



**University of Arkansas – CSCE Department**

**Capstone II – Final Report – Spring 2021**

**Quadcopter Drone Controller**

**Zachary Heil, Lily Phu, Stephanie Phillips, Spencer Ward, Andy McCoy, Joel Parker,  
Christ Somphounout, Dishoungh White II**

**Abstract**

Drones are fun to build and control. The CC3D drone controller board is a cheap microcontroller board. However, it currently does not support the more modern features such as a barometer, GPS (Global Positioning System) functionality, and Wi-Fi support. The project holds multiple components that were implemented: the PCB (Printed Circuit Board), a firmware program (Betaflight) that aids the drone’s flight capabilities with onboard sensors and other control systems logic, and an application that can pull simulated sensor data via HTTP (Hypertext Transfer Protocol) server. This project is a collaboration with the Electrical Engineering (ELEG) and Computer Science / Computer Engineering (CSCE) departments. The Electrical Engineering team designed the hardware components and layout for the controller board. The CSCE team were primarily responsible for configuring the firmware and a mobile application on the side as a stretch goal that sends/receives information to the drone’s Wi-Fi chip (ESP8266) for data if time permits or utilizing HTTP requests to a mock server for simulated data if time was a factor. These features will be elaborated further in the Design part of the report.

**1.0 Problem**

One of the issues that the Electrical Engineering team faced is deciding on the microprocessor to use for the board’s design. The CC3D board currently supports flight stabilization, satellite receiver support, and S-Bus support with the outdated STM32F1 processor. One of the problems for the CSCE team is to adapt the existing Betaflight open-source firmware for drone controllers into our custom boards with the STM32F4 processor. Betaflight is designed for primarily pre-made controller boards. While it is possible to configure Betaflight for custom pinouts, it is beyond the normal use-case and the official repository lacks documentation on the matter. Without the firmware adaptation, the drone would not be able to perform basic flight control functions such as hovering and power delivery management.

## 2.0 Objective

The goal of this project is to create a quadcopter flight controller based upon an existing obsolete flight controller model, the “CC3D” with similar or better functionality to commercially available flight controllers. The scope of this goal is focused on the flight controller primarily on the features, drone behavior, and method of flight control. This flight controller was mounted onto the DYS XCITE 320 quadcopter but also should be compatible with almost any quadcopter design like most off-the-shelf flight controllers. Because of this, the quadcopter remained unchanged as the modifications was focused on the power delivery circuits in the DYS 320 quadcopter as well as the flight controller PCB. The controller delivers PWM RC control signals and output servo control signals to control the quadcopter motors. Betaflight allows the drone to support hovering and is configured to accept PPM and S-Bus control signals from a receiver. Also, our secondary goals were to include attaching a GPS module and extended functionality to be controlled by an app on a device. However, due to time constraints, some of these stretch goals were not fully implemented.

## 3.0 Background

### 3.1 Key Concepts

**Analog-to-Digital Converter (ADC):** Converts analog signals (real-world signals) to digital (0 or 1) representations of that signal so that a computer can understand it.

**AnyDesk:** A remote desktop application that allows users to access someone’s computer remotely as if that user was there operating on the other’s system in-person.

**Betaflight [1]:** Betaflight is the firmware that was used as the firmware of our flight controller. The focus on flight performance as opposed to other firmware forks made it a good pick for the firmware for the project’s flight controller. It also includes a Betaflight configurator GUI that can flash and modify the firmware’s functionality on-the-fly.

**Development Board [11]:** This term refers to the Olimex STM-P405 board, which is a red development board that features the same STM32F405RG that is used in the production board.

**Direct Memory Access (DMA):** Direct Memory Access is a feature that Betaflight supports that allows peripherals or any other I/O device to directly access the system’s memory without calling the CPU to fetch a memory block for it. This can be important for reducing the processor’s load, especially during flight. It is good to keep Betaflight’s RTOS under a certain load so it will have time to compute critical tasks.

**ESP8266 [4]:** A budget Wi-Fi chip that allows microcontrollers to connect itself to a Wi-Fi network which allows protocols such as TCP/IP stacks to be able to communicate with it. Because of its low price, it allows communication with the quadcopter with ease and accessibility.

**Expo [5]:** Expo is a foundation for an app which can implement React Native [17]. To allow users to quickly refresh apps and compile them for testing while building itself on iPhone, Android, and web-based browsers.

**Firmware:** Firmware is a type of computer software that gives information and instructions for communicating between the device and hardware. The software is stored in the hardware and is interacted with by the device by sending signals most commonly by certain frequencies. Firmware is especially common for everyday hardware such as TV remotes, routers, refrigerators, and many other daily devices and appliances.

**Inter-Integrated Circuit (I<sup>2</sup>C) [22]:** A synchronous, serial communication interface developed by Phillips Semiconductor, now named NXP Semiconductor. It requires only 2 wires (Data and Clock) to run and allows multiple masters and peripherals. It is not as fast as SPI (see SPI below), but it is more flexible as you can easily add more peripherals to the bus since it only requires 2 lines.

**Multiwii Serial Protocol [10]:** A serial protocol that Betaflight and other Cleanflight derived flight controllers use.

**PID Tuning [14]:** Tuning the PID means to adjust the coefficients of the P (Proportional) value, I (Integral) value, and D (Derivative) value. This allows a control loop that is acting upon feedback to give the appropriate response. In the case of our quadcopter, too much or too little of these values could cause the drone to wobble when trying to level or too slow to respond accordingly. PID tuning is important for configuring the drone to be “smooth” and “responsive” during flight.

**PPM Signals [13]:** Pulse Position Modulation signals are signals composed of pulses of a fixed length in a series to send information between the transmitter and receiver. It is like Pulse Width Modulation signals however PPM changes the position of the impulse without variation to the amplitude. The final design may or may not have PPM signals integrated in since it is a stretch goal.

**Production Board:** This term refers to the green flight controller board used for the drone. In the report, “flight controller board” and “flight controller” are used interchangeably to refer to the green production board.

**PWM Signals [13]:** Pulse Width Modulation signals are signals that are transmitted with the average length of information between the transmitter and receiver. The width of the pulses varies which is why it is pulse width modulation. The amplitude is constant while the positions are changed based on the signal. This is the signal that the final design of the controller will use if stretch goals of PPM signals are not met.

**React Native [17]:** React Native is a simple JavaScript based development foundation that is open source which allows users to create simple naïve apps. It allows for components to be implemented easily and with its fast refresh, it coordinates well with Expo to create a framework for a mobile app.

**Serial Peripheral Interface (SPI) [22]:** A synchronous, serial communication interface developed by Motorola. Most of the time, SPI requires 4 wires (Clock, Master Out-Slave In (MOSI), Master In-Slave Out (MISO), Chip Select), but sometimes it can be configured with a 3-wire setup. The biggest advantage to SPI is speed as it does not have a specified maximum speed. However, SPI is commonly configured to run at around 10-20 Mbps while I<sup>2</sup>C's maximum speed is around 3.4 Mbps with "high-speed mode".

**Serial Wire Debug [18]:** Provides microcontrollers a debug port for pin-limited packages.

**STM32Cube:** STM32Cube is an IDE (Integrated Development Environment) that allows developers to program STM32 processors, which is the processor that is used on the flight controller boards.

**System:** The system refers to the circuitry that makes up the quadcopter controller. In the design, the system features will refer to the features of the final design overall.

**Timer:** A timer is used for counting operations to give the hardware some sense of timing. It is used in the Betaflight configuration process.

**Unified Target:** A unified target, in Betaflight terms, is a flexible configuration file that the Betaflight Configurator accepts and describes the flight controller's behavior.

**Universal Asynchronous Receiver Transmitter [22]:** An asynchronous, serial communication interface and is one of the earliest communication protocols developed. Since it is an asynchronous protocol, it does not need a wire for the clock. It only requires 2 lines: transmit and receive. USART (Universal Synchronous and Asynchronous Receiver Transmitter) adds optional synchronous operation to the UART interface.

### 3.2 Related Work

The quadcopter is based off the original CC3D board which ran from OpenPilot firmware. This project uses Betaflight which is derived from Cleanflight and it significantly improves on the flight performance compared to the old OpenPilot. As discussed in the Problem section, Betaflight lacks documentation for creating custom unified target configurations. After configuring the custom flight controller, more documentation has been made in the official BetaFlight repository via pull request [16] so others in the future could get clearer instructions on how to configure a custom board like what was done here.

Furthermore, the CC3D mentioned in [3] has additional features that the project improved from the original design. The system has a IMU (Inertial Measurement Unit) sensor as well as barometer and magnetometer. The flight controller also included a RDQ Mini 8 GPS module, allowing Betaflight to receive GPS coordinates for navigation purposes. The GPS module always allows tracking of the controller location, which can help with understanding the altitude and elevation of the quadcopter. These give the quadcopter more advanced flight functionality.

Aside from the quadcopter, related work on the app side focuses on an MSP protocol app. There is also a Cordova-based app [9] to control the drone and its design shows a basic UI which connects to a TCP network through a URL and controls the whole drone via two radars controlled by numbers on the side. While this is similar to what the project at hand, the quadcopter's app is envisioned to be a joystick on the sides. Still, it serves as a possible inspiration however, the app created will not be utilizing Cordova but instead Expo.

## 4.0 Design

### 4.1 Requirements and Use Cases and Design Goals

#### 4.1.1 Requirements

For this project, there were a few important functional requirements for the quadcopter flight controller. The controller is a fully functional controller that operates a quadcopter drone using Betaflight firmware. The controller decodes remote control Pulse-Width Modulation signals to control the drone during flight. The drone must also output a minimum of 4 motor control signals that are, each, sent to the respective motor's Electronic Speed Controller (ESC) unit. The updated controller has hardware support for communication protocols such as Wi-Fi and GPS. When the drone is hovering (maintaining a near-constant altitude), no other control signals such as throttle are sent from a receiver source using a barometric pressure sensor (BMP280). The drone also has a mechanism for battery protection to prevent permanent damage to the battery due to weather or any external phenomena.

Alongside the functional requirements, there are some non-functional requirements as well. The drone should include LEDs and audio output devices to aid the user while operating the drone by indicating location, if crashed, fallen, lost, low battery, and other things. Features implemented past this point would exceed the foundational scope of this project. Another feature that we were looking into that was not a functional requirement is an external application to control the drone with a smartphone device instead of an RC receiver. This application would allow the pilot to view sensor data such as altitude and accelerometer/gyroscope data, and battery life.

#### 4.1.2 Use Cases

In the UML Use Case Diagram, the diagram shows three "actors" (the pilot with any kind of compatible receiver, the Betaflight firmware, and the power delivery circuit) interacting with the "system" or the drone in this case. The pilot, either by means of RC controller or through Wi-Fi mobile app, controls the drone's movements such as hovering and steering. The pilot through the receiver, sends control signals for throttle (sets how fast the motors will spin to gain altitude or stall to lower altitude), roll (controls the drone's longitudinal axis), pitch (controls the drone's traversal axis), and yaw (controls the drone's vertical axis). Alongside the pilot as the main user, a battery and the power distribution/monitoring circuit built into the controller delivers power and protects the controller against any voltage errors. The Betaflight firmware assists the pilot in flying the drone by using the onboard peripherals.

# Quadcopter Drone Controller

## Quadcopter Drone Controller (Use Case Diagram)

Group 3 | April 19, 2021

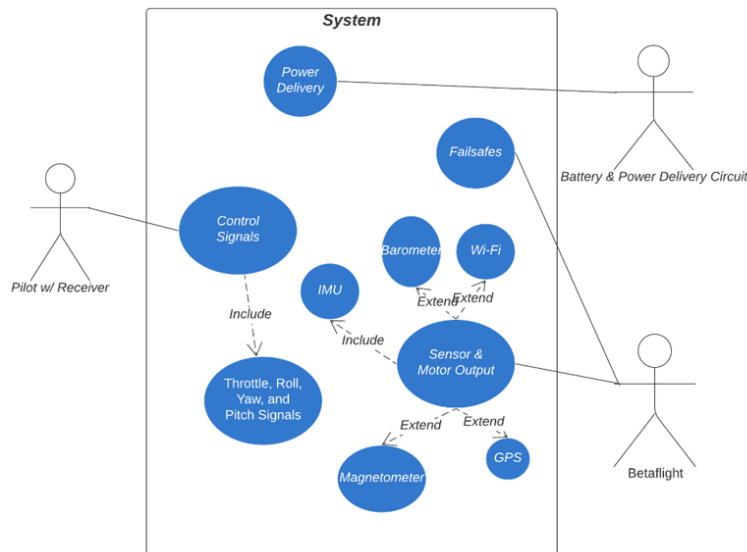


Figure 1: UML Use Case Diagram

### 4.1.3 Design Goals

The goal of this project is to create a flight controller board that supports more modern features than the outdated CC3D controller such as simultaneous S-Bus and PPM support for more flexibility, GPS support, Wi-Fi support, status alerts, and flight stabilization. The end goal of this flight controller is to have it control a quadcopter drone in a DYS Xcite 320 chassis, allowing it to hover and do other basic maneuvers. Some stretch goals have been established such as configuring LED strip and beeper support and creating a mobile application to control the drone through Wi-Fi.

# Quadcopter Drone Controller

## 4.2 High Level Architecture

There were four stages to the Quadcopter Drone Controller project. All four of those stages are discussed below.

### 4.2.1 PCB Design

The first phase of the project's implementation is to design the flight controller PCB (Printed Circuit Board). Below are two figures: the connection schematic for the entire "drone system" (Fig. 2) and the microcontroller schematic (Fig. 3). The processor chosen for our microcontroller is the STM32F405RG [19], which is a high-performance Arm® Cortex®-M4 based processor that can operate at a frequency up to 168 MHz. It can also have up to 1 MB of flash memory and up to 192 KB of SRAM (Static Random Access Memory). Its low cost, extensive I/O capabilities, and debug interfaces made a good choice for our flight controller board. For our drone, a USB circuit, GPS module, Wi-Fi module, an S.Bus connection, and power monitoring circuit for the drone's motors were added as external connections.

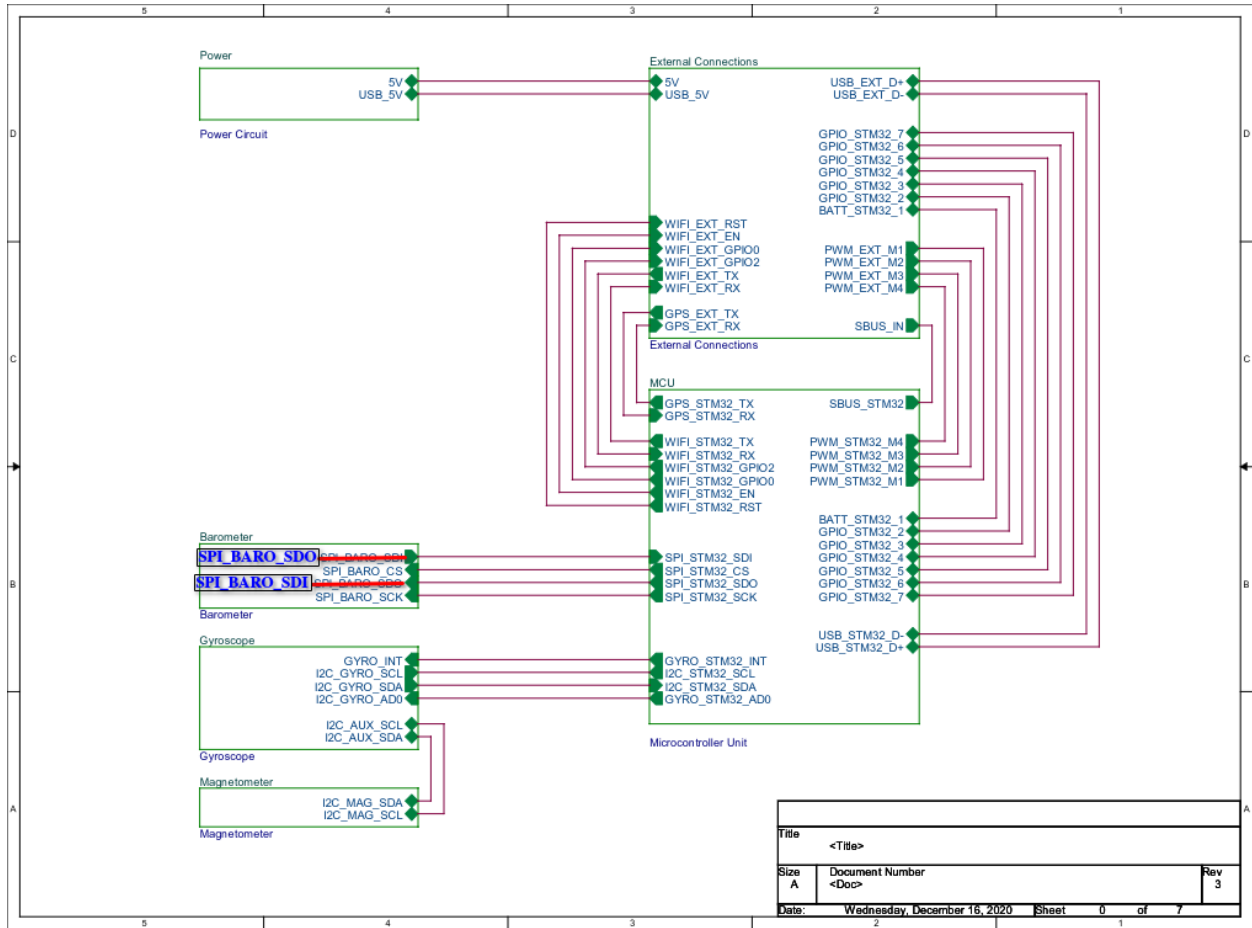


Figure 2: Component Connections Schematic





## Quadcopter Drone Controller

ESP-8266 Wi-Fi chip to a USART, connecting a GPS to another USART, connecting a reset switch to the BOOT0 pin to reset the processor when pressed, and connecting the motors to PWM output.

The main challenge for the PCB design was getting ordered parts in a timely manner. ELEG team incrementally soldered each module on the production board while CSCE tested the peripherals to ensure they could be programmed and showed expected behavior. Another challenge was correcting any mistakes that were encountered during the peripheral testing process.

### 4.2.2 Peripheral Integration

While ELEG team was incrementally adding peripherals and other headers to the flight controller board, CSCE team was tasked with testing the functionality of the peripherals using STM32Cube IDE, which is a development tool that allows the STM32F405 to be programmed in C. For much of February, the production board was yet to arrive or had any parts to test yet. So, the teams ordered a development board, which is the Olimex STM32-P405 (Fig. 4) [11]. It features the same processor. The teams also ordered a ST-Link USB Debugger. This provides the STM32 processor an interface to be programmed by a computer through Serial Wire Debugging (SWD). Many STM32 microcontrollers do not feature an on-board debugging interface.



Figure 4: Image of Olimex STM32-P405 Development Board

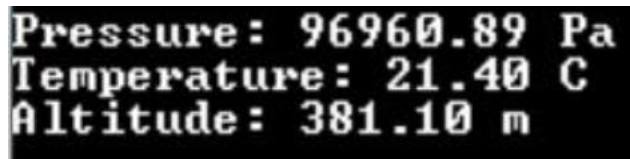
After getting the production boards from order, CSCE team tested the board's USB Serial interface to read output from the board. To set this up, a virtual COM port driver had to be installed into a Windows machine and there are libraries in the STM32Cube IDE that supports USB serial communication, which is the USB\_CDC library.



Figure 5: MISO and MOSI Switch Highlighted in Red Circle

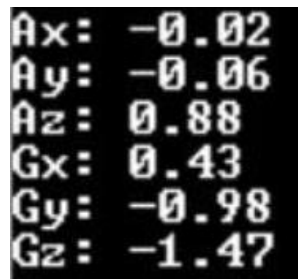
One of the first peripherals to be tested was the barometer. During that period of testing, a major PCB design flaw was detected. One of the mistakes made in the original PCB design was the processor's SDI line, which its input (MISO), was connected to the barometer's SDI line. Also, the processor's SDO line was connected to the peripheral's SDO. The correct connection is supposed to be: STM32\_SDO → BARO\_SDI & STM32\_SDI ← BARO\_SDO. Realizing this, adjustments had to be made to the board. Using tiny, enameled wires, traces were cut to bridge the correct SDO and SDI lines as shown in Fig. 5. After the fix, the barometer could output temperature and atmospheric pressure data using STM\_HAL functions for SPI communication. Altitude could then be derived from those variables as shown in the sample output (Fig. 6). Datasheets that include the derived algorithms used and code for the BMP280 test driver are provided in the GitHub repositories [21].

Next, the CSCE team tested the MPU-6050. Fortunately, the IMU did not reveal any further problems with the PCB that were major. At worst, the IMU had to be resoldered on some of the boards since some of the pads were not making good contact. After creating another test driver for the IMU using STM\_HAL functions for I<sup>2</sup>C communication, the IMU could output gyroscope and accelerometer data as shown in Fig. 7. Datasheets and code for the MPU-6050 test driver are provided in the GitHub repositories [21].



```
Pressure: 96960.89 Pa
Temperature: 21.40 C
Altitude: 381.10 m
```

Figure 6: Sample Barometer Output



```
Ax: -0.02
Ay: -0.06
Az: 0.88
Gx: 0.43
Gy: -0.98
Gz: -1.47
```

Figure 7: Sample IMU Output

After the IMU, the last peripheral that was tested was the LIS2MDL magnetometer. The magnetometer was a little trickier to test, since unlike the other peripherals on the board, it is not directly connected to the processor. According to the MPU6050 datasheet, the IMU can be configured to be a separate I<sup>2</sup>C master without help of the CPU. This frees up more pins that could be used for other things for our controller, but it makes it hard to debug this transaction as the processor cannot see it. To set the IMU to be its own master, the I2C\_BYPASS\_EN bit in register address 0x37 (INT\_PIN\_CFG) must be reset, which disallows the processor of accessing the auxiliary I<sup>2</sup>C bus between the IMU and magnetometer. Then, the I2C\_MST\_EN bit in register address 0x6A (USER\_CTRL) must be set, which allows the MPU-6050 to act as master for the magnetometer. Doing this will initiate a separate master-slave transaction between the IMU and magnetometer, which produces the similar output shown in Fig. 7, but the board



## Quadcopter Drone Controller

of hardware, setting custom configurations for custom board hardware like this is beyond the normal use-case, according to one of the Betaflight developers. The normal use case was loading pre-configured unified targets as Betaflight includes many different targets. The official Betaflight repository did not have sufficient documentation for configuring custom hardware after flashing generic STM32F405 firmware. In fact, there was some important documentation that was missing. Notably, the user must configure timers and DMA (Direct Memory Access) channels for some features.

The timers give the processor some sense of timing for operations such as motor output, I<sup>2</sup>C operations, and the Analog-to-Digital converters for the battery indicator. Direct Memory Access can also be important depending on the hardware setup. Direct Memory Access allows peripherals to, as the name implies, directly access memory instead of accessing by calling to the CPU to get the data from memory. This can be important for reducing load off the scheduler in Betaflight's RTOS. There could be many things trying to interact with the CPU at a time such as the IMU, the barometer, a GPS, video feed data, etc. While the drone is in flight, it is important to minimize the amount of load onto the CPU as much as possible to reduce the risk of Betaflight's scheduler not being able to handle a critical task on time. After this was discovered, a pull request was made and is merged to the official Betaflight repository to include this documentation.

The next peripheral to configure was the GPS module, a RDQ Mini 8. This is a GPS module that works out of the box. Not much had to be done to the configuration to set the GPS to work. All that was done was setting the USART3 port to a GPS sensor input and setting the GPS protocol to UBLOX, which is a standard GPS protocol. After that, Betaflight configuration was mostly done. ELEG team scheduled for a certified drone pilot to test our drone and it flew successfully.

The last thing that was configured on Betaflight was the PID (Proportional, Integral, Derivative) controller in Betaflight. In the first test flight of the drone, it was observed that the drone tended to overcompensate to feedback. The drone could hardly fly and could not land due to the hypersensitivity to the external feedback. This was due to not tuning the PID controller in Betaflight. For the next test, additional configurations were made to tune the P value, D value, and RC rate values. These settings, in addition to enabling a "Horizon" mode, made the drone fly much smoother and easier. The final drone is shown in Fig. 10 below. Any future improvements or tuning that could be done to the drone will be discussed in the Future Work subsection.



*Figure 9: Final Quadcopter Photo*

## Quadcopter Drone Controller

### 4.2.4 App Design

For the app design, the app documentation is provided along with the TSDoc comments implemented into the code. But for the baseline, the app started as a testing ground for HTTP requests. It was created using expo and react native as a naïve basic app. A mock drone server was created first which held the POST and GET methods for the mock drone to communicate with the app and vice versa. The app would utilize a useEffect hook to obtain information every few seconds within an interval. The fetching was done asynchronously from an endpoint and the information sent would be simulated such as the battery percentage of the drone, the acceleration and heading of the drone, the gyroscope's current axis, the altitude of the drone, the temperature and pressure which will help determine the absolute altitude of the drone, and other sorts of information necessary. Initializing these to a reasonable number allows the app to be used as a proof of concept. Through this information, the information can be sent back after the joystick was implemented.

Before the joystick's debut, the app was designed with a map and the foundation was changed using clever hooks that contained useState. The map allowed the usage of location data which allowed the drone to communicate its place with the phone. Initializing it from earlier, the drone would be able to be planted in 3D space with the current altitude, heading, yaw, throttle, etc. Another issue was the map itself would sometimes break so an override was created to ignore errors and test for issues when necessary. The app's imports of these libraries did not slow down the app by very much however when it was introduced that the joystick's compatibility would slow down the app by a large margin, the joystick would have to be done from scratch.

The joystick consisted of creating an inner and outer circle. However, the inner circle cannot hold in its own container unless a function prevented it from doing so. This caused a function called boundsLimit to be the key feature in holding the joystick together. Implementing a panResponder, the joystick would be in motion. The joystick had minor setbacks which were fully documented in the documentation but empty testing grounds were created to circumvent this situation. The joysticks were needed to be differentiated for their different purposes and using this information, it was sent back to the fly screen to be sent directly to the server. Because the server is the main computation component, if the ESP8266 was implemented, it would have been necessary for it to hold the computation as well. The mock server computed based on the joystick's location and allowed the drone to fully move and change the coordinates which would be reflected on the map.

The last part of the app would be to create an app with TCP protocol similar to the Cordova-based app for TCP. With expo, it would be difficult because of time-constraints, so it would be something to be pushed to future work. The app is shown in Fig. 11 below.



Figure 10: Final App Photo

### 4.2.5 Future Work

Due to time constraints, there were a few things that could have been done if the teams had extra time. One of the bugs that were encountered for the board design is whenever the board loses power, the board also loses Betaflight and the peripherals. When the board receives power again, Betaflight isn't re-enabled unless the reset button is held. However, still, most peripherals won't be enabled again as they should unless the board is connected to a computer again to set up Betaflight. The configuration is saved in the board, but not everything turns on as it should unless Betaflight is reset. It doesn't impact the core functionality of the board just as so long the board is receiving power. It's just a major inconvenience to connect and disconnect the board repeatedly every power cycle. This could be a flaw in the PCB design or a timer initialization problem in Betaflight. The board is also currently missing support for an LED strip and a beeper to signal the pilot on critical statuses such as low battery, GPS lost, and other failsafe protocols. This can simply be implemented with more commands in the Betaflight Configurator.

A major feature that the team intended on implementing is an app to send control signals (Throttle, Roll, Pitch, and Yaw). The app was to also be able to receive sensor values from the barometer, IMU, and GPS modules. There is a Cordova-based app that is like how we envisioned the app to be. It connects to the ESP8266 via TCP connection, and it controls the flight controller as a Wi-Fi receiver using MSP (Multiwii Serial Protocol) commands. While it does pull CPU Usage and AUX (Auxiliary) channel data, it currently does not receive data sensors data. This proves that the implementation is possible. There just was not enough time to reverse-engineer and create a new app from the ground up.

Another thing that could be done is to verify the sensor fusion between the IMU and magnetometer. In our schematic, the IMU and magnetometer are connected between their own I<sup>2</sup>C interaction. Betaflight can be coded to enable the IMU to be a I<sup>2</sup>C master without the CPU's help. The point of the magnetometer is to provide the IMU calibration data without having to set it in Betaflight. Using oscilloscopes show that there's data being passed in the auxiliary SDA line, but in Betaflight's perspective, there is no way in verifying if the IMU interfacing with the magnetometer.

## Quadcopter Drone Controller

### 4.3 Risks

Risk	Risk Reduction
Board Failure	<p>This design is focused on the two boards: First the flight controller board, then the power monitoring board. In the case of a complete failure of the flight controller board, the motors will lose power and the drone will fall out of the air. If the board were to have a partial failure of the flight controller board, systems of the drone could be shut down reducing the chance of injury to bystanders. In the case of power monitoring board failure, there is the chance that the lithium ion (or lithium polymer) battery could be ruined. To prevent this from happening, we will be incorporating a sensor to monitor the level of the battery to avoid under-volting. Using the battery monitor and Betaflight's failsafe configuration tools, the user will be notified that the battery is running low, giving them a chance to land the drone before it loses power.</p>
Misuse	<p>If not implemented properly, the design of the controller can be misused. The controller can be attached to almost any quadcopter with any intentions. The end user must have a warning about safety and misuse of the device, as well as the laws surrounding it.</p>
Safety Issues	<p>Without regulation, the quadcopter can fly up to 400 ft into the air, where it has the possibility to interfere with low flying air traffic. This must be monitored to fit within drone flying regulations set by the government. Also, the end user must be warned about the risks of flying near power/utility lines. The quadcopter must also be used by a licensed flyer.</p>
Environmental Issues	<p>Since the finished quadcopter will use a lithium polymer battery, there is a concern about proper disposal of such battery. Proper disposal must be advised on final product. A licensed flyer can also help check for environmental concerns of the drone.</p>
Destroyed Board	<p>In case if one of the controller boards were destroyed for any reason, 3 duplicate boards were developed.</p>

CPU Overload	Direct Memory Access channels were configured in Betaflight to prevent the CPU from being overloaded with tasks. This is especially important during flight so Betaflight will always have enough time to handle critical tasks.
--------------	--

### 4.4 Tasks

#### *Understanding:*

- Base flight controller design
- STM32Cube IDE
- BMP280 Functionality
- MPU6050 Functionality
- LIS2 Functionality
- Betaflight Configuration
- Timer & DMA Channel Setup (Betaflight)
- Receiver Signal (PPM or PWM or S.Bus)
- React-Native Framework for App
- TCP/MSP Back-end
- Drone Test Flight

#### *Design:*

- Create schematic
- Final PCB layout
- Betaflight custom unified target configuration
- Design react-native front-end
- Design HTTP back-end for app

#### *Implementation:*

- Print new PCB
- Program board
  - BMP280 driver
  - MPU6050 driver
  - MPU/LIS2 auxiliary I<sup>2</sup>C driver
- Hardware
  - Solder flight controller
  - Battery protection circuit
  - Support LED and sound indicators
- Betaflight (Software)
  - Flash base firmware through DFU
  - Configure primary peripherals



## Quadcopter Drone Controller

- Configure motors
- Configure receiver

### ***Testing:***

- Peripheral drivers (BMP, MPU, LIS2)
- Base Betaflight firmware (generic STM32F405 hex file)
- Drone flight (after Betaflight configuration)
- HTTP mock server
- App communication with mock server

### ***Documentation:***

- Pin layout
- MCU and Peripheral schematics
- Betaflight Configurator (unified target)
- USB Flashing
- Peripheral Testing
- All required FAA warnings
- App Documentation

## Quadcopter Drone Controller

### 4.5 Schedule

Tasks	Dates
1. Project Planning	9/15/20 - 12/1/20
2. Design: Schematics, Layouts, Test Plan	10/1/20 - 12/1/20
3. Hardware Implementation: PCB Layout, Print Board, Program Board, Control Signals, Hardware Soldering, Battery Protection Circuit	10/15/20 - 1/20/21
4. Flight Controller Peripheral Testing: BMP280, MPU6050, LIS2	2/15/21 - 3/21/21
5. Betaflight Configuration (Unified Target Creation): Peripherals, Motors, Receiver	3/21/21 - 4/16/21
6. ESP8266 Configuration: Flash esp-link firmware into Wi-Fi module and Research TCP/MSP Interface for App	3/21/21 - 4/23/21
7. App Development and Integration: Create HTTP Mock Setup & Reverse-Engineer MSP/TCP Controller	4/1/21 - 4/22/21
8. Finishing Touches on Drone: Fix Minor Problems and Test Flight	4/15/21 - 4/22/21
9. Final Deliverables: Poster, Final Report, Final Presentation, and Website	4/24/21 - 4/29/21

### 4.6 Deliverables

The deliverables for this project are as follows:

- A System Block Diagram & Specifications: These contain both the hardware and software components and the PCB layout.
- Flight controller and Peripheral Datasheets: These contains the architecture of the flight controller.
- Project Website [15]: All related deliverables will be contained on the website such as the Poster, Presentation Slides, Final Report, and the GitHub repositories with the relevant source files, config files, and documentation.
- Hardware: The finished flight controller and accompanying power board and other hardware used for the board will be delivered back to the ELEG department.
- GitHub organization [21]: This contains all the following repositories:
  - Betaflight: A forked repository from the official Betaflight repository to make any needed changes to the codebase.
  - DroneCTRL: Contains the source code for the mobile application using the React Native framework in TypeScript.
  - KnowledgeBase: Contains all documentation relating to the project such as configuring Betaflight, firmware flashing through DFU (Device Firmware Update), ESP8266 development, STM32Cube debugging, and peripheral datasheets.
  - STM32F4\_PeripheralTest: Contains the source code for the drivers to verify the peripherals on the flight controller in an isolated manner prior to flashing BetaFlight firmware, using STM32Cube IDE.

## 5.0 Key Personnel

**Zachary Heil** - Heil is a senior Computer Engineering and Electrical Engineering double major. He was responsible for coordinating and leading both teams. On the ELEG team, he was responsible for soldering. On the CSCE team, he was responsible for app development.

**Lily Phu** – Phu is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. She has completed software engineering and is experienced in many programming languages. She was responsible for app development.

**Stephanie Phillips** – Phillips is a senior Computer Engineering major at the University of Arkansas. She has completed software engineering, digital design, computer organization, embedded systems and is experienced with C, Verilog, VHDL, and Python. She was responsible for the backend development of the ESP-8266.

**Spencer Ward** – Ward is an undergraduate senior computer engineer at the University of Arkansas. He has experience with VHDL and Verilog, as well as C and C++. Currently in training for Dr. Di, he is researching asynchronous design technologies. He was responsible for the backend development of the ESP-8266.

**Dishoungh White II** – White is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Digital Design, Software Engineering, and Embedded Systems. He was responsible for testing primary peripherals on STM32Cube and configuring the Betaflight firmware on the drone controller.

**Andy McCoy** – McCoy is a senior Electrical Engineering major in the Engineering Department at the University of Arkansas. He was responsible for hardware testing.

**Joel Parker** – Parker is a senior Electrical Engineering major in the Engineering Department at the University of Arkansas. He was responsible for lab setup.

**Christ Somphounout** – Somphounout is a senior Electrical Engineering major in the Engineering Department at the University of Arkansas. He was responsible for log book & data.

**Skyler Hudson** – Hudson is a certified drone pilot and helped test the drone for flight.

**Alex Cutsinger (Champion)** – Cutsinger is a Software and Electrical Engineer for L3 Technologies, who graduated from the University of Arkansas with a Bachelor's Degree in Electrical Engineering. Cutsinger's interests are robotics and mathematics.

## 6.0 Facilities and Equipment

- Electrical Engineering Senior Design Lab (Facility) - The ELEG lab has a lot of important equipment and tools such as soldering irons, oscilloscopes, and other kits to help debug and create/modify our board.
- Olimex STM-P405 Development Board (Equipment) - The Olimex (red) board is a pre-made development board that we used to test the barometer and the STM32F405 processor in an integrated development environment (STM32Cube).
- ESP-8266 Wi-Fi Chip (Equipment) - The Wi-Fi chip was used as a bridge between our flight controller board and our proposed app.
- DYS Xcite 320 Drone Chassis (Equipment) - The DYS chassis is the body of our drone.
- Betaflight Configurator (Equipment) - The Betaflight Configurator is the GUI application used to flash generic STM32F405 firmware and to configure the hardware on our board to be used for flight.

## 7.0 References

- [1] Betaflight Home Page, <https://betaflight.com/>
- [2] BMP280 Pressure Sensor, <https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp280/>
- [3] CC3D Flight Control Board (Users Manual), <https://www.geeetech.com/Documents/CC3D%20flight%20control%20board.pdf>
- [4] ESP8266, <https://en.wikipedia.org/wiki/ESP8266>
- [5] Expo, <https://expo.io/>
- [6] KiCad STM32 Hardware Design and JLCPCB Assembly, [https://www.youtube.com/watch?v=t5phi3nT8OU&feature=emb\\_title](https://www.youtube.com/watch?v=t5phi3nT8OU&feature=emb_title)
- [7] LIS2MDL Product Overview, <https://www.st.com/en/mems-and-sensors/lis2mdl.html>
- [8] MPU-6050 Product Overview, <https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/>
- [9] MSP-Controller, <https://github.com/cs8425/msp-controller>
- [10] Multiwii Serial Protocol Overview, <https://ardupilot.org/copter/docs/common-msp-overview.html>
- [11] Olimex STM32-P405 Product Overview, <https://www.olimex.com/Products/ARM/ST/STM32-P405/>
- [12] Olimex STM32-P405 Schematic, [https://www.olimex.com/Products/ARM/ST/STM32-P405/resources/STM32-P103\\_P405\\_sch.pdf](https://www.olimex.com/Products/ARM/ST/STM32-P405/resources/STM32-P103_P405_sch.pdf)
- [13] PAM vs PWM vs PPM, <https://circuitglobe.com/difference-between-pam-pwm-and-ppm.html>
- [14] PID Explained for Process Engineers: Part 2 – Tuning Coefficients, <https://www.aiche.org/resources/publications/cep/2016/february/pid-explained-process-engineers-part-2-tuning-coefficients>
- [15] Project Website, [https://wordpressua.uark.edu/capstone/fall-spring-2020-2021/teams-1-5-f20/quadcopter\\_drone\\_controller/](https://wordpressua.uark.edu/capstone/fall-spring-2020-2021/teams-1-5-f20/quadcopter_drone_controller/)
- [16] Pull Request, <https://github.com/betaflight/betaflight/pull/10676>
- [17] React Native, <https://reactnative.dev/>
- [18] Serial Wire Debug, <https://developer.arm.com/architectures/cpu-architecture/debug-visibility-and-trace/coresight-architecture/serial-wire-debug>

## Quadcopter Drone Controller

- [19] STM32F405RG Product Overview, <https://www.st.com/en/microcontrollers-microprocessors/stm32f405rg.html>
- [20] ST-Link-V2 Description, <https://www.st.com/en/development-tools/st-link-v2.html>
- [21] UARK Quadcopter Flight Controller GitHub Organization, <https://github.com/UARK-Quadcopter-Flight-Controller>
- [22] UART vs SPI vs I2C Protocol Differences, <https://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html>
- [23] XCITE 320 Quadcopter, [https://hobbyking.com/en\\_us/dys-320-glass-fiber-folding-quadcopter-with-storage-case-pnf.html](https://hobbyking.com/en_us/dys-320-glass-fiber-folding-quadcopter-with-storage-case-pnf.html)