A Thesis

entitled

Speech Assisted Interface for Quadcopter Flight Control

by

Yaswanth Palivela

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the

Masters of Science Degree in

Engineering

_____

Dr. Jackson Carvalho, Committee Chair

_____

Dr. Henry Ledgard, Committee Member

_____

Dr. Ezzatollah Salari, Committee Member

_____

Dr. Amanda Bryant-Friedrich, Dean
College of Graduate Studies

The University of Toledo

May 2018

An Abstract of

Speech Assisted Interface for Quadcopter Flight Control

by

Yaswanth Palivela

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the
Masters of Science Degree in
Engineering

The University of Toledo

May 2018

Drone control is a complex task that requires knowledge of working principles of both the drone being controlled and its controller. The six degrees of freedom of quadcopters are achieved by planned maneuvers that make use of the four controllable motors on the quadcopter. Keeping track of the quadcopter's trajectory, as well as simultaneously manipulating the transmitter controls, can be a difficult task for most users. Human performance is inversely proportional to the amount of information to be processed. Therefore, minimizing the cognitive load associated with the transmitter operation is the focus of this research. This thesis proposes speech commands as a replacement for the standard RF transmitter based form of quadcopter control. The solution presented takes advantage of the widespread availability of smartphones and internet access.

To my parents Palivela Soma Sekhara Rao and Palivela Laliteswari, who have always put my needs ahead of theirs.

Acknowledgements

First and foremost, I would like to thank Dr. Jackson Carvalho for mentoring me throughout my time here at University of Toledo. Without his guidance, I would not be half the student I am today. I would like to thank Dr. Mansoor Alam for believing in me and offering me tuition scholarship to pursue my graduate studies here at UT. I would also like to convey my gratitude to Dr. Daniel Georgiev and Dr. Vijay Devabhaktuni for being exemplary professors, whose words of wisdom have always guided me in tough times.

I am grateful to Dr. Henry Ledgard and Dr. Ezzatollah Salari for taking the time to be on my committee. I would also like to take this opportunity to thank Mr. Marcello Carvalho for his effort and dedication throughout the project, which inspired me a lot in accomplishing my goals. I would like to thank Dr. Ahmad Javaid and his students for helping me with the drone hardware. I would also like to thank James R. Champion, Colin P. Elkin, and Ragadeepika Pucha for helping me revise this thesis.

Last but not least, I would like to thank all my friends and family for supporting me throughout my time here at UT.

Contents

List of Figures

List of Abbreviations

RPi …… Raspberry Pi

PC ……. Personal Computer

RTL ……... Return to Home

SITL …….. Software in the Loop

MAV ……. Micro Air Vehicle

UDP …….. Use Data Protocol

TCP ……... Transmission Control Protocol

UAV …….. Unmanned Aerial Vehicle

APM …….. ArduPilot Mega

GUI …… Graphic User Interface

RPM ……... Rotations per minute

CPU …… Control Processing Unit

Preface

This thesis is original, unpublished, and independent work by the author, Yaswanth Palivela under the tutelage of Dr. Jackson Carvalho.

Chapter 1

Introduction

A problem well experienced by most computer users is the slow response of a machine, especially when many programs are running at the same time. This is the same concept for humans as well. Performance falls when the amount of information to be handled exceeds the capacity of processing. A computer can easily be upgraded to a newer, more powerful unit; however, no such improvement is possible for humans. One way around this challenge is to design user-computer interfaces that accommodate the users' limitations. The idea of improving user accessibility to the control of quadcopters is the main motivation for this research.

1.1. The Problem

Quadcopters have an automation process that competes with some of the most complex aerial frameworks in aviation. The clock and counter-clock nature of any pair of its engines produce the required torque, which empowers the Unmanned Aerial Vehicle (UAV) to fly. With various exact turns of its four propellers, directional developments of a quadrotor are achievable. A quadcopter is a UAV with the capacity to perform challenging tasks, which include: flying in domains where constant supervision is required, and flying in locations where human presence has a high-risk factor. Quadcopters have both linear motions about the x, y, and z axes and rotations around the x, y, and z axes [1]. These six degrees of freedom are achieved via planned maneuvers that make use of their

four controllable motors. Keeping track of the trajectory, the task to be performed by the UAV, as well as simultaneously manipulating the transmitter controls can overwhelm novice users. Operating the UAV system may be made easier by minimizing the attention required by the user to control the transmitter.

The computing power from Mobile phones can be used to cut down the cognitive load of the user by using speech commands, which are easy to remember and require less cognitive processing. Replacing the standard manual form of control for quadcopters with a speech based one would reduce or even eliminate the need for an RF transmitter based form of control. Such a tool would take advantage of widespread availability of mobile phones and access to the Internet. This research proposes to coordinate these technologies into a viable system. This work addresses the usability of quadcopter flight control and proposes to improve the quadcopter flight handling using speech.

1.2. Proposed Solution

Mobile phones are equipped with powerful microchips and offer solid communication handling capacities. The processing capabilities of these gadgets can be utilized while controlling UAV flights. This research proposes a speech based form of controlling quadcopters by providing a proof of concept, which uses a dedicated module to process speech input and communicate the received commands to the controlling hardware. Pixhawk, a Raspberry Pi, and an Android based smartphone were the major hardware components used. The client/server architecture was used for handling the communication between the smartphone and the Raspberry Pi [2].

1.2.1. Assumptions

The proposed solution can be achieved through some user oriented modifications. An Android mobile phone was used as the speech processing component. This optimizes the speech recognition process, as mobile phones are equipped with advanced level speech recognizers, capable of dealing with a wide variety of user speech characteristics. For the development of the prototype to test the proposed solution, Pixhawk, Raspberry Pi, and Xiaomi phone (Redmi note 3) were used. Python and Java were the programming languages selected for the development of all necessary software.

A schematic overview of the concept is given in Fig. 1.

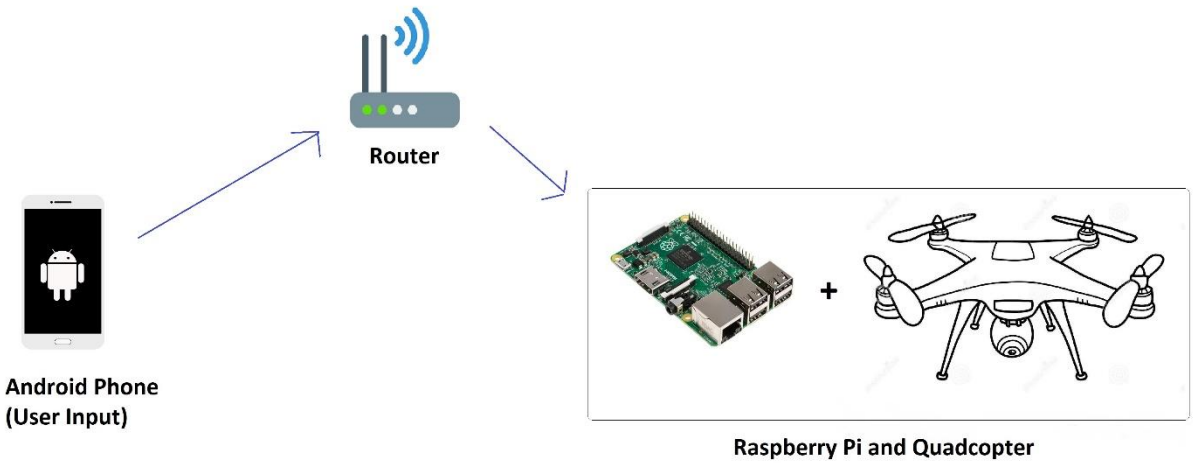


Figure 1: Overview of the working concept.

1.3. Organization of Thesis

This work addresses various aspects related to user interactions with drones, which include handling, automation, construction, and speech control. Chapter 2 covers handling of a drone by humans using a remote controller. It also introduces user interfaces for drones and some special equipment to assist users in flying them. Chapter 3 discusses the development of a quadcopter and introduces the special components related to it. The design of the proposed prototype is addressed in Chapter 4. The communication between the user and the drone is also discussed. In addition, Chapter 4 also addresses the challenges faced during the development of the core module. Chapter 5 covers the work in simulation of the proposed quadcopter. More specifically, this chapter discusses the arrangements made for testing the developed software on a simulator. In Chapter 6, both the results of the simulations, as well as the results from the trial runs of the developed software on two different quadcopters are shown. Chapter 7 presents the conclusion, weaknesses and the scope for future work.

Chapter 2

Related Work

This chapter addresses state of the art drone technologies and smarter ways to access or control drones.

2.1. Interpreting Drone Interfaces

This section assumes that the interface used for flight control will only interact with drones. It also assumes that a collision avoidance system is a built-in feature. This feature ensures that the operator will not make any additional effort for preventing mid-air collisions [3].

Aurora Flight Sciences is an aeronautics research company that primarily specializes in the design and construction of UAVs. According to them, the objectives relating to the interaction between the drones, operator, and interface are classified on a hierarchy of values shown in Fig. 2. The main focus is on human usability and performance, which depend on system performance. Reliability, Maintainability, and Functionality will be the key things to focus on, as well as determining the capabilities of the interface [4].

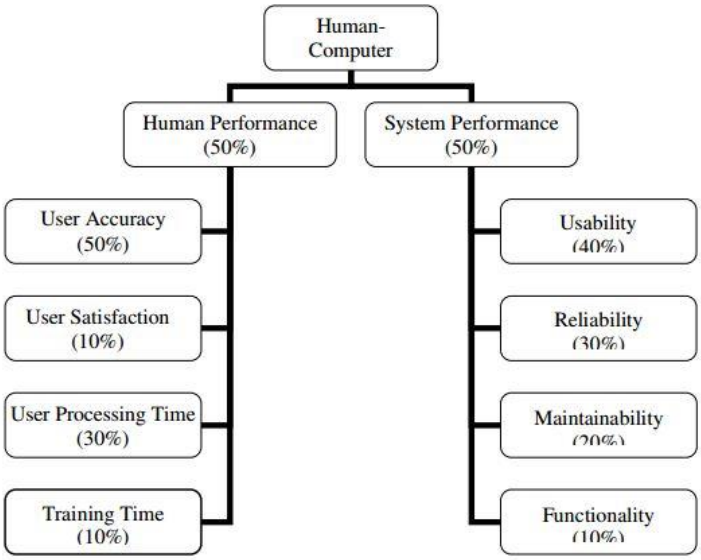


Figure 2: Value Hierarchy.

Utility function: Aurora Flight Sciences has defined a utility function that represents the stakeholder's values in a linear equation.

The following is Aurora's elicited utility function:

$U(X) = .50\ U_{HP}(X) + .50\ U_{SP}(X)$

$U_{HP}(X) \rightarrow$ Human Performance Utility

$U_{SP}(X) \rightarrow$ System Performance Utility

U_{HP}= .5U User Accuracy + .3U User Processing Time + .1U Training time + .1U User satisfaction

U_{SP}= .4U Usability +.3U Reliability +.2U Maintainability + 0.1U Functionality

These equations define and prioritize the enhancement of efficiency in humans. This equation holds true because the system performance is mainly dependent on hardware characteristics. Therefore, enhancing the right tools to the interface and assuming the system attributes will make it qualified.

2.2. User Interfaces for Drones

Mission Planner: With a variety of features to engage with, Mission Planner is a popular drone control interface. It is used for ArduPilot open source autopilot projects and ground station applications. It is compatible exclusively with Windows and is used as a dynamic control supplement to help with configuration for autonomous vehicles.

The list of features associated with the mission planner are:

- Setup, configure, and tune drones to optimal performance.
- Load firmware into Pixhawk that controls the drone.
- Plan autonomous missions by selecting waypoints and targets.
- Download logs to analyze the flight data.
- Interface with flight simulators to create a full hardware-in–loop simulator.
- Monitor the vehicle status in operation.
- View and analyze the telemetry logs.
- Operate the vehicle in first person view.

Figure 3: Main screen of Mission Planner.

Figure 4: Mission Planner with waypoint selection screen.

Q Ground Control: This interface provides full flight control support as well as configuration for the ArduPilot vehicles. It runs on Windows, Linux, and Mac.

The following are the features that are available with Q Ground Control:

- Create a compatible link for MAVLink capable vehicles.
- Plan and execute autonomous flight missions.
- Track flight position using satellite flight maps.
- Stream video with instrument display overlays.

Figure 5: Q ground control on an Android platform.

Although Mission Planner and Q Ground Control are similar, there are also a few differences. One of those differences is that Mission Planner runs only on Windows. This

could potentially cause performance issues if other machines are running Windows through a virtual machine. Another difference is that Mission Planner requires Pixhawk for controls. Q Ground Control offers a different approach and gives users additional freedom when controlling a drone.

2.3. Automation and its Consequences

Why automate a system?

An automated system performs any function more efficiently and accurately than a human operator. The expectation is that an automated drone can perform at a lower cost than one with a human operator [5]. With higher reliability, it could be a secure system as well as a safer system. System failures often lead to incidents in which people are injured or damage is caused to the surroundings. In a way, keeping human operators out of the system is a way of making the system more secure. Therefore, even though economics has been a main driving force to increase automation, it could be said that an automated system is a safer system [6].

What can go wrong in an automated system?

One of the problems with an automated system is that it only allows full human interaction before or after the automation process is completed [7]. This approach works when the designer considers problems such as feedback performance to the operator, the level to which the automation allows the operator to remain engaged in the operation, and the speed at which the operator must respond back to the system. Due to the inability to consider these performance and training issues, designers of automated systems have not fully integrated these human variables systematically [8]. This fact has caused incidents

with airplanes operating on an automatic pilot, which have lost control when flown into a terrain operating with speed constraints.

Thus, an important question to ask is: why is there this disconnect between human operators and automated system designers? The literature [9] says that the incidents involving automated systems must deal with human operators abusing and misusing the automated components of the system that they are operating. Misuse is treated as over-reliance or under-reliance on the automated systems. Under-reliance defines an operator not relying on the automation when it is required. Over-reliance on automation refers to an operator completely turning over the operation to the automated control system and withdrawing users' own valuable input affecting the system's performance.

As the level of automation increases, operators have more of a chance to tell what is going wrong in a process. If human operators are not able to tell that there is an impending problem in the system, they must rely on the automated system to find the problem. Automation cannot be built to process every option in which a human operator is required to make judgments. Without using the knowledge that a human operator develops from experiencing all the faults from using a drone, the risk of incident is higher. The more automated a system becomes, the more over-reliant an operator becomes and the more the user stays in the "habits of mind" mode and is thus less likely to switch cognitive gears when needed.

How to reduce cognitive load on human operators?

Artificial intelligence (AI) control paradigms tend to be one-on-one in nature and cater to monolithic systems. An example is the paradigm between an operator and a large multi-

functional robot. However, the future of AI is likely to be with smaller, more distributed systems and on a larger scale. Major advances have been made in the commercialization of smaller unmanned autonomous systems like quadcopters and small ground-based robots. On coupling of distributed AI algorithms with conventional map-based and minimalistic interfaces, there are a decreased amount of required user actions in carrying out a better way point based guidance systems [9].

Chapter 3

Development of the Quadcopter

Development of a quadcopter involves making some key decisions. Every quadcopter is constructed and later developed according to its design, i.e. by the shape or the type of frame. This case involves testing on two quadcopters with two different types of frames [10] [11].

The list of the components involved in developing the quadcopter are:

- Frame
- Motors and propellers
- Speed Controllers
- Flight Controller (Pixhawk (PX4))
- Battery
- Raspberry Pi
- External GPS and compass module
- Radio Receiver

3.1. Types of Frames

The type of frames involved in this thesis are regular “Quad-X” and “Angular V tail” shaped drones [12].

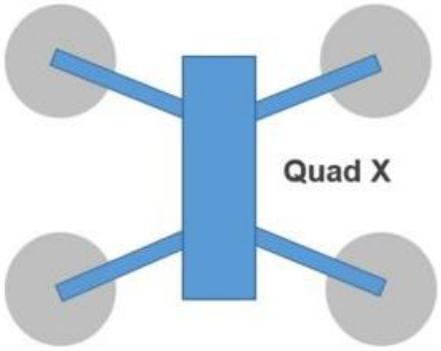


Figure 6:Quad-X frame.

Figure 7: Angular V tail frame.

Both frames have their own advantages because of their frame shapes. A *Quad-X frame* is one of the most basic frame types for a drone. Although the Quad-X's frame is

basic, it still offers structural advantages that make it stable in the air and one of the easiest types to control. The *Angular V Tail* frame has some unique abilities compared to the former. Its advantages and disadvantages are explained as follows.

Advantages with using the Angular V Tail frame

- Improved yaw authority over a quad (when yawing, a V tail creates a relatively large roll moment)
- Easy identification of which way the drone is facing when it is in air
- Improved maneuverability

Disadvantages with using the Angular V Tail frame

- Reduced flight times due to loss of downward thrust
- Complex flight control by having only two lifting surfaces
- Difficulty maneuvering while taking off

At first, developments on the *Angular V Tail* frame were quite successful in building and assembling it. However, there was a problem with this frame's weight distribution. The drone always spun on its rear leg while taking off, and there was no proper control over it when it is in the air. The reason behind this was mainly due to the additional weight outside the safe weighing area. Figure 8 explains how the weight distribution is monitored on the drone [13].

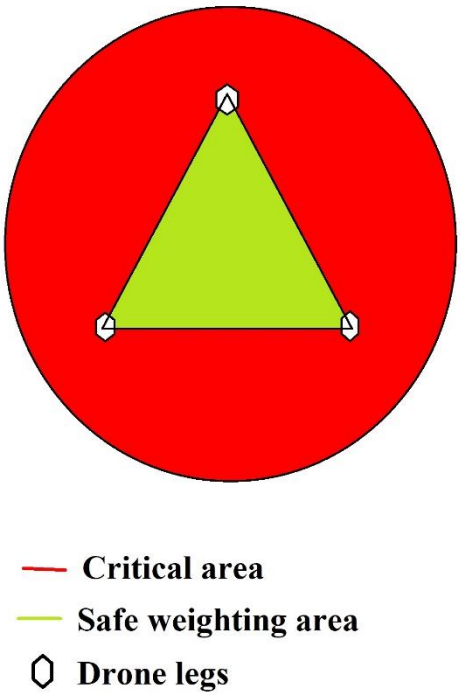


Figure 8: Safe weighting zones for a drone.

Earlier efforts did not follow any weight distribution logics and hence experienced the drifting problems. The problems encountered led to the development of another model based on the Quad-X frame type. The *Quad-X* model has much easier access to its components compared to the *Angular V tail* model. It also had a better safe weighting area, which allows one to take advantages of the extra space.

3.2. Configuration of basic quadcopter software

Configuring basic software actually involves two major hardware components of the quadcopter [14].

- Flight Controller (Pixhawk (PX4))
- Raspberry Pi

Flight Controller: Configuring the flight controller involves many calibration requirements. The flight controller has several components inside it, such as a compass, acceleration, GPS, etc., which are calibrated using the Mission Planner software. Once the Pixhawk is connected, the Mission Planner software allows for various calibration settings for the Pixhawk components. The respective type of frame on which the Pixhawk is going to be used also needs to be selected [15].

Raspberry Pi: A Raspberry Pi (RPi) is a mini computer that contains processor speeds up to 1.4 GHz. Raspbian OS is the operating system selected for this device, as it is the most commonly used operating system, and loading it on to the RPi is a basic task. Raspbian is optimized for the RPi hardware, thereby allowing easy ability to run the basic programs and utilities required for this implementation [16].

3.3. Additional equipment for quadcopter assistance

This section discusses the additional sensors that we thought of adding to the quadcopter. Their association with the drones will bring added strength to them and improve the flight quality of the drone.

3.3.1. Sonar Sensor

An ultrasonic sensor is a gadget that can gauge separation from an object by utilizing sound waves. By recording the time difference between the sound wave being produced and the sound wave coming back, it is conceivable to compute the separation between the sonar sensor and the object. Here, the main idea of using a sonar sensor is to detect the conditions of the floor where the drone lands. This can be achieved with the help of four individual sensors attached on each wing of the quadcopter. These sensors calculate the

distance from each wing of the quadcopter to the ground and sends the information to the on-board computer. The onboard computer then verifies the data received from the sensors and tallies the data from the other sensors. If the calculated distance from the floor to the drone is almost equal from the four sides of the drone, then the quadcopter will proceed for its landing operations.

3.3.2. Setup for Sonar Sensor

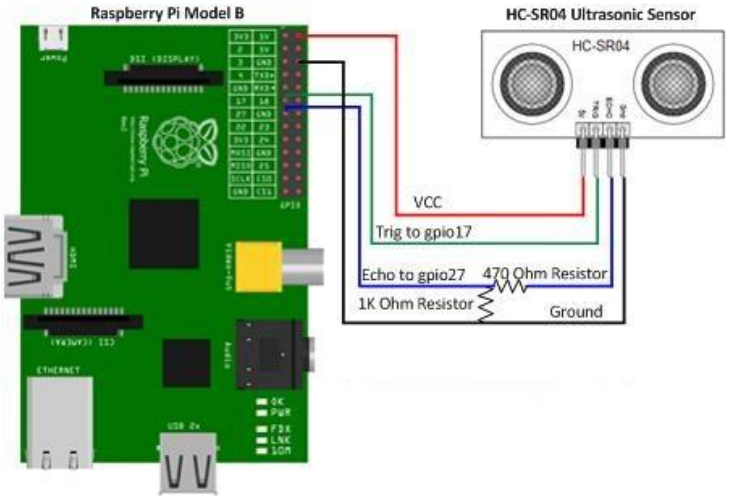


Figure 9: Connection between the Raspberry Pi and a sonar sensor *[17]*.

The connection with a Raspberry Pi is shown in Fig. 9. The Raspberry Pi calculates the distance from the object to the sensor. A Python script is needed to be run on the Raspberry Pi in order to achieve the calculated results. The Ultrasonic sensor yield (ECHO) will dependably yield low voltage (0V) unless it has been activated, in which case it will yield 5V (3.3V with a voltage divider). The next step is to set one GPIO stick on the RPi as a yield to trigger the sensor and set another as a contribution to identify the ECHO voltage change. Later, the necessary libraries for the GPIO pins are used in the script to collect the

data from the sensor [18]. The information received from these sensors includes the time it takes for the signal to travel to and from the drone. Once this time is established, the distance traveled can be calculated using the following formula:

$$Speed = \frac{Distance}{\text{Time}}$$

This formula is used to write an algorithm using Python. The result shown in Fig. 10 was achieved when the Python script containing the algorithm is executed on the RPi.

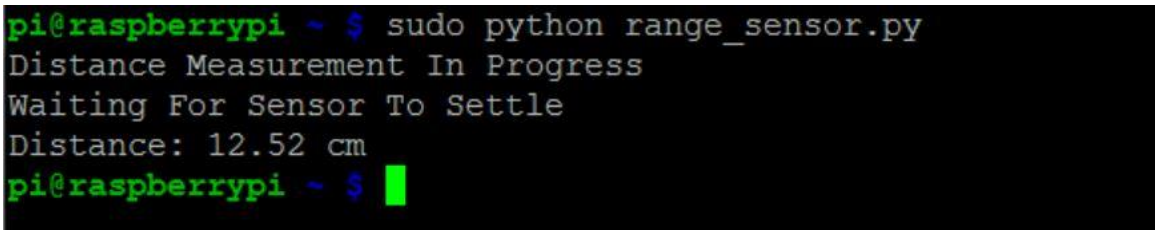
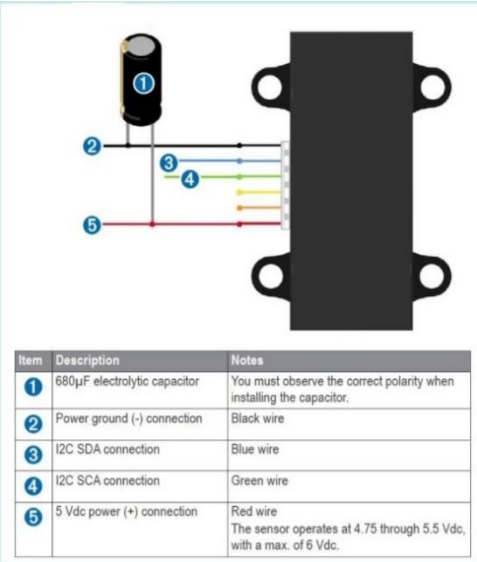


Figure 10: The result obtained from a sonar sensor.

3.3.3. LIDAR Lite

The LIDAR Lite also does the same job as the sonar sensor but with the help of light instead of sound as the means. Since light travels faster than sound, the sensor provides more data in the given instance to process. This additional data is then used to achieve more accurate results compared to a sonar sensor. This sensor is mounted on the front of the quadcopter to bring on several advantages. It can be used to detect the obstacles when the drone is in motion, or it can also be used to map a three-dimensional view for the drone [19] [20].



Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	Power ground (-) connection	Black wire
3	I2C SDA connection	Blue wire
4	I2C SCA connection	Green wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

Figure 11: Connection diagram for the Raspberry Pi *[21]*.

Figure 12: LIDAR Lite *[22]*.

A setup similar to the sonar sensors is made, and an algorithm is used, to calculate the distance. Fig. 13 shows the results from using LIDAR Lite to calculate the distance of the object in front of it.

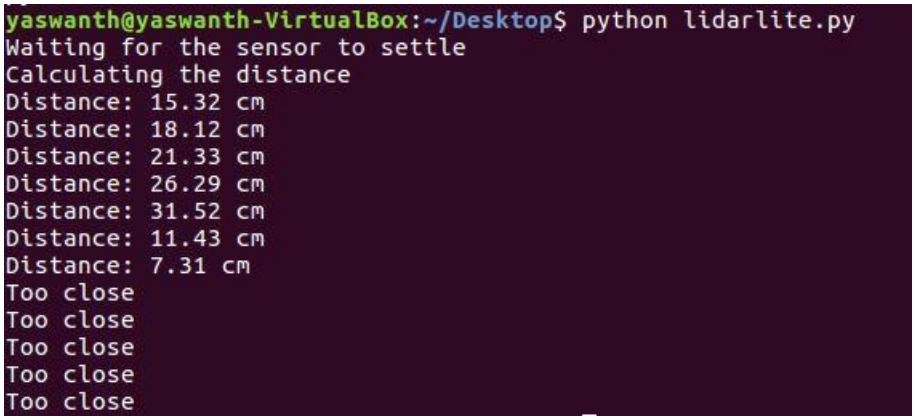


Figure 13: The result obtained from LIDAR Lite.

Chapter 4

Internal Architecture

Connecting the internal components and configuring the software modules serve together as the backbone of the whole communication process. An RPi monitors movements onboard the vehicle and communicates with the Pixhawk over a low-latency connect. Applications running on the RPi can perform computationally time-delicate and challenging errands. The RPi includes significantly more information than what is given by the Pixhawk alone.

DroneKit is a software development kit that helps develop applications for drones. It can be utilized with onboard computers running variations of Linux that help both Python and the establishment of Python bundles from the internet. DroneKit allows the RPi to decipher complex codes and process it to the flight controller. It contains a library of predefined basic functions, which can ease work with scripted missions.

MAVProxy is a fully functioning ground control station for drones that runs on the RPi. Using MAVProxy, a protocol for communication is created, known as MAVLink. MAVProxy is used to set up a connection between the RPi and the Pixhawk. Establishment of all these connections and their working conditions are well discussed in the sections below. Fig. 14 features the blueprint of the whole communication process.

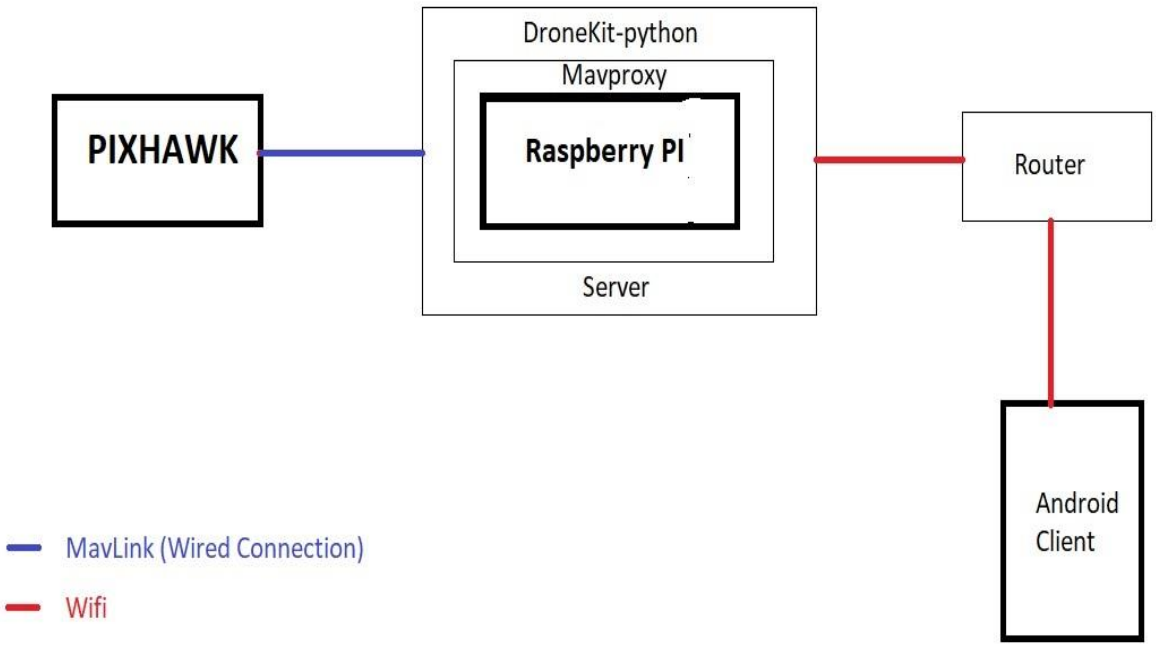


Figure 14: Internal networking between client and Pixhawk.

4.1. Proposed Connection Path

The connection between the components plays a vital role in communication between the user and the drone. The communication begins with the user accessing the wireless network. It is then carried between various components on the drone with the help of hard wired connections. After the message from the user is received by the Raspberry Pi, it is then sent to the flight controller (Pixhawk (PX4)).

The Pixhawk and the Raspberry Pi communicate with each other with the help of MAVProxy. MAVProxy is a powerful command-line based “developer” ground station software that complements popular GUI ground station, such as Mission Planner, APM Planner, etc. [23]. A key element of MAVProxy is its ability to forward the messages from the drone over the system to numerous other ground station by means of UDP. For instance,

a ground station can run on a workstation alongside the other receptors. Then, one can send information through *Wi-Fi* to a cell phone or tablet, which allows the controller to effectively maneuver the drone.

4.2. Installation of MAVProxy

Before proceeding, there are some software packages that need to be installed on the Raspberry Pi. The following commands will help complete the installation of MAVProxy and other supporting packages [25].

sudo apt-get install screen python-wxgtk2.8 python-matplotlib python-opencv

python-pip python-numpy python-dev libxml2-dev libxslt-dev

sudo pip install future

sudo pip install pymavlink

sudo pip install mavproxy

4.2.1. Configuring MAVProxy to always run

It is important to make sure MAVProxy is always running when the server starts. This is the only way the message from the user can reach the flight controller as soon as the drone is powered on [26]. As a result, the flight controller will always be able to listen to the RPi. To enable MAVProxy when the RPi is restarted, two additional steps need be

taken. First, one must open a terminal window and edit the **/etc/rc.local** file. Then, add the following lines just before the final "exit 0" line:

```
(
date
echo $PATH
PATH=$PATH:/bin:/sbin:/usr/bin:/usr/local/bin
export PATH
cd /home/pi
screen -d -m -s /bin/bash mavproxy.py --master=/dev/ttyAMA0 --baudrate 57600 --aircraft MyCopter
) > /tmp/rc.log 2>&1
exit 0
```

Figure 15: Setting up MAVProxy.

When the RPi connects to the Pixhawk, three files will be created in the /home/Pi/MyCopter/logs/YYYY-MM-DD directory:

- mav.parm : a text file with all the parameter values from the Pixhawk
- flight.tlog : a telemetry log including the vehicle's altitude that can be opened using the mission planner
- flight.tlog.raw: a raw data file with the .tlog mentioned above and any other serial data received from the Pixhawk

If one wishes to connect to the MAVProxy application that has been automatically started, the following command should be entered by logging in into the RPi:

sudo screen -x

The physical connection between the Raspberry Pi and the Pixhawk can be made in two ways. This can be done through a USB cable or a wired connection between the GPIO ports on the RPi to the telemetry port on the Pixhawk. The former is more reliable due to

its quality of communication. A wired connection with GPIO ports is more prone to interference which may present undesired results.

4.3. Dronekit – Python

DroneKit-Python enables developers to make applications that run on a RPi and communicate with the ArduPilot flight controller utilizing a low-idleness interface. Locally available applications can essentially improve the autopilot. These applications add more prominent knowledge to vehicle conduct and perform errands that are computationally escalated or time sensitive (PC vision, way arranging, or 3D displays). DroneKit-Python can likewise be utilized for ground station applications by speaking with vehicles over a higher dormancy RF-connect [27].

The DroneKit speaks with the Pixhawk through the MAVLink protocol. It gives automatic access to an associated vehicle's telemetry, state, and parameter data. MAVLink empowers both mission administration tasks and direct control over vehicle development [28]. DroneKit-Python, a Python based version of DroneKit chosen for this thesis, works better with vehicles that impart utilizing the MAVLink convention. It can run on Linux, Mac, or Windows.

DroneKit-Python provides the following classes and methods to:

- Connect to a vehicle (or multiple vehicles) from a script
- Set vehicle state/telemetry and parameter information
- Receive asynchronous notification of state changes
- Guide a UAV to specified position (GUIDED mode)

- Send custom messages to control UAV movement
- Create and manage waypoint missions (AUTO mode).
- Override RC channel settings.

4.3.1. Installation

DroneKit-Python and the Dronekit-SITL simulator are installed on the RPi from a software package management system known as *pip*. Here, DroneKit-SITL simulator is installed as a companion support package for the DroneKit-Python. The use of DroneKit-SITL is shown in Chapter 5.

On Linux, the first step is to install *pip* and *python-dev* with the help of the following command:

sudo apt-get install python-pip python-dev

Then, *pip* is used to install DroneKit-Python and DroneKit-SITL. Mac and Linux may require a prefix to these commands such as "*sudo*".

pip install dronekit

pip install dronekit-sitl

Upon installing DroneKit-Python, there are a few general considerations. The first is that some orders might be quietly overlooked by the autopilot. If this is the case, it is in a state in which it cannot securely follow up. Secondly, messages are interrupted or not sent

most of the time. The last point to consider is that commands can be received by the autopilot from numerous sources.

4.4. Creating Server Client Connection

In this case, the server will be the Raspberry Pi, and the client will be an Android mobile phone. The communication is accomplished using a socket connection between the two. This type of connection with a drone has various advantages. For instance, in the future, if the drone must be operated from any far distance, communication with the drone can persist for as long as the server and client are connected to the internet [29].

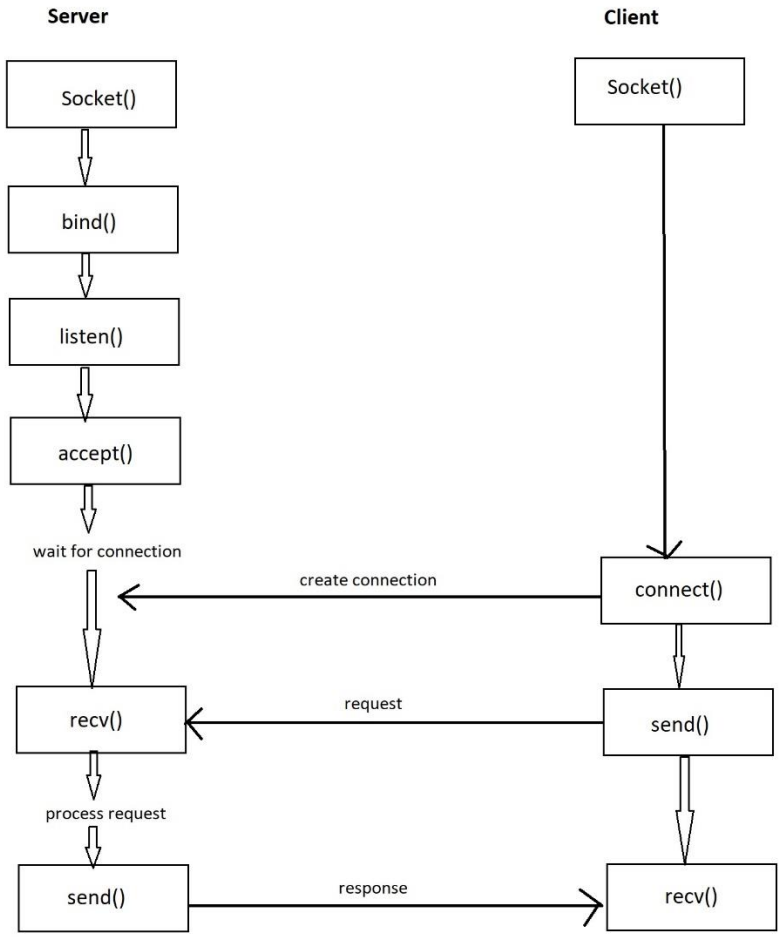


Figure 16: Working principle of a socket connection.

Here, a socket connection is used between the server and the client. A socket is one end-point of a two-way communication link between two programs running on the network. This type of connection can make the communication worthy because there is a lot of information flowing in both the directions. Most of the application-level protocols like FTP, TCP, and UDP make use of sockets to establish connections between a client and a server to exchange data.

4.5. Creating a Server on a Raspberry Pi

Here, a Python server running on the RPi is created. This is the most important part of the setup, as it is where major information exchanges take place between the client and the server. The received information by the server is processed, followed by triggering of the necessary protocol for a proper flow of information to the drone. This server functions in such a way that it always listens to the client, even when other messages are still processing. Every time a client request is received by the server, it initiates a sub-process. This sub-process executes in a Unix subshell. The server should be designed in such a way that there are "priority" based requests from the client. If a high priority request has been placed by the client, then the server should terminate or pause all the existing processes and initiate the requested one. All the challenges and sub-process modules used in creating this server is explained in detail in Section 4.4.

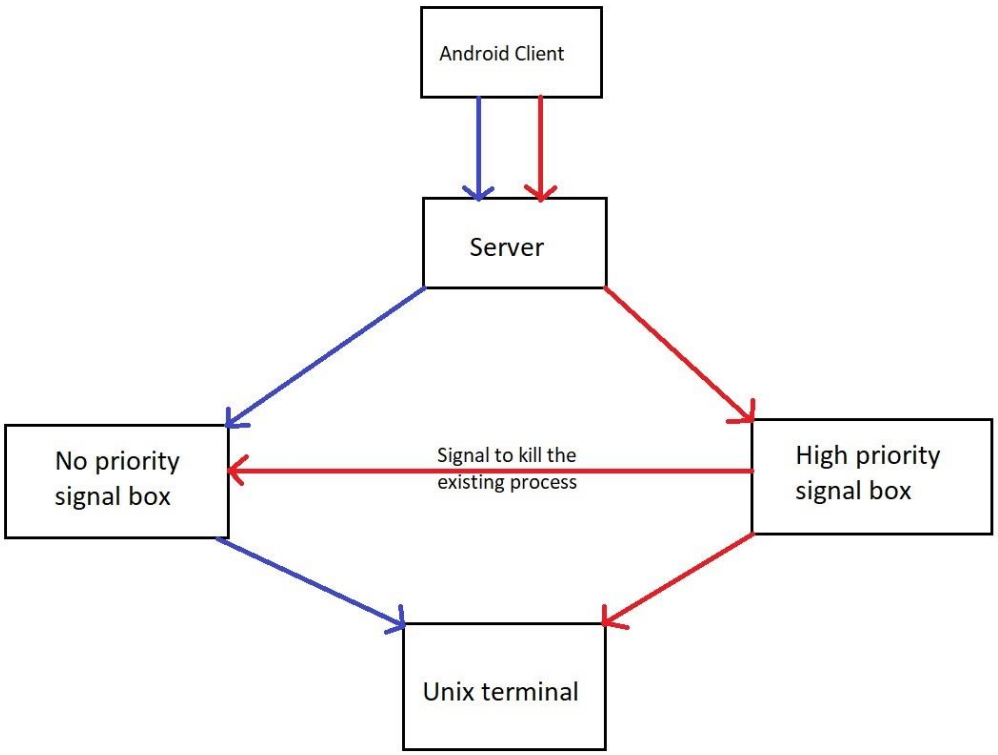


Figure 17: Handling of messages on the server.

4.6. Developing an Android Client

A client can be any source that can make some request to the server. The reason for choosing Android as the client in this case is due to its developer friendly nature. The client makes requests to the server using socket connections. Developing the client using an Android device is used because of its in-built speech recognition skills. Every Android device is supported by a well renowned speech recognition engine called “Google voice engine”. Taking advantage of this speech engine skills, one can append this feature to the client application to collect speech requests from the users. This Android application is developed using an Integrated Development Environment (IDE) called Android Studio.

The figures below give an idea on how the client sends the information to the server:

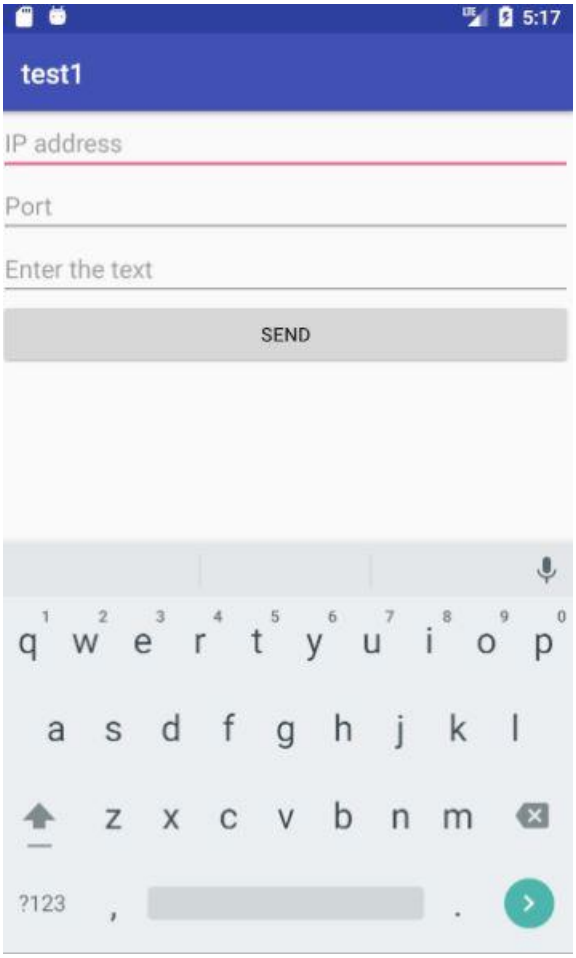


Figure 18: An android client interface.

Fig. 18 shows that the user can use basic keyboard input methods to enter values.

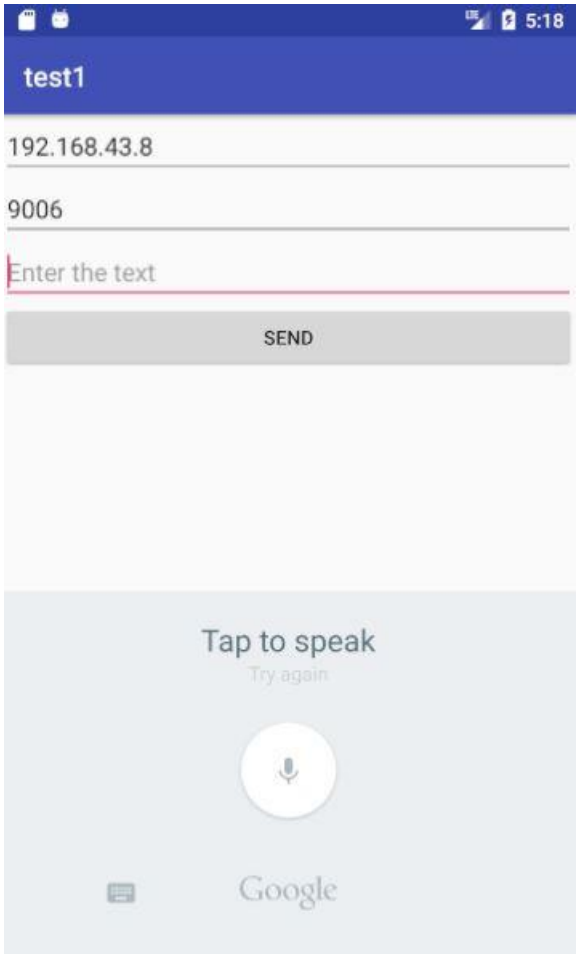


Figure 19: Speech input on an Android client.

Figure 19 contains the following provisions for users:

- IP address field: contains the IP address of the server
- Port Number field: contains the port that is open to the particular IP address
- Text field: allows the user to type in the requests for the server
- Mic symbol: allows the user to use voice commands

4.7. Challenges

This section describes the solutions to some complications which occurred while developing the above results.

4.7.1. Use of Sub-process Module to Trigger a Sub-process

The sub-process module will enable running multiple processes at the same time. These processes, known as sub-processes, run as completely independent entities, each with its own private system state and main thread of execution. Since a sub-process is free, it executes simultaneously with the first procedure. The procedure that made the sub-process can work ahead to deal with different things, while the other processes are working in the background.

In this case, the server is trying to evoke a script stored on the local drive of the server. Upon calling the script, a proper function should be used to ensure that the run is successful. Execution of this local script is an example of a sub-process used. These sub-processes must be handled carefully because the control of the sub-process is handled by various functions. Some of the functions used to call sub-processes are:

- os.system()
- os.popen()
- os.spawn()
- os.call()

When these functions are used, the command that initiates the sub-process will be inserted inside the parentheses of the submodule function.

4.7.2. Control over the Sub-process Module

Getting control over the sub-process is critical. Usually, when a sub-process is called, the parent process remains in the background and resumes once the child process has completed its job. There are some situations demanding the parent process to be killed to proceed with the child process. Such control over the sub-process can only be achieved by using the proper calling functions. All the parent processes are called by creating an active pipe connection. *Popen*() function is used here, which unlike the *call()* function, satisfies the above requirements. *Popen*() *d*oesn't restrict the parent process, allowing the user to interact with the parent process while it's running. This gives freedom to continue with other things in a Python program. *Call()*, does restrict the parent process. While it supports all the same arguments as the *Popen()* constructor, one can still set the process' output, environmental variables etc. The script waits for the program to complete, and *call()* returns a code representing the process' exit status.

4.7.3. Achieving immediate LAND command

The necessity of an immediate *LAND* helps to rescue the drone in case of any emergency. This section discusses how to kill the parent process and initiate a child process. The importance of the *LAND* command is crucial, as there is always a chance of force landing a drone due to unavoidable crash scenarios. This can be achieved when the *LAND* command on the server is written under a separate condition. When a user calls for an immediate landing, the server kills the existing parent process and initiates a child

process called *LAND*. Since all the parent sub-processes are called using the *Popen()* function, the main server always listens to any new messages. This gives an advantage to a user to call any high priority commands such as *LAND*. This command takes control of the drone at any given point of time, thereby adding a major advantage to its controllability.

Chapter 5

Simulation Setup

While testing a quadcopter, one must be careful and take all the necessary safety precautions to avoid any damage to the surroundings. To understand the UAV's behavior, the quadcopter software is first tested on a simulator. Simulation is always the best way to start examining the behavior of anything. To make this possible, one must set up a simulated vehicle. The Software in The Loop (SITL) simulator allows developer to create and test DroneKit-Python apps without a vehicle. It can be installed on the same computer as DroneKit or on any other computer on the same network. The following subsection discusses how to introduce and run SITL as well as how to interface with DroneKit-Python and ground stations.

5.1. DroneKit-SITL

DroneKit-SITL is the least complex approach to run SITL on Windows, Linux (x86 only), or Mac OS X. It is installed from Python's pip tool on all platforms It works by downloading and running pre-built vehicle binaries that are appropriate for the host operating system.

- **Installation:** The following command allows the user to install the DroneKit-SITL:

pip install dronekit-sitl -UI

5.1.1. Running SITL

Here, one also gets to decide what kind of drone simulator is requested by the user. Some examples of different kinds of drone simulation software are Copter, Plane, My Copter, Etc. After choosing a software, the user can also set the current location as the home location for the quadcopter. The command used to set and initiate the drone simulation with the current location is shown below:

dronekit-sitl copter –home=41.656946, -83.606942,0,180

This will lead to the following results:

Figure 20: Running DroneKit-SITL.

5.1.2. Connecting to DroneKit-SITL

DroneKit-SITL waits for a protocol known as Transmission Control Protocol (TCP) when connecting. Then, TCP is associated with the following IP and port address: 127.0.0.1:5760. The contents of DroneKit-Python running on the RPi can interact with

TCP utilizing the association with this IP and port address. The following command shows the syntax for a successful connection:

vehicle = connect('tcp:127.0.0.1:5760', wait_ready=True)

To broadcast the information to multiple ground stations, a user needs to set a particular device as the master device with the help of MAVProxy. This is to make sure the device under the master category gets the highest priority. Every device connected is assigned an IP address along with their port numbers. An example of this working connection is shown below in Fig. 21.

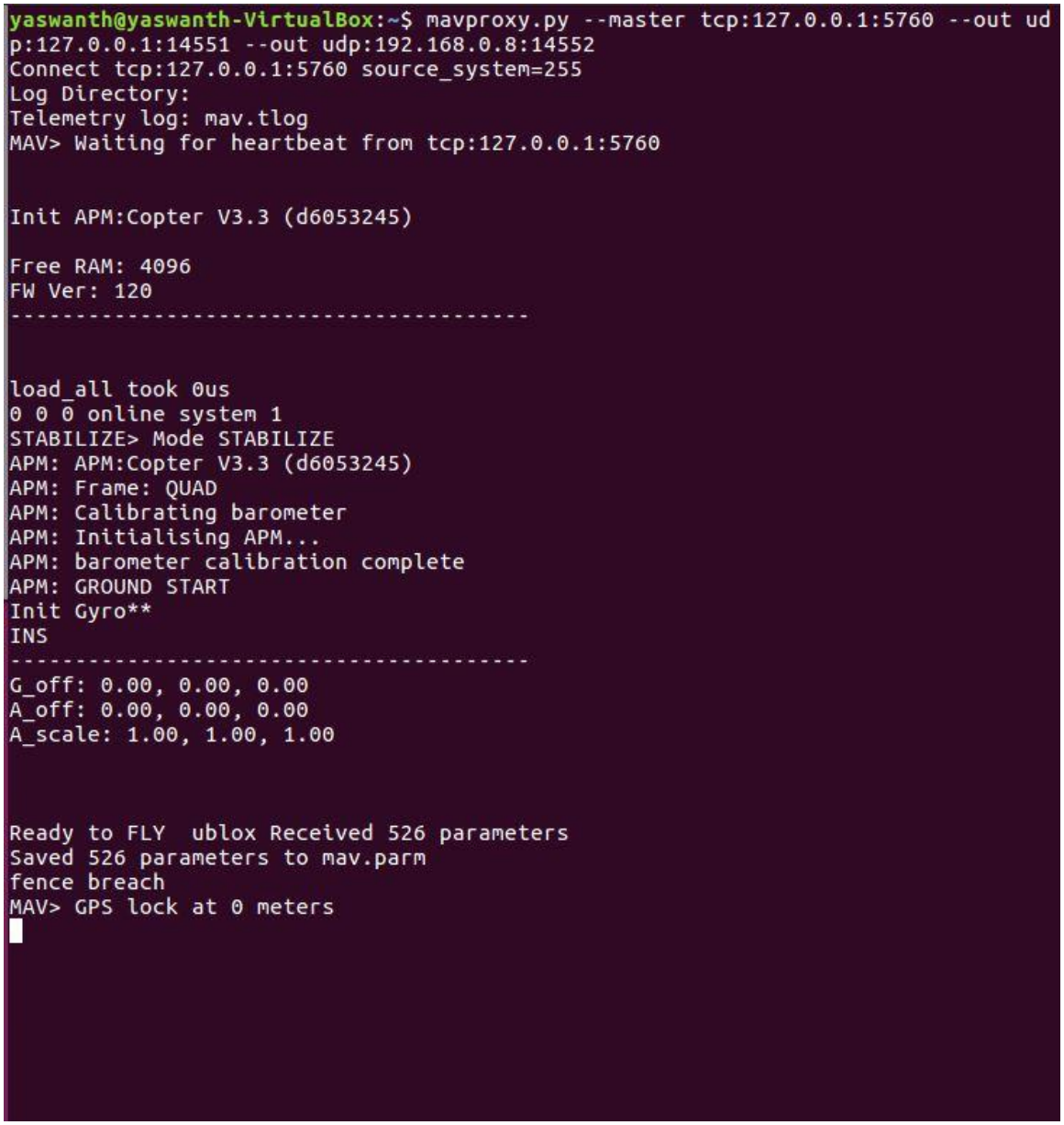


Figure 21: Running MAVProxy to create a link with the simulator.

Now the test code for the drone can be executed, and the drone's simulation can be observed on any ground station monitor. In this case, the basic code is written to initiate simple take-off and landing of the drone.

5.1.3. Code Schema

The vehicle connects by setting "*wait_ready=True"* to ensure that the vehicle is already populated with attributes when the *connect()* returns the following:

---

from dronekit import connect

 # Connect to the Vehicle (in this case a UDP endpoint)

vehicle = connect('REPLACE_connection_string_for_your_vehicle',

wait_ready=True)

After the connection has been established with the drone, about a need arises to determine the launch sequence of the drone. Consider the standard launching sequence described in "*Taking Off".* This is a predefined function from the DroneKit library, which consists of the following procedure:

- ensure *Vehicle.is_armable* is set
- set the *Vehicle.mode* to GUIDED
- set *Vehicle.armed=True* and poll on the same attribute
- call *Vehicle.simple_takeoff* with a target altitude.
- poll on the altitude and allow the code to continue only when it is reached.

This approach guarantees that commands are sent to the vehicle, where it can be successfully executed (e.g. knowing that Vehicle.is_armable is True before endeavoring

to arm, knowing that Vehicle.armed is True before we take off). It thereby makes troubleshooting departure issues much less demanding [30].

Once the drone satisfies and completes all the defined system calls in the "*Take Off"* function, it is ready to move to any desired point (e.g. move to waypoint 1, send it on a mission). The script can also be put to sleep when not in use, which can reduce the CPU overload. At low speeds, one may just need to check only when an objective at regular intervals is achieved. Using the *time.sleep()* function provides accurate reaction time from the drone and makes the flight proficient.

Once the drone is done with the mission or time of flight, the script tries to change the state of the drone from "*GUIDED MODE"* to either "*RETURN TO HOME MODE*" (RTL MODE), or "*LAND MODE*". These modes will make the drone reach a safe location and land. Thereafter, the system can be brought to rest by shutting down the vehicle, by calling the function: *Vehicle.close().*

Chapter 6

Results and Observations

The results are classified into two different segments. The simulation results show the behavior of the software with respect to an ideal drone conditions. The real-world results reveal the factors that can affect the drone's flight.

6.1. Simulation Results

The simulator helped in testing the software confidently without bringing any physical damage to the drone. The simulator always presents the results based on ideal conditions. Keeping in mind the setup discussed in Chapter 5, the following figures in this chapter will give an idea on how the drone behaves when it is autonomous. An additional software known as *TOWER* [31] was used to monitor the travel path and other parameters of the flight. *TOWER* [32] was installed on the Android phone from the Android Play Store. It gives the geographical location of the drone in real time along with the altitude reached and the state of the drone. The communication with the Android application and the drone is made through a UDP connection. This section details the virtual behavior of the drone, as the result of a user's request of a mission from an Android device.

Three tests were performed using the simulator. The first mission shows only the takeoff and landing of the drone with certain amounts of wait time in between. The second involves the drone moving towards a set waypoint for some given time and returning to

the home location. The final mission tests the immediate land function for the drone during mission.

Test 1: Results for a simple takeoff and landing of the drone.

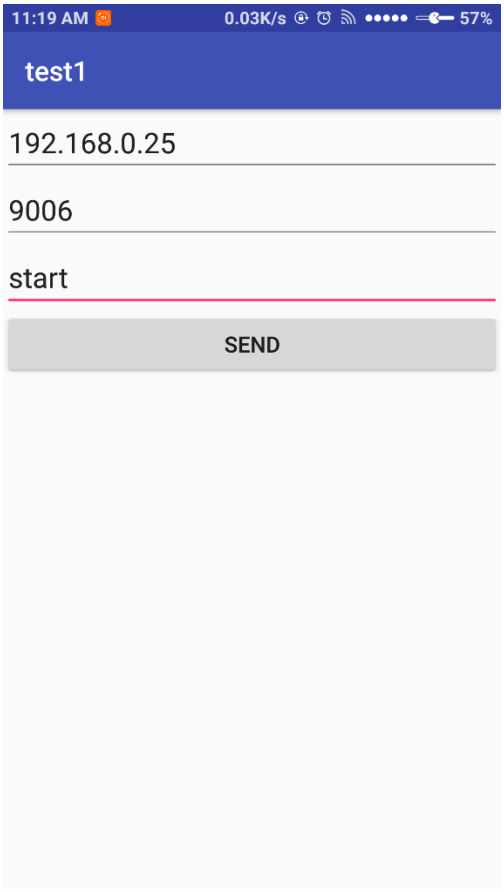


Figure 22: An Android client sending "start" command.

In this test, observation of basic take off and land can be seen below:

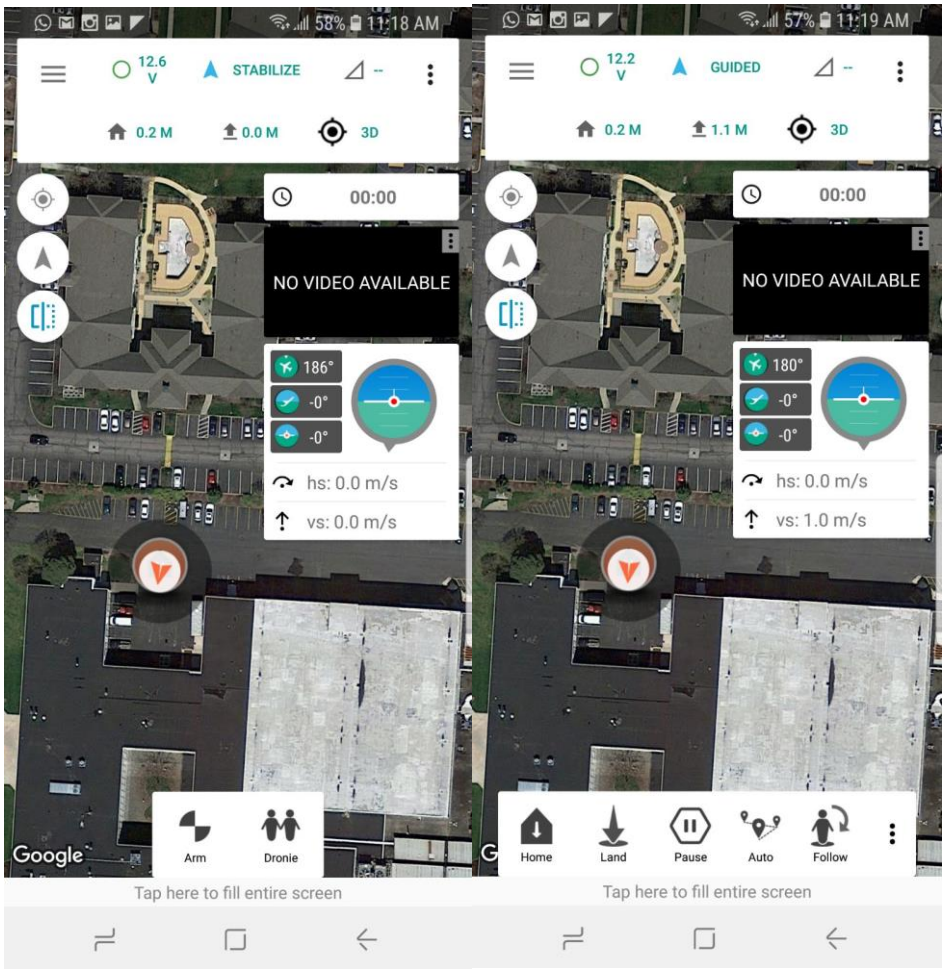


Figure 23: (left) The drone getting ready to take off. (right) The drone reaching the required altitude and is in GUIDED mode.

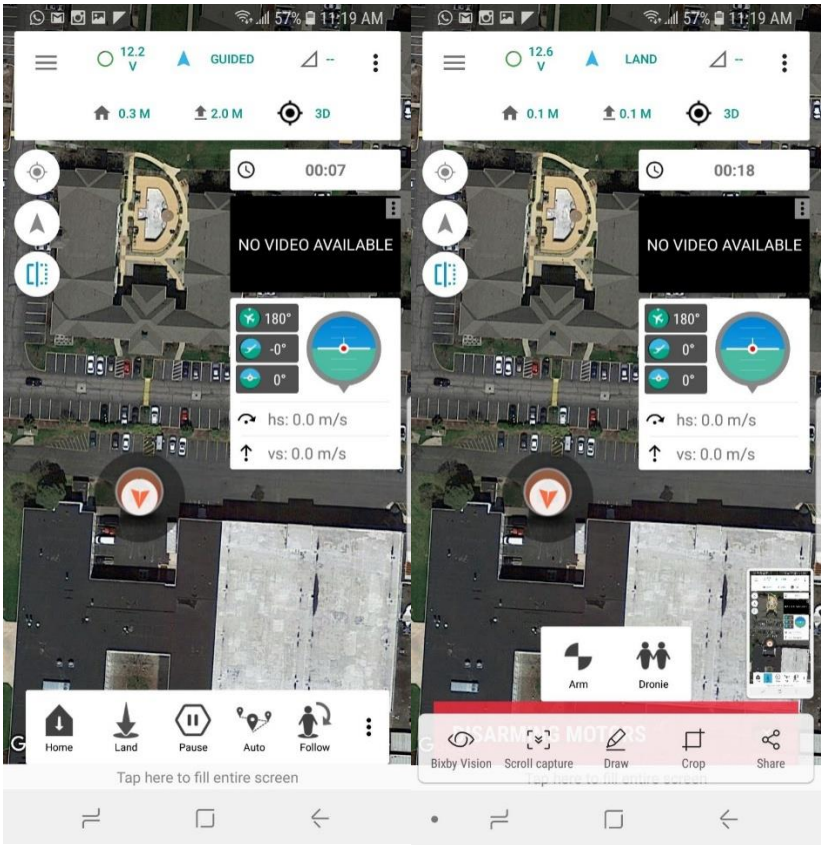


Figure 24: (left) The drone reaching the target altitude. (right) The script automatically trying to land the drone.

```
yaswanth@yaswanth-VirtualBox:~$ dronekit-sitl copter --home=41.656946,-83.606942
,0,180
os: linux, apm: copter, release: stable
SITL already Downloaded and Extracted.
Ready to boot.
Execute: /home/yaswanth/.dronekit/sitl/copter-3.3/apm --home=41.656946,-83.60694
2,0,180 --model=quad
Started model quad at 41.656946,-83.606942,0,180 at speed 1.0
bind port 5760 for 0
Starting sketch 'ArduCopter'
Serial port 0 on TCP port 5760
Starting SITL input
Waiting for connection ....
bind port 5762 for 2
Serial port 2 on TCP port 5762
bind port 5763 for 3
Serial port 3 on TCP port 5763
Hit ground at 0.507120 m/s
```

Figure 25: The simulator result of the landing speed of the drone.

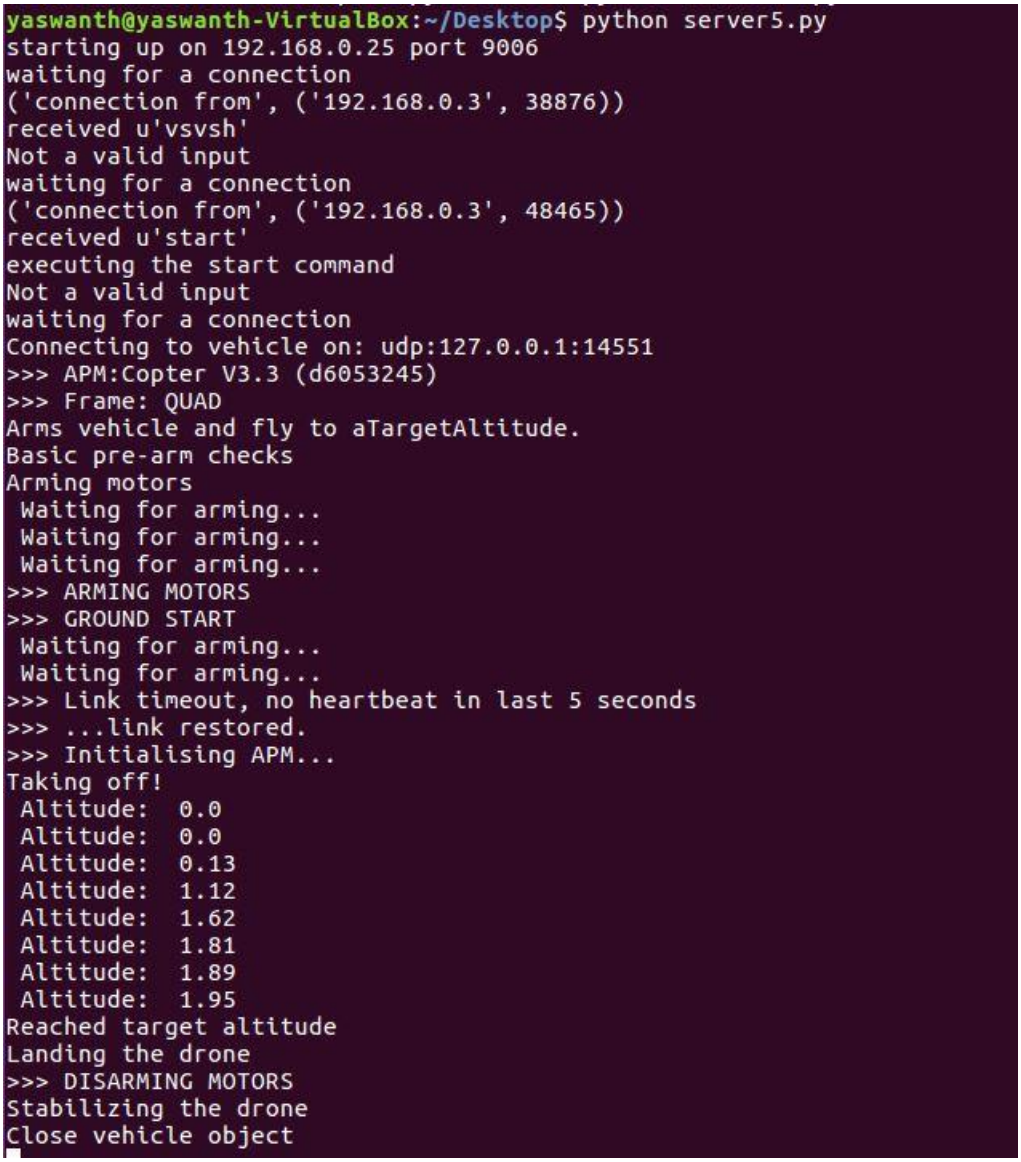


Figure 26: Complete command line results of the first test.

Test 2: Moving the drone to a desired waypoint and returning it to its home position.

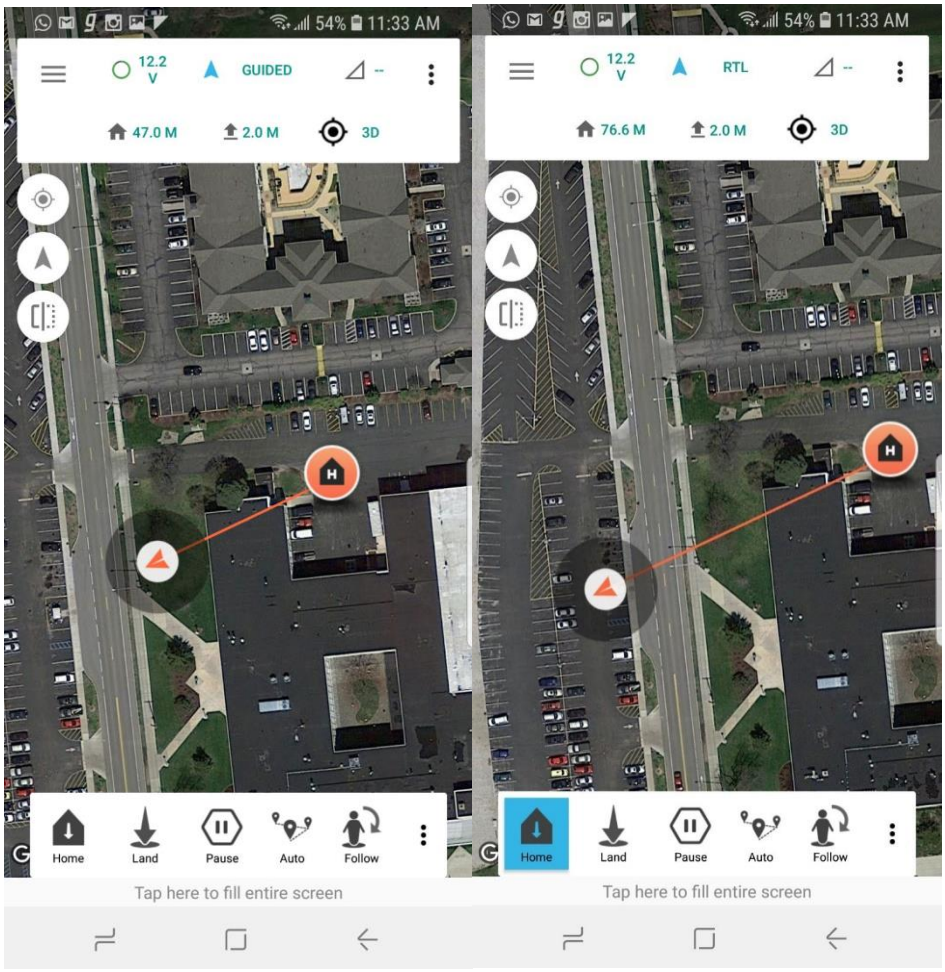


Figure 27: (left) The drone taking off and moving towards a waypoint for a given amount of time. (right) The position of the drone at the end of 30th second and the change of mode to RTL.

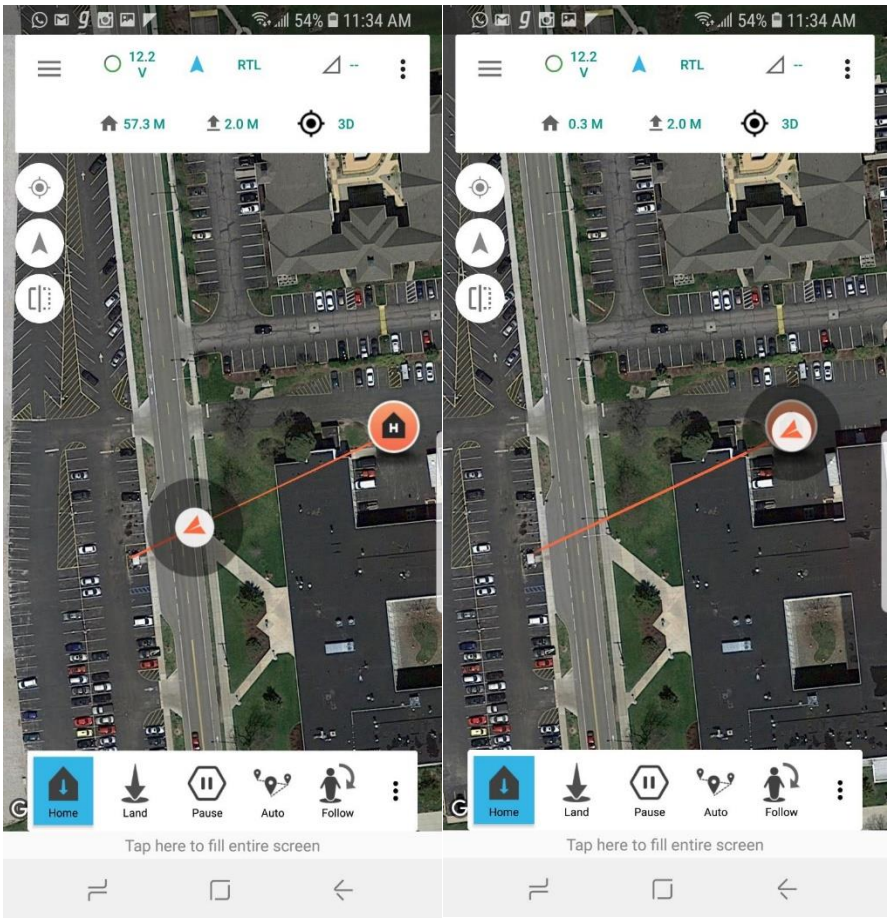


Figure 28: The change of mode from guided to RTL, the drone seen heading towards its home location.

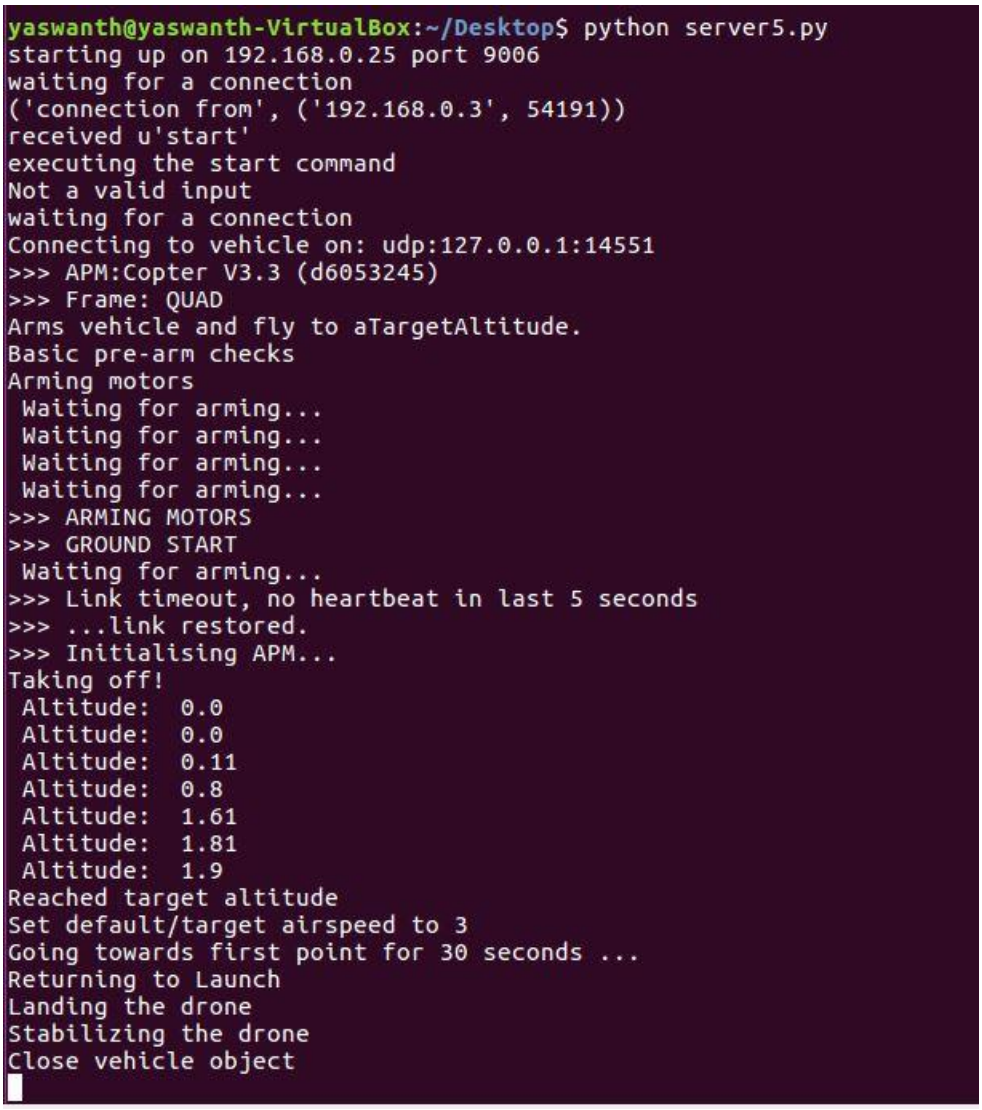


Figure 29: Complete command line results of the second test.

Test 3: How the drone behaves when a mission is interrupted and is forced to land.

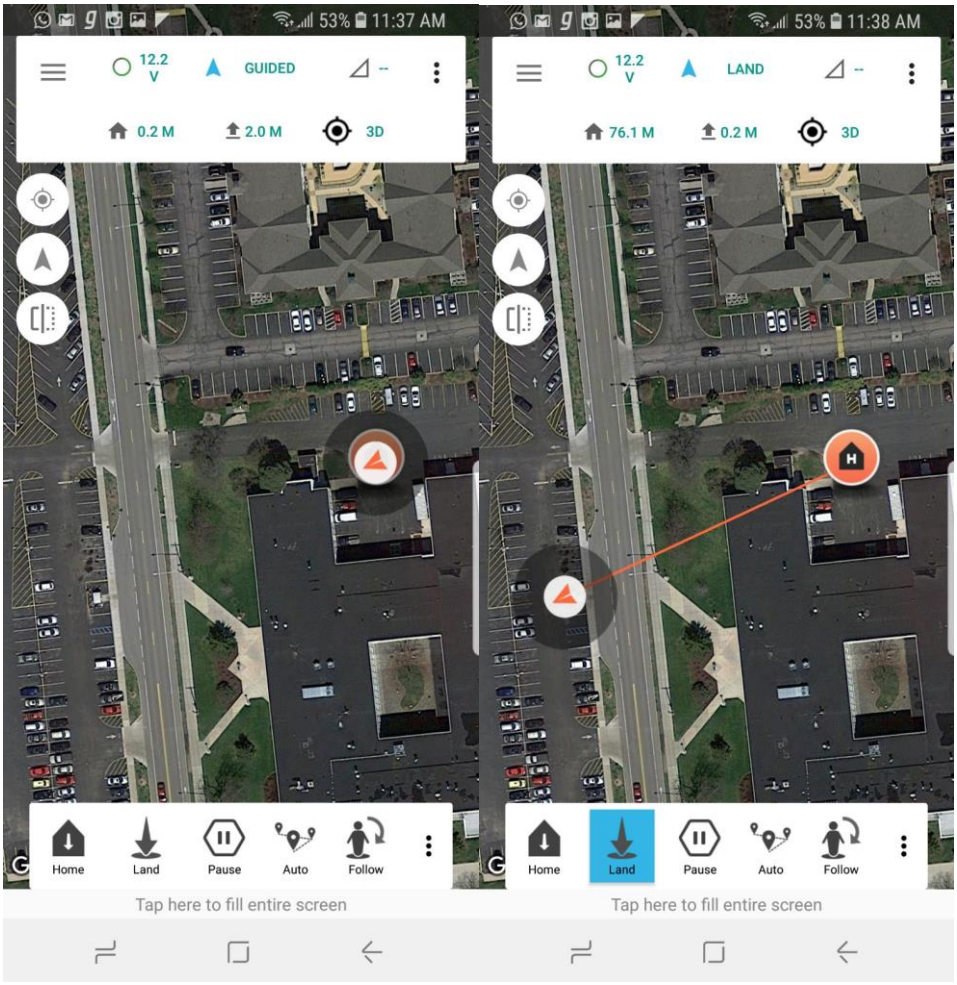


Figure 30: The drone being instructed to move towards a waypoint and come to its home location, but being interrupted by LAND command forced to land.

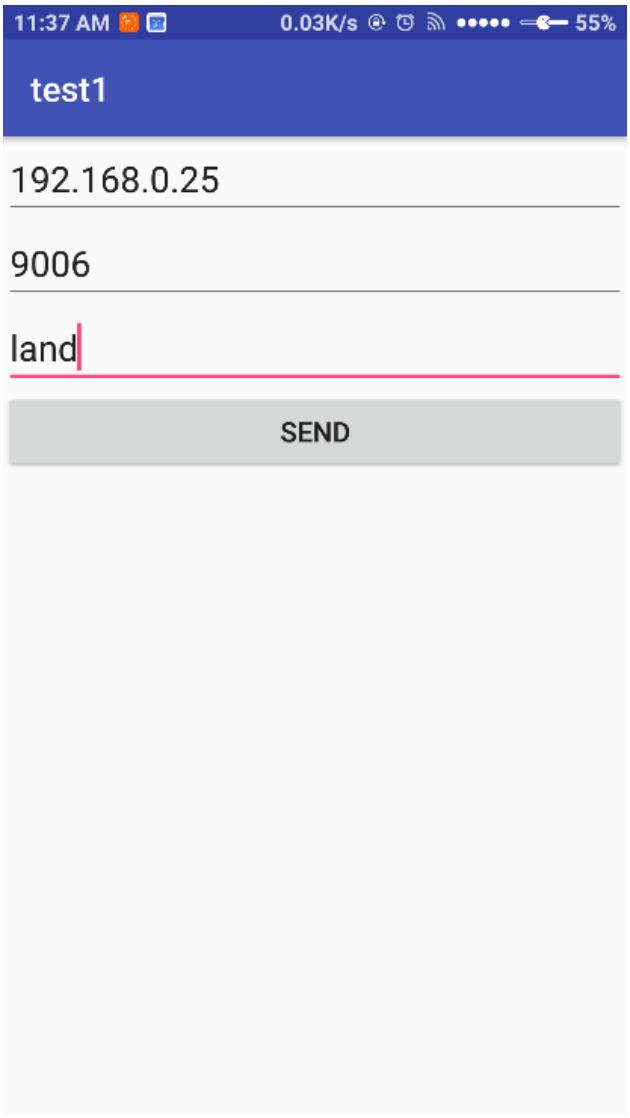


Figure 31: The LAND command from the user, which was used to interrupt the drone's mission.

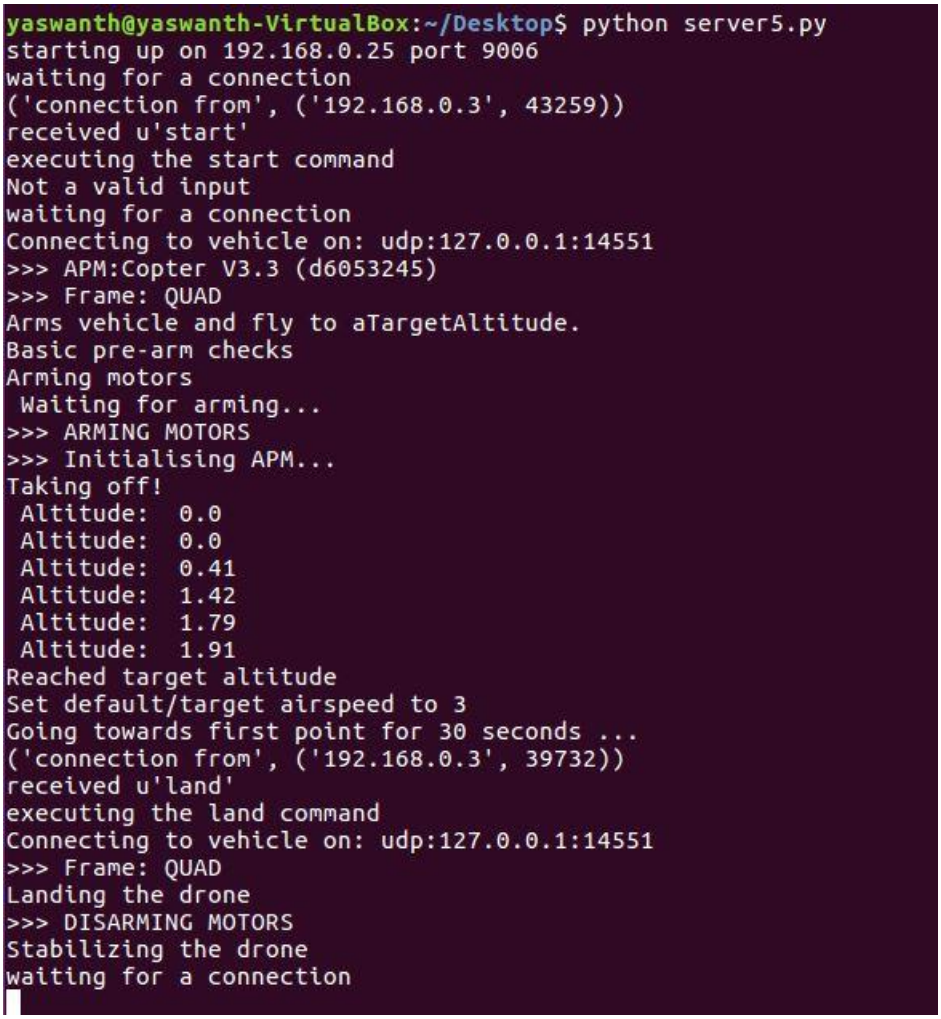


Figure 32: Complete command line results of the third test.

6.2. Real World Results

This section discusses the implementation of ideas tested on the simulator. Before directly testing the software on the drone, the drone could have been flown manually with the help of a remote controller. This was tried by flying the drone with the *Angular V Tail* drone, which proved unsuccessful. It has some weight distribution problems, which did not allow the propellers enough thrust for the drone to takeoff. The drone also had a heavy

battery with an inefficient discharge rate, which failed to supply enough power to the propellers when it is under load. The *Angular V tail* model is a failure in its design.

The next attempt was with the *Quad-X* model, which was tested by flying the UAV manually with help of a remote controller, which proved successful. The flight was stable, and the propeller sizes were upgraded. Larger propellers provided more thrust to the drone and had lower rotations per minute. Since the flight was stabilized, this model served as the basis for testing the software.

Initially, the software was used only to land the drone, meaning that the drone is taken into the air using the remote controller. After reaching a height, the drone was to be landed using the land command. It was quite successful for a good number of attempts until the same experiment was repeated at a much higher altitude. Suddenly, it started drifting in an undesired direction when the drone was prompted with the land command. This experiment resulted in a severe crash, and the drone was destroyed. There are several reasons that might have caused the crash. These reasons include: strong air pushing to the drone on one side, a bad GPS signal, and multiple inputs at an instance.

Therefore, the real-world testing process evolved to lifting the drone physically and giving it a virtual flight situation. This was done to avoid unnecessary crashing of the drone. Simple scenarios served in the testing process, which primarily included taking off and landing the drone. When a signal was sent to the drone for takeoff, the drone was lifted to a required height in anticipation of the next signal. Once the land command had been sent, the drone was manually landed on the ground. The result for this test is shown below.

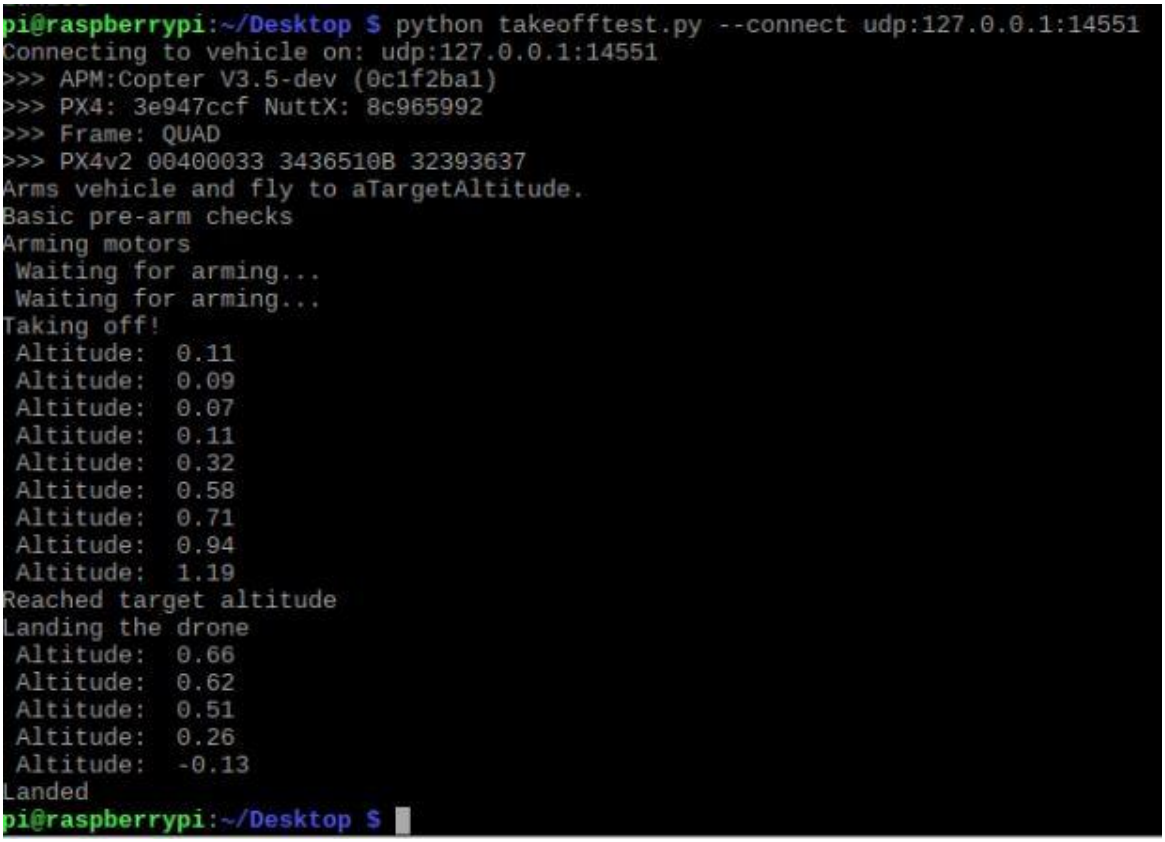


Figure 33: Complete command line results of real world test.

Even though it is a biased testing of the drone, there are some conclusions from this experiment. The communication between the user and drone is done without any loss, and the script was executed perfectly.

Chapter 7

Conclusions, Points of Weakness, and Future Work

7.1. Conclusions

This research suggests a better method of communication between a drone and its user. Voice control of a drone ensures that the user has privacy and reduces the need for a complex hand controller. Most Android smartphones can exhibit voice control for a drone by installing some developed applications. However, complete control of the drone is still not accessible to the user. The user can only control some basic movements and predefined missions loaded in the drone.

Two different drone models were involved in testing the controllability using the software. Several issues related to the design and hardware were noticed in both the models. Angular V tail model had an inappropriate weight distribution issues, and a complex system to cope with. The Quad-X model was more successful and was more stable during its flight. The results from the simulator were successful with some promising scenarios. Whereas the real-world testing scenarios always involved with some biased

conditions. Nonetheless, external usage of the software is still not recommended due to various real-time factors. Reliable hardware components will eliminate any inconsistencies and can lead to more accurate and unbiased results.

7.2. Points of Weakness

Though the results demonstrated some successful feedback with the software, there are other factors that affected some segments of this thesis.

- There are major limitations on hardware components. The components used were not reliable and took longer to fix.
- The project was started with a complicated drone system (*Angular V tail*), which involved weight distribution problems.
- A bad GPS signal can always add an unexpected drift to the drone.
- Unexplainable drifting (shown in fig. 34) of the drone is seen when the drone is connected to any ground station like *Mission Planner* or *Tower*.

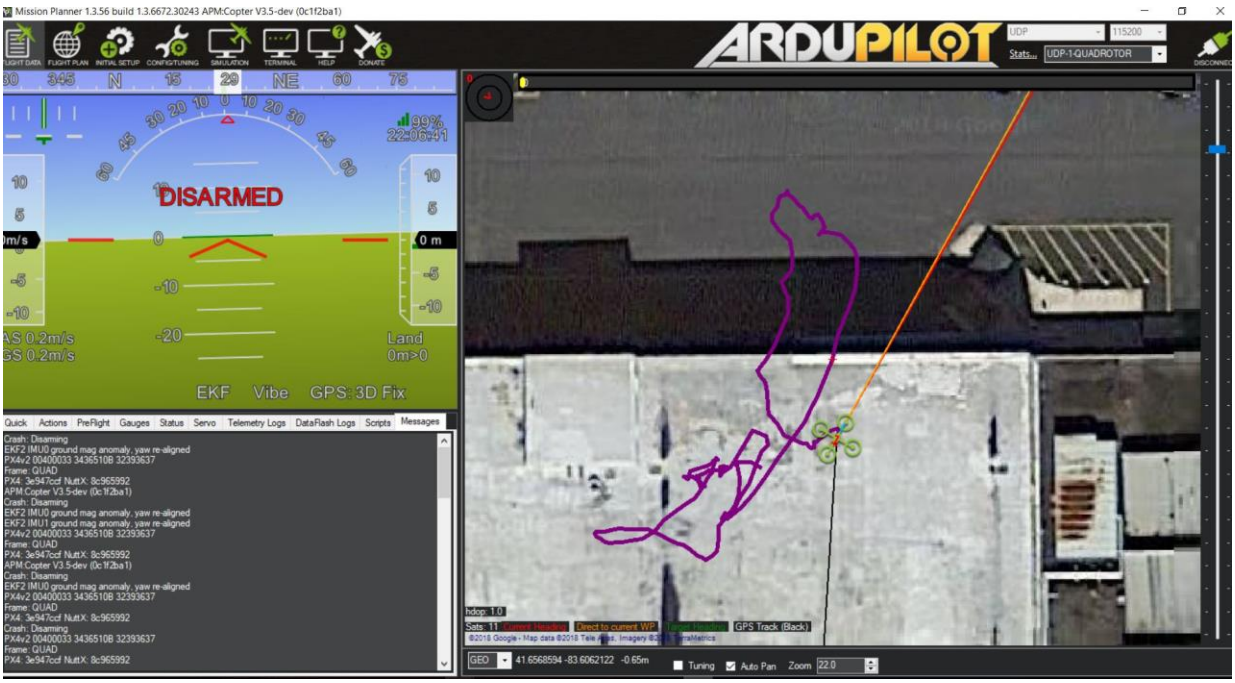


Figure 34: The figure shows the drifting problem generated by a stationary drone.

7.3. Future Work

The results of this research can be used as the building blocks of several possibilities for future work. Eliminating some basic issues, like complex drone design and the usage of unreliable hardware components, can make improvements to the current scenario. With respect to the software, the Android application can be upgraded with more luxury features. One such feature could be real-time feedback of the drone by showing its location on a map. The connectivity between the user and the drone can be upgraded from a local area network to a mobile data communication. This gives the freedom of controlling the drone from anywhere in the world. There is also a scope of working on the privacy of the connection. Adding a security feature to prevent the messages shared between the Android device and the quadcopter from being intercepted by unauthorized devices will provide an

additional level of protection. Providing more freedom of motion to the drone and helping avoid any unconditional landings can be achieved with additional sensors. These sensors also have the option of helping the drone avoid any obstacles and give the drone an “eye” to operate. Future steps could include building on all these possible upgrades and making the drone more independent and user friendly. The process can be endless and yet can lead to many possibilities in the future.

References

[1] J. P. Hansen, A. Alapetite, I. Scott Mackenzie and E. Mollenbach, "The use of Gaze to Control Drones," *ACM Digital Library*, 2014.

[2] C. Synan, "Voice Controlled Drone with RasPi, Amazon Echo and 3DR IRIS+," hackster.io,

2016. [Online]. Available: https://www.hackster.io/veggiebenz/voice-controlled-drone-with-raspi-amazon-echo-and-3dr-iris-c9fd2a.

[3] C. Rubens, S. Braley, A. Gomes, D. Goc, X. Zhang, J.-P. Carrascal and R. Vertegaal, "BitDrones: Towards Self-Levitating Programmable Matter via Interactive #D Quadcopter DIsplays," *antpgomes, squarespace,* 2015.

[4] A. Heath, S. Narayanan and A. Draper, "Human Interaction with Levels of Automation and

Decision-Aid Fidelity in the Supervisory Control of Multiple Simulated Unmanned Air Vehicles," *Presence: Teleoperators and Virtual Environments,* vol. 11, no. 4, 2007.

[5] V. Dennis A., T. Brent A. and C. David, "Unmanned Aerial System (UAS) Human-machine

Interfaces: New Paradigms in Command and Control," *Science Direct,* 2015.

[6] P. Hancock, "Automation: how much is too much?," *Ergonomics,* 2013.

[7] D. Cavett, M. Coker, R. Jimenez and B. Yaacoubi, "Human-Computer Interface for Control

of Unmanned Aerial Vehicles," *IEEE,* 2007.

[8] A. Kochan, "Automation in the sky," *Industrial Robot: An International Journal,* vol. 32, no. 6, 2005.

[9] S. O. Y. a. L. R. Scott Xiang Fang, "Development of Small UAS Beyond-Visual-Line-of-Sight

(BVLOS) Flight Operations: System Requirements and Procedures," *mdpi.com,* 2018.

[10] J. Lee, Y. Seong and k. yoonso, "Journal of the Korean Society for Aeronautical & Space Sciences," *The Korean Society for Aeronautical & Space Sciences,* vol. 44, no. 5, 2016.

[11] P. Wallich, "Arducopter Parenting," *IEEE Spectrum,* 2012.

[12] S. Cass, "Beyond the quadcopter: New designs for drones showcased at CeBIT,"

IEEE Spectrum, vol. 53, no. 5, 2016.

[13] C. P. S Jenkins, "Dynamic Loads on Airplane Structures," *SAE National Aeronautic and Air Transport Meeting* .

[14] "Techradar," 2016. [Online]. Available: https://www.techradar.com/how-to/computing /how-to-build-your-own-drone-1329806.

[15] "Flight controller," [Online]. Available: https://docs.px4.io/en/getting_started/flight _controller_selection.html.

[16] "INSTALLING OPERATING SYSTEM IMAGES," [Online]. Available: https://www. Raspberrypi.org/documentation/installation/installing-images/.

[17] A. Audet, "Linux Journal," 16 December 2014. [Online]. Available: https://www. linuxjournal.com/content/raspi-sump.

[18] A. Khoshnood, F. Khajemohammadi and S. Sina Zehtabchi, "Trajectory and control design of a quadcopter for crossing obstacles using sliding mode method," Tarbiat Modares University, 2016.

[19] S. S. J. W. A. M. a. R. S. Riccardo Polvara, "Vision-Based Autonomous Landing of a Quadrotor on the Perturbed Deck of an Unmanned Surface Vehicle," *mdp.com,* 2018.

[20] D. Wang and J. Lin, "Obstacle detection and avoiding of quadcopter," *SPIE Digital Library,* vol. 10457, 2017.

[21] M. Strip, "Setting up Lidar Lite 3 on Raspberry Pi 3," Wordpress.com, 26 December 2016. [Online]. Available: https://mobiusstripblog.wordpress.com/2016/12/26/first

-blog-post/.

[22] Garmin, "Garmin," [Online]. Available: https://www.google.com/search?q=lidar+lite&source=lnms&tbm=isch&sa=X&ved=0ahUKEwiOlLCdouzaAhUCj1kKHVTEBUkQ_AUIDCgD&biw=1536&bih=757#imgrc=hQLZp5JiSxiE3M:.

[23] 3DR, "Connecting pixhawk with Raspberry Pi," [Online]. Available: http://ardupilot.org/dev/docs/Raspberry-Pi-via-mavlink.html.

[24] "Getting the Raspberry Pi up and running with Raspbian," ardupilot.org, [Online]. Available: http://ardupilot.org/dev/docs/making-a-mavlink-wifi-bridge-using-the-Raspberry-Pi.html.

[25] 3DR, "Startup options for MavProxy," [Online]. Available: https://ardupilot.github.io/MAVProxy/html/getting_started/starting.html.

[26] ArduPilot, "ArduPilot," Dev Team, 2016. [Online]. Available: http://ardupilot.org/dev/docs/Raspberry-Pi-via-mavlink.html.

[27] 3DR, "Dronekit-python," [Online]. Available: http://python.dronekit.io/.

[28] "Raspberry udp issue," 2015. [Online]. Available: https://github.com/ArduPilot/MAVProxy/issues/121.

[29] Kenxpert1700, "PYTHON WEB SERVER FOR YOUR RASPBERRY PI," Instructables, 2015. [Online]. Available: http://www.instructables.com/id/Python-Web-Server-for-your-Raspberry-Pi/.

[30] 3. Services, "DroneKit," 3D Robotics, 2015. [Online]. Available: http://python.dronekit.io/1.5.0/examples/running_examples.html.

[31] 3. Services, "3DR DroneKit," 3D Robotics, 2015. [Online]. Available: http://android.dronekit.io/pebble_app.html.

[32] D. Labs, "Google Play Store," droidplannerlabs@gmail.com, [Online]. Available: https://play.google.com/store/apps/details?id=org.droidplanner.android&hl=en_US.