (Milgram & Kishino, 1994). Because the user interface combines satellite pictures, a map, and topography data into a cohesive model that is displayed as a virtual 3D world, it is extremely easy to use. The virtual environment, which includes (a) a map surface projected by real-time video and (b) a virtual UAV depicting the real UAV's position and pose, as well as the relative posture of the gimbaled camera, supplements the real data from the drone, such as video and telemetry. Different viewpoints are used by the drone for different observations.

(a) Chase perspective plus carrot-and-camera control model.

(b) Track-up perspective.

(c) North-up perspective.

(d) Pilot perspective.

Fig 2,4: Showing different perspectives of observation of drone.

In [13] Roman Bartak et.al., Discusses situations where the drone is controlled by an external computer, but the drone is still operated manually the majority of the time. Video frames are treated as matrices in order to detect objects. The operation is carried out utilizing the Tracking-Learning-Detection algorithm as the basic algorithm. C. A. Baker et al. described how contrast enhancement and image segmentation were used, then a CNN for human detection in their study [14]. To produce synthetic search and rescue datasets for their rescue application, they employ a 3D game editor. By integrating the initiation prototype with then SVM model, [15] provides a novel method for finding people trapped in an avalanche. A. J. Gallego et al focus in [16] Humans on the sea are being detected by a UAV furnished with a multi-spectral camera. An altered MobileNet convolutional neural network framework is employed in this method. The research of J. Sun et al. and colleagues in [17] did not provide any findings, but it did lay the groundwork for future research. It's an object recognition strategy that blends UAV hardware suggestions with semi-supervised and supervised machine learning approaches. T. Y. Lin et al. discuss cascade CNN in [18]. Cascade R-CNN is a object detection algorithm for multi-stage purposes that boosts detection by constantly increasing intersection. Cascade C-RNN was used for various applications, followed by CenterNet and RetinaNet. The study by et. al. in [19] suggests that GPS signals could be used in search and rescue operations. Assuming that the wounded individual has a mobile gadget turned on, a GSM radio wave from the mobile gadget is utilized to record the wounded individual's location based on the strength of the transmission and the drone's GPS position. On the Okutama-action dataset, R. Geralds et al. developed a system for recognizing individuals and action recognition while generating GPS points [20]. For identifying objects, a system that has been refurbished to MobileNetv2 and POINet was employed. The Tiny YOLO V3 framework was combined with the NVIDIA Jetson X1 microprocessor in [21], resulting in a platform for detecting persons in water.

All these applications are very much convergent towards the goal, but these are concentrated in a particular field and they are narrowed down to a single purpose. But theirs is more of a feature centered implementation rather than an application oriented.

2.3.4 "*Face Tracking for Flying Robot Quadcopter based on Haar Cascade Classifier and PID Controller, 2021*"

The main goal of this work was to create or improve a face detection system that could track face objects. The face detection technique is written in the Python and OpenCV programming languages. Mini drones are commonly selected because of their small and light structure, which makes it simple to conduct tests while causing no harm. The main advantage of using a quadcopter DJI Tello drone was having features like VPS (Vision Positioning System) and an onboard camera.

The primary reason for consulting this document was to learn more about the SDK. The Mini Quadcopter DJI Tello connects through WiFi and includes an app for Android or iOS. DJI has issued a Software Development Kit (SDK) that includes a number of read and write commands for use with UDP connection. The sensory data of the DJI Tello drone can be accessed and any issue commands that are meant to move which are based on the control signal that we provide can be done using the djitellopy library. A dual-ended full-duplex connection is established for receiving and sending instructions among the drone and the laptop or device and to have a video streaming of the drone on to a laptop a one-way half-duplex connection is established, these connections are the two fundamental network streams. For face recognition python and OpenCV are used by the drone once a very solid connection is established with the laptop with the live streaming function in it.

These pre-programmed read and write commands, which are utilized to move the drone and carry out the activities, are crucial. Damià Fuentes Escoté was responsible for translating the SDK to the Python programming language [23]. this Github repository is consist of the all the commands and as well as the methods of SDK in phyton programming language which makes it easier for us to understand the methods and implement them in our work.

2.4 Course resources

In the course [24], a website was developed for the purpose of controlling a DJI Tello drone using Tello SDK. The front-end part of the website provided all the options and controls needed to carry out a full-pledged drone movement.

So, references were made, from the course, for the sake of front-end appeal and had to include the course as a gesture towards the reference.

2.5 Existing System

Drone programming automation is discussed in the course [25], in which the drone tracks and follows 'any' individual. This project emphasizes the drone following any face. Not a just a particular target face, but any face comparatively closer to the drone than other. This is simply an implementation and do not have a broad range of applications. For sure, if it was developed further in the future, it may reap useful applications, but at the present place, it does not. ‘The flying pet’ project is a couple of steps ahead of this ‘any face’ project, as here the project not only follows a face, but follows a target face. This reaps enormous range of applications, like in surveillance, tracking and such. So, in comparison, ‘the flying pet’ project is much more ahead and pretty much novel than ‘any face’ project and can be considered superior in terms of applicable scope. The team consulted this project for minor help in aspects like the components of a drone, as well as how the drone flies. In addition, the training included information on the installations and basic movements, as well as image acquisition.

Chapter 3

Proposed System

Fig 3.1 Proposed Overview

The proposed project is a drone, acting as the body, and an associate processor, acting as the mind. The processor is maintained individually and is connected to the drone for the purpose of controlling. The processor is coded with the commands like takeoff, land, streamon, streamoff and the directions for the drone to move.

The Processor is equipped with a Machine learning model to differentiate objects from one-another, identifying the faces in the frame and distinguishing the anonymous from the target face. And give input to the drone to follow the target face and helps to find target face and follow the target even though multiple faces are on the frame. The main intention is that the drone sends the environment data that it had collected through its camera and IR sensor, so that it may send the data to the processor and the may take appropriate commands for the drone.

An IR device is arranged as an auxiliary to the drone, it is able to detect the current elevation of the drone. For the prototype the IR device is coming attached to it.

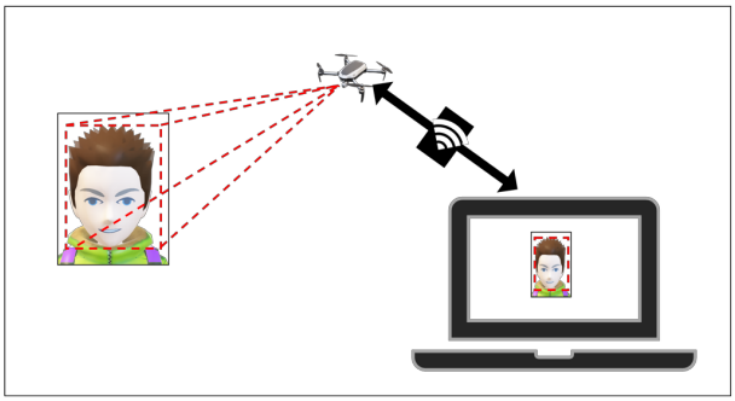


Fig 3.2 : Interface overview

The drone, acting as the body, is used for locomotion and takes commands from the processor regarding the direction and speed of the movement, and follows the target based on the input given using the face recognition. There is an option for manual

overriding of the instructions from end user. If the target face is missing from the program then the code will movement not send any command to the drone. A security measure where the drone lands automatically if it do not explicitly receive any commands in a 7 second span.

This image is analyzed by the automation software for provided instruction by the processor to the drone. The drone then follows the target wherever it may move and revolves with the target by fixing the face as per given fb_area.

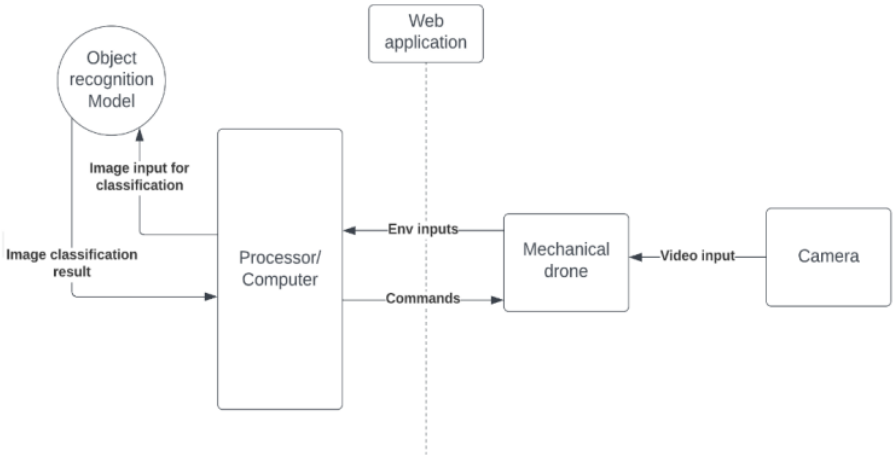


Fig 3.3 Architecture Diagram

Chapter 4

Software Design

4.1 Use-case diagrams

The driving force behind use case diagrams is to capture the dynamic aspects of the system. Though, the description is very broad for describing purpose, since the purpose of the other 4 diagrams: activity, sequence, State chart & collaboration) is the same. Investigation is done with a specific purpose that sets it apart from other 4 illustrations.

Use case diagrams are made use of for collecting system's requirements, with external & internal influences. Most of the requirements are designed ones. As a result, features and actors are identified when analysing the system for recording use cases. When a preliminary task is done, use-case diagrams are shaped for displaying the outside view.

In short, use-case diagrams assist with the following functions:

- Used for system's requirements collection
- Used for getting the bird's-eye sight of a system.
- Regulate both internal & external factors which are influencing the structure.
- Display interaction of requirements as the actors.

Use case diagrams considered requirement analysis for higher levels of the system. While a system's requirements are analysed, in use cases functionalities are captured. Use cases can be defined as system features described in an organised way. Actors are the second one related to use cases. Actors are objects that interact with the system. Actors are external applications, internal applications, or human users. These factors need to be considered when creating use case diagrams.

- As a use case, functionalities will be signified.
- Actors
- Relationships between actors and use-cases

4.1.1 System

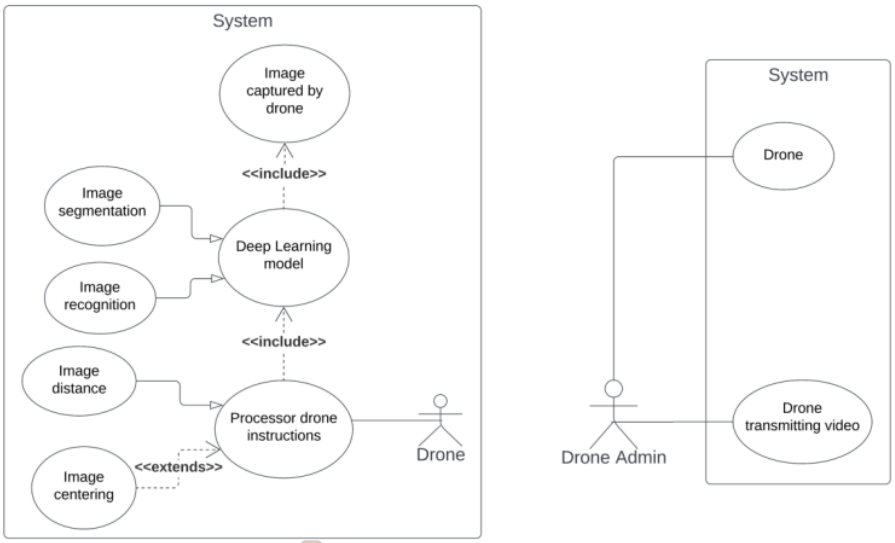


Fig 4.1 Use case diagram of the entire system

The above displayed is the Use-case diagram of the overall system of the project. This use-case diagram displays the behavior of the systems and helps the developer in understanding the requirements of the overall system.

- The execution starts with the system capturing an image and transferring the image to the machine learning model for processing. This model implements image segmentation and image recognition on the particular frame.
- And then this face identification 'metadata' is sent to an algorithm which then calculate the directions for the drone to move, so that the target face is centered in the frame.
- These directions are then sent to the processor, which commands the drone accordingly.
- The Drone admin, end-user who controls the drone, consists of the access to the video transmitted by the drone. Drone Admin can manually override the control to the drone, where he/she can observe the video transmission.

This work flow briefs the requirements for the project, and also narrates part of execution flow.

4.2 Sequence diagrams

A sequence diagram is a kind of interplay diagram for explaining how hard and fast objects interact with each other (and where in order). Business people and Software developers use the diagrams for understanding the requirements of new documents and systems existing processes. Event scenarios and event diagrams are aliases for sequence diagrams.

Companies and other organizations can benefit from using sequence diagrams as a reference. Create the following sequence diagram

- Describes the particulars of the UML use-case.
- Model the logic of complex procedures, functions, or operations.
- Find out how components and objects interrelate to complete the process.
- Plan & understand specific features of current/future scenarios
-

The following scenarios lend themselves well to the use of a sequence diagram:
Usage scenarios are expressions that describe how to use the system efficiently. This ensures that the logic of all possible system usage scenarios is executed.

Method logic:

Simply as you can use UML series diagrams to discover the logic of use instances, you could also use UML series diagrams to discover the common sense of any function, process, or complicated system. Collection diagrams are an extremely good manner to symbolize provider common sense whilst you think about a carrier as a high-level technique used by distinctive customers.

Vision sequence diagram:

Any Vision-sequence diagram which is created can be uploaded into Lucid chart. Lucid charts a great Microsoft Visio alternative that imports .vdx and supports.vsd file imports. Almost all the images on this site's UML section were created with Lucid chart

Object:

An object has a country, a lead, and a persona. The structure and path of objects which are, for all intents and functions, indistinguishable are depicted in the fundamental class. Within the diagram, each object represents a selected instance of magnificence. Order case is an item that's not named.

Message:

A message is an exchange of information between two items that trigger an event. Information is sent from the control convergence source point to the control convergence destination point by means of a message.

Link:

An existing association between two objects containing a class. This means an association between the opposite classes. If the object is linked to itself, use the image hover adjustment.

Lifeline:

As it moves down to show the passage of time. This dashed line represents the events that occur sequentially on the object during the recorded process. Lifelines can start with an engraved rectangle or an actor's symbol.

Actor:

Shows the entities that interact with the system or that are outside the system.

4.2.1 Face recognition

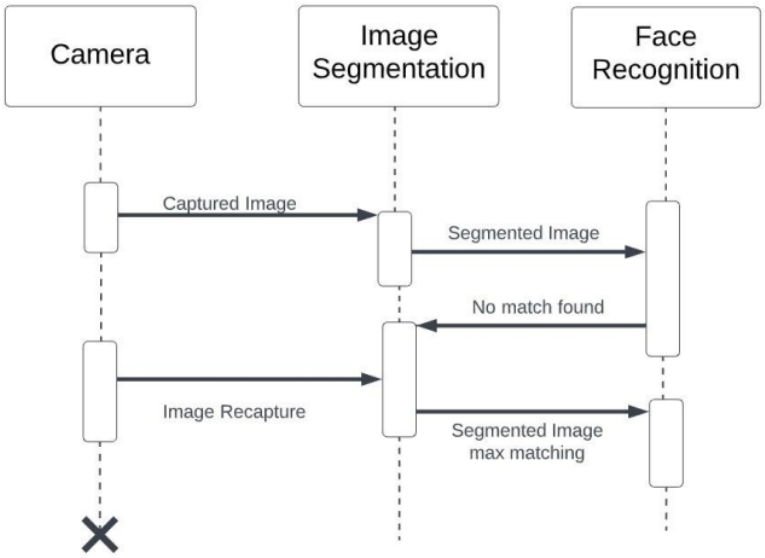


Fig 4.2 Sequence diagram of face recognition

This sequence diagram depicts how all the components in the project interacts and the sequence of interaction, of the particular portion of image processing in the total system.

- Drone used for prototype purposes, DJI Tello, comes with an integrated camera. The camera is kept of continuous streaming mode for the project. The images captured are sent towards the processor for image segmentation.
- After the image is segmented, for each and every face identified in the image, the face is sent towards face recognition. For the face that is matching with the target face by face-recognition module, corresponding commands will be operated upon the drone as discussed before.
- This whole sequence will be repeated continuously for each frame captured by the drone camera, for the remaining of the drone session.
- Thus, the system provides a continuous process for following the face.

4.2.2 Processor

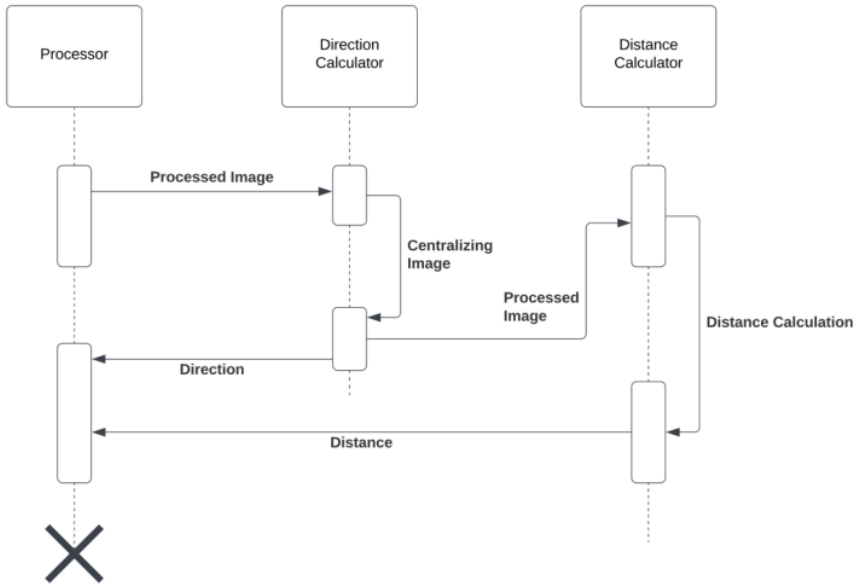


Fig 4.3 Sequence diagram of the processor

This particular diagram depicts on how the commands given to the drone are calculated using the image recognition input to the processor.

- After receiving the processed image from the machine learning model, the processor transfers the image to the 'Direction Calculator'.
- This Direction Calculator calculates the distance and direction of the centre of the identified face to the centre of the frame and scripts the commands to the drone to move in its direction.
- The Direction Calculator always ensures that the centre of the identified face and centre of the frame are aligned.
- After this, the same processed image received by the Direction Calculator is then forwarded to the Distance Calculator.
- Distance calculator calculates the distance for the drone needs to move towards or backwards of the face.
- Distance Calculator measures the ratio of the area of the bounding box covering the face to the area of the frame, implies that if the area is small, the face is far away and the drone needs to move closer to the face, and if the area is large, the face is too close and the drone needs to move farther back.

4.2.3 Drone communication

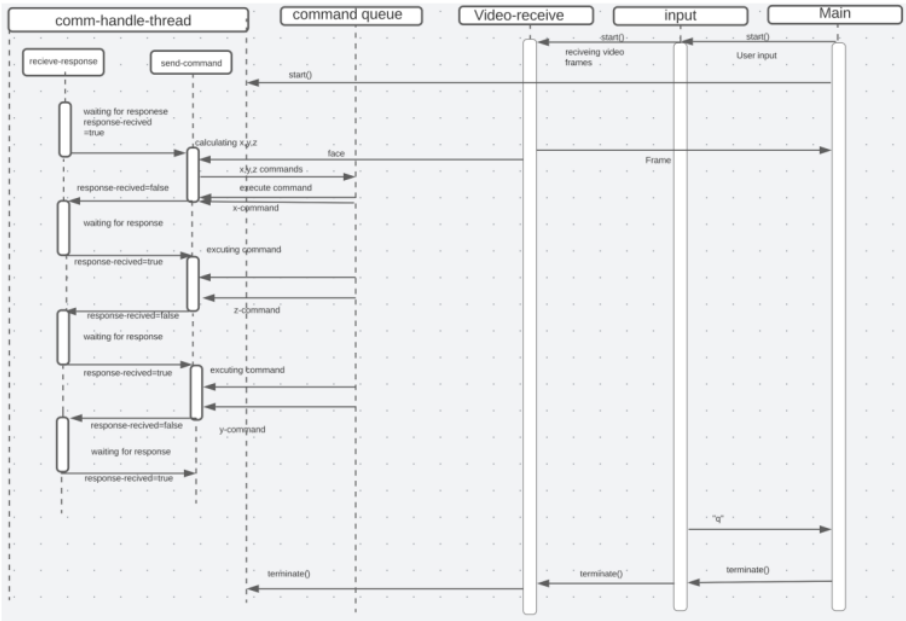


Fig 4.4 Sequence diagram of Tello drone communication

This overly complex sequence diagram explains the communicational sequence that undergoes between the drone and the processor. The process of how the drone sends the video recordings to the processor. And also, how the drone receives and interprets the commands.

- The comm-handle-thread maintains and verifies the commands received by the drone and executes it.
- The command queue maintains the remaining commands that needs to be executed.
- The video and the inputs need to be in a sequence for it to be transferred to the processor.

4.3 Class diagrams

4.3.1 System

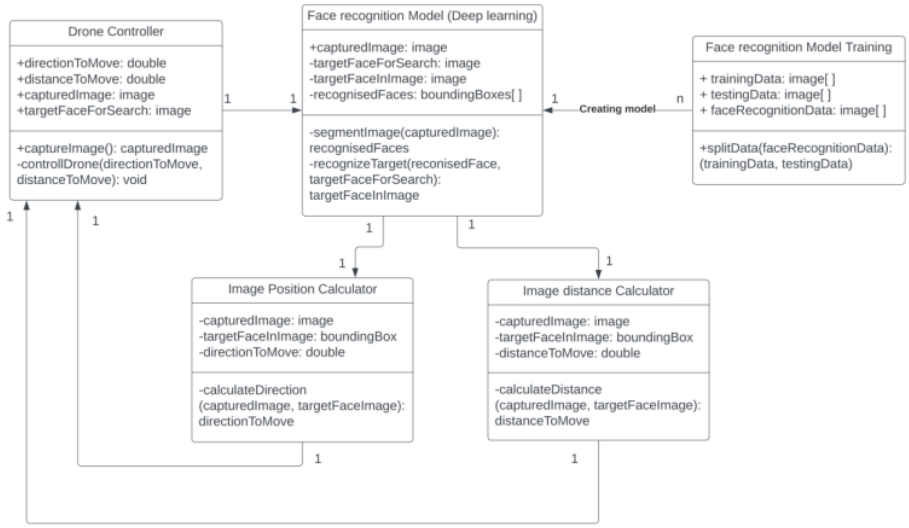


Fig 4.5 Class diagram of the entire system

This class diagram describes the outer view of the operations and attributes maintained by the whole system.

- Drone controller, in this prototype is a processor that maintains the attributes for the directions and distance commands that needed to be conveyed to the drone. The processor also maintains the image captured and its processed version.
- The face recognition model, used for identifying the target face in a frame, maintains the attributes such as captured image, the target face which is to be searched for in the frame, and the bounding box coordinates of the identified image in the frame.
- The face recognition model also carries out the operations of segmentation of image and recognizing the target face from the group photo.
- Two auxiliary objects for calculating the commands that are sent to the drone are Image position calculator and Image distance calculator.
- Image position calculator maintains the attributes of captured image, the target face and a result variable, direction to move.
- For calculating the direction to move, Image position calculator maintains the operations such of calculation the direction, by comparing the position of face to the center of the frame.

- Image distance calculator maintains the attributes of captured image, target face and a result variable, the distance to move.
- To calculate the distance to be moved by the drone, Image distance calculator conducts an operation of 'calculateDistance', which goes through face area comparison.

4.3.2 Drone

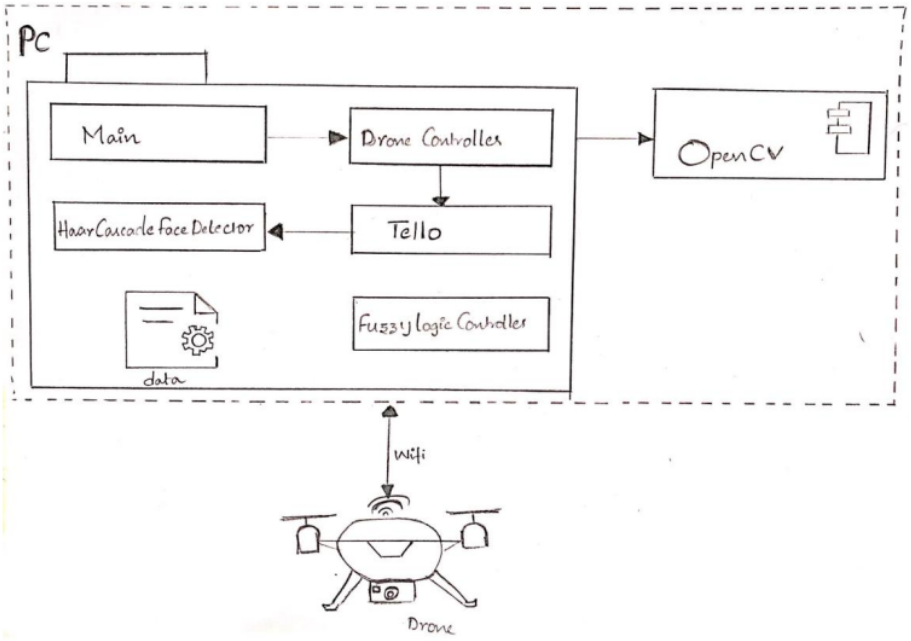


Fig 4.6 Class diagram of drone initialization

The class diagram depicts the attributes and operations required for the drone communication.

- As the main function in the web application gets executed, it contacts the Drone_controller model for all the operation it provides, so that the processor may control the drone through commands.
- The Drone_controller gets assistance from tello package for drone framework and HaarCascadeFaceDetector for the xml relations in face recognition.
- All these are processed through OpenCV libraries and the fuzzy logic is maintained in the process of drone following the target face.

- The data generated through the image processing and face identification will then be sent to the main function. With the assistance of functions calculating distance and direction, the processor builds up the commands that needs to be sent to the drone.
- For all this execution to be successful, attributes and operations are maintained by each class utilized in the program, namely DroneController, OpenCV, and the web application.

4.4 Activity diagrams

4.4.1 Face recognition model

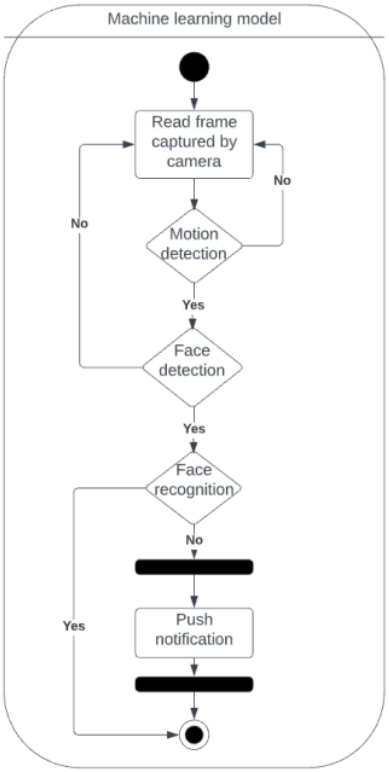


Fig 4.7 Activity diagram of Face recognition model

The activity diagram is of the machine learning model. This describes in a clear fashion the flow of work through the model for comparisons of two faces.

- Firstly, the model reads the frame captured by the camera, and then it evaluates the frame for motion detection by comparing it with the previous frame.

- Then, after the motion is detected, the model furthers evaluates the frame for a face, any face. And if any face is distinguishable from the frame, it marks it with a bounding box.
- And then all the faces with the bounding boxes are then matched against the target face. All the images will surely get a positive comparison value with the target face.
- The face with the largest confidence is considered to be the required target face, and the marks it and pushes a notification to the processor stating the required face is found in the frame.
- To avoid the detection of faulty faces, the minimum confidence required for a face to qualify was set to 50%.

4.4.2 System

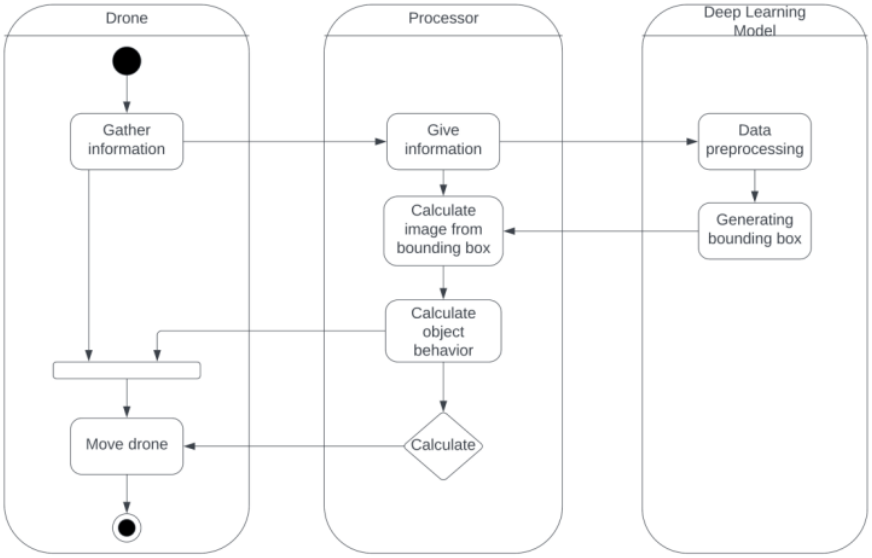


Fig 4.8 Activity diagram of the entire system

The whole activity of the system can be seen above. The interactions between the hardware drone, the processor that controls it and the deep learning model which instructs the processor.

- The drone gathers the information from the environment through camera and IR sensor, and sends this information to the processor.
- The processor segregates the information and send the image date to the face recognition deep learning model for face identifications.

- The deep learning model identifies all the faces in the frame, and target face from them, and generates bounding box around it.
- This bounding box is then again sent to Processor. The processor, with help for other auxiliary algorithm calculates the distance and direction in which the drone has to move, and generates the commands.
- These commands are then sent to the drone, which then acts them out, and thus implies that the drone is following the face.

Chapter 5
Technologies & Requirements

To understand the technologies and their requirements we need to know about the basic drone dynamics. Knowing drone dynamics will help in finding suitable software technologies to achieve the target use case.

5.1 Drone Dynamics

From the days of its uses in the military to their current habitual and usage as a toy, drones have come a long way in terms of technology. With improvements in battery and MEM technology, we are now able to design a drone of the size of a palm.

The quadcopter (quad–four propellers) design is the most used to build a drone. A copter with a single propeller would only move in one direction i.e. up and down like an elevator (of course the mechanics are different for both the functionalities). A two-propeller copter would be harder to balance while travelling. Similarly, a three-propeller copter would be hard to balance due to the forces (generated by the propeller's spin) and thrust generated with respect to the mass of the copter.

There are two ways to design a quadcopter body. They are called the plus shape + or the X shape. There are no functional differences between the two shapes of the design. The designers may use the shape which makes the drone components like sensors controllers sit on the body perfectly. Along with that, most designers go for a shape which looks appealing.

5.1.1 Drone Components

Though it fits in a pocket, there are several components in a drone, thanks to the advanced engineering of microprocessors and MEM technology.

5.1.1.1 Frame

A drone's frame is the body of the drone which holds all the components intact. The shape and design of the frame depend on the number of rotors or propellers used. As discussed earlier, a quadcopter which will have 4 rotors can be of two shapes + and X. We call each extending point an arm, so for a quad, there are four arms.

5.1.1.2 Electronic Speed Controllers and Motors

There are always motors and controllers attached to each arm in the frame. With the function of these controllers and BLDC motors, the levitation and piloting of the drone are done. The **Electronic Speed Controller** acts as an interface between the motor and the processor and operates the motor by telling it how fast/slow it should go.

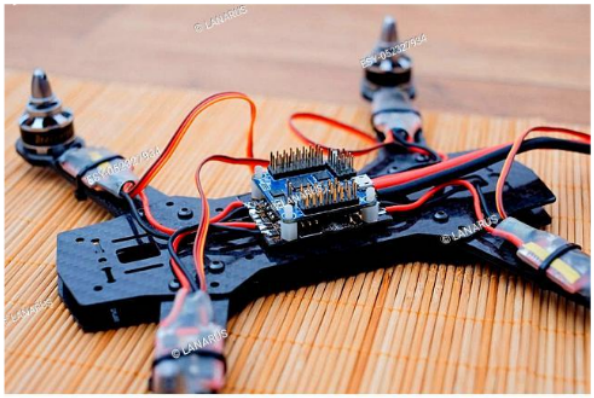


Fig 5.1 ESCs and motors in a drone

5.1.1.3 Propellers

The propellers are the wings of the drone. They are connected to the motors and they rotate with the speed of the motor's spin. Every propeller is designed in a way to push the air down while rotating which creates low pressure on the top of the propeller while at the bottom a higher pressure is created. Thus, due to the pressure difference, the drone is pushed up. Similarly, when the speed is decreased the pressure will vary on

Fig 5.2 airfoil shape propeller

both sides of the propeller make the drone come down. This is achieved with the airfoil shape of the propeller.

5.1.1.4 Flight Controller

The flight controller is like a motherboard for the drone. Later in this chapter, we will various drone motions and how a series of such motions are required to perform a small task like taking off or moving forward/backwards. We usually call the flight controller the brain of the drone which makes more sense when we look at the way the flight controller works. Our brain stimulates a lot of actions by giving appropriate directions to all the body parts, in the same way, the flight controller looks after all the necessary movements to perform the drone tasks.

Suppose the drone gets hit by a sudden gust of wind, it will be impossible for the operator to fix all the propellers' speeds within seconds to prevent the drone from crashing. This is where the flight controller comes in handy. To achieve this, the flight controller requires a lot of input from various sensors.

5.1.1.5 Inertial Measurement Unit (IMU)

The IMU of a drone is the most important sensor which measures rotation and acceleration. It consists of 3-axis accelerometers, 3-axis gyroscope sensors and magnetometers. A MEMs-based barometer sensor is used to check the air pressure.

To enhance the drone's flight and also to avoid the noise in the data from only a single sensor and magnetic deflections, modern drones use techniques called **sensor-fusion.** This technique is using two or more sensor data to improve the accuracy.
Few sensors which are widely used in drones are:

- GPS Sensor
  This provides the precisive altitude of the drone when fused with the data from the IMU's three sensors.
- IR Sensors
  These help for thermal imaging and also to check the distance below the drone.
- Ultrasonic Sensors
  These help to navigate the distance and obstacle avoidance.

5.1.1.6 Drone Camera

We have been seeing a lot of aerial footage and many of the applications discussed in chapter 1 are with a visual recording from the drone. Modern drones have high-resolution cameras such as 4K.

Cameras are also used as sensors for positioning and balancing the drone.

5.1.1.7 Drone Battery

A battery is to supply the power to the drone. Having a good battery is essential for the drone as it should be light in weight, powerful enough to power all the sensors and motors, and should be long-lasting and rechargeable.

5.1.2 Quadcopter-drone Physics

5.1.2.1 Quadcopter Frame Configuration: + & X

We discussed that a quadcopter's frame can be configured in two shapes. Now let us look at how and why it should be configured thus.

In general, the propellors in opposite arms should spin in opposite directions. This is to cancel out the torques generated by the spin of the propellors. If all the propellers spin in the same direction, then the torque generated in the opposite direction of the spin would make the drone swift away in a non-desired direction.

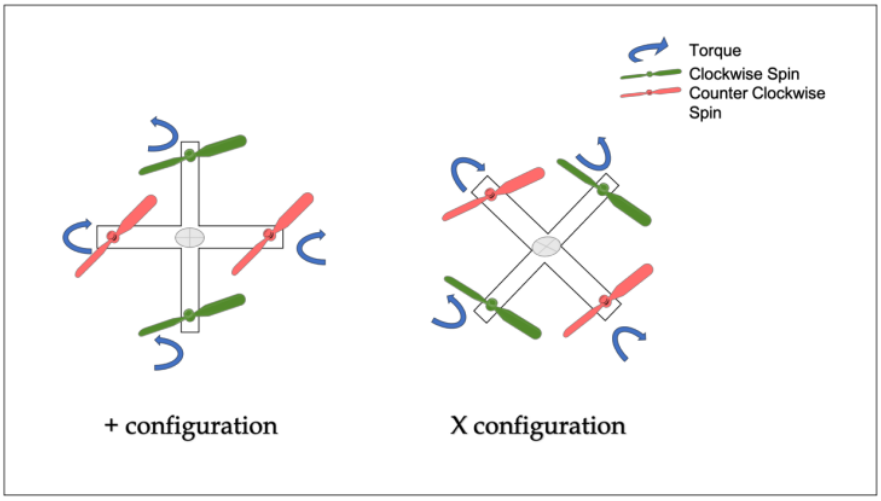


Fig 5.3 quadcopter + vs X

In fig 5.3, we can see the representation of + and X configurations. The propellers spinning in one direction should be kept parallelly and this is universally followed by all the drone designers. The reason for such a notation is to control the yaw's interaction with the roll and pitch of the drone. These topics will be discussed later in the chapter.

5.1.2.2 Basic functional movements of a drone

There are 3 translational motions and 3 rotational motions for a drone. Together with the three rotational motions along with one translational motion, the entire drone piloting is done. These motions are moving up and down (translational motion) and roll, pitch and yaw. Due to the 3-directional motions we need a 3d accelerometer and 3d gyroscope(discussed in 5.1.1.5).

Before looking at the rotational motions of the drone let us look at the translational motions.

5.1.2.3 Translational movements of a drone

The three translational movements of a drone are up/down, right/left and forward/backwards.

5.1.2.3(a) move up and down

For a drone to move Upwards and Downwards all the propellers' speeds should be uniform. When all the propellers' speeds are reduced, which reduces their spin and reduces the overall **thrust** generated, thus lowering the drone down. When speed is increased the drone moves up. **Landing** and **taking off** operations are also done in the same way.

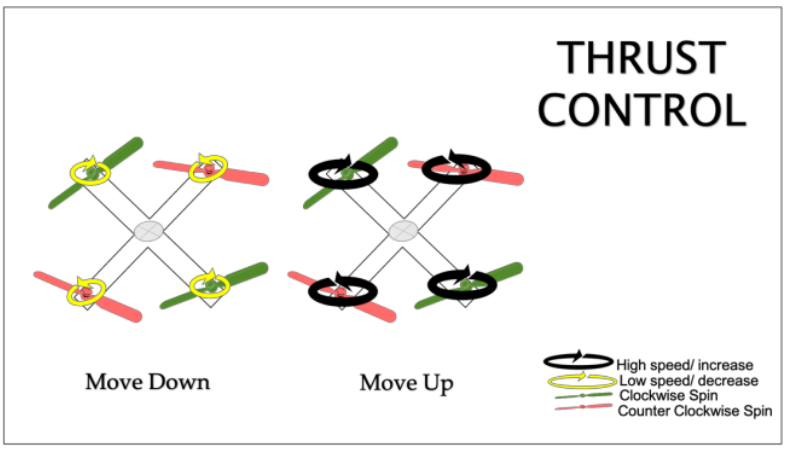


Fig 5.4 thrust control

5.1.2.3(b) move left and right

This movement is a drone simply moving either to its left or towards its right.

5.1.2.3(c) move forward and backwards

This movement is a drone simply moving either to its front or towards its back.

5.1.2.4 Requirement for rotational motion

It is obvious that with the above three translational movements, a drone can move from one place to other. It can perform all the moments with the above three movements but to perform the left/right and forward/backwards motion, the drone will implicitly require rotational motions like roll and pitch.

Actions like forwards, backwards, left and right are performed by rotating the drone at an inclined altitude where the thrust vector is partially in the line and opposite to the gravity and partially towards the force exerted.

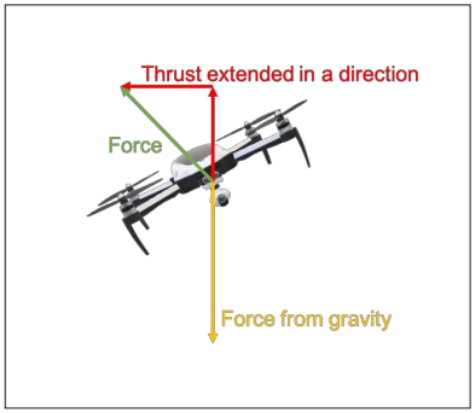


Fig 5.5 Forces behind Rotational motion

Here the force is the direction in which the drone has to follow. (This is achieved by manipulating the speeds of the individual propellors which will be covered next). Now, due to the thrust extending towards the direction of force after cancelling the downward pull of gravity, the drone will be moving in the direction of the force applied. To maintain or fly the drone at a certain altitude we need to increase thrust (as in fig 5.4) and this will cancel out the gravity vector yet again and extends towards the direction of the force applied.

To perform these actions like bending the drone towards the right, left, forwards and backwards we use rotational motions.

5.1.2.5 Rotational Motion of a Drone

5.1.2.3(a) Roll (x-axis)

Roll is to rotate the drone at a certain angle towards the left or towards the right. This helps the drone to further move in that direction as seen in fig 5.5. We can understand the roll motion by looking at fig 5.6.

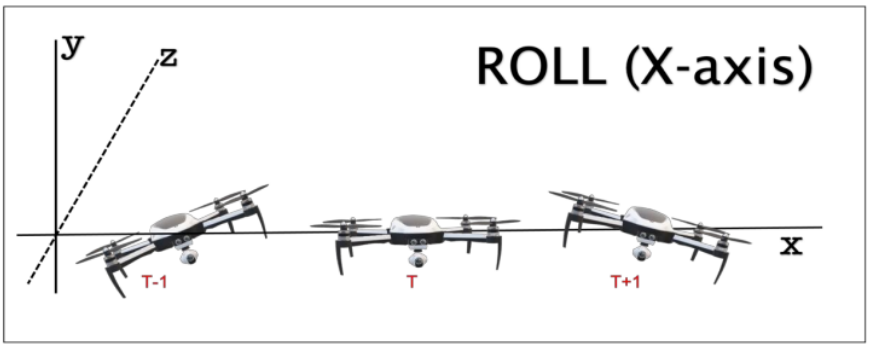


Fig 5.6 Roll motion

This roll is achieved by the speeds of the propeller. In an X-configured quadcopter, increasing one side's propellers' speed while decreasing the propellers' speeds on the other side will make it bend towards the lesser speed side.

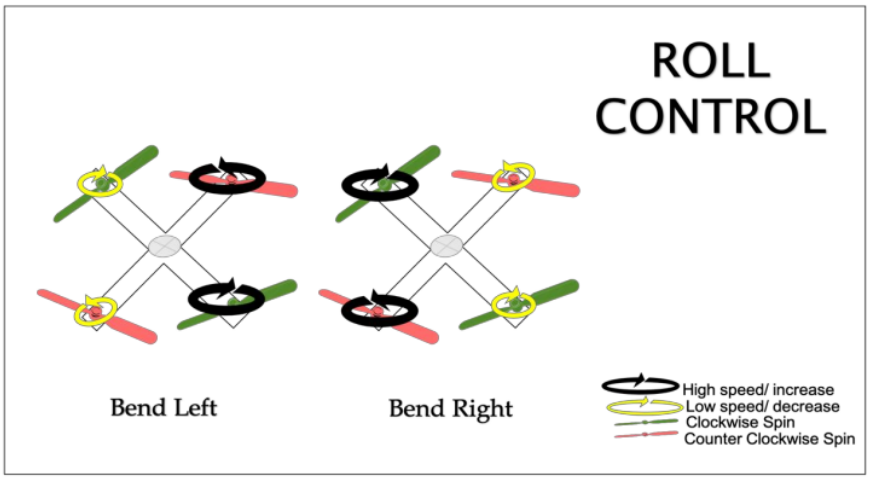


Fig 5.7 Roll Control

5.1.2.3(b) Pitch (y-axis)

Roll is to rotate the drone at a certain angle towards the front or towards the back. This helps the drone to further move in that direction as seen in fig 5.5. We can understand the roll motion by looking at fig 5.8.

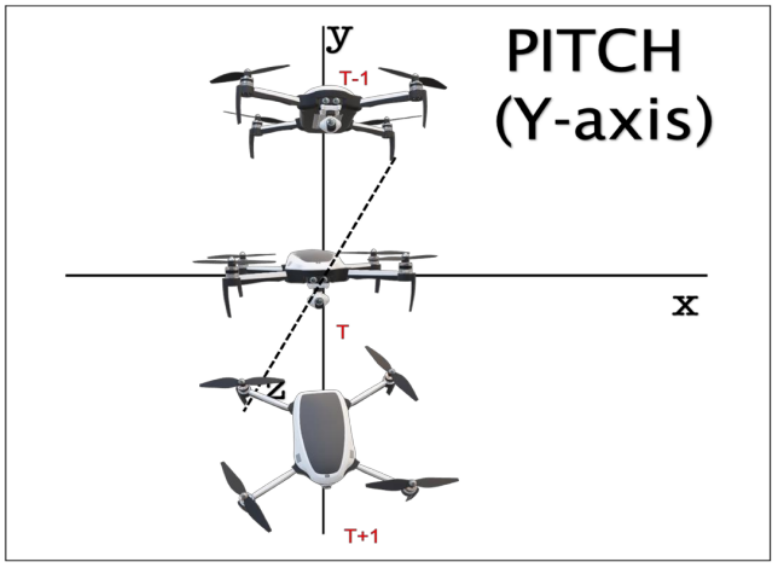


Fig 5.8 Pitch motion

This pitch is achieved by the speeds of the propeller. In an X-configured quadcopter, increasing one side's propellers' speed while decreasing the propellers' speeds on the other side will make it bend towards the lesser speed side.

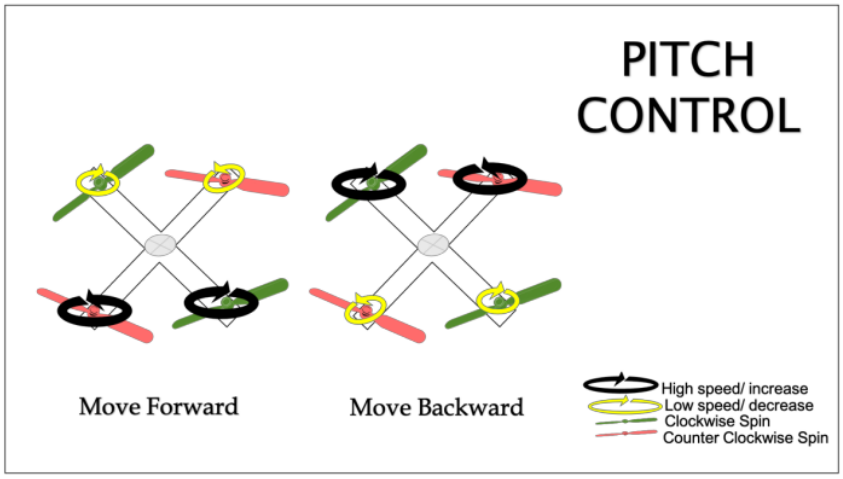


Fig 5.9 Pitch Control

5.1.2.3(c) Yaw (x-axis)

Yaw helps to spin the drone at a certain angle in a certain direction. We can understand the roll motion by looking at fig 5.10.

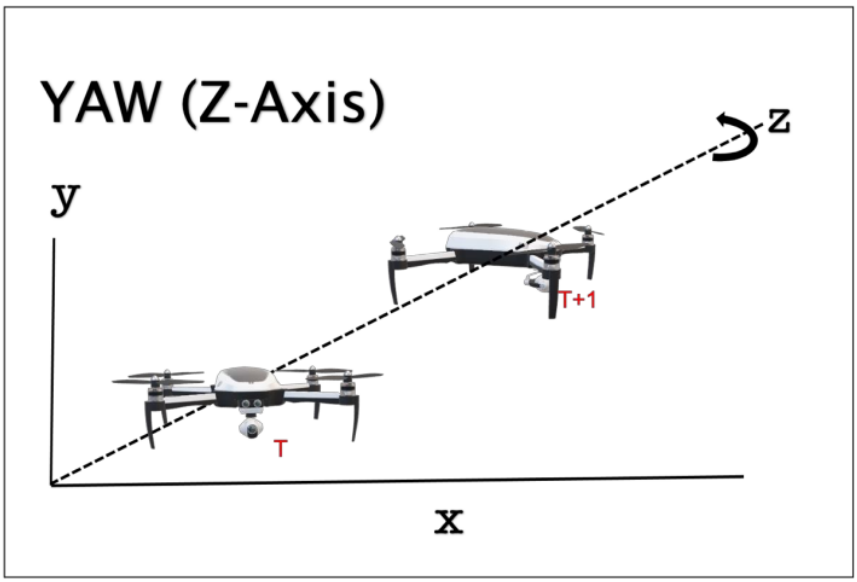


Fig 5.10 Yaw motion

Previously we saw that the X configured or the + configured quadcopters have the same directional spinning propellers facing each other. This is to achieve the yaw. This can be understood by looking at fig 5.11.

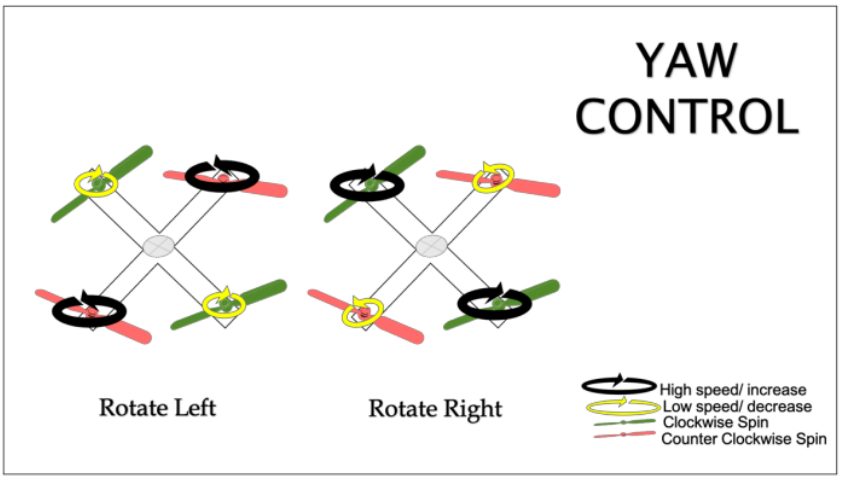


Fig 5.11 Yaw control

5.1.3 Quadcopter-drone Flight Coding

To code the drone, we need to control the three rotational movements or axis of the drone i.e., roll, pitch and yaw along with the thrust. We take these values and give them to the motor values to spin the propellers accordingly. These values can be calculated by the following formulae.

```
motor1 = rc_throttle - roll_adjust - pitch_adjust - yaw_adjust;
motor2 = rc_throttle + roll_adjust + pitch_adjust - yaw_adjust;
motor3 = rc_throttle - roll_adjust + pitch_adjust + yaw_adjust;
motor4 = rc_throttle + roll_adjust - pitch_adjust + yaw_adjust;
```

It is because of these formulae the drone rc_controls which will be seen in chapter 6, signs will be different from the traditional coordinate system.

We know that we need to give the roll, pitch and yaw values but we don't know how to calculate them. For that, we use the PID controller which is a Proportional-Integrative-Derivative Controller.

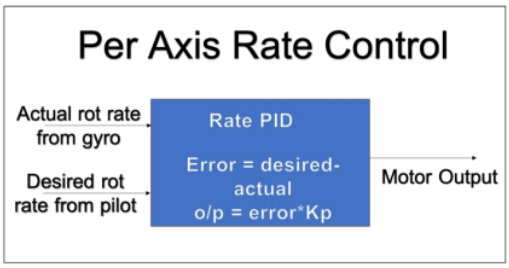


Fig 5.12 PID controller gate

It will take the Rotational rate from the gyroscope and calculate the error with the desired rotational rate and gives the motor output. For our use case, we need to use three PID controllers one for each roll, pitch and yaw. We will discuss more about the PID and their values in chapter 7 of this book.

5.2 The Prototype – Ryze Tello Drone

For this project, we used an educational drone by Ryze robotics. The reason for using this drone is because of the impressive Flight Controller which can be coded to our own controls using python.

5.2.1 Prototype Drone Components

The following is taken from the user manual of the Ryze Tello drone which can be found online at **ryzerobotics.com/tello**[26].

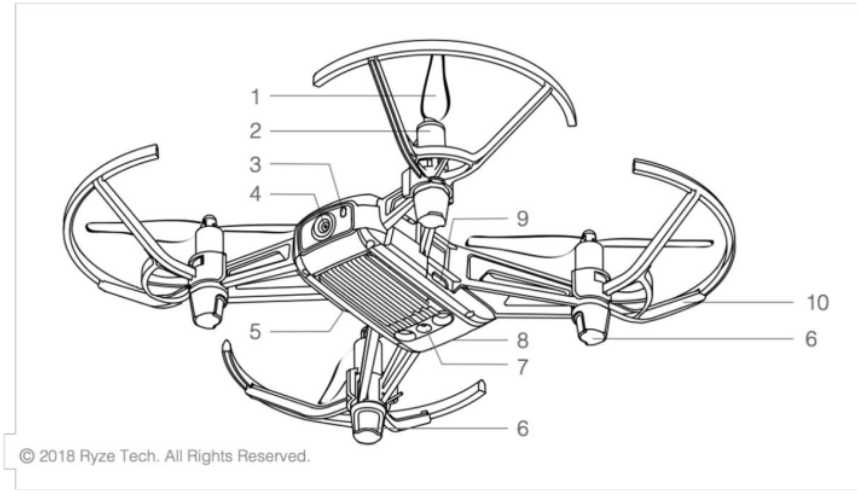


Fig 5.13 Tello Drone Components

"*1. Propellers | 2. Motor | 3. Aircraft Status Indicator | 4. Camera | 5. Power Button | 6. Antennas | 7. Vision Positioning System | 8. Flight Battery | 9. Micro USB Port | 10. Propeller Guards.*"

5.2.2 Prototype Drone Connection

The Architectural IDs and port numbers can be found at Tello SDK. This SDK can be found at
ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf[27].

Architecture

Use Wi-Fi to establish a connection between the Tello and PC, Mac, or mobile device.

Send Command & Receive Response

Tello IP: 192.168.10.1 UDP PORT: 8889 <<- ->> PC/Mac/Mobile

Step 1: Set up a UDP client on the PC, Mac, or mobile device to send and receive messages from the Tello via the same port.

Step 2: Before sending any other commands, send "command" to the Tello via UDP PORT 8889 to initiate SDK mode.

Receive Tello State

Tello IP: 192.168.10.1 ->> PC/Mac/Mobile UDP Server: 0.0.0.0 UDP PORT: 8890

Step 3: Set up a UDP server on the PC, Mac, or mobile device and check the message from IP 0.0.0.0 via UDP PORT 8890. Steps 1 and 2 must be completed before attempting step 3. For more details, refer to the Tello State section.

* Refer to the Tello State section for details.

Receive Tello Video Stream

Tello IP: 192.168.10.1 ->> PC/Mac/Mobile UDP Server: 0.0.0.0 UDP PORT: 11111

Step 4: Set up a UDP server on the PC, Mac, or mobile device and check the message from IP 0.0.0.0 via UDP PORT 11111.

Step 5: Send "streamon" to the Tello via UDP PORT 8889 to start streaming. Steps 1 and 2 must be completed before attempting step 5.

5.14 Tello communication flow

5.2.3 Prototype Drone Dynamics – Movements

The Tello drone can be programmed using python. There is an API called **djitellopy** which will allow us to control the drone through our program. One such source code is pasted below. All the API calls can be found at **djitellopy.readthedocs.io/en/latest/tello**[28]**.**

The following fig 5.15 is from the DJI Tello API. For any sample codes and references, we can look at the GitHub repository at **github.com/dji-sdk/Tello-Python**[28].

```
move(self, direction, x)
```

Tello fly up, down, left, right, forward or back with distance x cm. Users would normally call one of the move_x functions instead.

Parameters:

Name	Type	Description	Default
direction	str	up, down, left, right, forward or back	*required*
x	int	20-500	*required*

Source code in `djitellopy/tello.py`

```
def move(self, direction: str, x: int):
    """Tello fly up, down, left, right, forward or back with distance x cm.
    Users would normally call one of the move_x functions instead.
    Arguments:
        direction: up, down, left, right, forward or back
        x: 20-500
    """
    self.send_control_command("{} {}".format(direction, x))
```

Fig 5.15 Internal code for ‘move’ API

5.3 Software Technologies

5.3.1 Python

Python, created by Guido van Rossum, is an interpreted high-level programming language. Python’s design philosophy is using easier notations and whitespaces. Using thus, will improve the code readability.) Python supports dynamic code processing as well as garbage collection. Python also supports other libraries written in a different language with the language makers one such package in python is CMake. It helps to use other programming language libraries to compute complex mathematical solutions within a module or code.

Python is chosen for this project's coding purposes and machine learning model training purposes. Python is virtually considered the native programming language for Machine learning. This is due to the abundance of libraries that the language provides for machine learning-related tasks. Due to this same reason, the team has decided to go with python for project coding.

5.3.2 OpenCV

OpenCV is one of the libraries in python which is offered to manipulate videos and images. With the name, Open Source Computer Vision, it stands as a library which is producing the vision for a computer just like humans by performing in-numerous tasks. It is with the help of OpenCV we are now able to teach robots to differentiate and recognize many objects. It is with the help of OpenCV we can auto-edit our images and tune them.

There is an open community for OpenCV and their contributions are held in **opencv_contrib.**

5.3.3 Flask

Flask is a Python API developed for Web application development. Flask requires very less base code is required in Flask to create a fully working server-backed-up web application, making it easy to learn. Flask makes web development easier by making the developer not worry about low-level codes like threading and protocols. Flask uses Web Server Gateway Interface (WSGI) toolkit. Jinja2 can be utilized for templating.

This project makes use of Flask to maintain uniformity in every aspect. As the machine learning code will be done in Python and using OpenCV, Flask is utilized for web application development so that all the code in the project may be in Python programming language. And prioritizing Django over Flask is because Flask is lightweight and at the same time is not as powerful as Django. And since the project is in the development phase, just a lite website may suffice for the requirement. So, not too much strain on the purpose, the project utilizes Flask and all its readily built web tools for server and front-end side development.

The general layout of a Flask application is,

1. Module Init

   Module Init file is the primary requirement for the Flask project to be implemented, it is required to enable the application.

2. Module URL

   This module contains and maintains the URLs of every other module.

3. App root Init

   It is not always necessary to define the whole project in the __init__.py file, but the project execution starts at this point.

4. Module Views

   This module maintains the View part of MVC. Particularly this python file makes sure that the front-end files like HTML - CSS and JS files are accessible to other parts of the project for rendering.

5. Module Templates

   This is just a template folder, a modelling format for the project structure.

Flask has in-built support for HTTP methods regarding security, links and database connectivity.

5.4 Functional Requirements

- The requirement of distinguishing the target face from many other potential faces in the frame is the heart of the project.
- Streaming options are required, so whenever the drone is on manual control, the end-user might need to watch the environment the drone is in.
- Drone following the target person through facial recognition is the main problem statement and the top requirement.

5.5 Non-Functional Requirements

- One main functional requirement of the project is to make face recognition feasible to the end-user by limiting the input images required to train the model for facial recognition.
- Minimum facial match confidence of 50% is to be maintained, so that the machine learning model may not take random faces to be the target face.
- The prototype drone is DJI Tello, a mini and educational drone, for which a closed room is advisable for testing, as harsh environmental conditions may hinder its navigation.

5.6 Software Requirements

- Python
- Flask
- Python modules
    - OpenCV
    - face_recognition
    - djitellopy
    - NumPy
- Code Editor
- Tello Mobile application

5.7 Hardware Requirements

- Processor

  The processor requirement for the project is Intel i7 equivalent or above. As the face recognition model requires only a single photo for identification, a whole lot of load is conveyed in the processing phase, and a minimum of Intel i7 processor is required to handle this load and maintain a smooth video stream from the drone.

Also, the flask framework, being implemented in python, is slow in response by a small margin of 900 milliseconds. Having a weak processor will add to this and make the system unresponsive. So, a powerful processor is also ideal for this scenario too.

- Graphics Card

  A machine learning model that deals with images requires an above ordinary graphic card for its basic execution. And in the case of this project, since the ML model only requires a single photo for identification, a huge load of processing is done upon the single image to find as many facial points as the system can. And a well-powered graphic card is required for the smooth execution of the program.

- Drone

  Of course, a physical drone is required to follow its target. Currently, for the prototype phase, the project utilises the DJI Tello drone as it is a ready-made and easy-to-connect drone. But for the production, this model is not advisable, as it is simply an educational drone and the production drone is implied to carry a payload following its target and to have a large battery capacity to run long life.

Chapter - 6

Implementation

6.1 The MVC Architecture

To achieve our problem statement i.e., to make the drone follow us as a pet, we made an end-to-end web application that interfaces between a drone, an OpenCV model and the end-user.

For synergy between a drone, OpenCV algorithm and the web UI we hosted a local server in our local machine. The server takes controls from the user with the help of UI and sends commands to the drone. Our OpenCV implementation lies inside the local machine. The image frames sent by the drone are received through the server and hosted on the website. It is in our local machine the OpenCV code runs after receiving the image frames from the drone.

The architecture of the whole project has been modelled against the "**MVC – Model, View & Controller**" pattern.

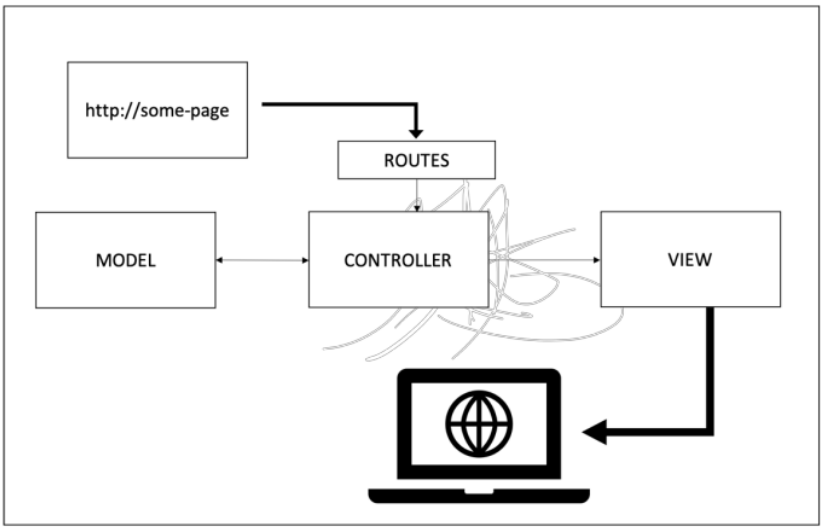


Fig. 6.1 – MVC Model Architecture

Model of the project is the "DroneController" class, which acts as a client wrapper for the drone. This provides access to the drone connection and contains most of the business logic. Both automated and manual controls of the project require DroneManager's access to control the drone as required.

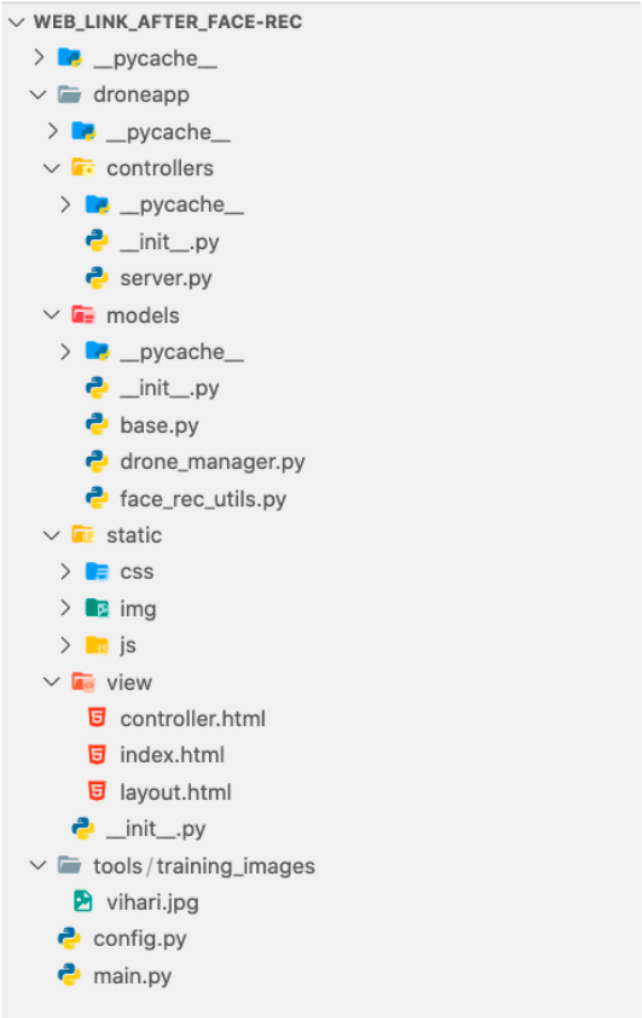


Fig. 6.2 – Directory structure using MVC

The **View** part is simply the frontend rendering of the web application. It renders the drone controls and streaming, in a web application, for manual access and also automation display.

The **Controller** part of the project is the flask (python) server, which acts as the communicational interface between the Front-end part of the web application and the DroneManager.

6.2 Face Recognition

No new face recognition model has been trained, but we utilized open source tools from OpenCV face detection and recognition.

Firstly, the primary encodings of the human face, namely the identification points are found using “face_recognition” of python. This module finds the encoding in a single image and returns them so that these encodings can be used for facial comparison.

Secondly, an infinite loop is computed, with sentinel values for termination, to take in the live streaming video and break them into individual frames. Here, each frame is searched for a potential human face-like structure, and when found a bounding box is used to mark them in the frame. These individual face images are again sent through the encoding procedure as discussed above. Now the encodings of these current faces and encoding of the target face are matched against each other. The match with the least face distance is rendered as the target face. To reduce the matching of random faces, the minimum confidence of matching is 50%. So, it is very much unlike for the system to perceive a random face as the target.

Finally, after the image is found, the system calculates the position of the face to the center of the frame, accordingly the system sends commands to the drone. For example, if the target face is to the right of the frame center, then the drone is commanded to move left so that the face stays in the center. This method is used for tracking the face around the drone. The advantage of using this procedure is that only a single clear image of the target is required for the system to identify him/her in the video frame. But this takes a lot of computational power, and as per the experience gathered during the major project, an equivalent of an intel I7 processor and a powerful GPU are required for smooth execution of the automation program. But it is

still worth the power as no dataset but a single image is required for the model to train its recognition ability.

6.2.1 Training the face to our model

As said earlier, our model requires only a single picture is required to train a particular person. Currently, there is no direct way to add new photos or faces through the website and a link to which it can be trained. We simply added a photo to the local machine where the server is hosted and a path of its location is provided to the OpenCV model. Photos with extension **jpg** and **png** are preferred. The name of the file should be named after the person as the model takes that as a value and generates the same name on the output frame when it detects a similar face.

```
path = '/Users/vihari/Desktop/Major_project/Web_link_after_face-rec/tools/training_images'
```

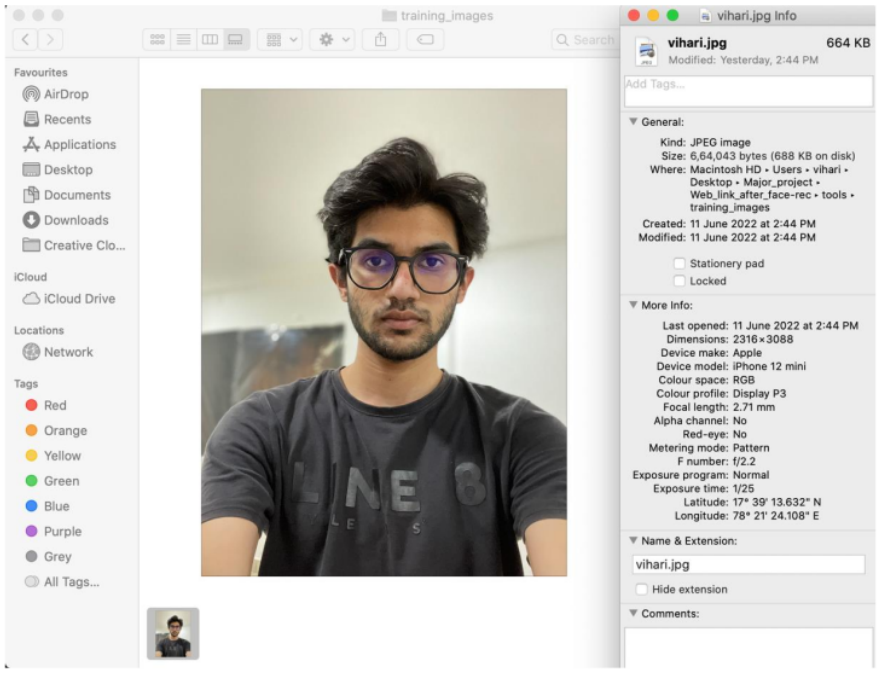


Fig 6.3 Trained face

6.2.2 face_rec_utils.py

```
from operator import contains
import cv2
import numpy as np
import face_recognition
import os

def getImages(path =
'/Users/vihari/Desktop/Major_project/Web_link_after_face-
rec/tools/training_images'):
    myList = os.listdir(path)
    if ".DS_Store" in myList:
        myList.remove(".DS_Store")

    images=[]
    for cl in myList:
        curImg = cv2.imread(f'{path}/{cl}')
        images.append(curImg)
    return images

def getClassNames(path =
'/Users/vihari/Desktop/Major_project/Web_link_after_face-
rec/tools/training_images'):
    myList = os.listdir(path)
    if ".DS_Store" in myList:
        myList.remove(".DS_Store")

    classNames=[]
    for cl in myList:
        classNames.append(os.path.splitext(cl)[0])
    return classNames

def findEncodings(images):
    encodeList = []
    for img in images:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encode = face_recognition.face_encodings(img)[0]
```

```
        encodeList.append(encode)
    return encodeList

def followFace(fbRange, faceCenter, faceArea, width, pid,
pError):
    area = faceArea
    x,y = faceCenter
    fb = 0

    error = x - width//2
    speed = pid[0]*error + pid[1]*(error-pError)
    speed = int(np.clip(speed, -100, 100))

    if area>fbRange[0] and area<fbRange[1]: ## stable
        fb = 0
    if area> fbRange[1]: ## too close
        fb = -20
    elif area < fbRange[0] and area != 0: ## too far
        fb = 20
    if x == 0:
        speed = 0
        error = 0

    return (fb, speed, error)

def markFace(img, x1, x2, y1, y2, name):
    cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0),
2)
    cv2.circle(img, (x1, y1), 5, (255, 0, 0), cv2.FILLED)
    cv2.circle(img, (x2, y2), 5, (0, 0, 255), cv2.FILLED)
    cv2.putText(img,name,(x1+6,y2-
6),cv2.FONT_HERSHEY_TRIPLEX,1,(255,255,255),2)
    cx = (x1+x2) //2
    cy = (y1+y2) //2
    cv2.circle(img, (cx,cy), 5, (255,255,255),cv2.FILLED)
```

6.3 Drone Functionalities

The project has been structured as the drone being the body/slave part of the system and a processor/controller acting as the mind/master component. The information of the environment as observed by the drone is transmitted to the processor, where the processor analyses the data, makes decisions on how the drone shall react, and finally sends them to the drone. The drone acts these instructions out.

For prototype and keeping financial aspects in consideration, the project is utilizing the DJI Tello drone, an educational drone, manufactured by the DJI industries. This drone is utilized as the body of the project as mentioned above. As per the processor role, an individual computer with a high standard processor and GPU is employed. These components consist of the total architecture. The communication between the processor and the drone is done through the utilization of djitellopy API, a python API developed for fulfilling the main purpose of communicating and conveying commands to a DJI Tello drone, programmatically.

In the **models,** the heart of the functionality lies within **drone_manager.py**. As the name suggests it manages all the functionalities of the drone. From taking off to landing, from generating a live stream to the website to manual controls to tracking a particular face and moving accordingly.

DroneManager is a class that when instantiated will be useful to use all the methods which we have written inside. All the basic functionalities of the drone such as takeoff, land, move up, down, left, and right are included. All the methods use **djitellopy API** internally which is pre-documented and can be found on the DJITellopy website.

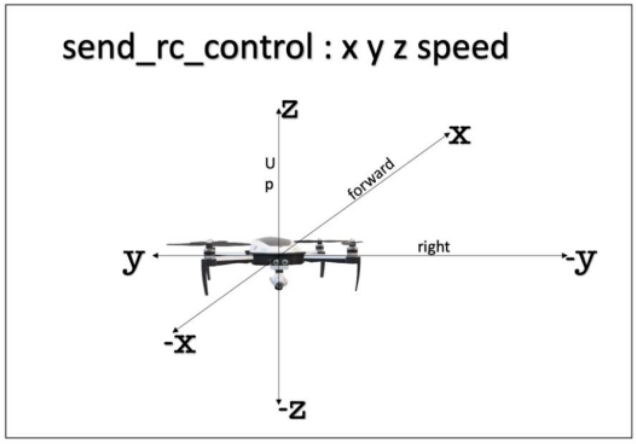
6.3.1 Face-Tracking Using Drone

Our main objective of the project is to have a domestic drone which travels alongside us and only us. The unique features of one's face are identified and used by the drone

to track and follow. As we know the **drone_manager.py** is the heart of the project pumping the required functionalities to the drone as required, the brain is the **face_rec_utils.py** identifying and recognizing the faces.

To achieve this objective, the important methods in **drone_manager.py** are **generate_facerec_video** and **read_cam_frame**. The **read_cam_frame** uses the **get_frame_read** function from djitellopy API which generates the actual frame read by the drone. **generate_facerec_video** uses the **read_cam_frame** to get the frames upon which the face identification and recognition are done. The very same method is also used to generate the video bits into the website which shows the live tracking of the drone's camera output. This is achieved using the **yield** keyword in python. We use yield instead of return keyword when we need to produce a sequence of values and the iteration of the sequence is not needed to be stored in the memory.

When we load our server up (which will be discussed later), the **stream_on** from djitellopy API will start sending frames from the drone which are later captured by **read_cam_frame**. These frames are rendered through **generate_facerec_video** and generated to the website. Until we shut down the server our face recognition model



will be continuously processing in the background trying to detect our trained face.

Fig 6.4 RC Controls graph

We have a variable in our **DroneManager** class **is_faceTrac.** When we want to start our drone automatically fly by tracking our trained face, we can click the start button under face tracking and our **is_faceTrac** which takes boolean values turns **True** and then only it sends the **rc_controls arguments** which our face recognition model will

calculate at every instance to the drone.

It is with the **rc_controls** we can automate our drone to track and face and follow it. We use **send_rc_control** to perform the specific task. This method takes four arguments say y, x, z and speed. It is quite unusual from the coordinate geometry but we have y for left and right velocity; x for forward and backward, z for up and down velocity and speed is the yaw velocity.

Whenever the target is lost from the view area of the drone, manual control override of drone is possible by the end-user, where he can rearrange the drone's view to where the target is visible again.

6.3.2 drone_manager.py

```
import cv2
import numpy as np
import face_recognition
from djitellopy import Tello

from droneapp.models.base import Singleton
from droneapp.models import face_rec_utils


DEFAULT_DISTANCE = 0.30
DEFAULT_SPEED = 10
DEFAULT_DEGREE = 10


class DroneManager(metaclass=Singleton):
    def __init__(self, is_imperial=False,
is_faceTrac=False):
        self.is_imperial = is_imperial

        self.drone = Tello()
        self.drone.connect()
        self.is_faceTrac = is_faceTrac
```

```
def get_battery(self):
    return self.drone.get_battery()

def takeoff(self):
    self.drone.takeoff()

def land(self):
    self.drone.land()

def imperialDistance(self, distance):
    distance = float(distance)
    if self.is_imperial:
        distance = int(round(distance * 30.48))
    else:
        distance = int(round(distance*100))
    return distance

def up(self, distance=DEFAULT_DISTANCE):
    distance = self.imperialDistance(distance)
    self.drone.move_up(distance)

def down(self, distance=DEFAULT_DISTANCE):
    distance = self.imperialDistance(distance)
    self.drone.move_down(distance)

def left(self, distance=DEFAULT_DISTANCE):
    distance = self.imperialDistance(distance)
    self.drone.move_left(distance)

def right(self, distance=DEFAULT_DISTANCE):
    distance = self.imperialDistance(distance)
    self.drone.move_right(distance)

def forward(self, distance=DEFAULT_DISTANCE):
    distance = self.imperialDistance(distance)
    self.drone.move_forward(distance)

def back(self, distance=DEFAULT_DISTANCE):
```

```
        distance = self.imperialDistance(distance)
        self.drone.move_back(distance)

    def set_speed(self, speed):
        self.drone.set_speed(speed)

    def clockwise(self, angle=DEFAULT_DEGREE):
        self.drone.rotate_clockwise(angle)

    def counter_clockwise(self, angle=DEFAULT_DEGREE):
        self.drone.rotate_counter_clockwise(angle)

    def flip_front(self):
        self.drone.flip_forward()

    def flip_back(self):
        self.drone.flip_back()

    def flip_right(self):
        self.drone.flip_right()

    def flip_left(self):
        self.drone.flip_left()

    def stream_on(self):
        self.drone.streamon()

    def stream_off(self):
        self.drone.streamoff()

    def read_cam_frame(self):
        return self.drone.get_frame_read().frame

    def generate_facerec_video(self):
        width, height = 360, 240
        fbRange = [3000, 5000]
        pid = [0.4, 0.4, 0]
        pError = 0
```

```
        faceCenter = [0, 0]

        images = face_rec_utils.getImages()
        classNames = face_rec_utils.getClassNames()
        encodings = face_rec_utils.findEncodings(images)

        # self.drone.send_rc_control(0,0,23,0)

        while True:
            img = self.read_cam_frame()
            img = cv2.resize(img, (width, height))
            imgS = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            facesCurFrame =
face_recognition.face_locations(imgS)
            encodeCurFrame =
face_recognition.face_encodings(
                imgS, facesCurFrame)

            for encodeFace, faceLoc in
zip(encodeCurFrame, facesCurFrame):
                faceDis =
face_recognition.face_distance(encodings, encodeFace)

                if faceDis < 0.5:
                    matchIndex = np.argmin(faceDis)
                    name = classNames[matchIndex].upper()
                    y1, x2, y2, x1 = faceLoc
                    cx = (x1+x2) // 2
                    cy = (y1+y2) // 2
                    faceCenter = [cx, cy]
                    w = abs(x1-x2)
                    h = abs(y1-y2)
                    area = w*h

                    face_rec_utils.markFace(img, x1, x2,
y1, y2, name)
```

```
                    fb, speed, pError =
face_rec_utils.followFace(
                        fbRange, faceCenter, area, width,
pid, pError)
                    if self.is_faceTrac:
                        self.drone.send_rc_control(0, fb,
0, speed)
                        print(fb, speed)

            (flag, encodedImage) = cv2.imencode(".jpg",
img.copy())
            yield(b'--frame\r\n' b'Content-Type:
image/jpeg\r\n\r\n' + bytearray(encodedImage) + b'\r\n')
```

6.4 Web Application

A web application is fabricated for the project purpose, which gives full control over the drone, be it in general or in face recognition mode. The web application is programmed using **flask** and acts as the interface between the end-user and the application.

We started by creating a configuration file for routing. The **config.py** is to tell what data is going to which file and where the logs are created. Creating this will allow us to make sure that the files from specific folders are being called when needed. We give the **WEB_ADDRESS** to 0.0.0.0 which is the local host and port number of 5000 which is the standard for flask.

6.4.1 config.py

```
import os
from flask import Flask
WEB_ADDRESS = '0.0.0.0'
WEB_PORT = 5000
PROJECT_ROOT = os.path.dirname(os.path.abspath(__file__))
TEMPLATES = os.path.join(PROJECT_ROOT, 'droneapp/view')
STATIC_FOLDER = os.path.join(PROJECT_ROOT,
'droneapp/static')
DEBUG = False
LOG_FILE = 'pytell.log'

app = Flask(__name__,
            template_folder=TEMPLATES,
            static_folder= STATIC_FOLDER)
```

As discussed earlier, the application followed MVC (Model, View, Controller) architecture during development. All services in the web application, that need to control the drone part have to employ the DroneManager object which is in **Models.**

The **View part** is a simple front-end part, consisting of just two HTML pages. The first page the **home page** or the **index page** provides the redirection to the **controller page**. The controller page contains which has controller options for the drone. Options like left, right, front, back movement and top, bottom, clockwise and counterclockwise rotations are available on the controller page. These control options enable the end-user to manually control the drone either during or out of the automation phase. The controller page uses the **POST** method to send the commands to the back-end server which will then be executed by the machine.

For both index and controller to have a consistent UI, we first made a layout and then the two pages will follow the same layout. The framework used for styling is **JQuery**. We added those JS and CSS as links to HTML.

6.4.2 layout.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta name="viewport" content="width=device-
width, initial-scale=1">
        <link rel="stylesheet"
href="/static/css/jquery.mobile-1.4.5.min.css" />
        <script src="/static/js/jquery-
1.11.1.min.js"></script>
        <script src="/static/js/jquery.mobile-
1.4.5.min.js"></script>
    </head>
    <body>
        <div data-role="page">
            <div data-role="header">
                <a href="/" data-icon="home" data-dom-
cache="false">Home</a>
                <h1>A following Pet</h1>
            </div>

            <div data-role="content">
                {% block content %} {% endblock %}
            </div>
        </div>
    </body>
</html>
```

6.4.3 index.html

```
{% extends "layout.html" %}

{% block content %}

    <div align="center">
        <h1>I wanna follow follow follow you...</h1>
```

```
        <img width="100%" src="/static/img/drone.png">
    </div>

    <ul data-role="listview">
        <li><a href="/controller/">Controller</a></li>
    </ul>


{% endblock %}
```

6.4.4 controller.html

```
{% extends "layout.html" %}

{% block content %}
<style>
    .controller-box{
        text-align: center;
    }
</style>

<script>
    alert(" Your battery % is " +{{battery}})

    function sendCommand(command, params={}) {
        console.log({action: 'sendCommand', command:
command, params: params});
        params['command'] = command;
        $.post("/api/command/", params).done(function
(json) {
            console.log({action: 'sendCommand', json:
json});
        }, 'json')
    }

    $(document).on('pageinit', function() {
        $('#slider-speed').on("slidestop", function
(event) {
            let params = {
```

```
                speed: $("#slider-speed").val(),
            };
            sendCommand('speed', params);
        });
    });

</script>

<div class="controller-box">
    <h1>Remote Controller</h1>
</div>

<div class="controller-box">
    <div data-role="controlgroup" data-type="horizontal">
        <a href="#" data-role="button"
onclick="sendCommand('takeOff'); return false;">Take
Off</a>
        <a href="#" data-role="button"
onclick="sendCommand('land'); return false;">Land</a>
    </div>
</div>


<div style="display: flex; justify-content: center;">
    <table style="align-self: center;">
        <td>
            <div style="text-align: center;">
                <div class="ui-nodisc-icon ui-alt-icon">
                    <a id="up" href="#"
                       class="ui-btn ui-shadow ui-corner-
all ui-icon-carat-u
                              ui-btn-icon-notext ui-btn-
inline"
                       onclick="sendCommand('up'); return
false;"></a>
                </div>
                <div class="ui-nodisc-icon ui-alt-icon">
                   <a href="#" class="ui-btn ui-shadow
```

```
ui-corner-all ui-icon-forward
                                        ui-btn-icon-notext
ui-btn-inline"
                        onclick="sendCommand('clockwise');
return false;"></a>
                    <a href="#" class="ui-btn ui-shadow
ui-corner-all ui-icon-back
                                        ui-btn-icon-notext
ui-btn-inline"

onclick="sendCommand('counterClockwise'); return
false;"></a>
                </div>
                <div class="ui-nodisc-icon ui-alt-icon">
                    <a href="#" class="ui-btn ui-shadow
ui-corner-all ui-icon-carat-d
                                         ui-btn-icon-notext
ui-btn-inline"
                        onclick="sendCommand('down');
return false;"></a>
                </div>
            </div>
        </td>

        <td>
            <div style="text-align: center;">
                <div class="ui-nodisc-icon ui-alt-icon">
                    <a href="#"
                        class="ui-btn ui-shadow ui-corner-
all ui-icon-arrow-u
                                         ui-btn-icon-notext
ui-btn-inline"
                        onclick="sendCommand('forward');
return false;"></a>
                </div>
                <div class="ui-nodisc-icon ui-alt-icon">
                    <a href="#"
                        class="ui-btn ui-shadow ui-corner-
```

```
all ui-icon-arrow-l
                            ui-btn-icon-notext ui-btn-
inline"
                      onclick="sendCommand('left');
return false;"></a>
                   <a href="#"
                      class="ui-btn ui-shadow ui-corner-
all ui-icon-arrow-r
                            ui-btn-icon-notext ui-btn-
inline"
                      onclick="sendCommand('right');
return false;"></a>
                </div>
                <div class="ui-nodisc-icon ui-alt-icon">
                    <a href="#"
                       class="ui-btn ui-shadow ui-corner-
all ui-icon-arrow-d
                             ui-btn-icon-notext ui-btn-
inline"
                       onclick="sendCommand('back');
return false;"></a>
                </div>
            </div>
        </td>
    </table>
</div>

<div class="controller-box">
    <h3>Speed</h3>
    <input type="range" name="slider-2" id="slider-speed"
data-highlight="true" min="0" max="100" value="10">
</div>

<div class="controller-box">
    <h3>Flip</h3>
    <div data-role="controlgroup" data-type="horizontal">
        <a href="#" data-role="button" data-inline="true"
onclick="sendCommand('flipFront'); return
```

```
false;">Forward</a>
        <a href="#" data-role="button" data-inline="true"
onclick="sendCommand('flipLeft'); return false;">Left</a>
        <a href="#" data-role="button" data-inline="true"
onclick="sendCommand('flipRight'); return
false;">Right</a>
        <a href="#" data-role="button" data-inline="true"
onclick="sendCommand('flipBack'); return false;">Back</a>
    </div>
</div>

<div class="controller-box">
    <h3>Face Tracking</h3>
    <div data-role="controlgroup" data-type="horizontal">
        <a href="#" data-role="button" data-inline="true"
onclick="sendCommand('faceTracStart'); return false;"
>Start</a>
        <a href="#" data-role="button" data-inline="true"
onclick="sendCommand('faceTracStop'); return
false;">Stop</a>
    </div>
</div>

<div align = "center">
    <img src="{{ url_for('video_feed') }}">
</div>

<!-- <div class="controller-box">
    <h3>ADVANCED MODE</h3>
    <div data-role="controlgroup" data-type="horizontal">
        <a href="#" data-role="button" data-inline="true"
onclick="sendCommand('patrol'); return false;">Patrol</a>
        <a href="#" data-role="button" data-inline="true"
onclick="sendCommand('stopPatrol'); return false;">Stop
Patrol</a>
    </div>
</div> -->
{% endblock %}
```

The **Controller** is which contains the server that is hosted locally. This server conveys the frontend requests and responses to the Model end. The setup of the server is done in **server.py.** This server uses logging and config (which we create). Since we use different pages having different routes we need to give a decorator for every function we write providing the route. We can write functions for every page that returns values or performs any actions or shows results on the web page. There should be a **run** function that when executed will load the server up to the local machine and runs it.

6.4.5 server.py

```
from flask import flash
import logging
import mimetypes
import cv2

from flask import request, Response
from flask import jsonify
from matplotlib.pyplot import flag

import config


from droneapp.models.drone_manager import DroneManager
from flask import render_template

logger = logging.getLogger(__name__)
app = config.app

def get_drone():
    return DroneManager()


@app.route('/video_feed/')
def video_feed():
    drone=get_drone()
```

```
    return Response(drone.generate_facerec_video(),
mimetype="multipart/x-mixed-replace; boundary=frame")

@app.route('/')
def index():
    drone = get_drone()
    drone.stream_off()
    return render_template('index.html')


@app.route('/controller/')
def controller():
    drone = get_drone()
    drone_battery = str(drone.get_battery())
    drone.stream_on()
    return render_template('controller.html', battery =
drone_battery)

@app.route('/api/command/', methods = ['POST'])
def command():
    cmd = request.form.get('command')
    logger.info({'action':'command', 'cmd': cmd})
    drone = get_drone()

    if cmd == 'takeOff':
        drone.takeoff()
    if cmd == 'land':
        drone.land()
    if cmd == 'speed':
        speed = request.form.get('speed')
        logger.info({'action': 'command', 'cmd': cmd,
'speed' : speed})
        if speed:
            drone.set_speed(int(speed))

    if cmd == 'up':
        drone.up()
    if cmd == 'down':
```

```
            drone.down()
        if cmd == 'forward':
            drone.forward()
        if cmd == 'back':
            drone.back()
        if cmd == 'clockwise':
            drone.clockwise()
        if cmd == 'counterClockwise':
            drone.counter_clockwise()
        if cmd == 'left':
            drone.left()
        if cmd == 'right':
            drone.right()
        if cmd == 'flipFront':
            drone.flip_front()
        if cmd == 'flipBack':
            drone.flip_back()
        if cmd == 'flipRight':
            drone.flip_right()
        if cmd == 'flipLeft':
            drone.flip_left()

        if cmd == 'faceTracStart':
            drone.is_faceTrac = True


        if cmd == 'faceTracStop':
            drone.is_faceTrac = False

        return jsonify(status = 'success'), 200




def run():
    app.run(host=config.WEB_ADDRESS,
port=config.WEB_PORT, threaded = True)
    #remove threaded
```

Now we have seen that the controller page uses the **POST** method to send the commands to the back-end server and there is a function command in **server.py.** To connect these commands so that the drone would move in the given direction. The **DroneManager** should only be instantiated once. So we created a **Singleton** class in **base.py** so that a single instance is used always. This Singleton is given as a Metaclass argument to the **DroneManager** class in models. We have seen above that in **server.py** we are having a "drone=get_drone()" which actually sends this Singleton to the variable drone and thus that Singleton is used to send the commands to the drone which operates the drone with the help of predefined methods in **DroneManager.**

6.4.6 base.py

```
class Singleton(type):

    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            print('call')
            cls._instances[cls] = super(Singleton,
cls).__call__(*args, **kwargs)
        return cls._instances[cls]
```

Next, we made a **main.py** file which launches a server with the address of localhost:5000 when executed. We can start our application by executing the main file.

6.4.7 main.py

```
import logging
import sys
import droneapp.controllers.server
```

```
logging.basicConfig(level=logging.INFO, stream =
sys.stdout)

if __name__ == '__main__':
    droneapp.controllers.server.run()
```

For the web application to be functional, the computer and drone are to be connected through WiFi network that every Tello drone provides (having a unique name for every Tello) and no additional setup is needed.

Chapter 7
Testing and Results

Testing of this project had been a non-conventional process from the beginning. By now it is clear that we have used OpenCV libraries to achieve our facial recognition. As there was no deep learning model developed for the purpose of face recognition, no training and testing had been conducted from machine learning point of view. To check that facial recognition with the drone, we used many trial and error methods by flying the drone and observing its flight and logs of RC controls. This chapter is not a traditional software testing oriented with metrics, but it gives the various errors we found while working with the hardware for the first time, and the results are what changes we made and what we learned. All the drone testing phases are captured in a video using our mobile phones and the required snaps in the form of frames are attached below.

7.1 Drone's functionality with respect to facial tracking

7.1.1 A basic code to fly the drone

Firstly, after getting the drone we wanted to check whether the drone is functioning properly and how well it is responding to the programs written with **djitellopy** libraries.

In this case, we just connected the drone and made sure that the drone is taking off and hovering at its place for a few seconds and landing. This is done with a sample code taken from the **djitellopy GitHub**. The main weightage in this testing was for Drone Stability testing. Drone Stability in simple terms is verifying whether the drone stays still in the air without any movement, after just being given the "takeoff" command.

7.1.1.1 Result: A basic code to fly the drone

The challenge was flying the drone in an open environment. Being a very less-weighted drone, the DJI Tello drone is easily overpowered by the surrounding wind.

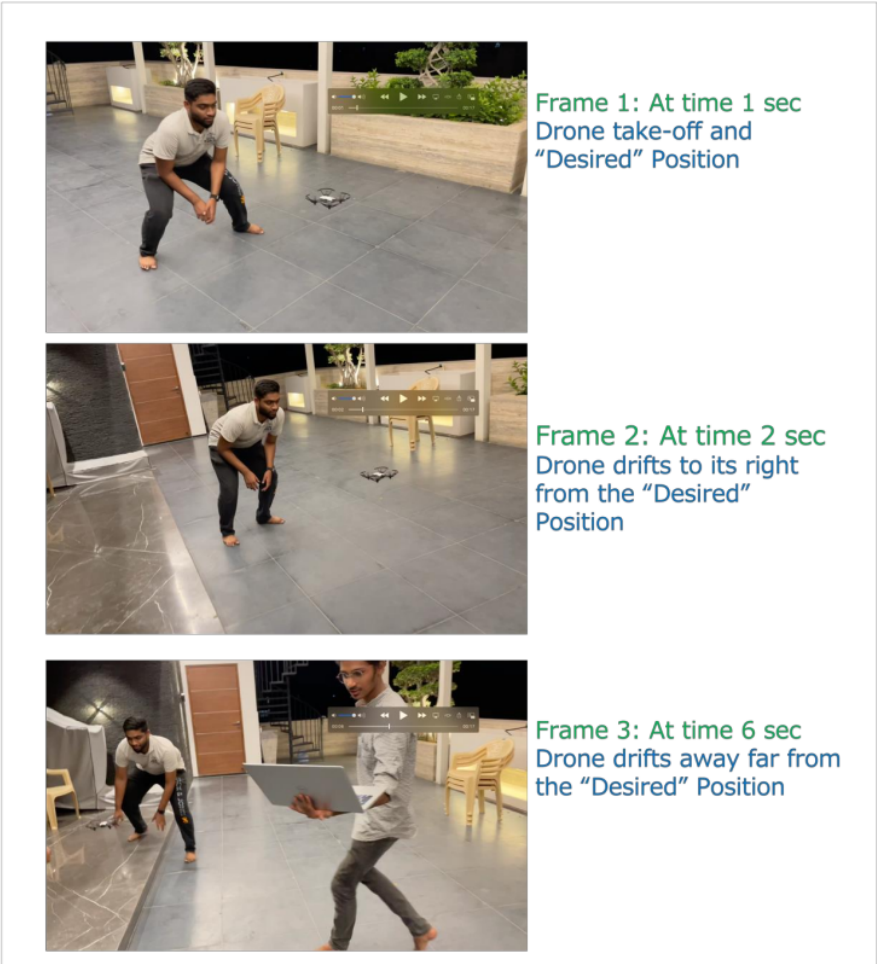


Fig 7.1 shows the result of flying the drone in one environmental condition.

Tests were conducted in various wind conditions, and finally, it was decided that the drone performed at its best in a closed and wind-less environment.

7.1.2 Drone Calibration

There were situations where the drone was not stable even in a closed environment. This was a confusing situation, and after observation and deliberation, the team found out that the propeller shields provided were damaged and deformed and were making the drone lose its balance mid-air. Removing them solved the issue.

7.1.2.1 Result: Drone Calibration

During this course of seeing why the drone is acting such a way, the team came to know the process of drone calibration through Tello mobile application. Sometimes, the drone may not maintain a stable stance in mid-air without any reason, this is perfectly all right, and just a quick process of calibration of the drone as instructed in the Tello mobile application solves the issue.

In the Tello application, we can calibrate the drone by following this path:
Settings->More-> "three dots symbol" -> calibrate. The process is self-explanatory once you look into the app.

7.1.3 Understanding GPU Performance

A skeleton program (without the web interface) is connected to the drone and is executed, making the drone fly and track the target face. The face recognition algorithm is very much efficient in terms of training. It only needs a single photo of the target person to train the model and identify the person in the testing phase. During the execution, the response of the machine learning was very slow. The process of 'drone recording the video and sending the video to the processor for analyzing and the processor giving a response of controls' was slowed down making the whole system to lag behind. Since the drone is being left without any response (duration: 7 seconds) it lands as a security measure (an in-built feature of the Tello drone).

7.1.3.1 Result: Understanding GPU Performance

Since the load is taken off the training phase, the model requires much higher computation power, in terms of both CPU and GPU. During the project's testing phase, this face recognition model could not perform at its best even with an Intel I5 processor and a moderate GPU. So, testing with a low power configuration never gave the chance for the drone to be tested. The result of this testing is that a better CPU and GPU are required for the smooth execution of the system.

7.1.4 OpenCV file issues faced while face tracking

The issue, which was unexpected, was that the OpenCV library is not compatible with the .heic format of the image (generally images taken on an iPhone are of this format).

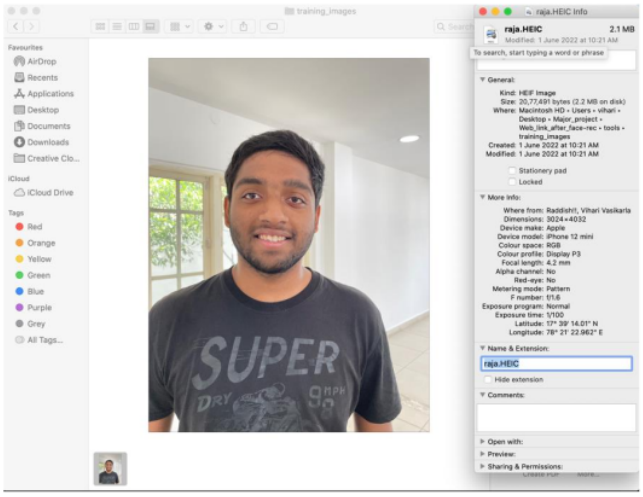


Fig 7.2 Trained image in HEIC format

<u>7.1.4.1 Result: OpenCV files issues faced while face tracking</u>

The issue was rectified by manually converting the image to .jpg format.

<u>7.1.4.1 Result: OpenCV files issues faced while face tracking – “Error 215”</u>

And also, while processing, the OpenCV library regularly gave out “Error:215 Assertion Error”, while reading a folder which contained only images in .jpg format. While debugging, the team came across an invisible file in the image folder, named “.DS_Store”, which when accessed by the OpenCV library gave rise to the error. This may be an OS-related file which cannot be accessed directly. So, the file was manually removed in the program making the error to be resolved.

7.1.5 Setting PID values

The stability of a drone is determined primarily by the values of the PID controller. PID stands for “proportional–integral–derivative”. The main goal of a PID controller is to correct an “error”, which is the difference between the desired value and the actual value. Each component of a PID controller has a different purpose.

The proportional component determines the amount of the error that is being corrected by the controller. The controller will adjust the value of the output of the component to try to achieve the desired flight path. Higher the P gain value sharper the movement of the drone. If the P is set too high, the drone becomes sensitive, and if it's set too low it becomes messy.

The Integral value is responsible for holding the drone’s movement against external forces like wind, gravity, etc. The more powerful the external force, the more the drone struggles to maintain its current orientation. The drone will hold its current orientation more easily if the external force is small. D gain works as a damper and lowers the over-correcting and overshoots generated by P gain.

The PID values should be assigned by calibrating the drone or trial and error method. The calibrated method is the most effective method to use while the trial-and-error method is the easiest method to use.

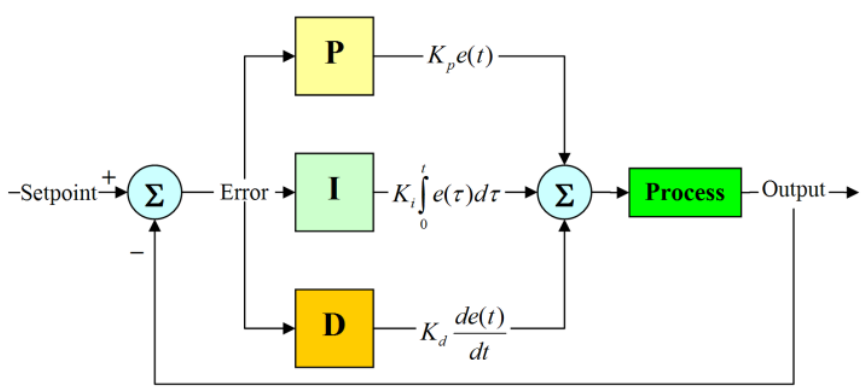


Fig 7.3 PID controller circuit equation

The error and speed values are calculated using the below formulas and then fed to the drone.

```
error = x-W // 2
speed = pid[0]*error + pid[1]*(error-pError)
speed = int(np.clip(speed, -100, 100))
```

x – detected face centre

W – frame width

pError – previous error

7.1.5.1 Result: Setting PID values

These PID values are very integral in drone navigation, and yet there is no readily available formula to calculate these values. There are calculated only through Trial and Error and through a patient calibration process. For every little change in PID values, the project is deployed and tested for the movement speed of the drone, the proximity to the face and other positional attributes, and the PID values with minimum error rate i.e., "[*0.4, 0.4, 0]"* are finalized.

7.1.6 Setting FB values

The face area is calculated by the processor and the ratio of the bounding box area to the whole frame area is calculated, and based on this area and an arbitrary range hard coded into the program, the processor decides whether the drone needs to move forward or backwards.

The arbitrary hard-coded range is named fbRange (forward-backwards range). If at all the face area recorded in the frame is in the range mentioned in fbRange, the drone stays stable. If the face area is less than the lower bound of fbRange, the drone is moved towards the face until the face area falls in the range as seen in fig 7.4.

Fig 7.4 Drone moving too close

If the face area in the frame exceeds the fbRange, the drone moves farther away from the face, until the face area falls in the fbRange as seen in fig 7.5.

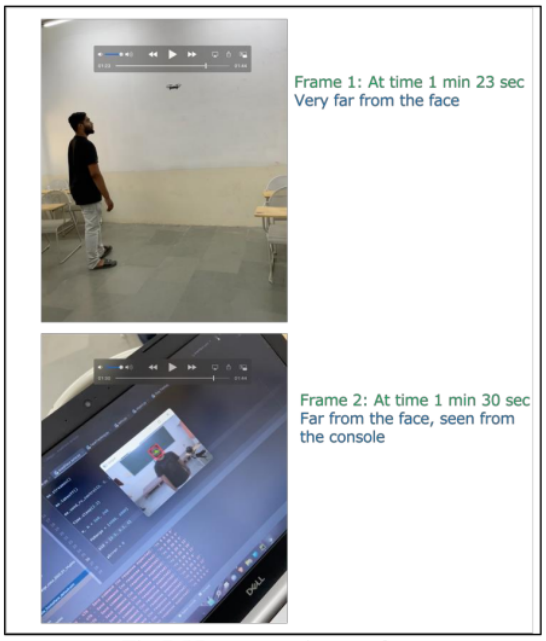


Fig 7.5 Drone moving too far

7.1.6.1 Result: Setting FB values

The FB values cannot be theorised and need to be calculated through a procedure of trial and error. So, through practical testing, by deploying the drone with a face recognition algorithm, random but thoughtful fbRange values were tested and adjusted, where the project finally settled down at the range of *"[3000, 5000]"*.

7.1.7 IMU – lighting conditions

During testing, Naveen (trained image in fig 7.6), one of the team members, was not identified by the drone in weak lighting. There were errors raised mentioning some IMU errors in this situation. And there were instances when drones refused to take off in low lighting conditions.

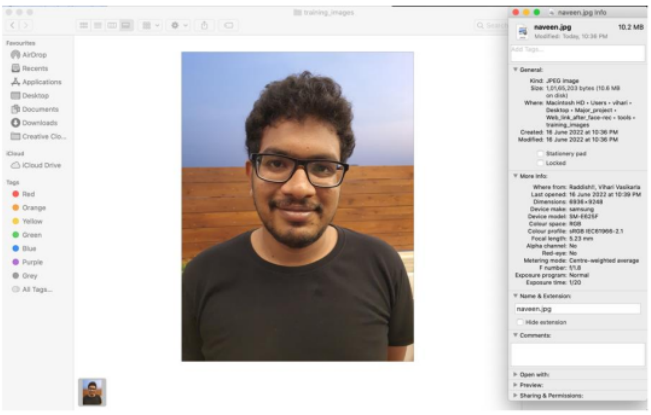


Fig 7.6 Naveen training image

7.1.7.1 Testing conditions

Figure 7.7 is the place used for testing. The room dimensions are 24'9" L–14'W–8'7" H. As seen in the figure, the windows generated the hard light which when the drone facing opposite, was hard to detect faces due to increased black-point (as seen in fig 7.8).

Fig 7.7 Facility used in drone testing

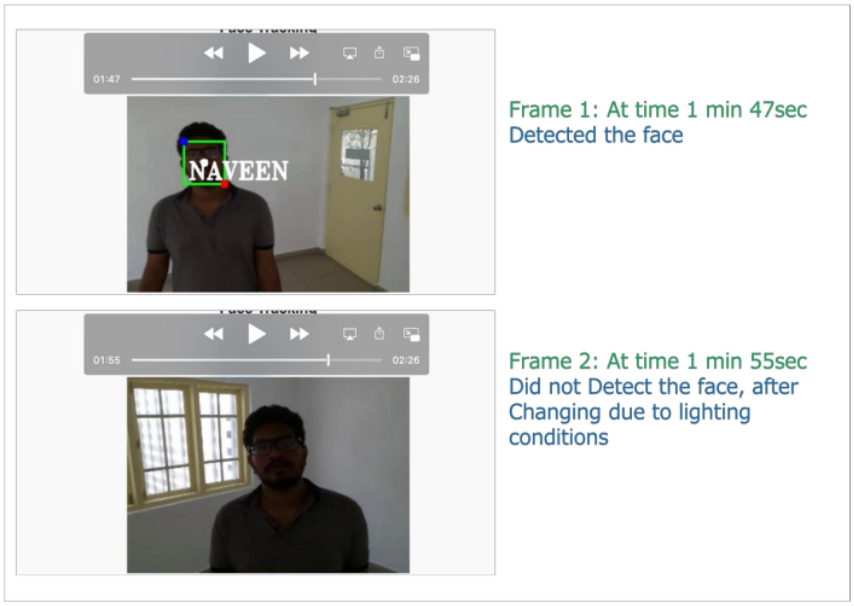


Fig 7.8 Detection failure due to low light

7.1.7.2 Result: IMU lighting conditions

Not sure what the error meant, and there was also no requirement for further investigation because everything in execution was played perfectly in sufficient lighting conditions. The outcome of the testing was that sufficient lighting is needed to be provided for the smooth execution of the project. To get the better results we covered all the windows with black polythene covers (as seen in fig 7.9) so that no sunlight gets inside and only the lights placed in the room will come to play.

Fig 7.9 Facility with windows covered

7.1.8 Face Recognition & tracking: Confidence

In a phase of testing, the model was fed with the image of Raja Yashaswi, one of the team members, for face recognition testing.

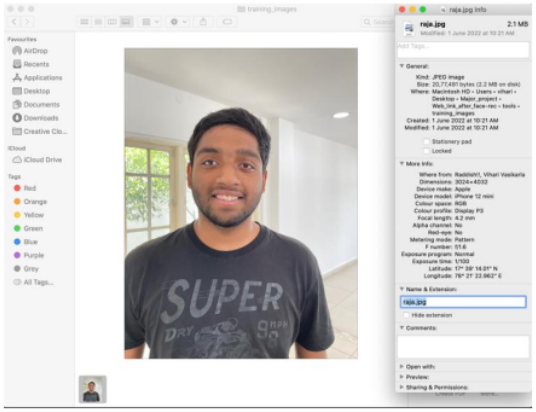


Fig 7.10 Raja training image

Vihari, another team member, mocked the drone by interfering with the drone vision, while the drone was in flight mode and following Raja Yashaswi.

<u>7.1.8.1 Result: Face recognition and tracking: Confidence</u>

The face detection algorithm is customized to only point to a face only if the confidence of the match is above 50%. The drone after recognising Vihari's face chose to follow Raja Yashaswi (as seen in fig 7.11 and fig 7.12). This can be regarded

as one of the biggest successful outcomes in the testing face. This made the team confident that the face recognition algorithm is working rigidly.

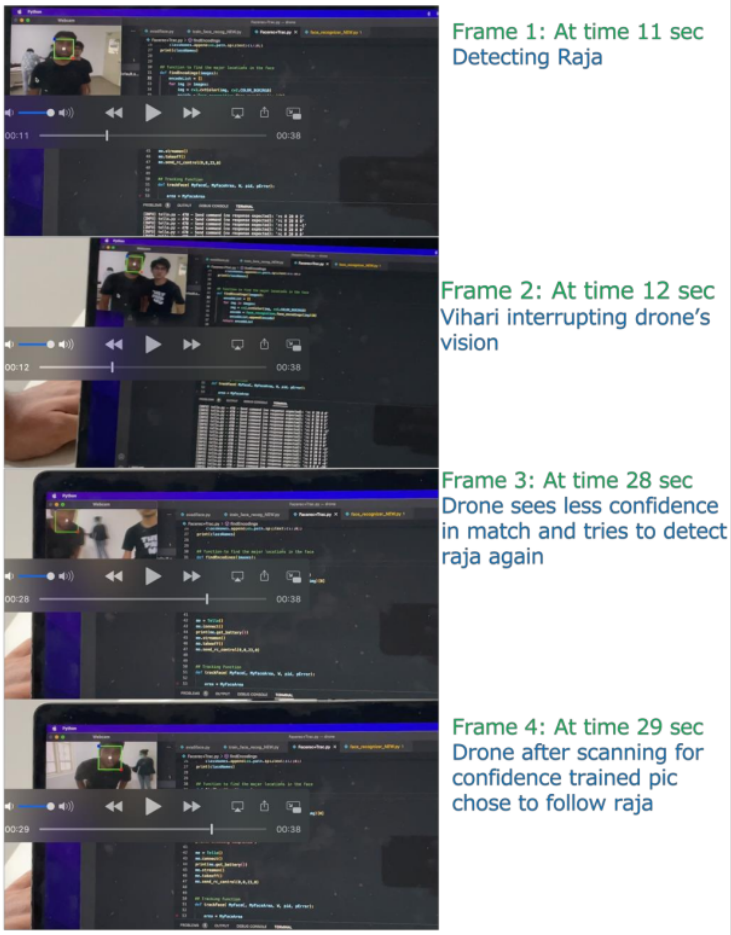


Fig 7.11 System detecting the target face

7.11 is showing the console output while 7.12 is showing the third-person point of view. This is the reason there is a change in timings in both the pictures.

Fig 7.12 TPP of drone detecting target face

7.2 Website

As discussed previously, a website is developed for the purpose that users may directly interact with the drone, without simply depending on the automation process of the face recognition model. This website gives the access to the user, so may directly control the navigation of the drone, and also the user can directly control when the drone's automation (target following) phase may start.

7.2.1 UI

7.2.1.1 Result: Home page or Index page

Figure 7.13 shows the UI of the index page. The picture on the index page is taken from a google search with advanced search properties where it can be used commercially.

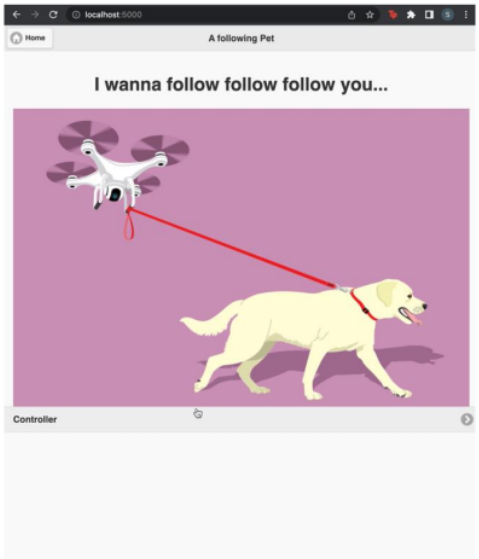


Fig 7.13 index page

7.2.1.2 Result: Battery alert

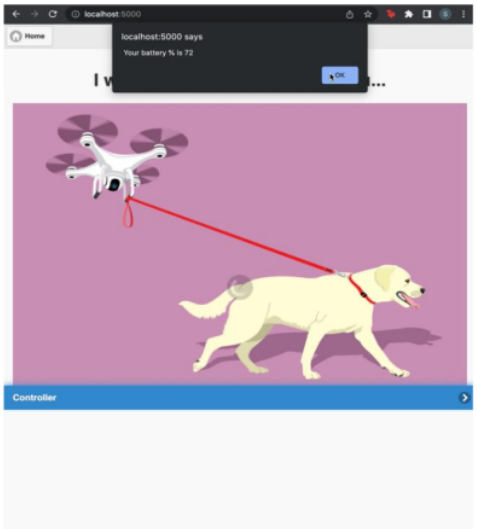


Fig 7.14 battery alert

When the user goes to the controller, they are alerted with the remaining battery in the Tello drone.

7.2.1.3 Result: Controller page

Figure 7.15 shows the UI of the controller page which has all the buttons to control the drone.

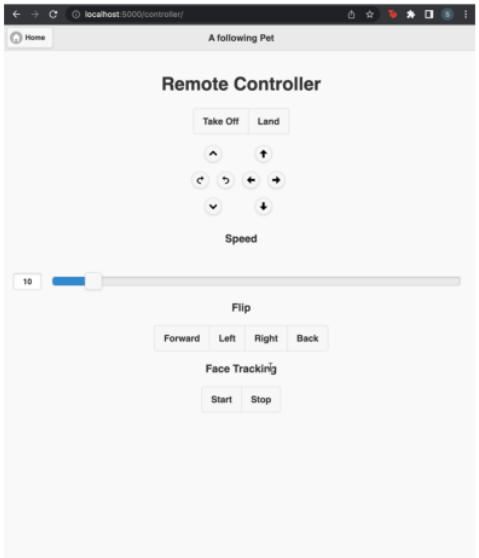


Fig 7.15 Controller page

7.2.2 Controlling drone

The controller page gives the end-user various options in the form of buttons to control the drone. The options were to, move forward, backward, left and right, upward and downward along with the clockwise and anti-clockwise rotation of the drone. Two of the movements are presented in the below pictures.

7.2.2.1 Result: Drone moving down with down arrow

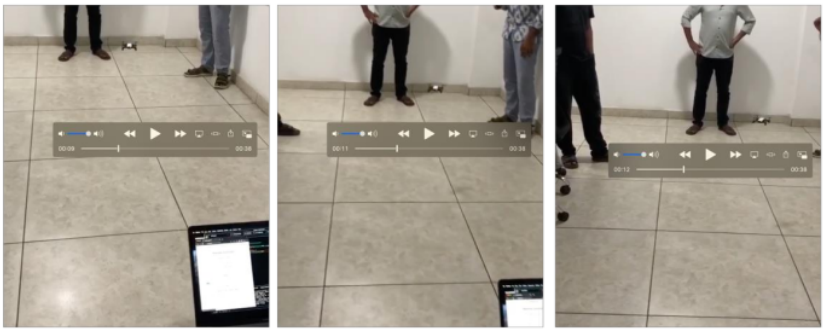
Figure 7.16 shows the drone taking commands from the web UI and going down.

Frame 1: At time 5 sec Frame 2: At time 7 sec Frame 3: At time 9 sec

Fig 7.16 Drone obeying 'Down' command

7.2.2.2 Result: Drone towards its right (our left) with rotate option

Figure 7.17 shows the drone taking commands from the web UI and rotating.



Frame 1: At time 9 sec Frame 2: At time 11 sec Frame 3: At time 12 sec

Fig 7.17 Drone obeying 'Right' command

7.3 Face tracking through the website

Finally, the main testing of the project is done by executing the automation (target following) phase of the drone on Vihari (as seen in chapter 6). This test was around 5 minutes long, and the outcomes were recorded in the form of a video.

7.3.1 Commands log, stream and RC controls

At our server-side, we can see logs of commands given from the website. These logs are actually built-in with djitellopy itself. We can also see the stream of the drone on the website. In chapter 6, we have seen the RC commands. Only after choosing the face tracking option, the RC commands are sent to the drone and this can be verified by looking at the logs at the server-side.

7.3.1.1 Result: Commands log, stream and RC controls

Figure 7.18 shows command logs received at the server-side.

```
[INFO] tello.py - 461 - Response takeoff: 'ok'
INFO:djitellopy:Response takeoff: 'ok'
INFO:werkzeug:127.0.0.1 - - [11/Jun/2022 14:48:22] "POST /api/command/ HTTP/1.1
" 200 -
INFO:droneapp.controllers.server:{'action': 'command', 'cmd': 'up'}
[INFO] tello.py - 437 - Send command: 'up 30'
INFO:djitellopy:Send command: 'up 30'
[INFO] tello.py - 461 - Response up 30: 'ok'
INFO:djitellopy:Response up 30: 'ok'
INFO:werkzeug:127.0.0.1 - - [11/Jun/2022 14:48:29] "POST /api/command/ HTTP/1.1
" 200 -
INFO:droneapp.controllers.server:{'action': 'command', 'cmd': 'up'}
[INFO] tello.py - 437 - Send command: 'up 30'
INFO:djitellopy:Send command: 'up 30'
[INFO] tello.py - 461 - Response up 30: 'ok'
INFO:djitellopy:Response up 30: 'ok'
INFO:werkzeug:127.0.0.1 - - [11/Jun/2022 14:48:34] "POST /api/command/ HTTP/1.1
" 200 -
```

Fig 7.18 shows command logs received at the server-side.

Figure 7.19 shows the streaming facility on the website.

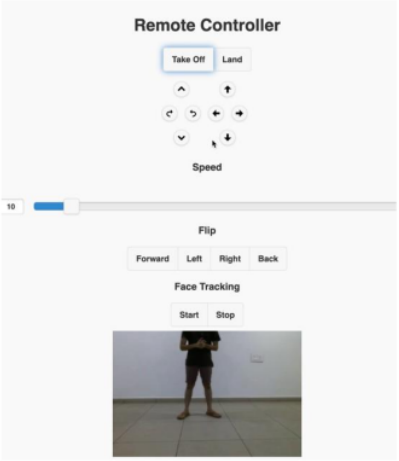


Fig 7.19 shows the streaming facility on the website.

Figure 7.20 shows the RC control's command logs received at the server-side

```
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -2'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -2'
20 -2
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -5'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -5'
20 -5
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -3'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -3'
20 -3
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -3'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -3'
20 -3
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -3'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -3'
20 -3
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -2'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -2'
20 -2
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -4'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -4'
20 -4
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -3'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -3'
20 -3
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -3'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -3'
20 -3
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 0'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 0'
20 0
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -2'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -2'
20 -2
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -2'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -2'
20 -2
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -2'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -2'
20 -2
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -2'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -2'
20 -2
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -2'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -2'
20 -2
[INFO] tello.py - 470 - Send command (no response expected): 'rc 0 20 0 -2'
INFO:djitellopy:Send command (no response expected): 'rc 0 20 0 -2'
20 -2
```

Fig 7.20 shows the RC control's command logs received at the server-side

7.3.2 Final test

The implementation i.e., chapter 6 of this book is the output of this final test. So, the trained face here used is the face mentioned in chapter 6. Figures 7.20 and 7.21 depict this final test.

7.3.2.1 Result: Final test

Figure 7.21 is a screenshot from a screen-recording of the website and server-side put side to side.

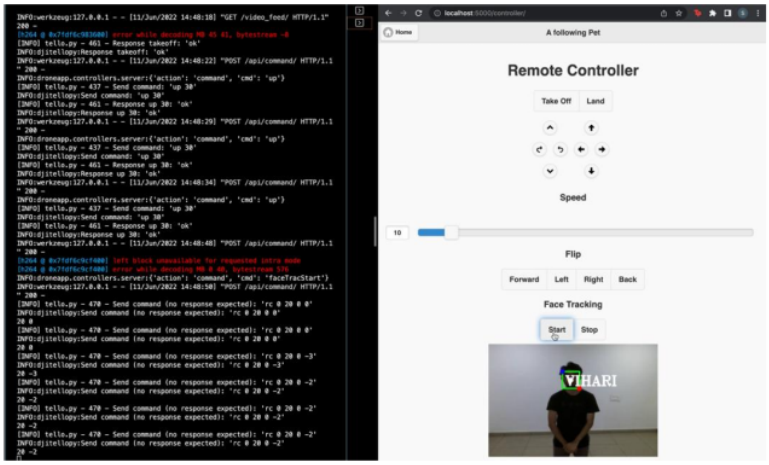


Fig 7.21 screenshot from a screen-recording of the website and server-side

Figure 7.22 is the snap from the third-person point of view which was shot during the drone flight recorded during the time of fig 7.21.

Fig 7.22 The snap from the third person point of view

This flight lasted around 4 minutes and a stitched video where both third person point of view and the screen recorded video can be seen at **clipchamp.com/watch/C1Dp37dBHWU**.

Chapter 8
Conclusion

8.1 Conclusion

The techno-stats say, drones are the future. We wanted to bring the future closer by automating the pilot service using facial recognition and thus making the drone a following pet. Bringing Sci-fi movies' version of drones like EDITH (Marvel & Sony produced Spiderman Far from Home) is a timely process and such cognitive automation (as discussed earlier in introduction) in a drone requires the initial baby-steps like these projects to act as a baseline and provide "data". With a following drone, one can monitor and track their children in an open park (provided that we include GPS tracking features in further additions).

Generic solutions like these, help us in providing many diverse application-oriented solutions or a base level abstraction upon which a problem-specific solution can be built. One of such solutions is the face tracking drone that can be used by filmmakers replacing the slider and dolly shots which consume more set space (the area specifically designed or erected in order to perform movie/drama shooting). **Vlogging** can be taken to the next level with this following drone. Vloggers can now travel places without the concern of holding a big camera or a selfie camera. With the help of the manual controls, vloggers can shoot their surroundings too (a win-win situation).

Similarly, with a following drone, we can further enhance its payload and use as a transport system which follows us, like a following shopping cart. In this way, with our project we tried to add a brick to the wall of advanced cognitive automation and make way for the near future somewhat less nearby.

8.2 Future Scope

8.2.1 Improvement of Web Application

In this project we were able to develop a working web application to manually control, watch the stream and turn on the face tracking. In chapter 6, we have seen that the training face is served to the device directly and the OpenCV runs in the local machine. Though this works for the prototype, we need to have a portable way of letting the users give the trained images through their personal systems or directly via camera.

8.2.2 Migration to Mobile Applications

The current project acts as a drone controller with face recognition and tracking. To access it, a web browser is required. Though the website has responsive UI which will best fit to the mobile browsers, having a mobile application will be useful to future enhancements and easier communication with the user. Having a mobile application will aid us in achieving solutions which are discussed further on in this chapter.

8.2.3 Code and Technical changes

The project is currently in a prototype phase, where the drone slowly follows the person by reading his frontal face. Currently it tracks only a single person even if there are a lot of people in the frame which was the desired output. Soon when we develop a real-time mobile application, multiple users need to train their faces and an option where the drone takes an input on whom to follow and switch between the faces mid-flight.

A Roadmap has also been formulated for the improvement in the algorithm for side face tracking, which enables the drone in following the owner beside him/her, rather than in front of the face as it is trained now. Such implementation may raise the scope towards vast and comfortable applications like:

1. Flying carts

   There has always been the idea of commercial usage of drones, them being used as flying carts, carrying our payloads for us. Having a drone carry everything for us and following us as a pet does feels right out of a sci-fi movie. Amazon is already trying to implement drone delivery of packages; the discussed application is similar in some manner, but a bit more person specific. This application particularly requires the "side-profile" recognition. As it would be annoying for the drone flying in front of the face while shopping, it is better for it to be flying by the side of the person/user.

2. Pet tracking

   The 2020 events had everyone locked behind their doors along with their pets. Isolation was a mandatory measure for humans to survive, but it was not the same for pets. They had to be locked in because, their owners were not allowed to risk taking them for a walk outside. This could be quite stressful and uncomfortable for them.

   So, this project also injected an idea on what if the drone is trained to recognise animals and take them for a walk in a pre-designated path. Since, the drone also has a manual override option, the owner can look over the process and take control of the drone whenever required. Facial recognition of pets is much more complicated than that of humans. So, the idea in this application is that, the drone distinguishes the animal as far as possible based on species, breed and body colour. And since the drone will also be leashed to the pet, that will narrow down the cases where this ideology may stutter.

8.3 RoadMap: Motivation to Future Scope

By coming up with the problem statement, "An automated drone which follows you like a flying pet". we are trying to make drones domesticated, just like smart phones. We are now seeing robots (in which drones are also a part) replacing many human tasks like automobile assembly.

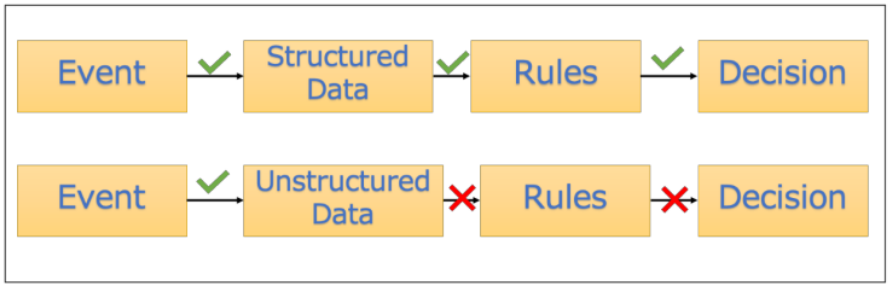


Fig 8.1 RPA

In general these tasks are done using RPA, which is similar to if-else statements and doesn't have any intelligence. They just perform according to the rules. Working with new data or the unstructured data in real-time environment is difficult with RPA as the robots need specific rules (as seen in figure 8.1). To overcome this, Cognitive Automation is introduced where robots/machines are taught to think with cognition.

The following figure (fig 8.2) shows the block diagram of how a cognitive automated robots takes decisions.

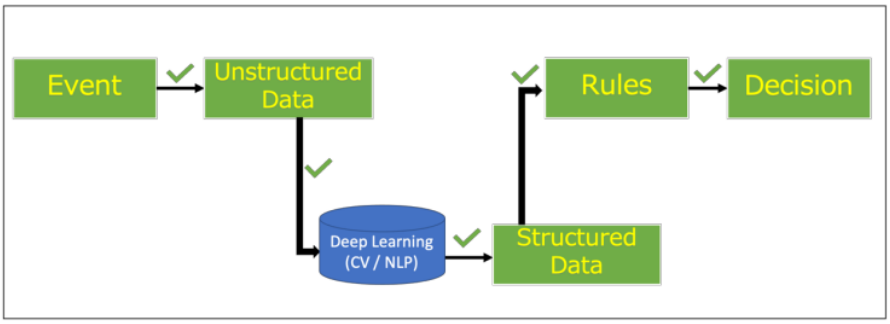


Fig 8.2 CA

Though we are not at a stage of what scientists call it automation, but this project is a small attempt to reach that. The OpenCV for robots is to see and identify the things and here we used the OpenCV to recognise a face. Though achieving a small automated task, this in future of cognitive automation helps largely.

We discussed about the flying carts and amazon drone delivery system. With much improved cognitive automation, the carts can scan the face, recognise, automate the payment, get to our houses all without any human interaction. We also discussed about pet care. The current project can only track the pet and follow it. Giving it a path to take the pet out walking is simple RPA. It is with cognitive automation that we can make sure the drone itself will take care of the pet. To achieve such a feat, people need data, proper hardware, more intelligent systems and high-powered processors. Our slightly automated project can help in developing the intelligent systems by tracking a pet, looking at its behaviour and recording it. Further, this data can be used to build an algorithm which gives cognition to the drone and thus the drone becomes a domestic being which will know what to do or to be precise which will think what to do.

The idea behind this section (8.3) is to make sure that, despite this being the conclusion, hoping there's a cognitive future ahead, which makes life more sustainable and easier.

Major Project Document_Group1

ORIGINALITY REPORT

4%	3%	2%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	github.com Internet Source	<1%
2	www.slideshare.net Internet Source	<1%
3	www.mathaelectronics.com Internet Source	<1%
4	www.irjmets.com Internet Source	<1%
5	discuss.python.org Internet Source	<1%
6	Arun K. Pande. "jQuery 2 Recipes", Springer Science and Business Media LLC, 2014 Publication	<1%
7	aisel.aisnet.org Internet Source	<1%
8	gmap.pucrs.br Internet Source	<1%
9	Rachmat Muwardi, Huangyao Qin, Hongmin Gao, Harun Usman Ghifarsyam, Muhammad	<1%

Hafizd Ibnu Hajar, Mirna Yunita. "Research and Design of Fast Special Human Face Recognition System", 2020 2nd International Conference on Broadband Communications, Wireless Sensors and Powering (BCWSP), 2020
Publication

10	Submitted to The Robert Gordon University Student Paper	<1 %
11	Submitted to Informatics Education Limited Student Paper	<1 %
12	Submitted to University of Adelaide Student Paper	<1 %
13	eprints-phd.biblio.unitn.it Internet Source	<1 %
14	kacavis.zemris.fer.hr Internet Source	<1 %
15	A S Priambodo, F Arifin, A Nasuha, A Winursito. "Face Tracking for Flying Robot Quadcopter based on Haar Cascade Classifier and PID Controller", Journal of Physics: Conference Series, 2021 Publication	<1 %
16	www.coursehero.com Internet Source	<1 %
17	qiita.com Internet Source	<1 %

18	Submitted to Visvesvaraya Technological University Student Paper	<1 %
19	Submitted to Greenwich School of Management Student Paper	<1 %
20	note.com Internet Source	<1 %
21	ela.kpi.ua Internet Source	<1 %
22	Submitted to Jawaharlal Nehru Technological University Kakinada Student Paper	<1 %
23	Submitted to Institute of Research & Postgraduate Studies, Universiti Kuala Lumpur Student Paper	<1 %
24	Submitted to Queen Mary and Westfield College Student Paper	<1 %
25	snte-tics-educacion.blogspot.com Internet Source	<1 %
26	Kushtrim Breslla, Gianmarco Bortolotti, Alexandra Boini, Giulio Perulli et al. "Sensor-fusion and deep neural networks for autonomous UAV navigation within orchards", 2020 IEEE International Workshop on	<1 %

Metrology for Agriculture and Forestry (MetroAgriFor), 2020
Publication

27	Submitted to Manchester Metropolitan University Student Paper	<1 %
28	Meng-Ru Hsieh, Yen-Liang Lin, Winston H. Hsu. "Drone-Based Object Counting by Spatially Regularized Regional Proposal Network", 2017 IEEE International Conference on Computer Vision (ICCV), 2017 Publication	<1 %
29	techpr.info Internet Source	<1 %
30	Turnitin 한국 DB, 국민대학교 Publication	<1 %
31	codecrawl.com Internet Source	<1 %
32	funinc.org Internet Source	<1 %
33	gist.github.com Internet Source	<1 %
34	notes.bweston.uk Internet Source	<1 %
35	"Rule-based Automatic Shot Sequence Generation", Reports of the Technical	<1 %

Conference of the Institute of Image Electronics Engineers of Japan, 2003
Publication

36	Lecture Notes in Computer Science, 2016. Publication	<1 %
37	www.csdn.net Internet Source	<1 %
38	"Computer Vision – ECCV 2018", Springer Science and Business Media LLC, 2018 Publication	<1 %
39	Brad Broulik. "Pro jQuery Mobile", Springer Science and Business Media LLC, 2011 Publication	<1 %
40	Sasa Sambolek, Marina Ivasic-Kos. "Person Detection in Drone Imagery", 2020 5th International Conference on Smart and Sustainable Technologies (SpliTech), 2020 Publication	<1 %
41	Submitted to University of Hertfordshire Student Paper	<1 %
42	docplayer.info Internet Source	<1 %
43	dokumen.pub Internet Source	<1 %
44	pierrelucayotte.ca Internet Source	<1 %

45	shengyu7697.github.io Internet Source	<1 %
46	anditianr.blogspot.com Internet Source	<1 %
47	"Computer Vision – ACCV 2018", Springer Science and Business Media LLC, 2019 Publication	<1 %
48	"Image and Graphics Technologies and Applications", Springer Science and Business Media LLC, 2019 Publication	<1 %
49	Adam Freeman. "Pro jQuery 2.0", Springer Science and Business Media LLC, 2013 Publication	<1 %
50	Eleftherios Lygouras, Nicholas Santavas, Anastasios Taitzoglou, Konstantinos Tarchanidis et al. "Unsupervised Human Detection with an Embedded Vision System on a Fully Autonomous UAV for Search and Rescue Operations", Sensors, 2019 Publication	<1 %
51	Ignacio Martinez-Alpiste, Gelayol Golcarenarenji, Qi Wang, Jose Maria Alcaraz-Calero. "Real-Time Low-Pixel Infrared Human Detection From Unmanned Aerial Vehicles", Proceedings of the 10th ACM Symposium on	<1 %

Design and Analysis of Intelligent Vehicular Networks and Applications, 2020
Publication

52	Jon Westfall, Rocco Augusto, Grant Allen. "Chapter 7 Building an Impressive User Experience with jQuery Mobile", Springer Science and Business Media LLC, 2012 Publication	<1 %
53	stackoverflow.com Internet Source	<1 %
54	version.aalto.fi Internet Source	<1 %
55	Submitted to Univerza v Ljubljani Student Paper	<1 %
56	yl33643.github.io Internet Source	<1 %

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	Off		

Major Project Document_Group1

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51

PAGE 52

PAGE 53

PAGE 54

PAGE 55

PAGE 56

PAGE 57

PAGE 58

PAGE 59

PAGE 60

PAGE 61

PAGE 62

PAGE 63

PAGE 64

PAGE 65

PAGE 66

PAGE 67

PAGE 68

PAGE 69

PAGE 70

PAGE 71

PAGE 72

PAGE 73

PAGE 74

PAGE 75

PAGE 76

PAGE 77

PAGE 78

PAGE 79

PAGE 80

PAGE 81

PAGE 82

PAGE 83

PAGE 84

PAGE 85

PAGE 86

PAGE 87

PAGE 88

PAGE 89

PAGE 90

PAGE 91

PAGE 92

PAGE 93

PAGE 94

PAGE 95

PAGE 96

PAGE 97

PAGE 98

PAGE 99

PAGE 100

PAGE 101

PAGE 102

PAGE 103

PAGE 104

PAGE 105

PAGE 106

PAGE 107

PAGE 108

PAGE 109