

# Kalyani Government Engineering College

(Affiliated to Maulana Abul Kalam Azad University of Technology, West Bengal)

Kalyani - 741235, Nadia, WB



Project Report

on

## IOT BASED UNMANNED AERIAL VEHICLE (UAV) FOR SURVEILLANCE USING WIRELESS SENSOR NETWORK (WSN)

(A dissertation submitted in partial fulfillment of the requirements of  
Master of Technology in Computer Science and Engineering of Maulana  
Abul Kalam Azad University of Technology, West Bengal)

Submitted by

Shubham Khan

University Roll Number: 10211219006

Session 2019-2021

4th Semester

Under the guidance of  
Prof. Swapan Kumar Mondal

Professor,  
Department of Computer Science and Engineering

2019-2021

কল্যাণী - ৭৪১ ২৩৫  
নদীয়া, পশ্চিমবঙ্গ



Kalyani 741 235  
Nadia, West Bengal, India

পত্রাঙ্ক / Ref. No.:

তারিখ / Date :

কল্যাণী গভঃ ইঞ্জিনিয়ারিং কলেজ  
**Kalyani Government Engineering College**  
( Govt. of West Bengal )

**Certificate of Approval**

This is to certify that the project report on “IoT based Unmanned Aerial Vehicle (UAV) for surveillance using Wireless Sensor Network (WSN)” is a record of bonafide work, carried out by Shubham Khan under my guidance and supervision.

In my opinion, the report in its present form is in conformity as specified by Kalyani Government Engineering College and as per regulations of the Maulana Abul Kalam Azad University of Technology. To the best of my knowledge, the results presented here are original in nature and worthy of incorporation in the project report for the M.Tech. program in Computer Science and Engineering.

Signature of Supervisor:

Name and affiliation: Professor, Dept. of CSE

Signature of Head, Dept. of CSE

## **Declaration by the student**

I Shubham Khan, a student of M.Tech, CSE 2<sup>nd</sup> year declare that I have submitted this report in partial fulfillment of the requirements of Master of Technology in Computer Science and Engineering of Maulana Abul Kalam Azad University of Technology, West Bengal.

I also declare that I have checked plagiarism of the report using duplichecker and the plagiarism value is 0.821%. I have appended the plagiarism results at the end of this report.

Name of the student: Shubham Khan

University Roll Number: 10211219006

Full signature: 

Date: 14/07/2021

## ACKNOWLEDGEMENT

I would like to express my profound gratitude and deep regards to my project guide Professor Swapan Kumar Mondal (Professor, Department of Computer Science and Engineering, Kalyani Government Engineering College, West Bengal) for his encouragement and inspiration in this project. This project would never have been successful without his great guidance.

I would also like to specially thanks to Dr. Kousik Dasgupta (Assistant Professor, Department of Computer Science and Engineering, Kalyani Government Engineering College, West Bengal) for his advice and valuable suggestions.



Signature

Shubham Khan.  
Name of the student

Roll No: 10211219006

## **ABSTRACT**

The main goal of our project is to build a video surveillance UAV. Here we can remotely control the UAV by using a remote and send the video directly to the base station with the help of the camera mounted on the UAV. To make this project more advanced, we will monitor human movement using UAV video footage and for that, we will use Object Detection where first we have to build a dataset and train it then we will finally get a model that we can use for Object Detection. Moreover, we can do mission planning here, where mission planning is done first in the UAV and then it can complete its mission automatically. This project can be used for search and rescue, military application, product delivery, and news reporting.

**Keywords:- IoT, UAV, Quadcopter, Surveillance, WSN, Object Detection**

# CONTENTS

<b>ACKNOWLEDGEMENT</b>	<b>iv</b>
<b>ABSTRACT</b>	<b>v</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>1</b>
1.1 Motivation	1
1.2 Background	1
1.3 Summary of present work	2
1.4 Organization of the thesis	3
1.5 Hardware/Software used	3
<b>CHAPTER 2 INTERNET OF THINGS</b>	<b>5</b>
2.1 Overview	5
2.2 Unmanned Aerial Vehicle (UAV)	5
2.2.1 Overview	5
2.2.2 Types of UAVs	5
2.2.3 UAVs components	9
2.3 Wireless Sensor Network (WSN)	17
2.3.1 Overview	17
2.3.2 Type of UAVs action	18
<b>CHAPTER 3 OBJECT DETECTION</b>	<b>19</b>
3.1 Overview	19
3.2 Working principle of object detection	19
<b>CHAPTER 4 PROPOSED METHODOLOGY</b>	<b>21</b>
4.1 Overview	21
4.2 Implementation of UAV	22
4.2.1 Hardware Implementation	23
4.2.2 Flight controller configuration	25
4.2.3 Developed object detection models	34

<b>CHAPTER 5 RESULT AND DISCUSSION</b>	<b>49</b>
<b>5.1 Result</b>	<b>49</b>
<b>5.2 Discussion</b>	<b>53</b>
<b>CHAPTER 6 CONCLUSION AND FUTURE WORK</b>	<b>54</b>
<b>6.1 CONCLUSIONS</b>	<b>54</b>
<b>6.2 FUTURE WORK</b>	<b>54</b>
<b>REFERENCES</b>	<b>55</b>

## LIST OF FIGURES

Figure 2.1: Multi-Roter UAV.	6
Figure 2.2: Fixed Wing UAV.	7
Figure 2.3: Single Router Helicopter.	8
Figure 2.4: Fixed Wing Hybrid VTOLs.	8
Figure 2.5: Quadcopter Frame 4-axis.	9
Figure 2.6: Motor A2212 1000KV.	10
Figure 2.7: ESC (Electronic Speed Control).	11
Figure 2.8: Propeller.	11
Figure 2.9: Ardupilot APM 2.8.	12
Figure 2.10: Lipo Battery 11.1V.	13
Figure 2.11: GPS Module.	13
Figure 2.12: FlySky FS i6.	14
Figure 2.13: FPV Camera.	15
Figure 2.14: FPV 5.8G Transmitter and Receiver.	16
Figure 2.15: 3D Radio Telemetry 433Mhz.	17
Figure 2.16: UAVs action type.	18
Figure 4.1: UAV hardware connection	23
Figure 4.2: UAV motor rotation direction.	24
Figure 4.3: Installing mission planner.	25
Figure 4.4: Installing Firmware.	26
Figure 4.5: Frame Type Configuration.	27
Figure 4.6: Calibrate Acceleration.	27
Figure 4.7: Compass Calibration.	28
Figure 4.8: Radio Control Calibration.	29
Figure 4.9: Electronic Speed Controller Calibration.	30
Figure 4.10: Flight Mode Configuration.	31
Figure 4.11: Failsafe.	32
Figure 4.12: Mission Planning.	33
Figure 4.13: Object detection project structure.	34
Figure 4.14: Pre-training model configuration.	35
Figure 4.15: Installing Required Packages.	36



Figure 4.16: Imports library.	36
Figure 4.17: Downloading Images and Annotations.	37
Figure 4.18: Organizing Images and Annotations.	38
Figure 4.19: Splitting labels into 80% training and 20% testing.	38
Figure 4.20: Converting the annotations from XML files to two CSV files.	39
Figure 4.21: Checks images box position in CSV files.	40
Figure 4.22: Downloading and Preparing Tensorflow model.	41
Figure 4.23: Generating TF record files.	42
Figure 4.24: Downloading the selecting model (Pretrained model).	43
Figure 4.25: Configuring the Training Pipeline.	44
Figure 4.26: Training model output directory setup.	45
Figure 4.27: Training model.	45
Figure 4.28: Training evaluation model.	45
Figure 4.29: Exporting the trained model.	46
Figure 4.30: Run inference graph model test.	47
Figure 5.1: UAV project.	49
Figure 5.2: Object detection test result.	49
Figure 5.3: Object detection evaluation matrix.	50

## LIST OF TABLES

Table 4.1: Estimated project cost	23
Table 5.1: Test image prediction	50
Table 5.2: Confusion matrix	51

# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation

After learning about DRDO's Bharat drone, I became interested to work on such a project, So I chose this project. Then I try to learn more about this and with that my enthusiasm grows a lot more.

It is a small prototype of the Bharat drone with fewer features. Nowadays surveillance and monitoring systems are more important mainly for our military forces to monitor the border area.

### 1.2 Background

The history of unmanned aerial vehicles (UAVs) or drones is not so short development. The development of surveillance technology with the help of these unmanned aerial vehicles has been going on for centuries. The chronological development of surveillance UAV is discussed below.

- In **1858**, French photographer Gaspard Felix Tournachon takes the first aerial photograph in a Dark Room with a Hot-Air Balloon [1].
- In **1896**, Famous dynamite inventor Alfred Nobel launches a camera attach rocket. This is the first time cameras were placed on a UAV [2].
- In **1915**, British forces are used aerial photography to build a map of the German front during the Battle of Neuve Chapelle [3].
- In **1929**, Hungarian scientist Tihanyi invented the television camera-based defense application for remotely-guided aircraft in London for the British Air Ministry [4].
- In **1935**, The First Modern Drone was invented by de Havilland DH82B "Queen Bee" aircraft [5], which was used as a radio-controlled drone developed for aerial target practice.

## 1.2 Background

---

- In **1943**, The Beginnings of First-Person View (FPV) Flight BQ-7 was invented by Boeing and the U.S. Airforce [6], which operated with a crude FPV system. It was an old bomber that effectively designated for stripped the target, Once the target was in view, the autopilot was engaged, and the pilot moves out from the plane.
- In **1973**, Israel Develops Surveillance and Scouting UAVs for the Military [7] which were able to increase their situational awareness significantly.
- In **1991**, During the Gulf War for the first time at least one UAV Flying 24/7 from the conflict start until its conclusion [8].
- In **2006**, UAVs were first allowed into U.S. citizen airspace [9].
- In **2010**, French company Parrot Control a Drone with a Smartphone [10]. And smartphone app was operating the drone safely control by a pilot.
- In **2013**, DJI builds the first phantom drone [11]. This drone started as a modern camera-equipped drone.
- In **2014**, The Use of Drones Rapidly Grows with Industry and Consumers.
- In **2020**, drones would play a key role in helping to deliver mass disinfection and medical supply with maintaining social distance and defending against the spread of the coronavirus.

## 1.3 Summary of present work

Nowadays A lot of work is being done on this project at the moment, one of the main goals being to do a lot of surveillance with less energy. It also makes it suitable for surveillance of more areas. And to monitor in a low sound manner.

Here we can use UAV surveillance to monitor from the sky and do human detection. With a camera mounted on the UAV, we can capture video and monitor that video from a distance. We will also improve our UAVs using machine learning and artificial intelligence technology.

Most of our programming will usually be on video captured on the UAV's camera and for that, we will use Python programming. In addition to UAV Flight Controller with Ardupilot APM 2.8, we will run various programs such as "Return to Home", "Mission Planning" and "Automatic Surveillance". We also use ultrasonic sensors in this project to protect our UAV from being hit by any object.

### **1.4 Organization of the thesis**

The rest of the thesis is described in this section. The second chapter describes the Internet Of Things, Unmanned Aerial Vehicle over Internet Of Things, different types of Unmanned Aerial Vehicle base on their structure, and discuss Unmanned Aerial Vehicle components what is used in this project. Also, discuss Unmanned Aerial Vehicle over Wireless Sensor Network. And some basic actions of Unmanned Aerial Vehicle.

The third chapter describes Object Detection, and there a basic concept of Object Detection models. Choices for Object Detection that are best suited for video surveillance UAV applications. And also how to train a custom Object Detection model.

The fourth chapter is the main part where describes hole work by giving an overview of the proposed methodology and Implementation of the video surveillance Unmanned Aerial Vehicle project.

The fifth chapter describes the project result and discussion. And The sixth chapter Finally, the thesis ends with the conclusion and the scope for future work.

### **1.5 Hardware/Software used**

Here is a list of components used in the project.

Hardware Requirements:

1. Quadcopter Frame 4-axis F450
2. 4 x Motor A2212 1000KV

## 1.5 Hardware/Software used

---

3. 4 x ESC 30A
4. 4 x Propeller
5. Ardupilot APM 2.8
6. Lipo Battery 11.1V
7. GPS Module
8. FlySky FS-i6 6-channel 2.4 GHz Tx and Rx
9. FPV Camera
10. FPV 5.8G AV TX & RX
11. 3DR Radio Telemetry 433Mhz

### Other Hardware:

1. Processor: Intel(R) Core(TM) i5-4258U CPU@ 2.40GHz
2. Memory (RAM): Minimum 8.00 GB

### Software Requirements:

1. Operating System: Windows 10 Pro
2. System Type: 64-bit Operating System
3. Ground Control Station application: Mission Planner 1.3.74
4. Firmware: ArduCopter v3.2.1
5. Programming Language: Python 3.6.7
6. Code Editor: Visual Studio Code

Here I use an online tool, Google Colaboratory notebook for Training object detection model.

## **CHAPTER 2**

### **INTERNET OF THINGS**

#### **2.1 Overview**

The Internet of Things (IoT) is a network of devices that communicate with one another. "Things" basically refers to a collection of sensors that can connect to the Network and swap data between devices.

UAVs is a very important role in the Internet of Things. Thus, the UAV works properly for this type of IoT device, which collects data and can communicate with other devices outside.

Intelligent automation in UAVs can make a highly integrated network, which can enable to completion of a mission by reducing time and energy. IoT drones can also be used as remote monitoring devices.

#### **2.2 Unmanned Aerial Vehicle (UAV)**

##### **2.2.1 Overview**

UAV stands for unmanned aerial vehicles. Before starting this chapter you need to know that Nowadays UAVs is popularly known as Drone. There are various types of UAVs are available in the market here we discuss the basic knowledge about all types of UAVs in the next paragraph. In the last couple of years UAVs, global market growth is increasing day by day. It is going to take on a bigger size in the coming days.

##### **2.2.2 Types of UAVs**

There are several other types of UAVs on the globe, but we'll go through the four common choices dependent on their construction. The following is a list of UAV types.

1. Multi-Rotor UAVs
2. Fixed-Wing UAVs

3. Single Rotor Helicopter
4. Fixed-Wing Hybrid VTOL

### Multi-Rotor UAVs



Figure 2.1: Multi-Roter UAV.

Multi-rotor UAVs are the most popular drones used by professionals and hobbyists, as seen in Figure 2.1. These are frequently used for several applications, such as video surveillance and photography. There are also conventional UAV racing for hobby purposes, or flying UAVs during leisure time. If you want to take pictures with a small camera in the air for a short time, it is very easy to do so with a multi-rotor.

The disadvantage of multi-rotors is their speed, they are large so it is not possible to fly very fast in the air. Moreover, it needs a lot of energy to fight against gravity to keep it in the air, so it can stay in the air for a very short time. These are usually limited to flying for about 20-30 minutes carrying low payloads. However, if the battery capacity is increased or solar is used, it may be possible to blow it longer.

Some of these multi-rotors can be classified into a series. These are 3 rotors are known as Tricopter, 4 rotors are known as Quadcopter, 6 rotors are known as Hexacopter, and 8 rotors are known as Octocopter.



Quadcopters are by far the most popular and commonly used of them. This quad does not require any particular training to operate. You may simply take them to an open area and blow them up there.

### **Fixed Wing UAVs**



Figure 2.2: Fixed Wing UAV.

Fixed-wing UAVs shown in Figure 2.2 are completely different in design from multi-rotor type UAVs. It uses a 'wing' like normal aircraft that helps it float in the sky. This is why they only use energy to move forward, so it should be easier to fly longer in the air. Most fixed-wing UAVs usually can fly for hours. Fixed-wing UAVs are ideal for high and long-distance flights so they are mostly used for aerial mapping or surveillance. However, these aircraft cannot be used for photography where the UAV needs to be kept in the air for some time.

Another disadvantage of fixed-wing UAVs is that they require specialized flight training. It is not easy to keep a fixed-wing UAV fixed in a certain place in the air. It also requires a specific 'runway' or catapult launcher to fly and land safely.

### Single Rotor Helicopter



Figure 2.3: Single Router Helicopter.

Single rotor UAVs shown in Figure 2.3 differ from other UAVs in terms of structure, very similar to helicopters. A single-rotor is mounted to the UAV's tail to control this single-rotor aircraft. Multi-rotor UAVs are typically less efficient than single-rotor UAVs. Single-rotor UAVs, on the other hand, is far more complicated and dangerous. Larger rotor blades are more dangerous. To operate properly, they need special training.

If you want long endurance and the ability to move fast, with heavy payloads, the single-rotor helicopter may be the best for you. Its downsides are complexity, vibration, and also danger to its large spinning blades. These UAVs can move around a single sport effortlessly, and can easily fly and land.

### Fixed Wing Hybrid VTOLs



Figure 2.4: Fixed Wing Hybrid VTOLs.

## 2.2.2 Types of UAVs

---

These hybrid UAVs shown in Figure 2.4 have the benefits of both rotor-based (hover) and fixed-wing-based (with high flight times). It uses vertical lifts to lift UAVs from the ground to the air, which acts like rotor-based UAVs and also works in automatic mode like a fixed-wing UAV to keep the UAV stable in the air. These UAVs are used for remotely or programmable control. There are just a few of these hybrid fixed-wing UAVs on the market.

### 2.2.3 UAVs components

Here we will discuss all the components used in our project. This chapter is primarily intended to describe general concepts of hardware components. Here you can find out what the material is, what it is used for, and how it works.

#### Quadcopter Frame 4-axis 450

It is the structure of the UAV, which holds all the components of a UAV together. Figure 2.5 is shown a Quadcopter UAV Frame. This frame is an important part of the quadcopter or UAV because it prevents motors and all other electronics and their vibrations. If its weight is light it will help reduce the weight of the UAV and help the UAV to fly in the air longer.



Figure 2.5: Quadcopter Frame 4-axis.

But it needs to be strong with light. Usually, it is made of carbon fiber material, but you can make it with any light and hard material and make it more advanced than the structure purchased from the market. Here we have used the 4-axis F450 Quadcopter Frame.

### **Motor A2212 1000KV**

BLDC stands for Brussels DC Motor, a high-speed brushless motor used exclusively in UAVs or planes. In our project, we have used a 1000 kV motor. For example, a 1000 kV motor means that a motor will spin at 1000 rpm per volt. Figure 2.6 is shown a Brussels DC Motor.



Figure 2.6: Motor A2212 1000KV.

If a 3-cell lipo battery has a voltage of 11.1V, we can calculate from this that a 1000 kV motor will spin  $1000 \times 11.1$  or 11,100 rpm. These motors help to rotate the UAV's propellers so that the UAV can fly in the air. The BLDC motor has three input wires that are connected to the electronic speed control.

### **ESC 30A**

An electronic speed control (ESC) is an electronic circuit that helps control the speed of an electric BLDC motor. It is commonly used in UAVs but is also used to control the speed of other small, fast, and advanced motors.



Figure 2.7: ESC (Electronic Speed Control).

Figure 2.7 is shown an electronic speed control. It has three wires at one end which are connected to the BLDC motor. These are called electronic speed control outputs. By changing the two wires at the two ends of these three wires and connecting them to the BLDC motor, the motor can rotate in its opposite direction. And at the other end, two wires help the power supply and three more thin wires are attached to it which are connected to the output pin of the flight controller of the UAV. These three use for controlling ESC and these three wires are known as VCC, Signal, and Ground.

### Propeller



Figure 2.8: Propeller.

A propeller is a device that converts rotary motion into linear thrust. Figure 2.8 shows a Propeller. Because the UAV propeller is designed like an airflow, when it rotates with the help of a motor, it generates lift, allowing the differential pressure between the top and bottom of the propeller to be observed and helping the UAV in flying. There are usually two types of propellers, clockwise propellers, and counter-clockwise propellers. These are made of very hard and light material.

## Arduilot APM 2.8

Ardupilot Mega (APM 2.8) is a flight controller, it is the main component of UAV. It is possible to control all the components of the UAVs. All the components used in the UAV are attached to it. It has to be installed in the firmware and then it has to be configured. The firmware has to be chosen according to the device on which it will work. If it is configured with some issue then there can be many problems so it needs to be done carefully.

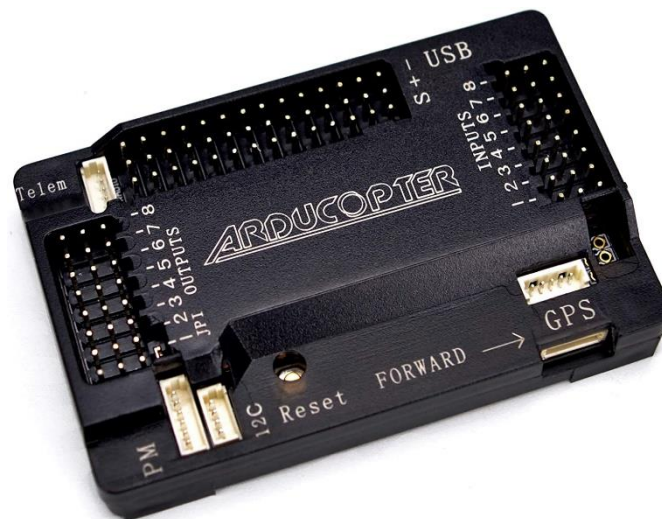


Figure 2.9: Ardupilot APM 2.8.

There is a variety of software available for this configuration, but Mission Planner is the most common. An Ardupilot APM 2.8 flying controller is shown in Figure 2.9. The Ardupilot Mega is a high-performance autopilot IMU built on the Arduino Mega platform. Fixed-wing aircraft, multi-rotor UAVs, helicopters, and a variety of so many other remotely controlled devices may all be controlled with this Ardupilot.

### Lipo Battery 11.1V



Figure 2.10: Lipo Battery 11.1V.

Li-Po or Lithium Polymer batteries are very strong. Figure 2.10 shows a Lipo Battery. Lithium polymer batteries are the most common batteries used for UAVs, as they are capable of delivering higher energy than their size and weight. Because each cell has a higher voltage, the UAV needs less energy to fly than other rechargeable batteries. There are two more common types of chemistry used for UAVs, LiHv (Lithium High Voltage) and Graphene. Here we used an 11.1V 2200mAh 3S Lipo battery for the power supply to the UAV.

### GPS Module



Figure 2.11: GPS Module.

### 2.2.3 UAVs components

The GPS is used in UAVs for navigation. The GPS module is shown in Figure 2.11. The position of the UAV is determined using GPS. GPS may be used to determine the relative location and speed of the UAVs. GPS modules feature small CPUs and antennas that can receive data from satellite-derived RF frequencies directly. They get timestamps as soon as they get data from the internet. They receive timestamps as they receive data from satellites. It is always attached to the flight controller board. Some GPS has a compass inbuilt.

GPS navigation helps a UAV to predetermine its destination or the location from which the UAV is located. It also gives instructions on where the UAV should fly. It is very good if it is under open sky when configuring. This is also why it is used to keep the UAV stable. For all these reasons it is very much used to do in the mission planning program. Its use in some modes of flying the UAV is vital.

#### **FlySky FS-i6 6-channel 2.4 GHz Tx and Rx**



Figure 2.12: FlySky FS i6.

A transmitter, often known as a radio transmitter, is an electrical device that produces radio waves and transmits them to a receiver via antennas. A radio transmitter and receiver are shown in Figure 2.12. The FlySky FS-i6 is a cost-effective entry-level 6-channel 2.4 GHz transmitter and receiver which is used for Stable Automatic Frequency Hopping Digital System (AFHDS) technology. Here it is used to control the UAV.



The receiver is linked to the UAV's flight controller, and the transmitter sends the signal to the receiver manually, allowing the base station to operate the UAV.

#### **FPV Camera**



Figure 2.13: FPV Camera.

Pilots used First-Person View (FPV), also known as Remote-Person View (RPV), to operate a radio-controlled vehicle even while capturing photos and video of the area. An FPV camera is shown in Figure 2.13. It's designed to be compatible with a radio-controlled plane or another unmanned aerial vehicle (UAV).

We are deploying it for surveillance. It is connected to the UAV's FPV transmitter. Because FPVs can broadcast live video via transmitter and receiver communication technologies, UAVs are becoming extremely popular.

### FPV 5.8G AV TX & RX



Figure 2.14: FPV 5.8G Transmitter and Receiver.

An FPV transmitter-receiver is a type of device used in remote-control (RC) UAVs. Figure 2.14 is shown an FPV Transmitter and Receiver. It is used to send video or images captured by a small video camera to the ground station.

FPV transmitter is attached with a UAV that is connected to the camera and the receiver device is connected to the ground station system. The transmitter can be used with any receiver but the transmitter just needs to bind with the receiver first.

An FPV transmitter can typically transmit video using the following frequencies: 27MHz, 72MHz, 433MHz, 900MHz, 1.3GHz, and 2.4Ghz, 433Mhz, 900Mhz, and 1.3GHz

### 3DR Radio Telemetry 433Mhz

The 3DR radio telemetry system for UAVs is designed as an open-source Xbee replacement radio set, capable of providing long-range and higher performance on XB radios.



Figure 2.15: 3D Radio Telemetry 433Mhz.

Scientists have been using radio telemetry since the 1960s to track and identify movements of any object. Figure 2.15 is shown Radio Telemetry. Radio telemetry uses radio signals, which are composed of electromagnetic waves to determine position. And It is associated with the UAV's flight controller.

A telemetry radio uses the MAVLink protocol to help the UAV communicate with the ground station while in the air. This helps the UAV to supply all the data to the ground station in real-time. It is even possible to program or control the UAV without any wire so that it is important for mission planning.

Keep in mind that if you use SIK radio, you will need your country's approved frequency version - 915 MHz (US) and 433 MHz (Europe).

## 2.3 Wireless Sensor Network (WSN)

### 2.3.1 Overview

WSN stands for Wireless Sensor Network. In this chapter, we understand the basic concept of Wireless Sensor Network and its uses over UAV. Wireless Sensor Network is a bunch of sensors that are communicated with each other and send data to the master node. UAVs have used many types of sensors that help fly by collecting data. Here a list of basic sensors describes below.

### 2.3.2 Type of UAVs action

A radio transmitter can transmit commands through channels. Each channel has a separate activity program. Figure 2.16 shows the basic function that any UAV device can perform. They are described below.

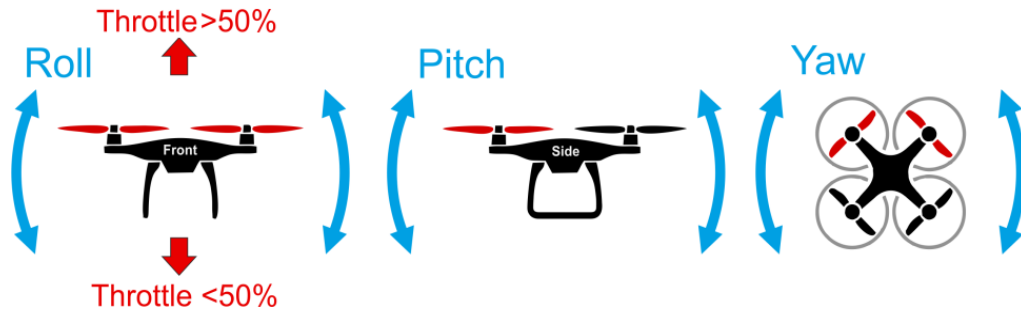


Figure 2.16: UAVs action type. Source: Adapted from [31]

- Roll: This helps move the UAV in the air to the left or right.
- Pitch: This helps the UAV to move forward or backward.
- Yaw: This helps the UAV to rotate clockwise or counterclockwise.
- Throttle: This helps the UAV to move faster or slower.

Many UAVs can form a network together. Where a UAV can be associated with one or more UAVs and transmit data with each other. And in this case, the UAV near the ground station helps to transmit all the data to the ground station. Each UAV in this network is known as a node. Such networks are usually used to cover a wide area together.

This type of network is known as a swarm network. Moreover, when it is decided to send a UAV over a long distance, a few more UAVs are placed between the ground station and the long-distance UAV so that they can form a network among themselves. And control of the long-distance UAV is possible from the ground station.

## CHAPTER 3

### OBJECT DETECTION

#### 3.1 Overview

Object detection is a part of computer vision, where an automated method is used to identify interesting objects in an image. Like other computer vision tasks, deep learning is a sophisticated method for object detection where the computer learns itself and builds a model. Data science and ML programming are useful in object detection projects. Popular algorithms used for most object detection include Convolutional Neural Networks (CNN), Region-Based Convolutional Neural Networks (R-CNN), Fast R-CNN, and You See Only Once (YOLO), etc.

#### 3.2 Working principle of object detection

Here we will discuss the object detection model in detail. and learn how to do it. Here we will describe step by step how to train the object detection model below.

- **Object Detection API**

There are several object detection APIs available for object detection. Popular models are TensorFlow, Keras, and Pytorch. Here we have selected TensorFlow for our project. TensorFlow is easily accessible Online, it provides a system in the cloud for model training. This platform is powered by Google.

- **Choosing a pre-training model**

Tensorflow also has many types of object detection pre-trained models, e.g. CenterNet, EfficientDet, SSD MobileNet, SSD ResNet, Faster R-CNN ResNet, Faster R-CNN Inception ResNet, and Mask R-CNN Inception ResNet. Here we choose the SSD MobileNet pre-trained model for this project.

- **Installing Required Packages**

Here we install some important packages and libraries that will be required for this project.

- **Preprocessing Images and Labels**

Here we collect a set of images and then label those images by making annotations files (XML files). Then splitting those into two directories named train and test. After that converting the XML files to the CSV file for each folder. and also create a pbtxt file for listing all class names.

- **Generating TF record**

Generating two TFRecords files for the train and test CSVs. Tensorflow accepts the data from TFrecords which is a binary file that run fast with low memory usage.

- **Downloading and Preparing Tensorflow model**

Here we are downloading and preparing the Tensorflow pre-trained model. In our case, we have chosen SSD MobileNet. Now configuring the training model pipeline by changing file path and value.

- **Training and Testing**

Finally training the model. It may take some time-dependent to system's configuration. Here model train itself. After completing the training the trained model needs to be exported as well as the data needs to be tested to complete this project with the models.

## CHAPTER 4

### PROPOSED METHODOLOGY

#### 4.1 Overview

I have planned a UAV surveillance system, which will have four motors and each motor attached to a propeller for fly in the sky. All motors will be connected to the ESC, which will help to control the speed of the motor so that the UAV can be moved in any direction.

Here all the ESCs will be connected by a flight controller so that we can control the ESCs. Moreover, GPS will be added to the flight controller to know the location of the UAV and if the UAV goes out of the range from the remote control, it will be automatically returned to its place before the end of the UAV power supply.

A receiver will be connected to the flight controller and it will be possible for the receiver to control the flight controller. This receiver will receive all signals from the transmitter and the transmitter will be operated by man.

The UAV will also have a camera so that we can do live video surveillance. This camera will be connected to an FPV transmitter so that the video captured on the camera can be sent to the FPV receiver and that video can be monitor through a mobile device.

Below is a step-by-step short description of the project work according to our planning.

**Phase 1:** First, we take the frame of the UAV and attach the motors to it, then we attach the propeller to each motor. Now connect the ESC to the motor and finally connect all the ESCs with the UAV flight controller.

**Phase 2:** I will connect the receiver to the flight controller and add a Lipo battery for the power supply. Then I will check the UAV is flying or not.

**Phase 3:** Now I will add radio telemetry to the flight controller and program the UAV.

**Phase 4:** I will mount the camera on the UAV. Then I will connect the transmitter and receiver to the camera and monitor the video on the mobile device.

**Phase 5:** Now I will do programs like Human Tracking, Recognition, and Identification on the Video using machine learning and artificial intelligence.

### 4.2 Implementation of UAV

Here is a step-by-step summary of the literature review we did before we started the project.

**Part 1:** First, we talked to the project guide and decided that we would do the project on Wireless Sensor Networks (WSN) with the Internet of Things (IoT).

**Part 2:** Prof. of Kharagpur IIT. Sudip Misra Sir's Wireless Sensor Networks and Internet of Things course I study online. Also, I read some research papers that are suggested by the project guide from google scholar.

**Part 3:** Talking to the guide I decided to do a project on IoT-based Unmanned Aerial Vehicles (UAV) for surveillance using Wireless Sensor Networks (WSN).

**Part 4:** I started reading research papers on UAVs from Google Scholar. From there I came to know the working process of UAVs. And I know what we can do with UAVs. I also got help from many websites to know about UAVs.

**Part 5:** I researched what components are needed to complete the project. And I write down an estimated copy then came to a specific decision.

**Part 6:** I bought all the components according to the estimated copy from the market and made a plan for the project how to complete. The estimated copy of Table 4.1 is attached below with the component market price base on 2021.



Table 4.1: Estimated project cost

SL No	Name	Price (₹)
1	Quadrotor Frame 4-axis F450	820
2	4 x Motor A2212 10000KV	1,399
3	4 x ESC 30A	1,559
4	4 x Propeller	274
5	Ardupilot APM 2.8	4,000
6	Lipo Battery 11.1V	1,899
7	FlySky FS-i6	4,850
8	Lipo Battery Charger	465
9	4 x Ultrasonic Sensor	260
10	GPS Module	2,140
11	FPV 5.8G AV TX & RX	4,245
12	FPV Camera	1,599
13	3DR Radio Telemetry 433Mhz	2,874
14	XT60 Connector 5 pairs	279
<b>Total</b>		<b>26,663</b>

### 4.2.1 Hardware Implementation

Hardware Implementation is given below:

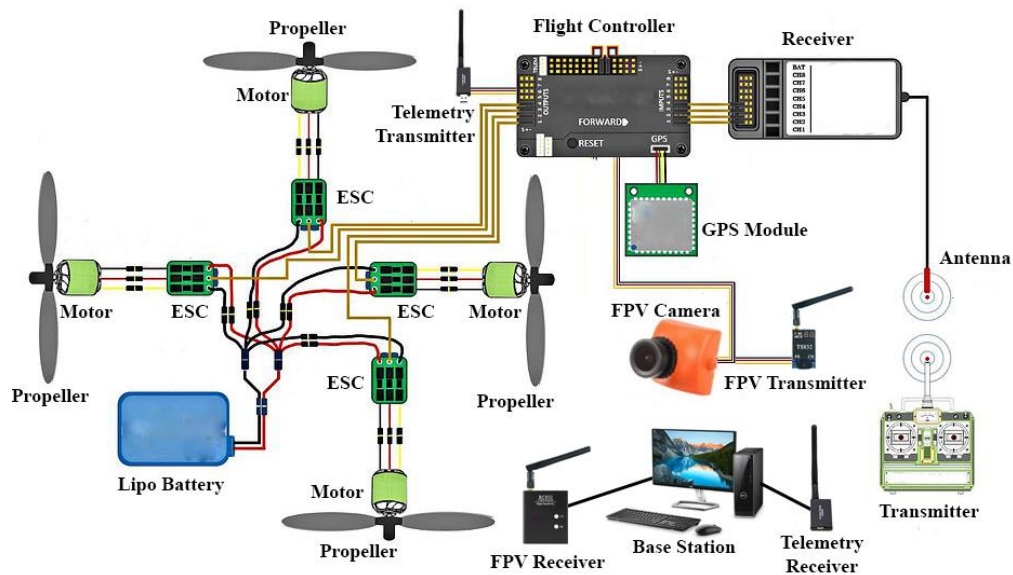


Figure 4.1: UAV hardware connection

After we got all the components, our first task was to check them properly and see if they were working properly. Figure 4.1 shows the UAV hardware connection. In this situation, we have to face the problem of low quality of ESC and Motor.

#### 4.2.1 Hardware Implementation

Then we observed the problem of ESC and Motor well, we solve it by welding the wires. Although every problem brings us to despair, we find the solution to that problem without bowing down. Moreover, after solving the problem every time, our interest in completing the project goes much further.

Now our main task begins by connecting all the components of the UAV. Before making the structure of the UAV, we have to keep in mind that the wires of ESC and APM Power Module are well welded in the power distribution board.

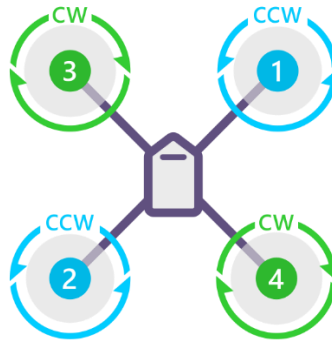


Figure 4.2: UAV motor rotation direction. Source: Adapted from [30]

After constructing the structure of the UAV frame, we connected the wires of the motor to the wires on the other end of the ESC. Figure 4.2 shows the UAV motor rotation direction. Here we need to combine two Motor clock wish and two other motors counter-clockwise, with ESC because this will help our UAV not to revolve around itself, i.e. to stay stable. Attached below is a figure where the direction of rotation of the motor is shown clockwise, CW, and counter-clockwise, CCW.

We then attached the flight controller and the GPS module in the center of gravity of the UAV. After attaching, one of the GPS module wairs was connected with the GPS port in the flight controller, and the other wair with the I2C port. Here I2C is for using a compass sensor. Then we attached the receiver, telemetry, and lipo battery to the UAV. Here the receiver is connected to the input ports of the flight controller and the telemetry to the telemetry port. We also attached ESC signal cables to the flight controller's output ports at the end, but it's not finished. Now we need to configure the UAV this will describe in the Flight controller configuration section.

## 4.2.1 Hardware Implementation

After configuration First, we attached a propeller with each motor then we will take a test flight of the UAV, if it is successful we will train ourselves in the training of a good UAV pilot so that we can fly our UAV like an expert UAV pilot.

When this is done we will connect the FPV Camera to the UAV and check if it is working properly, And also check if it can connect to the ground station or not. After that, we will collect video footage of the FPV Camera from the ground station and try to do programs like Human Tracking, Recognition, and Identification with it.

### 4.2.2 Flight controller configuration

Now we have installed mission planner software in the ground station which will help us to control the UAV from the ground station. After connecting the flight controller to the ground station, the firmware has to be installed in the flight controller so that the hardware of the UAV can be instructed properly.

Then we complete the configuration part of the UAV with the help of mission planner software, such as compass calibration, ESC calibration, and radio calibration, etc. It is described below using screenshots.

### Installing mission planner

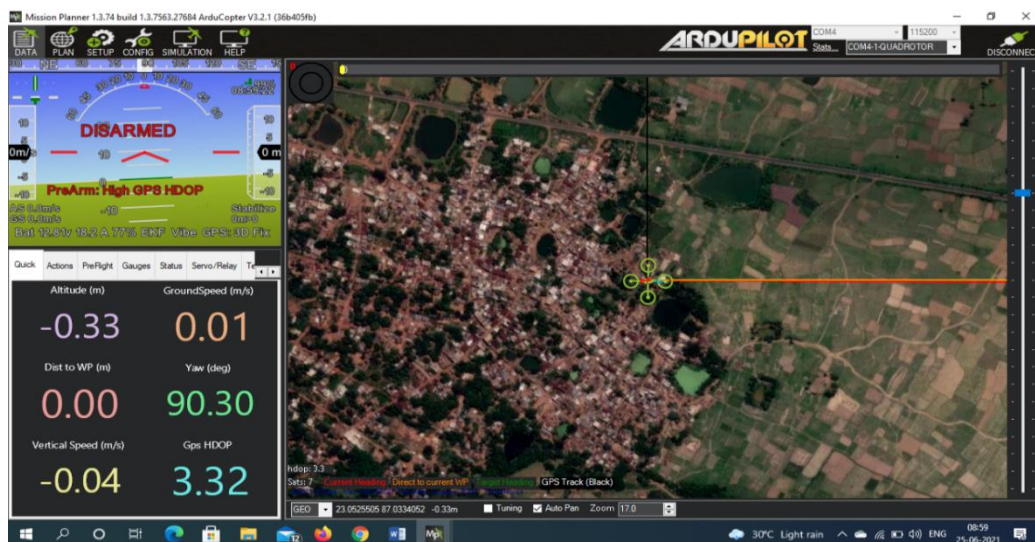


Figure 4.3: Installing mission planner.

## 4.2.2 Flight controller configuration

At first, we download mission planner 1.3.74 from the official website. Figure 4.3 is shown the mission planner Installing process. It is open-source software. Then we Install it on our ground station system.

Then connect the UAV flight controller by using a USB cable. Another possible way is available to connect through 3DR Radio Telemetry, but here we connect it the first time, and firmware not installed so we didn't use this process. With this software, we can monitor UAV from the base station.

### Installing Firmware

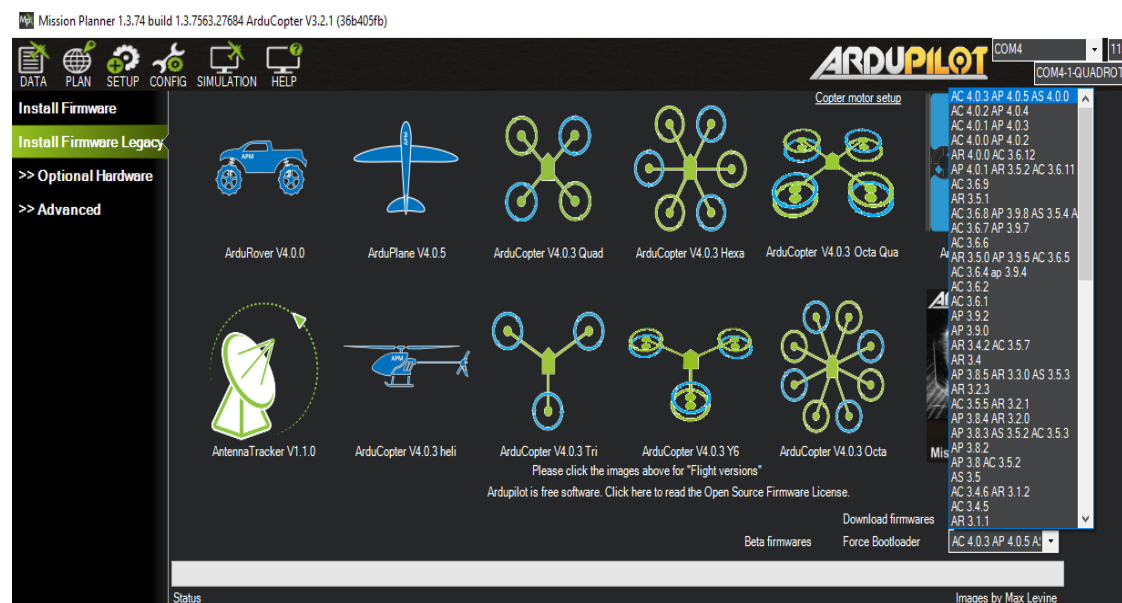


Figure 4.4: Installing Firmware.

Next, We open the Install Firmware tab (shown in Figure 4.4) by clicking on the Setup menu, select and install any firmware depending on the UAV Flight Controller.

It can take some time because it downloads from the internet then install in the UAV flight controller.

### Frame Type Configuration

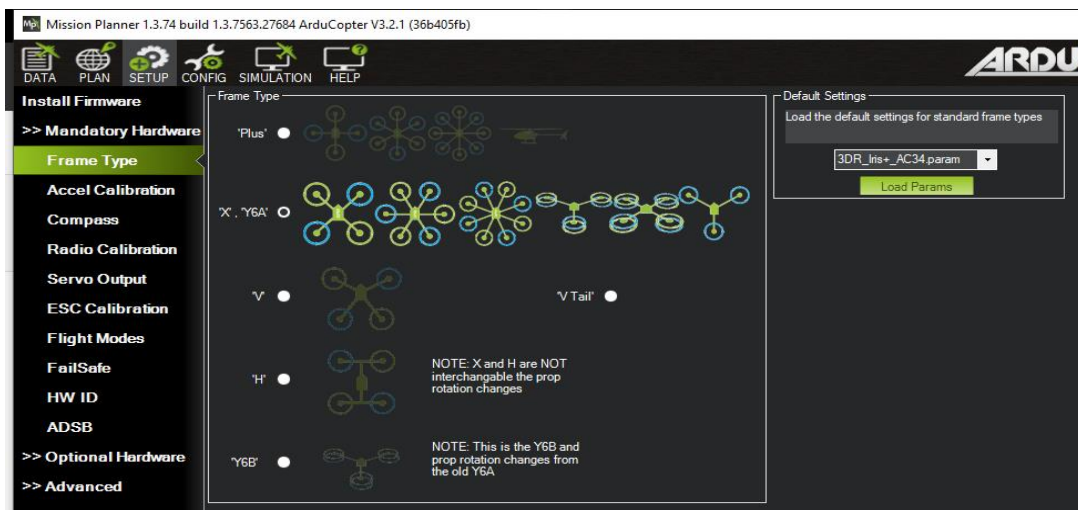


Figure 4.5: Frame Type Configuration.

After Firmware Installation we need to configure the flight controller by using some setup. The first configuration is started with choosing the frame type shown in Figure 4.5 is depending on the UAV structure. In our case, we used a quadcopter and the structure is like an X so we select 'X', 'Y6A'.

### Calibrate Acceleration

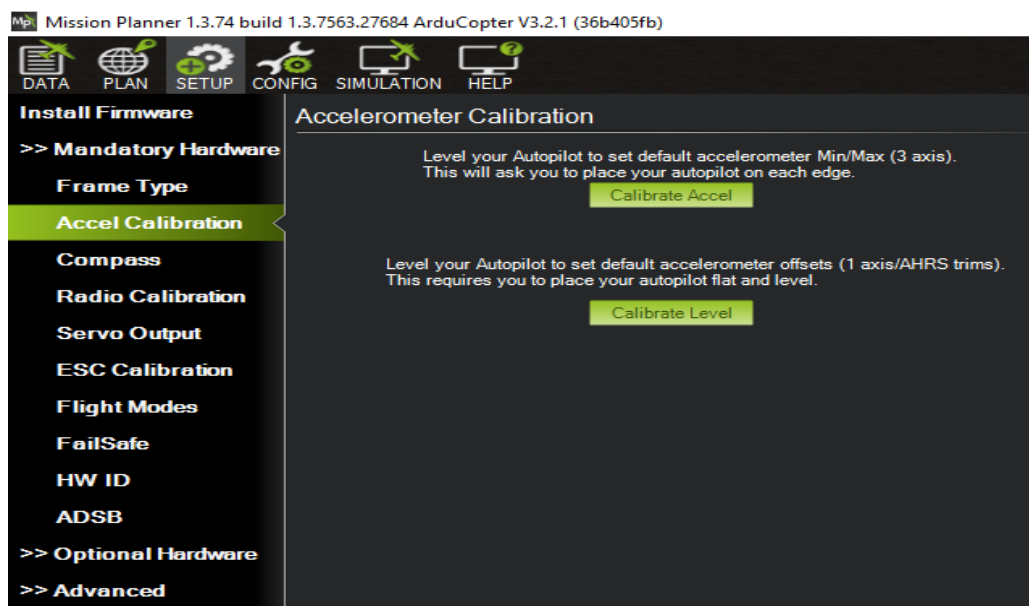


Figure 4.6: Calibrate Acceleration.

## 4.2.2 Flight controller configuration

Then we configure Accel Calibration shown in Figure 4.6 is also known as Accelerometer Calibration. It will be used to set up the UAV Accelerometer in the flight controller. Click the button Calibrate Accel to start this process.

One thing you need to keep in mind before starting this process is to first place the UAV on a level and press any key. Then set your UAV position left side and press any key. Then set your UAV position right side and press any key.

Then set your UAV position nose down and press any key. Then set your UAV position nose up and press any key. Then set your UAV position backside and press any key to finish this process. The next button Calibrate Level is not required to set up here because it is optional.

### Compass Calibration

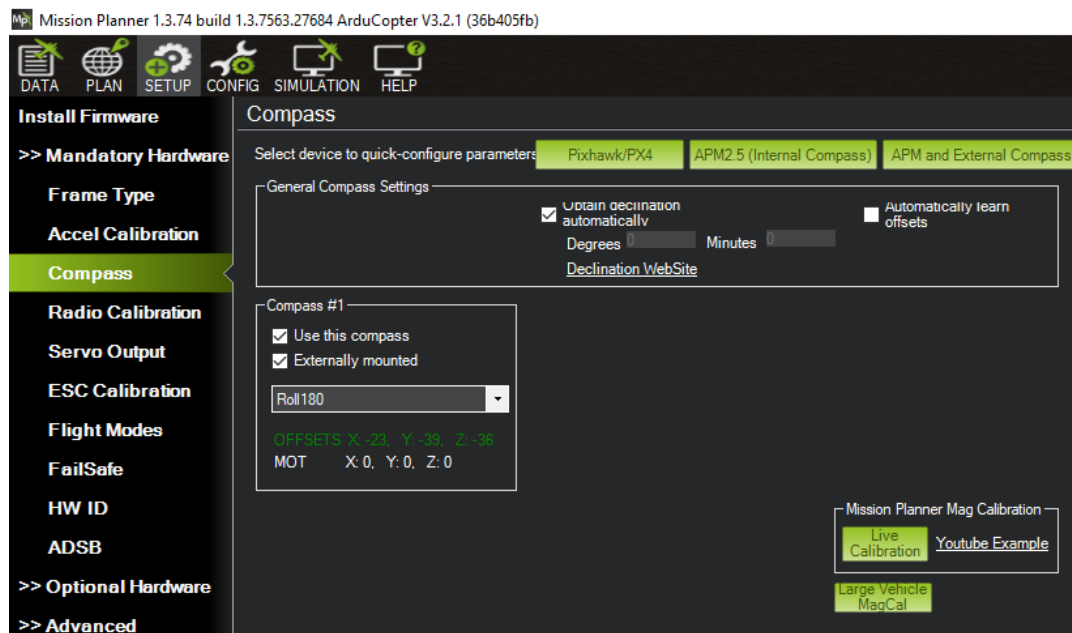


Figure 4.7: Compass Calibration.

The next configure step is Compass Calibration shown in Figure 4.7. Here two types of the compass are available in the UAV. One is the internal compass that belongs to the flight controller board and another one belongs to GPS. If you didn't use a GPS module then you may use compass calibration by using an Internal compass otherwise External compass.

## 4.2.2 Flight controller configuration

This process is started by clicking the Live Calibration button. The points are then collected by rotating the UAV 360 degrees in all positions. This process is then done by saving the offset value.

### Radio Control Calibration

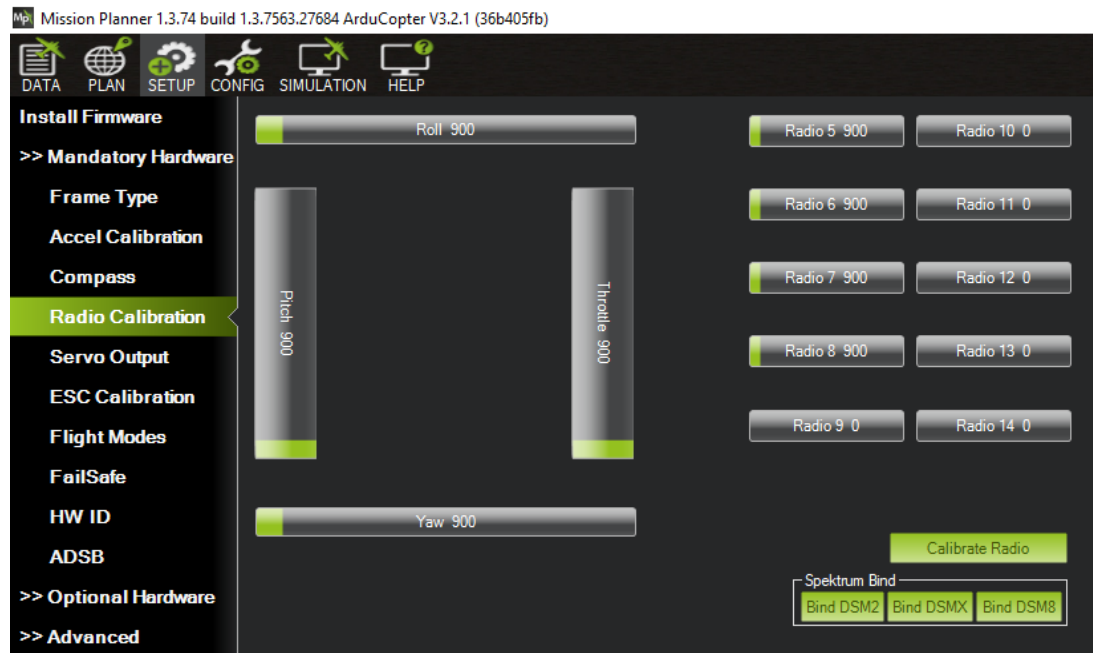


Figure 4.8: Radio Control Calibration.

Radio Control Calibration shown in Figure 4.8 is used for UAV controlling by Remote Control. This process requires Radio Control Transmitter and Receiver. This process starts with clicking Calibrate Radio button after turn on the Transmitter. Then move all switches to their maximum and minimum position and save them by clicking the click when done button.

This Calibration is mainly used for identifying the maximum and minimum value of all switches of the Transmitter. UAVs default channel mappings are ch1 for Roll, ch2 for Pitch, ch3 for Throttle, ch4 for Yaw, and ch5 for Flight modes.



### Electronic Speed Controller Calibration

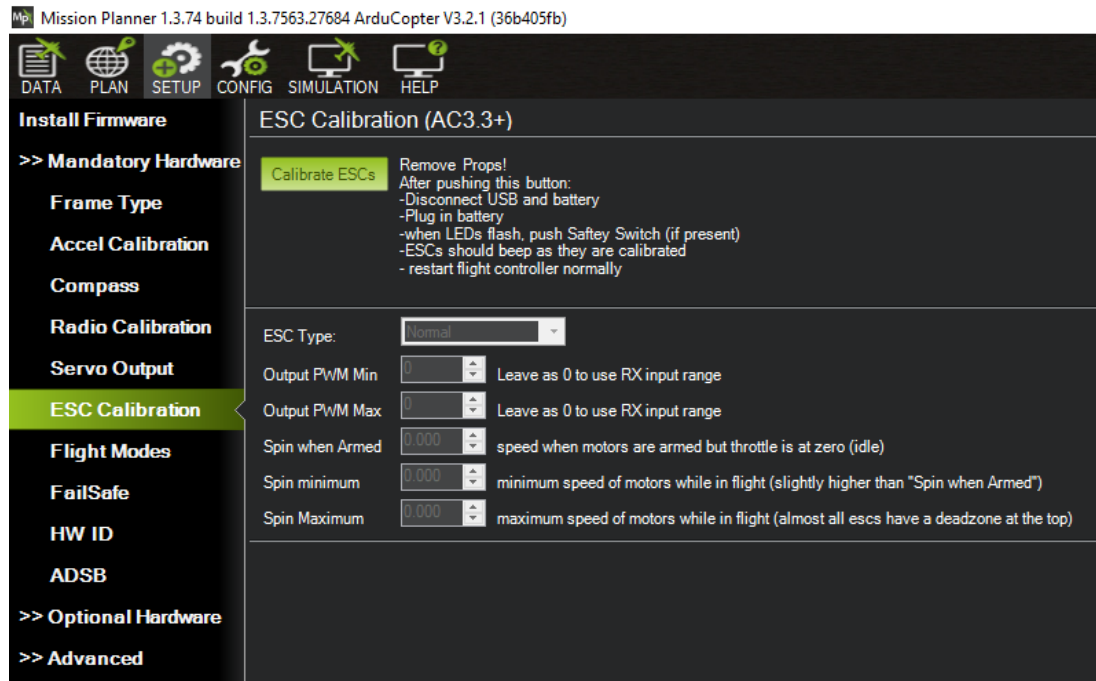


Figure 4.9: Electronic Speed Controller Calibration.

ESC calibration (shown in Figure 4.9) is a basic calibration of a UAV. It can configure by using Mission Planner software but manually configuration is easier over it.

In manual configuration at first turn on the transmitter then set the throttle to its maximum position. Then connect battery for power supplying to the UAV and wait for six-second then disconnects the battery. Now reconnect battery then ESC makes two beep sounds. Then you set the transmitter throttle to its minimum position and ESC makes three more beep sounds (For 3S battery) to complete the ESC calibration process.

This ESC calibration required to set the same value for each ESC, So that all motors rotated at the same speed.



### Flight Mode Configuration

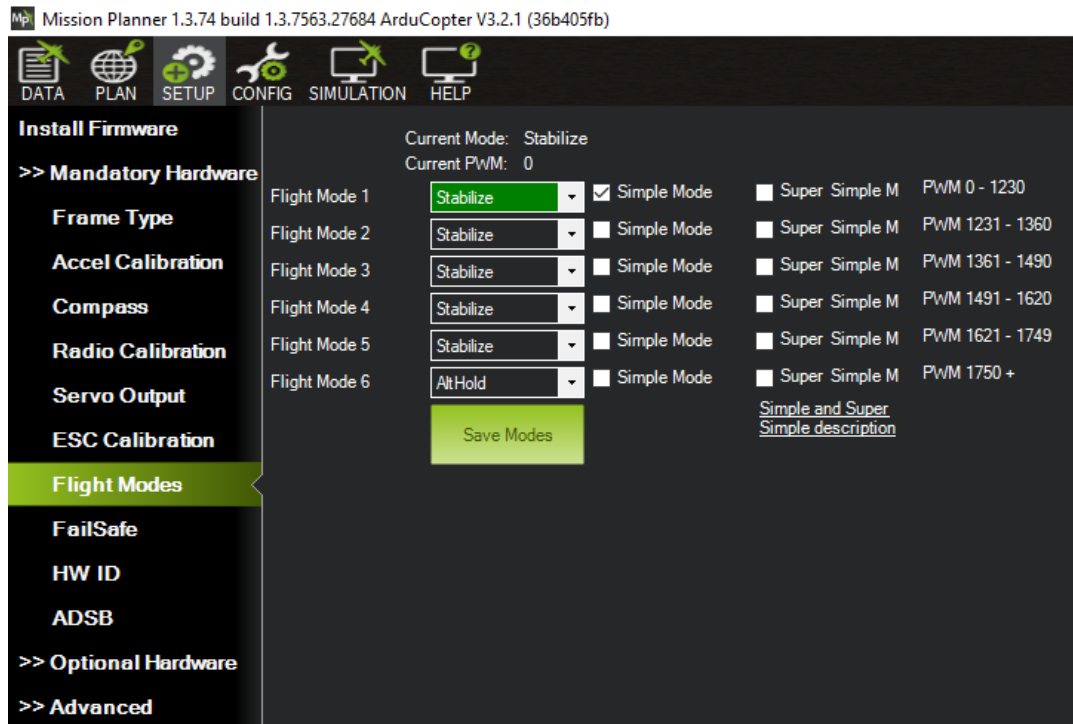


Figure 4.10: Flight Mode Configuration.

In this process, we use for Flight modes configuration shown in Figure 4.10. Here we choose flight modes. Maximum six modes are possible to set here. Some popular models name is Stabilize, AltHold, Loiter, Acro, and RTL, etc.

Those each mode has a different flying function. This process is required UAV Remote Control Transmitter and Receiver. And it uses over channel ch7, ch8 option switches. To complete this process by clicking the Save Modes button.

### Failsafe

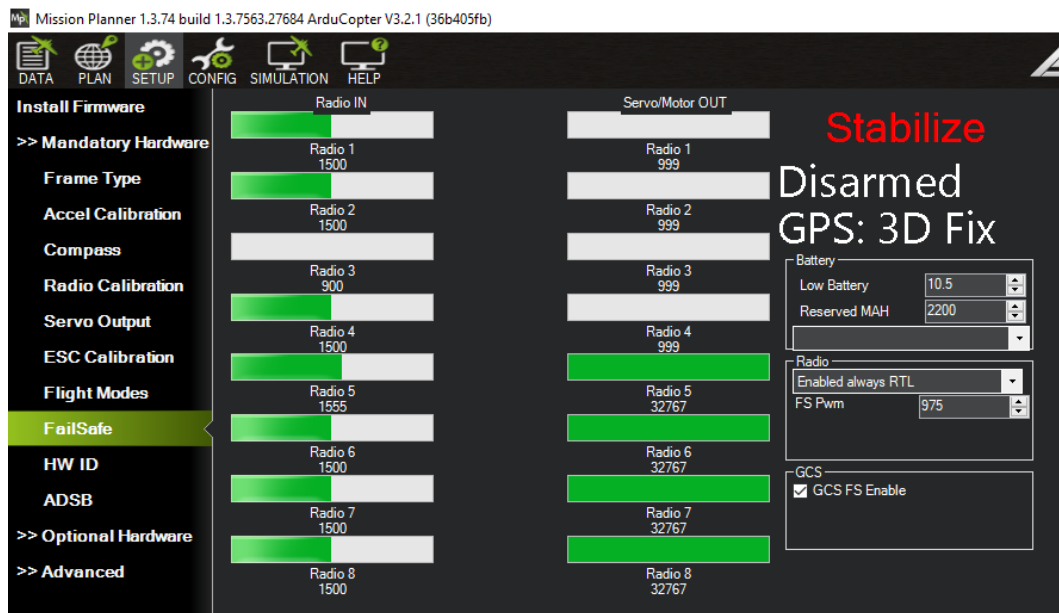


Figure 4.11: Failsafe.

UAV failsafe shown in Figure 4.11 is used for recovery/prevent failure during flying. There are two types of mechanisms available one is by using mission planner software and another one is by using Radio Transmitter.

Once the UAV faced some problem then they did not change modes automatically, so that using this failsafe process it changes modes automatically after configuration. UAVs Fail problem is Radio Failsafe, GSC Failsafe, Battery Failsafe, etc.

### Mission Planning



Figure 4.12: Mission Planning.

Nowadays mission planning shown in Figure 4.12 is the best feature of a UAV. This is easily done using mission planner software. In this process, you will be able to set a mission that can be complete automatically. In the above screenshot, we show you an example.

The mission starts with an automatic takeoff and then continues until Waypoint 2. Once there, proceed on the way to Waypoint 3 from there and finally return to the launch. Upon reaching the launch position, the UAV will land the craft.

This time we have to face the problem again, due to not support firmware in all hardware. Moreover, we know that this step is very important for UAVs, so we solve the problem without giving up hope.

### 4.2.3 Developed object detection models

#### Object detection project structure

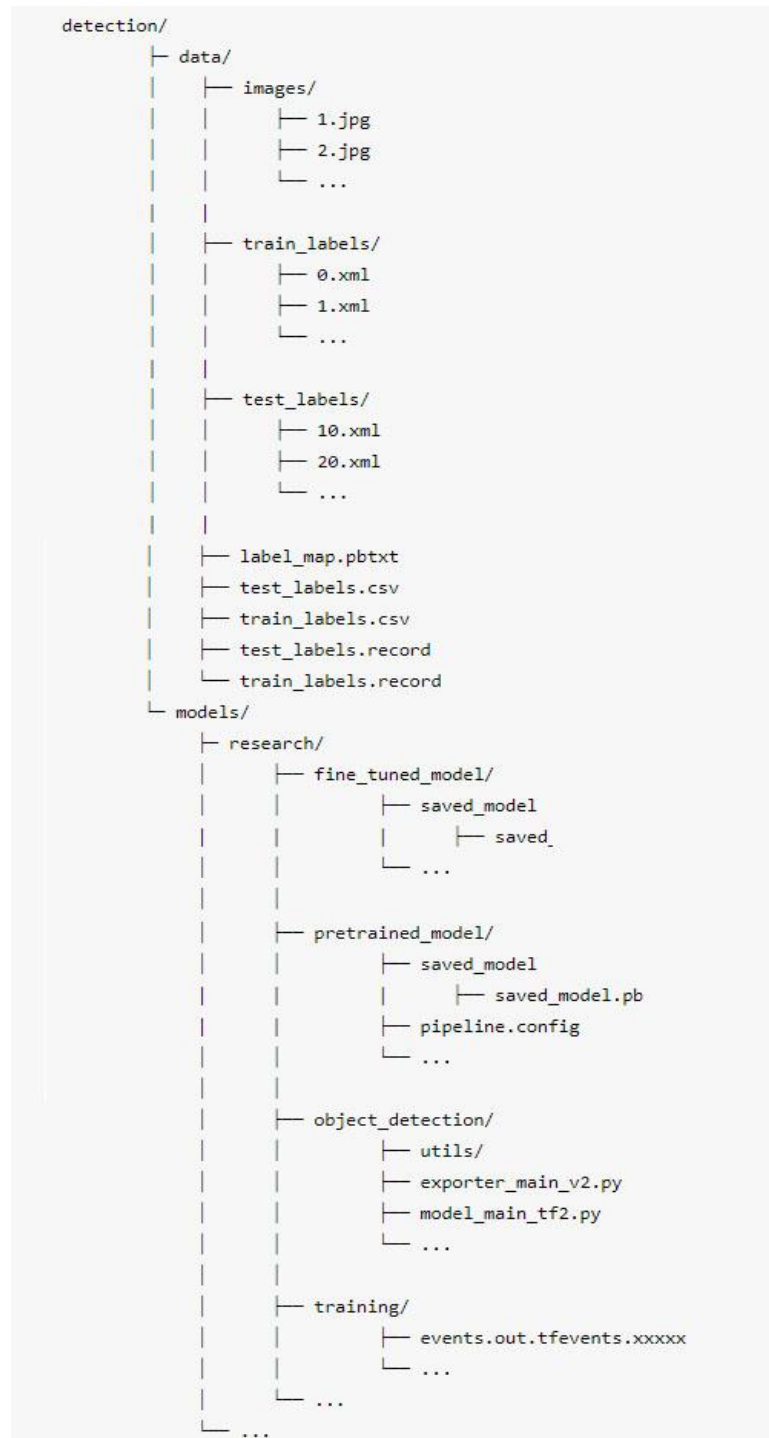


Figure 4.13: Object detection project structure.

### 4.2.3 Developed object detection models

The structure shown in Figure 4.13 is the above folders and files has been used in this project to develop a custom object detection model. The "detection/" here is the main folder where we have stored all the programs in this project. This folder contains the "data/" folder which is used to store the images in the "images/" folder, the labels annotations files divide into two-part in the "tain\_labels/" and "test\_labels/" folder, and also a \*.pbtxt file, two \*.csv files, and two \*.record files.

This folder also contains the "models/" folder which is used to store the TensorFlow model, here "object\_detection/" folder holds the TensorFlow object detection model, "pretrained\_model/" holds the pre-trained model that we choose for this project, "training/" folder holds the training data when we train our model and "fine\_tuned\_model/" folder holds exported our custom model.

#### Pre-training model configuration

```
import os
# Number of training steps.
num_steps = 20000 # 200000
# Number of evaluation steps.
num_eval_steps = 50
# set fine_tune_checkpoint_type.
checkpoint_type = 'detection'

test_record_file = '/content/detection/data/test_labels.record'
train_record_file = '/content/detection/data/train_labels.record'
label_map_pbtxt_file = '/content/detection/data/label_map.pbtxt'
fine_tune_checkpoint = '/content/detection/models/research/pretrained_model/checkpoint/ckpt-0'

# Some models to train on
MODELS_CONFIG = {
    # SSD MobileNet V2 FPNLite
    'ssd_mobilenet_v2_fpn_keras': {
        'model_name': 'ssd_mobilenet_v2_fpn_lite_640x640_coco17_tpu-8',
        'pipeline_file': 'pipeline.config',
        'batch_size': 8
    }
}

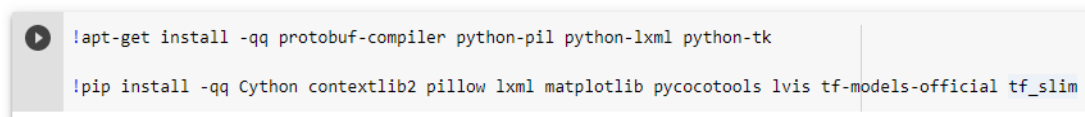
# Select a model in `MODELS_CONFIG`.
selected_model = 'ssd_resnet101_v1_fpn_keras'
# Name of the object detection model to use.
MODEL = MODELS_CONFIG[selected_model]['model_name']
# Name of the pipeline file in tensorflow object detection API.
pipeline_file = MODELS_CONFIG[selected_model]['pipeline_file']
# the path to the folder containing all the sample config files
CONFIG_BASE = "/content/detection/models/research/pretrained_model/"
# path to the specified model's config file
model_pipeline = os.path.join(CONFIG_BASE, pipeline_file)
# Training batch size fits in Colabe's Tesla K80 GPU memory for selected model.
batch_size = MODELS_CONFIG[selected_model]['batch_size']
```

Figure 4.14: Pre-training model configuration.

### 4.2.3 Developed object detection models

The program above (shown in Figure 4.14) is used for defined purposes where we initiate some variables for further use. In this set of code, we use our entire program to configure from one place. Here we choosing a pre-training model for this project is "ssd\_mobilenet\_v2\_keras". Because the interests of this project are to interfere with real-time video surveillance, SO choosing a model that has a high inference speed (ms) with relatively high (mAP) on COCO.

#### Installing Required Packages

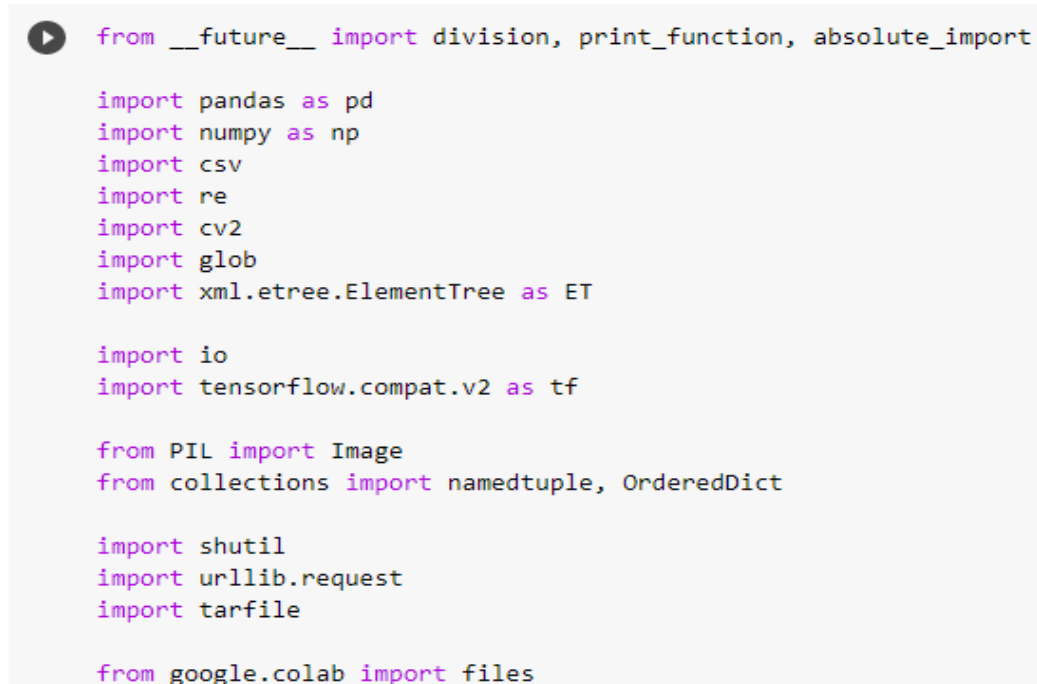


```
!apt-get install -qq protobuf-compiler python-pil python-lxml python-tk
!pip install -qq Cython contextlib2 pillow lxml matplotlib pycocotools lvis tf-models-official tf_slim
```

Figure 4.15: Installing Required Packages.

The above code is shown in Figure 4.15 used for installing Required Packages. Here we install all Packages that we need for developing a custom detection model.

#### Imports library



```
from __future__ import division, print_function, absolute_import

import pandas as pd
import numpy as np
import csv
import re
import cv2
import glob
import xml.etree.ElementTree as ET

import io
import tensorflow.compat.v2 as tf

from PIL import Image
from collections import namedtuple, OrderedDict

import shutil
import urllib.request
import tarfile

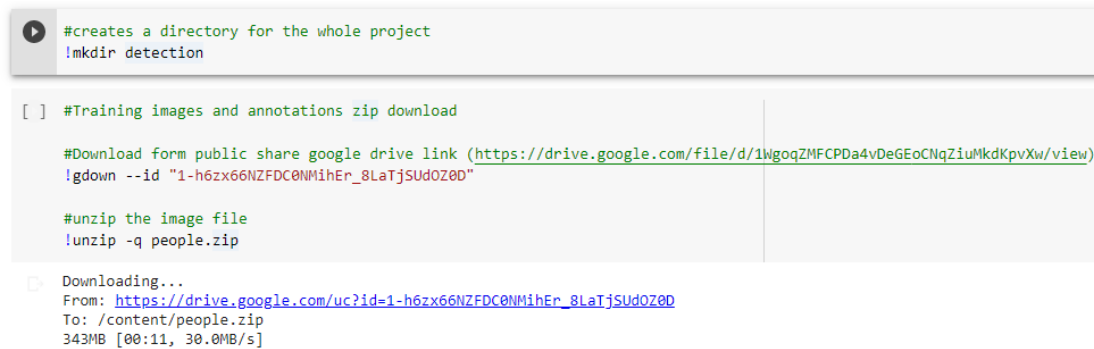
from google.colab import files
```

Figure 4.16: Imports library.

### 4.2.3 Developed object detection models

The above code (shown in Figure 4.16) is used for the Imports library will be done after downloading and installing some packages. Here we Import all libraries that we need for developing a custom detection model.

#### Downloading Images and Annotations



```
#creates a directory for the whole project
!mkdir detection

[ ] #Training images and annotations zip download

#Download form public share google drive link (https://drive.google.com/file/d/1WgoqZMFCDa4vDeGEoCNqZiuMkdKpvXw/view)
!gdown --id "1-h6zx66NZFDC0NMihEr_8LaTjSUd0Z0D"

#unzip the image file
!unzip -q people.zip

Downloading...
From: https://drive.google.com/uc?id=1-h6zx66NZFDC0NMihEr_8LaTjSUd0Z0D
To: /content/people.zip
343MB [00:11, 30.0MB/s]
```

Figure 4.17: Downloading Images and Annotations.

The above code is shown in Figure 4.17 used for making a folder named "detection/" in the root. After that code will be download from the dataset folder from google drive and unzip it. We previously prepared this dataset by using LabelImg software. It creates a \*.xml annotation label file with help of our capture footage.

### Organizing Images and Annotations

```
[ ] cd /content/detection

/content/detection

# creating a directory to store the training and testing data
!mkdir data

# folders for the training and testing data.
!mkdir data/images data/train_labels data/test_labels

# combining the images and annotation in the training folder:
# moves the images to data folder
!mv /content/data/images/* /content/detection/data/images

# moves the annotations to train_labels folder
!mv /content/data/annotations/* /content/detection/data/train_labels

[ ] # Deleting the zipped and unzipped folders
!rm -rf /content/data /content/people.zip
```

Figure 4.18: Organizing Images and Annotations.

The above code is shown in Figure 4.18 used for the first entry in the "detection/" folder then creates a new folder named "data/". Then create three folders under the "data/" folder named "images", "train\_labels/" and, "test\_labels/". After that moving our download dataset image and annotation to the "images" and "train\_labels/" folder. And then remove the download dataset empty folder.

### Splitting labels into 80% training and 20% testing

```
!ls data/train_labels/* | sort -R | head -300 | xargs -I{} mv {} data/test_labels

[ ] # 1200 "images"(xml) for training
!ls -l /content/detection/data/train_labels/ | wc -l

1200

[ ] # 300 "images"(xml) for testing
!ls -l /content/detection/data/test_labels/ | wc -l

300
```

Figure 4.19: Splitting labels into 80% training and 20% testing.



### 4.2.3 Developed object detection models

The above code shown in Figure 4.19 is used for "train\_labels/" folder XML files splitting into two-part 80% : 20% and store it on "train\_labels/" and "test\_labels/" folder. After that, we check by counting the number of files in the folder.

#### Converting the annotations from XML files to two CSV files

```
%cd /content/detection/data

def xml_to_csv(path):
    classes_names = []
    xml_list = []

    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            classes_names.append(member[0].text)
            value = (root.find('filename').text,
                    int(root.find('size')[0].text),
                    int(root.find('size')[1].text),
                    member[0].text,
                    int(member[4][0].text),
                    int(member[4][1].text),
                    int(member[4][2].text),
                    int(member[4][3].text))
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    classes_names = list(set(classes_names))
    classes_names.sort()
    return xml_df, classes_names

# for both the train_labels and test_labels csv files, it runs the xml_to_csv() above.
for label_path in ['train_labels', 'test_labels']:
    image_path = os.path.join(os.getcwd(), label_path)
    xml_df, classes = xml_to_csv(label_path)
    xml_df.to_csv(f'{label_path}.csv', index=None)
    print(f'Successfully converted {label_path} xml to csv.')

# Creating the `label_map.pbtxt` file
label_map_path = os.path.join("label_map.pbtxt")

pbtxt_content = ""

#creates a pbtxt file the has the class names.
for i, class_name in enumerate(classes):
    # display_name is optional.
    pbtxt_content = (
        pbtxt_content
        + "item {\n    id: {0}\n    name: '{1}'\n"
        + "display_name: '{1}'\n }\n\n".format(i + 1, class_name)
    )
pbtxt_content = pbtxt_content.strip()
with open(label_map_path, "w") as f:
    f.write(pbtxt_content)

/content/detection/data
Successfully converted train_labels xml to csv.
Successfully converted test_labels xml to csv.
```

Figure 4.20: Converting the annotations from XML files to two CSV files.

### 4.2.3 Developed object detection models

The above code is shown in Figure 4.20 used for Preprocessing Images and Labels. Converting the annotations from \*.xml files to two \*.csv files for each "train\_labels/" and "test\_labels/" folders. \*.csv file contains image name, width, height, class, xmin, ymin, xmax, ymax. if the \*.xml file contains more than one box/label, it will create more than one row for the same image. each row contains the info for an individual box. Also creating a \*.pbt.txt file that specifies the number of classes and display\_name is optional.

#### Checks images box position in CSV files

```
%cd /content/detection/data
images_path = 'images'

for CSV_FILE in ['train_labels.csv', 'test_labels.csv']:
    with open(CSV_FILE, 'r') as fid:
        print('[*] Checking file:', CSV_FILE)
        file = csv.reader(fid, delimiter=',')
        first = True
        cnt = 0
        error_cnt = 0
        error = False
        for row in file:
            if error == True:
                error_cnt += 1
                error = False
            if first == True:
                first = False
                continue
            cnt += 1
            name, width, height, xmin, ymin, xmax, ymax = row[0], int(row[1]),
                int(row[2]), int(row[4]), int(row[5]), int(row[6]), int(row[7])
            path = os.path.join(images_path, name)
            img = cv2.imread(path)
            if type(img) == type(None):
                error = True
                print('Could not read image', img)
                continue
            org_height, org_width = img.shape[:2]
            if org_width != width:
                error = True
                print('Width mismatch for image: ', name, width, '!=', org_width)
            if org_height != height:
                error = True
                print('Height mismatch for image: ', name, height, '!=', org_height)
            if xmin > org_width:
                error = True
                print('XMIN > org_width for file', name)
            if xmax > org_width:
                error = True
                print('XMAX > org_width for file', name)
            if ymin > org_height:
                error = True
                print('YMIN > org_height for file', name)
            if ymax > org_height:
                error = True
                print('YMAX > org_height for file', name)
            if error == True:
                print('Error for file: %s' % name)
                print()
        print()
        print('Checked %d files and realized %d errors' % (cnt, error_cnt))
        print("-----")
```

Figure 4.21: Checks images box position in CSV files.

### 4.2.3 Developed object detection models

The above code is shown in Figure 4.21 used for checking \*.csv file entry data for "images/" folder image. Checking if the annotations for each object are placed within the range of the image width and height. If any error occurred then that will be displayed here.

#### Downloading and Preparing Tensorflow model

```
[ ] # Downloads Tensorflow
%cd /content/detection/
!git clone --q https://github.com/tensorflow/models.git

/content/detection

[ ] %cd /content/detection/models/research
#compiling the proto buffers (not important to understand for this project but you can learn more about them here:
!protoc object_detection/protos/*.proto --python_out=.

# exports the PYTHONPATH environment variable with the reasearch and slim folders' paths
os.environ['PYTHONPATH'] += ':/content/detection/models/research:/content/detection/models/research/slim/'

/content/detection/models/research

[ ] # testing the model builder
!python3 object_detection/builders/model_builder_tf2_test.py
```

Figure 4.22: Downloading and Preparing Tensorflow model.

The above code is shown in Figure 4.22 used for downloading and preparing the Tensorflow model. Firstly cloning Tensorflow models from the official Github repo. The repo contains the object detection API we are interested in. Then compiling the proto buffers and adding folders to the os environment variable with the "research/" and "slim/" folders' paths. After that testing the model builder.

### Generating TF record files

```

from object_detection.utils import dataset_util
%cd /content/detection/models

DATA_BASE_PATH = '/content/detection/data/'
image_dir = DATA_BASE_PATH + 'images/'

def class_text_to_int(row_label):
    if row_label == 'people':
        return 1
    else:
        return None

def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in zip(gb.groups.keys(), gb.groups)]

def create_tf_example(group, path):
    with tf.io.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
        encoded_jpg_io = io.BytesIO(encoded_jpg)
        image = Image.open(encoded_jpg_io)
        width, height = image.size

        filename = group.filename.encode('utf8')
        image_format = b'jpg'
        xmins = []
        xmaxs = []
        ymins = []
        ymaxs = []
        classes_text = []
        classes = []

        for index, row in group.object.iterrows():
            xmins.append(row['xmin'] / width)
            xmaxs.append(row['xmax'] / width)
            ymins.append(row['ymin'] / height)
            ymaxs.append(row['ymax'] / height)
            classes_text.append(row['class'].encode('utf8'))
            classes.append(class_text_to_int(row['class']))

        tf_example = tf.train.Example(features=tf.train.Features(feature={
            'image/height': dataset_util.int64_feature(height),
            'image/width': dataset_util.int64_feature(width),
            'image/filename': dataset_util.bytes_feature(filename),
            'image/source_id': dataset_util.bytes_feature(filename),
            'image/encoded': dataset_util.bytes_feature(encoded_jpg),
            'image/format': dataset_util.bytes_feature(image_format),
            'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
            'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
            'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
            'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
            'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
            'image/object/class/label': dataset_util.int64_list_feature(classes),
        }))
    return tf_example

for csv in ['train_labels', 'test_labels']:
    writer = tf.io.TFRecordWriter(DATA_BASE_PATH + csv + '.record')
    path = os.path.join(image_dir)
    examples = pd.read_csv(DATA_BASE_PATH + csv + '.csv')
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())

    writer.close()
    output_path = os.path.join(os.getcwd(), DATA_BASE_PATH + csv + '.record')
    print('Successfully created the TFRecords: {}'.format(DATA_BASE_PATH + csv + '.record'))

/content/detection/models
Successfully created the TFRecords: /content/detection/data/train_labels.record
Successfully created the TFRecords: /content/detection/data/test_labels.record

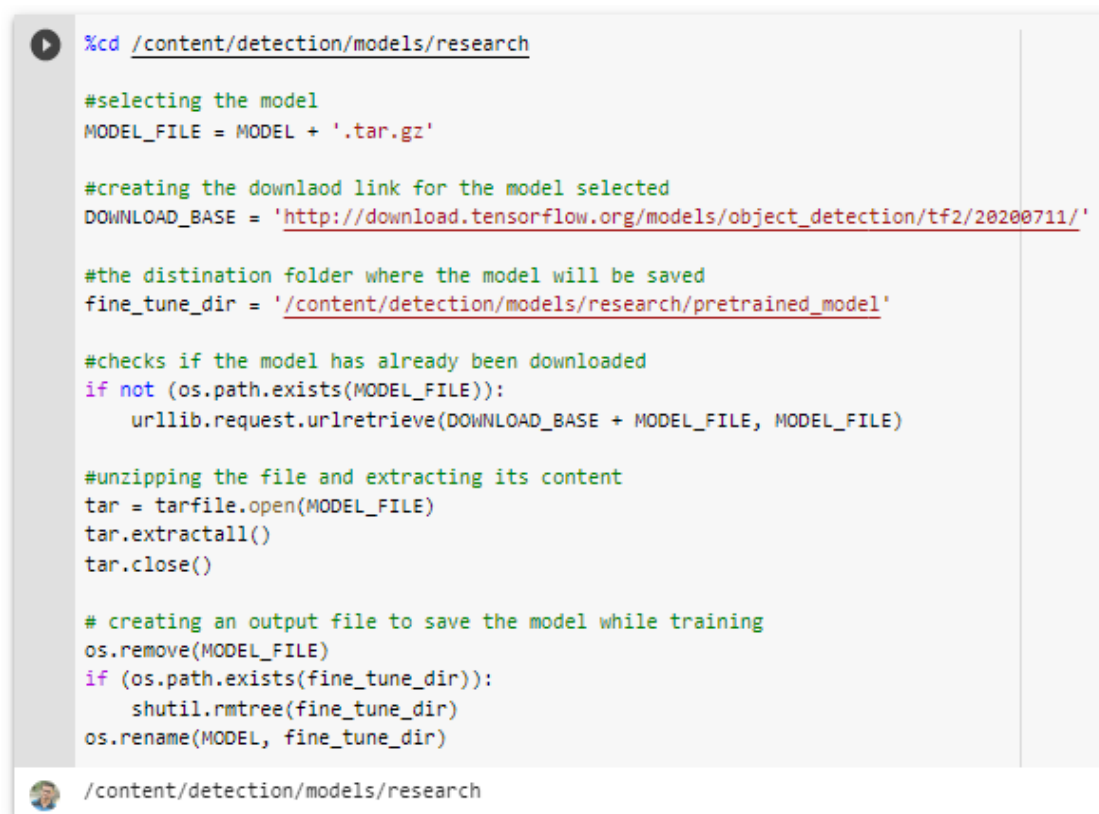
```

Figure 4.23: Generating TF record files.

### 4.2.3 Developed object detection models

The above code is shown in Figure 4.23 used for Generating two TFRecords files for the training and testing \*.csv file. Tensorflow accepts data in the form of TFRecords, a binary file that runs quickly and uses little memory. Instead of loading the entire data set into memory, Tensorflow uses these TFRecords to automatically divide the data into chunks.

#### Downloading the selecting model (Pretrained model)



```
%cd /content/detection/models/research

#selecting the model
MODEL_FILE = MODEL + '.tar.gz'

#creating the download link for the model selected
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/'

#the destination folder where the model will be saved
fine_tune_dir = '/content/detection/models/research/pretrained_model'

#checks if the model has already been downloaded
if not (os.path.exists(MODEL_FILE)):
    urllib.request.urlretrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)

#unzipping the file and extracting its content
tar = tarfile.open(MODEL_FILE)
tar.extractall()
tar.close()

# creating an output file to save the model while training
os.remove(MODEL_FILE)
if (os.path.exists(fine_tune_dir)):
    shutil.rmtree(fine_tune_dir)
os.rename(MODEL, fine_tune_dir)
```

/content/detection/models/research

Figure 4.24: Downloading the selecting model (Pretrained model).

The above code is shown in Figure 4.24 used for downloading the base model selected and extracting its content. Here based on the model selecting a previously pre-training model, what we used for this project. And creating the destination folder named "pretrained\_model/", where the model will be saved.



## Configuring the Training Pipeline

```
def get_num_classes(pbtxt_fname):
    from object_detection.utils import label_map_util
    label_map = label_map_util.load_labelmap(pbtxt_fname)
    categories = label_map_util.convert_label_map_to_categories(
        label_map, max_num_classes=90, use_display_name=True)
    category_index = label_map_util.create_category_index(categories)
    return len(category_index.keys())

num_classes = get_num_classes(label_map_pbtxt_file)

#open the file
with open(model_pipeline) as f:
    s = f.read()

with open(model_pipeline, "w") as f:

    # fine_tune_checkpoint
    s = re.sub('fine_tune_checkpoint: ".*?"',
               'fine_tune_checkpoint: "{}".format(fine_tune_checkpoint), s)

    # tfrecord files train and test.
    s = re.sub('input_path: ".*?"',
               'input_path: "{}".format(test_record_file), s, 0)

    s = re.sub('input_path: ".*?"',
               'input_path: "{}".format(train_record_file), s, 1)

    # label_map_path
    s = re.sub('label_map_path: ".*?"',
               'label_map_path: "{}".format(label_map_pbtxt_file), s)

    # fine_tune_checkpoint_type
    s = re.sub('fine_tune_checkpoint_type: ".*?"',
               'fine_tune_checkpoint_type: "{}".format(checkpoint_type), s)

    # Set training batch_size.
    s = re.sub('batch_size: [0-9]+',
               'batch_size: {}'.format(batch_size), s)

    # Set training steps, num_steps
    s = re.sub('num_steps: [0-9]+',
               'num_steps: {}'.format(num_steps), s)

    # Set number of classes num_classes.
    s = re.sub('num_classes: [0-9]+',
               'num_classes: {}'.format(num_classes), s)
    f.write(s)
```

Figure 4.25: Configuring the Training Pipeline.

The given code shown in Figure 4.25 is used to configure the training pipeline by putting the path to the TFRecords files, checkpoint file, and \*.pbtxt file to the configuration file, as well as batch\_size, num\_steps, and num\_classes. And set fine\_tune\_checkpoint\_type as "detection" in this project.

### Training model output directory setup

```
# where the model will be saved at each checkpoint while training
model_dir = 'training/'

# Optionally: remove content in output model directory to fresh start.
!rm -rf {model_dir}
os.makedirs(model_dir, exist_ok=True)
```

Figure 4.26: Training model output directory setup.

The above code is shown in Figure 4.26 is used for creating a folder named "training/", where the model will be saved at each checkpoint while training and remove the content in the output model folder to a fresh start.

### Training model

```
[ ] %cd /content/detection/models/research

/content/detection/models/research

[ ] !python3 /content/detection/models/research/object_detection/model_main_tf2.py \
    --pipeline_config_path={model_pipeline} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --num_eval_steps={num_eval_steps}
```

Figure 4.27: Training model.

The above code shown in Figure 4.27 is used to enter in the "research/" folder then Training the model by using the model\_main\_tf2.py file.

### Training evaluation model

```
[ ] !python3 /content/detection/models/research/object_detection/model_main_tf2.py \
    --pipeline_config_path={model_pipeline} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --num_eval_steps={num_eval_steps} \
    --checkpoint_dir={model_dir} \
```

Figure 4.28: Training evaluation model.

### 4.2.3 Developed object detection models

---

The above code shown in Figure 4.28 is used to display evaluation metrics while gathering evaluation results. This code is also the same as the model training code adding with `checkpoint_dir` path. Here we check the Average Precision and Average Recall value.

#### Exporting the trained model

```
[ ] #the location where the exported model will be saved in.
    output_directory = '/content/detection/models/research/fine_tuned_model'

#goes through the model is the training/ dir and gets the last one
#exports the model specified and inference graph
!python3 /content/detection/models/research/object_detection/exporter_main_v2.py \
    --input_type=image_tensor \
    --pipeline_config_path={model_pipeline} \
    --output_directory={output_directory} \
    --trained_checkpoint_dir={model_dir} \
```

Figure 4.29: Exporting the trained model.

The coding shown in Figure 4.29 above specifies the place where the export model will be saved. It retrieves the final model in the "training/" folder and exports the model specified as well as the inference graph into "fine\_tuned\_model/."



## 4.2.3 Developed object detection models

### Run inference graph model test

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # Suppress TensorFlow logging (1)
import pathlib
import tensorflow as tf
import cv2
import argparse
from google.colab.patches import cv2_imshow
IMAGE_PATHS = '/content/detection/data/images/0000357_00981_d_0000661.jpg'
PATH_TO_MODEL_DIR = '/content/detection/models/research/fine_tuned_model'
PATH_TO_LABELS = '/content/detection/data/label_map.pbtxt'
MIN_CONF_THRESH = float(0.60)

# LOAD THE MODEL
import time
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils

PATH_TO_SAVED_MODEL = PATH_TO_MODEL_DIR + "/saved_model"

print('Loading model...', end='')
start_time = time.time()
# LOAD SAVED MODEL AND BUILD DETECTION FUNCTION
detect_fn = tf.saved_model.load(PATH_TO_SAVED_MODEL)
# LOAD LABEL MAP DATA FOR PLOTTING
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
                                                                    use_display_name=True)

import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

def load_image_into_numpy_array(path):
    return np.array(Image.open(path))

print('Running inference for {}... '.format(IMAGE_PATHS), end='')

image = cv2.imread(IMAGE_PATHS)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_expanded = np.expand_dims(image_rgb, axis=0)

# The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
input_tensor = tf.convert_to_tensor(image)
# The model expects a batch of images, so add an axis with `tf.newaxis`.
input_tensor = input_tensor[tf.newaxis, ...]

# input_tensor = np.expand_dims(image_np, 0)
detections = detect_fn(input_tensor)

num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
              for key, value in detections.items()}
detections['num_detections'] = num_detections
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
image_with_detections = image.copy()
viz_utils.visualize_boxes_and_labels_on_image_array(
    image_with_detections,
    detections['detection_boxes'],
    detections['detection_classes'],
    detections['detection_scores'],
    category_index,
    use_normalized_coordinates=True,
    max_boxes_to_draw=200,
    min_score_thresh=0.4,
    agnostic_mode=False)

print('Done')
# DISPLAYS OUTPUT IMAGE
cv2_imshow(image_with_detections)
```

Figure 4.30: Run inference graph model test.

### 4.2.3 Developed object detection models

---

The above code shown in Figure 4.30 is used to run the inference graph model test, which is also known as the custom train model. Here we test images with the "fine\_tuned\_model/" folder. At first, we specify the location of the picture folder., model folder, and label map file, then need to provide a minimum confidence threshold.

## CHAPTER 5

### RESULT AND DISCUSSION

#### 5.1 Result



Figure 5.1: UAV project.

Since our project is hardware-based, So we have shown in Figure 5.1 above is how far our project has come. Our project is a real one where we have built a UAV by hand. And to make it more advanced, I programmed it like Object Detection with UAV video footage and described some of the results in Figure 5.2 is below.



Figure 5.2: Object detection test result.

This is the evaluation matrix shown in Figure 5.3 is our object detection model. Here precision, recall value is shown depending on small, medium, and large area.

```

creating index...
index created!
creating index...
index created!
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.08s).
Accumulating evaluation results...
DONE (t=0.03s).
Average Precision (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.748
Average Precision (AP) @[ IoU=0.50 | area= all | maxDets=100 ] = 1.000
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.917
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.717
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.773
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.408
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.767
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.767
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.733
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.783

```

Figure 5.3: Object detection evaluation matrix.

We have calculated here to evaluate the training model of our project. To test the training model, I first took 10 random images. Then I prediction them and listed the results in Table 5.1 is the list below.

Table 5.1: Test image prediction

Image	1.jpg	2.jpg	3.jpg	4.jpg	5.jpg	6.jpg	7.jpg	8.jpg	9.jpg	10.jpg
Truth	People	People	No People	People	No People	People	People	People	No People	No People
Predict	People	No People	People	People	No People	People	People	People	People	No People

The Image row above contains the names of the images we tested. The correct prediction is based on the images in the Truth row. And based on the results we got after testing our training model in Predict Row.

Based on the prediction results we create a confusion matrix in Table 5.2 is below.

There are some things you need to know before you know the confusion matrix.

- When an object is present in the image and the model can detect the object, it is called **True Positive** (TP).
- When an object is missing in the figure and the model can detect the object, it is called a **False Positive** (FP).
- When an object is present in the figure but the model fails to identify the object, it is called **False Negative** (FN).
- When an object is missing in the image and the model fails to identify the object, it is called **True Negative** (TN).

Table 5.2: Confusion matrix

		Actual	
		Positive	Negative
Predicted	Positive	TP = 5	FP = 2
	Negative	FN = 1	TN = 2

We can test our model by following some calculation formulas:

Accuracy is used to evaluate classification models. Accuracy is the ability to verify that a fraction of predictions are correct for a model or not. Accuracy is not always selected as the best model. The formula for accuracy is as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total no of predictions}}$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Accuracy} = \frac{5 + 2}{5 + 2 + 2 + 1} = 0.70$$

Here we get the accuracy of our model is 70%

The number of correct positive predictions made by the model is known as Precision. Following is the Precision formula:

$$\text{Precision} = \frac{\text{True Positive}}{\text{Actual Result}}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Precision} = \frac{5}{5 + 2} = 0.71$$

The number of correct positive predictions made out of all positive predictions made by the model is known as Recall. Following is the Recall formula:

$$\text{Recall} = \frac{\text{True Positive}}{\text{Predicted}}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Recall} = \frac{5}{5 + 1} = 0.83$$

The other measure is the F1 Score which is a function of Precision and Recall. F1 Score is required when you balance between Precision and Recall. It is a better measure technic over Accuracy. Following is the F1 Score formula:

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\text{F1 Score} = 2 * \frac{0.71 * 0.83}{0.71 + 0.83} = 0.76$$

In this project, we have started from the very basics and now we have reached this far. However, in this short time, it was not possible to develop software using the program of object detection with UAV. We wanted to do more with this project as described in the Future work below.

### 5.2 Discussion

What we wanted in our project has been fully implemented. We have to face many problems to make it happen, even though the problems are small, since we are new to this technology, it takes a lot of time to find a solution. The biggest problem is when configuring UAVs while flying. From this project, we can first learn the full functionality of UAVs. Then learn to fly it and finally build an object detection model based on it.

## **CHAPTER 6**

### **CONCLUSION AND FUTURE WORK**

#### **6.1 Conclusion**

We have completed our proposed work on time and we have included a description of it here. A lot of work is being done on this project nowadays. And our idea is that next time it will occupy one of the leading positions in our environment. But to do that we need to do a lot more with UAVs. You have to add a lot of features every day. It is also possible to improve our UAVs using machine learning and artificial intelligence technology.

This project is based on surveillance. Here a UAV has been made suitable for surveillance of a specific area. We launched this project to monitor from the sky and use UAV surveillance for human detection. With a camera mounted on a UAV, we can capture video and use that video for surveillance.

This is how we decided to finish our project. However, this is not the end because I am very keen to work on this project, so I am willing to do some more work on this project if I get any next opportunity in the future.

#### **6.2 Future work**

Following is what I want to do in the future with this project:

- Increasing UAV coverage diameter.
- Enable 24-hour surveillance even at night.
- Camera video footage compression without losing video quality.
- UAVs flying in residential areas by generating less noise.
- Maximum work with minimum power loss.
- Developing UAV controlling software for the ground station.



## References

1. Aber, J.S., 2004. Lighter-than-air platforms for small-format aerial photography. *Transactions of the Kansas Academy of Science*, 107(1), pp.39-44.
2. SPEARMAN, M., 1978, January. Historical development of world wide guided missiles. In 16th Aerospace Sciences Meeting (p. 210).
3. de Meyer, M., 2005. Houthulst and the A19-Project. Inventory of World War I Heritage based on Wartime Aerial Photography and Trench maps. In *Aerial Photography and Archaeology 2003: A Century of Information; Papers Presented During the Conference Held at the Ghent University, December 10th-12th, 2003* (Vol. 4, p. 87). Academia Press.
4. HAKA, A.T., Scientific Foundations and Stages in the Development of Thermography Thermography emerged from a number of scientific findings and established itself as a technology during the twentieth century. Its foundations can be traced to the Enlightenment.
5. Awange, J. and Kiema, J., 2019. Unmanned aircraft vehicles. In *Environmental Geoinformatics* (pp. 265-289). Springer, Cham.
6. Blom, J.D., 2010. Unmanned aerial systems: A historical perspective (Vol. 45). Kansas: Combat Studies Institute Press.
7. Zaloga, S.J., 2011. Unmanned aerial vehicles: robotic air warfare 1917–2007. Bloomsbury Publishing.
8. Wall, T. and Monahan, T., 2011. Surveillance and violence from afar: The politics of drones and liminal security-scapes. *Theoretical criminology*, 15(3), pp.239-254.
9. Carr, E.B., 2013. Unmanned aerial vehicles: Examining the safety, security, privacy and regulatory issues of integration into US airspace. National Centre for Policy Analysis (NCPA). Retrieved on September, 23, p.2014.
10. Kim, S.J., Jeong, Y., Park, S., Ryu, K. and Oh, G., 2018. A survey of drone use for entertainment and AVR (augmented and virtual reality). In *Augmented Reality and Virtual Reality* (pp. 339-352). Springer, Cham.
11. Chen, S., F Laefer, D. and Mangina, E., 2016. State of technology review of civilian UAVs. *Recent Patents on Engineering*, 10(3), pp.160-174.

12. Nakazawa, Anton, and Bai Xiang Jin. "Quadcopter Video Surveillance UAV." University of Victoria (2013).
13. Berrahal, Sarra, et al. "Border surveillance monitoring using quadcopter UAV-aided wireless sensor networks." *Journal of Communications Software and Systems* 12.1 (2016): 67-82.
14. Cox, T.H., Nagy, C.J., Skoog, M.A., Somers, I.A. and Warner, R., Civil UAV Capability Assessment.
15. Szczepański, C., 2015. UAVs and their avionic systems: development trends and their influence on Polish research and market. *Aviation*, 19(1), pp.49-57.
16. Fahlstrom, P. and Gleason, T., 2012. Introduction to UAV systems. John Wiley & Sons.
17. Bieda, R., Jaskot, K., Jędrasiak, K. and Nawrat, A., 2013. Recognition and location of objects in the visual field of a UAV vision system. In *Vision Based Systems for UAV Applications* (pp. 27-45). Springer, Heidelberg.
18. De Bruin, A. and Booysen, T., 2015. Drone-based traffic flow estimation and tracking using computer vision: transportation engineering. *Civil Engineering= Siviele Ingenieurswese*, 2015(8), pp.48-50.
19. Cazzato, D., Cimorelli, C., Sanchez-Lopez, J.L., Voos, H. and Leo, M., 2020. A survey of computer vision methods for 2d object detection from unmanned aerial vehicles. *Journal of Imaging*, 6(8), p.78.
20. Gleason, J., Nefian, A.V., Bouysounousse, X., Fong, T. and Bebis, G., 2011, May. Vehicle detection from aerial imagery. In *2011 IEEE International Conference on Robotics and Automation* (pp. 2065-2070). IEEE.
21. Xia, G.S., Bai, X., Ding, J., Zhu, Z., Belongie, S., Luo, J., Datcu, M., Pelillo, M. and Zhang, L., 2018. DOTA: A large-scale dataset for object detection in aerial images. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3974-3983).
22. Liew, C.F. and Yairi, T., 2020. Companion Unmanned Aerial Vehicles: A Survey. *arXiv preprint arXiv:2001.04637*.
23. Skorobogatov, G., Barrado, C. and Salamí, E., 2020. Multiple UAV systems: A survey. *Unmanned Systems*, 8(02), pp.149-169.

24. Carrio, A., Sampedro, C., Rodriguez-Ramos, A. and Campoy, P., 2017. A review of deep learning methods and applications for unmanned aerial vehicles. *Journal of Sensors*, 2017.
25. Karantanellis, E., Marinos, V., Vassilakis, E. and Christaras, B., 2020. Object-based analysis using unmanned aerial vehicles (UAVs) for site-specific landslide assessment. *Remote Sensing*, 12(11), p.1711.
26. "Introducing Copter – Copter documentation," Mar, 22, 2016. [Online]. Available at: <https://ardupilot.org/copter/docs/introduction.html>. [Accessed: Sep. 15, 2020].
27. S. Misra, "Introduction to internet of things–Courses," Sep, 14, 2020. [Online]. Available at: [https://onlinecourses.nptel.ac.in/noc20\\_cs66/preview](https://onlinecourses.nptel.ac.in/noc20_cs66/preview). [Accessed: Oct. 22, 2020].
28. S. Misra, "NOC | Wireless Ad Hoc and Sensor Networks," Nov. 20, 2017. [Online]. Available at: <https://nptel.ac.in/noc/courses/noc18/SEM1/noc18-cs09/>. [Accessed: Nov. 20, 2020].
29. S. Misra, "Introduction to Industry 4.0 and Industrial Internet of Things–Courses," Jan. 18, 2021. [Online]. Available at: [https://onlinecourses.nptel.ac.in/noc21\\_cs20/preview](https://onlinecourses.nptel.ac.in/noc21_cs20/preview). [Accessed: Jan. 25, 2021].
30. "Connect ESCs and Motors – Copter documentation" Mar, 22, 2016. [Online image]. Available at: <https://ardupilot.org/copter/docs/connect-escs-and-motors.html>. [Accessed: Sep. 15, 2020].
31. "Flying 101 · PX4 v1.8.2 User Guide" Mar, 22, 2021. [Online image]. Available at: [https://docs.px4.io/v1.8.2/en/flying/basic\\_flying.html](https://docs.px4.io/v1.8.2/en/flying/basic_flying.html). [Accessed: Jun. 15, 2021]
32. "Connect ESCs and Motors – Copter documentation" Mar, 22, 2016. [Online image]. Available: <https://ardupilot.org/copter/docs/connect-escs-and-motors.html>. [Accessed: Sep. 15, 2020].

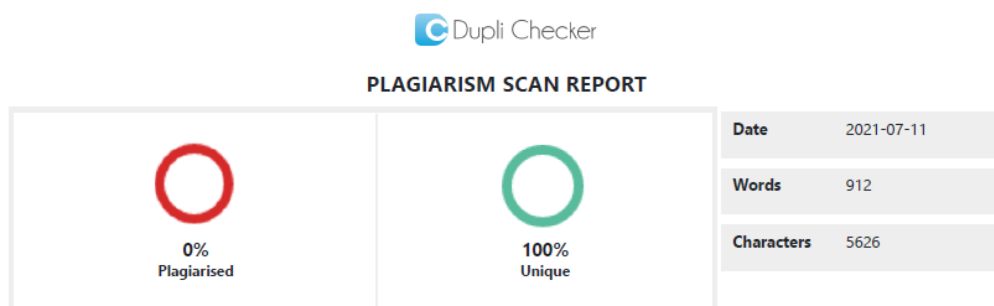
## Plagiarism statement

I check plagiarism from Introduction to conclusion. I used free software named dupliChecker. This software can check plagiarism 1000 words at a time.

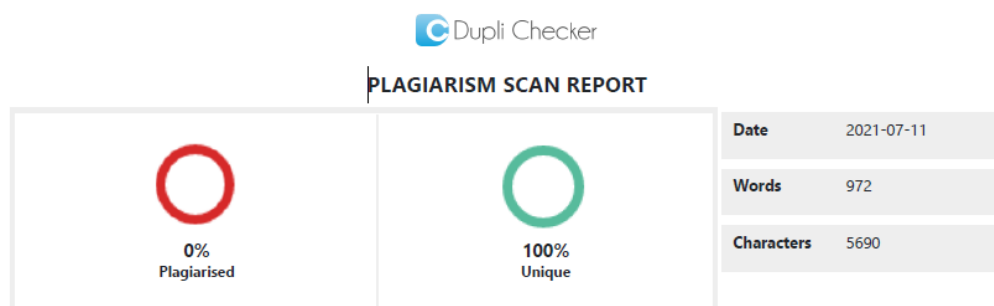
My report has a total of 8387 words. So I check it 9 times (912, 973, 931, 932, 965, 986, 966, 924, and 798). And results are coming (0, 0, 0, 0, 0, 0, 0, 4, and 4). So the final plagiarism number will be  $(4*0.924 + 4*0.798)/8.387 = 0.821$ .

Plagiarism report:

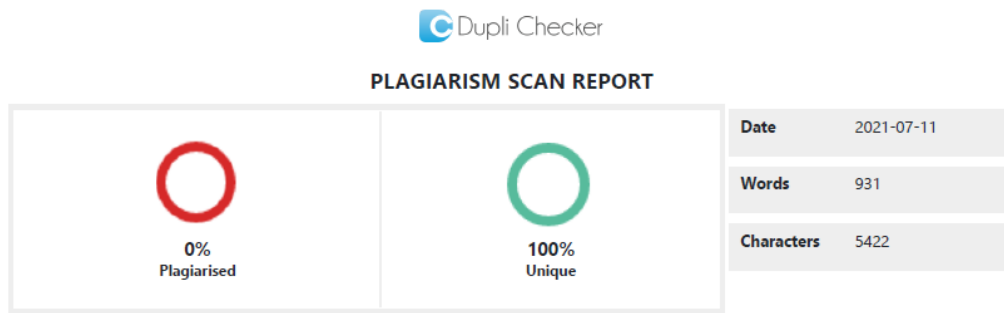
Plagiarism results 1<sup>st</sup> time:



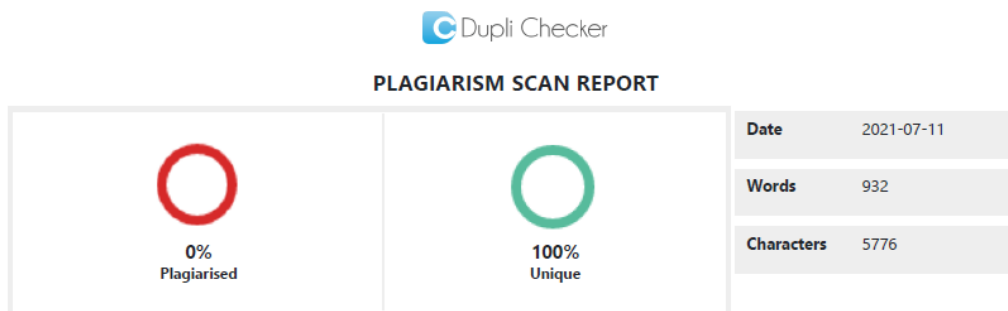
Plagiarism results 2<sup>nd</sup> time:



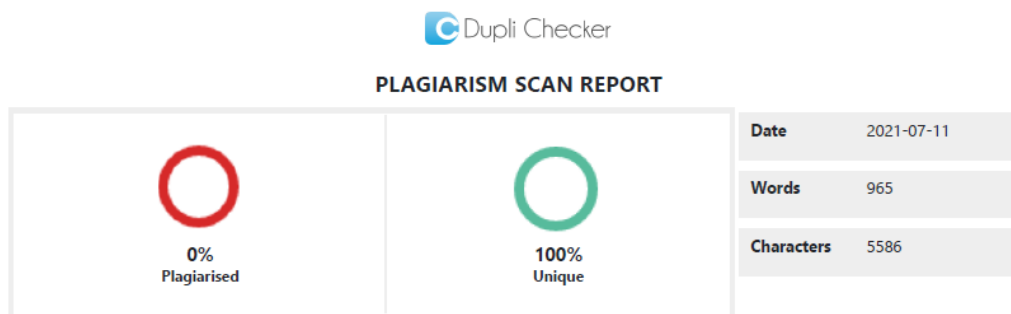
Plagiarism results 3<sup>rd</sup> time:



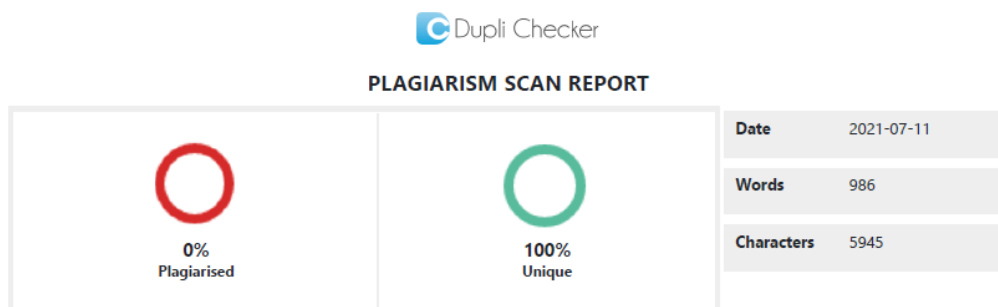
Plagiarism results 4<sup>th</sup> time:



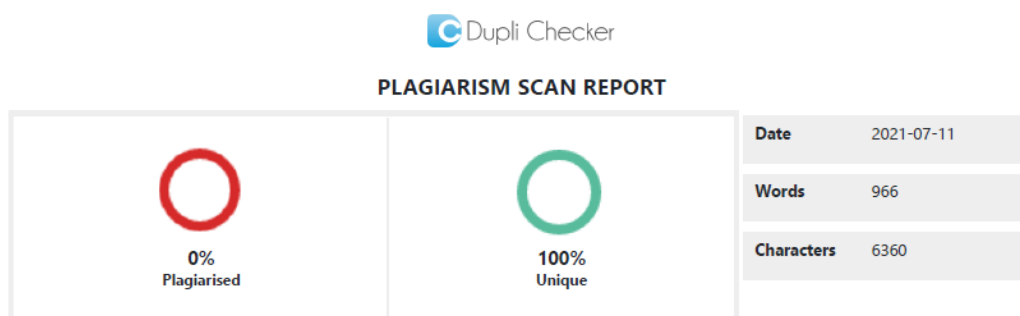
Plagiarism results 5<sup>th</sup> time:



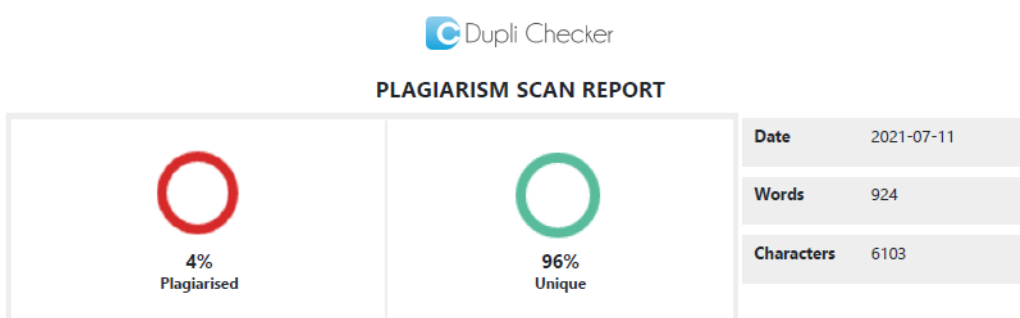
Plagiarism results 6<sup>th</sup> time:



Plagiarism results 7<sup>th</sup> time:



Plagiarism results 8<sup>th</sup> time:



Plagiarism results 9<sup>th</sup> time:

