TECHNICAL UNIVERSITY OF CRETE

DIPLOMA THESIS

Embedded Gimbal System for Land-based Tracking of Unmanned Aerial Vehicles

---

Author:
Athanasios MANESIS

Thesis Committee:
Prof. Apostolos DOLLAS
Prof. Sotirios IOANNIDIS
Prof. Panagiotis PARTSINEVELOS
(MRED TUC)

A thesis submitted in fulfillment of the requirements
for the diploma of Electrical and Computer Engineer

in the

School of Electrical and Computer Engineering
Microprocessor and Hardware Laboratory

October 4, 2024

TECHNICAL UNIVERSITY OF CRETE

Abstract

School of Electrical and Computer Engineering

Electrical and Computer Engineer

Embedded Gimbal System for Land-based Tracking of Unmanned Aerial Vehicles

by Athanasios MANESIS

Embedded systems are one of the most important and rapidly developing fields in engineering. They find application from small household appliances to critical airplane safety systems that have hundreds of them to perform functions such as in-flight entertainment systems, temperature control, speed control, flight management systems, flight data recorders, engine control and many more. With the use of artificial intelligence and machine learning algorithms these systems become more intelligent, efficient and have much more autonomy than in the past. In this thesis we studied and implemented such an embedded system that has the ability to detect and track UAVs (drone) through optical recognition. Detection and monitoring is visible via a monitor which is connected to the embedded system and has three different modes, one for horizontal scanning, one for vertical scanning and one for tracking. The embedded system consists of a rotation mechanism (PTZ) in which a camera is mounted and an RPi-4 which through the camera recognizes the drone using a machine learning model for object recognition and generates the appropriate angles to the gimbal to turn in that way so the drone stays as close to the center of the camera as possible. Communication between the RPi and the PTZ device is achieved via RS485 using the Pelco-P and Pelco-D protocols that the rotation mechanism supports. To convert the angles into the appropriate Pelco-P and Pelco-D frames, a python library was developed so that by using the corresponding functions, the appropriate messages can be produced without anyone needing to know exactly how these frames are created.

TECHNICAL UNIVERSITY OF CRETE

Abstract

School of Electrical and Computer Engineering

Electrical and Computer Engineer

Embedded Gimbal System for Land-based Tracking of Unmanned Aerial Vehicles

by Athanasios MANESIS

Τα ενσωματωμένα συστήματα είναι ένα ταχέως αναπτυσσόμενο πεδίο στην επιστήμη των Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών. Βρίσκουν εφαρμογή από μικρές οικιακές συσκευές έως κρίσιμα συστήματα ασφάλειας αεροπλάνων στα οποία εκτελούν λειτουργίες όπως έλεγχος θερμοκρασίας και ταχύτητας, έλεγχος κινητήρα, κλπ. Με τη χρήση αλγορίθμων τεχνητής νοημοσύνης και μηχανικής μάθησης αυτά τα συστήματα γίνονται πιο αποτελεσματικά και με συνεχώς αυξανόμενη αυτονομία. Στην παρούσα διπλωματική εργασία μλετήσαμε και υλοποιήσαμε ένα τέτοιο ενσωματωμένο σύστημα που έχει τη δυνατότητα να ανιχνεύει και να παρακολουθεί μη επανδρωμένο εναέριο όχημα μέσω οπτικής αναγνώρισης. Η ανίχνευση και η παρακολούθηση είναι ορατή μέσω μιας οθόνης που συνδέεται με το ενσωματωμένο σύστημα. Υπάρχουν τρεις διαφορετικές λειτουργίες, μία για οριζόντια σάρωση, μία για κάθετη σάρωση και μία για παρακολούθηση. Το ενσωματωμένο σύστημα αποτελείται από έναν μηχανισμό περιστροφής (PTZ) στον οποίο είναι τοποθετημένη μια κάμερα και ένα RPi-4 που μέσω της κάμερας αναγνωρίζει το drone χρησιμοποιώντας ένα μοντέλο μηχανικής μάθησης για την αναγνώριση αντικειμένων και δημιουργεί τις κατάλληλες γωνίες προς τον μηχανισμό στρέψης για να στρέφεται με κατάλληλο τρόπο ώστε το drone να παραμένει όσο το δυνατόν πιο κοντά στο κέντρο της κάμερας. Η επικοινωνία μεταξύ του RPi και της συσκευής PTZ επιτυγχάνεται μέσω του RS485 χρησιμοποιώντας τα πρωτόκολλα Pelco-P και Pelco-D που υποστηρίζει ο μηχανισμός στρέψης. Για την μετατροπή των γωνιών στα κατάλληλα Pelco-P και Pelco-D μυνήματα, αναπτύχθηκε μια βιβλιοθήκη python , ώστε με τη χρήση των αντίστοιχων συναρτήσεων, να παράγονται τα κατάλληλα μηνύματα χωρίς παρέμβαση του χρήστη.

Acknowledgements

First of all, I would like to thank my supervisor, Prof. Apostolos Dollas, for his teaching work on the areas related to Hardware Architecture throughout my studies as well as for his guidance in my thesis.

I would also like to thank my thesis committee, Prof. Panagiotis Parcinevelos, and Prof. Sotirios Ioannidis, for evaluating my thesis, as well as the head of the Microprocessor and Hardware Laboratory (MHL), Mr. Markos Kimionis, for being available for any problem I had regarding the equipment.

Finally, I would like to thank my family who supported me throughout my studies and the friends and colleagues we worked with during our studies at the Technical University of Crete.

My sincere thanks to all of you.

Contents

List of Figures

List of Tables

List of Algorithms

Dedicated to my family and friends…

Chapter 1

Introduction

An embedded system is a device combining hardware and software which performs dedicated functions. Found in consumer electronics, process control systems, aircraft, in-car systems and many other applications, they need to be extremely reliable, but because of their small size and limited resources, present challenges for designers and developers. With growth and advancements in the field of electronics, wireless communications, networking, cognitive and affective computing and robotics, devices around us communicate in more ways than we ever imagined. Those times are not very distant when every object around us will have a small processor/sensor embedded within itself, invisible to us but still communicating with all other devices around, making our lives more connected and accessible than ever before.

1.1 Motivation

There are many new applications that embedded systems play a key role, such as Internet of Things, Autonomous Vehicles, Ubiquitous computing, Smart Agriculture and many more. The possibilities they offer are huge as they are low cost devices, consist of microcontrollers, single-board computers and sensors, designed so that they can operate independently and in special applications in real time. With the parallel development of other fields such as communications and artificial intelligence, their use will increase. Already most appliances we have in our home, eg a refrigerator or air-condition has an embedded system, or a car has dozens of them to provide information to the driver or improve vehicle safety, e.g. abs.

Also through this thesis, protocols that are used in the industry will be implemented, and they are useful to be familiar to an Embedded Engineer, in our case Pelco-D and Pelco-P for handling the Gimbal, as well as it is a very

good opportunity to learn how to build a prototype of a product, even if it is not in final form for professional use.

All these make embedded systems a very attractive and creative field, since the applications are countless and the designer can build simple or more complex automations based on his interests.

1.2 Scientific Goals & Contributions

In this thesis, we will explore the use of an embedded system to automate the movement of a rotational base (gimbal) to detect and track a UAV (e.g., drone) within its optical field. The automation of the gimbal's movement will be accomplished through image processing and AI/ML algorithms. Through this process, we will gain insights into techniques for solving similar problems and undertaking projects in which embedded systems are useful.

A variety of industries are experiencing innovation and advancement due to the integration of embedded systems and AI. Many AI applications rely on embedded systems, which are computer systems built into other products or devices. They offer the physical capabilities and control required. On the other hand, AI offers the decision-making capabilities that increase these systems' autonomy and effectiveness.

The contribution of this thesis is that we saw how we can implement a low-cost prototype embedded system that performs the desired functions, how we implement protocols related to the handling of industrial devices such as the AS20RS485 rotation-mechanism/gimbal and how we dealt with errors or failures that we had not foreseen according to the initial design of the embedded system.

1.3 Thesis Outline

- **Chapter 2 - Background Concepts:** The theoretical concepts of Microcontrollers, Single Board Computers (SBC), protocols and devices that used or rejected for the embedded system implementation of this thesis.
- **Chapter 3 - Related Work:** Example of systems, with similar functions or using similar technologies, from industry or academia.

- **Chapter 4 - System Design and Implementation:** This chapter describes the system design process and the choice of technologies that implement the subsystems for tracking and detection.
- **Chapter 5 - Verification and Results:** Presentation of the results of the final system and comparison with the requirements that were set in the initial description of the system.
- **Chapter 6 - Conclusions and Future Work:** This work is being concluded, and directions for future work and possible extensions are given.

Chapter 2

Background Concepts

2.1 Gimbals (Rotation Base)

Gimbals are mechanical devices that enable rotation of an object around a specific axis. They typically consist of a pivoted support that can be a single gimbal or a set of three gimbals, each mounted one on top of the other with orthogonal pivot axes. This design allows an object mounted on the innermost gimbal to remain independent of the rotation of its support. The history of gimbals can be traced back to ancient times, for example, in ancient Greece, the philosopher Archimedes is known for studying the principles of similar devices, and in the Middle Ages, gimbals were utilized for navigation to keep instruments such as the astrolabe and the mariner's compass level on ships. In modern times, gimbal technology is widely employed in various fields such as inertial navigation, rocket engines, marine chronometers, photography and videography, due to its ability to provide precise and accurate movement, stabilization, and control. [1]

2.2 Microcontrollers (ATMEGA16)

A microcontroller is a small computer that contains one or more processors, memory and I/O peripherals. Microcontrollers is very useful for products and devices like control system for automobile, medical devices, power tools and other embedded systems. They have low power consumption, some of them can operate at frequencies as low as 4kHZ and they have the ability to retain their functionality while waiting for an event, eg interrupts. The first single-chip microcontroller was the Intel 4004, released on a single MOS LSI chip in 1971.

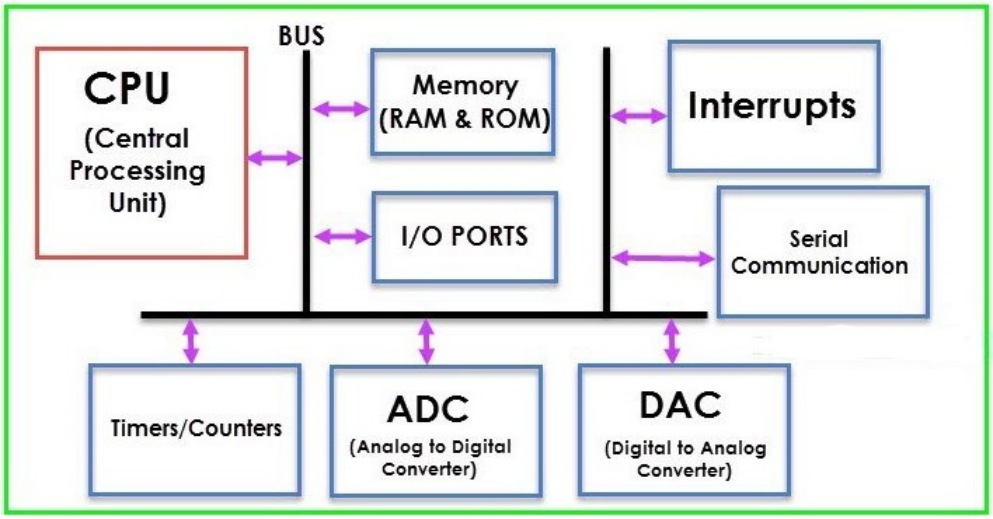

FIGURE 2.1: Microcontroller Basic Structure URL

More specifically, a microcontroller consists by CPU, Program Memory (ROM), Data Memory (RAM), Timers-Counters, I/O Ports, Serial Communication Interface, Clock Circuit and Interrupt Mechanism. Modern microcontrollers also have Serial Peripheral Interface (SPI), Inter Integrated Circuit (I2C), Analog/Digital to Digital/Analog Converter (ADC/DAC), Control Area Network (CAN), Universal Serial Bus (USB) and even more peripherals. The integration of these features on the same chip CPU makes it more efficient and cheaper than to use seperate chips. Below we can see the basic structure of a microntroller.

In our case we will use a microcontroller of the AVR family. AVR developed since 1996 by Atmel and they are available in three categories.

AVR Categories			
Series Name	Pins	Flash Memory	Special Feature
TinyAVR	6-32	0.5-8 KB	Small in size
MegaAVR	28-100	4-256 KB	Extended peripherals
XmegaAVR	44-100	16-384 KB	DMA , Event System included

TABLE 2.1: AVR Categories

The AVR microcontrollers are based on RISC architecture and have 32 8-bit general purpose registers. In RISC architecture the instruction set are fewer

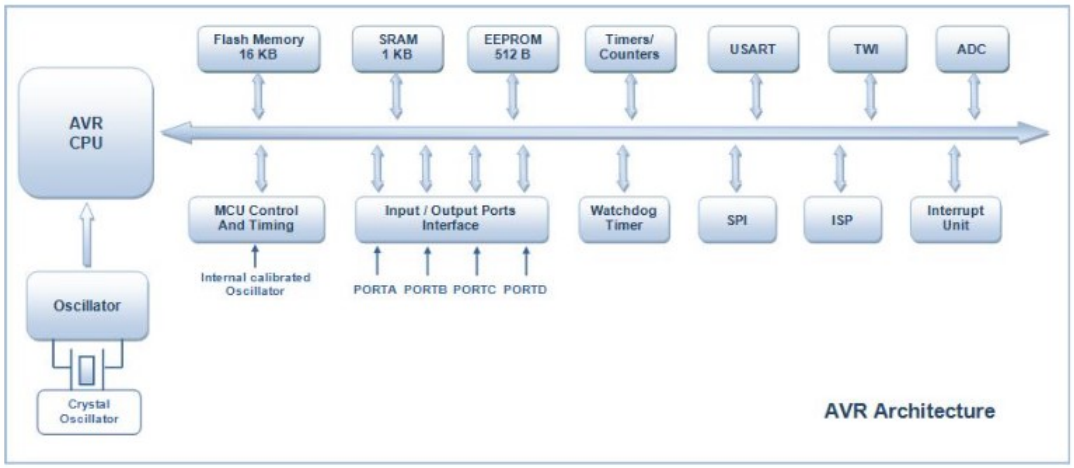

FIGURE 2.2: AVR Architecture of a 16KB microcontroller URL

in number and simpler and faster in operation.ALU can take inputs from two of these registers, perform the requested operation and write back the result to an arbitrary register in a single clock cycle.[2] Below we can see a block diagram explaining the AVR architecture of a 16KB microcontroller

In AVR architecture the memory is separated in three different sections, the Flash EEPROM, the Byte Addressable EEPROM and the SRAM. Flash EEPROM or flash memory is used to store the program dumped on the microcontroller and it is non-volatile. Byte Addressable EEPROM is used to store data such as variables value and it is also non-volatile. SRAM offers portions for the general purpose registers used by the microcontroller's CPU and peripheral subsystems and is volatile. Timers and Counters are useful for generating precision actions such as creating delays between two operation. Watchdog Timer continuously monitors and resets the control if the code gets stuck for more than a defined time interval. Interrupts can be internal or external for peripherals support. [3]

2.3 Raspberry Pi

The Raspberry Pi (Rpi) is a small, low-cost, and powerful single-board computer that has become a popular choice for embedded systems projects. Embedded systems are specialized computer systems that are integrated into other devices or products, and the Raspberry Pi's small form factor, low cost, and powerful capabilities make it a great option for these types of projects.

Some examples of embedded systems applications for the Raspberry Pi include industrial automation, robotics, home automation, Internet of Things (IoT) devices, digital signage, smart cities, and medical devices. The Raspberry Pi can be used to control and monitor industrial equipment, control the movement and sensor data of robots, connect and control various IoT devices, create and run digital signage, collect and process data from sensors and cameras in smart cities, and develop medical devices and applications.
In terms of technical specifications, the Raspberry Pi 4 Model B 4GB (RPi4) is equipped with a Broadcom BCM2711 64-bit quad-core ARM Cortex-A72 CPU, 4GB RAM, and a powerful VideoCore VI GPU. It features various interfaces, including Ethernet, dual HDMI, USB 3.0, and GPIO pins, enabling connections to different devices and peripherals like cameras, displays, and sensors. This versatility allows the RPi4 to run a full-featured operating system such as Linux, empowering it to perform tasks that would typically demand a more potent and costly computer.
The RPi4 boasts 40 General Purpose Input/Output (GPIO) pins, providing a broad spectrum of expansion possibilities. These pins can be utilized to connect sensors, motors, and other devices to the board. They are configurable as inputs or outputs and can be controlled by software to read the state of external devices or to control them. Additionally, the RPi4 comes with built-in Wi-Fi and Bluetooth, enabling seamless wireless connectivity and communication with other devices.

2.4 RPi4 - AtMega16 Comparison

The Raspberry Pi 4 (Rpi4) is a suitable choice for a UAV detection and tracking project as compared to AtMega16 microcontroller. The Rpi3 offers more computational power and flexibility for image processing and AI/ML algorithms, making it the best choice for the UAV detection and tracking application. The RPi4's built-in Ethernet, USB, and 40 GPIO pins also allow for connecting a camera and other sensors, as well as controlling and communicating with the gimbal and other systems. Additionally, the Rpi4 also has the ability to provide a live video feed through its built-in HDMI and USB interfaces and it also has the capability to send notifications or alerts through email or other messaging platforms when a UAV is detected thanks to its built-in Ethernet and WiFi capabilities.
On the other hand, while the AtMega16 may be suitable for low power consumption and high precision control systems, it lacks the computational

power and flexibility that is necessary for processing images and detecting UAVs. The AtMega16 is an 8-bit microcontroller that is based on the AVR architecture and operates at a clock speed of up to 16 MHz and has 16 kB of flash memory, 1 kB of SRAM, and various peripheral interfaces like Timers, UART, SPI, I2C, and ADC. It is ideal for applications that require low power consumption and high precision, such as control systems, sensor networks, and IoT devices.
The Rpi4 is a more suitable option for the UAV detection and tracking project as it offers more computational power and flexibility, as well as the ability to provide a live video feed and send notifications or alerts. While the AtMega16 may be suitable for some low power consumption and precision control applications, it lacks the necessary computational power and flexibility for image processing and AI/ML algorithms that are critical for detecting and tracking UAVs.

2.5 Pelco D and Pelco P Protocols

Pelco D and Pelco P are two communication protocols used for controlling pan-tilt-zoom (PTZ) cameras, commonly used in the security and surveillance industry. Both protocols are based on RS-485 serial communication and are designed to control the movements and various functions of PTZ cameras.
Pelco D is an ASCII-based protocol that uses a series of command and data bytes to control the camera's movements. It defines a set of commands for controlling the pan, tilt, zoom, and focus of the camera, as well as for setting preset positions and activating various camera functions such as night mode and autofocus. The command bytes consist of two characters, the first of which is always the letter 'A' and the second is a letter or number that represents the specific command. Data bytes consist of two characters, the first of which is a letter or number that represents the specific data value, and the second is a number that represents the value of the data. The format of a Pelco D command message is as follows:

Pelco-D Format	
Field	Value
Start byte	0xFF
Address byte	0x00 - 0xFF
Command 1	1 byte
Command 2	1 byte
Data 1 (optional)	1 byte (optional)

TABLE 2.2: Pelco D Format

Pelco D also supports extended commands, these commands are used to control additional camera functions such as backlight compensation, white balance, and privacy masking. The extended commands are similar to the standard commands, but are preceded by an extended command byte (0x00) and followed by additional data bytes.
Pelco P is also an RS-485 based protocol and it's similar to Pelco D, but it uses a slightly different command structure and data format. The format of a Pelco P command message is as follows:

Pelco-D Format	
Field	Value
Start byte	0xFF
Address byte	0x00 - 0xFF
Command	1 byte
Data 1	1 byte
Data 2	1 byte
Stop byte	0x00

TABLE 2.3: Pelco P Format

Pelco P commands are sent in hexadecimal format, unlike Pelco D which is sent as ASCII characters. Also, Pelco P commands include a checksum byte at the end, which is used to validate the integrity of the command. Unlike Pelco D, Pelco P does not support extended commands and it has a different set of commands for controlling the camera's functions.

In summary, Pelco D and Pelco P are both widely used protocols for controlling PTZ cameras, they are both based on RS-485 serial communication

and are commonly used in the security and surveillance industry. Pelco D is an ASCII-based protocol that uses a series of command and data bytes to control the camera's movements and it also supports extended commands. Pelco P is similar to Pelco D but it uses a slightly different command structure and data format, it doesn't support extended commands and it has a different set of commands for controlling the camera's functions. It's important to check the compatibility of the camera with the control system before using any protocol.

Chapter 3

Related Work

This chapter explores the various applications of gimbal systems in different fields, including aerospace, industrial engineering, military, medical, consumer electronics, photography and filmmaking, agriculture, and marine. These applications demonstrate the versatility and effectiveness of gimbal systems in providing precise and accurate movement, stabilization, and control.
It is important to note that many of the applications presented in this chapter are from industrial or military systems, and therefore, the technical information available may be limited.

3.1 Gimbal Systems in Consumer Electronics

In the field of consumer electronics, gimbal systems are utilized to enhance the functionality and user experience of devices such as smartphones and action cameras. These systems employ advanced control algorithms and motors to counteract unwanted movements and vibrations, resulting in smooth and stable footage, even while in motion.
Smartphones, for instance, are now equipped with gimbal systems to provide users with the ability to capture high-quality images and videos. This is particularly useful in situations where a steady shot is needed, such as during the recording of videos or capturing images while in motion. Additionally, gimbal systems have also been integrated into action cameras, allowing for even more versatile use, such as in extreme sports or during outdoor activities.
Furthermore, gimbal systems are also used in virtual and augmented reality systems, to provide smooth and realistic head and hand tracking. This allows for a more immersive and interactive experience for the user.

3.2 Gimbal Systems for Medical Diagnostics and Therapy

Gimbal systems are also used in the field of medical applications, such as rehabilitation, surgery, and diagnostics.
In the field of medical applications, gimbal systems are used to provide stability and precision in various applications. One of the most notable applications is in rehabilitation, specifically in the treatment of conditions such as Parkinson's disease. In this field, gimbal systems are used to stabilize the movement of patients, allowing for more precise and effective treatment. For example, gimbal-controlled mechanical spoons have been developed to help Parkinson's patients with hand tremors to eat independently.
In the field of surgery, gimbal systems are used to stabilize cameras and sensors, providing clear and stable images and data during procedures. They are also used to provide precision in robotic-assisted surgeries, allowing for greater accuracy and control during the procedure. This is particularly beneficial in minimally invasive surgeries, where the use of gimbal systems allows for a smaller incision and less trauma to the patient.
Gimbal systems are also utilized in diagnostic applications, such as endoscopy, where they are used to stabilize the camera and provide clear and stable images and data. This allows for more accurate diagnosis and treatment.

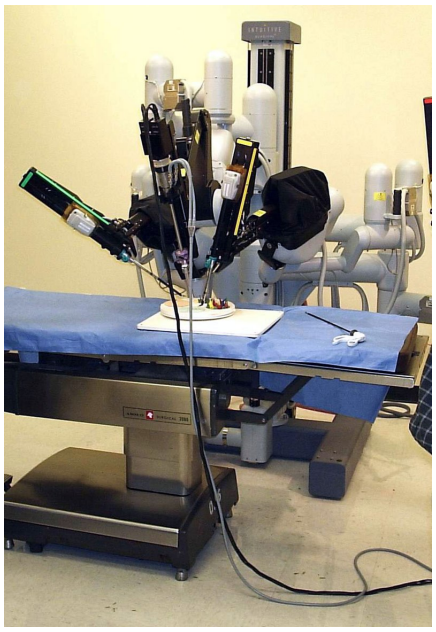

FIGURE 3.1: Da Vinci Surgical System by Intuitive Surgical URL

Overall, gimbal systems play a vital role in the field of medical applications, providing stability and precision in various applications and allowing for more precise and effective treatment and diagnosis.
Some real-world applications of gimbal systems in medical applications include the da Vinci Surgical System developed by Intuitive Surgical, which is a robotic surgical system that uses a gimbal system to control the movement of its instruments, and the ReoGo developed by Reha Technology, which is a gimbal-controlled mechanical spoon that is used to help Parkinson's patients with hand tremors to eat independently. The gimbal system allows for precise and accurate movement of the spoon, making it useful for tasks such as eating.

3.3 Gimbal Systems for Aerospace Navigation and Control

Aerospace Engineering is one of the fields that greatly benefits from the use of gimbal systems. These systems are used for a variety of applications such as stabilization of cameras and sensors on drones and spacecraft, attitude control of satellites, and pointing of antennas and communication systems. Gimbal systems are used to stabilize cameras and sensors on drones and spacecraft, allowing them to capture stable and clear images and data despite the movement of the vehicle. The use of gimbal systems in drones and spacecraft is critical for capturing high-quality images and data that can be used for a wide range of applications, such as surveying, cartography, and inspection of structures. The stabilization gimbal system works by using motors and sensors to actively counteract the movement of the vehicle and maintain the desired orientation of the camera. They typically use a combination of accelerometers, gyroscopes, and magnetometers to sense the movement of the vehicle and adjust the position of the camera accordingly.
Attitude control of satellites is another application of gimbal systems. In this application, gimbal systems are used to control the pointing of the satellite's antennas, solar panels, and other devices. Attitude control systems use a combination of sensors and actuators to sense the satellite's orientation and adjust it as necessary. This is important for maintaining communication and power for the satellite, and for pointing scientific instruments or cameras at a specific location on the Earth.

Lastly, gimbal systems are also used in the pointing of antennas and communication systems. This is the process of aligning an antenna with a specific location on the Earth. This allows the antenna to communicate with a specific satellite or ground station. Gimbal systems are used to point the antenna in the correct direction, and to keep it pointed in that direction despite the movement of the vehicle or the satellite.
These applications of gimbal systems in aerospace engineering demonstrate their importance in capturing clear and stable images and data, maintaining communication and power of the satellite, and keeping antenna pointed in the correct direction for communication purposes.
Some real-world applications of gimbal systems in aerospace include the Mars Helicopter developed by NASA's Jet Propulsion Laboratory [4], which uses a gimbal system to stabilize its camera and capture clear and stable images of the Martian surface, and the Space Shuttle's Remote Manipulator System (RMS), which uses a gimbal system to control the movement of the robotic arm and gripper, allowing for precise and accurate movement.

FIGURE 3.2: Mars Helicopter developed by NASA's Jet Propulsion Laboratory URL

3.4 Gimbal Systems in Industrial Engineering

Gimbal systems are also widely used in the field of Industrial Engineering to achieve precise control and stability in various applications. One of the most common applications is the stabilization of cameras and sensors on industrial robots and cranes. This allows for capturing clear and stable images and data, even in challenging environments, such as in manufacturing plants or construction sites. The gimbal system uses advanced control algorithms and motors to counteract the movement of the robot or crane and maintain the desired orientation of the camera.

Another application of gimbal systems in industrial engineering is the positioning and alignment of machine tools and welding equipment. This is the process of aligning a machine tool or welding torch with a specific location on a workpiece. This allows the machine tool or welding torch to perform its operation in the correct location and with the correct orientation. Gimbal systems are used to position and align the machine tool or welding torch, and to keep it in the correct position and alignment despite the movement of the machine or the workpiece. This allows for achieving precision and accuracy in industrial processes, such as welding, cutting, and drilling, resulting in higher efficiency and quality of the final product.

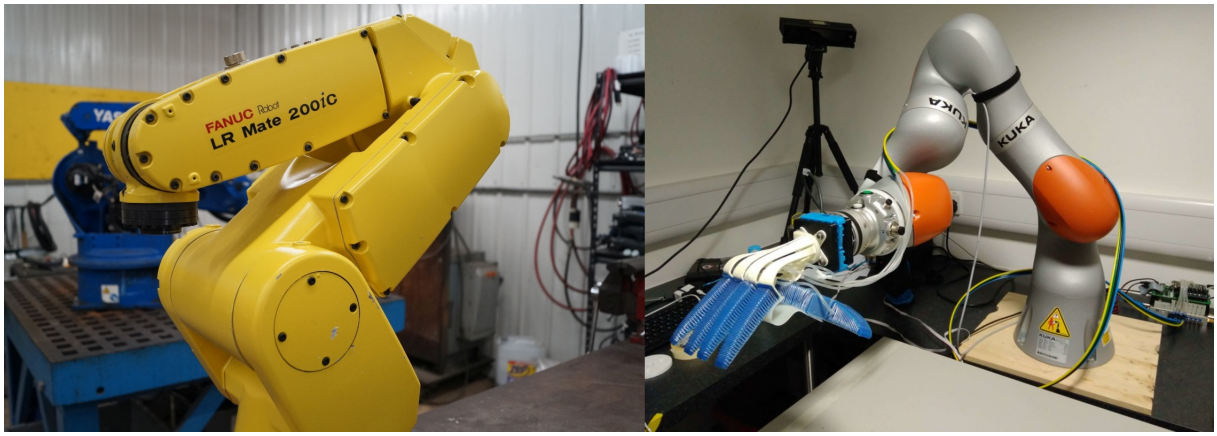

FIGURE 3.3: FANUC LR Mate 200iC (left) and KUKA LBR iiwa (right). URL

Some real-world applications of gimbal systems in industrial engineering include the FANUC LR Mate 200iC robot [5], which is equipped with a gimbal system that allows it to rotate its camera and sensors in any direction, and the KUKA LBR iiwa robot [6], which uses a gimbal system to control its movement, allowing for precise and accurate movement.

3.5 Military Applications of Gimbal Systems

Gimbal systems are utilized in various military applications such as stabilizing cameras and sensors on military vehicles and aircrafts, aiming weapons, and in surveillance and reconnaissance.
The stabilization of cameras and sensors on military vehicles and aircrafts is essential for capturing clear and stable images and data in dynamic and challenging environments, such as during combat or surveillance missions. Gimbal systems are specifically designed to counteract the movement of the

vehicle and maintain the desired orientation of the camera, providing stable and precise imagery.
In addition, gimbal systems are employed in weapons systems to ensure accuracy during aiming while in motion. They are used to point the weapon in the correct direction, and maintain its alignment despite the movement of the vehicle or the target, resulting in greater precision and efficiency in combat.
Lastly, surveillance and reconnaissance systems also utilize gimbal systems to capture clear and stable images and data from a distance. This information is vital for military operations, providing situational awareness and intelligence to the command center, for example for border control, maritime surveillance, among others.

FIGURE 3.4: EO/IR Sensor Turret developed by Lockheed Martin (left) and RQ-21A Blackjack developed by Insitu (right) URL

Some real-world applications of gimbal systems in military technology include the EO/IR Sensor Turret developed by Lockheed Martin, which uses a gimbal system to stabilize its cameras and sensors, and the RQ-21A Blackjack developed by Insitu [7], which is an unmanned aerial vehicle (UAV) that uses a gimbal system to stabilize its cameras and sensors, making it useful for tasks such as surveillance and reconnaissance.

3.6 Gimbal Systems in Agriculture

Gimbal systems are also used in the field of renewable energy, specifically in the tracking of solar panels and wind turbines. They are utilized to optimize

the energy production by aligning the solar panels and wind turbines with the sun or wind direction."
Solar tracking systems are designed to adjust the tilt and azimuth angle of photovoltaic (PV) panels to align them with the sun's position in the sky throughout the day. This results in a significant increase in the amount of electricity generated by the solar panels, as the PV cells are able to receive more direct sunlight.
The gimbal systems used in solar tracking are typically composed of motors, sensors, and control algorithms that work together to adjust the position of the solar panels in real-time. The motors are responsible for physically moving the solar panels, while the sensors, such as encoders, potentiometers, and inclinometers, are used to measure the position and orientation of the panels. The control algorithms then use this information to determine the optimal position for the panels to align them with the sun.
Wind turbine tracking systems are designed to adjust the pitch angle of the blades to align them with the wind direction throughout the day. This results in a significant increase in the amount of electricity generated by the wind turbine, as the blades are able to receive more direct wind.
The gimbal systems used in wind turbine tracking are also typically composed of motors, sensors, and control algorithms that work together to adjust the position of the turbine blades in real-time. The motors are responsible for physically moving the blades, while the sensors, such as wind vanes and anemometers, are used to measure the wind direction and speed. The control algorithms then use this information to determine the optimal position for the blades to align them with the wind.
In summary, gimbal systems are critical in the field of renewable energy, providing stability and accuracy in the tracking of solar panels and wind turbines. This allows for greater efficiency and energy production, making it an important technology in the effort to move towards more sustainable energy sources.

3.7 Marine Applications of Gimbal Systems

Gimbal systems are also used in the field of maritime and naval engineering. They are utilized to stabilize cameras and sensors on ships and submarines, position and align sonar systems, and point and stabilize navigation and communication systems.

The stabilization of cameras and sensors on ships and submarines is necessary for capturing clear and stable images and data in the challenging marine environment. Gimbal systems are engineered to counteract the movement of the vessel and maintain the desired orientation of the camera, providing stable and accurate imagery.
Positioning and aligning sonar systems is another key application of gimbal systems in maritime and naval engineering. This allows for accurate detection and localization of underwater objects, such as ships, submarines, and mines. Gimbal systems are used to position and align the sonar system and keep it in the correct position and alignment despite the movement of the vessel.
Lastly, gimbal systems are also used in navigation and communication systems on ships and submarines. They are used to point and stabilize antennas and other devices, allowing for accurate communication and navigation even in rough seas. This is crucial for the safety and operation of the vessel.
One of the real-world applications of gimbal systems in marine technology is the ROTA project developed by Technical University of Crete [8]. The project aims to provide fast network connections for ships navigating through densely populated island areas, like the Aegean sea, by utilizing gimbal-controlled directional antennas on ships and stationary antennas on the islands. This approach creates a Virtual Private Network (VPN) by connecting the stationary antennas on the islands to existing high-speed computer networks, and it is more cost-effective than traditional satellite communication methods.

3.8 Photography and Filmmaking Applications of Gimbal Systems

The field of cinematography and photography also makes extensive use of gimbal systems. These systems are used to stabilize cameras and ensure smooth and steady shots.
The stabilization of cameras is essential for capturing clear and stable images and footage in various environments. Gimbal systems use advanced control algorithms and motors to counteract the movement of the camera and maintain the desired orientation, resulting in smooth and stable footage. This feature is particularly beneficial in situations that require a steady shot, such as during the filming of movies, commercials, and music videos.

Furthermore, gimbal systems are also utilized in aerial photography and videography, enabling photographers and videographers to capture clear and stable images and footage from drones and other aerial platforms. This allows for a new perspective and a wider range of possibilities.
In summary, gimbal systems play a vital role in the field of cinematography and photography, providing stability and accuracy for capturing smooth and steady shots, resulting in high-quality images and footage.

3.9 Similar Work and Papers

3.9.1 Flying Objects Detection from a Single Moving Camera

The paper "Flying Objects Detection from a Single Moving Camera" by Artem Rozantseva, Vincent Lepetit, and Pascal Fua [9] proposes a method for detecting flying objects in the sky using a single moving camera. The proposed method involves first estimating the camera motion using visual odometry techniques, which involves tracking features in the image and estimating the camera's movement based on the motion of those features. The camera motion estimate is then used to predict the position of the sky region in the next frame of the video. The authors then use a background subtraction technique to remove the static background from the image, leaving only the moving objects. They then apply a motion filter to the remaining pixels to remove noise and further refine the flying object detection. The authors evaluate their method on a dataset of aerial videos and compare their results to other state-of-the-art flying object detection methods. Their method outperforms the other methods in terms of detection accuracy and computational efficiency. Overall, the proposed method provides an effective and efficient way to detect flying objects in the sky using a single moving camera, which has potential applications in various fields such as surveillance, meteorology, and drone detection.

3.9.2 Moving target tracking method for UAV/UGV based on AprilTags

The paper "Moving target tracking method for unmanned aerial vehicle/unmanned ground vehicle heterogeneous system based on AprilTags" by Xiao Liang and Guodong Chen [10] proposes an innovative solution to the challenge of tracking moving targets using a UAV-UGV heterogenous

system. The authors utilize AprilTags, which are easy-to-detect visual markers, to track the target and estimate its position and velocity.The proposed method addresses the problem of occlusion, which can occur when the target moves behind an obstacle and cannot be detected by the camera. The authors use a particle filter to handle such situations and ensure that the target is continuously tracked even when the AprilTag is temporarily lost.One of the advantages of the proposed method is its computational efficiency. The authors use a Kalman filter to estimate the target's position and velocity, which is a widely used technique in tracking applications. The use of AprilTags simplifies the detection process and reduces the computation required for tracking.
The authors evaluate their method using a dataset of moving targets and compare their results to other state-of-the-art tracking methods. Their method outperforms the other methods in terms of tracking accuracy, particularly in situations where the target is occluded or the camera view is obstructed.
Overall, the proposed method has potential applications in various fields such as surveillance, search and rescue, and precision agriculture. The use of a UAV-UGV heterogenous system enables the tracking of moving targets over a large area and provides more comprehensive surveillance compared to using a single platform. The use of AprilTags simplifies the detection process and reduces computation, making the method efficient and practical for real-world applications.

3.9.3 Low-Cost RPi-Based UAS Detection and Classification System

The paper "Low-Cost Raspberry-Pi-Based UAS Detection and Classification System Using Machine Learning" by Carolyn J. Swinney and John C. Woods [11] presents a low-cost system for detecting and classifying unmanned aerial systems (UAS) using machine learning techniques and a Raspberry Pi. The proposed system consists of a Raspberry Pi connected to a low-cost radio frequency (RF) receiver and a machine learning algorithm. The RF receiver detects the signals emitted by the UAS and sends the data to the Raspberry Pi, which processes the data and uses a machine learning algorithm to classify the UAS based on its characteristics. The authors evaluate their system using a dataset of UAS signals and compare their results to other state-of-the-art UAS detection systems. Their system performs well in terms of detection

and classification accuracy, and is significantly more cost-effective compared to other systems.
One of the advantages of the proposed system is its low cost, which makes it accessible to a wider range of users and allows for deployment in a variety of settings. Additionally, the use of machine learning algorithms allows for improved accuracy and flexibility in detecting and classifying UAS. The authors suggest that the proposed system has potential applications in various fields such as security, defense, and law enforcement, as well as in research and hobbyist settings. The system could be used for detecting and tracking UAS in sensitive areas or for studying the behavior of UAS in different environments.

3.9.4 Edge device based Military Vehicle Detection and Classification from UAV

The detection and recognition of military vehicles from images or video frames using unmanned aircraft systems is a significant concern for defense agencies. Identifying and classifying military vehicles from resource-constrained devices integrated with intelligent object detection algorithms can provide significant support. However, there is currently no openly available dataset with various military classes. To address this issue, the authors propose a dataset comprising 6772 images with classes such as military trucks, tanks, aircraft, helicopters, civilian cars, and civilian aircraft. They trained quantized SSD Mobilenet v2 and Tiny Yolo v3 deep learning models on the dataset and compared their performance on resource-constrained edge devices. The results showed that Tiny Yolo v3 performed well and was highly efficient, making it suitable for real-time classification of military and civilian vehicles using edge devices over UAVs. The paper also provides a mathematical calculation for determining the number of flight paths and frames required for surveillance in a given area using available hardware specifications. [12]

Chapter 4

System Design and Implementation

Object detection and tracking are challenging problems in computer vision that involve identifying objects of interest in an image or video and then tracking their movement over time. With the advancement of deep learning techniques, object detection and tracking have become more accurate and efficient.

4.1 High Level System Design

A general structure of the system is shown below. Essentially, we need a subsystem that will do the detection and a subsystem that, using the data from the detection, will move the camera accordingly in order to center the object we are tracking on the camera.

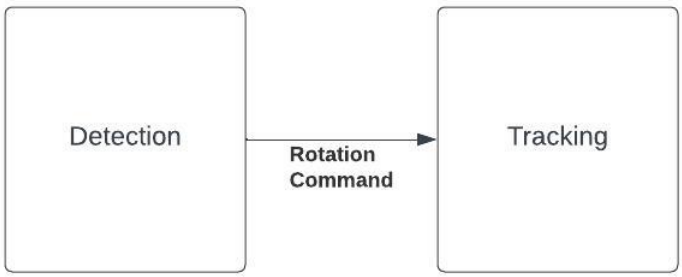

FIGURE 4.1: Top-Level System Block Diagram

In more detail in the block diagram below we can see more specifically the functionality of the subsystems we will need. Initially for the control we will need a microcontroller or a single-board computer, which will communicate with a rotation mechanism and will also be responsible for locating the drone

through the camera. To verify the correct operation and also for monitoring needs, there must be a terminal so that we can see live the image of the drone we are monitoring.

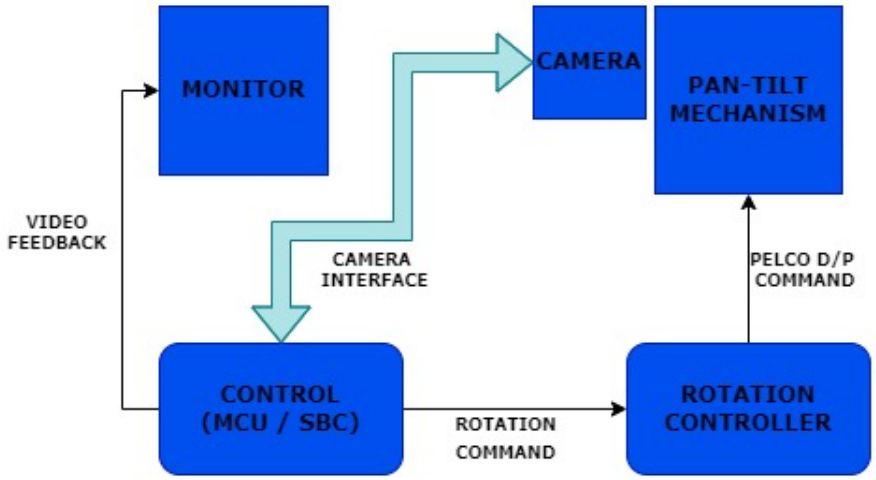

FIGURE 4.2: Sub-Systems Diagram

The rotation command will essentially be a vertical and horizontal rotation angle, so it will need to be formatted according to the Pelco-P and Pelco-D protocol frames. In order to convert the angle into Pelco-P or Pelco-D, a suitable library should be developed so that by calling simple functions the conversion can be done, which is important so that different PTZ devices can be used without affecting the system's functionality.

4.2 Keeping Objects in the Camera Center

Algorithm 1 Calculate Gimbal Rotation Angles

Object's bounding box position (x, y) and image dimensions (w, h) Gimbal rotation angles θ_v and θ_h

Initialize:

$\theta_v \leftarrow 0$; $\theta_h \leftarrow 0$;

Calculate vertical angle:

$\Delta_y \leftarrow y - \frac{h}{2}$; $\theta_v \leftarrow \arctan(\frac{\Delta_y}{h/2})$; $\theta_v \leftarrow \theta_v \cdot \frac{180}{\pi}$;

Calculate horizontal angle:

$\Delta_x \leftarrow x - \frac{w}{2}$; $\theta_h \leftarrow \arctan(\frac{\Delta_x}{w/2})$; $\theta_h \leftarrow \theta_h \cdot \frac{180}{\pi}$;

Output:

return θ_v, θ_h;

To calculate the rotation angle needed to keep the detected object in the center of the camera using a gimbal, we need to first determine the distance of the object from the center of the camera. Let's assume that the camera's field of view (FOV) is rectangular with width W and height H. The center of the camera is then located at the point $(W/2, H/2)$.
Next, we need to find the location of the detected object in the camera's image. Let's assume that the object is represented by a bounding box with its top-left corner at the pixel coordinates (x_1, y_1) and its bottom-right corner at (x_2, y_2). We can then calculate the center of the bounding box as the midpoint coordinates between points (x_1, y_1) and (x_2, y_2) are given by:

$$\left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right).$$

To determine the distance of the object from the center of the camera, we can calculate the horizontal and vertical offsets of the object from the camera center as follows:

$$dx = \frac{x_1 + x_2}{2} - \frac{W}{2} \tag{4.1}$$

$$dy = \frac{y_1 + y_2}{2} - \frac{H}{2} \tag{4.2}$$

We can then calculate the distance d from the object to the camera center using the Pythagorean theorem:

$$d = \sqrt{dx^2 + dy^2} \tag{4.3}$$

Finally, we can calculate the rotation angle needed to keep the object in the center of the camera by taking the arc-tangent of the vertical and horizontal offsets:

$$\theta_h = \text{atan2}(dx, W/2) \tag{4.4}$$

$$\theta_v = \text{atan2}(dy, H/2) \tag{4.5}$$

Note that theta_h and theta_v are the horizontal and vertical rotation angles, respectively, and are measured in radians, so they need to be converted to degrees.

$$\theta_h^{\text{Deg}} = \theta_h \cdot \frac{180}{\pi} \tag{4.6}$$

$$\theta_v^{\text{Deg}} = \theta_v \cdot \frac{180}{\pi} \tag{4.7}$$

4.3 Subsystems Implementation

The system consists of two subsystems: the detection subsystem, which is responsible for identifying objects of interest in a live video stream from a Raspberry Pi camera, and the tracking subsystem, which is responsible for tracking the movement of these objects using a gimbal.

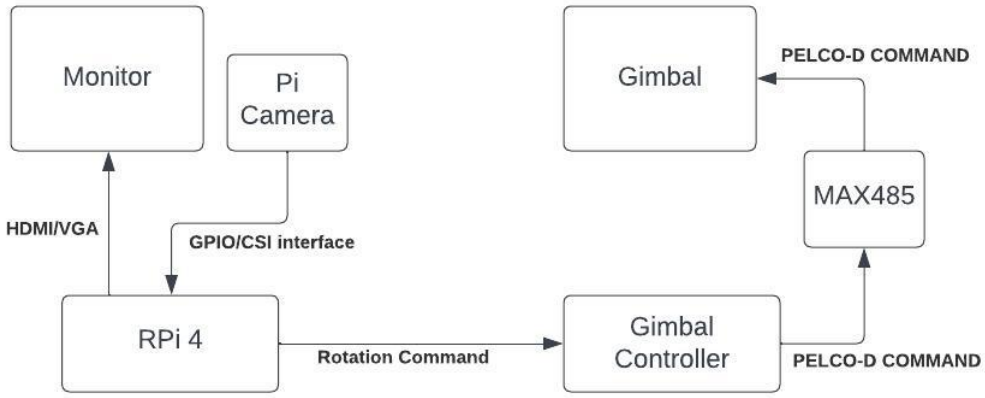

FIGURE 4.3: Sub-Systems Diagram Implementation with RPi

In the Detection section, we will cover the process of collecting and annotating a dataset, training a Tensorflow Lite model, and deploying the model on a Raspberry Pi 4 to perform object detection in real-time. In the Tracking section, we will shift our focus to the tracking subsystem, where we will use the detection results from the Raspberry Pi to control the gimbal and track the movement of the detected objects. By combining these two subsystems, we can create a comprehensive object detection and tracking system.

4.4 Detection

4.4.1 Dataset Collection and Annotation

Dataset Script

To train an effective object detection model, creating a high-quality dataset with accurate labels is crucial. In my project, I am creating my own dataset of images that include my drone. This involves capturing images of my drone from various angles, distances, and environments to make my dataset diverse and representative of real-world scenarios.
To create the dataset, I wrote a script using OpenCV to capture pictures. The script simplifies the process of gathering images for training a machine learning vision model and can be used by anyone, regardless of the camera they use.
To execute the script, initiate a command terminal, navigate to the directory where AutoPictureForTrainingData.py [13] is located, and enter the following command: python3 AutoPictureForTrainingData.py
Upon execution, a window displaying the camera's view will appear. Press the 'p' key on the keyboard to capture an image. The initial image will be stored in a folder named "Pics" with the filename Pics1.jpg. If the "Pics" folder is absent, it will be automatically generated. Continue pressing 'p' to capture additional images, each saved as Pics2.jpg, Pics3.jpg, and so forth. Press 'q' to exit the program.
In the event that Pics1.jpg already exists in the "Pics" folder, the program will dynamically increment the picture number until it identifies a filename that does not exist. For instance, if Pics1.jpg through Pics20.jpg are present in the "Pics" folder, the program will commence with Pics21.jpg for the first saved picture. This approach ensures seamless picture-taking without concerns about overwriting existing images.
By default, the script operates at a resolution of 1280x720, saves pictures in a folder named "Pics," and uses "Pics" as the base filename. You can modify these defaults using the following command arguments:

`-res`: Specify the camera resolution in WxH to adjust the resolution.

`-imgdir`: Specify the folder to save pictures (and serve as the base filename).

For instance, to capture images at a resolution of 1920x1080 and store them

in a folder named "Birds," issue the following command:

```
python3 AutoPictureForTrainingData.py -res=1920x1080
-imgdir=DroneDataset
```

Annotation Tools

To label the images, I am using labeling tools such as LabelImg. With LabelImg, I am drawing bounding boxes around the drone in each image and assigning a label to each box. Accuracy and precision are paramount in this step to ensure the labels are as tight as possible around the drone. This will help avoid any false positives or negatives during the training process.
Once I have labeled all my images, I will split my dataset into two parts: a training set and a validation set. The training set will be used to train the object detection model, while the validation set will be used to evaluate the performance of the model during training. It is essential to have a balanced representation of images that include the drone in both sets.

4.4.2 Model Training and Evaluation

First, I needed to train a custom object detection model to detect objects of interest in images. After researching available options, I decided to use TensorFlow Lite.
To start, I gathered and labeled training images for my model. This involved identifying objects of interest in images and drawing bounding boxes around them. I then used a tool like LabelImg to create an XML file for each image, which contained information about the location and class of each object.
Next, I installed the necessary dependencies for TensorFlow Object Detection, including TensorFlow 2, protobuf, and COCO API. This allowed me to use pre-built scripts to convert my labeled images into the format required for training a TensorFlow Lite object detection model.
Once the dependencies were installed, I uploaded my image dataset to Colab and split it into separate folders for training, validation, and testing. This helped me evaluate the performance of my model on unseen data. I also created a labelmap file and used a script to convert my labeled images into TFRecord format, which is required for training a TensorFlow Lite model.
After setting up the training data, I configured the training process by creating a pipeline configuration file that defined the architecture of my model, the location of my training data, and various other hyperparameters. I chose

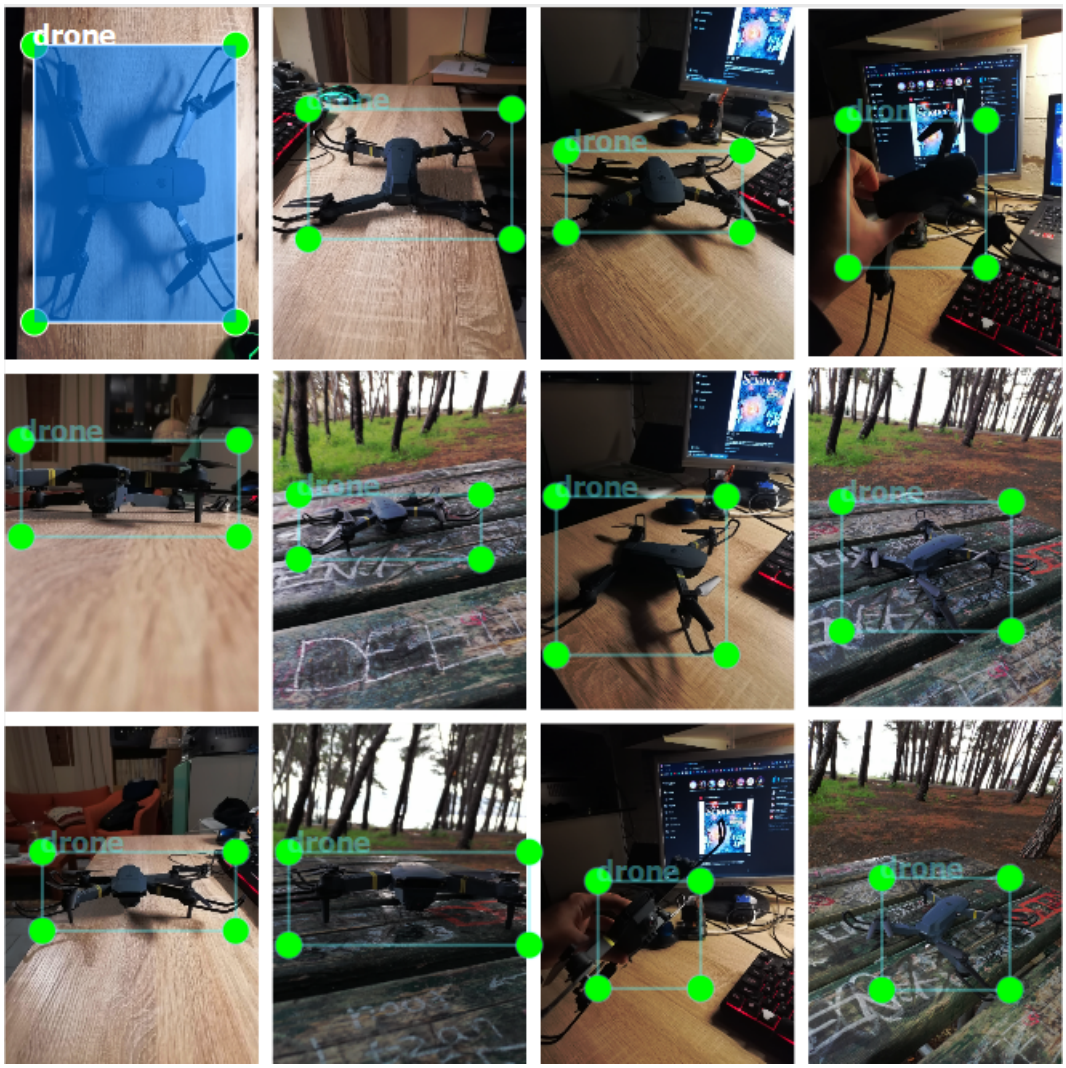

FIGURE 4.4: Sample Dataset for Training

to use an SSD-MobileNet-FPNLite architecture, which is optimized for mobile devices and can detect objects quickly and accurately.

With the training configuration set up, I trained my model by running a script that used the TensorFlow Object Detection API to iterate through my training data and optimize the model's parameters. This process took several hours and required a GPU to run efficiently.

Once training was complete, I converted my model to TensorFlow Lite format using another script. This allowed me to deploy my model to a wide range of devices, including mobile phones and edge devices like the Raspberry Pi. [14]

To test the performance of my model, I ran inference on a set of test images and calculated the mean average precision (mAP), a commonly used metric for object detection performance.

FIGURE 4.5: X-axis: Loss Classification - Y-axis: Training Steps

The Figure 4.5 shows the classification loss graph which focuses on how well the model is performing in predicting the correct class labels for the detected objects. This metric specifically measures the error in classifying objects, which is crucial for ensuring accurate drone detection. The X-axis represents the number of training steps and the Y-axis shows the classification loss value. The decreasing trend, despite fluctuations, indicates that the model's accuracy in classifying objects is improving over time.

The localization graph of Figure 4.6, tracks the progress of the model's training by measuring the localization loss, which indicates how well the model is learning to predict the bounding boxes around detected objects (drones). The X-axis represents the number of training iterations or steps and theY-axis represents the localization loss, which measures the error in the predicted bounding box coordinates. The graph shows a high loss at the beginning, which is expected as the model starts with initial weights that are not yet fine-tuned for the specific task of drone detection. There is a rapid decrease in localization loss in the early stages of training, indicating that the model is quickly learning to predict the locations of the drones more accurately. As training progresses, the rate of decrease in the loss slows down, and the loss

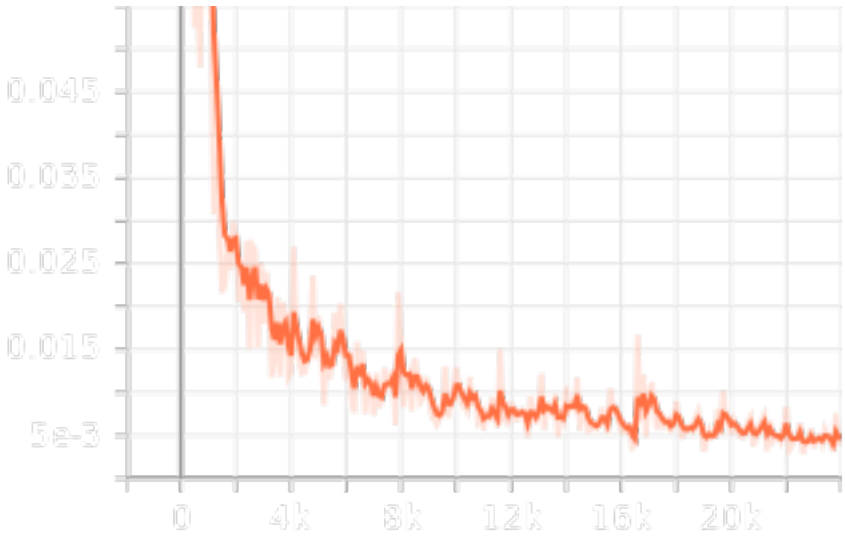

FIGURE 4.6: X-axis: Loss Localization - Y-axis: Training Steps

eventually plateaus. This suggests that the model is converging, meaning it is making only minor improvements with additional training steps.

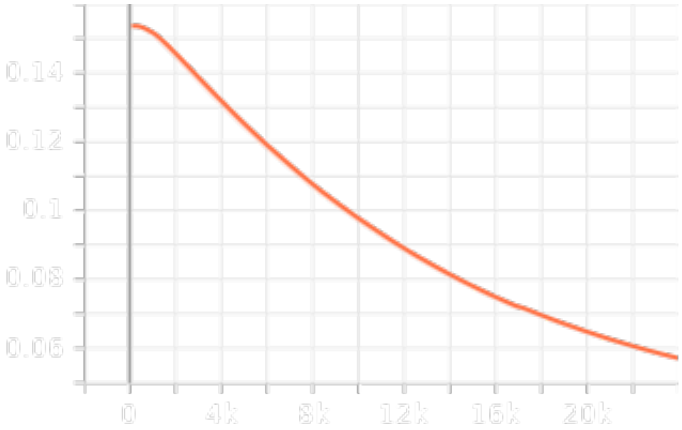

FIGURE 4.7: X-axis: Regularization Loss - Y-axis: Training Steps

The regularization loss graph of Figure 4.7 helps monitor the impact of regularization techniques used to prevent overfitting. Regularization terms (like L2 regularization) are added to the loss to penalize large weights, encouraging the model to generalize better to unseen data. The X-axis represents the number of training steps and the Y-axis shows the regularization loss value. A steady decrease in this loss indicates that the model's weights are being effectively constrained, reducing the risk of overfitting.
The total loss graph of Figure 4.8 serves as an overall indicator of the model's training progress. It combines various loss components (e.g., classification loss, localization loss) into a single metric. Monitoring this graph helps you assess whether the model is learning effectively and improving over time.

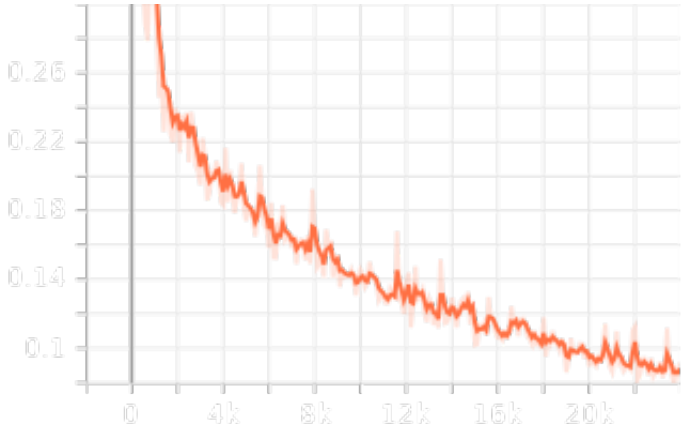

FIGURE 4.8: X-axis: Loss Total - Y-axis: Training Steps

The X-axis represents the number of training steps, which is the iterations the model has undergone. The Y-axis shows the total loss value. The decreasing trend indicates that the model's predictions are becoming more accurate and the overall error is reducing.

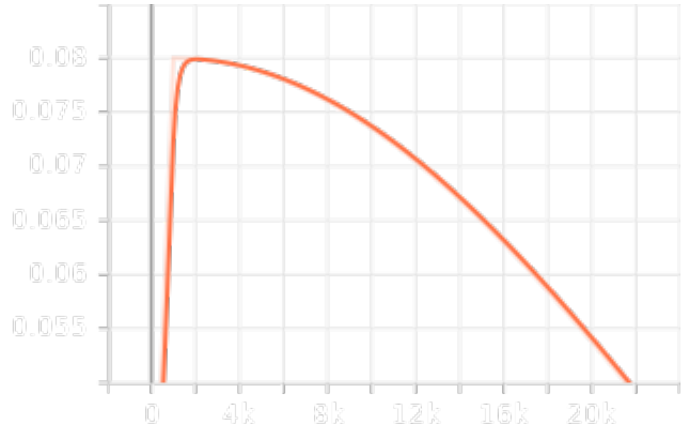

FIGURE 4.9: X: Learning Rate - Y-axis: Training Steps

The Learning Rate graph of Figure 4.9 illustrates the learning rate schedule used during training, showing how the learning rate changes over time.The X-axis represents the number of training iterations or steps and the Y-axis represents the value of the learning rate used at each training step. The learning rate starts low and increases. This is often part of a learning rate warm-up strategy, where a smaller learning rate is used initially to stabilize training. After reaching a peak, the learning rate decreases. This strategy helps the model to make large updates in the early stages for rapid learning and smaller updates later to fine-tune and converge to the optimal solution. The full process described in the appendix chapter outlines the steps taken to

train a custom object detection model using TensorFlow Lite in Colab, including data gathering and labeling, dependency installation, data preparation, model configuration, training, model conversion, testing, and post-training quantization. A

4.4.3 TF Lite Models Performance Comparison

TensorFlow Lite provides several object detection models, so I had to test a few of them to determine which one was most suitable for implementing the final system.
Choosing a deep learning object detection model for an embedded TensorFlow Lite application involves navigating a series of tradeoffs. Within the array of available lightweight models, each possesses its unique balance of speed and accuracy. While larger models such as EfficientDet deliver heightened accuracy, their expanded size contributes to slower inference speeds. On the other hand, smaller models like SSD-MobileNet offer rapid inference speeds but come with a trade-off of lower accuracy.
Below, I will present empirical data demonstrating the speed and accuracy of various TensorFlow Lite models when trained with a custom dataset and executed on a Raspberry Pi 4.

Models and Metrics

The models I concentrated on are those accessible through TensorFlow, encompassing models from both the TF1 and TF2 model zoo, as well as TFLite Model Maker. I specifically selected a subset of these models and assessed their performance. The chosen models include SSD-MobileNet-v2, SSD-MobileNet-v2-FPNLite-320x320, EfficientDet-d0, SSD-MobileNet-v1, and EfficientDet-Lite-D0.
Each model has its speed and accuracy metrics measured in inference speed per TensorFlow benchmark tool, FPS achieved when running in an OpenCV webcam pipeline, accuracy per COCO metric (mAP @ 0.5:0.95) and total number of objects correctly labeled in 75 test images. COCO mAP (Mean Average Precision) is a widely used evaluation metric in object detection tasks, especially those involving the COCO (Common Objects in Context) dataset. At the core of COCO mAP are precision and recall. Precision is the fraction of correctly identified objects (true positives) among all detected objects (true positives plus false positives), while recall is the fraction of correctly identified objects (true positives) among all actual objects (true positives plus false

negatives). These metrics provide insight into how well the model identifies objects and how many true objects it captures. IoU, or Intersection over Union, is another critical component. IoU measures the overlap between the predicted bounding box and the ground truth bounding box, calculated as the area of overlap divided by the area of union. Higher IoU values indicate better predictions. In COCO mAP, AP (Average Precision) is calculated for different IoU thresholds to evaluate the model's performance at various levels of localization accuracy. COCO mAP averages the AP values across ten IoU thresholds, ranging from 0.50 to 0.95 in increments of 0.05, providing a mean AP value. This averaging over multiple IoU thresholds ensures a robust assessment of the model's performance. COCO mAP is essential because it provides a comprehensive evaluation by considering multiple IoU thresholds, making it more robust than a single IoU threshold. It is a standard benchmark in the computer vision community, enabling consistent and comparable evaluations across different models and research studies. By focusing on localization accuracy, COCO mAP not only evaluates whether the objects are detected but also how accurately their bounding boxes are predicted. This makes COCO mAP a crucial metric for developing and assessing high-performance object detection models.

Measuring Model Speed

A model's inference speed is the amount of time it takes to process a set of inputs through neural network and generate outputs. When an object detection model runs inferencing on an image, it must propagate the input image through each layer of the network, performing layer operations and calculating values of nodes until the final output layer is reached. The more layers and nodes a model has, the longer inferencing takes. To measure raw inference time, I used Google's performance benchmark tool for TensorFlow Lite models. The tool provides information on minimum, maximum, and average inference speeds based on 100 iterations.

Measuring Model Accuracy

A widely used measure for assessing model accuracy is known as mAP, or mean average precision. The model performs inference on a set of test images, and the results (predicted classes and object locations) are compared to the ground truth data (the correct classes and object locations). The mAP score is then calculated based on the accuracy of these predictions. Simply

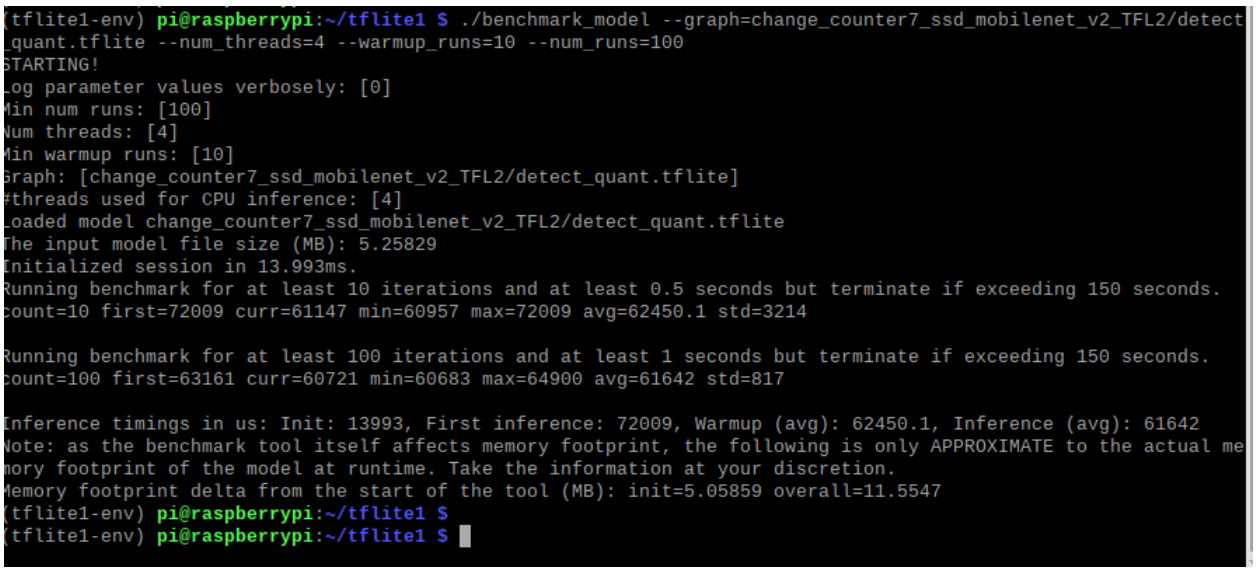

FIGURE 4.10: Example for SSD-MobileNet-v2 Model

put, the higher the mAP score, the more proficient the model is at recognizing objects in images.

Speed and Accuracy Results

Model	Inference Time (ms)	Throughput (FPS)
SSD-MobileNet-v2 (FP32)	142.1	2.85
SSD-MobileNet-v2 (INT8)	68.96	3.83
SSD-MobileNet-v2-FPNLite (FP32)	169.2	2.23
SSD-MobileNet-v2-FPNLite (INT8)	95.08	2.83
SSD-MobileNet-v1 (INT8)	65.59	3.76
EfficientDet-Lite-D0 (INT8)	112.6	2.58
EfficentDet-D0 (FP32)	1520	0.55

FIGURE 4.11: Model inference time and overall throughput

Here's a summarized overview of each model's performance based on the provided data:

SSD-MobileNet-v2

Demonstrates an inference time of 68.96 ms and a COCO mAP score of 60.99%. The quantized version achieves a favorable balance between speed

Model	Accuracy Score (COCO mAP @ 0.5:0.95)	Objects Correctly Labeled (out of 335)
SSD-MobileNet-v2 (FP32)	75.33%	306
SSD-MobileNet-v2 (INT8)	60.99%	264
SSD-MobileNet-v2-FPNLite (FP32)	84.81%	326
SSD-MobileNet-v2-FPNLite (INT8)	76.41%	302
SSD-MobileNet-v1 (INT8)	49.77%	225
EfficientDet-Lite-D0 (INT8)	60.79%	302
EfficentDet-D0 (FP32)	46.91%	235

FIGURE 4.12: Model accuracy and total number of correctly labeled objects in dataset

and accuracy. Consider utilizing the floating-point version for a modest increase in accuracy with only a minor reduction in speed.

SSD-MobileNet-v2-FPNLite

Slightly slower than the regular SSD-MobileNet-v2 but exhibits excellent accuracy, particularly given the small size of the training dataset. Successfully detects 306 out of 335 total objects in test images. Notably, it maintains consistent confidence levels (90% - 99%) in its predictions, unlike the varying confidences of the EfficientDet-Lite-D0 mode

EfficientDet-Lite-D0

Taking fifth place in accuracy and fourth place in speed, this model has middle-of-the-road performance. It may be a good alternative to try if other models don't have high enough accuracy on your dataset.

EfficientDet-D0

Proves slow and inaccurate on this dataset, likely due to the limited size of the dataset used for training. Theoretically capable of achieving higher accuracy than other models listed, but recommended only for applications with ample compute power and a larger training dataset.

4.4.4 Real-Time Object Detection on RPi

Finally, the results of the training are visible. The camera successfully detects the drone, but there's a challenge— the Mobile SSD Net is quite resource-intensive for the Raspberry Pi 4, resulting in a lower frames-per-second (fps). Below we can see some screenshots from the feed of the camera. As we can see, the model detects the drone with a good accuracy.

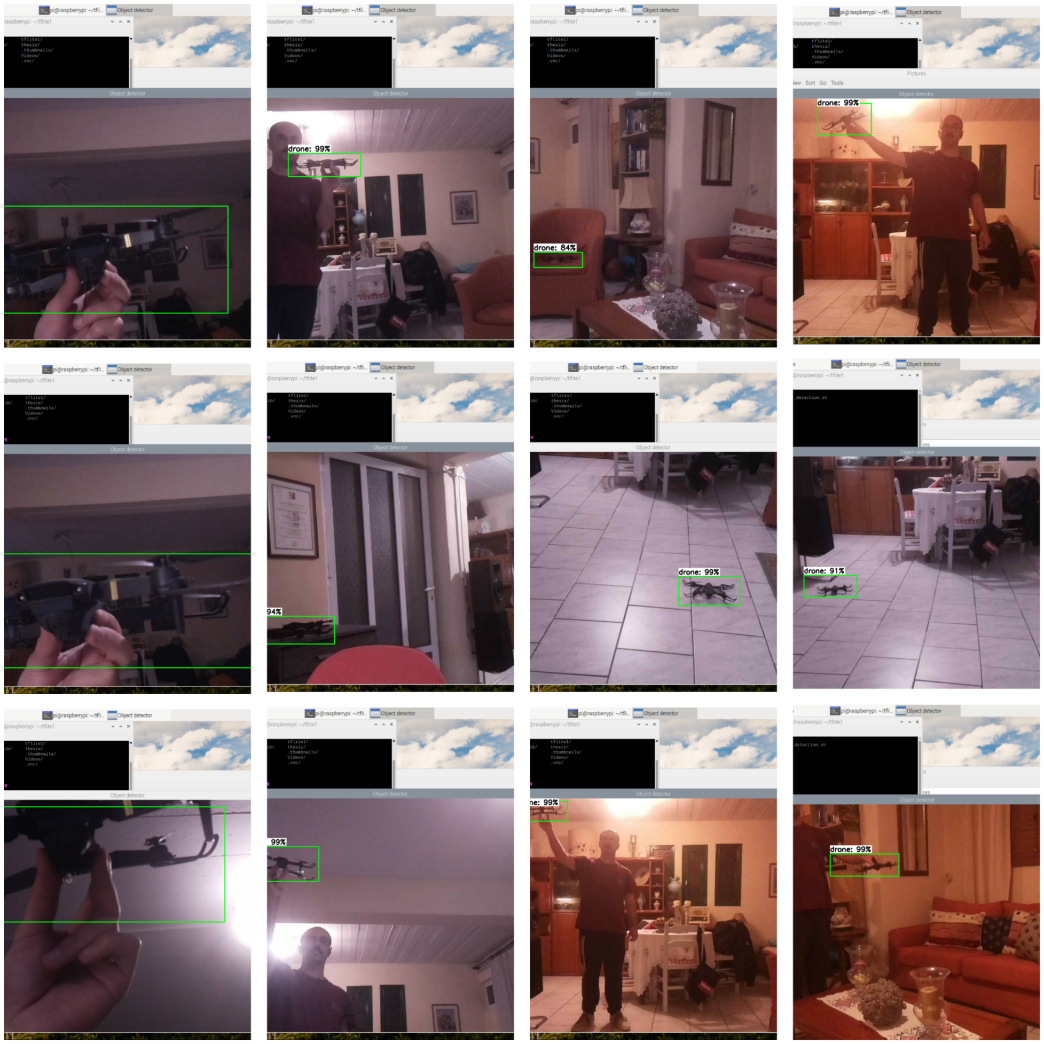

FIGURE 4.13: Detection Screenshots

For achieving greater accuracy and higher frames per second (fps), we would undoubtedly require technology with enhanced processing power, more memory, and a professional-grade camera. However, the effectiveness of the model's training also plays a significant role. The training was conducted on a modest dataset of 200-300 photos, each with a resolution of 640x480, which is relatively low. Training on datasets with very high resolutions and larger

sizes would demand substantial hardware resources and extended training hours, making it impractical for the Google Colab platform I used.
The training process itself took a minimum of 5 hours, and more when we increased the training steps. Unfortunately, due to frequent internet disconnections, the session was occasionally disrupted, requiring me to restart from the beginning.

4.5 Tracking

4.5.1 Generate Pelco-D Command Frame

The gimbal_transmit function is used to transmit a Pelco-D command. It takes in eight bytes as parameters, which together form the command. The _delay_us function is used to introduce a delay in the execution of the program, taking in a time in microseconds as a parameter. The stop function is used to stop the gimbal's motion, and up and down functions are used to move the gimbal up and down respectively, with no parameters needed. For moving the gimbal up or down by a specific number of degrees, the up_deg and down_deg functions respectively take in the number of degrees as a parameter. Similarly, the left function is used to move the gimbal left with no parameters, while the function moves the gimbal left by a certain number of degrees, taking in the number of degrees as a parameter. The output of these function calls will be a hexadecimal string that represents the Pelco-D command to be transmitted to the gimbal.

```
# Main program
if __name__ == '__main__':

    right_deg(45)
    up_deg(42)
    left_deg(21)
    down_deg(15)
```

FIGURE 4.14: Function Calls for Pelco-D command construction

The following diagram provides an overview of the different functions and interactions involved in controlling the gimbal, at a high level of abstraction.

```
C:\Users\thana\OneDrive\Desktop\Detection-Tracking-UAV>py main.py
0xa0 0x0 0x0 0x2 0x20 0x0 0xaf 0x2d
0xa0 0x0 0x0 0x8 0x0 0x20 0xaf 0x27
0xa0 0x0 0x0 0x4 0x20 0x0 0xaf 0x2b
0xa0 0x0 0x0 0x10 0x0 0x20 0xaf 0x3f
```

FIGURE 4.15: Output

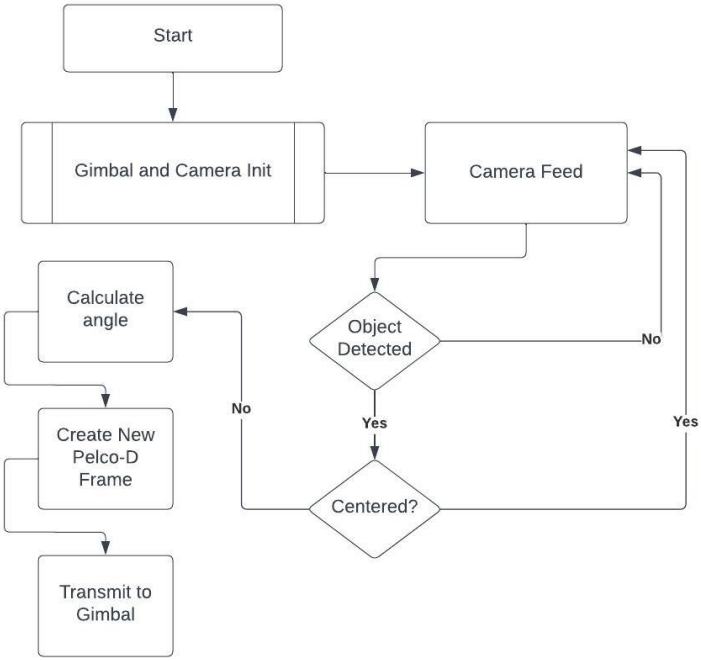

FIGURE 4.16: Top-level functionality block diagram

4.5.2 AS20RS485 Gimbal

The COP-USA AS20RS485 [15] is a pan/tilt mount that is Pelco P compatible and can handle a max load of 12Lbs. The AS20RS485 runs on 24VAC power and can be used both indoors and outdoors.
It needs a 230VAC/24VAC transformer to power it. The AS20RS485 is a device that communicates through the RS485 protocol, which is a standard for serial communication over long distances. A USBtoRS485 converter used for this purpose which uses the CH340 chip. [16]

FIGURE 4.17: AS20RS485 Pan/Tilt Mount

FIGURE 4.18: CH340 USBtoRS485 Converter

The gimbal also supports Pelco-D, B01, Longomity, Hunda600, LILIN,ALEC protocols. The protocol and baud-rate is selected via dip-switches.

Protocols	Selection of Protocols				Normal Baud Rate	
	DIP-1	DIP-2	DIP-3	DIP-4	DIP-5	DIP-6
SAMSUNG	ON	OFF	OFF	OFF	OFF	ON
B01	ON	OFF	OFF	OFF	OFF	ON
PELCO-D	ON	ON	OFF	OFF	OFF	OFF
PELCO-P/4800	OFF	OFF	ON	OFF	ON	OFF
PELCO-P/9600					OFF	ON
Longcomity	OFF	ON	ON	OFF	OFF	ON
HUNDA600	ON	ON	ON	OFF	OFF	ON
LILIN	OFF	OFF	OFF	ON	OFF	ON

FIGURE 4.19: Protocol Selection

FIGURE 4.20: AS20RS485 dip-switches selection for Pelco-P

As can be seen, a PELCO-P command consists of 8 bytes. The first byte (always 0xA0) functions as the start byte (start of text). The second byte refers to the address of the camera, which is set via dip switches. The data bytes are presented in detail below. Byte 7 (always 0xAF) is the end byte (end of text). Byte 8 is used for error checking in transmission and is calculated as the xOR of the previous bytes.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
STX	Camera Address	Data 1	Data 2	Data 3	Data 4	ETX	Checksum

FIGURE 4.21: Pelco-P Bytes

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Data 1	Fixed to 0	Camera On	Auto Scan On	Camera On/Off	Iris Close	Iris Open	Focus Near	Focus Far
Data 2	Fixed to 0	Zoom Wide	Zoom Tele	Tilt Down	Tilt Up	Pan Left	Pan Right	0 (for pan / tilt)
Data 3	Pan speed 00 (stop) to 3F (high speed) and 40 for Turbo							
Data 4	Tilt speed 00 (stop) to 3F (high speed)							

FIGURE 4.22: Pelco-P: Data 1-4 Bits

The bits used are bit4 to bit1 of the Data 2 which determine the type of movement the base will make. Data 3 and Data 4 bytes were set to the default value of 0x20, as this particular base does not support different rotation speeds. The Data 1 byte was set to 0x00 as it relates to camera handling. Based on the above information, the basic functions left( ), right( ), up( ), down( ), stop( ) become clear. Regarding the functions that make the corresponding movements but also take as an argument the degrees of the rotational movement,

it should be mentioned that their operation is based on the introduction of a delay where it leaves the corresponding simple rotation command active for an appropriate time, before the stop( ) function is called.

4.5.3 Pelco-P and Pelco-D support

Most industrial gimbals communicate with a PC or an embedded system via RS485 and Pelco-P/D protocols, so it is important to provide support for protocols and their functions, even though not all of them will be used. For this reason, the appropriate libraries were written so that each user can connect his own gimbal, depending on the application's requirements, or another developer can extend or change the functionality according to his own project.
The functions supported by the two files "pelco_d.py" and "pelco_p.py" are left, right, down, up, zoom wide, zoom tele, focus far, focus near, stop.
Accordingly, the motion functions with an angle argument are also supported, which produce the same frame but knowing that the gimbal moves at 5 degrees/second, we hold the command for the corresponding time until it reaches the desired angle.
Below we will see the pelco-p and pelco-p frames produced by the libraries but also by the analyzer232 software that offers us a GUI and programmable buttons to test the movements faster.

```
C:\Users\thana\OneDrive\Documents\Github\thesis>py pelco_d.py
0xff 0x1 0x0 0x4 0x20 0x0 0x25
0xff 0x1 0x0 0x2 0x20 0x0 0x23
0xff 0x1 0x0 0x8 0x0 0x20 0x29
0xff 0x1 0x0 0x10 0x0 0x20 0x31
0xff 0x1 0x0 0x40 0x0 0x0 0x41
0xff 0x1 0x0 0x80 0x0 0x0 0x81
0xff 0x1 0x1 0x0 0x0 0x0 0x2

C:\Users\thana\OneDrive\Documents\Github\thesis>py pelco_d.py
0xff 0x1 0x0 0x4 0x20 0x0 0x25
0xff 0x1 0x0 0x2 0x20 0x0 0x23
0xff 0x1 0x0 0x8 0x0 0x20 0x29
0xff 0x1 0x0 0x10 0x0 0x20 0x31
0xff 0x1 0x0 0x20 0x0 0x0 0x21
0xff 0x1 0x0 0x40 0x0 0x0 0x41
0xff 0x1 0x0 0x80 0x0 0x0 0x81
0xff 0x1 0x1 0x0 0x0 0x0 0x2
```

FIGURE 4.23: Pelco-P and Pelco-D frames

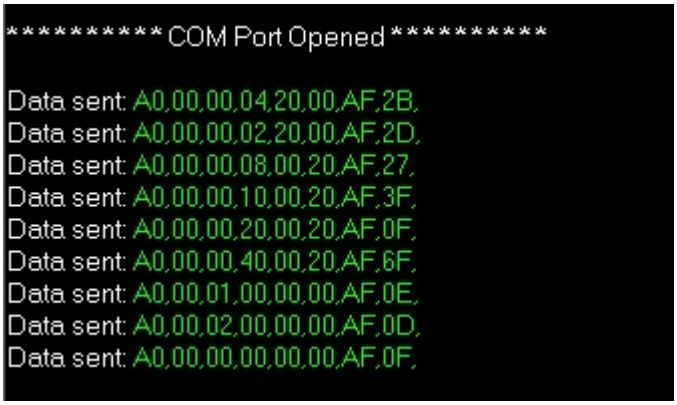

FIGURE 4.24: 232Analyzer: Pelco-P frames

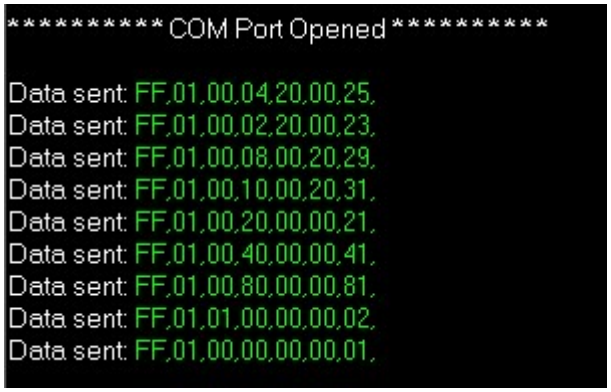

FIGURE 4.25: 232Analyzer: Pelco-D frames

4.5.4 Gimbal Modes and Scalability via RS485

In this section we will analyze the modes that will be created and the way the gimbal will work in each of them.

Gimbal Modes

Mode 1: Vertical Scan

In vertical scan mode the gimbal will repeatedly do up-down operations. The tilt range supported by the AS20RS485 is -50 degrees/+50 degrees, so this repetitive motion will be done within this range of degrees. In case of changing or expanding the code and the equipment by another developer, there are detailed instructions in the comments for changing the range according to the requirements of the project or the device that will be connected.
If a drone is detected during the scan, a bounding box around the drone will be created.

Mode 2: Horizontal Scan

Corresponding to the Vertical Scan mode, in Horizontal Scan will scan in the range of 355 degrees supported by the AS20RS485. Most gimbals support a pan range close to 360 degrees, but there are relevant instructions for modifying the range in the comments.
To make better use of the camera's FOV, its tilt will be set to 25 degrees, right in the middle of the [0.50] degree range.
If a drone is detected during the scan, a bounding box around the drone will be created.

Mode 3: Idle until drone detection

In this mode, the gimbal is initialized to a predetermined position and remains inactive until a drone is detected. When this happens, the gimbal moves accordingly to keep the drone within the camera's FOV.

Mode 4: Idle

In IDLE mode the gimbal does not perform any movement. It is only useful in cases where the user does not want the device to operate or for consumption purposes such as powering the embedded system and gimbal with battery.

Scalability

RS485 can drive up to 32 devices, but using isolated repeaters can drive multiples of 32. So the implementation of modes as well as support for Pelco-P and Pelco-D protocols can be used in a network of gimbals.
Also, it is not restrictive for the gimbal to carry any camera on it, but according to the needs of each application, it can carry any other sensor, any antenna or any laser. The implementation of the protocols remains the same, with minor changes that may be needed such as the pan/tilt ranges that are predefined by each gimbal.
More on issues of scalability, changes and alternative uses in the Future Work chapter. 6.2

4.5.5 Multithreading for Concurrent Gimbal Control

Threading Implementation in the Gimbal Control Script

Threading is implemented through the use of the threading module in Python. mode_lock is initialized as a threading lock to synchronize access to the shared variable mode.

```
speed_deg_per_sec = 5  # Gimbal rotation speed in degrees per second
mode = 0  # Initialize mode variable
active_thread = None  # Initialize active_thread variable
mode_lock = threading.Lock()  # Lock for synchronizing mode changes
```

FIGURE 4.26: Initialization Section

```
def gimbal_control_loop(ser):

    """...



    """
    global mode
    global active_thread

    while True:
        new_mode = input("Enter gimbal mode (1-4) or 'exit' to quit: ")

        if new_mode == "exit":
            break

        new_mode = int(new_mode)

        if new_mode == mode:
            print("Already in that mode.")
            continue

        with mode_lock:
            mode = new_mode

            if active_thread is not None:
                stop_active_thread(active_thread)

            if mode == 1:
                active_thread = threading.Thread(target=gimbal_mode1, args=(ser,), daemon=True)

            elif mode == 2:
                active_thread = threading.Thread(target=gimbal_mode2, args=(ser,), daemon=True)

            elif mode == 3:
                active_thread = threading.Thread(target=gimbal_mode3, args=(ser,), daemon=True)

            elif mode == 4:
                active_thread = threading.Thread(target=gimbal_mode4, args=(ser,), daemon=True)

            else:
                print("Invalid mode! Please enter a valid mode (1-4) or 'exit'.")
                continue

            active_thread.start()

    stop_active_thread(active_thread)
```

FIGURE 4.27: Gimbal Control Loop Section

Threading is applied to the gimbal control loop, allowing for concurrent execution of gimbal modes. The with mode_lock statement ensures proper synchronization when modifying the shared variable mode. Daemon threads are used for flexibility in program termination.

Active Thread Management with Daemons

The stop_active_thread function is responsible for stopping the currently active thread. It utilizes the active_thread.join() method to ensure the program waits for the thread to finish before proceeding.

```
# Function to stop the active thread
def stop_active_thread(active_thread):
    if active_thread is not None and active_thread.is_alive():
        active_thread.join()
```

FIGURE 4.28: Stop Active Thread

Before starting a new thread, the active thread is stopped using the stop_active_thread function. This ensures proper management of active threads and prevents conflicts between different modes.
The implementation of threading with locks in critical sections serves to prevent race conditions, ensuring robust thread safety. The management of active threads is executed in a way that guarantees the orderly stopping of threads prior to initiating new ones. The integration of daemon threads contributes to the program's exit flexibility, facilitating a smooth termination process even when threads remain active.

```
with mode_lock:
    mode = new_mode
    if active_thread is not None:
        stop_active_thread(active_thread)
    # ...
```

FIGURE 4.29: Stop Active Thread

4.5.6 Tracking Algorithm Logic

The algorithm is designed to determine the necessary gimbal movement to center a detected object in a live camera feed. The gimbal system has a rotation speed of $s_{\text{deg/sec}}$ degrees per second.

Inputs:

1. Detected object bounding box coordinates: $x_{\min}, y_{\min}, x_{\max}, y_{\max}$.
2. Camera frame dimensions: width and height.
3. Gimbal rotation speed: $s_{\text{deg/sec}}$.

Outputs: Pelco command sequence for gimbal movement.

Steps:

1. Calculate the center coordinates of the camera frame $(\text{center}_x, \text{center}_y)$.
2. Determine the center coordinates of the detected object $(\text{detected\_center}_x, \text{detected\_center}_y)$.
3. Calculate the angular displacement (δ_x, δ_y) needed to center the detected object based on the difference between camera frame center and object center.
4. Convert the angular displacement to the corresponding Pelco commands using the function `GeneratePelcoCommands` (replace this with actual Pelco-D or Pelco-P function calls).
5. Return the Pelco command sequence for gimbal movement.

This algorithm is a simple approach of an algorithm from "OBJECT TRACKING CONTROL USING A GIMBAL MECHANISM" [17] of University of Florianapolis.

Algorithm 2 Gimbal Movement Calculation using Pelco-D or Pelco-P

Input: Detected object bounding box $[x_{\min}, y_{\min}, x_{\max}, y_{\max}]$, Camera frame dimensions $(\text{width}, \text{height})$, Gimbal rotation speed $s_{\text{deg/sec}}$
Output: Pelco command sequence for gimbal movement
$\text{center}_x \leftarrow \text{width}/2$
$\text{center}_y \leftarrow \text{height}/2$
$\text{detected\_center}_x \leftarrow (x_{\min} + x_{\max})/2$
$\text{detected\_center}_y \leftarrow (y_{\min} + y_{\max})/2$
$\delta_x \leftarrow \lfloor (\text{detected\_center}_x - \text{center}_x) \times s_{\text{deg/sec}} \rfloor$
$\delta_y \leftarrow \lfloor (\text{detected\_center}_y - \text{center}_y) \times s_{\text{deg/sec}} \rfloor$
pelco_commands $\leftarrow$ GeneratePelcoCommands(δ_x, δ_y) ▷ Use Pelco-D or Pelco-P functions
Return *pelco_commands*

Chapter 5

Verification & Results

After analyzing the individual subsystems and their functionalities, the photo below illustrates the final form of the embedded system.

FIGURE 5.1: Embedded Gimbal System

5.1 Detection Results

As analyzed in the Sysstem Design and Implentation chapter and showcased in the demos, the detection component performs at a good level. The problems lie in the low frames per second (fps) and the limited dataset, occasionally resulting in false detections. Nevertheless, despite these issues, the drone recognition is mainly accurate.

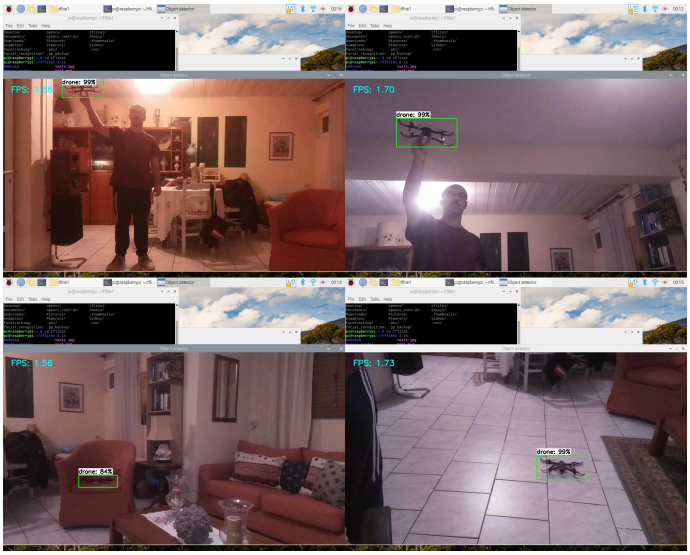

FIGURE 5.2: Detection Tests

In order to improve the performance, the model will have to be trained on a much larger dataset of higher resolution, which requires much more computing power, and therefore a much more powerful gpu, which greatly increases the cost. Also the increased resolution leads to the requirement of a professional camera.

5.2 Tracking Results

One of the first goals of this thesis was to create code that fully supports PELCO-P/D protocols so that it can be reused by simply calling implemented functions, without anyone needing to know which PELCO-P/D frame is needed to move the gimbal, eg left or right. The implementation of these functions was done even for functions that were not used. This is a good starting point for those who want to expand the functions, eg add a professional camera and use zoom-in/zoom-out or focus functions.
As it is obvious, in order to achieve tracking, it should be possible for the gimbal to move correctly in the direction and at the angle that is calculated,

so the implementation of these functions, apart from being very useful, is also extremely convenient. In the corresponding chapter we analyzed that the gimbal movement commands produce the correct frame, so we have a good implementation of the protocol.

5.3 Full System Results

By combining the two subsystems, we notice that the final system meets the requirements we set. Despite this there are also problems as the limited rotation speed of the gimbal (5 degrees / second) as well as the low fps and the small dataset make the device to have a slow response. More specifically, if the drone moves very fast and goes out of the camera's field of view, the gimbal will not have time to rotate and will not be able to follow it. We also notice that in low light conditions the response of the system drops much more as detection becomes even more difficult.
The system was tested in indoor conditions, where its response was satisfactory, as it detects the drone and follows it when it moves at low speed. It should be noted that for the tests I move the drone by hand as the development of high speeds limit the detection of the drone, since as mentioned above the response of the system due to low fps is slow.
However, through the know-how acquired through this thesis, by upgrading individual units of the system, we could achieve much better results and implement a system with better response, greater turning speed and better target recognition.
More about possible upgrade and use of the final system will be discussed in the next chapter.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Embedded systems and IoT in conjunction with tinyML is a rapidly growing field. New microcontrollers and single board computers make it possible to run machine learning models that were not possible in the past. This opens the way for new applications that will make the devices much more intelligent and autonomous. Through this thesis we examined the topic of detection and tracking and how it can be implemented through a cheap computer system in a rudimentary way. I also gained hands-on experience how to combine machine learning models with protocols used in industry or CCTV type devices and implement a device that has more complex functions.

6.2 Future Work

There are many extensions and variations that could be made to the embedded system. As I mentioned in the Verification and Results chapter, many functions could be optimized by using a board with greater computing power and better training of the model. Also scalability is something important as we could imagine a project with several gimbals tracking a common object-target and its turning angles being produced by a common SBC.
Additionally, remote notification functions could be implemented, e.g. via email or SMS, with the use of corresponding modules, where the user will be notified of the presence of the drone by sending a photo or text.
The target is not limited to being a UAV, in our case a drone, for example the gimbal could have a fabric structure on it that provides shade to crops at very high temperatures to protect them and move accordingly or an irrigation device that can to turn, depending on the low moisture readings in the soil.

Another idea would be the combination of the current thesis with Mr. Apostolakis thesis, so that in addition to tracking a drone, directional antennas can also be used to communicate with the drone. This assumes that the drone will support corresponding communication systems with the embedded system. Also, through communication, the integrated system could know its exact position, i.e. its geographical coordinates, and thus the tracking would be much more acurate.[18] [19]
Finally, for the improvement of this thesis, larger datasets could be used where we would achieve much better detection, but also with the use of SBC with greater computational power, for example Google Coral board and a better camera, the overall operation and response of the system would be improved.

Appendix A

TFLite2 Object Detection Model

The appendix chapter provides a detailed description of the process I followed to train a custom object detection model using TensorFlow Lite 2. This process involved gathering and labeling training images, installing the necessary dependencies, configuring the training process, and evaluating the performance of the model on test data. This chapter serves as a guide for anyone looking to develop their own custom object detection models using TensorFlow Lite 2. we will walk through the process of installing the TensorFlow Object Detection API in a Google Colab instance. This involves cloning the TensorFlow models repository, running installation commands, and verifying the installation with a test script. The latest version of TensorFlow that is verified to work with this Colab is TF v2.8.0. [20] [21]
1. Clone the TensorFlow models repository from GitHub:

```
# Clone the tensorflow models repository from GitHub
!git clone --depth 1 https://github.com/tensorflow/models
```

FIGURE A.1: Clone TF Models

2. Copy setup files into models/research folder:

```
# Copy setup files into models/research folder
%%bash
cd models/research/
protoc object_detection/protos/*.proto --python_out=.
#cp object_detection/packages/tf2/setup.py .
```

FIGURE A.2: Setup Files

3. Modify setup.py file to install the tf-models-official repository targeted at TF v2.8.0:
4. Install the Object Detection API[22]:
5. Test the installation
6. Connect to Google Drive:

```
# Modify setup.py file to install the tf-models-official repository targeted at TF v2.8.0
import re
with open('/content/models/research/object_detection/packages/tf2/setup.py') as f:
    s = f.read()

with open('/content/models/research/setup.py', 'w') as f:
    # Set fine_tune_checkpoint path
    s = re.sub('tf-models-official>=2.5.1',
               'tf-models-official==2.8.0', s)
    f.write(s)
```

FIGURE A.3: Modify Setup File

```
# Install the Object Detection API
!pip install /content/models/research/

# Need to downgrade to TF v2.8.0 due to Colab compatibility bug with TF v2.10 (as of 10/03/22)
!pip install tensorflow==2.8.0
```

FIGURE A.4: Install Object Detection API

```
from google.colab import drive
drive.mount('/content/gdrive')

!cp /content/gdrive/MyDrive/images/images.zip /content
```

FIGURE A.6: Drive connection

7. Split images into train, validation, and test folders:

```
!mkdir /content/images
!unzip -q images.zip -d /content/images/all
!mkdir /content/images/train; mkdir /content/images/validation; mkdir /content/images/test
```

FIGURE A.7: Drive connection

8. Next, we'll split the images into train, validation, and test sets. Here's what each set is used for:
Train: These are the actual images used to train the model. In each step of training, a batch of images from the "train" set is passed into the neural network. The network predicts classes and locations of objects in the images. The training algorithm calculates the loss (i.e. how "wrong" the predictions were) and adjusts the network weights through backpropagation.
Validation: Images from the "validation" set can be used by the training algorithm to check the progress of training and adjust hyperparameters (like learning rate). Unlike "train" images, these images are only used periodically during training (i.e. once every certain number of training steps).

```
# Run Model Bulider Test file, just to verify everything's working properly
!python /content/models/research/object_detection/builders/model_builder_tf2_test.py
```

FIGURE A.5: Test Installation

Test: These images are never seen by the neural network during training. They are intended to be used by a human to perform final testing of the model to check how accurate the model is.
I use a Python script to randomly move 80% of the images to the "train" folder, 10% to the "validation" folder, and 10% to the "test" folder.

```
!wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/mast
!python train_val_test_split.py
```

FIGURE A.8: Drive connection

9. Finally, we need to create a labelmap for the detector and convert the images into a data file format called TFRecords, which are used by TensorFlow for training. We'll use Python scripts to automatically convert the data into TFRecord format. Before running them, we need to define a labelmap for our classes.

```
### This creates a a "labelmap.txt" file with a list of classes the object detection model will detect.
%%bash
cat <<EOF >> /content/labelmap.txt
drone
EOF
```

FIGURE A.9: Labelmap example

```
# Download data conversion scripts
! wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/mas
! wget https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/mas

# Create CSV data files and TFRecord files
!python3 create_csv.py
!python3 create_tfrecord.py --csv_input=images/train_labels.csv --labelmap=labelmap.txt --image_dir=images/train --output_
!python3 create_tfrecord.py --csv_input=images/validation_labels.csv --labelmap=labelmap.txt --image_dir=images/validatio
```

FIGURE A.10: TFRecord Files

```
train_record_fname = '/content/train.tfrecord'
val_record_fname = '/content/val.tfrecord'
label_map_pbtxt_fname = '/content/labelmap.pbtxt'
```

FIGURE A.11: TFRecord and labelmap location

10. In this section, we'll set up the model and training configuration. We'll specifiy which TensorFlow model we want to use from the TensorFlow 2 Object Detection Model Zoo. Each model also comes with a configuration file that points to file locations, sets training parameters (such as learning rate and total number of training steps), and more. We'll modify the configuration file for our custom training job.

```
# Change the chosen_model variable to deploy different models available in the TF2 object detection zoo
chosen_model = 'ssd-mobilenet-v2-fpnlite-320'

MODELS_CONFIG = {
    'ssd-mobilenet-v2': {
        'model_name': 'ssd_mobilenet_v2_320x320_coco17_tpu-8',
        'base_pipeline_file': 'ssd_mobilenet_v2_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint': 'ssd_mobilenet_v2_320x320_coco17_tpu-8.tar.gz',
    },
    'efficientdet-d0': {
        'model_name': 'efficientdet_d0_coco17_tpu-32',
        'base_pipeline_file': 'ssd_efficientdet_d0_512x512_coco17_tpu-8.config',
        'pretrained_checkpoint': 'efficientdet_d0_coco17_tpu-32.tar.gz',
    },
    'ssd-mobilenet-v2-fpnlite-320': {
        'model_name': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8',
        'base_pipeline_file': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.config',
        'pretrained_checkpoint': 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz',
    },
    # The centernet model isn't working as of 9/10/22
    #'centernet-mobilenet-v2': {
    #    'model_name': 'centernet_mobilenetv2fpn_512x512_coco17_od',
    #    'base_pipeline_file': 'pipeline.config',
    #    'pretrained_checkpoint': 'centernet_mobilenetv2fpn_512x512_coco17_od.tar.gz',
    #}
}
```

FIGURE A.12: Model Selection

```
# Create "mymodel" folder for holding pre-trained weights and configuration files
%mkdir /content/models/mymodel/
%cd /content/models/mymodel/

# Download pre-trained model weights
import tarfile
download_tar = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/' + pretrained_checkpoint
!wget {download_tar}
tar = tarfile.open(pretrained_checkpoint)
tar.extractall()
tar.close()

# Download training configuration file for model
download_config = 'https://raw.githubusercontent.com/tensorflow/models/master/research/object_detection/configs/tf2/'
!wget {download_config}
```

FIGURE A.13: Weights and configuration files

```
# Set training parameters for the model
num_steps = 40000

if chosen_model == 'efficientdet-d0':
  batch_size = 4
else:
  batch_size = 16

# Set file locations and get number of classes for config file
pipeline_fname = '/content/models/mymodel/' + base_pipeline_file
fine_tune_checkpoint = '/content/models/mymodel/' + model_name + '/checkpoint/ckpt-0'

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

FIGURE A.14: Training parameters

```
# Run training!
!python /content/models/research/object_detection/model_main_tf2.py \
    --pipeline_config_path={pipeline_file} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --sample_1_of_n_eval_examples=1
```

FIGURE A.15: Run training

11. Convert Model to TFLite.

First, we need to export the model graph (a file that contains information about the architecture and weights) to a TensorFlow Lite-compatible format.

```
# Make a directory to store the trained TFLite model
!mkdir /content/custom_model_lite
output_directory = '/content/custom_model_lite'

# Path to training directory (the conversion script automatically chooses the highest checkpoint file)
last_model_path = '/content/training'

!python /content/models/research/object_detection/export_tflite_graph_tf2.py \
    --trained_checkpoint_dir {last_model_path} \
    --output_directory {output_directory} \
    --pipeline_config_path {pipeline_file}
```

FIGURE A.16: Export graph

Next, we'll take the exported graph and use the TFLiteConverter module to convert it to .tflite FlatBuffer format.

```
# Convert exported graph file into TFLite model file
import tensorflow as tf

converter = tf.lite.TFLiteConverter.from_saved_model('/content/custom_model_lite/saved_model')
tflite_model = converter.convert()

with open('/content/custom_model_lite/detect.tflite', 'wb') as f:
  f.write(tflite_model)
```

FIGURE A.17: Convert exported graph file into TFLite model file

12. Test TensorFlow Lite Model and Calculate mAP

```
%cd /content/mAP
!python calculate_map_cartucho.py --labels=/content/labelmap.txt
```

FIGURE A.18: mAP Calculation

13.Download TensorFlow Lite Model

```
# Move labelmap and pipeline config files into TFLite model folder and zip it up
!cp /content/labelmap.txt /content/custom_model_lite
!cp /content/labelmap.pbtxt /content/custom_model_lite
!cp /content/models/mymodel/pipeline_file.config /content/custom_model_lite

%cd /content
!zip -r custom_model_lite.zip custom_model_lite
```

```
from google.colab import files

files.download('/content/custom_model_lite.zip')
```

FIGURE A.19: Download trained model

References

[1] "Gimbal". In: (). URL: https://en.wikipedia.org/wiki/Gimbal.

[2] Vegard Wollan Alf-Egil Bogen. "AVR Enhanced RISC Microcontrollers". In: (1997). URL: http://www.compass-lab.com/pdf/AVR_RISC.pdf.

[3] Atmel-Microchip. "Atmega16 Manual". In: (). URL: http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf.

[4] "Mars Helicopter". In: (). URL: https://mars.nasa.gov/technology/helicopter/#Quick-Facts.

[5] "FANUC LR Mate 200iC". In: (). URL: https://www.robots.com/robots/fanuc-lr-mate-200ic.

[6] "Kuka LBR iiwa". In: (). URL: https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa.

[7] "RQ-21Q". In: (). URL: https://www.insitu.com/products/rq21a.

[8] Sotiriades Evripidis Dollas Apostolos Papaefstathiou Ioannis. "ROTA: an Archipelago-Wide Area Network for High Speed Communication to Ships". In: (2012). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6377404.

[9] "Flying Objects Detection from a Single Moving Camera". In: (2014). URL: https://openaccess.thecvf.com/content_cvpr_2015/papers/Rozantsev_Flying_Objects_Detection_2015_CVPR_paper.pdf.

[11] John C. Woods Carolyn Swinney. "Low-Cost Raspberry-Pi-Based UAS Detection and Classification System Using Machine Learning". In: (2022). URL: https://www.researchgate.net/publication/365676991_Low-Cost_Raspberry-Pi-Based_UAS_Detection_and_Classification_System_Using_Machine_Learning.

[12] Gaurav Singal D. Vijay Rao Priyanka Gupta Bhavya Pareek. "Edge device based Military Vehicle Detection and Classification from UAV". In: (2022). URL: https://www.researchgate.net/publication/365676991_Low-Cost_Raspberry-Pi-Based_UAS_Detection_and_Classification_System_Using_Machine_Learning.

[13] "AutoPictureForTrainingData.py Script". In: (). URL: https://github.com/amanesis/thesis/blob/main/AutoPictureForTrainingData.py.

[14] "Migrating your TFLite code to TF2". In: (). URL: https://www.tensorflow.org/guide/migrate/tflite.

[15] "AS20RS485 Manual". In: (). URL: https://www.manualsdir.com/manuals/656477/cop-usa-as20rs485.html.

[16] "CH340 Chip". In: (). URL: https://www.mpja.com/download/35227cpdata.pdf.

[17] "OBJECT TRACKING CONTROL USING A GIMBAL MECHANISM". In: (). URL: https://www.researchgate.net/publication/352810314_OBJECT_TRACKING_CONTROL_USING_A_GIMBAL_MECHANISM/fulltext/60daa3aaa6fdccb745f0b47b/OBJECT-TRACKING-CONTROL-USING-A-GIMBAL-MECHANISM.pdf.

[18] Stavros Apostolakis. "Design and Implementation of an Embedded Real-Time System for Tracking Directional Antennas from a Moving Vehicle". In: (). URL: https://dias.library.tuc.gr/view/64855.

[19] Stavros Apostolakis. "Embedded System for Tracking Fixed Directional Antennas from Moving Vehicles". In: (). URL: https://dias.library.tuc.gr/view/12789?locale=el.

[20] "TensorFlow Lite for Microcontrollers (TFLM)". In: (). URL: https://developer.arm.com/documentation/109267/0100/ML-software-development-on-Arm-Cortex-M-Processors/TensorFlow-Lite-for-Microcontrollers--TFLM-.

[21] "Object Recognition tutorial". In: (). URL: https://core-electronics.com.au/guides/raspberry-pi/object-identify-raspberry-pi/.

[22] "TensorFlow API Overview". In: (). URL: https://www.tensorflow.org/guide/core.