

**No Permission No Take-off (NPNT) Compliance IOT
device and software for Digital sky platforms for Drone
operations in India**

A

Project Report

*Submitted in partial fulfilment of the
Requirements for the award of the Degree of*

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

By

B. PRANAVNATH (1602-16-737-028)

K. SREERAM (1602-16-737-047)

S. HARSHA RAO (1602-16-737-048)

Under the guidance of

Dr. VASAPPANAVARA RAMESH, PhD

PROFESSOR



**Department of Information Technology
Vasavi College of Engineering (Autonomous)
(Affiliated to Osmania University)
Ibrahimbagh, Hyderabad-31**

2020

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology



DECLARATION BY THE CANDIDATE

We, **B.PRANAVNATH, K.SREERAM, S.HARSHARAO**, bearing hall ticket numbers, **1602-16-737-028, 1602-16-737-047, 1602-16-737-048**, hereby declare that the project report entitled “**No Permission No Take-off (NPNT) Compliance IOT device and software for Digital sky platforms for Drone operations in India**” under the guidance of **Dr. VASAPPANAVARA RAMESH**, PhD, PROFESSOR, Department of Information Technology, Vasavi College of Engineering, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering in Information Technology**

This is a record of bonafide work carried out by me and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

B.PRANAVNATH
1602-16-737-028

K.SREERAM
1602-16-737-047

S.HARSHA RAO
1602-16-737-048

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Information Technology



BONAFIDE CERTIFICATE

This is to certify that the project entitled “**No Permission No Take-off (NPNT) Compliance IOT device and software for Digital sky platforms for Drone operations in India**” being submitted by **B.PRANAVNATH, K.SREERAM, S.HARSHARAO** bearing **1602-16-737-028, 1602-16-737-047, 1602-16-737-048** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Information Technology is a record of bonafide work carried out by him/her under my guidance.

Dr. VASAPPANAVARA RAMESH, PhD
PROFESSOR
Internal Guide

Dr. K. Ram Mohan Rao
HOD, IT

External Examiner

ACKNOWLEDGEMENT

I am obliged and grateful to **Dr. S.V. Ramana**, Principal and **Dr. K. Ram Mohan Rao**, Professor, and Head of the Department of IT, Vasavi College of Engineering (A), Hyderabad, for their valuable suggestions and guidance in all respects during the BE course.

I am highly indebted to **Dr. K. Raghavendra**, Scientist, ADRIN, for his valuable guidance and constant encouragement, and guided me all along, till the completion of my project work by providing all the necessary information for developing a good system. He projected me with much-needed direction and motivation to take up a project of this kind.

I owe my deep gratitude to our project co **Dr. VASAPPANAVARA RAMESH, PhD, internal guide** and **Mr. G. Rajashekhar, project coordinator**, who took keen interest on my project work.

I express my sincere thanks to all faculty members and lab technical assistants of the IT Department for their help and kind cooperation.

Finally, I would like to take the opportunity to thank my family and friends for their support throughout the work.

ABSTRACT

Drone/UAV (Unmanned Aerial Vehicle) is an emerging technology which is continuously evolving and being put to several novel uses cases like aerial photography, shipping/delivery, geographic mapping, disaster management, search and rescue, military applications etc. However, while operating the drone, safety precautions have to be taken to avoid any unwanted/undesired damage taking place either to lives or the properties of the individual or public. Hence, regulating the drone operations is mandatory to overcome such unwanted problems and to implement smooth drone operations through the regulations for the security and safety of human kind is very essential. Towards the safety and security of drone operations, the ministry of Civil – Aviation, India has been working for several years to establish a world class leading drone ecosystem in India for Drone Regulations through Civil Aviation Requirement (CAR) regulations. Towards these CAR developments, DGCA has put effort for the development of Digital sky operations platform for drones, which is an effort to overcome the traditional method of digitizing a paper based process for registering and operating drones. This effort has resulted in the development of system called as Digital Sky Platform which is the first of its kind national unmanned traffic management (UTM) platform in India for drone air traffic management and regulated control starting from registration of drones, issue of the pilot license, drone unique identification number monitoring the drone operations etc. This platform implements “**No Permission No Take Off**” (NPNT) system as a part of **Digital Sky Platform** which allows the drones operators, for carrying smooth operations through the digital platform. At present, for realization of NPNT compliance device and compatible to digital sky platform and its sub systems, many developments are taking place by several starts-up companies and industries to use for as per regulations to operate in India, and such NPNT systems are under still evolving and are under development.

Here, in this project, we propose to develop one such **NPNT compliance** system as an Internet of Things (IOT) device to use for drone operations as per CAR regulations for Drones in India. The proposed NPNT device is based on Open Standard devices and protocols widely used for building and operating the Drones.

Here, we propose to use Open Standard Flight Controller model aka Pixhawk, GPS module NeoM8N, Raspberry Pi, Micro Air Vehicle Link Protocol (MAVLINK) and Internet Protocols (IPV4) for the development of the device. We also propose the development of Map Control for Graphical User Interface (GUI) for viewing the drones under flying operations to monitor and view on the backdrop of map. To store the telemetry information of all such flying drones and their locations, here we propose to use Google Cloud Firebase platform, and such fetched locations are viewed on the Map Control.

We also propose to develop a client server application, where in client software runs on the IOT device (Raspberry pi), and the server aka Digital Sky platform used to monitor the drones operating in the preloaded path aka geo fence area. The project also implements client/server TCP/IP application, for commanding the drone through the issue of the commands like LAND, RTL (Return To Land Home Position) from the Digital Sky platform server to the IOT Drone Device, in case of flight is out of the pre-loaded geo fence area. This command and controlling would allow the Drone under flying to have the control of Digital Sky platform and to take any actions to overcome any unforeseen threats/damages taking place if any during the flight operations by the drone operator.

INDEX

1. Introduction.....	1
1.1. Drones/UAV Classification.....	1
1.2. Drone Major Sectors and Applications.....	2
1.3. Anatomy of Drone.....	3
1.4. DGCA regulations	5
1.5. NPNT Guidelines - NO PERMISSION NO TAKE-OFF.....	5
1.6 Literature Survey.....	5
2. Proposed Framework.....	9
2.1. Hardware Components.....	9
2.1.1. Raspberry pi.....	9
2.1.1.1. Raspberry Pi Setup.....	11
2.1.2. Flight Controller.....	13
2.1.3. GPS module.....	16
2.2. Software Components.....	17
2.2.1. Mission planner Overview.....	17
2.2.2. Loading Firmware.....	18
2.2.3. Putty.....	21
2.2.4. VNC viewer.....	24
3. Design and Implementation.....	28
3.1. Architecture Diagram.....	28
3.2. Flow Diagram.....	29
3.3. Use Case Diagram.....	29
3.4. Sequence Diagram.....	30
3.5. Drone Kit.....	31
3.6. MAVLINK.....	43
3.7. Firebase.....	44
3.8. Map Control.....	48
3.9. Pyqt5.....	56
4. Results and Testcases.....	61
5. Conclusion &Future work.....	64
6.References.....	65

List of Figures

Figure 1.1 Drone Categories/classification as per Civil Aviation, India.....	1
Figure 1.2.1 Drone sectors and applications.....	2
Figure 1.2.1 Drone Application Sectors.....	3
Figure 1.3.1 Drone Anatomy.....	4
Figure 2.1 Proposed Framework – NPNT Compliance IOT Device.....	9
Figure:2.1.1.1 Raspberry pi.....	10
Figure:2.1.1.1.1 Raspbian Versions.....	11
Figure:2.1.1.1.2 Win32 Disk Imager.....	11
Figure:2.1.2.1 Pixhawk.....	13
Figure:2.1.2.2 Labelling of Pixhawk.....	15
Figure:2.1.3.1 GPS Module.....	17
Figure:2.2.1.1 Mission Planner.....	17
Figure:2.2.2.1 Mission Planner Setup window.....	19
Figure:2.2.2.2 Pixhawk with USB connection.....	19
Figure:2.2.2.3 Select COM.....	20
Figure:2.2.2.4 Mission Planner: Install Firmware Prompt.....	20
Figure:2.2.3.1 PuTTY Configuration window.....	21
Figure:2.2.3.2 PuTTY window with translation window.....	22
Figure:3.2.1.3 PuTTY configuration window with IP address.....	23
Figure:3.2.1.4 Terminal window of raspberry pi.....	23
Figure:2.2.4.1 Raspberry pi software config. Tool setup.....	24
Figure:2.2.4.2 Raspberry pi software config. Tool setup.....	25
Figure:2.2.4.3 Raspberry pi software config. Tool setup.....	25
Figure:2.2.4.4 VNC Viewer window.....	26
Figure:2.2.4.5 Desktop window of Raspberry pi In VNC viewer.....	27
Figure:3.1.1 Architecture Diagram.....	28
Figure:3.2.1 Flow Diagram Diagram.....	29
Figure:3.3.1 Use Case Diagram.....	29
Figure:3.4.1 Sequence Diagram.....	30
Figure:3.5.1 Initiating simulator and waiting for other connections.....	32
Figure:3.5.2 Selecting the type of connection in mission planner.....	33
Figure:3.5.3 Selecting the type of connection in mission planner.....	33

Figure: 3.5.4 Drone symbol after connecting mission planner to simulation.....	34
Figure:3.5.5 Starting drone using a python program.....	35
Figure:3.5.6 Drone movement in mission planner.....	36
Figure:3.5.7 Client-Server communication between windows system and raspberry pi.....	37
Figure:3.5.8 Drone landing.....	37
Figure:3.6.1 RasPi Configuration Utility.....	43
Figure:3.6.2 And then “Serial”.....	44
Figure:3.7.1 Shows different types of databases in firebase.....	45
Figure: 3.7.2 shows the location data which was sent from raspberry pi.....	46
Figure:3.8.1 Creating Geofence area in Mapbox API.....	48
Figure:3.8.2 Map Accessing location data and showing it.....	49
Figure:3.9.1 QT Designer.....	57
Figure:3.9.2 Sample DGCA GUI	58
Figure:3.9.3 DGCA GUI (not final).....	58
Figure:4.1 shows the data in pi which is being sent from Pixhawk.....	61
Figure:4.2 Loading GPS data into firebase.....	62
Figure:4.3 GUI which displays multiple drones and control drones.....	62
Figure:4.4 Sending commands through client & server.....	63
Figure:4.5Drone landing after command being sent.....	63

1. INTRODUCTION

Unmanned Aerial Vehicles (UAV) / Drone are quickly becoming omnipresent in our lives with the new dimensions of UAVs entering into the new emerging markets. Below, we describe the classification of UAV's based on the All Up Weight (AUW) they can carry and discuss by the major application sectors they are operated. This classification however may have small variations across nations.

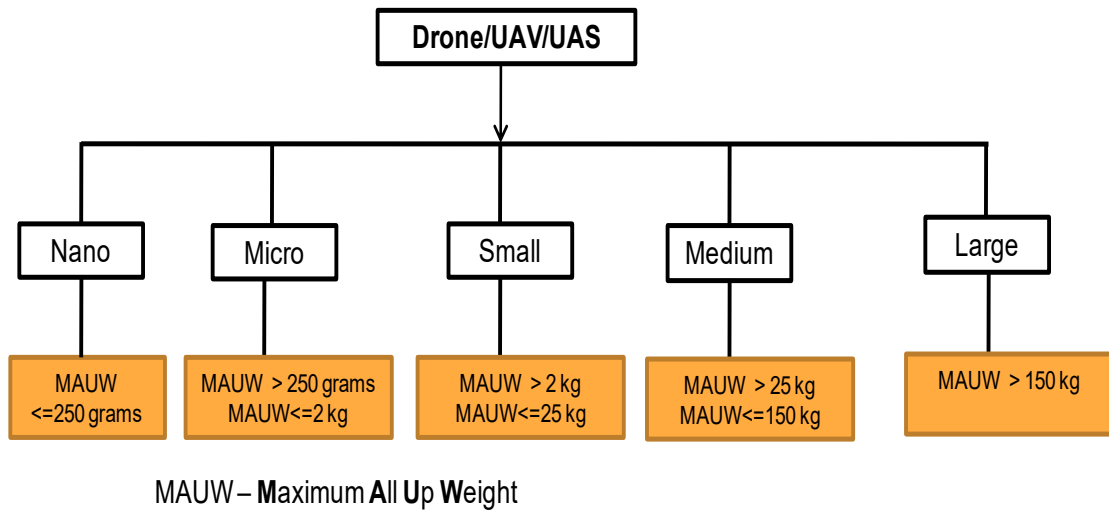


Figure 1.1: Drone Categories/classification as per Civil Aviation, India

NPNT or ‘No Permission – No Take-off’ is a software program that enables every RPA (except Nano) to obtain a valid permission through digital sky platform before operating in India. The NPNT compliant drone should have RFM (Registered Flight Module) and it is verified by the DGCA with a unique identification number (UIN) for each client/module.

1.1 Drones/UAV Classification

Drones are classified into five major categories as shown in Figure 1.1 (Director of Civil Aviation, 2018) to operate in India. The classifications are purely based on the maximum AUW(i.e. including payload weight) that drone can carry during the flight, and these classifications are stated as a part of drone regulations to operate unmanned aerial systems in India by December 2018. The drone

regulations also mention to have the unique ID for each drone from micro category to large category class to operate with the license being taken by the certified drone operator for operating the drone.

1.2 Drone Major Sectors and Applications

There are a variety of use cases for the drones such as: military, agriculture, law enforcement, natural resources, entertainment, commercial delivery and hobby usage (JoergSchlinkheider et.al, 2014) based on AUV they can carry. The several applications of drone in various major sectors are shown in Figure 1.2 , they are classified into the major sectors viz. Security applications to use in national security applications, traffic monitoring, and accessing to the remote areas, rescue operations etc. The second major sector is surveillance such as aerial Survey- to use in very high resolution image acquisition on demand and processing them to generate 3D images to use for several applications like disaster management, oil and gas exploration, Urban mapping, water resource monitoring etc.

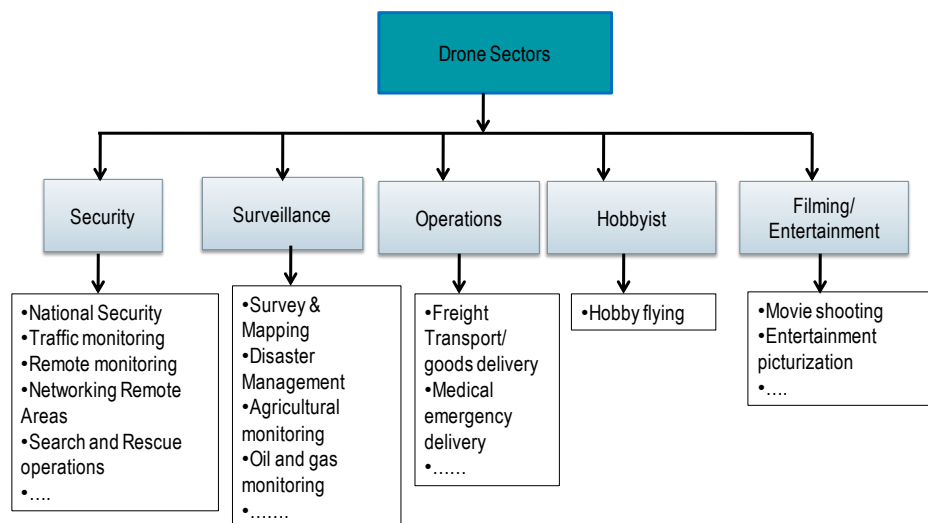


Figure 1.2.1 Drone sectors and applications

The aerial survey applications are further can be classified into three major areas such as

- Short range surveillance - Conducting short range surveillance, image capture, and analytics
- Long range surveillance - Conducting long-range surveillance, image capture and analytics

- Filming and Entertainment - Filming and entertaining industry where the drones are used for film shooting.
- Commercial service sector - To use for freight transport, goods delivery, emergency medical equipment delivery from one place to another etc.

The several drone application sectors are depicted pictorial in Figure 1.3.

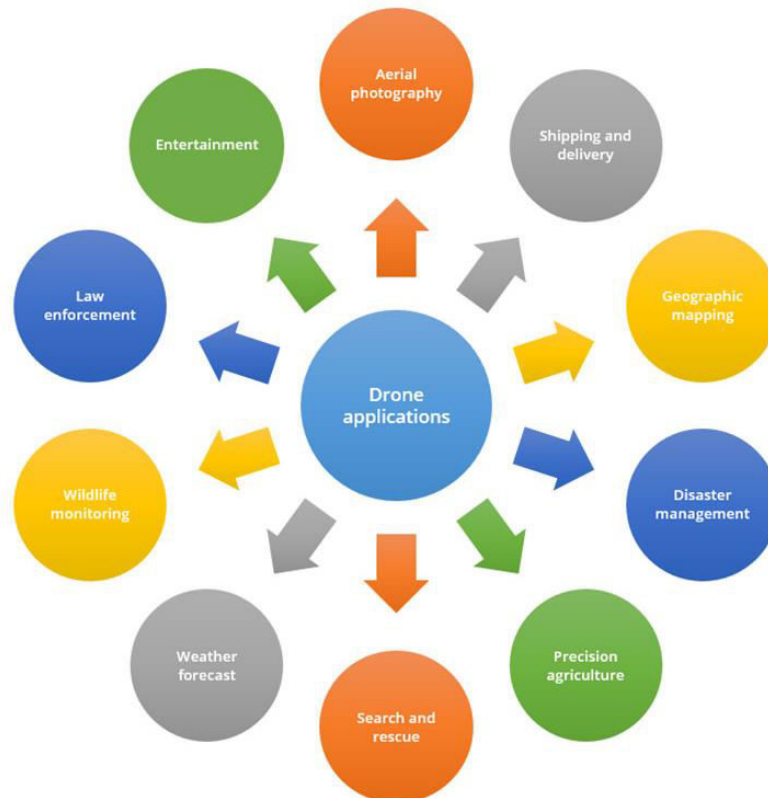


Figure 1.2.1: Drone Application Sectors

1.3Anatomy of Drone

The drone and its several components are depicted in Figure 1.3., the major components are Frame – around which all other parts get mounted, Arms – used for mounting the motors, Motors- Brushless DC motor with specified RPM to rotate for lifting of the drone, Power Distribution board – to supply the necessary essential power to all the sensors with necessary power connectors, Propellers – the major element to vertical lift off the drone with the proper pitch, Battery- another essential element for the drone to operate, Navigation or GPS – for the drone to navigate in the auto pilot mode, On Board computing (SOC) –used for

computing purpose, Flight Planner – the mission planning element for the mission to fly, payload – sensor payload which can be a camera in the case of aerial remote sensing applications with gimbals mount, telemetry – radio frequency device for monitoring the flight logs during the mission, Radio controller to operate the drone in manual mode and emergency cases and during lift off time, flight controller – the essential element to operate the drone, IMU sensors, and ESC – for supplying the current to the motors.

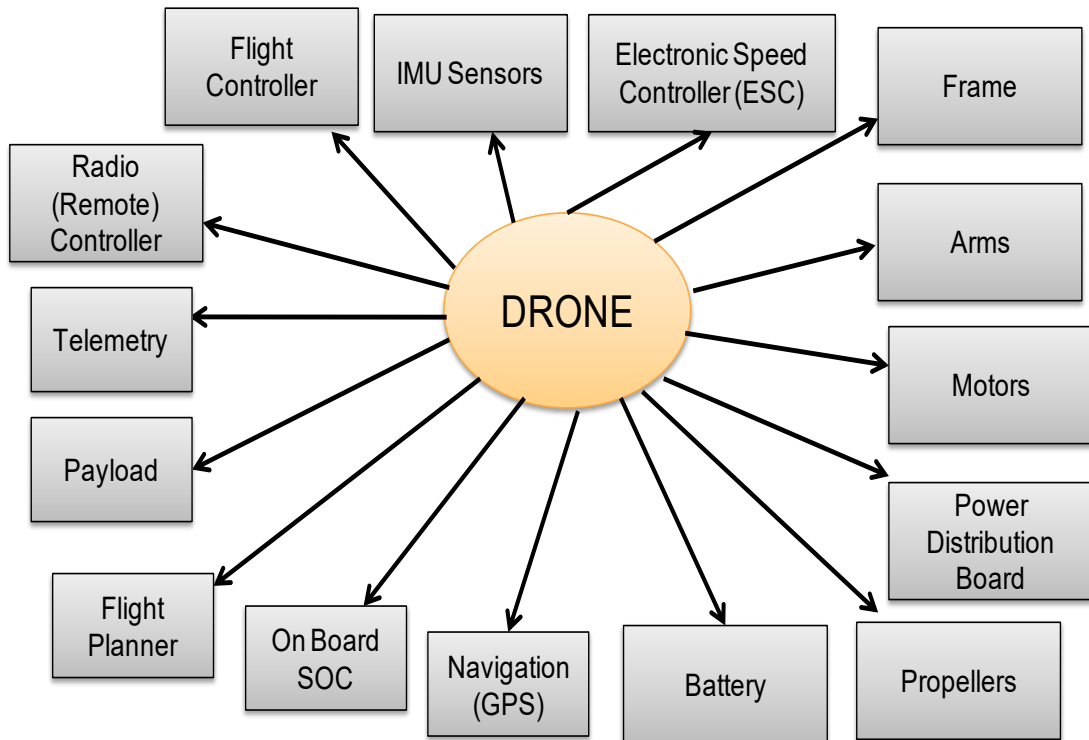


Figure 1.3.1: Drone Anatomy

The specifications of all such components used in integrating and building Drones are described in Chapter 2.

Drones are being manufactured and are being used increasingly in surveying and GIS, Construction, Mining, Agriculture, Insurance and Structural inspection, Search and Rescue operations etc. Though Drones are being used increasingly in aerial surveillance to save time and costs on surveying and mapping projects, but they pose major challenges in customization of payload sensors, stability of the drone platform, imaging and flight planning techniques, collection of the data and processing techniques.

All the Drones being used for several operations are being regulated by DGCA, as described below.

1.4 DGCA regulations

The following are the guide lines issued by DGCA for operating the drones within India.

- Certification of safe and controlled operation of drone hardware and software,
- Air space management through automated operations linked into overall airspace management framework,
- Beyond visual-line-of-sight operations,
- Contribution to establishing global standards,
- Suggestions for modifications of existing RPAs and/or new RPAs

1.5 NPNT Guidelines - NO PERMISSION NO TAKE-OFF.

NPNT requires all manufacturers to implement firmware & hardware changes that only allow flights authorized by DGCA to physically take-off.

No drone will be able to fly without first specifying to the DGCA its:

1. Intended flight envelope
2. Time of flight
3. Pilot credentials

An NPNT compliant drone tries to breach geo-fencing, the in-built software will compel the drone to return-to-home (RTH).

1.6 Literature Survey

The literature survey is majorly conducted in three areas – i) The commercial manufactures/vendors of the drones who offer COTS drone in several drone market segments, which are ready to unbox and use ii) major parts offered by several vendors /third party providers for building the custom drone iii) open source software tools required in building the drone and essentiality of NPNT compliance device.

Drones (Self-flying planes / robotic flight) are becoming as the recent automation of vehicles such as cars and planes promise to fundamentally alter the microelectronics of transporting people and goods, and several collision

avoidances through air traffic control research methods while drones are operated are discussed (Aislan, 2015). Surveyors and GIS professionals are using drone mapping to achieve tremendous time and cost savings on surveying and mapping projects. Time spent on collecting accurate data is vastly reduced, by acquiring data from the sky in the form of geo referenced digital aerial images – surveyors can gather millions of data points on one short flight (An overview of Drone Applications, 2016). The paper also discussed about the several applications to use the drones and several commercial players for the drone manufacturing and service providers. DJI (DJI Drones web link) is one such commercial vendor builds different classes/range of drones to operate in several segments varying from hobby to filming industry. DJI provides several types of drones to operate in several segments of the markets ranging from filming/TV industry to aerial survey etc. One major limitation with all these DJI drones are vendor lock in and prone to security issues, as the flight logs can be transmitted over a network while the flight controller is connected to internet for flight planning. Here, first we describe several major commercial players in the drone manufacturing followed by the several open source and third party software tools used for mapping applications.

Recently DJI has opened up (An overview of Drone Applications, 2016) drones on board software systems, introducing an Application programming Interface (API), which meant third parties could now develop apps to fly and control the aircraft in different ways. This is the key to developers coming up with the flight control Apps specifically for land surveying and mapping. However, these Apps are still vendor lock in and does operate with the GPS signal and the flight logs can be monitored, hence this kind of system care not ideal suited for any security applications.

There are other solution providers, apart from DJI who does the commercial manufacturing of the drones in many areas, offering drones specific to the land mapping field, and few manufactures who could potentially provide viable alternatives such as senseFlyeBee (SenseFly, weblink). SenseFlyeBee a is fixed wing drone designed mainly for larger land areas and agricultural applications. SenseFly is a subsidiary of Parrot, one of the leading toy and hobby makers. eBee has a 40-minute flight time, and can cover areas up to 8 sq. km, using COTS third party cameras from Sony or Canon in a fixed nadir (straight down) position, and

adapted for control by the aircraft in flight software. However, SenseFly operate with the pre mounted camera with the COTS mission planning software, which does not provide an option for putting the camera of our own choice and open source mission planning software.

Yeneec, is another Chinese drone manufacturer, and which is similar to DJI in the professional drone market. However, Yuneec has yet to offer an API for third party developers and at present the focus is primarily the video and film market, with limited applications in land mapping and surveying (Introducing MANTIS Q, weblink).

Trimble Navigation has been producing drones specifically for surveying and aerial data collection for a few years (Trimble, weblink). There are two fixed wing models and one multi rotor model, along with associated processing software packages for producing high resolution Orth mosaics, digital surface models (DSM), and coloured point clouds. However, these drones are highly vendor lock in solutions.

3DR form 3D robotics (3DR, 2014) is another promising solution provider with back end support from Qualcomm, and have two drone models available. Solo quadcopter uses an integrated GoPro Hero 4 Camera, along with their Site Scan application, cloud based post processing of image sets, and integration with Autodesk Recap 360. This is also vendor lock and not suitable for serious mapping applications.

All the above said drone models from major drone companies produce drones which are often highly vendor lock in, prone to security breaches, often locked with the camera models and flight planning software tools. The Drone *Milind-1* is completely built on open standard software protocols, integrated and custom built Octocopter drone with the swappable payload camera. Milind-1's flight controller and mission planning software tools are also based on open standard along with open source software. Hence, it is highly secured and configurable with any third party plug and play components, however, NPNT compliance device is still not yet present with this device. For example, GPS used is a third party component, GAGAN (Inside GNSS, 2016) Satellite Based Augmentation System (SBAS) to

provide precise location with corrections applied to produce more/accurate imaging positional and height accuracies.

The essential hardware/software architecture and tools required for drones' payloads and mission control is described in (Enric, 2006). Drone/Aircraft hardware components include batteries, Gimbals, frame, motors, payloads, IMU sensors, flight controller etc. The parts have been manufactured and sold by several manufactures who in turn indicates the specifications of the items being manufactured for the drones.

However, the literature survey describes, there is a need for development of NPNT compliance device to operate for both COTS and Custom built drone to operate within India as per DGCA CAR regulations. Hence, in this project, we propose to develop such one device as IOT compliance adhering to the open standards.

In the next chapter we describe the proposed framework, several IoT components used in the system development.

2. PROPOSED FRAMEWORK

In this chapter, we describe the proposed framework, several hardware and software components used for the system development. Figure 2.1 describes the proposed system framework, divided into four major components as given below.

r

1. **Drone and Its components** – It contains the drone ready to operate for which NPNT compliance device can be plugged through the standard interface like MAVLINK and other serial interface.
2. **NPNT Compliance Device:** The device proposed to develop which will be interfaced with the Drone.
3. **Digital Sky Platform** – The DGCA platform which regulates and monitor the drone and its geo locations
4. **Map Control** – A graphical user interface that reads the Geo Locations of the corresponding Drone system and presents them on the Back drop of Map.

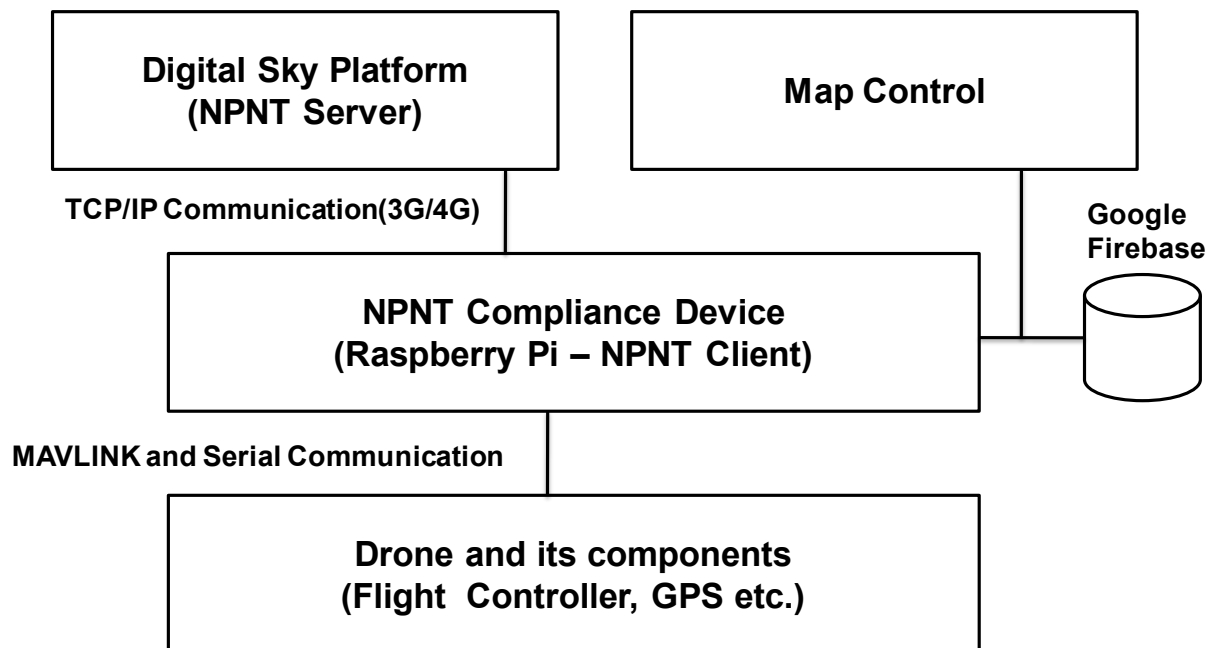


Figure 2.1: Proposed Framework – NPNT Compliance IOT Device

The details components list used in the system development is described below.

2.1 HARDWARE COMPONENTS

2.1.1 Raspberry pi:

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. The Raspberry Pi operates in the open source ecosystem: it runs Linux (a variety of distributions), and its main supported operating system, Raspbian, is open source and runs a suite of open source software. The Raspberry Pi Foundation contributes to the Linux kernel and various other open source projects as well as releasing much of its own software as open source. Some people buy a Raspberry Pi to learn to code, and people who can already code use the Pi to learn to code electronics for physical projects. The Raspberry Pi can open opportunities for you to create your own home automation projects, which is popular among people in the open source community because it puts you in control, rather than using a proprietary closed system. The Raspberry Pi Foundation works to put the power of computing and digital making into the hands of people all over the world. It does this by providing low cost, high-performance computers that people use to learn, solve problems, and have fun. It provides outreach and education to help more people access computing and digital making—it develops free resources to help people learn about computing and making things with computers and also trains educators who can guide other people to learn

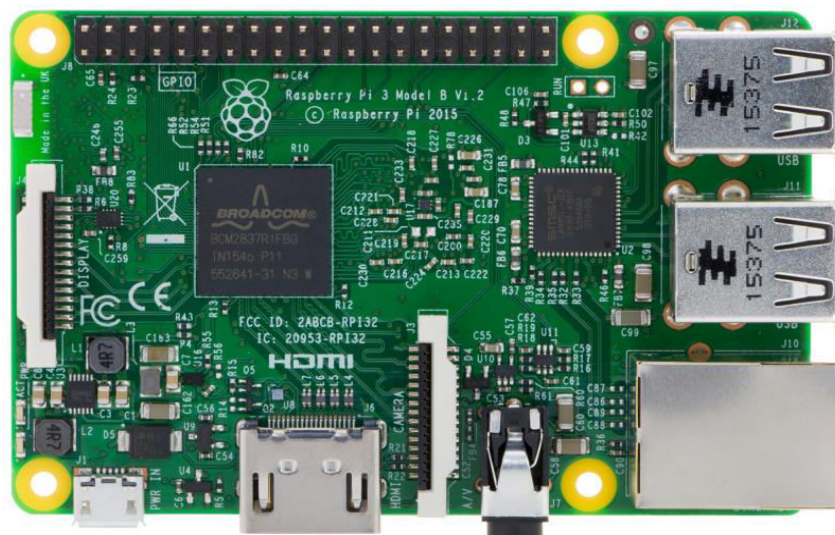


Figure:2.1.1.1Raspberry pi

2.1.1.1. Raspberry Pi Setup:

STEP 1: DOWNLOAD RASPBIAN

The Raspbian OS can be found on Raspberry Pi Foundation's official website <https://www.raspberrypi.org/downloads/>. Download this file.



Figure:2.1.1.1 Raspbian Versions

STEP 2: UNZIP THE FILE

The Raspbian disc image is compressed, so you'll need to unzip it. The file uses the ZIP64 format, so depending on how current your built-in utilities are, you need to use certain programs to unzip it such as WinRAR.

STEP 3: WRITE THE DISC IMAGE TO MICROSD CARD

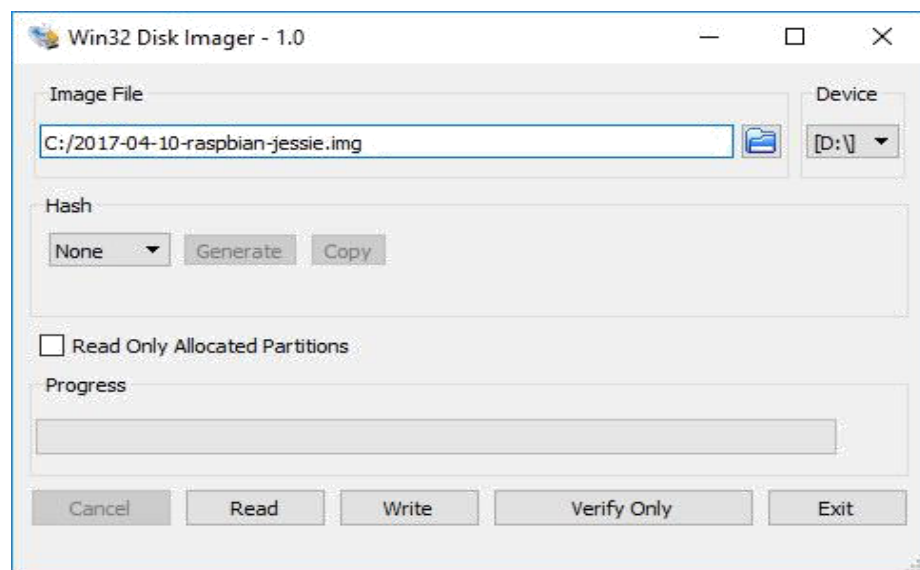


Figure:2.1.1.1.2 Win32 Disk Imager

Next, pop your microSD card into your computer and write the disc image to it. You'll need a specific program to do this. In Windows we use Win32DiskImager. The process of actually writing the image will be slightly different across these programs, but it's pretty self-explanatory no matter what you're using. Each of these programs will have you select the destination (make sure you've picked your microSD card!) and the disc image (the unzipped Raspbian file). Choose, double-check, and then hit the button to write.

STEP 4: PUT MICROSD CARD IN PI AND BOOT UP

Once the disc image has been written to the microSD card, you're ready to go! Put the SD card into your Raspberry Pi, plug in the peripherals and power source. The current edition to Raspbian will boot directly to the desktop. Your default credentials are username **pi** and password **raspberry**. Firstly, we need to update and upgrade the raspberry pi to prevent future problems. This can be done by the following commands:

\$sudo apt-get update

\$sudo apt-get upgrade

Now type sudo raspi-config in terminal:

\$sudo raspi-config

2.1.2. Flight Controller:



Figure:2.1.2.1 Pixhawk

The flight controller is the brain of your drone taking into account the angle of your drone and your control input it calculates how fast the motors should spin and sends the signals to the ESCs. Flight controllers are normally built for certain software such as Beta flight, KISS or Mission Planner so your software choice may affect your decision. In this project we will be using Pixhawk 2.4.8 which is an excellent flight controller.

Pixhawk Overview:

Specifications:

Processor

- 32-bit ARM Cortex M4 core with FPU
- 168 MHz/256 KB RAM/2 MB Flash
- 32-bit failsafe co-processor

Sensors

- MPU6000 as main accel and gyro

- ST Micro 16-bit gyroscope
- ST Micro 14-bit accelerometer/compass (magnetometer)
- MEAS barometer

Power

- Ideal diode controller with automatic failover
- Servo rail high-power (7 V) and high-current ready
- All peripheral outputs over-current protected, all inputs ESD protected

Interfaces

- 5x UART serial ports, 1 high-power capable, 2 with HW flow control
- Spektrum DSM/DSM2/DSM-X Satellite input
- Futaba S.BUS input (output not yet implemented)
- PPM sum signal
- RSSI (PWM or voltage) input
- I2C, SPI, 2x CAN, USB
- 3.3V and 6.6V ADC inputs

Dimensions

- Weight 38 g (1.3 oz)
- Width 50 mm (2.0")
- Height 15.5 mm (.6")
- Length 81.5 mm (3.2")

ABOUT FLIGHT CONTROLLER AND ITS ATTACHMENTS

The final component to mount is the flight controller! **This is the brain of your drone and we will be connecting nearly all of our signal wires here.** The hardest part of wiring the flight controller is knowing what goes where since all flight controllers have a slightly different layout. The very first thing I suggest you do is **search for a pinout diagram of your board**; it should look something like this:



Figure:2.1.2.2 Labelling of Pixhawk

Typically, you will be looking to connect the following wires to their respective pads:

Power –As with all other components we need to power them, almost all flight controllers require 5V however some have their own regulator and will run off battery voltage. Pixhawk requires 5 volts.

Vbat –If your flight controller runs off of 5V it will still need to read the main battery voltage if you want to make use of features such as the OSD or beeper. You will often have a positive and negative wire to do this connecting to the Vbat and ground pads.

Motors –Each of the four motors will have one signal wire (typically white) and one round wire (black). Refer to the motor layout diagram for the order!

Receiver – You’ll have one signal in wire to connect to either an UART RXport or a dedicated SBUS port etc. You may also have a telemetry wire which will connect to a different UART TX!

OSD –If you have an OSD you will have connectors for video in, video out and then grounds for both signals. It is important that you use these grounds for both your camera and VTX if you want clean video. Some extras you could also include could be

Buzzer –This works as a mean to find your lost drone in a crash or to warn you if the battery gets low. Flight controllers typically have a + and – buzzer pad to use here.

LEDs –You can run all kinds of LEDs with all kinds of patterns on your drone which are great for distinguishing your drone whilst racing. LED strips are typically powered by any + and – 5V pads with a signal wire connecting to the flight controller. As with most components I would recommend powering your LEDs off of the PDB if possible.

Pixhawk comes with a variety of attachments which include GPS module, buzzer, switch and much more. The first three components are basic requirements to fly the quad. You'll be having labels on top of the Pixhawk. Each label refers to the component which needs to be connected.

2.1.3 GPS Module

This device is used to calibrate the latitude and longitude coordinates of the flight controller. PX4 supports global navigation satellite systems (GNSS) using receivers that communicate via the UBlox, MTK Ashtech or Emlid protocols, or via UAVCAN. It also supports Real Time Kinematic (RTK) GPS Receivers which extend GPS systems to centimetre-level precision.

PX4 can be used with the following compass parts (magnetometers): Bosch BMM 150 MEMS (via I2C bus), HMC5883 / HMC5983 (I2C or SPI), IST8310 (I2C) and LIS3MDL (I2C or SPI).

Up to 4 internal or external magnetometers can be connected, though only one will actually be used as a heading source. The system automatically chooses the best available compass based on their internal priority (external magnetometers have a higher priority). If the primary compass fails in-flight, it will failover to the next one. If it fails before flight, arming will be denied.



Figure:2.1.3.1 GPS Module

2.2. Software Components:

2.2.1. Mission Planner Overview

Mission Planner is a full-featured ground station application for the ArduPilot open source autopilot project. This page contains information on the background of Mission Planner and the organization of this site.

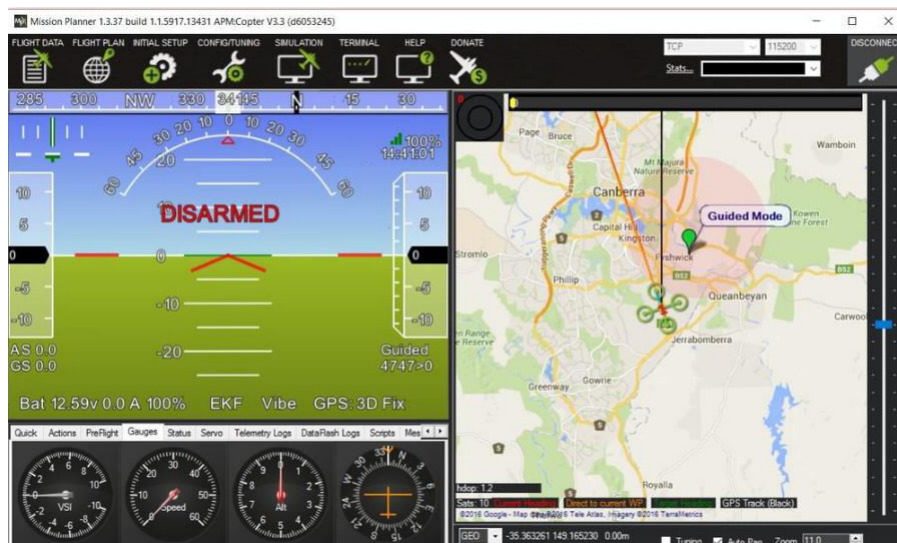


Figure:2.2.1.1 Mission Planner

Mission Planner is a ground control station for Plane, Copter and Rover. It is compatible with Windows only. Mission Planner can be used as a configuration utility or as a dynamic control supplement for your autonomous vehicle. Here are just a few things you can do with Mission Planner:

- Load the firmware (the software) into the autopilot board (i.e. Pixhawk series) that controls your vehicle.

- Setup, configure, and tune your vehicle for optimum performance.
- Plan, save and load autonomous missions into you autopilot with simple point-and-click way-point entry on Google or other maps.
- Download and success mission logs created by your autopilot.
- Interface with a PC flight simulator to create a full hardware-in-the-loop UAV simulator.

With appropriate telemetry hardware you can:

- Monitor your vehicle's status while in operation.
- Record telemetry logs which contain much more information the on-board autopilot logs.
- View and success the telemetry logs.
- View and success the telemetry logs.
- Operate your vehicle in FPV (first person view)

It is a free open source software and can be downloaded and installed from here

“<http://ardupilot.org/planner/docs/mission-planner-installation.html>”

2.2.2 LOADING FIRMWARE

These instructions will show you how to download the latest firmware onto the flight controller using the Mission Planner ground station.

Connect flight controller to computer

Once you've installed a ground station on your computer, connect the flight controller using the micro USB cable as shown below. Use a direct USB port on your computer (not a USB hub).



Figure:2.2.2.1 Mission Planner Setup window.



Figure:2.2.2.2 Pixhawk with USB connection

Windows should automatically detect and install the correct driver software.
Select the **COM** port

If using the Mission Planner select the COM port drop-down on the upper-right corner of the screen (near the **Connect** button). Select **AUTO** or the specific port for your board. Set the Baud rate to **115200** as shown. Don't hit **Connect** just yet.

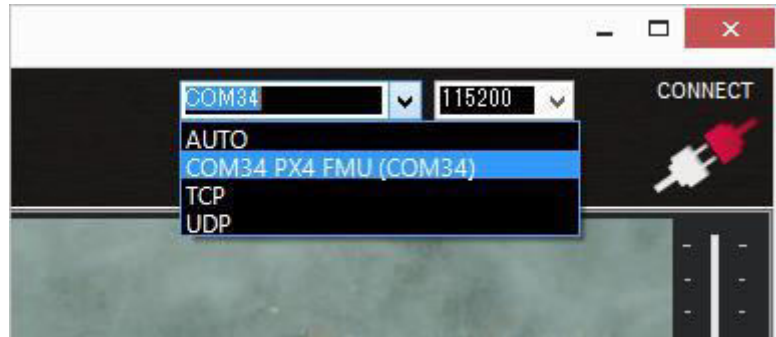


Fig:2.2.2.3 Select COM

Install firmware

On the Mission Planner's **Initial Setup | Install Firmware** screen select the appropriate icon that matches your frame (i.e. Quad, Hexa). Answer **Yes** when it asks you "Are you sure?". Mission Planner: Install FirmwareScreen

After the GCS detects which board you are using it will ask you to unplug the board, plug it back in and then press **OK** within a few seconds (during this brief period the bootloader accepts requests to upload new firmware).

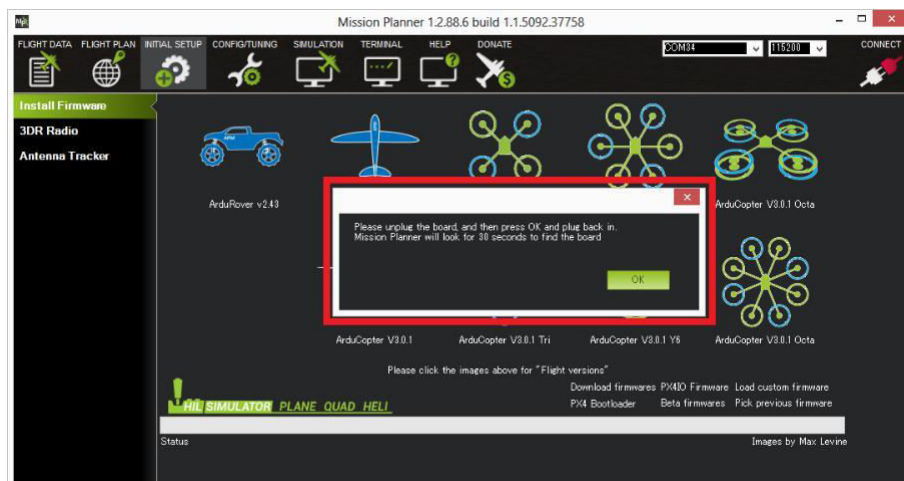


Figure:2.2.2.4 Mission Planner: Install FirmwarePrompt

Next you may be asked "Upload ChibiOS?". Most users will not notice a significant difference regardless of how this is answered.

“Yes” will load ArduPilot using the newer ChibiOS operating system which results in a smaller and more efficient firmware but has a few missing features

“No” will load ArduPilot using the older NuttX operating system.

If all goes well you will see some status appear on the bottom right including the words, “erase...”, “program...”, “verify.” And “Upload Done”. The firmware has been successfully uploaded to the board.

2.2.3. PuTTY:

PuTTY is a terminal emulation program . If you want to manage or configure a device from pc you can use PuTTY for that purpose. It supports SSH, Telnet and Serial ports, so that you can connect directly through serial cable or may take remote session.

Connecting to the raspberry pi using putty:

1. Make sure that SSH is enabled on your raspberry pi
2. Find out the IP Address of raspberry pi using any software (in our case we used advanced IP scanner)
3. Download and install putty
4. Putty window opens

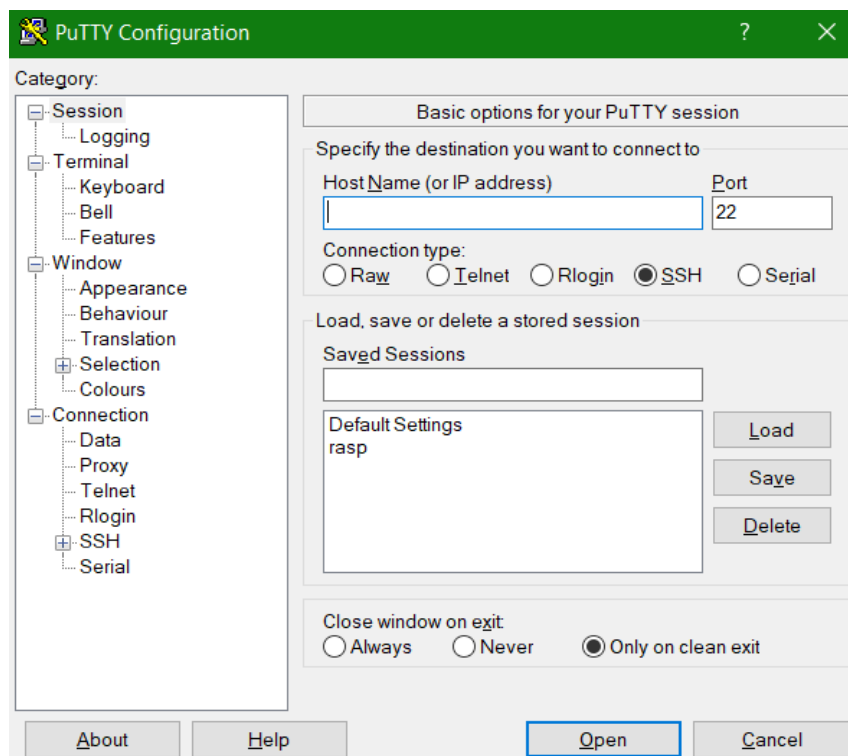


Figure:2.2.3.1 PuTTY Configuration window

5. Enter the Raspberry Pi IP address, select SSH as connection type and under the menu item “Window” → “Translation” select UTF-8

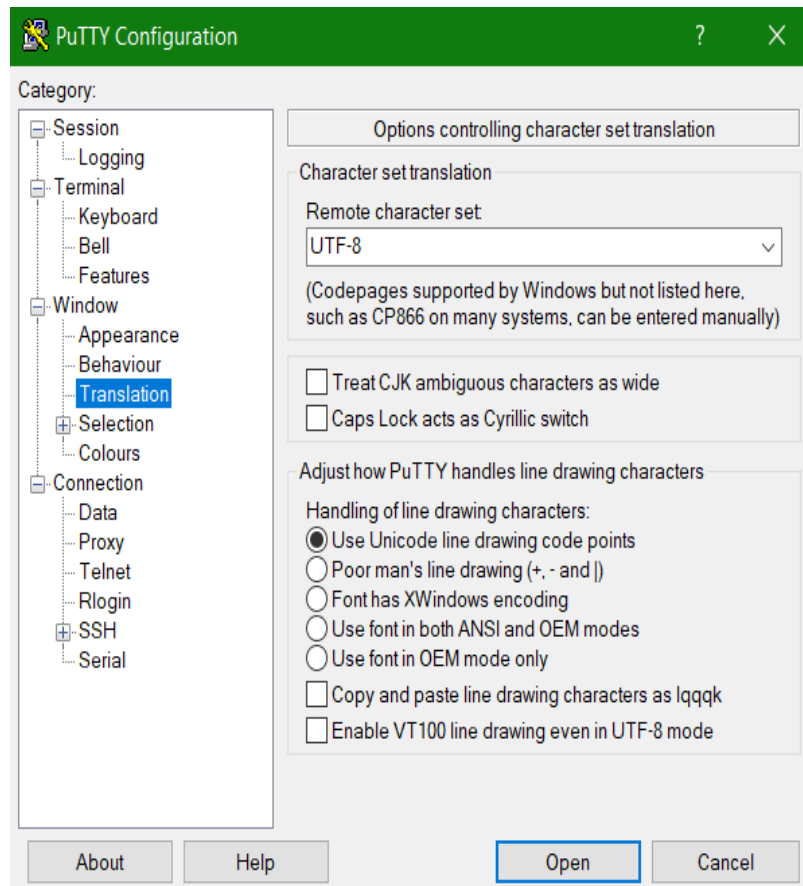


Figure:2.2.3.2 PuTTY window with translation window

After setting up putty we have to enter the IP Address of raspberry pi by checking it properly and can connect to it. Here we are using SSH (secured Socket Shell) connection type and click the open button.

This all we can view in fig.3.2.1.3 like entering IP address and other things. We can also save the sessions by entering our IP Address and save it such that we can load our sessions whenever required.

And whenever we try to open the terminal of raspberry pi by using putty or any other software it asks for a login and password where login is pi

And password is raspberry which were already mentioned. Ans also whenever we want to login into terminal both raspberry pi and our systems should be connected

to same network (Wi-Fi). If you didn't connect your both windows and raspberry pi to the same network then you can't see the login option to enter into terminal.

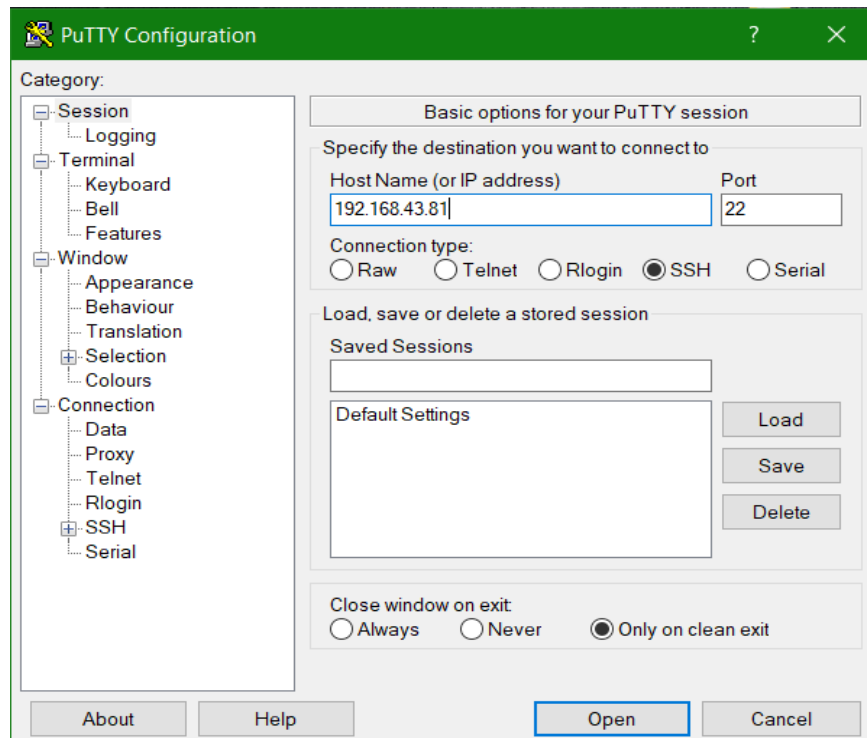


Figure:3.2.1.3 PuTTY configuration window with IP address

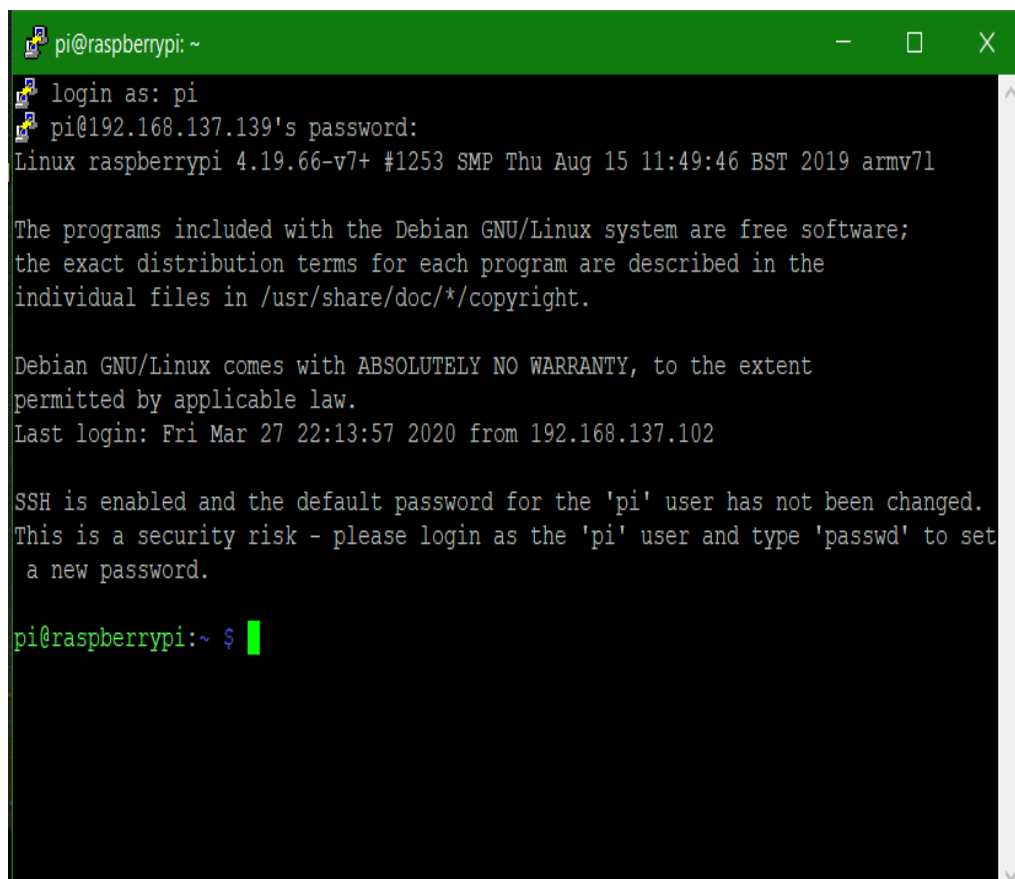


Figure:3.2.1.4 Terminal window of raspberry pi

2.2.4. VNC Viewer

Virtual network computing (VNC) is a type of remote-control software that makes it possible to control another computer over a network connection. Keystrokes and mouse clicks are transmitted from one computer to another, allowing technical support staff to manage a desktop, server, or other networked device without being in the same physical location. In this VNC viewer we can have the desktop view of raspberry pi and also, we can control the raspberry pi using keyboard, mouse etc.

Enabling VNC:

We can enable VNC graphically or at command line.

We are enabling VNC at command line

We can enable VNC at command line using:

\$sudo raspi-config

Now enable VNC server by doing the following:

- Navigate to interfacing options

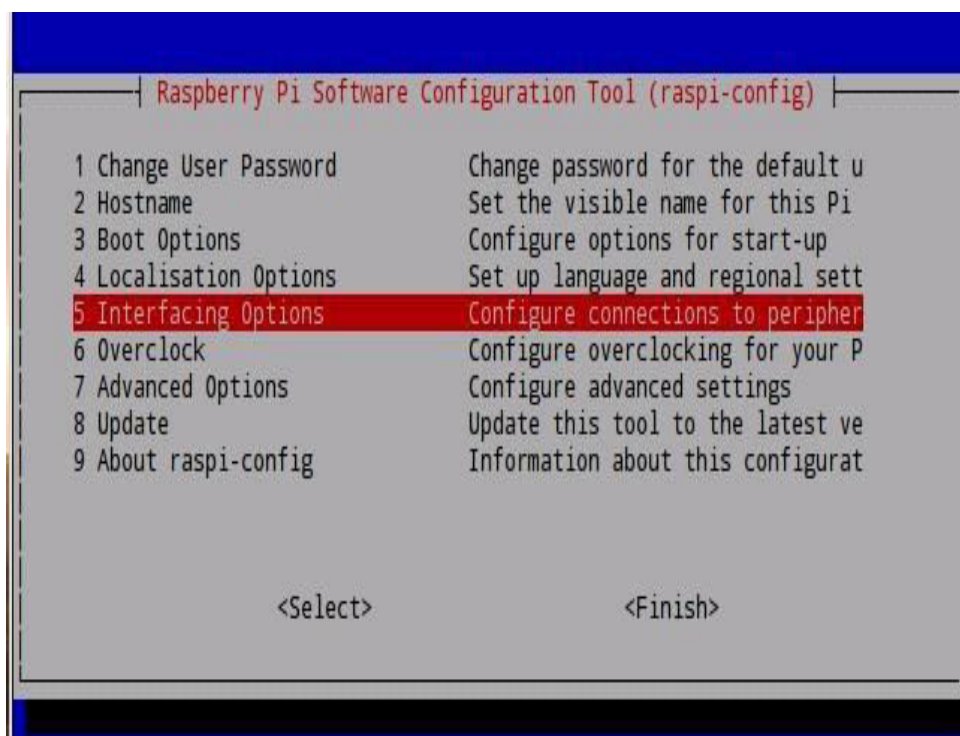


Figure:2.2.4.1 raspberry pi software configuration tool setup

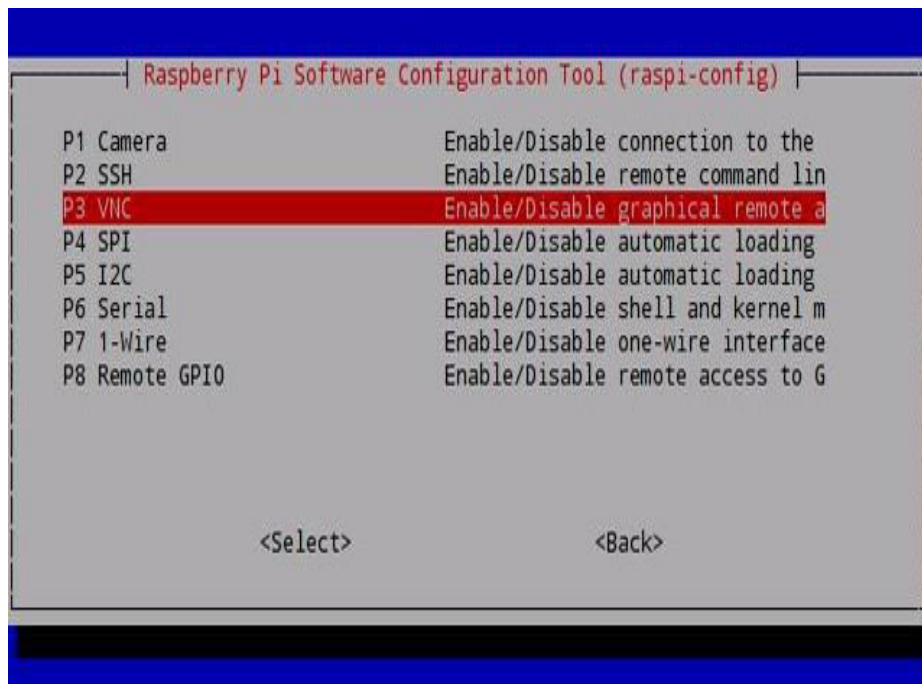


Figure:2.2.4.2 raspberry pi software configuration tool setup

- Scroll down and select VNC > YES

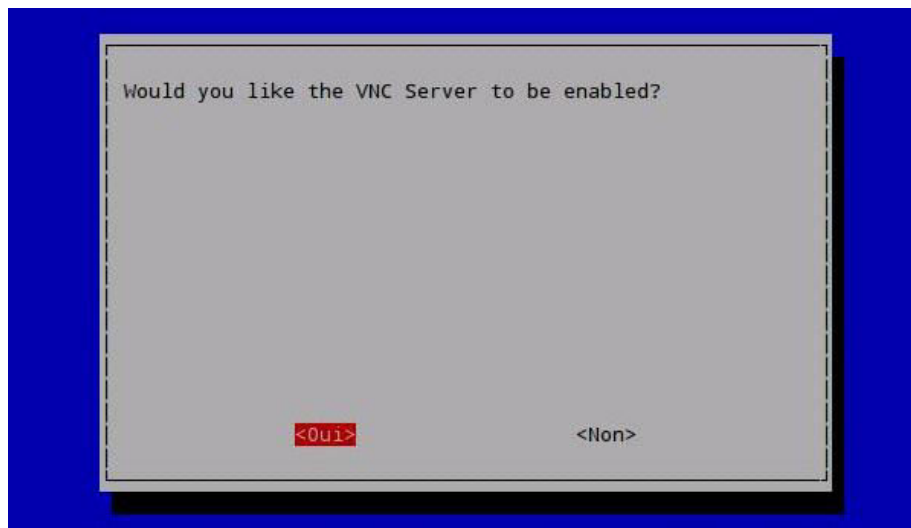


Figure:2.2.4.3 raspberry pi software configuration tool setup

Connecting to your Raspberry Pi with VNC Viewer:

There are two ways to connect to your Raspberry Pi. You can use either or both, depending on what works best for you.

Establishing a direct connection:

Direct connections are quick and simple providing you're joined to the same private local network as your Raspberry Pi. For example, this might be a wired or wireless network at home, at school, or in the office.

On your Raspberry Pi (using a terminal window or via SSH) use these instructions or run `ifconfig` to discover your private IP address.

On the device you'll use to take control, download VNC Viewer. For best results, use the compatible app from RealVNC.

Enter your Raspberry Pi's private IP address into VNC Viewer.

To make the above things to be done firstly we need to have RealVNC account.

So, we have to create a RealVNC account and sign in using those credentials such that RealVNC viewer works.

And also, in order to use VNC viewer we have to type `$vncserver` in raspberry pi command line such that it makes the VNC Viewer works.

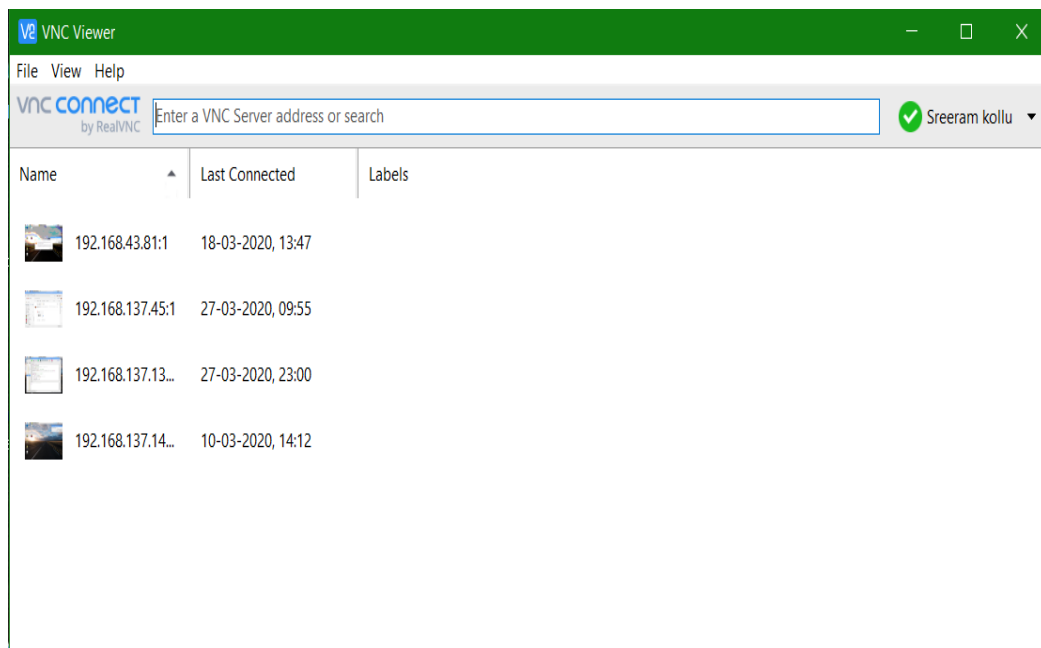


Figure:2.2.4.4 VNC Viewer window

Here we are giving IP Address of raspberry pi in format 192.168.137.134:1 such that it asks for some.

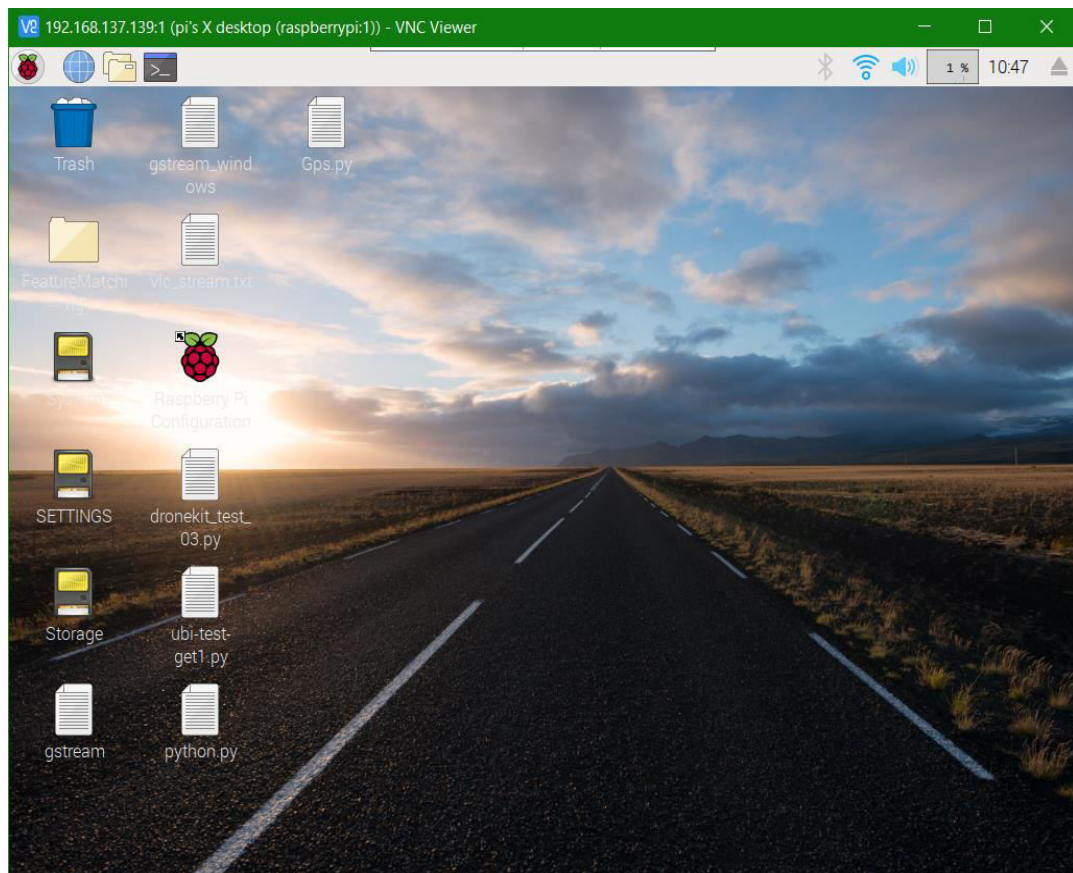


Figure:2.2.4.5 Desktop window of Raspberry pi In VNC viewer

1.4. Hardware Requirements

1. Pixhawk
2. Raspberry pi 3
3. GPS Module

1.5. Software Requirements

1. PuTTY
2. VNC Viewer
3. Map box Studio
4. Firebase
5. PyQt5

3. DESIGN AND IMPLEMENTATION

In this chapter, we present architecture, system flow, use case diagrams and the implementation.

3.1. Architecture Diagram

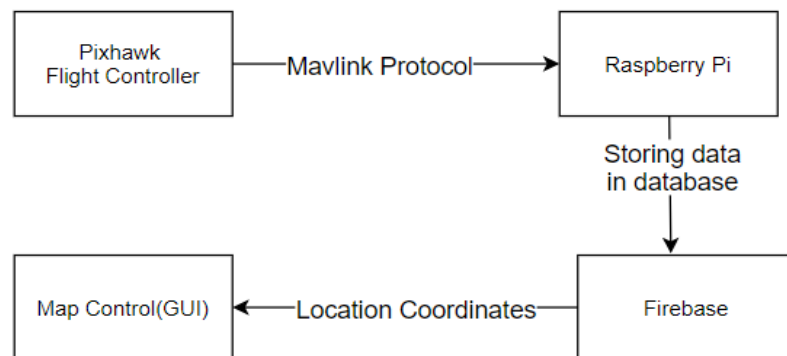


Figure:3.1.1 Architecture Diagram

The flight controller of the Drone, communicates with the IoT device- Raspberry Pi through the USB/Serial communication. An application has been designed which runs on the Pi to read the telemetry data and sends the Geo location information to the Firebase Cloud. The map control would fetch those geo locations to view on the Map Control. The Client/Server application would talk each other via MAVLINK command messaging for Command and Control of the drone.

3.2. Data Flow Diagram:

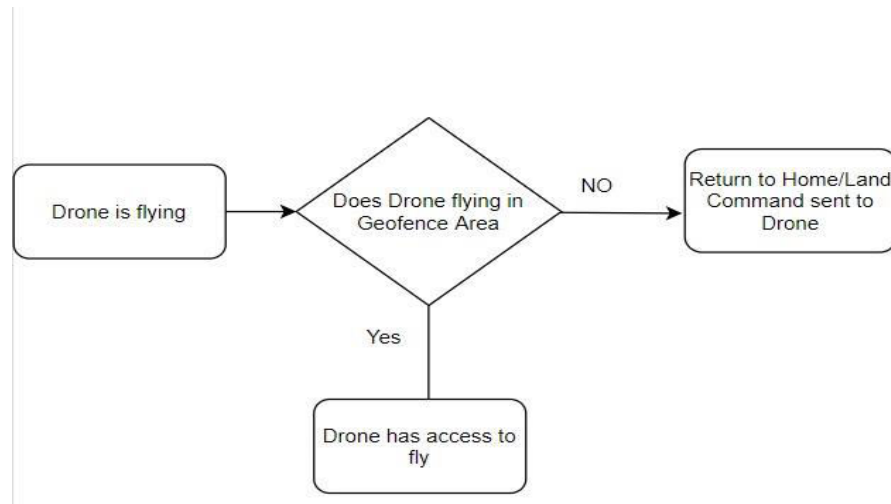


Figure:3.2.1 Flow Diagram

If drone is flying out of Geo-fence area then the server will send Land or Return to Land command to drone so that the drone follows the command and land at the same place or it will return to land.

3.3. Use Case Diagram

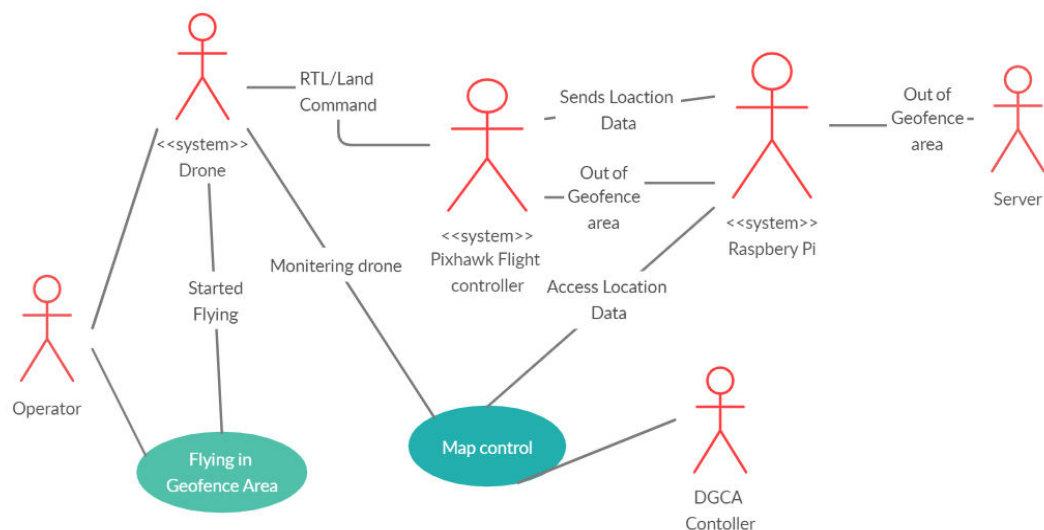


Figure:3.3.1: Use Case Diagram

The use case diagram of the proposed system is depicted in the above picture. Here, the operator operates the drone through the joystick or through the Ground Control Stations, and the respective geo location is sent Raspberry Pi which in turn will forward the location to Google fire base Cloud, then the map control will access the data for viewing. In case if the drone is out of Geofence area the server will send out of geofence area command to Raspberry pi and it will send to Pixhawk and Pixhawk will send RTL/Land command to Drone. Digital Sky platform controls the drone by monitoring them in map control and through the TCP/IP client server application which works over 3G/4G communication network.

3.4. Sequence Diagram:

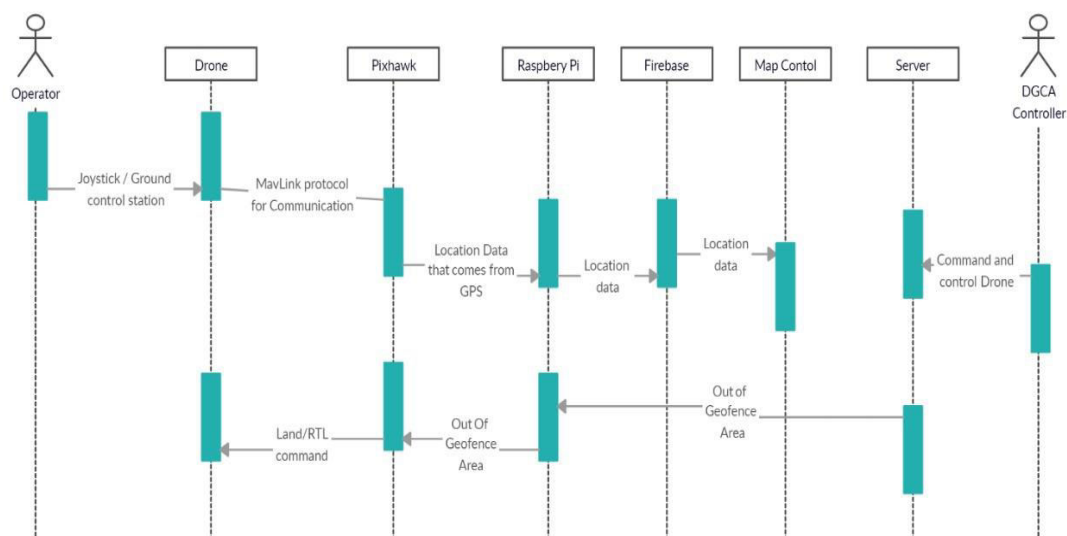


Figure:3.4.1: Sequence Diagram

The operator operates the drone from GPS the location data is sent to Pixhawk and then it is send to Raspberry Pi and from Raspberry Pi it is sent to fire base then the map control will access the data and when the drone comes out of Geofence area the server will send out of geofence area command to Raspberry pi and it will send to Pixhawk and Pixhawk will send RTL/Land command to Drone.

3.5. DRONEKIT:

DroneKit-Python can be used for ground station apps, communicating with vehicles over a higher latency RF-link.

The API communicates 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.

a).SITL

(Note: SITL is used to check some implemented features without a real vehicle)

The SITL (Software in The Loop) simulator allows you to create and test DroneKit-Python apps without a real vehicle (and from the comfort of your own developer desktop!).

SITL can run natively on Linux (x86 architecture only), Mac and Windows, or within a virtual machine. It can be installed on the same computer as DroneKit, or on another computer on the same network.

The sections below explain how to install and run SITL, and how to connect to DroneKit-Python and Ground Stations at the same time.

DroneKit-SITL is the simplest, fastest and easiest way to run SITL on Windows, Linux, or Mac OS X. It is installed from Python's pip tool on all platforms, and works by downloading and running pre-built vehicle binaries that are appropriate for the host operating system.

Steps to show how SITL works:

1). Firstly dronekit-sitl need to maintain connection in mission planner using the command

dronekit-sitl copter --home=78.5660961,17.3302002,0,0

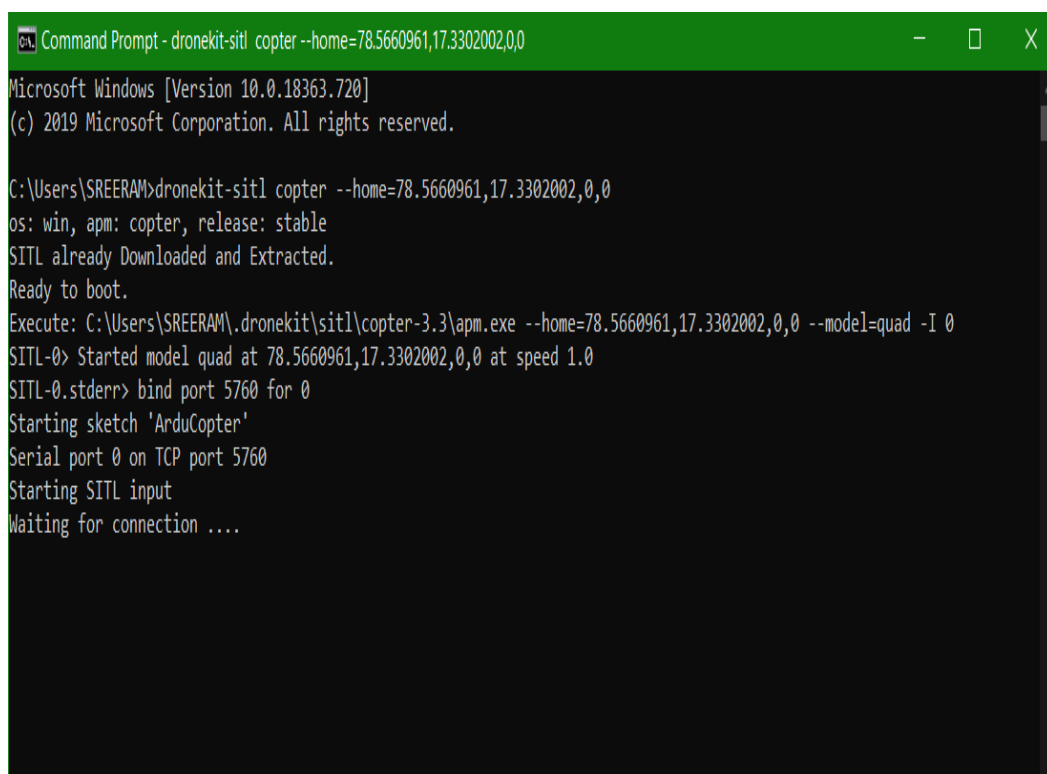
where the numbers(78.5660961,17.3302002) represent the latitude and longitude of the present location (can give any random location).

Now this sitl waits for connection.

Usually this sitl works on TCP protocol where different port numbers will be used where in this case ports like 5760, 5762, 5763 and up to 3ports can be connected.

In this simulation there were things which we connected were mission planner, raspberry pi, and python program which actually tells the information about our drone like at which altitude it is flying, vehicle speed etc.

In our scenario we have given port numbers 5760 to mission planner (which will be explained in next step),5762 to vehicle info python program which can connect to sitl when it was made to run and 5763 to raspberry pi from which commands will be sent to the drone.



```
Command Prompt - dronekit-sitl copter --home=78.5660961,17.3302002,0,0
Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\SREERAM>dronekit-sitl copter --home=78.5660961,17.3302002,0,0
os: win, apm: copter, release: stable
SITL already Downloaded and Extracted.
Ready to boot.
Execute: C:\Users\SREERAM\.dronekit\sitl\copter-3.3\apm.exe --home=78.5660961,17.3302002,0,0 --model=quad -I 0
SITL-0> Started model quad at 78.5660961,17.3302002,0,0 at speed 1.0
SITL-0.stderr> bind port 5760 for 0
Starting sketch 'ArduCopter'
Serial port 0 on TCP port 5760
Starting SITL input
Waiting for connection ....
```

Figure:3.5.1 Initiating simulator and waiting for other connections

2)Open mission planner and select the type of connection required at top right direction from the drop-down. TCP connection need to be selected for maintaining connection with dronekit-sitl.

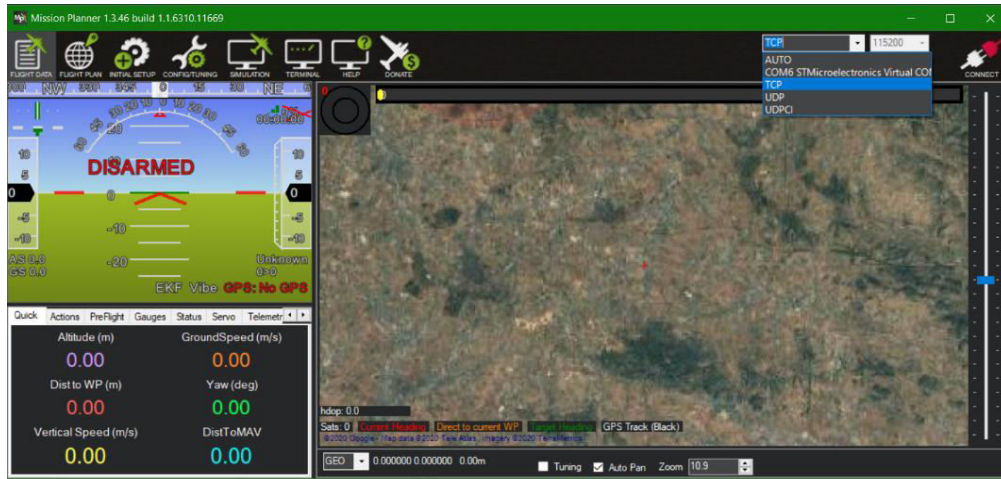


Figure:3.5.2 Selecting the type of connection in mission planner

3) Now host needs to be selected as localhost and port number needs to be required as dronekit contains 3 different types of ports like 5760, 5762, 5763.

In the step 1 we have mentioned which ports are given to what. As we are giving 5760 as port number to this mission planner, mission planner will be connected to sitl with port number 5760.

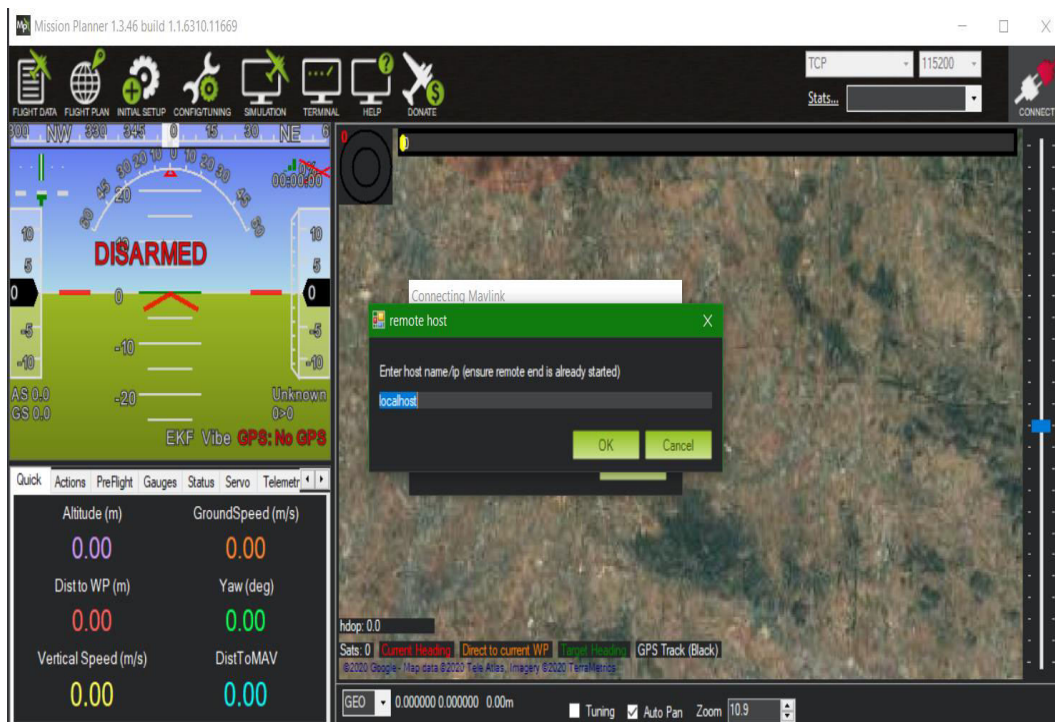


Figure:3.5.3 Mission planner establishing connection with simulation

4) Now after pressing the connect button we can see the drone symbol at the location where we represented latitude and longitude while running sitl.

And also after connecting we can see the yaw, pitch, roll etc... in bottom left side which changes according to the drone situation (Note: In this simulator the values will be of the drone-sitl that is flying, but when we connect the actual Pixhawk everything will be according to the drone).

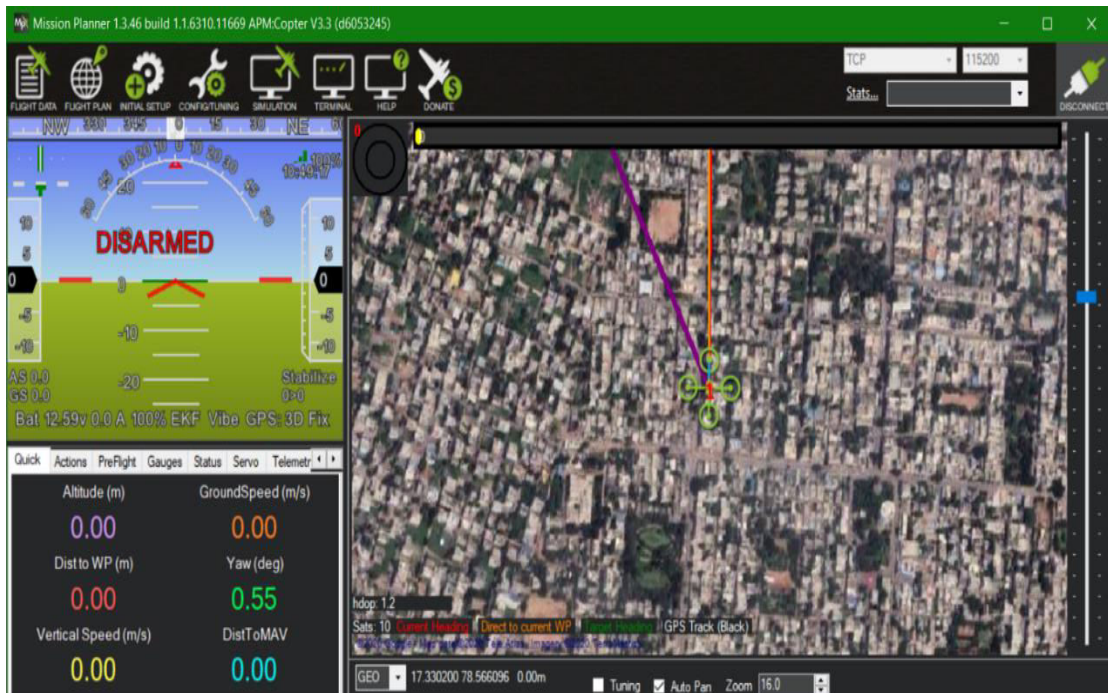


Figure: 3.5.4 Drone symbol after connecting mission planner to simulation

5) Now a program for drone movement needs to be run such that drone movement starts towards the location which will be represented in that program. To run this program successfully port numbers like 5762 or 5763 need to be given so that it gets connected to that simulation(sitl). And in this case, we are running this program in windows system and this can run from anywhere with proper port numbers and also with dronekit, dronekit-sitl etc... which we should import in our program.

And after running this program drone will start moving upwards until it reaches certain altitude as we can see in the map the movement of drone towards the specified location (specified location in the program can be whatever but just used to move the drone to our required direction) after reaching a specific altitude.

And there were certain attributes like Target altitude, pre-arm checks etc... in program which actually checks the motors and different things like arming and when everything got checked the drone will start moving upwards until it reaches certain altitude and in the normal conditions maximum altitude will be around 92 m, so until reaches 92 m it goes on moving upwards.

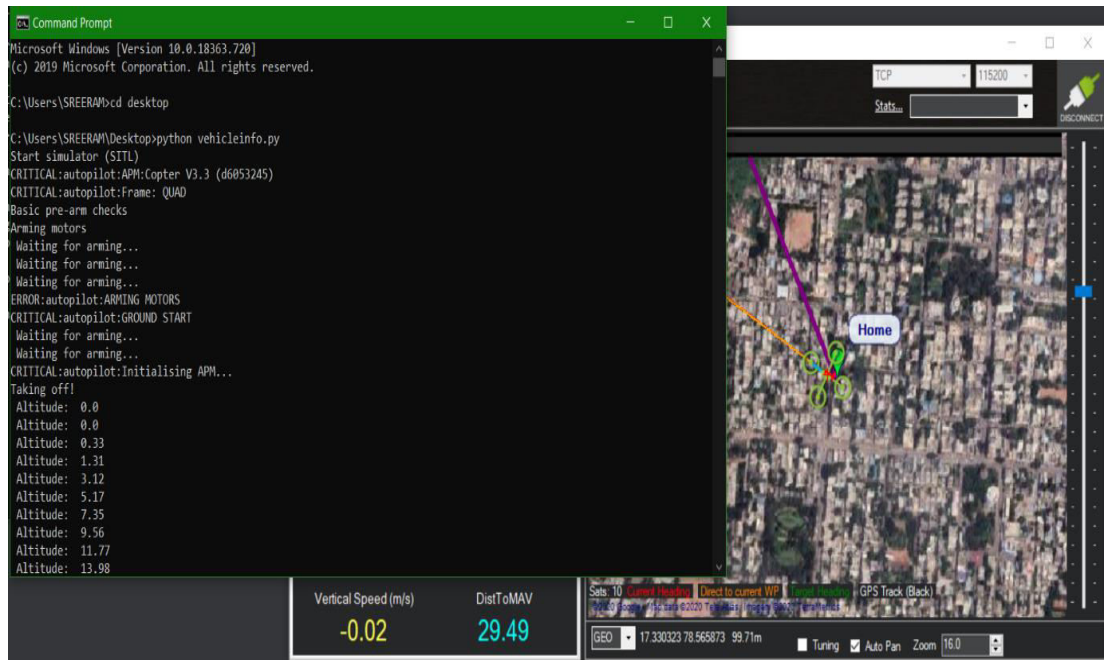


Figure:3.5.5 Starting drone using a python program

6) Now the drone will be moving towards the location which was mentioned in the program with normal speed. We can see the drone movement in map of mission planner and also we can create geofence area in this mission planner which is not our ultimate goal as we need to create this geofence in our own map control but just to test we can create geo fence in this mission planner.

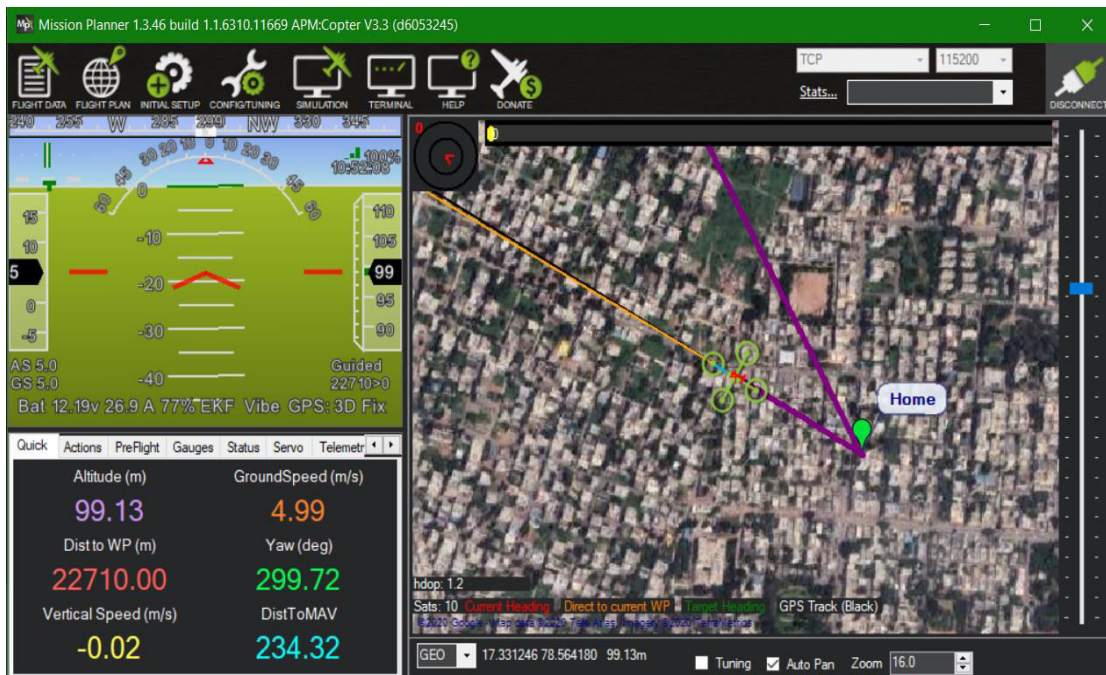


Figure:3.5.6 Drone movement in mission planner

7) Now we are running the client and DGCA server where this DGCA server will be present in our system and client will be present at raspberry pi.

We use this DGCA server program to send commands from our system to raspberry pi to control the drone.

So in this scenario firstly we need to establish connection between raspberry pi and simulation. So in the program client which was present in raspberry pi those TCP connect command and also port number 5763 as well as our system's IP address and raspberry pi's IP address which acts as host need to be mentioned in order to establish connection. Firstly, we are running the client program in order to establish connection with the simulation and after connecting we are printing some random statement so at that time we will be running DGCA Server program which was established in our system which actually sends commands to the client (raspberry pi). Here we are having two commands namely RTL(Return To Land) as well as Land such that RTL brings the drone to the location which was started and LAND command lands the drone wherever it is flying.

```

pi@raspberrypi:~$ vncserver
New 'X' desktop is raspberrypi:1
Starting applications specified in /home/pi/.vnc/xstartup
Log file is /home/pi/.vnc/raspberrypi:1.log

pi@raspberrypi:~$ python server.py
CRITICAL:autopilot:Frame: QUAD
WARNING:dronekit:Link timeout, no heartbeat in last 5 seconds
listening for command
('Got connection from', ('192.168.137.102', 60386))
LAND
WARNING:dronekit:Link timeout, no heartbeat in last 5 seconds

```

```

Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\SREERAM>python client.py
python: can't open file 'client.py': [Errno 2] No such file or directory

C:\Users\SREERAM>cd desktop
C:\Users\SREERAM\Desktop>python client.py
C:\Users\SREERAM\Desktop>

```

Figure:3.5.7 Client-Server communication between windows system and raspberry pi

8) In the above program we are sending command namely RTL and LAND such that whenever we send RTL command to raspberry pi it will make drone to return to the starting location and land there. And if we send LAND command it will be landing at the same place where it receives the command.

In the below picture LAND command is been sent to the drone and drone got landed at some location.

So, the main idea of doing all this is to make sure that sending commands from our system works or not because we are sending commands from our system to raspberry pi and raspberry pi controls the drone using Pixhawk

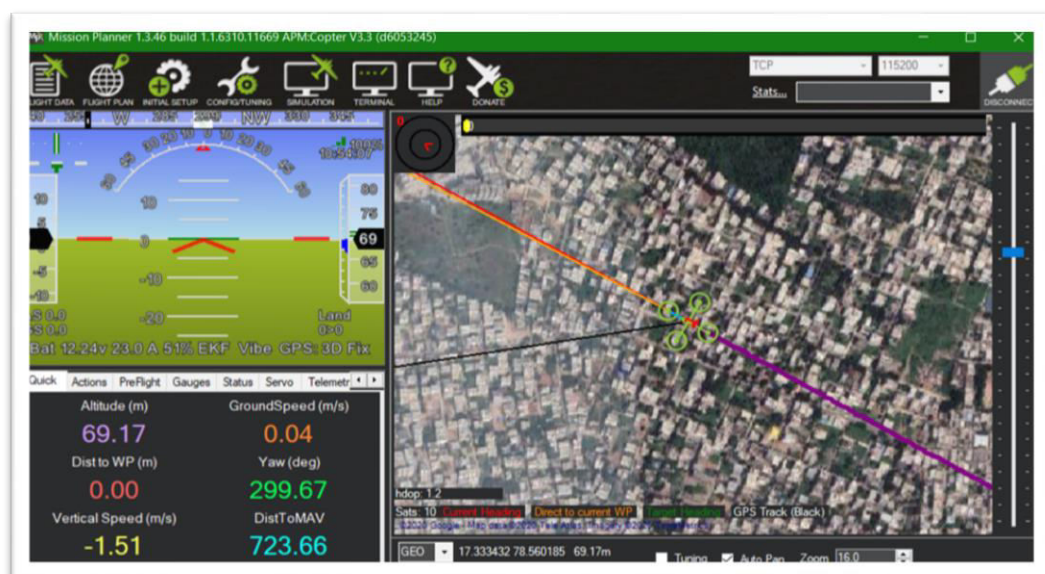


Figure:3.5.8 Drone landing

Implementation code:**a). Code which gets the data of Pixhawk and GPS module
droneinfo.py**

```
from dronekit import connect, VehicleMode
import time
import dronekit_sitl
from firebase import firebase
connection_string = "/dev/ttyACM0"
baud_rate = 115200
from datetime import datetime
now=datetime.now()

#--- Now that we have started the SITL and we have the connection string
(basically the ip and udp port)...

print(">>> Connecting with the UAV <<<")
vehicle = connect(connection_string,baud=baud_rate, wait_ready=True) #-
wait_ready flag hold the program untill all the parameters are been read (=, not .)

#-- Read information from the autopilot:
#- Version and attributes
vehicle.wait_ready('autopilot_version')
firebase = firebase.FirebaseApplication('https://npnt-acf2d.firebaseio.com/')

print('Autopilot version: %s'%vehicle.version)

#- Does the firmware support the companion pc to set the attitude?
print('Supports set attitude from companion:
%s'%vehicle.capabilities.set_attitude_target_local_ned)

#- Read the actual position

print('Latitude: %s'% vehicle.location.global_relative_frame.lat)
```

```
print('Longitude: %s'% vehicle.location.global_relative_frame.lon)
```

```
print('Altitude: %s'% vehicle.location.global_relative_frame.alt)
```

```
#- Read the actual attitude roll, pitch, yaw
```

```
print('Attitude: %s'% vehicle.attitude)
```

```
#- Read the actual velocity (m/s)
```

```
print('Velocity: %s'%vehicle.velocity) #- North, east, down
```

```
print('time: %s' %now)
```

```
#- When did we receive the last heartbeat
```

```
print('Last Heartbeat: %s'%vehicle.last_heartbeat)
```

```
#- Is the vehicle good to Arm?
```

```
print('Is the vehicle armable: %s'%vehicle.is_armable)
```

```
#- Which is the total ground speed? Note: this is settable
```

```
print('Groundspeed: %s'% vehicle.groundspeed) #(%)
```

```
#- What is the actual flight mode? Note: this is settable
```

```
print('Mode: %s'% vehicle.mode.name)
```

```
#- Is the vehicle armed Note: this is settable
```

```
print('Armed: %s'%vehicle.armed)
```

```
#- Is the state estimation filter ok?
```

```
print('EKF Ok: %s'%vehicle.ekf_ok)
```

```
#----- Adding a listener
```

```
#-- dronekit updates the variables as soon as it receives an update from the UAV
```

```
#-- you can define a callback function for predefined messages or define one for
```

```
#-- any mavlink message
```



```

@vehicle.on_message('SYSTEM_TIME')
def listener(slef, name, message):
    print(message.time_unix_usec)

def attitude_callback(self, attr_name, value):
    print(vehicle.attitude)

print("")
print("Adding an attitude listener")
vehicle.add_attribute_listener('attitude', attitude_callback) #-- message type,
callback function
time.sleep(5)
#--- Now we print the attitude from the callback for 5 seconds, then we remove
the callback
vehicle.remove_attribute_listener('attitude', attitude_callback) #(.remove)

#--- You can create a callback even with decorators, check the documentation out
for more details

#---- PARAMETERS
print("Maximum Throttle: %d"%vehicle.parameters['THR_MIN'])

#-- You can read and write the parameters
vehicle.parameters['THR_MIN'] = 50
time.sleep(1)
print("Maximum Throttle: %d"%vehicle.parameters['THR_MIN'])
result = firebase.put("/test pos", "position:", 200)
print(result)
vehicle.close()
print("done")

```

b).Code which tells the info of drone.

Vehicleinfo.py

```
import time
import dronekit_sitl

from dronekit import connect,
VehicleMode,LocationGlobal,LocationGlobalRelative
def arm_and_takeoff(aTargetAltitude):
    """
    Arms vehicle and fly to aTargetAltitude.
    """
    print ('Basic pre-arm checks')
    # Don't try to arm until autopilot is ready
    while not vehicle.is_armable:
        print ( ' Waiting for vehicle to initialise...')
        time.sleep(1)
    print ('Arming motors')
    # Copter should arm in GUIDED mode
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True
    # Confirm vehicle armed before attempting to take off
    while not vehicle.armed:
        print ( ' Waiting for arming...')
        time.sleep(1)
    print ('Taking off!')
    vehicle.simple_takeoff(aTargetAltitude) # Take off to target altitude
    # Wait until the vehicle reaches a safe height before processing the goto
    (otherwise the command
    # after Vehicle.simple_takeoff will execute immediately).
    while True:
        print ( ' Altitude: ', vehicle.location.global_relative_frame.alt)
        #Break and return from function just below target altitude.
        if vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95:
            print ('Reached target altitude')
```

```

        break
    time.sleep(1)
print ('Start simulator (SITL)')
#sitl = dronekit_sitl.start_default()
vehicle = connect('tcp:127.0.0.1:5762', wait_ready=True)
#vehicle.airspeed = 5 #m/s
#vehicle.groundspeed = 10 #m/s
arm_and_takeoff(100)
#a_location = LocationGlobalRelative(-34.364114, -149.166022,30)
a_location = LocationGlobalRelative(17.4325600, 78.3786881,30)
vehicle.simple_goto(a_location,groundspeed=10)
print ('Groundspeed: %s' % vehicle.groundspeed)

```

3.6. MAVLINK

We used Mavlink protocol to establish communication between Raspberry Pi and Pixhawk so the two devices can communicate and send commands.

The micro air vehicle link (MAVLINK in short) is a communication protocol for unmanned systems (e.g., drones and robots). It specifies a comprehensive set of messages exchanged between unmanned systems and ground stations. This protocol is used in major autopilot systems, mainly ArduPilot and PX4(Pixhawk), and provides good features for not only for monitoring and controlling unmanned systems missions but also for their integration into the Internet.

To install mavlink in raspberry pi following commands need to be executed..

```
$sudo apt-get install python3-dev python3-opencv python3-wxgtk3.0 libxml2-dev python3-pip python3-matplotlib python3-lxml
```

```
$sudo pip3 install future
```

```
$sudo pip3 install pymavlink
```

```
$sudo pip3 install mavproxy
```

```
sudoraspi-config
```

Configure the serial port (UART)

Use the Raspberry Pi configuration utility for this.

Type:

And in the utility, select “Interfacing Options”:

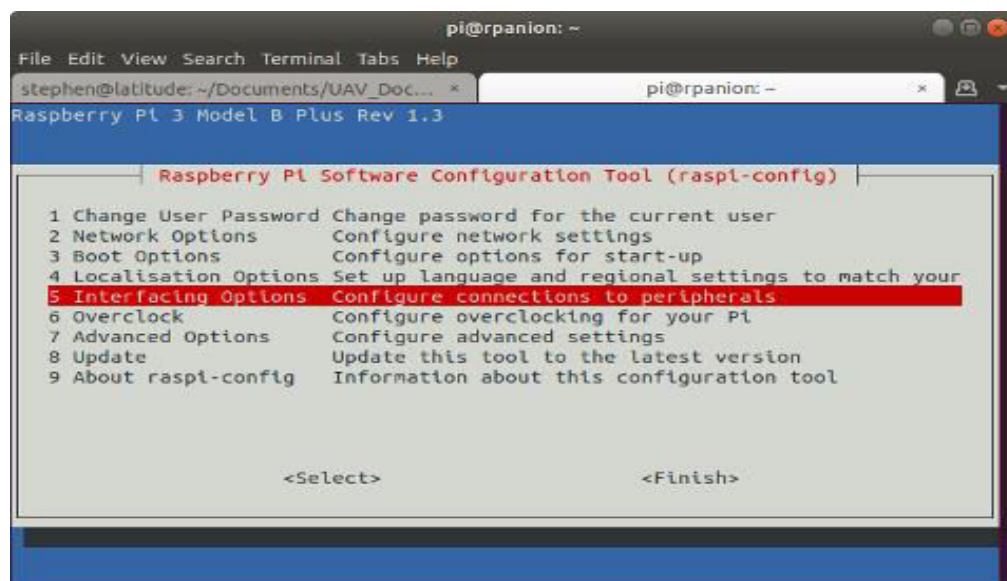


Figure:3.6.1 Raspberry Pi Configuration Utility

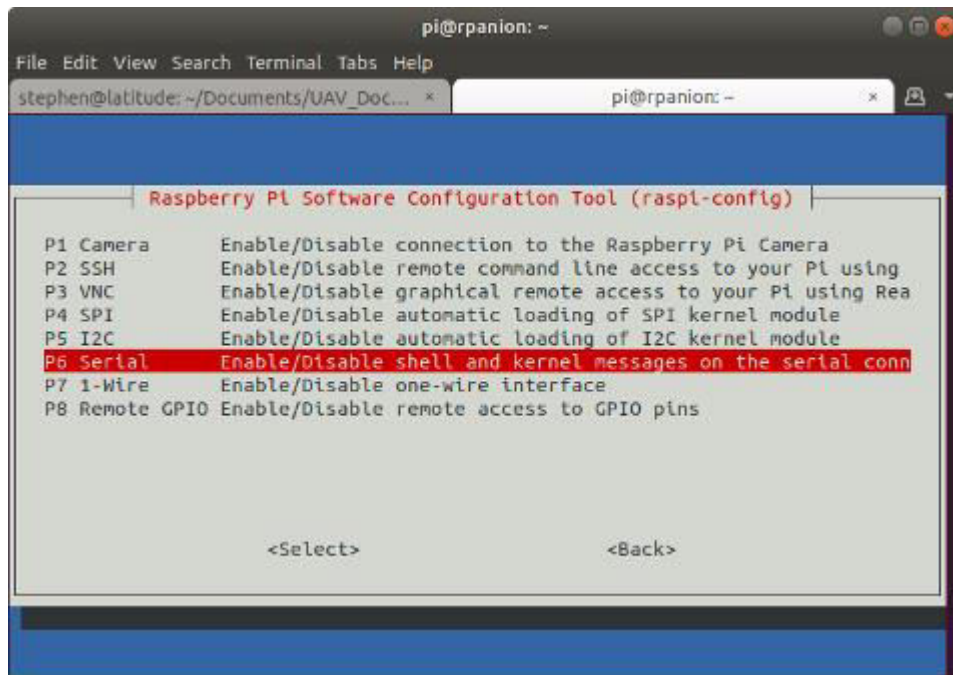


Figure:3.6.2 And then “Serial”

When prompted, select “No” to “Would you like a login shell to be accessible over serial?”.

When prompted, select “Yes” to “Would you like the serial port hardware to be enabled?”.

Reboot the Raspberry Pi when you are done.

3.7. FIREBASE

We used Firebase to store location data that is sent from Raspberry pi.

Firebase provides a real-time database and back-end as a service. The service provides application developers an API that allows application data to be synchronized across clients and stored on Firebase's cloud. The database is also accessible through a REST API and bindings for several JavaScript frameworks such as Angular.js, React, Ember.js and Backbone.js. The REST API uses the Server-Sent Events protocol, which is an API for creating HTTP connections for receiving push notifications from a server. Developers using the real-time database can secure their data by using the company's server-side-enforced security rules.

Firebase Realtime Database and Cloud Firestore provide database services. I listed them both as “Realtime, cloud hosted, NoSQL databases”. They have individual strengths and weaknesses, and you may need to do some research to figure out which one is better for your needs. Hint: start with Cloud Firestore, as it likely addresses more of your needs (and it’s also massively scalable). You can use either one, or both together, if that suits your application.

Here we are using Real-time database where we store data which we are sending from raspberry pi. So the actual data which we are sending from raspberry pi is location, time, altitude and velocity and it will get stored in database in required format.

The data which we are sending to database will be accessed by map control.

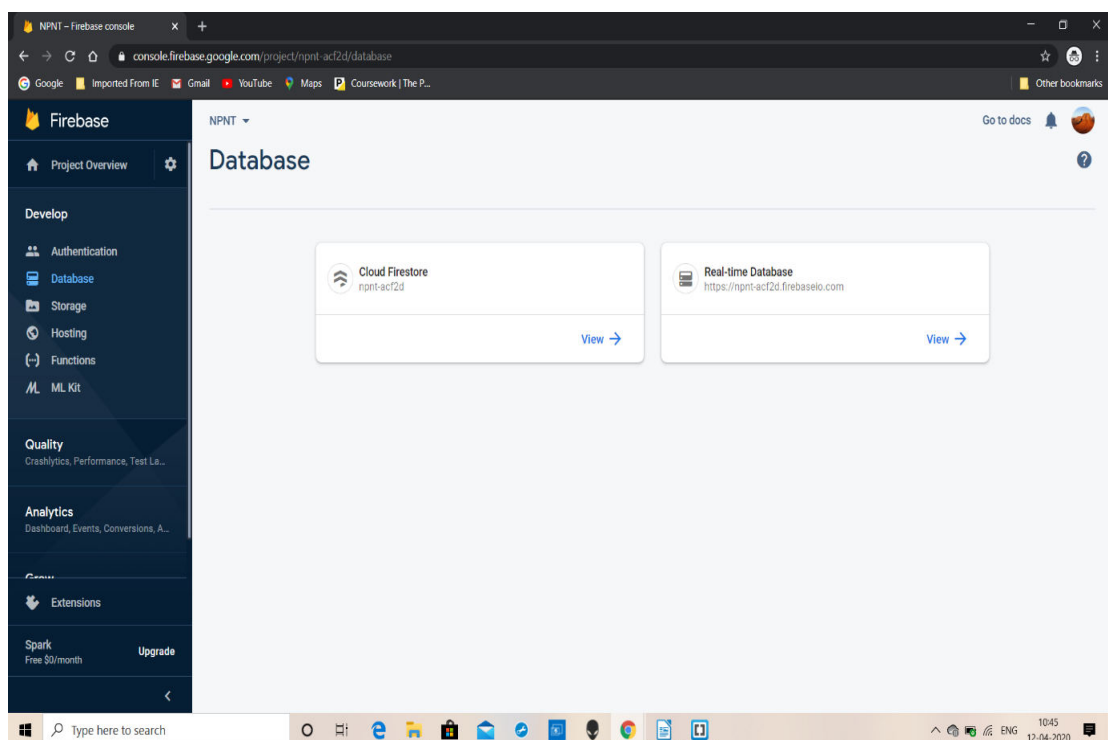


Figure:3.7.1 different types of databases in firebase

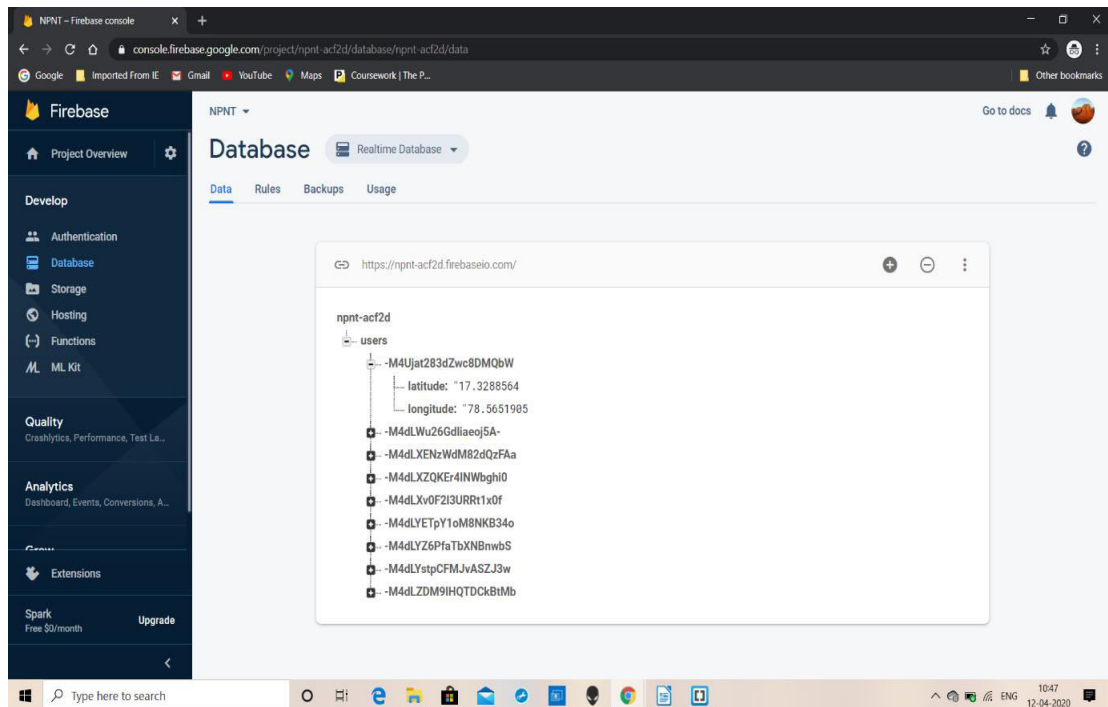


Figure: 3.7.2 location data sent from raspberry pi

Implementation code:

Code used to send data from raspberry pi to firebase

Firestore_test.py

```
from firebase import firebase
from dronekit import connect, VehicleMode
import time
import dronekit_sitl

firebase = firebase.FirebaseApplication('https://npnt-acf2d.firebaseio.com/',None)

connection_string = "/dev/ttyACM0"
baud_rate = 115200

from datetime import datetime
now=datetime.now()
```

#--- Now that we have started the SITL and we have the connection string
(basically the ip and udp port)...

```
print(">>> Connecting with the UAV <<<")
vehicle = connect(connection_string,baud=baud_rate, wait_ready=True) #-
wait_ready flag hold the program untill all the parameters are been read (=, not .)
```

#-- Read information from the autopilot:

#- Version and attributes

```
vehicle.wait_ready('autopilot_version')
```

while True:

```
    latitude = str(vehicle.location.global_relative_frame.lat)
```

```
    longitude = str(vehicle.location.global_relative_frame.lon)
```

```
    altitude = str(vehicle.location.global_relative_frame.alt)
```

```
    time = str(now)
```

```
    result = firebase.post('/users/', {'latitude':latitude,'longitude':longitude})
```

```
    #result = firebase.post('/users',{ "type":"feature",
```

```
        #           "geometry":{
```

```
        #           "type":"point",
```

```
        #           "coordinates":[latitude,longitude]} })
```

```
    #result1 = firebase.post('/time/', time)
```

```
    #result2 = firebase.post('/velocity/', vehicle.velocity)
```

```
    print(result)
```

```
    #print(result1)
```

```
    #print(result2)
```


3.8. MAP CONTROL:

Mapbox is the location data platform for mobile and web applications. It provides building blocks to add location features like maps, search, and navigation into any experience you create.

Mapbox provides a Maps SDK for iOS and Android for publishing user maps in native applications. The Maps SDKs for iOS and Android are designed to be drop-in replacements for Apple's MapKit and the Google Maps SDKs. The Maps SDKs should be familiar to mobile developers who have experience with either. Often, your maps can be swapped for Mapbox by changing a single line of code.

Using mapbox studio we can:

1. Create custom maps
2. Upload custom data as tilesets
3. Create data visualizations
4. Draw data in the dataset editor

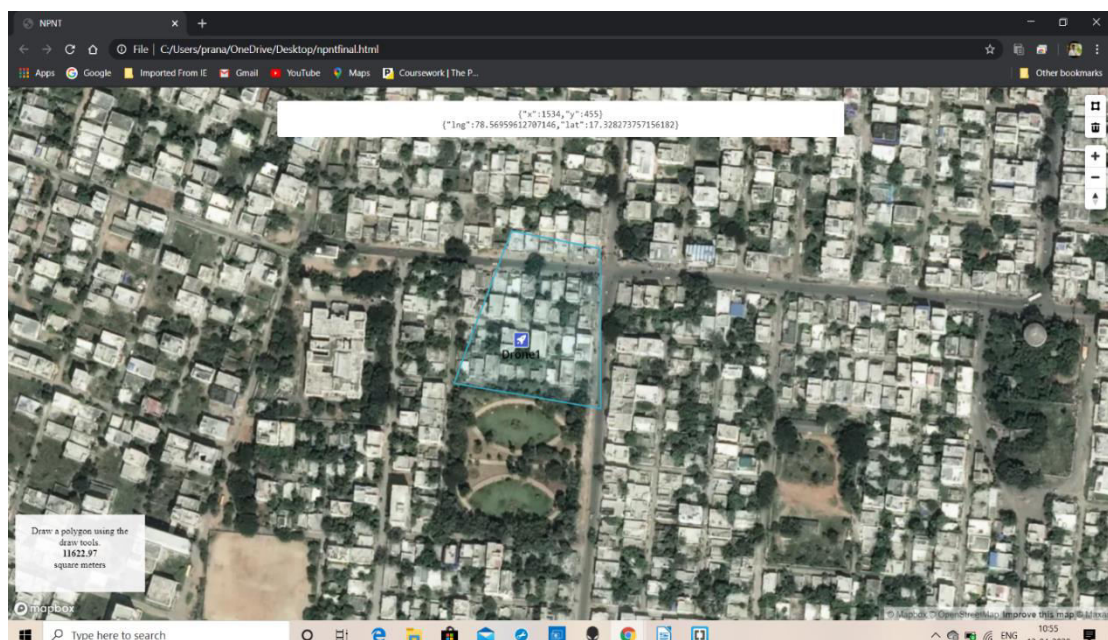


Figure:3.8.1 Creating Geofence area in Mapbox API

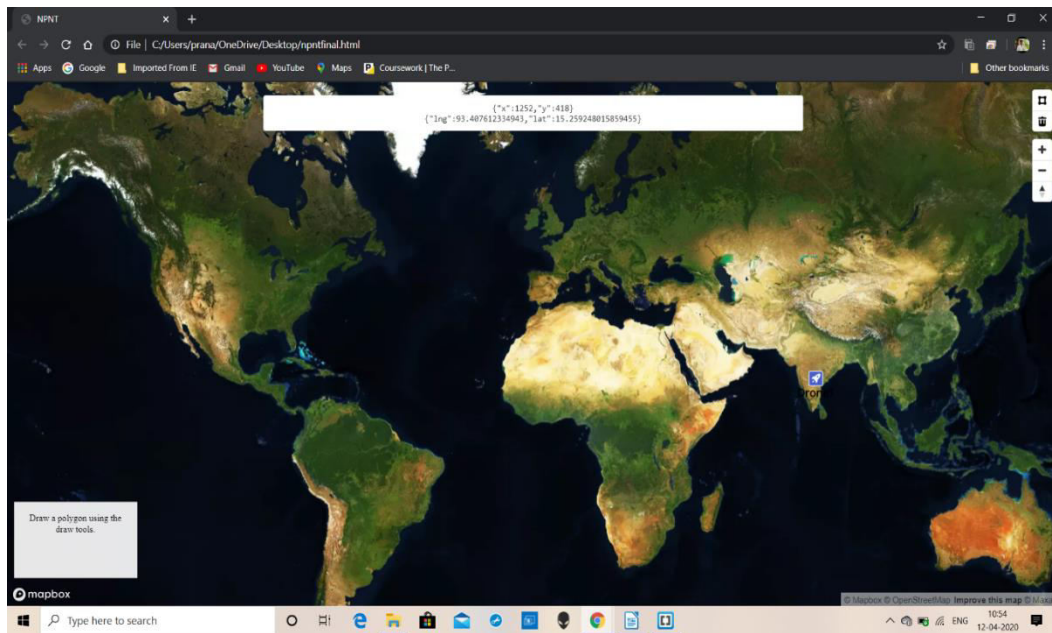


Figure:3.8.2 Map Accessing location data and showing it

Implementation code (HTML):

```
<!DOCTYPE html>
<html>

<head>
<meta charset="utf-8" />
<title>NPNT</title>
<meta name="viewport" content="initial-scale=1,maximum-scale=1,user-
scalable=no" />
  //mapbox
<script src="https://api.mapbox.com/mapbox-gl-js/v1.9.0/mapbox-
gl.js"></script>
<link href="https://api.mapbox.com/mapbox-gl-js/v1.9.0/mapbox-gl.css"
rel="stylesheet" />
<style>
  body {
    margin: 0;
    padding: 0;
  }
```

```

    #map {
        position: absolute;
        top: 0;
        bottom: 0;
        width: 100%;
    }
</style>
//firebase scripts
<script defer src="https://www.gstatic.com/firebasejs/7.13.2/firebase-
app.js"></script>
<script defer src="https://www.gstatic.com/firebasejs/7.13.2/firebase-
auth.js"></script>
<script defer src="https://www.gstatic.com/firebasejs/7.13.2/firebase-
firestore.js"></script>
<script src="https://www.gstatic.com/firebasejs/live/3.0/firebase.js"></script>

</head>

<body>
<style type="text/css">
    #info {
        display: block;
        position: relative;
        margin: 0px auto;
        width: 50%;
        padding: 10px;
        border: none;
        border-radius: 3px;
        font-size: 12px;
        text-align: center;
        color: #222;
        background: #fff;
    }
</style>

```

```

<style>
    .calculation-box {
        height: 75px;
        width: 150px;
        position: absolute;
        bottom: 40px;
        left: 10px;
        background-color: rgba(255, 255, 255, 0.9);
        padding: 15px;
        text-align: center;
    }
    p {
        font-family: 'Open Sans';
        margin: 0;
        font-size: 13px;
    }
</style>
<script
src="https://api.tiles.mapbox.com/mapbox.js/plugins/turf/v3.0.11/turf.min.js"></s
cript>
<script src="https://api.mapbox.com/mapbox-gl-js/plugins/mapbox-gl-
draw/v1.0.9/mapbox-gl-draw.js"></script>
<link rel="stylesheet" href="https://api.mapbox.com/mapbox-gl-
js/plugins/mapbox-gl-draw/v1.0.9/mapbox-gl-draw.css"
    type="text/css" />

<div id="map"></div>
<pre id="info"></pre>
<script
    src="https://api.mapbox.com/mapbox-gl-js/plugins/mapbox-gl-
geocoder/v4.4.2/mapbox-gl-geocoder.min.js"></script>
<link rel="stylesheet"
    href="https://api.mapbox.com/mapbox-gl-js/plugins/mapbox-gl-
geocoder/v4.4.2/mapbox-gl-geocoder.css"

```

```

    type="text/css" />
<div class="calculation-box">
<p>Draw a polygon using the draw tools.</p>
<div id="calculated-area"></div>
</div>

<!-- Promise polyfill script required to use Mapbox GL Geocoder in IE 11 -->
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-
promise.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/es6-promise@4/dist/es6-
promise.auto.min.js"></script>
<script>
    var lat;
    var lng;
    var alti;
    //setup mapbox

    mapboxgl.accessToken =
'pk.eyJ1IjoibnBudCIImEiOiJjazhibG93c2kwZGVrM3JvYW4wd3o2MzllIn0.oOz
u6V4M8euRVerkgCci9A';

    var map = new mapboxgl.Map({
        container: 'map',
        style: 'mapbox://styles/mapbox/satellite-v9'
    });

    map.fitBounds([
        [-168.30520240075913,64.8358548708973],
        [152.77377797786426,-29.77236723145483]
    ]);

    //geofence
    var draw = new MapboxDraw({
        displayControlsDefault: false,
        controls: {
            polygon: true,
            trash: true

```

```

    }
  });
  map.addControl(draw);
  map.on('draw.create', updateArea);
  map.on('draw.delete', updateArea);
  map.on('draw.update', updateArea);

function updateArea(e) {
  var data = draw.getAll();
  var answer = document.getElementById('calculated-area');
  if (data.features.length > 0) {
    var area = turf.area(data);
    // restrict to area to 2 decimal points
    var rounded_area = Math.round(area * 100) / 100;
    answer.innerHTML =
      '<p><strong>' +
      rounded_area +
      '</strong></p><p>square meters</p>';
  } else {
    answer.innerHTML = "";
    if (e.type !== 'draw.delete')
      alert('Use the draw tools to draw a polygon!');
  }
}

//firebase
// Web app's Firebase configuration
var Config = {
  apiKey: "AIzaSyBNcURCd6BDbek7H5WMdCRimbAO5IuFOIs",
  authDomain: "npnt-acf2d.firebaseio.com",
  databaseURL: "https://npnt-acf2d.firebaseio.com",
  projectId: "npnt-acf2d",
  storageBucket: "npnt-acf2d.appspot.com",
  messagingSenderId: "909672195598",

```

```

        appId: "1:909672195598:web:19c6ecee582c7d2a927e5f",
        measurementId: "G-XH7EFWC400"
    };

    // Initialize Firebase
    firebase.initializeApp(Config);

    var database = firebase.database();
    var childData;
    var childKey;

    function getData() {
        return new Promise((resolve, reject) => {
            database.ref().on('child_added', function (snapshot) {
                snapshot.forEach(function (childSnapshot) {
                    childKey = childSnapshot.key;
                    childData = childSnapshot.val();
                    lat = childData.latitude;
                    lng = childData.longitude;
                    resolve(childData);
                })
            })
        })
    }

    showPoints();

    // mouse movement

    map.on('mousemove', function (e) {
        document.getElementById('info').innerHTML =
            // e.point is the x, y coordinates of the mousemove event relative
            // to the top-left corner of the map
            JSON.stringify(e.point) +
            '<br />' +
            // e.lngLat is the longitude, latitude geographical position of the event

```

```

        JSON.stringify(e.lngLat.wrap());
    });
    //showing points
    function showPoints() {
        map.on('load', function () {
            getData().then((res) => {
                const lat = parseFloat(res.latitude);
                const lng = parseFloat(res.longitude);
                map.addSource('points', {
                    'type': 'geojson',
                    'data': {
                        'type': 'FeatureCollection',
                        'features': [
                            {
                                // feature for Mapbox DC
                                'type': 'Feature',
                                'geometry': {
                                    'type': 'Point',
                                    'coordinates': [lng,lat]
                                },
                                'properties': {
                                    'title': 'Drone1',
                                    'icon': 'rocket'
                                }
                            }
                        ]
                    }
                });
                map.addLayer({
                    'id': 'points',
                    'type': 'symbol',
                    'source': 'points',
                    'layout': {
                        // get the icon name from the source's "icon" property

```



```

        // concatenate the name to get an icon from the style's sprite
sheet

        'icon-image': ['concat', ['get', 'icon'], '-15'],
        // get the title name from the source's "title" property
        'text-field': ['get', 'title'],
        'text-font': ['Open Sans Semibold', 'Arial Unicode MS Bold'],
        'text-offset': [0, 0.6],
        'text-anchor': 'top'
    }
});
})
});
}

map.addControl(new mapboxgl.NavigationControl());
</script>
</body>
</html>

```

3.9. PyQt5

PyQt5 is cross-platform GUI toolkit, a set of python bindings for Qt v5. One can develop an interactive desktop application with so much ease because of the tools and simplicity provided by this library.

A GUI application consists of Front-end and Back-end. PyQt5 has provided a tool called ‘QtDesigner’ to design the front-end by drag and drop method so that development can become faster and one can give more time on back-end stuff.

PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android.

PyQt5 may also be embedded in C++ based applications to allow users of those applications to configure or enhance the functionality of those applications. PyQt5 is released under the GPL v3 license and under a commercial license that allows for the development of proprietary applications.

PyQt5 is one of the most used modules in building GUI apps in Python and that's due to its simplicity as you will see.

Another great feature that encourages developers to use PyQt5 is the PyQt5 designer which makes it so easy to develop complex GUI apps in a short time. You just drag your widgets to build your form.

Installation:

First, we need to install PyQt5 library. For this, type the following command in the terminal or command prompt:

\$Pip install pyqt5

If successfully installed one can verify it by running the code:

>>>Import PyQt5

PyQt5 provides lots of tools and Qt Designer is one of them. For this run this command:

\$pip install PyQt5-tools

Here we can design a GUI by directly writing a code according to PyQt5 platform by importing PyQt5 in the code or you can also design the GUI using Qt designer which will be available at location:

C:\Program Files\Python36\Lib\site-packages\pyqt5-tools

You can also open qt designer by typing and running designer in command prompt. By running it automatically opens qt designer window which looks like..

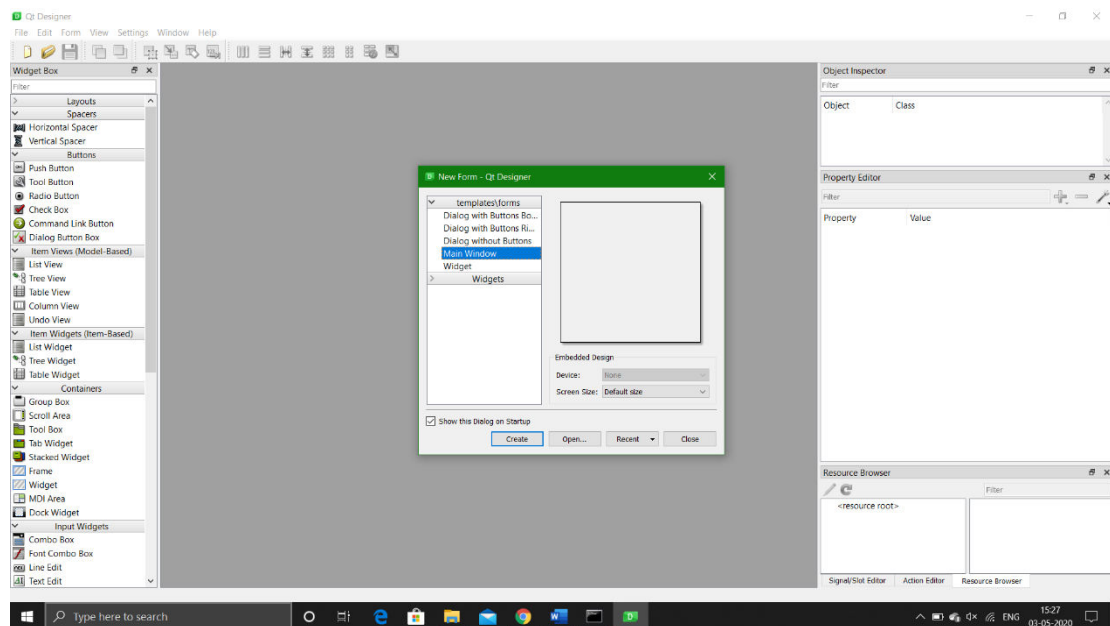


Figure 3.9.1: QT designer

And to Create a new GUI window by selecting main window and create the new window. And now you can drag and drop required containers, buttons, widgets etc. to create a GUI. Our projects ultimate goal is to have a GUI with map control and some buttons which somewhat looks like this..

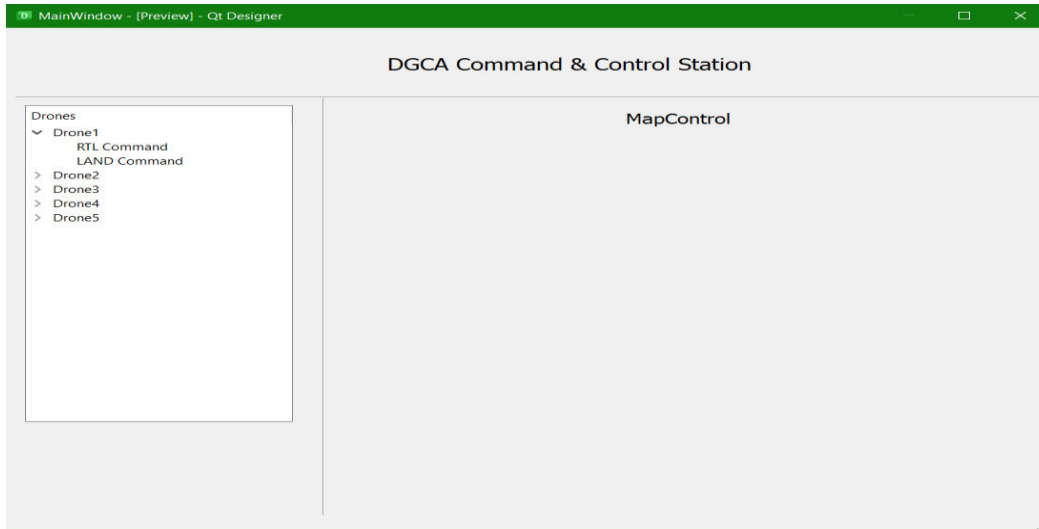


Figure 3.9.2: Sample DGCA GUI (not yet completed)

But the map control which we have created using mapbox earlier was a HTML File. We can insert html file in this PyQt but we can't control the drone using cursor which was not at all useful. So, we wrote a program for this where we created two windows one for html file and other for our drop-down container which looks like..

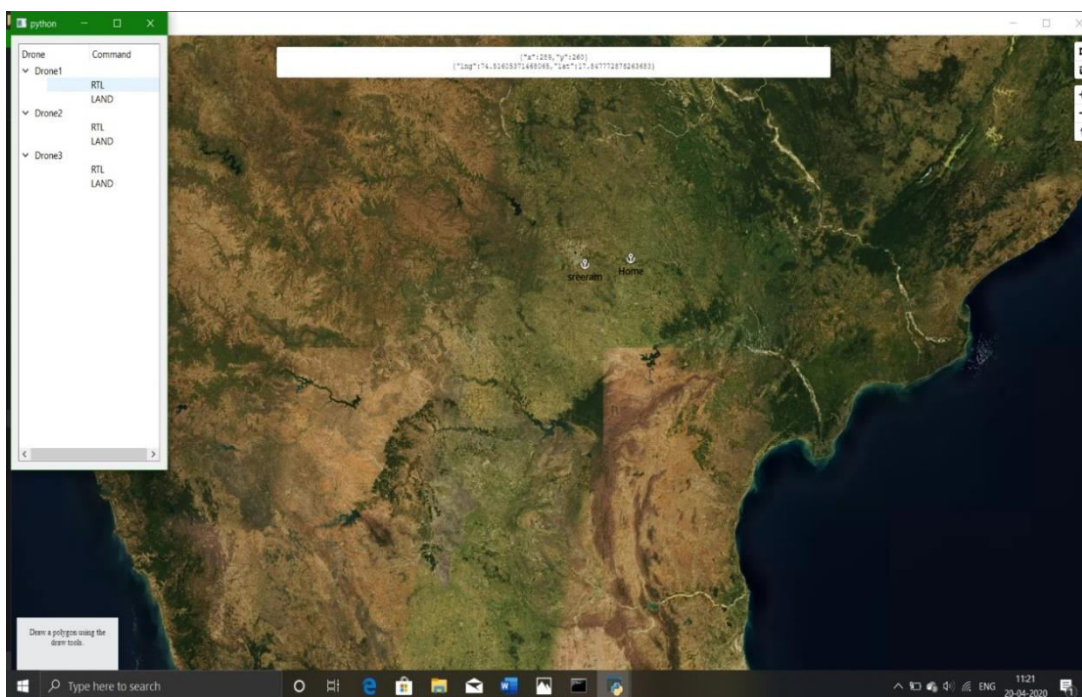


Figure 3.9.3: DGCA GUI (Not final)

The program used to create the fig. gui :

Pyqt5_qwebengineview.py:

```
import sys
import os
from PyQt5 import QtCore, QtWidgets, QtWebEngineWidgets
from PyQt5.QtWebEngineWidgets import *
from PyQt5.QtCore import *

app = QtWidgets.QApplication(sys.argv)
view = QtWebEngineWidgets.QWebEngineView()
window = QtWidgets.QWidget()
window.setGeometry(0,0,280,400)
view.setGeometry(0,0,2000,2000)

layout = QtWidgets.QVBoxLayout(window)

#view.load(QtCore.QUrl().fromLocalFile(os.path.split(os.path.abspath(__file__))[
0]+r'C:\Users\SREERAM\Desktop\project\test.html'))
#web = QWebEngineView()
#web.load(QUrl(r"C:\Users\SREERAM\Desktop\project\test.html"))
#htmlView.setSource(QtCore.QUrl.fromLocalFile("test.html"))
#web.show()

tw = QtWidgets.QTreeWidget()
tw.setHeaderLabels(['Drone','Command'])
#tw.setGeometry(0,0,100,300)

cg = QtWidgets.QTreeWidgetItem(tw,['Drone1'])
c1 = QtWidgets.QTreeWidgetItem(cg,['', 'RTL'])
c2 = QtWidgets.QTreeWidgetItem(cg,['', 'LAND'])
ch = QtWidgets.QTreeWidgetItem(tw,['Drone2'])
c3 = QtWidgets.QTreeWidgetItem(ch,['', 'RTL'])
```

```
c4 = QtWidgets.QTreeWidgetItem(ch,['', 'LAND'])
ci = QtWidgets.QTreeWidgetItem(tw,['Drone3'])
c5 = QtWidgets.QTreeWidgetItem(ci,['', 'RTL'])
c6 = QtWidgets.QTreeWidgetItem(ci,['', 'LAND'])
fname = os.getcwd()+ '/npntfinal.html'
url = QUrl.fromLocalFile(fname)
layout.addWidget(tw)
view.setUrl(url)
view.show()
window.show()
sys.exit(app.exec_())
```

4. RESULTS AND TESTCASES

```
pi@raspberrypi: ~/drone/how_do_drones_work/myscripts
pi@raspberrypi:~/drone/how_do_drones_work/myscripts $ clear
pi@raspberrypi:~/drone/how_do_drones_work/myscripts $ python test_connect.py
>>>> Connecting with the UAV <<<
CRITICAL:autopilot:PreArm: Hardware safety switch
CRITICAL:autopilot:PreArm: 3D Accel calibration needed
CRITICAL:autopilot:PreArm: Compass not calibrated
CRITICAL:autopilot:PreArm: Compass not calibrated
CRITICAL:autopilot:PreArm: Board (4.0v) out of range 4.3-5.8v
Autopilot version: APM:Copter-4.0.3
Supports set attitude from companion: True
Position: LocationGlobalRelative:lat=17.3301368,lon=78.5660371,alt=2.338
Attitude: Attitude:pitch=0.034065309912,yaw=1.32045662403,roll=3.03807377815
Velocity: [1.06, 0.57, 0.4]
time: 2020-03-28 11:23:10.068614
Last Heartbeat: 0.758479561
Is the vehicle armable: True
Groundspeed: 1.21111178398
Mode: STABILIZE
Armed: False
EKF Ok: True

Adding an attitude listener
Attitude:pitch=0.0340605191886,yaw=1.31637716293,roll=3.03819012642
1585374803211077
Attitude:pitch=0.0340286828578,yaw=1.31023192406,roll=3.03823590279
1585374803460785
Attitude:pitch=0.03387330845,yaw=1.30404365063,roll=3.03822803497
1585374803708916
Attitude:pitch=0.0339213870466,yaw=1.29777896404,roll=3.03827667236
1585374803961040
Attitude:pitch=0.0339255742729,yaw=1.29152202606,roll=3.03844475746
1585374804210802
Attitude:pitch=0.0338965430856,yaw=1.28534603119,roll=3.03847718239
1585374804458704
Attitude:pitch=0.0338404029608,yaw=1.27917456627,roll=3.03854107857
1585374804710835
Attitude:pitch=0.0338222533464,yaw=1.27294313908,roll=3.03880023956
1585374804959009
Attitude:pitch=0.0337462723255,yaw=1.26675117016,roll=3.0388777256
1585374805208791
Attitude:pitch=0.0336538664997,yaw=1.2604868412,roll=3.03896689415
1585374805458556
Attitude:pitch=0.0335920266807,yaw=1.25423121452,roll=3.03909063339
1585374805710794
Attitude:pitch=0.0335556715727,yaw=1.24800240993,roll=3.03912234306
1585374805960972
Attitude:pitch=0.033519975841,yaw=1.24176084995,roll=3.03904271126
1585374806211033
Attitude:pitch=0.0334504842758,yaw=1.2354850769,roll=3.03882288933
1585374806458896
```

Figure:4.1 shows the latitude, longitude, time, altitude etc. in raspberry pi which is being sent from Pixhawk using a python code.

The above data has got by running a python program in raspberry pi. The main idea of getting these data was to send it to firebase and display the location on the GUI map. Here details like position, altitude, velocity, time and other data which was not actually useful.

In the figure 4.2 there was data like latitude and longitude which was sent from raspberry pi and was not manually written. The data has been sent by using the above program which automatically finds the location of drone(Pixhawk) and send it to firebase accordingly. There will be an unique code for every data coming into firebase which is actually a key in an encrypted form to send the data securely into firebase.

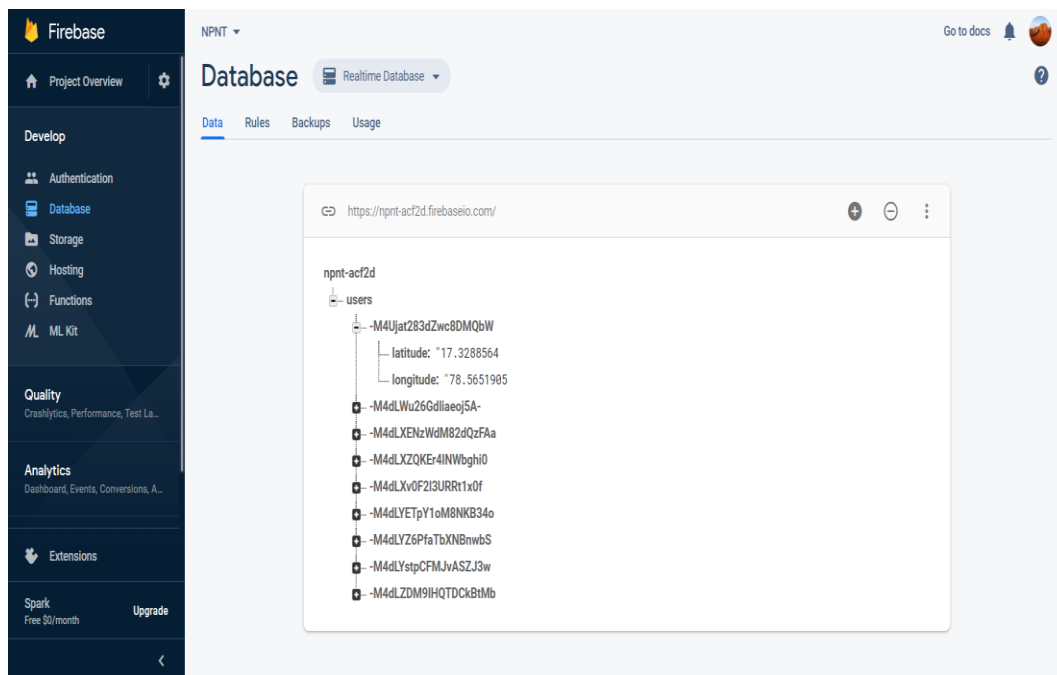


Figure 4.2: Loading GPS data into firebase

Now, the data from firebase will be used in map control to display the location of drone as we can see in the figure 4.2 which contains latitude and longitude points and that data will be used in map control which automatically fetches that firebase data and displays it on map. Now again this map control will be used in a python-based GUI where we can control the drone by automatically sending the commands when it reaches the geo fence area.

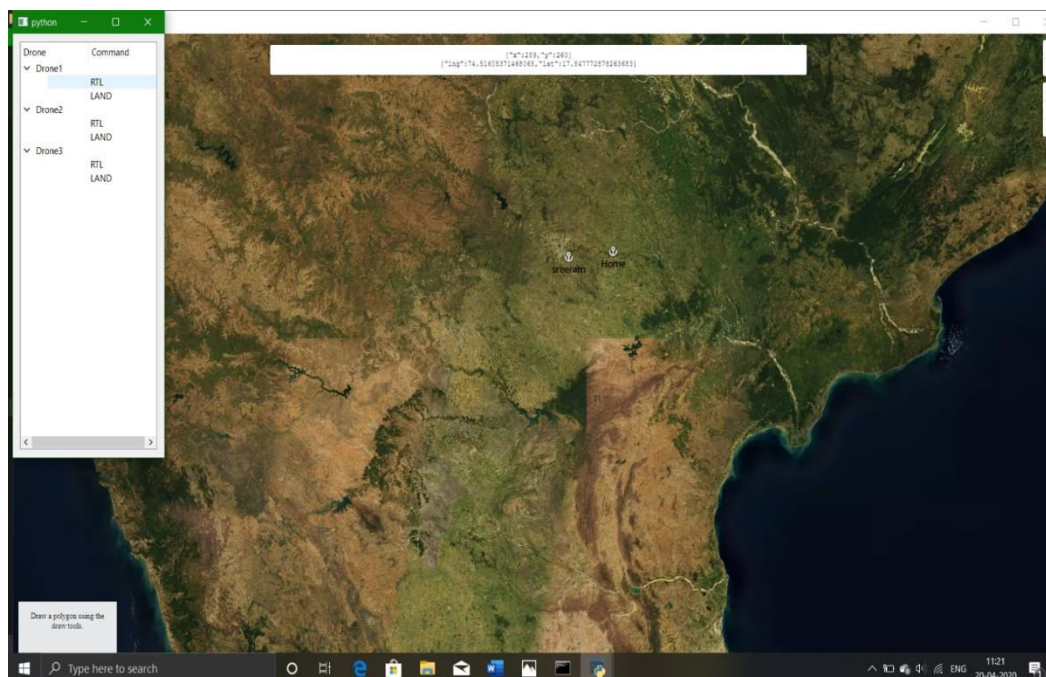


Figure:4.3 GUI which displays multiple drones and control drones

Now, to test whether commands were being sent or not there is a simulation which was already explained in design and implementation chapter. Now whenever drone reaches the geo fence area commands need to be sent to land or return the drone.

Here is a client and server programs where client is raspberry pi and server is our windows system where commands will be sent through a program.

```

pi@raspberrypi:~$ vncserver
New 'X' desktop is raspberrypi:1
Starting applications specified in /home/pi/.vnc/xstartup
Log file is /home/pi/.vnc/raspberrypi:1.log

pi@raspberrypi:~$ python server.py
CRITICAL:autopilot:Frame: QUAD
WARNING:dronekit:Link timeout, no heartbeat in last 5 seconds
listening for command
('Got connection from', ('192.168.137.102', 60386))
LAND
WARNING:dronekit:Link timeout, no heartbeat in last 5 seconds

Microsoft Windows [Version 10.0.18363.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\SREERAM>python client.py
python: can't open file 'client.py': [Errno 2] No such file or directory

C:\Users\SREERAM>cd desktop
C:\Users\SREERAM\Desktop>python client.py
C:\Users\SREERAM\Desktop>

```

Figure 4.4. Sending commands through client & server

As we sent the command LAND to the pi system drone has been landed here in this simulation map which is mission planner.

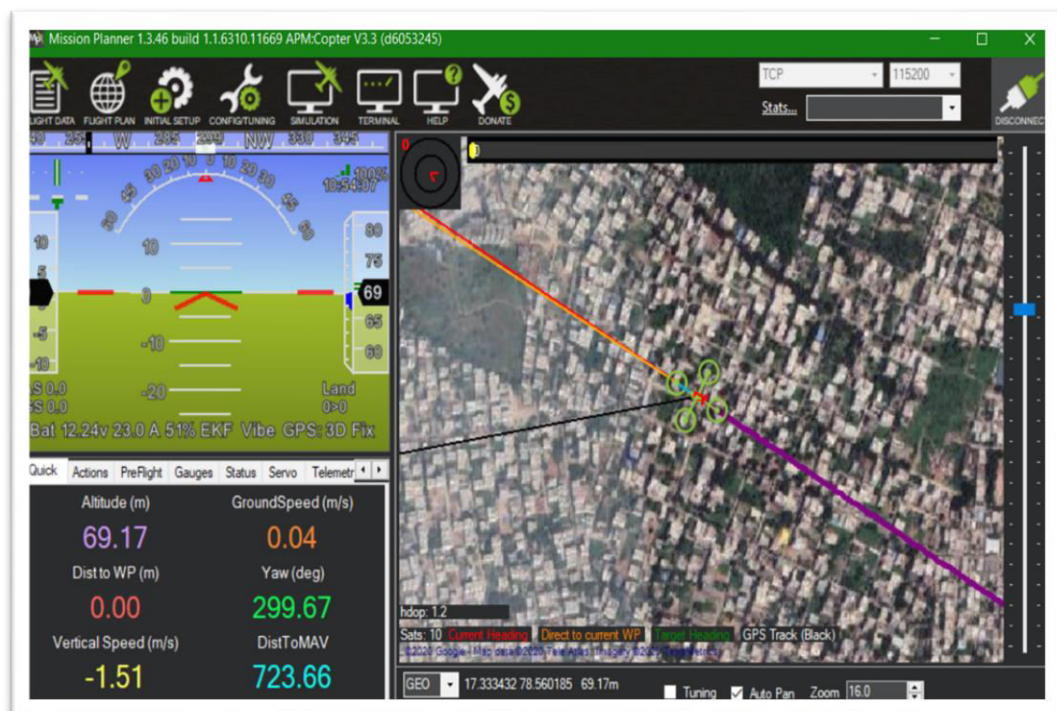


Figure 4.5: Drone landing after command being sent

5. Conclusion &Future Work

The main aim of this project is to design and develop NPNT compliance IOT device to operate for Digital Sky Platform drone operations in India for safely and securely operations by adhering to DGCA Drone regulations. Here, we developed one such NPNT IOT device, for operations that would transmit the geo locations to Digital Sky platform operating via flight controller Pixhawk and raspberry pi using python programs and sending that location information to google firebase cloud for viewing.

A map control takes that location data from firebase and displays the drone location on map. Now whenever drone out of geo-fence area commands like land and RTL will be sent from our server system to raspberry pi which can bring back or land the drone.

Presently, the map control used for display contains minimal features such as view of the geo locations, however, in future it is proposed to extend the dynamic view the drone locations and for multiple drones at a time A selection based method for viewing of the drones is also proposed to extend.

At present, the command and communication to the drone happens at the server end as a back end script, however, in future it is proposed to send commands from Map control GUI for better access and controlling of the drones. Here, for the extension the python code/library of sending commands is to be integrated with the Map library/Control.

Here, for the system testing, all the commands are tested with Drone kit and SITL (simulation in the loop) however this needs to be tested with actual drone. Also, Geo fence area and automated alert logging facility in Digital Sky platform to be tested.

6.References

- [1] Milind-1: Custom Built Drone Platform for Remote Sensing Applications with Swappable Payload and Sensors, INCA38, 2018.
- [2] 3DR, DIY Quad, Build Manual, 2014.
- [3] 3D Reality Modeling Software, Context Capture,
<https://www.bentley.com/en/products/brands/contextcapture>
- [4] An Overview of Commercial Drone Applications, By Drone Survey 2016, Hong Kong, 20th May 2016.
- [5] Aislan Gomide Foina, Raja Sengupta, Patrick Lerchi, Zhilong Liu and Clemens Krainer, Drones in Smart Cities: Overcoming Barriers through Air Traffic Control Research, 2015, Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS), Mexico.
- [6] Agisoft Photoscan photogrammetry software, <http://www.agisoft.com/>
- [7] Agisoft Photoscan User Manual, Professional Edition, Version 1.0.0, 2013.
- [8] Agoston Restas, Drone Applications for Supporting Disaster Management, World Journal of Engineering and Technology, 2015, 3, pp. 316-321.
- [9] ArduPilot, open source auto flight software ,<http://ardupilot.org/>
- [10] Autodesk Recap360, 3D Scanning Software,
<https://www.autodesk.com/products/recap/overview>
- [11] Cramer M, Przybill H.J, and Zurhorst A, 2017, UAV Cameras: Overview and Geometric Calibration Benchmark, The international Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XLII-2/W6, 2017.
- [12] Drone2Map for ArcGIS, <https://www.esri.com/en-us/arcgis/products/drone2map/overview>
- [13] Drone mapping software, <https://www.opendronemap.org/>
- [14] DJI Drones web link, <https://www.dji.com>

- [15] Drone Review, <https://www.tomshardware.com/reviews/multi-rotor-quadcopter-fpv,3828-6.html>
- [16] DroneDeploy, The leading Drone Software Platform with Unlimited Flying, Mapping and Sharing, <https://www.dronedeploy.com/>
- [17] Drone Mapping Software, OpenDroneMap, <https://www.opendronemap.org/>
- [18] Enric Pastor, Juan Lopez and Pablo Royo, 2006, A Hardware/Software Architecture for UAV payload and mission control, 25th Digital Avionics Systems Conference, 2006.
- [19] JoergSchlinkheider, PrashantRamarao, Tim Tully, VineetBanga, VipulDeokar, 2014, Commercial Drone Are Coming- Sooner Than You Think? Insights in Engineering Leadership White Paper, <https://ikhlaqsidhu.files.wordpress.com/2015/01/commercial-drones-white-paper-1.pdf>
- [20] Introduction to Photogrammetry, SasankaMadawalagama, GeoinformaticsCenter, Asian Institute of Technology, Thailand.
- [21] Introducing MANTIS Q, <http://us.yuneec.com/>
- [22] Inside GNSS, 2016, A.S. Gangeshan, S.V. Satish, A. Kartik, S. Nirmala, and G. Ramesh, Indian Space Research Organization and Airports Authority of India, India's Satellite-Based Augmentation System, GAGAN- Redefining Navigation Over the Indian Region, January/February 2016.
- [23] Konstantin Kakaes, Faine Greenwood, Mathew Lippincott, Shannon Dosemagen, Patrick Meier, and Serge Wich, 2015, Drones and Aerial Observation: New Technologies for Property Rights, Human Rights, And Global Development A Primer, New America press
- [24] KhaulaAlkaabi, AbdelgadirAbuelgasim, International Journal of Social Sciences Arts and Humanities, 2017, Vol. 5, No.1, 2017, pp. 4-11.
- [25] Richard Szeliski,, Image Alignment and Stitching; A Tutorial, Foundation and Trends in Computer Graphics and Vision, Vol.2, No 1 (2006) 1-104.
- [26] Maps Made Easy, OrthoPhoto Map and 3D model generation, <https://www.mapsmadeeasy.com/>

- [27] Pix4D Mapper, Professional drone mapping software, <https://pix4d.com/>
- [28] Pixhawk Overview, xxxxxxxx
- [29] Requirements for operation of Civil Remotely Piloted Aircraft Systems (RPAS), Office of the Director General of Civil Aviation, August 2018.
- [30] SenseFlyeBee, <https://www.sensefly.com/>
- [31] The Rise of The Drones, 2016, An overview of Commercial Drone Applications, By Drone survey, Hong Kong.
- [32] Trimble, Transforming World Solutions, <https://www.trimble.com/>
- [33] UC Berkely Report, Insights in Engineering Leadership, 2014, Commercial Drones Are Coming- Sooner Than You Think.
- [34] Unmanned Aerial Vehicle System Association, Unmanned aerial vehicle system association – uav or uas, 2012b, URL <http://www.uavs.org/advantages>.
- [35] <https://digitalsky.dgca.gov.in/>
- [36] https://dronekit-python.readthedocs.io/en/latest/examples/simple_goto.html
- [37] <https://www.raspberrypi.org/documentation/remote-access/ssh/windows.md>
- [38] <https://maker.pro/raspberry-pi/projects/how-to-connect-a-raspberry-pi-to-a-laptop-display>
- [39] <https://ardupilot.org/planner/docs/mission-planner-initial-setup.html>
- [40] https://dronekit-python.readthedocs.io/en/latest/examples/follow_me.html
- [41] https://dronekit-python.readthedocs.io/en/latest/examples/drone_delivery.html
- [42] <https://gist.github.com/vo/9331349>
- [43] <https://docs.mapbox.com/api/>
- [44] <https://docs.mapbox.com/mapbox-gl-js/example/live-geojson/>
- [45] <https://www.raspberrypi.org/documentation/remote-access/vnc/>
- [46] <https://dronekit-python.readthedocs.io/en/latest/automodule.html>
- [47] <https://ardupilot.org/dev/docs/sitl-native-on-windows.html>
- [48] <https://pypi.org/project/PyQt5/>