CUSTOM FLIGHT CONTROLLER FOR FPV DRONE

Jaelynn-Darshawn Abdullah

Hulya Goodwin

Muhammad Rabbani

Final Report for ECE 445, Senior Design, Spring 2025

Professor: Viktor Gruev

TA: Jason Jung

7 May 2025

Project #42

Abstract

Our beginner friendly FPV drone allows customers to control a drone that is budget friendly, has additional sensors to increase the robustness of the drone, and utilizes open-source software to run diagnostics and calibrate their drone with ease. The proposed system features full integration with Betaflight, ensuring access to a robust suite of configuration and tuning tools widely adopted by the FPV community. To enhance flight stability and reliability, additional sensors are incorporated to increase environmental awareness and system redundancy. This modular and scalable flight controller serves as an accessible platform for new drone enthusiasts while offering upgrade potential for more advanced applications.

Contents

1. Introduction

First Person View Drones, or FPV drones, first were invented in 1999, but the term FPV was coined officially in 2002/2003 by an online forum user “Cyber-Flyer” [1]. Quickly after, these drones went from something engineers were building themselves to being commercially available. As FPV drones became more accessible for hobbyists and the technology was getting better, these drones became even more popular. It’s common to even see one flying across the quad on a warm Spring day. The problem with this popularity is that the average person who wants to pick up this hobby can be intimidated by all of the expensive drones that aren’t beginner friendly. In the FPV Reddit thread FPV drone users have said to have crashed at least 8 drones before they finally got the hang of how to use them safely without damage. [2] The cost of building an FPV drone is estimated to be between $400-1800, which can be a steep price for beginners who will most likely crash their first drones. [3] This makes getting into the hobby of FPV drone flying daunting for people who may not have an income that would allow them to spend -at minimum estimation- $400*6 = $2400 just to get the hang of flying drones without crashing them. When looking at the current drone market, there are categories of ‘quadcopters’, ‘GPS’, ‘FPV’, ‘Mini”, etc. There is a gap in the current market for beginner friendly drones that are specialized to be cheap, durable, and an easy way to ease into the FPV drone hobby.

1.1 Proposed Solution

To address this gap in the current market for FPV, we will create our own custom flight controller and put together a drone to display its functionality. This custom flight controller (FC) will have all the necessary components to control the FPV drone ie. camera, IMU, radio controller/transmitter. On top of these basic components, we will add a humidity sensor to the FC, to let the user know if it’s going to rain while they’re flying. The FC will be compatible with the most common Open Source software for FPV drones, which is Betaflight, so that new users can get accustomed to the software they will most likely be using for their more advanced future FPV drones as well. If the user is content with our FC, they can easily continue to use it for their future drones, since they can customize their frames/Electronic Speed Controllers (ESCs) because our FC will be compatible with a range of ESCs and drone frames.

1.3 High Level Requirements

To determine that we have successfully made the custom FC, our project must complete the following requirements:

1. Demonstrate a functional flight controller that controls the motors to make balanced motions in the entire 360° plane. Motors can be powered and controlled for up to a full minute.
2. Demonstrates that the flight controller receives/sends accurate data from/to the microcontroller and integrated sensors. Video can stream 20+ FPS and sensors receive accurate live data within 5% error. Latency is below 100ms.
3. Drone has a functional system that turns on an LED and beeps when humidity sensors sense 90% air humidity or any raindrops. Humidity alert is audible and visible to users up to 10 feet away.

We believe fulfilling all these requirements demonstrates fully functional subsystems that should come together to accomplish our initial goals.

2 Design

2.1 Physical Design

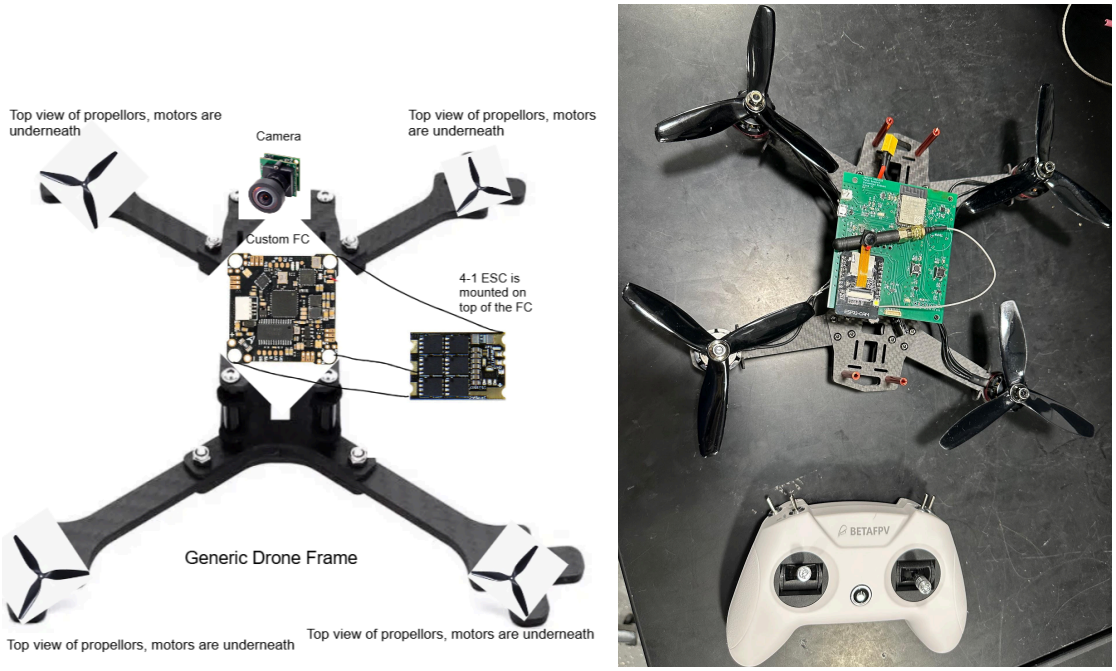

Figure 1: Physical Drone Design

2.2 Block Diagram

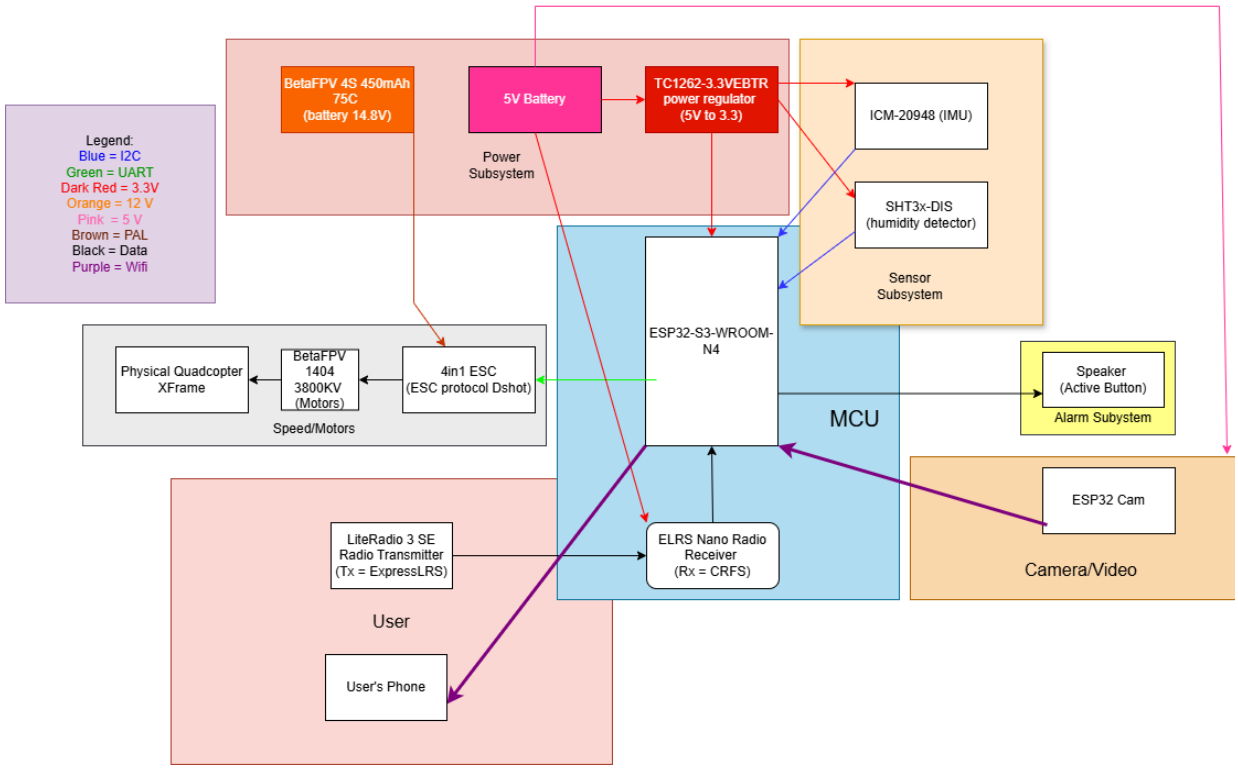

Figure 2: Block Diagram of Flight Controller Subsystem Integration

2.3 Functional Overview and Subsystems

2.3.1 Control Subsystem

For the main control unit, we are opting for an ESP32S3 microcontroller to provide communication between all the subsystems. By modifying the open-source Betaflight software to add additional capabilities, it will be responsible for controlling the ESC to control the speeds of the brushless motors and receiving data back on the motor speed, triggering the alarm system in high humidity conditions, transmitting video data to the user's device, receiving input from the remote controller to change the direction of flight through the radio receiver, and recording IMU data.

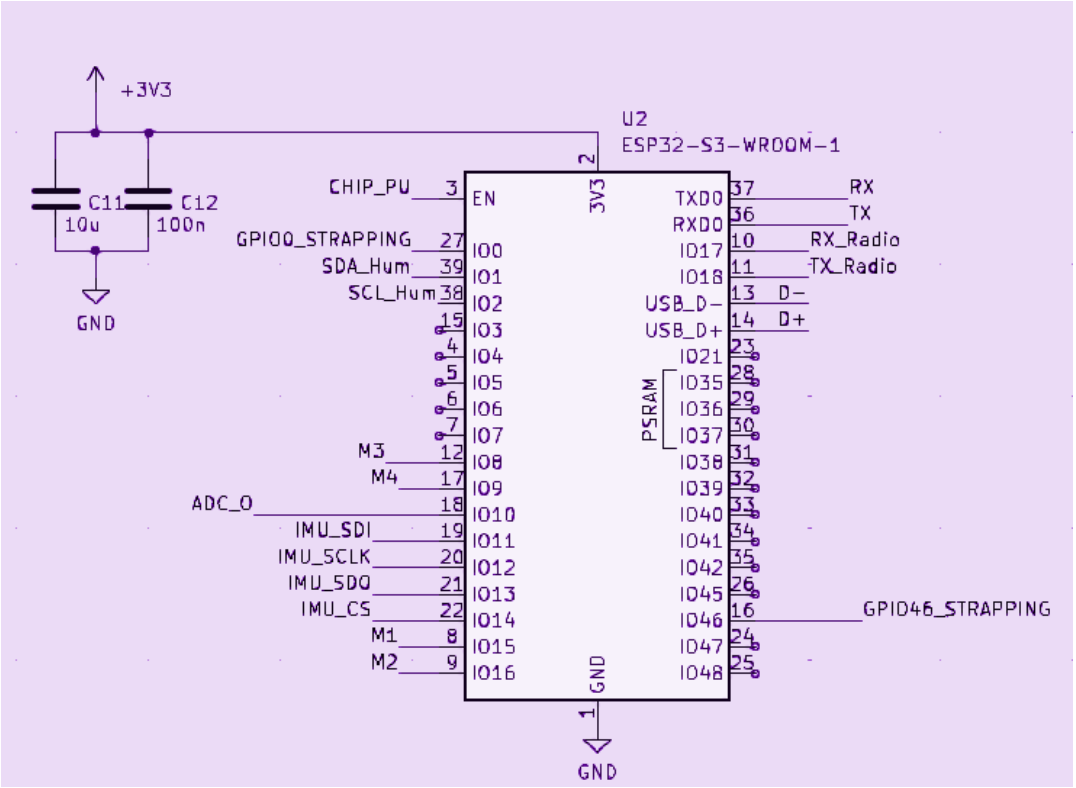

Figure 3: Schematic of ESP32-S3 on the custom flight controller with assigned GPIO pins.

For our ESP32, we are utilizing primarily a direct USB connection with a backup plan to utilize an external UART to USB device. For using the direct USB connection, there are two concerns: the GPIO0 pin and the physical traces. For flashing, we must ensure we are able to pull the pin to low. A user must hold the button low during the entire duration of the reprogramming process, which is tedious, however, most of the breakout boards provided within class use this configuration. With this in mind, we continued with this method as the primary way to flash our ESP32. Using the USB method, however, impedance matching, trace length, and trace spacing has been heavily studied. Due to the high-speed nature of USB, the lack of symmetry and distance will negatively impact the signal being transmitted. The USB traces also has a TVS array for overvoltage protection. With the varying levels of voltages being carried on voltage lines, it is easy to accidentally short devices when going from higher voltage devices to lower voltage devices. We have a JTAG that may be used as backup, but the USB worked. Additional capacitors at the VDD and Vin pins in conjunction with the TVS arrays an logic level shifters will provide enough protection for our FC.

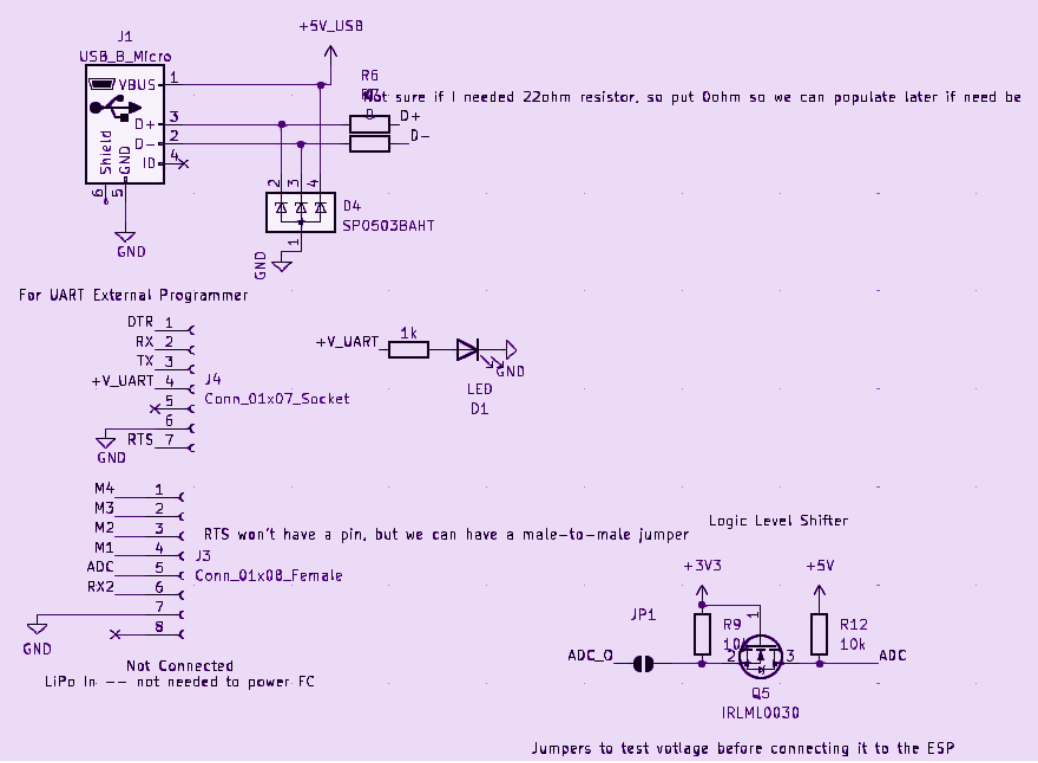

Figure 4: Schematic of USB and JTAG that is utilized for flashing

2.3.2 Power Subsystem

The power subsystem is responsible for powering the components on board. The 14.8V LiPo battery is lightweight and a commonly used component for FPV drones. The total calculated current draw is around 33A with a predicted motor control time of around one 1 minute. The regulators will step down the voltage to 5V and 3.3V to power the different subsystems. Specifically, the control and sensor subsystems will require 3.3V while the remaining subsystems (outside of the Motor Subsystem) will require 5V. The 5V will be provided by an additional battery to reduce the noise generated from the 14.8V battery and increase modularity of our design.

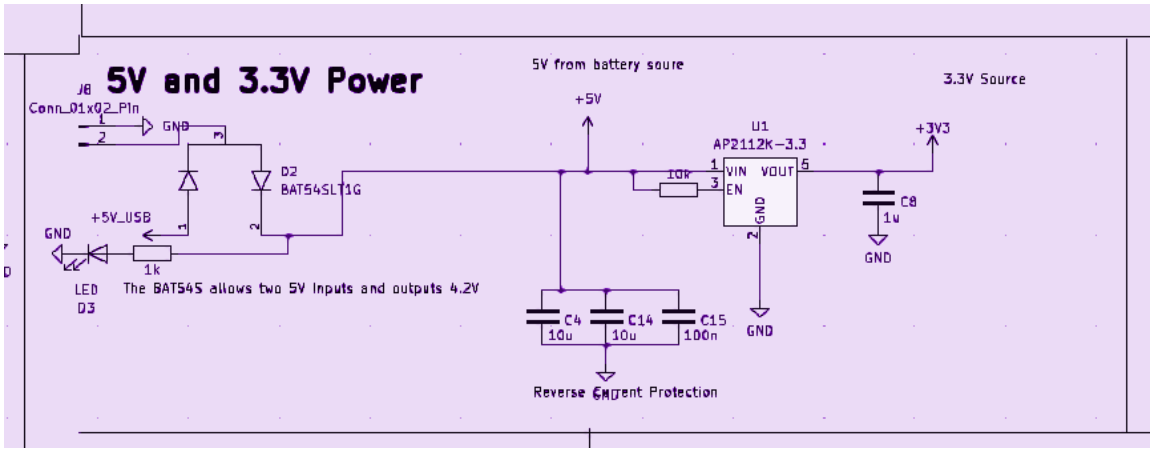

Figure 5: Power Subsystem containing a TVS, 5V to 3.3V voltage regulator, coupling capacitors, and an indicator LED that signifies when 5V is supplied to the board.

2.3.3 Sensor Subsystem

The IMU consists of an accelerometer, gyroscope, and magnetometer. Linear acceleration, angular rotation, and magnetic field of the drone are measured by the IMU. Using this data, we can detect at what speed the drone is under maximum load, what angle the drone is at to allow the user to correct the flight path, and the position of the drone. Having a functional and accurate IMU is integral to controlling the drone properly. All data collected by the IMU should be communicated to the ESP32 through SPI. This data is used to calculate roll, pitch, yaw, and throttle to allow for user control of motors through the radio transmitter.

The humidity sensor measures both humidity and temperature. All data collected by the humidity sensor should be communicated to the ESP32 through I2C. This data is then used to calculate the potential for rain/water damage to the flight controller. Once past a certain humidity threshold, an alert signal is sent to the alarm subsystem.

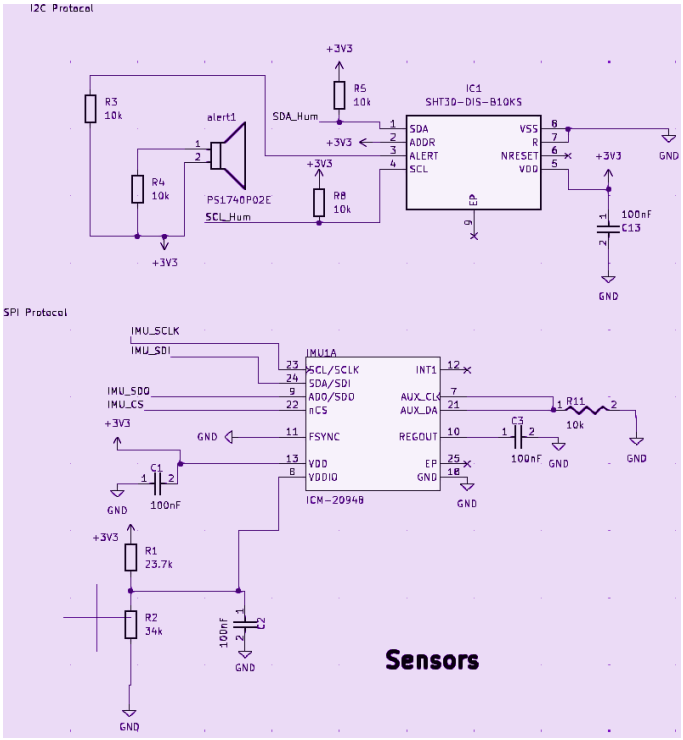

Figure 6: Sensor Subsystem containing IMU and Humidity Sensor communicating to ESP32 through SPI and I2C respectively

2.3.4 Alarm Subsystem

The alarm subsystem is what we will define as our Passive Buzzer, controlled by the alert signal sent from the ESP32 and Humidity Sensor. This allows for audible feedback to the user when humidity levels are dangerous/potentially damaging for the drone to fly in. A 5V Passive Buzzer will be used for our Alarm Subsystem.

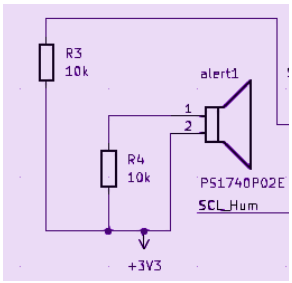

Figure 7: Alarm Subsystem consisting of the Passive Buzzer triggered by an Alert Signal

2.3.5 Motor and ESC Subsystem

The motors subsystem consists of the speed controller, the brushless motors, propellers, and quadcopter frame. Located in all 4 corners of the quadcopter frame will be the brushless motors that are controlled

by the ESC to control the direction and speed of travel. The drone must be able to navigate through the entire 360° range of motion using the joysticks on the remote controller. User inputs should be communicated through the flight controller to the ESC, controlling all motors.

Each motor draws up to 8A and must be continuous during flight. Additionally, their maximum thrust output must be able to support beyond the physical weight of the flight controller and frame. Our motors and propellers' maximum thrust is calculated to be 1375 grams, which can fully support our estimated 305 gram drone.

Our quadcopter frame must be within our estimated weight limit while being able to house our PCB, ESC, wiring, and batteries.

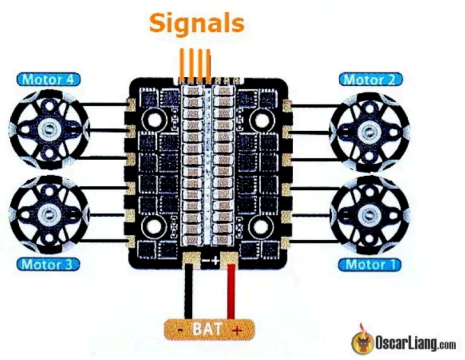

Figure 8: Wiring Diagram of ESC and motors

2.3.6 User Subsystem

The user should be able to fully control the drone via a radio transmitter. The radio transmitter must be able to communicate to the radio receiver to ESP32 sending roll, pitch, yaw, and throttle signals.

Additionally, the user can view a live camera view during flight through their smartphone.

2.3.7 Camera Subsystem

The camera subsystem is what we will define as an ‘FPV’ for our drone. This allows new users to view what the drone sees. The ESP32 Cam Module will be responsible for taking digital video data and sending it to the Control Subsystem to be streamed.

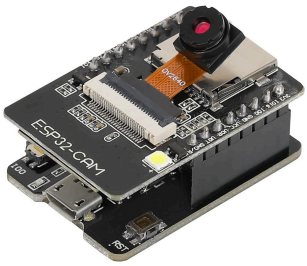

Figure 9: ESP32 Cam Module

3. Design Verification – go into R&V details here (table is in appendix)

The verifications of BetaFlight compatibility can be seen in Figures 10 and 11, where we have connected to BetaFlight successfully. In Figure 10, the DSHOT600 protocol that we used to communicate between the motors, ESC, and ESP32 can is visible and in the right corner of the 4 motors and slide bars is how each individual motor was spun and tested.

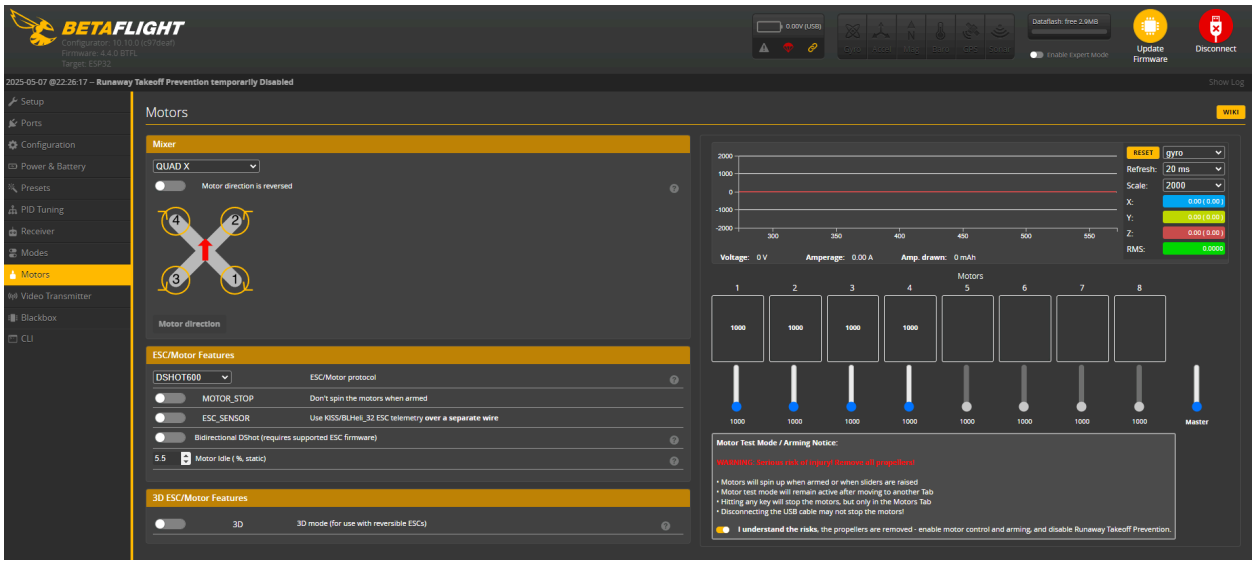

Figure 10: BetaFlight Interfacing with ESC and Motors

As seen in Figure 11 below, BetaFlight was also able to verify that our radio transmitter, radio receiver, and ESP32 connections to the receiver were correct. The visualized drone in the center of Figure 11 moves according to the yee, yaw and pitch of the radio transmitter. In Figure 12, there is a graph from data that my group collected as we moved the radio transmitter's switches. This data actually is provided in BetaFlight and the software creates its own graph, but for the sake of clarity our own graphed data can be seen below. As we increased the throttle using the transmitter, the throttle would simultaneously increase in the graph as well.

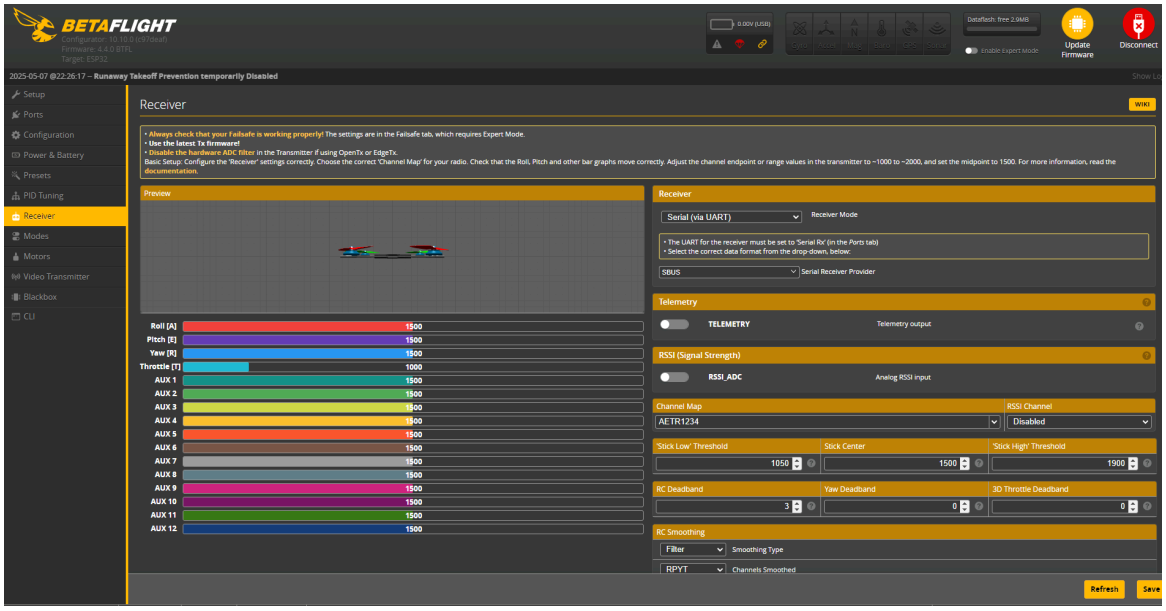

Figure 11: BetaFlight Interfacing with Radio Transmitter

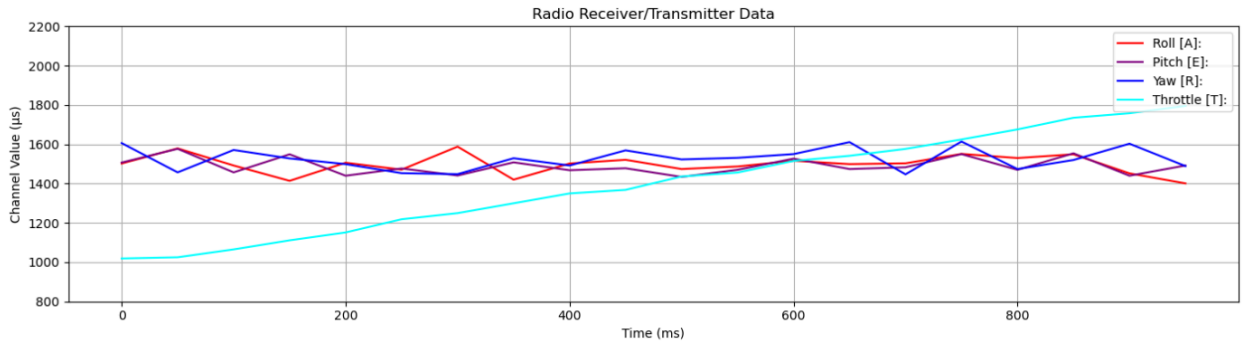

Figure 12: BetaFlight's Radio Receiver/ Transmitter Data

The sensor verifications were performed by using the Serial Monitor in Arduino to print out the data calculated by the humidity sensor and IMU. For the humidity sensor, my team increased the humidity by breathing onto the sensor. In Arduino, the instantaneous increase was visible. An example of this increase with the data that was measured by the humidity sensor is seen in Figure 13. The alert was also verified this way by setting the max humidity level in Arduino to send out the high signal to the buzzer. For the IMU's verification Arduino would confirm if the IMU was abruptly moved its acceleration would increase and if the IMU was flipped upside down, the gyroscope would display it in the serial monitor. The data from the accelerometer and gyroscope of the IMU can be seen in Figures 14 and 15, respectively.

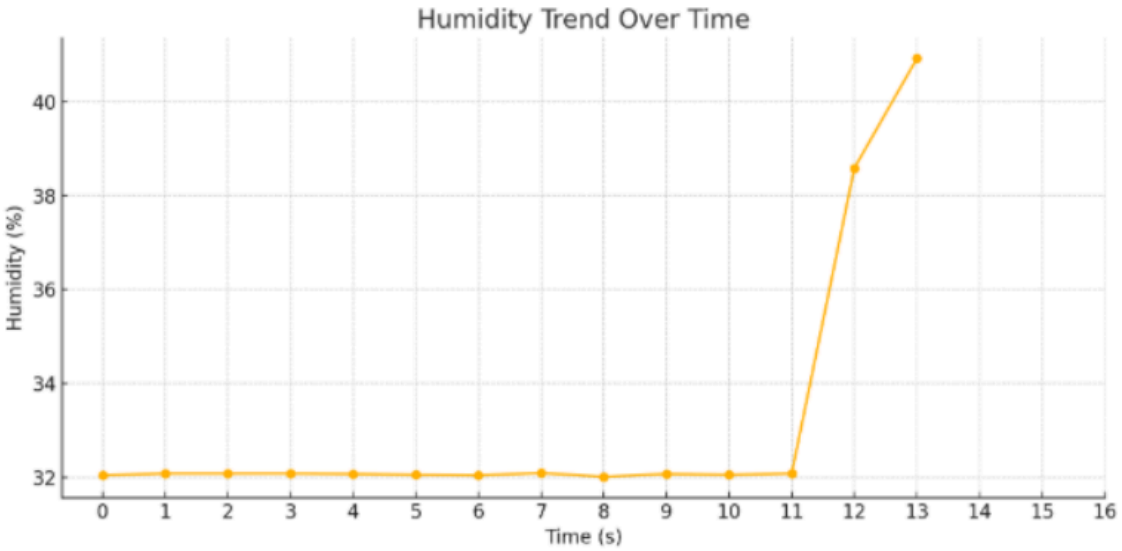

Figure 13: Humidity Sensors Data

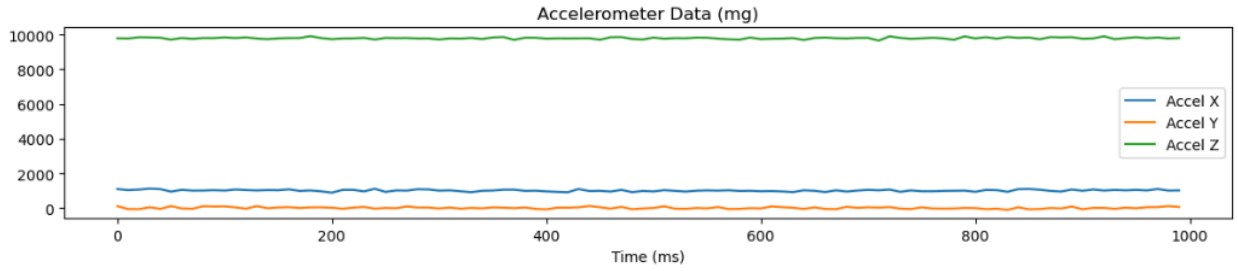

Figure 14: Raw Data From the IMU's Accelerometer

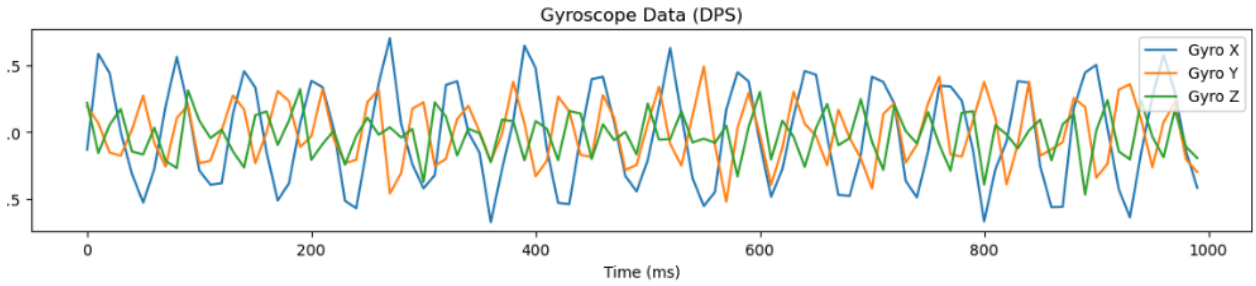

Figure 15: Raw Data From the IMU's Accelerometer

3.1 Hardware and PCB Design Choices

3.1.1 Signal Integrity and Protection

Since noise and there are critical parts needed to operate in order ensure the success of our project, voltage regulators as well as additional resistors and capacitors were used across power traces. From the battery, a 5V voltage regulator and a 3.3V voltage regulator. Our camera, receivers, and other peripherals

need a constant 5V to operate while our microcontroller and sensors require a constant 3.3V. 100 microfarad decoupling capacitors are placed at the input and 10 nanofarad decoupling capacitors output terminals of the voltage regulators in order to prevent static-hazard glitches as well as send in cleaner signals.

For the EN and VDD pins, we are using 10 kOhm resistors as pull-up resistors to ensure we have a constant high during operation. Similar to the using I2C and UART lines, we will be using 10 kOhm to ensure the signals are clean at the input due to the importance of sensor data for our drone.

3.1.2 ESP32 Microcontroller Communication with Sensors and Choices

With BetaFlight being able to flash onto the ESP32, we can run diagnostics to ensure communication is being handled properly. In Figure 10, Betaflight can test the ESC and the radio receiver. The motors can receive DSHOT signals replicating a signal from a radio transmitter to spin up and spin down the motors. For the radio receiver, there is a 3D simulation of a quadcopter drone that can simulate the current yee, yaw, and pitch. With this in mind, we are able to verify that we can achieve the right thrust and on top of that see the potential latency from the radio transmitter as seen in Figure 11.

3.1.3 Power Subsystem Design

To verify our 5V to 3.3V process, we utilized a PMOS and a voltage regulator as shown in the simulation in Figure 11 to protect from overvoltage events and clean any noise from the voltage source. Due to the inherent voltage drop across the PMOS when conducting, the output at the source of the PMOS is approximately 4.25V. This 4.25V power source then serves as the input to our voltage regulator which steps the voltage down to a regulated 3.3V output. From here, we now have power that can supply both the entire flight controller assembly.

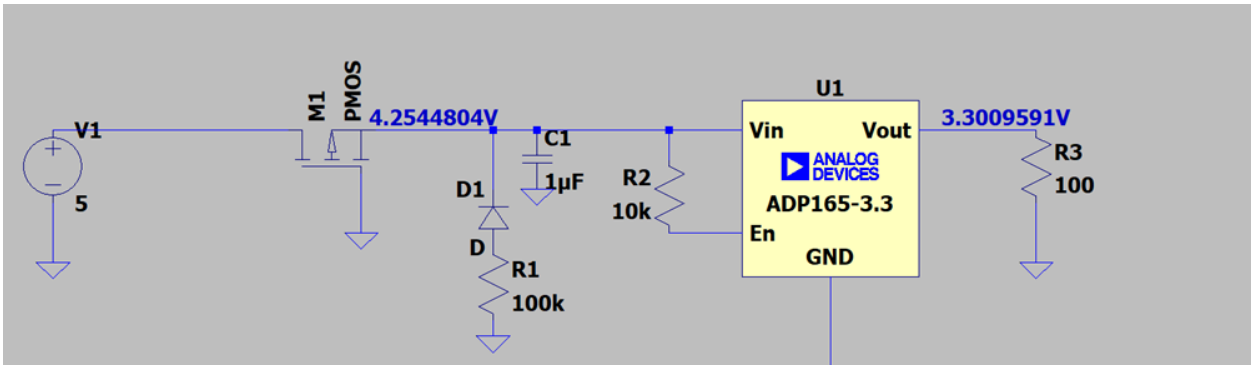

Figure 16: LTSpice simulation of Power Subsystem.

3.1.5 Weight and Size Management

Our main limitations with our FPV drone to achieve lift under its total physical weight are our motor RPM, propeller size, and battery capacity. Most sensors and control subsystems are chosen beforehand to achieve functionality. Their total weight (PCB & ESC) comes out to around 42.5 grams. For ease and efficiency, we chose a standard 5” frame to house our components as all components laid out fit comfortably within such a standard frame. With 5” propellers, we can decide on an efficient motor paired with a compatible battery. A motor with high documentation weighing below 50 grams we found

was the EMAX RS2205S 2300KV. Pairing this with the LiPo battery, Or 14.8V 4S 650mAh 80C, gives us a total compatible system of motor and battery weighing under 100 grams total. This gives us ample room in terms of weight for our 5” frame and additional components, as this motor consistently provides 442+ grams of thrust at 50% PWM [7].

3.2 Software Design Choices

3.2.1 BetaFlight

We chose to use Betaflight for its popularity and open-source documentation. This provides us and future users with widespread support and debugging resources. Additionally, the long-time development of Betaflight gives it ease of use, polish, and quicker debugging.

However, Betaflight lacks support for fully custom flight controllers. Most flight controllers use STM32 F405 microcontrollers. This provided some difficulties in flashing an ESP32 microcontroller with Betaflight firmware. Additionally, Betaflight contains a limited number of IMU libraries, preventing integration of most outside IMU’s. Lastly, Betaflight lacks support for custom sensor integration.

3.2.2 Arduino for IMU and Humidity Sensor

The lack of flexible sensor integration in Betaflight required us to write custom Arduino code to communicate with our sensor subsystem. This allowed us to have fully functional sensor and alarm subsystems.

However, there was difficulty in actually integrating our Arduino code with our Betaflight firmware, preventing us from compiling all subsystems on one ESP32.

3.3 Tolerance Analysis

3.3.1 Motor Thrust vs Weight

Another risk to our design is achieving lift despite the drone’s weight. The table below shows our motor paired with a similar ESC and similar 5” propellers with thrust measurements at different percentage PWMs [7]. As seen below, all propellers provide 350+ [g] thrust at 50% PWM, achieving lift with our design specifications. Additionally, calculating the total weight of our drone and its parts, we find that our weight is well below 442g. This gives us plenty of weight room to prevent potential risks.

Note: this time I had to conduct this test on a different ESC – the Racerstar MS35A.

Props	%	Peak Thrust – g	Peak Current – A	Max Power – W	Efficiency – g/W
KingKong 5040	50	364	5.8	98.2	3.71
	100	1257	30.1	483.7	2.60
Gemfan 5040×3	50	441	6	103.4	4.26
	100	1327	29.58	479	2.77
Gemfan 5045 HBN	50	442	6.2	104.8	4.22
	100	1375	30.3	485.3	2.83
DAL 5045×3 HBN	50	488	7.21	121.9	4.00
	100	1440	35.87	572.3	2.52
Racekraft 5051×3*	50	459	7.31	121.9	3.77
	100	1313	50.61	810.4	1.62

Figure 16: Table of thrust tests and measurements for RS2205S Motor

4. Costs

4.1 Parts

Part	Manufacturer	Cost	Quantity	Description
RS2205 2300KV Brushless Motors	EMax	$8.5	4	Brushless motors for drone
Lumenier Mini Razor Pro ESC 45A	GetFPV	$59.99	1	ESC for motors
ICM-20948	DigiKey	$7.11	1	9-axis IMU that has accelerometer, gyro, and uses I2C and SPI protocol
SHT30-DIS-B10kS	DigiKey	$2.70	1	Humidity Sensor

Humidity Sensor				for drone that uses I2C protocol
ESP32 Cam	AI Thinker	$5.80	1	Mini camera for FPV camera
ELRS LiteReceiver V1.1	BetaFPV	$9	1	Radio Receiver
ESP32-S3-WROOM-U1-N4	DigiKey	$2.95	1	Microcontroller
LiteRadio 2 Radio Transmission	BetaFPV	$24	1	Radio Transmission
LiPo 14.8V 4S 650 mAh 80C	Flyfive33	$15	1	14.8V battery
5 Inch Propellers (16 Pack)	Gemfan Hurricane	$13	1	Propellers for motors
5-Inch Frame (Carbon Fiber)	Amazon	$13.47	1	Frame for Drone
Tax (11.5%)				
Shipping (5%)				
		Total: $217.88		

4.2 Labor

Our team is made up of two Computer Engineers and one Electrical Engineer. Looking at the UIUC report about starting pay of $109,176 and $87,769 respectfully, after taking the average of these two salaries and assuming 30 days per month as well as 8 hour work days, we can assume (if we are charging hourly and not on a salary) that each member of the team has an hourly rate of around $34. [5] Each week, members are expecting to work 6 hours for 12 weeks, giving us a total of **$7344** for labor. However, we can expect finals costs to be around 7344*2.5 or **$18360.**

4.3 Schedule

Week	Tasks	Person
February 23 - March 1st	• Finalize and order parts • Flash Betaflight onto ESP32 using GitHub port • Begin schematic design	• Jaelynn • Jaelynn & Muhammad

		• Hulya
March 2nd - March 8th	• Begin breadboard demo for placement • Update schematic design and create PCB	• All • Hulya & Jaelynn
March 9th - March 15th	• Second round of PCB Design • Work on ESC and motors • Solder PCB • Demo breadboard	• All • Jaelynn & Muhammad • Hulya • All
March 16th - March 22nd	• SPRING BREAK	
March 23rd - March 29th	• Debug PCB • Redesign PCB • Functional motors/ESC with Betaflight • Have the frame ordered	• Hulya • Hulya & Jaelynn • Muhammad & Jaelynn • Muhammad
March 30th - April 5th	• Correct weight balancing • Functional camera stream video to WiFi • Debug PCB • Redesign PCB	• Muhammad • Jaelynn • Hulya • Hulya & Jaelynn
April 6th - April 12th	• Order third round of PCB • Have Humidity Sensor calibrated to activate LED and Alarm system at desired humidity environment • Ensure Motors are calibrated to Radio Transmitter	• Jaelynn & Hulya • Muhammad & Hulya • Muhammad & Jaelynn
April 13th - April 19th	• Order final round of PCB • Solder the new PCB • Debug PCB • Work on IMU and sensors connecting to ESP32	• Jaelynn • All • Hulya & Jaelynn • Hulya & Jaelynn • Muhammad
April 20th - April 26th	• PCB flashing and integration • Final debugging • System integration • Ensure entire system	• All • All • All • All

	achieves high level requirements	
April 27th - May 3rd	Mock Demo	All
May 4th - May 10th	Final Presentation Final Paper	All

5. Conclusion

5.1 Accomplishments

Our project has achieved several key milestones, marking significant progress in both hardware and software integration. One of the major successes was successfully flashing the ESP32 directly on our custom PCB, which validated our hardware design. We also integrated Betaflight, which allowed us to leverage its advanced flight control features and tuning capabilities, greatly enhancing our development efficiency. The use of breakout boards for the IMU and humidity sensor ensured modularity and ease of debugging, giving us flexibility in sensor placement and data acquisition during early testing.

On the control side, we achieved full functionality of the motors and our ESC, which are now reliably receiving commands from the radio transmitter via the ESP32. This demonstrates robust signal processing and control pipeline execution from user input to mechanical actuation. The system's responsiveness to radio commands confirms successful UART communication and software coordination across multiple subsystems. Collectively, these achievements show that our hardware-software approach is effective and that the foundation is firmly in place for further development and testing of more complex autonomous behaviors.

5.2 Uncertainties

Despite our many successes, the project faced several notable challenges that highlighted areas for improvement. One major failure was within the camera subsystem: although the camera module was functional during initial tests, after soldering it onto the PCB, a critical design mistake was discovered — the 5V trace and ground pin were reversed, resulting in a short and rendering the camera inoperative. This issue might have been avoided by designing the PCB with a dedicated ribbon head connector, eliminating the need to rely on a separate camera module and reducing the risk of manual soldering errors. Another challenge arose with the integration of the IMU and humidity sensor into Betaflight. We found that Betaflight supports only a limited selection of IMUs and sensors, restricting our ability to bring all sensor data into the flight control system. Additionally, the use of small, QFN-packaged ICs made debugging more difficult, suggesting that future iterations should consider non-QFN components for ease of access during troubleshooting. Finally, full PCB integration proved problematic; although flashing the ESP32 worked, integration with the IMU and humidity sensor failed due to shorts between the small IC pins and unresolved layout issues. More time spent on breadboard prototyping before finalizing the

PCB layout could have provided critical insights, allowing us to catch and address these integration challenges earlier in the prototyping cycle.

5.3 Ethical considerations

With open-source hardware and software, we have a responsibility to ensure that our final product should be available for the public to continue the development of beginner friendly drones in accordance with IEEE 7.8.I.2 [6]. Moreover, with our criteria, we want to ensure that the user has honest performance reports with the appropriate tolerancing to upkeep safety and the integrity of our project.

Additionally, we have a responsibility to address ethical concerns regarding military applications and privacy violations. While our drone is meant for civilian recreational use, we acknowledge the potential misuse of our FPV drone in illegal and unauthorized aerial reconnaissance and surveillance. To prevent such abuse, we provide strict guidelines to any users and implement altitude restrictions and geofencing capabilities through our intended drone flight capability range. The limited range of our drone will allow us to mitigate any risk of potential military or surveillance abuse. Through this, we uphold IEEE 7.8.I.1 by prioritizing public welfare to ensure responsible technological use [6].

5.4 Safety and Safety Procedures

These and other safety considerations are based around 7.8.I.1 in IEEE's Code of Ethics to ensure the safety and use ethical design practices [6]. Moreover, by recognizing that the users are newer, in accordance with IEEE 7.8.I.6 [6], considerations were made into the design to limit the amount of training experience required to fly the drone and diagnose issues that occur. The alarm system and the built-in protection for the parts we want to order heavily contribute to our goal of protecting our users. Ensuring user and environmental safety is a top priority when operating an FPV drone, in alignment with IEEE Code of Ethics. Key procedures include checks of batteries, wiring, and propellers, flying only in authorized areas under safe weather conditions, and maintaining visual line of sight for FPV. Electrical safety involves proper battery handling, charging practices, and thermal management. Post-flight, users should disconnect batteries, inspect components, and review logs. For FAA/FCC regulations, privacy considerations, and community respect further support responsible and safe drone use.

5.4 Future Work

To continue this project, further research must be done into sensor integration with the FC firmware. IMU integration can be done through either the usage of a different compatible IMU or experimenting with other open-source firmware. Some potential operating firmware to use may be with Flix, Madflight, or BLHeli. Additionally, in terms of design considerations, removing the JTAG can help simplify the PCB design and open up RX/TX pins on the ESP32. Similarly, for the camera subsystem, using the camera ribbon directly instead of the entire camera module will allow for further PCB design simplicity and subsystem safety.

References

[1] CurryKitten. “The History of FPV.” CurryKitten. https://www.currykitten.co.uk/the-history-of-fpv/ (accessed Mar. 5, 2025).

[2] “How many drones have you crashed since you started flying?” Reddit. https://www.reddit.com/r/fpv/comments/12qhckv/how_many_drones_have_you_crashed_since_you (accessed Mar. 5, 2025).

[3] Le, S. Supporting, I. 11, G. Wi-Fi, and Le), “ESP32-S3 Series Datasheet 2.4 GHz Wi-Fi + Bluetooth Including.” Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf

[5] ECE Illinois. “Salary Averages.” University of Illinois Urbana-Champaign. https://ece.illinois.edu/admissions/why-ece/salary-averages (accessed Mar. 5, 2025).

[6] IEEE. “IEEE Policies: Section 7 – Statement of Policy.” IEEE. https://www.ieee.org/about/corporate/governance/p7-8.html (accessed Mar. 5, 2025).

[7] Max, et al. “Review - EMAX RS2205S 2300KV Motors.” Oscar Liang, 24 Apr. 2017, https://oscarliang.com/emax-rs2205s-2300kv-motors/

Appendix A Requirement and Verification Table

Table 1 System Requirements and Verifications

Requirements	Verifications
The ESP32 must be able to take in data from the sensor subsystem i.e. the speed/orientation of the drone and the humidity sensor and send the alert signal to the alarm subsystem if humidity is too high.	• Use a multimeter to ensure that the pin on the microcontroller will be an input to the alert subsystem and send a voltage>1 V to be accepted as a logical 1. • Connect the micontroller's sensor ports as outputs of the microcontroller to the computer's terminal and send the IMU data to the terminal through UART. See the data changes as we move the IMU's speed/orientation.
The ESP32 must be able to take in the RF data from the Radio Receiver on how to move the drone through UART.	• Set up a UART port on the ESP32 that prints incoming CRSF data to the computer's monitor using Arduino Framework. • Check on BetaFlight's Ports Tab in Betaflight Configurator and ensure the UART connected to ESP32 has Serial RX enabled and set the receiver protocol to CRSF under the "Receiver" tab.
The ESP32 must be able to receive data from the ESCs on the speed of the motors.	• In Betaflight Configurator: Go to Configuration Tab → ESC/Motor Features and enable Bi-directional DShot. In the Ports Tab, enable ESC Telemetry (RX) on the appropriate UART that's set to receive the data from the ESCs. Go to the Motors Tab in Betaflight and verify that ESC telemetry values (RPM, voltage, temperature) update

	when motors are running.
The ESP32 must be able to send the digitized camera data to the user's phone through Wifi.	• In the code to connect the ESP32 to Wifi, have print statements that print out if Wifi is connected and print the IP Address that the ESP is connected to. • Ping the IP address on a nearby computer. • Open a web browser on a phone and type in the provide IP address to stream the data on the phone.
The radio receiver must be able to get the data from the radio transmitter (determines direction of movement) within the range of a 500Hz- 1000Hz receive refresh range from the user subsystem.	• Check the LED on the radio receiver and ensure it's a solid light, indicating a stable connection to the radio transmitter. • In Betaflight Configurator, go to the Receiver tab to check the receiver's status and make sure that the receiver is enabled and properly bound to the transmitter and that the correct ExpressLRS protocol is selected under the Ports tab in Betaflight. • Go to the Receiver tab in BetaFlight and observe the real time receiver's input on the screen. Test by moving the radio transmitter's stick and seeing how quickly that data appears on screen at ranges of 1fr, 4ft, and 10ft away.
The radio receiver must be able to send the received RF data into the ESP32 through UART.	Verify the same way that we verify the ESP32 is inputting that data.

Voltage Regulator 5V to 3.3V must provide clean 3.3V to power microcontroller and sensor subsystem (+/- .1V)	• Connect a multimeter to the input pin of the 5-3.3 regulator and make sure the input is 5V +/- .1V. • Connect a multimeter to the output pin of the 5-3.3 regulator and make sure the output is 3.3V +/- .1V.

Humidity Detector must be able to accurately represent the relative humidity(RH) percentage in the air around the PCB by taking 6 measurements/second of the RH and saving this value into a 16-bit register.	• Connect the humidity sensor to the ESP32 through I2C protocol: Sensor's Vdd is connected to 3.3V, Sensor's GND is connected to ESP32's GND, Sensor's SDA is connected to ESP32's SDA (Data Line) and the sensor's SCL is connected to the ESP32's SCL (Clock Line) • In Arduino IDE, download the SHT3x library (our sensor's library) and write code to receive the temperature and humidity measurements from the sensor that's uploaded to the ESP32. Then check the Arduino Serial Monitor to see these measurements and verify their accuracy in given environments.
The humidity sensor must be able to communicate relative humidity% to the Control sub-system and send the ALERT interrupt to the ESP32 if RH is above 90%.	• Connect the ALERT pin of the sensor to a GPIO pin on the ESP32 and configure this GPIO pin as an interrupt input to detect when the ALERT pin is triggered in Arduino IDE. • Simulate a high humidity environment and ensure that the

	ALERT interrupt goes off when RH is 90%.
The IMU's Accelerometer + Gyroscope must be able to accurately represent the acceleration of the physical drone by taking measurements in the X,Y,Z axis with maximum measurable acceleration before saturation set to +16g (g equals about 9.81 m/s^2)	• Write Arduino code to configure the accelerometer to +16g and print out the measured X,Y,Z values in terms of gravity. Move the IMU around at different forces to simulate different g's and verify that the raw data should be in the range of -32768 to 32767 for +16g. • Write Arduino code to configure the gyroscope to 2000m/s and print its measurements to the serial monitor. Ensure when it's flat on a surface the X,Y,Z values are 0. Rotate the IMU on only the x-axis and ensure only X values change, etc.
Must be able to accurately represent the pitch, yaw, and roll of the physical drone throughout the full scale 360° range within an error of ±15% to the Microcontroller through I2C protocol at 400 kHz.	• Betaflight provides a real-time view of the pitch, yaw, and roll of the drone in the "Flight Data" tab, under the "Angle" indicator. To verify this data: gently move the IMU in all directions (pitch up, pitch down, roll left, roll right, yaw left, yaw right) and observe the changes in the 3D model or the angle indicator in the Betaflight Configurator and verify that the pitch, roll, and yaw values match the expected orientation based on our physical movements. • To test error margin, rotate the drone to known angles (e.g., 0°, 90°, 180°, 270° for each axis), and compare the Betaflight displayed values to the expected angles. Ensure the displayed angles stay

	within the required ±15% margin of error.

5V Passive Buzzer must be able to receive high DC input from ESP32 Alert in response to the humidity sensor to create a sound of 100 dB audible within 100 ft.	● Apply a 5V high signal directly to the buzzer, using a standalone multimeter and voltage source. ● Use a calibrated sound level meter to measure the buzzer's dB output to 5V at 3ft, 10ft, 100ft distances ● Measure voltage from designated GPIO pin during ESP32 Alert to ensure 3.3V-5V ● Ensure Alarm buzzes from ESP32 Alert once all connected

Radio Transmitter must be able to wirelessly send data to the Radio Receiver in accordance with physical joystick inputs under 100ms latency.	● In BetaFlight, under the Receiver section, set the Receiver Mode to Serial-based Receiver and choose CRSF for the receiver protocol and set the Serial Receiver Provider to CRSF. Also, set the Serial Baud Rate to 400,000. ● In the Ports Tab on BetaFlight, find the UART that the ELRS receiver is connected to and enable Serial RX for the appropriate UART port. In the Receiver tab in Betaflight Configurator while moving the joystick of the LiteRadio2 transmitter, observe the channels (e.g., Throttle, Roll, Pitch, Yaw) and make sure that the values change in real-time as we move the joysticks. This confirms that the receiver is receiving signals from

	the transmitter and correctly mapping them to the flight controller. ● Measure the seconds between the joystick movement and mapping onto the channel, ensuring it's consistently less than 100ms at different distances with range of the transmitter.
Smart phone must be able to receive live video stream from the ESP32's Wifi in 20+FPS quality and below 100ms latency.	● Make sure that the user's phone can connect to Wifi (any Wifi) and then specifically the Wifi of the ESP32. ● Open the video stream on our smartphone and start counting frames in a 10-second interval to ensure that the number of frames displayed in this interval equals or exceeds 200 frames (for 20 FPS). ● Setup an LED in front of the camera and flash the LED on and off every second. Start a stopwatch as soon as the LED flashes and observe the time it takes for the flash to appear on the smartphone screen. The latency is the time between the moment we initiate the flash and the moment we see it on the smartphone.

Electronic Speed Control must be able to receive control inputs from the ESP32 from the Radio Transmitter to send varying Dshot signals to speed up or slow down the motors/propellers.	● In Betaflight Configurator, manually adjust the throttle slider in the Motors tab (in the Motors tab set Master Switch to do this) to test each motor and verify that it is

	receiving and responding to the DShot signal. Make sure that the motor's speed increases or decreases as we vary the throttle signal in Betaflight, which will show the Dshot values should change accordingly. ● Disable Master Switch and use the radio transmitter to see how the motors will move and if the Dshot values are changing in the BetaFlight Configurator.
Quadcopter X-Frame must be able to house motors, propellers, PCB, and battery and must be able to move throughout the 3D plane according to motor controls.	● Take measurements of the motors, propellers, PCB, and battery to make sure that they will fit onto the frame. ● When moving the joysticks of the radio transmitter, make sure that the drone moves in a balanced manner in any X,Y,Z direction.