



## **An Exploration of a connected network of drones using IOT protocols**

Kevin Troy

Student number: A00195358

Date of Submission: 9<sup>th</sup> May

Supervisor: Tom Bennett

Submitted for the requirements for the Bsc(Honours) Degree in Software Design  
with Mobile Apps & Connected Devices

## STUDENT PLAGIARISM DISCLAIMER FORM

### PLAGIARISM DISCLAIMER

STUDENT NAME: Kevin Troy

STUDENT NUMBER: A00195358

PROGRAMME: Mobile Apps and Connected devices 4

YEAR: 4

MODULE: Mobile apps & connected devices Project 4

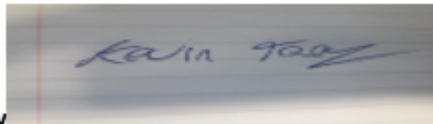
DATE SUBMITTED: 9<sup>th</sup> May

ADDITIONAL INFORMATION: Word count of main body is 9500 words; this excess of the word count was discussed with my supervisor.

I understand that plagiarism is a serious academic offence, and that AIT deals with it according to the AIT Policy on Plagiarism.

I have read and understand the AIT Policy on Plagiarism and I agree to the requirements set out therein in relation to plagiarism and referencing. I confirm that I have referenced and acknowledged properly all sources used in preparation of this assignment. I understand that if I plagiarise, or if I assist others in doing so, that I will be subject to investigation as outlined in the AIT Policy on Plagiarism.

I understand and agree that plagiarism detection software may be used on my assignment. I declare that, except where appropriately referenced, this assignment is entirely my own work based on my personal study/or research. I further declare that I have not engaged the services of another to either assist in, or complete this assignment.



Signed: Kevin Troy

Dated: 09/05/2021

---

## Contents

1.Introduction .....	3
1.1 Context.....	5
1.2 Research Questions.....	5
2 Background Research.....	6
2.1 Current Work – Multiple drone control projects using RC.....	6
2.2 Current Work – Swarmtouch & SwarmGlove .....	7
2.3 Studies on Swarm Robotics .....	9
2.4 How multiple drone systems communicate.....	10
2.5 How Drones operate .....	11
2.6 Gap in subject Area.....	14
2.7 IOT Protocols – MQTT .....	14
2.8 Zigbee.....	17
2.9 Bluetooth .....	18
2.10 PID vs ON/OFF control.....	18
3 System Design.....	20
3.1.1 Requirements - DRONE .....	20
3.1.2 Requirements - APPLICATION .....	23
3.2 Hardware & Architecture.....	25
3.3 Implementation.....	30
3.3.1 Grouping .....	30
3.3.2 Toggling drones and subgroups .....	31
3.3.3 Latency.....	32
3.3.4 Additional JAVA classes .....	34
3.3.5 Drone Commands & nueltral Reset .....	35
3.3.6 Using QoS 1 and 2 .....	36
4 Testing & Results .....	38
4.1 Testing LATENCY with different brokers.....	38
4.2 Testing THE Hc – SrO4 .....	42
4.3 Testing THE VIABILITY OF PID .....	44
4.4 Testing android activities & shared preferences .....	45
5 Conclusions.....	47
5.1 Summary.....	47
5.2 Implications.....	47
5.3 Recommendations.....	48
5.4 Future Work.....	49

## ACKNOWLEDGEMENTS

Throughout my work on this project, I have received support and assistance from several people, both in academia and in terms of support from outside college.

I would like to first thank my Supervisor Tom Bennett for meeting with me when I needed it and always being available to meet. Tom was a massive support throughout guiding me to appropriate research topics and helping me better understand the scope of my project.

I would like to acknowledge my classmates in year 4 Mobile Apps & Connected devices for participating and collaborating on a number of different assignments throughout the year, many of which taught useful skills for this project. I would like to specifically acknowledge Daniel, Kevin and Samantha for their work ethic and companionship.

I would like to thank my housemates for being supportive through this hard year of college as there has been many times they have helped me throughout the year.

I would like to thank my girlfriend, Amy who has been a huge support throughout a tough academic year and without whom I might never have gone back to college.

## **ABSTRACT**

In this paper, I present my strategy for applying IOT safe protocols to the control of multiple Unmanned Aerial Vehicles, in this paper referred to as Drones. This project focuses particularly on the application of MQTT over WiFi in the control of these devices while comparing it to other potential protocols such as Zigbee and Bluetooth and the implementation of such control using an Android Application. The project also examines the use of different strategies for commanding multiple drones and ensuring that each drone understands its position in a formation and the position of the drones around it.

## **1.Introduction**

### **1.1 CONTEXT**

My project is a system that will control a connected swarm of drones in multiple different dynamic formations. The system will simulate the control of these items which will be represented using the ESP 8266's microcontroller functionality. The system will explore IOT (Internet of Things) safe protocols and the application of them in a system of interconnected drones. While applications exist that either control a single drone using a mobile phone through Bluetooth or WiFi, and there is research for methods to control multiple drones with special controllers, my project suggests a novel approach for moderately technically proficient drone enthusiasts to implement a system with multiple drones using their smartphone as the controller. My application is designed so that each drone self identifies, without the need to run multiple different programs on each microcontroller.

### **1.2 RESEARCH QUESTIONS**

- What protocols do drones currently use?
- Can IOT protocols be used to control a network of drones?
- What products currently exist that use MQTT, Zigbee or Bluetooth?
- What protocol will provide the least latency?
- What protocol will provide the greatest scalability?
- What projects exist that control multiple drones? How do they Communicate?
- Can MQTT be used to reliably deliver the same commands to different devices?
- What are the advantages of each type of QOS that MQTT offers and where might a simulation use them?
- What microcontroller would provide the most suitable simulation of a drone?

- How do Zigbee, Bluetooth and MQTT work?
- How do traditional Drone control systems work?

## 2 Background Research

In this chapter I will explore existing work in the field of controlling multiple drones, explore swarm robotics and its application in my project as well as looking at each Internet of Things protocol relevant for the study of this project and compare and contrast each one in terms of latency, range, overhead and their scalability. I will also look at commercial drones in an attempt to understand the requirements in a UAV for the purposes of this project.

### 2.1 CURRENT WORK – MULTIPLE DRONE CONTROL PROJECTS USING RC

Typical drones work by sending and receiving radio signals from remote controllers, typically large custom controllers similar to those used to control hobbyists RC cars. Radio controllers typically feature the following controller layout shown in Figure 1.<sup>[1]</sup>

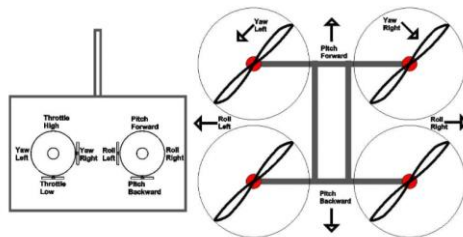


Figure 1

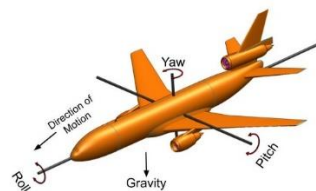


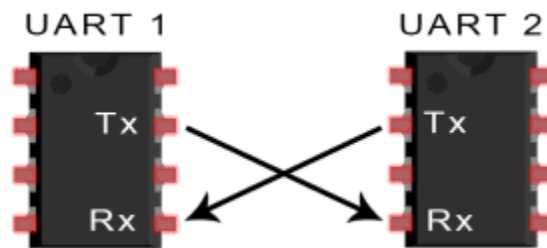
Figure 2

Pushing the left stick vertically will control the throttle and accordingly the speed of the motors of the drone while pushing it horizontally will control the Yaw, while the right stick controls pitch and roll respectively, for reference see Figure 2.

There exists some work to modify existing Radio Controller systems to work with a fleet of drones. Stephen Lazarro of Wisconsin university used an Arduino Uno

to connect to a drones flight controller via UART using modified arduinoCopter firmware to send GPS coordinates to the Arduino, finally using a 2.4GHz transceiver to wirelessly transmit commands and GPS data to neighbouring copters.<sup>[2]</sup> This was done using a common 2.4Ghz RC controller, seen in many RC vehicle control systems.

UART (Universal Asynchronous Receiver and Transmitter) is a circuit in a microcontroller with the purpose of sending and receiving serial data. As UARTS communicate asynchronously, there isn't a clock signal to synchronize the output of bits the sender sends with what the receiver receives so instead they define the beginning and end of a packet in order for the receiver to know when to start receiving bits. It is important to note that each UART must be set up to receive and send the same data structure.<sup>[3]</sup>



*Figure 3*

## **2.2 CURRENT WORK – SWARMTouch & SWARMGLOVE**

There are multiple studies on the use of Vibrotactile gloves to control drones through the movement of the human hand <sup>[4],[5]</sup>. This is a novel approach to interfacing with multiple flying devices by a specialized controller not available for commercial use.



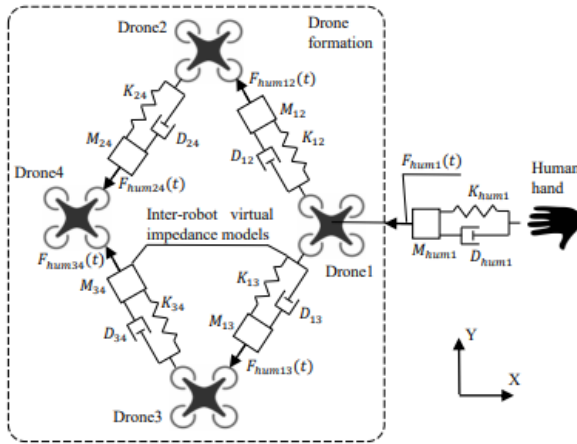


Figure 4

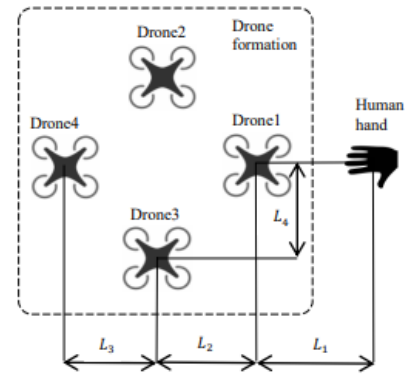


Figure 5

The advantage of this application is that it allows for the user to easily understand how to move and command each agent intuitively. While this cannot be applied to my project, both Swarmitouch and Swarmitouch touch on a fundamental difference between two different models for swarm communication. In Figure 4 we see the use of position-based impedance control while Figure 5 shows PID position control. A key difference between these two models is how they interact with the controller.

However as outlined in the Swarmitouch paper and shown in figure 6, a system where a single drone receives a command and instructs the others to follow it can lead to tangible delays between the first and last drone carrying out their commands.

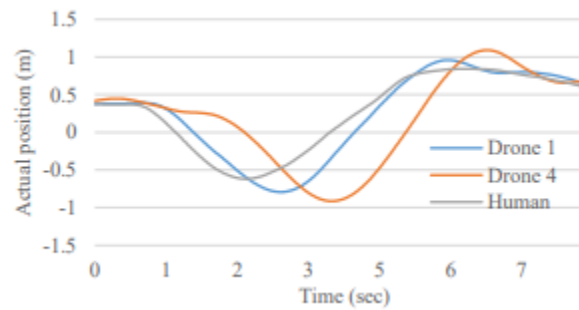


Figure 6

## 2.3 STUDIES ON SWARM ROBOTICS

As defined by E.Sahin of the Middle East Technical University, Swarm Robotics is “a novel approach to the coordination of large numbers of robots ... with realistic interactions among the individuals and the environment”[1]. Swarm Robotics is a field of study worth examining for this project as one can examine the qualities that make for an optimal swarm system and apply them to their work.. The benefits of examining swarm robotics are clear:

- Swarm robotics means that a network of controlled devices can be robust, allowing for the malfunction or disconnection of multiple machines in the network, as such the implemented system should feature dynamic grouping.
- Swarms are naturally capable of greater scalability, allowing for a greater number to be controlled. As such a system that implements this should allow for the number of connected devices to change without disrupting the use of the controller.
- Giulia De Masi and Eliseo Ferrante of TIE Abu Dhabi posit that due to economies of scale, it would also be easier and cheaper to use a swarm of simple machines to accomplish a task than a single complex machine<sup>[6]</sup>

The advantages of swarm control as such are **Adaptability**, **Scalability** and **Robustness**<sup>[6]</sup>.

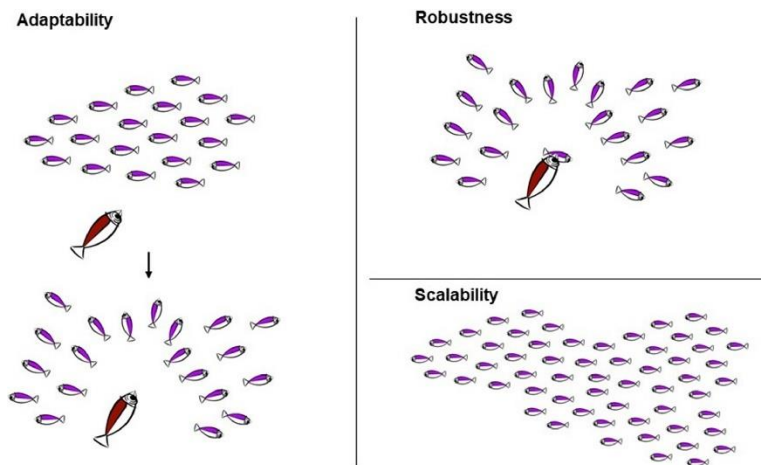


Figure 7

In many applications of swarm robotics that exist today, swarms are either very large groups of machines with very little in the way of computational power and sensors that largely exist in a proof-of-concept space<sup>[7]</sup> or smaller connected networks of more complex machines that are not by their nature swarms. <sup>[8]</sup>

## 2.4 HOW MULTIPLE DRONE SYSTEMS COMMUNICATE

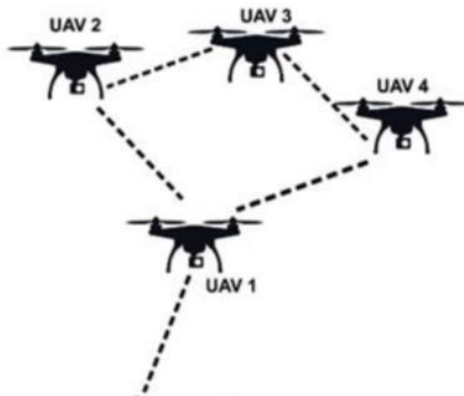


Figure 8

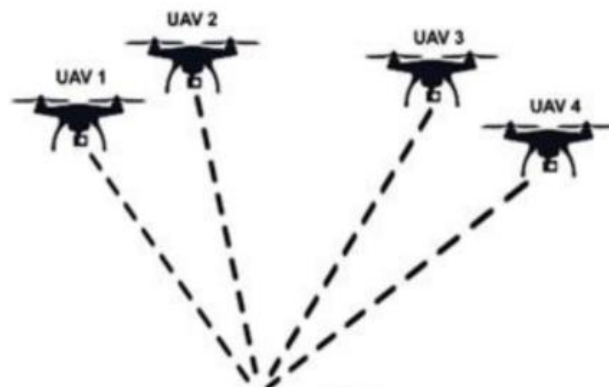


Figure 9

I have Identified a few fundamental structural differences between how multiple drone control systems operate, though all send out commands globally using the controller, most use a single device to receive data and send those commands to

its fellow drones, only the PID position control of swarmtouch<sup>[5]</sup> doesn't rely on a single leader drone to issue commands to the other machines. Relying on a mesh network of drones can slow down time between communication and cause noticeable delay when attempting to issue the same command to multiple drones.

## 2.5 HOW DRONES OPERATE

As part of my research it is critical that I understand how existing commercially available drones operate and the various necessary component parts that make them work, as such in addition to my research I have performed an autopsy on a commercial drone that is no longer functional to understand the necessary design requirements.



*Figure 10*



*Figure 11*

In Figure 10 and 11 we can see a top down and undercarriage view of the standard commercial UAV. Figure 11 shows the presence of LEDs assigned to each motor, these are set to active when the aircraft hovers in place or when the

drone pitches in the opposite direction, indicating that the LEDs signal the direction the drone is intending to move.

In Figure 12, we can see the motor and Electronic Speed Controller, which translates signal to electrical supply. Each drone has four ESCs with one corresponding to each motor. ESCs are used to make sure that the drone can hover in place and remain upright when receiving commands. An ECS works in tandem with the flight controller to determine the amount of power needed for certain maneuvers.<sup>[1]</sup>



*Figure 12*

In Figure 13, we can see the flight controller and receiver. I was able to identify that the receiver is a LT5910 2.4Ghz wireless transceiver(highlighted in blue), this is where the drone receives radio signals from the drones controller, converting them to alternating current pulses, and sending them to the flight controller. While I was unable to identify a specific flight controller, I have learned from my research that a drones flight controller receives and processes signals from the receiver and use on board sensors to communicate those commands to the motors in reasonable ways<sup>[1]</sup>.

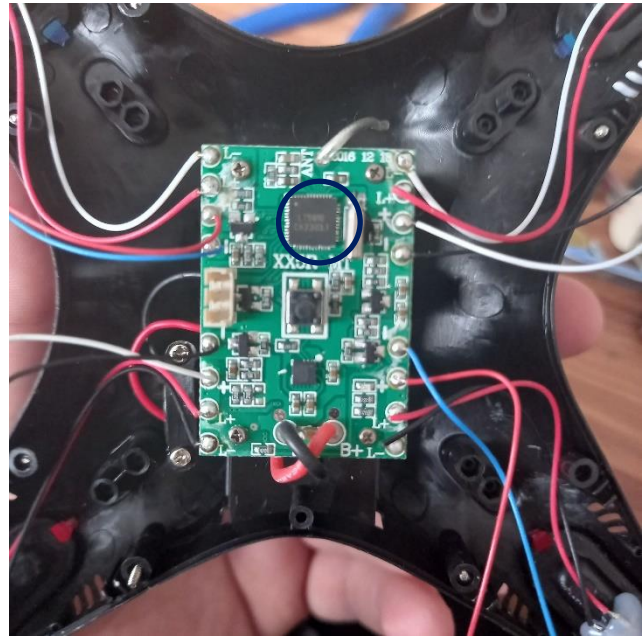


Figure 13

Figure 14 shows the power supply for the drone, a 3.7v battery that provides enough power for 20 minutes of continuous flight and takes about 1 hour to charge. Though there are multiple known ways of powering drones, this project will be focusing on battery powered. The ESP 8266 wifi module has a power output of 3.3v and as such will be examined as a potential emulation for a drone.



Figure 14



## 2.6 GAP IN SUBJECT AREA

As outlined in the section 2.1, Existing strategies for controlling multiple drones are currently being explored through academic space, however most commercially available drone products use either specialized controllers or specialized microdrones. The main gap exists in that while there are apps that control single drones and custom RC controllers for multiple drones, there exists at time of writing no smartphone application that can control multiple drones. Hobbyists, Surveyors, Photographers and Security personnel could each benefit from a system that provides an aerial view of the same location from multiple angles. A further advantage of my proposed idea is that it will use largely cheap materials and test for control of these drones using IOT safe protocols, meaning that it can be reproduced by small business without the need for potentially expensive or difficult to use technology.

Furthermore, much of the research in the field of multiple UAV control follows predetermined flight paths <sup>[4],[5]</sup> and while this is useful for the purpose of delivery drones, the transport of freight in general or some specialized security personnel, the purpose of this project is to provide an application that can control multiple drones at once, with little training.

## 2.7 IOT PROTOCOLS – MQTT

MQ Telemetry Transport (herein referred to as MQTT) is a commonly used protocol that was developed by IBM as part of a suite of products called MQ<sup>[9]</sup>. MQTT is a publish/subscribe protocol designed to be simple and lightweight. MQTT allows devices to subscribe to messages on a set topic and publish using similar topics. In this way it is a one-to-many messaging service that can run over WiFi or Bluetooth. As such, it is common for IOT systems to use MQTT due to its flexible and lightweight nature. MQTT is a machine-to-machine protocol where neither machine communicates directly but rather through the use of something

called an MQTT broker. An MQTT broker is a server that intercepts each message and distributes it based on the subscription status of devices. The broker acts as a courier in this case, with all messages needing to go through it before reaching their destination.<sup>[10]</sup>

MQTT utilizes 3 different types of QoS:

- **0 - At most once:** This is the most common type of message, unassured service. At this QoS level, the publisher sends the message once and never again, with no handshake or confirmation that the message was received.



Figure 15

- **1 – At least once:** This is where the message is guaranteed to arrive at the receiver. In this case the sender stores the message until an acknowledgement is received, resending it when a timeout is reached, this can lead to multiple messages being sent on the same topic with the same message.



Figure 16

- **2 – Exactly once:** This is the highest level of QoS offered by MQTT and as such can be quite slow, however, it offers a guarantee that a message will be delivered once and exactly once. This is done using a 4-part-



handshake that is done by the publisher waiting to receive an acknowledgement, sending duplicate flags until it does. When the publisher gets the acknowledgement, they discard the publish packet and send the PUBREL packet, telling the receiver to discard all stored states and reply with a PUBCOMP packet. The receiver finally stores a reference to the packet ID. This way the message is processed once and only once. [10]

In addition to this, in a situation where a packet is dropped during transit, the message is resent, but due to the nature of the 4 way handshake, it will only be received once.

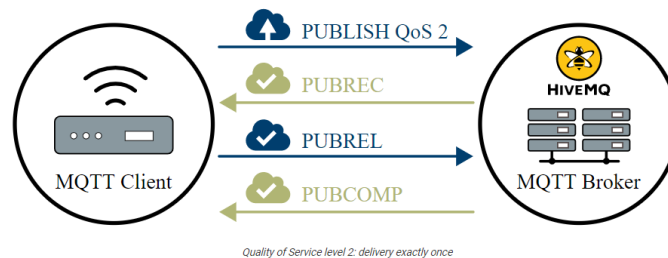


Figure 17

MQTT uses two ports as standard, 1883 and 1888 for unencrypted and encrypted communication respectively, with 1883 using SSL (Secure Sockets Layer) and TLS(Transport Layer Security to validate the server. This is not specifically part of MQTT, however it is customary for brokers to support this. Due to MQTTs nature as a lightweight protocol, these additional features may provide too much overhead to be useful, using SSL and a high QoS makes the benefits of MQTT as a lightweight protocol seem redundant.

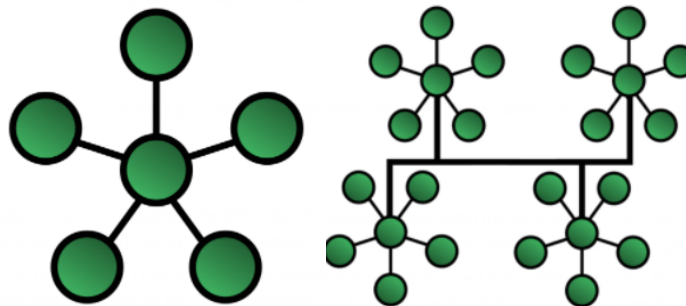
In addition to this, communication between between QoS 1 and 0 and Qos 2 and 0 can have significant impact on latency, this will be explored further in section 3.3.6. With an Operational range of up to 50 meters when run over WiFi, MQTT over WiFi will be the focus of this project.

## 2.8 ZIGBEE

Widely considered as an alternative to Bluetooth and WiFi in the realm of IOT, Zigbee is a low-power wireless mesh network that connects battery-powered devices for the use of wireless control. Most commercial devices that use Zigbee use 2.4Ghz for home use. On first glance this seems very appropriate for creating a network of drones. As of 2006, the Zigbee alliance changed how items in a network communicate, and since then it has used a cluster library that categorizes items in a system with identifier names such as “Smart Lighting” or in our case “Drone group 1”.<sup>[25]</sup>

Zigbee communicates via the same band as WiFi and Bluetooth, IEEE 802.15.4 over 2.4Ghz with a far shorter range, with a maximum range of around 20 meters and significant packet loss after 15<sup>[11]</sup>

To compensate for Zigbees lack of operational range, it supports a range of mesh networks, where multiple devices operate as receivers and transmitters in order to effectively route data<sup>[12]</sup>. In addition to supporting generic mesh networks, Zigbee also supports star networks and tree networks.



A **star network** is an implementation of a spoke–hub distribution paradigm in computer networks. In a star network, every host is connected to a central hub. In its simplest form, one central hub acts as a conduit to transmit messages. The star network is one of the most common computer network

Figure 18

A **tree network**, or **star-bus network**, is a hybrid network topology in which star networks are interconnected via bus networks. Tree networks are hierarchical, and each node can have an arbitrary number of child nodes.

Figure 19

If a zigbee network is sending a payload between less than 5 hops, and the payload is smaller than 150 bytes, the latency should not be expected to be more than 200 milliseconds, with less than 140 milliseconds if the payload is 50 bytes or less, according to a study on Zigbee Mesh Network performance by Silicon labs<sup>[11]</sup>.

## 2.9 BLUETOOTH

Much like Zigbee, Bluetooth was introduced as a low power answer to Personal Area Networks. Notoriously resistant to noise and interference and equipped with a frequency hopping algorithm that prevents interference when multiple WiFi channels are active. Bluetooth is a common choice for app systems that control drones and Bluetooth drones are widely commercially available. Similar to Zigbee however, it can see significant packet loss at distances of 10 meters or greater.

Bluetooth is used for low-cost, low power networks and creating local wireless networks on an ad hoc basis<sup>[12]</sup>. Bluetooth audio devices have an ideal latency of 34 milliseconds but have been observed at latencies over 100 milliseconds, becoming disconnected after frequent messages with a delay of 300 milliseconds or higher

As of android Pie, 5 bluetooth audio devices can be connected at once, to allow for seamless swapping between devices, such as streaming music in one room speaker and switching to another. This does not however mean that meaningfully sized packets can be sent to different devices at the same time <sup>[13]</sup>. As such it may not prove useful for this project.

## 2.10 PID VS ON/OFF CONTROL

The final topic to research is the question of how drones handle moving out of bounds or into forbidden spaces, such as too close to another drone or obstacle. PID (Proportional Integral Derivative) is a common algorithm used to control smart devices in this way<sup>[14]</sup>. The formula for PID is as follows:

$$u(t) = K_p e(t) + K_i \int e(t) dt + K_p \frac{de}{dt}$$

Figure 20

PID uses the Proportional term (Blue), the integral Term (Yellow) and the differential term (Red) to calculate the control variable (Green). In our case, PID would use the speed, distance to a target and our position to modulate our speed accordingly.

PID is a closed loop control method, meaning the system gives feedback based on whether a variable is in its set or reference value<sup>[15]</sup>. A Drone using PID control would slow down when approaching a boundary or obstacle.

On/Off control is often chosen for its simplicity, with it functioning as a very simple way to control a device. In On/Off control the device has a binary state, On or Off. This is often used for commercial drones who fly too high, or common in smart heating systems, when the threshold is reached, the device is shut off until the variable is below the threshold again, in this way the drone will fall until it is at an acceptable height and the heater will turn off until the room needs more heat. On/Off control is simple but easy to implement and PID can require the sort of overhead and calculation that may not be available during this project.

### 3 System Design

In this section I will discuss the design of the system, the system architecture and design of the system, along with a full list of functional and non functional requirements for the system. I will discuss the different prospective components of the simulation and come to a more definitive outline on what the project will look like.

#### 3.1.1 REQUIREMENTS - DRONE

TITLE	Description
FUNCTIONAL REQUIREMENTS	
Drone Commands – UP	Each drone should respond to an UP command by simulating moving forward
Drone Commands - DOWN	Each drone should respond to a DOWN command by simulating moving backward
Drone Commands - LEFT	Each drone should respond to a LEFT command by simulating banking left
Drone Commands - RIGHT	Each drone should respond to a RIGHT command by simulating banking RIGHT
Drone Information - TOGGLE	Each drone should know when the application says it is toggled off and they should be consistent.
Drone Information - FORMATION	Each drone should know its position in a formation (X).

<b>Drone information – FORMATION TYPE</b>	Each drone should know the total number of drones in a formation and therefore its formation type.
<b>Drone Latency - MQTT</b>	Each drone will subscribe to the latency topic and reply to all latency requests with a timestamp in milliseconds
<b>Drone Latency -RECORD TIME</b>	Each drone shall note the time it received a latency request
<b>NON-FUNCTIONAL REQUIREMENTS</b>	
<b>Drone display - LEDS</b>	Each drone should display the status of it's motors through the use of LEDS
<b>Drone information – Other toggled</b>	Each drone should know when another in its formation has been toggled off
<b>Drone formation – RESPONSIVE TOGGLE</b>	Each drone should respond to a toggle by changing group size and their number in that group
<b>Drone Latency - RTC</b>	The drone shall use a real time clock module to keep track of time
<b>Drone Latency – Bluetooth Connectivity</b>	The drone shall feature an LED that can be toggled by the use of Bluetooth using a HC105 bluetooth module

<b>Test Drone Ultrasonic - PID</b>	Test drones shall use HC-SR04 ultrasonic sensors and multiple LEDs to illustrate PID control.
<b>Test Drone Ultrasonic – On/Off</b>	Test drones shall use HC-SR04 and a single LED to illustrate On/Off control.

### 3.1.2 REQUIREMENTS - APPLICATION

TITLE	Description
<b>FUNCTIONAL REQUIREMENTS</b>	
<b>App functionality – Drone Commands</b>	The app should allow the user to send commands to all connected drones
<b>App functionality – Drone Toggle</b>	The app should allow the user to group selected drones using the toggle feature
<b>App functionality – Auto determine mode</b>	The app should be able to determine how many drones are connected to it at once before selecting a flight mode
<b>App functionality – Mode override</b>	The user should be able to override the automatically determined mode using a dropdown menu
<b>App functionality - Menu</b>	The app should feature a menu navigation system that allows the user to navigate between activities
<b>App display - Formation</b>	The app should display the selected formation and each drones position in it.
<b>App functionality - Latency</b>	The app should be able to calculate the time for a reply to be received.
<b>App Functionality – Latency graph</b>	The app should graph the changes in Latency data over time



<b>App Functionality – Average Latency</b>	The app should calculate the rolling average Round-Trip-Time and update it when a new reply is received.
<b>NON-FUNCTIONAL REQUIREMENTS</b>	
<b>App Display – Drone status</b>	The app should allow the user to see the current movement status of each drone they are controlling.
<b>App Display – Drone toggled</b>	The app should notify the user which drones are currently toggled on and off.
<b>App Connection – Connection Complete</b>	The user should be notified when the device has successfully connected to the MQTT broker.
<b>App Connection – New drone</b>	The app should notify the user when a new drone is connected, displaying the new total.
<b>App notifications – New Drone</b>	A push notification should be sent to the user when a new drone is connected.
<b>App Functionality – Joystick Control</b>	The app shall use a responsive joystick to interpret the users commands.
<b>App Functionality – Joystick Power</b>	The app shall only respond to joystick commands if the Joysticks power reading is over 50%.

### 3.2 HARDWARE & ARCHITECTURE

In order to create an application that simulates the control of multiple drones, I needed to identify the necessary parts to make an effective simulation of a drone, to that end this next segment will examine potential parts that would be considered for the design of the drone simulations

**ESP 8266 NodeMCU** is a cheap and powerful chip with WiFi capabilities invented by a Chinese company called Espressif Systems<sup>[16]</sup>. The Esp8266 allows microcontrollers to connect to WiFi and also has microcontroller capabilities itself. The Esp8266 is used commonly for DIY IOT projects due to its low price and lack of external components. The ESP 8266 was updated in 2014, removing the need for external microcontrollers and allowing the ESP to be programmed directly. In addition to SDKs provided by Espressif, Arduino can be used to program the ESP using external libraries. The cheap, replaceable nature of this chip make it the main focus for my project as it falls in line with the values my project is based on and the target market my solution is aimed at.

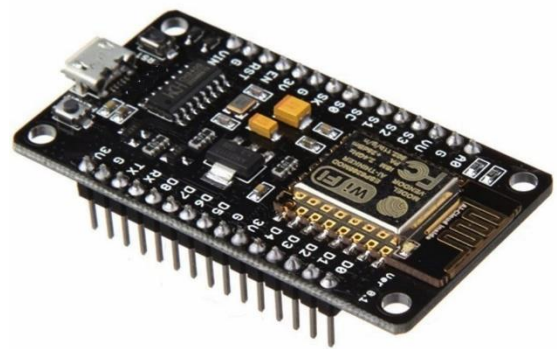


Figure 21

**The HC – SR04 Ultrasonic Sensor** is a commonly used tool to detect distance in Arduino projects as it comes with part of most Arduino Kits. The HC – SR04 is capable of measuring the distance between the sensor and an obstacle within a range of 2 centimeters up to a maximum of 450 centimeters away<sup>[17]</sup> The

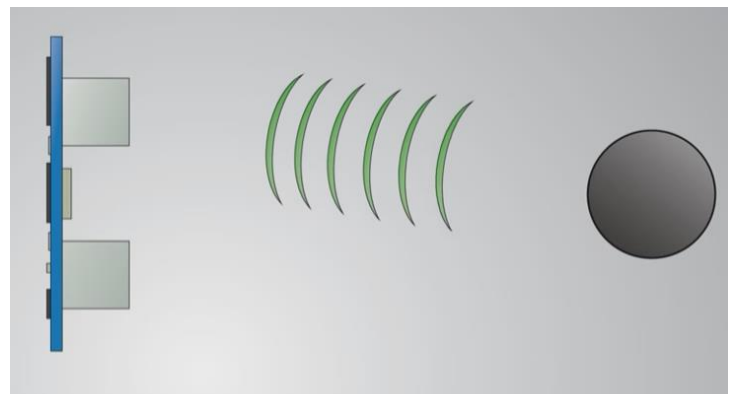


Figure 22

module uses SONAR (Sound Navigation and Ranging) in order to determine distance by sending out a burst of sound and seeing how long it takes to come back. While its range is limited, it is worth exploring the feasibility of this module to help the drones to avoid collision with obstacles and other members of their swarm. In addition to this, the HC – SRO4 will provide adequate testing ground to explore an implementation of a PID like system.

**MQTT** over WiFi seems the most likely candidate for a connection protocol to control multiple drones based on my research of IOT protocols, however this project features a test to check the validity of a bluetooth connected system, it is not the focus of this project. MQTT over WiFi will allow the use of ESP8266 wifi modules as microcontrollers, potentially keeping the cost down even further than using an Arduino Uno.

Protocol	Maximum Range	Latency
Bluetooth	10 meters	34 – 300 milliseconds
MQTT over WiFi	50 meters	50 Milliseconds
Zigbee	20 meters	< 140 Milliseconds

**MAVLINK** is a lightweight protocol that provides a messaging service to communicate with drones and their components. Similar to MQTT, it follows a publish subscribe model where its commands are defined in XML files send out as topics. Licensed and supported by MIT for the control of drones, it would seem the ideal protocol to use, however Mavlink 2 does not support java and as such is outside the scope of an app development project using android studio<sup>[18]</sup>.

**Motor Simulation-** due to the low power nature of the ESP8266 WiFi Module and restrictions on what materials can be acquired, it would not be possible to use real DC motors to represent the drones, accordingly a simple simulation of the drones will simply take inspiration from the model used in the autopsy and represent its external motors with LEDs. Each LED will represent a corner of an aerial view of a quadcopter and therefore represent a rotor. As a command is received to go left, the right 'motors' will be written to low power in order to simulate pitching the drone in that direction.

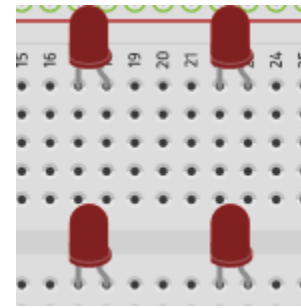


Figure 23

**User Control** had to be done by either buttons, joystick or gyroscope input. There are several different joystick implementations developed by android developers, most of which take in the user input in a direction and receive that as an analogue command. In the process of designing this project I downloaded and sampled 3 different joystick projects, one by instructables<sup>[19]</sup> shown in figure 24,

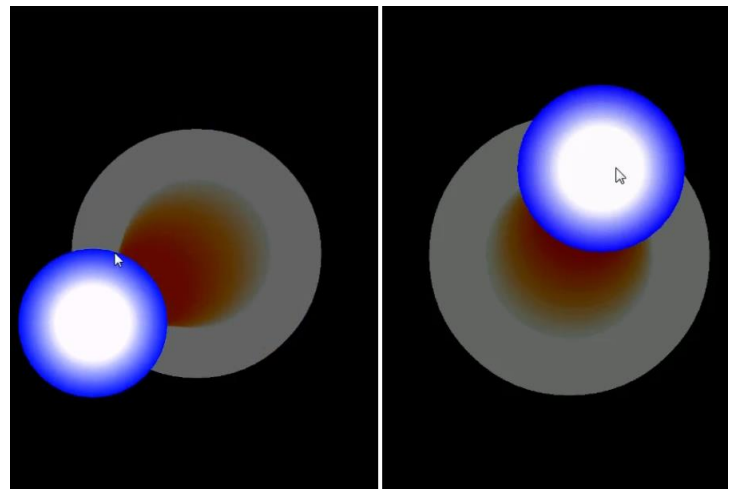


Figure 24

was the first one I tried, it featured little in the way of reading the inputs other than using radial calculation, which proved cumbersome and prompted me to switch to a different Joystick Implementation.

The second Joystick implementation from ControlWear is based on a project called Joystick view<sup>[20]</sup> and has, at time of writing a number of known issues posted on its GitHub page related to the refresh rate of commands so in the end I decided to use JoystickView for its simplicity, easy categorization of directions over angles and calculation of power as a % of distance away from center.

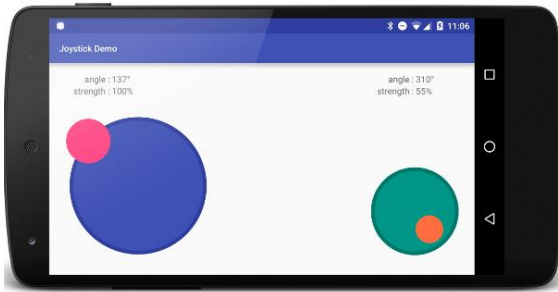


Figure 25

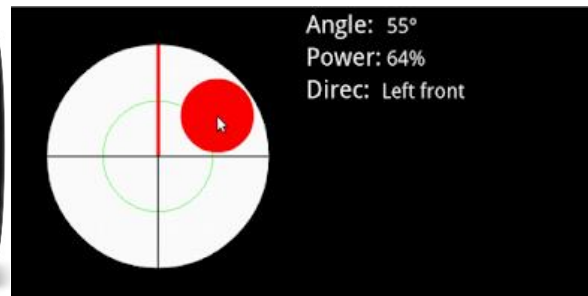


Figure 26

In order to communicate to the user which direction each drone is intending to move, it is important that there be some manner of visual representation on the app side. Accordingly, I used photoshop to create a slew of different drone representations to show which direction the user's drone(s) were headed in. Each of these change in resources will be handled by the android OS and used only when a message can be successfully sent to subscribed clients. Each drone will have a representation on screen during the activity where it is connected, with it's associated image resource changing depending on the current status of the connected drone it represents.

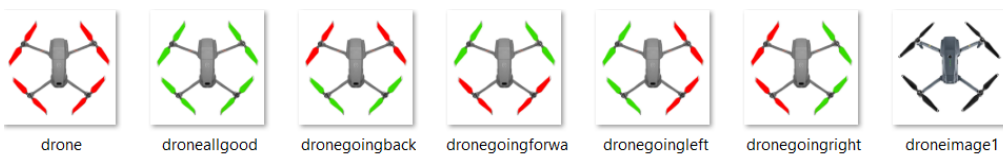


Figure 27

**RTC (Real Time Clock)** is a module for remembering date and time, as such they are useful for limited computing power microcontrollers that cannot keep track of global date and time without additional modules. The RTC keeps track of DATE and TIME by using millisecond calculation whereas an Arduino and the ESP 8266 simply keep track of the number of milliseconds since they were turned on and

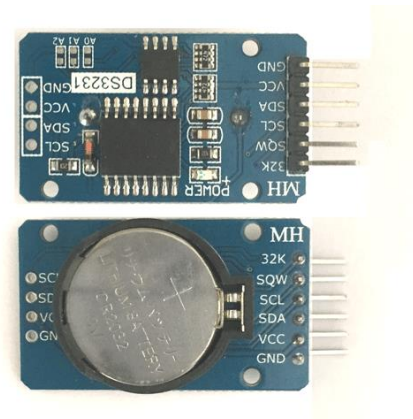


Figure 28

not the number of milliseconds since the 1<sup>st</sup> of January 1970, as is commonly used in real time clock modules and external time libraries. In our case, a RTC module could be used to ensure that all drones are receiving commands within the same rough timeframe and also to test the latency of commands sent to and from each drone.

Every MQTT system needs a broker in order to act as a router for messages. The MQTT broker receives each message and sends it to each device subscribed to that topic. For much of my testing I used a public broker known as EMQX, however I also set up my own local broker using a raspberry PI by installing Mosquitto MQTT in order to test if it could reduce latency, the results of which will be discussed further in my testing section.

### 3.3 IMPLEMENTATION

In this section I will discuss the process of implementing the requirements into the system and the process behind each activity and function the android app carries out will be explained using flowcharts and sequence diagrams.

#### 3.3.1 GROUPING

The first key problem to address in terms of implementing a system of controlling multiple drones is grouping. Each drone needs to know how many other drones are connected to it within its network. Using MQTT over WiFi, each device will publish and subscribe in this exchange. The android app subscribes to requests for formation and the drone subscribes to the formation assignment topic. As part of

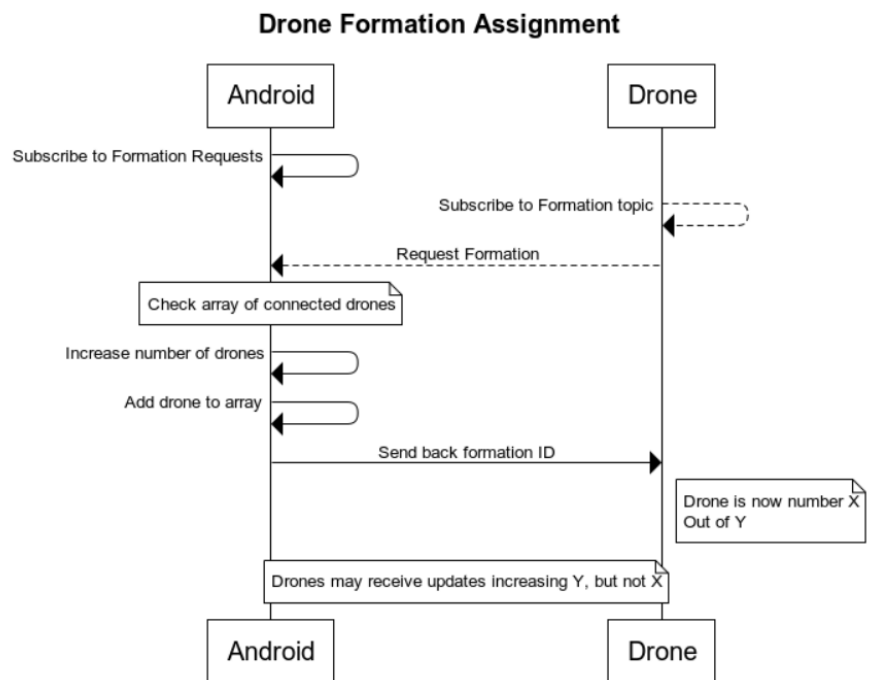


Figure 29

the setup sequence, each drone requests a formation and waits to receive a reply from the android device before unsubscribing from that topic. When the App receives a request for connection it checks its list of connected drones objects from local storage using shared preferences (Shared Preferences acts as small scale local storage for android applications). The drone is then added to the array of drone objects and assigned an index X (which represents its number in the list of currently connected drones) The variable Y represents the total number of drones connected in the list, which at this point will always be equal to X, so the variable X is sent back to the drone. The drone then correctly identifies itself as

number X out of a swarm of Y drones. Later the variable Y may change and as such the drone will subscribe to a topic where this can be updated later. This ensures that while X stays correct no matter how many drones connect to the system, Y will remain consistent with the number of drones the user sees are connected via the apps user interface. At this point each drone knows only the number of other drones in its network and its own number within it, however this is enough information for a smart microcontroller to begin to orient itself appropriately within the swarm, for example in the application a triple drone swarm will feature drone 1 out of 3 in the front of the swarm and drones 2 and 3 in the back, horizontally aligned each equidistant from each other. With each drone knowing its position in this formation and the makeup of this formation, future applications can easily instruct the drones to format themselves accordingly.

### 3.3.2 TOGGLING DRONES AND SUBGROUPS

Whenever the user presses on one of the drone icons on the activity for controlling their desired amount of drones, they will publish on the drone toggle topic with the id (X) of the drone they have selected to toggle, each drone subscribes to this topic for the duration of its uptime and has a different response depending on whether or not the command is for them. As covered in the section on Grouping, each drone knows that it is part of a

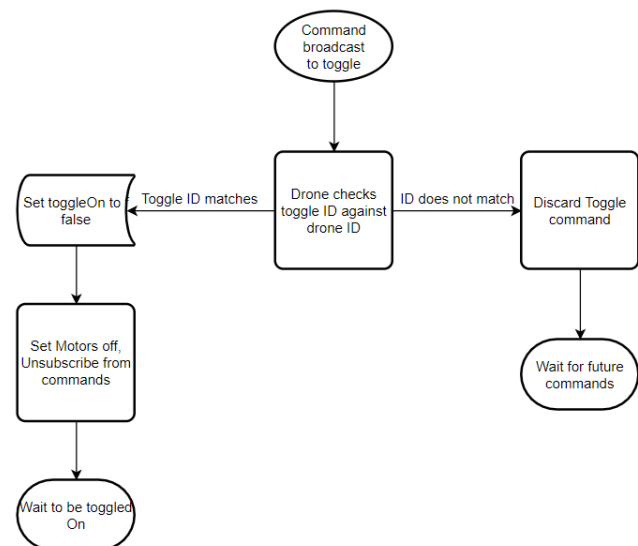


Figure 30

group of drones and that it is number X out of Y , this is demonstrated by the fact



that each drone responds accordingly to a toggle command. When a drone receives a message on the Toggle topic, it checks it against its own id within the swarm X and if they match it changes its status to toggled off and no longer responds to incoming commands. This creates subgroups within the swarm and allows for smart, dynamic control of multiple drones at once. If the drone receives a command that does not match up with its ID, the command is recognized but discarded. This is the only way of communicating this command over MQTT as MQTT does not allow for identification of devices on a network through something like a ping system. As such while there is some extra overhead, it is necessary to assure accurate grouping.

### **3.3.3 LATENCY**

Before using the local broker with Mosquito MQTT, I used a public broker for testing the connectivity of devices and getting a simple MQTT connection, this lead to an observable delay between sending the commands and the machine responding to them, so I began designing an activity that would determine the latency in a way that could be displayed to the user simply in a manner that they can easily understand. In order to do so it is important to understand how each device keeps track of time. The android System.getTime defaults to milliseconds and keeps track of the number of milliseconds passed since the 1<sup>st</sup> of January 1970. This allows the system to keep accurate track of time but also understand what day, date and month it is currently. The ESP 8266 and other microcontrollers only keep track of the number of milliseconds since they were powered on, as such a Real Time Clock module is necessary for an ESP to know the exact time. In order to calculate the time for a simple payload to travel from one device to another, the android calculates its current time and saves it as a long, before sending it over to the ESP which will use its RTC module to compare the current time with the time clocked on the App. The difference represents the time taken for the message to reach the ESP in milliseconds. Once this is calculated it is sent back to the android app, who can then log the most recent latency data, using a list of latency entries to calculate rolling average.

## Latency Calculation w/ RTC

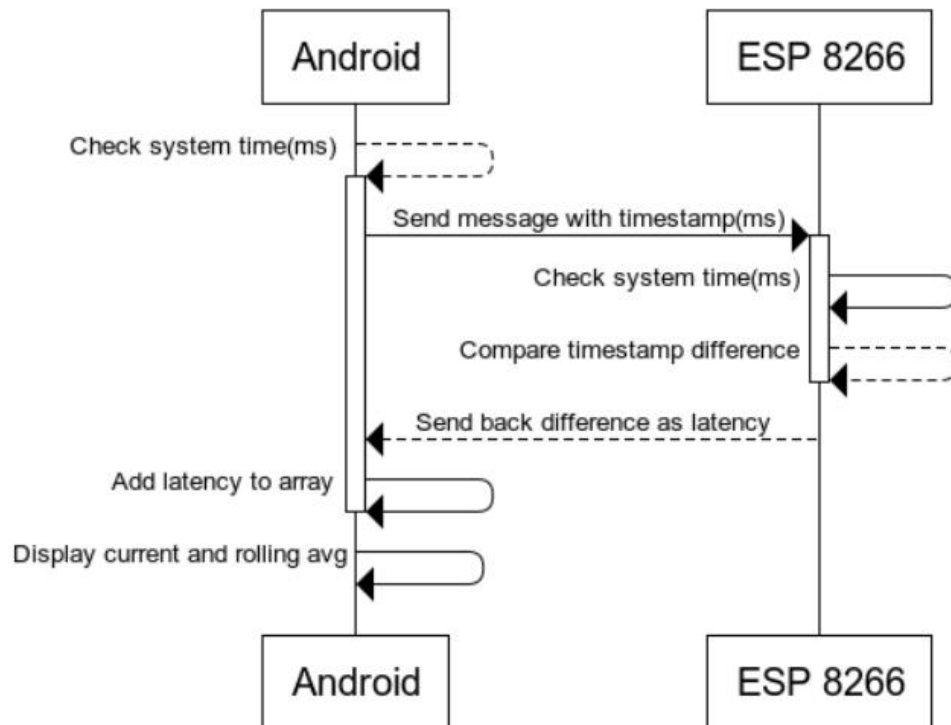


Figure 31

Due to technical problems and time constraints, I was unable to integrate the RTC module into my project in the way that I wanted to. This made calculating latency difficult without using repeated back and forth communication between the two, and due to the fact that the ESP 8266 has a limited number of topics that it can subscribe to in a short space of time without performance being affected, finding another solution was necessary. Instead of calculating latency from device A to device B, I could instead calculate the time it takes for a message to go from A to B and back to A and consider it as round trip time, with the latency being approximately half of the Round Trip Time. In order to calculate round trip time, the android app records system time, publishes on the latency request topic and subscribes to latency reply. The ESP then records its own time in milliseconds, While it is not necessary for the ESP to keep track of time, it is useful for analytics purposes. When the app receives a reply from the ESP 8266, it compares the current clock time with the previously recorded clock time and

the difference is the Round Trip Time, from which the latency can be derived. Finally it is added to an array of latency times and the entry is recorded, with the rolling average latency being adjusted according to the number of entries.

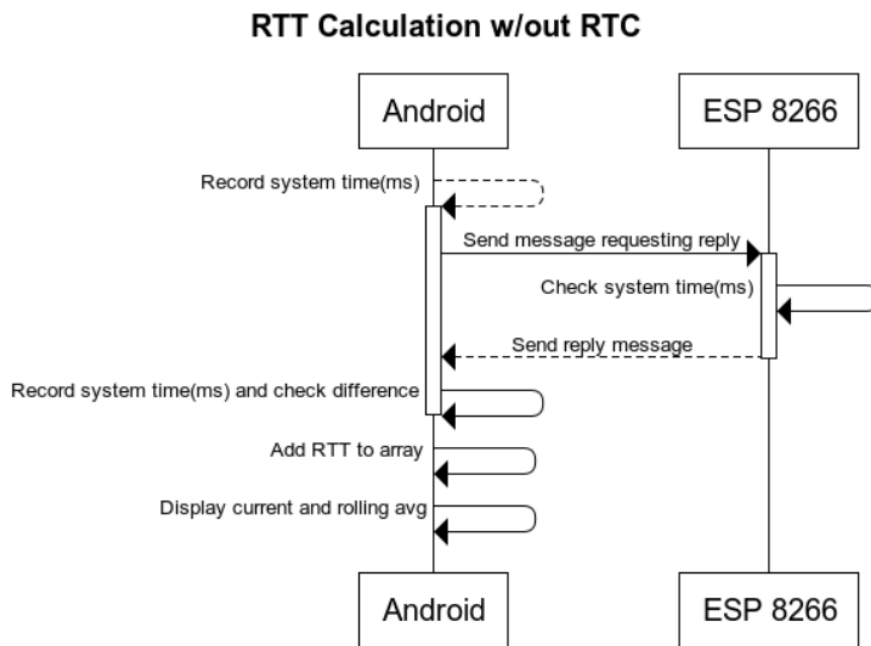


Figure 32

### 3.3.4 ADDITIONAL JAVA CLASSES

In order to implement some of the more complex functionality, additional java classes had to be created to store object values. For the calculation of Rolling latency, a rolling object had to be created which would store 2 data points, the number of entries and the rolling total added to it. Using the implementation of a function that calculated the average, this could be reused for any average calculation. Each drone object initialized on the system had an object representation in the android app with its position in the swarm (its ID, X) and its current status (Toggled on or off).

### 3.3.5 DRONE COMMANDS & NUETRAL RESET

In order to communicate to the user that the drone is moving in a natural way in accordance with their inputs, each activity features a display of one or more drones in accordance with how many have been selected to fly. When the joystick registers an input with greater than 50% power (this is calculated by the sensors distance from the central point) the joystick view uses the radius of the input to determine a general direction with an option to use 8 directional input (North, South, East, West, NorthEast, NorthWest, SouthEast and SouthWest), however for the purpose of this project, only 4 cardinal directions are needed. When the user indicates movement in a cardinal direction it is translated to a command of either Up, Down, Right or Left and interpreted as such. When a command is interpreted, all active drones are sent a command to move in the appropriate direction, this takes the form of sending an MQTT payload to drones with appropriate ID's depending on whether they are

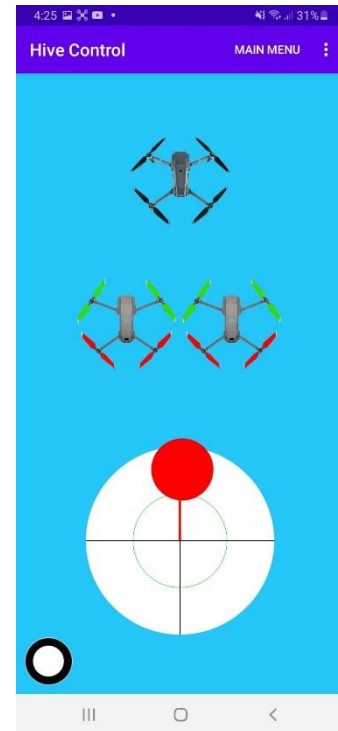


Figure 33

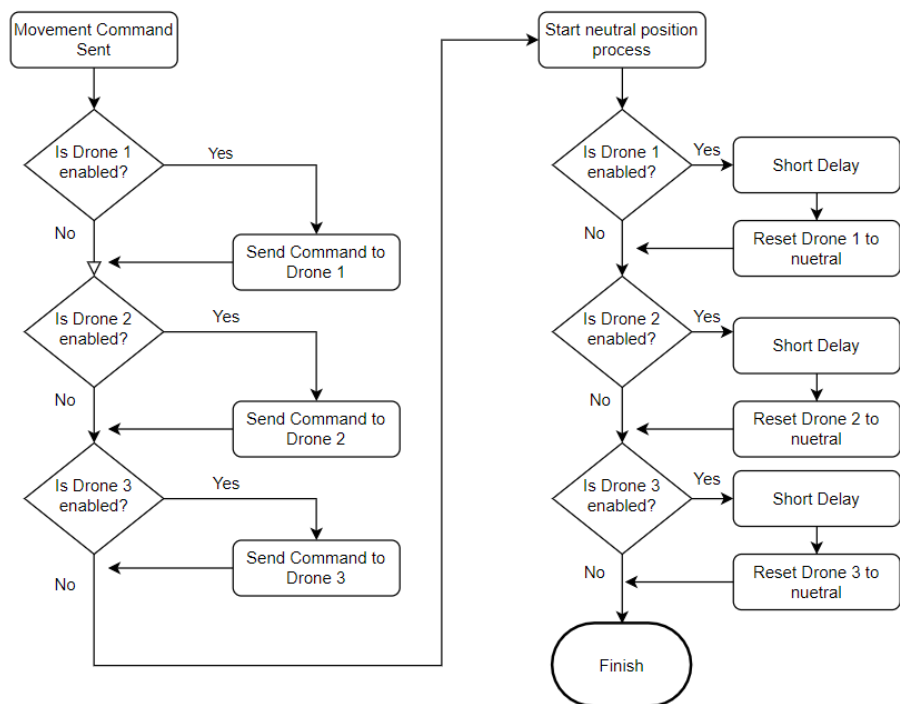


Figure 34

part of the active group or not. Once this is done, the ESP receives the appropriate command based on what the payload is, with up commands containing the character 'u', the left commands containing the character 'l', the right commands containing the character 'r' and the back commands containing the character 'b'. Next the ESP begins the process of moving in the appropriate direction by turning the appropriate LED's on or off depending on the direction. Accordingly, this should be reflected in the User interface that is displayed to the user. When the LEDs change on the drone, the image resource for all drones changed this way should change. Each command will only be represented on the drone for about half a second per entry. In order to represent the drone returning to neutral, there is a process for resetting it to neutral. When a drone's command has executed, it will start a thread that will intend to set the active drones to neutral if they are not busy after a short delay.

### **3.3.6 USING QoS 1 AND 2**

When implementing the system, I found that sometimes I would turn on a single drone and it would notify the app that 2 or sometimes even 3 drones had connected, this is due to the QoS used by default in the Paho MQTT implementation being QoS 1 or "at least once". It was at this time that I decided to reevaluate the ideal QoS for each topic. According to research on MQTT delay<sup>[21]</sup>, sending and receiving messages to different QoS can introduce significant delays, as such each of these desired QoS reflects both the publisher and subscriber. In order to ensure the least latency between drone commands, some have been left on 0 or "fire and forget" in order to maximize the speed with which they are sent. While these do not represent the implemented QoS for each test version of the project, below is a list of topics and their desired QoS.

<b>Topic</b>	<b>Publisher</b>	<b>Desired QoS</b>
<b>Latency Request</b>	Android	0
<b>Latency Reply</b>	ESP 8266	0
<b>Drone Command - Directional</b>	Android	0
<b>Drone Command - Toggle</b>	Android	0
<b>Formation Assign Topic</b>	Android	2
<b>Formation Request Topic</b>	ESP 8266	2
<b>Formation Y update</b>	Android	2

Ultimately there are some topics that desire speed over reliability, in this sense almost every topic uses either QoS 2 or 0, based on whether accuracy or speed is crucial. In the case of updating a drone as to the Y variable of drones in the network, the command needs to be sent exactly once or each drone could receive different Y variables, causing errors. In the case of the formation request and assignment, these were the first topics I implemented QoS 2 for, as it was important that the number of connected drones remain consistent throughout application and simulation. For Latency, speed is the only thing that is important and the loss of a latency request packet will not cause any significant problems for the user.

In this next section I will discuss testing each part of the system and how I came to certain results and conclusions.

## 4 Testing & Results

In this upcoming section I will discuss the testing of different parts of my implementation, including testing the latency activity with different brokers, testing the viability of a PID implementation using the HC – SR04 ultrasonic sensor and the implementation of the ultrasonic sensor with the ESP 8266.

### 4.1 TESTING LATENCY WITH DIFFERENT BROKERS

One of the most important aspects for control of a drone or system of drones is latency, if there is a latency. Therefore the latency activity exists to calculate the rolling average Round Trip Time in communication between a drone and the android application different MQTT brokers. Acceptable latency differs from person to person, with human reaction times varying greatly with an average of 250 milliseconds. However drones receive multiple complex commands per second and it can be more easily noticeable to the human eye. Studies on online gaming indicate that a delay between input and command under 100 milliseconds does not impact the users enjoyment, while Dr SC de Vries writing for the Dutch Ministry of defense notes that while military drones which use satellite navigation can tolerate a maximum of 1 millisecond latency, while commercial drones such as those examined in this report can tolerate delays up to 70 milliseconds without any impact on the perception of performance<sup>[22]</sup>. It is my belief however that a latency below 200 milliseconds will prove MQTT to be the best IOT solution for this subject as it would approach the latency of Zigbee, while possessing all of the additional advantages that MQTT offers, as explored in the research chapter. The Latency activity uses a library called MPAndroidChart to create line charts from incoming data using splash threads. This functionality was used to measure the time it takes for a message to be sent from the android device and back, however as I do not have an RTC module, this functionality tests Round Trip Time instead of latency as I had intended, latency can therefore be approximately derived using half of the round trip time. While this method doesn't account for added latency due to MQTTs overhead, studies

suggest that MQTTs delay when communicating on the same QoS level is insignificant, with delays only cropping up between QoS 1 and 0 and Qos 2 and 0 communication.

Initially for most of this testing, this project has used a public broker, EMQX in order to handle the delivery of topics in order to facilitate faster testing and development. However I needed to know if the public broker was slower than a local broker and as such I implemented a local broker through the use of a raspberry PI, the instructions for doing so can be found on Hackster.ios website<sup>[26]</sup>. Mosquitto MQTT is apparently the fastest MQTT broker on the market<sup>[23]</sup> so I wanted to try it out. Upon setting up the broker on the Raspberry Pi 4, I began testing and noticed slightly reduced latency. After 50 consecutive trials, I noticed that the delay from EMQX was no longer deviating more than ~10 milliseconds per entry, meaning that it had more or less evened out in terms of calculating average latency. Therefore I used this metric to test EMQX and decided I would use 50 trials for both in order to ensure a fair test.

This is then compared using a line graph simulation of Bluetooth delay, that simulates a random value between the lowest expected Bluetooth latency and highest tolerable latency.





Figure 35

Figure 36

As shown in the first screenshot, the rolling average round trip time for MQTT is never lower than Bluetooth, this confirms what we can expect from our research of Bluetooth being generally faster than MQTT over WiFi on my network, with some occasional bumps in latency. In addition to this, the difference between EMQX(left) and Mosquitto(Right) are not insignificant, while an average of 395 milliseconds is approximately 197 milliseconds for a single trip and 429 milliseconds round trip time translates to 214 milliseconds. In the case of my 50

trials an average improvement of 17 milliseconds was noted. While this seems like conclusive evidence in favour of Mosquitto, it should be noted that there are a number of variables that would need to be eliminated. I cannot say with certainty that Mosquitto is faster as I was running a local broker in my own home as opposed to a public broker with potentially significant traffic, It should be noted also that at entry 30, there was an outlier entry of 1000 milliseconds during EMQX testing that may have altered the average more than intended, even so these outliers should be observed and not discounted since we are judging the measure of this protocols ability to control a drone in flight and a single delay of 1 second could cause an unintended crash. As there are too many variables unaccounted for, including single hop latency, I cannot definitively say that one is more effective than the other.

Broker	Trials	Round Trip Time	Estimated Latency
EMQX	50	429	214 milliseconds
Local Mosquitto (Pi4)	50	395	197 milliseconds
Simulated Bluetooth Latency	100	335	167 milliseconds

While this is outside the bounds of what some industry experts would consider acceptable latency, and it is indicated to come with a visually noticeable dip in performance while in flight, it is my belief that this indicates a success, a local MQTT broker with multiple drones with flight controllers that can compensate for the additional delay may be the solution to an IOT based solution for multiple drones.

## 4.2 TESTING THE HC – SR04

The HC – SR04 ultrasonic module starts up by sending out a 10  $\mu$ s trigger signal in order to establish the distance of things around it, this is followed by 8 cycles of a quick sonic burst in order to calculate the range. As the speed of sound is equal to 340 m/s (or 0.034 cm/ $\mu$ s) and the formula for time is distance/speed, the distance in centimeters can be calculated using the formula shown in the second image. Though similarly to round trip time calculation, this variable needs to be divided by 2 in order to give us the distance in centimeters as it represents the total distance in centimeters travelled instead of the distance between the sensor and the object.

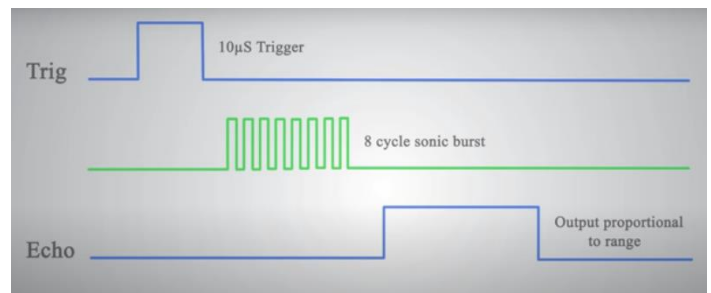


Figure 37

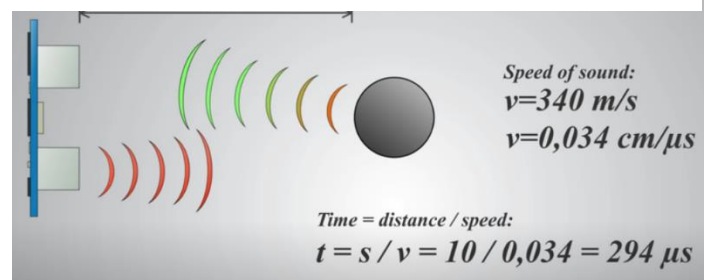


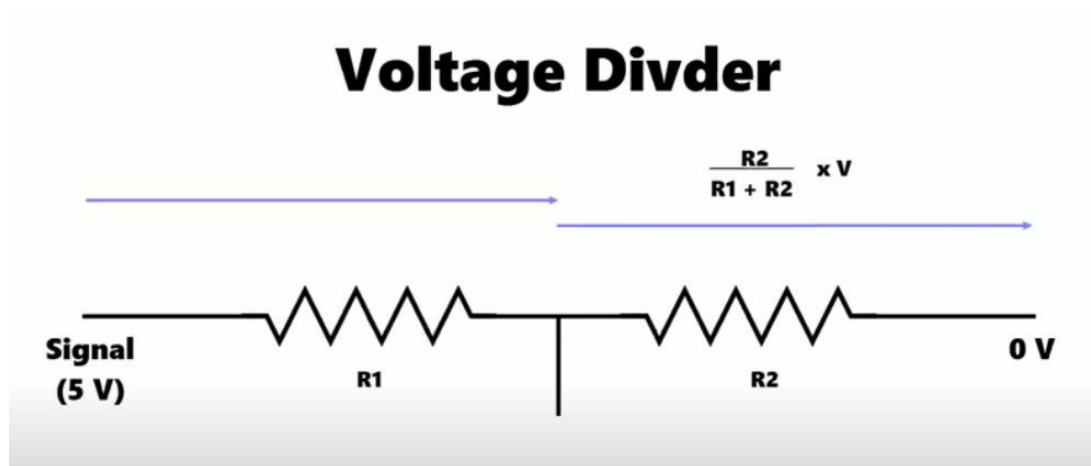
Figure 38

Crucially, the HC – Sr04 is a 5v device as opposed to a 3.3v device. The Esp 8266 is a 3.3v device. As such, in most cases a 3.3v signal in a 5v device will seldom cause problems as the 5v device considers the 3.3v signal 'Logic High' and can therefore interpret it. Thusly, the ESP 8266 has no problem sending the trigger signal from its GPIO pin as it is a 3.3v signal. If a 5v signal attempts to pass through a 3.3v device it can cause damage to the device as the device as the maximum these devices can read is usually around 3.6v<sup>[24]</sup>. This does not bode well for the ESP 8266 as a simulation of a drone with an ultrasonic sensor

as the 5v echo signal could cause damage to the device if left unchanged.

There are two ways to solve this problem, Voltage dividers and Logic Level Shifting.

Voltage dividers are the simplest and most applicable solution to this so this is what was tested. This is accomplished by using multiple resistors to help ground some of the signal, reducing the voltage dropped across the connection from 5v to a manageable number.



*Figure 39*

In order to determine the type of resistors needed to appropriately ground the signal, we can use the formula  $(r2/(r1+r2)) * V$  to find an appropriate voltage. Therefore using a 10k Ohm resistor for R1 and a 22k Ohm Resistor for R2 gives us an output voltage of 3.4v which is within acceptable parameters for most 3.3v devices. In practice however I was unable to acquire a 22k Ohm Resistor and for this reason had to chain other resistors together. Ultimately, despite this, the readings that came in from the Echo pin seemed to be inconsistent with what I expected, often it would have trouble accurately reading the position of my hand, I suspect that this may be due to the difference in voltage, an error with the voltage divider or damage done to the device when I was initially trying to read a 5v signal into a 3.3v device. Regardless, the implementation of a HC – SR04 ultrasonic sensor is ultimately not viable for this simulation as it cannot read accurate data under these testing conditions.

While a logic level shifter could be implemented to reduce the voltage down to an acceptable level, this would require the use of a piece of hardware that I was unable to test with. Logic level shifters interpret signals that come in at 5v to 3.3v before outputting them to the ESP. Currently it is my recommendation to use logic level shifters instead of voltage dividers as they are a simple solution to a complex problem.

## Logic Level Shifter

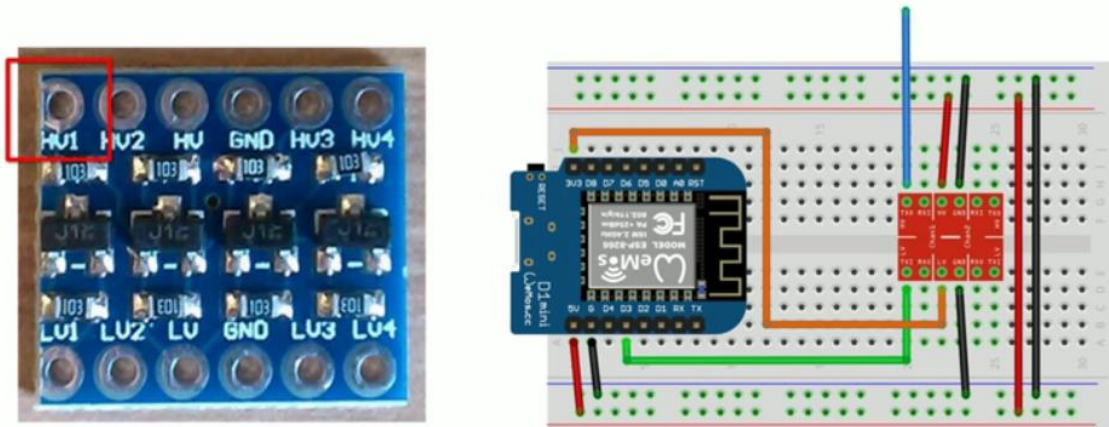


Figure 40

### 4.3 TESTING THE VIABILITY OF PID

In order to test the Viability of PID vs On/Of control within the scope of my application, I set up an ultrasonic sensor to connect to the 5v signal Arduino Mega, hooked up to a string of LEDs, one that would be controlled using a simple range detection function that would toggle the LED off when an obstruction is within a certain range and the others would be connected in a line, using a mock PID simulation that would reduce the number of LEDs lit up depending on the distance sensed in centimeters between the sensor and the nearest obstruction.

The string of blue LEDs would be fully lit when there is no obstruction nearby,

with subsequent LEDs becoming low at intervals of approximately 10 centimeters, there is a short delay in between these actions in order to simulate PID. The On/Off LED would simply be written low until the obstruction is removed. This same principle can be applied to the control of the drones, where the flight controller might instruct the ESC to write a lower output to each rotor, slowing the speed of the drone as it approaches an obstacle until it stops. While the Implementation of a PID system is outside the scope of this simulation, it is important to examine it to make recommendations for future work. Accordingly, PIDs main drawback is that it reduces the control an individual has over what their device is doing, this coupled with a 200 millisecond latency may prove too much of a difference between input to action for an average user.

#### **4.4 TESTING ANDROID ACTIVITIES & SHARED PREFERENCES**

When a new drone is detected via the formation request topic on the app, it is added to a list of items stored in local storage known as “shared Preferences” despite the name, SP is not only used to store a users preferences, but is a common way to access information from one activity to the next without the implementation of a local database. The number of total drones along with their IDs is stored in shared preferences and this is used to navigate automatically to the next activity based on the number of drones connected. During testing, it became necessary to add functionality that would allow the user to manually select the number of drones they wished to enter the fly activity with. While it proved useful for testing, I decided to leave it in, slightly modified in order to allow the user to select a number of drones smaller than the number of drones they had connected or to reset drone IDs. Accordingly each of these activities is correctly implemented as shown in the screenshots below. Each activity correctly displays the current movement status of the drones according to the users input and this is consistent with the movements displayed on the simulation. Accordingly the number of drones selected

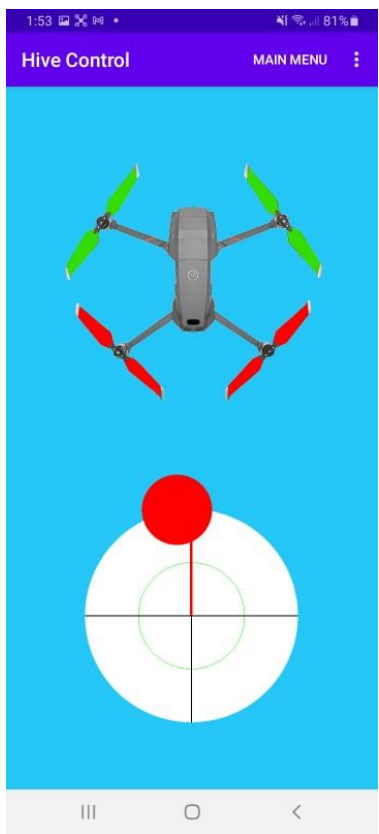


Figure 41

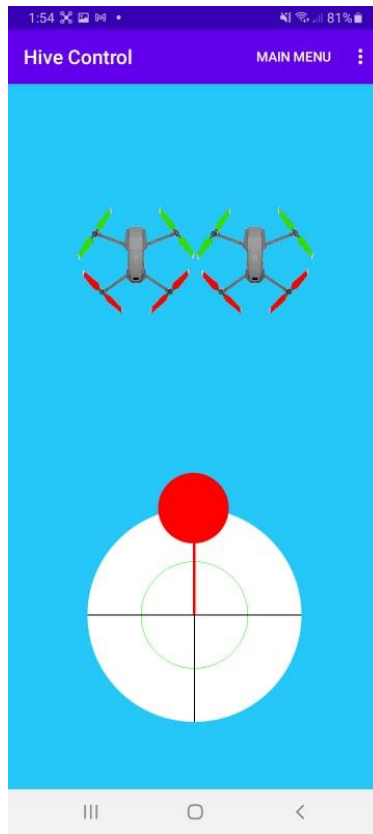


Figure 42

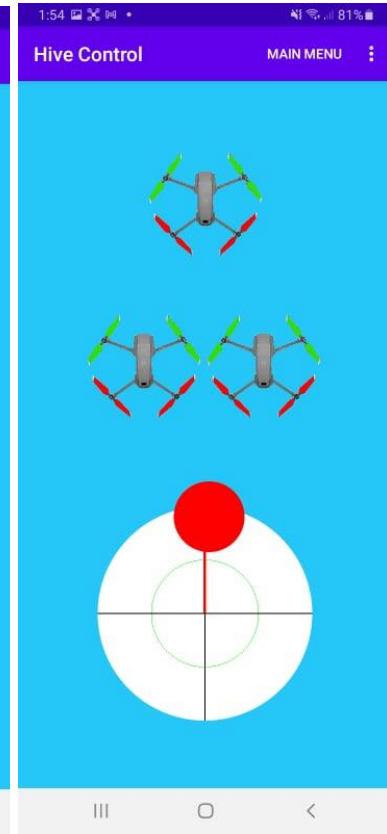


Figure 43

With the manual drone selection or the number of drones detected using the auto detect feature is always consistent with a notification the user will receive whenever a new drone is detected. A change in drone numbers will always publish a notification to the user , alerting them that a new drone is detected and displaying the number of total drones connected. This will transpire regardless of activity open or if the application is suspended, as it will run as a background service, checking for connected devices.

In this upcoming final section I will discuss my thoughts and conclusions on this project and the implementation of it, whether the project was successful or not in achieving its goals and answering its research questions as well as making recommendations for future works in this subject area.

## 5 Conclusions

### 5.1 SUMMARY

In summary, my project used ESP 8266 microcontrollers as simulated representations of Unmanned Aerial Vehicles for commercial use. My project explored various protocols used to communicate data between UAVs and smarthome devices alike. Ultimately it was decided that I would use MQTT over WiFi as the cornerstone of this project. Each drone simulation subscribes to commands and receives them via the implementation of an android app that can automatically determine which drone is which based on a program of my design. Each microcontroller can be outfitted with the same program as I designed it in such a way to allow for the controllers to automatically determine their own IDs within a group. My project also explored the validity of using Zigbee, Bluetooth and MQTT over WiFi as protocols for use in this project and determined that while MQTT over WiFi can be slow in some cases, and particularly slower than Bluetooth in most, the ability to contact multiple devices at once makes it invaluable. Each drone correctly receives commands using MQTT, each drone correctly assigns itself to an ID in a group or subgroup and each drone can be toggled dynamically on or off.

While the latency is beyond that which most research considers acceptable for optimum performance, as the design goals of this project was to accommodate for the average drone users needs and not for that of racing or military drones, I would ultimately consider the implementation to be a limited success.

### 5.2 IMPLICATIONS

My research into the latency requirements of drones and the impact that MQTT can have on Latency seem to indicate that the ESP 8266 cannot be properly utilized as a flight controller for a drone without significant modifications. However, MQTT is not a lost cause in the field of drone control. My research indicates a latency comparable to that of Zigbee. This, coupled with an increased



range over both alternatives make it the most viable IOT protocol for this project and projects of this scope. I started this project with the goal of creating an easy to implement system for controlling multiple drones, though what I have discovered is that there seems to be a noticeable difference in the latency between using a public broker vs a local broker on a raspberry Pi 4 and while it isn't difficult to implement one, it certainly impacts the ease of which the project can be adapted to suit its target audience.

In short, the implications of this are that MQTT over WiFi is a viable, if slightly slow method of controlling multiple drones and further research will be needed to implement a flight controller that can accommodate for this latency.

### **5.3 RECOMMENDATIONS**

It is my formal recommendation, therefore that any system that wishes to issue identical commands or group objects in an IOT environment should use MQTT over WiFi as it provides the greatest balance of scalability, adaptability and robustness, each of which are critical to designing a system to implement the concepts of swarm robotics.

It is my recommendation that any system seeking to utilize this concept use an Arduino Uno, Mega or similar other microcontroller in conjunction with the ESP 8266 to make up for its shortcomings in terms of processing power as for my project, several concessions had to be made to account for its inability to subscribe to more than 4 MQTT topics. Many drones feature cameras and in order to carry out their function, such as those meant for security or photography thus an implementation of a camera module on one of these drones could easily be achieved, however for this I recommend using Dynamic Streaming over HTTP as most IOT protocols such as Zigbee will deliver lower refresh rates.

Accordingly, I would recommend the use of a local broker in order to make up for the lack of security provided by a public broker, particularly if businesses wished to adopt this technology.

## **5.4 FUTURE WORK**

This technology provides much opportunity to allow for the completion of future projects. Future work could explore the technical requirements to get a swarm of drones airborne using an android device. There is room to explore a concept of sharing control between devices using MQTT. In order to make sure the drones have as much data as possible, future projects could implement a local database service such as Firebase, which is supported by Android studio. By using Firebase, each drone could be aware of the status of each drone around it in an easily accessible way. In addition to this, each drone could track its own latency data over time, providing complete data along its flight. On the subject of latency, more testing is recommended before a sound conclusion can be made as to what kind of broker should be used, however at this time I recommend a local broker using Mosquitto on a Raspberry Pi 4.

If I personally had more time and resources I would like to implement a feature that displays live latency data on each activity, for each drone however with the hardware limitations of the ESP 8266, subscribing to enough topics to make this work was not possible.

## Appendix

### ABBREVIATIONS

Abbreviation	Definition
ESC	Electronic Speed Controller
ESP	Embedded Systems Programming
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
IOT	Internet of Things
LED	Light Emitting Diode
MQTT	MQ Telemetry Transport
PID	Proportional, Integral, Derivative
PUBREL	Publish Release
PUBCOMP	Publish Complete
QoS	Quality of Service
RC	Radio Control
RTC	Real Time Clock
RTT	Round Trip Time
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver/Transceiver
UAV	Unmanned Aerial Vehicle
V	Volts, voltage
WiFi	Wireless Fidelity
μ	Micro

## TERMS GLOSSARY

Abbreviation	Definition
Activity (Android)	A screen of an apps User Interface
Android Pie (v9)	The 9 <sup>th</sup> version of android software
Arduino (Language)	A programming language modelled after c for programming microcontrollers
Arduino	Programmable open source circuit board
Bluetooth	Short range wireless protocol
C	A programming language
FireBase	A database application system supported by android
Flight Controller	A chip used in drones to control flight, see section 2.5
HC-SR04	An ultrasonic distance sensor
HC-06	A Bluetooth receiver
Overhead (data)	Additional information used to route packets
Pitch	Rotation about the side-to-side axis
Protocol	A set of rules allowing communication
Roll	Rotating about the front-to-back axis
Shared Preferences (Android)	A method of storing data locally in an app
Swarm Robotics	A field of science governing the control of many grouped machines
Transceiver	A piece of technology used to receive and sent data
Voltage Divider	A circuit which converts voltage
Yaw	Vertical Rotation
Zigbee	A wireless protocol used in IoT

## LIST OF FIGURES

Figure	Description
Figure 1	Diagram of drone controller
Figure 2	Diagram of Pitch/Yaw/Roll
Figure 3	Diagram explaining UART
Figure 4	Diagram explaining swarmtouch's Position based impedance
Figure 5	Diagram explaining swarmtouch's PID position control
Figure 6	Line Chart showing latency of drones in swarmtouch
Figure 7	Explanation of the principles of swarm robotics
Figure 8	Diagram showing an example of drone communication
Figure 9	Diagram showing an example of drone communication
Figure 10	Drone autopsy photograph - overhead
Figure 11	Drone autopsy photograph - Undercarriage
Figure 12	Drone autopsy photograph - Motor/Rotor/ESC
Figure 13	Drone autopsy photograph - Flight controller/Transceiver
Figure 14	Drone autopsy photograph - Power Supply
Figure 15	QoS 1 Diagram MQTT
Figure 16	QoS 2 Diagram MQTT
Figure 17	QoS 3 Diagram MQTT

Figure 18	Diagram Explaining Star networks – Wikipedia
Figure 19	Diagram Explaining Tree networks – Wikipedia
Figure 20	Formula for PID
Figure 21	Esp 8266 microcontroller
Figure 22	Example of sound returning to an ultrasonic sensor
Figure 23	Picture of 4 LEDS representing motors
Figure 24	Screenshot of Joystick implementation from instructables
Figure 25	Screenshot of emulator running Joystick implementation from ControlWear
Figure 26	Screenshot of Joystick implementation Joystickview
Figure 27	A display of different drone status images created for this project
Figure 28	An RTC module
Figure 29	A sequence diagram explaining the assignment of drones to a formation
Figure 30	A flowchart explaining the process of toggling drones
Figure 31	A sequence diagram using explaining Latency calculation with a RTC module
Figure 32	A sequence diagram using explaining RTT calculation without a RTC module

Figure 33	A screenshot of my applications commanding 3 drones
Figure 34	A flowchart explaining how drones are given commands
Figure 35	A screenshot of my latency activity using EMQX
Figure 36	A screenshot of my latency activity using Mosquitto MQTT
Figure 37	A diagram explaining the setup sequence for ultrasonic sensors
Figure 38	A diagram and formula explaining how distance is derived from sound
Figure 39	A diagram explaining voltage dividers and the formula required to calculate necessary resistance
Figure 40	A photo and representation of a logic level shifter
Figure 41	A screenshot of the single drone activity
Figure 42	A screenshot of the double drone activity
Figure 43	A screenshot of the triple drone activity
Table 1	Functional Requirements – Drone
Table 2	Non-Functional Requirements – Drone
Table 3	Functional Requirements – Application
Table 4	Non-Functional Requirements – Application

Table 5	IOT protocol comparison table
Table 6	A table of MQTT topics and their ideal QoS in my application
Table 7	A table comparing the latency of EMQX, Mosquitto and a simulated Bluetooth latency

## BIBLIOGRAPHY

- [1] Dukowitz, Z. (2019). *Drone Controllers: A Look at How They Work*. Available: <https://uavcoach.com/drone-controller/>. Last accessed 9th May 2021.
- [2] Lazarro, S (2015). *Flying multiple drones from 1 remote controller*. Wisconsin: University of Wisconsin-Madison. all.
- [3] Campbell, S. (2016). *Basics of UART*. Available: <https://www.circuitbasics.com/basics-uart-communication/>. Last accessed 9th May 2021.
- [4] Agishev, R et al (2019). *Tactile Interaction of Human with Swarm of Nano-Quadrotors augmented with Adaptive Obstacle Avoidance*. Glasgow: Published under creative commons. 4.
- [5] Labazanova, L et al (2017). *SwarmGlove: A wearable Tactile Device for Navigation of Swarm of Drones i*. Moscow, Russia: Skolkovo Institute of Science and Technology.
- [6] M. Dorigo and E. S. ahin, "Swarm robotics," *Autonomous Robots*, pp. 111–113, 2004.
- [7] Schranz, M et al (2021). *Swarm Intelligence and cyber-physical systems: Concepts, challenges and future trends*. Science Direct: Self Published Journal. 1.
- [8] Rubenstein, M (2014). *Kilobot: A low cost robot with scalable operations designed for collective behaviors*. Science Direct: Self Published. 12.
- [9] Brush, K & Bernstein, C. (2021). *MQTT (MQ Telemetry Transport)*. Available: <https://internetofthingsagenda.techtarget.com/definition/MQTT-MQ-Telemetry-Transport>. Last accessed 9th May 2021.
- [10] HiveMQ team. (2015). *MQTT Essentials: Quality of Service 0,1 & 2*. Available: <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/>. Last accessed 9th May 2021.
- [11] Silicon Labs (c.2014). *Zigbee Mesh Network Performance*. Austin, TX: Self Published. 4.
- [12] Uncredited Homey Staff. (2021). *What is Zigbee? Explaining the World's Most Popular Smart Light Network Technology*. Available: <https://homey.app/en-ie/wiki/what-is-zigbee/>. Last accessed 9th May 2021.



- [13] Android Developers Platform. (2018). *Android 9 features and APIs*. Available: <https://developer.android.com/about/versions/pie/android-9.0>. Last accessed 9th May 2021.
- [14] Uncredited NI staff. (2020). *PID Theory Explained*. Available: <https://www.ni.com/en-ie/innovations/white-papers/06/pid-theory-explained.html>. Last accessed 9th May 2021.
- [15] Freltas, R. (2017). *PID controllers and drones*. Available: <https://visaya.solutions/en/article/pid-controller-small-quad-copter-drones>. Last accessed 9th May 2021.
- [16] Lewis, J. (2015). *Four ESP8266 Gotchas and a tip for a first time users*. Available: <https://www.baldengineer.com/four-esp8266-gotchas.html>. Last accessed 9th May 2021.
- [17] How To Mechatronics. (2015, July 26). *Ultrasonic Sensor HC-SR04 and Arduino Tutorial* [Video]. YouTube. <https://www.youtube.com/watch?v=ZejQOX69K5M>
- [18] Mavlink. (2019). *Mavlink Developer Guide*. Available: <https://mavlink.io/en/>. Last accessed 9th May 2021.
- [19] Instructables (2017) <https://www.instructables.com/A-Simple-Android-UI-Joystick/>
- [20] JoystickView  
[https://github.com/zerokol/JoystickView?fbclid=IwAR3Ixx3iiJBnSqq1FEpeRF1FbemHEpri\\_9Ad8pFYLBH7ZAmuCUq6MxZNew2A](https://github.com/zerokol/JoystickView?fbclid=IwAR3Ixx3iiJBnSqq1FEpeRF1FbemHEpri_9Ad8pFYLBH7ZAmuCUq6MxZNew2A)
- [21] Lagerqvist, A & Lakshminarayana, T (2018). *IoT Latency and Power consumption*. Jönköping: School of Engineering in Jönköping. 17-30.
- [22] De Vries, SC (2005). *UAVs and control delays*. Soesterberg, Netherlands: TNO Defence, Security and Safety.
- [23] *MQTT Broker Comparison – MQTTRoute vs Mosquitto*. Available: <https://www.bevywise.com/blog/mqtt-broker-comparison-mqttroute-vs-mosquitto>. Last accessed 9th May 2021.
- [24] Brian Lough. (2017, June 5). *[TMT] Logic Level Shifting* [Video]. YouTube.  
[https://www.youtube.com/watch?v=2i\\_VMaX8lco](https://www.youtube.com/watch?v=2i_VMaX8lco)
- [25] Araman, A. (2019). *Difference between Bluetooth and Zigbee*. Available: <https://www.geeksforgeeks.org/difference-between-bluetooth-and-zigbee/>. Last accessed 9th May 2021.
- [26] Parikh, D. (2020). *Running a MQTT Broker on Raspberry Pi*. Available: <https://www.hackster.io/dhairya-parikh/running-a-mqtt-broker-on-raspberry-pi-63c348>. Last accessed 9th May 2021.
- [27] N. Anju Latha Et Al (2016). *Distance Sensing with Ultrasonic Sensor and Arduino*. Anantapur, A.P., India: Department of Instrumentation, Sri Krishnadevaraya University, Anantapur, A.P., India. 2-3

[28] IBM Documentation . (c.2010). *IBM MQ Telemetry Transport format and protocol*. Available: <https://www.ibm.com/docs/en/ibm-mq/9.1?topic=reference-mq-telemetry-transport-format-protocol>. Last accessed 9th May 2021.

[29] DroneNodes. (c.2021). *How to Choose a Flight Controller for FPV Quadcopter*. Available: <https://dronenodes.com/drone-flight-controller-fpv/>. Last accessed 9th May 2021.

[30] Dahiya, M (2017). *A Short Range Wireless Network: Bluetooth*. JANAKPURI, DELHI, INDIA: MAHARAJA SURAJMAL INSTITUTE. all.

[31] Uncredited Digi Staff. (c.2021). *Zigbee Wireless Mesh Networking*. Available: <https://www.digi.com/solutions/by-technology/zigbee-wireless-standard>. Last accessed 9th May 2021. – what is zigbee

[32] Uncredited HelloTech Staff. (2020). *What is the Difference Between Bluetooth and WiFi?*. Available: <https://www.hellotech.com/blog/what-is-the-difference-between-bluetooth-and-wifi> . Last accessed 9th May 2021. – difference between Bluetooth and wifi

[33] O'Gorman, S. (2015). *WiFi, Bluetooth or Zigbee?*. Available: <https://www.electronicsspecifier.com/news/analysis/wifi-bluetooth-or-zigbee>. Last accessed 9th May 2021.

[34] Koubasa, A (2019). *Micro Air Vehicle Link (MAVLink) in a Nutshell: A Survey*. El Manar, Tunisia: University of El Manar, Tunisia.