Faculty of Electrical and Electronic Engineering Technology

DEVELOPMENT OF DRONE DETECTION SYSTEM USING ARDUINO FOR ENHANCE PRIVACY PURPOSE

MUHAMMAD ELWAN BIN MOHD ROSLAN

Bachelor of Electronics Engineering Technology (Telecommunications) with Honours

2023

DEVELOPMENT OF DRONE DETECTION SYSTEM USING ARDUINO FOR ENHANCE PRIVACY PURPOSE

MUHAMMAD ELWAN BIN MOHD ROSLAN

A project report submitted
in partial fulfillment of the requirements for the degree of
Bachelor of Electronics Engineering Technology (Telecommunications) with Honours

Faculty of Electrical and Electronic Engineering Technology

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2023

DECLARATION

I declare that this project report entitled “Development of Drone Detection System Using Arduino for Enhance Privacy Purpose” is the result of my own research except as cited in the references. The project report has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature : 

Student Name : Muhammad Elwan Bin Mohd Roslan

Date : 27 JANUARY 2023

APPROVAL

I approve that this Bachelor Degree Project 1 (PSM1) report entitled “Project Title” is sufficient for submission.

Signature : 

Supervisor Name : DR. A K M ZAKIR HOSSAIN

Date : 27.01.2022

APPROVAL

I hereby declare that I have checked this project report and in my opinion, this project report is adequate in terms of scope and quality for the award of the degree of Bachelor of Electronics Engineering Technology (Telecommunications) with Honours.

Signature :

Supervisor Name :

Date :

Signature :

Co-Supervisor :

Name (if any)

Date :

DEDICATION

To my beloved mother, Salbiah Binti Nasir

and

To my dearest siblings Irfan, Aiman, Iman and Ain.

ABSTRACT

Today, we continue to see significant increases in the use of robots (airborne or non-airborne) in various aspects of our lives. Drones are common examples of unmanned airborne robots. Drones are not only used for military purposes; they are also used in the civilian sector. Some of these applications include logistical operations, reconnaissance, search-and-rescue, disaster assessment, and others. The Federal Aviation Administration (FAA) reports that over 1.5 million drones are registered in the United States. Twenty-eight percent of these registered drones are used for commercial purposes, while the rest are purchased for recreational purposes. Because there are numerous benign applications for drones and no strict regulations on who can buy and operate a drone, there is also an increase in safety and security concerns. The use of a drone to infiltrate a secured or restricted area is a common security concern. Extremist or terrorist groups can also use drones to deliver explosive payloads or chemicals to a specific location, putting public safety at risk. Radar has been widely used in the detection of drones and small aircraft. However, there are some restrictions. Radar, for example, cannot tell the difference between birds and drones. Several other detection methods, based on sensing mechanisms such as sound, video, thermal, and radio frequency, have been used in DDI systems (RF). There are various tradeoffs when using any of these approaches, and these tradeoffs influence system performance. Weather conditions can have a significant impact on the thermal approach. When there is a lot of noise around, the effectiveness and efficiency of a sound detection system suffers. Similarly, low light visibility and coverage are disadvantages for using video detection mechanisms.

ABSTRAK

Hari ini, kita terus melihat peningkatan ketara dalam penggunaan robot (bawaan udara atau bukan udara) dalam pelbagai aspek kehidupan kita. Dron adalah contoh biasa robot bawaan udara tanpa pemandu. Drone bukan sahaja digunakan untuk tujuan ketenteraan; ia juga digunakan dalam sektor awam. Beberapa aplikasi ini termasuk operasi logistik, peninjauan, mencari dan menyelamat, penilaian bencana dan lain-lain. Pentadbiran Penerbangan Persekutuan (FAA) melaporkan bahawa lebih 1.5 juta dron didaftarkan di Amerika Syarikat. Dua puluh lapan peratus daripada dron berdaftar ini digunakan untuk tujuan komersial, manakala selebihnya dibeli untuk tujuan rekreasi. Oleh kerana terdapat banyak aplikasi jinak untuk dron dan tiada peraturan ketat tentang siapa yang boleh membeli dan mengendalikan dron, terdapat juga peningkatan dalam kebimbangan keselamatan dan keselamatan. Penggunaan dron untuk menyusup ke kawasan yang selamat atau terhad adalah kebimbangan keselamatan yang biasa. Kumpulan pelampau atau pengganas juga boleh menggunakan dron untuk menghantar muatan bahan letupan atau bahan kimia ke lokasi tertentu, meletakkan keselamatan awam pada risiko. Radar telah digunakan secara meluas dalam pengesanan dron dan pesawat kecil. Walau bagaimanapun, terdapat beberapa sekatan. Radar, sebagai contoh, tidak dapat membezakan antara burung dan dron. Beberapa kaedah pengesanan lain, berdasarkan mekanisme penderiaan seperti bunyi, video, terma dan frekuensi radio, telah digunakan dalam sistem DDI (RF). Terdapat pelbagai pertukaran apabila menggunakan mana-mana pendekatan ini, dan pertukaran ini mempengaruhi prestasi sistem. Keadaan cuaca boleh memberi kesan yang ketara ke atas pendekatan terma. Apabila terdapat banyak bunyi di sekeliling, keberkesanan dan kecekapan sistem pengesanan bunyi terjejas. Begitu juga, keterlihatan dan liputan cahaya rendah adalah kelemahan untuk menggunakan mekanisme pengesanan video.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my supervisor, Dr. AKM Zakir Hossain for his precious guidance, words of wisdom, and patience throughout this project.

I am also indebted to Universiti Teknikal Malaysia Melaka (UTeM) for the financial support through the Faculty of Electrical and Electronic Engineering Technology which enables me to accomplish the project. Not forgetting my fellow colleagues, for their willingness of sharing their thoughts and ideas regarding the project.

My highest appreciation goes to my parents and family members for their love and prayer during the period of my study. An honorable mention also goes to my mother Salbiah Binti Nasir for all her motivation and understanding.

Finally, I would like to thank all the staff at the Faculty of Electrical and Electronic Engineering Technology, fellow colleagues and classmates, the Faculty members, as well as other individuals who are not listed here for being cooperative and helpful.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

m	-	meter
s	-	seconds

LIST OF ABBREVIATIONS

UAV	-	unmanned aerial vehicle
RF	-	radio frequency
GPS	-	global positioning system
DOA	-	direction of arrival
RCS	-	radar cross-section
FMCW	-	frequency-modulated continuous-wave
MIMO	-	multiple-input multiple-output
MDR	-	motion detection radar
SDR	-	Software-defined radio
FPV	-	First-Person View
OUI	-	Organizationally Unique Identifier
3D	-	Three dimentional
COTS	-	commercially available off-the-shelf
LOS	-	line of sight
FOV	-	field of view
NLOS	-	non-line of sight
RSS	-	received signal strength
LBP	-	local binary pattern
DPM	-	deformable parts model
GFD	-	generic Fourier descriptor
SIFT	-	scale-invariant feature transform
HOG	-	histogram of oriented gradients
SSD	-	single shot detector
YOLO	-	you only look once
KCF	-	kernelized correlation filter
CNN	-	convolutional neural network
CRNN	-	Convolutional Recurrent Neural Network
RNN	-	Recurrent Neural Network
GAN	-	Generative Adversarial Network
DSP	-	digital signal processing
SVM	-	Support Vector Machine
PIL	-	Plotted Image Learning
K-NN	-	K-Nearest Neighbour
SfM	-	structure from the motion
BA	-	bundle adjustment
GMM	-	Gaussian Mixture Models
KF	-	Kalman Filter
GSM	-	Global System for Mobiles

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

Over the past few years, we have seen drones have become very popular among all ages of people. We can see them being sold in all kinds of places whether it is a department stores, online, or even at the convenience store. Nowadays, it came at all sizes and prices. You can get a drone that has the same size as an apple and with a price that everyone can afford. The drone also can be referred to as an unmanned aerial vehicle (UAV). A drone is a flying robot that may be commanded remotely or fly autonomously by using a radio frequency (RF) controller or software-controlled flight plans that connect with onboard sensors and a global positioning system (GPS)[1]. A drone most commonly refers to a multirotor, which has three or more propellers and may hover or fly in any direction. A quadcopter is the most common type, with four propellers[2]. Drones have so many cool features that make them so popular. Many drones include cameras that allow you to observe the world through the eyes of the drone. You may also make videos with the camera to share your flying experiences with others. A flight controller is embedded into every drone to keep it stable. If it tips over due to a gust of wind, the flight controller will immediately alter the propeller speeds to level it out[3]. This makes learning to fly easier for novices. A full-sized airplane or helicopter was formerly necessary for aerial flight. Drones can now perform many of the same tasks at a fraction of the cost. Drones require a power source, such as a battery or fuel, to fly. Rotors, propellers, and a frame are all included. To save weight and improve manoeuvrability, drone frames are often composed of lightweight composite materials[4]. Drones need a controller, which enables the user

to use remote controls to launch, navigate, and land the drone. Controllers communicate with the drone via radio waves such as Wi-Fi.

1.2 Problem Statement

Drones that master the art of data collection efficiently are now part of the current inspection standard[2]. UAV technology has prompted various businesses to adopt new methods. However, all of the benefits come with a few drawbacks. Even while Drones strive toward perfection, they aren't perfect. When a new technology is introduced, some new problems will come with it. The most common problem that everyone has with a drone is a breach of privacy[5]. Drones give users the ability to position a flying camera practically anywhere they choose, including on other people or property. Even though there are rules governing where drones can fly, some users disregard them. The drone is easily manipulated and can invade the privacy of a group or individual. While many people want to use drones to keep themselves safe, doing so could violate a variety of individual liberties for the sake of public safety[2]. Drones are frequently used by criminals to target their victims and keep track of them[6]. The loud propeller noises are no longer a problem because they are unnoticed, allowing attackers to breach someone's privacy. Many drones equipped with thermal and night sensors detect vital signs and efficiently target people the spy is now interested in[7]. Because UAVs can collect reliable data, they can track routine habits and detect suspicious activity without requiring authorization.

Unmanned Aircraft Systems (UAS) are already widely used, yet because it is a new technology in the market, the legislation is continually evolving[8]. Specific rules for small drones apply to commercial and recreational users as well but are still ambiguous in various ways. Drone movement regulation and property protection from airborne trespassing are still

in the works, therefore UAV technology operates in a legal grey area. There are several conflicts between federal standards and any state or local laws governing aerial property rights, which can lead to drone pilots breaking statutes they are unaware of[9]. Drones' rapid acceptance over the last decade has aroused privacy, security, and safety concerns. Drones are used by voyageurs and paparazzi to photograph people in their homes and other formerly private locales. Drones are often employed in dangerous regions like cities and near airports[3]. Increased commercial and personal drone use has raised the risk of mid-air crashes and drone control loss. Concerns about drones flying too close to commercial planes have led to calls for regulation. UAV rules have been created in many nations. Laws are constantly changing as drone use becomes more widespread[8]. Drone pilots, both personal and commercial, must familiarise themselves with the rules of the country and region in which they are flying the devices. Drones pose a threat to large planes and helicopters, and they can obstruct firefighting and rescue efforts. Wildfires have grown deeper in certain cases because a drone hovered nearby, preventing firefighting planes and helicopters from reaching the blaze. So it has become part of airspace issues.

1.3 Project Objective

The primary goal of this research is to present a functional and methodical progress approach. The following are the specific objectives:

a) To design and simulate an efficient system for detecting drones.

b) To fabricate the proposed prototype for detecting drones.

c) To benchmark and compare the result with the current trend and industry.

1.4 Scope of Project

For this project, the scope is to design a system that can be used in detecting the unknown drone and prevent it from trespassing the private property. The way that the drone can be detected is by using the radio frequency (RF) signal of the drone. A camera also will be used to feed the live footage of the intruder. Both the inputs that are being used will be controlled by Arduino which can also be the brain of this system. The range of frequency of the signal that can be detected is around 2.4GHz and the distance is around 100 meters.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Small unmanned aerial vehicles (UAV) or drones have become increasingly popular nowadays. People are using drones for recreational use or as a hobby[10]. There is also a company that replaces their staff with drones such as food delivery. Drones are becoming a necessity in our daily lives. However, as technology improves, it becomes easier to spy on people in otherwise private places, such as a person's home, posing a greater risk to privacy[10]. Anybody nowadays can buy a drone and fly it over someone's fences without having difficulty climbing over the fence to get the inside view. The best solution for this situation is to equip ourselves with something that can detect and counter the personal drones that invade our private property[10]. Unfortunately, the existing product of drone detection systems are inaccessible to the general public because they require specialized equipment and costly deployment procedures.

The creation of an automatic drone detection system has become necessary in every home to have[6]. But the market for drone detection only targeting the industrial uses or bigger, that can cover a large area, so the product they offer are not affordable for civilian[10]. The manufacturer of this drone detector comes in many types of sensors implements inside the system depending on the needs of the users, so that it can become more flexible based on the situation of the area they are implementing it on[8]. But for the use of household properties they doesn't need so many types of sensors just one or two and also it only need to cover a small space enough to cover their house and yard.

In this report, I am going to present a type of drone detection that doesn't require a complicated setup to configure and only cost less than the industrial kind of product. The hardware and equipment we will be using are also off the shelves equipment and component. This system will only detect the drone that invade the privacy of home owner but also include the way to counter the intruder drone. Drones may carry out operations using a variety of technologies[6]. As a result, drone detection systems must be comprehensive enough to detect any type of drone. Current technologies, on the other hand, usually fail to recognise the many types of drones with harmful missions[8]. But in this system, it will detect all types of drones but it will leave the decision whether the drone are malicious or safe on the user hands.

2.2 Introduction to Drone detection System

The purpose of distribution feeder (cable or line) is to provide path for energy flow from GSS all the way to the distribution customer. Traditional distribution feeders (without DER) are usually operated in radial configurations - the energy flows uni-directionally from the GSS to the load. The feeders are typically categorized by its: (i) voltage level, (ii) conductor material, (iii) conductor size (cross sectional area), (iv) insulation type and (v) no of phases. These feeders scattered all over different supply zones. Hence, they are extensive and large in numbers.

2.3 Methods of Drone Detection

There are many ways to detect and track drone. There are five methods that are mostly used by the drone detection manufacturer which is Video-based detection, sound- based detection, radar-based detection, radio-based detection and Wi-fi based detection[14].

These five methods require their own types of sensors in order for the system to operate and detecting the drone.

2.3.1 Video-Based Detection

Video-based detection relies on the graphical and electrical camera sensors to detect and study the movable object and identify the objects in the monitored area[12]. The sensor can cover up to 100 meter area of monitoring. A quiet environment can be achieved by using this kind of detection method. The camera sensors that are being used in this system must able to characterized object based on their colour, contour lines, shapes, and edges in order for the system to differentiate the drone from the other things. This method require the complex monitoring algorithm that can evaluate various elements over consecutive frames. The algorithm are required so that it can identify and differentiate all the moving object that comes on the camera and it can automatically detect the drone and not bird or other unnecessary object. This kind of camera sensor usually very sensitive on the lighting condition and require to be place at the place that receive a suitable amount of lighting in order for the sensor to operate successfully.

2.3.2 Sound-Based Detection

An acoustic camera is used as sensor in sound-based detection system based on its ability to detect the sound and voice through the air[12]. There are many research and study that focussing on identifying and detecting the sound that drone makes so that they can make an algorithm that works against drones. The other way is to use direction of arrival (DOA) rating in order to track the drone. The famous method is to use the emission sound of drone and detecting it using the acoustic camera to distinguishing the difference from other things[7].

2.3.3 Radar-Based Detection

A broad radar cross-section (RCS) is used in a conventional radar-based detection system that observe the small aircraft and flying objects[2]. The theory behind radar-based detection uses electromagnetic principle of backscattering to detect drone[12]. However, as proven by the existing

literature, the majority of current drones are mechanical quadcopters with a low RCS. The conventional radar detection depends strongly on the build materials and it has trouble detecting the dielectric materials that have characteristic similar to the air[12]. That where the electronic principle of backscattering comes in to cover the inconvenience from the previous types of radar[4]. Additionally, drones' scope, kind, range, and radial velocity were measured using frequency-modulated continuous-wave (FMCW) radar. A multiple-input multiple-output (MIMO) were introduced to create virtual components and can be used to determine the position of the drone whether it stays within the range of angular cells[12]. This method is more mobile and motion detection radar (MDR) usually comes with it onboard. Some researchers suggested a method that can automatically identify and detect three types of drones in a laboratory environment. Lastly, a radar that relies on the vision of computer to differentiate the drone from a bird are also used as a way to detect a drone.

2.3.4 Radio Frequency Detection

The fact that all drones fly by using the RF signal to connect with the controller makes this method can be used to detect drones. Drone networking protocols are usually carried by the same radios that are used for Wi-Fi communications, particularly in the 2.4 and 5 GHz bands[12]. Drones with cameras often transmit a video stream to their control system over the same wireless channel. The RF sensor has a range of more than 400 meters

area of detection. It is one of the longest distances operational among all types of sensors but it relies deeply on the transmission energy and response of the detector. Software-defined radio (SDR) was introduced and by combining it with wireless acoustic sensors it can estimate the direction and track the location of the drone[12]. A new portable universal radio minor that uses SDR was introduced recently that can monitor the drone location and simulate the drone the different situations.

2.3.5 Wi-Fi-Based Detection

Many operational drones are built and developed to be operated through Wi-Fi, allowing experts to monitor the drone using their own smart devices. These systems frequently contain a First-Person View (FPV) video capability, which allows the integrated camera to relay the stream straight to the intelligent device monitor[12]. Few proposals in prior research have looked into exploiting the Wi-Fi signal to detect the presence of illegal drones. The main idea is to use a Wi-Fi channel network packet capture to capture drone power and video transfer packet streams. This approach is used in drones that are designed to detect specific sorts of adjacent machinery. However, these solutions are usually reliant on prior knowledge of the remotely controlled aircraft, such as information about the Organizationally Unique Identifier (OUI) vendor that is used to classify the sender/receiver of unique packets[12]. There is also some proposal about a model for classifying drone positions in the monitored area based on the statistical fingerprints of Wi-Fi traffic[12]. A conducted research on drone identification using Wi-Fi sniffing by statistically analysing Wi-Fi traffic for drone fingerprints has been introduced. A power-saving system was installed so that the system can detect and remove video streams from Wi-Fi-connected drones, which might be a useful mitigation tool for privacy-conscious systems.

Methods	Advantages	Disadvantages
Video	• Can see info on the UAV and any cargo it carries immediately. • Takes photos and videos that could be used as proof in the event of a prosecution.	• Impractical for detection on its own • False reports are frequent. • Hard to use in low-light, fog, and or poor-visibility situations
Sound	• very accurate in drone detection	• very expensive equipment • have a very complicated algorithm for drone detection

Radar	• long range • can handle multiple targets simultaneously • Visual conditions do not limit their use • can catch autonomous drones	• required transmission licence • Frequency checks are required to prevent interference
Radio Frequency	• Capable of simultaneously detecting, identifying, and locating several drones and their handlers	• limited range • In places where there is a mix of radio frequencies, it is less efficient.

Table 2.1 Comparison of Methods of Detection

2.4 Drone Monitoring Equipment

Drone monitoring equipment is often divided into four categories. This equipment is basically the sensors that can be installed inside the detection method.

2.4.1 Radio Frequency (RF) Analyzers

One type of electromagnetic energy is radio frequency (RF). Undesired wireless signals usually cause interference as a result of one or more variables influencing communication devices' receiving systems, resulting in a deterioration in the required signal's requirements or the information loss about the signal that is present if the signal disappears unexpectedly[12]. Any of the following phenomena or acts can cause radio frequency interference.

2.4.2 Acoustic Sensors (Microphones)

Ultrasound waves that are higher in frequency than human-made sounds but lower than 20,000 Hz are captured by acoustic sensors (microphones). Acoustic sensors are used in a variety of sectors including physics, chemistry, technology, and medicine[12].

2.4.3 Optical Sensors (Cameras)

Digital sensors, such as optical sensors (cameras), are a sort of digital sensors. Optical sensors detect objects using light. Optical sensors detect the presence or absence of light using a light detector (receiver) and a light source (transmitter)[12].

2.4.4 Radar

Radar is an electromagnetic sensor that is used to monitor, locate, and recognise things at long distances. The size and shape of these objects may also be determined via radar. The performance of radar equipment is dependent on the transmission of electromagnetic energy towards certain targets and the monitoring of the echoes that return. Because radio waves can travel longer distances and accomplish work even when the signal is weak, radar devices use them instead of sound waves[10].

Figure 2.1 Drone Monitoring Equipment

2.5 Drone Countermeasures Equipment

There are many kinds of drone countermeasures equipment in the market nowadays. There are some that is passive which only tracking and monitoring and there are also which is active that can send the signal out and analysing the response and perform a variety of tasks, including detection, classification and identification, locating and tracking and alerting[6]. But the drone countermeasures that we are about to show are all about countering and fighting back against the unwanted drone. There are some technologies that we can obtain anywhere and there is also some equipment that only military personnel has access to.

2.5.1 RF Jammers

Using RF frequency and GPS signal, the technology shoots down dangerous drones[12]. It employs dual remote control, navigation signals, and control, preventing the malicious drone from entering the defence zone, which is located outside the emergency landing zone.

2.5.2 GPS Spoofers

GPS spoofers are technologies that prohibit drones from getting signals from control towers by jamming their GPS signals. They are sophisticated systems in which the jamming is accomplished via GPS systems on the system layer, which transmit incorrect data[7].

2.5.3 High Power Microwave (HPM) Device

One of the new high-voltage applications, such as diodes, is this high-power device[12]. Large power sources demand the use of extremely high-quality equipment. High-power microwave devices consistently perform their task when utilised for drone detection, allowing attacking drones to be efficiently halted utilising non-kinetic neutralisation.

2.5.4 Nets & Guns

The net gun is a non-lethal weapon that fires projectiles from a network that obstructs and foils target movement[2]. It's used to detect drones, to sway birds away from aircraft, and even save wild birds. The net gun technology identifies drones and captures them physically, allowing for better forensics and prosecution. They have a high mission success accuracy, a minimal danger of collateral damage, and a large range for deployed

nets[12].

2.5.5 High-Energy Laser

UAV and drones take off from a safe distance are being developed by military forces all over the world. The present armament systems, on the other hand, are huge and heavy and cannot be mounted on motorised vehicles or combat planes[12]. This encouraged big defence industry firms to investigate fibre laser weapons as a possible alternative.

2.5.6 Birds of Prey

Eagles and falcons are birds of prey recognised for their excellent vision and great speed, as well as their fluid wings that allow them to fly quickly[3]. They have big bodies, sharp beaks, and powerful claws that can crush prey.

Figure 2.2 Drone Countermeasures Equipment

2.6 Camera Tracking

2.6.1 Single-View Tracking

Most trackers have trouble monitoring small things like flying birds, and even with infrared cameras, tracking many small targets is challenging. A recent study has attempted into creating realistic sense-and-avoid programs for distance flying objects employing passive cameras that can manage low-resolution pictures and movable cameras. These approaches, however, are unable to retrieve precise 3D UAV trajectories[13].

2.6.2 Multi-View Tracking

Tracking objects in a 3D scene is substantially more reliable with synchronised passive multi-camera setups[13]. They have traditionally been offered for comprehending human activities, assessing sports scenes, and surveillance of indoor, outdoor, and traffic situations. These approaches are rarely used to monitor small objects in a large 3D volume when the above optimization methods are unfeasible.

2.6.3 Multi-view 3D reconstruction

While most existing solutions necessitate meticulous pre-calibration, several techniques allow cameras to be calibrated on the fly. A researcher presented a method for simple linear or conical object motion, which was later expanded to include curved and generic planar trajectories[13]. For joint tracking and camera calibration, more recent approaches have taken advantage of other geometric limitations. These approaches, however, necessitate precise feature monitoring and matching among views and are ineffective for small objects.

2.6.4 Constrained Bundle Adjustment

In classic bundle adjustment, camera settings are modified alongside the 3D structure, which is typically displayed as a 3D point cloud. Planarity, 3D symmetry, bound limits, and prior knowledge of 3D shapes are examples of soft geometric constraints on 3D structures, which are sometimes used to incorporate regularisation into bundle adjustment[13].

2.7 Previous Project Research

2.7.1 Detecting Drone Attacks Using Wi-fi

Here's how to make a drone detector out of commercially available off-the-shelf (COTS) gear that targets homeowners as consumers[10]. Network fingerprints from the drone's Wi-Fi connectivity can be used in some approaches. They must rely on an unencrypted connection between the drone and the controller for this strategy to work, which is not the case with newer models. They based their model based on three-phase of attack which is approach, surveillance, and escape[10]. To carry out the attack, the attacker launches the drone from outside at launch distance and then flies it to surveillance distance. The window has a motion detector fitted. The detector's line of sight (LOS) is limited by the window's field of view (FOV)[10]. In the window, there is a

little Wi-Fi receiver. The receiver is set up to catch traffic in monitor mode, switching channels on a regular basis to cover the full band. The detection part consists of three phases.

Pre-processing provides the stream of packets received for processing in the first phase. Individual transmitters are next tested statistically to see if they are traveling and working in free space. Unless the statistical checks pass, the attack analysis phase represents the current attack phase and whether the drone has approached within close range. They are testing four kinds of approach patterns to the window which are direct, zig-zag, back and forth, and non-line of sight (NLOS)[10]. Raspberry Pi Model A15 was used as receivers and a Wi-Pi USB Wi-Fi adaptor was added to received packets. The drone was equipped with a Raspberry Pi, which interacted with a computer using the iPerf 217 benchmark program at a rate that was comparable to live footage (4 MB/s, 400 packets/s)[10]. And a 3D Robotics X8+ drone was used. Background communication from the other Wi-Fi transmitters in the neighbourhood was used to calibrate the noise limit[10]. Finally, using the data from the studies, they began evaluating how effectively the metrics can recognize a drone. They discovered that the detection of drones happens quicker than they expected. A drone's initial movement takes off, but communication begins far earlier. This earlier communication aids our detection since the received messages generate a beginning measurement in each window, such that in many situations, the takes off alone is sufficient to raise the standard deviation above the detection threshold[10]. As a result of the early identification, the user has plenty of time to be alerted to a privacy violation and take precautionary measures. It also gives you the option of just informing a user of nearby drone activity that isn't, or hasn't yet escalated into, a privacy violation effort.

From their observation, as it becomes possible to see the drone firsthand while flying, the operator's reliance on the drone's FPV live feed will grow. When executing a privacy assault on one's own, the operator requires fast feedback to guarantee that the drone has a decent view of the interior of the house, and it has proven easy to record detailed film of a window frame by

accident[10]. It would have been beneficial to investigate the influence of cross-traffic on detection in order to truly comprehend the system's predicted effectiveness in an urban setting. The amount of successfully recorded received signal strength (RSS) samples in a given period decreases as the cross-traffic packet rate increases, and we would anticipate this to slow down recognition speed and accuracy, albeit the exact behaviour is unknown[10]. Because this detection method relies solely on RSS measurements, the equipment employed in this study is just one of many options. RSS is supported by the vast majority of Wi-Fi hardware. This information is available thanks to data and a large number of drivers. In short, their system can identify drones flying nearby and can provide an alert when one approaches a window. Even for the greatest physical windows, detection occurs at such a distance that an attacker will not have had time to perform comprehensive surveillance before the alarm is raised. Only low-cost, readily available hardware was employed in their implementation. Furthermore, the system is based on measures found in the vast majority of Wi-Fi-capable equipment, allowing for a wide range of deployment options.

2.7.2 Video with a Static Background

Handcrafted feature-based approaches and deep learning-based algorithms are used to detect drones using visual data[14]. Handcrafted feature-based approaches use classic descriptors and are centred on traditional machine learning algorithms for example local binary pattern (LBP), deformable parts model (DPM), generic Fourier descriptor (GFD), scale-invariant feature transform (SIFT), and histogram of oriented gradients (HOG) that have low-level handcrafted characteristics (colour information, drops, edges, and blobs) and classical classifiers (support vector machine (SVM), AdaBoost)), the second group, on the other hand, uses two-stage learning to rely on learned traits (Mask R-CNN, Fast R-CNN, Faster R-CNN, and region-based convolutional neural network (R-CNN)) and single-stage (single shot detector (SSD), RetinaNet, and you only look once (YOLO)) deep object detectors[14]. GFD vision-based features were developed for this method to

characterise the binary patterns (like silhouettes) of drones and birds which are insensitive to rotation and translation alterations. Using the deep learning technology, a real-time drone detector are created. Because developing a successful detector necessitates a large number of training images, the scientists first created a semi-automatic dataset using a KCF (kernelized correlation filter) tracker rather than hand labelling[14]. The KCF tracker-based semi- automatic approach of labelling datasets sped up the process of preparing the training images. Deep learning-based detectors have produced the best results in drone detection. This is demonstrated by the fact that most studies on drone detection have depended on CNNs to solve the problem in part or entirely[14].

Drone-vs.-Bird Challenge is the challenge's main goal is to identify and differentiate drones from birds in brief films captured by a static camera from a great distance. A researcher tested the Faster R-CNN object detector with several CNN backbones to identify flying objects. Real photographs of drones and birds were removed from their surroundings and mixed with frames from coastal region films to produce a fake dataset[14]. The authors concentrated their research on detecting drones in real-time against a static background. A background subtraction method was used to create the motion detector. This module's outputs are all of the scene's moving objects. Drones are distinguished from other objects moving by placing all observed objects into a classifier[14]. The classifier is a CNN that was trained on the dataset of birds, drone, and background photos that we gathered.

2.7.3 Deep Learning Techniques

The goal of this study is to address the shortcomings of drone detection methods by developing an autonomous system that, in addition to recognizing drones, can also distinguish them depending on their acoustic signatures that use distinctive deep learning methods such as the Recurrent Neural Network (RNN), Convolutional Recurrent Neural Network (CRNN), and

Convolutional Neural Network (CNN), all without the need for human interference[5]. However, there are two issues. The first is a shortage of big acoustic drone datasets, that are required to adequately train deep learning algorithms. The second is that most drone databases only include a few different types of drones. As a result, failing to include all types and models of drones accessible degrades the detection process and exposes it to new drone types. To solve these challenges, we use the Generative Adversarial Network (GAN), a cutting-edge deep learning method for generating artificial data, to construct a huge artificial drone acoustic dataset with the goal of enhancing the presence of drone recognition[5].

Analyze the efficiency of the chosen deep learning algorithms in drone identification and detection using specified assessment methods such as F1 score, recall, accuracy, and F1 score, as well as the computational time necessary to develop and test the proposed models[5]. The first is to compare the reliability and effectiveness of merging an artificially induced dataset with an actual drone audio dataset in improving drone identification. The next step is to make an open-source drone audio collection comprising both recorded and generated drone audio available to academic researchers in order to address the lack of drone training samples for deep learning models.

Here also has conducted research and developed a new way of detecting the presence of drones in a region using digital signal processing (DSP). Similarly, the scientists developed a new drone detection technique by merging DSP with Machine Learning algorithms for example the Support Vector Machine (SVM) algorithm in their study[5]. The studies report on the performance of SVM in drone recognition, with high accuracy, but the investigation was limited to clear ambient noises. Furthermore, to fine-tune the model, SVM hand-crafted features must be extracted and optimised manually, which is an extra step in the categorization process. Deep learning models, on the other hand, have the potential to overcome these flaws and eliminate the additional steps necessary in traditional machine learning algorithms by automatically training the model from start to finish[5]. In this vein, proposed a method for detecting drones utilising DSP and two Machine

Learning techniques, the K-Nearest Neighbour (K-NN) and the Plotted Image Learning (PIL)[5]. While the algorithms exhibited their efficacy and identification ability, the KNN technique's total accuracy was surprisingly small. To minimise biases and overfit the noise, PIL requires a large number of pre-stored image datasets with consistently variable background noises; consequently, deploying such a system in the real world is difficult[5]. Two of the most significant obstacles to implementing an acoustic-based method for drone recognition have been identified. The first one is the impact of a noisy environment on the performance of an acoustic-based approach, and the other is the abundance of diverse drone acoustic data[5]. Deep learning algorithms are useful in audio applications such as speech recognition a recent study. However, nothing is known about the use of deep learning approaches in drone recognition based on the drone's auditory properties at this time. The absence of acoustic drone datasets limits the capacity to use deep learning techniques to build an effective solution.

2.7.4 UAV Trajectory Based On Flight Dynamics

This study offers a new approach for estimating a flying object's 6-of trajectory inside a 3D airspace under surveillance by numerous fixed ground cameras, such as a quadrotor UAV[13]. It uses a new structure from movement formulation to recreate a single moving point with known motion dynamics in 3D. Their significant benefit is a unique bundle adjustment approach that improves camera positions while also regularising the point direction using a motion dynamics-based prior or, more precisely, flight dynamics[13]. The fundamental control input delivered to the UAV's autopilot, which dictated its flight route, can also be deduced. Existing single-camera tracking and detection methods are typically inappropriate for drone surveillance due to their limited field of view and the challenge of properly calculating the length of objects far from the camera that occupies very low pixels in the image sequence[13]. These constraints can be overcome by using many overlapping cameras. Existing multi-camera tracking approaches, on the other hand,

are designed to follow people and vehicles for interior and outdoor surveillance activities, where the targets are frequently on the ground. Small drones, on the other hand, must be tracked inside a three- dimensional volume that is orders of magnitude larger. The 6-of motion trajectory of a quadrotor recorded by numerous fixed cameras is recovered using a new structure from the motion (SfM) formulation presented in this research[13].

The simulation manage to achieve contributions as three folds, researchers offer a new bundle adjustment (BA) approach that not only optimises camera poses and 3D trajectory, but also regularises it using a prior relying on an existing flight dynamics model[13]. This strategy allows them to identify the essential control inputs that dictated the trajectory of the UAV's autopilot. This might supply drone pilots with analytics or allow them to learn controllers from the demos. Finally, a new cost function is used in their BA technique. It is based on standard picture reprojection error but does not require explicit information association formed from photo correspondences, which is usually regarded a requirement in traditional point-based SfM[13].

Ceres non-linear least squares minimizer was used to implement their approach in C++[11]. To detect the UAV, we used the OpenCV implementation of Gaussian Mixture Models (GMM)-based noise reduction. In a word, it creates a GMM model for the background and updates it with each new frame. Furthermore, foreground parts of the image that do not match the model are considered foreground and are used as detections. However, because this results in a huge number of false positives for the outside films, we used a Kalman Filter (KF) with a constant acceleration model to process the detections[13]. Our BA optimization uses detections from the generated tracks. We only considered tracks with more than three times steps to decrease the number of false positives. Researchers then tested their suggested technique on synthetic results acquired from a realistic quadrotor flight sim to examine the precision and resilience of the predicted courses and command inputs in the face of picture distortion, outliers in tracing, and faults in the initial camera posture settings. They also provide many results based on real data recorded

indoors and in a huge outside scenario where ground truth trajectory information is available. By strongly aligning our trajectory estimate to the ground truth trajectory, we can assess the accuracy of our predicted trajectories. The root means the squared error is then calculated in the ground truth coordinate system (RMSE)[13]. They demonstrated a unique method for recreating the 3D route of a quadrotor UAV from several cameras. We've demonstrated that using motion data enhances the precision of the rebuilt route of an object in a three-dimensional environment. Moreover, the system can infer the quadrotor's internal characteristics, such as pitch angles, thrust, and roll from the operator's commands.

2.8 Comparison of Previous Research Paper

Authors	Research Title	Software/hardware main	Features
Simon Birnbach, Richard Baker, Ivan Martinovic[10]	Wi-Fly? : Detecting Privacy Invasion Attacks by Consumer Drones	Raspberry Pi, DJI Phantom 3 Standard, Parrot Bebop,	Considering three phase of attack which is approach, surveillance, and escape. Also testing different kind of approach patterns.
Aishah Moafa[12]	DRONES DETECTION USING SMART SENSORS	Only proposing and comparing different models	Proposing camera-based and camera and radar-based model and listing its risk assessment
Ulzhalgas Seidaliyeva, Daryn Akhmetov,	Real-Time and Accurate Drone Detection in a Video with a Static Background	GoPro, Faster R-CNN, Inception v2 and ResNet-101	Drone detection is based on deep learning to distinguish the drones from birds

Lyazzat Ilipbayeva, Eric T. Matson[14]			with static background
Sara Al-Emadi, Abdulla Al-Ali, Abdulaziz Al-Ali[5]	Audio-Based Drone Detection and Identification Using Deep Learning Techniques with Dataset Enhancement through Generative Adversarial Networks	Static camera, RNN, CNN, CRNN	Training and testing different kinds of audio from the drone and studying the difference between each result and trying to differentiate the results to every type of drone used
Artem Rozantsev, Sudipta N Sinha, Debadeepta Dey, Pascal Fua[13]	Flight Dynamics-based Recovery of a UAV Trajectory using Ground Cameras	Six ground cameras, laptop	Uses six cameras which consist of two Top, two Middle, and two Bottom cameras in order to get a 3D volume within the set amount of space
M, Vidyasagar P, Shoaib, Syed, Prasad, Shiva M, Kashyap, Sathwik R, N, Lethan M[15]	Detection and Surveillance of UAVs Based on RF and Radar Technology	Radar, RF Analyser, Web Camera, Arduino	Uses three types of surveillance which is RF, radar and video and a microcontroller board to interface all these components.
Xiufang Shi, Chaoqun Yang, Weige Xie, Chao Liang, Zhiguo Shi and Jiming Chen[6]	Anti-Drone System with Multiple Surveillance Technologies: Architecture, Implementation, and Challenges	Spectrum analyzer, optical camera, acoustic arrays	Uses three types of sensing unit which is sound, video and RF and a central processing unit to conduct drone feature extraction, drone detection and drone localization.
Al-Emadi, Sara, Al-Senaid, Felwa[8]	Drone Detection Approach Based on Radio-Frequency Using Convolutional Neural Network	2 different CPU, 2 different GPU, 2 different OS and RF sensor	Employed deep learning in the processes of detection, identification and classification problem.
Huynh-The, Thien, Pham, Quoc Viet, Nguyen, Toan	RF-UAVNet: High-Performance Convolutional Network	2 units of Universal Software Radio Peripheral (USRP)	Intercept the RF signal from the drone by the software-defined radio configurable

Van, Costa, Daniel Benevides Da, Kim, Dong Seong[4]	for RF-Based Drone Surveillance Systems		devices and store in the lacal database repository for processing.
Basak, Sanjoy, Rajendran, Sreeraj, Pollin, Sofie, Scheers, Bart[2]	Combined RF-based drone detection and classification	RF analyser, SDR	The YOLO-lite architecture is recreated from scratch and modified to perform the combined drone signal detection and classification.
Phuc Nguyen, Mahesh Ravindranathan, Anh Nguyen, Richard Han and Tam Vu[11]	Investigating Cost-effective RF-based Detection of Drones	RF analyser, USRP, SDR	Presented different approached for drone detection using RF technology

Table 2.2 Journals Comparison

2.9 Summary

Based on the previous research and existing product, we can conclude that the best sensor for drone detection is radar and the acoustic sensor comes second with the last place being a ground camera. However, both the best sensors have complex construction and simulation and the price for each sensor is too expensive and is not affordable, especially if the target market for this product is the homeowner and not big company. So, we settle with RF and camera because of their efficiency and affordable market price.

Aside from that, the main component which is microcontroller will be Arduino Uno. This microcontroller is easy to program and edit by using its own software editor, Aduino IDE. Arduino is able to operate multiple sensors at once and has been used by thousands all around the world so it is a trusted component.

CHAPTER 3

METHODOLOGY

3.1 Introduction

The methodology chapter will be about the project in more detailed parts. In this part of the report, we will discuss more in detail how the project will be conducted in order to meet the goal of this project. In this chapter, we will include three parts of the workflow, the first one is the flow chart. In this part, we will show the flow of process of the system that will be conducted. Then the next part is the hardware, this part will include all the equipment and materials that are used in the process of completing this project. Lastly, the project will show the complete circuit diagram of the system that will run the program in order to run this project smoothly.

3.2 Study Design

The main objective of this project is to develop a system of drone detection that is accurate but also affordable. In order to achieve this aim, we must use the type of sensor that is easy to get your hands on but also must be dependable. Based on this objective we have chosen the RF sensor and also using video-based detection. Both of these methods have an easy to get equipment that is affordable and also precise. An Arduino Uno will be used as a microcontroller that will control and run the process of detection in the system. Arduino IDE software will be used in order to edit and compile the program that is being run in the system.

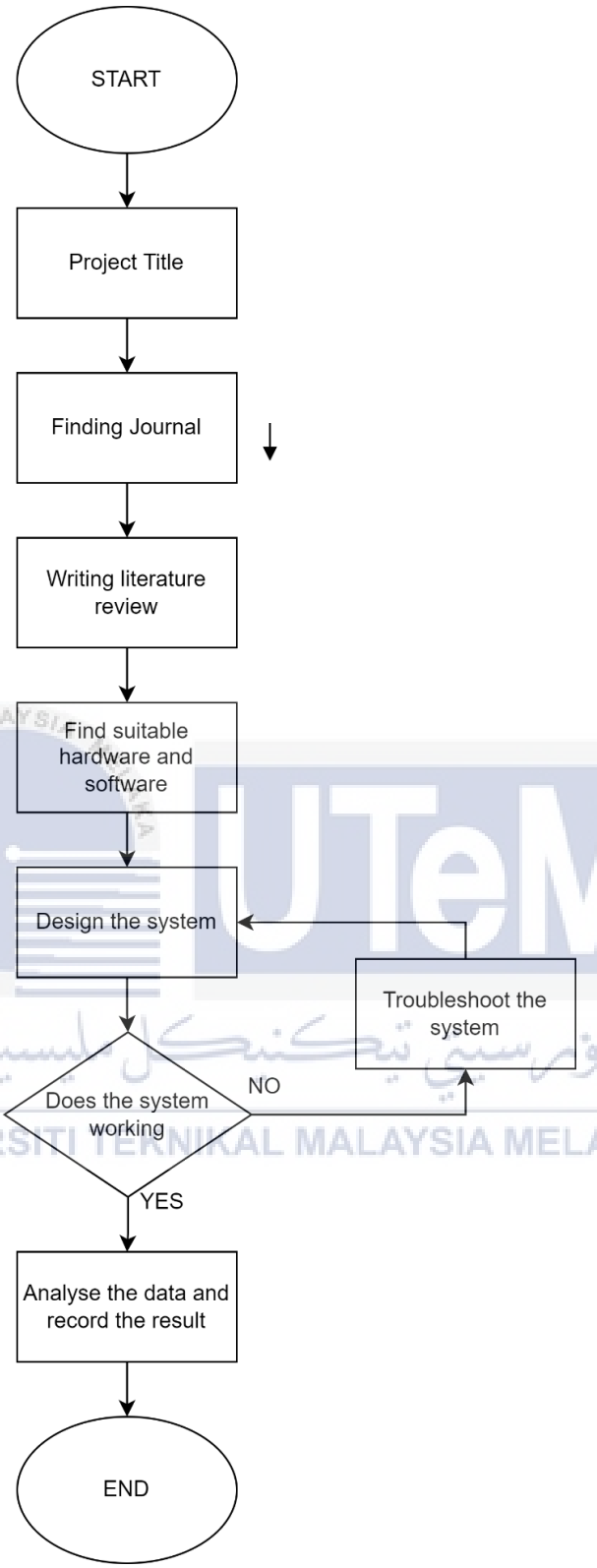

Figure 3.1 Flowchart of Project

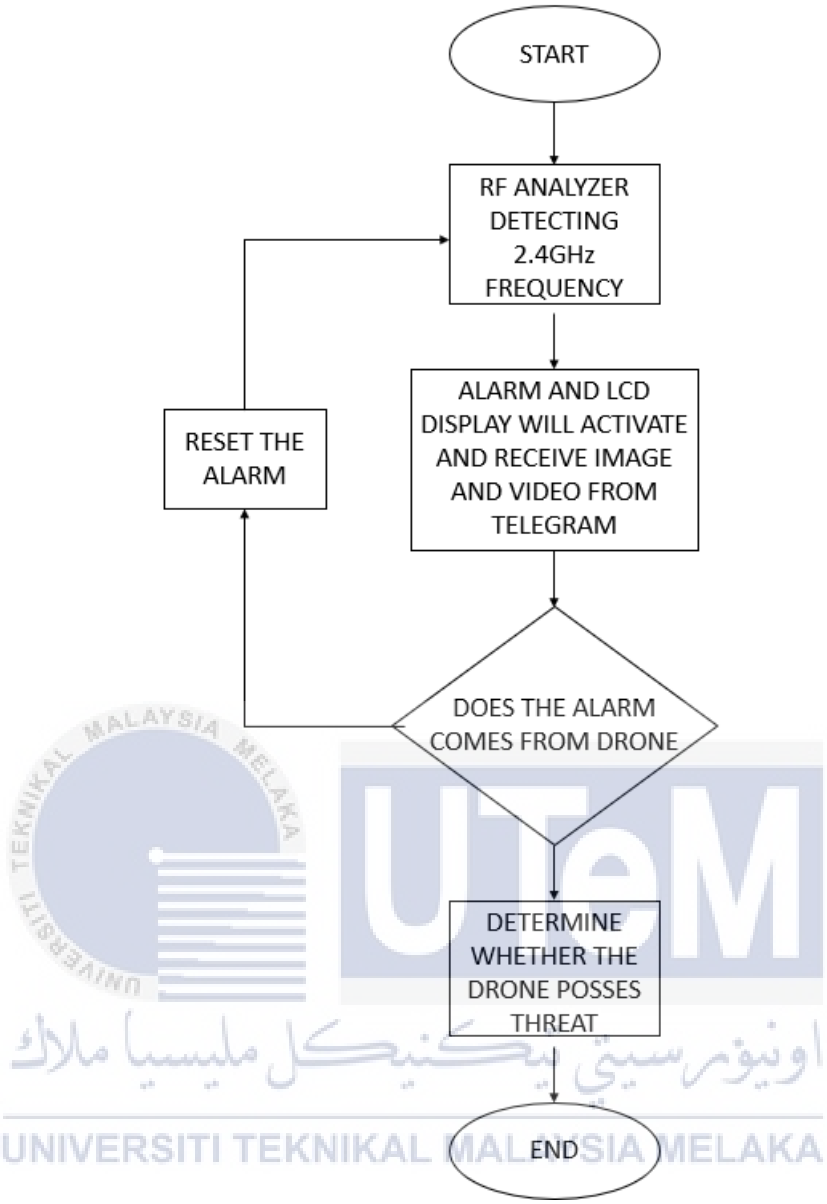

Table 3.1 Flowchart Of The System

3.3 Flowchart Explanation

This project uses two types of drone detection methods in order to achieve the objective of this project. If either of the sensors detects any abnormality, the system will start processing the data from the sensor and determine whether the abnormality comes from any

UAVs or other things. If the system has determined that the trespasser is a drone, it will continue to determine the level of threat of that drone. This step is important to prevent the

destruction of an innocent drone. Because nowadays many couriers company use drones as their way to transport something replacing humans. If the drone is harmless, the system will contact the homeowner to inform the arrival of the drone. But if the drone poses threat to the property, the system will move to the last step, which is to stop the drone from advancing any further. This flow will continue after the system has determined that the environment is safe.

3.4 Hardware specification

The hardware that we will be focussing on in this project is the microcontroller and sensors. These components are the most important part of the system. We are using two types of sensors RF sensors and also Wi-fi connection sensors.

3.4.1 Arduino microcontroller

In this project, we are using Arduino Uno as a microcontroller. This type of microcontroller can be programmed by using C and C++ programming languages. Arduino Nano are also used in this project as a controller for the receiver side on the drone system. Arduino Nano has a small form factor which are perfect to be place inside the body of the drone. Although this microcontroller has a small size, the functionality of this Arduino is still the same as the other kind of Arduino. The only significant differences between it and the UNO are the lack of a DC power jack, the use of a Mini USB port rather than a USB B port, and the USB-TTL converter chip unlike Arduino UNO that is powered by ATMega328P microcontroller. Arduino UNO has a much more input/output port compared to Arduino Nano.

Figure 3.2 Arduino UNO

3.4.2 RF analyser

The nRF24L01 is a single-chip radio transceiver that operates in the global 2.4 - 2.5 GHz ISM band[15]. A fully integrated frequency synthesiser, a power amplifier, a crystal oscillator, a demodulator, a modulator, and an Enhanced ShockBurstTM protocol engine comprise the transceiver. A SPI interface allows you to easily programme output power, frequency channels, and protocol setup. The current consumption is extremely low, only 9.0mA at -6dBm output power and 12.3mA in RX mode. Built-in Power Down and Standby modes make power conservation simple.

Figure 3.3 nRF24L01 RF Analyser

3.4.3 Camera

The ESP32-CAM are used as the device for monitoring and collecting image and video data of the intruding drone. The small size and the low-cost of this cam module are perfect for this project. This camera module are widely used in todays IoT application because of its small factor and its special features. The ESP32-CAM are commonly used in the QR identification, home applications and wireless surveying. It is a perfect answer for the IoT applications.

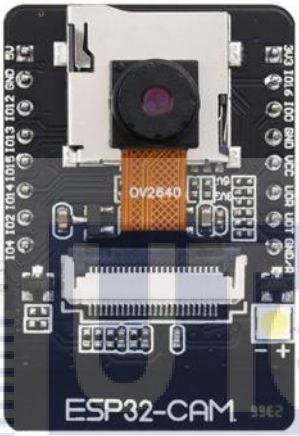

Figure 3.4 ESP32-CAM

3.4.4 Battery

Lithium Polymer (LiPo) type of battery used as the power supply for the Arduino Nano and Rx placed at drone body. LiPo rechargeable battery are used widely in many electronic sector including drone manufacturing. We are using 7.4V of ordinary voltage at 1400mAh capacity. The battery must be charge with LiPo battery charger itself. It has a small size and lightweight that became the critical factor in choosing this kind of battery.

Figure 3.5 Lipo SM connector battery LJ 501855 7.4v 1400mah

3.4.5 Buzzer

The buzzer is a sounding device capable of converting audio signals into sound signals. It is typically powered by direct current (DC). It is widely used as a sound device in alarm clocks, computers, printers, and other electronic products.

Figure 3.6 Buzzer

3.4.6 LCD Display

These LCDs are designed specifically for displaying text/characters, hence the name 'Character LCD.' The display has an LED backlight and can display 32 ASCII characters in two rows of 16, each with 16 characters.

Figure 3.7 LCD Display 16x2

3.5 Software Application

3.5.1 Arduino IDE

The Arduino Integrated Development Environment (IDE), also known as the Arduino Software (IDE), includes a code editor, a message area, a text console, a toolbar with buttons for common functions, and a series of menus. It communicates with and uploads programmes to the Arduino hardware.

3.5.2 Telegram

Telegram messenger is a online chatting planform that can be used to send image, video, document, location, audio file and animated sticker to other users. Unlike Whatsapp, telegram includes build-in browser that allow users to browse website while still inside the app. Telegram also supports third party bot that provide additional features. These bots can do almost any task including converting files, playing games and translates words. But in our case, we used a bot to create a new bot that can interact with ESP32-CAM. This new bot will allow the ESP32-CAM to capture image and record video and send the files to the telegram app.

3.6 Block Diagram

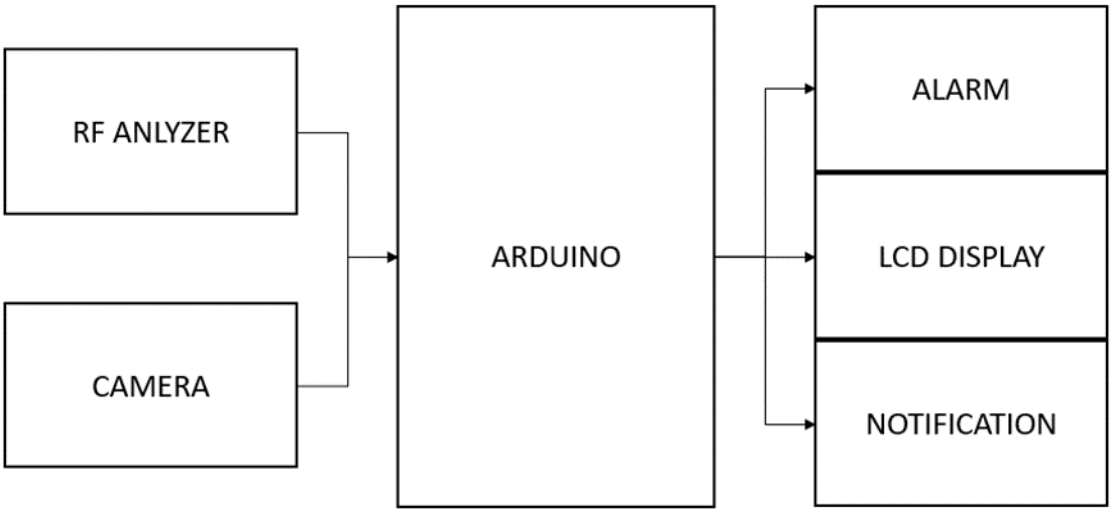

Figure 3.8 Block Diagram of the System

From this block diagram, the radio frequency or RF analyser is acting as a detector for any incoming drones. The camera used to feed live feed of footage in situation where the

RF analyser detected RF signal. The footage will be displayed to the homeowner for making a decision whether the drone is safe or not. The data that it collects when detecting the drones then will be transferred to the Arduino and the Arduino will also determine whether the drone is harmless or a threat. If the drone possess a threat to the homeowner, the alarm will go off and the GSM module will notify the owner about the intruder. However, if the drone was proved harmless the alarm will not go off but the GSM module will still notify the owner about the arrival of the drone.

3.7 Circuit Diagram

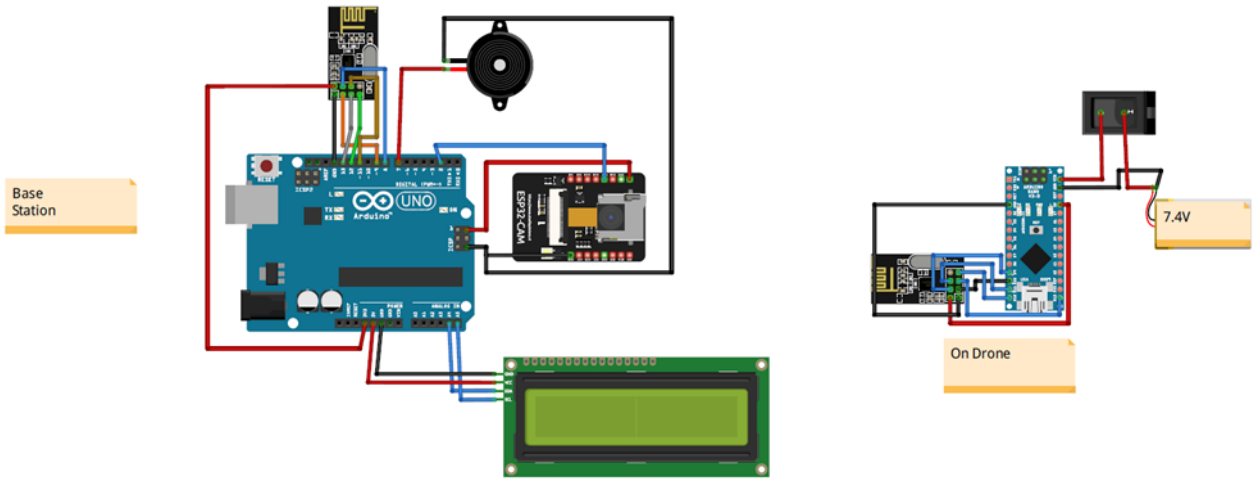

Figure 3.9 Circuit Diagram

From the figure above, the circuit of drone detection consist of Modula TX and Modula RX as RF module that act as input for the circuit. The Arduino UNO are used as central processing unit that process the flow and transmit the data to the output. For the output part we have three components. The first one is buzzer that act as alarm to warn the user about the intruder. Then, the ESP32-Cam will send the notification through the Telegram application to your phone using ESP32-Cam Bot. The last component we have is LCD display that will be displaying all the process or the output of the process.

3.8 Summary

This chapter presents the proposed methodology in order to develop a new, effective and integrated approach in estimating large scale/system wide TL of medium voltage (MV) network. The primary focus of the proposed methodology is in accomplishing a simple, less rigorous and effective estimation in such a way that it would not cause a significant loss of accuracy of the results. The methods also intended to use the generally available and limited data of the network and load from the power utilities. The ultimate intend of the method is not to obtain highest

accuracy, but, for efficiency, easy to use and manipulate and practicality of deployment on a large-scale distribution network.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

This chapter will follow through with the development of software and hardware used in this project. The simulation and result will be analysed and discussed based on the development of drone detection system using Arduino to enhance privacy purposes. This chapter will also discuss the existing current market device and compare it with the actual project. This part will help with the outcome of this project based on the availability and the current market of the component.

4.2 Software Development

For this project, Arduino IDE are used in software development. This project's software development is done using the Arduino IDE. The Arduino IDE is code and compiler software in which we begin by selecting the library that will be implemented and applying it to the software. The library recognises the software and makes it easier for the developer to use the component that will be constructed.

4.3 Interface Of The System

The system of this project starts when the nRF24L01 detecting the radio frequency emitted from the drone to its controller. The frequency will be transmitted to the receiver end of the RF analyzer. When the microcontroller has receive the detection from the RF analyzer, it will activate the buzzer, displaying intruder alert on LED display and send the image and also video of the event through ESP32-CAM. The data will be sent to the telegram application. The image will be sent to the bot on the app on the spot, however the video has delay around one minute before reaching the

user. The duration of the video are around 10 seconds and the buzzer and LED display will continue active until the drone leave the area or by resetting the system.

Figure 4.1 Telegram Bot

4.4 Hardware development

Hardware used in this project consist of two types Arduino which is Arduino Uno and Arduino Nano. Next, for the sensor we have nRF24L01 and ESP32-CAM. For the output, we have buzzer and also LCD display. Last but not least, we also use Lipo Lipo SM connector battery LJ 501855 7.4v 1400mah as power supply for Arduino Nano. Figure 4.2 shows the configuration of hardware for the base station.

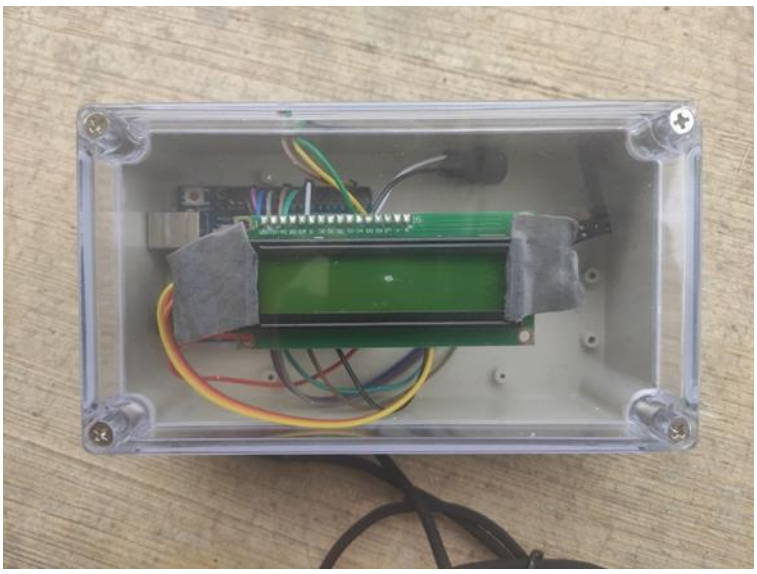

Figure 4.2 Base Station

Figure 4.3 Hardware on Drone

In the figure 4.3 shown how the systems looks like on the body of the drone. On the drone's body , we connect the battery, Arduino Nano and transmitter without the enclosure. The other components are placed at the base station.

4.5 Analysis Of The System

The analysis of this system will contain a few numbers of test on the distance of the receiving of the RF of the drone to the system. We will test how much distance it takes for the system to detect the drone and also the difference between RC car and drone.

4.5.1 Analysis On The Distance of Detection Using Power of 5V 2A

Distance (meter)	Detection
10	YES
20	YES
30	YES
40	YES
50	NO

Table 4.1 Data of the Detection Using Power of 5V 2A

Based on the Table 4.5.1.1, the test was conducted to measure the distance of detection of the system using power bank which power is 5V and 2A. the maximum distance that the system can detect is around 40 meter.

Figure 4.4 Distance of the Testing Area

4.5.2 Analysis On The Distance of Detection Using Power of 11V 3A

Distance (meter)	Detection
20	YES
40	YES
60	YES
80	YES
100	NO

Table 4.2 Data on The Distance of Detection Using Power of 11V 3A

Based on Table 4.5.2.1, the test was conducted using a power adapter with the power of 11V 3A. As we can see from the data, the distance of detection are much bigger than when using a power bank. The maximum distance of the detection that we get is around 80 meter.

Figure 4.5 Distance of the Testing Area 100m

4.5.3 Analysis On The Time Delay Vs Distance Using Power of 5V 2A

Distance (meter)	Delay (second)
10	0
20	0.5
30	0.5
40	1
50	0

Table 4.3 Data On The Time Delay Vs Distance Using Power of 5V 2A

Based on Table 4.5.3.1, we run the test same as the distance but we added the delay time for the system to detect the frequency of the drone. The test ran on the same place with addition of timer to record the time of detection. From the data that was recorded, the longest time for the system to detect the drone was about 0.9 seconds at 40 meters.

4.5.4 Analysis On The Time Delay Vs Distance Using Power of 11V 3A

Distance (meter)	Delay (second)
20	0.5
40	1
60	1.5
80	2
100	0

Table 4.4 Data On The Time Delay Vs Distance Using Power of 11V 3A

Based on Table 4.5.4.1, the test ran for the distance of 100 meter. And during this test once again testing on the time delay of the drone detection system using the power adapter. Then we record the time taken for the system to detect the active drone. At the maximum distance of 80 meters we get the time delay of 1.2 seconds.

4.6 Summary

This section discussed the project's development, which included the hardware, software, and system interface. Aside from that, the analysis in this chapter compares an RF based monitoring and Arduino to a generic device to determine the tolerance value.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

From this project, we can conclude that the aim to detect the drone using Arduino has been achieved. The drone detection using Arduino is successful by using the RF analyser and camera as the sensor and by adding other technologies into the system, the system has been completed. Aside from that, the system also is simple and affordable and can be afforded by all people that want their property safe from any invasion from drones.

Based on the result, the analysis showed that the objective of this project has been successfully achieved. However, there are a lot of improvements that can be added into the system to make it more reliable and convenient such as adding storage that can store footage of the invasion and upload it into the cloud.

5.2 Future Works

This system can still be improvised to make it more reliable and convenient such as adding storage that can store footage of the invasion and upload it into the cloud. The footage that has been stored can be used as proof for future use. The RF sensor also can be improved, by using new and advanced technologies such as Universal Software Radio Peripheral (USRP) software defined radios (SDR). This type of sensor is more advanced and reliable compared to the RF analyser because it has a bigger range of frequency and it is a software- defined RF architecture provided for rapidly designing, prototyping, and deploying wireless systems with custom signal processing.

REFERENCES

[1] P. Nguyen *et al.*, "DroneScale: Drone load estimation via remote passive RF sensing," in *SenSys 2020 - Proceedings of the 2020 18th ACM Conference on Embedded Networked Sensor Systems*, Nov. 2020, pp. 326–339. doi: 10.1145/3384419.3430778.

[2] "Combined RF-based drone detection and classification", doi:

10.36227/techrxiv.14991999.v1.

[3] P. Nguyen, H. Truong, M. Ravindranathan, A. Nguyen, R. Han, and T. Vu, "Matthan: Drone presence detection by identifying physical signatures in the drone's RF communication," in *MobiSys 2017 - Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, Jun. 2017, pp. 211–224. doi: 10.1145/3081333.3081354.

[4] T. Huynh-The, Q. V. Pham, T. Van Nguyen, D. B. Da Costa, and D. S. Kim, "RF-UAVNet: High-Performance Convolutional Network for RF-Based Drone Surveillance Systems," *IEEE Access*, vol. 10, pp. 49696–49707, 2022, doi: 10.1109/ACCESS.2022.3172787.

[5] S. Al-Emadi, A. Al-Ali, and A. Al-Ali, "Audio-based drone detection and identification using deep learning techniques with dataset enhancement through generative adversarial networks†," *Sensors*, vol. 21, no. 15, Aug. 2021, doi: 10.3390/s21154953.

[6] X. Shi, C. Yang, W. Xie, C. Liang, Z. Shi, and J. Chen, "Anti-Drone System with Multiple Surveillance Technologies: Architecture, Implementation, and Challenges," *IEEE Commun. Mag.*, vol. 56, no. 4, pp. 68–74, Apr. 2018, doi: 10.1109/MCOM.2018.1700430.

[7] V. Matić, V. Kosjer, A. Lebl, B. Pavić, and J. Radivojević, “Methods for Drone Detection and Jamming,” 2020.

[8] S. Al-Emadi and F. Al-Senaid, “Drone Detection Approach Based on Radio-Frequency Using Convolutional Neural Network,” in *2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies, ICIoT 2020*, Feb. 2020, pp. 29–34. doi: 10.1109/ICIoT48696.2020.9089489.

[9] “2003.02656”.

[10] S. Birnbach, R. Baker, and I. Martinovic, “Wi-Fly?: Detecting Privacy Invasion Attacks by Consumer Drones,” May 2017. doi: 10.14722/ndss.2017.23335.

[11] P. Nguyen, M. Ravindranathan, A. Nguyen, R. Han, and T. Vu, “Investigating cost-effective RF-based detection of drones,” in *DroNet 2016 - Proceedings of the 2nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use, co-located with MobiSys 2016*, Jun. 2016, pp. 17–22. doi: 10.1145/2935620.2935632.

[12] A. Moafa, “DRONES DETECTION USING SMART SENSORS,” 2020.

[13] A. Rozantsev, S. N. Sinha, D. Dey, and P. Fua, “Flight Dynamics-based Recovery of a UAV Trajectory using Ground Cameras.”

[14] U. Seidaliyeva, D. Akhmetov, L. Ilipbayeva, and E. T. Matson, “Real-time and accurate drone detection in a video with a static background,” *Sensors (Switzerland)*, vol. 20, no. 14, pp. 1–18, Jul. 2020, doi: 10.3390/s20143856.

[15] V. P. M, S. Shoaib, S. M. Prasad, S. R. Kashyap, and L. M. N, “Detection and Surveillance of UAVs Based on RF and Radar Technology,” *Int. Res. J. Eng. Technol.*, 2021, [Online]. Available: www.irjet.net

[16] OmniVision Technologies, “OV7670/OV7171 CMOS VGA (OmniPixel ®) CAMERACHIP™ Sensor Omnivision,” pp. 1–43, 2006, [Online]. Available:

https://www.voti.nl/docs/OV7670.pdf

APPENDICES

Coding For Transmitter

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

RF24 radio(10, 9); // CE, CSN

const byte address[6] = "00001";

void setup() {
  Serial.begin(9600);
  if (!radio.begin()) {
    Serial.println(F("radio hardware is not responding!!"));
    while (1) {}  // hold in infinite loop
  }
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}

void loop() {
  const char text[] = "Hello World";
//  radio.write(&text, sizeof(text));
  int tests=1288;
  radio.write(&tests, sizeof(tests));
  delay(1000);
}
```


Coding For Receiver

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

RF24 radio(9, 8); // CE, CSN
LiquidCrystal_I2C lcd(0x27, 16, 2);
#define buzzers 7
#define trigger 2
const byte address[6] = "00001";
```

```
void setup() {
  Serial.begin(9600);
  // initialize the LCD
  lcd.init();
  lcd.backlight();
  lcd.print("Drone Detection");
  lcd.setCursor(3,1);
  lcd.print("System");
  //lcd.clear();
  pinMode(buzzers,OUTPUT);
  pinMode(trigger,OUTPUT);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  digitalWrite(buzzers,0);
  digitalWrite(trigger,0);
}

void loop() {
  radio.startListening();
  int rec=0;
  if (radio.available()>0) {
//    char text[32] = {0};

//    radio.read(&text, sizeof(text));
    radio.read(&rec, sizeof(rec));

    Serial.print("Receive : ");
    Serial.println(rec);
    Serial.print("Sizeof : ");
    Serial.println(sizeof(rec));
//    if(sizeof(rec)>2)
  if(rec==1288)
    {
      digitalWrite(trigger,1);
      lcd.setCursor(0,0);
      lcd.print("Drone Detected!");
      lcd.setCursor(0,1);
      lcd.print("            ");
      digitalWrite(buzzers,1);
      delay(1000);
       digitalWrite(buzzers,0);
      delay(1000);

      }
  }
  else
  {
    lcd.setCursor(0,0);
```

```
    lcd.print("Drone Detection");
    lcd.setCursor(0,1);
    lcd.setCursor(3,1);
    lcd.print("System");
    digitalWrite(buzzers,0);
    digitalWrite(trigger,0);
   }


}
```

Coding For ArduinoJson Library

```
{
  "name":"Arduino on ESP32",
  "toolchainPrefix":"xtensa-esp32-elf",
  "svdFile":"esp32.svd",
  "request":"attach",
  "postAttachCommands":[
    "set remote hardware-watchpoint-limit 2",
    "monitor reset halt",
    "monitor gdb_sync",
    "thb setup",
    "c"
  ],
  "overrideRestartCommands":[
    "monitor reset halt",
    "monitor gdb_sync",
    "thb setup",
    "c"
  ]
```


Coding For Creating Telegram Bot

```
#include "UniversalTelegramBot.h"

#define ZERO_COPY(STR)    ((char*)STR.c_str())
#define BOT_CMD(STR)      buildCommand(F(STR))

UniversalTelegramBot::UniversalTelegramBot(const String& token, Client &client) {
  updateToken(token);
  this->client = &client;
}

void UniversalTelegramBot::updateToken(const String& token) {
  _token = token;
}
```

```
String UniversalTelegramBot::getToken() {
  return _token;
}

String UniversalTelegramBot::buildCommand(const String& cmd) {
  String command;

  command += F("bot");
  command += _token;
  command += F("/");
  command += cmd;

  return command;
}

String UniversalTelegramBot::sendGetToTelegram(const String& command) {
  String body, headers;
  bool avail;

  // Connect with api.telegram.org if not already connected
  if (!client->connected()) {
    #ifdef TELEGRAM_DEBUG
       Serial.println(F("[BOT]Connecting to server"));
    #endif
    if (!client->connect(TELEGRAM_HOST, TELEGRAM_SSL_PORT)) {
      #ifdef TELEGRAM_DEBUG
       Serial.println(F("[BOT]Conection error"));
      #endif
    }
  }
  if (client->connected()) {

    #ifdef TELEGRAM_DEBUG
       Serial.println("sending: " + command);
    #endif

    client->print(F("GET /"));
    client->print(command);
    client->println(F(" HTTP/1.1"));
    client->println(F("Host:" TELEGRAM_HOST));
    client->println(F("Accept: application/json"));
    client->println(F("Cache-Control: no-cache"));
    client->println();

    readHTTPAnswer(body, headers);
  }

  return body;
}
```

```
bool UniversalTelegramBot::readHTTPAnswer(String &body, String &headers) {
  int ch_count = 0;
  long now = millis();
  bool finishedHeaders = false;
  bool currentLineIsBlank = true;
  bool responseReceived = false;

  while (millis() - now < longPoll * 1000 + waitForResponse) {
    while (client->available()) {
      char c = client->read();
      responseReceived = true;

      if (!finishedHeaders) {
        if (currentLineIsBlank && c == '\n') {
          finishedHeaders = true;
        } else {
          headers += c;
        }
      } else {
        if (ch_count < maxMessageLength) {
          body += c;
          ch_count++;
        }
      }

      if (c == '\n') currentLineIsBlank = true;
      else if (c != '\r') currentLineIsBlank = false;
    }

    if (responseReceived && ch_count > 5) { //jz
      #ifdef TELEGRAM_DEBUG
        Serial.print(">");
        Serial.print(body);
        Serial.println("<");
      #endif
      break;
    }
  }
  return responseReceived;
}

String UniversalTelegramBot::sendPostToTelegram(const String& command, JsonObject
payload) {

  String body;
  String headers;

  // Connect with api.telegram.org if not already connected
  if (!client->connected()) {
```

```
  #ifdef TELEGRAM_DEBUG
     Serial.println(F("[BOT Client]Connecting to server"));
  #endif
  if (!client->connect(TELEGRAM_HOST, TELEGRAM_SSL_PORT)) {
   #ifdef TELEGRAM_DEBUG
     Serial.println(F("[BOT Client]Conection error"));
   #endif
  }
 }
 if (client->connected()) {
  // POST URI
  client->print(F("POST /"));
  client->print(command);
  client->println(F(" HTTP/1.1"));
  // Host header
  client->println(F("Host:" TELEGRAM_HOST));
  // JSON content type
  client->println(F("Content-Type: application/json"));

  // Content length
  int length = measureJson(payload);
  client->print(F("Content-Length:"));
  client->println(length);
  // End of headers
  client->println();
  // POST message body
  String out;
  serializeJson(payload, out);

  client->println(out);
  #ifdef TELEGRAM_DEBUG
     Serial.println(String("Posting:") + out);
  #endif

  readHTTPAnswer(body, headers);
 }

 return body;
}

String UniversalTelegramBot::sendMultipartFormDataToTelegram(
   const String& command, const String& binaryPropertyName, const String& fileName,
   const String& contentType, const String& chat_id, int fileSize,
   MoreDataAvailable moreDataAvailableCallback,
   GetNextByte getNextByteCallback,
   GetNextBuffer getNextBufferCallback,
   GetNextBufferLen getNextBufferLenCallback) {

 String body;
 String headers;
```

```
  const String boundary = F("------------------------b8f610217e83e29b");

  // Connect with api.telegram.org if not already connected
  if (!client->connected()) {
    #ifdef TELEGRAM_DEBUG
       Serial.println(F("[BOT Client]Connecting to server"));
    #endif
    if (!client->connect(TELEGRAM_HOST, TELEGRAM_SSL_PORT)) {
      #ifdef TELEGRAM_DEBUG
        Serial.println(F("[BOT Client]Conection error"));
      #endif
    }
  }
  if (client->connected()) {
    String start_request = "";
    String end_request = "";

    start_request += F("--");
    start_request += boundary;
    start_request += F("\r\ncontent-disposition: form-data; name=\"chat_id\"\r\n\r\n");
    start_request += chat_id;
    start_request += F("\r\n" "--");
    start_request += boundary;
    start_request += F("\r\ncontent-disposition: form-data; name=\"");
    start_request += binaryPropertyName;
    start_request += F("\"; filename=\"");
    start_request += fileName;
    start_request += F("\"\r\n" "Content-Type: ");
    start_request += contentType;
    start_request += F("\r\n" "\r\n");

    end_request += F("\r\n" "--");
    end_request += boundary;
    end_request += F("--" "\r\n");

    client->print(F("POST /"));
    client->print(buildCommand(command));
    client->println(F(" HTTP/1.1"));
    Serial.print("*")  ; delay(jzdelay);

    // Host header
    client->println(F("Host: " TELEGRAM_HOST));       //jz <<<<<<<<<<<<<<<<<<<<<<
println !!!!!!!! >>>>>>>>>>>>>>>>>>>
    client->println(F("User-Agent: arduino/1.0"));
    Serial.print("*")  ; delay(jzdelay);
    client->println(F("Accept: */*"));
    Serial.print("*")  ; delay(jzdelay);

    int contentLength = fileSize + start_request.length() + end_request.length();
```

```
  #ifdef TELEGRAM_DEBUG
    Serial.println("Content-Length: " + String(contentLength));
  #endif
  client->print(F("Content-Length: "));
  client->println(String(contentLength));
  Serial.print("*")  ; delay(jzdelay);
  client->print(F("Content-Type: multipart/form-data; boundary="));
  client->println(boundary);
  //Serial.print(" --- bef ---")  ; delay(jzdelay);
  client->println();   // client->println(F"");
  //Serial.print(" --- aft ---")  ; delay(jzdelay);
  client->print(start_request);
  Serial.print("*")  ; delay(jzdelay);

  #ifdef TELEGRAM_DEBUG
   Serial.print("Start request: " + start_request);
  #endif

  if (getNextByteCallback == nullptr) {
    while (moreDataAvailableCallback()) {
      client->write((const uint8_t *)getNextBufferCallback(),
getNextBufferLenCallback());
      #ifdef TELEGRAM_DEBUG
       Serial.println(F("Sending photo from buffer"));
      #endif
      }
  } else {
    #ifdef TELEGRAM_DEBUG
      Serial.println(F("Sending photo by binary"));
    #endif
    //byte buffer[jzblocksize];
    int count = 0;
    char ch;
    while (moreDataAvailableCallback()) {
      buffer[count] = getNextByteCallback();
      count++;
      if (count == jzblocksize) {
        // yield();
        #ifdef TELEGRAM_DEBUG
          //jz Serial.println(F("Sending binary photo full buffer"));
        #endif
        client->write((const uint8_t *)buffer, jzblocksize);
        Serial.print("*")  ; delay(jzdelay);
        count = 0;
      }
    }

    if (count > 0) {
      #ifdef TELEGRAM_DEBUG
```

```
            Serial.println(F("Sending binary photo remaining buffer"));
          #endif
          client->write((const uint8_t *)buffer, count);
          Serial.print("*")  ; delay(jzdelay);
        }
    }

    client->print(end_request);
    Serial.print("*")  ; delay(jzdelay);

    #ifdef TELEGRAM_DEBUG
       Serial.print("End request: " + end_request);
    #endif

    //Serial.print("... Done Sending.  Client.Available = ");  Serial.println(client->available());
    //delay(2000);
    //Serial.print("... 2 secs later.  Client.Available = ");  Serial.println(client->available());
    readHTTPAnswer(body, headers);
  }

  closeClient();
  return body;
}

String UniversalTelegramBot::sendMultipartFormDataToTelegramWithCaption(
    const String& command, const String& binaryPropertyName, const String& fileName,
    const String& contentType, const String& caption, const String& chat_id, int fileSize,
    MoreDataAvailable moreDataAvailableCallback,
    GetNextByte getNextByteCallback,
    GetNextBuffer getNextBufferCallback,
    GetNextBufferLen getNextBufferLenCallback) {

  String body;
  String headers;

  const String boundary = F("------------------------b8f610217e83e29b");

  // Connect with api.telegram.org if not already connected
  if (!client->connected()) {
    #ifdef TELEGRAM_DEBUG
       Serial.println(F("[BOT Client]Connecting to server"));
    #endif
    if (!client->connect(TELEGRAM_HOST, TELEGRAM_SSL_PORT)) {
      #ifdef TELEGRAM_DEBUG
        Serial.println(F("[BOT Client]Conection error"));
      #endif
    }
  }
  if (client->connected()) {
```

```
    String start_request = "";
    String end_request = "";

    start_request += F("--");
    start_request += boundary;
    start_request += F("\r\ncontent-disposition: form-data; name=\"chat_id\"\r\n\r\n");
    start_request += chat_id;
    start_request += F("\r\n" "--");

// jz caption start
    start_request += boundary;
    start_request += F("\r\ncontent-disposition: form-data; name=\"caption\"\r\n\r\n");
    start_request += caption;
    start_request += F("\r\n" "--");
// jz caption end

    start_request += boundary;
    start_request += F("\r\ncontent-disposition: form-data; name=\"");
    start_request += binaryPropertyName;
    start_request += F("\"; filename=\"");
    start_request += fileName;
    start_request += F("\"\r\n" "Content-Type: ");
    start_request += contentType;
    start_request += F("\r\n" "\r\n");

    end_request += F("\r\n" "--");
    end_request += boundary;
    end_request += F("--" "\r\n");

    client->print(F("POST /"));
    client->print(buildCommand(command));
    client->println(F(" HTTP/1.1"));
    Serial.print("*")  ; delay(jzdelay);

    // Host header
    client->println(F("Host: " TELEGRAM_HOST));       //jz <<<<<<<<<<<<<<<<<<<<<<
println !!!!!!!! >>>>>>>>>>>>>>>>>>>
    client->println(F("User-Agent: arduino/1.0"));
    Serial.print("*")  ; delay(jzdelay);
    client->println(F("Accept: */*"));
    Serial.print("*")  ; delay(jzdelay);

    int contentLength = fileSize + start_request.length() + end_request.length();

    #ifdef TELEGRAM_DEBUG
       Serial.println("Content-Length: " + String(contentLength));
    #endif
    client->print(F("Content-Length: "));
    client->println(String(contentLength));
    Serial.print("*")  ; delay(jzdelay);
```

```
    client->print(F("Content-Type: multipart/form-data; boundary="));
    client->println(boundary);
    Serial.print("*")  ; delay(jzdelay);
    client->println();    // client->println(F(""));         <--- jz 1.05 bug
    Serial.print("*")  ; delay(jzdelay);
    client->print(start_request);
    Serial.print("*")  ; delay(jzdelay);

    #ifdef TELEGRAM_DEBUG
     Serial.print("Start request: " + start_request);
    #endif

    if (getNextByteCallback == nullptr) {
       while (moreDataAvailableCallback()) {
          client->write((const uint8_t *)getNextBufferCallback(),
getNextBufferLenCallback());
          #ifdef TELEGRAM_DEBUG
           Serial.println(F("Sending photo from buffer"));
          #endif
          }
    } else {
       #ifdef TELEGRAM_DEBUG
          Serial.println(F("Sending photo by binary"));
       #endif
       //byte buffer[jzblocksize];
       int count = 0;
       char ch;
       while (moreDataAvailableCallback()) {
          buffer[count] = getNextByteCallback();
          count++;
          if (count == jzblocksize) {
             // yield();
             #ifdef TELEGRAM_DEBUG
                //jz Serial.println(F("Sending binary photo full buffer"));
             #endif
             client->write((const uint8_t *)buffer, jzblocksize);
             Serial.print("*")  ; delay(jzdelay);
             count = 0;
          }
       }

       if (count > 0) {
          #ifdef TELEGRAM_DEBUG
             Serial.println(F("Sending binary photo remaining buffer"));
          #endif
          client->write((const uint8_t *)buffer, count);
          Serial.print("*")  ; delay(jzdelay);
       }
    }
```

```
    client->print(end_request);
    Serial.print("*")  ; delay(jzdelay);

    #ifdef TELEGRAM_DEBUG
       Serial.print("End request: " + end_request);
    #endif

    //Serial.print("... Done Sending.  Client.Available = ");  Serial.println(client-
>available());
    //delay(2000);
    //Serial.print("... 2 secs later.  Client.Available = ");  Serial.println(client->available());
    readHTTPAnswer(body, headers);
  }

  closeClient();
  return body;
}

bool UniversalTelegramBot::getMe() {
  String response = sendGetToTelegram(BOT_CMD("getMe")); // receive reply from
telegram.org
  DynamicJsonDocument doc(maxMessageLength);
  DeserializationError error = deserializeJson(doc, ZERO_COPY(response));
  closeClient();

  if (!error) {
    if (doc.containsKey("result")) {
      name = doc["result"]["first_name"].as<String>();
      userName = doc["result"]["username"].as<String>();
      return true;
    }
  }

  return false;
}

/*********************************************************************
*********
 * SetMyCommands - Update the command list of the bot on the telegram server     *
 * (Argument to pass: Serialied array of BotCommand)                             *
 * CAUTION: All commands must be lower-case                                      *
 * Returns true, if the command list was updated successfully                    *

*********************************************************************
*******/
bool UniversalTelegramBot::setMyCommands(const String& commandArray) {
  DynamicJsonDocument payload(maxMessageLength);
  payload["commands"] = serialized(commandArray);
  bool sent = false;
  String response = "";
```

```
  #if defined(_debug)
  Serial.println(F("sendSetMyCommands: SEND Post /setMyCommands"));
  #endif  // defined(_debug)
  unsigned long sttime = millis();

  while (millis() < sttime + 8000ul) { // loop for a while to send the message
    response = sendPostToTelegram(BOT_CMD("setMyCommands"),
payload.as<JsonObject>());
    #ifdef _debug
    Serial.println("setMyCommands response" + response);
    #endif
    sent = checkForOkResponse(response);
    if (sent) break;
  }

  closeClient();
  return sent;
}


/**********************************************************************
 * GetUpdates - function to receive messages from telegram *
 * (Argument to pass: the last+1 message to read)          *
 * Returns the number of new messages          *
 **********************************************************************/
int UniversalTelegramBot::getUpdates(long offset) {

  #ifdef TELEGRAM_DEBUG
    Serial.println(F("GET Update Messages"));
  #endif
  String command = BOT_CMD("getUpdates?offset=");
  command += offset;
  command += F("&limit=");
  command += HANDLE_MESSAGES;

  if (longPoll > 0) {
    command += F("&timeout=");
    command += String(longPoll);
  }
  String response = sendGetToTelegram(command); // receive reply from telegram.org

  if (response == "") {
    #ifdef TELEGRAM_DEBUG
       Serial.println(F("Received empty string in response!"));
    #endif
    // close the client as there's nothing to do with an empty string
    closeClient();
    return 0;
  } else {
    #ifdef TELEGRAM_DEBUG
```

```
  Serial.print(F("incoming message length "));
  Serial.println(response.length());
  Serial.println(F("Creating DynamicJsonBuffer"));
#endif

// Parse response into Json object
DynamicJsonDocument doc(maxMessageLength);
DeserializationError error = deserializeJson(doc, ZERO_COPY(response));

if (!error) {
  #ifdef TELEGRAM_DEBUG
    Serial.print(F("GetUpdates parsed jsonObj: "));
    serializeJson(doc, Serial);
    Serial.println();
  #endif
  if (doc.containsKey("result")) {
    int resultArrayLength = doc["result"].size();
    if (resultArrayLength > 0) {
      int newMessageIndex = 0;
      // Step through all results
      for (int i = 0; i < resultArrayLength; i++) {
        JsonObject result = doc["result"][i];
        if (processResult(result, newMessageIndex)) newMessageIndex++;
      }
      // We will keep the client open because there may be a response to be
      // given
      return newMessageIndex;
    } else {
      #ifdef TELEGRAM_DEBUG
        Serial.println(F("no new messages"));
      #endif
    }
  } else {
    #ifdef TELEGRAM_DEBUG
       Serial.println(F("Response contained no 'result'"));
    #endif
  }
} else { // Parsing failed
  if (response.length() < 2) { // Too short a message. Maybe a connection issue
    #ifdef TELEGRAM_DEBUG
       Serial.println(F("Parsing error: Message too short"));
    #endif
  } else {
    // Buffer may not be big enough, increase buffer or reduce max number of
    // messages
    #ifdef TELEGRAM_DEBUG
       Serial.print(F("Failed to parse update, the message could be too "
                      "big for the buffer. Error code: "));
       Serial.println(error.c_str()); // debug print of parsing error
    #endif
```

```
    }
   }
   // Close the client as no response is to be given
   closeClient();
   return 0;
  }
}

bool UniversalTelegramBot::processResult(JsonObject result, int messageIndex) {
  int update_id = result["update_id"];
  // Check have we already dealt with this message (this shouldn't happen!)
  if (last_message_received != update_id) {
    last_message_received = update_id;
    messages[messageIndex].update_id = update_id;
    messages[messageIndex].text = F("");
    messages[messageIndex].from_id = F("");
    messages[messageIndex].from_name = F("");
    messages[messageIndex].longitude = 0;
    messages[messageIndex].latitude = 0;
    messages[messageIndex].reply_to_message_id = 0;
    messages[messageIndex].reply_to_text = F("");
    messages[messageIndex].query_id = F("");

    if (result.containsKey("message")) {
      JsonObject message = result["message"];
      messages[messageIndex].type = F("message");
      messages[messageIndex].from_id = message["from"]["id"].as<String>();
      messages[messageIndex].from_name = message["from"]["first_name"].as<String>();
      messages[messageIndex].date = message["date"].as<String>();
      messages[messageIndex].chat_id = message["chat"]["id"].as<String>();
      messages[messageIndex].chat_title = message["chat"]["title"].as<String>();
      messages[messageIndex].hasDocument = false;
      if (message.containsKey("text")) {
        messages[messageIndex].text = message["text"].as<String>();

      } else if (message.containsKey("location")) {
        messages[messageIndex].longitude = message["location"]["longitude"].as<float>();
        messages[messageIndex].latitude  = message["location"]["latitude"].as<float>();
      } else if (message.containsKey("document")) {
        String file_id = message["document"]["file_id"].as<String>();
        messages[messageIndex].file_caption = message["caption"].as<String>();
        messages[messageIndex].file_name =
message["document"]["file_name"].as<String>();
        if (getFile(messages[messageIndex].file_path, messages[messageIndex].file_size,
file_id) == true)
          messages[messageIndex].hasDocument = true;
        else
          messages[messageIndex].hasDocument = false;
      }
      if (message.containsKey("reply_to_message")) {
```

```
        messages[messageIndex].reply_to_message_id =
message["reply_to_message"]["message_id"];
        // no need to check if containsKey["text"]. If it doesn't, it default to null
        messages[messageIndex].reply_to_text =
message["reply_to_message"]["text"].as<String>();
      }
    } else if (result.containsKey("channel_post")) {
      JsonObject message = result["channel_post"];
      messages[messageIndex].type = F("channel_post");
      messages[messageIndex].text = message["text"].as<String>();
      messages[messageIndex].date = message["date"].as<String>();
      messages[messageIndex].chat_id = message["chat"]["id"].as<String>();
      messages[messageIndex].chat_title = message["chat"]["title"].as<String>();

    } else if (result.containsKey("callback_query")) {
      JsonObject message = result["callback_query"];
      messages[messageIndex].type = F("callback_query");
      messages[messageIndex].from_id = message["from"]["id"].as<String>();
      messages[messageIndex].from_name = message["from"]["first_name"].as<String>();
      messages[messageIndex].text = message["data"].as<String>();
      messages[messageIndex].date = message["date"].as<String>();
      messages[messageIndex].chat_id = message["message"]["chat"]["id"].as<String>();
      messages[messageIndex].reply_to_text = message["message"]["text"].as<String>();
      messages[messageIndex].chat_title = F("");
      messages[messageIndex].query_id = message["id"].as<String>();
    } else if (result.containsKey("edited_message")) {
      JsonObject message = result["edited_message"];
      messages[messageIndex].type = F("edited_message");
      messages[messageIndex].from_id = message["from"]["id"].as<String>();
      messages[messageIndex].from_name = message["from"]["first_name"].as<String>();
      messages[messageIndex].date = message["date"].as<String>();
      messages[messageIndex].chat_id = message["chat"]["id"].as<String>();
      messages[messageIndex].chat_title = message["chat"]["title"].as<String>();

      if (message.containsKey("text")) {
        messages[messageIndex].text = message["text"].as<String>();

      } else if (message.containsKey("location")) {
        messages[messageIndex].longitude = message["location"]["longitude"].as<float>();
        messages[messageIndex].latitude  = message["location"]["latitude"].as<float>();
      }
    }
    return true;
  }
  return false;
}

/*******************************************************************
 * SendMessage - function to send message to telegram              *
 * (Arguments to pass: chat_id, text to transmit and markup(optional)) *
```

```
 *******************************************************************/
bool UniversalTelegramBot::sendSimpleMessage(const String& chat_id, const String& text,
                                             const String& parse_mode) {

  bool sent = false;
  #ifdef TELEGRAM_DEBUG
    Serial.println(F("sendSimpleMessage: SEND Simple Message"));
  #endif
  long sttime = millis();

  if (text != "") {
    while (millis() < sttime + 8000) { // loop for a while to send the message
      String command = BOT_CMD("sendMessage?chat_id=");
      command += chat_id;
      command += F("&text=");
      command += text;
      command += F("&parse_mode=");
      command += parse_mode;
      String response = sendGetToTelegram(command);
      #ifdef TELEGRAM_DEBUG
        Serial.println(response);
      #endif
      sent = checkForOkResponse(response);
      if (sent) break;
    }
  }
  closeClient();
  return sent;
}

bool UniversalTelegramBot::sendMessage(const String& chat_id, const String& text,
                                       const String& parse_mode) {

  DynamicJsonDocument payload(maxMessageLength);
  payload["chat_id"] = chat_id;
  payload["text"] = text;

  if (parse_mode != "")
    payload["parse_mode"] = parse_mode;

  return sendPostMessage(payload.as<JsonObject>());
}

bool UniversalTelegramBot::sendMessageWithReplyKeyboard(
    const String& chat_id, const String& text, const String& parse_mode, const String& keyboard,
    bool resize, bool oneTime, bool selective) {

  DynamicJsonDocument payload(maxMessageLength);
```

```
  payload["chat_id"] = chat_id;
  payload["text"] = text;

  if (parse_mode != "")
    payload["parse_mode"] = parse_mode;

  JsonObject replyMarkup = payload.createNestedObject("reply_markup");

  replyMarkup["keyboard"] = serialized(keyboard);

  // Telegram defaults these values to false, so to decrease the size of the
  // payload we will only send them if needed
  if (resize)
    replyMarkup["resize_keyboard"] = resize;

  if (oneTime)
    replyMarkup["one_time_keyboard"] = oneTime;

  if (selective)
    replyMarkup["selective"] = selective;

  return sendPostMessage(payload.as<JsonObject>());
}

bool UniversalTelegramBot::sendMessageWithInlineKeyboard(const String& chat_id,
                                                         const String& text,
                                                         const String& parse_mode,
                                                         const String& keyboard) {

  DynamicJsonDocument payload(maxMessageLength);
  payload["chat_id"] = chat_id;
  payload["text"] = text;

  if (parse_mode != "")
    payload["parse_mode"] = parse_mode;

  JsonObject replyMarkup = payload.createNestedObject("reply_markup");
  replyMarkup["inline_keyboard"] = serialized(keyboard);
  return sendPostMessage(payload.as<JsonObject>());
}

/**********************************************************************
 * SendPostMessage - function to send message to telegram             *
 * (Arguments to pass: chat_id, text to transmit and markup(optional)) *
 **********************************************************************/
bool UniversalTelegramBot::sendPostMessage(JsonObject payload) {

  bool sent = false;
  #ifdef TELEGRAM_DEBUG
    Serial.print(F("sendPostMessage: SEND Post Message: "));
```

```
    serializeJson(payload, Serial);
    Serial.println();
  #endif
  long sttime = millis();

  if (payload.containsKey("text")) {
    while (millis() < sttime + 8000) { // loop for a while to send the message
      String response = sendPostToTelegram(BOT_CMD("sendMessage"), payload);
      #ifdef TELEGRAM_DEBUG
        Serial.println(response);
      #endif
      sent = checkForOkResponse(response);
      if (sent) break;
    }
  }

  closeClient();
  return sent;
}

String UniversalTelegramBot::sendPostPhoto(JsonObject payload) {

  bool sent = false;
  String response = "";
  #ifdef TELEGRAM_DEBUG
    Serial.println(F("sendPostPhoto: SEND Post Photo"));
  #endif
  long sttime = millis();

  if (payload.containsKey("photo")) {
    while (millis() < sttime + 8000) { // loop for a while to send the message
      response = sendPostToTelegram(BOT_CMD("sendPhoto"), payload);
      #ifdef TELEGRAM_DEBUG
        Serial.println(response);
      #endif
      sent = checkForOkResponse(response);
      if (sent) break;

    }
  }

  closeClient();
  return response;
}

String UniversalTelegramBot::sendPhotoByBinary(
    const String& chat_id, const String& contentType, int fileSize,
    MoreDataAvailable moreDataAvailableCallback,
    GetNextByte getNextByteCallback, GetNextBuffer getNextBufferCallback,
GetNextBufferLen getNextBufferLenCallback) {
```

```
  #ifdef TELEGRAM_DEBUG
    Serial.println(F("sendPhotoByBinary: SEND Photo"));
  #endif

  String response = sendMultipartFormDataToTelegram("sendPhoto", "photo", "img.jpg",
    contentType, chat_id, fileSize,
    moreDataAvailableCallback, getNextByteCallback, getNextBufferCallback,
getNextBufferLenCallback);

  #ifdef TELEGRAM_DEBUG
    Serial.println(response);
  #endif

  return response;
}

String UniversalTelegramBot::sendPhoto(const String& chat_id, const String& photo,
                                       const String& caption,
                                       bool disable_notification,
                                       int reply_to_message_id,
                                       const String& keyboard) {

  DynamicJsonDocument payload(maxMessageLength);
  payload["chat_id"] = chat_id;
  payload["photo"] = photo;

  if (caption.length() > 0)
     payload["caption"] = caption;

  if (disable_notification)
     payload["disable_notification"] = disable_notification;

  if (reply_to_message_id && reply_to_message_id != 0)
     payload["reply_to_message_id"] = reply_to_message_id;

  if (keyboard.length() > 0) {
    JsonObject replyMarkup = payload.createNestedObject("reply_markup");
    replyMarkup["keyboard"] = serialized(keyboard);
  }

  return sendPostPhoto(payload.as<JsonObject>());
}

bool UniversalTelegramBot::checkForOkResponse(const String& response) {
  int last_id;
  DynamicJsonDocument doc(response.length());
  deserializeJson(doc, response);

  // Save last sent message_id
```

```
  last_id = doc["result"]["message_id"];
  if (last_id > 0) last_sent_message_id = last_id;

  return doc["ok"] | false;  // default is false, but this is more explicit and clear
}

bool UniversalTelegramBot::sendChatAction(const String& chat_id, const String& text) {

  bool sent = false;
  #ifdef TELEGRAM_DEBUG
    Serial.println(F("SEND Chat Action Message"));
  #endif
  long sttime = millis();

  if (text != "") {
    while (millis() < sttime + 8000) { // loop for a while to send the message
      String command = BOT_CMD("sendChatAction?chat_id=");
      command += chat_id;
      command += F("&action=");
      command += text;

      String response = sendGetToTelegram(command);

      #ifdef TELEGRAM_DEBUG
        Serial.println(response);
      #endif
      sent = checkForOkResponse(response);

      if (sent) break;

    }
  }

  closeClient();
  return sent;
}

void UniversalTelegramBot::closeClient() {
  if (client->connected()) {
    #ifdef TELEGRAM_DEBUG
       Serial.println(F("Closing client"));
    #endif
    client->stop();
  }
}

bool UniversalTelegramBot::getFile(String& file_path, long& file_size, const String&
file_id)
{
  String command = BOT_CMD("getFile?file_id=");
```

```
  command += file_id;
  String response = sendGetToTelegram(command); // receive reply from telegram.org
  DynamicJsonDocument doc(maxMessageLength);
  DeserializationError error = deserializeJson(doc, ZERO_COPY(response));
  closeClient();

  if (!error) {
    if (doc.containsKey("result")) {
      file_path  = F("https://api.telegram.org/file/");
      file_path += buildCommand(doc["result"]["file_path"]);
      file_size = doc["result"]["file_size"].as<long>();
      return true;
    }
  }
  return false;
}

bool UniversalTelegramBot::answerCallbackQuery(const String &query_id, const String
&text, bool show_alert, const String &url, int cache_time) {
  DynamicJsonDocument payload(maxMessageLength);

  payload["callback_query_id"] = query_id;
  payload["show_alert"] = show_alert;
  payload["cache_time"] = cache_time;

  if (text.length() > 0) payload["text"] = text;
  if (url.length() > 0) payload["url"] = url;

  String response = sendPostToTelegram(BOT_CMD("answerCallbackQuery"),
payload.as<JsonObject>());
  #ifdef _debug
    Serial.print(F("answerCallbackQuery response:"));
    Serial.println(response);
  #endif
  bool answer = checkForOkResponse(response);
  closeClient();
  return answer;
}
```

Coding For Importing Library

```
#ifndef UniversalTelegramBot_h
#define UniversalTelegramBot_h

//#define TELEGRAM_DEBUG 0                    //jz
#define ARDUINOJSON_DECODE_UNICODE 1
#define ARDUINOJSON_USE_LONG_LONG 1
#include <Arduino.h>
```

```
#include <ArduinoJson.h>
#include <Client.h>

#define TELEGRAM_HOST "api.telegram.org"
#define TELEGRAM_SSL_PORT 443
#define HANDLE_MESSAGES 1

//unmark following line to enable debug mode
//#define _debug

typedef bool (*MoreDataAvailable)();
typedef byte (*GetNextByte)();
typedef byte* (*GetNextBuffer)();
typedef int (GetNextBufferLen)();

struct telegramMessage {
  String text;
  String chat_id;
  String chat_title;
  String from_id;
  String from_name;
  String date;
  String type;
  String file_caption;
  String file_path;
  String file_name;
  bool hasDocument;
  long file_size;
  float longitude;
  float latitude;
  int update_id;

  int reply_to_message_id;
  String reply_to_text;
  String query_id;
};

class UniversalTelegramBot {
public:
  UniversalTelegramBot(const String& token, Client &client);
  void updateToken(const String& token);
  String getToken();
  String sendGetToTelegram(const String& command);
  String sendPostToTelegram(const String& command, JsonObject payload);
  String
  sendMultipartFormDataToTelegram(const String& command, const String&
binaryPropertyName,
                                  const String& fileName, const String& contentType,
                                  const String& chat_id, int fileSize,
                                  MoreDataAvailable moreDataAvailableCallback,
```

```
                        GetNextByte getNextByteCallback,
                        GetNextBuffer getNextBufferCallback,
                        GetNextBufferLen getNextBufferLenCallback);

//jz caption
  String
  sendMultipartFormDataToTelegramWithCaption(const String& command, const String&
binaryPropertyName,
                        const String& fileName, const String& contentType,
                        const String& caption,
                        const String& chat_id, int fileSize,
                        MoreDataAvailable moreDataAvailableCallback,
                        GetNextByte getNextByteCallback,
                        GetNextBuffer getNextBufferCallback,
                        GetNextBufferLen getNextBufferLenCallback);


  bool readHTTPAnswer(String &body, String &headers);
  bool getMe();

  bool sendSimpleMessage(const String& chat_id, const String& text, const String&
parse_mode);
  bool sendMessage(const String& chat_id, const String& text, const String& parse_mode
= "");
  bool sendMessageWithReplyKeyboard(const String& chat_id, const String& text,
                          const String& parse_mode, const String& keyboard,
                          bool resize = false, bool oneTime = false,
                          bool selective = false);
  bool sendMessageWithInlineKeyboard(const String& chat_id, const String& text,
                           const String& parse_mode, const String& keyboard);

  bool sendChatAction(const String& chat_id, const String& text);

  bool sendPostMessage(JsonObject payload);
  String sendPostPhoto(JsonObject payload);
  String sendPhotoByBinary(const String& chat_id, const String& contentType, int
fileSize,
                    MoreDataAvailable moreDataAvailableCallback,
                    GetNextByte getNextByteCallback,
                    GetNextBuffer getNextBufferCallback,
                    GetNextBufferLen getNextBufferLenCallback);
  String sendPhoto(const String& chat_id, const String& photo, const String& caption = "",
              bool disable_notification = false,
              int reply_to_message_id = 0, const String& keyboard = "");

  bool answerCallbackQuery(const String &query_id,
                    const String &text = "",
                    bool show_alert = false,
                    const String &url = "",
                    int cache_time = 0);
```

```
  bool setMyCommands(const String& commandArray);

  String buildCommand(const String& cmd);

  int getUpdates(long offset);
  bool checkForOkResponse(const String& response);
  telegramMessage messages[HANDLE_MESSAGES];
  long last_message_received;
  String name;
  String userName;
  int longPoll = 0;
  int waitForResponse = 1500;
  int _lastError;
  int last_sent_message_id = 0;
  int maxMessageLength = 1500;
  int jzdelay = 0;  // delay between multipart blocks
  //int jzblocksize = 32 * 512;
#define jzblocksize  32 * 512
  byte buffer[jzblocksize];

private:
  // JsonObject * parseUpdates(String response);
  String _token;
  Client *client;
  void closeClient();
  bool getFile(String& file_path, long& file_size, const String& file_id);
  bool processResult(JsonObject result, int messageIndex);
};

#endif
```


Coding For ESP32-CAM Initialization

```
// ----------------------------
// Standard Libraries - Already Installed if you have ESP32 set up
// ----------------------------

#include <WiFi.h>
#include <WiFiClientSecure.h>
#include "esp_camera.h"

// ----------------------------
// Additional Libraries - each one of these will need to be installed.
// ----------------------------

//#include <UniversalTelegramBot.h>
```

```
#include "UniversalTelegramBot.h"  // use local library which is a modified copy of an old version
// Library for interacting with the Telegram API
// Search for "Telegram" in the Library manager and install
// The universal Telegram library
// https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot

#include <ArduinoJson.h>
// Library used for parsing Json from the API responses
// Search for "Arduino Json" in the Arduino Library manager
// https://github.com/bblanchon/ArduinoJson


static const char vernum[] = "drone-detection-cam 8.9";
String devstr =  "base_station";
int max_frames = 150;
framesize_t configframesize = FRAMESIZE_VGA; // FRAMESIZE_ + QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
int frame_interval = 0;         // 0 = record at full speed, 100 = 100 ms delay between frames
float speed_up_factor = 0.5;        // 1 = play at realtime, 0.5 = slow motion, 10 = speedup 10x
int framesize = FRAMESIZE_VGA; //FRAMESIZE_HD;
int quality = 10;
int qualityconfig = 5;

// Initialize Wifi connection to the router and Telegram BOT

char ssid[] = "sbahri6@unifi";     // your network SSID (name)
char password[] = "0132523694"; // your WiFi network password
// https://sites.google.com/a/usapiens.com/opnode/time-zones  -- find your timezone here
String TIMEZONE = "SGT-8";//your timezone

// you can enter your home chat_id, so the device can send you a reboot message, otherwise it responds to the chat_id talking to telegram

String chat_id = "453141460";//your telegram ID
#define BOTtoken "5960330722:AAGQAFHGsWvpGfgwAt5HTG1C_sKqkNMWiQM"
// your Bot Token (Get from Botfather)

// see here for information about getting free telegram credentials
// https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot
// https://randomnerdtutorials.com/telegram-esp32-motion-detection-arduino/

bool reboot_request = false;

#define CAMERA_MODEL_AI_THINKER
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
```

```
#define SIOC_GPIO_NUM     27

#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

#include "esp_system.h"

bool setupCamera() {
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  //init with high specs to pre-allocate larger buffers
  if (psramFound()) {
    config.frame_size = configframesize;
    config.jpeg_quality = qualityconfig;
    config.fb_count = 4;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }
```

```
  //Serial.printf("Internal Total heap %d, internal Free Heap %d\n", ESP.getHeapSize(),
ESP.getFreeHeap());
  //Serial.printf("SPIRam Total heap   %d, SPIRam Free Heap   %d\n",
ESP.getPsramSize(), ESP.getFreePsram());

  static char * memtmp = (char *) malloc(32 * 1024);
  static char * memtmp2 = (char *) malloc(32 * 1024); //32767

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return false;
  }
  free(memtmp2);
  memtmp2 = NULL;
  free(memtmp);
  memtmp = NULL;
  //Serial.printf("Internal Total heap %d, internal Free Heap %d\n", ESP.getHeapSize(),
ESP.getFreeHeap());
  //Serial.printf("SPIRam Total heap   %d, SPIRam Free Heap   %d\n",
ESP.getPsramSize(), ESP.getFreePsram());

  sensor_t * s = esp_camera_sensor_get();

  //  drop down frame size for higher initial frame rate
  s->set_framesize(s, (framesize_t)framesize);
  s->set_quality(s, quality);
  delay(200);
  return true;
}

#define FLASH_LED_PIN 4

WiFiClientSecure client;
UniversalTelegramBot bot(BOTtoken, client);

int Bot_mtbs = 5000; //mean time between scan messages
long Bot_lasttime;   //last time messages' scan has been done

bool flashState = LOW;

camera_fb_t * fb = NULL;
camera_fb_t * vid_fb = NULL;

TaskHandle_t the_camera_loop_task;
void the_camera_loop (void* pvParameter) ;
static void IRAM_ATTR PIR_ISR(void* arg) ;

bool video_ready = false;
```

```
bool picture_ready = false;
bool active_interupt = false;
bool pir_enabled = false;
bool avi_enabled = false;

int avi_buf_size = 0;
int idx_buf_size = 0;

bool isMoreDataAvailable();


////////////////////////////////  send photo as 512 byte blocks or jzblocksize
int currentByte;
uint8_t* fb_buffer;
size_t fb_length;

bool isMoreDataAvailable() {
  return (fb_length - currentByte);
}

uint8_t getNextByte() {
  currentByte++;
  return (fb_buffer[currentByte - 1]);
}

////////////////////////////////  send avi as 512 byte blocks or jzblocksize
int avi_ptr;
uint8_t* avi_buf;
size_t avi_len;

bool avi_more() {
  return (avi_len - avi_ptr);
}

uint8_t avi_next() {
  avi_ptr++;
  return (avi_buf[avi_ptr - 1]);
}

bool dataAvailable = false;


////////////////////////////////

uint8_t * psram_avi_buf = NULL;
uint8_t * psram_idx_buf = NULL;
uint8_t * psram_avi_ptr = 0;
uint8_t * psram_idx_ptr = 0;
char strftime_buf[64];
```

```
void handleNewMessages(int numNewMessages) {
  //Serial.println("handleNewMessages");
  //Serial.println(String(numNewMessages));

  for (int i = 0; i < numNewMessages; i++) {
    chat_id = String(bot.messages[i].chat_id);
    String text = bot.messages[i].text;

    Serial.printf("\nGot a message %s\n", text);

    String from_name = bot.messages[i].from_name;
    if (from_name == "") from_name = "Guest";

    String hi = "Got: ";
    hi += text;
    bot.sendMessage(chat_id, hi, "Markdown");
    client.setHandshakeTimeout(120000);
    if (text == "/flash") {
      flashState = !flashState;
      digitalWrite(FLASH_LED_PIN, flashState);
    }

    if (text == "/status") {
      String stat = "Device: " + devstr + "\nVer: " + String(vernum) + "\nRssi: " +
String(WiFi.RSSI()) + "\nip: " +  WiFi.localIP().toString() + "\nEnabled: " + pir_enabled +
"\nAvi Enabled: " + avi_enabled;
      if (frame_interval == 0) {
        stat = stat + "\nFast 3 sec";
      } else if (frame_interval == 125) {
        stat = stat + "\nMed 10 sec";
      } else {
        stat = stat + "\nSlow 40 sec";
      }
      stat = stat + "\nQuality: " + quality;

      bot.sendMessage(chat_id, stat, "");
    }

    if (text == "/reboot") {
      reboot_request = true;
    }

    if (text == "/enable") {
      pir_enabled = true;
    }

    if (text == "/disable") {
      pir_enabled = false;
    }
```

```
if (text == "/enavi") {
  avi_enabled = true;
}

if (text == "/disavi") {
  avi_enabled = false;
}

if (text == "/fast") {
  max_frames = 150;
  frame_interval = 0;
  speed_up_factor = 0.5;
  pir_enabled = true;
  avi_enabled = true;
}

if (text == "/med") {
  max_frames = 150;
  frame_interval = 125;
  speed_up_factor = 1;
  pir_enabled = true;
  avi_enabled = true;
}

if (text == "/slow") {
  max_frames = 150;
  frame_interval = 500;
  speed_up_factor = 5;
  pir_enabled = true;
  avi_enabled = true;
}

/*
if (fb) {
  esp_camera_fb_return(fb);
  Serial.println("Return an fb ???");
  if (fb) {
    esp_camera_fb_return(fb);
    Serial.println("Return another fb ?");
  }
}
*/

for (int j = 0; j < 4; j++) {
camera_fb_t * newfb = esp_camera_fb_get();
if (!newfb) {
  Serial.println("Camera Capture Failed");
} else {
  //Serial.print("Pic, len="); Serial.print(newfb->len);
```

```
    //Serial.printf(", new fb %X\n", (long)newfb->buf);
    esp_camera_fb_return(newfb);
    delay(10);
   }
  }
   if ( text == "/photo" || text == "/caption" ) {

    fb = NULL;

    // Take Picture with Camera
    fb = esp_camera_fb_get();
    if (!fb) {
     Serial.println("Camera capture failed");
     bot.sendMessage(chat_id, "Camera capture failed", "");
     return;
    }

    currentByte = 0;
    fb_length = fb->len;
    fb_buffer = fb->buf;

    if (text == "/caption") {

     Serial.println("\n>>>>> Sending with a caption, bytes=  " + String(fb_length));

     String sent = bot.sendMultipartFormDataToTelegramWithCaption("sendPhoto",
"photo", "img.jpg",
                 "image/jpeg", "Your photo", chat_id, fb_length,
                 isMoreDataAvailable, getNextByte, nullptr, nullptr);

     Serial.println("done!");

    } else {

     Serial.println("\n>>>>> Sending, bytes=  " + String(fb_length));

     bot.sendPhotoByBinary(chat_id, "image/jpeg", fb_length,
                   isMoreDataAvailable, getNextByte,
                   nullptr, nullptr);

     dataAvailable = true;

     Serial.println("done!");
    }
    esp_camera_fb_return(fb);
   }

   if (text == "/vga" ) {

    fb = NULL;
```

```
    //sensor_t * s = esp_camera_sensor_get();
    //s->set_framesize(s, FRAMESIZE_VGA);

    Serial.println("\n\n\nSending VGA");

    // Take Picture with Camera
    fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera capture failed");
      bot.sendMessage(chat_id, "Camera capture failed", "");
      return;
    }

    currentByte = 0;
    fb_length = fb->len;
    fb_buffer = fb->buf;

    Serial.println("\n>>>>> Sending as 512 byte blocks, with jzdelay of 0, bytes=  " +
String(fb_length));

    bot.sendPhotoByBinary(chat_id, "image/jpeg", fb_length,
                          isMoreDataAvailable, getNextByte,
                          nullptr, nullptr);

    esp_camera_fb_return(fb);
  }


  if (text == "/clip") {

    // record the video
    bot.longPoll =  0;

    xTaskCreatePinnedToCore( the_camera_loop, "the_camera_loop", 10000, NULL, 1,
&the_camera_loop_task, 1);
    //xTaskCreatePinnedToCore( the_camera_loop, "the_camera_loop", 10000, NULL, 1,
&the_camera_loop_task, 0);  //v8.5

    if ( the_camera_loop_task == NULL ) {
      //vTaskDelete( xHandle );
      Serial.printf("do_the_steaming_task failed to start! %d\n", the_camera_loop_task);
    }
  }

  if (text == "/start") {
    String welcome = "ESP32Cam Telegram bot.\n\n";
    welcome += "/photo: take a photo\n";
    welcome += "/flash: toggle flash LED\n";
    welcome += "/caption: photo with caption\n";
```

```
      welcome += "/clip: short video clip\n";
      welcome += "\n Configure the clip\n";
      welcome += "/enable: enable trigger detector\n";
      welcome += "/disable: disable trigger detector\n";
      welcome += "/enavi: enable avi\n";
      welcome += "/disavi: disable avi\n";
      welcome += "\n/fast: 25 fps - 3  sec - play .5x speed\n";
      welcome += "/med: 8  fps - 10 sec - play 1x speed\n";
      welcome += "/slow: 2  fps - 40 sec - play 5x speed\n";
      welcome += "\n/status: status\n";
      welcome += "/reboot: reboot\n";
      welcome += "/start: start\n";
      bot.sendMessage(chat_id, welcome, "Markdown");
    }
  }
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~
//
// Make the avi functions
//
//   start_avi() - open the file and write headers
//   another_pic_avi() - write one more frame of movie
//   end_avi() - write the final parameters and close the file


char devname[30];

struct tm timeinfo;
time_t now;

camera_fb_t * fb_curr = NULL;
camera_fb_t * fb_next = NULL;

#define fbs 8 // how many kb of static ram for psram -> sram buffer for sd write - not really
used because not dma for sd

char avi_file_name[100];
long avi_start_time = 0;
long avi_end_time = 0;
int start_record = 0;
long current_frame_time;
long last_frame_time;

static int i = 0;
uint16_t frame_cnt = 0;
uint16_t remnant = 0;
uint32_t length = 0;
uint32_t startms;
```

```
uint32_t elapsedms;
uint32_t uVideoLen = 0;

unsigned long movi_size = 0;
unsigned long jpeg_size = 0;
unsigned long idx_offset = 0;

uint8_t zero_buf[4] = {0x00, 0x00, 0x00, 0x00};
uint8_t dc_buf[4] = {0x30, 0x30, 0x64, 0x63};    // "00dc"
uint8_t avi1_buf[4] = {0x41, 0x56, 0x49, 0x31};    // "AVI1"
uint8_t idx1_buf[4] = {0x69, 0x64, 0x78, 0x31};    // "idx1"

struct frameSizeStruct {
  uint8_t frameWidth[2];
  uint8_t frameHeight[2];
};

//  data structure from here https://github.com/s60sc/ESP32-
CAM_MJPEG2SD/blob/master/avi.cpp, extended for ov5640

static const frameSizeStruct frameSizeData[] = {
  {{0x60, 0x00}, {0x60, 0x00}}, // FRAMESIZE_96X96,    // 96x96
  {{0xA0, 0x00}, {0x78, 0x00}}, // FRAMESIZE_QQVGA,    // 160x120
  {{0xB0, 0x00}, {0x90, 0x00}}, // FRAMESIZE_QCIF,     // 176x144
  {{0xF0, 0x00}, {0xB0, 0x00}}, // FRAMESIZE_HQVGA,    // 240x176
  {{0xF0, 0x00}, {0xF0, 0x00}}, // FRAMESIZE_240X240,  // 240x240
  {{0x40, 0x01}, {0xF0, 0x00}}, // FRAMESIZE_QVGA,     // 320x240
  {{0x90, 0x01}, {0x28, 0x01}}, // FRAMESIZE_CIF,      // 400x296
  {{0xE0, 0x01}, {0x40, 0x01}}, // FRAMESIZE_HVGA,     // 480x320
  {{0x80, 0x02}, {0xE0, 0x01}}, // FRAMESIZE_VGA,      // 640x480   8
  {{0x20, 0x03}, {0x58, 0x02}}, // FRAMESIZE_SVGA,     // 800x600   9
  {{0x00, 0x04}, {0x00, 0x03}}, // FRAMESIZE_XGA,      // 1024x768  10
  {{0x00, 0x05}, {0xD0, 0x02}}, // FRAMESIZE_HD,       // 1280x720  11
  {{0x00, 0x05}, {0x00, 0x04}}, // FRAMESIZE_SXGA,     // 1280x1024 12
  {{0x40, 0x06}, {0xB0, 0x04}}, // FRAMESIZE_UXGA,     // 1600x1200 13
  // 3MP Sensors
  {{0x80, 0x07}, {0x38, 0x04}}, // FRAMESIZE_FHD,      // 1920x1080 14
  {{0xD0, 0x02}, {0x00, 0x05}}, // FRAMESIZE_P_HD,     //  720x1280 15
  {{0x60, 0x03}, {0x00, 0x06}}, // FRAMESIZE_P_3MP,    //  864x1536 16
  {{0x00, 0x08}, {0x00, 0x06}}, // FRAMESIZE_QXGA,     // 2048x1536 17
  // 5MP Sensors
  {{0x00, 0x0A}, {0xA0, 0x05}}, // FRAMESIZE_QHD,      // 2560x1440 18
  {{0x00, 0x0A}, {0x40, 0x06}}, // FRAMESIZE_WQXGA,    // 2560x1600 19
  {{0x38, 0x04}, {0x80, 0x07}}, // FRAMESIZE_P_FHD,    // 1080x1920 20
  {{0x00, 0x0A}, {0x80, 0x07}}  // FRAMESIZE_QSXGA,    // 2560x1920 21

};


#define AVIOFFSET 240 // AVI main header length
```

```
uint8_t buf[AVIOFFSET] = {
  0x52, 0x49, 0x46, 0x46, 0xD8, 0x01, 0x0E, 0x00, 0x41, 0x56, 0x49, 0x20, 0x4C, 0x49,
0x53, 0x54,
  0xD0, 0x00, 0x00, 0x00, 0x68, 0x64, 0x72, 0x6C, 0x61, 0x76, 0x69, 0x68, 0x38, 0x00,
0x00, 0x00,
  0xA0, 0x86, 0x01, 0x00, 0x80, 0x66, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00,
0x00, 0x00,
  0x64, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,
  0x80, 0x02, 0x00, 0x00, 0xe0, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x4C, 0x49, 0x53, 0x54, 0x84, 0x00,
0x00, 0x00,
  0x73, 0x74, 0x72, 0x6C, 0x73, 0x74, 0x72, 0x68, 0x30, 0x00, 0x00, 0x00, 0x76, 0x69,
0x64, 0x73,
  0x4D, 0x4A, 0x50, 0x47, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,
  0x01, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0A, 0x00,
0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x73, 0x74,
0x72, 0x66,
  0x28, 0x00, 0x00, 0x00, 0x28, 0x00, 0x00, 0x00, 0x80, 0x02, 0x00, 0x00, 0xe0, 0x01,
0x00, 0x00,
  0x01, 0x00, 0x18, 0x00, 0x4D, 0x4A, 0x50, 0x47, 0x00, 0x84, 0x03, 0x00, 0x00, 0x00,
0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x49, 0x4E,
0x46, 0x4F,
  0x10, 0x00, 0x00, 0x00, 0x6A, 0x61, 0x6D, 0x65, 0x73, 0x7A, 0x61, 0x68, 0x61, 0x72,
0x79, 0x20,
  0x76, 0x38, 0x38, 0x20, 0x4C, 0x49, 0x53, 0x54, 0x00, 0x01, 0x0E, 0x00, 0x6D, 0x6F,
0x76, 0x69,
};


//
// Writes an uint32_t in Big Endian at current file position
//
static void inline print_quartet(unsigned long i, uint8_t * fd) {
  uint8_t y[4];
  y[0] = i % 0x100;
  y[1] = (i >> 8) % 0x100;
  y[2] = (i >> 16) % 0x100;
  y[3] = (i >> 24) % 0x100;
  memcpy( fd, y, 4);
}

//
// Writes 2 uint32_t in Big Endian at current file position
//
```

```
static void inline print_2quartet(unsigned long i, unsigned long j, uint8_t * fd) {
  uint8_t y[8];
  y[0] = i % 0x100;
  y[1] = (i >> 8) % 0x100;
  y[2] = (i >> 16) % 0x100;
  y[3] = (i >> 24) % 0x100;
  y[4] = j % 0x100;
  y[5] = (j >> 8) % 0x100;
  y[6] = (j >> 16) % 0x100;
  y[7] = (j >> 24) % 0x100;
  memcpy( fd, y, 8);
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~
//
//  get_good_jpeg()  - take a picture and make sure it has a good jpeg
//
camera_fb_t *  get_good_jpeg() {

  camera_fb_t * fb;

  long start;
  int failures = 0;

  do {
    int fblen = 0;
    int foundffd9 = 0;

    fb = esp_camera_fb_get();

    if (!fb) {
      Serial.println("Camera Capture Failed");
      failures++;
    } else {
      int get_fail = 0;
      fblen = fb->len;

      for (int j = 1; j <= 1025; j++) {
        if (fb->buf[fblen - j] != 0xD9) {
        } else {
          if (fb->buf[fblen - j - 1] == 0xFF ) {
            foundffd9 = 1;
            break;
          }
        }
      }

      if (!foundffd9) {
        Serial.printf("Bad jpeg, Frame %d, Len = %d \n", frame_cnt, fblen);
```

```
      esp_camera_fb_return(fb);
      failures++;
     } else {
      break;
     }
    }

  } while (failures < 10);   // normally leave the loop with a break()

  // if we get 10 bad frames in a row, then quality parameters are too high - set them lower
  if (failures == 10) {
    Serial.printf("10 failures");
    sensor_t * ss = esp_camera_sensor_get();
    int qual = ss->status.quality ;
    ss->set_quality(ss, qual + 3);
    quality = qual + 3;
    Serial.printf("\n\nDecreasing quality due to frame failures %d -> %d\n\n", qual, qual +
5);
    delay(1000);
  }
  return fb;
}


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// the_camera_loop()

void the_camera_loop (void* pvParameter) {

  vid_fb = get_good_jpeg(); // esp_camera_fb_get();
  if (!vid_fb) {
    Serial.println("Camera capture failed");
    //bot.sendMessage(chat_id, "Camera capture failed", "");
    return;
  }
  picture_ready = true;

  if (avi_enabled) {
    frame_cnt = 0;

    ////////////////////////////// start a movie
    avi_start_time = millis();
    Serial.printf("\nStart the avi ... at %d\n", avi_start_time);
    Serial.printf("Framesize %d, quality %d, length %d seconds\n\n", framesize, quality,
max_frames * frame_interval / 1000);

    fb_next = get_good_jpeg();                  // should take zero time
```

```
  last_frame_time = millis();
  start_avi();

  ////////////////////////////// all the frames of movie

  for (int j = 0; j < max_frames - 1 ; j++) { // max_frames
    current_frame_time = millis();

    if (current_frame_time - last_frame_time < frame_interval) {
      if (frame_cnt < 5 || frame_cnt > (max_frames - 5) )Serial.printf("frame %d, delay
%d\n", frame_cnt, (int) frame_interval - (current_frame_time - last_frame_time));
      delay(frame_interval - (current_frame_time - last_frame_time));            // delay for
timelapse
    }

    last_frame_time = millis();
    frame_cnt++;

    if (frame_cnt !=  1) esp_camera_fb_return(fb_curr);
    fb_curr = fb_next;          // we will write a frame, and get the camera preparing a new
one

    another_save_avi(fb_curr );
    fb_next = get_good_jpeg();            // should take near zero, unless the sd is faster than
the camera, when we will have to wait for the camera

    digitalWrite(33, frame_cnt % 2);
    if (movi_size > avi_buf_size * .95) break;
  }

  ////////////////////////////// stop a movie
  Serial.println("End the Avi");

  esp_camera_fb_return(fb_curr);
  frame_cnt++;
  fb_curr = fb_next;
  fb_next = NULL;
  another_save_avi(fb_curr );
  digitalWrite(33, frame_cnt % 2);
  esp_camera_fb_return(fb_curr);
  fb_curr = NULL;
  end_avi();                          // end the movie
  digitalWrite(33, HIGH);         // light off
  avi_end_time = millis();
  float fps = 1.0 * frame_cnt / ((avi_end_time - avi_start_time) / 1000) ;
  Serial.printf("End the avi at %d.  It was %d frames, %d ms at %.2f fps...\n", millis(),
frame_cnt, avi_end_time - avi_start_time, fps);
  frame_cnt = 0;            // start recording again on the next loop
  video_ready = true;
 }
```

```
  Serial.println("Deleting the camera task");
  delay(100);
  vTaskDelete(the_camera_loop_task);
}


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~
//
// start_avi - open the files and write in headers
//

void start_avi() {

  Serial.println("Starting an avi ");

  time(&now);
  localtime_r(&now, &timeinfo);
  strftime(strftime_buf, sizeof(strftime_buf), "Cam from Base Station %F
%H.%M.%S.avi", &timeinfo);

  //memset(psram_avi_buf, 0, avi_buf_size);  // save some time
  //memset(psram_idx_buf, 0, idx_buf_size);

  psram_avi_ptr = 0;
  psram_idx_ptr = 0;

  memcpy(buf + 0x40, frameSizeData[framesize].frameWidth, 2);
  memcpy(buf + 0xA8, frameSizeData[framesize].frameWidth, 2);
  memcpy(buf + 0x44, frameSizeData[framesize].frameHeight, 2);
  memcpy(buf + 0xAC, frameSizeData[framesize].frameHeight, 2);

  psram_avi_ptr = psram_avi_buf;
  psram_idx_ptr = psram_idx_buf;

  memcpy( psram_avi_ptr, buf, AVIOFFSET);
  psram_avi_ptr += AVIOFFSET;

  startms = millis();

  jpeg_size = 0;
  movi_size = 0;
  uVideoLen = 0;
  idx_offset = 4;

} // end of start avi

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~
//
```

```
//  another_save_avi saves another frame to the avi file, uodates index
//          -- pass in a fb pointer to the frame to add
//

void another_save_avi(camera_fb_t * fb ) {

  int fblen;
  fblen = fb->len;

  int fb_block_length;
  uint8_t* fb_block_start;

  jpeg_size = fblen;

  remnant = (4 - (jpeg_size & 0x00000003)) & 0x00000003;

  long bw = millis();
  long frame_write_start = millis();

  memcpy(psram_avi_ptr, dc_buf, 4);

  int jpeg_size_rem = jpeg_size + remnant;

  print_quartet(jpeg_size_rem, psram_avi_ptr + 4);

  fb_block_start = fb->buf;

  if (fblen > fbs * 1024 - 8 ) {                  // fbs is the size of frame buffer static
    fb_block_length = fbs * 1024;
    fblen = fblen - (fbs * 1024 - 8);
    memcpy( psram_avi_ptr + 8, fb_block_start, fb_block_length - 8);
    fb_block_start = fb_block_start + fb_block_length - 8;
  } else {
    fb_block_length = fblen + 8  + remnant;
    memcpy( psram_avi_ptr + 8, fb_block_start, fb_block_length - 8);
    fblen = 0;
  }

  psram_avi_ptr += fb_block_length;

  while (fblen > 0) {
    if (fblen > fbs * 1024) {
      fb_block_length = fbs * 1024;
      fblen = fblen - fb_block_length;
    } else {
      fb_block_length = fblen  + remnant;
      fblen = 0;
    }

    memcpy( psram_avi_ptr, fb_block_start, fb_block_length);
```

```
    psram_avi_ptr += fb_block_length;

    fb_block_start = fb_block_start + fb_block_length;
   }

   movi_size += jpeg_size;
   uVideoLen += jpeg_size;

   print_2quartet(idx_offset, jpeg_size, psram_idx_ptr);
   psram_idx_ptr += 8;

   idx_offset = idx_offset + jpeg_size + remnant + 8;

   movi_size = movi_size + remnant;

} // end of another_pic_avi

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~
//
//  end_avi writes the index, and closes the files
//

void end_avi() {

  Serial.println("End of avi - closing the files");

  if (frame_cnt <  5 ) {
    Serial.println("Recording screwed up, less than 5 frames, forget index\n");
  } else {

    elapsedms = millis() - startms;

    float fRealFPS = (1000.0f * (float)frame_cnt) / ((float)elapsedms) * speed_up_factor;

    float fmicroseconds_per_frame = 1000000.0f / fRealFPS;
    uint8_t iAttainedFPS = round(fRealFPS) ;
    uint32_t us_per_frame = round(fmicroseconds_per_frame);

    //Modify the MJPEG header from the beginning of the file, overwriting various
placeholders

    print_quartet(movi_size + 240 + 16 * frame_cnt + 8 * frame_cnt, psram_avi_buf + 4);
    print_quartet(us_per_frame, psram_avi_buf + 0x20);

    unsigned long max_bytes_per_sec = (1.0f * movi_size * iAttainedFPS) / frame_cnt;
    print_quartet(max_bytes_per_sec, psram_avi_buf + 0x24);
    print_quartet(frame_cnt, psram_avi_buf + 0x30);
    print_quartet(frame_cnt, psram_avi_buf + 0x8c);
```

```
    print_quartet((int)iAttainedFPS, psram_avi_buf + 0x84);
    print_quartet(movi_size + frame_cnt * 8 + 4, psram_avi_buf + 0xe8);

    Serial.println(F("\n*** Video recorded and saved ***\n"));

    Serial.printf("Recorded %5d frames in %5d seconds\n", frame_cnt, elapsedms / 1000);
    Serial.printf("File size is %u bytes\n", movi_size + 12 * frame_cnt + 4);
    Serial.printf("Adjusted FPS is %5.2f\n", fRealFPS);
    Serial.printf("Max data rate is %lu bytes/s\n", max_bytes_per_sec);
    Serial.printf("Frame duration is %d us\n", us_per_frame);
    Serial.printf("Average frame length is %d bytes\n", uVideoLen / frame_cnt);


    Serial.printf("Writng the index, %d frames\n", frame_cnt);

    memcpy (psram_avi_ptr, idx1_buf, 4);
    psram_avi_ptr += 4;

    print_quartet(frame_cnt * 16, psram_avi_ptr);
    psram_avi_ptr += 4;

    psram_idx_ptr = psram_idx_buf;

    for (int i = 0; i < frame_cnt; i++) {
      memcpy (psram_avi_ptr, dc_buf, 4);
      psram_avi_ptr += 4;
      memcpy (psram_avi_ptr, zero_buf, 4);
      psram_avi_ptr += 4;

      memcpy (psram_avi_ptr, psram_idx_ptr, 8);
      psram_avi_ptr += 8;
      psram_idx_ptr += 8;
    }
  }

  Serial.println("---");
  digitalWrite(33, HIGH);
}
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~
//
// setup some interupts during reboot
//
//  int read13 = digitalRead(13); -- pir for video

int PIRpin = 13;

static void setupinterrupts() {

//  pinMode(PIRpin, INPUT_PULLDOWN) ; //INPUT_PULLDOWN);
```

```
  pinMode(PIRpin,INPUT);

  Serial.print("Setup PIRpin = ");
  for (int i = 0; i < 5; i++) {
    Serial.print( digitalRead(PIRpin) ); Serial.print(", ");
  }
  Serial.println(" ");

  esp_err_t err = gpio_isr_handler_add((gpio_num_t)PIRpin, &PIR_ISR, NULL);

  if (err != ESP_OK) Serial.printf("gpio_isr_handler_add failed (%x)", err);
  gpio_set_intr_type((gpio_num_t)PIRpin, GPIO_INTR_POSEDGE);


}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~
//
//  PIR_ISR - interupt handler for PIR  - starts or extends a video
//
static void IRAM_ATTR PIR_ISR(void* arg) {

//  int PIRstatus = digitalRead(PIRpin) + digitalRead(PIRpin) + digitalRead(PIRpin) ;
  int PIRstatus = digitalRead(PIRpin);
//  if (PIRstatus == 3)
  if (PIRstatus == 1){
    Serial.print("PIR Interupt>> "); Serial.println(PIRstatus);

    if (!active_interupt && pir_enabled) {
      active_interupt = true;
      digitalWrite(33, HIGH);
      Serial.print("PIR Interupt ... start recording ... ");
      xTaskCreatePinnedToCore( the_camera_loop, "the_camera_loop", 10000, NULL, 1,
&the_camera_loop_task, 1);
      //xTaskCreatePinnedToCore( the_camera_loop, "the_camera_loop", 10000, NULL, 1,
&the_camera_loop_task, 0);  //v8.5

      if ( the_camera_loop_task == NULL ) {
        Serial.printf("do_the_steaming_task failed to start! %d\n", the_camera_loop_task);
      }
    }
  }
}

#include "esp_wifi.h"
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include <ESPmDNS.h>
```

```
bool init_wifi() {
  uint32_t brown_reg_temp = READ_PERI_REG(RTC_CNTL_BROWN_OUT_REG);
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  // Attempt to connect to Wifi network:
  Serial.print("Connecting Wifi: ");
  Serial.println(ssid);

  // Set WiFi to station mode and disconnect from an AP if it was Previously connected
  devstr.toCharArray(devname, devstr.length() + 1);       // name of your camera for
mDNS, Router, and filenames
  WiFi.mode(WIFI_STA);
  WiFi.setHostname(devname);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }

  wifi_ps_type_t the_type;

  //esp_err_t get_ps = esp_wifi_get_ps(&the_type);

  esp_err_t set_ps = esp_wifi_set_ps(WIFI_PS_NONE);

  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, brown_reg_temp);

  configTime(0, 0, "pool.ntp.org");
  char tzchar[80];
  TIMEZONE.toCharArray(tzchar, TIMEZONE.length());       // name of your camera for
mDNS, Router, and filenames
  setenv("TZ", tzchar, 1);  // mountain time zone from #define at top
  tzset();

  if (!MDNS.begin(devname)) {
    Serial.println("Error setting up MDNS responder!");
    return false;
  } else {
    Serial.printf("mDNS responder started '%s'\n", devname);
  }
  time(&now);

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  return true;
}
```

```
////////////////////////////////////////////////////////////////////////////////

void setup() {
  Serial.begin(115200);
  Serial.println("--------------------------------");
  Serial.printf("ESP32-CAM Video-Telegram %s\n", vernum);
  Serial.println("--------------------------------");

  pinMode(FLASH_LED_PIN, OUTPUT);
  digitalWrite(FLASH_LED_PIN, flashState); //defaults to low

  pinMode(12, INPUT_PULLUP);        // pull this down to stop recording

  pinMode(33, OUTPUT);              // little red led on back of chip
  digitalWrite(33, LOW);            // turn on the red LED on the back of chip

  avi_buf_size = 3000 * 1024; // = 3000 kb = 60 * 50 * 1024;
  idx_buf_size = 200 * 10 + 20;
  psram_avi_buf = (uint8_t*)ps_malloc(avi_buf_size);
  if (psram_avi_buf == 0) Serial.printf("psram_avi allocation failed\n");
  psram_idx_buf = (uint8_t*)ps_malloc(idx_buf_size); // save file in psram
  if (psram_idx_buf == 0) Serial.printf("psram_idx allocation failed\n");

  if (!setupCamera()) {
    Serial.println("Camera Setup Failed!");
    while (true) {
      delay(100);
    }
  }

  for (int j = 0; j < 7; j++) {
    camera_fb_t * fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Camera Capture Failed");
    } else {
      Serial.print("Pic, len="); Serial.print(fb->len);
      Serial.printf(", new fb %X\n", (long)fb->buf);
      esp_camera_fb_return(fb);
      delay(50);
    }
  }

  bool wifi_status = init_wifi();

  // Make the bot wait for a new message for up to 60seconds
  //bot.longPoll = 60;
  bot.longPoll = 5;

  client.setInsecure();
```

```
  setupinterrupts();

  String stat = "Reboot\nDevice: " + devstr + "\nVer: " + String(vernum) + "\nRssi: " +
String(WiFi.RSSI()) + "\nip: " +  WiFi.localIP().toString() + "\n/start";
  bot.sendMessage(chat_id, stat, "");

  pir_enabled = true;
  avi_enabled = true;
  digitalWrite(33, HIGH);
}

int loopcount = 0;


void loop() {
  loopcount++;

  client.setHandshakeTimeout(120000); // workaround for esp32-arduino 2.02 bug
https://github.com/witnessmenow/Universal-Arduino-Telegram-
Bot/issues/270#issuecomment-1003795884

  if (reboot_request) {
    String stat = "Rebooting on request\nDevice: " + devstr + "\nVer: " + String(vernum) +
"\nRssi: " + String(WiFi.RSSI()) + "\nip: " +  WiFi.localIP().toString() ;
    bot.sendMessage(chat_id, stat, "");
    delay(10000);
    ESP.restart();
  }

  if (picture_ready) {
    picture_ready = false;
    send_the_picture();
  }

  if (video_ready) {
    video_ready = false;
    send_the_video();
  }

  if (millis() > Bot_lasttime + Bot_mtbs )  {

    if (WiFi.status() != WL_CONNECTED) {
      Serial.println("***** WiFi reconnect *****");
      WiFi.reconnect();
      delay(5000);
      if (WiFi.status() != WL_CONNECTED) {
        Serial.println("***** WiFi rerestart *****");
        init_wifi();
      }
    }
```

```
    int numNewMessages = bot.getUpdates(bot.last_message_received + 1);

    while (numNewMessages) {
      //Serial.println("got response");
      handleNewMessages(numNewMessages);
      numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    }
    Bot_lasttime = millis();
  }
  if(Serial.available()>0)
  {
    int v=Serial.read();
    if(v==49)
    {
      Serial.println(v);
      xTaskCreatePinnedToCore( the_camera_loop, "the_camera_loop", 10000, NULL, 1,
&the_camera_loop_task, 1);
    }

  }
}


void send_the_picture() {
  digitalWrite(33, LOW);        // light on
  currentByte = 0;
  fb_length = vid_fb->len;
  fb_buffer = vid_fb->buf;

  Serial.println("\n>>>>> Sending as 512 byte blocks, with jzdelay of 0, bytes=  " +
String(fb_length));

  if (active_interupt) {
    String sent = bot.sendMultipartFormDataToTelegramWithCaption("sendPhoto",
"photo", "img.jpg",
                  "image/jpeg", "Drone Event!", chat_id, fb_length,
                  isMoreDataAvailable, getNextByte, nullptr, nullptr);
  } else {
    String sent = bot.sendMultipartFormDataToTelegramWithCaption("sendPhoto",
"photo", "img.jpg",
                  "image/jpeg", "Telegram Request", chat_id, fb_length,
                  isMoreDataAvailable, getNextByte, nullptr, nullptr);
  }
  esp_camera_fb_return(vid_fb);
  bot.longPoll =  0;
  digitalWrite(33, HIGH);         // light oFF
  if (!avi_enabled) active_interupt = false;
}
```

```
void send_the_video() {
  digitalWrite(33, LOW);         // light on
  Serial.println("\n\n\nSending clip with caption");
  Serial.println("\n>>>>> Sending as 512 byte blocks, with a caption, and with jzdelay of
0, bytes=  " + String(psram_avi_ptr - psram_avi_buf));
  avi_buf = psram_avi_buf;

  avi_ptr = 0;
  avi_len = psram_avi_ptr - psram_avi_buf;

  String sent2 = bot.sendMultipartFormDataToTelegramWithCaption("sendDocument",
"document", strftime_buf,
              "image/jpeg", "Drone Detected!", chat_id, psram_avi_ptr - psram_avi_buf,
              avi_more, avi_next, nullptr, nullptr);

  Serial.println("done!");
  digitalWrite(33, HIGH);         // light off

  bot.longPoll = 5;
  active_interupt = false;
}
```

Gant Chart PSM 1

PROJECT ACTIVITY /TASK	WEEK													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Project Briefing	X													
Research project	X													
Background, Problem statement & Objective		X	X	X										
Identify component			X	X	X	X	X	X						
Project flow chart						X	X	X		X	X			
Methodology						X	X	X		X	X			
Review report						X	X	X		X	X	X		
Submit report												X	X	
Presentation														X

Gant Chart PSM 2

PROJECT ACTIVITY /TASK	WEEK													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Preparing hardware	X	X	X	X	X	X	X	X	X					
Researching code				X	X	X	X	X	X	X	X			
Testing Prototype						X	X	X	X	X	X			
Troubleshoot Project						X	X	X	X	X	X	X		
Project Hardware Planning						X	X	X	X	X	X			
Project Design Planning						X	X	X	X	X	X			
Review report						X	X	X	X	X	X			
Final draft submission													X	
Submit PSM 2 Report Panel														X
Presentation														X