ECE445 FALL 2025

SENIOR DESIGN LABORATORY

FINAL REPORT

Omni-directional Aerial Vehicle

Team:

Ivan Ren - iren2@illinois.edu

Dhruv Satish - dsatish2@illinois.edu

Mahir Koseli - mkoseli2@illinois.edu

TA:

Jason Zhang - zekaiz2@illinois.edu

Abstract

This report details the motivations, design process, and eventual build of our group's ECE445 Senior Design project, an **Omnidirectional Drone**. The design is based on existing work from ETH Zurich, but implemented with a custom frame, electronics, and software. The system is an overactuated, 8 rotored vehicle with a motor configuration that allows for decoupled motion in each of the 6 DOF, originally based on a central STM system that handled motor speed calculations and control algorithms, interfacing with a 2.4 GHz ELRS remote controller, IMU, and our electronic speed controllers (ESCs) over PWM. Through the execution of our project, we were able to verify functionality of our mechanical subsystem, our IMU, radio transmission and receiving, off-the-shelf ESC functionality and power protection circuits. Our final experimental results demonstrated the integrated functionality of our frame, limited functionality of our BLDC motor drive system, and a rudimentary control scheme based off of IMU angular position.

1. Introduction **4**
1. 1 Problem 4
1.2 Solution 4
1.3 Block Diagram 6
1.4 High-level Requirements 6
2. Design **7**
2.1 Design Procedure 7
2.2 Design Details 8
2.21 Electrical Subsystem 8
2.22 Mechanical Subsystem 12
2.23 Flight Control + Telemetry 13
3. Verification 13
3.1 Electrical Subsystem 13
3.2 Mechanical Subsystem 14
3.3 Flight Control + Telemetry 15
4. Costs **16**
4.1 Labor Analysis 16
4.2 Cost Analysis/BOM 16
Appendix A **20**
A.1 Component Cost Tables 20
A.2 Requirement and Verification Tables 23
A.3 Additional PCB Images 27
A.4 Mechanical Design Images 28
A.5 Project Build Images 29
A.6 High Level Code 30
References **35**

1. Introduction

1. 1 Problem

The issue of aerial maneuvering has become an increasingly important consideration in the new age of drone deliveries, drone imaging, and necessity for automation in the fields of agriculture, construction, surveying, remote monitoring, and more. The current standard of drone technology remains limited to mostly quadcopters, a technology that has matured to enough of a degree to allow for complex directional motion, and extreme speed and stability. However, these vehicles have a notable issue of a lack of movement decoupling, with the translational and rotational motions being tied together. In a lot of speed-focused applications, this issue is trivial as most movement systems can compensate to move in 6DOF space by applying different amounts of power to different motor configurations. But in precision applications or in situations that require a certain orientation to be held, decoupling the rotational and translational degrees of motion allow for the drone to have unprecedented control. For example, in an omnicopter design by ETH Zurich, their demo of catching balls using a net showed impressive results, with the drone staying motionless midair while rotating to track the ball and catch it with a net (reference). Just considering a few simple scenarios, for precise filming, construction, or especially sensitive natural or urban areas, a drone with full control over its movement means the ability to hold an angle for a shot, to apply paints at all angles and move around objects through very tight spaces, or to survey wildlife or urban areas without interfering with the natural environments. In any situation not prioritizing speed or power, an omnicopter would provide significantly improved flexibility and control.

1.2 Solution

Our solution consists of three main components: build a robust motor drive system in an omnicopter configuration, designing and 3D printing a frame with the required orientation of motors, and creating the required controls and telemetry to move the drone in a stable manner. The motor drive system contained all required electronics to power and control the motors, including the ESCs, motors, current and voltage sensors, battery management system, and a central microcontroller that interfaces with the ESCs and remote controller. The system was

built to be modular, with each ESC and motor addition being its own module and being easily added to the overall electrical schematic to ensure flexibility with motor configuration, depending on power usage during testing. The frame of the omnicopter took on a 8 motor configuration and placed an emphasis on easy fabrication using quick prototyping methods like stereolithography (SLA) and fused deposition modeling (FDM) 3D printers, while also remaining lightweight and structurally sound. The communications and controls side read data from the remote controller and converted movement signals into different motor power combinations. The remote controller will be a simple dual-joystick system with additional channels being provided using auxiliary switches. This system also includes the inertial measurement unit required to track current orientation and balance the system once controls have been input, as well as the antennas required to communicate with the remote controller.

Our final experimental results demonstrated major subsystem functionality, verified through use of lab tools. Firstly, our physical frame achieved optimal motor orientation while staying within calculated weight limits and estimated bending stiffness requirements. Secondly, our motor drive system correctly handled input signals from our onboard MCU, with our power protection circuit providing the required 3.3V from our LIPO battery's 14.8V. Thirdly, our telemetry and controls were able to interface with the IMU, remote controller, and MCU to produce a differing motor thrust output depending on angular position and user input. Though our drone was unable to hover, we believe we were able to demonstrate overall feasibility.

1.3 Block Diagram

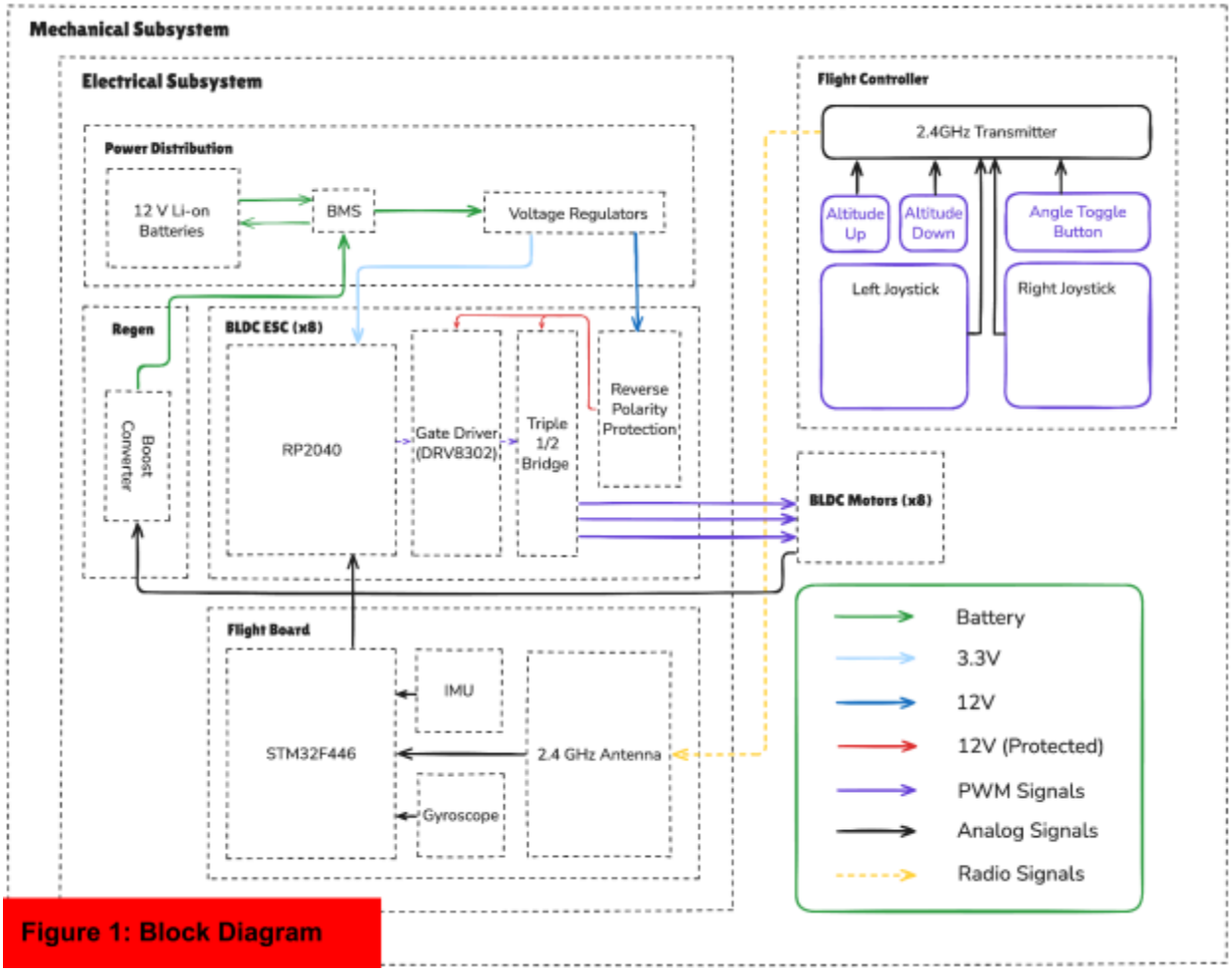


Figure 1: Block Diagram

1.4 High-level Requirements

- ☐ Individual ESC able to ramp up and down BLDC motor through PWM control
- ☐ Physical design with optimal motor orientation and mounted electronics/motors
- ☐ Move (1) vertically upwards **1 meter**, hold (2) **a fixed altitude ± 0.25 meter**

2. Design

2.1 Design Procedure

Due to the scope of our project, we had several reference designs that we originally planned to base our design off. In specific, VESC and DeepBlueEmbedded both had reference designs for robust ESCs. We planned to also develop a custom flight controller board to integrate our IMU, 2.4 GHz receiver, and main microcontroller. The optimization problem for motor orientation has already been solved in the official publication by ETH Zurich. There are also existing CAD files for other designs. We originally planned to use an existing build, including their carbon fiber dimensions to create our frame. Fabrication-wise, we planned to use primarily SLA 3D printers (resin printers) due to improved resolution and material selection. Robust controls are required to operate and balance the drone in each of the 6 DOF. With 8 motors to operate, the control concept comes down to an 8 by 6 matrix, where each of the motors have an output that produces a known translational acceleration (x, y, z) and a known rotational acceleration (alpha, beta, gamma) depending on its orientation. Filtered IMU data and CRSF data is transmitted over UART to our STM chip, allowing for our drone to balance itself.

However, due to time constraints, we made several changes to our original design procedure to simplify and optimize for our high level requirements. In specific, due to multiple revisions for trace width meant that we were unable to implement a flight board, deciding to instead use several breakout boards for the sensors we chose and then connecting them to a Nucleo Board. When verifying our final ESC, 2 of our high-side MOSFETs did not function, making it impossible for us to drive a BLDC motor using our board. We were able to demonstrate half-functionality using a drone battery and several LEDs. We used off-the-shelf 4-in-1 ESCs instead as a fallback. To reduce fabrication cost and time, as well as due to the lack of materials online matching the exact dimensions required, our mechanical lead decided to build a new frame from the ground up using comparable parts. Calculations were done to ensure the alternatives had similar or better stiffness values, weights, and that costs/availability were reasonable. Due to resolution and curing issues, we were unable to use the SLA printers for several of our parts and opted to use FDM 3D printers at a high-resolution layer height instead. For our controls, we were unable to execute a full 6 DOF balancing control schema; to

optimize for our high-level requirements, we simplified our system down to an 8 x 4 matrix that only focused on linear movement. This introduces issues with translational drift, but our time constraints meant we focused on hovering first. And instead of implementing a sensor fusion algorithm to filter raw IMU data using an Extended Kalman filter or Madgwick's Algorithm, we bought an IMU with an onboard M0 processor for built-in sensor fusion.

2.2 Design Details

2.21 Electrical Subsystem

The electrical subsystem will contain all required electronics to power and control the motors, including the ESCs, motors, current and voltage sensors, battery management system, and a central microcontroller that interfaces with the ESCs and remote controller. Within the motor drive system, the power management system handles power distribution and contains the required safety measures to protect against back energy from the motors. In specific, each custom ESC has:

- 6 MOSFETs in a triple half-bridge configuration connected to a main gate driver
- A sensor-less back EMF (BEMF) and phase voltage measurement system
- 3 comparators connected to each phase voltage and the virtual neutral point for zero crossing detection (ZCD)
- A reverse polarity protection unit that operates using a power management chip
- TVS diode clamp
- Large parallel capacitor bank
- Buck converter to convert to 3.3V output for digital electronics
- Battery state monitoring

Calculations

In order to provide the thrust required, the motors we decided to use are the FLASH HOBBY Arthur 2207.5 Outrunner Brushless Motor 1900KV and Nihewo 4S Lipo Battery 14.8V 2200mAh RC Battery Soft Case 100C with XT60 Plug for FPV Drone RC Quadcopter Helicopter Airplane Car Truck Boat RC Models. Based on the datasheet of the motor, each motor can provide a thrust of 427g at 10A, using 5cm rotors. When multiplying the individual

thrust with the insphere radius, we get a max symmetrical thrust of 427 * 2.3 = 981g at about half thrust, which is our maximum takeoff weight.

We plan to use a 4s LiPo battery for our design and by using the formula P = IV, this comes out to a maximum current draw of 155A across all 8 motors or 20A per motor. The maximum current draw of a LiPo battery is given by $I_{max} = C_{rating} * Energy$. For our design, we're planning on using a 2200 mAh battery rated for 100C has I_{max} = 100C * 2200 mA = 220 A which provides more than the required maximum current draw. At an average hovering draw rate of 10A per motor, the total draw is 80 A, which can be supported for 2.2/80 = 0.028 hours = 1.65 minutes, enough to demonstrate hovering ability.

Schematics

The ESC schematic can be broken into a few sections: Interfacing, MOSFET Bridge, MCU, and Power. The Interfacing circuit shown below consists of the BEMF sensing resistor network placed in parallel with each 3-phase BLDC motor, the gate driver, a capacitor bank, and sensors for phase voltage/phase current as well as total current draw.

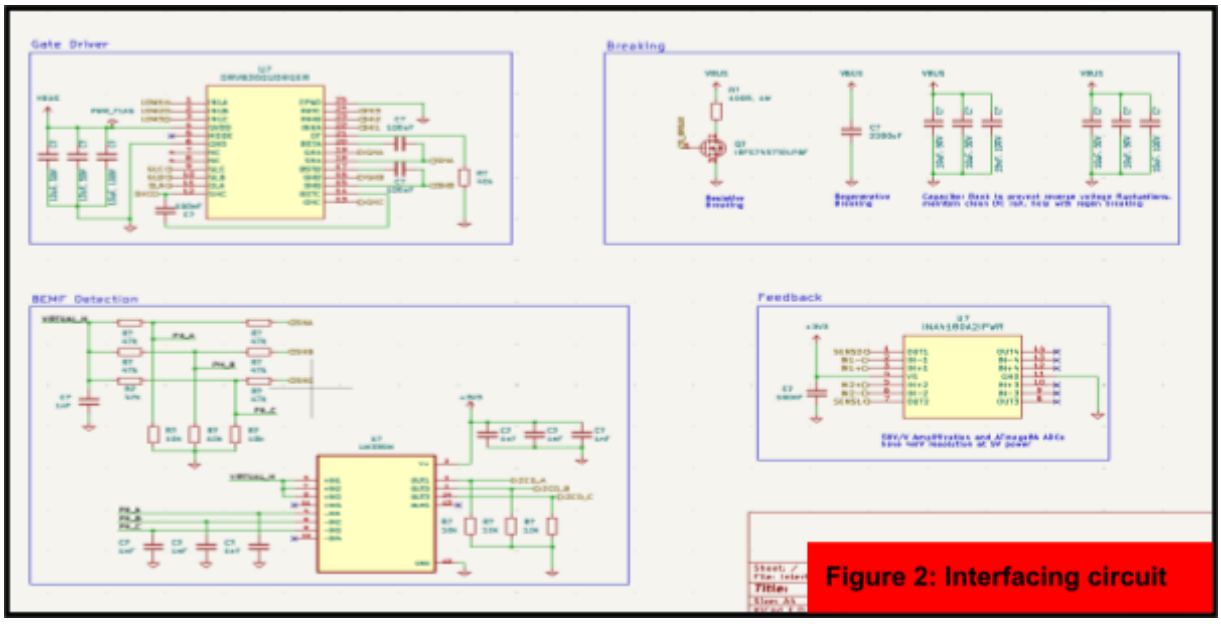


Figure 2: Interfacing circuit

The MOSFET Bridge setup consists of several high current, high voltage MOSFETs setup in a triple half-bridge configuration, with each half-bridge powering one of the phases for one BLDC motor.

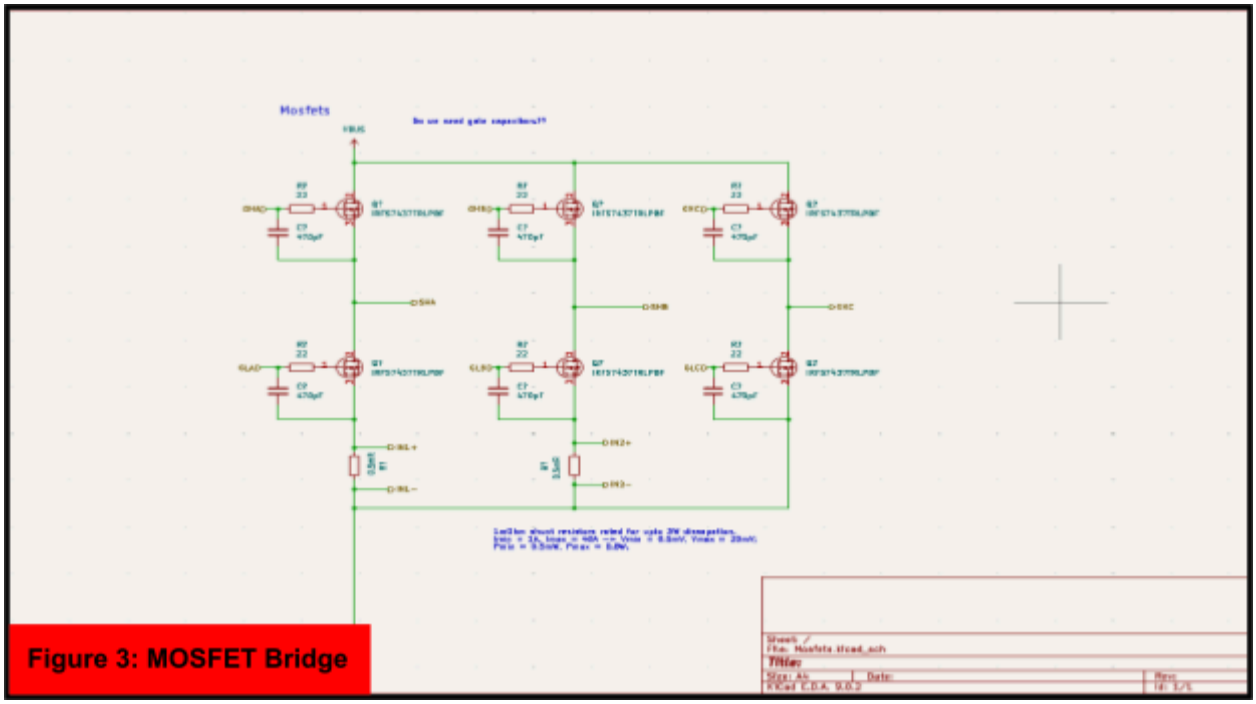


Figure 3: MOSFET Bridge

The MCU schematic consists of the RP2040 microcontroller with all necessary pull-down resistors, capacitors, GPIO connections, USB connectors, and flash memory.

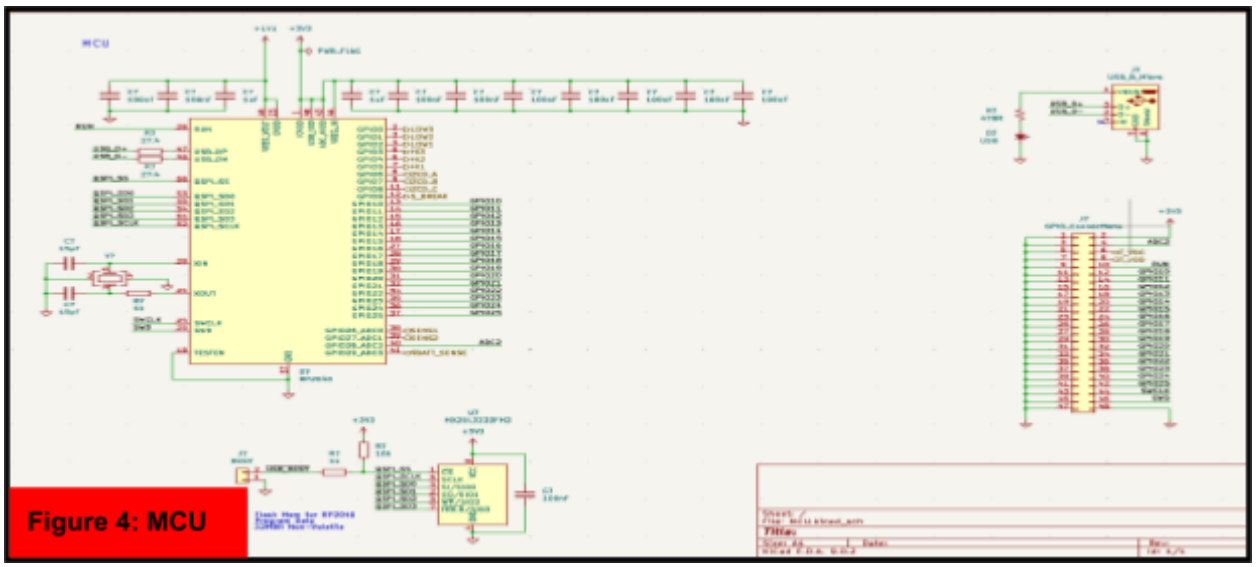
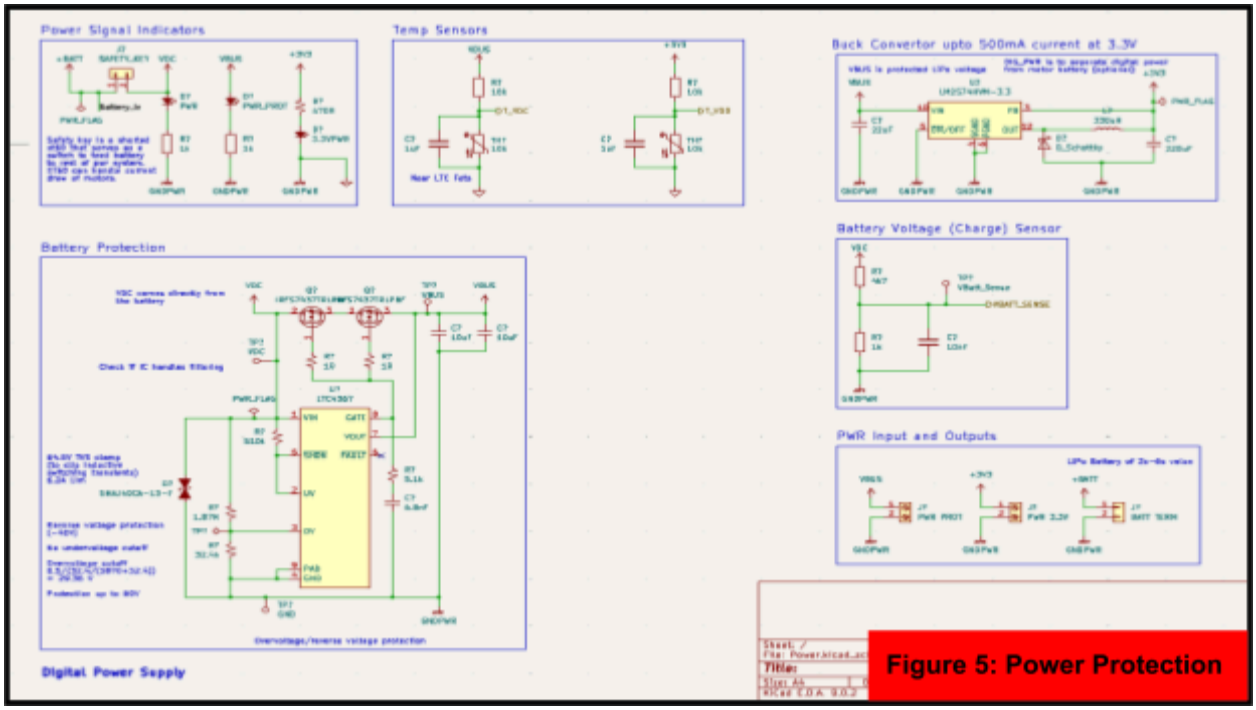


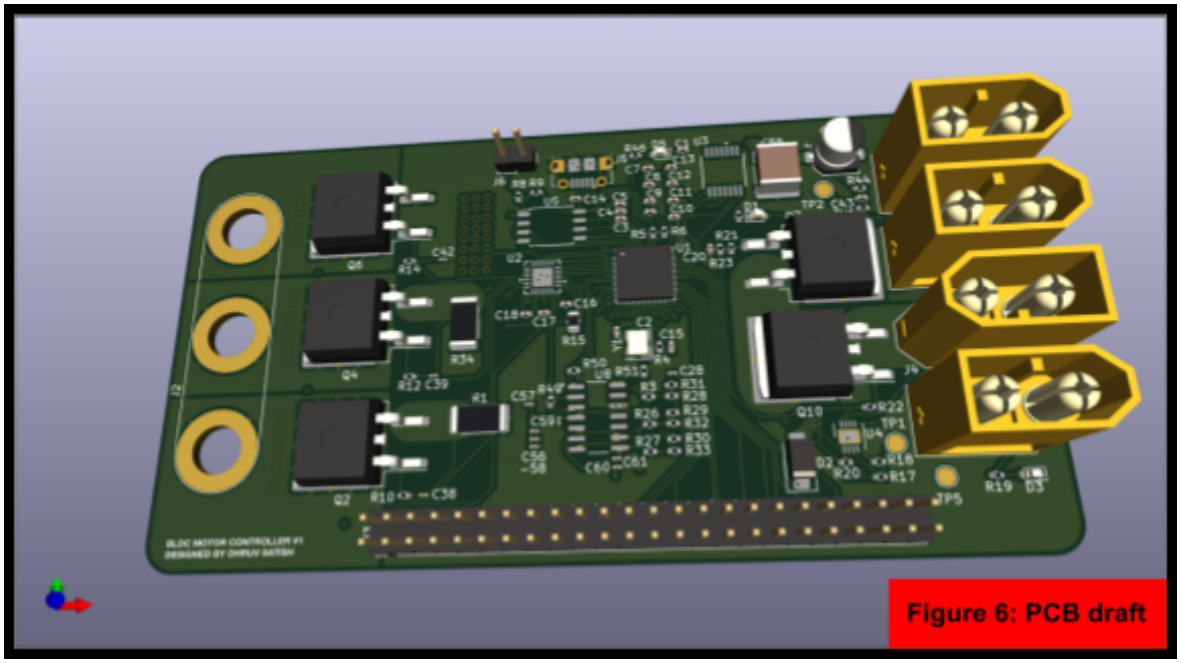
Figure 4: MCU

The Power schematic includes a buck converter, a battery protection and battery sensing.



PCB

Our final PCB implemented all of the subsystem requirements on a 4-layer design, with ground and power planes for heat dissipation and high-current handling. Extra PCB details included in the appendix.



2.22 Mechanical Subsystem

The overall mechanical design focuses on the design of the frame and configurations of the motors. The frame of the omnicopter took on a 8 motor configuration for maximum stability and due to existing online designs. We placed an emphasis on minimum weight and maximum strength while maintaining easy fabrication through quick prototyping methods like 3D printers. The frame can be split into 3 major parts:

- A central hub containing the control electronics and electrical subsystem.
- Mounting holes for all PCBs and brackets to safely hold high-power electronics
- 8 arms to hold each motor at the optimal orientation
- An external skeleton connecting each arm and providing structural rigidity to the overall frame.

Calculations

This weight needs to include the battery, motors, frame and electronics. The heaviest of these components are going to be the motors and batteries. The battery weight for our specific battery was 226g and total motor weight 36.3 * 8 = 290.4g. As such, we have a total electronics and frame budget of 464g.

For our frame, we decided on cube edges of 300 mm, with face diagonals of $\sqrt{(300^2) + (300^2)} \cong 420$ mm, with some extra gaps included for space to place the joints.

The space diagonals needed dimensions of $\sqrt{(300^2) + (300^2) + (300^2)} = 511$ mm; this had to be split into 2 arms for each motor. Leaving space for the interior hub, we decided on space diagonal dual arms of 200 mm. We were able to source the following diameters and dimensions for the final carbon tubes, verifying the construction using a CAD model (included in appendix):

- 8x 420mm (5 mm ID 3 mm OD)
- 12x 300 mm edges (5 mm ID 3mm OD)
- 16x 200 mm space diagonals (3 mm solid)

This resulted in a total estimated weight = 154.4 + 44 = 198.4 g < 250 g allocated. Since we were using a different set of carbon dimensions for the arms than the existing designs, we did some quick bending stiffness estimations. Stiffness estimates can be approximated as:

- $I_{total} = I_{center} + Ad^2 = 3.97 + 44.1786 = 48.2$ mm
    - $2 \times 48.15 \approx 96.3$mm, 4 times as stiff as original design

2.23 Flight Control + Telemetry

The controls and communications side took input from the remote controller and converted to a set of 8 PWM signals detailing the magnitude and direction of the force we want to apply to the overall drone system. The main microcontroller on board received said signal through a 2.4 GHz frequency band on an open source, drone-optimized communication system called ExpressLRS (ELRS), with a specific protocol called Crossfire (CRSF) that encodes the channel values. We wrote our own custom firmware to decode the CRSF signals, read motion data and then output PWMs for each individual motor.

For the controls, we created a cascaded PD controller that kept our angular orientation constant using an attitude controller and provided movement in one linear axis using an altitude controller, in line with our high-level requirements. Each motor's contribution to the 6 DOFs was calculated and then written as an 8 x 6 matrix, which we could solve given our current thrust state and our desired state to determine required motor thrust outputs. Our high level code is included in the appendix. The parts involved in this subsystem are as follows:

- 9 axis IMU to track translational motion and rotational motion
- 2.4 GHz antenna and receiver
- 2.4 GHz remote controller with least 8 channels of input
- Main STM32F446 microcontroller

3. Verification

3.1 Electrical Subsystem

In order to verify each unit of the electrical subsystem, we approached the system in 2 directions: our custom PCB, and our fallback plan utilizing off-the-shelf components. Our

custom PCB was tested using Test Points during the fabrication, and also continuity testing using a multimeter to confirm that each component was correctly soldered and connected to the pads. Afterwards, a power supply was used in conjunction to a multimeter to determine the output voltage of our buck converter. Finally, a function generator and oscilloscope was used to test different input PWM signals and confirm MOSFET outputs. Unfortunately, our ESC failed testing as two high-side MOSFETs did not function, leading us to trace the issue back to the motor driver, which wasn't outputting the expected signals. We turned to the backup plan, which was using off-the-shelf 4-in-1 ESCs with the required current output and PWM control.

We approached ESC testing similarly, Testing motors using lab power supply and Nucleo board to verify functionality of off-the-shelf ESCs. We applied different current limits to the power supply and tested each motor and ESC channel to confirm functionality. We used the oscilloscope to verify the PWM output from the Nucleo board was as expected. Once we could confirm that each motor could be individually ramped up and down from the Nucleo board, we began assembly. Unfortunately, during our first test run of our system all connected, something went terribly wrong and our Nucleo board started smoking when the battery terminals were connected. Mahir's laptop also was fried, not booting up anymore. We ported our project over to Arduino and carefully retested each component, finding an unexpected connection to the battery terminals through our MCU. Using one of the 4-in-1 ESCs in our redemption demo, we were able to demonstrate partial functionality.

3.2 Mechanical Subsystem

First successful skeleton frame testing using our custom joints, we ran into several tolerance issues. Our carbon rods measured out to almost exactly 3.00 mm using Mitutoyo Calipers (0.01 mm precision) but our joints took multiple iterations to print due to deformations and imperfect surfaces. We weighed this portion of the frame with a final value of ~210g, within the expected range of 198g to 250g, most likely due to the glue and print scaling (for tolerance).

We began assembling our main frame, ensuring all components fit properly on the body of the drone. We would eventually switch from this frame to a version printed by a Bambu Labs FDM printer due to issues with the alignment of the top and bottom plates that connected to the carbon skeleton. Small adjustments were made due to printing constraints with FDM

printers. Since SLA printers print with 100% infill, this part was heavier than expected at 67 grams, in comparison to our FDM part printed at 60% infill which weighed in at 37 grams.

For our previous breadboard demos, we had confirmed functionality of the IMU and receiver modules, so we began assembly of the drone on a newly printed frame. This included our custom PCB as our power protection unit attached to the battery. The final weight of all parts came out to ~ 952 grams, which is cutting close to our total 10A motor thrust value of 982 grams.

3.3 Flight Control + Telemetry

We approached this subsystem in two parts. First, we needed to verify our telemetry requirements, including our IMU, receiver, transmitter and all associated drivers. We did this in a few steps, beginning with the IMU.

For our first breadboard demo, we verified the performance of our IMU with our MCU, confirming through the Serial monitor that we could receive data over UART at the required rate of 100 Hz for our control loop.

For our second breadboard demo, we worked on creating a driver capable of receiving UART packets from the radio transmitter using the CRSF protocol, triggering an interrupt callback to store the values into a buffer, and then decoding the packet into the 12 channel outputs from our remote controller. We verified this also over the Serial monitor, confirming the correct output and response from our remote receiver. In our last integration step, we confirmed that we could receive IMU data at the required rate to update our current state and also recalculate our motor thrusts whenever we received a new desired thrust output from the remote controller.

Unfortunately, we ran out of time before we were able to test the cascaded controller Mahir implemented, instead opting for a simple control scheme that directly adjusted motor thrusts depending on IMU and remote controller inputs to demonstrate partial functionality.

4. Costs

4.1 Labor Analysis

We need to first consider labor costs by roughly estimating our salary according to our education and qualifications. Our project is split into 3 main portions: circuits design and electronics testing, mechanical design and fabrication, and controls/software development. Each of our roles will require a similar amount of time, considering a timeline of around 10 weeks (including planning and circuit design during the summer), we're assuming a total of 100 hours, at 10 hours per week. Considering each of these main roles, as well as our relative levels of experience, we used the following jobs titles to estimate our total cost: **$22,530**

Job Title (New Grad)	**Hourly Wage**	**Hours Worked**	**Total Cost**
Hardware Engineer	$58/hour	10 hours/week * 8 weeks 20 hours/week * 2 = 120	$6960
Mechanical Engineer + Integration	$51/hour	10 hours/week * 8 weeks + 20 hours/week * 2 = 120	$6120
Software Engineer	$63/hour	150 hours	$9450

4.2 Cost Analysis/BOM

The second portion of our analysis focuses on the actual estimated cost of acquiring all the parts for the drone. Several parts will be purchased in larger quantities than strictly required, however this is due to the soldering process and high probability that some boards run into some issues after soldering. The PCBs are cheaper to buy in a larger quantity, so the price is somewhat amortized by buying a larger set, while also providing us extras in case of mistakes. It is also important to mention the reasoning behind purchasing some redundant components

like the 4-in-1 ESCs, this is as a last resort in case of issues with our own design. Considering the importance of the ESC system, it is absolutely necessary to have a working set of ESCs to drive the 8 BLDC motors. The total cost based off of the table included in the appendix:

$643.59

5. Conclusions

5.1 Successes

We were able to write a custom driver to receive and decode data over UART for our specific ExpressLRS radio receiver/transmitter system that transmits channel data through the CRSF protocol. We also built a completely custom mechanical design based on available carbon dimensions, within stiffness and weight requirements. For our controls, we wrote a custom cascaded PD controller for linear translational movement utilizing IMU data and current state. And finally, we designed and laid out a custom 4-layer PCB for high-power BLDC applications, verifying robust power protection functionality and half-functionality of the triple half-bridge driver.

5.2 Failures and What We Learned

We ran into several project timing issues meant a rushed final product and massive integration issues when faced with unexpected setbacks i.e. Dhruv's flight was delayed 4 days, IMU model incorrect, frying Nucleo board, several motor delivery delays. Due to rushing, testing and verifications at each step were not as comprehensive as detailed in the original schedule leading to fried boards. We also only produced and tested one version of the custom PCB due to several revisions, leading to only partial functionality of the final board without time to revise. Overall, our project management needs improvement, including securing dedicated facilities meant for drone flight and buying parts sooner in case of delays.

The scope of our project was massive. We were aware of the level of challenge to some degree, which is why Dhruv and Ivan began planning and designing the schematic for our ESCs over the summer. However, during the school semester, it became clear our execution did not match the scope of our project, regardless of what reasons led to this. At some points

throughout the semester, we began to realize our performance was differing from the schedule, and though we worked late nights to catch up, this pattern already made it clear that our pacing was unsustainable. For any project of some complexity, proper delegation and ownership over deadlines is definitely vital and small, consistent steps are significantly better than attempting large scale changes. Making note of all deadlines and making resilient back up plans to account for delays is extremely important since things will go wrong when you don't expect. We are ultimately still proud of the work we were able to finish in the time we had. The root to our final issues was due to rushing and time constraints. We believe that better project management would've fixed the root of our issues, though a few other specifics also stand out to us.

1. Better parallelize our testing efforts, buy standalone sensors and then write drivers to integrate each sensor system with our MCU while testing our electronics and designing mechanical systems.
2. Simulate our systems – a lot of what we did depended on our final product, better simulation would have given us a chance to test our control systems.
3. Improved electronics testing by approaching each integration step with the proper measurement tools before hooking up any electronics, which would have prevented us from frying our board.

5.3 Future Plans

Overall, we were very impressed by the packaging and efficient design for the off-the-shelf 4-in-1 ESCs. Dhruv and Ivan would like to learn from the production quality of a commercial product by redesigning their ESCs to perform at a similar level while reducing the overbuiltness of some parts. The current frame can be improved on to improve manufacturability and structural rigidity; the current skeleton performs well, but weight reductions can be made without sacrificing strength. Mahir and Ivan would like to use the existing frame and motors, fix up the current parts and attempt to test Mahir's cascading control system/make tweaks to enable full omnidirectional movement as a controls project.

5.4 Ethical and Safety Considerations

Our omni-directional drone features high-speed rotating propellers and a Li-ion battery pack, which pose risks of injury, fire, or electrical failure. We implemented a kill switch just in case, though because our drone never used our full battery power, the power supply limited power draw. We also used preventative measures including reverse polarity protection, fuses, TVS diodes, and capacitor banks to absorb back-EMF from motors. Charging was supervised using manufacturer-recommended chargers, and the frame was designed for maximum air flow to prevent overheating.

Our project uses a 2.4 GHz wireless remote controller and telemetry module to transmit and receive flight commands. These components fall under unlicensed operation limits defined by the FCC Part 15 Subpart C rules for intentional radiators in the 2.4 GHz ISM band. However, by using low-power consumer-grade devices designed for hobbyist and research use, the expected RF exposure is well below harmful thresholds.

When working with 3D printing, solder, and carbon fiber, we took specific safeguards that need to be taken to avoid accidental ingestion of harmful materials. Additively manufactured parts, such as those made from nylon or resin, can release fine particulates during post-processing, therefore sanding or trimming was performed with gloves and masks in accordance with UIUC Laboratory Safety Guidelines for Additive Manufacturing. Additionally, certain 3D filaments like ABS will be avoided due their release of toxic fumes when printing; instead, alternatives like PETG or proprietary material blends that are certified safe will be utilized in our SLA and FDM printers. Soldering exposes operators to flux fumes and molten metal, therefore all soldering was performed in well-ventilated areas with smoke absorbers.

Appendix A

A.1 Component Cost Tables

ICs	Quantity	Cost
RP2040CT	12	$1.22 x 12 = $14.64
MX25L3233FM2	12	$0.48 x 12 = $5.76
DRV8300UDRGER	12	$1.44 x 12 = $17.28
INA4180A2IPWR	12	$1.00 x 12 = $12.00
IRFS7437TRLPBF	96	$0.4911 × 96 = $47.07
LM2574HVM-3.3	12	$5.31 × 12 = $63.72
LTC4367	12	$7.07 × 12 = $84.84

Capacitors	Quantity	Cost
0201, 10nF, 25V	12	$0.10 x 12 = $1.20
0201, 1nF, 25V	84	$0.10 x 84 = $8.40
0201, 470pF, 25V	72	$0.10 x 72 = $7.20
0402, 100nF, 25V	168	$0.10 x 168 = $16.80

0402, 15pF, 50V	24	$0.20 x 24 = $4.80
0402, 1µF, 6.3V	48	$0.11 x 48 = $5.28
0402, 6.8nF, 50V	12	$0.15 x 12 = $1.80
0805, 10µF, 50V	72	$0.32 x 72 = $23.04
0805, 10µF, 25V	24	$0.11 x 24 = $2.64
0805, 22µF, 10V	12	$0.32 x 12 = $3.84
0805, 220µF, 25V	12	$0.91 x 12 = $10.92
0805, 2200µF, 25V	12	$1.97 x 12 = $23.64

Resistors	**Quantity**	**Cost**
0402, 47kΩ, 0.1W	72	$0.10 x 72 = $7.20
0402, 1kΩ, 0.1W	60	$0.10 x 60 = $6.00
0402, 27.4Ω, 0.1W	24	$0.10 x 24 = $2.40
0402, 22Ω, 0.1W	72	$0.10 x 72 = $7.20
0402, 10kΩ, 0.1W	132	$0.10 x 132 = $13.20
0402, 1.87MΩ, 0.1W	12	$0.10 x 12 = $1.20
0402, 32.4kΩ, 0.1W	12	$0.10 x 12 = $1.20

0402, 470Ω, 0.1W	24	$0.10 x 24 = $2.40
0402, 510kΩ, 0.1W	12	$0.10 x 12 = $1.20
0402, 10Ω, 0.1W	24	$0.10 x 24 = $2.40
0402, 5.1kΩ, 0.1W	12	$0.10 x 12 = $1.20

Miscellaneous (ESC)	**Quantity**	**Cost**
STM32F446	2	$8.63 x 2 = $17.26
MPU-9250	2	$6.78 × 2 = $13.56
SX1280IMLTRT	2	$6.45 x 2 = $12.90

Drone	**Quantity**	**Cost**
MRM Titan 2208-1100KV	8	$7.50 x 8 = $60.00
Thunder Power RC TP1800-4SM70	1	$53.99 x 1 = $53.99
Carbon Fiber Rods	15	$10.99 x 3 = $32.97

Controller	**Quantity**	**Cost**

RadioLink 2.4 GHz 8Ch Controller with 2 Joysticks	1	$52.44 x 1 = $52.44

A.2 Requirement and Verification Tables

A.2.1 Electrical Subsystem

Requirement	**Verification**
☐ 12V battery capable of outputting at a nominal 12V ☐ 12V battery capable of power draw up to 8 · 150W = 1200W ☐ 12V battery capable of powering flight time of around 5 minutes, approximately an energy capacity of 8 · 30 W/12V · 1/12 hour = 240/144 Ah = 1667 mAh	☐ **Multimeter testing** between two battery terminals, test when **fully charged** and **mostly discharged** ☐ **Current sense tracking** through **STM32 microcontroller** for each of the ESCs to verify total current draw ☐ **Discharge** fully charged battery through all 8 loaded motors until fully discharged
☐ Buck converter capable of outputting a steady 3.3V from a 12 V input ☐ Buck converter capable of current draw up to 160 mA ☐ Buck converter capable of outputting 3 different PWM channels at 100 KHz	☐ **Multimeter testing** between buck converter output terminal**s**, test when discharging battery through motors and idle ☐ **Multimeter testing** between buck converter terminals with varying resistive loads, with up to 200 mA total current draw ☐ **Oscilloscope testing** at ESC inputs to read PWM output from MCU, powered by buck converter

☐ MOSFETs bridge capable of switching at 100 KHz ☐ MOSFET bridge capable of handling VDS as high as 30V	☐ **Oscilloscope testing** at MOSFET bridge outputs to verify correct output frequency ☐ Use **power supply** to artificially supply 30V VDS to MOSFETs and confirm output PWM functionality
☐ Gate drivers capable of handling input PWM frequencies of 100 KHz	☐ Verify motor operation **visually** and confirm correct phase outputs for each motor phase using **oscilloscope testing**
☐ BLDC motors run at 12V	☐ **Visually confirm** motor operation with applied trapezoidal BLDC motor control algorithm

A.2.2 Mechanical Subsystem

Requirement	**Verification**
☐ Central hub should have dimensions large enough to holding 12V lithium, ESC boards, flight control + telemetry, and MCU	☐ **Physically verify** the size of each component once ESC has arrived, alongside other parts and lay out overall positioning of central hub to confirm necessary dimensions
☐ Arms long enough that rotors do not touch each other and that they don't interfere with the rotors' airstream	☐ **Physically verify** using calipers the leeway between propellers placed at expected motor position and each

	portion of the drone frame and maximize
☐ Carbon fiber skeleton has rods with structural tolerances within ±1 mm deflection from the straight line to ensure consistent structure ☐ Carbon fiber rods have structural integrity, capable of less than ±1 mm deflection from starting when under 5N of load	☐ **Manually measure** deflection from the straight line using ruler and calipers ☐ **Manually measure** deflection from the straight line using ruler, weights and stable surface.

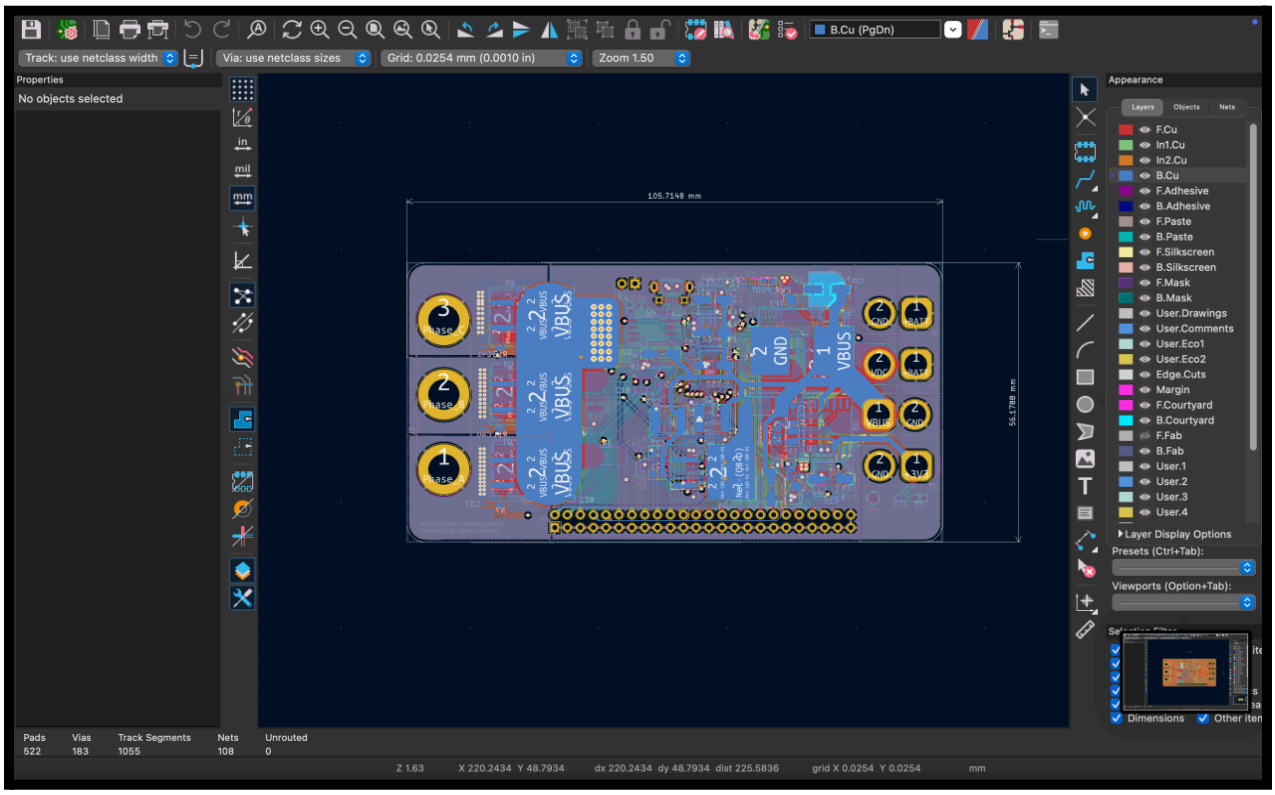
A.2.3 Control and Telemetry Subsystem

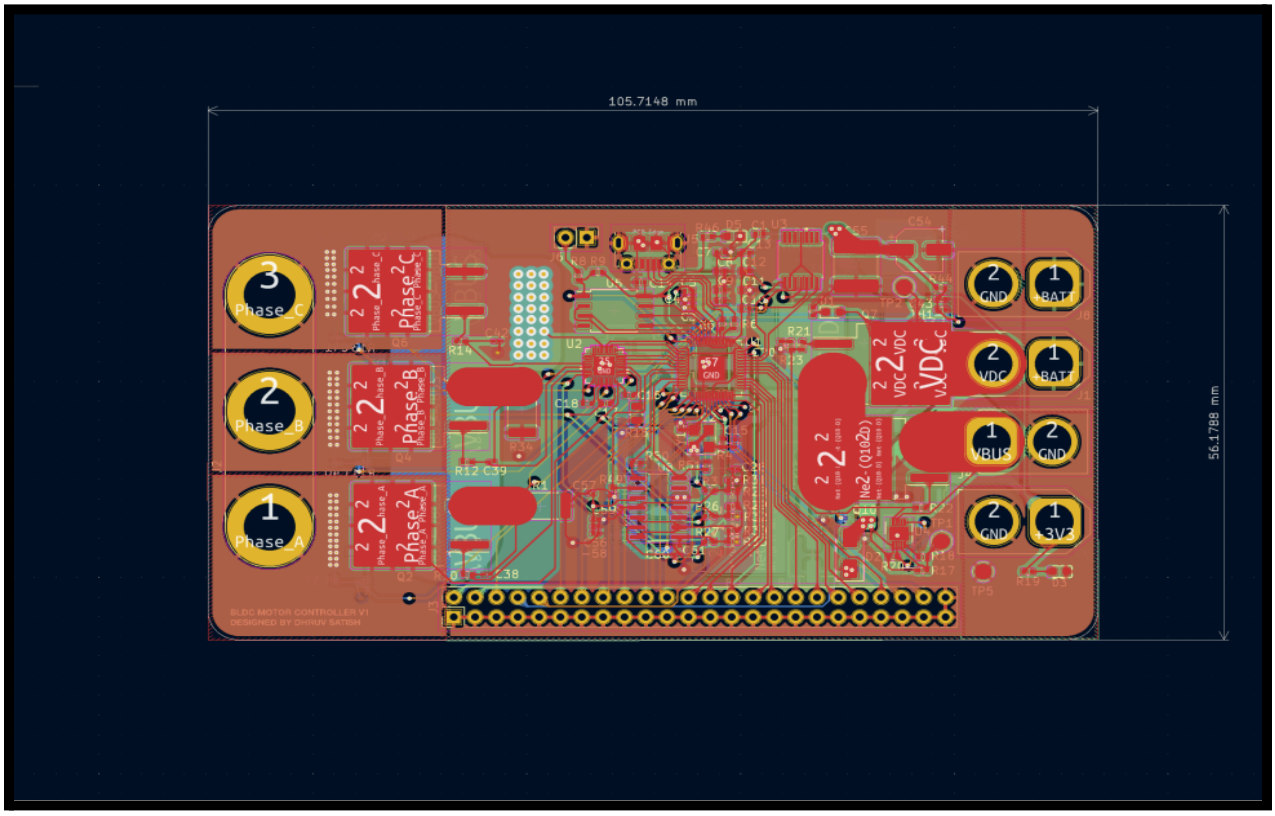
Requirement	**Verification**
☐ STM32f446ZE should stably operate using 3.3V digital input from buck converter ☐ Capable of SPI, I2C and UART communications at high bus clock rate of ~400 kHz for low latency and high sampling rate ☐ Processor clock rates of ~32MHz	☐ Verify operation of MCU by **connecting to PC** using **3.3V buck converter output** and confirming functionality ☐ Verify sampling rate by **polling data from IMU module** at specified rates and confirming correct size of data output ☐ **Verify through microcontroller datasheet**
☐ SX1280IMLTRT encoder must take digital signals and converts to 2.4GHz radio signals	☐ **Verify using 2.4 GHz receiver and antenna** connected to STM32, send and read arbitrary, distinct values ☐ **Verify through datasheet**

☐ Frequency accuracy: ±10–20 ppm (depends on crystal) ☐ TX output power tolerance: ±1.5–2 dB ☐ RX sensitivity tolerance: ±2 dB ☐ Phase noise: –100 dBc/Hz @ 100 kHz offset (typical)	

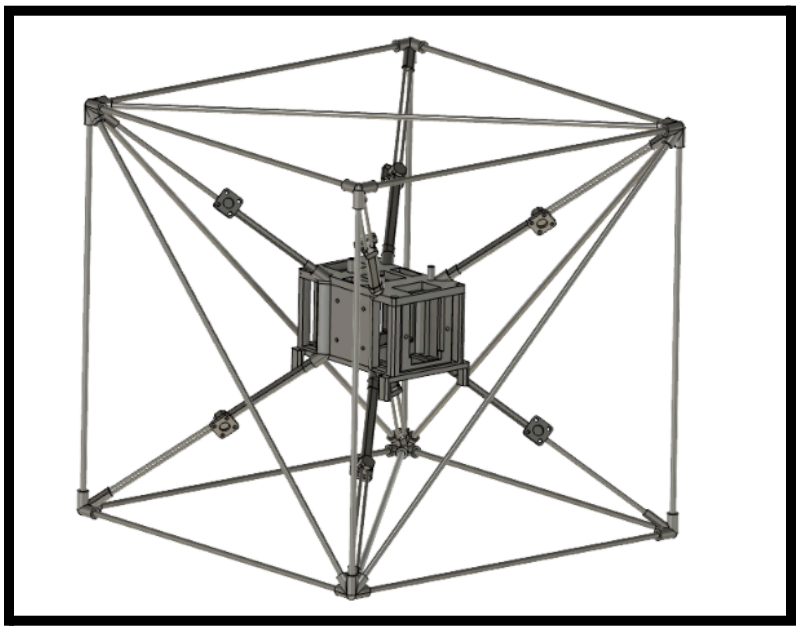
☐ Accelerometer (IMU)
  ☐ Zero-g offset: ±40–100 mg
  ☐ Sensitivity error: ±1–3%
  ☐ Noise density: ~100–300 µg/√Hz
☐ Gyroscope (IMU)
  ☐ Zero-rate offset: ±1–5 °/s
  ☐ Sensitivity error: ±1–3%
  ☐ Noise density: ~0.005–0.02 °/s/√Hz
	☐ **Verify physically and through the datasheet**, accelerate IMU in constrained directions by dropping it in different orientations and record accelerations
	☐ **Verify physically and through the datasheet**, test IMU at no rotation, rotate IMU using motor and confirm output readings match expected from final angular velocity and acceleration time.

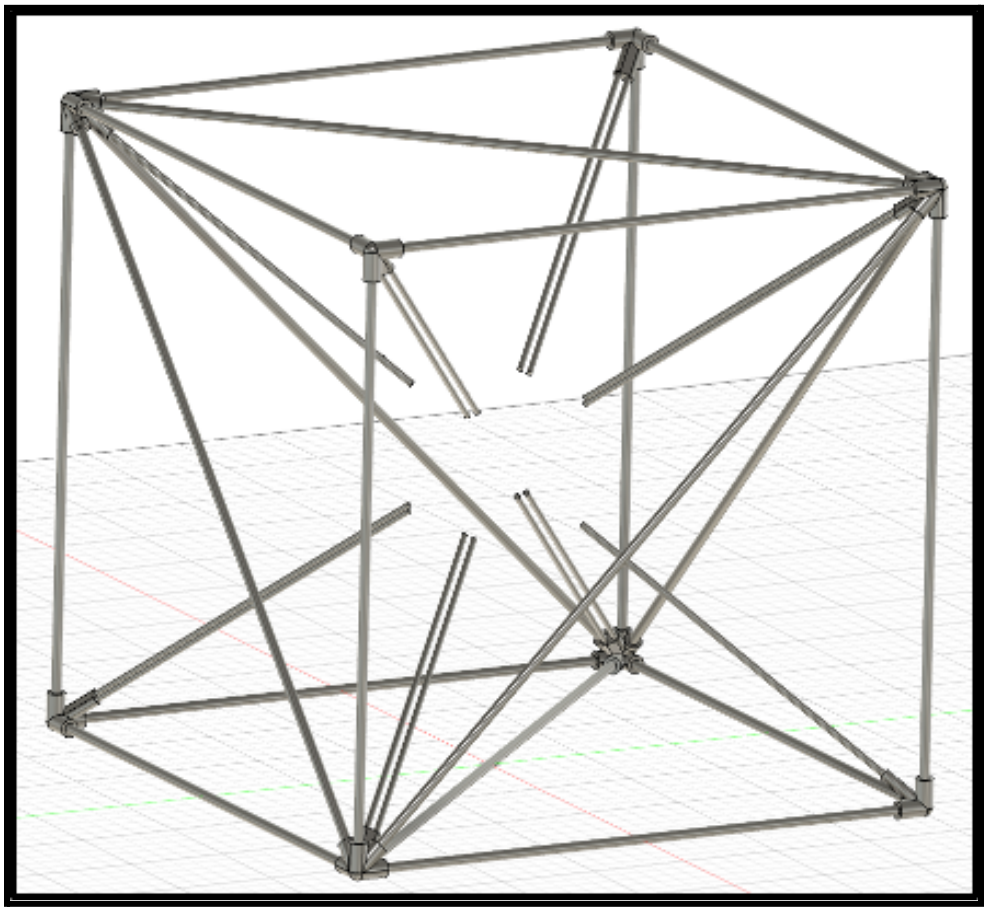
A.3 Additional PCB Images



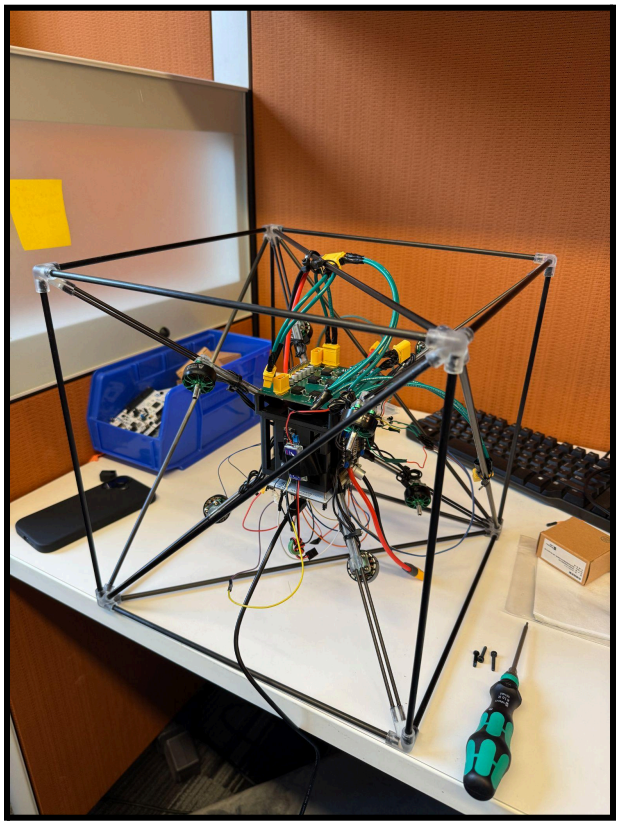


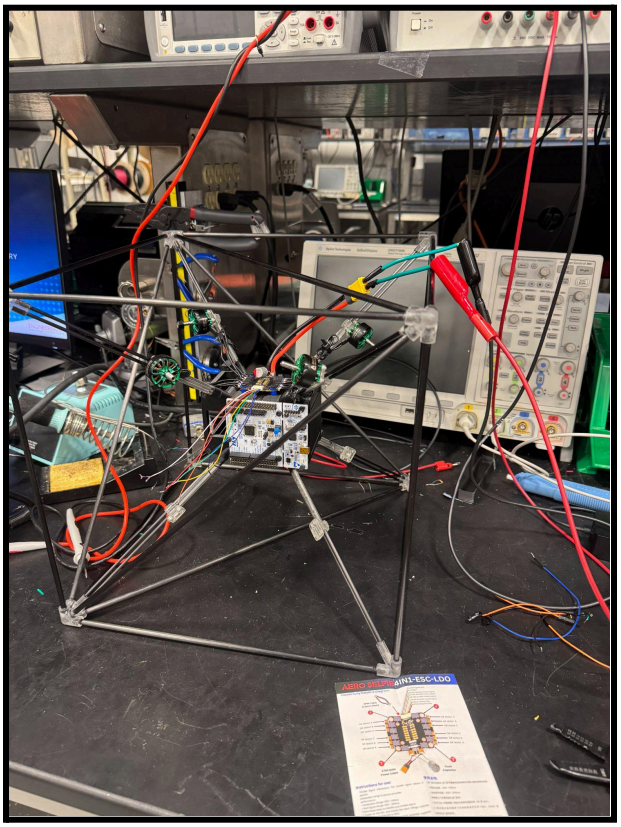
A.4 Mechanical Design Images

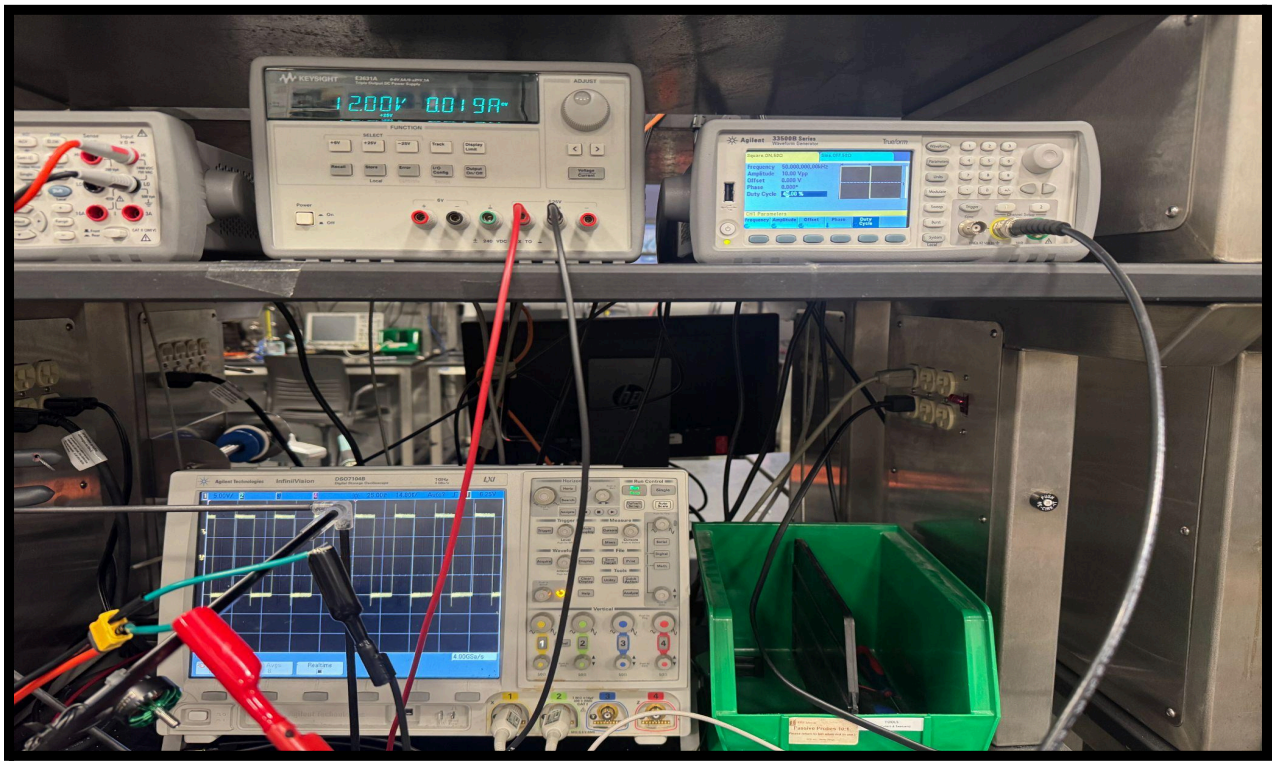




A.5 Project Build Images







A.6 High Level Code

C/C++

```
#include "control.hpp"
#include "geometry.hpp"
#include <algorithm> // for std::clamp

CascadedController::CascadedController() { }

void CascadedController::init(const VehicleParams& veh,
                              const PosCtrlParams& pos_cfg,
                              const AttCtrlParams& att_cfg,
                              const RateCtrlParams& rate_cfg)
{
    veh_      = veh;
    pos_cfg_  = pos_cfg;
    att_cfg_  = att_cfg;
    rate_cfg_ = rate_cfg;

    reset();
}

void CascadedController::reset()
{
    pos_state_.perr_int   = Vec3{0,0,0};
    att_state_.qerr_vec_int = Vec3{0,0,0};
}

Vec3 CascadedController::positionController(
    float dt,
    const VehicleState& state,
    const ControlSetpoints& ref)
{
    // Position and velocity errors (in inertial frame)
    Vec3 p_err = ref.p_des - state.p;
    Vec3 v_err = ref.v_des - state.v;

    // Integrate position error
    pos_state_.perr_int += p_err * dt;

    // Optional anti-windup clamp
    float I_MAX = 5.0f; // tune
    pos_state_.perr_int.x = std::clamp(pos_state_.perr_int.x, -I_MAX, I_MAX);
    pos_state_.perr_int.y = std::clamp(pos_state_.perr_int.y, -I_MAX, I_MAX);
    pos_state_.perr_int.z = std::clamp(pos_state_.perr_int.z, -I_MAX, I_MAX);
```

```
    // Gains from Eqs. (58)-(60)
    const float tau     = pos_cfg_.tau_pos;
    const float zeta    = pos_cfg_.zeta_pos;
    const float tau_i   = pos_cfg_.tau_pos_i;

    float k_p = 1.0f/(tau*tau) + 2.0f*zeta/(tau*tau_i);
    float k_i = 1.0f/(tau*tau*tau_i);
    float k_d = 2.0f*zeta/tau + 1.0f/tau_i;

    // Desired acceleration from controller + feedforward a_des + g
    Vec3 a_cmd = p_err * k_p
               + pos_state_.perr_int * k_i
               + v_err * k_d
               + ref.a_des          // feedforward desired accel
               + veh_.gravity;      // +g (Eq. (57))

    // Transform commanded accel to a *body-frame* thrust vector
    // Eq. (57): f_cmd = m * R(q) * ( ... )
    Mat3 R_IB = rotationInertialToBody(state.q); // I->B
    Vec3 f_cmd = R_IB * (a_cmd * veh_.mass);

    return f_cmd; // in body frame
}

Vec3 CascadedController::attitudeController(
    float dt,
    const VehicleState& state,
    const ControlSetpoints& ref)
{
    // q_err = q_des ⊗ q^{-1}
    Quaternion q_conj = state.q.conjugate(); // for unit quat, inverse =
conjugate
    Quaternion q_err  = ref.q_des * q_conj;
    q_err.normalize();

    float q0 = q_err.w;
    Vec3  qv{q_err.x, q_err.y, q_err.z};

    // sign(q_err,0) (scalar part)
    float sgn = (q0 >= 0.0f) ? 1.0f : -1.0f;

    // Error vector with sign
    Vec3 e = qv * sgn;
```

```
    // Integrate attitude error vector
    att_state_.qerr_vec_int += e * dt;

    float I_MAX = 5.0f; // tune
    att_state_.qerr_vec_int.x = std::clamp(att_state_.qerr_vec_int.x, -I_MAX,
I_MAX);
    att_state_.qerr_vec_int.y = std::clamp(att_state_.qerr_vec_int.y, -I_MAX,
I_MAX);
    att_state_.qerr_vec_int.z = std::clamp(att_state_.qerr_vec_int.z, -I_MAX,
I_MAX);

    // Gains from Eqs. (66)-(67)
    const float tau   = att_cfg_.tau_att;
    const float tau_i = att_cfg_.tau_att_i;

    float k_p = 2.0f/tau + 2.0f/tau_i;
    float k_i = 2.0f/(tau * tau_i);

    // Feedforward term R(q_err)^-1 * omega_des (Eq. (65))
    Mat3 R_err = rotationInertialToBody(q_err);  // R(q_err)
    Mat3 R_err_inv = R_err.transpose();          // inverse

    Vec3 omega_ff = R_err_inv * ref.omega_des;

    Vec3 omega_cmd = e * k_p
                   + att_state_.qerr_vec_int * k_i
                   + omega_ff;

    return omega_cmd; // body-frame desired angular velocity
}

Vec3 CascadedController::rateController(
    const VehicleState& state,
    const Vec3& omega_cmd)
{
    const float tau = rate_cfg_.tau_omega;

    Vec3 domega = omega_cmd - state.omega;   // ω_cmd - ω

    // J * ω
    Vec3 Jomega{
        veh_.inertia.m[0][0]*state.omega.x,
        veh_.inertia.m[1][1]*state.omega.y,
```

```
        veh_.inertia.m[2][2]*state.omega.z
    };

    // ω × (Jω)  (Coriolis term)
    Vec3 coriolis = state.omega.cross(Jomega);

    // t_cmd = (1/τ_ω) J(ω_cmd - ω) + ω × (Jω + rotor term)
    Vec3 t_cmd{
        (veh_.inertia.m[0][0] * domega.x) / tau,
        (veh_.inertia.m[1][1] * domega.y) / tau,
        (veh_.inertia.m[2][2] * domega.z) / tau
    };

    t_cmd += coriolis; // ignoring rotor inertia term for now

    return t_cmd;
}

void CascadedController::allocateThrust(
    const Vec3& f_cmd,
    const Vec3& t_cmd,
    float f_rot_cmd[8])
{
    // v_cmd = [f_cmd; t_cmd] (6x1)
    float v_cmd[6] = {
        f_cmd.x, f_cmd.y, f_cmd.z,
        t_cmd.x, t_cmd.y, t_cmd.z
    };

    // Initialize rotor thrusts
    for (int i = 0; i < 8; ++i)
        f_rot_cmd[i] = 0.0f;

    // f_rot = B_pinv * v_cmd
    for (int i = 0; i < 8; ++i) {
        float sum = 0.0f;
        for (int j = 0; j < 6; ++j) {
            sum += Geometry::B_pinv[i][j] * v_cmd[j];
        }
        f_rot_cmd[i] = sum;
    }

    // TODO later:
    //  - enforce |f_rot| within [f_min, f_max]
```

```
    //  - adjust along nullspace to reduce power / avoid reversals
    //  - hysteresis as in Section V-E
}

void CascadedController::step(
    float dt,
    const VehicleState& state,
    const ControlSetpoints& ref,
    float f_rot_cmd[8])
{
    // 1) Position controller: p, v, a_des -> f_cmd (body frame)
    Vec3 f_cmd = positionController(dt, state, ref);

    // 2) Attitude controller: q_des, omega_des + current q -> omega_cmd
    Vec3 omega_cmd = attitudeController(dt, state, ref);

    // 3) Angular velocity controller: omega_cmd + current omega -> t_cmd
    Vec3 t_cmd = rateController(state, omega_cmd);

    // 4) Control allocation: (f_cmd, t_cmd) -> rotor thrusts
    allocateThrust(f_cmd, t_cmd, f_rot_cmd);
}
```

References

[1] E. Ackerman, “ETH Zurich’s Omnicopter Plays Fetch,” *IEEE Spectrum*, May 17, 2017. [Online]. Available: https://spectrum.ieee.org/eth-zurich-omnicopter-plays-fetch. [Accessed: Sep. 18, 2025].

[2] B. Brescianini and R. D’Andrea, “An Omni-Directional Multirotor Vehicle,” *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 903–910, Apr. 2018.

[3] DeepBluEmbedded, “STM32 ESC PCB Design / FOC ESC (BLDC) Schematic.” [Online]. Available: https://deepbluembedded.com/stm32-esc-pcb-design-foc-esc-bldc-schematic/. [Accessed: Sep. 18, 2025].

[4] M. Hein, *Demystifying BLDC Motor Commutation: Trap, Sine, & FOC*, Texas Instruments, Apr. 2020. [Online]. Available: https://www.ti.com/lit/ml/slyp711/slyp711.pdf. [Accessed: Sep. 18, 2025].

[5] Jon’s Workshop, “Synchronous Rectifiers,” Nov. 2017. [Online]. Available: https://www.jons-workshop.com/syncrec.html. [Accessed: Sep. 18, 2025].

[6] Texas Instruments, “Integrated intelligence part 3: Motor startup from standstill position,” SSZT853, Dec. 2017. [Online]. Available: https://www.ti.com/lit/ta/sszt853/sszt853.pdf. [Accessed: Sep. 18, 2025].

[7] D. Torres and P. Heath, *Regenerative Braking of BLDC Motors*, Microchip Technology Inc. [Online]. Available: https://ww1.microchip.com/downloads/en/devicedoc/regenerative%20braking%20of%20bldc%20motors.pdf. [Accessed: Sep. 18, 2025].

[8] R. Tylor, “Minecraft, MCPE | How to make a Working Drone | No Commands, No Addons,” *YouTube*, Sep. 2, 2020. [Online]. Available: https://www.youtube.com/watch?v=FzrqVVbJ7EI. [Accessed: Sep. 18, 2025].

[9] Semtech, *SX1280 / SX1281 — Data Sheet*, Rev. 3.2, Mar. 2020. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/483/SX1280-81_Rev3.2_Mar2020.pdf. [Accessed: Sep. 18, 2025].

[10] “Omnicopter | PX4 Guide,” *PX4 Documentation*, [Online]. Available: https://docs.px4.io/main/en/frames_multicopter/omnicopter.html. [Accessed: Oct. 9, 2025].