Firefly - Low Cost Drone Light Show

Final year Project Final Report

Michelle Cheng 100696572

Rodaba Ebadi 100708585

Toluwanimi Elebute 100724471

Munazza Fahmeen 100701595

Nivetha Gnaneswaran 100695935

Group 34

A final report submitted in partial fulfillment for the final year Capstone Project in the Faculty of Engineering and Applied Science.

Advisor: Dr.Lixuan Lu, P.Eng.

Coordinator: Dr. V.K.Sood, P.Eng.

Submitted to Ontario Tech University

April 6th, 2023

Abstract

This project aims to make drone light shows affordable and easily accessible by using low-cost materials and developing a mobile application that can control the drones' movements and sequences. This report will cover the various amounts of research and techniques used to implement the entire Firefly system. An overview of different ways a low cost drone light show can be implemented will be discussed as well as the testing and software implementation of the system will be outlined. The final product was created using the Tello EDU drone and swarm programming, as well as an application for users. A downsized drone light show was successfully made with various formations, concluding that a low cost drone light show is possible.

Dedication

To our families and friends.

Acknowledgements

We would like to express our gratitude to our capstone advisor Dr. Lixuan Lu for her guidance and encouragement throughout this project. Her knowledge and impactful insights aided us in successfully completing our project. We would also like to thank our capstone coordinator Dr.V.K.Sood for leading and organizing the capstone course. His guidance throughout the term was extremely helpful in completing project milestones. We would also like to thank Dr. Khalid Hafeez for providing us helpful insight and technical recommendations. Finally, we would like to take this opportunity to express our profound gratitude to our parents for continuously supporting us throughout our journey.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AP	Access Point
DHCP	Dynamic Host Configuration Protocol
EDU	Education
ESC	Electronic Speed Controller
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IDE	Light Emitting Diode
IP	Internet Protocol
IQ	Installation Qualification
MVC	Model-View-Controller
OQ	Operational Qualification
OS	Operating System
PQ	Performance Qualification
RGBW	Red Green Blue White
SDK	Software Development Kit
SoC	Separation of Concerns
SSID	Service Set Identifier
UAV	Unmanned Aerial Vehicles,
UDP	User Datagram Protocol
UI	User Interface
VTX	Video Transmitter

1 Introduction

1.1 Problem Statement

Fireworks have been a popular way to celebrate special occasions across the globe, but their negative impact on the environment, animals, and human health have been increasingly concerning. The use of chemicals such as Barium and Aluminum in the creation of the bright colors of fireworks has been linked to detrimental effects on the environment. The loud noise and air pollution caused by fireworks can also be harmful to animals and humans, and the oxidizers used in fireworks are known to cause water pollution. In light of these issues, the "Low Cost Drone Light Show" project - Firefly, aims to provide a sustainable alternative to fireworks that creates a celebratory atmosphere without causing harm to the environment, animals, or humans.

While drone light shows have been gaining popularity worldwide, they are often expensive and not accessible to the general public. The Firefly project aims to address this issue by developing a low-cost drone light show that is affordable and accessible to a wider audience. The project will use a combination of drone technology and light displays to create stunning visuals that can replace traditional fireworks displays.

The Firefly project seeks to provide a sustainable alternative to traditional fireworks, without compromising on the excitement and celebratory atmosphere that they bring. The low-cost drone light show has the potential to not only reduce the negative impact of fireworks on the environment and wildlife but also to provide a safer alternative for communities that have concerns over the noise and air pollution caused by traditional fireworks. With the aim of making drone light shows more accessible to the general public, the Firefly project has the potential to revolutionize the way we celebrate special occasions and bring communities together.

1.2 Design Process

In order to successfully develop the system, the stakeholders interested in the proposed system must be identified along with their requirements. The primary stakeholders of this system would be individuals that would like to use drones for independent recreational purposes, along with event and entertainment organizations that will be bringing in this type of clientele. Their main priority is to have a safe and low cost experience for their guests, if the system is not able to meet these requirements then these stakeholders will opt for other alternatives. An example of these recreational purposes could include but is not limited to birthday parties, weddings, gender reveals etc. The secondary stakeholders of the system would be the viewers and audiences at the events where the drones are being used. The system should be able to give these viewers an enjoyable and entertaining experience.

In this research report, our capstone project has adopted an agile methodology due to its flexibility for change and evolving nature. This approach enables us to modify project requirements and solutions as needed. The agile software development life cycle consists of five main stages: ideation, development, testing, deployment, and operations. The first stage of agile development is ideation, where the purpose, requirements, and resource allocation are defined. In support of this initial development stage, an architecture schematic was created for the project. Using 3D animation software, we intend to provide the drones with a predefined flight path. This signal will be communicated through the transmitter to the drone, where the receiver will send it to the flight controller. The ESC will appropriately set the

motor speed to control the drone's motion. The signal will also include information for the LED, sensor, and other components required for smooth control.

1.3 Overview of Report

The remainder of this report is organized as follows. Section 2 presents the literature review which entails the background information conducted in order to understand the aim and deployment of the project. Section 3 outlines the technologies used and the details of the integration of the system, including the drones flight and the app development. Ethical and safety considerations are covered in section 4. Finally conclusion and future work are presented in section 5.

2 Literature Review

To understand the background of our project, we conducted research on already existing technologies for drone light shows, research into what core materials will be required for the project and research on how these technologies will be needed to achieve the goal of low costs for drone light shows. Below are these highlighted research areas.

2.1 Drones

The usage of consumer drones has become increasingly popular in recent years, with a variety of applications ranging from search and rescue, firefighting, surveillance, casual videography, and even light shows. These Unmanned Aerial Vehicles (UAV) have incorporated a range of complex mechanics, including rotors, connectivity, accelerometer, altimeter, motors, and landing gear.

In drone light shows, quadcopters with mounted LED lights are the most commonly used. These quadcopters have standard and pusher propellers and are equipped with Electronic Speed Controllers (ESC) for controlling the electric motor's speed and direction. ESC circuits are typically located in the quadcopter's main frame and can be customized using a development kit (SDK). The flight controller, which serves as the brain of the quadcopter, receives information about the battery, GPS coordinates, and steering, and monitors motor speeds in the ESC. The power distribution board regulates and distributes power to the ESC boards and connects to the Video Transmitter (VTX). A receiver connects to the flight controller, receiving signals from a remote control transmitter and providing control to the quadcopter through at least four channels. The use of an antenna enables greater signal distance.

The process of building a drone from scratch necessitates a comprehensive understanding of electrical engineering, particularly given the different components and systems involved. One of the critical parts of a drone is the flight controller, which functions as the drone's brain by regulating its motors and ESCs, enabling movement and flight. The flight controller is an electronics board composed of sensors, processors, transmitter pins, and communication protocols. Through research the flight controller facilitates the drones flight by using a Raspberry Pi, a small computer with numerous capabilities, to code the drone's movements by directing the flight controller on what to do, which would subsequently control the motors and sensors according to the given instructions.

Given the high cost associated with building a drone from scratch, alternative options to complete the project were sought, which focused on programming an affordable drone show. The Tello Drone series, which includes Tello, Tello Iron Man, and Tello EDU, was discovered from Ryze Tech, a technology company focused on making drones more accessible. The drones are equipped with DJI flight

control system, intel processor, and SDK version 10. The Tello EDU utilizes an upgraded SDK version 2.0 that enables more advanced configuration capabilities.

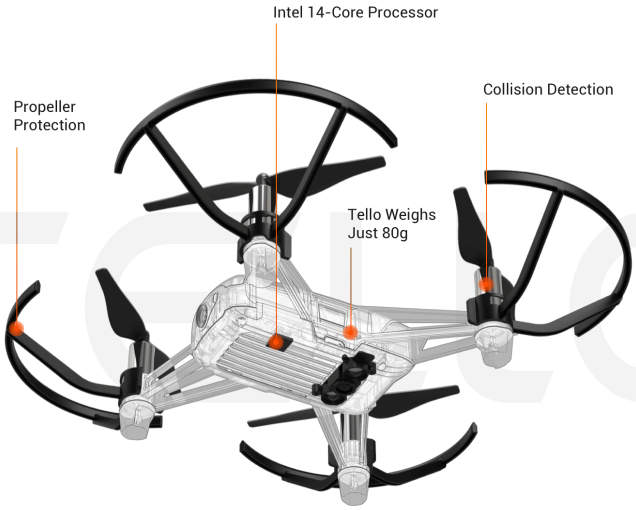

Fig 2.1 Basic Tello Drone Diagram

2.2 Conceptual System Design

The Tello Drone approach provides a significant advantage in programming by offering a baseline of predetermined electrical components. These components include two types of propellers in both clockwise and counterclockwise positions, motors, landing gears, a main control board, electronic speed controllers on all motors, a receiver, an antenna, a battery, and a camera. To enhance the lighting aspect, an additional RGBW LED light was connected to the drone, as the Tello Drone does not have light sensors available. Ryze Tech maintains a Github repository for easy installation of relevant dependencies through a script. The Tello SDK connects to the drone via a Wi-Fi UDP port, enabling scripts and commands to maneuver the drone. The UDP client can be configured on a PC or a mobile device, which we plan to utilize in creating a mobile application to control the light show. All Tello drone models permit basic Tello commands to be automatically configured, including control, set, and read commands. We chose the Tello EDU over the Tello because of the swarm programming mode, which enables multiple drones to be programmed simultaneously. This mode is only available through the Tello 2.0 SDK, which is exclusive to the Tello EDU model. During swarm programming, each Tello drone becomes a client to a router, which must support a 2.4 GHz bandwidth.

3 Implementation and Analysis

3.1 Drone Programming

To meet our budget requirement of under $1000CAD, three Tello EDU drones were purchased for the purpose of this project. The programming of the drones and the interactions between them are essential to achieve the drone light objective. This meant having a method to streamline all interactions through a common network and have swarm programming set up. The Tello EDU SDK 2.0 was specifically selected due to the capability to enable swarm programming with the use of Python.We were

able to customize python scripts that could read a command text file and send specified formations to each individual drone.

3.1.1 Network Initialization

The initialization of a Network was needed in order to streamline connection of the drones. To achieve this, an external wireless router was purchased to eliminate as many discrepancies from the network as possible. The FireFly network was configured with a speed of 2.4GHz bandwidth, IP 198.168.0.1 and subnet of 225.225.225.0. Setting up this external network made connection to Tello more stable as we could control the number of devices on the network. DHCP (Dynamic Host Configuration Protocol) was enabled in order for the router to dynamically assign an IP to each Tello. The starting IP was set to 198.168.0.100 allowing us to know that all connected devices would be assigned an IP between 198.168.0.100-198.168.0.225.

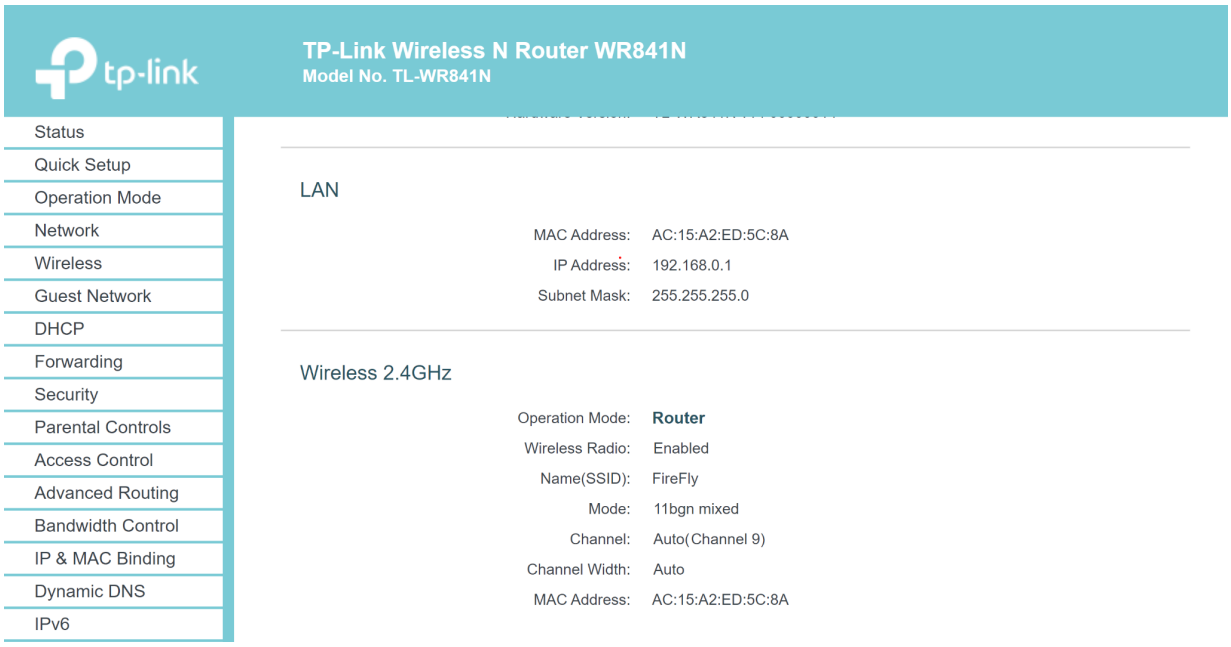

Fig 3.1 FireFly Network Information

Default, Tello EDU drones are set to station mode and have their own individual network. AP allows the drone to turn into a client to a router instead of being the router itself. To switch to AP mode and communicate through the FireFly network, the “configure ap” packet was sent to each drone while on its individual network. We were able to pass packets into the drone by using an application called PacketSender. This packet takes the name and the default password of the network and sends it to port 8889 with IP 192.168.10. When sent correctly, the drone will be rebooted and flash yellow indicating it is open for connections through the access point. As we know the start IP for the network, the Tello will have an IP of 198.168.0.101, 198.168.0.102 and 198.168.0.100 respectively on the FireFly network.

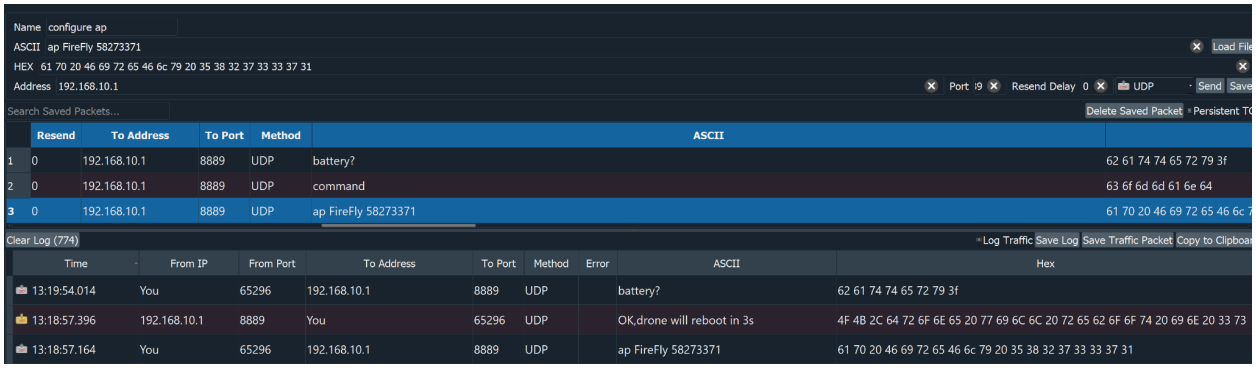

Fig 3.2 Example of configure AP Packet from PacketSender

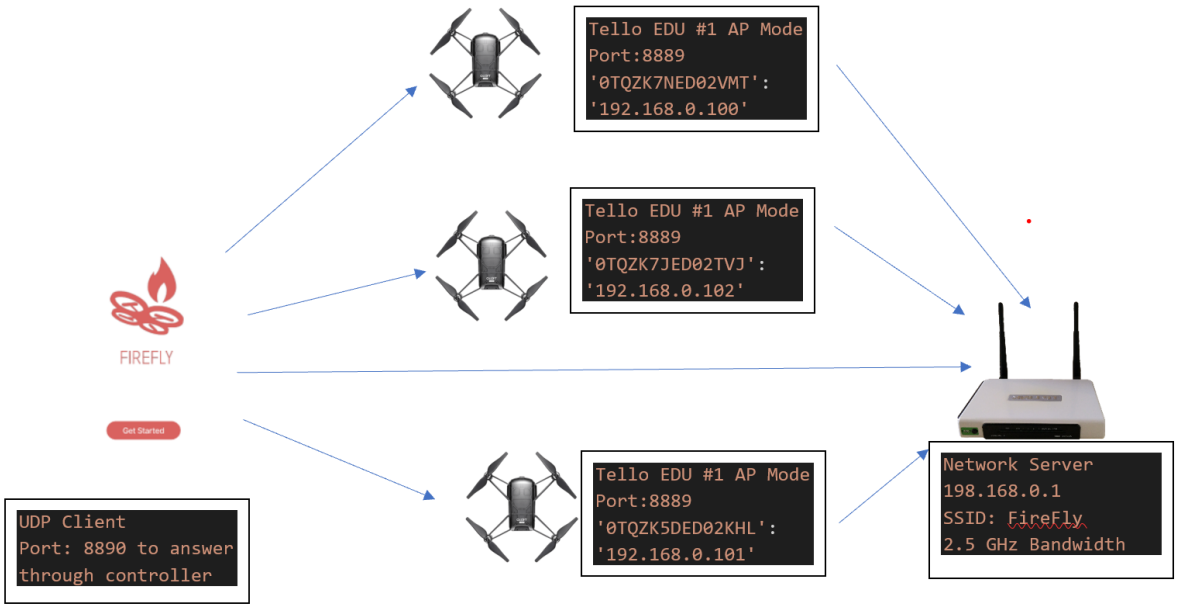

Fig 3.3 UDP connection between App, Drones and FireFly Server

3.1.2 Python Scripts Overview

The programming design consists of 3 main python scripts named planned-flight.py, tello.py and swarm.py. Within Tello.py the main classes include Stats, SubnetInfo, Tello and TelloManager. Swarm.py holds the SwarmUtil and Swarm classes. These work together in order to provide the necessary base functionality to the Tellos. Stats is used to receive various statistics on the drone such as start time, end time, duration and response. This is particularly important for troubleshooting purposes and to fill information for our log. SubnetInfo is used to set the network and get possible IPs in the subnetwork. Tello is a wrapper class which enables the IP and Tello Manager to send commands. The Tello Manager is the script that gets possible IPs and subnet information, in order to return a list of Tello for commands to be sent into. It is called within the swarm and swarm util classes which handles the execution pool and the queue of the Tello Manger for the drones. The Swarm class is used to model the Tello EDU swarm and

initializes the connection points such as the IP pool of the drones and is also where our coordinate system and formations are specified.

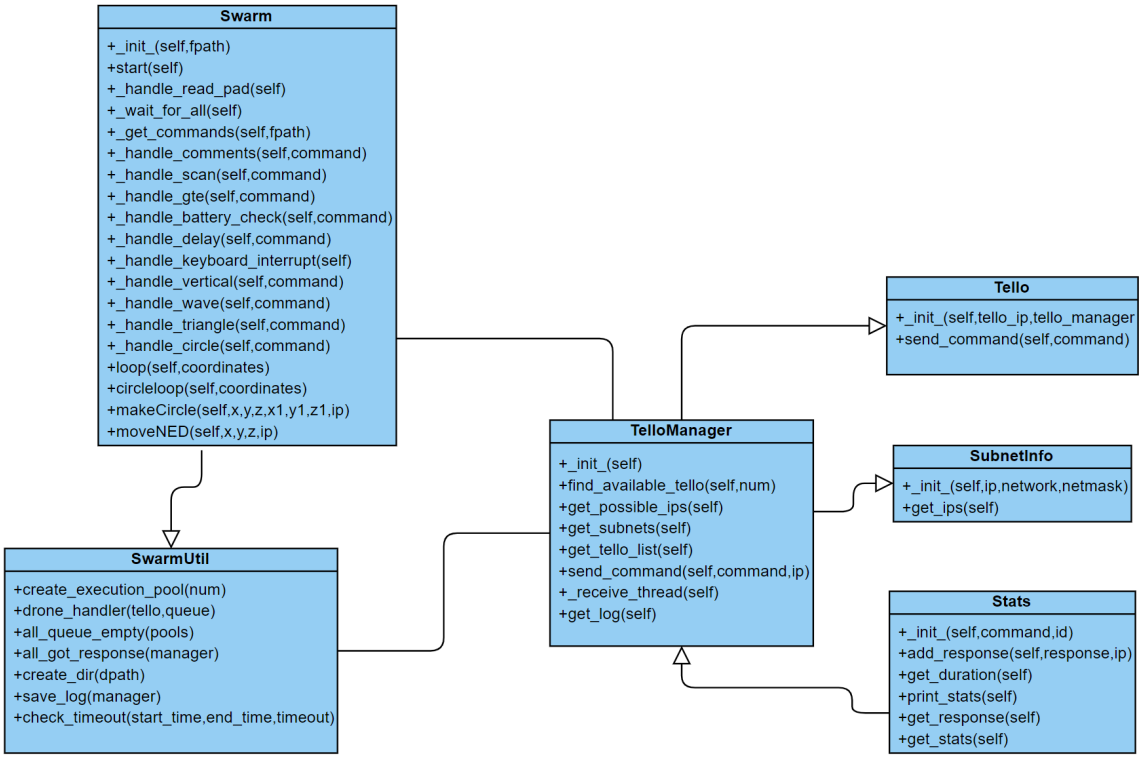

Fig 3.4 Class Diagram of Swarm Programming

Table 3.1: Swarm Class Method Summary

Method	**Parameters**	**Explanation**
init()	self.fpath	Called when an object of that class is created. Initializes the attributes of the object. Takes one parameter 'self.fpath'. Where the SN,IP and ID are declared.
start()	self	Main loop. Starts the swarm.
_handle_read_pad()	self	Sets up command file for reading
_wait_for_all()	self	Waits for all queues to be empty and for all responses to be received.
_get_commands()	self	Returns list of commands
_handle_comments()	self	Handles comments.

_handle_scan()	self, command	Handles scan.
_handle_gte()	self, command	Handles gte or >.
_handle_battery_check()	self, command	Handles battery check. Raises exception if any drone has battery life lower than specified threshold in the command.
_handle_delay()	self, command	Handles delay.
_handle_keyboard_interrupt()	self	Handles keyboard interrupt.
_handle_vertical()	self, command	Handles vertical Formation. Assumes start with horizontal line up.
_handle_wave()	self, command	Handles Wave Formation. Assumes horizontal start.
_handle_triangle()	self, command	Handles Triangle Formation. Assumes horizontal start.
_handle_circle()	self, command	Handles Circle Formation. Assumes horizontal start.
loop()	self, coordinates	Loops through IP to assign x,y,z coordinates.
circleloop()	self, coordinates	Loops through IP to assign start and end x,y,z coordinates.
makeCircle()	self, x, y, z, x1, y1, z1, ip	Sets up command with coordinates and IP to be sent with Tello Manager.
moveNED()	self, x, y, z, ip	Sets up command with coordinates and IP to be sent with Tello Manager.
_handle_correct_ip()	self, command	Handles correction of IPs.
_handle_eq()	self, command	Handles assignments of IDs to serial numbers.
_handle_sync()	self, command	Handles synchronization.

Table 3.2: SwarmUtil Class Method Summary

Method	**Parameters**	**Explanation**
create_execution_pool()	num	Creates execution pools.
drone_handler()	tello.queue	Drone handler.

all_queue_empty()	pools	Checks if all queues are empty.
all_got_response()	manager	Checks if all responses are received.
create_dir()	dpath	Creates a directory if it does not exists.
save_log()	manager	Saves the logs into a file in the ./log directory.
check_timeout()	start_time, end_time, timeout	Checks if the duration between the end and start times is larger than the specified timeout.

Table 3.3: TelloManager Class Method Summary

Method	**Parameters**	**Explanation**
init()	self	Initializes the Tello Manager Class.
find_available_tello()	self, num	Find Tellos that are available on the network.
get_possible_ips()	self	Gets all the possible IP addresses for subnets that the computer is a part of.
get_subnets()	self	Gets all subnet information.
get_tello_list()	self	Returns the list of Tellos detected on the network.
send_command()	self.command, ip	Sends a command to the IP address. Will be blocked until the last command receives an 'OK'. If the command fails (either b/c time out or error), will try to resend the command.
_receive_thread()	self	Listen to responses from the Tello. Runs as a thread, sets self.response to whatever the Tello last returned.
get_log	self	Get all logs.

Table 3.4: Tello Class Method Summary

Method	**Parameters**	**Explanation**
init()	self.tello_ip, tello_manager	Initializes the Tello Class.
send_command()	self.command	Sends a command into a specified Tello IP.

Table 3.5: SubnetInfo Class Method Summary

Method	**Parameters**	**Explanation**
init()	self.ip, network, netmask	Initializes the SubnetInfo Class.
get_ips()	self	Gets all the possible IP addresses in the subnet.

Table 3.6: Stats Class Method Summary

Method	**Parameters**	**Explanation**
init()	self.command, id	Initializes the Stats class
add_response()	self.response, ip	Adds a response.
get_duration()	self	Gets the duration.
print_stats()	self	Prints statistics.
get_response()	self	Checks if response was received.
get_stats()	self	Gets the statistics.

The list of commands are stated within a simple text file with the initialization of the “scan 3” and the three serial numbers for drones. Using the scripts, the program knows to search for 3 drones on the network and searches for a matching serial number to IP declared in the swarm _init_ method. A default set of commands are provided by the SDK 2.0 automatically such as basic up, down, left and right movement. For the purpose of a lightshow, additional commands were created in order to have various default formations. For 3 drones, we implemented a vertical line formation, triangle, wave and circle formation. The commands text file is feeded unto the swarm.py after being called with the initial planned-flight.py script.

Table 3.7: Serial Number, IP and ID for 3 Tellos

ID	**Serial Number (SN)**	**IP Address**
0	0TQZK7NED02VMT	192.168.0.100

1	0TQZK7JED02TVJ	192.168.0.101
2	0TQZK5DED02KHL	192.168.0.102

3.1.3 Coordinate System Initialization

Implementation of swarm programming required the initialization of a coordinate system in order to map out the flight paths for formations.This was a critical part in visualizing the movement of each drone and planning out the trajectory which was based on the x,y,z planes. All of the formations are based on the front facing camera to be positive x axis, right of the drone to be positive y axis and the upward direction to be positive z axis. We set the initial home coordinate for each Tello by assigning self.home_x=0, self.home_y=0 and self.home_z=0. The moveNED method was created to reflect this coordinate system. This is an integral part of our formations as it sends each coordinate to the appropriate IP with the corresponding directional command.

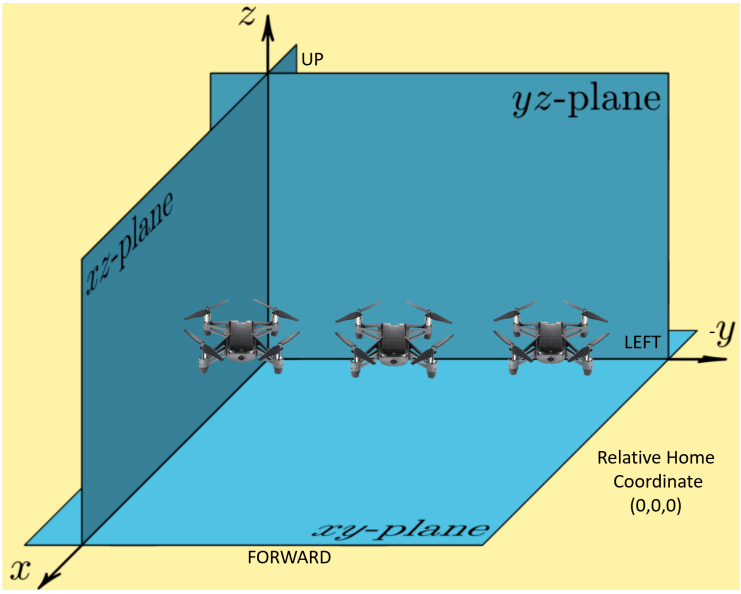

Fig 3.5 X,Y,Z Plane Initialization

Table 3.8 Coordinate and direction set in moveNED method

Coordinate	direction	Command sent to self.manager.send_command
x>0	forward	`"forward "+str(x),ip)`
x<0	back	`"back "+str(x2),ip`
y>0	right	`"right "+str(y),ip`

y<0	left	`"left "+str(y2),ip`
z>0	up	`"up "+str(z),ip`
z<0	down	`"down "+str(z2),ip`

```
def moveNED(self,x,y,z,ip):
    y2=y*-1
    x2=x*-1
    z2=z*-1

    if x>0:
        self.manager.send_command("forward "+str(x),ip)
    elif x<0:
        self.manager.send_command("back "+str(x2),ip)

    if y>0:
        self.manager.send_command("right "+str(y),ip)
    elif y<0:
        self.manager.send_command("left "+str(y2),ip)

    if z>0:
        self.manager.send_command("up "+str(z),ip)
    elif z<0:
        self.manager.send_command("down "+str(z2),ip)
```

Fig 3.6 Code Snippet of moveNED

In order to implement Circle formation on the coordinate system a separate circle method was implemented. The method took in 6 coordinates indicating the starting and ending coordinates for a semi circle. In order to complete the circle, the coordinates were multiplied by -1 to get the bottom half of the circle. This set of 12 was passed into a Tello along with the “curve” twice and IP address to form the circular shape.

Table 3.9 Coordinate Set and Circle set in makeCircle

Coordinate set	**Circle**	**Command sent to self.manager.send_command**
x,y,z,x1,y1,z1	Top Half	`"curve "+str{coordinate set},ip)`
(x,y,z,x1,y1,z1)*-1	Bottom Half	`"curve "+str{coordinate set},ip`

```
def makeCircle(self,x,y,z,x1,y1,z1,ip):
    x2_1=x*-1
    y2_1=y*-1
    z2_1=z*-1
    x2_2=x1*-1
    y2_2=y1*-1
    z2_2=z1*-1

    self.manager.send_command("curve "+ str(x) + " " +str(y) + " " +str(z)
                              +" "+ str(x1)+ " " +str(y1) + " "+str(z1)+" "+str(30),ip)

    self.manager.send_command("curve "+ str(x2_1) + " " +str(y2_1) + " " +str(z2_1)
                              +" "+ str(x2_2)+ " " +str(y2_2) + " "+str(z2_2)+" "+str(30),ip)
```

Fig 3.7 Code Snippet of makeCircle

3.1.4 Drone Light Show Formation

IP Loop

In order to implement the drone formation, looping through established IPs was required in order to send the specified coordinates to each. This was achieved by storing each Tello object in a id list which is parsed through to get the corresponding serial number by searching the self.id2sn list and ip by searching sn2ip. A counter was used to separate each drone, where then the coordinates can be pulled from the specified directory and sent as a parameter along with the IP, to the moveNED and makeCircle function.

```
def loop(self,coordinates):
    id_list=[]
    id_list=[t for t in range(len(self.tellos))]
    num=0

    for tello_id in id_list:
        sn = self.id2sn[tello_id]
        ip = self.sn2ip[sn]
        if num==0:
            print(f'{str(num)}{str(ip)}')
            print(f'{coordinates["x1"]},{coordinates["y1"]},{coordinates["z1"]}')
            self.moveNED(coordinates["x1"],coordinates["y1"],coordinates["z1"],ip)
        elif num==1:
            print(f'{str(num)}{str(ip)}')
            print(f'{coordinates["x2"]},{coordinates["y2"]},{coordinates["z2"]}')
            self.moveNED(coordinates["x2"],coordinates["y2"],coordinates["z2"],ip)
        elif num==2:
            print(f'{str(num)}{str(ip)}')
            print(f'{coordinates["x3"]},{coordinates["y3"]},{coordinates["z3"]}')
            self.moveNED(coordinates["x3"],coordinates["y3"],coordinates["z3"],ip)
        num=num+1
```

Fig 3.8 Code Snippet of Loop Function

```
def circleloop(self,coordinates):
    id_list=[]
    id_list=[t for t in range(len(self.tellos))]
    num=0

    for tello_id in id_list:
        sn = self.id2sn[tello_id]
        ip = self.sn2ip[sn]
        if num==0:
            print(f'{str(num)}{str(ip)}')
            print(f'{coordinates["x1_1"]},{coordinates["y1_1"]},{coordinates["z1_1"]},{coordinates["x1_2"]}
            self.makeCircle(coordinates["x1_1"],coordinates["y1_1"],coordinates["z1_1"],
                            coordinates["x1_2"],coordinates["y1_2"],coordinates["z1_2"],ip)

        elif num==1:
            print(f'{str(num)}{str(ip)}')
            print(f'{coordinates["x2_1"]},{coordinates["y2_1"]},{coordinates["z2_1"]},{coordinates["x2_2"]}
            self.makeCircle(coordinates["x2_1"],coordinates["y2_1"],coordinates["z2_1"],
                            coordinates["x2_2"],coordinates["y2_2"],coordinates["z2_2"],ip)

        elif num==2:
            print(f'{str(num)}{str(ip)}')
            print(f'{coordinates["x3_1"]},{coordinates["y3_1"]},{coordinates["z3_1"]},{coordinates["x3_2"]}
            self.makeCircle(coordinates["x3_1"],coordinates["y3_1"],coordinates["z3_1"],
                            coordinates["x3_2"],coordinates["y3_2"],coordinates["z3_2"],ip)
        num=num+1
```

Fig 3.9 Code Snippet of makeCircle

Triangle

One of the first formations we wanted to create was a triangle. The idea is that the drones would start from a horizontal alignment along the y-axis and assuming facing the front camera of the drones, IP 192.168.0.100 is on the far right,IP 192.168.0.102 in center and 192.168.0.101 on the far left. General overview of sequence:

1. Middle drone IP 192.168.0.102 moving upward to specific coordinates.
2. Drone returns to its original position along y axis.

The function is to be called anytime after takeoff, where the left and right drone stay stationary and the middle drone is set to coordinates (0,0,70) which is up 70 and stored in a directory as key-value pairs which is passed into the loop function. After that, it is sent coordinates (0,0,-70) which is down 70, to assume its regular position. Coordinates are established within the _handle_triangle method located in the swarm class and this method is called when “triangle” is found in the command text file by the start method.

```
coordinates={}

coordinates.update([('x1',self.home_x),('y1',self.home_y),('z1',self.home_z),
                    ('x2',self.home_x),('y2',self.home_y),('z2',self.home_z+70),
                    ('x3',self.home_x),('y3',self.home_y),('z3',self.home_z)])
self.loop(coordinates)
```

Fig 3.10 Code Snippet of Coordinates for Triangle Formation

```
coordinates.clear()

coordinates.update([('x1',self.home_x),('y1',self.home_y),('z1',self.home_z),
                    ('x2',self.home_x),('y2',self.home_y),('z2',self.home_z-70),
                    ('x3',self.home_x),('y3',self.home_y),('z3',self.home_z)])
self.loop(coordinates)
```

Fig 3.11 Code Snippet of Coordinates to Return to Original

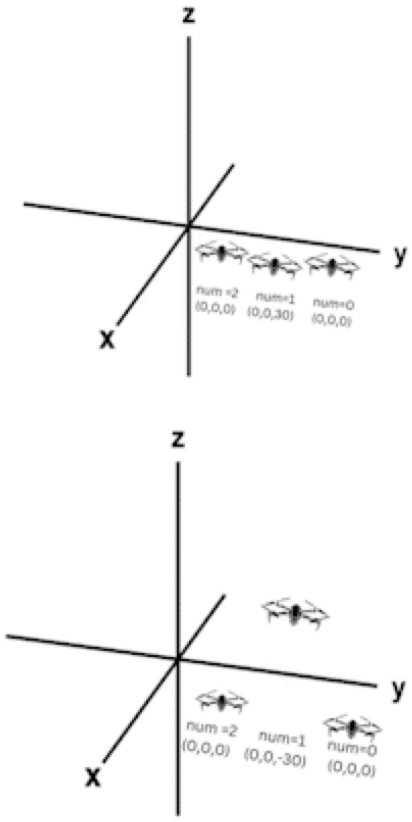

Fig 3.12 Triangle Formation

Wave

The second formation we implemented was a wave sequence between the three drones. Similar to the triangle formation, we assume that drones would start from a horizontal alignment along the y-axis and are aligned with IP 192.168.0.100 on the far right,IP 192.168.0.102 in center and 192.168.0.101 on the far left. A general overview of the sequence includes:

3. Left drone IP 192.168.0.100 and right drone IP 192.168.0.101 moving up to specified coordinates. Middle drone IP 192.168.0.102 moving down to specific coordinates.
4. Left drone IP 192.168.0.100 and right drone IP 192.168.0.101 moving down to specific coordinates. Middle drone IP 192.168.0.102 moving up to specific coordinates.
5. Drones return to their original position along y axis.

The function is to be called anytime after takeoff, where the left and right drone are set to coordinates (0,0,60) which is up 60 and the middle drone is set to coordinates (0,0,-60) which is down 60 to create the first wave formation. Coordinates are stored as key value pairs in the coordinates dictionary that get passed into the loop function. The same methodology was used to set a second wave, by setting coordinates of the left and right drone to (0,0,-60) which is down 60 and middle drone to (0,0,60) which is up 60. Lastly, the drones assume their regular position on the horizontal access by setting the left and right drones to (0,0,30) which is up 30 and middle drone to (0,0,-30) which is down 30. All coordinates are established within the _handle_wave method located in the swarm class and is called when "wave" is found in the command text file by the start class.

```
coordinates={}

coordinates.update([('x1',self.home_x),('y1',self.home_y),('z1',self.home_z+60),
                    ('x2',self.home_x),('y2',self.home_y),('z2',self.home_z-60),
                    ('x3',self.home_x),('y3',self.home_y),('z3',self.home_z+60)])

self.loop(coordinates)
```

Fig 3.13 Code Snippet of First Wave Coordinates

```
coordinates.clear()

coordinates.update([('x1',self.home_x),('y1',self.home_y),('z1',self.home_z-60),
                    ('x2',self.home_x),('y2',self.home_y),('z2',self.home_z+60),
                    ('x3',self.home_x),('y3',self.home_y),('z3',self.home_z-60)])

self.loop(coordinates)
```

Fig 3.14 Code Snippet of Second Wave Coordinates

```
 coordinates.clear()

 coordinates.update([('x1',self.home_x),('y1',self.home_y),('z1',self.home_z+30),
                     ('x2',self.home_x),('y2',self.home_y),('z2',self.home_z-30),
                     ('x3',self.home_x),('y3',self.home_y),('z3',self.home_z+30)])
 self.loop(coordinates)
```

Fig 3.15 Code Snippet of Coordinates to Return to Original

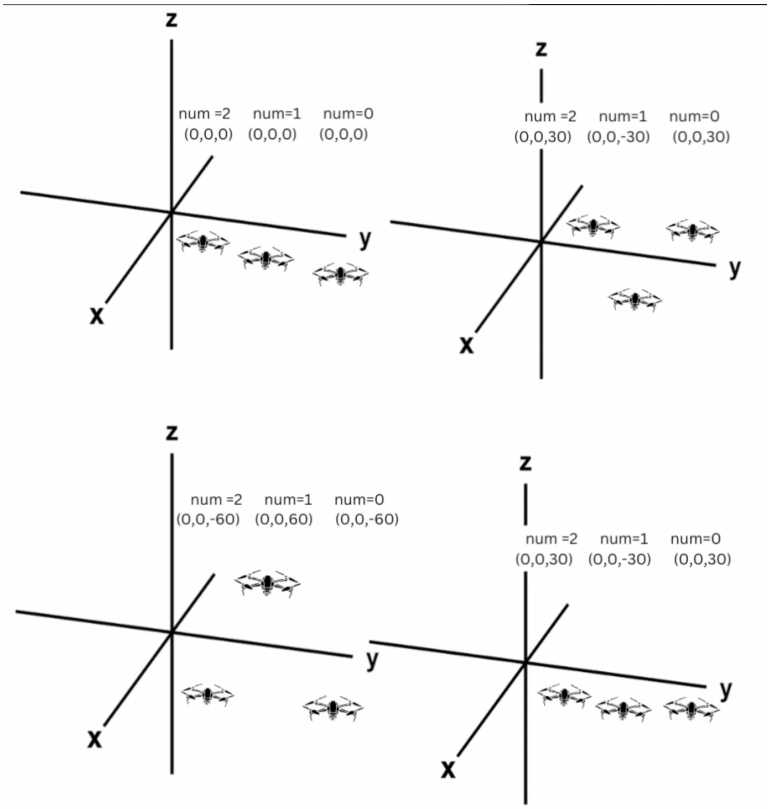

Fig 3.16 Wave formation

Vertical

The third formation we implemented was a vertical sequence between the three drones. Similar to the triangle formation, we assume that drones would start from a horizontal alignment along the y-axis and are aligned with IP 192.168.0.100 on the far right,IP 192.168.0.102 in center and 192.168.0.101 on the far left. The general overview of the sequence:

1. Drones align on the x axis with back to front drone arrangement being IP 192.168.0.100, 192.168.0.102 and 192.168.0.101.
2. Drones for a diagonal along x-z axis as back drone IP 192.168.0.100 flies up and front drone IP 192.168.0.101 flies down to the specified coordinates.
3. Back drone IP 192.168.0.100 flies left and front drone IP 192.168.0.101 flies right to the specified coordinate.
4. Drones return to their original position on the horizontal axis.

The function is to be called anytime after takeoff, where the left drone is set to coordinate (-50,30,0) indicating back 50 and right 30. The middle drone is set to (20,0,0) which is forward 20, and the right drone is set to coordinate (50,-30,0) which is forward 50 and left 30. This aligns the drones along the x axis. The coordinates are stored as key-value pairs in a dictionary and passed to the loop function.

```
coordinates={}

coordinates.update([('x1',self.home_x-50),('y1',self.home_y+30),('z1',self.home_z),
                    ('x2',self.home_x+20),('y2',self.home_y),('z2',self.home_z),
                    ('x3',self.home_x+50),('y3',self.home_y-30),('z3',self.home_z)])

self.loop(coordinates)
```

Fig 3.17 Code Snippet x-axis Alignment Coordinates

A diagonal along the x-z axis is then created by assigning (0,0,30) which is up 30 the back drone and (0,0,-30) which is down 30 to the front drone. The middle drone is set to (-20,0,0) which is backward 20.

```
coordinates.clear()

coordinates.update([('x1',self.home_x),('y1',self.home_y),('z1',self.home_z+30),
                    ('x2',self.home_x-20),('y2',self.home_y),('z2',self.home_z),
                    ('x3',self.home_x),('y3',self.home_y),('z3',self.home_z-30)])

self.loop(coordinates)
```

Fig 3.18 Code Snippet of x-z Diagonal Coordinates

The back drone is then assigned coordinate (0,-30,0) which indicates left 30 and dront drone was assignment coordinate (0,30,0) which indicates right 30 and middle drone (20,0,0) which is forward 20.

```
coordinates.clear()

coordinates.update([('x1',self.home_x),('y1',self.home_y-30),('z1',self.home_z),
                    ('x2',self.home_x+20),('y2',self.home_y),('z2',self.home_z),
                    ('x3',self.home_x),('y3',self.home_y+30),('z3',self.home_z)])

self.loop(coordinates)
```

Fig 3.19 Code Snippet of Left Right Coordinates

Lastly, the drones assume its regular position on the horizontal access by setting the left drone to (50,0,-30) which is forward 50 and down 30. The right drone is set to (-50,0,30) which is backward 50 and right 30. The middle drone is set to (-20,0,0) indicating a backward direction of 20. All coordinates are established within the _handle_wave method located in the swarm class and is called when “vertical” is found in the command text file by the start class.

```
coordinates.clear()

coordinates.update([('x1',self.home_x+50),('y1',self.home_x),('z1',self.home_z-30),
                    ('x2',self.home_x-20),('y2',self.home_y),('z2',self.home_z),
                    ('x3',self.home_x-50),('y3',self.home_y),('z3',self.home_z+30)])

self.loop(coordinates)
```

Fig 3.20 Code Snippet of Coordinates to Return to Original

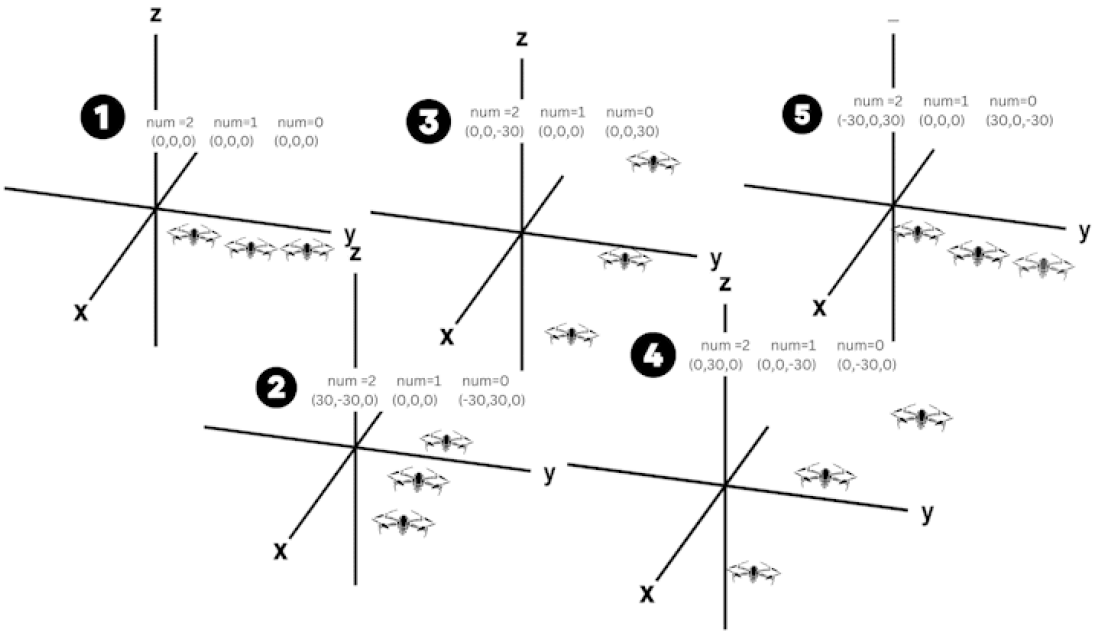

Fig 3.21 Vertical Formation Sequence

Circle

The circular formation takes in 6 coordinates for each drone. The left drone takes in coordinates (0,30,30,0,60,30) which are the starting and ending coordinates for the semi circle flying from starting position along the -y axis. The middle drone front takes in coordinates (30,0,30,30,0,30) and creates a semi circle flying from the starting point forwards along the x axis. The right drone takes coordinates (30,0,30,0,-30,-30) which flies from its starting position along the +y axis. The coordinates are stored as key-value pairs in a dictionary which are called by the circle loop function to assign to the correct IP. The full formation is made by calling two curve functions and multiplying the coordinate by -1 to get the reflected semi circle formation. Due to the nature of a circle, the drones will all end in its original formation along the y axis.

```
coordinates={}

coordinates.update([('x1_1',self.home_x),('y1_1',self.home_y+30),('z1_1',self.home_z+30),
                    ('x1_2',self.home_x),('y1_2',self.home_y+60),('z1_2',self.home_z+30),
                    ('x2_1',self.home_x+30),('y2_1',self.home_y),('z2_1',self.home_z+30),
                    ('x2_2',self.home_x+30),('y2_2',self.home_y),('z2_2',self.home_z+30),
                    ('x3_1',self.home_x),('y3_1',self.home_y-30),('z3_1',self.home_z+30),
                    ('x3_2',self.home_x),('y3_2',self.home_y-60),('z3_2',self.home_z+30)])

self.circleloop(coordinates)
```

Fig 3.22 Code Snippet of Coordinates for Circle

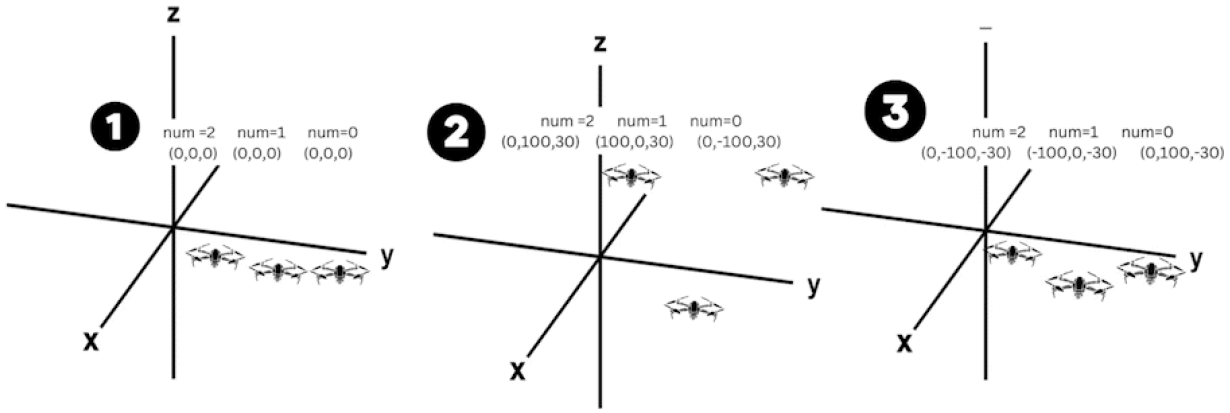

Fig 3.23 Circle Formation Sequence

3.2 Drone Test Cases

Testing protocol for this project has been divided into the following phases including Installation Qualification (IQ), Operational Qualification (OQ) and Performance Qualification (PQ). Due to the multiple parts associated with this project, testing on the drone and mobile application components will be done both separately simultaneously before testing the application as a whole.

3.2.1 Installation Qualification

The purpose of installation Qualification includes ensuring proper setup of equipment. This applies to the drones which is the main equipment in this project. The following test cases were used in order to validate this stage.

Table 3.10: Installation Qualification - Drone Testing

Test Case #	Description	Test Step	Expected Result	Status
TC-1	Wireless network	Connect external devices (3)	Devices shall connect to	PASS

	Configuration for the Router. SSID: FireFly Subnet Mask: 255.255.255.0	onto the network.	the network with no errors.	
TC-2	Test Network Connection for Drone 1.	Connect to Tello Network and send UDP command packet to 192.168.10.1 Port 8889.	Packet shall respond “ok”	PASS
TC-3	Test Network Connection for Drone 2.	Connect to Tello Network and send UDP command packet to 192.168.10.1 Port 8889.	Packet shall respond “ok”	PASS
TC-4	Test Network Connection for Drone 3.	Connect to Tello Network and send UDP command packet to 192.168.10.1 Port 8889.	Packet shall respond “ok”	PASS
TC-5	Test command response for Drone 1.	Connect to Tello Network and send UDP command Battery packet to 192.168.10.1 Port 8889.	Packet shall respond “{}/r/n” where {} is the battery of Tello.	PASS
TC-6	Test command response for Drone 2.	Connect to Tello Network and send UDP command Battery packet to 192.168.10.1 Port 8889.	Packet shall respond “{}/r/n” where {} is the battery of Tello.	PASS
TC-7	Test command response for Drone 3.	Connect to Tello Network and send UDP command Battery packet to 192.168.10.1 Port 8889.	Packet shall respond “{}/r/n” where {} is the battery of Tello.	PASS

3.2.2 Operational Qualification

The purpose of operational qualification is to test that the components function as intended. This stage applies to drones. The following test cases were used to validate this stage.

Table 3.11: Operational Qualification - Drone Testing

Test Case #	Description	Test Step	Expected Result	Status
TC-8	Configure and send command.txt file to drone 1 for Auto Flight Path. command.txt should	Connect to Tello network and send configured text file with app.py script through terminal.	Drone 1 shall execute 7 commands in order with no errors.	PASS

	include the following: command takeoff left 20 right 20 up 10 down 10 land			
TC-9	Configure and send command.txt file to drone 2 for Auto Flight Path. command.txt should include the following: command takeoff left 20 right 20 up 10 down 10 land	Connect to Tello network and send configured text file with app.py script through terminal.	Drone 2 shall execute 7 commands in order with no errors.	PASS
TC-10	Configure and send command.txt file to drone 3 for Auto Flight Path. command.txt should include the following: command takeoff left 20 right 20 up 10 down 10 land	Connect to Tello network and send configured text file with app.py script through terminal.	Drone 3 shall execute 7 commands in order with no errors.	PASS
TC-11	Test shape formation function “poly” which directs the drone to fly in a specified polygon formation. Add the following command to the txt file: command takeoff poly 5 land	Connect to Tello network and send configured txt file with app.py script through terminal.	Selected drone shall execute the 4 commands and the drone shall fly in pentagon formation.	PASS

TC-12	Switch Drone 1 to AP mode. Create command in Packet Sender with network SSID: FireFly and password: TelloEDU12345.	Connect to Tello Network and send UDP command configure AP packet to 192.168.10.1 Port 8889.	Packet shall respond “OK, drone will reboot in 3s”. Drone shall flash yellow.	PASS
TC-13	Switch Drone 2 to AP mode. Create command in Packet Sender with network SSID: FireFly and password: TelloEDU12345.	Connect to Tello Network and send UDP command configure AP packet to 192.168.10.1 Port 8889.	Packet shall respond “OK, drone will reboot in 3s”. Drone shall flash yellow.	PASS
TC-14	Switch Drone 3 to AP mode. Create command in Packet Sender with network SSID: FireFly and password: TelloEDU12345.	Connect to Tello Network and send UDP command configure AP packet to 192.168.10.1 Port 8889.	Packet shall respond “OK, drone will reboot in 3s”. Drone shall flash yellow.	PASS
TC-15	Locate IP of drones by running network-scan.py script.	Connect to FireFly Network and run network-scan.py in the terminal.	Terminal shall display a list of IPs and indicate online IPs for 3 drones.	PASS
TC-16	Swarmed drones should fly with one script sent. Test swarmed drone by running command.txt on FireFly Network where all drones are communicating. Add the following into command.txt: scan 3 battery_check 20 correct_ip 1=0TQZK7NED02VMT 2=0TQZK7JED02TVJ 3=0TQZK5DED02KHL *>takeoff sync 5 *>land	Connect to FireFly Network and run commands txt file with app.py script in the terminal.	All 3 drones shall execute commands in order with no errors.	PASS

TC-17	Test shape formation function “vertical” which directs the drone to fly into a vertical formation.	Connect to the Firefly network and send configured txt file through the terminal to swarmed drones.	Drones shall fly into vertical line formation.	PASS
TC-18	Test shape formation function “circle” which directs the drone to fly into a horizontal formation.	Connect to the Firefly network and send configured txt file through the terminal to swarmed drones. terminal.	Drones shall fly into horizontal line formation.	PASS
TC-19	Test shape formation function “wave” which directs the drone to fly in wave formation.	Connect to the Firefly network and send configured txt file through the terminal to swarmed drones. terminal.	Drones shall fly in wave formation.	PASS
TC-20	Test shape formation function “triangle” which directs the drone to fly into a triangle formation.	Connect to the Firefly network and send configured txt file through the terminal to swarmed drones. terminal.	Drones shall fly into a triangle formation.	PASS

3.3 App Development

To enable the ease of control of the drones and its light sequences by a user, the initiative to integrate an app into the project was taken. The purpose of the app is to allow users to choose and select various shapes and sequences that they desire the drones to perform without having to interact with the heavily coded backend of the drones.

The mobile application is being developed using the MVC Architecture. The MVC architecture was chosen as an architecture not only because it is a go-to for development of user interfaces but it also allows for Separation of Concerns (SoC). This means that the frontend and backend are separated into independent components[1] and as a result, it makes it easier to manage changes on these different ends of the project without them interfering with each other. MVC Architecture entails three aspects of the application: Model, View and Controller where the controller is the mediator between the view and the model and controls how data is displayed [1], the view being the Graphical User Interface (GUI) where data is displayed and the model, which is the backend that contains storage of the data likened as a database typically. Below is a diagram capturing the relationships between these entities.

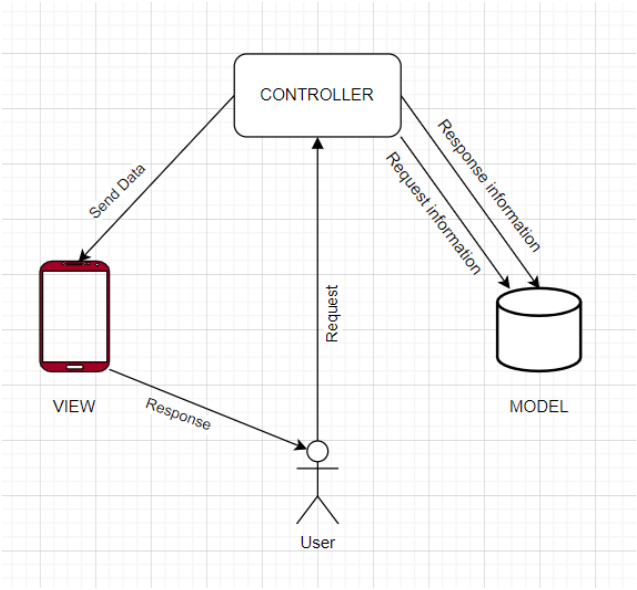

Fig 3.24 App Development: Stack Considerations

Research was conducted to decide on the stack and technologies to develop the application with. There are various pathways, IDEs, and frameworks to accomplish this therefore, pros and cons of each were taken into careful consideration upon coming to a decision. Android Studio, for example, was one that the team was familiar with from previous project experiences. It is an IDE used for Android app development in the languages of either Java or Kotlin with one of its main advantages being faster coding and completion time with instant evaluation of workflow from a feature-rich emulator [2]. However, this is also a limitation to this IDE as it is veered towards Android OS and will not be suitable for other Mobile phone operating systems like iOS.

Flutter was also considered to be used. It is a better option than Android Studio in the sense that it supports the running of mobile applications on both Android and iOS. It is an open-source UI toolkit from Google that is typically used for creating aesthetically-pleasing, natively compiled applications from a single codebase for mobile, desktop and web applications [3]. Although this would have worked perfectly to be used in the project, the disadvantage of Flutter in this context is the language used in app development here. Flutter apps are developed in the Dart language, a language that is completely unfamiliar with members of the team. Therefore, taking the time to learn the language first to then develop an app in it would have proven time consuming and inefficient.

App Development: Final Stack Decision

As a result, the final decision for the Stack to use for the app development was concluded as React Native. React Native is an open-source UI framework that not only supports both Android and iOS mobile phone operating systems, but applications using this framework are written in JavaScript which is a language that did not require to be learned newly by members of the team. The framework itself however, did require some learning but was easily picked up in understanding as it shares similar structure to HTML language web development. Furthermore, in contribution to the app development, Expo was used to build and scale the app.

There is an understanding of the importance of user interfaces and providing users an easy-to-use mode of interaction with the application which is why, before proceeding with the hands-on coding of the app, brainstorming and discussions were taken to formulate an idea of what it would look like. To easily visualize this, Figma, a widely used collaborative platform for interface design, was used. Below is a screenshot of the first few pages of the application. The users will be welcomed onto the app with a starting screen that displays the logo of the project (which was an original design created in regards to the project name—FireFly) with a 'Get Started' button that takes them to the next page. Users will then be able to login to their account or register if they don't already have. The reason for authentication in this project is to enable the feature of personalized and authorized connections to the user's drones.

After a user is logged in and authenticated, the user would be on the landing page which is the Dashboard page. The Dashboard page as shown in Fig 2.5 has the main profile page which can be navigated to and three options. The three options on this page are 'Connect Your Drone', 'Manage Your Drone' and 'Preview Your Lightshow'. Each of these options would help the user to navigate to the page where the user would be able to make changes. The 'Connect Your Drone' option is selected by the user to connect the Tello drone to the application, multiple drones can be connected to the app. The 'Manage Your Drone' option can be selected by the user to remove or add any more drones. Currently, the application only supports adding three drones which would be programmed with the use of swarm programming. The 'Preview Your Lightshow' option is selected by the user to preview any previous settings that the user had selected.

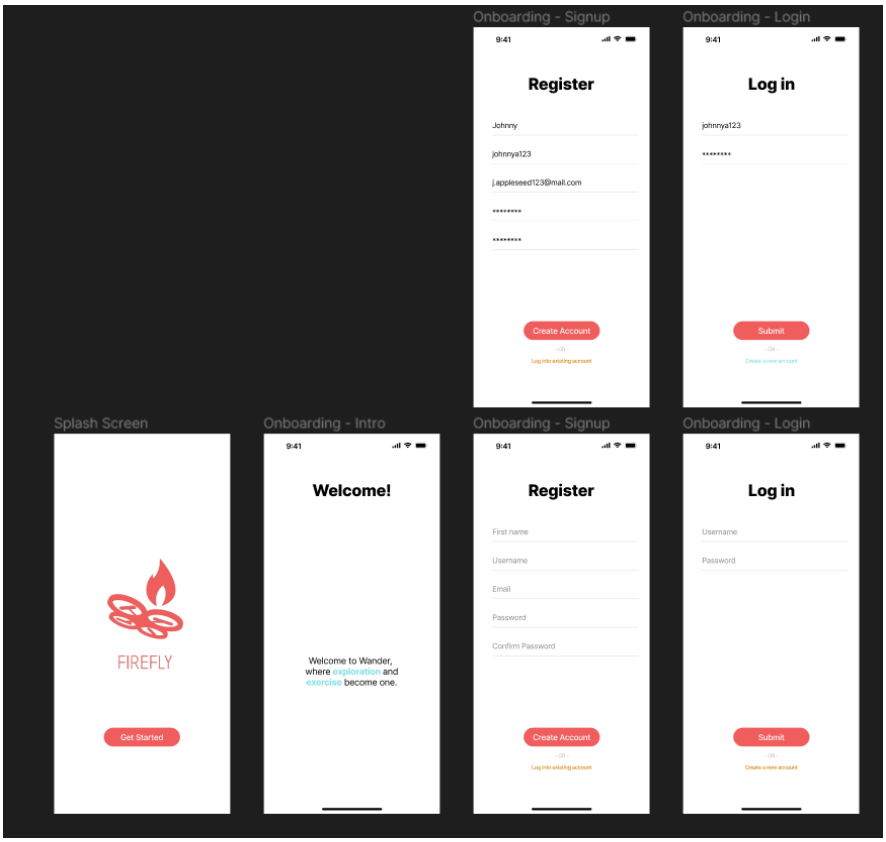

Fig 3.25 Figma Interface Design

Once the interface design using Figma was concluded, coding using the chosen stack—React Native—then began. Starting stages entailed developing and coding the welcome and authentication screens/pages. Below are screenshots of the results. Building and development of the app is an ongoing process as there will be added features.

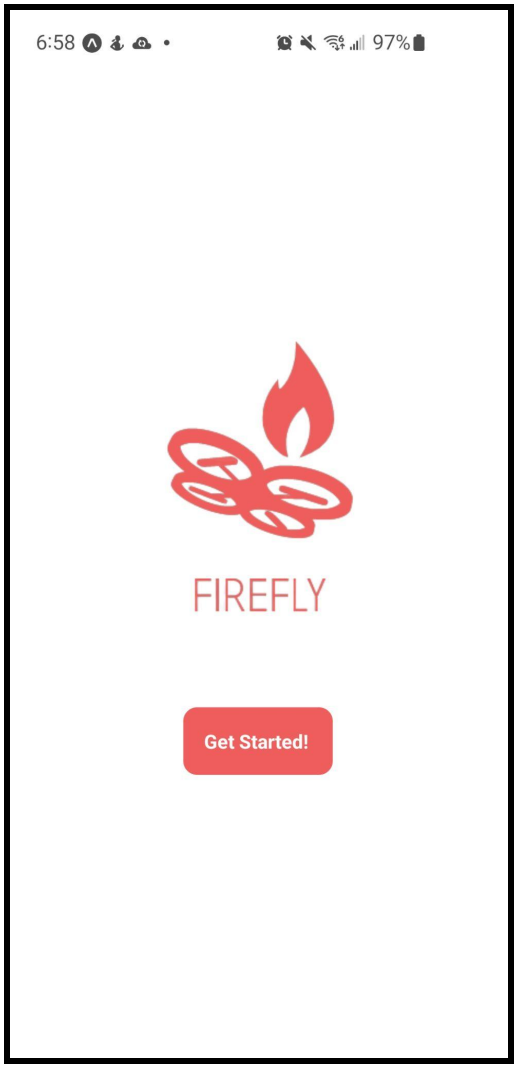

Fig 3.26
Home Screen

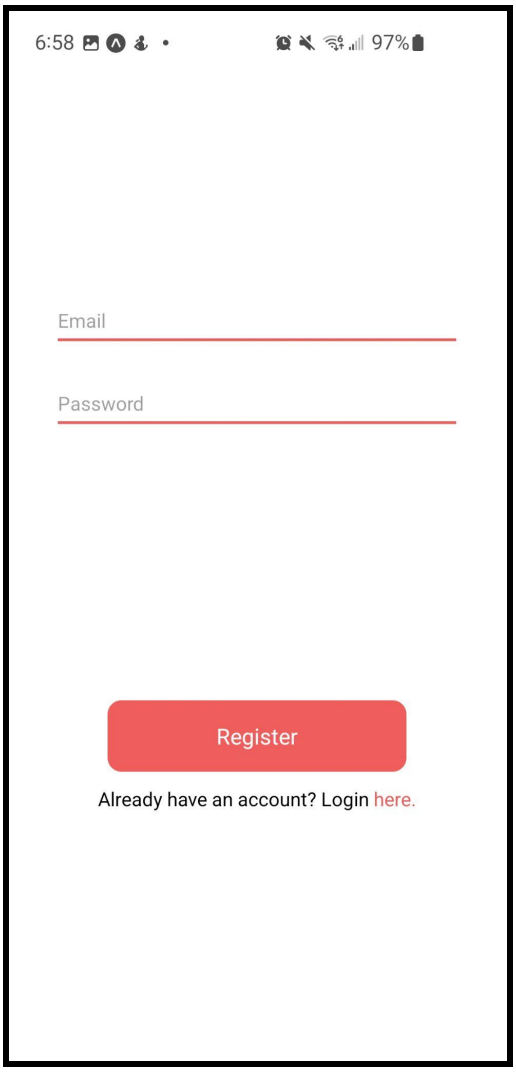

Fig 3.27
Profile Screen

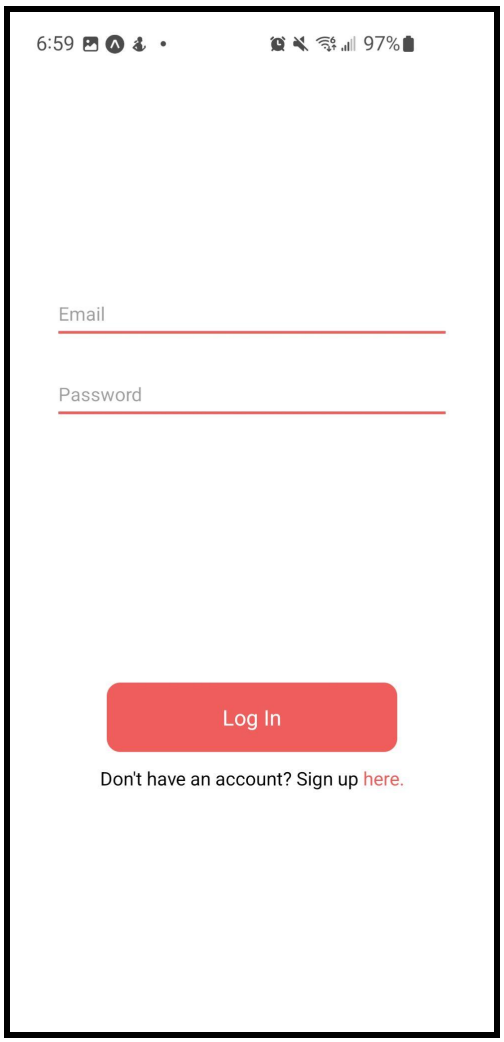

Fig 3.28
Home Screen

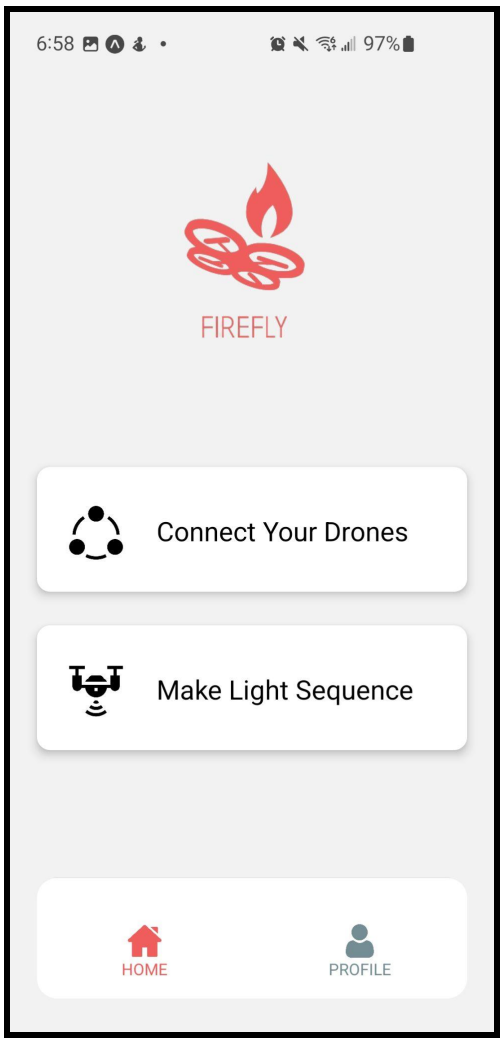

Fig 3.29
Profile Screen

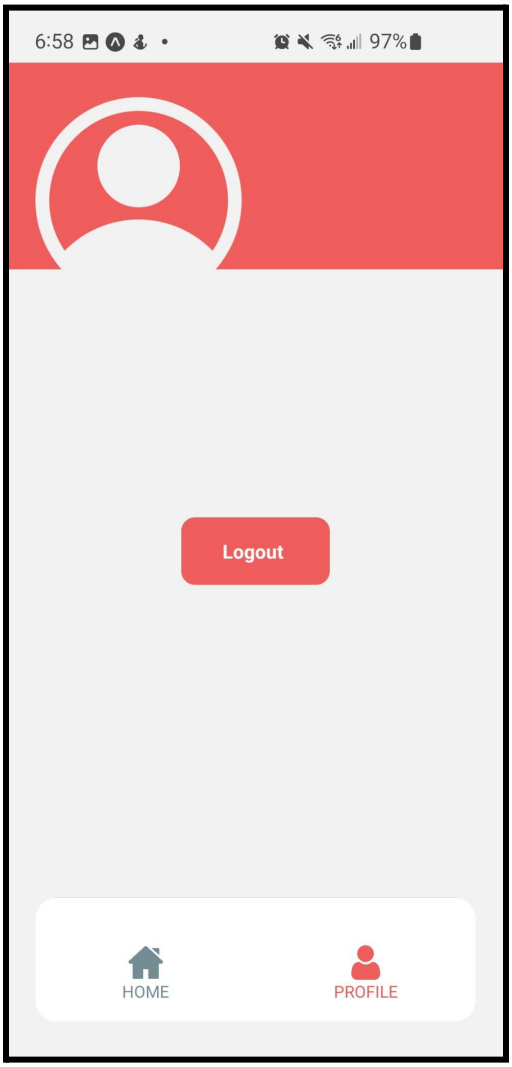

Fig 3.30
Home Screen

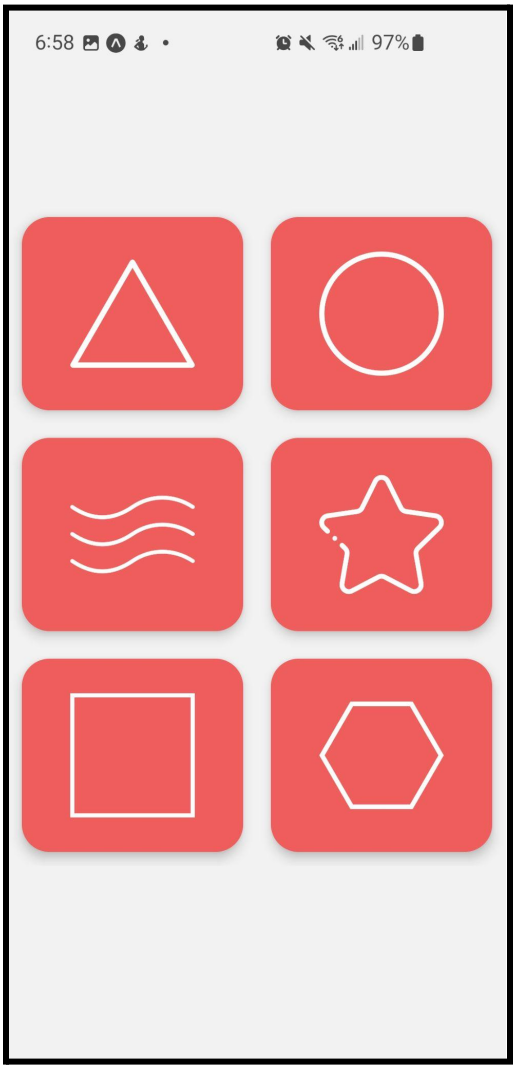

Fig 3.31
Profile Screen

3.4 App Test Cases

Table 3.12: App Testing

Test Case #	Description	Test Step	Expected Result	Status
	Get Started Button. This button should take the user from the splash screen of the Firefly app, to the login screen.	Click login.	User shall be navigated to the login screen.	PASS
	Login Button. This button should confirm an authorized user through authentication from Firebase database and take them to the Dashboard screen if successful.	Login with an authorized account and click login.	User shall be navigated to the dashboard screen.	PASS
	Login Button. This button should not allow unauthorized accounts to login.	Login with an unauthorized account and click login.	User cannot login and the following prompt appears on screen: “Unauthorized user, please try again with a valid username and password.”	PASS
	Signup Link on Login Screen. This link should take the user to the Signup screen.	Click on the Signup link. .	User shall be navigated to the Signup screen.	PASS
	Signup Button. This button should insert the user’s information into the Firebase Database and automatically to the Dashboard screen.	Fill in sign up information and click sign up. Then login with a newly created account.	User information shall be added to Firebase Database. User shall be navigated to the dashboard screen upon login.	PASS
	Home Icon on Navigation Bar. This icon should navigate the user to the Dashboard screen.	Click on Home Icon.	User shall be navigated to the dashboard screen.	PASS

	Profile Icon on Navigation Bar. This icon should navigate the user to their profile screen.	Click on Profile Icon.	User shall be navigated to the user to their profile screen.	PASS
	‘Connect Drones’ Button. This button should take the user to the ‘Connect Drones’ screen.	Click on Connect Drones.	User shall be navigated to the ‘Connect Drones’ screen.	PASS
	‘Make Light Sequence’ Button. This button should take users to the ‘Make Light Sequence’ screen.	Click on Make Light Sequence Button.	User shall be navigated to the ‘Make Light Sequence’ screen.	PASS
	User should be able to create a Light Sequence from options provided for the number of drones indicated. User can see the options on screen for selection for 3 drones: takeoff land vertical horizontal left right up down wave poly flip	Click on options to make a light sequence then click confirm.	A light sequence shall be created. User is directed to the confirmation screen when confirm is clicked.	PASS
	Logout Button. This button should log the user out of the app.	Button click	User shall be logged out of the app.	PASS

Ethical Considerations

With low cost drone light shows comes some ethical issues that must be taken into consideration. This section will highlight and discuss the key ethical considerations that must be considered when planning and executing a low-cost drone light show. The first ethical consideration when planning a low-cost drone light show is privacy. The use of drones for a light show could potentially infringe on people's privacy, particularly if the show is being conducted in a residential area. It is important to obtain the necessary permissions from property owners and to ensure that the drones do not capture any images or video that could infringe on people's privacy. This may involve placing restrictions on the altitude of the drones, or using technology to block or obscure the drone's camera. Noise pollution is another important ethical consideration. Drones can be quite loud, which could be a concern if the light show is being conducted in a residential area. It is important to consider the impact of the noise on nearby residents and to take steps to mitigate any negative effects, such as scheduling the show during non-peak hours or using noise-reducing technology. Finally, respect for wildlife is an important ethical consideration when planning a low-cost drone light show. Drones can disturb wildlife, especially if the show is being conducted in a natural area. It is important to consider the impact of drones on wildlife and to take steps to minimize any disturbance, such as avoiding areas where wildlife is known to congregate or using drones that are designed to be as quiet as possible.

Safety Considerations

This section outlines the safety considerations required for the operation of the drones in this system. These safety measures are regulated and must be followed to ensure safe flight operations. A comprehensive risk assessment was conducted to identify and evaluate potential hazards. This assessment included the prevention of collisions during drone operations which resulted in a system being implemented to ensure that each Tello EDU drone is correctly identified and assigned a unique IP address. This system utilizes a primary technique involving the transmission of packets and the use of pinging each drone to identify and determine its correct IP address prior to the flight operations. This method ensures each individual drone is correctly identified and configured to avoid potential conflicts and collisions during the operation of the drone light show system. Additionally, a comprehensive emergency response plan was developed to respond and manage any possible incidents that may arise during the drone light show operations, such as drone malfunctions or collisions. This plan consists of a series of procedures and protocols to be enacted in the event of an emergency and defines the roles and responsibilities of the trained members involved in the operation and response to manage the situation. By adhering to these safety protocols, the drone light show system mentioned in this report can be operated safely and legally, ensuring public safety and promoting the use of drone technology for recreational and entertainment purposes.

Conclusions and Future Work

There are several areas of future work that could potentially enhance Firefly's system capabilities. Currently, the system incorporates three Tello EDU drones to create a light show, however to expand the complexity of the light show, the system will purchase additional drones to create more intricate and dynamic light shows. In addition, the mobile application that controls the drone's movements and sequences could be further developed. The current version of the application has encountered some challenges connecting to the Python scripts that include pre-built functions used to fly the drones. Future iterations of the application would address this issue to provide a seamless user function. Furthermore, the system currently incorporates attachable LED lights to create the light segment of the drone light show, however there is potential to explore other lighting options that could be integrated with the drones. This could involve exploring the use of custom designed LED drone lights that could be affixed to the drones. Overall, Firefly's system has enormous potential to enhance and expand its capabilities, while maintaining its primary objective of providing a low cost, accessible drone light show. By implementing these proposed areas of improvement, Firefly can continue to innovate and provide an exceptional entertainment experience.

In conclusion, Firefly presents a viable solution to the growing concerns over the negative impacts of traditional fireworks on the environment, animals and human health. By providing a low cost and accessible alternative, Firefly aims to revolutionize the way the public audience celebrates occasions while promoting sustainability and safety. The use of drone technology and light displays has the potential to create stunning visuals that provide an environmentally friendly solution making it an ideal option for the public audience. Overall, with the aim of creating drone light shows that are more accessible to the general public, Firefly is an innovative step towards a sustainable and brighter future.

References

[1] R. D. Hernandez, “The model view controller pattern – MVC architecture and Frameworks explained,” freeCodeCamp.org, 20-Apr-2021. [Online] https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/..

[2] J. Ghanchi, “The major advantages of Android Studio App Development,” IndianAppDevelopers, 17-Nov-2022.[Online] https://www.indianappdevelopers.com/blog/advantages-of-android-studio-app-development/.

[3] “FAQ,” Flutter. [Online] https://docs.flutter.dev/resources/faq#:~:text=Performance%20FAQ-,What%20is%20Flutter%3F,is%20free%20and%20open%20source

[4] A. Tabassi, “5 benefits of using Google Firebase,” InfoTrust, 15-Aug-2022. [Online]. Available: https://infotrust.com/articles/5-benefits-of-using-google-firebase/#:~:text=Initially%20it%20started%20off%20as,%2C%20Web%2C%20and%20Unity%20products

[5] “6. python programming,” 6. Python Programming - Tello Programming 0.0.1 documentation, 25-Oct-2022. [Online]. Available: https://tello.oneoffcoder.com/python.html

Appendix A

A-1: Tello EDU drone SDK 2.0 User Guide

https://dl-cdn.ryzerobotics.com/downloads/Tello/Tello%20SDK%202.0%20User%20Guide.pdf

A-2: Github Repository

https://github.com/Shaeyi-lu/CapstoneApp

Contribution Matrix

	Michelle	Munazza	Rodaba	Nivetha	Toluwanimi
Introduction	✓	✓			
Literature Review			✓	✓	
Implementation & Analysis	✓	✓	✓	✓	✓
Ethical Considerations	✓				✓
Safety Considerations		✓	✓		
Conclusion & Future Work				✓	✓