Quadrotor Controlled by an Onboard Smartphone

ELEC 49X Final Report

Group 18

Jeremy Roy	jeremy.roy@queensu.ca
Jérémie Jollivet	jeremie.jollivet@queensu.ca
Justin Chow	justin.chow@queensu.ca
Riley Kirkpatrick	riley.kirkpatrick@queensu.ca

Supervisor: Dr. Carlos Saavedra

TA: Zack Babcock

April 6th, 2018

Queen's University

Department of Electrical and Computer Engineering

Abstract

Unmanned aerial vehicles (UAVs) have a multitude of uses including land surveying, photography, research, and recreation. However, the cost can be prohibitively high due to the need for a multitude of sensors, a flight controller, and telemetry equipment. Simultaneously, many people possess unused smartphones. These devices are equipped with a GPS sensor, gyroscope, accelerometer, camera, and antennas. This project outlines the design and implementation of a low-cost alternative to a standard quadrotor with payload capability where all components, excluding the frame, motors, speed controller and battery, are internal to the phone.

There are two primary aspects to this project. The first aspect is the independent control of multiple motors through a two-channel analog audio port. Since analog signals are amplitude and time-varying, multiple amplitude-unique signals can be layered together on a single channel to be then de-multiplexed through circuitry. As a demonstration of this concept, a custom circuit was designed to de-multiplex four distinct pulse wave modulated (PWM) signals from a 2-channel audio port. The circuit was modelled in PSpice, and the behaviour was optimized through simulation. A printed circuit board (PCB) was then made and assembled before advancing to the testing phase.

The second aspect of this project is the programming of a custom flight-controller. This software stack is comprised of two applications: an onboard smartphone application and a remote controller application. The onboard smartphone application collects sensor data from the phone, runs the quadrotor's flight controller, and sends motor commands to the phone's audio port. The remote controller application connects to the onboard smartphone over a local area network, displays sensor telemetry from the quadrotor, and sends user-specified joystick commands to the quadrotor. Both applications are implemented using Robot Operating System as a communication layer.

A prototype UAV was assembled using a DJI F450 quadrotor frame. This frame was determined to be large enough to house a smartphone, a 5400mAh battery, and the custom PCB. The quadrotor's motors and electronic speed controllers were chosen such that the vehicle has a thrust-to-weight ratio of 2.4. Given that the unloaded quadrotor has a mass of 1150 grams, the quadrotor is capable of lifting a payload of 500 grams with an estimated flight time of 15 to 20 minutes.

The assembled prototype was tested while tethered to a surface. A user was able to control each motor individually via a remote computer. Motor testing demonstrated the ability of the custom circuit to convert two audio channels into four digital PWM signals. Motor testing also demonstrated the successful implementation of the flight controller stack.

Most targets of this project, with the notable exception of stable flight, were achieved successfully by the team. Recommendations for future works include the development of a dynamic model of the quadrotor to improve flight controller tuning as well as further software development to implement autonomous functionality.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Background

A quadrotor is an unmanned aerial vehicle (UAV) or drone that is lifted and propelled by four rotors. While the concept of quadrotors has been around since the beginning of the 20th century, it wasn't until the early 2000s that they started to gain popularity amongst robotic researchers [1]. In 2015, hobbyist drones reached the mainstream when the American Federal Aviation Agency granted hundreds of new exemptions for companies to operate drones within the United States [2]. Since then, the hobbyist drone market has grown rapidly, and the quadrotor has become one of the most popular configurations for drones.

The increasing popularity of quadrotors has steadily decreased the price of commercial drones. However, even today, high-end quadrotors can cost over $1000 USD [3]. The high cost can largely be attributed to the vast number of sensors and actuators, a need for microcontrollers, and the telemetry equipment. Higher-end quadrotors usually include a Global Positioning System (GPS) module and a camera capable of recording 1080p or 4K video.

According to a 2017 survey conducted by Nanos Research, 62% of the 3000 Canadians surveyed said they had one or more unused smartphones, with an average of 2.1 smartphones per household [4]. Many of these unused smartphones contain ample processing power, a wide range of networking capabilities such as Wi-Fi and cellular data, and a suite of navigational sensors such as an accelerometer, gyroscope, and GPS. While considering the capabilities of these unused smartphones, the team explored the idea of creating a quadrotor controlled by an onboard smartphone. Such a device has the potential of reducing the cost of a quadrotor by eliminating the need for communication and telemetry equipment.

The idea of a quadrotor with an onboard smartphone is not new. For example, a team at the Vienna University of Technology presented "a low-weight and low-cost Unmanned Aerial Vehicle (UAV) for autonomous flight and navigation in GPS-denied environments" using an off-the-shelf smartphone as its core on-board processing unit [5]. In a separate study, Chirtel and colleagues attempted to create a spatially

aware, autonomous quadrotor using an onboard smartphone [6]. These studies both detail designs in which the quadrotor stabilization logic was implemented on a microcontroller that was interfaced to the phone via Universal Serial Bus (USB) connection.

1.2 Problem Scope and Definition

The team proposed to design and build a quadrotor controlled by an onboard smartphone without the need for an external microcontroller. The onboard phone would replace the quadrotor's antennas, transmitters, sensor module, flight controller, camera, and GPS. Avoiding the use of an external microcontroller reduces system complexity by removing a processing unit and its associated software. This is advantageous to the end user, as they would not have to perform microcontroller firmware updates. The team envisioned that the use of an external microcontroller could be avoided by implementing flight stabilization logic on the phone and outputting motor control signals to the phone's analog audio port. The electrical hardware would condition the phone's output signals via an analog signal processing circuit and convert them into a format understood by the quadrotor's electronic speed controllers (ESC).

The scope included the design and assembly of electrical and mechanical hardware, and the development of onboard smartphone and offboard remote control software. The hardware would include the drone frame as well as custom power, safety, and signal processing circuitry. The software would be responsible for generating the required audio output signals and allow the teleoperation of the quadrotor. The project would also be designed for compatibility with Samsung Galaxy S3, S4, and S6 smartphones, as these were the phones possessed by team members. From a technical point of view, this project was challenging due to the wide scope, breadth of knowledge required, and the numerous deliverables proposed in the blueprint document from September 2017.

The team aimed to adhere to two major constraints as dictated by the ELEC 49X course. First, a budget of $400 was specified. Second, the prototype was requested to be completed by February 15, 2018 and ready for demonstration at the ELEC 49X Open House event.

The stakeholders for this project are the team, Dr. Carlos Saavedra, Queen's University, and the Queen's Department of Electrical and Computer Engineering. Due to the diversity of a quadrotor's practical applications, the range of stakeholders can be expanded to law enforcement officers, enthusiasts and researchers. Each of these categories would potentially benefit from a low-cost alternative to the high-end systems currently on the market.

Although the environmental impacts of the project were considered, its effects are estimated to be small when considering the millions of unused smartphones in Canada. The re-use of old smartphones would, however, contribute to a reduction of electronic waste and eliminate the need for manufacturing new quadrotor sensors and telemetry equipment [4]. Upon returning the materials to the department of electrical and computer engineering, the team will recommend that the LiPo battery and electronics be re-used, or properly recycled.

The team also considered the societal impact of the project. It is noted that beyond the environmental impact of improper smartphone disposal, there are toxic heavy metals present in smartphones that can cause public health issues. Once again, the effects of reusing smartphones would be small, but may contribute to a reduction in e-waste.

Additional societal impacts lie in the quadrotor's potential in research, terrain mapping and photography. A cheaper alternative and readily available equipment would render these services more accessible; these services include maps that can be updated on a more frequent basis, aerial photography and drone footage for enthusiasts, or visual aid for law enforcement agencies. Quadrotors can also be used to survey and examine areas that are too hazardous or too remote for humans. The increased air traffic, however, is an important impact that will need to be studied and considered. As more enthusiasts and professionals take part in this activity, the risk of flight within restricted airspaces such as residential housing, airports, natural disaster areas, and helipads become more probable.

2 Design

The design section is divided into three parts. Sections 2.1 and 2.2 outlines the design considerations of the UAV and the circuitry, while Section 2.3 elaborates on the software requirements and design.

2.1 Quadrotor body

2.1.1 Design Requirements

Five design requirements were considered with stakeholder needs in mind. These drove the design of the quadrotor and are detailed in Table 8.

Aircraft type

The UAV is built on the existing DJI F450 quadrotor frame platform. A rotary wing design was chosen over a fixed wing design as it offers a high degree of maneuverability that allows the UAV to change directions quickly. The high degree of control is most frequently utilized in photography and mapping applications. The rotary wing is also capable of hovering which allows for minimal landing space and was critical to performing indoor tests. This capability is not possible with a fixed-wing aircraft that relies on the wings generating lift from the vehicle's forward velocity, and the shape of the airfoil.

However, this choice came with constraints, such as the additional complexity. A quadrotor requires control over four brushless motors whereas a fixed wing design uses one brushless motor and three servos. The complexity of additional motors is also the reason for which hexacopter or octocopter designs were not pursued. Furthermore, the rotary wing design is more susceptible to disturbance, which requires a more complex control algorithm for the quadrotor in comparison to a fixed wing design [7]. Finally, a rotary wing design has a low-efficiency caused by the necessity to generate lift continuously, and the stabilization software rapidly accelerating and decelerating the brushless DC motors.

Weight and payload fraction

The takeoff weight of the quadrotor is an important design requirement as it is the principal contributor to the flight time of the UAV. As a result, the weight of all individual components must be

carefully considered. For a fixed motor and ESC specification, reducing the takeoff weight of the UAV also results in a higher payload fraction. The payload fraction is the ratio of the payload weight to the takeoff weight of the aircraft. For this project, a list of material weights was developed, and is documented in Table 1. A takeoff weight of 1105 grams and a target payload fraction of 0.45 are thus considered.

The proposed UAV is categorized as a recreational UAV, with a weight range of 250 grams to 35 kilograms. Although Transport Canada laws on drones were considered, the upper limit on legal weight is large enough that it did not impact the design of the quadrotor [8].

Flight time

The flight time is influenced by the total weight of the UAV, the type of battery and its discharge capacity. The flight time specification is especially important because it closely impacts the number of applications for the UAV and the target user base. As a larger battery results in additional weight, a trade-off exists between payload fraction and flight time. For this project, a standard flight time of 15 minutes is considered to put the UAV on par with existing equally weighted quadrotors such as the DJI Phantom Series [9].

Cost

The proposed design is intended to be a low-priced and versatile alternative to more expensive drones. Upon researching existing commercial solutions, such as products by Eachine and Yuneec, a starting price of $200 was found. This price point, however, does not feature any payload capability, camera options or autonomous functionalities. As the complexity of the UAVs increases through specialized camera software, GPS and navigation capabilities, the price increases to over $1000 [9] [10] [11]. The cost of materials is therefore an important consideration for the proposed final product. The team aimed for an expected mass production cost of $100, not including the battery and smartphone. The estimated material cost put the solution well into the price range of the target audience. For this project, however, the team built a prototype and the model presented at the Open House is not a reflection of the final product cost.

Fall height

The quadrotor was designed to withstand a fall from three metres. This specification was added to ensure that rigidity and structural integrity were considered when designing the quadrotor in preparation for prototype testing. The required fall height also dictated that no loose wires should be exposed, which demanded the design of a custom printed circuit board (PCB), where all connections should be soldered or made with header pins. The phone is secured in a protective case, and the quadrotor has landing skids to absorb the shock and reduce damage. The only replaceable parts subject to breaking are the propellers.

2.1.2 Models

The quadrotor consists of a frame onto which 4 ESCs, four motors and propellers, the smartphone, a PCB and battery are attached. A list of components and their weights is detailed in Table 1.

Table 1: Expected weight breakdown of quadrotor

	QTY	UNIT WEIGHT	TOTAL WEIGHT
BATTERY	1	250g	250g
PHONE	1	140g	140g
FRAME	1	248g	248g
LANDING SKIDS	4	10g	40g
MOTORS	4	50g	200g
ESC	4	25g	100g
PROPELLERS	4	8g	32g
PCB	1	25g	25g
PHONE HOLDER	1	50g	50g
SAFETY CIRCUIT	1	20g	20g
TOTAL			1105g

The motors and ESCs had to be designed to accommodate the estimated weight of 1105 grams. In order to choose the correct set of specifications, a pull calculation was made to estimate the required maximum thrust to be exerted by each motor. The calculations use a thrust to weight ratio of 2:1 and a safety factor of 5%, for a total thrust factor of 2.1. These are seen in Table 2.

Table 2: Pull calculations

	DESCRIPTION	WEIGHT
TAKEOFF WEIGHT	Expected weight	1105g
PAYLOAD WEIGHT	Expected payload	500g
REQUIRED THRUST	$(Takeoff\ weight + payload\ weight) * Thrust\ to\ weight\ ratio\ of\ 2$	2210g
SAFETY FACTOR	$Required\ thrust * Safety\ ratio\ of\ 5\%$	111g
TOTAL REQUIRED THRUST	$Required\ thrust + Safety\ factory$	3371g
REQUIRED THRUST PER MOTOR	$\frac{Total\ required\ thrust}{4\ motors}$	842g

The motors chosen for the final design are required to pull 842 grams. With the selected motor, the maximum current specification of the motors determined the current rating of the ESCs.

The ESCs are designed to receive pulse width modulation (PWM) signals. As such, the PCB processes the audio signal to mimic this behaviour. The circuit outputs are connected to the signal port of the ESC which drives the brushless motors by applying the desired voltage. All four ESCs are connected in parallel to the battery. The circuit schematic of the quadrotor components is shown in Figure 1.

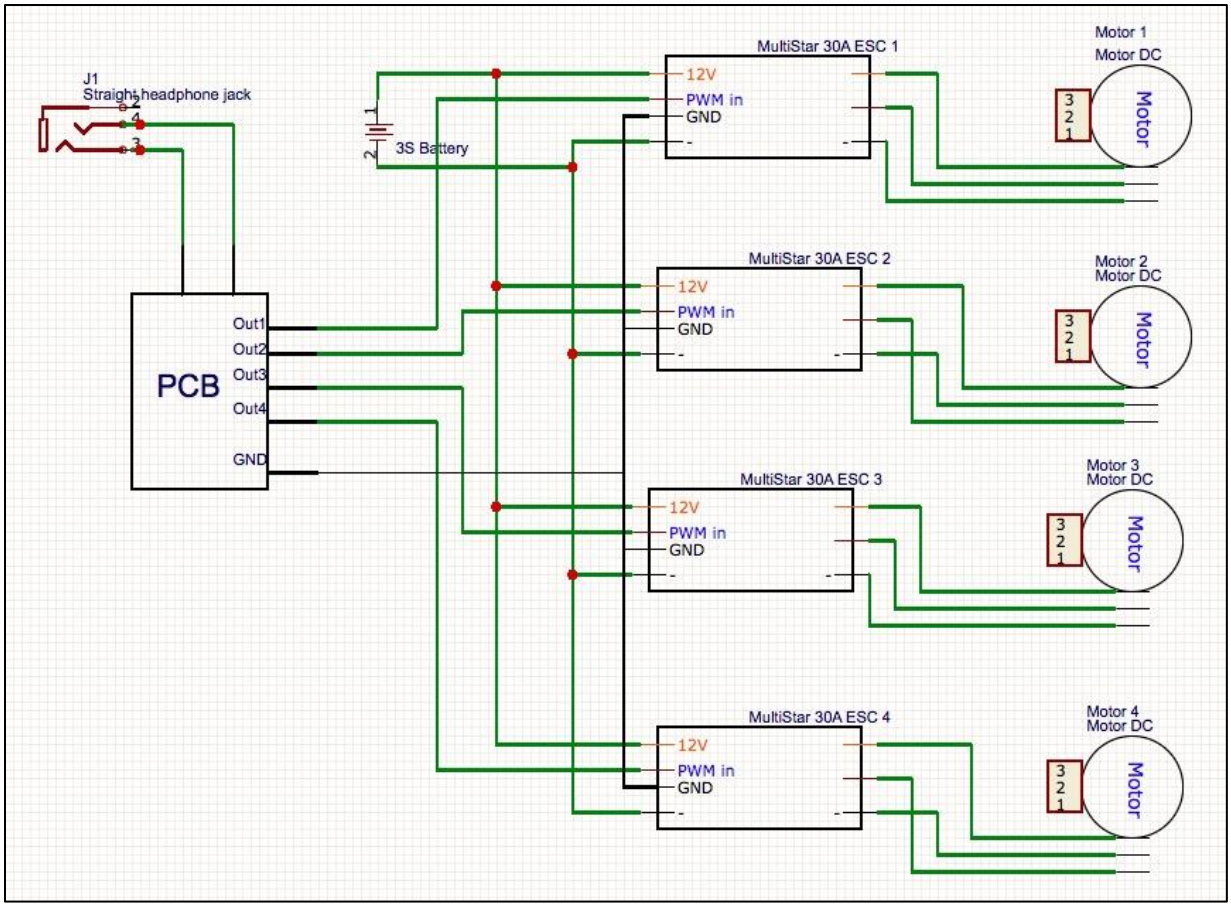

Figure 1: Wiring schematic of main components in the quadrotor

2.2 Electronics

2.2.1 Design Requirements

There are two primary functions to the circuit. The first function is the conversion of the phone's audio output into direct motor control signals, and the second is the On/Off powering of the motors from two switches, both onboard and remote. The power control functionality was added as a design requirement to permit safe testing of the prototype. For clarity, the Left and Right audio channels will subsequently be referred to as Input A and Input B.

The conversion circuit is required to convert an analog signal on each input channel (inputs A and B), to four output square signals (outputs 1, 2, 3 and 4). The circuit should be able to resolve any combination of layered inputs into accurate PWM outputs. The power circuit should be able to disable

motors through both a built-in switch and a remote switch. A detailed breakdown of the functional requirements can be found in Table 8. Metrics for circuit design were taken from Samsung Galaxy S3, S4 and S6 phones since these models were available for use by the team.

Conversion Circuit Design

The conversion of two inputs into four outputs is simplified by converting a single input into two outputs, with the assumption that outputs 1 and 2 do not depend on input B, and that outputs 3 and 4 do not depend on input A. The analog nature of the phone's audio output can be used to differentiate the input signals where the amplitude of the input will correspond to an output mode. The logic for the conversion begins with an analog comparison to DC threshold voltages A, B, and C. If the signal is greater than the threshold, the comparator will output 5V, or a logic 1. The output logic is found below.

$$Output\ 1 = \bar{A} * C$$
$$Output\ 2 = B * C$$

The relationship between input and output is visualized in Figure 2. This design is theoretically robust enough to resolve the correct outputs from any combination of frequency and duty varying signal inputs.

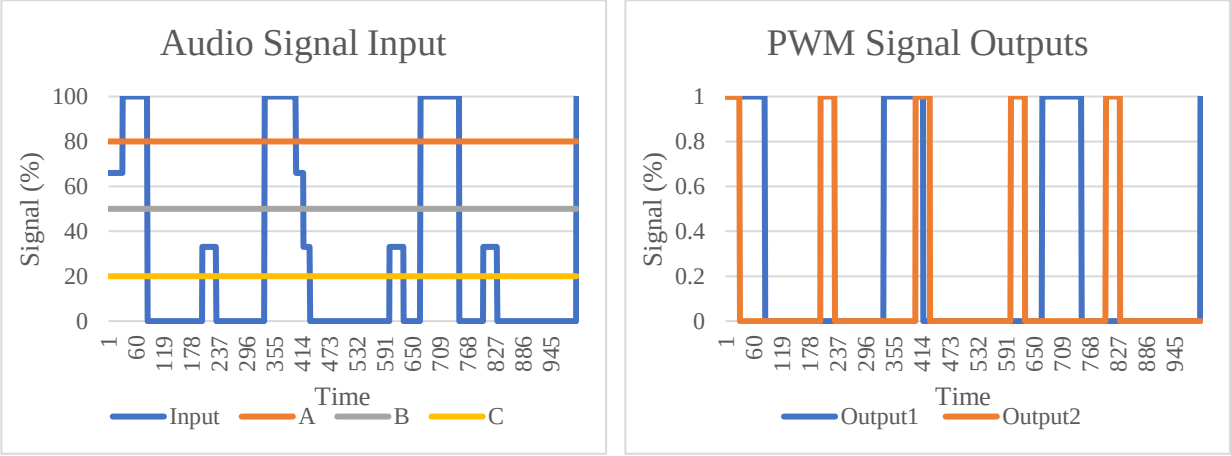

Figure 2: Visualization of PWM de-multiplexing from the multiplexed audio input

The Phone Output

The output of the Samsung Galaxy S4 is an analog signal with a measured output voltage of $\pm 150mV$. Figure 3 illustrates the output of the phone at maximum volume for a square wave at 5% duty.

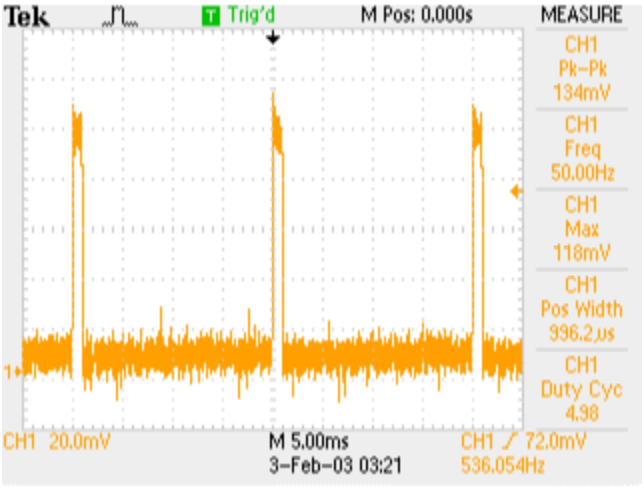

Figure 3: Audio Input for Samsung Galaxy S4 for 5% duty, 50Hz, 100% output

Each model of phone, however, has a different output power, resulting in a different maximum signal voltage. It is necessary to adjust the signal with an amplifier and a potentiometer such that the output voltage can be precisely selected. Figure 4 shows the calibration curves for the outputs of a Galaxy S3, Galaxy S4 and Galaxy S6 smartphone at different calibration targets.

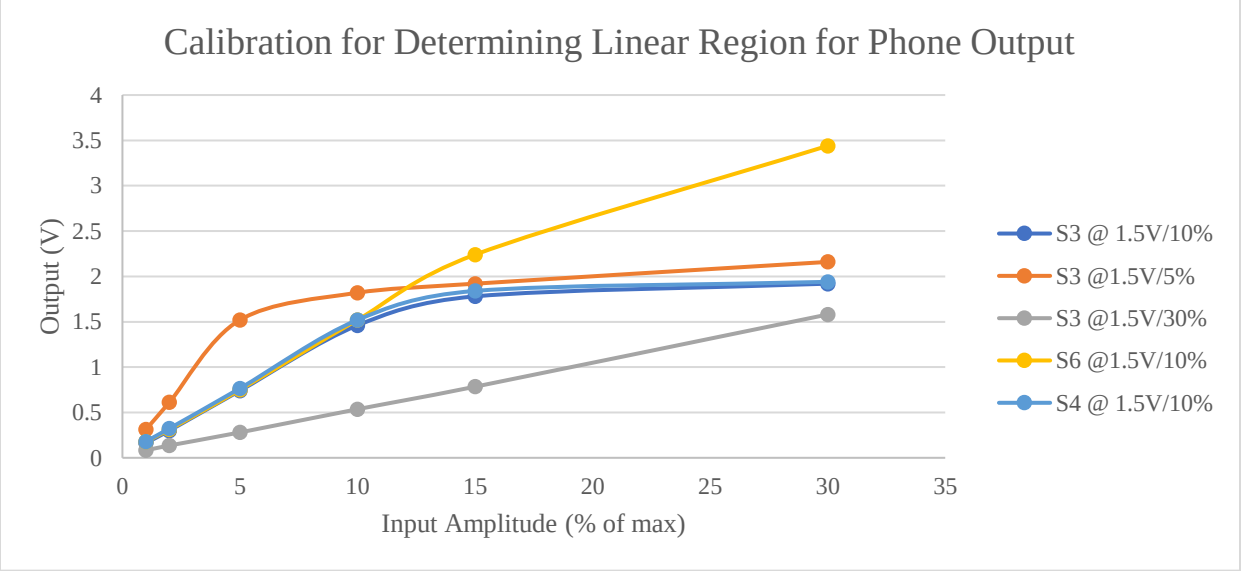

Figure 4: Phone Output after amplification for various models and calibration targets

This plot demonstrates that although different smartphones have different audio power specifications, a region of linearity exists between the output signal and output power. By altering the position of the calibration potentiometer, the output voltage can be varied for a constant volume input. The input amplitude of 10% of the maximum phone volume, which results in a voltage output of 1.5V, was chosen for all further

calibrations. This choice was made because all three phone models can output 1.5V while remaining within the bounds of the calibration potentiometer. The amplifier did introduce a DC bias to the signal, which was later removed with a passive high pass filter at a cut-off frequency of 0.7Hz. Figure 5 shows the oscilloscope output for an S4 at 5% duty after filtering.

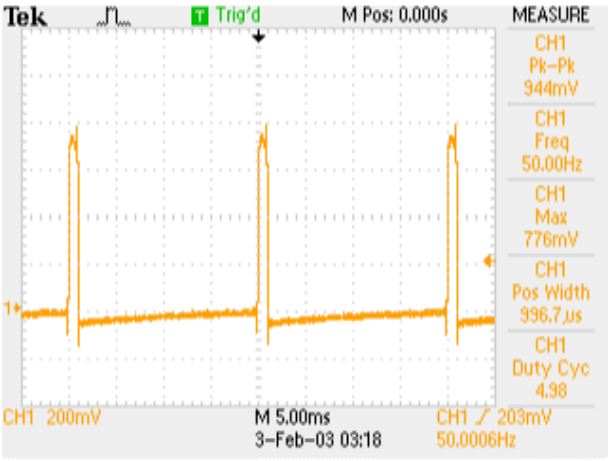

Figure 5: 5% duty input signal after calibration and filtering

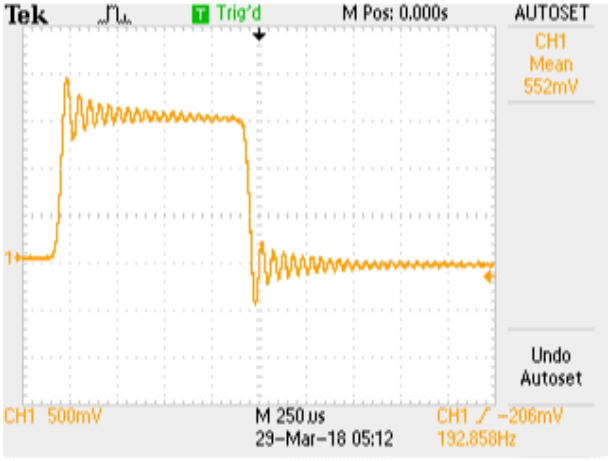

Figure 6: Ringing on 5% duty input signal

Upon inspection of Figure 6, it was noticed that the phone does not produce a pure square wave as intended. The cause of this unusual ringing effect was found to be the low pass filter built into the phone's audio unit. This is due to a square wave being constructed from an infinite sum of sine waves of increasing frequency, as defined by the equation below.

$$f(x) = \frac{4}{\pi} \sum_{n=1,3,5,\ldots}^{\infty} \frac{1}{n} \sin\left(\frac{n\pi x}{L}\right)$$

A low pass filter removes the high-frequency components after the cut-off, and the information required to make a perfect square wave is not recoverable. The result of the filter on a square wave as calculated in MATLAB in Figure 7.

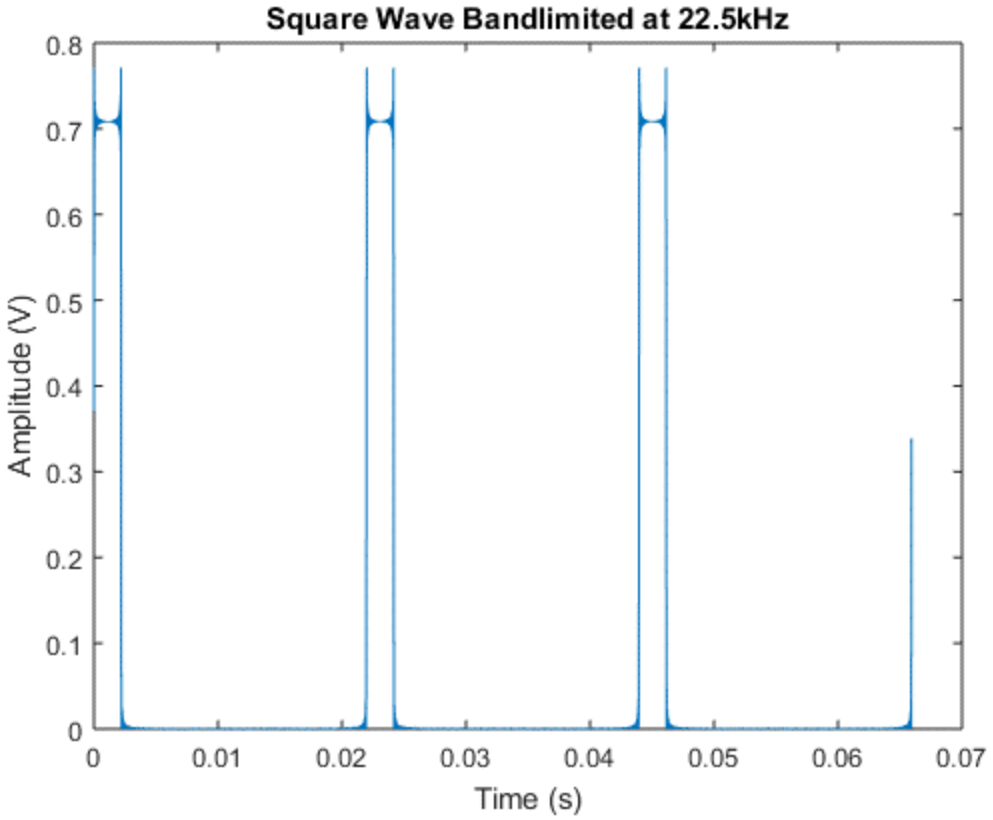

Figure 7: MATLAB model of the input signal

As seen in Figure 8, when the phone signal is observed in the Fourier domain by the oscilloscope's FFT function, the cut-off of the phone's filter appears to be 22kHz. A low pass filter with a cut-off frequency matching that of the phone is added to the high pass filter to reduce any high-frequency noise.

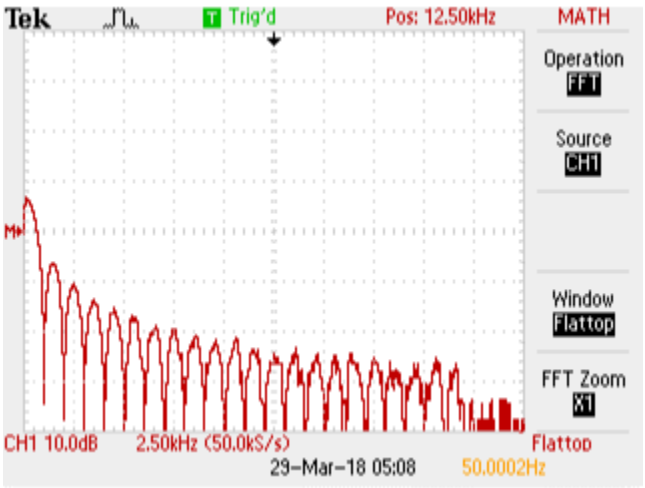

Figure 8: Capture of input signal's frequency spectrum

At 22kHz, the maximum percent overshoot/undershoot that will occur for any square wave duty will be less than 10%. Thus, threshold levels are chosen such that the ringing caused by filtering will not overlap and cause false detections. For the selection of 2%, 5%, and 8% of the maximum amplitude, the range of ringing and the range of observed voltage are summarized in Table 3. The final threshold selections are documented in Table 4.

Table 3: Input range of values for desired outcome amplitudes

OUTPUT		**DESIRED OUTCOME**	**RINGING RANGE (% MAX OUTPUT)**		**VOLTAGE (MV)**	
OUTPUT 1	OUTPUT 2		MIN	MAX	MIN	MAX
OFF	Off	0%	0	0.8	0	120
ON	Off	2%	1.8	2.2	220	330
ON	On	5%	4.5	5.5	675	825
OFF	On	8%	7.2	8.8	1080	1320

Table 4: Summary of selected threshold amplitude values

OUTPUT	**SELECTION**
A	900mV
B	500mV
C	150mV

2.2.2 Circuit Model and Simulation

The circuit was then modelled using PSpice, in which the results to the MATLAB signal shown in Figure 7 were inputted. The aim was to optimize the circuit design and the choice of electrical components. The results of the simulation are given in Figure 9.

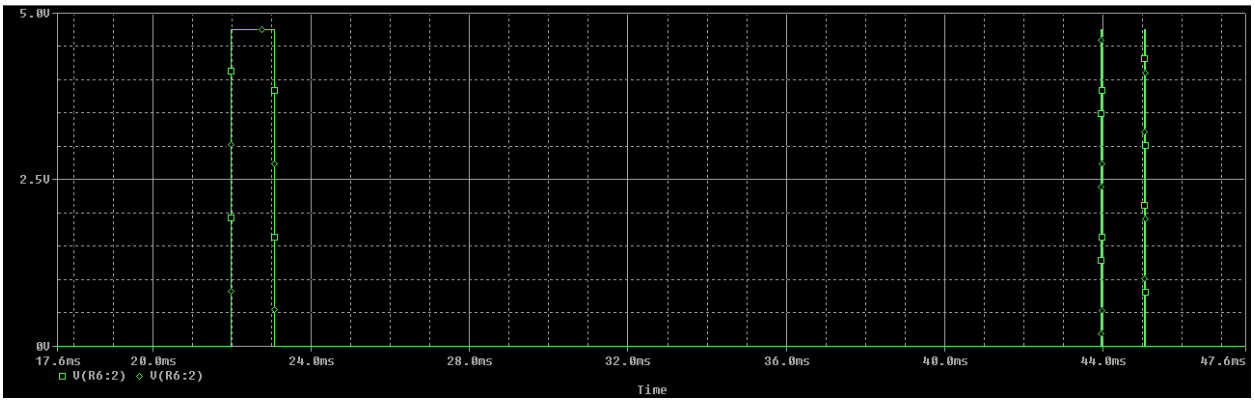

Figure 9: PSpice results of circuit modelling

The PSpice simulation revealed that the loss of frequency information causes a non-infinite rise time between amplitude levels. Consequently, the voltage comparisons are separated by up to tens of microseconds, resulting in unanticipated non-zero signals. To remove this effect, the output of the logic was filtered with a small capacitor as an AC shunt. Remodelling the circuit in PSpice, Figure 10 shows the result of the shunt capacitor filtering. This output was found to integrate with the ESCs to control motor speed.

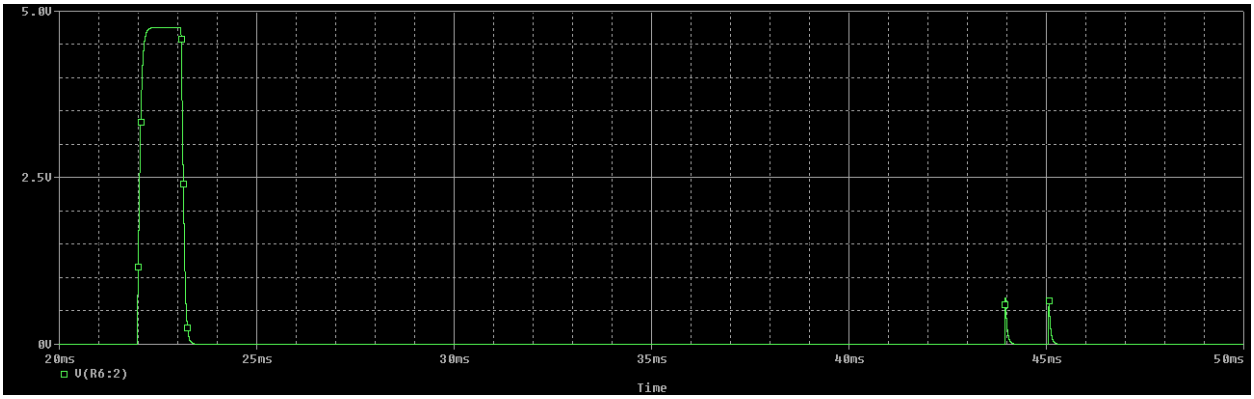

Figure 10: PSpice simulation results with capacitor shunt

2.2.3 Final Conversion Circuit

The final conversion circuit begins with amplification and is band pass filtered (BPF) before the logic blocks. The logic consists of analog comparisons between the signal and chosen thresholds, and AND gates. The output of the AND gates are the inputs for the ESCs. The block diagram for one audio channel of the circuit is given in Figure 11. The corresponding circuit diagram is given in Figure 12.

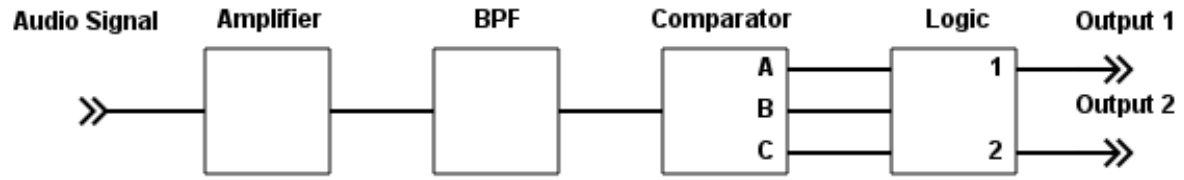

Figure 11: Block diagram of signal conversion for one audio channel

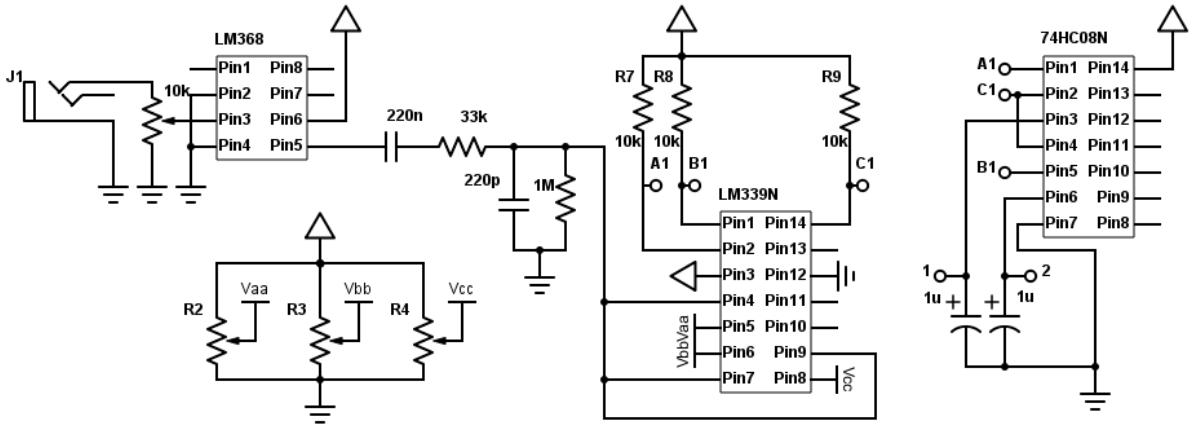

Figure 12: Circuit diagram of signal conversion for one audio channel

2.2.4 Power and Safety Circuit

To test and operate the drone safely, the power to the motors had to be cut on demand by interrupting the flow of current between the battery and the motors. This was achieved with a power MOSFET connected between the battery ground and circuit ground, and the gate was connected to a 5V switch. A secondary switch was added in the form of an RF receiver. The system was fail-secure, which means that the switch would become open if the button at the RF transmitter was released. This second switch acted as a remote kill switch and was originally designed to control a second MOSFET connected in series with the first. However, the initial tests revealed that the voltage drop across both transistors at typical load was over

3.5V, which was larger than the specified value of 1.8V in the datasheet [12]. The large voltage drop began causing unanticipated behaviour in the motors. The second MOSFET was consequently discarded. The circuit for the power control can be found in Figure 13.

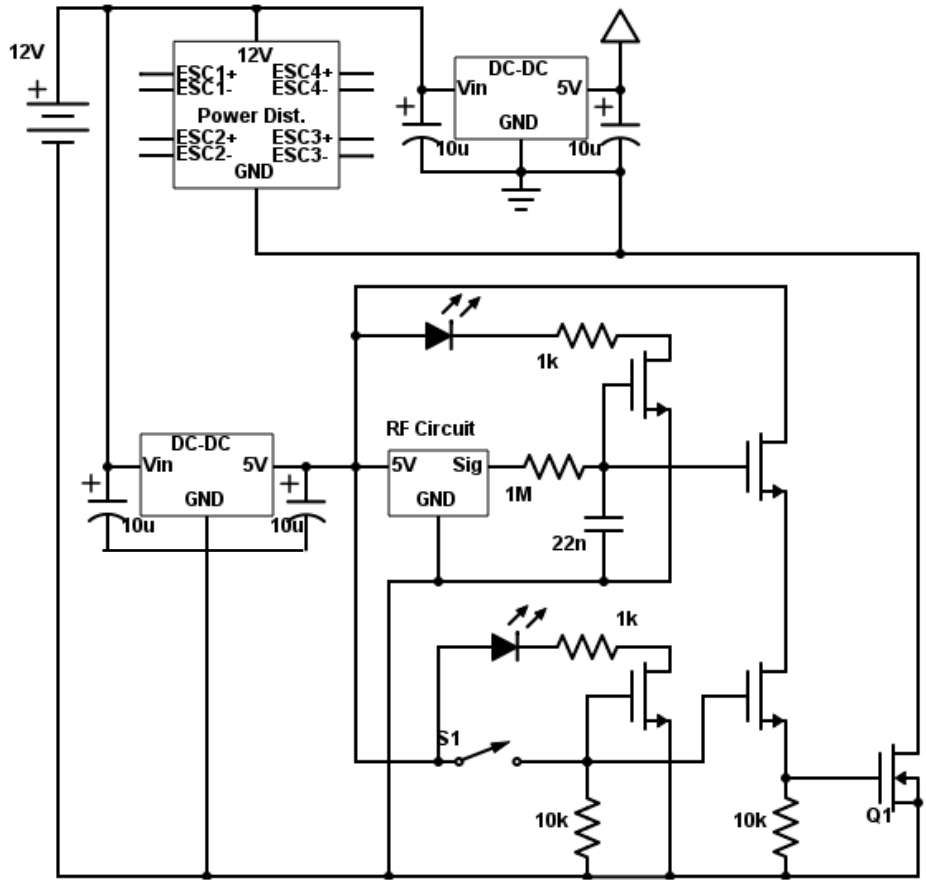

Figure 13: Power and safety circuit schematic

2.3 Software

2.3.1 Design Requirements

The primary goal of the system's software was to provide a user with a means of controlling the quadrotor from a remote location. To accomplish this goal, the team envisioned that the software should consist of two primary sub-systems:

1. An application on the onboard smartphone that is capable of outputting an amplitude-multiplexed PWM signal to the phone's audio port. This signal must be decipherable by the circuit as outlined in this report's previous section. The smartphone application must also contain the control logic required for the quadrotor to achieve stable flight.
2. An application on a remote computer that can wirelessly send commands to and receive telemetry from the quadrotor. This application should provide the operator with an intuitive means of controlling the quadrotor. This application should also allow the operator to engage an emergency-stop mechanism in the event of a hardware malfunction or in the event the quadrotor poses a danger to the surrounding populace.

Sections 2.3.2 through 2.3.5 of this report outline the design decisions made while further considering these subsystems. A complete list of the system requirements are provided in Table 9 and Table 10.

2.3.2 Software Back-End and Data Flow

The team designed the back-end of the software as a system of independent modules, or processes, that are interconnected by a series of communication channels. Each module is effectively a black box, with clearly defined inputs and outputs. A modular approach to the software design was appealing to the team as it facilitated the parallel development of individual modules. Furthermore, each module could be developed with little to no regard of the internal workings of the system's other modules. This, in term, facilitated the delegation of software development tasks.

The Android application contains three modules, and the remote application contains two. These modules and their interaction are outlined in Figure 14. The phone's IMU Reader module interfaces with the Android Sensor API to retrieve information from the phone's inertial sensors. This information is then sent to the remote application's Telemetry Display module and the Android application's Control Logic module. The Control Logic module, which encompasses the code for the quadrotor's flight controller, also receives directional commands from the remote application's User Command module. The module's outputs are the desired motor rpms to stabilize and guide the quadrotor. Finally, the Motor Driver module

receives these desired motor rpm commands and transforms them into amplitude-varying PWM signals that can be understood by the custom circuitry.

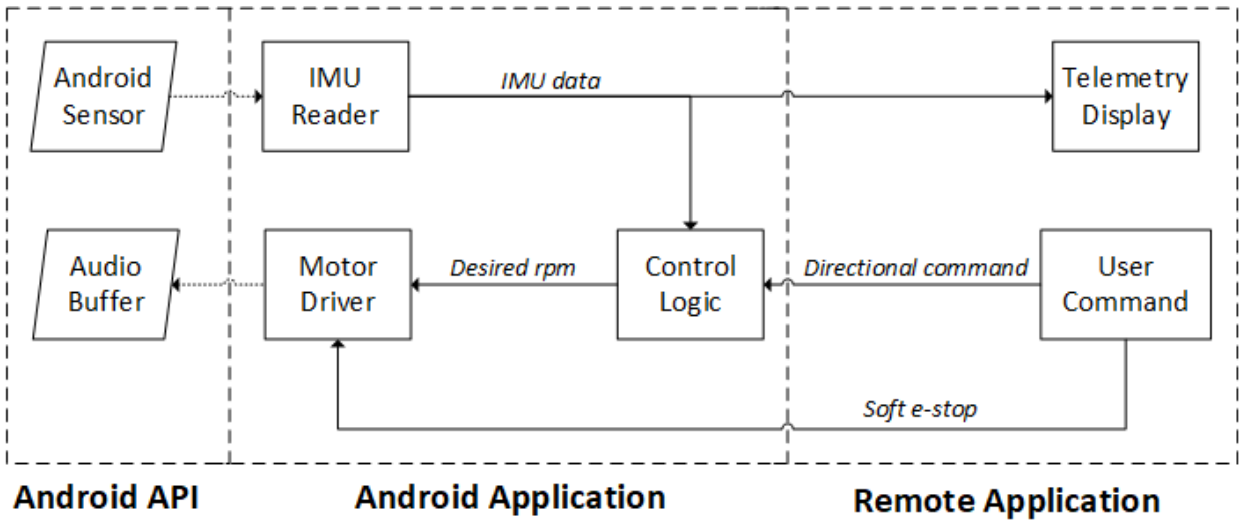

Figure 14: Software architecture. Parallelograms represent system APIs. Rectangles represent system modules. Dashed lines represent system boundaries. Dotted arrows represent system interfaces. Solid errors represent data flow.

2.3.3 Wireless communication

When considering the issue of teleoperation, the team identified four mechanisms of communicating with the quadrotor as summarized in Table 5. Firstly, the team considered making use of the smartphone's 3G or 4G cellular networking capabilities. A benefit of using a cellular network was that it provides long-range coverage, permitting the operator to control the quadrotor anywhere within range of a cellular tower. However, the disadvantage is that the cellular network is a paid service. The second option considered was to use the phone's 2.4GHz antenna. This solution had the benefits of no associated fees and lower communication latency, but at the cost of a shorter operating range. The 2.4GHz antenna was ultimately selected due to budgetary constraints.

Having chosen the medium for wireless communication, the media access control and physical layer protocol then had to be selected. IEEE 802.11 (Wi-Fi) and IEEE 802.15.1 (Bluetooth) were considered as both come standard in smartphones. Bluetooth has the advantage of having a lower communication latency than Wi-Fi. However, Wi-Fi has the advantage of having been tested over long ranges on the 2.4GHz band,

providing the operator with more freedom. Another advantage of Wi-Fi is that it is natively supported by many high-performance open-source messaging libraries and tools for distributed systems, such as ZeroMQ, DDS, and Robot Operating System (ROS) [13] [14] [15]. The use of such tools could provide a simple and elegant method of implementing the system's inter-module communication scheme. Due to these benefits, Wi-Fi was selected as the system's media access control and physical layer protocol.

Table 5: Comparison of different wireless communication technologies commonly available on cellular devices.

MEAN OF COMMUNICATION	RANGE	LATENCY (MS)	FEES
3G	Densely populated areas [16]	88.6 [16]	Yes
4G LTE	Densely populated areas [16]	47.3 [16]	Yes
WI-FI	Up to 20Km [17]	Up to 25 [1]	None
BLUETOOTH 4.0	Up to 100m [18]	7.5[2]	None

2.3.4 Android GUI

Although all navigation controls can be achieved through the remote application, an Android application was designed to confirm that the smartphone meets the required specifications, and to set up the telemetry options prior to flight. The Android application consists of four buttons to access the various features. These are explained further below, and an app flowchart is shown in Figure 15.

1. CREATE HOTSPOT. The button redirects the phone to the Android Wi-Fi hotspot settings. This allows the user to turn on the phone's mobile hotspot settings, allowing a remote computer to connect to the phone.
2. SENSOR STATUS VIEWER. The button launches another GUI that displays a list of sensors critical to the flight controller. It also indicates whether or not these sensors are present and functioning and warns the user if the necessary sensors are not available on the phone.

[1] Maximum Wi-Fi latency as experimentally determined by the team by sending "ping" requests between two devices connected to an ad-hoc network.
[2] The minimum connection latency of the HCI as specified in the Bluetooth Core 4.0 specification [26]. This was not experimentally determined by the team.

3. INITIALIZE MOTORS. The button starts the motor initialization process. The process includes on-screen prompts that the user should follow, and adjusts the audio port output signals as needed, according to the protocols described by the ESC manufacturer.
4. REMOTE CONTROL. The button launches the teleoperation back-end that allows remote flight control.

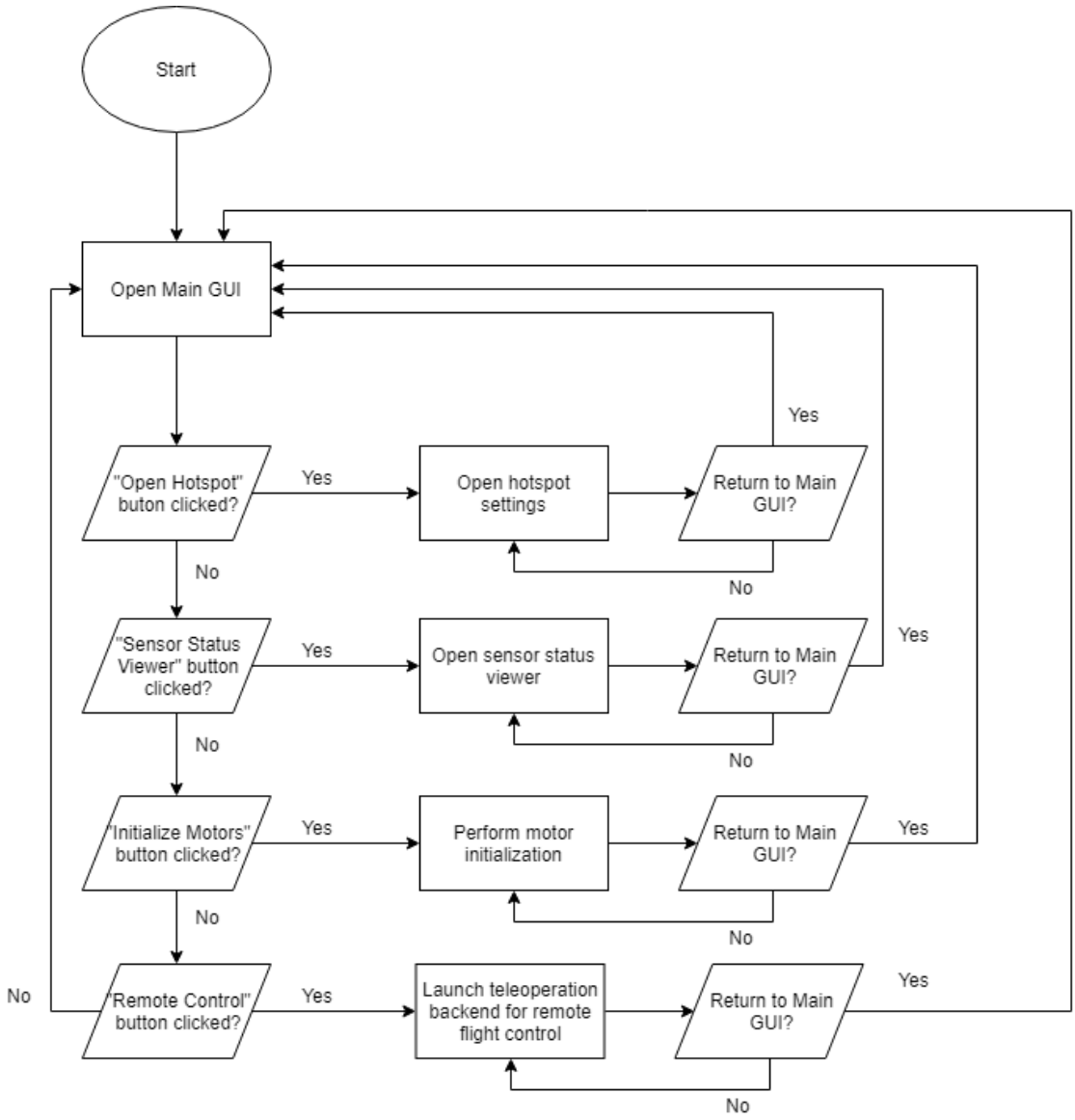

Figure 15: Flow chart of Android Application

2.3.5 Remote Application

The remote application was designed as a console-based program that starts both the User Command and Telemetry Display modules in separate processes. The Telemetry Display module prints a continuous stream of telemetry data to the console, while the User Command module asynchronously listens for directional commands from the user. The remote application keeps track of both the User Command and Telemetry Display process identifiers and terminates these processes as part of its shutdown procedure.

3 Implementation

The implementation section is divided into four parts. Sections 3.1, 3.2, and 3.3 focus on the quadrotor body, circuitry and software implementation of the UAV respectively. Section 3.4 establishes the timeline of this year-long project, and Section 3.5 recounts the budget and bill of material.

3.1 Quadrotor Body

As dictated by the requirements in Section 2.1, the final specifications of the components were chosen to be A2212 1000kV brushless motor[3], 30A brushless ESCs and 10x45 propellers[4]. This combination of the motor, ESC and propeller is rated to reach a total pull of 880 grams per motor, which satisfies the required 842 grams specification mentioned in Section 2.1 [19]. Thrust testing was performed to verify the correct functioning of parts before attaching them to the quadrotor frame and soldering them together. Using the apparatus shown in Figure 16, the maximum pull obtained was 860 grams.

[3] The two pairs of numbers (22 and 12) represent the diameter and length of the motor housing respectively in mm. The kV rating represents the rpm in thousand of revolutions per volt applied to the motor.
[4] A propeller is defined by two numbers. The first is the total propeller diameter and the second is the pitch of the propeller blades.

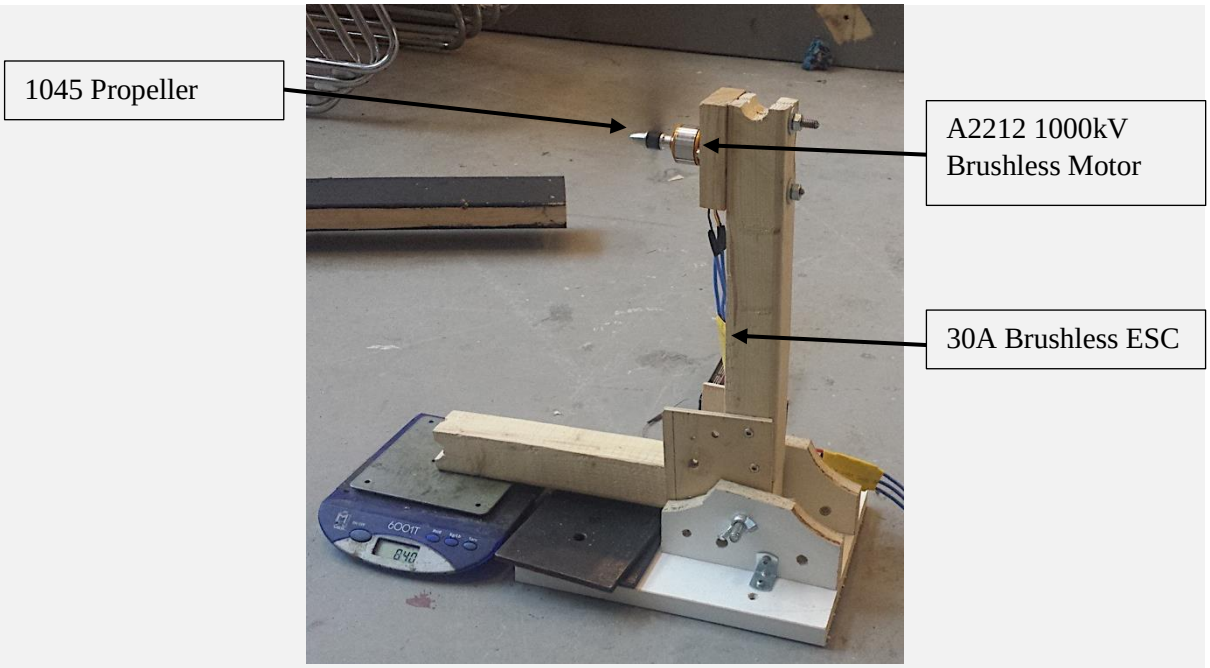

Figure 16: The thrust testing apparatus was borrowed from the Queen's aero design team. The balance shows the pull for one motor in grams.

The battery chosen was a 3-cell 5400mAh Lithium polymer battery. LiPo is the ideal choice since their high energy density and comparably lower weight results in longer flight time. The rated discharge capacity is 20C allowing for a potential maximum current draw of 108 Amps [20]. The chosen ESC and motor configuration is well within this specification, with an estimated maximum current draw of 12A.

The selected quadrotor frame allowed for a high degree of customization. The power distribution board is built into the lower frame which reduced the wiring complexity of the ESCs. Due to the larger surface area of the PCB, the board was placed on the upper frame, which gave quick access to the power switch. The battery was then placed in the centre between the two plates, and the phone was placed on the underside of the quadrotor. This configuration resulted in a lower centre of gravity, and the phone faces upwards to allow the rear camera to be used for recording capabilities.

SolidWorks was used to design a custom phone holder to be strapped under the quadrotor securely. The design was then printed using PLA material on a 3D printer. An Otterbox Galaxy S4 case provided a tighter fit within the phone holder and acted as a second layer of protection. One of the team members provided the smartphone and case. Finally, landing skids were ordered and screwed to the bottom frame.

They provide additional clearance between the ground and the smartphone and served to absorb shocks that would have compromised the integrity of the frame when performing our testing. The final design is seen in Figure 17.

Figure 17: Final assembled design of quadrotor

3.2 Electronics

During the design phase, several electronic components were selected to ensure real-world practicality of the design. Primarily, the LM368 integrated circuit chip and the Power FET component characteristics were carefully selected to meet specifications. The LM368 is a general-purpose audio amplifier with set gains of 20 and 100 that operates on a 0-5V power supply, and with a low power output. The default gain of 20 was chosen such that the output of the phone would have an operational range of 0-3V. The power MOSFET is a 100W transistor rated for 50A continuous DC and has a low drain to source resistance of 0.018Ω. Furthermore, the gate to source voltage required to switch the transistor is 5V. The 50A rating was needed as the MOSFET acts as a switch between the battery and the motors rated at 12A each, for a total of 48A. The remaining integrated circuits, including the 74HC08N AND gate and the

LM339N linear comparator, are common general-purpose components. They were found to function as expected and were incorporated into the final design.

After the completion and verification of the initial circuit was completed, the board was drawn using EAGLE, and a PCB was ordered from OSH Park. It should be noted that the version of the circuitry that the PCB is modelled on utilizes the two-power MOSFET solution that was later revised. The PCB was consequently altered upon arrival to accommodate the latest changes. Figure 18 shows a picture of the completed PCB. Appendix B shows the EAGLE diagram and schematic used.

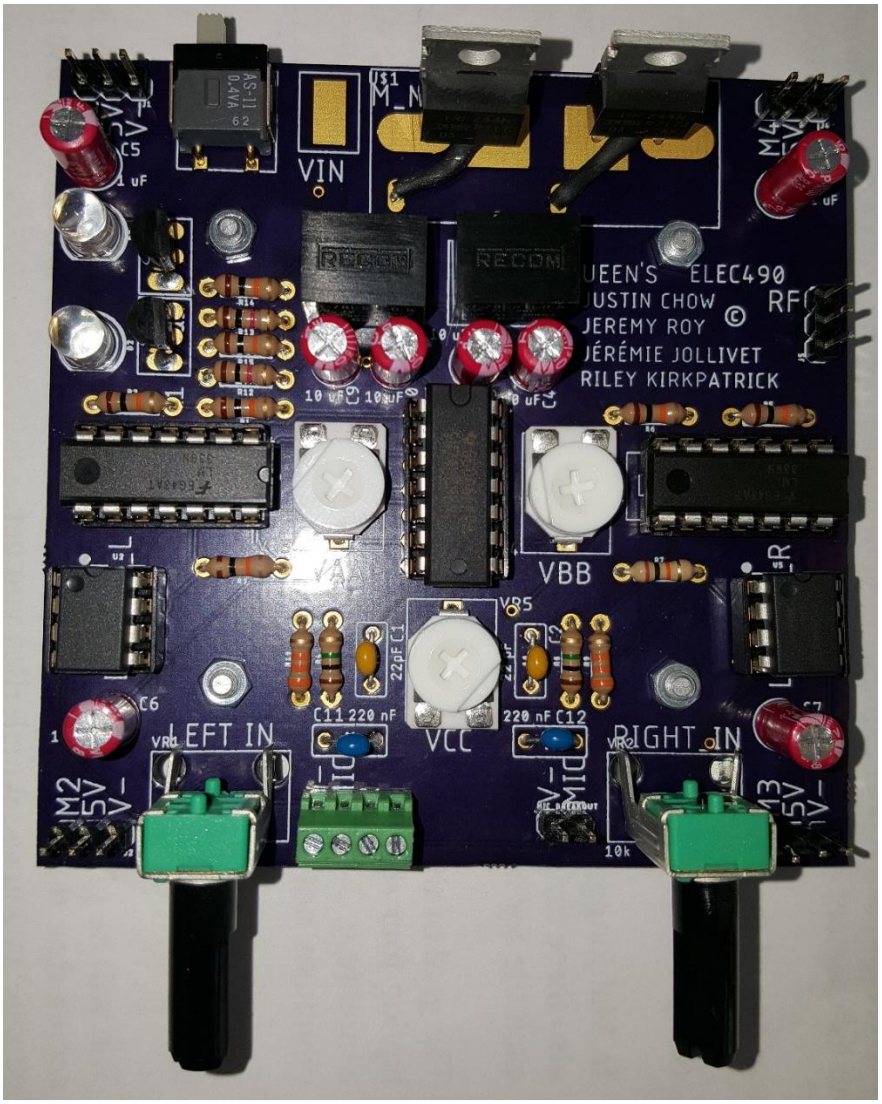

Figure 18: Picture of the completed PCB

3.3 Software

3.3.1 Software Framework

One of the first decisions made during the software implementation process consisted of choosing a set of messaging libraries and APIs that could support the software's modular framework outlined in Section 2.3.1 of this report. Two solutions were envisioned. Firstly, the team considered implementing inter-module messaging using the ZeroMQ messaging library, with messages formatted using Google Protocol Buffers. This solution had the benefit of being highly customizable, as ZeroMQ supports many messaging paradigms such as publisher/subscriber, push/pull, and client/server [13]. In addition, both ZeroMQ and Protocol Buffers are supported across many programming languages and mobile platforms [13] [21]. This is beneficial as it increases the portability of the software. However, very few open-source robotic modules that use this approach exist, therefore eliminating the possibility of re-using existing modules.

Secondly, the team considered implemented inter-module messaging using Robot Operating System (ROS). ROS is an open-source meta-operating system that provides distributed processes, or "nodes", a means of communicating between themselves [15]. ROS has the benefit of having a large open-source community that develops and maintains a considerable set of modules, such as teleoperation modules and localization modules, that are freely available for integration into robotics projects. However, ROS only natively supports two languages, C++ and python, although some members of the community have created experimental ports for JAVA and Lua. Furthermore, ROS is only available on a select number of platforms, including Ubuntu and Android.

Ultimately, the benefits brought about by the ROS community outweighed the benefits brought about by the flexibility of ZMQ + Protobuf, and the team pursued ROS.

3.3.2 Overview of ROS

A brief overview of ROS will now be provided, as its terminologies will frequently be used in subsequent sections of this report. A core component of ROS is the ROS Master. The ROS Master is a program that enables individual nodes to locate one another and to set up communication channels between themselves. The network can only contain one ROS Master. ROS provides two main mediums of communication; “topics” and “services”. Topics are named unidirectional buses over which nodes can exchange information. Topics use a many-to-many model of message passing, where an arbitrary number of nodes can either publish to or be subscribed to a topic. Services, on the other hand, offer nodes a means of bidirectional communication via a single channel of communication. Services are normally presented to programmers as a remote procedure call, where a client node can request for an action to be completed by a node acting as the service provider.

3.3.3 Development Environment

The choice of the development environment was highly dependent on the team’s choice to use ROS. ROS is released as a set of distributions each tied to a separate release of Ubuntu. When the project’s software development started in October 2017, the recommended distribution of ROS was called “ROS Kinetic”, which was tied to Ubuntu 16.04. A virtual machine accessed through VMWare Workstation 14 Player was used to provide all team members with access to a Ubuntu 16.04 machine. The software packages installed on the virtual machine include “ros-kinetic-desktop-full” and “ros-kinetic-rosjava”. ROS Android was compiled from source. To support Android application development, Android Studio and the Android SDK were also installed. Software version control was performed using Git and GitHub. All developed software is open source and freely available online [22] [23] [24] [25].

3.3.4 IMU Reader

The IMU Reader module was implemented as a ROS Android node that periodically publishes a `sensor_msgs/Imu` message to a ROS topic called `phone_imu/`. The sensor data in each message consists

of the smartphone's orientation formatted as a unit quaternion, the phone's angular velocities around its three orthogonal axes, and the phone's linear acceleration along its axes.

3.3.5 Teleoperation

The software's teleoperation bundle consists of the Telemetry Display and User Command modules as discussed in Section 2.3.1 of this report and shown in Figure 14. The Telemetry Display module, which prints an incoming stream of `sensor_msgs/Imu` messages to the remote computer's console, was implemented using the `rostopic echo` command line tool. The "User Command" module was implemented using the open-source `hector_quadrotor_teleop` and `joy` nodes, which interfaces with the Linux joystick API to provide quadrotor teleoperation commands using a gamepad (see Figure 19 for a graphical depiction). Gamepads that are natively supported by the `hector_quadrotor_teleop` node include the Sony DualShock3, Logitech Gamepad, and Xbox controller. Due to budgetary constraints, the team was unable to purchase one of these natively supported gamepads. Rather, the team borrowed a Sony DualShock4 controllers from a team member and wrote a custom launch script that mapped the Sony DualShock4 inputs to the `hector_quadrotor_teleop` node's expected ROS parameters. The `hector_quadrotor_teleop` node outputs user directional commands to the `command/thrust`, `command/yawrate`, and `command/attitude_adjusted` topics.

3.3.6 Motor Driver

Motor control is accomplished by outputting signals through the audio port. This is done by writing audio samples to an audio buffer. The buffer is then written to an `AudioTrack` object which allows streaming of the audio buffer to the Android audio sink for playback. A stereo audio buffer is used, as different signals are sent to the left and right audio channels. As a result, audio samples are written to the buffer in alternating order. The left samples are written first, followed by the right samples. The output signal is set at 50Hz to match PWM specifications. The first and second motors on each channel are associated with an output amplitude of 2% and 8% of the maximum respectively. These amplitudes were chosen to match the circuit's design in Section 2.2.1.

The Motor Driver module was implemented in two parts. First, the `MotorsAudio` class contains an algorithm to generate the appropriate audio samples for the desired motor outputs. The class provides a layer of abstraction between the audio buffer and the ROS node logic and accepts full-scale motor power commands for each motor from the upper layer.

To generate the appropriate audio samples at the appropriate time, the software keeps track of the phase of the signal that corresponds to the following audio sample. The first half of each period of the signal on each channel is dedicated to the channel's first motor, while the second half is dedicated to the channel's second motor. Appropriate audio samples for the appropriate channel were generated by keeping track of the number of samples that are written to the current audio buffer. An even numbered sample denotes a left channel sample, while an odd numbered sample denotes a right channel sample. With the above information, the software first identifies the channel and motor to be written to next. It then checks the desired duty cycle for the specified motor. If the current phase of the signal is within the desired duty cycle, the audio sample sets the appropriate amplitude (2% or 8%, depending on the motor). If the current phase of the signal does not fall within the desired duty cycle, the audio sample is set to 0.

The second part to the Motor Driver module is a ROS node written to interface a `MotorsAudio` object with the rest of the system. The node is subscribed to a topic called `motor_ctrl` and expects to receive a message of type `simone_msgs/MotorCTRL`. This custom message type was created by the team and allows a publishing node to specify the percent full-scale thrust of each motor individually. The node also provides an `EnableMotors` service that acts as the system's soft-estop.

3.3.7 Control Logic

The quadrotor's flight control logic was inspired by the flight controller design outlined by Dr. Gareth Owenson in his blog post "Build your own Quadcopter Flight Controller" [26]. The flight controller consists of a cascade controller with three inner-loop PID controllers and two outer-loop PID controllers. The inner-loop controllers regulate the angular rates of the quadrotor (RPY), whereas the outer-loop controllers adjust

the absolute roll and pitch angles of the quadrotor. The quadrotor operator directly commands the quadrotor's yaw rate, and thus it was not necessary for the flight controller to include an outer-loop controller for absolute yaw angle. The gains of the PID controller were tuned using a trial and error method.

Programmatically, the flight controller was implemented as a ROS Android node that is subscribed to the `phone_imu/`, `command/thrust`, `command/yawrate`, and `command/attitude_adjusted` topics. The flight controller node publishes the custom `simone_msgs/MotorCTRL` message to the `motor_ctrl` topic. The flight controller also offers an `update_pid` service that allows the user on a remote computer to update the gains of the PID controllers. The PID controllers were implemented using the open-source `MiniPID-Java` library written by GitHub user tekdemo and licensed under the GPL-3.0 software license.

3.3.8 Quadrotor Leveler

When testing the quadrotor, the team noticed that even when the quadrotor was on a level surface, the orientation returned suggested that the quadrotor was slightly pitched or rolled. In theory, this would cause the control logic to try to level to an angle that wasn't normal to the surface of the earth. Furthermore, it was observed that the estimated orientation of the quadrotor as returned by the gyroscope had a slight drift. Therefore, the quadrotor had an invalid and worsening sense of orientation. To solve this, a Quadrotor Leveler node was introduced as an intermediary node between the `hector_quadrotor_teleop` node and the Control Logic node (see Figure 19). The Quadrotor Leveler node provides a `recalibrate_attitude` service to the ROS system. When this service is invoked, the node assumes that the quadrotor is on level ground and takes an average of 200 orientation readings from the IMU topic. The error between the IMU's estimated orientation and the quadrotor's assumed level orientation is computed and then applied as a correction factor to the user-specified attitude command being output from the `hector_quadrotor_teleop` node. Finally, the Quadrotor Leveler node outputs the resulting corrected attitude commands to a new topic to which the Control Logic node is subscribed.

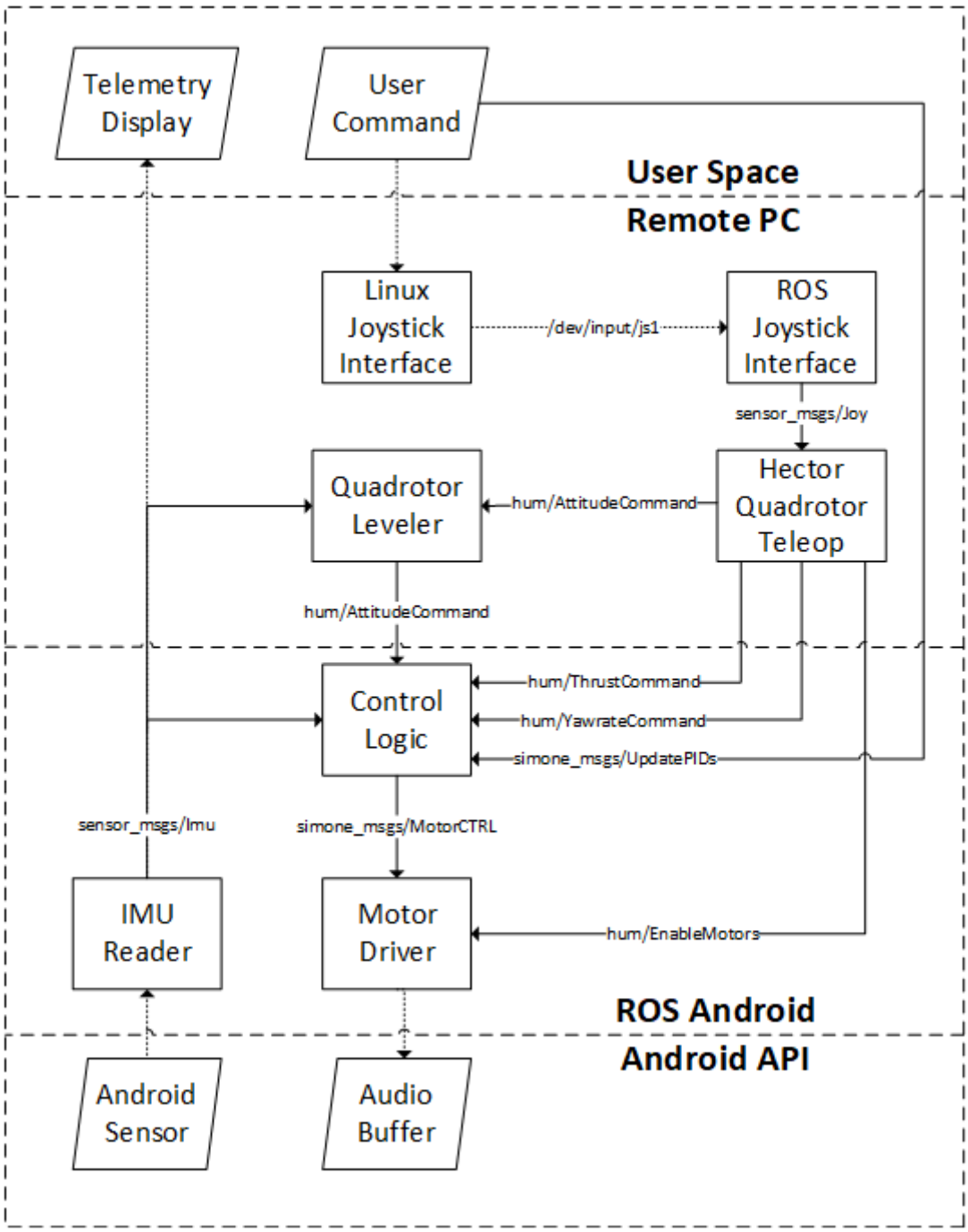

Figure 19: Final software layout. "hum" is an abbreviation for "hector_uav_msgs". Dashed lines represent system boundaries. Dotted arrows represent system interfaces. Solid errors represent ROS topics. ROS topics are annotated with the type of message being passed over them.

3.3.9 Wireless Security

Wireless security is an important concern when operating any mechatronic system over a wireless network. A compromised system could result in the loss of equipment, damages to property, or injury. A common method of enforcing wireless security is through the encryption of communications. Although the

application developed by the team does not impose a wireless encryption policy on the system's Wi-Fi network, the user is highly encouraged to use a secure protocol such as WPA2 PSK.

3.3.10 Android GUI

The Android GUI was developed in Java using Android Studio and the Android Framework. It was implemented according to the design outlined in Figure 15. Screen captures of the GUI are shown in Figure 20, Figure 21, and Figure 22.

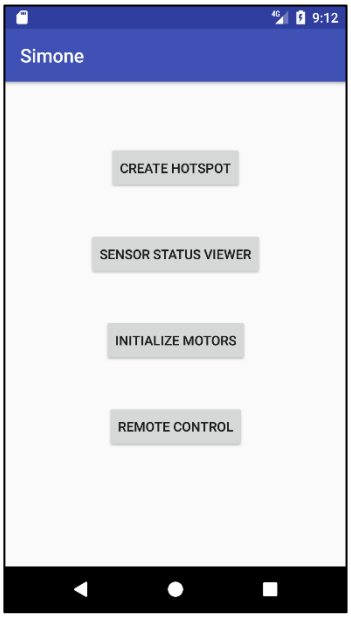

Figure 20: The main GUI

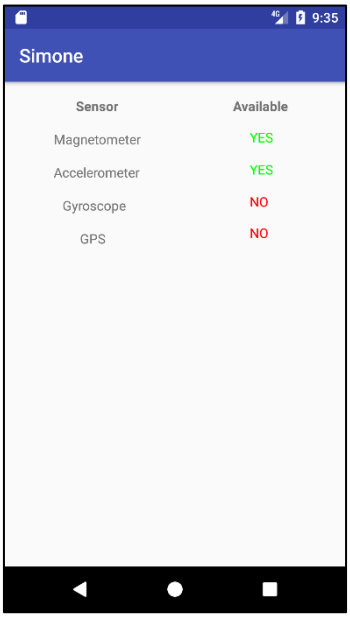

Figure 21: The sensor status viewer

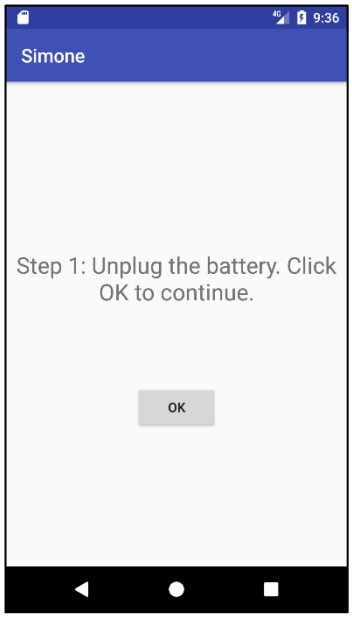

Figure 22: The first step of the motor initialization process

3.4 Timeline

Table 6 shows the planned and actual timelines for the project. The "PLANNED" column shows the proposed completion dates as specified in the project's blueprint document. The "ACTUAL" column shows the true completion dates. Most of the milestones were met on schedule. As the design process unfolded, however, certain design changes called for additional unexpected activities. These included the app design on the Android phone, the safety circuit design, the ROS environment setup and the ordering of new components. The activities were added to the table and italicized, and their corresponding "PLANNED" column was left blank. They include the design of the Android app, the safety circuit design, the ROS environment setup and the ordering of additional components.

Table 6: Timeline

NO.	MILESTONE	RESPONSIBLE MEMBER(S)	PLANNED	ACTUAL
1	Initial Proof of Concepts (Software)	Jeremy Roy	Week 5	Week 6
2	Initial Proof of Concepts (Hardware)	Riley Kirkpatrick	Week 5	Week 5
3	Major Materials Ordered	Jérémie Jollivet	Week 6	Week 5
4	Preliminary Design Package (Circuit Models)	Riley Kirkpatrick	Week 8	Week 8
5	Preliminary Design Package (Software Overview)	Jeremy Roy	Week 9	Week 9
6	Preliminary Design Package (Hardware Package)	Jérémie Jollivet	Week 10	Week 11
7	Minor Components Ordered	Jérémie Jollivet	Week 10	Week 10
8	*Safety circuit Design*	*Riley Kirkpatrick*	-	*Week 12*
9	*ROS environment setup*	*Jeremy Roy*	-	*Week 12*
10	Preliminary Design Package (PCB Layout)	Justin Chow	Week 11	Week 11
11	Construct Pre-Version A Prototype	Riley Kirkpatrick	Week 11	Week 12
12	Order PCB	Justin Chow	Week 12	Week 13
13	*Design of the Android app*	*Justin Chow*	-	*Week 14*
14	*Design of 3D printed smartphone case*	*Jérémie Jollivet*	-	*Week 16*
15	*Testing platform*	*Jérémie Jollivet, Riley Kirkpatrick*	-	*Week 17*

NO.	MILESTONE	RESPONSIBLE MEMBER(S)	PLANNED	ACTUAL
16	Construct V0 Prototype	Jeremy Roy	Week 17	Week 17
17	Open House Design Package	Riley Kirkpatrick	Week 18	Week 18
18	Final Deliverable – V1 Prototype	All members	Week 23	Week 19
19	Final project report, presentation	All members	Week 24	Week 24

The team held bi-weekly meetings with the course TA, Zack Babcock, and the faculty supervisor, Dr. Carlos Saavedra, for the duration of the first 18 weeks. These meetings were used to evaluate the team's progress and compare it to the schedule proposed in Week 2. Additionally, the team met weekly for two hours on Thursday afternoons to collaborate on the project tasks and identify the work for the coming week.

The team encountered delays during the implementation phase of the project. A major hardware delay was due to the late ordering of the PCB. Although the design was ready to be sent for manufacturing in Week 12, the team was seeking guidance with regards to the size of the traces and the high current through the safety circuit. The PCB design files were sent for manufacturing at the end of December, but the order was not processed until the second week of January. Another source of delay came from the software development process. The deadlines outlined in the blueprint document were primarily hardware-based, which resulted in prioritizing the circuit over the software. In retrospect, the team should have focused more on the software development timeline. The delays resulted in the flight controller software testing being pushed back to Week 17. As such, the PID gains were not tested correctly before the test flights.

3.5 Budget

The team spent a total of $407.42 over the course of this project, which is $7.42 more than the allocated amount. The expenses are shown in Table 7. This amount is also greater than the original amount of $355 proposed in the blueprint document and was primarily due to two reasons.

First, an additional 30 dollars were spent on electrical components. This was caused by a faulty 433Mhz RF transmitter and receiver obtained from Amazon. A new part was then ordered in the second Digi-Key order, as well as solder boards and new Power FETs to perform additional prototyping. The second unexpected expense came from an ESC burning out during testing. The part was temporarily switched with an ESC belonging to the Queen's Aero Design Team, and a replacement part was purchased from Leading Edge Hobbies a week later for $22.59.

Table 7: Bill of materials

ORDER No.	QTY	DESCRIPTION	SUPPLIER	UNIT PRICE	SHIPPING /TAX	TOTAL
1	4	XCSOURCE® A2212 13T 1000KV Brushless Motor 30A ESC 1045 Propeller for FPV DJI F450 550 quadrotor RC133	Amazon	$ 25.99	$ 8.32	$ 112.28
1	1	10pcs Solder Finished Prototype 5x7cm	Amazon	$ 7.99	$ 0.64	$ 8.63
1	1	433Mhz RF Transmitter With Receiver Kit	Amazon	$ 4.99	$ 0.40	$ 5.39
1	1	Venom 20C 3S 2100mAh LiPO Battery	Amazon	$ 26.59	$ 2.13	$ 28.72
2	1	F450 MultiCopter quadrotor Kit Frame	Amazon	$ 23.34	$ 1.87	$ 25.21
2	1	5 pcs 20cm Tie Down Strap	Amazon	$ 6.99	$ 0.56	$ 7.55
3	1	Digi-Key Order #1	Digi-Key	$ 49.35	$ 12.59	$ 61.94
4	1	Printed Circuit Board (3)	OSH Park	$ 62.00	$ 25.08	$ 87.08
5	1	DJI F450 Landing skids	Amazon	$ 9.24	$ 1.55	$ 10.79

ORDER No.	QTY	DESCRIPTION	SUPPLIER	UNIT PRICE	SHIPPING /TAX	TOTAL
6	1	Digi-Key Order #2	Digi-Key	$ 27.18	$ 10.06	$ 37.24
7	1	Replacement 20A ESC	Leading Edge	$ 19.99	$ 2.60	$ 22.59
TOTAL						$ 407.42

4 Testing and Evaluation

The section is divided into three parts. Sections 4.1 focuses on the hardware testing, including the circuitry delay. Section 4.2 focuses on the testing of nodes and latency in software. Finally, Section 4.3 focuses on the system as a whole.

4.1 Hardware Testing

The physical properties of the system were evaluated to analyze the performance of the circuit in relation to the ESCs. The parameters most important to the functionality of the drone are the circuit delay time, and the amount of lift needed to achieve flight.

4.1.1 Circuit Delay Testing

The input and output signals were connected to a TDS2000C oscilloscope to measure the circuit delay time. This value is equal to the difference in time between which the output transitions from zero to its maximum value. As both the input and output signals are captured at the same time, and the frequency of the signal is far less than the oscilloscope's maximum operating frequency of 50MHz, the oscilloscope data can be considered to be valid [27].

The circuitry delay was calculated using the oscilloscope. From Figure 23, the delay was found to be $40\mu s \pm 2\mu s$. A non-zero value was anticipated due to the passive components that introduced a delay in the circuitry. A delay of a few microseconds is typical in digital processing and is only one order of magnitude faster than this analog solution. In the context of the refresh rate required to fly a quadrotor, the circuit delay is adequate.

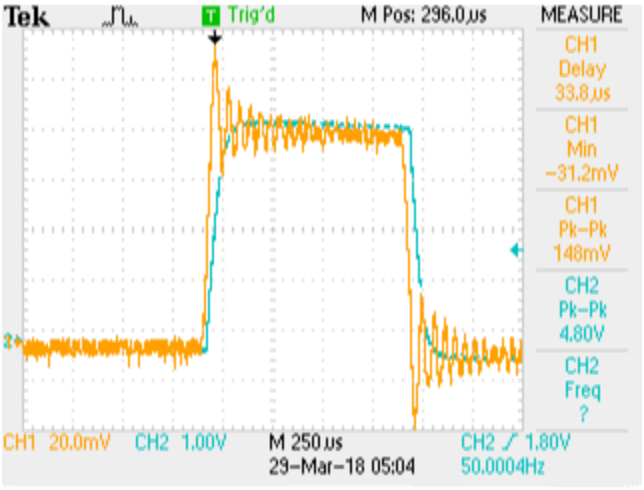

Figure 23: Capture of input (yellow) and output (blue) signals illustrating circuit delay

4.1.2 Motor Thrust Testing

The apparatus shown in Figure 16 was used to test the force output of the motor for a given audio input. Figure 24 shows the results obtained from this test. The correlation between the output duty cycle and the thrust output of the motor was found to be near linear. This test also revealed that the maximum thrust of the motor is in the approximate range required by design. The results of this test, however, have an unknown margin of error. As such, the plot only served to illustrate the behaviour of the motors and to provide an estimate of their thrust capabilities.

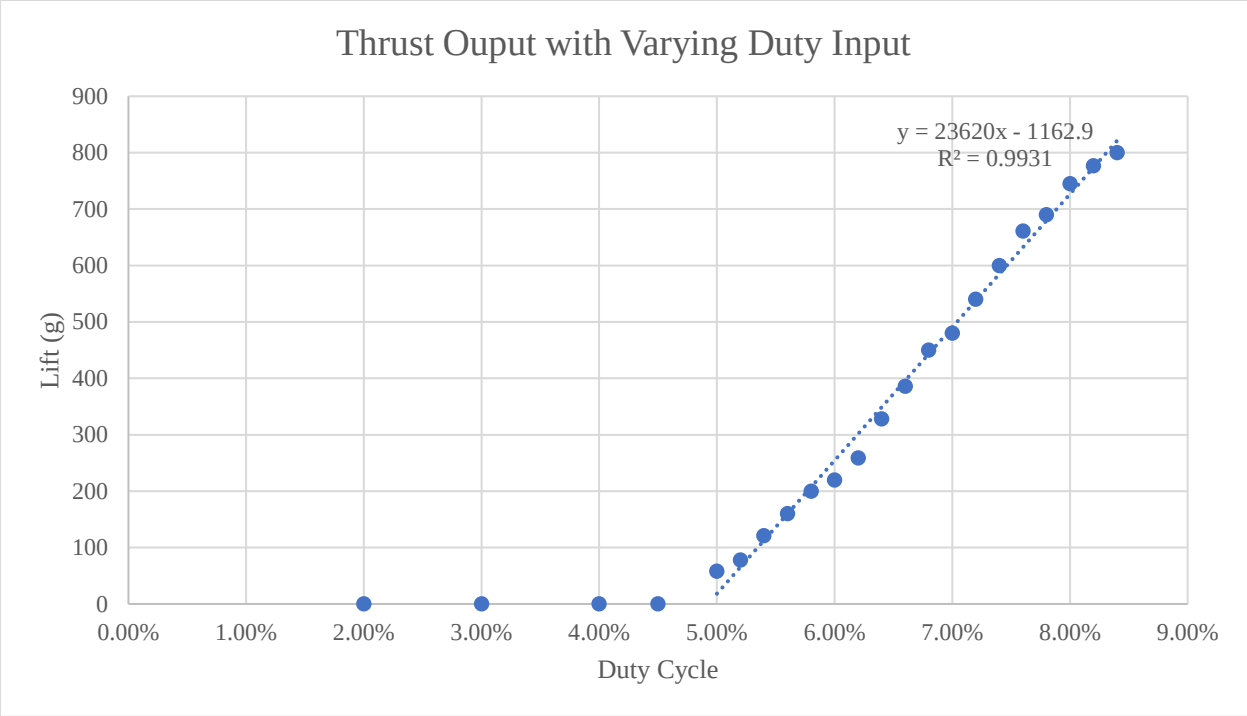

Figure 24: Plot of Thrust output and Duty input

4.2 Software Testing

Software testing was segmented into two groups; Android GUI testing and ROS back-end testing. The bulk of the project's software consisted of the ROS back-end, and therefore software testing focused on ensuring that the ROS system behaved as intended. Subsections 4.2.1, 4.2.2, and 4.2.3 outline the various tests performed on the system's ROS back-end. Android GUI testing mostly consisted of clicking on the GUI's various buttons and making sure the user was brought to the expected page.

The console tools `rostopic` and `rosbag` were extensively used during back-end software testing. `rostopic` is a tool that can, amongst other things, publish messages to topics and view messages being published to topics. `rosbag` is a tool that can record messages being passed over topics. These messages, which are stored in a "bag" file, can then be played back to the system at the same rate they were received.

4.2.1 Unit Testing

Extensive unit testing was performed on each ROS node both during development and before they were integrated to the rest of the system. Therefore, the flow of unit testing followed the order in which the

ROS nodes were developed. A detailed description of a subset of the unit tests that were performed are available in Table 12 in Appendix C – Testing procedure. The Sensors Viewer Page of the Android GUI was also tested by confirming that the indicated sensor availabilities matched the known hardware configuration of the phone.

4.2.2 Integration Testing

Once the ROS nodes passed their respective unit tests, they were flagged as ready for integration with the rest of the system. System integration was performed by combining flagged nodes one by one such that their inter-connections conformed to the system architecture as outlined in Figure 19. A series of black box tests were performed on the system following the addition of each new node. Each black box test consisted of a combination of the integrated nodes' unit tests. For example, when combining the IMU Reader and Control Logic nodes, three tests were performed. These consisted of publishing stationary user commands (zero thrust, yaw rate, and level attitude), rotating the phone along its axes of rotation, and finally examining the `simone_msgs/MotorCTRL` messages being output by the flight controller. This test was considered successful if the commanded motors would attempt to keep the phone level and counteract any yaw movement.

4.2.3 Latency Testing

Latency testing was performed to satisfy software requirements 3.1 and 7.1 of Table 9 and Table 10. The software requirement 3.1 stated that the delay between the time the Android application receives a directional command, and the time the signal is output at the audio port, should be no more than 150ms. To verify this requirement, both the Android application's processing latency and the phone's audio latency were measured. The processing latency of the Android application was measured by inserting two time probes. The first was placed at the entry point of directional commands in the Control Logic node, and the second at the exit point of the motor duty commands to the audio buffer in the Motor Driver node. These probes were set to measure the system's latency only when a "high thrust" command was received by the Android application. A maximum delay of 39ms out of five trials was measured between these probes.

The phone's audio latency was then measured using a third party application called "Zoiper Audio Latency Benchmark". After running the application five times, the worst-case round-trip audio latency from the phone's speaker to microphone measured on the Samsung Galaxy S4 was 246ms. Because the application measures the round trip latency, the output latency of the S4 is half of that number, or 123ms. Combining the Android application's processing latency and the phone's audio latency, the total worst-case phone latency was 162ms, which is 12ms over specification.

Software requirement 7.1 stated that the delay between the time the remote application receives a user command and the time the command is received by the android application must be no greater than 100ms. To verify this requirement, both the processing latency of the remote application and the Wi-Fi network latency were measured. The processing latency was measured by inserting the time probes at both the entry and exit point points of the attitude commands. These are located in the ROS Joystick Interface and Quadrotor Leveler nodes respectively. The probes were set to measure the system's time only when a "high roll" command was received from the user. A maximum delay of 11ms was measured between these probes over the course of five trials. The network's Wi-Fi latency was measured by "pinging" the smartphone from the remote computer. The worst case round-trip network delay was measured to be 50ms. Assuming the worst case remote-to-phone network delay is half that of the round-trip delay, the worst-case user-to-phone delay was found to be 36ms, which is 64ms below specification.

4.3 System Testing

Once the software and hardware were tested separately and the team was confident with their performance, the software and hardware were combined for full system testing. First, the Android GUI's motor initialization functionality was tested to ensure that the motors could be initialized properly by the software. Next, each motor was individually tested to ensure that the software could control each motor independently. This was done by running the Android application with the flight controller disabled. The quadrotor was then tethered to the testing apparatus as shown in Figure 17. The `rostopic` command line tool was then used to publish `simone_msgs/MotorCTRL` messages to activate each motor, one at a time.

After confirming that each motor could be controlled independently, the motors were tested by controlling multiple motors at the same time. This was done following the same procedure outlined for individual motor testing. However, instead of publishing `simone_msgs/MotorCTRL` messages to activate each motor independently, `simone_msgs/MotorCTRL` messages were published to active multiple motors. All combinations of motors were tested including testing two motors at a time, three motors at a time, and ultimately, four motors at a time.

Finally, once it was proven that the software could successfully control all the motors, the flight controller was tested. Flight controller testing proceeded in two phases. The first phase was performed indoors, with the quadrotor again tethered to the testing apparatus. For this test phase, there was slightly more slack in the tether to allow the quadrotor to lift off the ground. This phase of testing involved tuning of the PID gains in an attempt to have the quadrotor stably lift off the ground. This proved to be difficult as the tether appeared to pull on the quadrotor before it had a chance to stabilize. The team subsequently proceeded to attempt outdoor testing in an open field. Unfortunately, due to time constraints, the team was unable to achieve that goal and system requirement 4.1 from Table 9 was not met.

5 Compliance and Specifications

5.1 Hardware Specifications

As the design and implementation phases developed, several requirements were found to be outside the scope of the original project, and other unplanned requirements were discovered. From Table 8, hardware specification 3 was changed from "battery power" to "flight duration," as the latter is a more appropriate name. Hardware specifications 6 and 7 were added after the original blueprint was delivered, as the result of feedback from the team's supervisor, Dr. Saavedra. The hardware specification 2 was considered redundant to software requirements 3.1 and 7.1. Thus, this specification was not considered during compliance analysis and was marked as non-applicable.

The circuitry delay, as described in the hardware testing in Section 4.1, was found to be near instantaneous, thereby meeting hardware specification 1.

Since the quadrotor did not achieve stable flight, hardware specifications 3 and 4 could not be directly verified and were therefore marked as non-applicable. A measure of total flight duration was not directly obtained, as 15 minutes continuous flight were not achieved. However, the prototype ran all day during the demonstration, suggesting that 15 minutes of flight could be obtained with the selected battery.

Due to the destructive nature involved with verifying hardware specification 5, an actual test was not conducted. However, a quadrotor of similar weight with the same landing gear was tested by the Queen's Aero Design Team, and the UAV survived a fall of 2.5m with only one landing skid needing replacement. The integrity of the frame and the electronics were not impacted, suggesting that the hardware specification 5 can be met.

The weight of the final quadrotor was measured to be 1150 grams, which is 45 grams higher than expected. The motors were capable of providing more thrust than expected, with an actual pull of 880 grams per motor. A payload fraction of 46% was therefore achieved, meeting hardware specifications 6 and 7.

Table 8: Hardware requirements

SPEC. NO.	SPECIFICATION	TARGET VALUE	TOLERANCE	ACHIEVED VALUE
1	Circuitry Delay	< 1ms	0 %	$40\mu s \pm 2\mu s$
2	Reaction Delay	< 100ms	0 %	N/A
3	Flight Duration	> 15min	0 %	N/A
4	Steady State time	< 2s	0 %	N/A
5	Acceptable Fall Height	3m	± 33 %	< 2.5m
6	Weight *(added)*	1105g	10%	1150
7	Payload fraction *(added)*	45%	10%	46%

5.2 Software Requirements

With regards to the Android application requirements, 93% of the requirements were met. These are defined in Table 9. As the design of the quadrotor progressed, it was discovered that smartphones do not typically contain an altimeter, and instead measure altitude using the GPS module. As a result, software requirement 1.1 from Table 9 is met except for the altitude sensor data. Software requirements 1.2 to 1.7 were met, as described in the Design and Implementation sections.

The interface and performance requirements were all met, except for a change made in software requirement 2.2. The team did not display the sensor data within the Android application, but rather on the desktop application. This change was made because the phone is mounted on the quadrotor such that the screen is not visible during operation.

The performance requirement of the Android application, software requirement 3.1, was only partially met as described in Section 4.2.3 of this report. The worst-case latency was found to be 162ms, which is 12ms above the specification. However, the average latency of the five tests conducted was 122ms, which is within the required specification.

Although the quadrotor was proven to be able to move with six degrees of motion, the movement could not be controlled reliably, and stable flight could not be achieved. Therefore, software specification 4.2 was achieved while software specification 4.1 was not.

Table 9: Software requirements of onboard Android Application

1	**Functional requirements - Android application**	**TARGET MET?**
1.1	Detect which navigational sensors are available on phone. Navigational sensors include an accelerometer, gyroscope, magnetometer, altimeter, and GPS.	Yes (minus the GPS and altimeter)
1.2	Alert the user if the phone does not contain the necessary navigational sensors or if a navigational sensor is malfunctioning.	Yes
1.3	Send commands to the phone's output ports to control the vehicle's actuators.	Yes
1.4	Connect over a Wi-Fi network to a remote computer.	Yes
1.5	Send sensor data to a remote computer.	Yes
1.6	Receive navigational commands from a remote computer.	Yes
1.7	Have a soft e-stop mechanism that can be engaged in the event of an error.	Yes
2	**Interface requirements - Android application**	
2.1	The user must be able to test the vehicle's actuators in a safe manner.	Yes
2.2	The user must be able to view data from all navigational sensors on a page. *Changed to: The user must be able to view data on the remote desktop*	Yes
2.3	The user must be able to set the phone into wireless access-point mode (WPA2 PSK) and specify the SSID and the password.	Yes
2.4	The user must be able to set the vehicle into "remote control mode", where the vehicle is controlled by a remote application.	Yes
3	**Performance requirements - Android application**	
3.1	The delay between receiving a command from the desktop application, and sending a command to the vehicle's actuators should be no more than 150ms.	162ms worse case 122ms average case

4	**System Requirements - Android application**	
4.1	The drone must be able to hover in a stationary position for a minimum of 30 seconds.	No
4.2	The drone must be able to move in six degrees of motion.	Yes

Table 10 shows the software requirements for the remote desktop application. Requirement 6.1 was changed from “Control drone using the keyboard” to “Control the drone using an analog input device”, as the team determined that analog input devices offer a more intuitive means on controlling analog values such as thrust, attitude, and yaw rate. 100% of the desktop application requirements were met.

Table 10: Software requirements of remote desktop application

5	**FUNCTIONAL REQUIREMENTS - DESKTOP APPLICATION**	**TARGET MET?**
5.1	Connect to the phone via Wi-Fi.	Yes
5.2	Receive sensor data from the phone.	Yes
5.3	Send navigational commands to the phone.	Yes
6	**Interface requirements - desktop application**	
6.1	Control drone using the keyboard. *Changed to: Control drone using an analog input device*	Yes
6.2	Display incoming sensor data.	Yes
7	**Performance requirements - desktop application**	
7.1	The delay between receiving a user command and sending the command to the phone must be no greater than 100ms.	Yes

6 Conclusion

6.1 Commercial viability

The frame and electrical systems could potentially be commercialized with the software provided for free. However, the feasibility of commercialization is limited by two major factors. First, cost of the prototype was $407.42 which translates to a high production cost, but it could be reduced by taking advantage of economies of scale. Second, current software is unable to achieve stable flight, and a commercial product would need additional features such as autonomous functionality. The current setup contains all the required hardware, but further work on the software would be needed to make it possible.

The most significant reduction in price will be through the custom PCB. Quotes for 5000 units were obtained from PCBCart and Bittele Electronics. The final estimated cost was $0.98 per PCB, compared to the prototyping cost of $29.02 per PCB [28]. The second source of savings stems from the electrical components. The cost of the electrical components was researched on Digi-Key and Arrow.com, and prices for bulk quantities (500 units or more) are sold at a 30 to 50% discount when compared to the unit price. No bulk pricing information was found for the frame, landing skids, and propellers. Therefore, a general discount rate of 10% was applied to the costs from Table 7. The final bill of materials, not including the battery and smartphone, is estimated to be $88.75 per quadrotor.

6.2 Project Summary

Over the course of the project, the team has proven that converting two analog signals into four PWM signals is feasible. A proof of concept consisting of a quadrotor was built, in which the four motors were controlled by the left and right channels of a smartphone's audio port. Furthermore, the ROS environment was used, and its versatility was demonstrated. The group's software accomplishments include the design of a motor driver algorithm that interfaces with an Android smartphone's audio buffer, and the creation of a ROS back-end that enables the teleoperation of the quadrotor. Ultimately, the team was able to fulfill the majority of the requirements originally proposed in the blueprint submitted in September, 2017.

The primary technical advantages of the proposed quadrotor architecture are the reduced cost and its versatility. Although the UAV did not achieve stable flight and required further tuning of the PID gains, the current architecture allows for the eventual integration of additional phone sensors and camera. This system could therefore be used for a multitude of applications including research, photography, terrain mapping and recreational purposes. These are further expanded in Section 6.3. Another advantage of the proposed solution is that it encourages the re-use of old smartphones, thus having the potential for reducing electronic waste.

Despite its many advantages, the solution may present adverse societal impacts. For example, the creation of a more accessible quadrotor could lead to more incidences of UAVs in restricted areas. To mitigate this, the application could contain an internal map of no-fly zones. When a quadrotor approaches a no-fly zone, a warning could be displayed on the operator's control station. When the quadrotor arrives at the border of a no-fly zone, the software could implement a "virtual wall" feature that rejects further user commands in the direction of the no-fly zone.

6.3 Recommendations and future work

The team demonstrated that the circuit was an effective method of converting analog signals to PWM signals. A reason for unstable flight, however, was the unexpectedly high audio buffer latency in the Android operating system that caused motor commands to be executed 0.1 to 0.2 seconds late. Although a fixed-wing aircraft or land vehicle would be able to navigate properly at this delay, a quadrotor is an unstable system that requires a shorter delay. The team would, therefore, only recommend pursuing the method of motor control through the proposed circuit if:

1. The application cannot support the added complexity of a microcontroller, and
2. The application can operate with a delay greater than 100ms

Going forward with the proposed solution utilizing the audio port, recommendations include exploring the use of new software libraries for lower latency audio buffers in Android smartphones. Another

recommendation is to replicate this work on an iPhone, as the developer website states a minimum audio buffer of only 5 milliseconds [29].

If the application cannot operate with long output delays and can support the additional complexity of a microcontroller, the team would recommend exploring the use of the USB port and writing new libraries to permit communication between the Android phone and the external microcontroller.

The final recommendation for the hardware is an updated version of the custom PCB, based on the lessons learned when designing the prototype. Modifications include AND-ing the RF input and the physical switch into one signal. This would permit using only one power MOSFET, thereby resulting in a lower voltage drop across the transistor. The breakout board containing the RF receiver would also be integrated within the PCB.

Recommendations for the software involve allocating additional time to work on the stabilization algorithm and the tuning of the PID gains. This would be achieved by more frequent testing, and the development of a dynamic model of the quadrotor from which the gains can be calculated.

A number of steps can be taken to achieve stable flight and expand on the system's capabilities. Future work for this project includes reducing the audio buffer latency through the methods previously mentioned to allow for a faster refresh rate and motor control. The team would then recommend further pursuing the addition of ROS functionalities such as the collection and integration of GPS coordinates from the smartphone to permit semi-autonomous and autonomous navigation.

The second area of exploration is the use of external sensors to contribute to navigation and obstacle avoidance. This would be pursued by receiving signals using the microphone input of the audio port.

The final recommendation for future work consists in the incorporation of camera capabilities. This includes the use of OpenCV libraries aimed at real-time machine vision. By associating autonomous navigation with camera capabilities, the quadrotor could then be used as a research platform, as well as for area mapping and inexpensive access to aerial photography.

7 References

[1] E. Darack, "A Brief History of Quadrotors," Air & Space Magazine, 19 May 2017. [Online]. Available: https://www.airspacemag.com/daily-planet/brief-history-quadrotors-180963372/. [Accessed 3 April 2018].

[2] A. Meola, "Drone market shows positive outlook with strong industry growth and trends," Business Insider, 13 July 2017. [Online]. Available: http://www.businessinsider.com/drone-industry-analysis-market-trends-growth-forecasts-2017-7. [Accessed 3 April 2018].

[3] J. Fisher, "The Best Drones of 2018," Ziff Davis, LLC. PCMag Digital Group, 12 March 2018. [Online]. Available: https://www.pcmag.com/roundup/337251/the-best-drones. [Accessed 4 April 2018].

[4] Nanos Research, "Understanding cell phone recycling behaviours," Nanos Research, Ottawa, 2017.

[5] M. Leichtfried, C. Kaltenriner, A. Mossel and H. Kaufmann, "Autonomous Flight using a Smartphone as On-Board Processing Unit in GPS-Denied Environments," Interactive Media Systems Group, Vienna, 2013.

[6] E. Chirtel, R. Knoll, C. L. Bridget, N. Peck, J. Robarge and G. Lewin, "Designing a spatially aware, autonomous quadcopter using the android control sensor system," *Systems and Information Engineering Design Symposium*, pp. 35-40, 2015.

[7] Y. Gao, "What Makes The Quadcopter Design So Great For Small Drones?," Forbes, 23 December 2013. [Online]. Available: https://www.forbes.com/sites/quora/2013/12/23/what-makes-the-quadcopter-design-so-great-for-small-drones/#2b932ad4654f. [Accessed 30 March 2018].

[8] Transport Canada, "Flying your drone safely and legally," Transport Canada, 3 March 2018. [Online]. Available: https://www.tc.gc.ca/eng/civilaviation/opssvs/flying-drone-safely-legally.html. [Accessed 30 March 2018].

[9] DJI, "DJI Official Website," DJI, [Online]. Available: https://www.dji.com/. [Accessed 30 March 2018].

[10] Eachine, "Eachine Official Website," Eachine, [Online]. Available: http://www.eachine.com/. [Accessed 30 March 2018].

[11] Yuneec, "Yuneec Official Website," Yuneec, [Online]. Available: http://us.yuneec.com/. [Accessed 30 March 2018].

[12] International Rectifier, "IRFZ44NPbF HEXFET® Power MOSFET," International Rectifier, El Segundo, 2010.

[13] iMatix Corporation, "ZeroMQ - Distributed Messaging," 2014. [Online]. Available: http://zeromq.org/. [Accessed 14 October 2017].

[14] Object Management Group, Inc, "What is DDS?," 2017. [Online]. Available: http://portals.omg.org/dds/what-is-dds-3/. [Accessed 15 October 2017].

[15] Open Source Robotics Foundation, "ROS Documentation," 31 March 2017. [Online]. Available: http://wiki.ros.org/. [Accessed 14 October 2017].

[16] OpenSignal, "State of Mobile Networks: Canada (January 2017)," 2017. [Online]. [Accessed 22 October 2017].

[17] Lukac, Martin et al., "First-class meta-data: a step towards a highly reliable," in *International Conference, ADHOC-NOW 2010,* Edmonton, AB, Canada, 2010.

[18] J. Wright, "Dispelling Common Bluetooth Misconceptions," SANS Technology Institute, [Online]. Available: https://www.sans.edu/cyber-research/security-laboratory/article/bluetooth. [Accessed 24 October 2017].

[19] RC Timer, "Manual of RC Timer ESC 30A Brushless Motor Speed Controller," RCTimer Power Model Co.,Ltd, Hong Kong.

[20] Venom Power, "Venom 20C 3S 5400mAh 11.1V LiPo Battery," Vertical Partners West, LLC, Rathdrum.

[21] Google, "Protocol Buffers - API Reference," 26 September 2017. [Online]. Available: https://developers.google.com/protocol-buffers/docs/reference/overview. [Accessed 15 October 2017].

[22] J. Roy, R. Kirkpatrick, J. Chow and J. Jollivet, "Ros Utilities Git Repository," [Online]. Available: https://github.com/jeremyroy/ros_utilities.

[23] J. Roy, R. Kirkpatrick, J. Chow and J. Jollivet, "Simone GUI Git Repository," [Online]. Available: https://github.com/JustinChow/Simone-GUI.

[24] J. Roy, R. Kirkpatrick, J. Chow and J. Jollivet, "Simone Git Repository," [Online]. Available: https://github.com/jeremyroy/simone.

[25] J. Roy, R. Kirkpatrick, J. Chow and J. Jollivet, "Simone Remote Git Repository," [Online]. Available: https://github.com/jeremyroy/simone_remote.

[26] G. Owenson, "Build your own Quadcopter Flight Controller," 17 August 2013. [Online]. Available: https://blog.owenson.me/build-your-own-quadcopter-flight-controller/. [Accessed 24 November 2017].

[27] Tektronix, "TDS2000C and TDS1000C-EDU Series Manual," 14 April 2011. [Online]. Available: https://www.tek.com/oscilloscope/tds2000-manual/tds2000c-and-tds1000c-edu-series. [Accessed 5 April 2018].

[28] PCB Cart, "Home Page," PCBCART, [Online]. Available: https://www.pcbcart.com. [Accessed 2 Paril 2018].

[29] Apple Developer, "set Preferred IO Buffer Duration," Apple, 2018. [Online]. Available: https://developer.apple.com/documentation/avfoundation/avaudiosession/1616589-setpreferrediobufferduration. [Accessed 30 March 2018].

[30] Bluetooth SIG, "Host Controller Interface Functional Specification," in *BLUETOOTH SPECIFICATION Version 4.0 [Vol 2]*, Bluetooth SIG, 2010, p. 800.

8 Appendices

A – Effort

Table 11: Overall effort expended by each team member

NAME	OVERALL EFFORT EXPENDED
JEREMIE JOLLIVET	25%
JEREMY ROY	25%
JUSTIN CHOW	25%
RILEY KIRKPATRICK	25%

B – EAGLE schematic and board

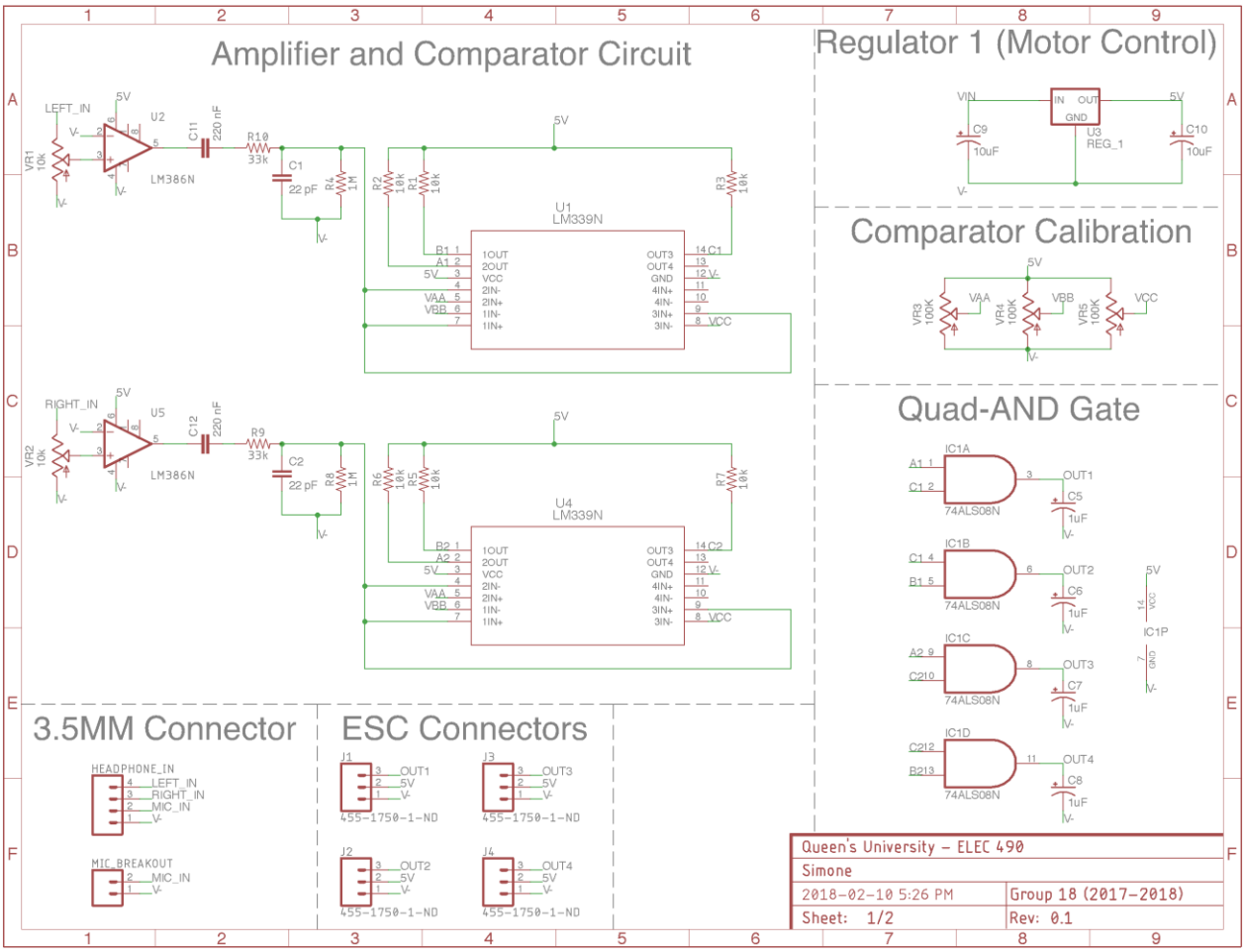

Figure 25: PCB Schematic (page 1)

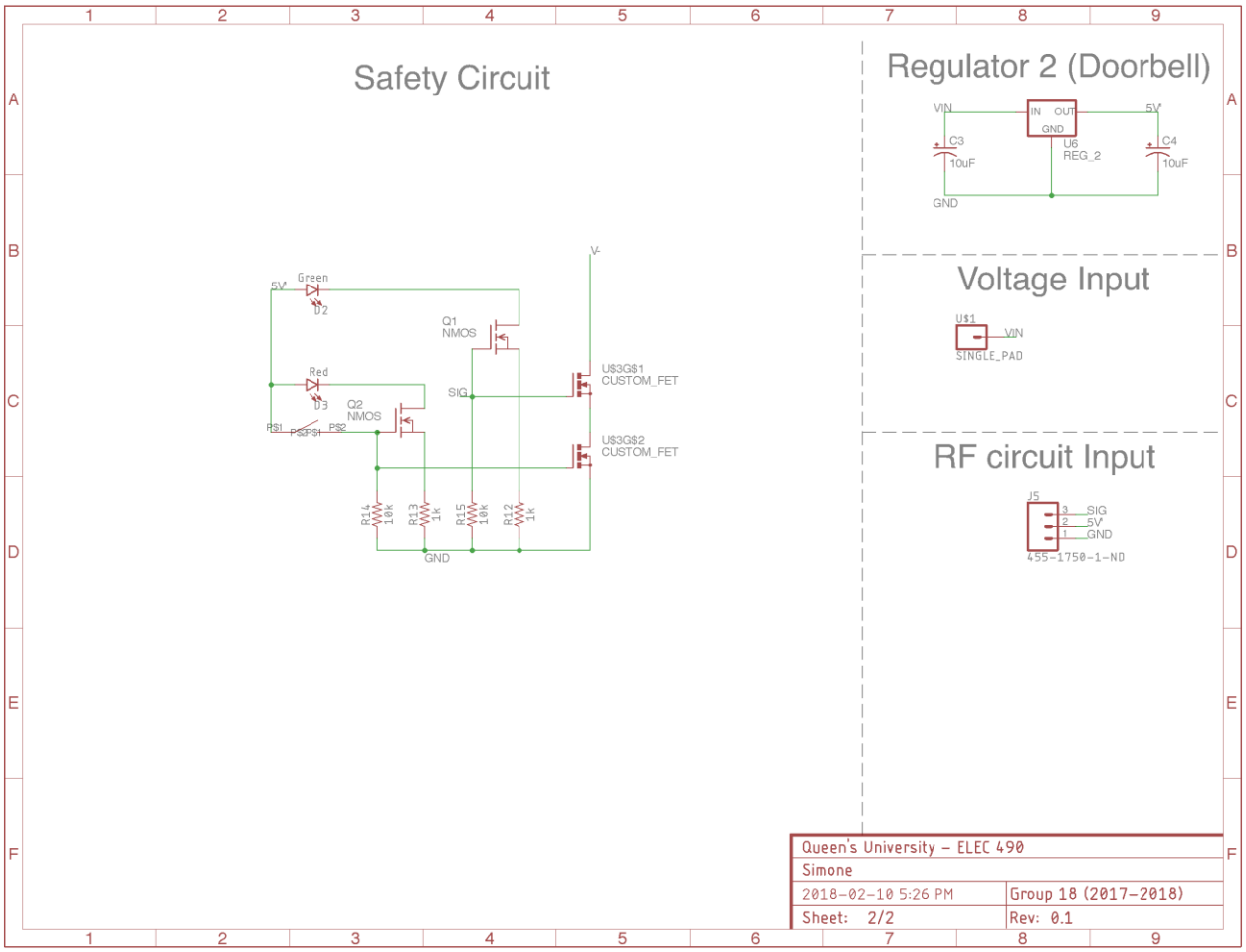

Figure 26: PCB schematic (page 2)

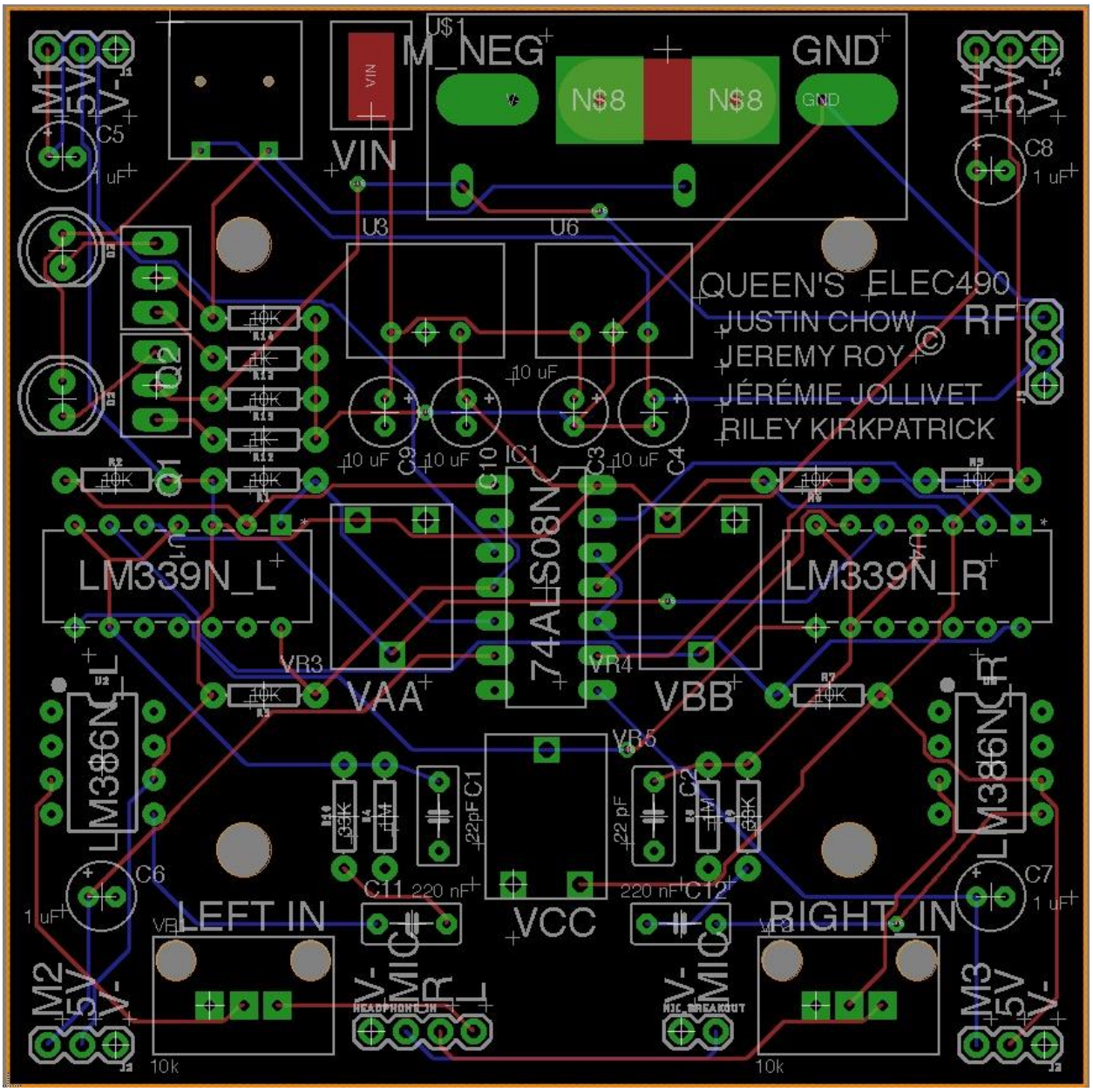

Figure 27: PCB Board layout

C – Testing procedure

Table 12: Example of Software Unit Testing procedure

NODE	TESTING PROCEDURE

IMU READER	Required equipment: a smartphone flashed with the software to be tested and a ROS-enabled remote computer.
	1. Create mobile hotspot on the phone.
	2. Connect remote computer to the phone's hotspot.
	3. Start a ROS Master on the phone.
	4. Start the IMU Reader node on phone.
	5. Configure the remote computer such that it can find the remote ROS Master. Type the following:
	`export ROS_IP=<IP address of computer>`
	`export ROS_MASTER_URL=https://<IP address of phone>:11311`
	6. In a console on the remote computer, type the following:
	`rostopic echo /phone_imu`
	This will print the stream of `sensor_msgs/Imu` messages received from the IMU Reader node.
	7. Rotate the phone along its axes and inspect the orientation and angular velocity fields of the sensor_msgs/Imu. Take note of any unexpected behavior.
	8. Rotate the phone along its axes and inspect the linear acceleration fields of the `sensor_msgs/Imu`. Take note of any unexpected behavior.
	9. Shut down all ROS nodes and terminate the ROS master.
	10. Repeat testing until no unexpected behavior is observed.

NODE	TESTING PROCEDURE

CONTROL LOGIC	Required equipment: a smartphone flashed with the software to be tested and a ROS-enabled remote computer.
	1. Create mobile hotspot on the phone.
	2. Connect remote computer to the phone's hotspot.
	3. Start a ROS Master on the phone.
	4. Start the Control Logic node on the phone.
	5. Configure the remote computer such that it can find the remote ROS Master. Type the following:
	`export ROS_IP=<IP address of computer>`
	`export ROS_MASTER_URL=https://<IP address of phone>:11311`
	6. In a console on the remote computer, type the following:
	`rostopic pub -r 25 /command/thrust <desired message>`
	7. In a second console on the remote computer, type the following:
	`rostopic pub -r 25 /command/yawrate <desired message>`
	8. In a third console on the remote computer, type the following:
	`rostopic pub -r 25 /command/attitude_adjusted <desired message>`
	9. In a fourth console on the remote computer, type the following:
	`rostopic pub -r 25 /phone_imu <desired message>`
	10. In a fifth and final console on the remote computer, type the following:
	`rostopic echo /motor_ctrl`
	This will print the stream of `simone_msgs/MotorCTRL` messages being output by the Control Logic node.

NODE	TESTING PROCEDURE
	11. Examine the output of the of the flight controller as output to the fifth console. Take note of any unexpected behavior. 12. Repeat steps 6 through 11 for different combinations of thrust, yaw rate, attitude commands, and dummy sensor data. 11. Shut down all ROS nodes and terminate the ROS master. 13. Repeat testing until no unexpected behavior is observed.

MOTOR DRIVER	Required equipment: a two-channel oscilloscope and probes, a 3.5mm audio cable extension cable, a smartphone flashed with the software to be tested, and a ROS-enabled remote computer.
	1. Turn on the oscilloscope and plug two probes into separate channels.
	2. Plug the stripped 3.5mm audio cable into the phone's audio port.
	3. Create mobile hotspot on the phone.
	4. Connect the remote computer to phone's hotspot.
	5. Start ROS Master on phone.
	6. Start the Motor Driver node on phone.
	7. Configure the remote computer such that it can find the remote ROS Master. Type the following:
	`export ROS_IP=<IP address of computer>`
	`export ROS_MASTER_URL=https://<IP address of phone>:11311`
	8. In a console on the remote computer, type the following:
	`rostopic pub -r 20 /motor_ctrl <desired message>`
	9. Using the oscilloscope, probe the audio cable's left and right channels respectively. Verify that the audio signals generated by the phone on both

NODE	TESTING PROCEDURE
	channels corresponds to the expected amplitude-varying PWM signals. Take note of any unexpected behavior. 10. Modify the motor commands being published to the `/motor_ctrl` topic. 11. Using the oscilloscope, verify that the duty of the signals output by the phone respond as expected to the new motor command. Take note of any unexpected behavior. 12. Repeat steps 10 and 11 for various motor commands. Shut down all ROS nodes and terminate the ROS master. 12. Repeat testing until no unexpected behavior is observed.
QUADROTOR LEVELER	Required equipment: a ROS-enabled computer. 1. Create a ROS Master. Type the following in a console: `roscore` 2. Start the Quadrotor Leveler node in another console. Type: `rosrun quadrotor_leveler quadrotor_leveler_node`). 3. In a third console, type the following: `rostopic pub -r 25 /command/thrust <desired message>` 4. In a fourth console, type the following: `rostopic pub -r 25 /command/yawrate <desired message>` 5. In a fifth console, type the following: `rostopic pub -r 25 /command/attitude <desired message>` 6. In a sixth console, type the following: `rostopic pub -r 25 /phone_imu <desired message>`

NODE	TESTING PROCEDURE
	7. In a seventh and final console on the remote computer, type the following: `rostopic echo /command/attitude_adjusted` This will print the stream of `hector_uav_msgs/AttitudeCommand` messages being output by the Quadrotor Leveler node. 8. Examine the output of the of the Quadrotor Leveler node as printed to the fifth console. Take note of any unexpected behavior. 9. Repeat steps 6 through 11 for different combinations of thrust, yaw rate, attitude commands, and dummy sensor data. 13. Shut down all ROS nodes and terminate the ROS master. 10. Repeat testing until no unexpected behavior is observed.
TELEMETRY VIWER	The command `rostopic echo /phone_imu` was used as the telemetry viewer. The rostopic tool was considered to operate properly. Therefore, no unit testing was performed on the Telemetry Viewer.
JOY + HECTOR QUADROTOR TELEOP	Unit tests were performed on the `Joy` and `hector_quadrotor_teleop` nodes simultaneously, as manually publishing the `sensor_msgs/Joy` messages that are accepted by the `hector_quadrotor_teleop` node is quite tedious. Required equipment: a ROS-enabled computer, Sony DualShock4 gamepad, and a USB Micro A to USB A cable. 1. Connect the Sony DualShock4 gamepad to the computer using the USB cable. 2. Launch the hector_quadrotor_teleop node in the DualShock4 configuration. Type the following in a console:

NODE	TESTING PROCEDURE

	`roslaunch hector_quadrotor_teleop sony_dualshock4.launch`
	3. In a second console, type the following:
	`rostopic echo /joy`
	This will print the stream of `sensor_msgs/Joy` messages being output by the Joy node.
	4. In a third, fourth, and fifth console, type the following commands respectively:
	`rostopic echo /command/thrust`
	`rostopic echo /command/yawrate`
	`rostopic echo /command/attitude`
	5. Vary the analog inputs on the DualShock4 controller and examine the resulting changes in the published messages. Take note of any unexpected behavior.
	6. Shut down all ROS nodes and terminate the ROS master.
	7. Repeat testing until no unexpected behavior is observed.