DE-42 (MTS)

ABU BAKAR, YOUSUF, ZAIN, AZMAT

DESIGN OF QUADCOPTER CONTROLLER WITH COMPUTER VISION FOR OBJECT DETECTION

COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
RAWALPINDI
2024

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DE-42 MTS
PROJECT REPORT

DESIGN OF QUADCOPTER CONTROLLER WITH COMPUTER VISION FOR OBJECT DETECTION

Submitted to the Department of Mechatronics Engineering
in partial fulfillment of the requirements
for the degree of
Bachelor of Engineering
in
Mechatronics
2024

Ayesha

Sponsoring DS:

Dr. Ayesha Zeb

A/P Kanwal Naveed

Industrial Supervisor:

Dir. Moazzam Aziz

Submitted By:

Abu Bakar

Azmat Ali Bakht

Zain Bin Sajid

Muhammad Yousuf Hasnain

ACKNOWLEDGMENTS

We express our sincere gratitude to Almighty Allah for granting us the opportunity to work on this revolutionizing project. We are immensely thankful for the knowledge, skill, and perseverance bestowed upon us, enabling the development of a PID Controlled Flight Controller with Object detection. We hope that this project will potentially advance indigenous UAV technology globally.

Our heartfelt appreciation goes to our parents for their unwavering support, kind words, prayers, and encouragement throughout our academic journey. Without their guidance and belief in our abilities, our success would not have been possible.

Lastly, we extend our deepest gratitude to our esteemed supervisors, Dr. Ayesha Zeb, Ma'am Kanwal Naveed and our industrial supervisor Mr. Moazzam Aziz. Their guidance and mentorship have been instrumental in driving our project to development. We acknowledge their profound expertise, which has significantly influenced our understanding and implementation of the project.

We would also like to acknowledge and appreciate the collective contributions of the faculty members in the Department of Mechatronics. Their guidance, valuable advice, and insightful perspectives have played an integral role in our journey as engineers and human beings.

ABSTRACT

This thesis presents the design and implementation of a comprehensive flight control system for a quadcopter. The system encompasses a 6-DOF mathematical model of the quadcopter, a PID-controlled flight controller, and integration with the quadcopter across various simulation platforms and hardware. The primary objectives of this project include designing and implementing a robust flight control system for a quadcopter, integrating the designed models in both simulation and real-life scenarios, and deploying the designed controller on PX4 hardware to enable RC-based flight.

The system is engineered to be highly customizable and adaptable to diverse scenarios and environments. The PID-controlled flight controller is designed to be robust and efficient, ensuring that the integration with various simulation platforms and hardware facilitates thorough testing and validation in multiple settings.

A significant aspect of this project involved integrating object detection and tracking capabilities using a Raspberry Pi mounted on the quadcopter. This addition enabled the quadcopter to autonomously recognize and track objects in its environment, enhancing its functionality for applications such as surveillance, search and rescue, and delivery. The camera system was successfully integrated with the flight control system, allowing for real-time processing and decision-making based on visual input.

The results of this project demonstrate the effectiveness of the designed flight control system in maintaining the quadcopter's flight stability and safety. The system's adaptability and customization potential make it suitable for a broad range of applications.

Overall, this thesis advances drone technology by offering a comprehensive and automated approach to designing and implementing flight control systems for quadcopters.

Table of Contents

List of Figures

List of Symbols

-PID	Proportional-Integral-Derivative
-DOF	Degree Of Freedom
-ROS	Robot Operating System
-SIL	Simulation In Loop
-RC	Radio Controlled
-UAV	Unmanned Aerial Vehicle
-HIL	Hardware In Loop
-GPU	Graphics Processing Unit
-RAM	Random Access Memory
-IMU	Inertial Measurement Unit
-GPS	Global Positioning System
-RGB	Red Green Blue
-3D	3-Dimensional
-BLDC	Brushless Direct Current
-LiPo	Lithium Polymer Battery
-ESC	Electronic Speed Controller
-BEC	Battery Eliminator Circuit
-PSP	Pixhawk Support Package
-QGC	Ground Control Station
-VTOL	Vertical Takeoff and Landing
-DC	Direct Current
- φ	Roll angle
- θ	Pitch angle

- ψ	Yaw angle
-OpenCV	Open-Source Computer Vision Library
-YOLO	You Only Look Once
-ReLU	Rectified Linear Unit
-R-CNN	Regional Convolutional Neural Networks
-MOT	Multiple Object Tracking
-MSV	Mean Shift Vector
-SVT	Support Vector Tracking
-SVM	Support Vector Machine
-PWM	Pulse Width Modulation
-LED	Light Emitting Diode
-uORB	micro-Object Request Broker
-API	Application Programming Interface
-Kp	Proportional Constant
-Ki	Integral Constant
-Kd	Derivative Constant
-CPU	Central Processing Unit
-FP	False Positive
-TP	True Positive
-FN	False Negative
-AP	Average Precision
-MOTP	Multiple Object Tracking Performance
-MOTA	Multiple Object Tracking Accuracy

List of Tables

Chapter 1 – INTRODUCTION

1.1 Background and Motivation

The rapid advancements in drone technology have led to an increased demand for sophisticated and reliable flight control systems. Among various drone types, the quadcopter has emerged as a popular platform due to its versatility in applications such as surveillance, inspection, and delivery. Ensuring the safety and effectiveness of these applications necessitates the development of robust and efficient flight control systems. This thesis aims to contribute to the advancement of drone technology by designing and implementing a comprehensive flight control system for a quadcopter.

1.2 Problem Statement

Current flight control systems for quadcopters frequently depend on manual tuning of PID controllers, a process that is both time-consuming and requires considerable expertise [1] [2]. Additionally, integrating these systems with the quadcopter's sensors and actuators presents significant challenges, particularly when dealing with customized firmware and hardware configurations. Adding to the complexity, incorporating object detection, and tracking capabilities within the flight control system further complicates the integration process. As a result, the current state of the art lacks a comprehensive and automated approach to designing and implementing flight control systems for quadcopters. The integration of object detection and tracking capabilities aims to address specific challenges related to vehicle and person detection. By leveraging camera technology embedded on the drone, this system enhances situational awareness and enables real-time detection and tracking of vehicles and individuals, thereby enhancing the quadcopter's functionality for applications such as traffic monitoring, security surveillance, and search and rescue operations.

1.3 Objectives

The primary objectives of this thesis are:

1. To design and implement a 6-DOF mathematical model of a quadcopter using MATLAB/Simulink.
2. To design a PID-controlled flight controller using MATLAB/Simulink and integrate it with the modeled quadcopter on ROS/Gazebo for simulation-in-the-loop (SIL) testing.
3. To integrate the designed PID-controlled flight controller with a realistic model of the F450 quadcopter on the FlightGear simulator.
4. To upload the designed controller onto PX4 hardware and fly the drone using RC.
5. To integrate object detection and tracking capabilities using a Raspberry Pi with the quadcopter.

1.4 Scope and Limitations

This thesis focuses on the design and implementation of a comprehensive flight control system for a quadcopter. The scope includes developing a 6-DOF mathematical model, designing a PID-controlled flight controller, and integrating the controller with the quadcopter across various simulation platforms and hardware. However, the thesis does encounter certain limitations, such as the complexity of integrating the designed models with the quadcopter in both simulation and real-world scenarios. Additionally, challenges arise when initializing and configuring sensors and rotors with customized firmware uploaded onto PX4 hardware.

1.5 Hardware

Choosing the right hardware for the right application is one of the most crucial parts of the development of any product because these factors affect the dynamics of any mobile platform such as UAV.

1.5.1 Ground Computer

The ground computer, or the personal computer, has several important functions. The first purpose it serves is development of the controller firmware through Simulink, followed by SIL simulations and code generation. It further allows for uploading the firmware to the flight controller and provides the interface between hardware and software for HIL simulations.
The other importance of the ground computer is that it is utilized for monitoring values and reading data communicated during and after the quadcopter flights, and therefore enabling sensor calibration, parameter tuning and real-time communications.

Choosing the right hardware for the right application is one of the most crucial parts of the development of any product because these factors affect the dynamics of any mobile platform such as UAV.

1.5.2 Frame

When choosing the frame for a quadcopter, the following factors are kept in mind to get the best frame according to the requirements [3]:

- **Size:** The size of the quadcopter is determined by the size of the frame. The frame must be of size enough to house all necessary components, including motors, propellers, battery, and flying controller [3].
- **Weight:** The overall weight of the quadcopter will be affected by the weight of the frame. A heavier frame will make flying the quadcopter more difficult [3].
- **Material:** The frame can be composed of several materials, including carbon fiber, aluminum, or plastic. The lightest and strongest material is carbon fiber, but it is also the most expensive. Aluminum has a good weight-to-strength ratio material. Plastic is the cheapest material, but it is also the heaviest and most brittle [3].
- **Price:** The cost of the frame will depend on its size, weight, and material. Before shopping for the frame, a budget should be constructed [3].

We have chosen an F450 quadrotor frame that is a 450mm frame when measured between opposite rotors. The F450 frame is large enough to carry all the other components. It weighs about 282g which is not too heavy a frame, thus making it an excellent option. The frame is shown in Figure 1 [3].

Figure 1 F450 Quadcopter Frame [3]

1.5.3 Microcontroller

Both the Jetson Nano [4] and the Raspberry Pi 3B [41] are single-board computers capable of object recognition. The Jetson Nano, on the other hand, has some advantages over the Raspberry Pi 3B, making it a better candidate for object recognition applications.

Advantages of Jetson Nano:

- It has a faster GPU, NVIDIA Maxwell [4] than the Raspberry Pi 3B which has the Broadcom VideoCore IV [5]. This improves its performance when executing machine learning and deep learning algorithms for object recognition.
- It has a larger and faster memory with a capacity of 4GB and a speed of 2133MHz [4] than the Raspberry Pi 3B which has 1GB of memory with

900MHz speed [5]. This is significant for object identification applications, which frequently demand a large amount of RAM to hold the machine learning models.

- It consumes less electricity than the Raspberry Pi 3B. This is critical for scenarios where the device will be powered by a battery.

Overall, the Jetson Nano outperforms the Raspberry Pi 3B in object recognition applications due to its more powerful GPU, larger RAM, and lower power consumption.

Figure 2 Image of Raspberry pi 3B [5]

Figure 3 Image of Jetson Nano [4]

1.5.4 Flight Controller

Pixhawk is a popular open-source autopilot system used in drones and other unmanned aerial vehicles (UAVs). It has a triple-redundant flight controller, a GPS receiver, an inertial measurement unit (IMU), and a magnetometer. Pixhawk is also

compatible with a diverse set of sensors and actuators, making it an adaptable platform for a wide range of UAV applications [6].

Here are a few reasons why Pixhawk is an excellent choice for autopilot applications:

- **Opensource:** Pixhawk is a free and open-source hardware and software platform. This means that anyone can help create it and that there is a wide community of users and developers who can help [7].
- **Full-featured:** Pixhawk is a full-featured autopilot system that comes with everything you need to operate a UAV autonomously. A flight controller, GPS receiver, IMU, and magnetometer are all included [7].
- **Versatile:** Pixhawk is compatible with a wide variety of sensors and actuators. As a result, it is a versatile platform for a wide range of UAV applications, including quadcopters, hexacopters, and octocopters [7].
- **Reliable:** Pixhawk is a dependable and well-proven autopilot system. It is employed in a variety of UAV applications, including commercial, military, and research [7].

Pixhawk's position and velocity are provided by the GPS receiver. Its orientation and acceleration are provided by the IMU. Pixhawk gets its heading from the magnetometer. Pixhawk estimates its state using data from the GPS receiver, IMU, and magnetometer. This includes its position, velocity, orientation, and acceleration. Then it utilizes its estimated state to operate the UAV's motors and actuators, allowing it to fly to its goal.

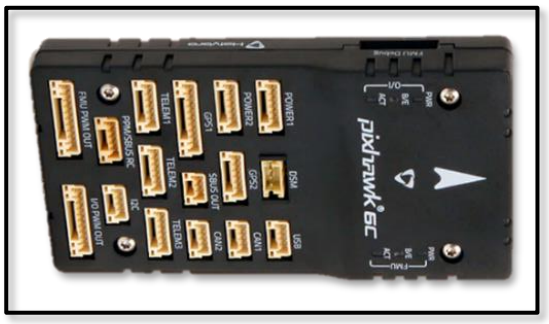


Figure 4 Pixhawk 6c [9]

Pixhawk is a sophisticated and adaptable autopilot system that is suitable for a wide range of UAV applications. It is open source, full-featured, dependable, and works with a variety of sensors and actuators. It is also compatible with Ardupilot which is an open-source and reliable platform for developers who want to autopilot their drones [8].

1.5.5 RGBD Camera

For object detection, three camera sensor options were available, the Etron stereo camera [9], Microsoft Kinect [10], and Xtion Pro Live RGB motion sensor [11] all of which are excellent sensors. The Xtion Pro Live RGB motion sensor, however, has several advantages over the other two sensors.

The following are the advantages of the Xtion Pro Live RGB motion sensor:

- It is more detailed than the other two sensors. This implies that it can capture more information in its photographs and videos [11].
- It covers a larger area than the other two sensors. This implies it can observe a bigger region at the same time [11].
- It can record in-depth information in addition to RGB data. This means that it can produce 3D photos and videos [11].

Because of these benefits, the Xtion Pro Live RGB motion sensor is a better option for object detection than the other two sensors. It can capture more detail and see a bigger area simultaneously due to its better resolution and broader field of view. This facilitates the identification of items using object recognition software. The ability to capture depth information helps the object identification software to generate a 3D representation of the object, which can increase object recognition accuracy.

Table 1 Comparison of Etron, Kinect & Xtion Pro Live [9], [10], [11]

Feature	Etron stereo camera	Microsoft Kinect	Xtion Pro Live RGB motion sensor
Resolution	1280x720	1024x768	1920x1080
Field of view	60°	57°	70°
Depth information	No	No	Yes

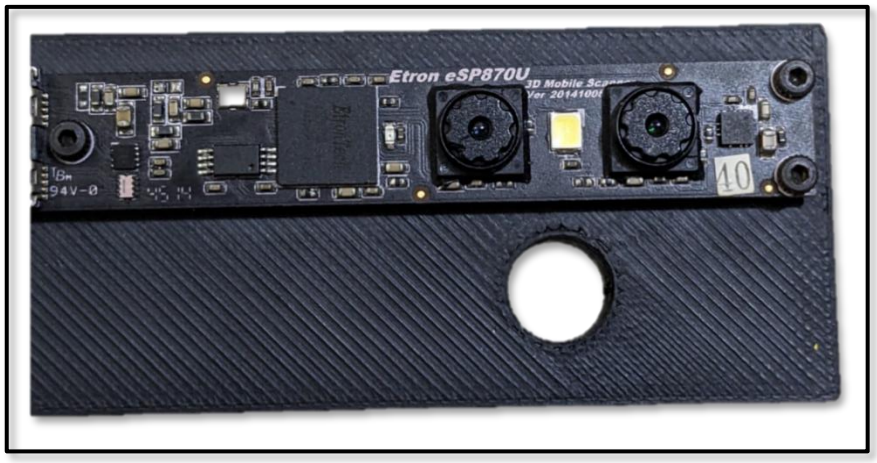


Figure 5 Etron ESP870U

Figure 6 Microsoft Kinect XBOX 360

Figure 7 ASUS Xtion Pro Live Motion Sensor [12]

1.5.6 Actuators

5010-650KV BLDC Motors can operate over the range 11.1V to 22.2V and can take from 10A to 40A. They can produce a thrust of 1000g-4000g. For a 4S LiPo battery which will be discussed later, it can produce 11,100 RPMs [12]. It has a weight of 140g which can be considered a little bit heavy. Because they are powerful and efficient, these motors are ideal for quadcopters. They can generate a lot of thrust, which is necessary for lifting the quadcopter and its payload. They are also efficient, meaning they can create the same amount of thrust with less battery power.

The following are some of the advantages of using 5010-750KV BLDC motors in the quadcopter:

- **Power:** These motors are quite powerful and can generate a significant amount of thrust. This is necessary for lifting the quadcopter and its payload [12].
- **Efficiency:** These motors are also incredibly efficient, which means they can create the same amount of thrust with less battery power [12].
- **Durability:** These motors are extremely tough and can withstand a great deal of abuse. This is especially crucial for quadcopters, which can experience much stress and vibration [12].

Though these motors are very good, they can have some disadvantages as well, such as:

- **Cost:** The cost of these motors is higher than that of other types of motors [12].
- **Weight:** These motors weigh more than other types of motors [12].
- **Noise:** These motors can be very loud [12].

Figure 8 5010-750KV BLDC Motors [13]

1.5.7 Electronic Speed Controller (ESC)

Rudnytsky SIMONK30A ESCs are an excellent choice for controlling the speed of 5010-750KV BLDC motors [13]. They are compact, light, and simple to use. They also have certain characteristics that make them perfect for quadcopter applications, such as:

- **High refresh rate:** SIMONK30A ESCs have a high refresh rate, which means they can react quickly to changes in the throttle signal. This is critical for quadcopters, which must respond swiftly to pilot input to maintain stability [13].
- **Linear BEC:** SIMONK30A ESCs contain a linear BEC, which means they can give a consistent voltage to the receiver and other electronics even when they are under load. This is critical for quadcopters, which require a stable power source to operate securely [13].
- **Low voltage cutoff:** SIMONK30A ESCs have a low voltage cutoff, which means they will automatically turn off the motors if the battery voltage falls too low. This is a vital safety feature that can prevent battery and motor damage [13].

To use SIMONK30A ESCs to control the speed of 5010-750KV BLDC motors, ESCs are connected to the motors and the receiver. ESCs must be calibrated according to the motors.

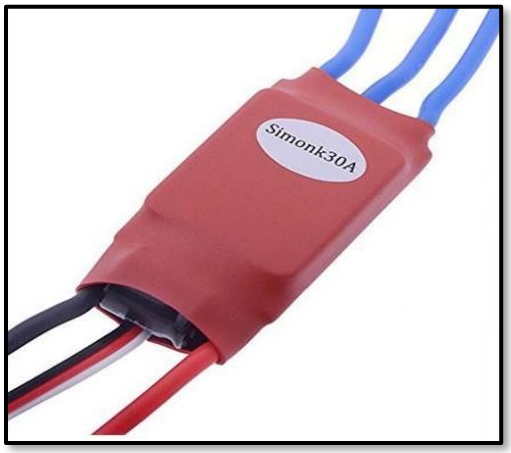


Figure 9 SIMONK30A ESC [14]

1.5.8 Lithium Polymer (LiPo) Battery

The CNHL G+ 5000mAh 4S 70C LiPo battery is a high-performance battery suitable for a wide range of applications such as quadcopters, helicopters, and airplanes [14]. It has a high discharge rate of 70C, which means it can supply a lot of power to motors even when they are under severe load. It also has a large capacity of 5000mAh, allowing it to power long flights or races [14].

The following are some of the advantages of using a CNHL G+ Plus 5000mAh 4S 70C LiPo battery:

- **High discharge rate**: The battery's high discharge rate of 70C means it can give a lot of power to the motors even while under heavy load. This is critical for quadcopters, helicopters, and airplanes, which require a great deal of power to fly.
- **Long capacity:** The battery's 5000mAh capacity implies that it can power long flights or races. This is critical for quadcopters and airplanes that can fly for extended periods of time.
- **High quality:** CNHL G+ Plus batteries are constructed with high-quality components. This implies they are long-lasting and can tolerate a lot of wear and abuse.

Here are some of the specifications of the CNHL G+ Plus 5000mAh 4S 70C LiPo

battery [14]:

- Capacity: 5000mAh
- Voltage: 14.8V
- Discharge rate: 70C
- Weight: 515g
- Dimensions: 38x50x145mm

Figure 10 CNHL G+ 70C 5000mAh Battery [15]

1.5.9 Propeller

The right propellers are crucial for the required thrust produced by the motors. The following factors can help while selecting propellers:

- **Size:** The size of the propeller will affect the thrust and efficiency of the quadcopter. Choosing propellers that are the right size for the quadcopter's motors and frame is essential [15].
- **Pitch:** The pitch of the propeller will affect the speed and thrust of the quadcopter. A higher-pitch propeller will provide more thrust, but it will also be less efficient. A lower-pitch propeller will be more efficient, but it will also provide less thrust [15].
- **Material:** Propellers are made from a variety of materials, such as plastic, wood, and carbon fiber. Carbon fiber propellers are the most expensive, but they are also the lightest and strongest [15].

We have selected 9.5in carbon fiber propellers because they are durable, efficient, have great responsiveness, and are very quiet, making them a good option to use for the quadcopter.

Figure 11 Carbon Fiber Propeller

1.6 Software

1.6.1 MATLAB/Simulink

For designing the controller and programming, we chose MATLAB/Simulink due to its extensive utilization in areas involving aerial vehicles and automobiles. It allows for simplified dynamic modeling, design, and simulation of controllers, along with accessibility towards linking to other software for SIL simulation.[16] Furthermore, the Pixhawk Support Package (PSP) Toolbox, combined with the Advanced Embedded coder and Simulink coder provide the ability to convert the Simulink model to executable C/C++ code. Simulink then also allows for uploading this code to the Pixhawk hardware for HIL simulations and outdoor hardware implementations. [16]

1.6.2 PSP Toolbox

PSP Toolbox is an official Simulink Toolbox which enables designing of controllers, conversion into code and uploading of the designed firmware onto the Pixhawk hardware. It consists of various sensor blocks and communication protocols.

1.6.3 Flight Gear—Flight Simulator

It is an open-source software for flight simulation utilized to observe the flight of a drone in real-time. It works in sync with Simulink; the flight controller designed in Simulink can be simulated, and Flight Gear displays the simulated flight in a 3D environment. Controls can be adjusted, and the effects observed in real-time.

1.6.4 PX4 Software—Source Code

It is a firmware built for flight control specifically tailored to run on the Pixhawk autopilot. Together, these two make up an integrated autopilot system, which is one of the most popular ones for the multi copters.

1.6.5 PX4 Toolchain—Compiling Environment

The PX4 toolchain's main purpose is to be used in conjunction with Simulink to compile the PX4 source code alongside the designed controller into a ". px4" firmware file, which can then be uploaded onto the Pixhawk controller.

1.6.6 QGroundControl (QGC)—Ground Control Station

QGroundControl is utilized to carry out pre-flight checks (frame assignment, sensor calibration, RC setup and tuning of parameters) for the Pixhawk autopilot and connected sensors. It can also be used to receive live flight data and send commands for control of the aerial vehicle using wireless radio telemetry. [17]

1.6.7 CopterSim—Real-Time Motion Simulation Software

This software provides Real-time motion simulation for the Pixhawk autopilot. Different parameters can be adjusted for the selected multi copter, and the Pixhawk hardware can be connected for performing HIL simulations.

1.6.8. 3DDisplay—3D Visual Display Software

3DDisplay is a real-time 3D visual display application that receives flight data from the CopterSim simulation model via UDP protocol. This allows for the real-time visualization of a multi copter's attitude and position within a 3D environment. CopterSim and this 3D display software together form an integrated hardware-in-the-loop (HIL) simulation platform. [18]

Chapter 2 – LITERATURE REVIEW

2.1 Six DOF Quadrotor Mathematical Model

In recent years, a plethora of hovercrafts have been utilized as UAVs, including fixed-wing airplanes, airships, and helicopters, among others [19]. Nonetheless, the aircraft that have garnered the most attention for remote investigation applications are multirotor copters, specifically quadcopters, the reason being that these aircraft boast various advantages over others. They have a straightforward structural design, excellent flight capacity and maneuverability, especially their capability for vertical take-off and landing (VTOL) and ability to hover [20]. A quadcopter obtains flight through four DC motors, each controlling an identical rotor, which is upward-facing and positioned in a square-shaped arrangement at equal distance from the center of mass of the quadrotor [21]. Each motor's speed can be controlled independently of the other, permitting balanced alteration of the rotors' speed, therefore generating thrust and accelerations in the desired directions. This allows quadrotors more stability.

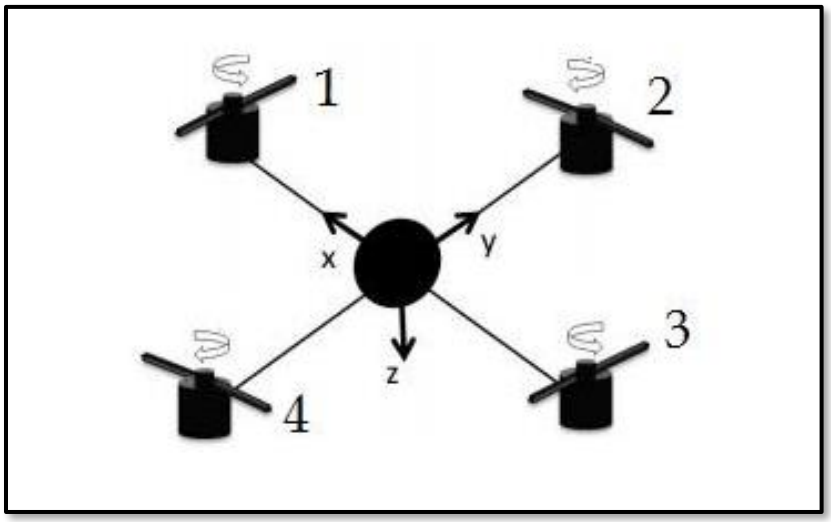


Figure 12 Quadrotor structure [12]

A quadcopter is considered a six-degrees-of-freedom (6-DoF) system, implying that it moves in 6 directions and 6 variables are required to show its altitude and attitude in space (x, y, z, φ, θ and ψ). The variables x, y and z represent the translational motion of the quadcopter. A fixed reference frame is used for this purpose and x, y and z respectively

indicate the distance of the copter's center of mass along x, y and z axes as depicted in Figure 1, from this frame. The remaining three variables are the Euler angles, used to indicate the orientation of the quadcopter and used for rotational movement. (φ) represents the angle about the x axis and is termed roll angle, (θ) represents the angle about the y axis and is termed pitch angle and (ψ) represents the angle about z axis and is termed yaw angle. These angles are represented in Figure 13.

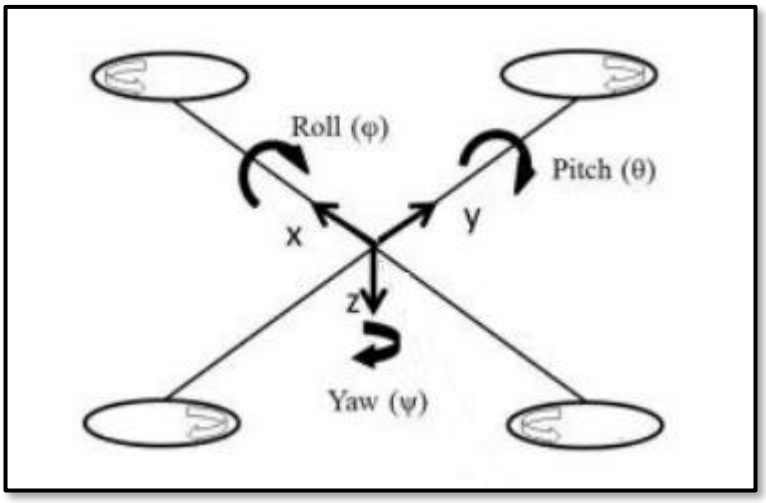


Figure 13 Euler angle representation for quadcopter [12]

For a quadcopter, the roll and pitch angles are called attitudes, the yaw angle is called heading and the vertical distance from the ground (z-axis) is termed as the altitude.

The quadcopter is an under-actuated system, which means that it has a lower number of actuators as compared to its Degrees of Freedom, it only allows for direct control over four DOF, rather than 6. These 4 DOF consist of altitude, attitude, and heading. The other 2 DOF can be controlled through 2 of the rotation angles as roll and pitch rotations generate aircraft movement along the Y and X axes, respectively [22].

Most researchers start with a basic dynamic model of the quadcopter, which is derived by taking the net external forces and torques into consideration. These are usually conventional. However, some have incorporated aerodynamic effects [23], [24], which vary according to various applications and environments. [21] and [25] make use of both Newton-Euler equations and Euler-Lagrange equations to derive differential equations that represent the dynamics of the quadcopter. Furthermore, the derivations of these equations for a quadrotor model have been well elaborated by Beard in [26], Corke in [27], Sidea in

[28], and Bresciani in [29]. Both methods of deriving these dynamical equations are equivalent, thus also producing an equivalent result. Bouabdallah in [25] incorporates various external effects into his model, including thrust force, hub force, drag moment, rolling moment, and ground effect. However, all these studies have focused on the plus-configuration of the quadcopter, as it is easier to explain the flight mechanisms in this configuration. A quadcopter may also be used in a cross-configuration, as depicted in Figure 14.

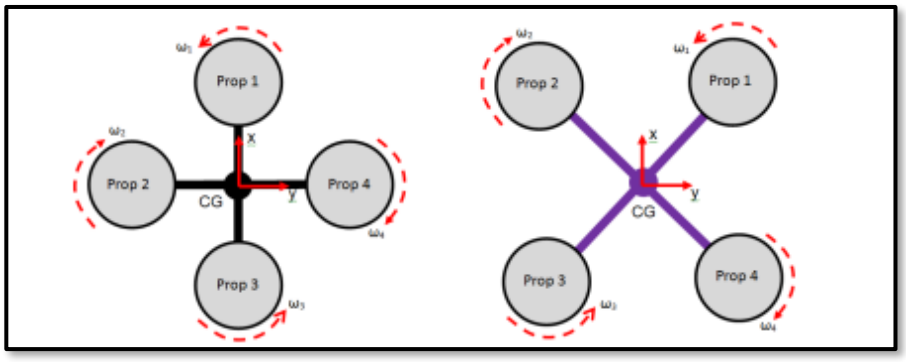


Figure 14 Quadcopter in plus- and cross-configurations [31]

The dynamics regarding thrust and yaw are similar for both, albeit there being different dynamics for roll and pitch.

Defining the reference frames is necessary for deriving quadcopter kinematic equations. Ruth Tesfaye [30] Mehmet Ikulak [31], and Alexander Lebedev [32] use the fact that quadcopter motion is described based on two frames, body frame and inertial frame [33] [34]. However, some researchers like Wei Zhong Fum [35] take the assumption that an additional frame corresponding to quadcopter motion is required, known as the vehicle frame.

Ultimately, the study in [36] puts together all this research to derive the equations for the dynamical model of the quadcopter and creates an adequate model, which they further use to design a controller. They include the effects of the gravitational force, the thrust forces of the propellers, the hub torque acting on the propellers, and the drag force acting due to the non-elliptical shape of the quadcopter, which is an aerodynamic effect. However, other gyroscopic effects like ground effect and gyroscopic effect have been ignored.

2.2 Cascaded Loop PID Control

After modeling the 6 DOF model of the quadcopter, a cascade control loop is used to linearize the non-linear model. Then linearized equations are used to design a PID-controlled drone in all 6 degrees of freedom. A cascaded control loop architecture is a control strategy that involves the use of multiple PID (Proportional-Integral-Derivative) controllers in a hierarchical or cascaded manner to control a complex system. The goal of cascaded control is to enhance overall control performance and reduce interactions between various control loops. Linearization is beneficial for the drone as it will help the drone maintain a set altitude, enabling it to capture clear images during flight. A common application of cascaded control is to control a primary process variable while using a secondary control loop to manage disturbances or improve the response of the primary loop. The linearization will be performed in Simulink, where linear equations will be used to compare the output of both the linear and non-linear systems. In this architecture, there will be two loops: an inner loop for controlling pitch rate and an outer loop for controlling vertical acceleration.

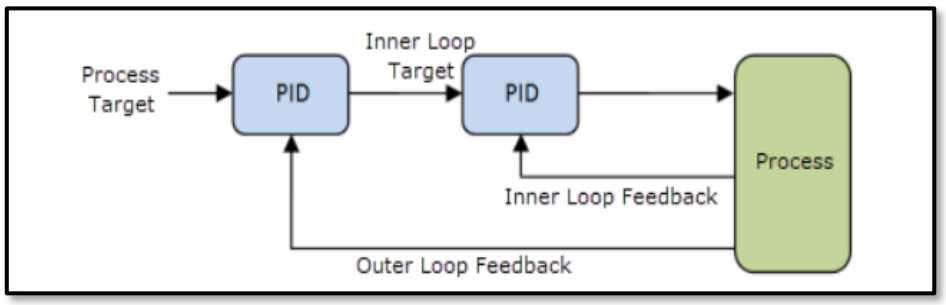


Figure 15 Cascaded Loop Control [38]

The foundations of control theory, on which the PID controller is based, can be traced back to the work of French mathematician Pierre-Simon Laplace in the 18th and 19th centuries. Laplace introduced differential equations and Laplace transforms, which are fundamental tools in control theory [37]. In the 19th century, the concept of the proportional constant, represented by the "P" in PID, was introduced by James Clerk Maxwell. The theoretical basis for integral control, denoted by the "I" in PID, was established by researchers Nicholas Minorsky and Harold Stephen Black in the early 20th century. The "D" in PID, signifying derivative control, was incorporated into control theory by researcher Albert Rea

during the early 20th century. In 1911, Elmer Sperry utilized the first-ever PID controller to automate a ship's steering mechanism. Then, in 1914, Laurence Sperry, Elmer Sperry's son, demonstrated the use of a gyroscope and ailerons to enable an aircraft to maintain a level position in the face of disturbances. In 1912, the Foxboro Company achieved a significant milestone in process control by introducing the first on-off controller (binary controller) featuring a calibrated set-point scale. In 1927, Foxboro introduced a pneumatic "narrow band" proportional controller, enhancing the precision and control capabilities of industrial processes [38]. Pneumatic control systems utilize air pressure to regulate various aspects of industrial processes. In 1922, Nicolas Minorsky implemented the PID control architecture to automate ship steering [39]. This work laid the foundation for the application of PID control not only in ship steering but also in the control of various dynamic systems. In 2008, the Department of Computer and Electrical Engineering at the Missouri University of Science and Technology conducted research on a PID-controlled drone that used cascaded loop control to enhance pitch control effectiveness and reduce pitch response lag by adjusting the integral and derivative gains of the pitch PID loop [40].

A group of bachelor's students from the Department of Computer and Electrical Engineering at the University of Khartoum designed a cascaded loop control architecture for an effective autonomous PID-controlled flight of a drone. Results of implementing the cascaded loop control system can be seen in the figure 17 [41].

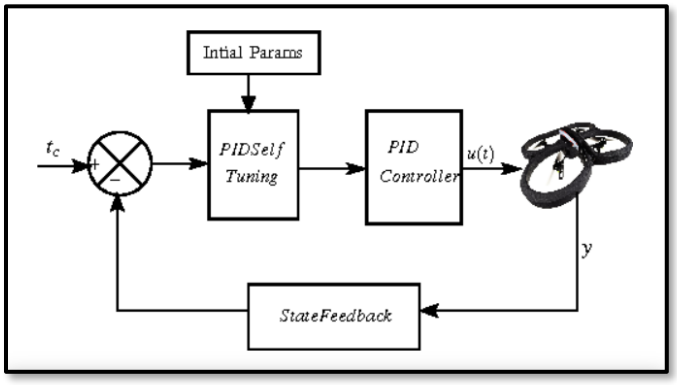


Figure 16 General PID loop for Drones

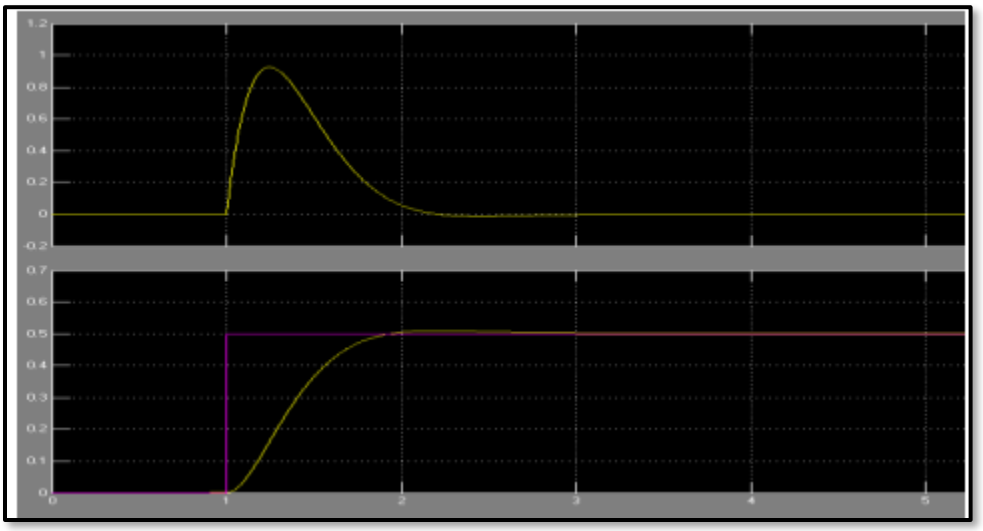


Figure 17 Altitude velocity and Position response after linearization

2.3 Software-in-the-loop (SIL) Simulation:

In this project, after designing the PID-controlled flight controller in Simulink, the controller will be linked with a quadcopter model in ROS/Gazebo for simulation and validation to ensure it operates correctly. Additionally, the designed controller will undergo final testing on FlightGear software before being uploaded onto hardware to ensure seamless operation in real-world environments, given FlightGear's realistic environment options and flight dynamics. These software tools are utilized for validating the flight controller's design due to their efficient communication with Simulink.

Similar projects have been undertaken previously, focusing on simulating simple PID-controlled algorithms designed in Simulink with Gazebo.[42] However, in this project, the goal is to design a complex PID-controlled flight controller that considers environmental and real-life factors. Moreover, a group of students previously worked on a similar project, implementing, and integrating the PX4 controller with a quadcopter modeled in Gazebo [43]. In contrast, this project aims to implement a PID-controlled flight controller from scratch, simulate it in Gazebo, and then upload it onto PX4 hardware.

2.4 Object Detection in RGB images

Object detection in real-time scenarios, particularly in the context of unmanned aerial vehicles (UAVs), has gained substantial attention in recent years due to its potential applications in surveillance, monitoring, and autonomous navigation. A comprehensive analysis of the current state-of-the-art methodologies for object detection in RGB images, with a specific focus on the integration of the YOLO (You Only Look Once) algorithm and the OpenCV (Open-Source Computer Vision Library) for real-time applications on autonomous UAVs, is discussed below. The review will encompass pre-processing and post-processing techniques to understand the complete pipeline for efficient object detection. The dilemmas concerned with 2D object detection are not new, they exist from the invention of computer vision. It is critical to highlight that there is no general agreement in the books on terminologies like image recognition, annotation and labeling of images, detection, clustering, and localization which are frequently described in several ways. [44].

• Object detection involves identifying whether any objects from specified categories are present in an image and, if so, determining their spatial locations and extents.

• On the other hand, Object classification and categorization focus on detecting the presence of objects from pre-defined classes in the image without pinpointing their locations.

• Object recognition, however, includes both identifying and localizing all objects within an image[45] thus combining elements of both image classification and detection [46].

2.4.1 Pre-Processing Techniques

• Image Enhancement and Filtering [47]: Various techniques to enhance the images, such as contrast adjustment and histogram equalization, have been used to improve the quality of input images. Researchers like Jain and Zhang have emphasized the significance of these techniques in enhancing the confidence rate of subsequent object detection algorithms.

• Image Registration and Stabilization [47]: Image registration techniques, including feature-based and intensity-based methods, have been employed to align images and compensate for UAV motion. Studies by Li et al. (2020) and Wang et al. (2021) have highlighted the importance of image registration in ensuring consistent input for real-time object detection algorithms.

• Feature Extraction and Augmentation [47]: Feature extraction techniques, including SIFT and SURF, have been utilized to extract robust features for improved object detection. Researchers such as Lowe (2004) and Bay et al. (2008) have extensively discussed the effectiveness of these feature extraction methods in the context of UAV-based object detection.

2.4.2 Review of Object Detectors

A computer cannot make sense of images, yet it can perform seemingly intelligent tasks like identifying and tracking objects in a video. Object detection algorithms achieve this by sectioning the image and performing classification and localization operations on it. Computers can pass labelled image data for several classes of objects through a neural network and make it constantly adjust its weights and biases at the nodes until the error in the predictions with respect to the labelled input is minimized. After training the neural network in this manner, it can predict different classes of objects in any new image input into it.

To localize an object within an image, the image is divided into rectangular segments represented by bounding boxes, and these are fed through the same classifier network which returns a confidence score for that image segment containing a particular object. The network outputs multiple predictions which are then filtered using algorithms like non-maximum suppression. The result is a single remaining bounding box which is most likely to contain the object of interest. Object detectors can be broadly classified into traditional detectors and deep learning-based detectors (Figure 18).

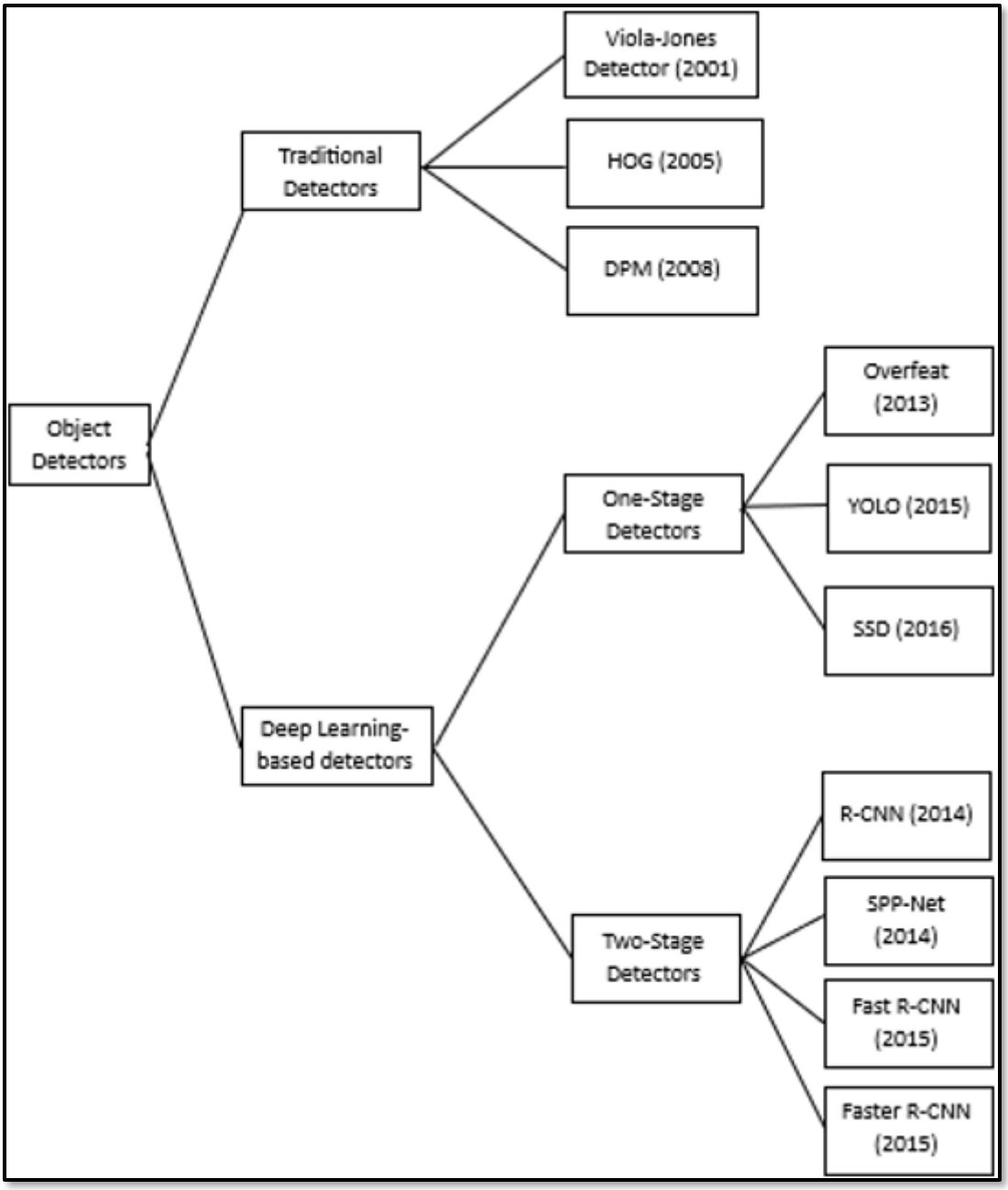


Figure 18 Classification of Object Detectors

Early object detection algorithms relied on a phase of handcrafted feature extraction, aiming to develop sophisticated feature representations that captured the essence of the image. During that period, numerous adjustments were made to enhance accuracy, achieve invariance to various geometrical and spectral factors,

and attain near real-time performance, culminating in a plateau around 2010. The groundbreaking work by Krizhevsky et al. in 2012 [48] which includes 5 convolutional layers and 3 fully connected layers, utilized two Graphics Processing Units (GPUs) running different layers of the network and communicating only at certain layers. This work introduced modern techniques such as data augmentation, multi-GPU training, Rectified Linear Unit (ReLU) activation, max-pooling layers for down sampling, and dropout for regularization, achieving a top-5 error rate of 15.3% on the ImageNet Large Scale Visual Recognition Challenge[49]. Since then, various architectures have been proposed to enhance accuracy in this domain.[50]. Notable contributions are reported below:

• R-CNN [51] (2014) A two-stage object detector that introduced region proposals for potential object locations using selective search, with separate feature extraction for each proposal, resulting in high computational load.

• Faster R-CNN [52] (2015Enhanced the speed of R-CNN by processing the entire image as input and introducing RoI pooling layers, improving both accuracy and speed.

• Mask R-CNN [53] (2017) Extended Faster R-CNN by adding a branch for predicting object masks alongside bounding box recognition.

• YOLO [54] (2016) A single-step object detector that significantly increased computational speed by using a single feature map for object detection, dividing the image into a grid for this purpose, like Faster R-CNN. It was proposed by Joseph Redmon in 2015. It performs feature extraction from the input image using the CNN-based architecture called Darknet as its backbone. The entire input image is divided into several grid cells, and bounding boxes are generated for detection in every grid cell. As a result, multiple bounding boxes are generated, and each has an associated class probability. To deal with objects of different sizes, YOLO uses anchor boxes, which are predefined reference templates of different sizes and aspect ratios for bounding box generation. After all the bounding boxes are generated, YOLO uses the technique of non-maximum suppression to only keep those detections that have a higher probability than a certain threshold value, and the rest are removed. The

YOLO algorithm became increasingly popular due to its simplicity and applicability in real-time scenarios. Hence, in subsequent years, its pioneers introduced many improved versions, and newer versions are still being introduced. These include YOLOv1, YOLOv2, and so on. Each of these comes in architectures of variable sizes having different numbers of trainable parameters. The lighter versions with fewer trainable parameters are more suitable for faster, real-time applications, even though they are a little less accurate. The YOLO algorithms still struggle to show the same performance for detecting small and crowded objects as they did for detecting larger objects captured from a clearer point of view.

2.4.3 Aerial Datasets

The datasets for aerial object detection and tracking are typically acquired by low-altitude drones. The parameters in these datasets can vary greatly based on the types of sensors used for the collection of data. Moreover, the number and position of targets, capturing angle, and altitude can all change drastically across the frames of a video. The UAV123[46], ALOV300++, Temple Color 128 [55] and VisDrone [56] are some popular aerial datasets. These are very diverse datasets comprising a wide variety of objects from urban and wildlife environments. Table 1 below shows some of these datasets and the no. of elements they have.

Table 2 Common Aerial Video Datasets

Name of Dataset	**No. of images/videos**
VOT2017 [50]	60 (21,000 frames)
VisDrone2019 [49]	288 (261,908 frames)
UAV20L [51]	20 (58,670 frames)
UAV123 [46]	120 (112,578 frames)
Temple Color 128 [48]	129 (55,346 frames)
ALOV300++ [47]	314 (151,657 frames)

2.4.4 Review of Trackers

Object tracking is a crucial task in deep learning/computer vision and has numerous applications in various fields. The plots of the trajectories of the objects from a video can be used to form meaningful conclusions about a scenario. Examples of this include monitoring traffic, crowd surveillance, and marine wildlife surveillance. Multi-Object tracking (MOT) algorithms usually involve two steps: detection, and data association. In detection, box-predictions are generated for the objects involved in each and every frame. Then, the stage of data association compares matching of the predicted boxes of the similar objects/classes across multiple frames based on their appearance and motion and assigns them the same ID number. Tracking from UAV can be challenging because of the moving camera on the UAV.

Most modern state-of-the-art modern trackers are based on some type of fundamental tracking strategy. These can be broadly classified into three categories- point tracking, kernel tracking, and silhouette tracking. Of these, generally the kernel-based tracking methods demonstrate higher accuracy, however, point tracking has a very less computational cost. The major algorithms belonging to these categories, as shown in Figure 19 below, have been discussed in the following sections.

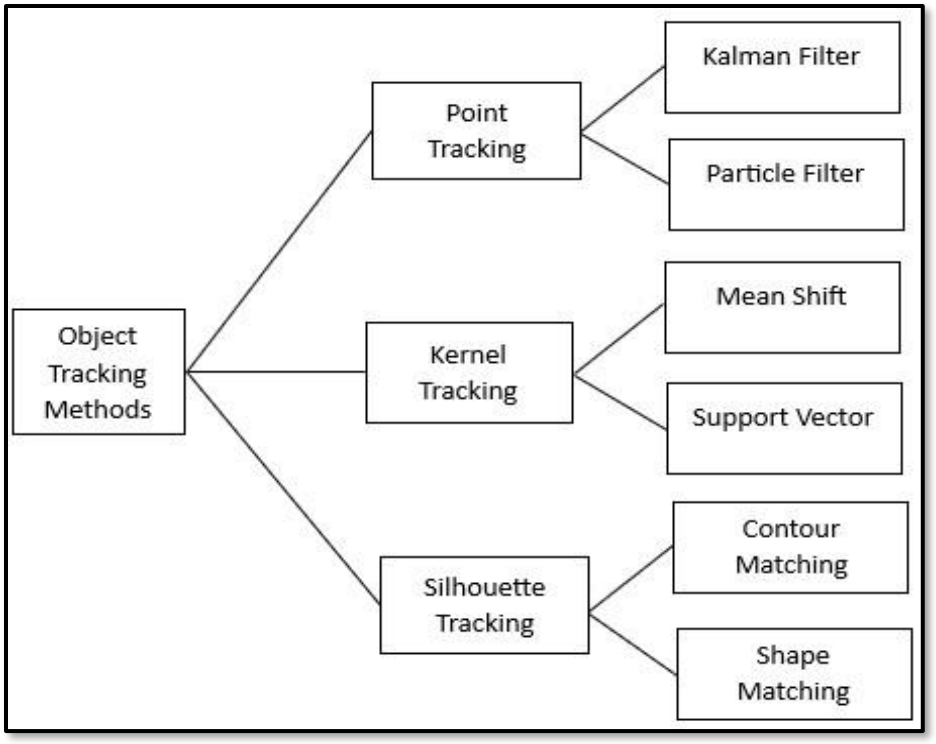


Figure 19 Classification of Object Trackers

2.4.4.1 Point Tracking Methods

•**Kalman Filter**[57]: was developed in the 1960s by Rudolf Kalman and has since become one of the most important state estimation algorithms for tracking applications. It is a recursive algorithm that uses the state transition model for prediction and then corrects it using the true measurement once the new state of the system is obtained. Since it needs to store information about only the immediately previous state of the system to predict the new state, it is computationally very light. It takes an initial input state and outputs the estimate uncertainty along with the next state estimate. Because of the fewer computations involved, it can be used in real-time tracking applications. Include reference of tracking in noisy images using Kalman filter.

•**The Particle Filter**[58]: was introduced in 1993 as an extension of the Kalman filter to compute state estimation for non-Gaussian systems. It consists of a set of weighted particles, which represent a hypothesis about

the state of the system. The first step of the algorithm is called importance sampling, in which the particles are propagated based on assigned weights and their fidelity to the true measurements. The particles with greater weight are more likely to be resampled in the next step. In this way, the system gradually attains a closer approximation to the true state of the system. Both the Kalman filter and the Particle filter form the basis of many tracking applications, however, the Particle filter-based system is computationally more complex. Include reference for good results in image with occultations.

2.4.4.2 Kernel Tracking Methods

Within the framework of machine learning algorithms, kernels are mathematical functions that transform input data into higher dimensional space and compute the similarity between different sets of data points. By transforming data into higher dimensional space, its features become more apparent and separable for the purpose of analysis. There are different types of kernels, like linear kernels, polynomial kernels, sigmoid kernels, etc., which can be used to compute the similarity between two different sets of data, e.g., image frames. Hence, these are employed in designing algorithms for tracking. The following two algorithms described below are kernel-based methods of tracking.

- **The Mean Shift Vector** indicates the direction of the maximum increase in the similarity between the target object and the image region in question. The algorithm[59] iteratively computes this vector and eventually converges to the target's location. To start with, it takes an initial location, computes the feature or color histogram of the region, and defines a kernel function to assess the similarity of the target region with the neighboring regions. The weighted average of the histogram gradients indicates the direction of maximum similarity improvement, as encoded in the mean shift vector. The target's location is gradually updated by shifting it in the

direction of this vector until convergence is achieved. The mean shift algorithm is robust to changes in appearance, illumination, and occlusions, therefore it can handle complex tracking scenarios. Include reference for real time application of MS.

• **Support Vector Tracking (SVT)** works on the basis of support vector machines (SVMs)[60]. SVMs are supervised learning algorithms employed to address both classification and regression models. The aim is to find the planes or boundaries in the data based on which different classes of objects can be classified. SVT combines kernelized correlation filters with SVMs. To start with, it first trains a binary classifier that can differentiate between target objects (positive samples) and backgrounds (negative samples). Then kernel functions map the data points into a higher-dimensional feature space. During tracking, kernelized correlation filters estimate the target's position in the frame and measure the similarity between the target model and search regions. The target location is then predicted based on the maximum response of the filter. Overall, SVT combines the discriminative abilities of the SVMs and the robustness of correlation filters in dealing with challenging tracking scenarios. Include reference for handling partial occlusions, and training.

2.4.5 Image Datasets

The datasets for object detection and tracking are typically acquired by low-altitude drones. The parameters in these datasets can vary greatly based on the types of sensors used for the collection of data. Moreover, the number and position of targets, capturing angle, and altitude can all change drastically across the frames of a video. The UAV123[61], Temple Color 128[62] and OIDv7[63] are some popular aerial datasets. These are very diverse datasets comprising a wide variety of objects from urban and wildlife environments.

2.4.6 Embedded Platforms Used for Real-Time Implementation

For deploying tracking applications to real-world scenarios, algorithms, after they have been tested and optimized, must be ported onto embedded platforms. The platform used is.

2.4.6.1 Raspberry Pi

Although not as powerful as the Nvidia Jetson boards, the Raspberry Pi provides a cost-effective solution. Owing to their less computational capacity, they are more suitable for lightweight computer vision applications with extremely optimized models.

Most practical object detection and tracking applications require extensive computational resources and storage capacity. For tasks where real-time processing is not required, the captured data can be transmitted to ground-based stations for processing. This helps save energy and battery power of the device; however, the data transfer can also make the process slow. Therefore, for applications requiring real-time processing, the UAV must be equipped with sensors and processing boards to perform faster operations.

The computational capacity of most embedded systems poses a limit to the complexity of the algorithms that can be used during onboard operations. Therefore, lighter versions of the popular deep learning algorithms have been proposed for real-time operation from UAVs. Examples of such detection models include, TensorFlow-Lite, UAV-Net [64], which is based on SSD architecture and is adapted to the unique features of aerial imagery, and the lighter version of Faster-RCNN, which uses a lightweight deep CNN feature extractor. A few small, lightweight detectors based on TinyYOLO architecture have also been proposed, which vary parameters like filter size and input image size. Examples of these include SmallYOLOv3[65], TinyYoloNet[66], and DroNet[67].

Chapter 3 – METHODOLOGY

3.1 Configuration of Quadcopter

The quadcopter is an underactuated system, which uses four inputs to control six degrees of freedom. A quadcopter with four rotors offers two design options: X-configurations and +-configurations. The thrust and yaw dynamics are comparable for both; however the roll and pitch dynamics differ. In other words, just two rotors generate the roll and pitch moment in + configurations, whereas four motors generate these moments in X configurations. Figure 20 depicts X- and +- arrangements. Note the direction of the axes that correspond to each setup[68].

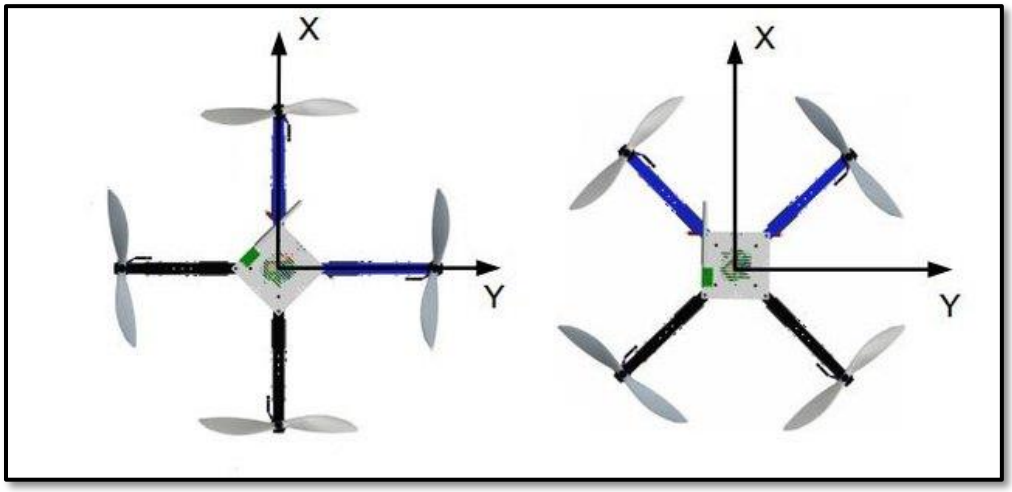


Figure 20 Quadrotor Dynamics (plus configuration) [71]

Vertical upward motion is generated by increasing rotor speed, and vertical downward motion is generated by lowering rotor speed. In the +-configuration, pitch rotation is achieved by adjusting the speeds of the front and back rotors, while roll rotation is accomplished by varying the speeds of the left and right rotors. For X-configurations, the speed of four rotors is adjusted based on the required rotation. This thesis utilizes the plus arrangement under study [71].

3.2 Mathematical Modeling

3.2.1 State Variables

For a quadrotor dynamic model, we are interested in the position and velocity of the quadrotor. So, we can take linear as well as rotational position and velocity values that are then used to determine the position of the quadrotor in the space. And like that 12 state variables are x, y, z position in the coordinate axis and φ, θ, ψ that are rotational angles around the x, y, and z axis respectively, and then we have u, v, w that are velocities in x, y, z directions respectively and lastly, we have p, q, and r that is rotational velocities around each axis.

$$X = [x; y; z; \varphi;\ \theta;\ \psi;\ u; v; w; p; q; r]$$

3.2.2 Measurements

All above-mentioned variables are measurable as recent GPS/IMU sensors provide all the above information.

3.2.3 Dynamics

We take our front direction as positive x-axis in the direction of the motor 1 and our positive Y-axis towards right (motor 2) and we have our Z-axis positive downwards as we know that our gravitational acceleration ($g = 9.8\,ms^{-2}$) is positive in downward direction so it will help us out whenever we are using gravitational acceleration value. To move in a forward direction speed of motors 2 and 4 remains constant while motor 3 speed should be greater than motor 1 speed () and vice versa for backward direction movement. To move towards the direction speed of motor 1 and 4 should remain constant and the speed of motor 4 should be greater than that of motor 2 and vice versa to move in the negative Y-axis direction. To move in an upward direction speed of all motors should be the same and of such value that the thrust produced is greater than the overall weight of the quadrotor and to move in a

downward position, the thrust force should be less than the weight of the quadrotor. These movements are used for take-off and landing purposes.

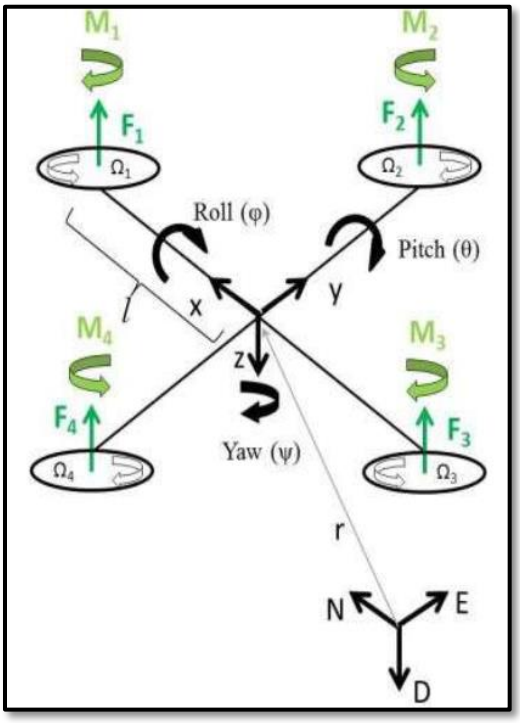


Figure 21 Quadrotor Reference Frame [72]

3.2.4 Control Variables

We have four rotors, and we have a 6-DOF system which means we have a complex underactuated system, so we will need to control these four rotors to move in space. And by that, we get to the controlling variables which are thrust, and moment values about each axis. The thrust causes the vertical motion of quadrotor and moments cause its motion in XY-plane.

$$\vec{u} = [u_1; u_2; u_3; u_4] = [T_{\sum}; M_1; M_2; M_3]$$

3.2.5 Thrust and Moment Distribution among Rotors

Total thrust produced and moment values are given as below,

$$T_{\sum} = \sum_{i=1}^{4} Kt\, \Omega_i^2 \qquad (1)$$

$$M_1 = -Kt \sum_{i=1}^{4} li\, sin(\alpha_i)\, \Omega_i^2 \tag{2}$$

$$M_2 = Kt \sum_{i=1}^{4} li\, cos(\alpha_i)\, \Omega_i^2 \tag{3}$$

$$M_3 = Kq \sum_{i=1}^{4} Oi\, \Omega_i^2 \tag{4}$$

And these equations give us the following matrix that calculates the necessary values of total thrust and moments.

$$\begin{bmatrix} T_{\Sigma} \\ M_1 \\ M_2 \\ M_3 \end{bmatrix} = \begin{bmatrix} Kt & Kt & Kt & Kt \\ 0 & Ktl & 0 & -Ktl \\ -Ktl & 0 & Ktl & 0 \\ Kq & Kq & Kq & Kq \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} \tag{5}$$

This matrix can be used to find out values of motor speeds given values of thrust and moments, for that we have:

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} Kt & Kt & Kt & Kt \\ 0 & Ktl & 0 & -Ktl \\ -Ktl & 0 & Ktl & 0 \\ Kq & Kq & Kq & Kq \end{bmatrix}^{-1} \begin{bmatrix} T_{\Sigma} \\ M_1 \\ M_2 \\ M_3 \end{bmatrix} \tag{6}$$

3.2.6 Equations of Motion by Newton-Euler Method

We can easily calculate equations of motion by using the Newton-Euler method.

3.2.7 Translational Motion

We can get equations of linear acceleration for translational motion using the following equations,

$$\ddot{X} = -\frac{1}{m} (sin\,\psi \cdot sin\,\varphi + cos\,\psi \cdot sin\,\theta \cdot cos\,\varphi) \tag{7}$$

$$\ddot{Y} = -\frac{1}{m} (-cos\,\psi \cdot sin\,\varphi + sin\,\psi \cdot sin\,\theta \cdot cos\,\varphi) \tag{8}$$

$$\ddot{Z} = -\frac{1}{m}\cos\theta \cdot \cos\varphi \cdot T_{\sum} + g \tag{9}$$

And then using values of acceleration from here we can calculate values of translational velocities and coordinates of the quadrotor in the earth frame.

3.2.8 Rotational Motion

Now for rotational motion, we can calculate rates of six rotational variables using the following relations,

$$\begin{bmatrix}\dot{\varphi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & s(\varphi)t(\theta) & c(\varphi)t(\theta)\\ 0 & c(\varphi) & -s(\varphi)\\ 0 & s(\varphi)\,sec(\theta) & c(\varphi)\,sec(\theta)\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{10}$$

As our IMU gives values in relation to earth frame, thus we must transform these values from earth frame to body fixed frame and above relation is used to do so.

$$\dot{p} = \frac{I_{yy} - I_{zz}}{I_{xx}} \cdot qr + \frac{1}{I_{xx}} \cdot M_1 \tag{11}$$

$$\dot{q} = \frac{I_{zz} - I_{xx}}{I_{yy}} \cdot pr + \frac{1}{I_{yy}} \cdot M_2 \tag{12}$$

$$\dot{r} = \frac{I_{xx} - I_{yy}}{I_{zz}} \cdot pq + \frac{1}{I_{zz}} \cdot M_3 \tag{13}$$

These equations give us values of rotational accelerations in earth frame and output from these equations can be integrated to get the values of rotational velocities that can be used to calculate φ, θ, ψ.

3.3 Control Strategy for Path Planning

For control of our quadcopter, we tried and tested two methods, the first is the control strategy designed to be used in conjunction with path planning, and the second strategy is designed for flying the quadcopter using a Remote Control. This section discusses the control strategy required for Path planning.

3.3.1 Nested Control Loop

As we are autopiloting our quadrotor, we give the way points to our quadcopter, and it follows them one by one and finally reaches the destination. These desired points are then fed to the position control block that then calculates the thrust deviation ($\Delta\omega_F$) and desired angles of the quadrotor about each axis. These angles then go to the attitude control block which then calculates the deviation in the motor speed that is required to obtain these angles. All these deviation values are then fed to the motor dynamics block that calculates values of speeds of all motors and then these values are used to calculate the thrust and moment values required. Then these values of thrust and moment are used to calculate the translational and rotational motion parameters of the quadrotor using dynamics equations as discussed above in the model.[69]

3.3.2 PID Controller for a Quadrotor

This study describes the modeling and stabilization of the quadrotor. On open loop, the developed quadrotor responds in a very nonlinear way. It is thus controlled by an external PID controller. This approach is employed as a recursive procedure to create control laws; all the computation stages relating to the tracking error are simplified[70].

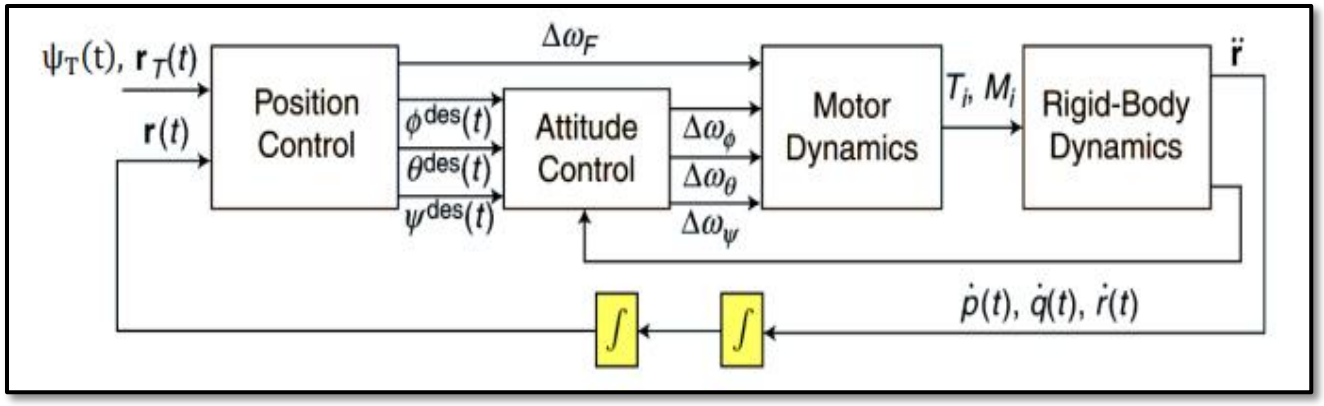


Figure 22 Nested control loops for position and attitude control [73]

Another factor to consider while selecting a controller is the UAV control mechanism. The model can be divided into two components based on the control

system used: one controls the angular rotation of the quadrotor UAV, and the other controls the height(z) of the modeled quadrotor. The equations above use PID control with inputs T, U2, U4, U4 and outputs phi, theta, psi, and altitude z. Although these control methods work well for local analysis and non-linear systems, typically fail when used to global analysis and non-linear systems with non-affine control. The fully actuated subsystem can be controlled and stabilized by applying a control algorithm to all the UAV's output states. For this underactuated system, rate bounded PID controllers must be built to move states to their desired values. Automatic tuning is utilized to linearize the output of the Quadrotor UAV model. The simulation results suggest that the quadrotor can be stabilized using PID controller. The results with and without a PID controller are examined in the next section[70].

3.3.2.1 Position Controller

In our position controller we have a way points profile and angles profile.

3.3.2.2 Way Points Profile

In our way points profile we provide three-way points, first of all we provide height of 2000m and when it reaches this height then using simple switches of SIMULINK we provide it 2000m in positive x direction after that it has to go to the position of [x, y] = [2000, 2000]. When it reaches that point then we command it to reach the position of [x, y] = [4000, 2000]. Throughout this journey the height is maintained at z = 2000m.

3.3.2.3 Desired Angles Profile

After getting waypoints we apply Cascade PID Controller[71] on the desired point and previous point to get the desired value of the angle. In the outer loop we apply PID on the desired way point and previous point and in the inner loop we apply PID to the rate of change of points to the velocity in the respective direction (either x or y).

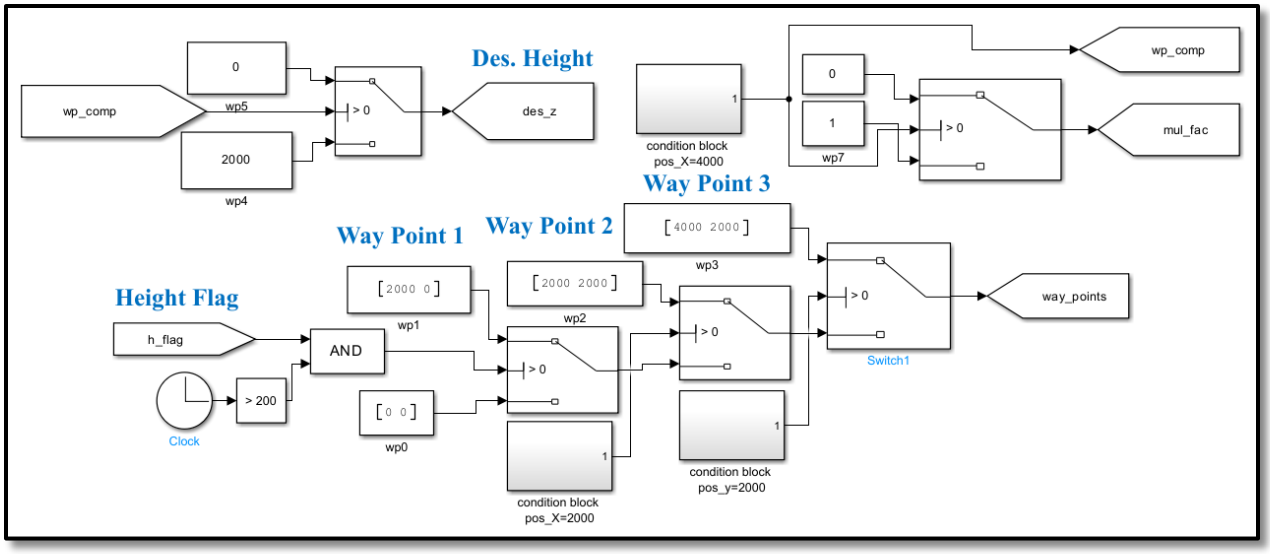


Figure 23 Way Points Profile

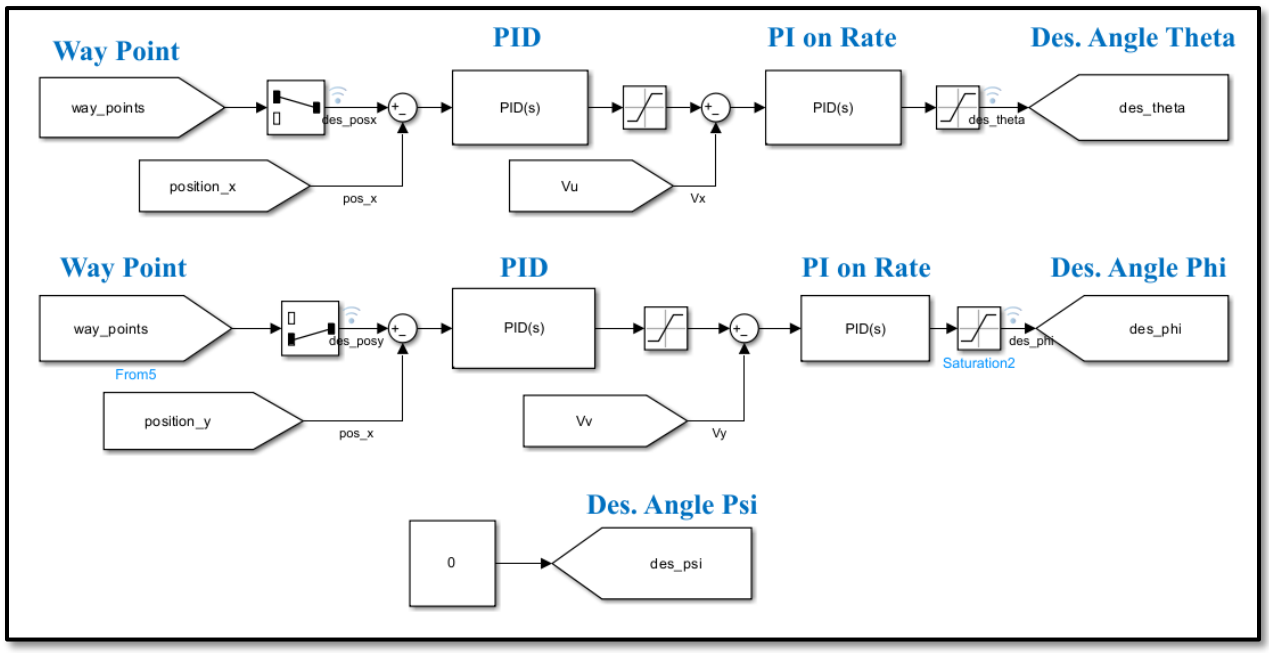


Figure 24 Desired Angles Profile

In Figure 24 upper cascade loop calculates the value of the desired angle θ because to move in a positive x-direction angle θ is responsible and the lower cascade loop calculates the desired angle φ because it causes motion in the positive y-axis. We have assumed our quadrotor faces in one direction $\psi = 0$.

3.3.2.4 Altitude Controller

We prefer to approach the given height before we make any movement in the lateral direction (XY). So desired height (des_z) is compared with our present height then PID is implemented on the error signal. This error signal is actually a change in the height that is then compared with velocity in the negative z direction (upward). This part is done in the inner loop, and it is crucial for the stability of the system. We also need control over our velocity otherwise we may not be able to control the velocity and our quadcopter system may behave in a disturbing way. By using output from our inner loop, we calculate the thrust required to get our quadrotor to the desired height.

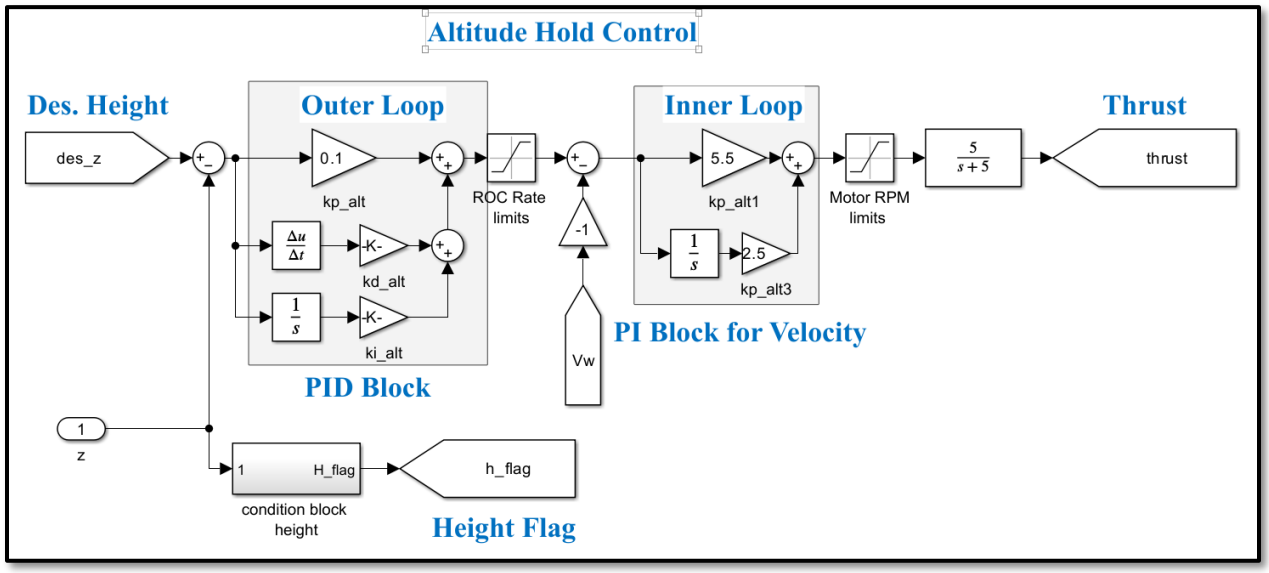


Figure 25 Altitude Controller using PID in SIMULINK

3.3.2.5 Attitude Controller

Throughout our mission, we have assumed that our quadrotor faces in a positive x direction means we have $\psi = 0$. So, we require control over φ and θ.

3.3.2.6 Roll Controller

We take our present value of roll rate and then taking integration we get the value of the present roll angle. This angle is compared with the desired roll angle which is des_phi that was calculated in our desired angles profile. After receiving the error signal, we compare it with the present roll rate this is again a crucial part for stability because otherwise, our rate of angle change may be so disruptive that our system might fail and cause a crash landing damaging our equipment. The error signal from the inner loop gives us the rate of change of velocity that can be considered as rotational acceleration about the x-axis that is multiplied by Ixx to get the required value of tau_phi that is our required moment value about the x-axis i.e M_1.

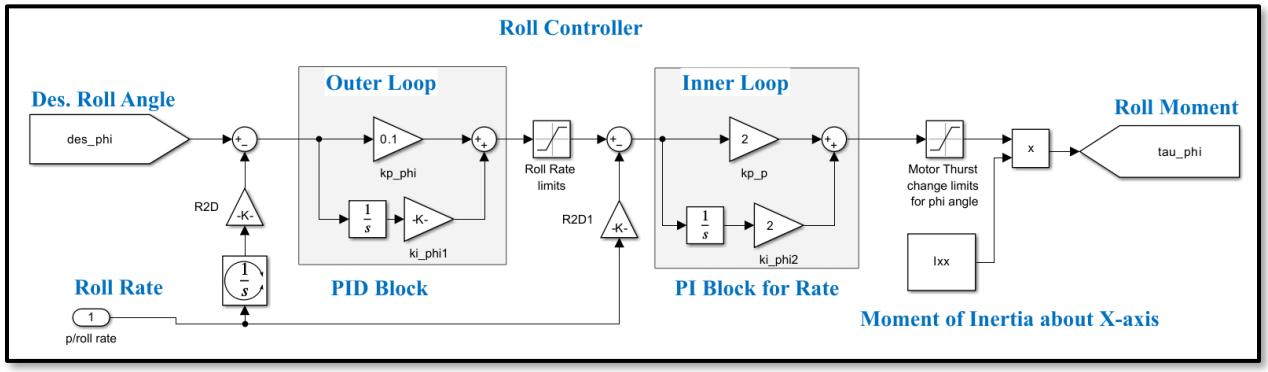


Figure 26 Roll Controller

3.3.2.7 Pitch Controller

In Figure 27 we take our present value of pitch rate and then taking integration we get value of the present pitch angle. This angle is compared with the desired pitch angle which is des_theta that was calculated in our desired angles profile. After receiving the error signal, we compare it with the present pitch rate this is again a crucial part of stability because otherwise, our rate of angle change may be so disruptive that our system might fail and cause a crash landing damaging our equipment like roll angle. The error signal from the inner loop gives us the rate of change of angular velocity about the y-axis that can be considered as rotational acceleration

about the y-axis that is multiplied by Iyy to get the required value of tau_theta that is our required moment value about the y-axis i.e M_2.

3.3.2.8 Yaw Controller

In Figure 28 though throughout our flight we considered our yaw angle to be zero for the case when we are given the yaw angle, we can control it just in a way we did for roll and pitch angles. This gives us the value of M_3.

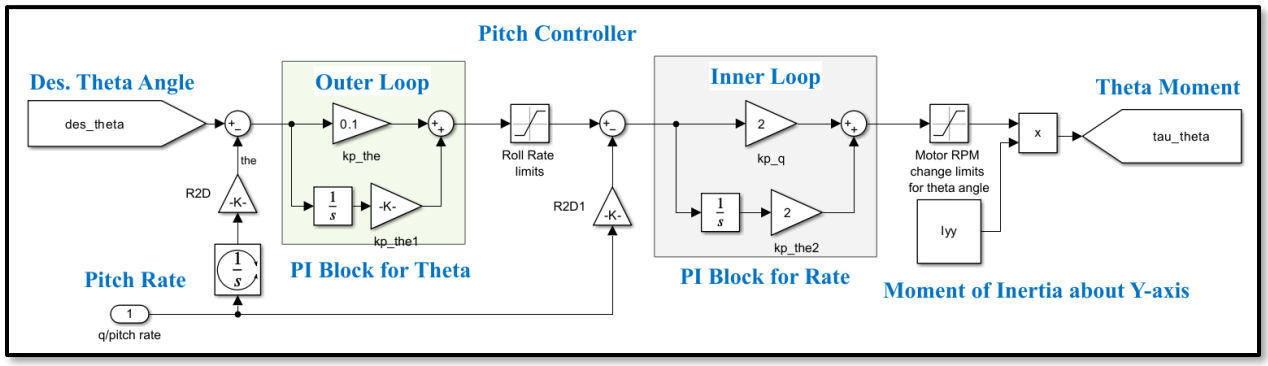


Figure 27 Pitch Controller

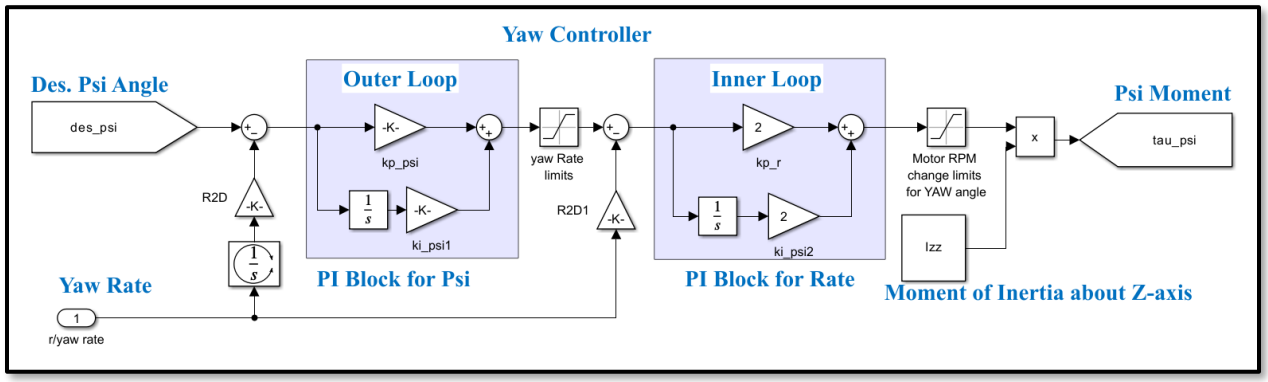


Figure 28 Yaw Controller

3.3.2.9 Motors Dynamics

After we have calculated the values of all our control parameters, we are able to calculate the speeds of all the motors. That are then fed to our 6-DOF model that will calculate values of our state variables.

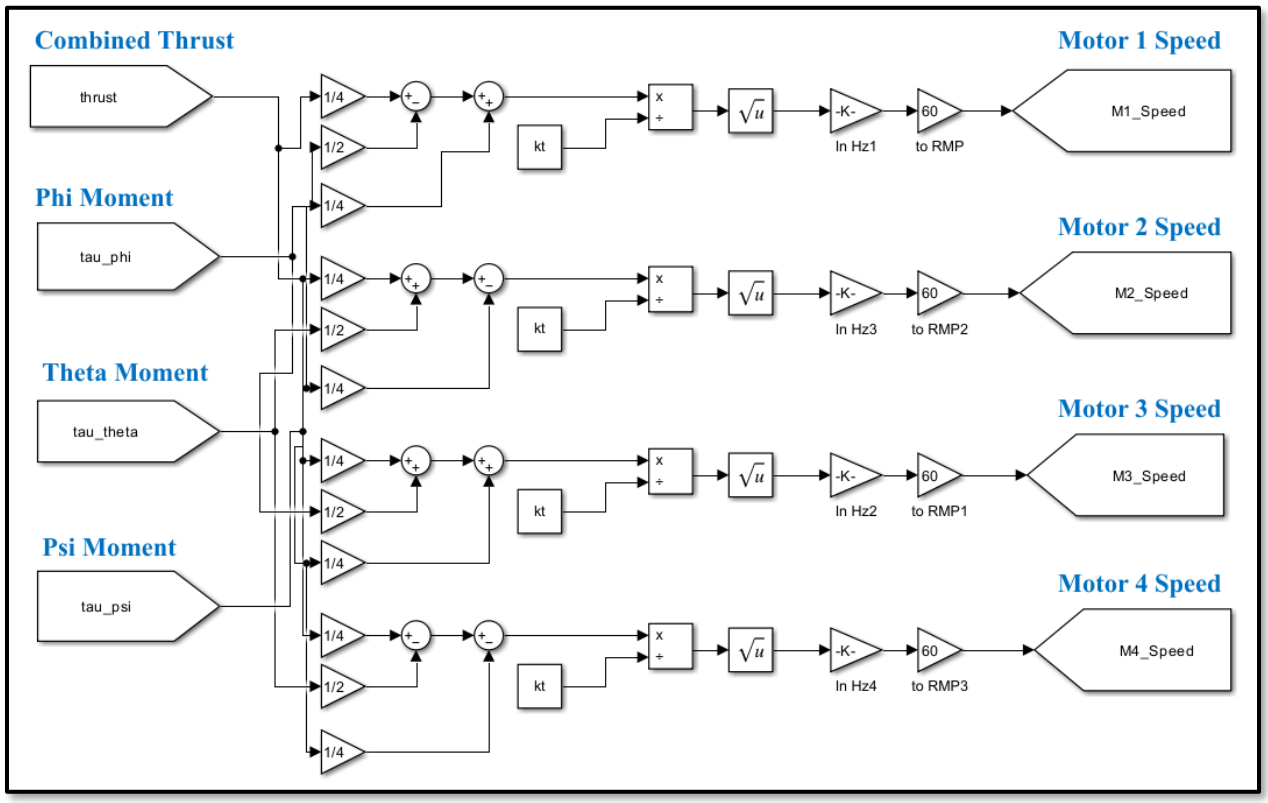


Figure 29 Motors Speed Calculation

3.3.2.10 Plant Function

We have our plant function in which we have implemented all the equations stated in sections 3.5.2.1 and 3.2.5.2. This is where we get our state variables calculated.

```
function x_dot = fcn(x,u,Thrust)


% All the parameters required
g = 9.81;  % Gravitational acceleration
R = 0.125; % Radius of central mass
b_m = 2.05;  % Body mass
a_l = 0.225; % Arm length
r_m = 0.112; % Rotor mass
```

```
% Moment of Inertia terms
Ixx = ((2*b_m*R^2)/5)+(2*r_m*a_l^2); % Moment of inertia for sphere = (2/5)mr^2
Iyy = ((2*b_m*R^2)/5)+(2*r_m*a_l^2); % Moment of inertia for pendulum = mr^2
Izz = ((2*b_m*R^2)/5)+(4*r_m*a_l^2);
kt = 0.00263;
% Output variable that gives derivative of state variables
x_dot = zeros(12,1); % States [x y z phi the psi u v w p q r]
% Thrust = kt*(w(1) + w(2) +w(3) + w(4));

% All components of Absolute rotational velocity
phi_dot = x(10) + sin(x(4))*tan(x(5))*x(11) + cos(x(4))*tan(x(5))*x(12);
theta_dot = cos(x(4))*x(11) - sin(x(4))*x(12);
psi_dot = sin(x(4))*(1/cos(x(5)))*x(11) + cos(x(4))*(1/cos(x(5)))*x(12);

% All the linear acceleration terms
u_dot = (-1/b_m) * (sin(x(6))*sin(x(4)) + cos(x(6))*sin(x(5))*cos(x(4))) * Thrust; %
Acceleration in X direction
v_dot = (-1/b_m) * (-cos(x(6))*sin(x(4)) + sin(x(6))*sin(x(5))*cos(x(4))) * Thrust; %
Acceleration in Y direction
w_dot = ((-1/b_m) * (cos(x(5))*cos(x(4))) * Thrust) + g; % Acceleration in Z
direction

% All the rotational acceleration terms
p_dot = (((Iyy-Izz)/Ixx)*x(11)*x(12)) + (1/Ixx)*u(1); % Acceleration about X
direction
q_dot = (((Izz-Ixx)/Iyy)*x(12)*x(10)) + (1/Iyy)*u(2); % Acceleration about Y
direction
r_dot = (((Ixx-Iyy)/Izz)*x(10)*x(11)) + (1/Izz)*u(3); % Acceleration about Z
direction

x_dot  = [x(7); x(8); x(9); phi_dot; theta_dot; psi_dot; u_dot; v_dot; w_dot; p_dot;
q_dot; r_dot];
end
```

3.3.3 SIMULINK Model

In Figure 30 we have our overall SIMULINK model that simulates quadrotor dynamics using the Newton-Euler method.

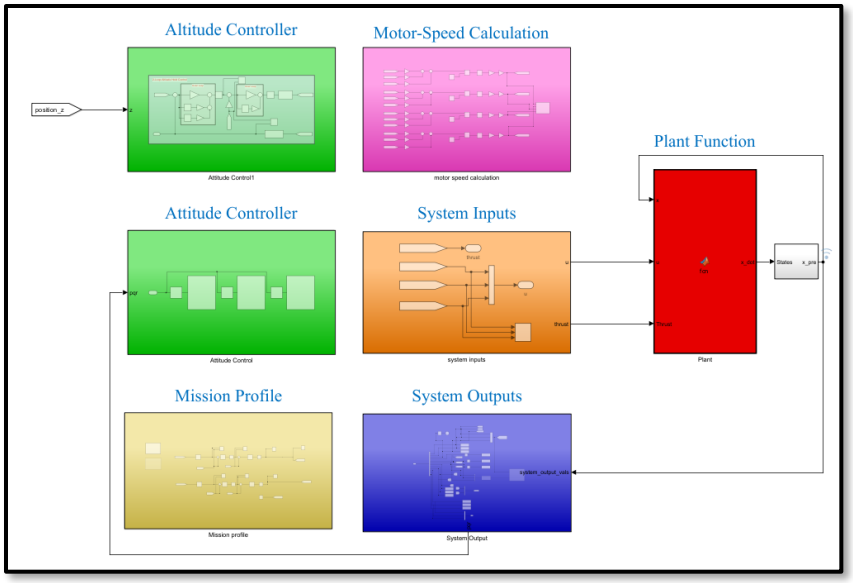


Figure 30 Complete SIMULINK Model

3.4 Cascade-Loop PID Controller for RC control

This section details the design of the attitude controller developed for RC control of the drone using Pixhawk.

3.4.1 PSP Toolbox

The PSP Toolbox contains various the following sensor modules to simplify connection of designed controller to the software and hardware components:

Figure 31 RC input block in PSP Toolbox

3.4.1.1 RC input

The RC or the Radio Control input block allows users to access the PWM input signals being sent by their RC transmitter. These signals are recorded by the RC receiver and sent to the Pixhawk. The sample time and RC channels which are to be used can be selected. The block provides access to a maximum of 18 channels.

3.4.1.2 Sensors Combined

This module allows access to sensor data coming in from the Pixhawk. Data from the following sensors is included: magnetometer, barometer, accelerometer, gyroscope. Users can change the sample rate and choose which sensor data they want to use from settings and parameters.

3.4.1.3 Vehicle attitude

The attitude blocks provide an estimate for the vehicle attitude, which involves the roll, pitch and yaw rates of the vehicle. The 4^{th} option is for a quaternion, which gives the orientation of the vehicle as a quaternion. A different function is required to convert these values to the Euler angles, to get the values of the phi, theta, psi angles.

3.4.1.4 Vehicle GPS

The GPS module gives the values coming into the Pixhawk from the GPS module, which gives the attitude of the multi copter along with its latitude and longitude values.

3.4.1.5 RGB LED

This block can be used to configure how the RGB LED installed on the Pixhawk would blink/flash/display for different scenarios, especially for arming/disarming.

3.4.1.6 PWM output

The PWM Output controls the output PWM values sent to the motors on the multirotor. The Arming output for the motors can be controlled, which is important for controlling the factor which decides suitability for arming/disarming. The number of PWM channels and their update rate (frequency) can also be configured.

Figure 32 PWM Output block in PSP Toolbox

3.4.1.7 uORB modules

uORB (micro–Object Request Broker) is a middleware used in the PX4 flight stack. It is utilized as an asynchronous API which publishes and subscribes messages within the PX4 architecture, thus allowing for inter-thread and inter-process communication. It enables various components to communicate by publishing messages to specified topics, preconfigured in the PSP Toolbox t (e.g., sensor_accel, vehicle_global_position). It further enables subscribing to these topics to receive required data through messages. The UAV Toolbox Support Package in Simulink includes blocks tailored specifically to sending and receiving uORB messages.

When code is generated from a Simulink model containing these blocks and

deployed to a Pixhawk flight controller, the Simulink controller installed on the autopilot becomes able to both read from and write to the uORB network, interacting with corresponding topics. The uORB protocol allows accessing of internal PX4 parameters for real-time tuning of controller gains. All the modules and blocks mentioned above are implemented in the Simulink model through reading and writing of uORB messages. This enables Simulink to access all intermediate variables and variables used in the PX4 autopilot. [17]

3.4.2 RC inputs

The input values being transmitted by the RC transmitter are divided into 5 channels for our use. Channel 5 is used as the arming/disarming channel. The three-way switch, SWA on the transmitter was configured as the arming/disarming switch to correspond to channel 5. Moreover, Channel 1 is the roll channel; similarly, channels 2,3 and 4 are the pitch, throttle, and yaw channels respectively. These channels take values from the 2 toggles and convert them to a PWM value which is then transmitted to the RC receiver. These PWM values are limited to between a minimum of 1100 and a maximum of 1900. Furthermore, 1500 is the 'rest' value for the roll, pitch and yaw channels, and '1100' is that of the throttle channel. A deadzone rate of 0.05 is used for the toggles to ensure that there is no stick drift and to give a "cushion" region. The MATLAB code for doing this is shown below:

```
RCMin = 1100;
RCMax = 1900;
deadZoneRate = 0.05;
deadZone = deadZoneRate*(RCMax-RCMin);
 %ensuring values do not exceed 1900 and don't go lower than 1100
if (u < RCMin)
   u = RCMin;
elseif (u > RCMax)
   u = RCMax;
```

```
end
%configuring the deadzone
 if (u > 1500+deadZone)
   y = u;
 elseif(u < 1500-deadZone)
    y = u;
 else
    y = 1500;
 end

 end
```

Since these values are for angles and throttle, they have to further be normalized. They are normalized between 0.5 and -0.5 for the angles and between 0 and 1 for the throttle. A 0.5 radian limit is set for the angles to ensure that sensitivity doesn't go too high.
The roll, pitch and yaw values from here are fed into the attitude controller, whereas the throttle value goes directly into the motor mixer.

3.4.3 Attitude Control

The attitude controller has a total of 8 inputs:

- The Angular rate values, p, q and r, are being feedback from the plant model.
- The normalized roll, pitch, and yaw values from the RC transmitter
- The roll and pitch feedback values from the plant model (the yaw rate does not require a cascade-loop controller; thus, no yaw feedback is involved)

3.4.4 PID

PID controllers are commonly used in quadcopters, since they provide stability, precision, and robust performance. They work on a feedback system by comparing the setpoint, which is the required value, with the actual value from the system,

which may have been obtained using sensors or otherwise. An error value is then calculated, and the PID then acts on this error value to reduce it to near zero (ideally zero). The transfer function of a PID system can be written as following,

$G(s) = K_p + K_i\left(\frac{1}{s}\right) + K_d s$ [72]

where K_p is the proportional gain, K_d is the derivative gain, K_i and is the integral gain.

These three gains have the following effect on the system, working best when used in conjunction:

1. **Proportional (P):** This part of the controller multiplies the error signal by a constant factor, K_p .This helps improve the transient response rise time and settling time, thus making the controller more reactive to changes, however making the system susceptible to oscillations and overshoots in the case of increasing it too much.

2. **Integral (I):** The integral part sums up all the past errors and multiplies them by a constant, K_i. This helps to eliminate any steady-state error over time and ensures the system reaches the exact desired value. However, increasing this value too far causes the system to become sluggish.

3. **Derivative (D):** The derivative component computes the rate of change of the error signal and multiplies it by a constant K_d. It anticipates future errors and adjusts the control input to smooth out the response and prevent overshooting or oscillations. It does this by limiting the reaction speed of the controller.

By combining these three elements, the PID controller can effectively control the system, providing a balance of quick response, accuracy, and smooth operation.

3.4.5 Cascade loop PID control

The designed attitude controller utilizes 3 PID controllers, 2 of which are cascade-loop PID controllers. This is because the quadcopter is an under-actuated system [73], which means that it has 6 DOF but it is being controlled using only four motors. [74] X and Y can be controlled directly using the pitch and roll respectively. Due to this fact, when establishing control for a quadcopter, the 3 position

coordinates are considered along with the yaw angle. Three orientation controllers have to be used for each roll, pitch and yaw. The control signals from these three position controllers make up a force vector (thrust) in the inertial coordinate system. A cascade loop PI-PID controller is used for Roll and Pitch values. A cascade loop controller as shown in Figure 33, works through 2 PID loops, the inner loop and the outer loop. The outer loop is used to control the attitude of the quadcopter. The inner loop controller takes the output of the outer loop controller as its setpoint input, and it is used to control a a swiftly varying variable, which in this case is angular velocity. [75]

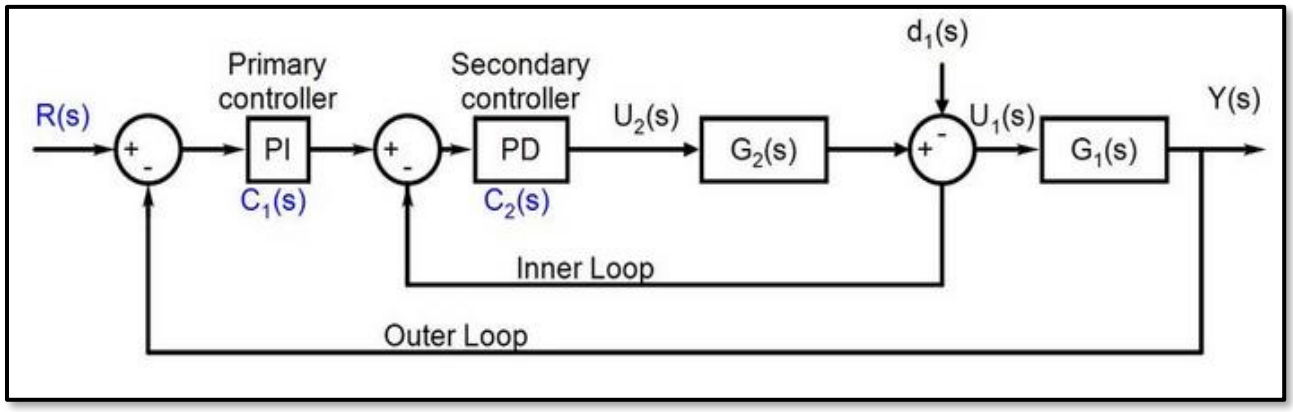


Figure 33 Architecture of a cascade-loop PID controller [80]

Thus, the outer loop controllers are designed by taking the normalized values from the RC transmitter as setpoints. The feedback values used for calculating the error are the Euler angle values taken from the plant model. The outputs from the outer loop controllers are used as the setpoints for the inner loop controllers, which use the values of the rates from the plant model as feedback to calculate the error values. This is done for control of each of the roll and pitch values. The p and q values are multiplied by a gain value of 1/3 to normalize the values coming in from the plant. A simple PID controller is used for Yaw, which just takes the normalized values from the RC transmitter as its setpoint and the Yaw angle value from the plant as its feedback, thus calculating the error and applying feedback control on it.

All of this is shown as a Simulink model.

Furthermore, for tuning of the PID parameters, trial-and-error was used. A step input was provided and then each parameter tuned one by one for each controller

until the output curves best resembles the step input curve.

The output values from these controllers can then be used to calculate the PWM values for each motor, in the Motor Mixer.

3.4.6 Motor Mixer

The output from the attitude controller, in addition to the thrust, represents the angular velocities of the quadcopter. The following equations are utilized to find the values of the PWM to be supplied to the 4 motors: [69]

$$P_1 = P_z + P_\alpha + P_\theta - P_\varphi \quad (14)$$

$$P_2 = P_z - P_\alpha + P_\theta + P_\varphi \quad (15)$$

$$P_3 = P_z - P_\alpha - P_\theta - P_\varphi \quad (16)$$

$$P_4 = P_z + P_\alpha - P_\theta + P_\varphi \quad (17)$$

where P_1, P_2, P_3 and P_4 represent the PWM values being output to each motor in the quadcopter. $P_\alpha, P_\theta, P_\varphi$ represent the values output by the roll, pitch, and yaw controllers, respectively. [69]

These PWM values are then normalized based on the idle PWM value of the brushless DC motor, and then output to the mathematical model.

3.5 Proposed Detection and Tracking Framework

3.5.1 Overview

This chapter outlines the detailed methodology of implementation for the proposed solution. The choice of dataset, the required preprocessing, and the algorithms used have been explained. Furthermore, the evaluation metrics used for quantitative assessment of detection and tracking have been discussed. Based on the comparison of two trackers analyzed, the further steps taken for implementing tracking on embedded hardware have also been described.

3.5.2 Analysis of Project Requirements

The following requirements were considered for designing the solution to the

proposed problem:

Video acquisition is performed by an operator manually flying the drone over a target point.

The video is fed into the detection and tracking algorithm, which outputs the detected boxes around the objects of interest (i.e., cars, etc.).

The system is capable of offline video processing, and analysis of its performance.

The system can perform real-time detection and tracking using the training model on Yolov8 for detection and DeepSORT for tracking on a windows computer, which is further implemented on TensorFlow Platform to interface with a Raspberry Pi.

3.5.3 Hardware Setup

To fulfill the above requirements, the hardware components employed are shown in Figure 5. The hardware set-up for real-time object detection and tracking comprises an ESP-870u camera equipped with an Intel Visual Processing Unit (VPU) for data acquisition and processing of the algorithms, a Raspberry Pi acting as its host device, and a battery for the Raspberry Pi, all mountable on a drone to form an integrated system. For off-line video processing, the acquired data from the UAV is fed into a ground-based GPU-enabled computer.

Data gathered from the UAV is first fed into the YOLOv8 detection algorithm, which is trained on the self-annotated OIDv7[63] dataset for detecting cars out of all traffic objects. Then, the tracker makes use of the trained weights to track these cars across the frames of the video. The details of the algorithm are given in section 3.5. The tracking results are saved for evaluation or output to the screen. For real-time processing, the PyTorch weights file is converted to an intermediate representation comprising an XML file and a .tflite file. Then, this model file is fed into the Raspberry Pi CPU for performing real-time detection and tracking.

3.5.4 Choice of Dataset and Annotations

Out of the numerous datasets available, one that closely matched the scenario being

worked on, was the OIDv7 and OIDv4[63] dataset. This dataset was collected by the Opensource Google team, and made available online, which includes various computer vision tasks on aerial imagery including object detection and tracking. It comprises 288 video clips, and over 10,000 static images of different classes and covers diverse urban and rural environments. The annotations were done by using "labeling"[76] annotator available in python w=environment along with online annotator made available by the Open Image Dataset team. [63]

3.5.5 Choice of Algorithms

3.5.5.1 Detection

Out of the two classes of deep-learning based object detectors, one-stage detectors are much better suited for embedded vision applications because of their speed. YOLO is a popular one-stage detection algorithm and has many variants. Their performance can vary based on the type of hardware they are running on. Out of all the algorithms in the YOLO family, small versions YOLOv7[77] and YOLOv8[78] have been shown to perform better on less advanced hardware like a CPU[78]. Therefore, YOLOv8 was selected and trained on the converted data, as described in the previous section. The training was done on Google's cloud GPU platform Collab. The dataset, which contained over 6000 images along with their annotations, was sliced and around 1000 images were utilized and annotated. Out of these, 700 images were used for training and 300 for validation. This was done to reduce the time associated with uploading and downloading of data and training time of the algorithm on the cloud platform.

The backbone of the YOLO algorithm is a convolutional neural network, which outputs a 3D feature map when an image is input into it. The output feature map is projected onto the input image, which is divided into a square grid of cells. The depth of this feature map is an array containing the parameters of bounding box coordinates, abjectness score, and class

probabilities for the grid cell that sits on its surface. Each grid cell can predict multiple bounding boxes. The bounding box with the highest probability is identified, and any overlapping boxes with an Intersection over Union (IOU) value of greater than 0.5 are deleted. This technique is called non-maximum suppression and is used to clean up the bounding box predictions and localize the object of interest.

The architecture of the YOLO algorithm comprises a convolutional neural network with 8 convolutional layers and 3 fully connected layers. It was trained with an input image size of 640x640 pixels. The modified YAML file with the paths to the training data and information on the class labels was given to the algorithm for the training to begin.

3.5.5.2 Tracking

Out of the three categories of trackers described, the particle tracking-based methods are computationally most efficient, and therefore the trackers of choice for real-time or embedded vision applications. The performance of the two trackers was compared, paired with the detection model- DeepSORT[79] and its improved, newer variant StrongSort[80].

The first step in tracking is to generate the detections. The tracker then extracts deep features from the detected objects which store high level information about them and are used for matching and association in the subsequent frames. Detected objects are assigned unique IDs, and tracks are initialized for every new ID. DeepSORT performs data association between subsequent frames using the Kalman Filter, and the state of each track is predicted based on past measurements and time elapsed since the last update. The features of the new detections are then compared to the tracks using distance metrics like squared distance or Cosine distance to ascertain if the objects belong to an existing track or a new track needs to be initialized. If the state of the track is not updated for a certain number of frames, it is considered deleted. Finally, the tracks are smoothed out using techniques like track interpolation and are output in the form of bounding

boxes with assigned IDs.

StrongSort is an improved version of DeepSORT, which uses a more powerful feature extractor and linking method and is therefore much more capable of distinguishing between different objects.

3.5.6 Choice of Evaluation Metrics

3.5.6.1 Evaluation Metrics for Detection

Qualitative results help to visualize the performance of the algorithms, however, to evaluate them more precisely, a quantitative measure of their performance is needed. An object detection algorithm works by predicting bounding boxes for objects in images it has not seen before. To evaluate the algorithm, we must also have the ground truth data for these images containing the true coordinates for the bounding boxes. The predicted bounding box is compared with the true bounding box, and the extent of overlap is expressed as Intersection over Union (IOU). An IOU value closer to one represents greater overlap.

Precision: A threshold value for the overlap can be set for the detection algorithm above which it must count the predicted detection as a true positive (TP). An incorrectly detected object is known as a false positive (FP). The ratio of TPs to the sum of TPs and FPs is known as precision. A higher precision means there are very few FPs (incorrect detections).

Recall: An object that the algorithm must detect but fails to do so is known as False Negative (FN). Recall is the ratio of true Positives to the sum of true positives and false negatives.

Because of the inherent nature of how detection algorithms work, there usually has to be a tradeoff between precision and recall. While detecting objects, the algorithm assigns a confidence score to each prediction for it being a TP. During evaluation, if this confidence score is greater than a certain IOU threshold, the detection is counted as a TP, otherwise it is classified as a false detection (FP). So, if the threshold is set higher, only

the predictions with very high confidence scores will be considered TPs, however, some true predictions with lower confidence scores may be missed (higher FNs), and there will be a very less chance of incorrect detections (FPs). Therefore, a higher threshold for evaluation is likely to increase precision and decrease recall.

Conversely, if the threshold is set lower, there is a very less chance of missed detections, but a greater chance of FPs creeping in too. So, a lower threshold will improve recall but decrease precision.

Average precision (AP): This is a derived metric which is computed by averaging precision at multiple recall values. It is represented by the area under the precision-recall curve. Average precision calculated and summed up for multiple classes is known as the mean AP, and it is usually reported at a range of IOU values for detection models.

F1 score: This is the harmonic mean of precision and recall. This is a single metric to evaluate a model when both precision and recall must be considered.

3.5.6.2 Evaluation Metrics for Tracking

Since detections in individual frames form the basis of tracking across the entire video, the metrics used to evaluate detection are also a measure tracking. However, they cannot give complete information about the performance of the tracker. There are several metrics which have been proposed for more holistic evaluation of object tracking in videos[81], [82], [83]. Two of these benchmark metrics are Multi-Object Tracking Accuracy (MOTA), and Multi-Object Tracking Precision (MOTP).

MOTA[84]**:** MOTA is defined as 1 minus the total error ratio. The error ratio is the sum of FPs, FN, and ID switches (IDSW), all divided by the total ground truth annotations (GT). To calculate MOTA, first spatial correspondence is established between predicted outputs and ground truth bounding boxes. Then, based on the set IOU threshold, FPs and FNs are calculated. Finally, value for MOTA is calculated using the equation given

below. MOTA can range from minus infinity to 1, with 1 indicating 100 percent accuracy.

$$MOTA = 1 - \frac{FP + FN + IDSW}{GT} \quad (18)$$

MOTP[84]: MOTP is a measure of how well the objects in the frames are localized. It is the ratio of distances between the prediction and the ground truth boxes to the total matches found between the ground truth and predicted boxes. The formula for MOTP is given in the equation below, where D represents the distance between the ground truth objects and the detection results, and M represents the total matches between the ground truth and the detection output.

$$MOTP = \frac{D}{M} \quad (19)$$

Inference Time: This is the measure of computational efficiency of the algorithm. It is the time taken during the forward pass through the network. the FPS is calculated by "1/inference time". Depending on the hardware being used to run the algorithms, inference speed can vary, and to overcome slower inference, other things about the model may need to be changed. For that reason, the model running on the Windows PC (with an Nvidia GTX1650 GPU) must be modified before it can run on any edge computing device like Raspberry Pi[85].

3.5.7 Deployment on OpenCV AI Kit and Raspberry Pi

3.5.7.1 Retraining the detection model

Usually, the performance of the AI models is heavily dependent on the hardware they are running on. Accuracy can be optimized when the speed of the operation is not a crucial factor, and the computational capacity of the hardware allows for it. However, when the same models are ported onto single board computers (SBCs) or embedded devices, the architecture of the models weighs them down. Therefore, modifications must be made to reduce their size, which improves their speed but also leads to some drop in

the accuracy. However, this is an inevitable compromise to reproduce a balanced performance on the embedded computers.[85]

For an optimal performance on the Raspberry Pi 4B, which was the hardware chosen for real-time implementation of the project, hence retraining a lighter object detection model. It was decided to train the Nano version of the YOLOv8[77] and to transfer the model's ".pt" file (weights) in TensorFlow-lite format as well for faster on-board processing, based on the comparison of performance provided in the official documentation for different models.

Table 3 Comparison of Etron, Kinect & Xtion Pro Live [9], [10], [11]

Model	**Image Size**	**FPS**	**Latency [milliseconds]**
YoloV7t	416x416	46.7	37.6
YoloV7t	640x640	17.8	97.0
YoloV8n	416x416	31.3	56.9
YoloV8n	640x640	14.3	123.6
YoloV8s	416x416	15.2	111.9
YoloV8m	416x416	6.0	273.8

4. RESULTS

4.1 Mathematical Model

4.1.1 Positional Variables:

We can see in the results in Figure 34 how our position changes in space with respect to time and how the angles in our body frame are changing as quadrotor goes through the course.

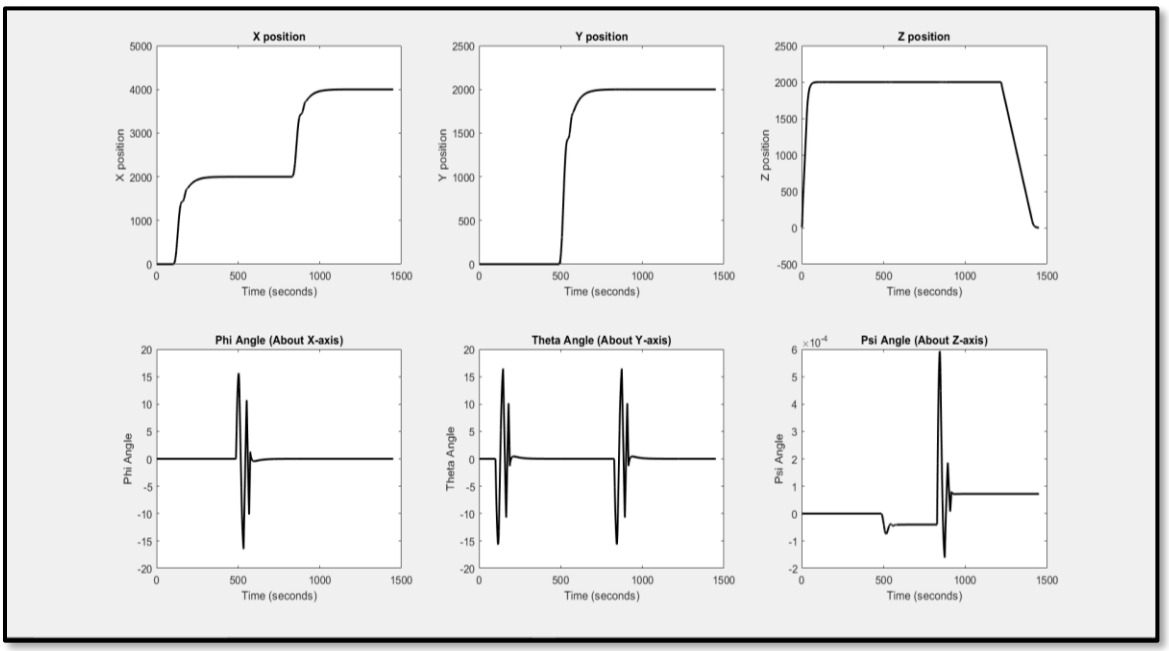


Figure 34 Positional Variables

4.1.2 Motion Variables:

In Figure 35 we have the results of all rates of change both translational and rotational, it includes velocities in the x, y, and z-axis and roll, pitch, and yaw rate.

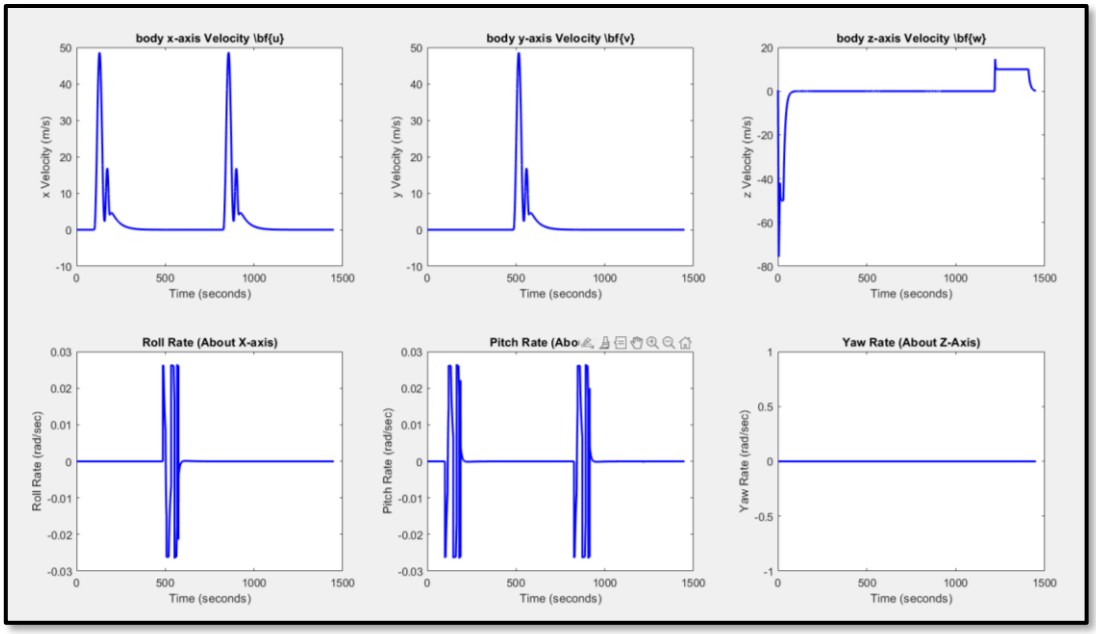


Figure 35 Motion Variables

4.1.3 3-Dimensional Trajectory in MATLAB:

Figure 36 shows us the track of the complete flight followed by our quadrotor.

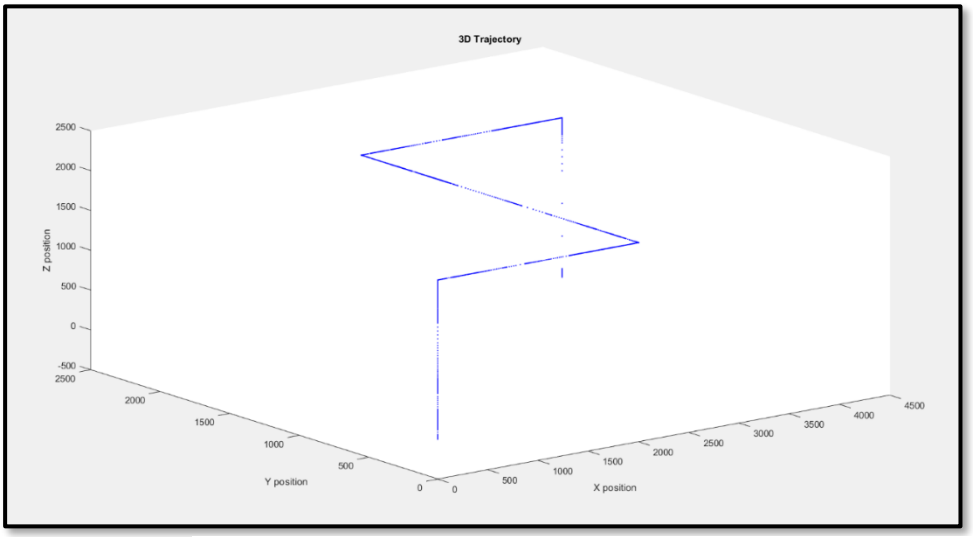


Figure 36 3-D Trajectory of the Flight

4.2 Software-in-the-loop (SIL)

"Software-in-the-loop" (SiL or SIL) is a testing method used in the development and validation of complex software systems, especially in fields like automotive, aerospace, and industrial automation. This method allows for testing software components in a controlled and virtual environment, providing a crucial step in verifying and validating the software's functionality and performance before deploying it to the hardware. SIL helps identify faults or errors in the software system, so they can be fixed before being deployed, preventing costly issues in real-world scenarios. Additionally, SIL allows testing the system's responses under different conditions and environments without risking the hardware. It is also useful for integration testing, where different software modules and components are tested together to ensure they interact correctly, helping resolve interface issues.

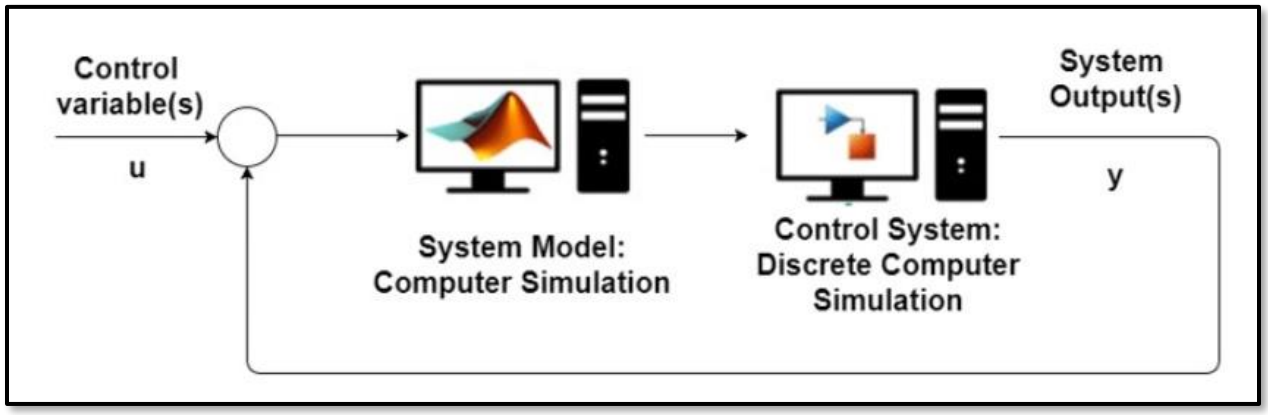


Figure 37 Software-in-the-loop Example [91]

We use the SIL approach for our flight controller designed on MATLAB/Simulink before uploading it onto the Pixhawk hardware. For this approach, we utilize Gazebo 11 and FlightGear-F450 software. Gazebo is an open-source robotics simulation software that offers a strong and flexible environment for developing and testing robots virtually. In contrast, FlightGear is an open-source flight simulator that offers a complete and highly realistic flight simulation experience.

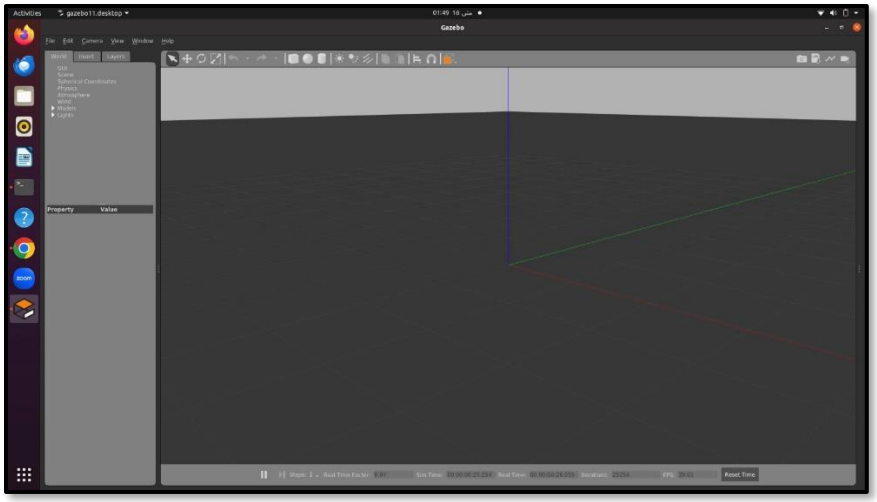


Figure 38 Gazebo default Environment

4.2.1 ROS/Gazebo

ROS, short for Robot Operating System, is a versatile framework for creating robot software. It consists of tools, libraries, and conventions aimed at simplifying the development of complex and reliable robot behavior across a wide range of robotic platforms. Additionally, ROS acts as a middleware, facilitating communication between different parts of a robotic system. It enables seamless data exchange between components such as sensors, actuators, and control algorithms. ROS also encourages the development of modular software components called nodes, which can be reused in various projects. Communication between nodes in ROS follows a publish-subscribe messaging model. Topics facilitate one-way, many-to-many communication, while services offer a synchronous, two-way communication mechanism. ROS provides a set of tools for tasks such as visualization (Rviz), simulation (Gazebo), data recording and playback (rosbag), and system introspection (rqt). These tools aid in the development, testing, and debugging of robotic applications. ROS offers hardware abstraction, allowing developers to write high-level code without needing to worry about the specific hardware details. This is achieved through standard interfaces and drivers for various sensors and actuators. Finally, ROS integrates smoothly with simulation environments like Gazebo, enabling developers to validate their algorithms in realistic virtual scenarios before deploying them onto physical robots.

The ROS-Gazebo connection merges the Robot Operating System (ROS) with the Gazebo simulation environment, forming a potent toolset for robotic simulation and development. ROS nodes can govern simulated robots in Gazebo via topics, services, and actions,

facilitating smooth communication between the simulation and the ROS ecosystem. This integration boosts the development process's effectiveness, enabling quick prototyping, testing, and iteration in a safe and economical manner. It's worth noting that this integration is exclusive to Linux software.

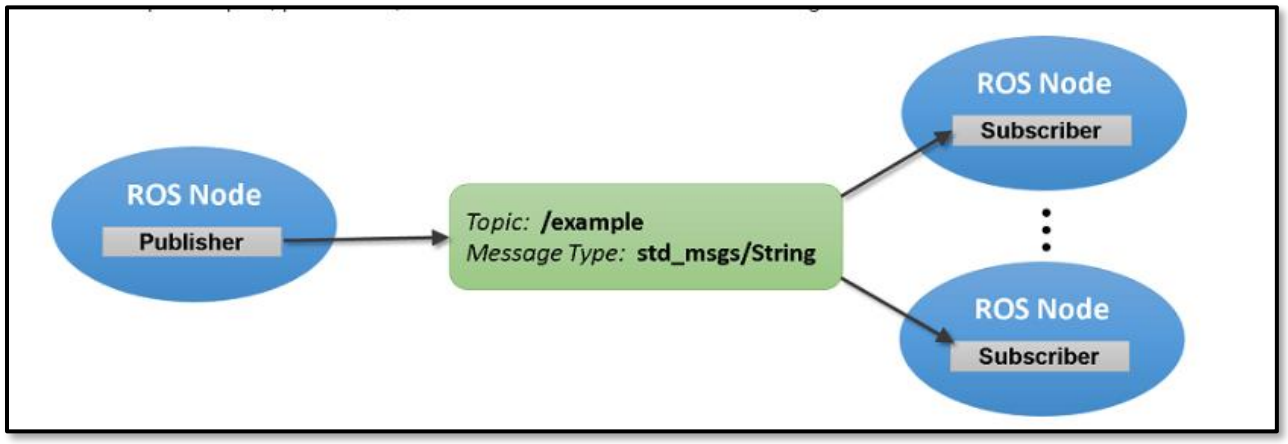


Figure 39 ROS Communication between nodes [92]

To simulate this project, the first task was to create the environment. The basic xacro format was used to design the xacro file of a quadcopter in Gazebo. To test the drone, Gazebo plugins for Ardupilot Master were used to automate and make it autonomous, allowing it to hover at a fixed level. Then, MAVProxy was installed as a ground control station (GCS) for UAVs. MAVProxy is a robust command-line ground station software designed for developers, which can be enhanced with additional modules or used in conjunction with other ground station software like Mission Planner, APM Planner 2, or QGroundControl to offer a graphical user interface. It boasts several important features, such as the capability to forward messages from your UAV over the network via UDP to multiple ground station software on different devices. Next, the ground station QGroundControl was installed, as it supports the full setup and configuration of Ardupilot and PX4 Pro powered vehicles. QGroundControl offers comprehensive flight control and vehicle setup for vehicles powered by PX4 or ArduPilot systems[86]. It provides the information related to the drone as it can be seen in Figure 6.

Then the terminal was used on Ubuntu 20 to launch Ardupilot, QGC and Gazebo and different commands were used to arm the drone and hover it by giving the value of throttle. The drone designed was fitted with a camera just to give a realistic feel

of a drone that we designed in real life. The fitted camera was able to record the environment and show the video in real-time.

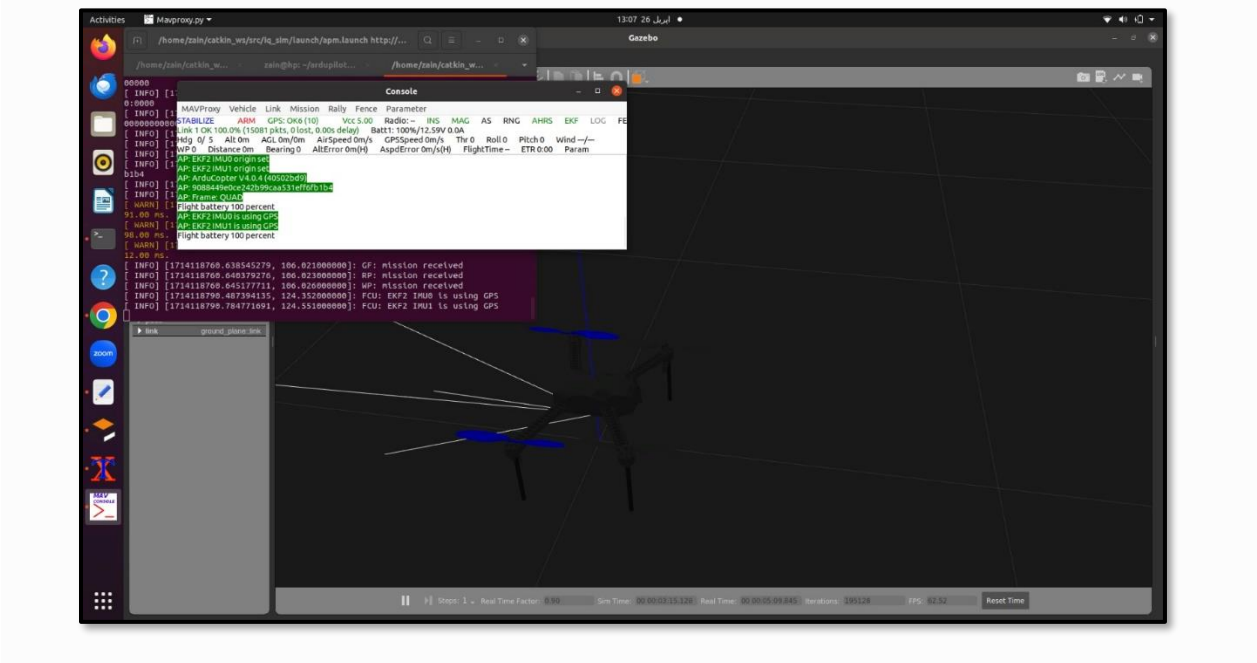
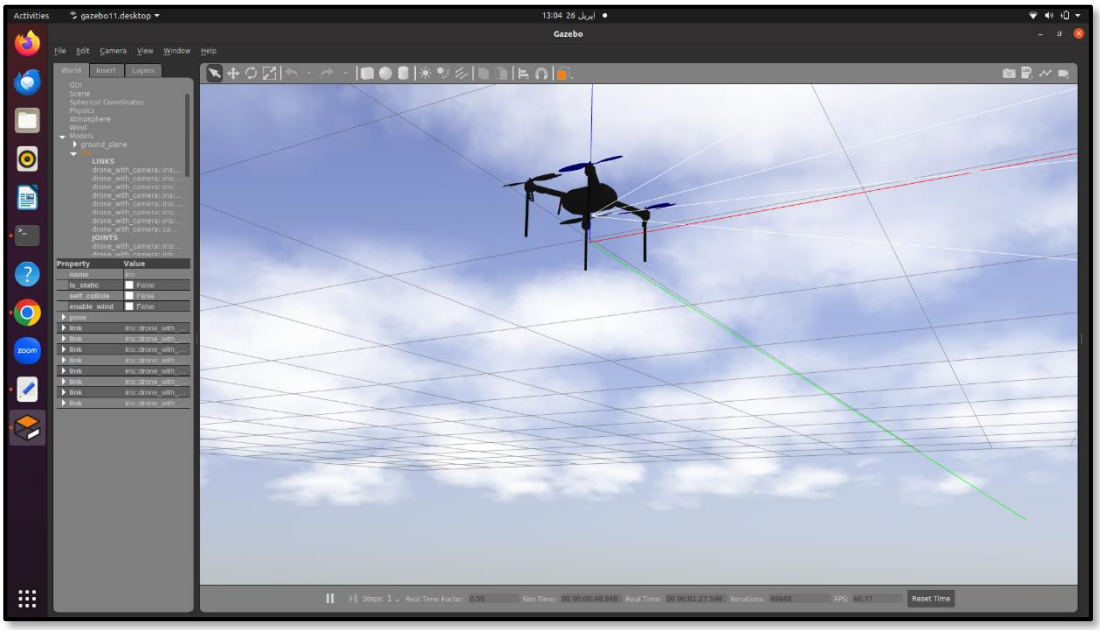


Figure 40 Ground control station tab



.

Figure 41 Designed quadcopter in gazebo environment

The setup of Ardupilot and QGroundControl was done to arm the drone, making

the rotors in the simulation ready to use and enabling it to hover. The main purpose of using Gazebo was to simulate and validate the flight controller designed in Simulink. To enable effective communication between Simulink and Gazebo, a toolbox from MathWorks was used. This toolbox provides various subscriber and publisher blocks, allowing any existing node to be a subscriber or publisher as needed and facilitating the sharing of topics carrying messages or information.[87]

Figure 42 Subscriber block from ROS toolbox in Simulink

The designed PID controller was connected to the links of the quadcopter model in Gazebo using the Subscriber/Publisher block, as shown in Figure 42. The PID controller's output, designed in Simulink, is transferred as rotor RPM to the drone. The drone then flies in Gazebo, travels along the given coordinates, and lands after completing the specified path.

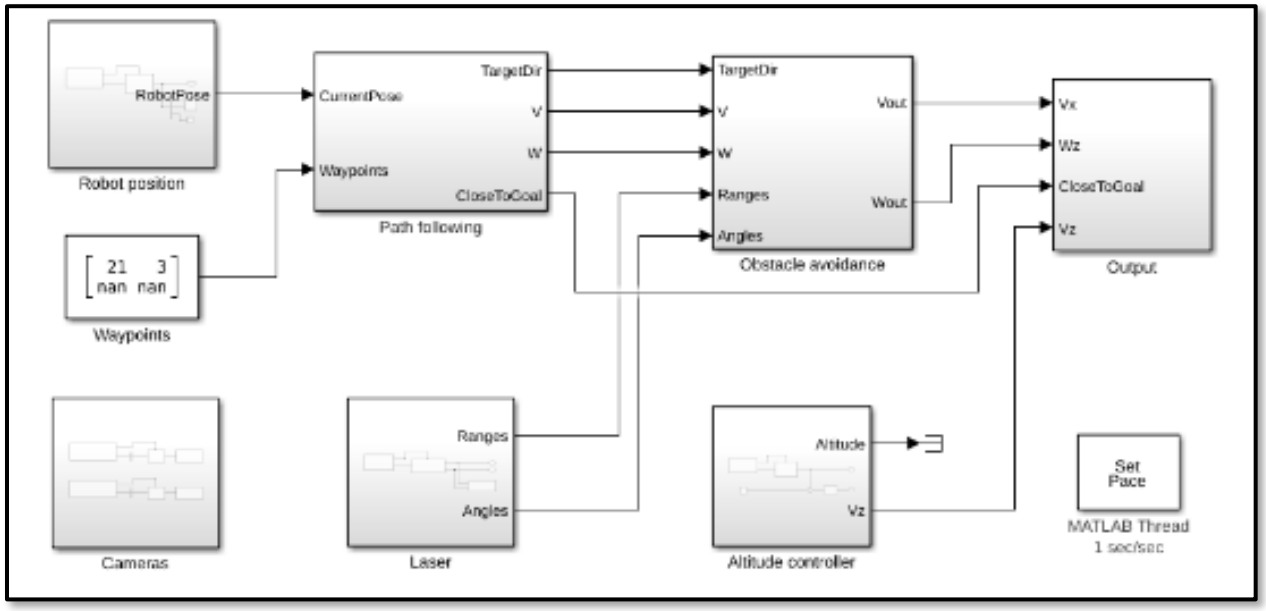


Figure 43 Flight Controller in Simulink

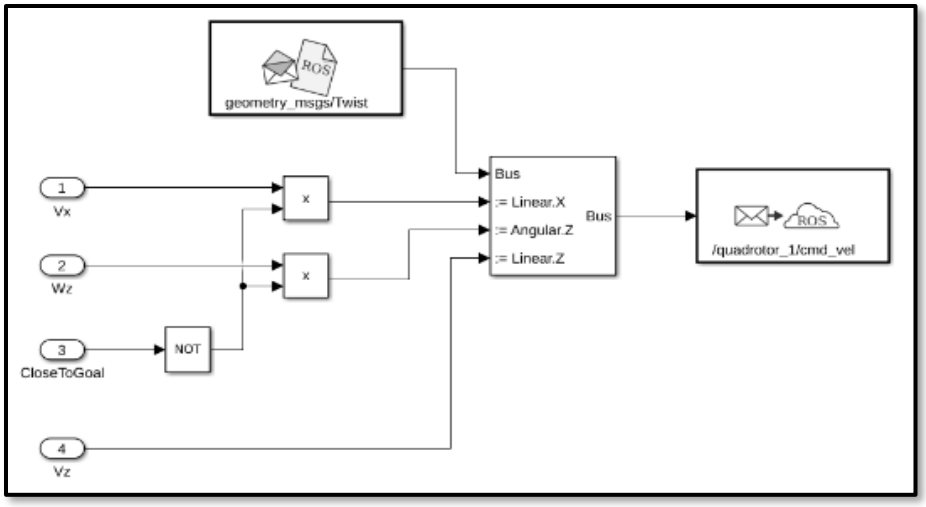


Figure 44 Output sub-block of flight controller

4.2.2 FlightGear F450

FlightGear is a widely used open-source flight simulator. It can seamlessly display the flight states of a simulated aerial vehicle in Simulink by receiving flight data from Simulink through a User Datagram Protocol (UDP) interface. UDP is a core protocol of the Internet Protocol (IP) suite, known for its simplicity and efficiency in transmitting data over a network. FlightGear is used by aviation enthusiasts, pilots, researchers, and educators for various purposes. It is effective for SIL purposes because it uses advanced flight dynamics models to accurately simulate the behavior of different types of aircraft. The simulator features a detailed and expansive global scenery database, including accurate terrain data, airports, landmarks, and weather conditions. Additionally, it offers advanced weather simulation capabilities, allowing users to experience real-time and historical weather conditions. The simulator also provides realistic cockpit instrumentation and avionics, replicating those found in real aircraft.

Figure 45 FlightGear simulator interface

For SIL purposes, a Simulink model is created with a variable knob at the start to adjust the values of roll, pitch, yaw, and throttle. These values are sent to the controller sub-block, as shown. The controller sub-block is then connected to the multicopter model, which is linked to the FlightGear interface sub-block, allowing the drone to be simulated in FlightGear.

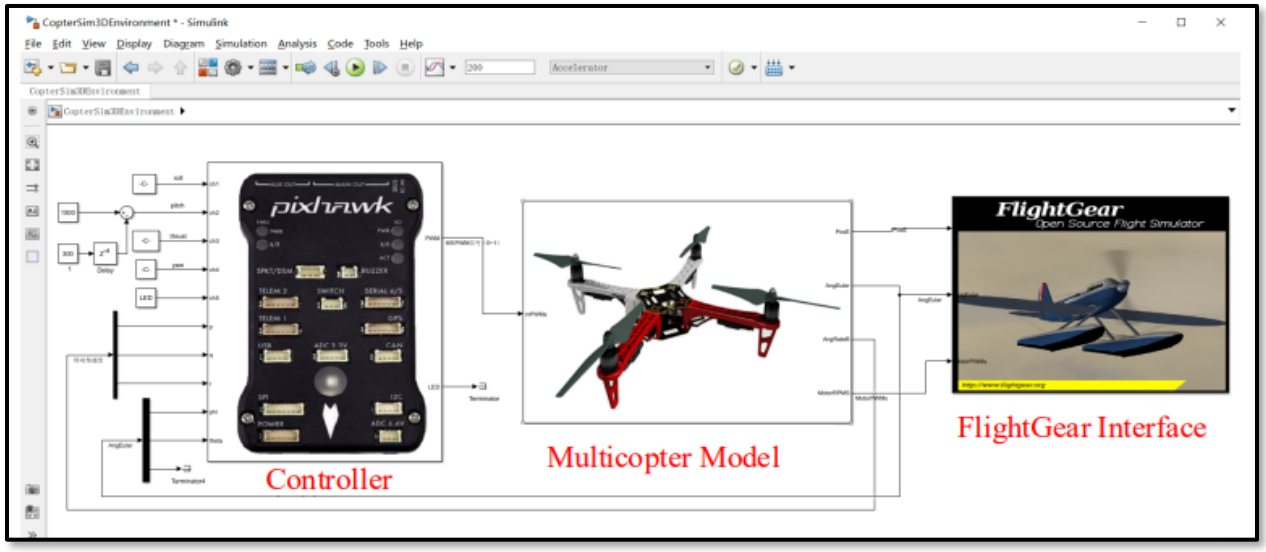


Figure 46 Simulink model for FlightGear[18]

The controller sub-block functions as an attitude controller for the quadcopter's pitch and roll angles. It processes control signals from the RC transmitter to adjust the quadcopter accordingly. The 'Input Conditioning' module translates the five-channel signals from the RC transmitter into the desired pitch and roll angles. The 'Attitude Controller' module then computes the required force and torque values to achieve the desired attitude. Finally, the 'Motor Mixer' module converts these force and torque values into control signals (ranging from 1000 to 2000) for the four motors. [17]

The 'Motor Model' in the multicopter model simulates the dynamics of the motors. The 'Force and Moments Model' module simulates all external forces and moments acting on the multicopter, including propeller thrust, fuselage aerodynamics, gravity, and ground support forces. The '6DOF' module calculates the kinematics of the vehicle, covering speed, position, and attitude. The 'Environmental Model' module provides environmental data, such as gravitational acceleration, air density, wind disturbances, and the geomagnetic field.

The FlightGear Interface subsystem features three input ports that correspond to the multicopter's position, Euler angles, and motor PWM signals. This subsystem transmits the multicopter's flight state information to FlightGear, enabling the observation of the quadcopter's flight attitude and trajectory in a 3D scene.[17]

4.3 SIL Results

The Software-in-the-loop objective was successfully achieved. Firstly, the Simulink flight controller, initially designed, successfully interfaced with the quadcopter model in Gazebo and managed to hover the drone at the specified coordinates. Secondly, the complex and more detailed flight controller was also simulated successfully in the FlightGear simulator.

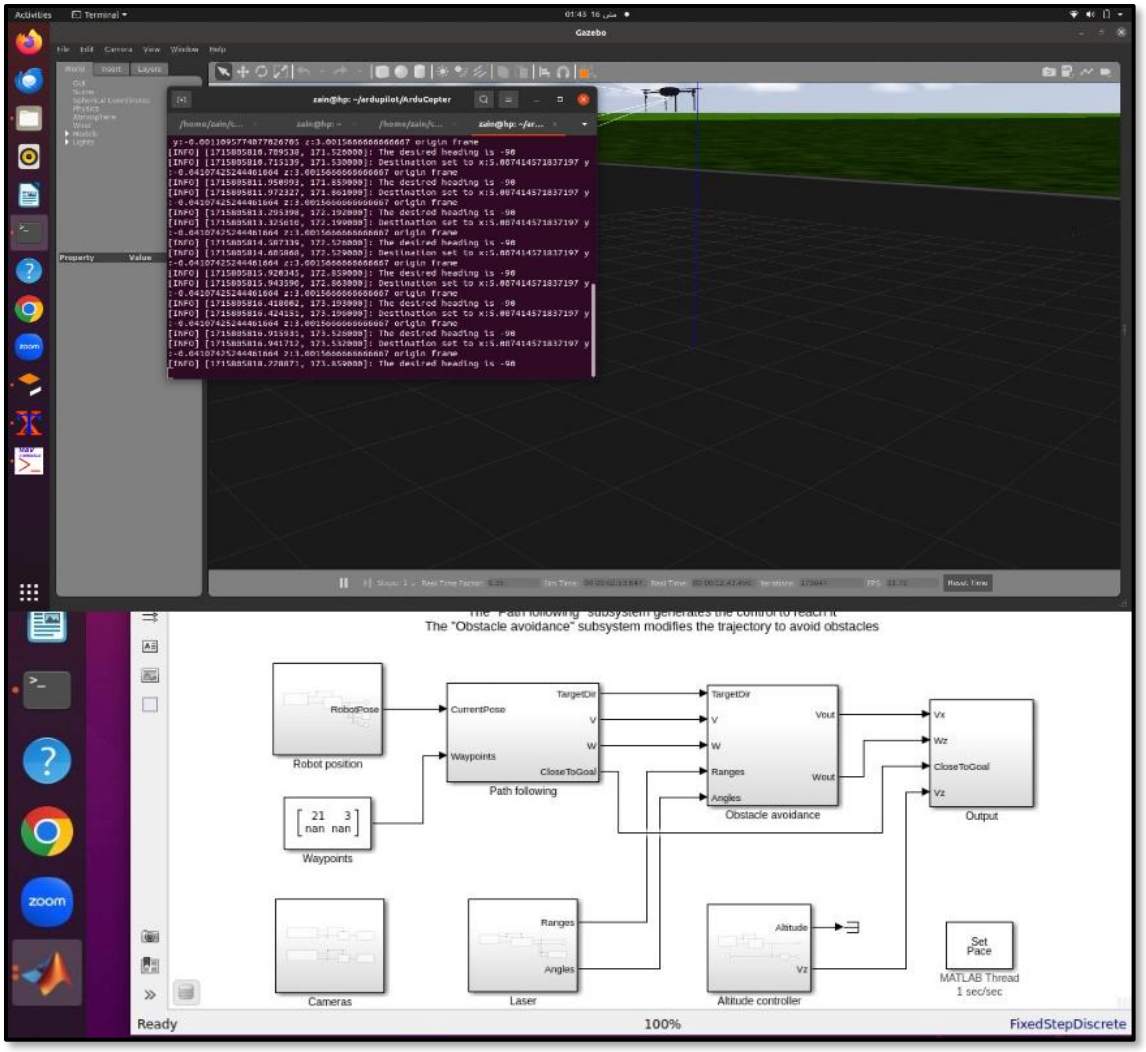


Figure 47 Quadcopter flight on Gazebo

Both Gazebo and FlightGear were utilized for SIL purposes, each serving different roles. Gazebo was utilized when a detailed or realistic environment or model for the quadcopter was not necessary, and only PWM and a simple PID controller were required. However, when a detailed flight controller was designed, intended for uploading onto actual PX4 hardware, it was first simulated on the FlightGear simulator. FlightGear provides specialized features tailored for aviation, which can be particularly advantageous for SIL testing of a quadcopter. These features include realistic flight dynamics, advanced weather simulation, detailed environmental interactions, and strong integration with aviation-specific tools and resources.

HIL uORB: The PSP toolbox in Simulink provides the uORB read/write block, which functions similarly to the Subscriber/Publisher block used in SIL on Gazebo. The main purpose of these uORB blocks is to read and write data to specific topics or nodes, facilitating communication between the simulation and the actual hardware.

The PX4 uORB Read block generates a Simulink nonvirtual bus corresponding to a specified uORB topic. During each simulation step, the block checks for new messages on this topic. If a new message is available, the block retrieves it, converts it to a Simulink bus signal, and outputs it through the Msg port. If no new message is available, it outputs the last received message. If no message has been received since the simulation started, it outputs a blank message. Conversely, the PX4 uORB Write block publishes messages to the uORB network. On each sample hit, it converts the Msg input from a Simulink bus signal into a uORB message and publishes it. The block publishes the message on every sample hit without distinguishing whether the input is a new message. [88]

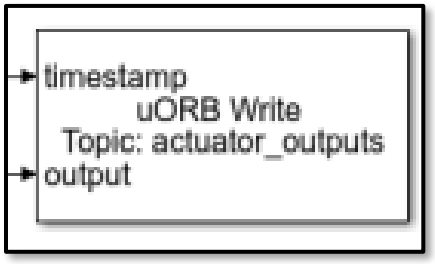


Figure 48 uORB Write Block

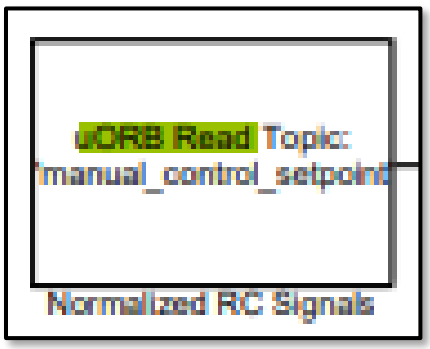


Figure 49 uORB Read Block

4.4 HIL Simulation

Hardware-in-the-loop (HIL) simulation is about integration of actual hardware components into a virtual simulation environment. This enabled testing of how the hardware would react to the designed control system in an environment similar to the real world. It allows tuning of the controller and ensuring its accuracy and correct working before deploying it on the quadcopter.

The hardware used in HIL simulation includes the Pixhawk autopilot, GPS sensor, and RC transmitter and receiver.

The following changes are made to the designed controller to enable it to run HIL simulations.

4.4.1 RC to controller in Simulink

The RC block from the PSP Toolbox is used in the Simulink model to read the RC channel values being received by the Pixhawk. 5 channels are connected to the controller, which read the roll, pitch, throttle, yaw values and the arming state. These values are then fed into the controller. The condition used for arming is an RC value >= 1500, which in the context of a three-way switch, means that the switch may be flicked to its middle or maximum position to initialize arming of the motors.

4.4.2 RGB mode

The RGB LED on the Pixhawk is utilized for information of the arming state of the Pixhawk.

The following conditions are used:

1. If the Pixhawk has successfully armed the motors (Arming state is 1), then the LED will be green and in breathing mode.

2. If the Pixhawk is not armed (Arming state is 0), then the LED will be red and in blinking mode.

4.4.3 Controller output PWM and uORB write

The plant model is no longer required, as simulation will now be carried out using hardware on a prepared environment. PWM outputs from the motor mixer are, therefore, now connected to the uORB write block, topic: actuator_outputs, which sends uORB message “actuator_outputs” to control motors instead of the PWM. This is done because for a HIL simulation, we need to send control signals to CopterSim instead of to actual motors.

Once these changes have been made, the controller is ready to deploy onto the Pixhawk, which can be done using the steps stated below.

4.4.4 Code generation

The Simulink coder enables generation of C code from a Simulink model. The hardware board needs to be selected as the Pixhawk PX4. The “Build” button, as shown in Figure 50, can be used to verify and build the model and generate a corresponding C code.

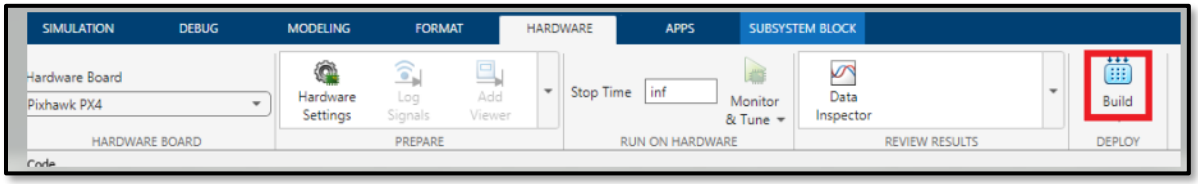


Figure 50 Build option in Simulink

4.4.5 Uploading to Pixhawk

Once the C code has been generated, the Pixhawk has to be connected to a COM port on the ground station. The “PX4Upload” command must be entered into the Command Window. Simulink itself detects the Pixhawk, and erases the previously installed firmware, installing the new firmware involving the controller designed.

4.4.6 QGC calibration

The Pixhawk must then be disconnected and reconnected after opening QGroundControl (QGC). GPS is then connected to the Pixhawk via I2C. QGC then requires calibration of GPS and the internal accelerometer, gyrometer and compass

of the Pixhawk. The RC receiver must then be connected to the Pixhawk, and a connection established with the RC transmitter. QGC further requires calibration of the RC system. Channel 5 for the RC is then set to Switch A and configured for Arming. Following that, the Pixhawk can be enabled for HIL simulations while still in QGC.

4.4.7 CopterSim and 3dDisplay

CopterSim transmits sensor data to the Pixhawk autopilot, which then computes the motor control signals and sends them back to CopterSim. This allows the Pixhawk autopilot to control both the simulated multicopter in CopterSim. At the same time, CopterSim shares the multicopter's attitude and position data over the local network using the UDP protocol. The 3DDisplay software then receives this data and creates a real-time 3D representation of the multicopter's flight, making it visible for monitoring and analysis purposes. [17]

The “Model Parameters” option, as shown in Figure 51, allows changing of various parameters of the aircraft, based on different parts available on the market and various physical parameters of the quadcopter, for example, mass.

Once the calibration and configuration in QGC is completed, the Pixhawk autopilot assembly is disconnected and then reconnected once CopterSim is launched. CopterSim automatically detects the Pixhawk hardware and confirms the status of the Pixhawk. 3dDisplay software is then launched simultaneously. Simulation is then started on CopterSim after confirming that the RC transmitter and receiver are connected. CopterSim transmits to and receives data from the Pixhawk in real-time, thus updating sensor values in real-time and sending motor signals and values to 3dDisplay. The RC transmitter is used to control the quadcopter, which can be seen flying in the 3d environment in 3dDisplay, as shown in Figure 52.

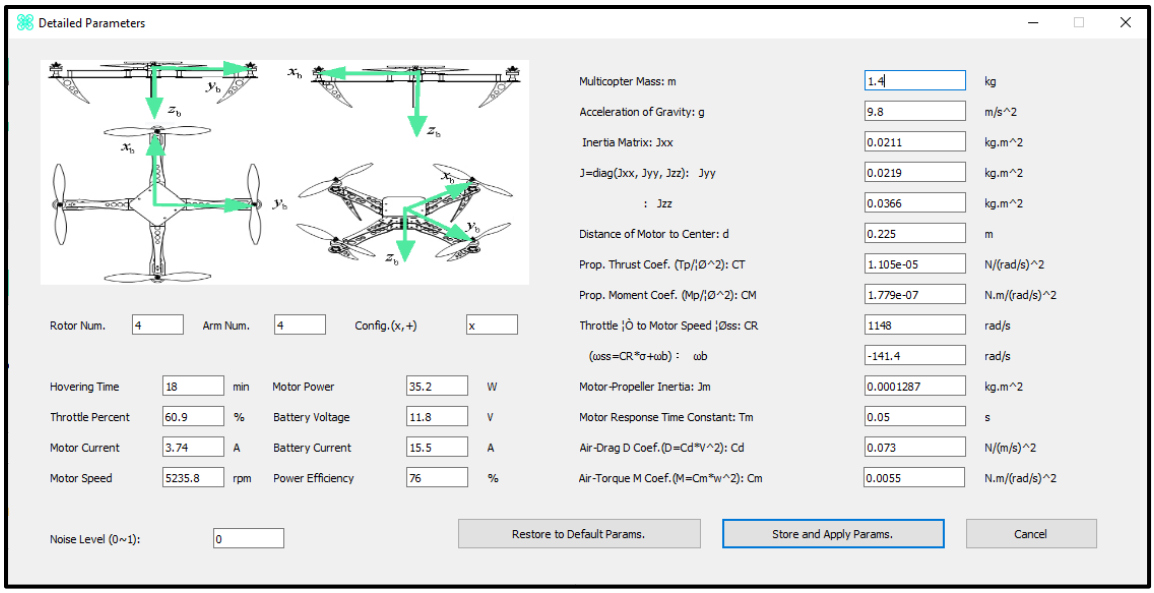


Figure 51 Model Parameters tab in CopterSim

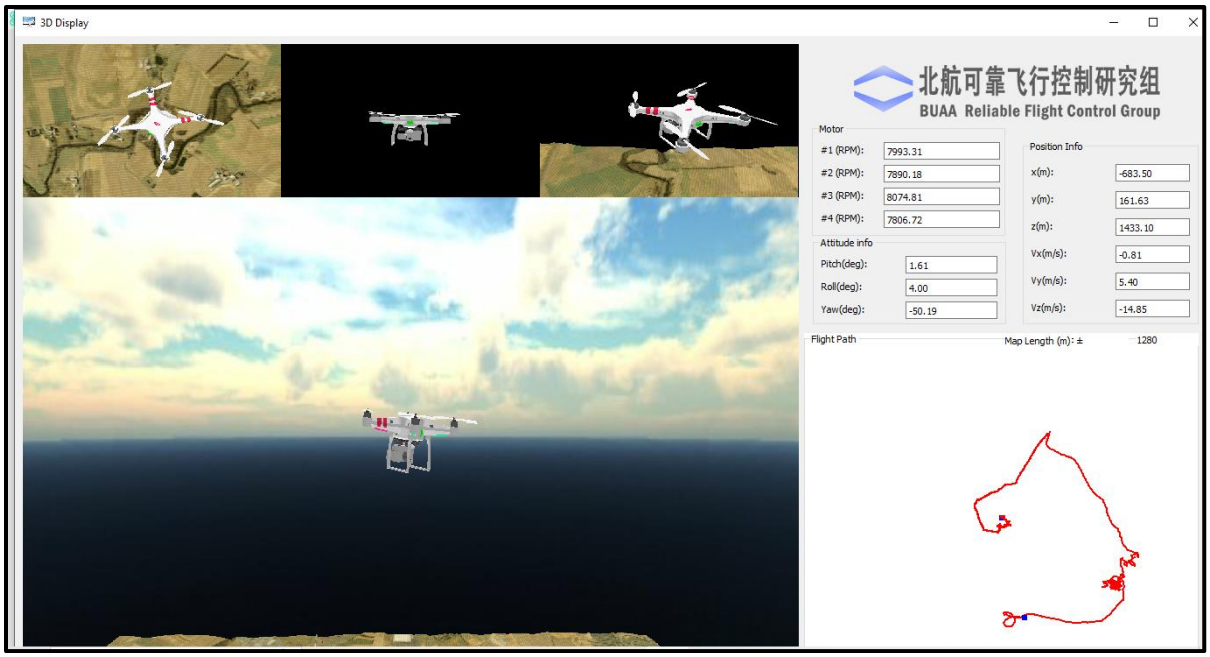


Figure 52 HIL simulation in 3D Display

As can be seen in Figure 52, HIL simulations were successfully carried out, with the drone achieving stable flying conditions, up to a high altitude. The X-Y path of the drone is plotted in red as can be seen in the bottom right.

4.5 Computer Vision

This section includes describing the experimental set up for data collection and contains the detailed results for both detection and tracking. Both qualitative and quantitative results have been demonstrated for the algorithms analyzed, and detailed analysis of their performance is made along with real-time implementation results.

4.5.1 Experimental Setup

The videos and images were taken from an ESP 870u camera module which was attached to F450 drone and flown over a height of 25m. Several short clips were taken, and the algorithm was tested and evaluated on a personal laptop equipped with GTX 1650 graphics card. Several open-source videos were also taken, and model was implemented on them as well since the videos and images captured from ESP 870u camera module were not of the best quality, and it also doesn't perform well when completely exposed to sunlight. For that purpose, some videos were also taken from DJI Phantom 4 pro drone. For real-time implementation Raspberry-Pi 4 was connected remotely to the laptop using remote desktop connection, and camera was also connected with Raspberry-Pi and model was converted to TFLite format for implementation on Raspberry-Pi.

Figure 53 Data Collected from DJI drone

4.5.2 Quantitative Results

As described in chapter 3.5, precision is the ratio of the actual true positives to all positive predictions made by the model, and recall is the fraction of total positive predictions out of all positives in the ground truth. The precision-confidence curve describes how the precision changes as the classification threshold for detecting the object “car” goes from 0 to 1. As expected, at lower thresholds, precision is lower owing to the higher number of FPs and increases as the classification threshold increases. The recall-confidence shows that recall is higher at lower thresholds, owing to smaller number of FNs or missed detections. The precision-recall curve describes the inverse relationship of both parameters. The F1 score is an average of precision and recall, and therefore gives a combined parameter for evaluating both. The curve for F1 score shows that model has an optimal precision and recall at threshold ~ 0.5. These curves are illustrated in the figures below.

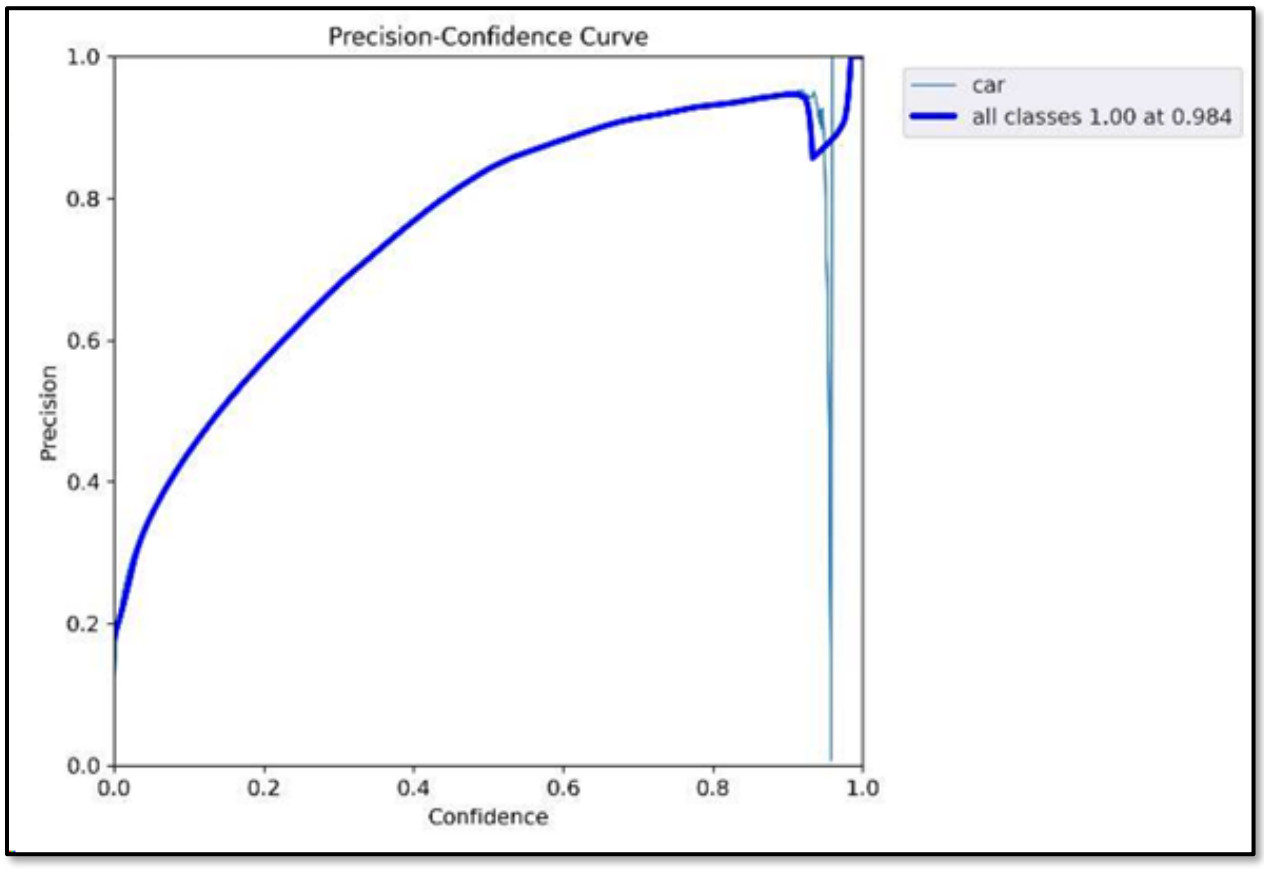


Figure 54 Precision Confidence Curve

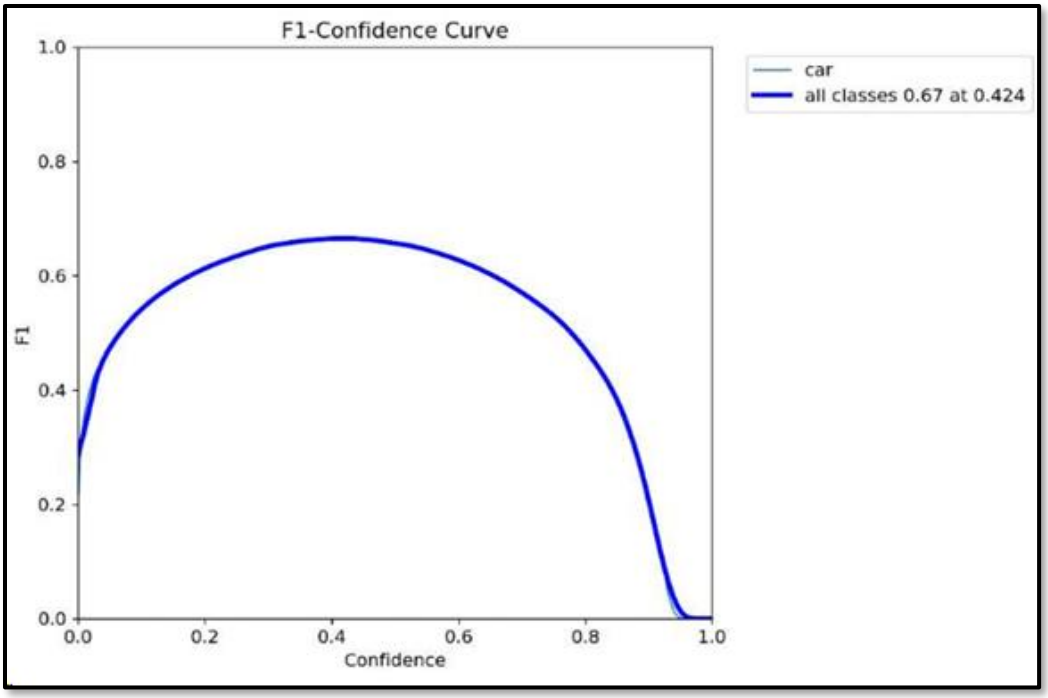


Figure 55 Recall-Confidence Curve

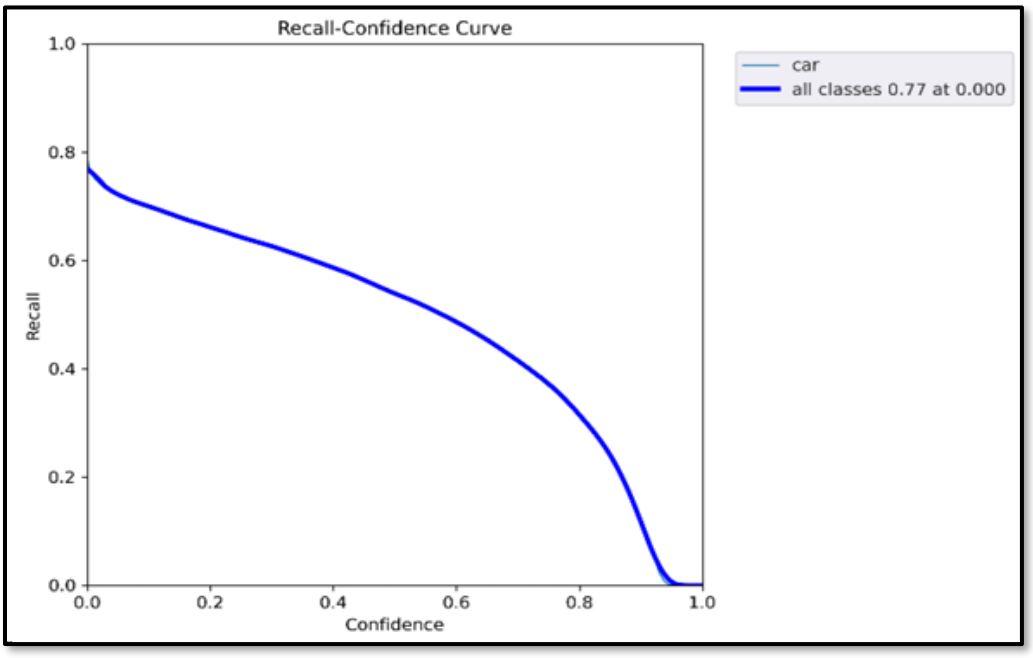


Figure 56 F-1 confidence curve

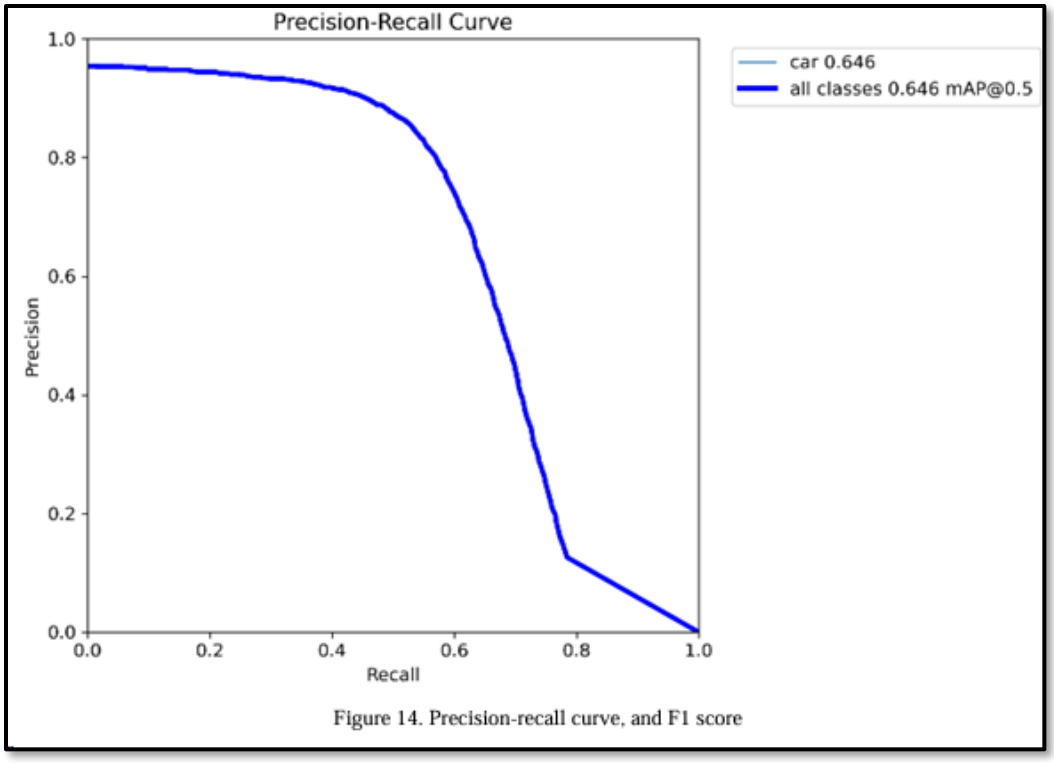


Figure 14. Precision-recall curve, and F1 score

Figure 57 Precision Recall Curve

4.5.3 Qualitative Results

4.5.3.1 Detection

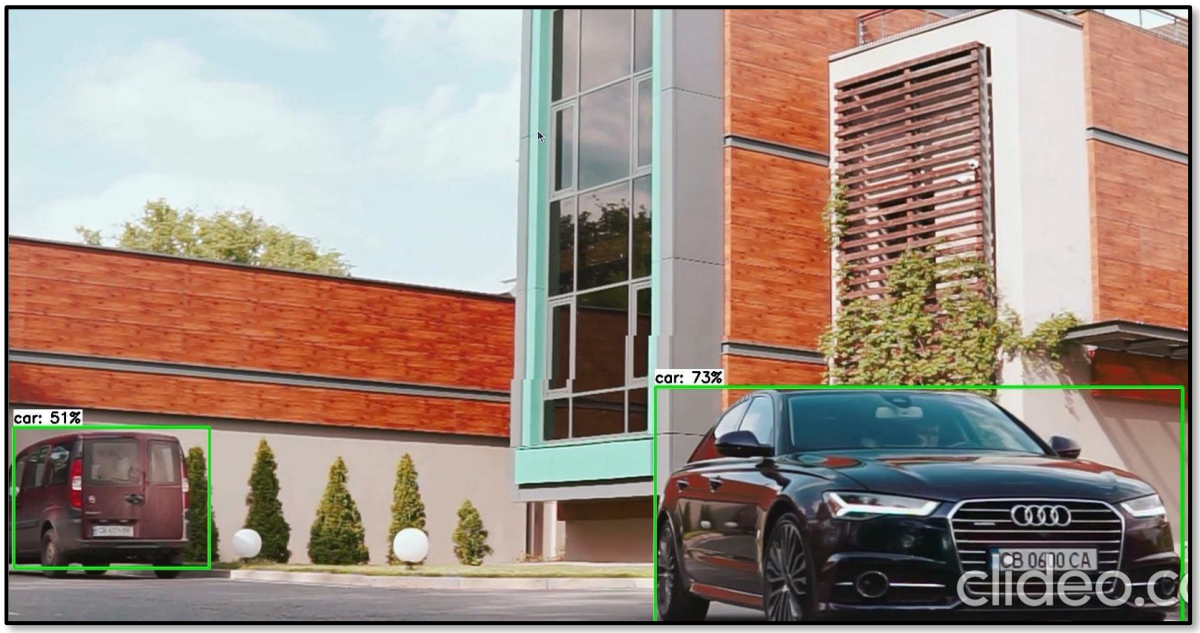


Figure 58 Detection results on Pi

The figures show the detection results from the online available data on which the model was run to obtain the detection results. Overall, the model exhibited good accuracy and seems capable of handling partial occlusions. The results are from both trained model and from. tflite. converted model.

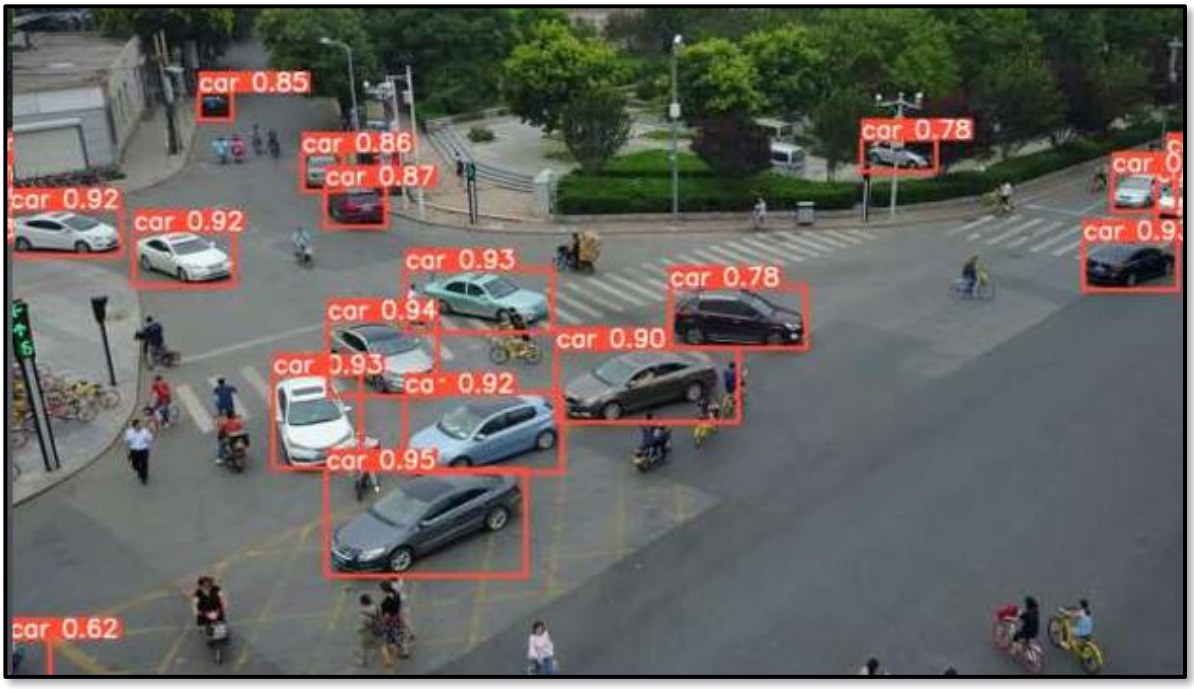


Figure 59 Detection Result on Laptop

4.5.3.2 Tracking

Figure 60 show the frame of a video on which deepsort tracker was applied. Visually, there were very few missed or wrong detections. The tracker seems capable of retaining most object IDs and is also able to recognize and retain partially occluded objects.

Figures 61 and 62 show the real time tracking results done on Raspberry-Pi 4 after converting the model into. tflie format which affected the accuracy of the model and requires good quality videos to give best results. the figure below is a snapshot from a video on which the model was run real-time using the camera as video was being displayed on a screen.

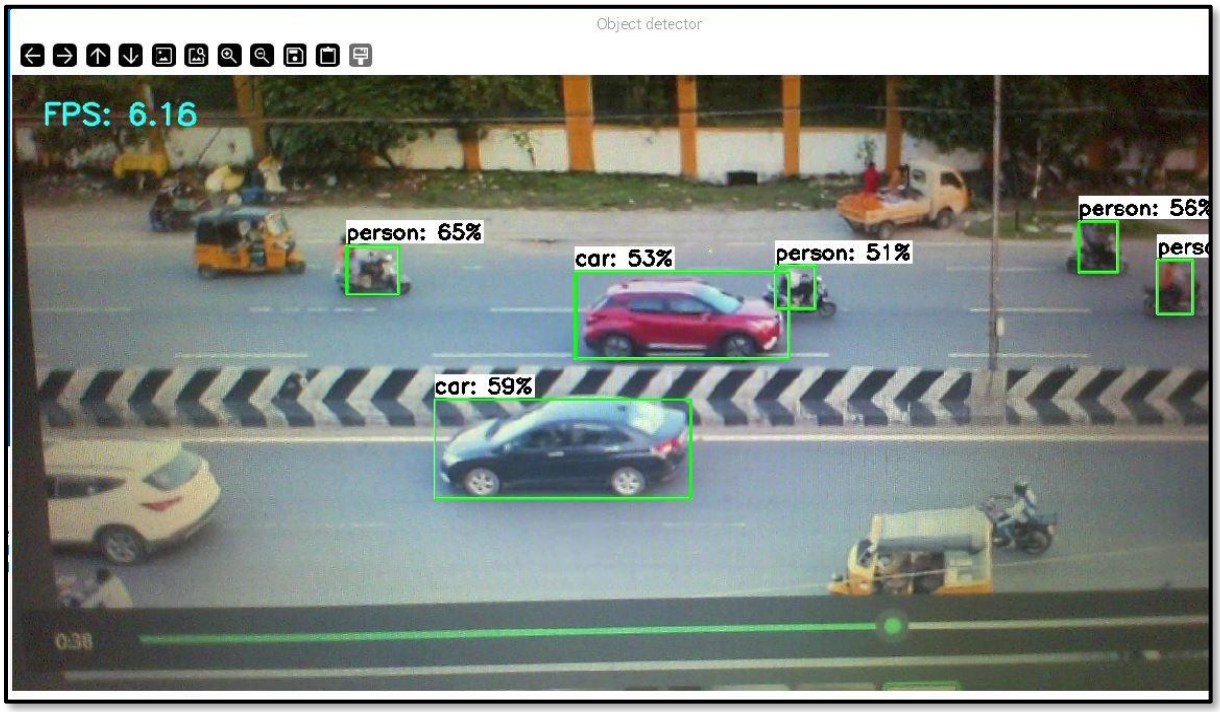


Figure 60 Tracking result on laptop

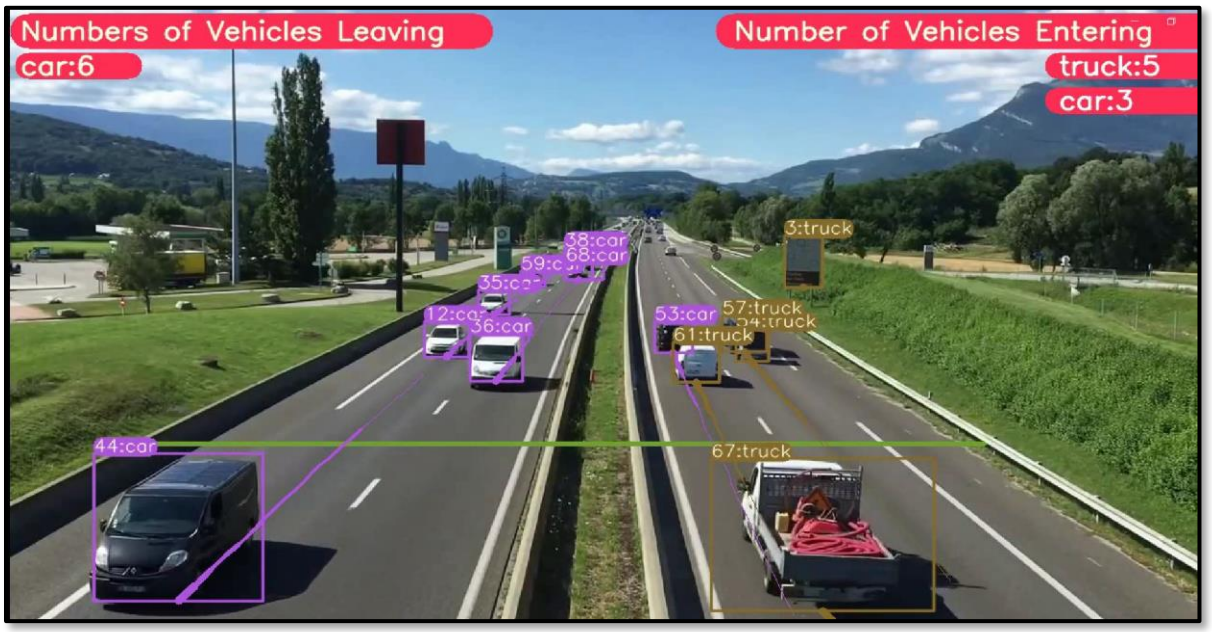


Figure 61 Real-time tracking results

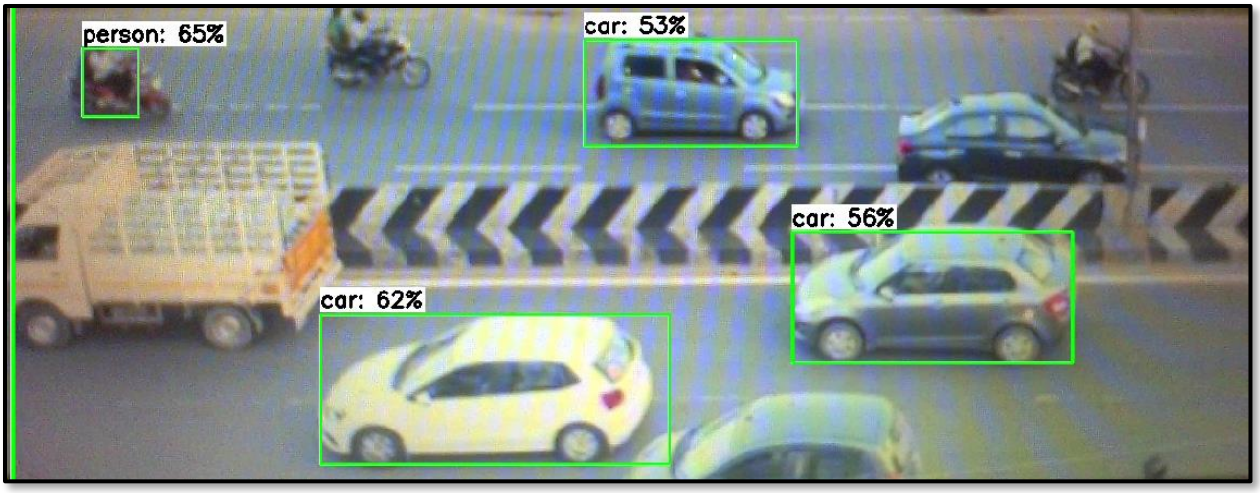


Figure 62 Real-time Tracking results

4.5.4 Analysis

Overall, the detection and tracking algorithms exhibited good visual results on the test data for cars and people. However, the accuracy of the detection algorithm in recognizing multiple viewpoints was not optimal due to the dataset being cut short multiple times during training. The tracking accuracy improved with a greater Intersection Over Union (IOU) threshold, a trend explained by the PyMotMetrics library's quantitative tracking results.

The YOLOv8 model, trained for deployment on the Raspberry Pi, showed a lower accuracy (mean Average Precision ~40) compared to the initial model. This decrease in accuracy is attributed to the smaller model architecture, a necessary compromise to achieve faster inference times on the embedded platform.

5.CONCLUSION

5.1 Conclusion

This thesis has successfully presented the design and implementation of a comprehensive flight control system for a quadcopter, addressing the need for advanced, reliable, and automated flight control solutions in modern drone technology. By developing a 6-DOF mathematical model and a PID-controlled flight controller, and integrating these with various simulation platforms and hardware, the project has achieved its primary objectives and demonstrated significant advancements in both theoretical and practical aspects of drone control systems.

The first major accomplishment was the creation of a detailed 6-DOF mathematical model of the quadcopter using MATLAB/Simulink. This model served as the foundation for developing and testing the PID-controlled flight controller. The PID controller was designed with robustness and efficiency in mind, ensuring precise control over the quadcopter's movements. The integration of this controller with simulation platforms such as ROS/Gazebo and FlightGear facilitated extensive testing and validation, allowing for the identification and resolution of potential issues in a controlled environment before real-world deployment.

One of the standout features of this system is its high degree of customizability and adaptability. The ability to tailor the flight control system to various scenarios and environments is crucial for real-world applications where conditions can be unpredictable and varied. This adaptability was further proven by the successful deployment of the designed controller on PX4 hardware, enabling remote control (RC)-based flight. The practical implementation on physical hardware underscored the system's reliability and effectiveness, bridging the gap between simulation and real-world application.

The integration of object detection and tracking capabilities using a Raspberry Pi added an additional layer of sophistication to the flight control system. This feature enhances the

quadcopter's functionality, making it suitable for applications requiring autonomous navigation and interaction with dynamic environments. The successful integration of these capabilities demonstrates the system's potential for further enhancements and scalability.

The results of this project have shown that the designed flight control system is effective in maintaining the quadcopter's flight stability and safety. Through extensive testing in both simulated and real-world environments, the system has proven its robustness and reliability. Its ability to maintain stable flight under various conditions highlights the success of the PID control strategy and the overall design approach.

Overall, this thesis contributes significantly to the advancement of drone technology by providing a comprehensive and automated approach to designing and implementing flight control systems for quadcopters. The methodologies and findings presented in this work offer valuable insights and practical solutions for future research and development in this field. By combining theoretical modeling, simulation, and practical implementation, this project lays a strong foundation for future innovations in drone flight control systems, paving the way for more advanced, autonomous, and reliable unmanned aerial vehicles (UAVs).

5.2 Future Work

A 6-DOF mathematical model for a quadcopter has been developed using the Newton-Euler method and implemented through simple ODE calculations. This model can be improved by using a state-space representation to handle the complex motion of the quadcopter. Additionally, a control system using cascade-loop PID control has been designed. This can be upgraded to more effective control algorithms, such as adaptive control, model predictive control (MPC), or linear quadratic regulator (LQR).

For the SIMULINK model, a position control block (Mission Profile) has been developed, directing the quadcopter through three predefined waypoints. This feature can be made more flexible to allow users to input any number of waypoints. Incorporating Simultaneous Localization and Mapping (SLAM) or other advanced navigational algorithms, such as

Extended Kalman Filter (EKF) or Particle Filter, can help the quadrotor navigate complex courses efficiently, particularly for indoor flights. Additional features like loiter (hovering in place) and return-to-home can be added to the flight controller. A low battery voltage warning system is another important feature to ensure safety.

The controller has been implemented on hardware through remote control (RC). Future enhancements could include mission planning to enhance user experience. For object detection and tracking, adding event recognition could significantly enhance the project. For example, in traffic monitoring, the system could recognize if a vehicle is moving in the wrong direction or identify accidents to automate rescue calls.

REFERENCES

[1] "(PDF) Robust PID Controller Design for an UAV Flight Control System." Accessed: May 16, 2024. [Online]. Available: https://www.researchgate.net/publication/257133119_Robust_PID_Controller_Design_for_an_UAV_Flight_Control_System

[2] "Design and Simulation of Drone Flight Control Using PID Controller Presentation | PPT." Accessed: May 16, 2024. [Online]. Available: https://www.slideshare.net/slideshow/design-and-simulation-of-drone-flight-control-using-pid-controller-presentation/266735385

[3] "Quadcopter Frame-F450 Online in India at Best Price # Matha Electronics #." Accessed: May 16, 2024. [Online]. Available: https://www.mathaelectronics.com/product/f450-quadcopter-frame/

[4] "Jetson Nano Brings the Power of Modern AI to Edge Devices | NVIDIA." Accessed: May 16, 2024. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/

[5] "Buy a Raspberry Pi 3 Model B – Raspberry Pi." Accessed: May 16, 2024. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

[6] "Homepage - Pixhawk." Accessed: May 16, 2024. [Online]. Available: https://pixhawk.org/

[7] "Open Source Autopilot for Drones - PX4 Autopilot." Accessed: May 16, 2024. [Online]. Available: https://px4.io/

[8] "ArduPilot - Versatile, Trusted, Open." Accessed: May 16, 2024. [Online]. Available: https://ardupilot.org/

[9] "Etron's 3D Stereo Camera and Depth-Map Controller IC — eSP870 Wins COMPUTEX 2014 Best Choice Golden Award - 鈺創科技." Accessed: May 16, 2024. [Online]. Available: https://etron.com/news_list/etrons-3d-stereo-camera-and-depth-map-controller-ic-esp870-wins-computex-2014-best-choice-golden-award/

[10] "Kinect for Windows - Windows apps | Microsoft Learn." Accessed: May 16, 2024. [Online]. Available: https://learn.microsoft.com/en-us/windows/apps/design/devices/kinect-for-windows

[11] "Xtion PRO LIVE." Accessed: May 16, 2024. [Online]. Available: http://xtionprolive.com/asus-xtion-pro-live

[12] "5010 750KV High Torque Brushless Motor for Drone – Havoc Hobby." Accessed: May 16, 2024. [Online]. Available: https://www.havochobby.com/products/5010-750kv-high-torque-brushless-motor-for-drone

[13] "SimonK 30A BLDC ESC 30A Brushless Motor Speed Controller." Accessed: May 16, 2024. [Online]. Available: https://udvabony.com/product/hobbypower-simonk-30a-esc/

[14] "CNHL G+Plus 5000mAh 14.8V 4S 70C Lipo Battery with EC5 Plug — ChinaHobbyLine." Accessed: May 16, 2024. [Online]. Available: https://chinahobbyline.com/products/cnhl-gplus-series-5000mah-14-8v-4s-70c-lipo-battery-with-ec5-plug

[15] "9 BEST Drone Propeller Replacements for Your UAV Drone." Accessed: May 16, 2024. [Online]. Available: https://projectgo.pro/drone-propeller/

[16] S. Wang, X. Dai, C. Ke, and Q. Quan, "RflySim: A Rapid Multicopter Development Platform for Education and Research Based on Pixhawk and MATLAB," *2021 International Conference on Unmanned Aircraft Systems, ICUAS 2021*, pp. 1587–1594, Jun. 2021, doi: 10.1109/ICUAS51884.2021.9476786.

[17] Q. Quan, X. Dai, and S. Wang, "Multicopter Design and Control Practice," *Multicopter Design and Control Practice*, 2020, doi: 10.1007/978-981-15-3138-5.

[18] "GitHub - RflySim/CopterSim: A high-fidelity simulation model developed in Simulink that compatible with different types of multicopters." Accessed: May 13, 2024. [Online]. Available: https://github.com/RflySim/CopterSim

[19] P. J. Benavidez, J. Lambert, A. Jaimes, and M. Jamshidi, "Landing of a quadcopter on a mobile

base using fuzzy logic," *Studies in Fuzziness and Soft Computing*, vol. 312, pp. 429–437, 2014, doi: 10.1007/978-3-319-03674-8_41.
[20] U. Profesional Adolfo López Mateos and S. M. Salazar en C Francisco Villanueva México DF, "INSTITUTO POLITECNICO NACIONAL," 2010.
[21] V. Desai, "MODELING, SIMULATION AND COMPLETE CONTROL OF A QUADCOPTER ME-440 Major Project Semester Report," 2017.
[22] S. Kurak and M. Hodzic, "Control and estimation of a quadcopter dynamical model," *Periodicals of Engineering and Natural Sciences*, vol. 6, no. 1, pp. 63–75, 2018, doi: 10.21533/PEN.V6I1.164.
[23] H. Huang, G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering," *Proc IEEE Int Conf Robot Autom*, pp. 3277–3282, 2009, doi: 10.1109/ROBOT.2009.5152561.
[24] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference 2007*, vol. 2, pp. 1670–1689, 2007, doi: 10.2514/6.2007-6461.
[25] "(PDF) Design and Control of quadrotors with application to autonomous flying." Accessed: May 15, 2024. [Online]. Available: https://www.researchgate.net/publication/37439805_Design_and_Control_of_quadrotors_with_application_to_autonomous_flying
[26] R. Beard and " Quadrotor, "Quadrotor Dynamics and Control Rev 0.1 Quadrotor Dynamics and Control Rev 0.1 Original Publication Citation Original Publication Citation null BYU ScholarsArchive Citation BYU ScholarsArchive Citation," *Dynamics and Control Rev*, no. 1, p. 1325, 2008, Accessed: May 15, 2024. [Online]. Available: https://scholarsarchive.byu.edu/facpubhttps://scholarsarchive.byu.edu/facpub/1325
[27] "Robotics, Vision and Control: Fundamental Algorithms in Python - Peter Corke - Google Books." Accessed: May 15, 2024. [Online]. Available: https://books.google.com.pk/books?hl=en&lr=&id=hhC-EAAAQBAJ&oi=fnd&pg=PR6&dq=P.+Corke,+Robotics,+Vision+and+Control.+Berlin:+Springer,+2013,+pp.+79%E2%80%9390.+&ots=Q1I-6HufPv&sig=lKLeBY0qS0-b6fZyyFq1a0SjcK8&redir_esc=y#v=onepage&q&f=false
[28] A. G. Sidea, R. Y. Brogaard, N. A. Andersen, and O. Ravn, "General model and control of an n rotor helicopter," *J Phys Conf Ser*, vol. 570, 2014, doi: 10.1088/1742-6596/570/5/052004.
[29] O. Bouaiss, R. Mechgoug, and R. Ajgou, "Modeling, control and simulation of quadrotor UAV," *CCSSP 2020 - 1st International Conference on Communications, Control Systems and Signal Processing*, pp. 340–345, May 2020, doi: 10.1109/CCSSP49278.2020.9151687.
[30] "Modeling and Control of A Quad-Rotor Unmanned Aerial | PDF | Quadcopter | Unmanned Aerial Vehicle." Accessed: May 15, 2024. [Online]. Available: https://www.scribd.com/document/710961368/Modeling-and-Control-of-a-Quad-rotor-Unmanned-Aerial
[31] M. S. Büyüksarıkulak, "Autopilot design for a quadrotor /," 2014, Accessed: May 15, 2024. [Online]. Available: https://open.metu.edu.tr/handle/11511/24365
[32] S. Montenegro and A. Lebedev, "Master Thesis Design and Implementation of a 6DOF Control System for an Autonomous Quadrocopter," 2013.
[33] "Dynamic Modeling and Control of a Quadrotor Using Linear and Nonlinear ... - Heba talla Mohamed Nabil Elkholy - Google Books." Accessed: May 15, 2024. [Online]. Available: https://books.google.com.pk/books/about/Dynamic_Modeling_and_Control_of_a_Quadro.html?id=nZr0rQEACAAJ&redir_esc=y
[34] H. Elkholy and M. K. Habib, "Dynamic modeling and control techniques for a quadrotor," *Handbook of Research on Advancements in Robotics and Mechatronics*, pp. 408–454, Dec. 2014, doi: 10.4018/978-1-4666-7387-8.CH014.
[35] "Implementation of Simulink controller design on Iris+ quadrotor : Fum, Wei Zhong : Free

Download, Borrow, and Streaming : Internet Archive." Accessed: May 15, 2024. [Online]. Available: https://archive.org/details/implementationof1094547258
[36] "(PDF) Autopilot Design for a Quadcopter." Accessed: May 15, 2024. [Online]. Available: https://www.researchgate.net/publication/331299062_Autopilot_Design_for_a_Quadcopter
[37] T. L. Hankins, "Pierre Simon Laplace, 1749–1827: A Determined Scientist (Book review)," *Phys Today*, vol. 59, no. 9, pp. 62–64, Sep. 2006, doi: 10.1063/1.2364251.
[38] A. O'dwyer, "ARROW@TU Dublin ARROW@TU Dublin Conference papers School of Electrical and Electronic Engineering," 2005, doi: 10.21427/q287-3p17.
[39] "Nicolas Minorsky and The Automatic Steering of Ships - S. Bennett | PDF | Control Theory | Physics." Accessed: May 15, 2024. [Online]. Available: https://www.scribd.com/document/219298014/Nicolas-Minorsky-and-the-Automatic-Steering-of-Ships-S-Bennett
[40] D. Erdos and S. E. Watkins, "UAV autopilot integration and testing," *2008 IEEE Region 5 Conference*, 2008, doi: 10.1109/TPSD.2008.4562731.
[41] A. Hussein and R. Abdalla, "Autopilot Design for a Quadcopter", doi: 10.13140/RG.2.2.17020.80008.
[42] M. Nithya and M. R. Rashmi, "Gazebo - ROS - Simulink Framework for Hover Control and Trajectory Tracking of Crazyflie 2.0," *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, vol. 2019-October, pp. 649–653, Oct. 2019, doi: 10.1109/TENCON.2019.8929730.
[43] K. Dang Nguyen and T. T. Nguyen, "Vision-Based Software-in-The-Loop-Simulation for Unmanned Aerial Vehicles Using Gazebo and PX4 Open Source," *Proceedings of 2019 International Conference on System Science and Engineering, ICSSE 2019*, pp. 429–432, Jul. 2019, doi: 10.1109/ICSSE.2019.8823322.
[44] A. Andreopoulos and J. K. Tsotsos, "50 Years of object recognition: Directions forward," *Computer Vision and Image Understanding*, vol. 117, no. 8, pp. 827–891, 2013, doi: 10.1016/J.CVIU.2013.04.005.
[45] "Toward Category-Level Object Recognition | Request PDF." Accessed: May 15, 2024. [Online]. Available: https://www.researchgate.net/publication/238720169_Toward_Category-Level_Object_Recognition
[46] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *Int J Comput Vis*, vol. 115, no. 3, pp. 211–252, Dec. 2015, doi: 10.1007/S11263-015-0816-Y/FIGURES/16.
[47] 陳世芳, "Anil K Jain - Fundamentals of Digital Image Processing." Accessed: May 15, 2024. [Online]. Available: https://www.academia.edu/39878253/Anil_K_Jain_Fundamentals_of_Digital_Image_Processing
[48] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun ACM*, vol. 60, no. 6, pp. 84–90, Jun. 2017, doi: 10.1145/3065386.
[49] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," pp. 248–255, Mar. 2010, doi: 10.1109/CVPR.2009.5206848.
[50] "(PDF) Object Detection in 20 Years: A Survey." Accessed: May 15, 2024. [Online]. Available: https://www.researchgate.net/publication/367483279_Object_Detection_in_20_Years_A_Survey
[51] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 580–587, Sep. 2014, doi: 10.1109/CVPR.2014.81.
[52] "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." Accessed: May 15, 2024. [Online]. Available: https://www.researchgate.net/publication/319769911_Faster_R-CNN_Towards_Real-Time_Object_Detection_with_Region_Proposal_Networks
[53] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-October, pp. 2980–2988, Dec. 2017, doi: 10.1109/ICCV.2017.322.

[54] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-December, pp. 779–788, Dec. 2016, doi: 10.1109/CVPR.2016.91.
[55] “Temple Color 128.” Accessed: May 16, 2024. [Online]. Available: https://www3.cs.stonybrook.edu/~hling/data/TColor-128/TColor-128.html
[56] “VISDRONE.” Accessed: May 16, 2024. [Online]. Available: http://aiskyeye.com/
[57] C. Urrea and R. Agramonte, “Kalman Filter: Historical Overview and Review of Its Use in Robotics 60 Years after Its Creation,” *J Sens*, vol. 2021, 2021, doi: 10.1155/2021/9674015.
[58] S. Godsill, “Particle Filtering: The First 25 Years and beyond,” *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2019-May, pp. 7760–7764, May 2019, doi: 10.1109/ICASSP.2019.8683411.
[59] D. Comaniciu and P. Meer, “Mean shift: A robust approach toward feature space analysis,” *IEEE Trans Pattern Anal Mach Intell*, vol. 24, no. 5, pp. 603–619, May 2002, doi: 10.1109/34.1000236.
[60] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, “A comprehensive survey on support vector machine classification: Applications, challenges and trends,” *Neurocomputing*, vol. 408, pp. 189–215, Sep. 2020, doi: 10.1016/J.NEUCOM.2019.10.118.
[61] “A Benchmark and Simulator for UAV Tracking (Dataset) | IVUL | Image and Video Understanding Lab.” Accessed: May 16, 2024. [Online]. Available: https://cemse.kaust.edu.sa/ivul/uav123
[62] “Temple Color 128.” Accessed: May 16, 2024. [Online]. Available: https://www3.cs.stonybrook.edu/~hling/data/TColor-128/TColor-128.html
[63] “Open Images V7.” Accessed: May 16, 2024. [Online]. Available: https://storage.googleapis.com/openimages/web/index.html
[64] P. H. Chen and C. Y. Lee, “UAVNet: An Efficient Obstacel Detection Model for UAV with Autonomous Flight,” *2018 International Conference on Intelligent Autonomous Systems, ICoIAS 2018*, pp. 217–220, Oct. 2018, doi: 10.1109/ICOIAS.2018.8494201.
[65] D. Xu and Y. Wu, “Improved YOLO-V3 with DenseNet for Multi-Scale Remote Sensing Target Detection,” *Sensors 2020, Vol. 20, Page 4276*, vol. 20, no. 15, p. 4276, Jul. 2020, doi: 10.3390/S20154276.
[66] P. Adarsh, P. Rathi, and M. Kumar, “YOLO v3-Tiny: Object Detection and Recognition using one stage improved model,” *2020 6th International Conference on Advanced Computing and Communication Systems, ICACCS 2020*, pp. 687–694, Mar. 2020, doi: 10.1109/ICACCS48705.2020.9074315.
[67] C. Kyrkou, G. Plastiras, S. Venieris, T. Theocharides, and C.-S. Bouganis, “DroNet: Efficient convolutional neural network detector for real-time UAV applications,” *Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition, DATE 2018*, vol. 2018-January, pp. 967–972, Jul. 2018, doi: 10.23919/DATE.2018.8342149.
[68] “(PDF) Autopilot Design for a Quadcopter.” Accessed: May 14, 2024. [Online]. Available: https://www.researchgate.net/publication/331298873_Autopilot_Design_for_a_Quadcopter
[69] M. Idres, O. Mustapha, and M. Okasha, “Quadrotor trajectory tracking using PID cascade control,” *IOP Conf Ser Mater Sci Eng*, vol. 270, no. 1, Dec. 2017, doi: 10.1088/1757-899X/270/1/012010.
[70] “(PDF) Quadrotor Control Using PID Controller.” Accessed: May 15, 2024. [Online]. Available: https://www.researchgate.net/publication/326460585_Quadrotor_Control_Using_PID_Controller
[71] “Cascade Control | Basic Process Control Strategies and Control System Configurations | Textbook.” Accessed: May 15, 2024. [Online]. Available: https://control.com/textbook/basic-process-control-strategies/cascade-control/
[72] K. H. Ang, G. Chong, and Y. Li, “PID control system analysis, design, and technology,” *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, Jul. 2005, doi: 10.1109/TCST.2005.847331.
[73] H. Hou, J. Zhuang, H. Xia, G. Wang, and D. Yu, “A simple controller of minisize quad-rotor

vehicle," *IEEE International Conference on Mechatronics and Automation*, pp. 1701–1706, 2010, doi: 10.1109/ICMA.2010.5588802.
[74] S. Abdelhay and A. Zakriti, "Modeling of a Quadcopter Trajectory Tracking System Using PID Controller," *Procedia Manuf*, vol. 32, pp. 564–571, Jan. 2019, doi: 10.1016/J.PROMFG.2019.02.253.
[75] J. Ren, D. X. Liu, K. Li, J. Liu, Y. Feng, and X. Lin, "Cascade PID controller for quadrotor," *2016 IEEE International Conference on Information and Automation, IEEE ICIA 2016*, pp. 120–124, Jan. 2017, doi: 10.1109/ICINFA.2016.7831807.
[76] "GitHub - HumanSignal/labelImg: LabelImg is now part of the Label Studio community. The popular image annotation tool created by Tzutalin is no longer actively being developed, but you can check out Label Studio, the open source data labeling tool for images, text, hypertext, audio, video and time-series data." Accessed: May 16, 2024. [Online]. Available: https://github.com/HumanSignal/labelImg
[77] Y. Zhang, Z. Wu, X. Wang, W. Fu, J. Ma, and G. Wang, "Improved YOLOv8 Insulator Fault Detection Algorithm Based on BiFormer," *2023 IEEE 5th International Conference on Power, Intelligent Computing and Systems, ICPICS 2023*, pp. 962–965, 2023, doi: 10.1109/ICPICS58376.2023.10235397.
[78] B. Gašparović, G. Mauša, J. Rukavina, and J. Lerga, "Evaluating YOLOV5, YOLOV6, YOLOV7, and YOLOV8 in Underwater Environment: Is There Real Improvement?," *2023 8th International Conference on Smart and Sustainable Technologies, SpliTech 2023*, 2023, doi: 10.23919/SPLITECH58164.2023.10193505.
[79] R. Pereira, G. Carvalho, L. Garrote, and U. J. Nunes, "Sort and Deep-SORT Based Multi-Object Tracking for Mobile Robotics: Evaluation with New Data Association Metrics," *Applied Sciences*, vol. 12, no. 3, Feb. 2022, doi: 10.3390/APP12031319.
[80] Y. Du *et al.*, "StrongSORT: Make DeepSORT Great Again," *IEEE Trans Multimedia*, vol. 25, pp. 8725–8737, 2023, doi: 10.1109/TMM.2023.3240881.
[81] T. Nawaz, F. Poiesi, and A. Cavallaro, "Measures of effective video tracking," *IEEE Trans Image Process*, vol. 23, no. 1, pp. 376–388, Jan. 2014, doi: 10.1109/TIP.2013.2288578.
[82] T. Nawaz and A. Cavallaro, "A protocol for evaluating video trackers under real-world conditions," *IEEE Trans Image Process*, vol. 22, no. 4, pp. 1354–1361, 2013, doi: 10.1109/TIP.2012.2228497.
[83] L. Li, T. Nawaz, J. Ferryman, L. Li, T. Nawaz, and J. Ferryman, "Performance analysis and formative assessment of visual trackers using PETS critical infrastructure surveillance datasets," *JEI*, vol. 28, no. 04, p. 043004, Jul. 2019, doi: 10.1117/1.JEI.28.4.043004.
[84] "Evaluating multiple object tracking accuracy and performance metrics in a real-time setting." Accessed: May 16, 2024. [Online]. Available: https://visailabs.com/evaluating-multiple-object-tracking-accuracy-and-performance-metrics-in-a-real-time-setting/
[85] "OAK & Raspberry Pi — DepthAI Hardware Documentation 1.0.0 documentation." Accessed: May 16, 2024. [Online]. Available: https://docs.luxonis.com/projects/hardware/en/latest/pages/guides/raspberrypi/
[86] "GitHub - Intelligent-Quads/iq_tutorials." Accessed: May 15, 2024. [Online]. Available: https://github.com/Intelligent-Quads/iq_tutorials
[87] "ROS Toolbox - MATLAB." Accessed: May 15, 2024. [Online]. Available: https://www.mathworks.com/products/ros.html
[88] "Create a blank message using specified uORB topic - Simulink." Accessed: May 15, 2024. [Online]. Available: https://www.mathworks.com/help/uav/px4/ref/px4uorbmessage.html