Collision Avoidance for Unmanned Aerial Vehicle

A Thesis submitted in partial fulfilment of the requirement for the degree of

Master of Technology

Submitted by

Ankit Vaishnav

Roll No. 224102501

Supervisor

Dr. Indrani Kar

Co-Supervisor

Dr. Chayan Bhawal

Department of Electronics and Electrical Engineering

Indian Institute of Technology Guwahati

Assam - 781039, India

November, 2023

CERTIFICATE

This is to certify that the work contained in the thesis entitled, **Collision Avoidance for Unmanned Aerial Vehicle**, is a bonafide work of **Ankit Vaishnav (Roll No. 224102501)**, which has been carried out in the Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati under my supervision and this work has not been submitted elsewhere for a degree.

Dr. Indrani Kar
Associate Professor
Dept. of E&EE
IIT Guwahati

Dr. Chayan Bhawal
Associate Professor
Dept. of E&EE
IIT Guwahati

Guwahati
May, 2024

DECLARATION

I hereby declare that the work which is being presented in the thesis entitled, **Collision Avoidance for Unmanned Aerial vehicle**, in partial fulfilment of the requirements for the award of the degree of Master of Technology in Systems, Control and Automation at the Indian Institute of Technology Guwahati is an authentic record of my work carried out under the supervision of **Dr. Indrani Kar**, **Dr. Chayan Bhawal** and references to other researchers' work are duly listed in the reference section. The contents of this thesis, in full or in parts, have not been submitted to any other institute or university for the award of any degree or diploma.

Ankit Vaishnav
Dept. of E&EE
IIT Guwahati

Guwahati
May, 2024

ACKNOWLEDGEMENT

On the path of learning, the presence of an experienced guide is indispensable. I would like to thank my supervisor, **Dr. Indrani Kar**, and my Co-supervisor, **Dr. Chayan Bhawal** for their invaluable help, constant support and persistent guidance throughout the course of this project.

I express my sincere gratitude to the Indian Institute of Technology Guwahati for providing me with the opportunity and resources to accomplish my project.

I would like to thank my family, who helped me reach my goals with consistent encouragement and never-ending support. I want to thank my friends for helping me through this project.

I am grateful to everyone who helped me complete this project successfully.

ABSTRACT

UAVs, commonly known as drones, have emerged as a transformational tool with many applications, from military to agriculture to civilian purposes. This abstract investigates the technological advancement that has propelled UAVs to the forefront of modern technology. However, the risk of collisions, whether with other UAVs, obstacles, or terrain, poses significant challenges to their safe operation. The goal of this project is to improve the operational safety and dependability of unmanned aerial vehicles (UAVs) by creating an efficient collision avoidance system. The main goal is to design an algorithm that can detect and avoid potential collisions in real-time.

The proposed system leverages advanced sensor technologies, specifically the Intel RealSense camera, to capture depth information to identify obstacles and determine the optimal avoidance manoeuvres. By integrating the RealSense camera, the UAV can continuously monitor its surroundings and obtain precise depth measurements.

Path planning was done for a single drone. The collision avoidance algorithm was designed, but practical testing in the real world has not been conducted. Preliminary simulations show the approach's potential efficacy and point to a major increase in UAV safety and navigation in challenging conditions. Future work will focus on real-world validation and extending the system to accommodate multiple UAVs.

List of Tables

5.1 Planned circular waypoints 22
5.2 Actual circular waypoints 23
5.3 Planned rbitrary Wapoints 28
5.4 Actual arbitrary waypoints 28

List of Figures

2.1 Quadcopter connected with pixhawk 4

3.1 Fully Calibrated Drone 11
3.2 Raspberry Pi 12
3.3 Rpi imager 12

4.1 commands to run in terminal 14

5.1 Hover Control Mechanism 17
5.2 Altitude v/s Time Graph 18
5.3 Algorithm for circular path planning 19
5.4 Circular path simulation 21
5.5 Planned v/s Actual path 24
5.6 Planned v/s actual latitude 25
5.7 Planned v/s actual longitude 25
5.8 Algorithm for arbitrary path planning 26
5.9 Simulation result for arbitrary path planning 27
5.10 Simulation output Data 27
5.11 Planned vs actual Path 29
5.12 Simple Path Planning algo using NED cordinates 33
5.13 Path Planning using NED coordinates 34
5.14 NED path planning data 34

6.1 Collision Avoidance algorithm 37
6.2 42
6.3 42
6.4 43

Chapter 1

Introduction

Drones, also known as UAVs, are becoming more common these days, and it has dramatically increased their use in various fields, from precision agriculture and disaster relief to surveillance and reconnaissance. The necessity for advanced control systems to guarantee safe and effective operations grows as these UAVs become increasingly integrated into different sectors. This project aims to achieve this by developing and executing a collision avoidance control strategy for quadcopters.

Safety is the main driving force behind the creation of reliable collision avoidance systems. UAV crashes with other items or other objects in midair have the potential to cause serious material damage, financial loss, and even human injury. Keeping unmanned aerial vehicles (UAVs) safe to fly in both controlled and uncontrolled airspaces reduces the risks involved in using them.

Control system techniques guarantee precision and ensure optimal trajectories and collision avoidance in real-time. The following parts of this report will explore the details of the Collision Avoidance Control Strategy. Through simulations, we will test our strategy thoroughly to ensure it works well. If it does, it can help in control system engineering and make drones fly more independently and safely. [1] offers valuable insights into developing an intuitive understanding of the collision avoidance strategy employed by networked quadcopters.

1.1 Literature Survey

The initial phase of the project involves focusing on hardware calibration and PID tuning for the drone controller that is Pixhawk. Successful hardware calibration and PID tuning are prerequisites for the implementation of the methodologies. A detailed exploration of calibration and PID tuning has been documented in our report. Without successful calibration and PID tuning, it will not be possible to fly the drone in a stable position. MISSION PLANNER software has been utilized for calibration and successful PID tuning.

The main focus is designing a control strategy such that we can fly net-

worked quadcopters without collision. The challenge of designing and controlling networked quadcopters will be addressed in this research. To make the quadcopters communicate with each other, the Mavlink protocol can be utilized so that the quadcopters have information about each other's position and velocity.

The key component of the Ardupilot autopilot software stack is Ardupilot SITL. It makes it possible to simulate Ardupilot-controlled vehicles in a software environment, including vehicles with quadrotors. It is also compatible with real-world implementations given its connection with the Ardupilot flight stack. This simulation environment employs a variety of control algorithms, from more sophisticated techniques like Model Predictive Control (MPC) to the traditional PID controllers. These algorithms control how the quadrotors behave, enabling us to assess and measure their effectiveness.

It is essential to assess a single drone's path-planning algorithms before moving on to collision avoidance.[2] is extensively referred for path planning.Drones' planned course is determined by path planning algorithms. Predictable drone movement is ensured by a well-defined path, which is important for evaluating the way collision avoidance algorithms will work with the intended trajectory. By using path planning to establish a baseline, we can comprehend the drone's typical performance, including its speed, stability, and accuracy of the path. Testing path planning algorithms aid in detecting any basic problems with the drone's itinerary, such as barriers from the surroundings or errors in navigation. It is ensured that collision avoidance algorithms are evaluated in an ideal and regulated environment by addressing these problems first. Path planning and collision avoidance algorithms need to work seamlessly together. Testing path planning ensures that the navigation system is reliable, making integrating and testing the collision avoidance algorithms easier.[3] has been referred for designing an optimal algorithm for collision avoidance.

1.1.1 Work Done:

During PID tuning, drones were iteratively tested with various P, I, and D values to achieve a stable position. The complete procedure and the challenges faced during the tuning have been extensively discussed in this report. Journal articles, including 'Research on attitude controller of quadcopter based on cascade PID control algorithm' [4] were also consulted to supplement our understanding of relevant concepts [5] can be referred for studying the modelling and dynamics of a quadcopter.

The latitude, longitude, and altitude coordinates, also known as (lat, long, alt) coordinates, were initially capitalised to implement the algorithms used for path planning. However, the issue with this coordinate system is that the GPS produces a significant error, which prevents our drone from always going to the specified location. Because of this, utilizing the (lat, long, alt) coordinate system for our collision avoidance algorithm is completely inappropriate.

Instead, a different coordinate system that is far more precise and accurate should be used. The NED coordinate system—North, East, and Down—must be adopted for all of the stated reasons. All the algorithms were first tested in a simulation environment. ArduPilot SITL was utilized for simulation due to its ease of use. After successful implementation of the algorithms in simulation, they were finally tested on real hardware.

Path planning using NED (North, East, Down) coordinates was also conducted in a simulation environment. [6] gives a good idea about how path planning can be done utilizing NED coordinates. However, certain issues were observed with this approach, making the direct implementation of the algorithm on the hardware unsuitable. Further details will be provided in the latter part of the report.

Finally, a collision avoidance algorithm was designed for two quadcopters moving in a straight line, with the forward drone moving at a slower velocity and the back drone at a faster velocity. The faster drone would have access to the position and velocity data of the slower drone [7]. The slower drone leads while the faster drone follows in a straight line, avoiding collisions by following a specific algorithm. However, proper communication between the drones could not be established. In my project, two drones connected via the MAVLink protocol using ArduPilot were considered.

The strategy discussed can be accomplished with much more accuracy with the help of computer vision algorithms [8]. Computer vision is a branch of computer science that enables computers to identify and understand people and objects in images and videos. It seeks to replicate the way humans see, and the way humans make sense of what they see. The onboard cameras, which I have successfully installed in the drone with the help of Raspberry Pi [9], will work as a sensing device or to be more precise an image-sensing device. For this project, we have utilized the INTEL REALSENSE camera as a sensing device. Image sensors or vision sensors capture images or video frames, which will then be processed by computer vision algorithms to recognize and interpret the surroundings. This enables the system to build a visual understanding of the environment. Computer vision will enable drones to simultaneously build a map of their environment while determining their own position within that map [10]. This is particularly crucial for real-time localization. This technique is known as Simultaneous Localization and Mapping or SLAM in short. The technique discussed is part of future work. The path planning method was thoroughly evaluated on both real hardware and in simulation. The drone was moving to the designated arbitrary points based on path planning, which involved selecting random points. Additionally, the drone is moved in a circle using one of the path-planning algorithms. The drone moved along the designated path by using the GPS on the drone controller.

Chapter 2

Background

2.1 Basics of Quadcopter

An unmanned aerial vehicle (UAV) or drone having four rotors, each with a motor and propeller, is called a quadrotor or quadcopter. A quadcopter can be operated autonomously or manually. It is also known as a quadrotor or a quadrotor helicopter. It is a member of the multicopter or multirotor class of aerial vehicles, which is a broader category.

Due to their inexpensive cost, minimal inertial force, and easy flight control system, small quadcopters are simple to construct. Quadcopters are perfect for aerial photography and surveillance since they have steady flying characteristics. Delivery, land surveys, agricultural evaluation, weather broadcasting, and more are further application fields. There are several sizes of quadcopters, ranging from palm-sized to ones that can transport large amounts of freight or people.

Figure 2.1: Quadcopter connected with pixhawk

Design and Structure

Newton's Third Law of Motion serves as the fundamental theory governing a quadcopter's motion. The propellers of a quadcopter push downward, producing an equal and opposite thrust that raises the machine. The air movement results from Bernoulli's Principle, which states that larger propeller blades and quicker rotations amplify thrust.

However, because of torque, a quadcopter's propellers spinning in one direction tend to rotate oppositely. Helicopters use their tail rotors to overcome this.Unlike helicopters, which employ tail rotors, this is resolved by rotating two diagonal propellers clockwise and the other two counterclockwise.

When the thrusts generated from the diagonal propeller pairs point in opposing directions, there may be an issue since the quadcopter's ability to climb or fly may be hampered by the propellers rotating in opposite directions. By creating mirror images of each propeller blade in each diagonal pair, this problem is resolved. By doing this, the quadcopter may achieve steady flight since all of its propellers will push air downward regardless of the direction of rotation.

Flight Dynamics and Control

Quadrotors can execute complex maneuvers because of their dynamic flying characteristics. The quadcopters may perform pitch, roll, and yaw movements by varying the speed of each rotor. A number of complex control systems control the quadrotors' functions. PID (proportional-integral-derivative) controllers are frequently used to control translational and rotational motions. Further sophisticated control algorithms, such as Model Predictive Control, improve their flexibility and reactivity.

Sensors and Perception

Quadrotors use a variety of sensors, including GPS, accelerometers, gyroscopes, and Inertial Measurement Units (IMUs), for efficient navigation and stabilization. In addition, depth sensors, onboard cameras, and computer vision algorithms aid in obtaining a sense of the environment.

Autonomous Navigation

Thanks to AI and machine learning, quadrotors have advanced into the autonomous navigation space. These unmanned vehicles are independently capable of exact spatial location, obstacle avoidance, and course planning. These capabilities make mapping, exploration, and surveillance applications possible.

Applications

Quadrotors are used in many different fields. They support precision farming and crop monitoring in the field of agriculture. Their quick survey skills come in very handy during search and rescue operations. Quadrotors are used by the entertainment industry for aerial cinematography, which produces stunning and dynamic images.

2.2 Collision Avoidance

Importance of collision avoidance

The risk of mid-air collisions has grown significantly with the proliferation of UAVs, particularly in areas with high traffic or densities of population. Systems for avoiding collisions are essential to guaranteeing the dependability and safety of UAV operations. These technologies improve flight safety and operating efficiency by enabling UAVs to identify possible impediments and conduct evasive maneuvers to prevent crashes.

Types of collision avoidance

There are several types of collision avoidance systems employed in UAVs:

- **Passive Systems:** Rely on predefined flight paths and air traffic control to prevent collisions.
- **Active Systems:** Use sensors (e.g., radar, LiDAR, cameras) and real-time data processing to detect obstacles and make dynamic adjustments to the flight path.

Algorithms for collision avoidance

- **Reactive Algorithms:**Adjust the UAV's trajectory in real-time in response to impediments that are detected.
- **Predictive Algorithms:**Make use of sensor data to anticipate possible collisions and schedule preventive actions ahead of time.
- **Path Planning Algorithms:**Before the mission begins, take into account the UAV's surroundings and any potential obstructions to create safe flying pathways.

UAV Communication Protocols

Coordinated operations and collision avoidance among numerous UAVs depend on communication between them and ground control centers.

Critical flight data and directives can be sent more easily thanks to protocols like MAVLink (Micro Air Vehicle Communication Protocol).

Chapter 3

Basic Setup

Basic setup involves the calibration, PID tuning, and setup for the microcontroller used. In my project, I have utilized Raspberry as a microcontroller. For proper operation of the drone, communication is established between the Pixhawk and Raspberrypi such that commands can be sent to Pixhawk via Raspberrypi.

The flight controller that is put in the drone is responsible for PID tuning and calibration. The Pixhawk controller is the one utilized in this project.

3.1 Calibration

For caliberation we have used the Ardupilot software. ArduPilot is an open source, Unmanned vehicle Autopilot Software Suite, capable of controlling autonomous:

- Multirotor drones
- Fixed-wing and VTOL aircraft
- Helicopters
- Ground rovers
- Boats
- Submarines
- Antenna trackers
- Blimps

The ArduPilot software package includes ground station controlling software such as Mission Planner, APM Planner, QGroundControl, MavProxy, Tower, and others, in addition to navigation software that runs on the vehicle. In this project we have used mission planner.

The key components that requires caliberation are Accelerometers, Gyroscopes, Magnetometers (Compass), Barometers, GPS, Inertial Measurement Unit, Propellers and Motors. Now we will see the steps by which caliberation is achieved .

Choosing Frame Type: As our quadcopter is in an X shape, we will choose an "X" type frame.

Accelerometer Calibration: Mission Planner's accelerometer calibration is essential when configuring a drone. It guarantees accurate and stable sensor readings. Use Mission Planner to find it by going to "Initial Setup" > "Install Firmware" > "Accel Calibration." For best drone performance, calibrate and adhere to the on-screen instructions.

Compass calibration: Mission Planner's compass calibration is essential to guarantee precise heading data for a drone. Use Mission Planner to find it by going to "Initial Setup" > "Install Firmware" > "Compass Calibration." To improve the precision of our navigation, calibrate the compass by following the on-screen directions, which usually include rotating the drone around different axes.

Radio calibration: To guarantee precise communication between the remote control (RC) transmitter and the drone's flight controller, radio calibration in Mission Planner entails setting and calibrating the RC transmitter. In Mission Planner, radio calibration entails setting up our drone's remote control transmitter. Go to "Initial Setup" > "Radio Calibration" to access it. To ensure precise communication between the transmitter and drone, move each control through its entire range by following the on-screen directions.

ESC calibration: To ensure that the flight controller precisely regulates the speed of each motor on our drone, ESC (Electronic Speed Controller) calibration in Mission Planner is an essential step.Go to "Initial Setup" > "Motor Test" to access it. To calibrate ESCs, adhere to the on-screen instructions, which typically call for particular throttle inputs and power cycles. Verify that the motor responses are smooth, then save the setup to ensure steady flight control.

3.2 PID Tuning

We optimized the drone's stability and reactivity by fine-tuning the Proportional (P), Integral (I), and Derivative (D) components throughout the PID tuning procedure. We addressed identified oscillations or overshooting by methodically adjusting these variables through iterative testing, starting with default values. The importance of sensor calibration for precise feedback was stressed. The flight controller's tuned PID parameter integration was done with great care. The goal of the PID controller is to correct the "error", the difference between a measured value (gyro sensor measurement), and a desired set-point (the desired rotation speed).

The drone has three main axes of motion: roll (tilting left/right), pitch (tilting forward/backwards), and yaw (rotation around the vertical axis). The goal of the PID controller is to adjust the motor speeds to maintain the desired orientation and altitude.

3.2.1 Steps for PID tuning

Step 1: We begin with some default values of P,I, and D provided by our software.

Step 2: Modifying p along the yaw, pitch, and roll axes. Increase the value of p a little at a time while watching for drone manure if the drone exhibits oscillations, such as a regular up-and-down or side-to-side movement. Reduce the value of p by a small amount. Low P values can contribute to the sluggish response, and high P values can cause the drone to exceed its target position. This can be especially visible during fast movements.

Step 3: Including I term lowers the error in the steady state. Increase I's value gradually until the error shows a consistent shift. We want to set it at a reasonable price since it might lead to instability. When drifting occurs, even without external inputs, the drone cannot keep a steady location and wanders progressively in one direction. Incorrect I (Integral) values cause this.

Step 4: Introduce D term to dampen oscillations and enhance stability. Increase D until oscillations are adequately damped. Be cautious not to set it too high to avoid introducing noise. Inadequate D (Derivative) values can result in delayed corrections to changes in the drone's orientation. This can affect the drone's ability to stabilize itself quickly after disturbances.

One important thing to note is that we have simultaneously adjusted the P, I, and D values for each axis, i.e., roll, pitch, and yaw.

3.2.2 Challenges in Tuning P, I, and D Parameters

We encountered two primary challenges during the calibration process:

Yawing During Takeoff:

After taking flight, the drone began to yaw. To maximize flying performance, we made repeated adjustments to the proportional (P), integral (I), and derivative (D) values. PID parameter fine-tuning is an iterative process that requires cautious modifications and patience. PID performance can be affected by variables like payload weight and battery status, necessitating careful calibration. There may be some trial and error involved in finding the perfect balance for a gentle landing.

Soft Landing Difficulty:

Achieving a soft landing necessitated careful tuning of PID values. Patience is crucial, as minor adjustments are often needed. External factors, such as changes in payload weight and battery conditions, can impact PID performance. Safety remains a top priority during testing, and documenting findings helps refine tuning over time.

3.2.3 Result

The values of P,I, and D for which we achieved successful calibration are :
Stabilize Roll: P=6.5
Stabilize Pitch: P=6.5
Rate Roll: P=0.250, I=0.200, D=0.0036
Rate Pitch: P=0.250, I=0.200, D=0.006
Rate Yaw : p=0.180, I =0.018

Figure 3.1: Fully Calibrated Drone

3.3 Raspberry-Pi

Raspberry Pi (often abbreviated as RPi) is a series of single-board computers developed by the Raspberry Pi Foundation. Raspberry Pi boards are equipped with various hardware interfaces, including GPIO (General-Purpose Input/Output) pins, making them suitable for interfacing with external devices and sensors. They run on different operating systems, with Raspberry Pi OS (formerly Raspbian), Linux, etc.

The first step is to configure the Raspberry Pi by choosing an appropriate operating system before attaching the Intel RealSense camera to the Pi. Because Ubuntu 20.04.1 is compatible with ROS Noetic, we chose it for our

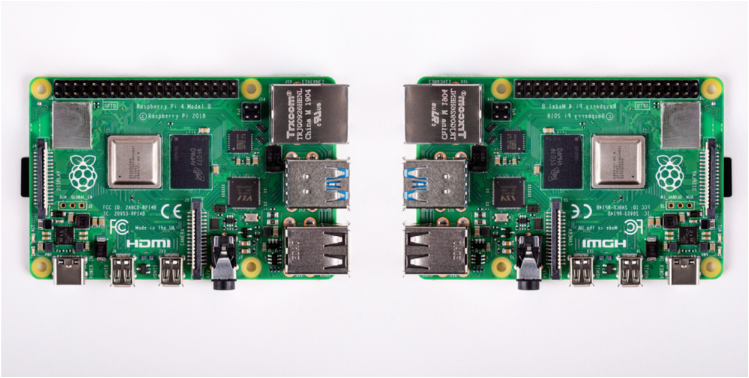

Figure 3.2: Raspberry Pi

project. The instructions for burning the selected operating system onto the Raspberry Pi are provided in the following phases:

Step 1: To begin, acquire an SD card with a minimum capacity of 32 GB. Subsequently, download the suitable image file for your SD card from the following link: https://releases.ubuntu-mate.org/20.04/arm64/. Once the download is complete, proceed to execute the image file.

Step 2: After launching the imager, we must choose our OS's storage, which is an SD card; once chosen, we click the write option. Our OS will then be loaded into the Raspberry Pi.

Figure 3.3: Rpi imager

Step 3: We now need to insert our SD card into the RPI's slot, link it to the display via an HDMI connection, and supply power for the RPI. OS is now loaded on RPI. The Raspberry Pi is now prepared for integration with Pixhawk.

Step 4: Now important packages and libraries like MAVproxy and dronekit needs to be installed in the Raspberry Pi. MAVproxy is used to send commands to the flight controller from the Pi. For this official documentation of Ardupilot is followed.

Step 5: We can now integrate our Intel-real sense camera with Raspberry Pi. After the successful integration of Raspberry Pi with the camera, it is necessary to calibrate our camera so that we can actual depth information and the camera can be used for object detection.We can find the Intel RealSense Depth D435i IMU Calibration document here: Intel RealSense Depth D435i IMU Calibration

Chapter 4

Simulation using Ardupilot autopilot and Gazebo

4.1 Installations

For the installation, setup, and commands, we have mainly used the official Ardupilot -Autopilot documentation and the official ROS-Wiki, which is an open-source software used for autonomous mission capabilities of UAVs and UGVs. Linux(Ubuntu 20.04) was used as the OS for the work, keeping in mind the compatibility as well as the limitations of the simulation environment and the Ardupilot-Software.

4.2 Running the simulation

In the Linux terminal we can run the following commands:

Launching the `ardupilot_sitl` gazebo file results in the following output at the terminal:

Figure 4.1: commands to run in terminal

4.3 Multi-quadrotor simulation

Here is the link to the tutorial on swarming with ArduPilot: Swarming with ArduPilot

Connecting Multiple Vehicles SITL to Gazebo

We can think of Ardupilot as purely a control system. It takes sensor inputs and outputs commands to actuators. Gazebo is playing the role of a Flight Dynamics Model (FDM). The FDM encompasses all of the following, sensor models, actuator models, and the dynamics of the vehicle. In real life ardupilot would communicate with your sensors via serial connections, but when you run SITL with a gazebo ardupilot talks to the sensors and actuators via UDP connections (an IP protocol). Because of this, we will need to specify different UDP ports in our robot models such that these streams do not conflict.

Connecting Multiple drones to ground station

To connect multiple unique vehicles to a ground station, we must ensure that the TCP or UDP connection ports do not conflict. In my simulation, I have used the TCP port. We first need to relaunch our SITL Ardupilot instances with a unique in/out TCP port for our GCS.

Setup ground station to accept multiple vehicles

For this, we need to navigate to the settings tab and click on "Comm Links". then find the fill-in below for the vehicle's unique TCP ports:

4.4 Challenges

The biggest challenge was running a multi-vehicle simulation, which required extensive internal modifications to the SITL files. Although I was able to connect multiple drones to the simulation environment, I was unable to establish communication between them since my multi-drone simulation could not be finished.

Note:

The simulation with one drone worked well. Every method for a single drone simulation was tested, and success was achieved.

Chapter 5

Path Planning

Path planning with drones involves designing a trajectory for the drone to follow from a starting point to a destination while avoiding obstacles and optimizing certain criteria.This procedure makes use of algorithms that may take into consideration the dynamic environment in which drones operate, which includes shifting terrain, shifting weather, and moving obstructions. For applications including delivery services, search and rescue operations, agricultural monitoring, and infrastructure inspection, effective path planning is essential to ensuring safe and efficient navigation.

5.1 Hover control

The first step before we can test for any path planning algorithm is testing the hover control mechanism, here we give the command to our drone to take off to a desired altitude hover for a while, and subsequently land after that.

Here is a simple algorithm for that:

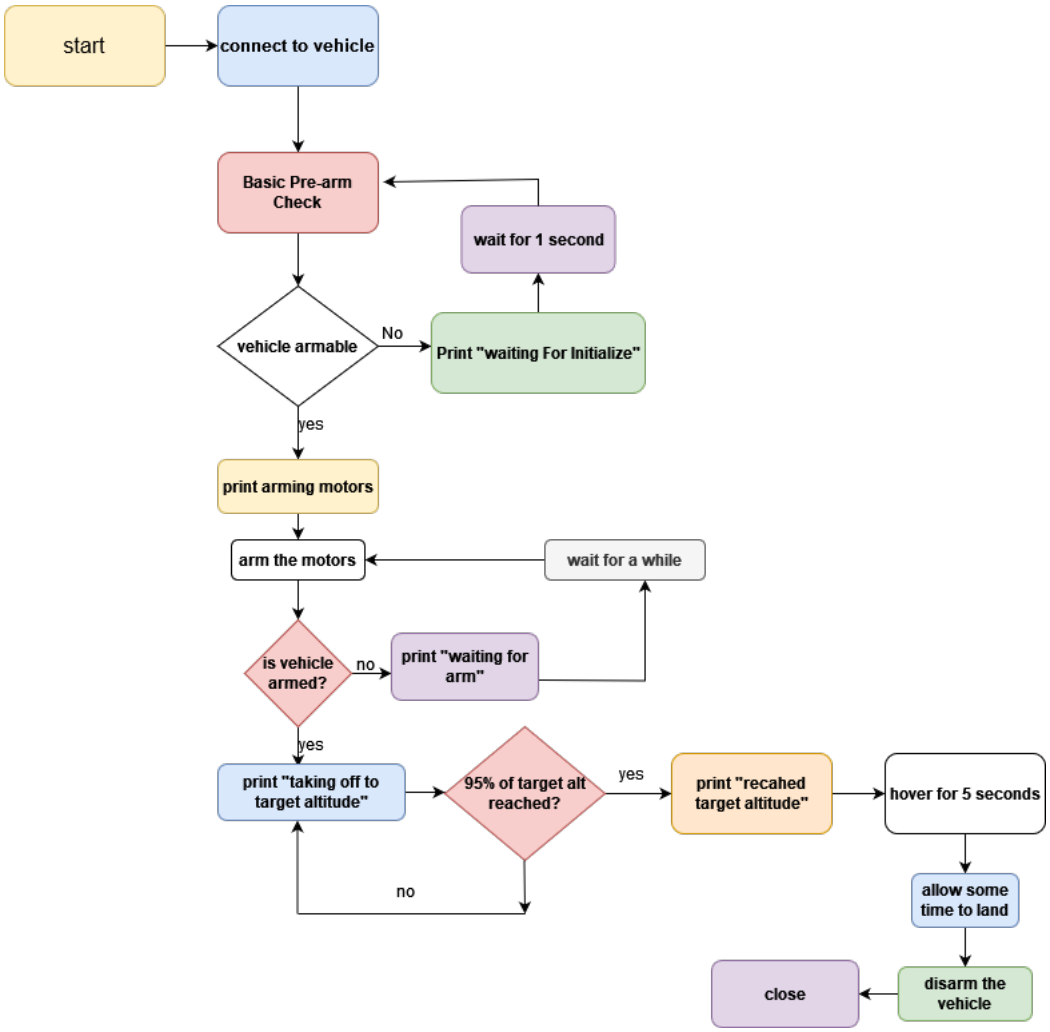

Figure 5.1: Hover Control Mechanism

Explanation:

The first step of the hover control algorithm is to connect to the vehicle and initialize the system. It waits for stabilization, carries out some basic pre-arm tests, and then signals that it is awaiting startup. After the vehicle is armed, it prints a message, engages the motors, and verifies that the arming process was successful. After that, the algorithm prints a confirmation message, announces the takeoff to the desired altitude, and checks to see if 95% of the target altitude has been attained. The UAV hovers for five seconds at the target height before beginning the landing procedure, disabling the motors and ending the mission.

Result:

The altitude log values are:

```
altitude_log = [0,0,0.03,0.36 0.527, 1.06, 1.47, 1.639,
    1.569, 1.561, 1.573, 1.56, 1.44, 1.027, 0.56, 0.06,0]
```

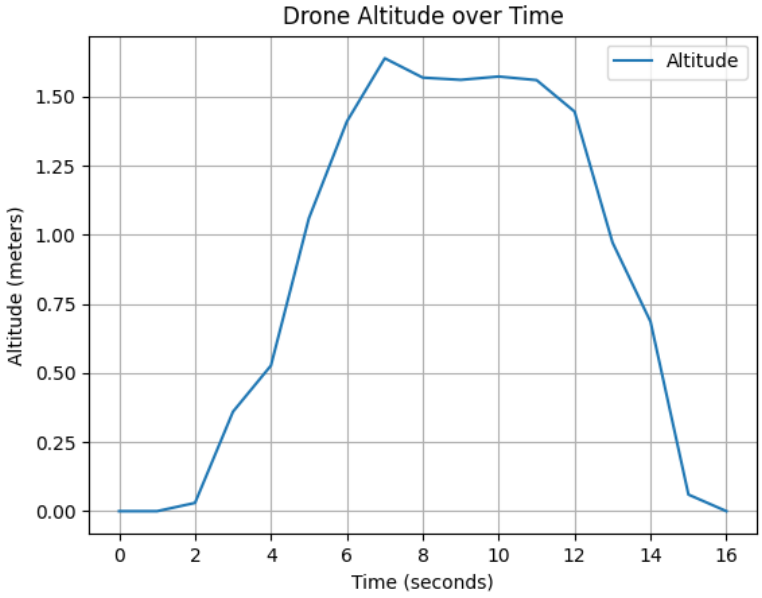

Figure 5.2: Altitude v/s Time Graph

5.2 Circular Path Planning

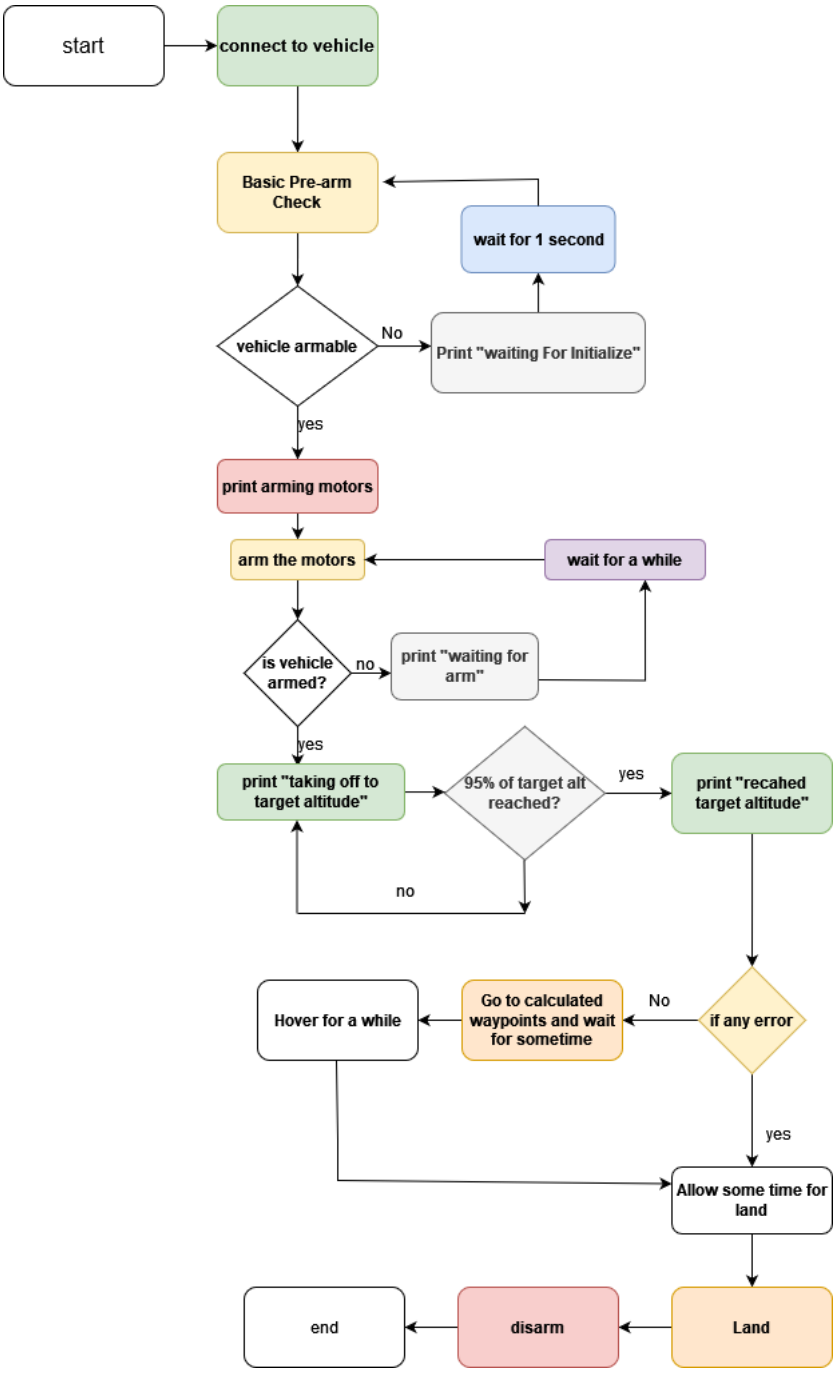

Figure 5.3: Algorithm for circular path planning

Explanation:

The purpose of this algorithm is to generate waypoints for a circular path around a given centre point. These waypoints can be used for navigating a drone in a circular trajectory. The algorithm involves defining a centre point, specifying a radius, determining the number of waypoints, and calculating the

latitude and longitude for each waypoint using trigonometric and geographic calculations.The algorithm uses trigonometric functions to determine the displacement of each waypoint from the centre point. These displacements are then converted from meters to degrees of latitude and longitude, taking into account the curvature of the Earth.

Inputs

- Center latitude (`center_lat`)
- Center longitude (`center_lon`)
- Radius (`radius`)
- Number of waypoints (`num_points`)

Calculation of Waypoints

Explanation of waypoint calculation:

Angle Calculation

The circle is divided into equal segments based on the number of waypoints. The angle for each waypoint is calculated as:

$$\text{angle} = i \times \left(\frac{360}{\text{num\_points}} \right) \tag{5.1}$$

Convert the angle to radians:

$$\text{angle\_radians} = \text{math.radians(angle)} \tag{5.2}$$

Displacement Calculation

The x (east-west) and y (north-south) displacement calculation in meters:

$$dx = \text{radius} \times \cos(\text{angle\_radians}) \tag{5.3}$$

$$dy = \text{radius} \times \sin(\text{angle\_radians}) \tag{5.4}$$

Latitude Conversion

Convertion of north-south displacement (dy) from meters to degrees of latitude:

$$\begin{aligned} dLat = \frac{dy}{111132.92 - 559.82\cos\left(2 \times \text{math.radians(center\_lat)}\right)} \\ +1.175\cos\left(4 \times \text{math.radians(center\_lat)}\right) \\ -0.0023\cos\left(6 \times \text{math.radians(center\_lat)}\right) \end{aligned} \tag{5.5}$$

Longitude Conversion

Convertion of east-west displacement (dx) from meters to degrees of longitude:

$$\begin{aligned} dLon = \frac{dx}{111412.84\cos\left(\text{math.radians}(\text{center\_lat})\right)} \\ -93.5\cos\left(3 \times \text{math.radians}(\text{center\_lat})\right) \\ +0.118\cos\left(5 \times \text{math.radians}(\text{center\_lat})\right) \end{aligned}$$

Calculation of Waypoint Coordinates

Calculate the latitude and longitude of the waypoint by adding the displacements to the centre point:

$$\text{lat} = \text{center\_lat} + dLat \tag{5.7}$$

$$\text{lon} = \text{center\_lon} + dLon \tag{5.8}$$

Results:

Figure 5.4 can be referred to for simulation results.

Figure 5.5 shows the comparison between the planned path and the actual path followed by the drone.

To see how planned latitude and longitude differ from actual latitude and longitude, consult Figures 5.6 and 5.7.

Figure 5.4: Circular path simulation

.

Table 5.1 shows the planned circular waypoints and Table 5.2 shows the actual circular waypoints.

Table 5.1: Planned circular waypoints

Point	**Latitude**	**Longitude**
1	26.19020131	91.6972566
2	26.19017025	91.6972747
3	26.19018283	91.69728829
4	26.19019922	91.697291827
5	26.19020143	91.697284464
6	26.19029281	91.69726818
7	26.19010615	91.697242868
8	26.19011260	91.69722475
9	26.19013185	91.697211173
10	26.1901576	91.69720764
11	26.19018198	91.69721400
12	26.19019778	91.697230281
13	26.19020131	91.69725668
14	26.19019486	91.6972747
15	26.19017561	91.6972882
16	26.19014978	91.69729182
17	26.19012548	91.69728446
18	26.19010968	91.6972681
19	26.19010615	91.69724286
20	26.19011260	91.6972247

Table 5.2: Actual circular waypoints

No.	Latitude	Longitude
1	26.1901536	91.69730661
2	26.19014507	91.69730677
3	26.19014089	91.69730676
4	26.19013401	91.69730148
5	26.1901269	91.69729396
6	26.19011903	91.69729185
7	26.19011607	91.69728556
8	26.19011439	91.6972772
9	26.19011132	91.69727185
10	26.19011056	91.69726505
11	26.19010866	91.69725453
12	26.19010692	91.69724496
13	26.19011342	91.69724019
14	26.19011523	91.69723005
15	26.19012048	91.69722539
16	26.19012241	91.69722224
17	26.19012925	91.69721368
18	26.19013621	91.69720838
19	26.19014228	91.69720825
20	26.19014927	91.69720791
21	26.19015842	91.69720433
22	26.19016338	91.69720853
23	26.19016895	91.69721114
24	26.19017725	91.69721252
25	26.19018432	91.69721637
26	26.19018706	91.69722251
27	26.19019207	91.69723402
28	26.1901962	91.6972376
29	26.19019941	91.69724898
30	26.1902006	91.69725441
31	26.19019935	91.69726021
32	26.19019913	91.69727064
33	26.19019324	91.69728065
34	26.19018877	91.69728205
35	26.19018482	91.69728922
36	26.19018322	91.69729401
37	26.19017389	91.69730271

In our case the centre latitude = 26.190634, centre longitude = 91.696732.

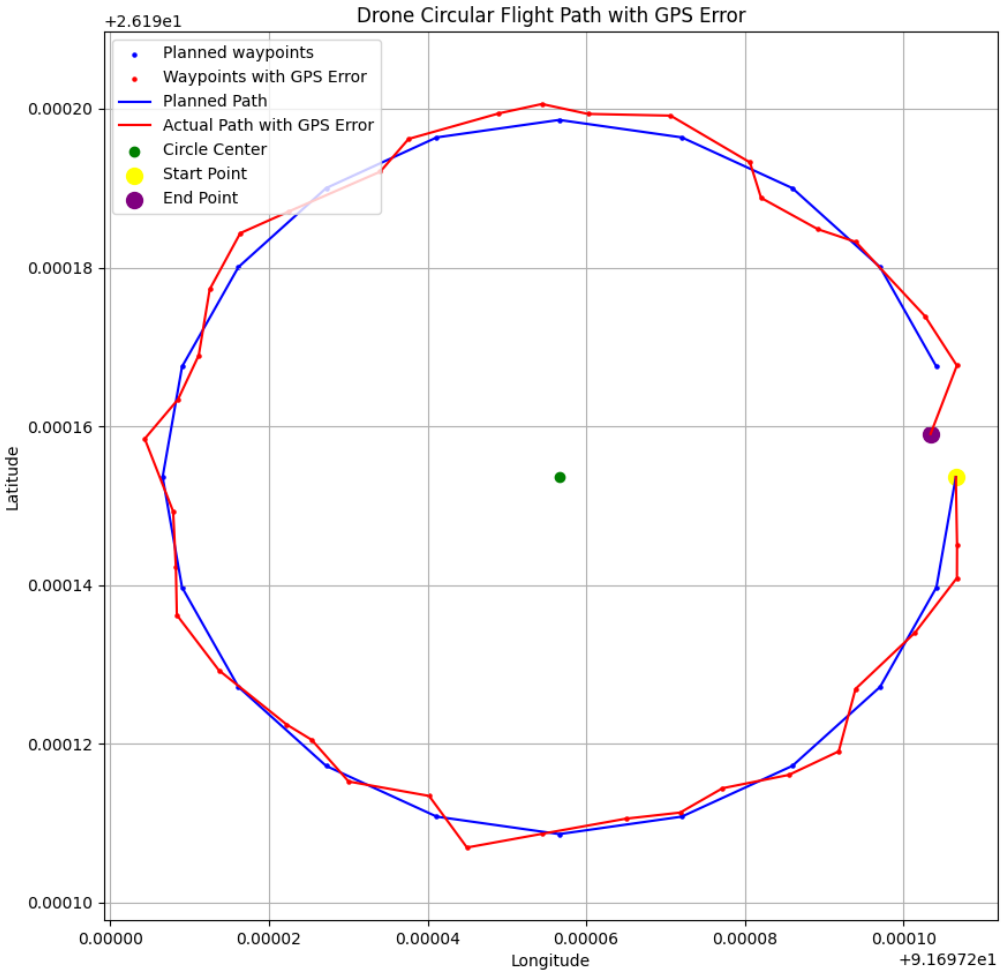

Figure 5.5: Planned v/s Actual path

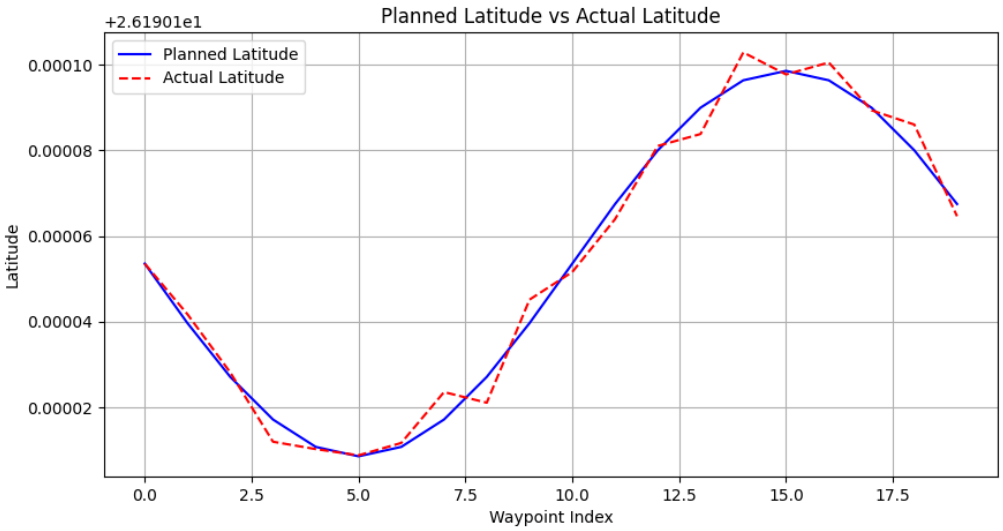

Figure 5.6: Planned v/s actual latitude

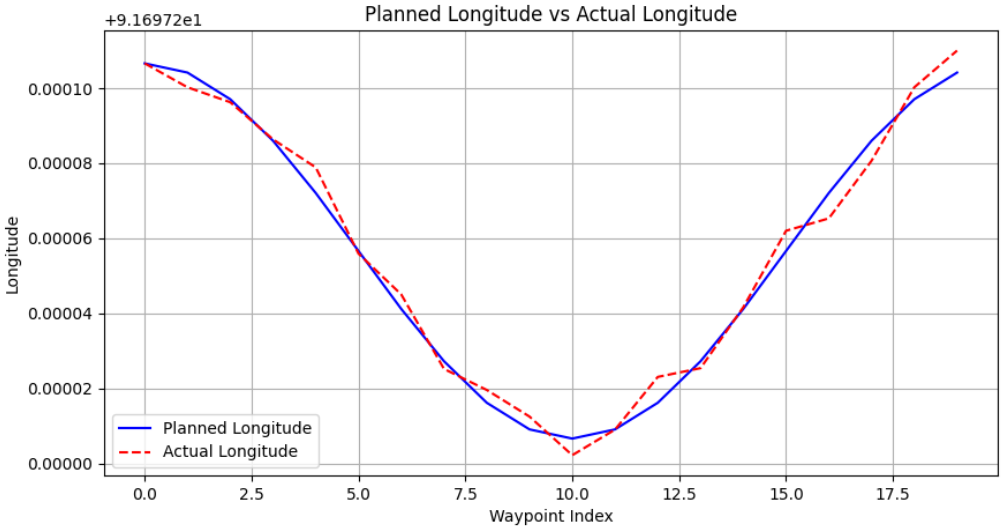

Figure 5.7: Planned v/s actual longitude

5.3 Arbitrary path planning

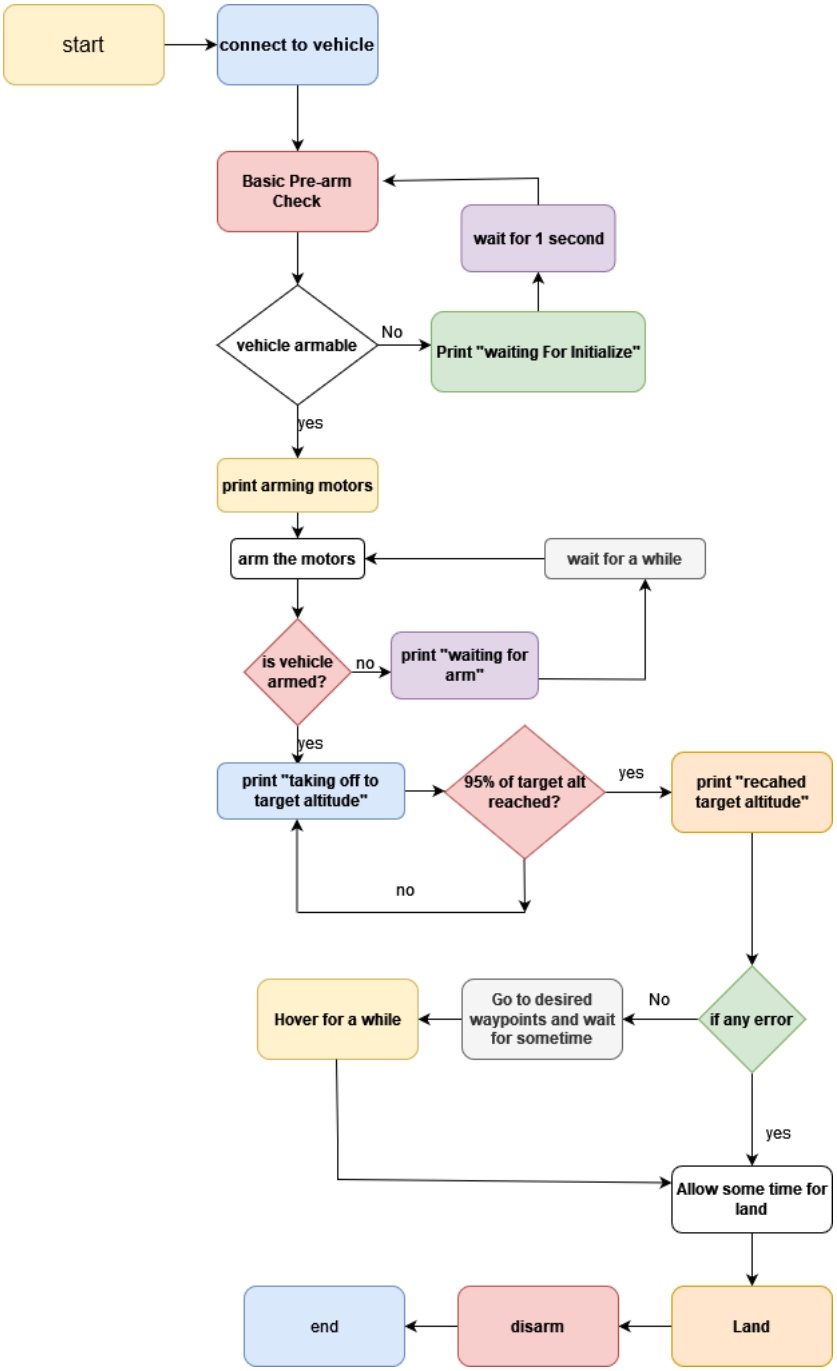

Figure 5.8: Algorithm for arbitrary path planning

Explanation:

The algorithm operates similarly to altitude control until the drone reaches the target altitude. Once this altitude is achieved, the drone proceeds to navigate to the specified waypoints. These waypoints, which can be predefined by the user, require inputting the desired latitude and longitude coordinates. The drone will then follow the path through all the designated waypoints.

Results:

Figure 5.9 is a simulation result for arbitrary path planning.

Figure 5.9: Simulation result for arbitrary path planning

```
drone1_log.txt
1   Basic pre-arm checks
2   Arming motors
3    Waiting for arming...
4   Taking off!
5    Altitude:  0.0
6    Altitude:  0.0
7    Altitude:  0.0
8    Altitude:  0.0
9    Altitude:  0.428
10   Altitude:  1.652
11   Altitude:  2.847
12   Altitude:  3.815
13  Reached target altitude
14  -35.3632622 149.1652375
15  -35.3834622 149.1763385
16  -35.3829622 149.1973375
17  -35.3936622 149.2075375
18  -35.4131622 149.2681375
19  Landing...
20  Disarming...
21
```

Figure 5.10: Simulation output Data

Table 5.3 shows the planned arbitrary waypoints and Table 5.4 shows the actual arbitrary waypoints

Table 5.3: Planned rbitrary Wapoints

Latitude	Longitude
26.1901444	91.6972991
26.1901556	91.6971508
26.1904888	91.697171
26.1904635	91.6972626
26.1901444	91.6972991

Table 5.4: Actual arbitrary waypoints

Point	Latitude	Longitude
1	26.190150	91.697282
2	26.190149	91.697271
3	26.190147	91.697261
4	26.190146	91.697250
5	26.190145	91.697240
6	26.190143	91.697229
7	26.190142	91.697219
8	26.190141	91.697208
9	26.190139	91.697198
10	26.190138	91.697187
11	26.190137	91.697177
12	26.190135	91.697166
13	26.190134	91.697156
14	26.190161	91.697156
15	26.190189	91.697156
16	26.190217	91.697156
17	26.190244	91.697156
18	26.190272	91.697156
19	26.190299	91.697156
20	26.190327	91.697156
21	26.190354	91.697156
22	26.190382	91.697156
23	26.190410	91.697156
24	26.190437	91.697156
25	26.190465	91.697156

Point	Latitude	Longitude
26	26.190465	91.697163
27	26.190464	91.697172
28	26.190463	91.697180
29	26.190462	91.697188
30	26.190462	91.697196
31	26.190461	91.697204
32	26.190461	91.697212
33	26.190460	91.697221
34	26.190459	91.697229
35	26.190458	91.697237
36	26.190458	91.697245
37	26.190457	91.697253
38	26.190435	91.697258
39	26.190413	91.697263
40	26.190391	91.697268
41	26.190369	91.697272
42	26.190346	91.697277
43	26.190324	91.697282
44	26.190302	91.697286
45	26.190280	91.697291
46	26.190258	91.697296
47	26.190236	91.697301
48	26.190213	91.697305
49	26.190191	91.697310

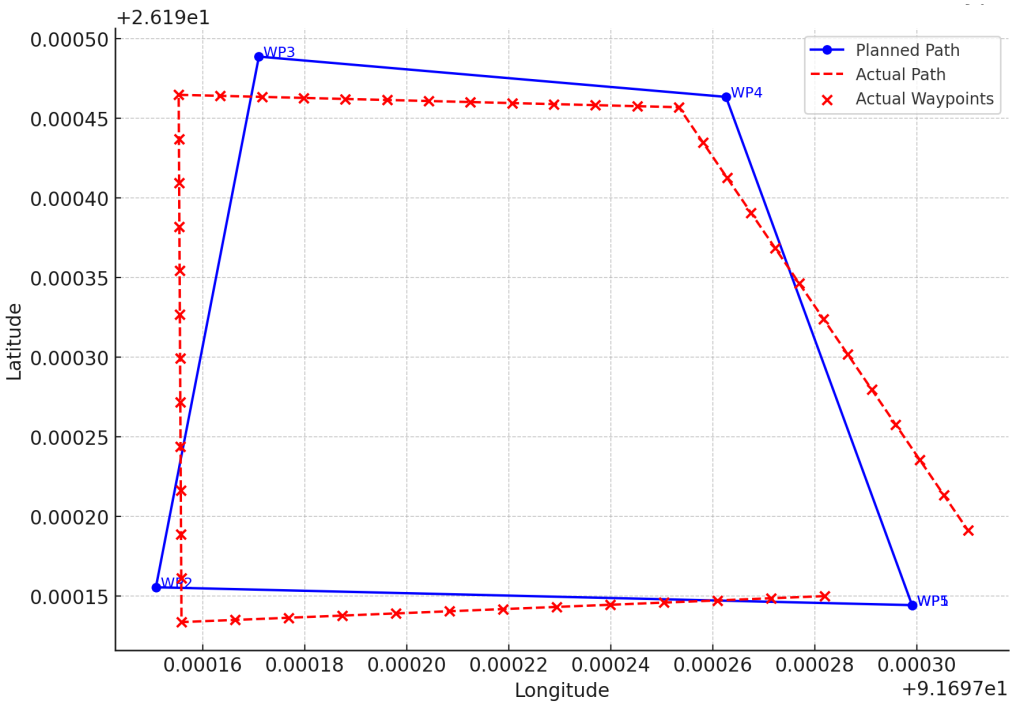

Figure 5.11: Planned vs actual Path

5.4 Path planning using NED coordinates

5.4.1 Introduction to NED coordinate system

The NED (North East Down) coordinate frame is used to express velocities and relative positions. The centre of the coordinate frame is the current position. From that origin, the north axis points to the true north and remains parallel with the landline of latitude at that point. The east axis is normal to the north axis and points tangent to the line of longitude at that point – The down axis points directly toward the centre of the Earth ↓.

When planning a path, using the (lat long alt) coordinate system results in substantial mistake. It generates inaccuracies in the range of one to three meters, making it inappropriate for local navigation. Particularly for local navigation, employing NED coordinates rather than latitude, longitude, and altitude can result in reduced error in some path planning calculations.

The North-East-Down (NED) coordinate system is a local Cartesian coordinate system used in local navigation and aerospace applications. The origin is typically set at a reference point, such as the takeoff location of a UAV. The axes are defined as follows:

- **North (x-axis)**: Points towards the local geographic north.
- **East (y-axis)**: Points towards the local geographic east.
- **Down (z-axis)**: Points downwards, perpendicular to the Earth's surface.

5.4.2 Coordinate Transformation

We convert to an Earth-Centered, Earth-Fixed (ECEF) coordinate system first, then to the local NED frame, in order to translate global coordinates (latitude, longitude, altitude) to NED coordinates.

Convert LLA to ECEF Coordinates

The ECEF coordinate system has its origin at Earth's centre. The transformation from LLA to ECEF is given by:

$$X = (N + h)\cos\phi\cos\lambda \tag{5.9}$$

$$Y = (N + h)\cos\phi\sin\lambda \tag{5.10}$$

$$Z = \left(\frac{b^2}{a^2}N + h\right)\sin\phi \tag{5.11}$$

where:

- ϕ is the latitude,
- λ is the longitude,
- h is the altitude,
- a is the semi-major axis of the Earth,
- b is the semi-minor axis of the Earth,
- $N = \frac{a^2}{\sqrt{a^2\cos^2\phi + b^2\sin^2\phi}}$ is the prime vertical radius of curvature.

Transform ECEF to NED Coordinates

We define a rotation matrix that aligns the ECEF coordinates with the local NED frame in order to convert from ECEF to NED. This rotation involves the latitude (ϕ_0) and longitude (λ_0) of the reference point (origin of NED frame):

$$R_{NED}^{ECEF} = \begin{bmatrix} -\sin\phi_0\cos\lambda_0 & -\sin\phi_0\sin\lambda_0 & \cos\phi_0 \\ -\sin\lambda_0 & \cos\lambda_0 & 0 \\ -\cos\phi_0\cos\lambda_0 & -\cos\phi_0\sin\lambda_0 & -\sin\phi_0 \end{bmatrix} \tag{5.12}$$

The position vector in the NED frame ($\mathbf{p}_{NED}$) can be obtained by:

$$\mathbf{p}_{NED} = R_{NED}^{ECEF}(\mathbf{p}_{ECEF} - \mathbf{p}_{ECEF_0}) \tag{5.13}$$

where:

- $\mathbf{p}_{ECEF}$ is the position vector in ECEF coordinates,
- $\mathbf{p}_{ECEF_0}$ is the ECEF position vector of the reference point.

5.4.3 Vector Operations in NED Coordinates

The Cartesian structure of the NED coordinate system makes vector operations simple. These are a few typical operations:

Position Vector

The position of a point P in the NED frame relative to the origin can be represented as:

$$\mathbf{P}_{NED} = \begin{bmatrix} x_{NED} \\ y_{NED} \\ z_{NED} \end{bmatrix} \tag{5.14}$$

where x_{NED}, y_{NED}, and z_{NED} are the north, east, and down coordinates, respectively.

Displacement Vector

The displacement vector between two points P_1 and P_2 in the NED frame is:

$$\mathbf{d}_{NED} = \mathbf{P}_{2_{NED}} - \mathbf{P}_{1_{NED}} = \begin{bmatrix} x_{2_{NED}} - x_{1_{NED}} \\ y_{2_{NED}} - y_{1_{NED}} \\ z_{2_{NED}} - z_{1_{NED}} \end{bmatrix} \tag{5.15}$$

Distance Calculation

The Euclidean distance between two points P_1 and P_2 in the NED frame is given by:

$$d = \sqrt{(x_{2_{NED}} - x_{1_{NED}})^2 + (y_{2_{NED}} - y_{1_{NED}})^2 + (z_{2_{NED}} - z_{1_{NED}})^2} \tag{5.16}$$

5.4.4 Type Mask in MAVLink Messages

Key to MAVLink communications is the type mask, which defines which fields in a message are ignored or enabled. Each bit in the bitmask corresponds to a distinct message field. Each bit determines whether a field is enabled (0) or ignored (1). This form of mask is commonly used during Path Planning with NED coordinates.

Structure of the Type Mask

In MAVLink, a type mask is an integer of 16 bits. A distinct field in the message is represented by each bit in this number. The type mask's detailed description is provided below:

- **Bit 0**: Position x (North) - 0 (enabled) / 1 (ignored)
- **Bit 1**: Position y (East) - 0 (enabled) / 1 (ignored)

- **Bit 2**: Position z (Down) - 0 (enabled) / 1 (ignored)
- **Bit 3**: Velocity x (North) - 0 (enabled) / 1 (ignored)
- **Bit 4**: Velocity y (East) - 0 (enabled) / 1 (ignored)
- **Bit 5**: Velocity z (Down) - 0 (enabled) / 1 (ignored)
- **Bit 6**: Acceleration x (North) - 0 (enabled) / 1 (ignored)
- **Bit 7**: Acceleration y (East) - 0 (enabled) / 1 (ignored)
- **Bit 8**: Acceleration z (Down) - 0 (enabled) / 1 (ignored)
- **Bit 9**: Force x (North) - 0 (enabled) / 1 (ignored)
- **Bit 10**: Force y (East) - 0 (enabled) / 1 (ignored)
- **Bit 11**: Force z (Down) - 0 (enabled) / 1 (ignored)
- **Bit 12**: Yaw - 0 (enabled) / 1 (ignored)
- **Bit 13**: Yaw rate - 0 (enabled) / 1 (ignored)
- **Bit 14-15**: Reserved

Example Type Mask

Consider the type mask `0b0000111111111000`, which enables only the position fields (north, east, down) and ignores all other fields. Here's how each bit is set:

- **Bit 0 (Position x)**: 0 (enabled)
- **Bit 1 (Position y)**: 0 (enabled)
- **Bit 2 (Position z)**: 0 (enabled)
- **Bit 3 (Velocity x)**: 1 (ignored)
- **Bit 4 (Velocity y)**: 1 (ignored)
- **Bit 5 (Velocity z)**: 1 (ignored)
- **Bit 6 (Acceleration x)**: 1 (ignored)
- **Bit 7 (Acceleration y)**: 1 (ignored)
- **Bit 8 (Acceleration z)**: 1 (ignored)
- **Bit 9 (Force x)**: 1 (ignored)
- **Bit 10 (Force y)**: 1 (ignored)

- **Bit 11 (Force z)**: 1 (ignored)
- **Bit 12 (Yaw)**: 1 (ignored)
- **Bit 13 (Yaw rate)**: 1 (ignored)
- **Bit 14-15**: Reserved

Setting the Type Mask

We utilize a binary integer to set the type mask, with each bit representing either 0 or 1, depending on whether we wish to enable or ignore the corresponding field. A generic form for configuring the type mask is as follows:

$$\text{type\_mask} = \text{Bit } 0 \times 2^0 + \text{Bit } 1 \times 2^1 + \text{Bit } 2 \times 2^2 + \cdots + \text{Bit } 13 \times 2^{13} \quad (5.17)$$

For example, if we want to enable only the position fields and ignore all others, we set bits 0, 1, and 2 to 0 and all other bits to 1:

$$\text{type\_mask} = 0\times2^0+0\times2^1+0\times2^2+1\times2^3+1\times2^4+\cdots+1\times2^{13} = 0b0000111111111000 \quad (5.18)$$

5.4.5 Example

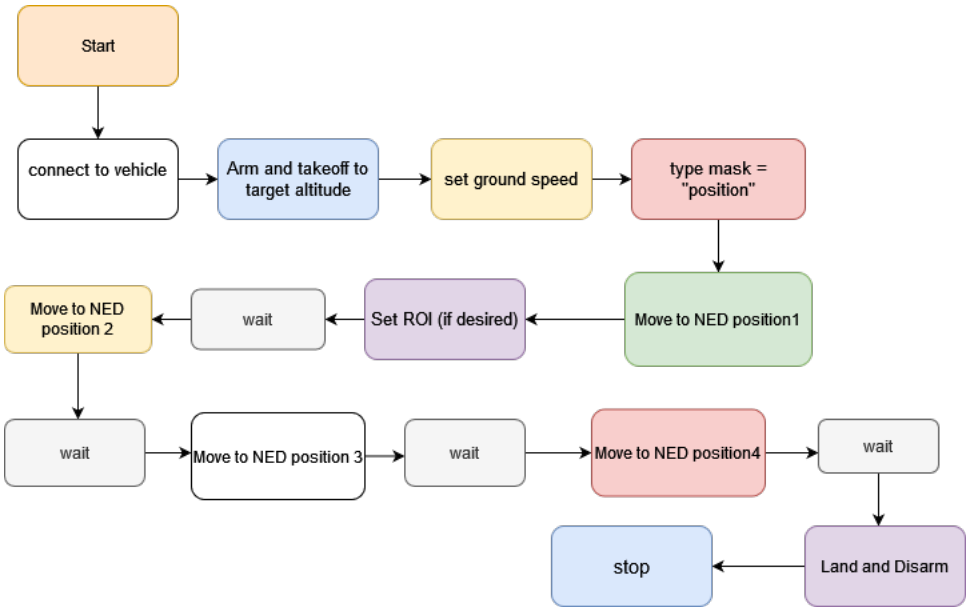

Figure 5.12: Simple Path Planning algo using NED cordinates

Result

Various path planning with NED coordinates was tested in simulation, but due to the jerk movement of the drone, it was not possible to test it in real hardware.

Figure 5.13 can be referred to for simulation for path planning using NED coordinate system.

Figure 5.13: Path Planning using NED coordinates

```
 Waiting for arming...
Taking off!
Set groundspeed to 5m/s.
going
SQUARE path using SET_POSITION_TARGET_LOCAL_NED and position parameters
North 50m, East 0m, 10m altitude for 8 seconds
Point ROI at current location (home position)
North 50m, East 50m, 10m altitude
Point ROI at current location
North 0m, East 50m, 10m altitude
North 0m, East 0m, 10m altitude
Disarming...
```

Figure 5.14: NED path planning data

5.5 Challenges and Errors

Major challenges faced during the hardware test were:

• Path planning algorithms need to be efficient to run in real-time on the drone's limited computational resources. The algorithm needs to be tested on simulation to be sure that the algorithm works properly without any error. If the algo is directly tested on hardware there is a chance of hardware getting damaged.

• Because the Pixhawk and Raspberry Pi are communicating via the ACM 0 port, there was a communication lag of approximately 10 seconds between the command being provided and the execution of the command

• Limited communication range can affect the drone's ability to receive updates or commands.

• Drone flight periods and distances are constrained by their short battery life. For the drone to accomplish its objective and return safely, energy consumption must be taken into consideration during the path planning process.
• Considerable inaccuracy was seen when path planning was done using the (lat, long, alt) coordinate system.
• When the simulation was done for path planning using the NED coordinate system, the drone was not controllable even after setting the ground speed. the drone was moving with a jerk from one point to another, due to which real-world implementation of algo was not possible.

Chapter 6

Collision Avoidance

Collision avoidance is a critical component in the safe operation of UAVs, particularly in scenarios where multiple drones operate nearby. Effective collision avoidance systems are essential to ensure that UAVs can navigate dynamically and avoid obstacles, thereby preventing accidents and ensuring smooth operation.

This chapter focuses on the development and implementation of a collision avoidance algorithm for UAVs, specifically designed for scenarios where two drones move in a linear path—one slower and one faster. The faster drone must be capable of detecting the slower drone ahead and executing manoeuvres to avoid a collision.

The primary objective of this study is to create a robust collision avoidance system that leverages advanced algorithms and real-time data from sensors. By integrating technologies such as MAVLink for communication and DroneKit for control, I aim to demonstrate a practical solution for UAV collision avoidance.

6.1 Collision Avoidance Algorithm for Drones

Here is the basic structure for the collision avoidance algorithm:

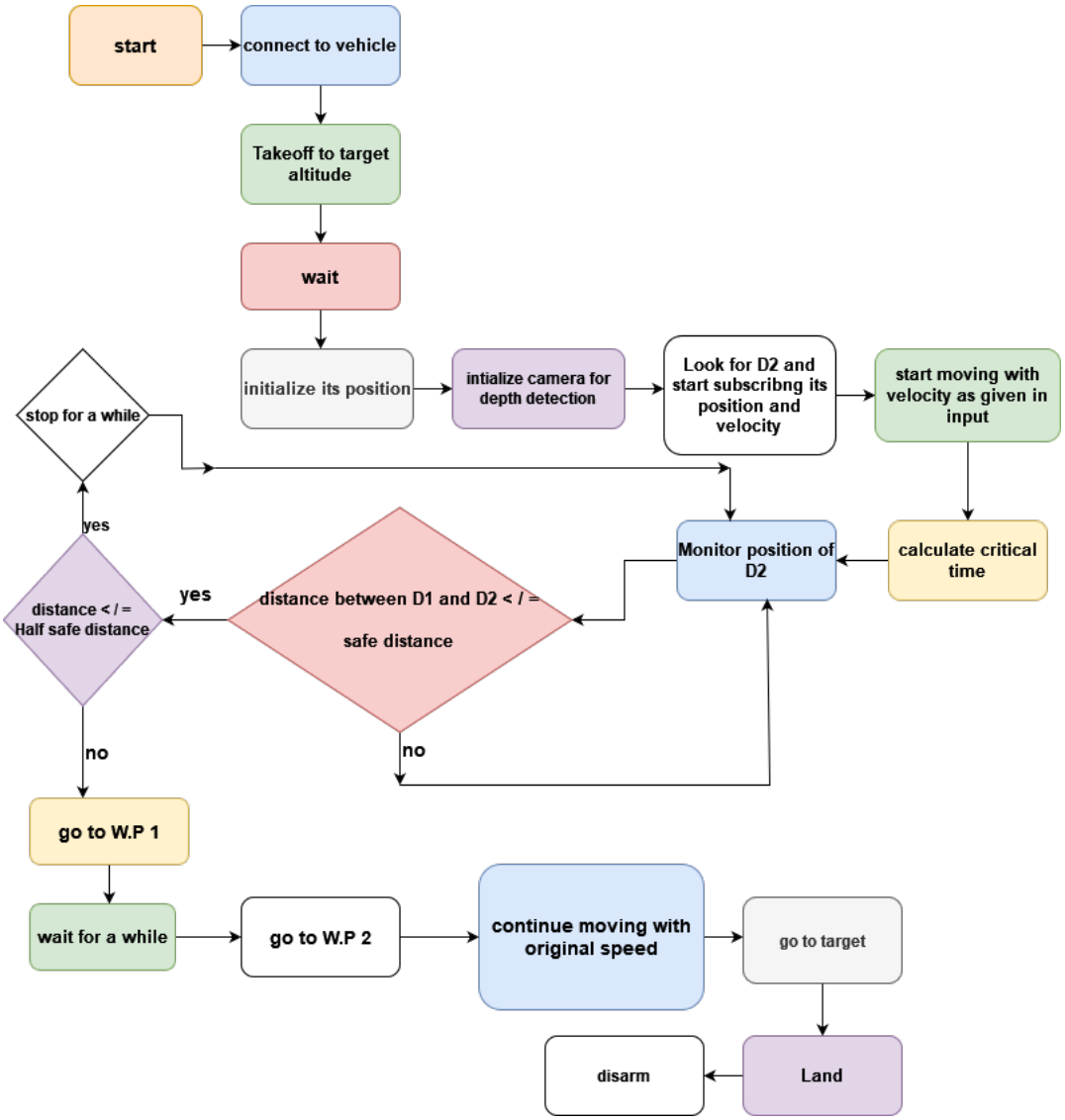

Figure 6.1: Collision Avoidance algorithm

Initialization

Let D_1 and D_2 represent the slower and faster drones, respectively.

- **Position Initialization:**
  - Let $\mathbf{p}_1(t)$ and $\mathbf{p}_2(t)$ denote the positions of D_1 and D_2 at time t.
  - Let $\mathbf{v}_1(t)$ and $\mathbf{v}_2(t)$ denote the velocities of D_1 and D_2 at time t.

Inputs

- $v_1(t)$: velocity of D_1
- $v_2(t)$: velocity of D_2
- K_s: safety factor

- $v_{1\text{new}}$: overtaking velocity
- s: safe distance at the end of the overtaking algorithm
- d_{overtake}

Safe Distance Calculation

Define the safe distance d_s and half the safe distance d_{hs}. The critical distance conditions are given by:

$$d_s = k_s \cdot v_2 \tag{6.1}$$

$$d_{hs} = \frac{1}{2} \cdot d_s \tag{6.2}$$

where k_s is a safety factor.

Monitoring and Calculation

- **Distance Between Drones:**

$$d(t) = \|\mathbf{p}_2(t) - \mathbf{p}_1(t)\| \tag{6.3}$$

- **Relative Velocity:**

$$\mathbf{v}_{rel}(t) = \mathbf{v}_2(t) - \mathbf{v}_1(t) \tag{6.4}$$

- **Critical Time:**

$$t_c = \frac{d(t) - d_s}{\|\mathbf{v}_{rel}(t)\|} \tag{6.5}$$

Descision Making

- **If** $d(t) \leq d_s$**:**
  - **If** $d(t) \leq d_{hs}$**:**
    - Stop D_1 for a while:

$$\mathbf{v}_1(t) = (0, 0, 0)$$

    - After waiting for a predefined duration t_{wait}, D_1 resumes monitoring the position of D_2.
  - **Else:**
    - Increase speed of D_1 to $v_{1,new}$ (horizontal component: $v_{1x,new}$, vertical component: $v_{1y,new}$), and to go calculated waypoints.

Post Maneure Action

After completing the overtaking algorithm the faster drone will resume to its original speed and move to its target location.

Steps for waypoints calculation for faster drone

- Calculation for waypoint 1:
  - **Time to reach Waypoint 1 (t_{overtake}):**

$$t_{\text{overtake}} = \frac{d_{\text{overtake}}}{v_{1y,\text{new}}} \tag{6.6}$$

$$v_{1_y\text{new}} = \sqrt{(v_{1\text{new}})^2 - (v_2(t))^2} \tag{6.7}$$

$$\text{as,} \quad v_1(t) = v_{1_x\text{new}} \tag{6.8}$$

  - **Position of D_2 after t_{overtake}:**

$$\mathbf{p}_2(t_c + t_{\text{overtake}}) = (x_2(t_c) + v_2 \cdot t_{\text{overtake}}, y_2(t_c), z_2(t_c)) \tag{6.9}$$

  - **Position of D_1 after t_{overtake}:**

$$\mathbf{p}_1(t_c + t_{\text{overtake}}) = (x_1(t_c) + v_{1x,\text{new}} \cdot t_{\text{overtake}}, d_{\text{overtake}}, z_1(t_c)) \tag{6.10}$$

- Calculation for waypoint 2:

$$\frac{x_{1,\text{final}} - x_{1,\text{overtake}}}{v_{1x,\text{new}}} = \frac{x_{1,\text{final}} - x_{2,\text{overtake}} - s}{v_2(t)} \tag{6.11}$$

  where d is the safe distance between D1 and D2 after the completion of overtaking maneure of D1

  we can calculate from the above equation, such that the final position of D_1 will be $(x_{1,\text{final}}, y_1(t_c), z_1(t_c))$ and the final position of D_2 will be $(x_{1,\text{final}} - s, y_2(t_c), z_1(t_c))$.

For monitoring the position of the slower drone we need some kind of sensor, for my project intel-real sense camera is utilized, as it has the inbuilt property of giving the depth information, but the algorithm cannot run well with only depth sensing mode, using RGB mode with depth sensing can be a better option as it enhances the algorithm implementation.

6.2 Results

A 3-D simulation was done considering point masses of drones which shows how the algorithm will work. The animation was created considering the point mass for the drones.

Simulation Parameters

For my simulation, I considered:

- Initial position of slow drone: $[5, 0, z]$
- Initial position of fast drone: $[0, 0, z]$
- Initial speed of slow drone: $1\,\text{m/s}$
- Initial speed of fast drone: $2\,\text{m/s}$
- Overtaking velocity of fast drone: $3\,\text{m/s}$
- Safety factor, $K = 4$
- Safe distance after overtake: $5\,\text{m}$
- Overtake distance: $5.8\,\text{m}$

Calculation

Calculating critical time, t_c:

$$t_c = \frac{5-4}{2-1} = 1\,\text{sec}$$

The position of fast drone after 1 second will be:

$$(2, 0, z)$$

The position of slow drone after 1 second will be:

$$(6, 0, z)$$

Waypoint 1 Calculation

Now the overtaking algorithm will start.

- Overtaking velocity: $3\,\text{m/s}$
- Horizontal velocity of fast drone: $2\,\text{m/s}$
- Vertical velocity: $\sqrt{(3^2) - (2^2)} = 2.236\,\text{m/s}$

Now calculating time to reach waypoint 1:

$$t_{\text{overtake}} = \frac{\text{overtake distance}}{2.236}$$

$$\frac{5.8\,\text{m}}{2.236\,\text{m/s}} = 2.593\,\text{sec}$$

Horizontal Displacement Calculation

The horizontal displacement of fast drone after 2.593 seconds will be:

$$2 + (2.593 \times 2) = 7.186$$

The horizontal displacement of slow drone after 2.593 seconds will be:

$$6 + 2.593 = 8.593$$

Therefore, the position of the fast drone after 2.593 seconds will be:

$$(7.187, 5.8, z)$$

The position of the slow drone after 2.593 seconds will be:

$$(8.593, 0, z)$$

Calculation of Waypoint 2

Let x_f be the final position of the fast drone, and the safe distance, d_s after the overtaking algorithm ends be 5 m.

$$\frac{x_f - 7.187}{2} = \frac{x_f - 8.593 - 5}{1}$$

$$x_f - 7.187 = 2x_f - 27.186$$

$$-x_f = -27.186 + 7.187$$

$$x_f = 20.001$$

Therefore, the final position of the fast drone is:

$$(20.001, 0, z)$$

The final position of the slow drone is:

$$(15.001, 0, z)$$

We can see from Figure 6.2 that when the overtaking algorithm ends the position of faster drone is 20m and the position of the slower drone is 15m from the starting point.

Figure 6.4 shows that after the overtaking algorithm starts the speed of the faster drone becomes 3m/s till it reaches its final position.

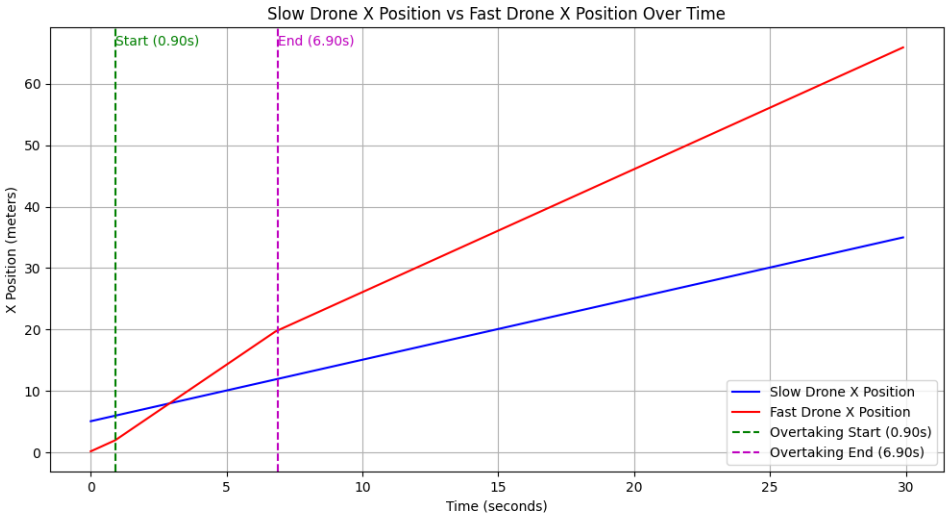

Figure 6.2

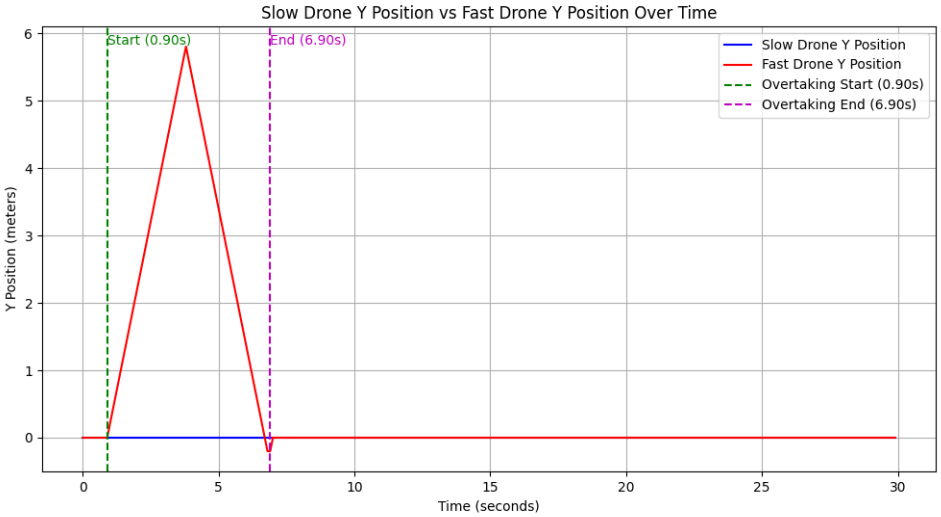

Figure 6.3

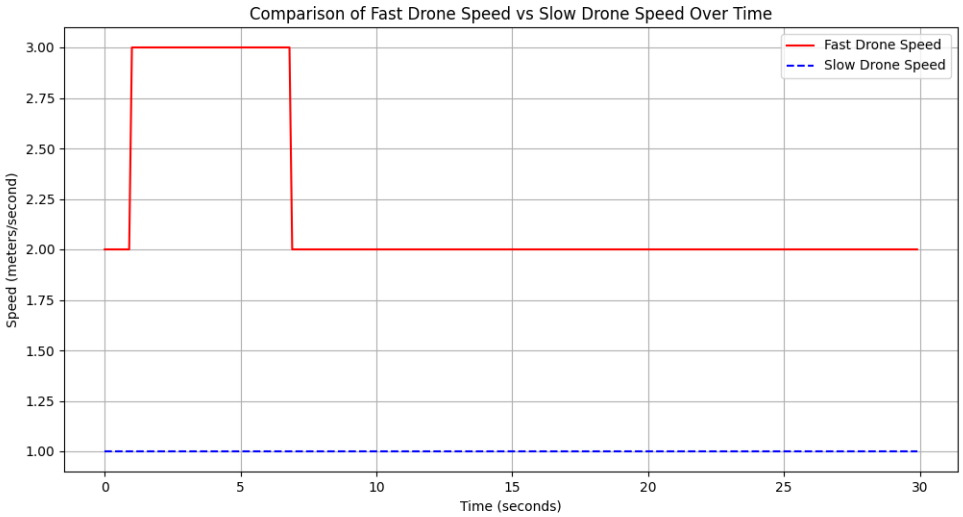

Figure 6.4

6.3 Challenges

Intel real sense camera was not compatible with our microcontroller which is Raspberry, as I was not able to work with RGB mode. Working only with depth mode decreases the efficiency of the algorithm. I also need to design some type of controller so that the drones go to desired waypoints. The algorithm needs to be tested in Gazebo Simulation for measuring the efficiency of the algorithm.

Chapter 7

Conclusion and Future work

Collision avoidance for UAVs is an enormous area of research interest. This report presents a comprehensive analysis of the hardware calibration of a quadcopter using mission planner software. There are many other platforms for hardware calibration of quadcopters, but calibration can quickly be done in a mission planner. I successfully calibrated the drone by adjusting the P, I, and D values for each axis, i.e., roll, pitch, and yaw. After the Raspberry Pi was successfully integrated with Pixhawk, I was able to perform path planning for arbitrary paths and circular paths in simulation and also in hardware. Also, I was able to perform path planning using NED coordinates in the Gazebo simulation, but testing in the real hardware was not done.

The collision avoidance algorithm was created but not tested in simulation and also in hardware. Therefore the next task will be testing the collision avoidance algorithm in simulation and also in the hardware to test the efficacy.

References

[1] S. Mastellone, D. M. Stipanović, C. R. Graunke, K. A. Intlekofer, and M. W. Spong, "Formation control and collision avoidance for multi-agent non-holonomic systems: Theory and experiments," *The International Journal of Robotics Research*, vol. 27, no. 1, pp. 107–126, 2008.

[2] S. Aggarwal and N. Kumar, "Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges," *Computer communications*, vol. 149, pp. 270–299, 2020.

[3] Y. Watanabe, A. Calise, and E. Johnson, "Vision-based obstacle avoidance for uavs," in *AIAA guidance, navigation and control conference and exhibit*, p. 6829, 2007.

[4] N. Bao, X. Ran, Z. Wu, Y. Xue, and K. Wang, "Research on attitude controller of quadcopter based on cascade pid control algorithm," in *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 1493–1497, IEEE, 2017.

[5] P. Wang, Z. Man, Z. Cao, J. Zheng, and Y. Zhao, "Dynamics modelling and linear control of quadcopter," in *2016 International Conference on Advanced Mechatronic Systems (ICAMechS)*, pp. 498–503, IEEE, 2016.

[6] F. R. Rochin, H. Yamazoe, and J.-H. Lee, "Autonomous coverage path planning and navigation control system for search operations using a uav," in *2019 16th International Conference on Ubiquitous Robots (UR)*, pp. 194–199, IEEE, 2019.

[7] C. Bernardeschi, A. Fagiolini, M. Palmieri, G. Scrima, and F. Sofia, "Ros/gazebo based simulation of co-operative uavs," in *Modelling and Simulation for Autonomous Systems: 5th International Conference, MESAS 2018, Prague, Czech Republic, October 17–19, 2018, Revised Selected papers 5*, pp. 321–334, Springer, 2019.

[8] J.-W. Park, H.-D. Oh, and M.-J. Tahk, "Uav collision avoidance based on geometric approach," in *2008 SICE Annual Conference*, pp. 2122–2126, IEEE, 2008.

[9] L. Sha, "Promoting the development of intelligent community management system: a raspberry pi-based handheld 3d scanner," in *International Conference on Image, Signal Processing, and Pattern Recognition (ISPP 2023)*, vol. 12707, pp. 1227–1232, SPIE, 2023.

[10] Z. Chen, J. Samarabandu, and R. Rodrigo, "Recent advances in simultaneous localization and map-building using computer vision," *Advanced Robotics*, vol. 21, no. 3-4, pp. 233–265, 2007.