ABSTRACT

Drones commonly referred, as UAVs are mostly associated with military, industry and other specialised operations but with recent developments in the area of sensors and Information Technology in the last two decades the scope of drones has been widened to other areas like Agriculture. The drones manufactured these days are becoming smarter by integrating open-source technology, smart sensors, better integration, and more flight time, tracking down criminals, detecting forest and other disaster areas. In this abstract, we demonstrate the design and development of an aircraft type autonomous drone suitable for pesticides spraying in agriculture. Nowadays, autonomous drones fly by with the support of external software like Agri-assistant, Q-controller etc. instead of using external software, we are going to make real time mapping, constrained path for the movements. It is an AI powered drone, which flies in a fixed path and has an obstacle sensing system. The drone can fly under different altitudes varying from 10 to 50 meters. The agricultural drone has a return to home feature for autonomous return to the starting position, when its battery voltage reduces down to a certain level or any communication failure is detected.

LIST OF FIGURES

Figure 1.1. Drone System………………..……..……......3
Figure 1.2 Raspberry Pi drone……………………………4
Figure 1.3. Raspberry Pi Module……..……………………6
Figure 1.4. Pixhawk Flight Controller……………………7
Figure 1.5. Telemetry……………………………………..8
Figure 1.6. Drone Programming System…………………9
Figure 2.1 SITL Firmware……………………………….12
Figure 2.2 Mission Planner………………………………14
Figure 2.3 Ground Station……………………………….14
Figure 3.1 DJI Tello Drone……………………………...19
Figure 3.2 Open CV……………………………………..24
Figure 5.1 Object Tracking………………………………38
Figure 5.2. Mapping……………………………………..39
Figure 5.3. Line Follower………………………………43

CHAPTER 1
DRONE SYSTEM

1.Drone System

Drone system are classified into three types;

- Drone or UAV
- Ground Control System
- Communication System

1.1 UAV System

A drone is an unmanned aerial vehicle (UAV), which is an aircraft that does not have a human pilot, crew, or passengers. Unmanned aerial vehicles (UAVs) are a subset of unmanned aircraft systems (UAS), which also include a ground controller and a communications network with the UAV. UAV flight can be remotely controlled by a human pilot, or it can have varying degrees of autonomy, such as autopilot assistance, all the way up to fully autonomous aircraft that do not allow for human intervention.

1.1.1 Ground Control System

A ground control station (GCS) for an unmanned aerial vehicle (UAV) is a land- or sea-based control centre that provides the infrastructure for human control (UAVs or "drones"). However, this is more commonly known as a Mission Control Center. It could also refer to a system for controlling rockets within or above the atmosphere.

1.1.2 Communication System

Unmanned aerial vehicles (UAVs) and drones use a variety of communication methods to establish data links with ground control stations (GCS) and, in some cases, multiple aircraft (swarm technology). In addition to broadcasting telemetry data, UAV communication systems may be used to send video and data from sensors and payloads back to the control station.

1.1.3 Technical Illustration of Drone System

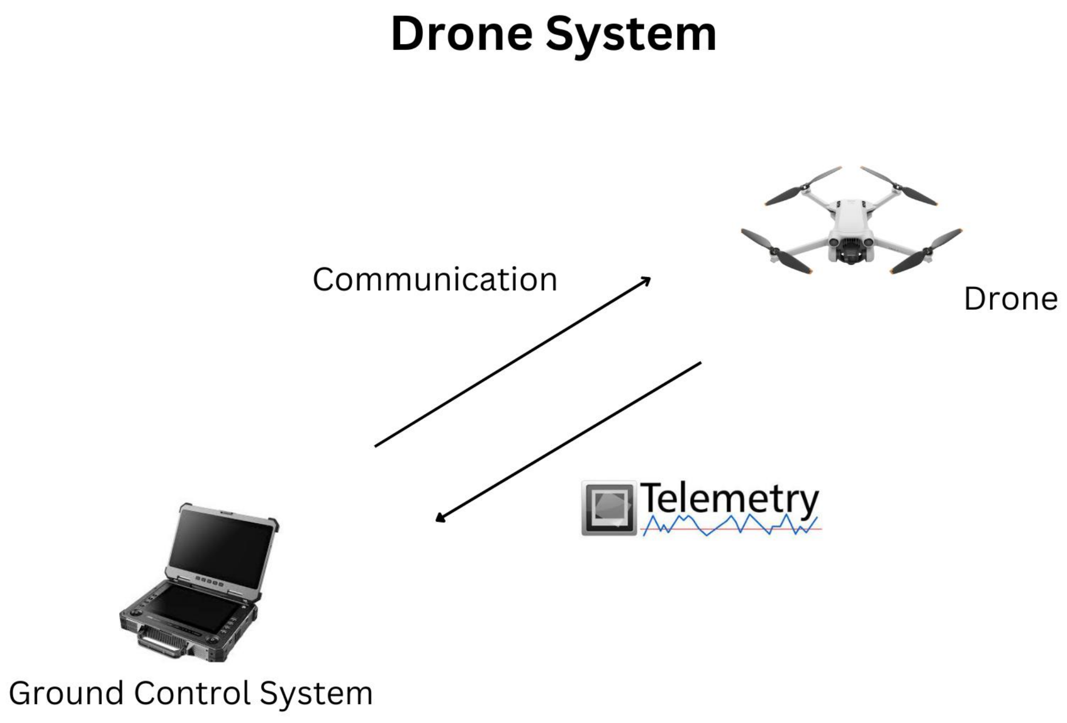


Figure 1.1. Drone System

1.2 Drone Programming (Python):

As the drone industry evolves from simple RC robots in the sky to actual industries, one thing is certain: drone programming will be crucial in fulfilling the realisation of truly autonomous drones. Think about it, whether we're talking about drone delivery or surveillance drones, human pilots will not suffice. The program for drones that we will use is Python programming language. Python is a programming language that lets you work quickly and integrate systems more effectively.

The application of mathematical and physical laws allows us to fly drones, but much accuracy is required for stabilised hovering. We are developing obstacle avoidance and keyboard control features in UAVs using various platforms such as ROS, Linux, Gazebo (modelling or simulating), Python, and C++.

1.2.1 Raspberry Pi Drone

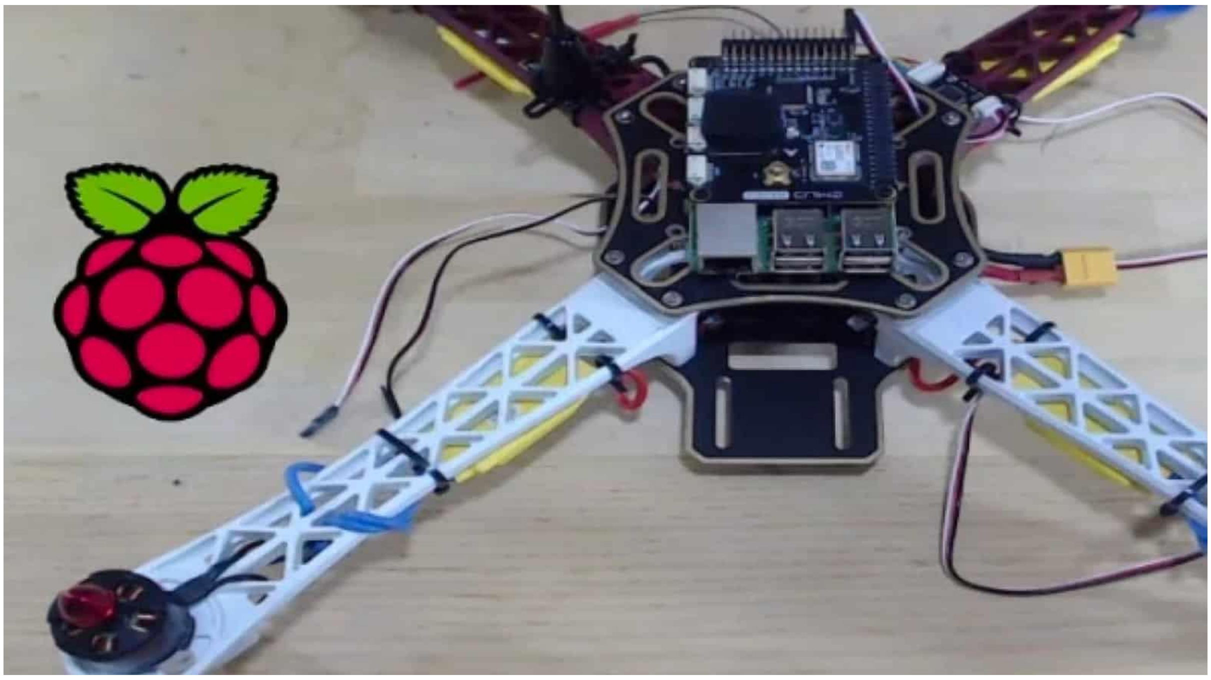


Figure 1.2 Raspberry Pi drone

This drone, built with a Raspberry Pi 2 Model B paired with the Multiwii flight controller, is capable of recording video to a USB drive aboard the drone. The Styrofoam frame shows the versatility present when building your own drone.

Components

- Raspberry Pi 4B
- Pixhawk Flight Controller
- Drone Frame
- Four Brushless Motors
- Four ESCs
- RC Transmitter and Receiver
- 3D Printed Vibration Dampening Plate
- SD Card and SD Card Reader
- Mounting materials

- Lipo Battery
- Lipo Battery Charger
- Lipo Battery Fire-proof case
- Eight Propellers
- GP S and Compass Module
- GPS mount
- Telemetry Modules
- Screws, screwdrivers, hex keys

1.2.2 Raspberry Pi 4B

➔ The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It’s capable of doing everything you’d expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

➔ What’s more, the Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting bird houses with infra-red cameras. We want to see the Raspberry Pi being used by kids all over the world to learn to program and understand how computers work.

➔ The Raspberry Pi 4 Model B is the latest version of the low-cost Raspberry Pi computer. The Pi isn't like your typical device; in its cheapest form it doesn't have a case, and is simply a credit-card sized electronic board of the type you might find inside a PC or laptop, but much smaller.

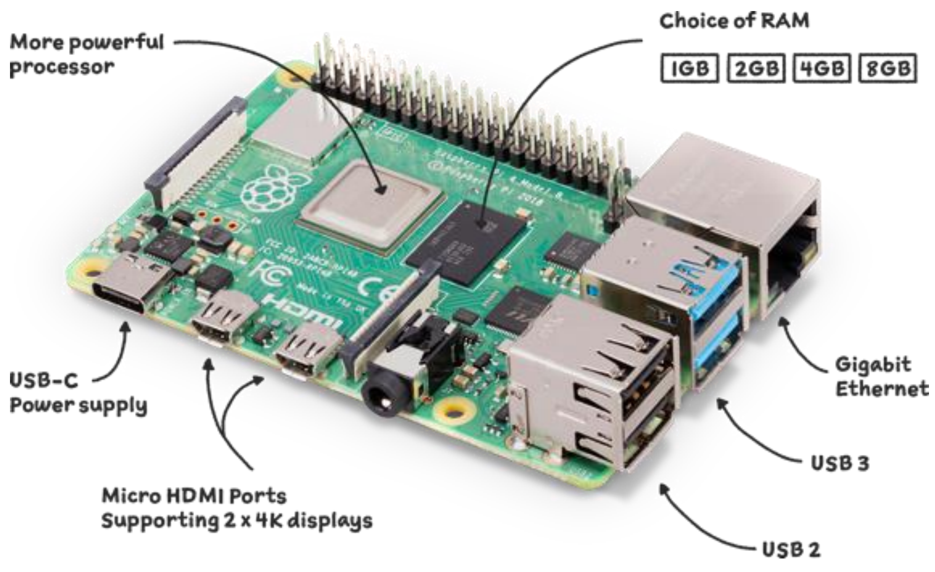


Figure 1.3. Raspberry Pi Module

1.2.3 Pixhawk Flight Controller

- Pixhawk 4 is an advanced autopilot designed and made in collaboration with Holybro and the PX4 team. It is optimised to run PX4 v1.7 and later, and is suitable for academic and commercial developers.
- The Pixhawk is a very popular flight controller that has been around for quite some time. What is cool about this is its open source hardware, meaning the schematics are free for anyone.
- Due to its open source nature, it has turned into an extremely dependable board. Just two decades ago, flight controllers and the accompanying firmware would cost thousands to hundreds of thousands of dollars.

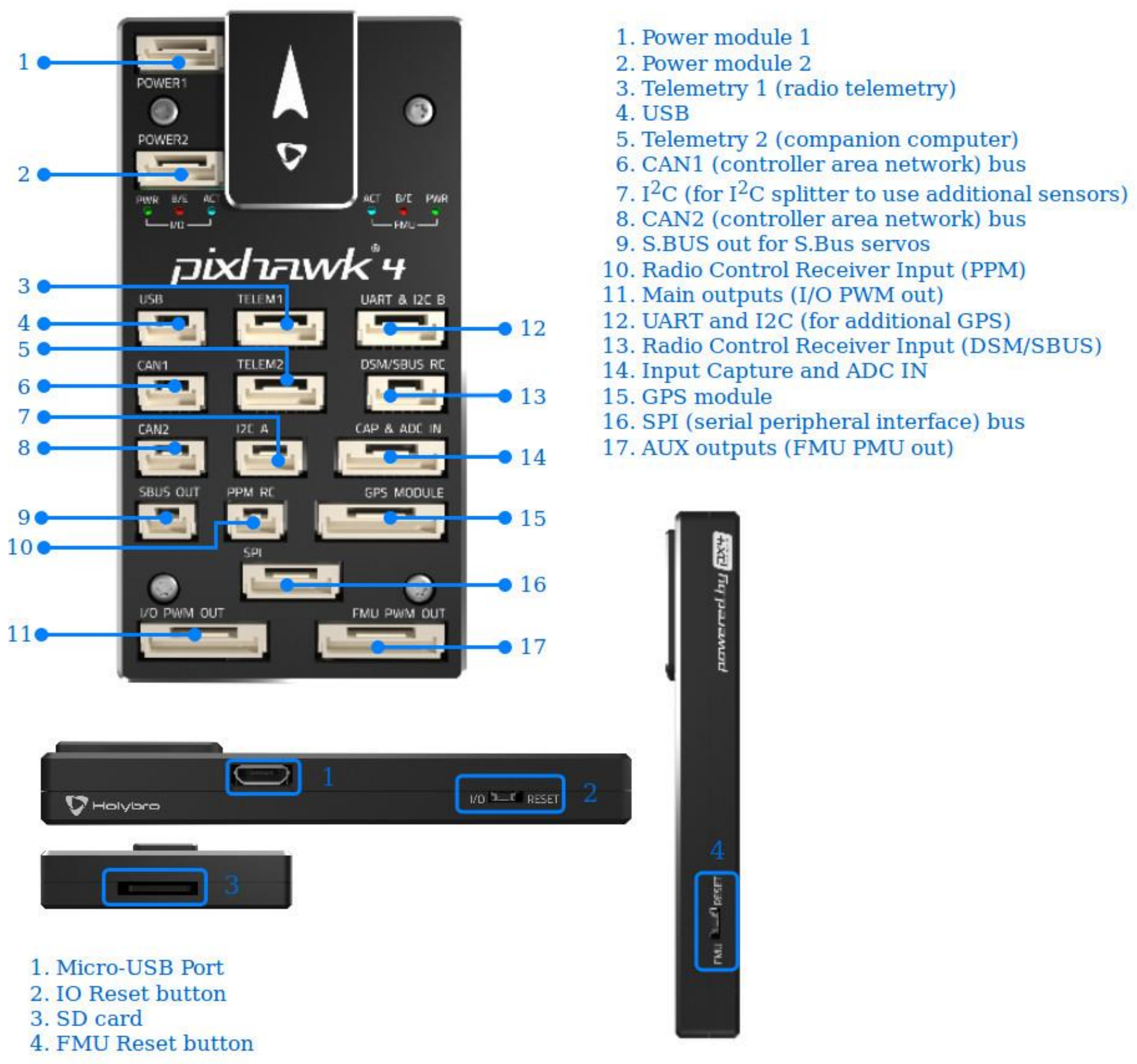

1. Power module 1
2. Power module 2
3. Telemetry 1 (radio telemetry)
4. USB
5. Telemetry 2 (companion computer)
6. CAN1 (controller area network) bus
7. I2C (for I2C splitter to use additional sensors)
8. CAN2 (controller area network) bus
9. S.BUS out for S.Bus servos
10. Radio Control Receiver Input (PPM)
11. Main outputs (I/O PWM out)
12. UART and I2C (for additional GPS)
13. Radio Control Receiver Input (DSM/SBUS)
14. Input Capture and ADC IN
15. GPS module
16. SPI (serial peripheral interface) bus
17. AUX outputs (FMU PMU out)
POWER1
POWER2
pixhawk 4
USB
TELEM1
UART & I2C B
CAN1
TELEM2
DSM/SBUS RC
CAN2
I2C A
CAP & ADC IN
SBUS OUT
PPM RC
GPS MODULE
SPI
I/O PWM OUT
FMU PWM OUT
Holybro
I/O
RESET
powered by PX4
FMU
1. Micro-USB Port
2. IO Reset button
3. SD card
4. FMU Reset button

Figure 1.4. Pixhawk Flight Controller

1.2.4 Telemetry

Telemetry is the automatic recording and transmission of data from remote or inaccessible sources to an IT system in a different location for monitoring and analysis. Telemetry data may be relayed using radio, infrared, ultrasonic, GSM, satellite or cable, depending on the application (telemetry is not only used in software development, but also in meteorology, intelligence, medicine, and other fields). In the software development world, telemetry can offer insights on which features end users use most, detection of bugs and issues, and offering better visibility into performance without the need to solicit feedback directly from users.

1.2.5 How Telemetry Works

In a general sense, telemetry works through sensors at the remote source which measures physical (such as precipitation, pressure or temperature) or electrical (such as current or voltage) data. This is converted to electrical voltages that are combined with timing data. They form a data stream that is transmitted over a wireless medium, wired or a combination of both. At the remote receiver, the stream is disaggregated and the original data displayed or processed based on the user's specifications.

In the context of software development, the concept of telemetry is often confused with logging. But logging is a tool used in the development process to diagnose errors and code flows, and it's focused on the internal structure of a website, app, or another development project. Once a project is released, however, telemetry is what you're looking for to enable automatic collection of data from real-world use. Telemetry is what makes it possible to collect all that raw data that becomes valuable, actionable analytics.

Figure 1.5. Telemetry

1.2.6 OpenCV Kit

OAK—D is a spatial AI powerhouse, capable of simultaneously running advanced neural networks while providing depth from two stereo cameras and colour information from a single 4K camera in the centre. All OAK—D hardware pledges come with a high-quality 1 metre USB-C cable, a 5V power supply.

1.3 Firmware Based Drone Programming

The core element in the software flight stack you will be learning about is the ArduPilot firmware. The great thing about ArduPilot, is it is very widely supported by a TON of flight control boards on the market, including any Pixhawk or Cube based drone.

1.3.1 Technical Illustration of Drone System using Python programming:

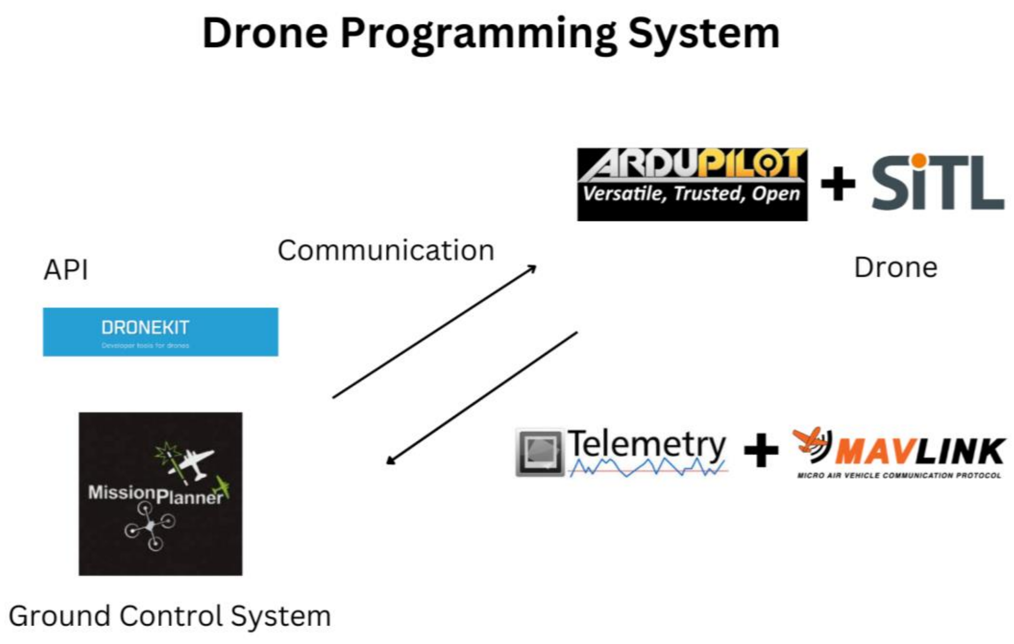


Figure 1.6. Drone Programming System

CHAPTER 2

FIRMWARE

2.1 UAV System using Firmware:

2.1.1 Dronekit

DroneKit-Python allows developers to create apps that run on an onboard companion computer and communicate with the ArduPilot flight controller using a low-latency link.DroneKit-Python (formerly DroneAPI-Python) contains the python language implementation of DroneKit.

The API allows developers to create Python apps that communicate with vehicles over MAVLink. It provides programmatic access to a connected vehicle's telemetry, state and parameter information, and enables both mission management and direct control over vehicle movement and operations.

The API is primarily intended for use in onboard companion computers (to support advanced use cases including computer vision, path planning, 3D modelling etc). It can also be used for ground station apps, communicating with vehicles over a higher latency RF-link.

2.1.2 Ardupilot:

ArduPilot is a trusted, versatile, and open source autopilot system supporting many vehicle types: multi-copters, traditional helicopters, fixed wing aircraft, boats, submarines, rovers and more. The source code is developed by a large community of professionals and enthusiasts. New developers are always welcome! The best way to start is by joining the Developer Team Forum, which is open to all and chock-full of daily development goodness.

ARDUPILOT
Versatile, Trusted, Open

PURPOSE OF FIRMWARE

Firmware is software that contains basic machine instructions that enable hardware to function and communicate with other software on a device. Firmware controls the hardware of a device at a low level.

2.1.3 SITL Firmware

You choose the firmware to match your vehicle and mission: Copter, Plane, Rover, Sub, or Antenna Tracker.

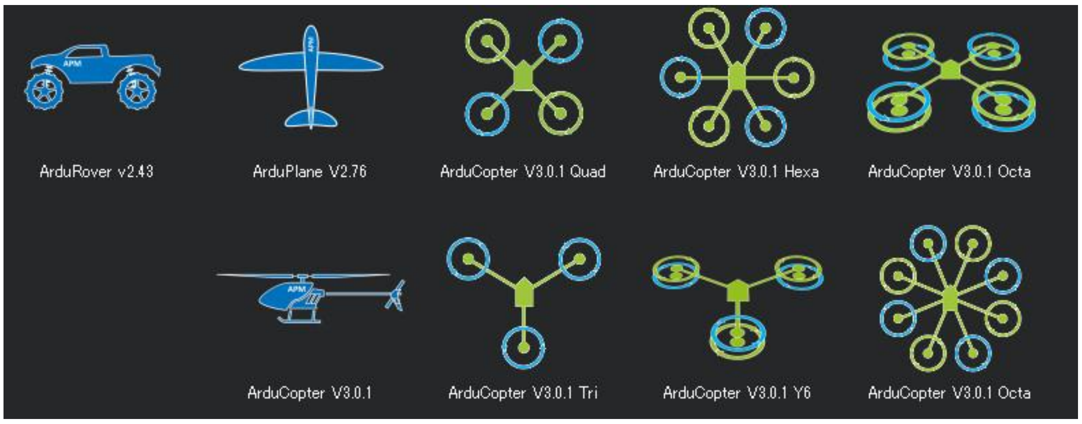


Figure 2.1 SITL Firmware

The ArduPilot software suite consists of navigation software (typically referred to as firmware when it is compiled to binary form for microcontroller hardware targets) running on the vehicle (either Copter, Plane, Rover, AntennaTracker, or Sub), along with ground station controlling software including Mission Planner, APM Planner, QGroundControl, MavProxy, Tower and others.ArduPilot source code is stored and managed on GitHub, with almost 400 total contributors.

The software suite is automatically built nightly, with continuous integration and unit testing provided by Travis CI, and a build and compiling environment including the GNU cross-platform compiler and Waf. Pre-compiled binaries running on various hardware platforms are available for user download from ArduPilot's sub-websites.

SITL

The SITL (software in the loop) simulator allows you to run Plane, Copter or Rover without any hardware. It is a build of the autopilot code using an ordinary C++ compiler, giving you a native executable that allows you to test the behaviour of the code without hardware.

Overview

SITL allows you to run ArduPilot on your PC directly, without any special hardware. It takes advantage of the fact that ArduPilot is a portable autopilot that can run on a very wide variety of platforms. Your PC is just another platform that ArduPilot can be built and run on.

When running in SITL the sensor data comes from a flight dynamics model in a flight simulator. ArduPilot has a wide range of vehicle simulators built in, and can interface to several external simulators. This allows ArduPilot to be tested on a very wide variety of vehicle types. For example, SITL can simulate:
A big advantage of ArduPilot on SITL is it gives you access to the full range of development tools available to desktop C++ development, such as interactive debuggers, static analyzers and dynamic analysis tools. This makes developing and testing new features in ArduPilot much simpler.

2.1.4 Running SITL (Software in the loop)

The ArduPilot SITL environment has been developed to run natively on both Linux and Windows. For setup instructions see Setting Up SITL for more information. Using SITL is explained in Using SITL. For examples of starting and using SITL for a particular vehicle see Examples of using SITL by Vehicle.

Mission Planner (Windows) also provides a simple means of running SITL for the master branch and stable branches of vehicles. See Mission Planner Simulation.

2.1.5 Mission Planner

It is a full-featured GCS supported by ArduPilot. It offers point-and-click interaction with your hardware, custom scripting, and simulation.

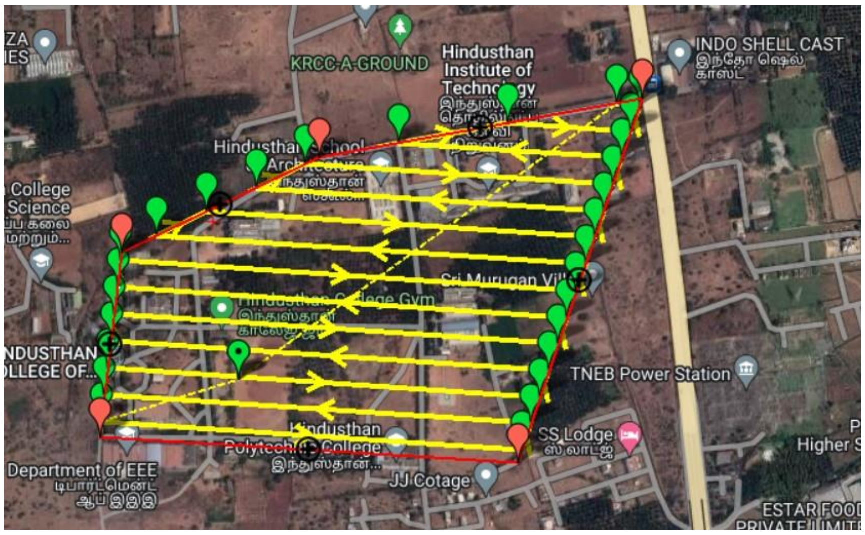


Figure 2.2 Mission Planner

A mission planner helps plan missions for an organisation or military entity. Job duties vary, but they may include preparing an autopilot route for aircraft or deciding what vehicle is necessary to complete a mission.

2.1.6 Ground Station

Ground stations for UAVs, or ground control stations for UAVs are land-based communications and control systems typically used for direct piloting and communication between the crew and a UAV. These ground control systems typically allow for both piloting of the craft and streaming live video and data.

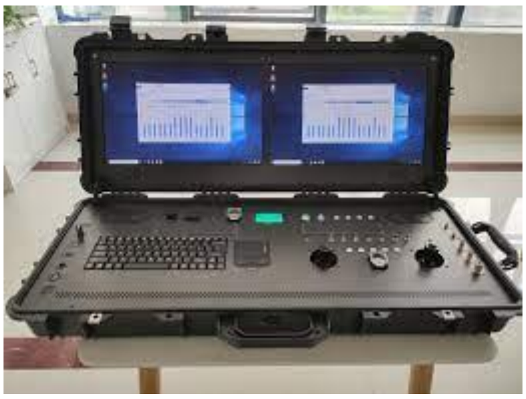


Figure 2.3 Ground Station

Ground Control Stations (GCS) are sets of ground-based hardware and software that allow UAV operators to communicate with and control a drone and its payloads, either by setting parameters for autonomous operation or by allowing direct control of the UAV.

2.1.7 MAVLINK

MAVLINK

MICRO AIR VEHICLE COMMUNICATION PROTOCOL

MAVLink is a very lightweight messaging protocol for communicating with drones (and between onboard drone components). MAVLink follows a modern hybrid publish-subscribe and point-to-point design pattern: Data streams are sent / published as topics while configuration sub-protocols such as the mission protocol or parameter protocol are point-to-point with retransmission. Messages are defined within XML files. Each XML file defines the message set supported by a particular MAVLink system, also referred to as a "dialect". The reference message set that is implemented by most ground control stations and autopilots is defined in common.xml

2.1.8 Pymavlink

This is a Python implementation of the MAVLink protocol. It includes a source code generator (generator/mavgen.py) to create MAVLink protocol implementations for other programming languages as well. Also contains tools for analysing flight logs.

2.2 Dronekit Python Script

```
#############DEPENDENCIES#######################
from dronekit import connect,
VehicleMode,LocationGlobalRelative,APIException
import time
import socket
import exceptions
import math

##############FUNCTIONS#########################
##Function to arm the drone props and takeoff at targetHeight (m)
def arm_and_takeoff(targetHeight):
        while vehicle.is_armable!=True:
                print("Waiting for vehicle to become armable.")
                time.sleep(1)
        print("Vehicle is now armable")
        vehicle.mode = VehicleMode("GUIDED")
        while vehicle.mode!='GUIDED':
                print("Waiting for drone to enter GUIDED flight mode")
                time.sleep(1)
        print("Vehicle now in GUIDED MODE. Have fun!!")
        vehicle.armed = True
        while vehicle.armed==False:
                print("Waiting for vehicle to become armed.")
                time.sleep(1)
        print("Look out! Virtual props are spinning!!")
        vehicle.simple_takeoff(targetHeight)
```

```
    while True:
        print("Current Altitude:
%d"%vehicle.location.global_relative_frame.alt)
        if vehicle.location.global_relative_frame.alt>=.95*targetHeight:
            break
        time.sleep(1)
    print("Target altitude reached!!")
    return None
############MAIN EXECUTABLE#############
####sim_vehicle.py opens up port on localhost:14550
vehicle = connect('127.0.0.1:14550',wait_ready=True)
###Arm the drone and takeoff into the air at 5 meters
arm_and_takeoff(5)
print("Vehicle reached target altitude")
####Once drone reaches target altitude, change mode to LAND
vehicle.mode=VehicleMode('LAND')
while vehicle.mode!='LAND':
    print("Waiting for drone to enter LAND mode")
    time.sleep(1)
print("Vehicle now in LAND mode. Will touch ground shortly.")
```

CHAPTER 3

DJI TELLO

3. DJI TELLO FOR PROGRAMMING:

1. Collision avoidance
2. Programmable drone
3. 5MP Camera, 720p Recording
4. Flight time - 13 Min
5. Flight distance - 100m

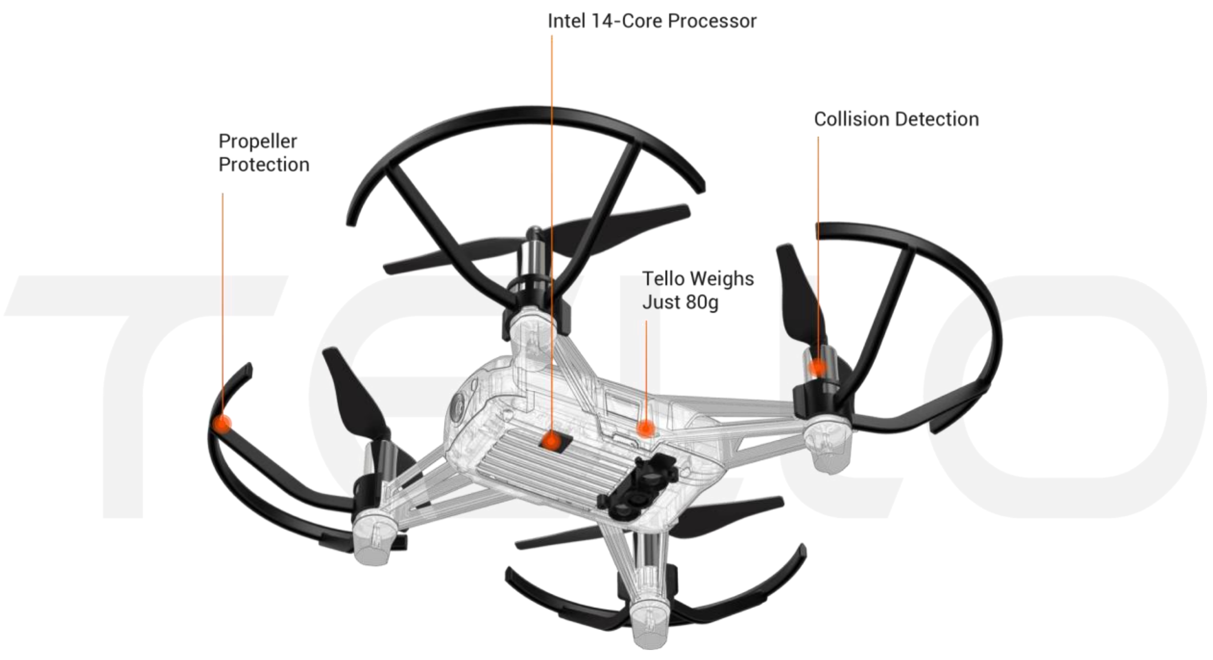


Figure 3.1 DJI Tello Drone

3.1 Description of components of the Tello drone:

1. Propellers
2. Motors
3. Drone Status Indicator
4. Camera
5. Power Button
6. Antennas

7. Vision Positioning System
8. Battery
9. Micro USB port
10. Propeller protectors

3.2 Description of the aircraft

Its dimensions and characteristics make it very manageable.

Weight	87 g
Dimensions	98×92.5×41 mm
Propeller	3 inches
Integrated functions	Telemetric sensor
	Barometer
	LED
	Vision System
	Wi-Fi 2.4 GHz 802.11n
	Real-time streaming 720p
Port	Port USB battery charging port
Operating temperature range	from 0° to 40°
Operating frequency range	from 2.4 to 2.4835 GHz
Transmitter (EIRP)	20 dBm (FCC)

	19 dBm (CE)
	19 dBm (SRRC)

Details in operation

Being a drone for mainly indoor use, its characteristics that define its operations are quite limited.

Maximum distance of flight	100 m
Maximum speed	8 m/s
Maximum flight time	13 min
Maximum flight height	30 m

3.3 WIFI EXTENDER

The Ryze Tello is controlled via a WiFi connection to your Smartphone. We wanted to boost the power of this signal and see if we could seek out some more range and better video transmission from the Tello. We did some research on antennas and by far the most popular budget option seems to be the Xiaomi MI WiFi Extender. So we got the MI Wifi extender to give it a shot – we were not disappointed.

What is Need

1. DJI Tello (OK, it is really a “Ryze Tello”).
2. The Xiaomi MI WiFi Extender.
3. A USB Battery Backup. We recommend a rectangular shape to sit flat on any surface and hold your antenna up vertically.

How Does the MI Wifi Extender Work?

The MI WiFi Extender (or WiFi repeater) works by going between your connected device and the WiFi source. It passes the information between the connected device and the source, thereby extending the range between the two. In your home, the source will typically be a WiFi router, while the connected device is your smartphone, tablet, computer, smart TV, etc. We typically only think of an extender working if it is positioned at a midpoint between the source and the connected device – but that's now how we used the MI WiFi Extender.

When flying a drone you can't position the extender between your controlling device (smartphone) and the source (the drone). Assuming you keep the extender near the pilot, the only way the extender helps is if its antenna is somehow superior to the antenna in your phone.

How Much Extra Range?

We tested the MI Wifi extender with the DJI Tello to see how much range it has with the default Tello network as compared to the network created by the MI Wifi Extender. The results were shocking.

We tested in two locations. At the first location, we got about 100 feet (30 metres) of range without the extender and 250 feet (75 metres) with the MI Wifi extender. That's a 2.5X improvement. Not bad. At 250 feet your Tello is a small spec that requires 20/20 vision to see.

Tello Range Test Map

At our first test location, the Tello flew more than 200 feet in several directions.We did even better at the second location. We were again only able to fly about 100 feet unaided by the improved antenna. With the MI Wifi extender turned on, but with the antenna lying flat parallel to the ground, we only added about 50 more feet. When we turned the antenna up we again reached 250 feet.

Then we discovered something interesting. When we turned the Tello around facing the pilot the signal improved. So we were then able to fly it backward another 300 feet! So with the Tello facing the pilot and flying backward, it was able to go a whopping 550 feet (170 metres). That’s more than 5 times the range when flying forward unassisted. Apparently, the Tello was built for “dronies” (selfies, pictures and videos taken from a drone)

3.4 Supporting Software for the System:

3.4.1 PYTHON - (Version 2.7)

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

3.4.2 IDE - Pycharm

PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive Python, web, and data science development.

3.4.3 PYTHON DRONE PROGRAMMING

Now that we have a basic understanding of the open source drone software stack, let’s actually start drone coding with python dronekit! As noted previously, we can even begin drone programming without an actual drone! We’ll do this with the ArduPilot SITL simulator.

3.4.4 OPENCV (LIBRARY)

OpenCV is the huge open-source library for computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human.

Figure 3.2 Open CV

Building a drone that can be operated by computer vision can seem challenging, but with the appropriate information, it is a lot simpler than you might think. What you need to construct an opencv drone is provided here.

3.4.5 TELLOPY (LIBRARY)

This is a python package which controls the DJI toy drone 'Tello'. The major portion of the source code was ported from the driver of the GOBOT project.The Tello-SDK is a Java project. It contains several packages, each with one or more classes that present the data and methods used to control a Tello drone. The first package is called tello.

CHAPTER 4

SIMPLE MISSION ON TELLO DRONE

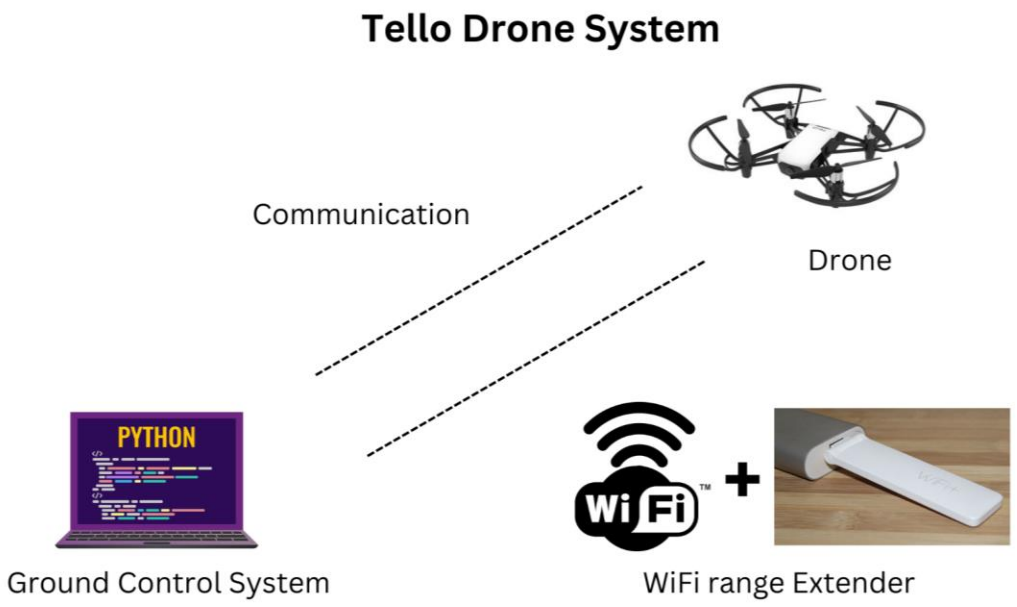
4. Simple Mission on Tello Drone

Forward and Backward: Start with the value of DESIRED_OBJECT_SIZE (The value is the size from the left upper corner coordinate to the right lower corner), then subtract the distance between the left upper corner of the object and the right lower corner and multiply that result to the value of MAX_SPEED_FORWARDBACK. A positive result means the object is far, and the drone will approach it. Negative results will cause the drone to go backward. Value equal or near cero makes the drone maintain the distance.

Up and down: First calculate the Y coordinate of the centre of the object, then subtract the Y coordinate of the centre of the frame, multiply that result to the value of MAX_SPEED_UPDOWN, and multiply the result by -1. A positive result means the object is in the lower part of the image and the drone needs to move up. Negatives will make the drone go down. With a value near or equal to zero the drone maintains the altitude.

Yaw: First calculate the X coordinate of the centre of the object, then subtract the X coordinate of the centre of the frame, finally multiply that result to the value of MAX_SPEED_YAW. A positive result means the object is to the left and the drone will rotate to that direction. Negative will rotate to the right. Values near or equal to cero cause the drone to not rotate.

4.1 Illustration of Tello Drone System



4.2 Python for Drones

The program for drones that we will use is Python programming language. Python is a programming language that lets you work quickly and integrate systems more effectively. To do this we are going to install Python and use IDE which is an integrated programming environment for writing the code. First, we are going to python.org in the download section where we can download the latest version. Then we can go to the PyCharm download section and download IDE. When Python and PyCharm are installed we are ready to start. To program a Tello drone we need libraries. So we have to install them as well. We need the dji tellopy library and opencv-python library. After we import these libraries, we can start coding.

```
from djitellopy import tello
```

We will need to import a time library. This will allow us to insert delays between each command.

```
From time import sleep
```

We have to create a tello object.

```
me = tello.Tello()
```

Now we can simply connect this object and it will take care of all ip addresses and all the communication parts for you.

```
me.connect()
```

The created object allows us to issue commands to the drone and to read some value from the drone, for example if we want to know what the battery status is we use the get_battery () method.

```
print(me.get_battery())
```

The list of methods is available if you click on the Ctrl and then click the left mouse button.

Now we can see all of the functions we can use. Before we start our code we need to check if our drone is connected to wifi. We have to turn on the drone. After that, go to wifi settings and connect the drone. When we have done that we can start our code.

Now we need to see how we can control our drone. We can simply write - go forward:

```
me.move_forward(30)
```

But we won't be able to control the speed, like we discussed in the introduction of the drone. It can translate into a tree so we want to control that. We will write:

```
me.send_rc_control(left_right_velocity,forward_backward_velocity,up_down_velocity,yaw_velocity)
```

This function has four figures, first is left_right_velocity, second is forward_backward_velocity, third is up_down_velocity and forth is yaw_velocity. All of these speeds are used for drone control and all of these can be used from minus 100 to 100. By combining these speeds, we achieve the desired movement of the drone.

After issuing the command, the duration of the action given by the command should be determined, and we do this by calling the sleep function with the parameter given in seconds.

```
sleep(2)
```

Before using the command for moving the drone, we should command the drone to take off and after all, command for landing. These commands are:

```
me.takeoff()
```

and

```
me.land()
```

So, these are the basic movements that we can control. We can control up and down, left and right, forwards and backwards and we can control the jaw. Now, we are going to learn how to capture images from our drone. What we need to do is to turn on the stream. This stream gives us all the frames one by one and we can process it.

```
me.stream_on()
        while True:
        img=me.get_frame_read().frame
         img=cv2.resize(img,(360,240))
        cv2.imshow("Image",img)
        cv2.waitKey(1)
```

With the help of this we are going to work on the Topics;

- Basics movements
- Object tracking
- Collision avoidance
- Swarming
- Mapping
- Line follower

Task 1: Basic Movements

```
from djitellopy import tello
from time import sleep
drone = tello.Tello()
drone.get_battery()
drone.connect()
print(drone.get_battery())

# parameter
lr = input("Left Right value\n")
fb = input("Forward value\n")
ud = input("Altitude\n")
yv = input("Rotate angle\n")
repeat = input("Loop value\n")

drone.takeoff()
counter = 0
while counter <= repeat:
   drone.send_rc_control(0, 0, 0, 0)
   sleep(2)
   drone.send_rc_control(0, 0, ud, 0)
   sleep(2)
   drone.send_rc_control(0, fb, 0, 0)
   sleep(2)
   drone.send_rc_control(lr, 0, 0, 0)
   sleep(2)
   drone.send_rc_control(0, -fb, 0, 0)
   sleep(2)
   drone.send_rc_control(lr, 0, 0, 0)
   sleep(2)
   counter = counter+1
```

Task 2:

Let the drone go 50cm forward, then 50cm left, then 50cm back, then 50cm right. Before landing, give an order to go 10 cm up and then 10 cm down.

```
from djitellopy import *
from time import sleep
me = tello.Tello()
me.connect()
me.takeoff()
me.move_forward(50)
sleep(2)
me.move_left(50)
sleep(2)
 me.move_back(50)
sleep(2)
me.move_right(50)
sleep(2)
me.move_up(50)
sleep(2)
me.move_down(50)
sleep(2)
me.land()
```

Task 3

Drone movement is repeated 5 times before landing.

```
from djitellopy import tello
from time import sleep
me = tello.Tello()
me.connect()
me.takeoff()
def movements():
        me.move_forward(50)
        sleep(2)
        me.move_left(50)
        sleep(2)
        me.move_back(50)
        sleep(2)
        me.move_right(50)
        sleep(2)
        me.move_up(50)
        sleep(2)
        me.move_down(50)
        sleep(2)
for i in range(5):
        movements()
me.land()
```

Task 4

Establish a connection with the camera on the drone and show what it "sees". Let the drone lift and turn 5 times while sending the video.

```
from djitellopy import tello
import cv2
me = tello.Tello()
```

```
me.connect()
me.takeoff()
me.streamon()

i = 0
while i < 5:
        me.send_rc_control(0, 0, 0, 50)
        img = me.get_frame_read().frame
        img = cv2.resize(img, (360, 240))
        cv2.imshow("Image", img)
        cv2.waitKey(1)
        i = i + 1
me.land()
```

Task 5

Make the program so that the drone is controlled by keyboard.

```
import KeyPress as kp
from djitellopy import tello
from time import sleep
kp.init()
me = tello.Tello()
me.connect()
me.takeoff()
def getKeyboardInput():
lr, fb, ud, yv = 0, 0, 0, 0
speed = 50
if kp.getkey("LEFT"):
        lr = speed
elif kp.getkey("RIGHT"):
        lr = -speed
    if kp.getkey("UP"):
        fb = speed
    elif kp.getkey("DOWN"):
        fb = -speed
    if kp.getkey("w"):
        ud = speed
    elif kp.getkey("s"):
        ud = -speed
    if kp.getkey("a"):
        yv = speed
    elif kp.getkey("d"):
        yv = -speed
    if kp.getkey("q"):
        me.land()
    if kp.getkey("i"):
        me.takeoff()
    return [lr, fb, ud, yv]
while True:
    vals = getKeyboardInput()
    me.send_rc_control(vals[0], vals[1], vals[2], vals[3])
    sleep(0.05)
```

Task 6

Modify the previous program by adding a button that will save the current camera image to
a file.

```
import KeyPress as kp
```

```
from djitellopy import tello
import time
import cv2
kp.init()
me = tello.Tello()
me.connect()
global img
me.streamon()
me.takeoff()
def getKeyboardInput():
        lr, fb, ud, yv = 0, 0, 0, 0
        speed = 50
         if kp.getkey("LEFT"):
        lr = speed
 elif kp.getkey("RIGHT"):
         lr = -speed
 if kp.getkey("UP"):
        fb = speed
 elif kp.getkey("DOWN"):
        fb = -speed
 if kp.getkey("w"):
         ud = speed
 elif kp.getkey("s"):
         ud = -speed
 if kp.getkey("a"):
        yv = speed
 elif kp.getkey("d"):
        yv = -speed
 if kp.getkey("q"): me.land(); time.sleep(3)
 if kp.getkey("i"): me.takeoff()
 if kp.getkey("z"):
         cv2.imwrite(f' Resources/Images/{time.time()}.jpg', img)
         time.sleep(0.3)
 return [lr, fb, ud, yv]

while True:
        vals = getKeyboardInput()
        me.send_rc_control(vals[0], vals[1], vals[2], vals[3])
         img = me.get_frame_read().frame
         img = cv2.resize(img, (360, 240))
         cv2.imshow("Image", img)
         cv2.waitKey(1)
```

CHAPTER 5

IMPLEMENTATION

5.1 OBJECT TRACKING CODE

Figure 5.1 Object Tracking

```
import numpy as np
import cv2
def run_main():
    cap = cv2.VideoCapture('upabove.mp4')
    # Read the first frame of the video
    ret, frame = cap.read()
    # Set the ROI (Region of Interest). Actually, this is a
    # rectangle of the building that we're tracking
    c,r,w,h = 900,650,70,70
    track_window = (c,r,w,h)
    # Create mask and normalised histogram
    roi = frame[r:r+h, c:c+w]
    hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv_roi, np.array((0., 30.,32.)),
np.array((180.,255.,255.)))
    roi_hist = cv2.calcHist([hsv_roi], [0], mask, [180], [0, 180])
    cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
    term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT,
80, 1)

    while True:
        ret, frame = cap.read()
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        dst = cv2.calcBackProject([hsv], [0], roi_hist, [0,180], 1)
        ret, track_window = cv2.meanShift(dst, track_window, term_crit)
        x,y,w,h = track_window
        cv2.rectangle(frame, (x,y), (x+w,y+h), 255, 2)
        cv2.putText(frame,                'Tracked',                (x-25,y-10),
cv2.FONT_HERSHEY_SIMPLEX,
            1, (255,255,255), 2, cv2.CV_AA)

        cv2.imshow('Tracking', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
if __name__ == "__main__":
    run_main()
```

5.2 MAPPING CODE

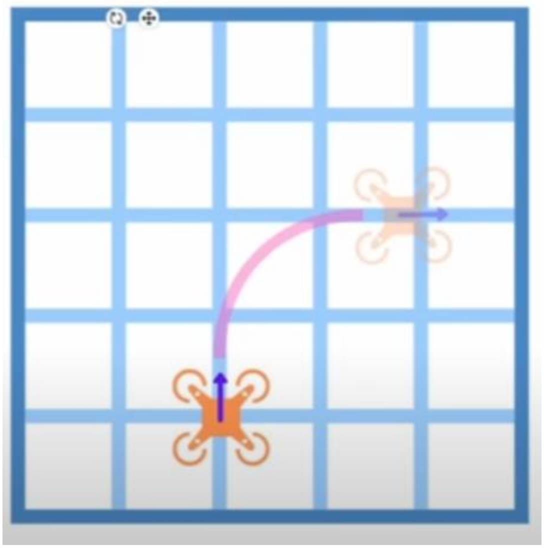


Figure 5.2. Mapping

```
from djitellopy import tello
import KeyPressModule as kp
import numpy as np
from time import sleep
import cv2
import math

######## PARAMETERS ###########
fSpeed = 117 / 10  # Forward Speed in cm/s   (15cm/s)
aSpeed = 360 / 10  # Angular Speed Degrees/s  (50d/s)
interval = 0.25
dInterval = fSpeed * interval
aInterval = aSpeed * interval

x, y = 500, 500
a = 0
yaw = 0
kp.init()
me = tello.Tello()
me.connect()
print(me.get_battery())
points = [(0, 0), (0, 0)]

def getKeyboardInput():
    lr, fb, ud, yv = 0, 0, 0, 0
    speed = 15
    aspeed = 50
    global x, y, yaw, a
    d = 0
```

```
if kp.getKey("LEFT"):
    lr = -speed
    d = dInterval
    a = -180
elif kp.getKey("RIGHT"):
    lr = speed
    d = -dInterval

    a = 180
if kp.getKey("UP"):
    fb = speed
    d = dInterval
    a = 270

elif kp.getKey("DOWN"):
    fb = -speed
    d = -dInterval
    a = -90
if kp.getKey("w"):
    ud = speed
elif kp.getKey("s"):
    ud = -speed
if kp.getKey("a"):
    yv = -aspeed
    yaw -= aInterval
elif kp.getKey("d"):
    yv = aspeed
    yaw += aInterval
    if kp.getKey("q"): me.land(); sleep(3)
    if kp.getKey("e"): me.takeoff()
    sleep(interval)
    a += yaw
    x += int(d * math.cos(math.radians(a)))
    y += int(d * math.sin(math.radians(a)))
    return [lr, fb, ud, yv, x, y]

def drawPoints(img, points):
    for point in points:
        cv2.circle(img, point, 5, (0, 0, 255), cv2.FILLED)
    cv2.circle(img, points[-1], 8, (0, 255, 0), cv2.FILLED)
    cv2.putText(img, f'({(points[-1][0] - 500) / 100},{(points[-1][1] - 500) /
100})m',
                (points[-1][0] + 10, points[-1][1] + 30),
cv2.FONT_HERSHEY_PLAIN, 1,
                (255, 0, 255), 1)
while True:
    vals = getKeyboardInput()
    me.send_rc_control(vals[0], vals[1], vals[2], vals[3])
    img = np.zeros((1000, 1000, 3), np.uint8)
    if points[-1][0] != vals[4] or points[-1][1] != vals[5]:
        points.append((vals[4], vals[5]))
    drawPoints(img, points)
    cv2.imshow("Output", img)
    cv2.waitKey(1)
```

5.3 LINE FOLLOWER CODE

Figure 5.3 Line Follower

```
import numpy as np
from djitellopy import tello
import cv2
me = tello.Tello()
 me.connect()
print(me.get_battery())
me.streamon()
#me.takeoff()
cap = cv2.VideoCapture(1)
hsvVals = [0,0,188,179,33,245]
sensors = 3
threshold = 0.2
width, height = 480, 360
senstivity = 3  # if number is high less sensitive
weights = [-25, -15, 0, 15, 25]
fSpeed = 15
curve = 0
def thresholding(img):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    lower = np.array([hsvVals[0], hsvVals[1], hsvVals[2]])
    upper = np.array([hsvVals[3], hsvVals[4], hsvVals[5]])
    mask = cv2.inRange(hsv, lower, upper)
    return mask
def getContours(imgThres, img):
    cx = 0
    contours, hieracrhy = cv2.findContours(imgThres, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
    if len(contours) != 0:
        biggest = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(biggest)
        cx = x + w // 2
        cy = y + h // 2
        cv2.drawContours(img, biggest, -1, (255, 0, 255), 7)
         cv2.circle(img, (cx, cy), 10, (0, 255, 0), cv2.FILLED)
    return cx
def getSensorOutput(imgThres, sensors):
    imgs = np.hsplit(imgThres, sensors)
    totalPixels = (img.shape[1] // sensors) * img.shape[0]
    senOut = []
    for x, im in enumerate(imgs):
        pixelCount = cv2.countNonZero(im)
        if pixelCount > threshold * totalPixels:
            senOut.append(1)
        else:
            senOut.append(0)
        # cv2.imshow(str(x), im)
    # print(senOut)
    return senOut
def sendCommands(senOut, cx):
    global curve
    ## TRANSLATION
    lr = (cx - width // 2) // senstivity
    lr = int(np.clip(lr, -10, 10))
    if 2 > lr > -2: lr = 0
    ## Rotation
    if   senOut == [1, 0, 0]: curve = weights[0]
    elif senOut == [1, 1, 0]: curve = weights[1]
    elif senOut == [0, 1, 0]: curve = weights[2]
    elif senOut == [0, 1, 1]: curve = weights[3]
    elif senOut == [0, 0, 1]: curve = weights[4]
    elif senOut == [0, 0, 0]: curve = weights[2]
    elif senOut == [1, 1, 1]: curve = weights[2]

    elif senOut == [1, 0, 1]: curve = weights[2]
    me.send_rc_control(lr, fSpeed, 0, curve)
while True:
    #_, img = cap.read()
    img = me.get_frame_read().frame
    img = cv2.resize(img, (width, height))
    img = cv2.flip(img, 0)
    imgThres = thresholding(img)
    cx = getContours(imgThres, img)  ## For Translation
    senOut = getSensorOutput(imgThres, sensors)  ## Rotation
    sendCommands(senOut, cx)
    cv2.imshow("Output", img)
    cv2.imshow("Path", imgThres)
    cv2.waitKey(1)
```

Color Picker

Python

```
from djitellopy import tello
import cv2
import numpy as np
frameWidth = 480
frameHeight = 360
me = tello.Tello()
me.connect()
print(me.get_battery())
me.streamon()
def empty(a):
    pass
cv2.namedWindow("HSV")

cv2.resizeWindow("HSV", 640, 240)
cv2.createTrackbar("HUE Min", "HSV", 0, 179, empty)
cv2.createTrackbar("HUE Max", "HSV", 179, 179, empty)
cv2.createTrackbar("SAT Min", "HSV", 0, 255, empty)
cv2.createTrackbar("SAT Max", "HSV", 255, 255, empty)
cv2.createTrackbar("VALUE Min", "HSV", 0, 255, empty)
cv2.createTrackbar("VALUE Max","HSV", 255, 255, empty)
#cap = cv2.VideoCapture(1)
frameCounter = 0
while True:
    img = me.get_frame_read().frame
    #_, img = cap.read()
    img = cv2.resize(img, (frameWidth, frameHeight))
    img = cv2.flip(img,0)
    imgHsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h_min = cv2.getTrackbarPos("HUE Min", "HSV")
    h_max = cv2.getTrackbarPos("HUE Max", "HSV")
    s_min = cv2.getTrackbarPos("SAT Min","HSV")
    s_max = cv2.getTrackbarPos("SAT Max", "HSV")
    v_min = cv2.getTrackbarPos("VALUE Min", "HSV")
    v_max = cv2.getTrackbarPos("VALUE Max", "HSV")
    lower = np.array([h_min, s_min, v_min])
    upper = np.array([h_max, s_max, v_max])
    mask = cv2.inRange(imgHsv, lower, upper)
    result = cv2.bitwise_and(img, img, mask=mask)
    print(f'[{h_min},{s_min},{v_min},{h_max},{s_max},{v_max}]')
    mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)
    hStack = np.hstack([img, mask, result])

    cv2.imshow('Horizontal Stacking', hStack)
    if cv2.waitKey(1) and 0xFF == ord('q'):
        Break
```

5.4 REAL TIME INPUT FROM USER

```
from djitellopy import tello
from time import sleep

drone = tello.Tello()

drone.get_battery()
drone.connect()
print(drone.get_battery())
```

```
# parameter
lr = input("Left Right value\n")
fb = input("Forward value\n")
ud = input("Altitude\n")
yv = input("Rotate angle\n")
repeat = input("Loop value\n")

drone.takeoff()
counter = 0
while counter <= repeat:
    drone.send_rc_control(0, 0, 0, 0)
    sleep(2)
    drone.send_rc_control(0, 0, ud, 0)
    sleep(2)
    drone.send_rc_control(0, 0, 0, 0)
    sleep(2)
    drone.send_rc_control(0, fb, 0, 0)
    sleep(2)
    drone.send_rc_control(0, 0, 0, 0)
    sleep(2)
    drone.send_rc_control(lr, 0, 0, 0)
    sleep(2)
    drone.send_rc_control(0, 0, 0, 0)
    sleep(2)
    drone.send_rc_control(0, -fb, 0, 0)
    sleep(2)
    drone.send_rc_control(0, 0, 0, 0)
    sleep(2)
    drone.send_rc_control(lr, 0, 0, 0)
    sleep(2)
    counter = counter + 1

drone.send_rc_control(0, 0, 0, 0)
sleep(2)
drone.land()
```

RESULT AND DISCUSSION

Conclusion

The tests related to this project have been performed in an open environment and a few are in a closed controlled environment, at different locations at the university campus. The machine/laptop/ Notebook is used for this test is HP VICTUS, which equipped with a Ryzen processor (Ryzen 5 5000H, 8 core) and 8GB RAM, no GPU is used for the computing process, which makes it more versatile and adaptable. Because a lot of the pose recognition frameworks which are available in the market are mostly required a powerful computer with GPU, but this framework can be used in low-end devices without GPU power. First, an investigation was carried out to view what were the other possible drones available in the market which offers the same functionality as Tello and Tello EDU. And, by doing that investigation it observed that there were a lot of other drones present in the market. But Tello played a noticeable role in the field of education, which is why it was decided to use that in this project along with that it also provides some base for future developers who wanted to start programming with the drone for fun and other applications.

Second, another research was carried out regarding the pose estimation, as we already discussed in our Pose Detection chapter, where we mentioned the different types of pose recognition models and ML frameworks available and can be used to create a project, but we choose the one which is more versatility and required only CPU.

In addition, we go through many contents and paperwork which consists of controlling a drone through body movements or gestures or pose. During the research, we found a lot of scripts that can also control a drone through body movements, but these scripts were more complicated and required a lot of processing power for both CPU & GPU. Although, these scripts were provided me with the basic knowledge for programming this project.

Third, our objective of this project is to make the script more versatile, easy and adaptable, so that it can be used on any platform which is capable to run the python script. In our project, I used Windows 11 OS along with PyCharm as a Python IDE. The purpose of using PyCharm because it provides smart code completion, code inspections, on-the-fly error highlighting and quick-fixes, along with automated code refactoring and rich navigation capabilities and its good for beginners using Windows OS instead of Linux OS or Mac OS as people are more familiar with is OS which also motivates new coders to learn more. However, users can use any other Python IDE and OS according to their convenience. In the beginning, this project seems to be so simple and easy but as we make progress and start working on the project to achieve our main application, we faced a lot of errors and failures while testing the drone flight and the code. However, from these mistakes and problems, we learned a lot of things and keep me motivated to pursue further in this project.

In this project, we have also created a few test programs to test the functionality of every function before applying them to the main application. These test codes can help others to understand the programming topology and its works. The testing codes can be used as building blocks for further developments in the application. You can easily access these testing codes and test them. These codes contain some basic manoeuvres of the Drone and the testing of Pose class along with another miscellaneous test like camera working or not, test flight take-off and landing, and testing other functions of the class.

Finally, this document can provide all the basic and advanced knowledge which allow for future developers to develop more complicated applications without any hustle and also save the time to do extensive research on the internet.

CONCLUSION

The aim of this work was to study, create and evaluate autonomous navigation algorithms allowing a small drone with few sensors (a Tello EDU, mainly with a single RGB frontal camera) to navigate without the supervision of a human pilot in an indoor environment (the corridors of the Montefiore Institute) free of dynamic obstacles. The assumption that the drone has access to a simple representation of its environment was made.

Firstly, after a preliminary review of the state of the art about the subject, a simulated environment was set up to perform initial tests in a safe way and without real world constraints. The simulator chosen was Unreal Engine 4 with the AirSim plugin and two simulated indoor environments have been created.

Secondly, a high-level control interface was designed. The latter is an additional layer to existing drone's manufacturer control interface allowing navigation algorithms to be usable with any drone model.

Thirdly, a representation of the environment, exploitable by the drone, was created. Using a binary occupancy grid, this representation allows the planning of paths and the extraction of information (called "key point") such as the position of turns, crossroads and staircases and their associated actions.

Fourthly, based on the implemented resources, several autonomous navigation algo- rithms have been designed and tested, first on simulator and then in the real world. Van-ishing point detection methods, via line detection and neural network, are used to align the drone. Three main methods have been used to design algorithms: image classification (using Deep Learning), depth estimation (using Deep Learning) and markers (ArUco) detection. Initially designed to follow a simple path in a single floor, more advanced techniques to manage staircases but also battery stations have been implemented.

Finally, the future of such autonomous systems, from a technical and legal point of view, was discussed.

REFERENCES

REFERENCES:

[1] K. M. Hasan, W. S. Suhaili, S. H. Shah Newaz, and M. S. Ahsan, “Development of an aircraft type portable autonomous drone for agricultural applications,” in *2020 International Conference on Computer Science and Its Application in Agriculture, ICOSICA 2020*, Sep. 2020. doi: 10.1109/ICOSICA49951.2020.9243257.

[2] D. Floreano and R. J. Wood, “Science, technology and the future of small autonomous drones,” *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 460–466, May 27, 2015. doi: 10.1038/nature14542.

[3] K. O. Said *et al.*, “On the application of drones: a progress report in mining operations,” *Int J Min Reclam Environ*, vol. 35, no. 4, pp. 235–267, 2021, doi: 10.1080/17480930.2020.1804653.

[4] U. R. Mogili and B. B. V. L. Deepak, “Review on Application of Drone Systems in Precision Agriculture,” in *Procedia Computer Science*, 2018, vol. 133, pp. 502–509. doi: 10.1016/j.procs.2018.07.063.

[5] H. Chen, X. M. Wang, and Y. Li, “A survey of autonomous control for UAV,” in *2009 International Conference on Artificial Intelligence and Computational Intelligence, AICI 2009*, 2009, vol. 2, pp. 267–271. doi: 10.1109/AICI.2009.147.

[6] V. Puri, A. Nayyar, and L. Raja, “Agriculture drones: A modern breakthrough in precision agriculture,” *Journal of Statistics and Management Systems*, vol. 20, no. 4, pp. 507–518, Jul. 2017, doi: 10.1080/09720510.2017.1395171.

[7] B. Majidi and A. Bab-Hadiashar, “Real Time Aerial Natural Image Interpretation for Autonomous Ranger Drone Navigation,” 2005.

[8] University of Mauritius, Institute of Electrical and Electronics Engineers. Mauritius Subsection, Mauritius Research Council, and Institute of Electrical and Electronics Engineers, *2017 1st International Conference on Next Generation Computing Applications (NextComp) : 19th-21st July 2017, Mauritius*.

[9] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous UAV guidance," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 57, no. 1–4, pp. 65–100, Jan. 2010, doi: 10.1007/s10846-009-9383-1.

[10] Y. Lu, Z. Xue, G. S. Xia, and L. Zhang, "A survey on vision-based UAV navigation," *Geo-Spatial Information Science*, vol. 21, no. 1, pp. 21–32, Jan. 2018, doi: 10.1080/10095020.2017.1420509.

[11] D. Rathbun, S. Kragelund, A. Pongpunwattana, and B. Capozzi, "AN EVOLUTION BASED PATH PLANNING ALGORITHM FOR AUTONOMOUS MOTION OFA UAV THROUGH UNCERTAIN ENVIRONMENTS."

[12] J. Tisdale, Z. W. Kim, and J. K. Hedrick, "Autonomous UAV path planning and estimation: An online path planning framework for cooperative search and localization," *IEEE Robot Autom Mag*, vol. 16, no. 2, pp. 35–42, 2009, doi: 10.1109/MRA.2009.932529.