The Perfect Shot is Fleeting.

AIRFRAME

ELEC3117 Final Report

Aither Technologies | M12A Group F
Chris Pagolu (z5440317)
& Ahmad Rana (z5481463)

1. EXECUTIVE SUMMARY

AirFrame by Aither Technologies is an **AI-powered** modular, three-axis **self-stabilising tripod** system designed for content creators and travel enthusiasts who require professional-grade stability without the complexity of traditional setups. The architecture is partitioned into the **AirMount**, a stationary tripod base with high-power **USB-C PD charging**, and the **AirBox**, a portable, battery-powered control unit. This modularity allows AirFrame to function as both a static, intelligent tripod and a handheld gimbal.

The system utilises a high-precision **6-axis Inertial Measurement Unit** (IMU) and an **ESP32** microcontroller in a closed-loop feedback system. Raw sensor data is processed using a user-tuneable Kalman filter for sensor fusion, and a Proportional-Integral-Derivative (**PID**) controller actively corrects for handheld motion and external disturbances to maintain a level horizon. This core functionality is enabled by four custom-designed PCBs managing power distribution, protection, and control logic.

Key features include a magnetic MagSafe/Qi2 wireless charging mount, an integrated iOS app (**AirLink**) for remote control and PID tuning, and a powerful AI assistant, **Aerial**, which uses multiple LLMs for voice commands and intelligent scene analysis.

A functional prototype successfully met key performance targets. Tested battery endurance exceeded **1.8 hours** under a continuous 250g payload. The system is conservatively rated for a full operational temperature range of **-20°C to +60°C,** bounded by component and Li-ion battery limitations.

The project also provides a clear five-year roadmap for commercialisation. Future work will focus on **design-for-manufacturing (DFM)** using more durable materials (PETG/ASA or aluminium), enhancing the power system with a higher-rated buck converter, and migrating to a more powerful microprocessor to support advanced **on-device AI**. Following compliance testing and consultation reviews, AirFrame is positioned for a phased domestic market entry, with a scalable supply chain planned for future international expansion.

Table of Contents

INDEX OF TABLES

GLOSSARY

- **AirFrame:** The overall product comprising AirMount (tripod/dock), AirBox (electronics/power), and ProHalo (3-axis gimbal).
- **AirOS:** The embedded firmware on the ESP32 handling sensing, estimation, control, actuation, BLE, and safety.
- **AirLink:** The iOS companion app providing telemetry, mode control, person tracking, PID tuning, and access to Aerial.
- **Aerial:** The AI assistant integrated into AirLink that supports natural-language control, tracking, scene analysis, and image tools.
- **Gimbal:** A multi-axis mechanical stage that stabilises an attached payload—in this case, a smartphone—about roll, pitch, and yaw.
- **Degrees of Freedom (DOF):** Independent axes of motion. AirFrame stabilises three rotational DOFs: roll, pitch, yaw.
- **Roll/Pitch/Yaw:** Rotation about the phone's longitudinal (roll), lateral (pitch), and vertical (yaw) axes.
- **Inertial Measurement Unit (IMU):** A sensor measuring linear acceleration and angular rate (BMI088, 6-axis).
- **Sensor Fusion:** Combining accelerometer and gyroscope data to estimate orientation with higher accuracy and lower drift.
- **Kalman Filter:** A recursive estimator used to fuse IMU signals and track angle and gyro bias for stable orientation estimates.
- **PID Controller:** A feedback controller using proportional, integral, and derivative terms to minimise orientation error smoothly.
- **Set-point:** The desired orientation the controller maintains; mode-dependent in AirOS.
- **Overshoot/Settling Time:** Measures of transient response i.e. how far past the target the system moves and how quickly it stabilises.
- **Backlash:** Mechanical play that can degrade precision and introduce small errors or chatter.
- **Servo:** Position-controlled actuator driven by a PWM signal (typically 50 Hz, ~1–2 ms pulse width).
- **Bluetooth Low Energy (BLE):** Wireless link used by AirFrame for telemetry and control (GATT service/characteristics).
- **GATT (Profile):** The BLE schema defining services/characteristics for reading angles, modes, and sending commands/tuning.
- **Person/Object Tracking:** Vision-based target detection on the phone; AirLink computes pan/tilt velocities sent to AirOS.
- **Magnetic Mount (MagSafe/Qi2):** Magnetic alignment system for secure phone attachment and standards-based wireless charging.

- **Pogo Pins / Contact Pads:** Spring-loaded pins and matching pads providing dock power between AirMount and AirBox.
- **USB Power Delivery (PD):** Protocol used on the dock to negotiate voltage (e.g., 9 V) for faster charging.
- **Buck Converter (Step-Down):** Switch-mode regulator converting the 2S battery voltage to 5 V (e.g., MP2393).
- **LDO (Low-Dropout Regulator):** Linear regulator providing low-noise 3.3 V logic supply from 5 V (e.g., TL1963A-33).
- **Ideal-Diode OR-ing:** MOSFET arrangement preventing reverse current and enabling seamless source selection (dock vs battery).
- **PPTC Fuse:** Resettable fuse that limits fault current on the dock side to protect upstream supplies.
- **ENIG:** Electroless Nickel Immersion Gold finish on pads for low contact resistance and durability (e.g., contact pads).
- **Hot-dock/Hot-swap:** Safe docking/undocking under power without back-feed or large inrush current.
- **Ripple (Rail):** Residual AC component on a DC power rail; kept low to avoid audio/servo interference.
- **NVS (Non-Volatile Storage):** ESP32 flash storage used to persist gains, filter parameters, and mode preferences.
- **Model Context Protocol (MCP):** A protocol developed by Anthropic that Aerial can use to coordinate tools/models for AI-driven features.

ACRONYMS

- A2DP: Advanced Audio Distribution Profile (Bluetooth audio streaming profile).
- ADC: Analog-to-Digital Converter.
- BLE: Bluetooth Low Energy.
- dps: Degrees per second (angular rate unit).
- DOF: Degrees of Freedom.
- EMC/EMI: Electro-Magnetic Compatibility / Electro-Magnetic Interference.
- ENIG: Electroless Nickel Immersion Gold (PCB finish).
- ESR/DCR: Equivalent Series Resistance / DC Resistance (of capacitors/inductors).
- GATT: Generic Attribute Profile (BLE data model).
- I2C: Inter-Integrated Circuit serial bus.
- I2S: Inter-IC Sound digital audio bus.
- IMU: Inertial Measurement Unit.
- KPI (optional): Key Performance Indicator (e.g., battery life, settling time).
- LDO: Low-Dropout Regulator.

- Li-ion: Lithium-ion battery chemistry.
- MCP: Model Context Protocol.
- MCU: Microcontroller Unit (ESP32).
- NVS: Non-Volatile Storage.
- ODR: Output Data Rate (sensor sampling rate).
- OVP/OTP: Over-Voltage Protection / Over-Temperature Protection.
- PCB: Printed Circuit Board.
- PD: Power Delivery (USB PD).
- PID: Proportional–Integral–Derivative controller.
- PPTC: Polymer Positive Temperature Coefficient (resettable fuse).
- PWM: Pulse-Width Modulation (servo and motor control signal).
- UART: Universal Asynchronous Receiver/Transmitter (serial).
- VREG: Voltage Regulator.

2. Project Description

2.1. PROBLEM STATEMENT

Many content creators, videographers, and travellers struggle to get smooth, level shots when filming in unpredictable environments. Current stabilisers often require fiddly setup steps and still can't fully handle wind gusts or the natural shake of handheld shooting. As a result, creators sometimes miss the perfect moment or end up with shaky, unusable footage which can be a frustrating setback when trying to produce professional-quality work.

2.2 NEEDS ASSESSMENT

Primary research was conducted to determine the needs of users. The results of this research are included in Appendix G.

2.2.1 Target Audience

Online research demonstrated that while our target audience can primarily be social media content creators and travel/wildlife enthusiasts, it also found that it could address the contemporary concerns of casual photographers, sports spectators, real estate videographers, physics teachers, product marketing videographers, and specialists in cinema, sports, vehicles, modelling and educational media.

Thus, this research overall signalled the broad diverse range of customers from different fields that could use this product, and its immense potential profitability.

2.2.2 User requirements

From our research, users want one thing above all: footage that stays smooth and steady, no matter if they're filming in windy streets, crowded events, or cold outdoor conditions. They also value extra conveniences like Bluetooth audio playback and built-in phone charging. Portability is key, with the device being light enough to carry all day without fatigue. Long battery life is essential for extended shoots, and the system should be tough enough to handle a variety of weather conditions without compromising performance.

2.2.3 Use Cases

Content creators, travel enthusiasts and all types of media photographers and videographers listed require the camera hold stable while counteracting any jittering that occurs during photo shooting or adjusting camera movement to any movement in its target.

Current Competitors and Limitations

Below is a summary of the current competitors on the market, including their expertise, strengths and weaknesses.

Ecosystem Builders (e.g. DJI):

- DJI dominates through a powerful integrated hardware and software ecosystem.
- Their strengths are in software integration and cutting-edge features.
- But their weak areas come down to high costs, proprietary "walled garden" ecosystem, and significant geopolitical/supply chain risk.

Value Challengers (e.g. Zhiyun, FeiyuTech and Hohem):

- They compete aggressively on payload capacity, features, and price.
- They have excellent performance-per-dollar proposition.
- However, they have less polished software experience and weaker brand ecosystems.

Design Innovators (Peak Design and Manfrotto):

- They focus on premium mechanical design, build quality and durability.
- They have good build quality and reputation.
- But they are slower to adopt electronic innovation and smart features.

There are also other areas that competitors' products fall behind in:

a) **Being Slow to Set Up** - Many stabilisers require multiple adjustments before filming, and users would find that frustrating when they need to start shooting quickly.
b) **Having Static Nature** - most users need a dynamic stabiliser not a static one
c) **Stability vs portability vs cost** - users face a compromise between two out of three of these factors from the manufacturing process
d) **A vulnerability to External Forces** - these gimbals face difficulties adjusting with wind, vibrations, random human motions which reduce the entertainment quality of footage for consumers.
e) **Causing handheld fatigue** - some devices make users extend their arms for long periods of time which would strain their limbs.

We are proud to say our gimbal has also adapted to the newer technological landscape of Qi2 Charging and the rise of AI, where the alternatives have fallen way behind.

Unique Selling Points

1. **Intelligent AI Gimbal Control:** Aerial (see Section 4.4.3) acts as an intelligent AI assistant orchestrating multiple Large Language Models (LLMs) like Gemini 2.5 Flash, Claude 4 Sonnet, GPT-5 all for advanced natural language processing and Model Context Protocol (MCP) to determine optimal response and control AirFrame Gimbal.
2. **Software-Tuneable PID Control:** The orientation of the device can be changed via use of PID tuning in the connected AirFrame Software.

3. **Integrated AI Image Generation & Editing:** Aerial can analyse the camera to determine the right amount of each photographic feature, meaning focus, exposure, white balance, and even composition, and this enables the automated capture of high-quality content. It can also use GPT-5's image generation and editing capabilities to enhance your images using AI.
4. **Advanced Person & Object Tracking:** Aerial is capable of real-time, precise person and object tracking, automatically moving AirFrame's gimbal to keep the subject perfectly framed within shot.
5. **Natural Language Voice & Text Control:** Natural language commands voice input or text chat through which users can intuitively control AirFrame and access Aerial's features.
6. **Adaptive Learning & Contextual Memory:** Aerial delivers a more personalised user experience by learning user preferences, utilising chat history and making the appropriate decisions.
7. **Superior 6-Axis Stabilisation:** 3-Axis Stabilisation with 6-Axis Sensing: we use a 6-axis IMU (accel + gyro) and precise motor control to actively counteract roll, pitch and yaw. Translational disturbances are indirectly mitigated by the tripod base.
8. **Bluetooth Speaker:** Built-in wirelessly connected speaker allows users to monitor or playback audio directly from their connected smartphone.
9. **2-in-1 Firm Grip Magnetic Safe Wireless Charging Stand:** On the top is a secure magnetic mounting solution for smartphones that provides wireless charging and a convenient display stand when not actively recording.
10. **Built-in Battery Charging Modules:** Integrated circuitry for direct battery charging via standard USB-C input, allowing users to charge the gimbal's internal battery pack without removing the batteries.

Product Concept

AirFrame is a next-generation smartphone stabiliser tripod and gimbal hybrid, engineered to address the market's need for perfectly stable levelled camera footage across diverse environments.

The goal is to provide an automated stabilisation, to dynamically counteract the unwanted camera jittering and maintain a level horizon, despite the terrain or disturbances like wind or handheld motion. This aims to empower content creators, a broad range of videographers, and travel enthusiasts by freeing them from technical setups and redirecting their efforts toward their creative vision, eliminating missed opportunities and unusable footage. AirFrame aims to provide a robust modular portable design for real-world conditions by including features like battery charging modules, magnetic wireless charging, and Bluetooth audio speaker that all elevate the user experience.

Beyond stabilisation, AirFrame provides a modern solution to enhance user experience through integrated AirFrame software and more intelligent, AI-driven functionalities. Aerial (our AI solution, see Section 4.4.2) acts as a powerful tool for photo taking enhancement, offering intuitive voice/text control, smart scene analysis and advanced tracking. It further streamlines workflow and elevates creative possibilities.

Design Concept

AirFrame would be a control integration of electrical hardware, mechanical systems and sophisticated control software with cutting-edge AI.

It would have a secure designated mount with three degrees of freedom, which are Roll, Pitch, Yaw. An IMU present in the same system would measure the gimbal's linear and angular acceleration. There are 3 servo motors and each of the rotational degrees of freedom are actuated by their respective motor. The IMU acceleration data will transfer over to the microcontroller.

This motion data would be filtered then compared with the desired stable values to counteract unwanted impulses and rotations. The resulting three control signals are each fed into their own motor driver to achieve stability through position acceleration feedback. The device will be powered by a series combination of two rechargeable batteries. It will also have a charging module to extend the life of the system.

In addition, the microcontroller must have Bluetooth in order to connect to the software networks, to successfully embed Aerial with the smartphone and AirFrame system. The software should have manual movement tuning.

3. Product Overview

AirFrame comprises 3 subsystems:

1. AirMount: Stable charging mount for 9 V power delivery to the AirBox, and it has tripods attached with a USB-C input power delivery PCB for the delivery of power to the batteries via pogo pins at the height of the base by which it delivers output power when charged.

2. AirBox: Portable, durable and re-attachable box containing the sensors and brain of the system, whereby it contains a 2S AT18650 battery pack nominal 7.4 V that is stepped down by a Buck Converter (Power Fabric) to power the CoreFrame PCB. CoreFrame consists of built-in Wi-Fi and Bluetooth ESP32 microcontroller and BMI088 IMU sensors that have a built-in accelerometer and gyroscope for 6-axis movement measurement data. It also has contact pads at the bottom for the reception of the 9 V input power from AirMount to charge its batteries via a TP5100 charging module.

3. ProHalo (gimbal): The actuation and stabiliser system holding the phone firm and steady. AirFrame actively stabilises rotational motion about three axes (yaw, pitch, roll) using three servos, informed by the BMI088 IMU. Translational disturbances are indirectly mitigated by the tripod base. There is also software for the device itself which can display the Roll, Pitch, Yaw data from IMU sensors, along with built-in Aerial, and software PID tuning.

The following figure showcases the high-level block diagram of the system:

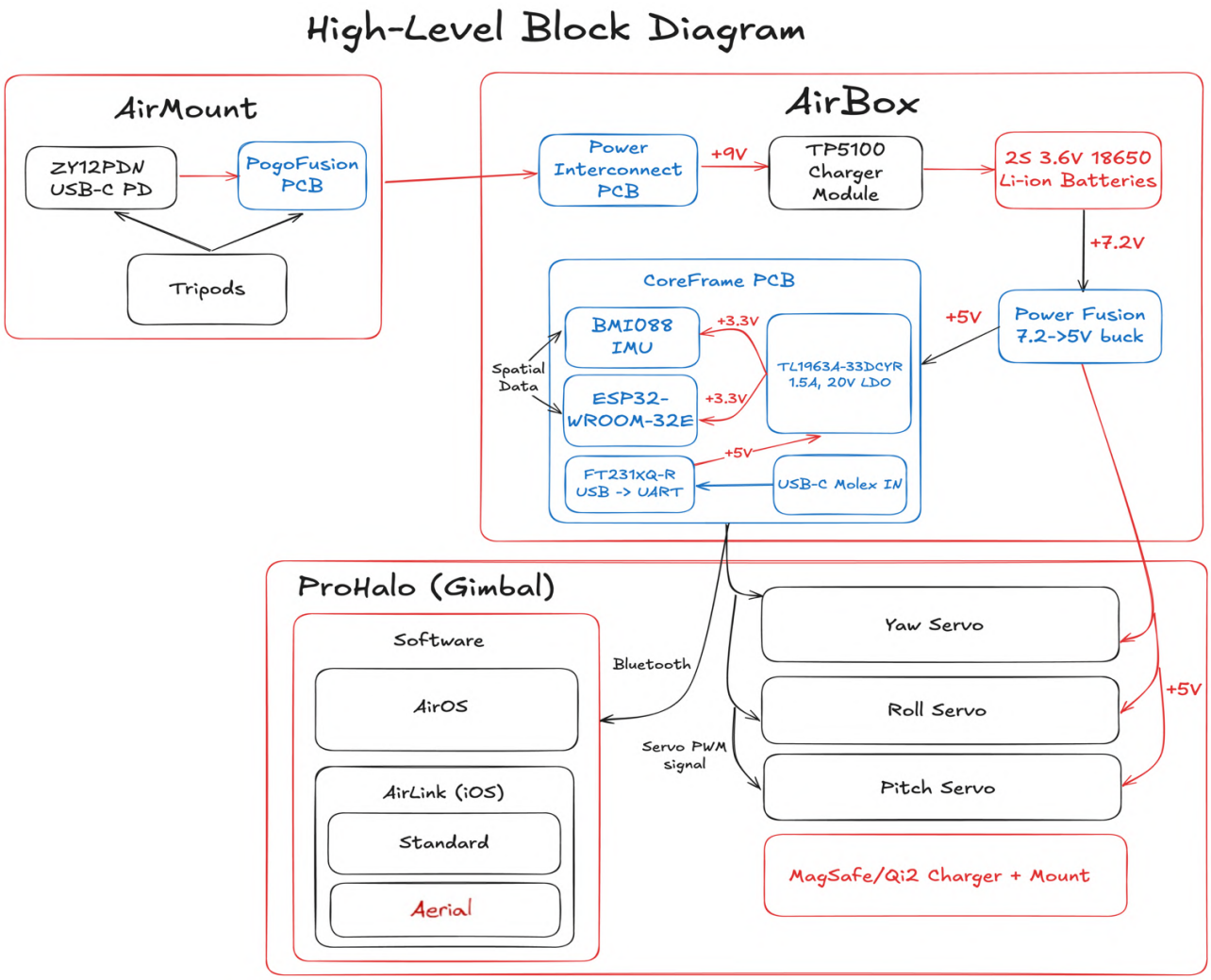


Figure 3.1. High-level Block Diagram of the AirFrame. Additional block diagrams for power and data are provided in Section 4.

Principle of Operation

Complex Stabilisation Mechanism

AirFrame operates on a sophisticated closed-loop feedback control system. This system is designed to counteract external accelerative and rotational forces acting on mounted smartphone cameras, thereby continuously maintaining a perfectly stable levelled camera orientation (roll, pitch, yaw) across diverse environments. This is done by detecting and then sampling changes in the acceleration and gyroscopic behaviour of the phone mount via a 6 Degrees-of-Freedom IMU, which is the BMI088. This IMU is physically integrated within the AirBox which is also connected to the phone mount for accurate motion sensing. The BMI088 streams acceleration and angular-rate data to the ESP32 over I^2C (addresses 0x18 and 0x68,

datasheet Table 3). We run both accelerometer and gyroscope at 200 Hz (BMI088 4.4), then fuse them on the ESP32.

Stabilisation occurs independently for each rotational axis. Inside the data micro-processing of the ESP32, raw accelerometer measurements are low-pass filtered to eliminate noise, and gyroscope measurements high-pass filtered to remove drift effects. The filtered measurements are then combined into a Kalman Filter sensor fusion algorithm. The algorithm provides an accurate real-time estimate of the phone mount's true orientation. This orientation estimate is then utilised to determine an estimate of the phone mount's deviation from initial position.

The determined error signal is passed through a Proportional-Integral-Derivative (PID) control algorithm. This PID controller produces an overdamped response to minimise unwanted oscillations in the mount's motion and provide smooth mechanical damping to any forced motion. The resulting control signal released from the ESP32 can also be modified by AirFrame Software PID tuning. Afterward, it is provided to the motor drivers. These drivers translate commands into precise electrical signals that adjust phone mount positioning accordingly. As time goes on, the system undergoes ongoing stabilisation through a continuous, dynamic closed-loop feedback cycle.

Battery Charging

The battery charging system is facilitated by a magnetically attachable AirMount. The AirMount contains a USB-C Power Delivery PCB and an elevated PogoFusion PCB, equipped with pogo pins designed to align with contact pads located on the underside of the AirBox.

When the AirBox rests on the AirMount, this connection delivers a 9 V power input to the TP5100 charging module within the AirBox. This module quickly charges the Li-Ion batteries in the AirBox, thereby enhancing the battery life of the system.

Speaker Connections

AirFrame has a built-in speaker feature for audio playback. Its ESP32 has an established Bluetooth A2DP connection with the user's smartphone, enabling wireless reception of digital audio streams. This ESP32 interfaces with DAC via I2S Digital Audio Interface. This I2S connection ensures a high-quality audio path from microcontroller to the audio processors. The processed analogue audio signal is then fed to the speaker. Unfortunately, in this prototype, audio could not be fully integrated by demo time due to power budget; see Section 5.3.1.

MagSafe/Qi2 Charging Phone Mount

AirFrame incorporates a MagSafe/Qi2-compatible phone mount, providing a secure and stable magnetic attachment for modern smartphones (including iPhones and upcoming Android smartphones). The positioning of this phone mount is manually adjustable through moving the platform along the pitch axis. This mount also leverages strong magnetic attraction for firm device retention even during dynamic stabilisation. This integrated mount is capable of wireless magnetic inductive charging to smartphones, although it draws power from external power sources.

AirLink app with PID Tuning

AirLink, AirFrame's smartphone application, retains a Bluetooth connection with the ESP32, providing direct real-time access to the system's rotational movement data. Beyond monitoring, it can also facilitate manual PID tuning by allowing users to input and send specific control parameters to ESP32's PID algorithm. This lets users fine-tune stabilisation characteristics for their preferred performance. In addition, the control system can also be commanded by the software's Aerial commands.

Meeting Design Requirements

The following comparisons were made between the original product design and the original design requirements from the planning stage. This is expanded on later in Section 5.

- Device levelling of the phone charge mount was manually achieved.
- The product does have universal compatibility with Qi2 magnetic safe devices, which currently includes modern iPhones and upcoming Android smartphones.
- Imminent prototyping deadlines deemed height adjustment a non-essential requirement and by use of 3D printed Tripods, this was traded off for product quality considering small assembly timeframe
- The product achieves a battery life of 1.8 hours, considering average power consumption by the three servo motors, battery capacity, and software integration. This undercounts the original battery life metrics of 8 hours however that metrics did not realistically consider total motor power consumption or software.
- Product successfully met the constraints for small weight, but the dimensions differ from original requirements metrics because PCBs were too wide, and those metrics were likely unrealistic.
- Manual Pitch control enabled for the MagSafe charging platform.
- Grounding adaptability was achieved, as the tripod legs can stand on any floor in a stable manner.
- A versatile Operational Temperature Range from –20°C to +60°C, as determined by battery, ESP32, IMU and the MG996R servos.

4. Detailed Design

4.1. SYSTEM ARCHITECTURE OVERVIEW

AirFrame is a modular, three-axis self-stabilising tripod system designed to deliver professional-grade camera stability for smartphones in both static and dynamic shooting scenarios. The architecture is primarily partitioned into three primary physical subsystems: AirMount, AirBox, and ProHalo Gimbal System. Each subsystem has its own distinct mechanical, electrical and control responsibilities.

This modularity allows AirFrame to operate either **docked** to the tripod for stationary use or **undocked** as a handheld gimbal stabiliser.

4.1.1. Subsystem Overview

AirMount provides the height-adjustable tripod and a precise locating cradle for the AirBox. It negotiates a 9 V profile from a standard USB-C wall adapter using a ZY12PDN PD trigger and routes that power through the PogoFusion PCB. PogoFusion uses MP015219 spring-loaded pogo pins for reliable contact and a YAGEO SMD2920B300TF/30 PPTC resettable fuse, so faults are limited at the mount before power ever reaches the AirBox.

AirBox houses the complete power subsystem and all control electronics. Two Samsung 25R 18650 cells are connected in 2S $(7.2\ \mathrm{V}\ nominal, 8.4\ \mathrm{V}\ max)$ and are charged by a TP5100 2S charger. Dock power enters via the Power Interconnect PCB, which uses an FDD4141 P-channel MOSFET configured as an ideal high-side element for reverse-polarity, hot-dock and back-feed protection. The Power Fabric PCB then generates a regulated 5 V rail using an MP2393 synchronous buck converter with a 2.2 µH SPM6530T inductor. A TL1963A-33 LDO on the CoreFrame board generates a clean 3.3 V rail for logic. Processing and sensing are handled by the CoreFrame PCB with an ESP32-WROOM-32E-N4 MCU, a Bosch BMI088 IMU connected over I²C, and an FT231XQ-R USB-UART bridge. For sound, the AirBox includes an Adafruit MAX98357A I²S class-D amplifier and a 3 W, 4 Ω loudspeaker. Interfaces include USB-C for programming and logs, BLE/Wi-Fi (BLE used in the prototype), and the pogo dock for power.

ProHalo is the three-axis gimbal head. Three MG996R high-torque servos provide yaw, pitch and roll, each driven by a dedicated ESP32 PWM output. The payload mount accepts a MagSafe/Qi2 charger or bracket so an iPhone can be powered while filming. Rigid arm sections and short lever arms reduce flex and backlash.

4.1.2. Power and Data Flow

Power Path (Docked mode):

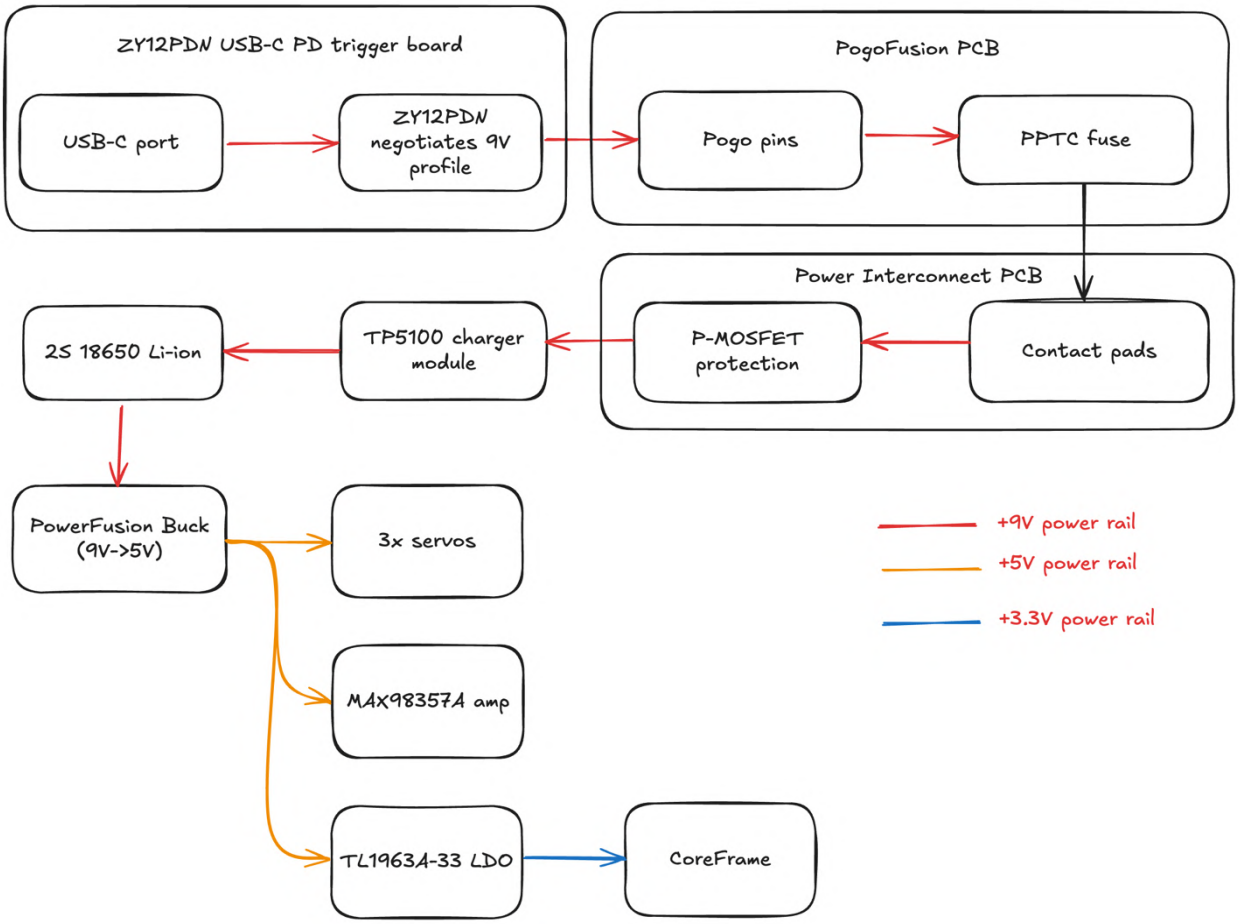


Figure 4.1. Power path diagram showing voltage rails and load distribution.

Data Path:

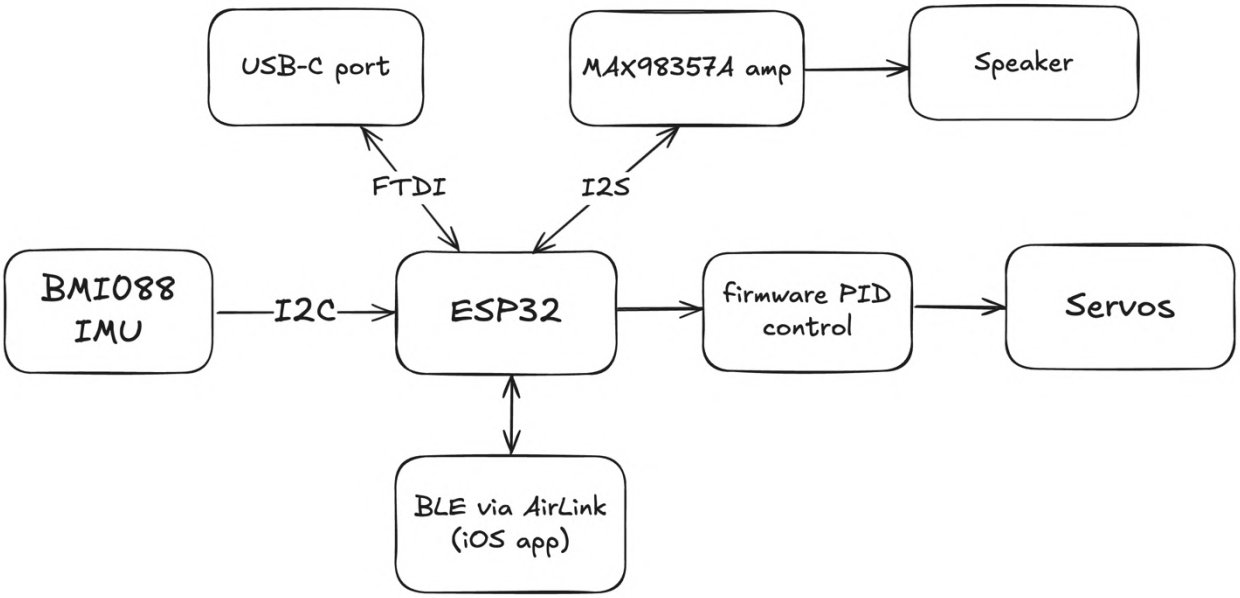


Figure 4.2. Data path diagram showing different communication protocols between different components.

4.2. HARDWARE IMPLEMENTATION

This section details every physical subsystem that makes up AirFrame, focusing on the complete power path and protections, the custom PCBs, actuator electronics, layout/EMC choices, and the electrical interfaces between boards. Where helpful, design equations and calculated margins are provided, based on datasheet values.

4.2.1. Power & Protection (end-to-end)

1. **Docked mode:**

- **USB-C PD negotiation (AirMount):** The ZY12PDN trigger requests a 9V profile from a compliant USB-C wall adapter. The negotiated 9V rail is routed to the PogoFusion PCB.
- **PogoFusion PCB:** Two MP015219 spring-loaded pins provide low-resistance, compliant contact to the AirBox contact pads. A series PPTC fuse (YAGEO SMD2920B300TF/30, $\mathrm{I_{hold}} \approx 3\ \mathrm{A}$) sits upstream to limit fault energy during mis-dock/short events so faults are contained on the mount side before power reaches the AirBox.
- **Power Interconnect PCB (on AirBox side):** A high-side P-channel MOSFET (ONSEMI FDD4141, $R_{\mathrm{DS(on)}} \approx 12.3\ \mathrm{m\Omega}$) is wired as an ideal-diode element. The body diode points from the dock input toward the internal node such that the FET enhances only when $V_{\mathrm{dock}} > V_{\mathrm{load}}$). This arrangement blocks reverse current from the battery into the mount, reduces inrush on hot-dock and provides polarity protection without the forward drop of a diode. Conduction loss worst-case at $I = 3\ \mathrm{A}$: $P \approx I^2 R_{\mathrm{DS(on)}} = 9 \times 12.3\ \mathrm{m\Omega} \approx 0.11\ \mathrm{W}$.

2. **Battery subsystem:**

- **Cells:** 2x Samsung 25R 18650 in series (7.2V nominal, 8.4V max, ~2x 2.5 Ah), wired with appropriately rated silicone-insulated leads and a battery holder. Pack energy is ~ 18Wh.
- **Charging:** TP5100 2S charger module (CC/CV) is fed from the docked PD rail. Charge current is configured via the module's sense resistor to meet thermal limits and cell recommendations; the status pins are available to the MCU for UI/telemetry (however, this was left unused in the actual project due to time constraints). The charger's OVP, OTP, and CV taper protect the cells during charge.
- AirBox operates from the battery when undocked, and from the dock input (while also charging) when docked; the ideal-diode FET and the charger's internal ORing realise seamless source selection.

3. **5V conversion (Power Fabric PCB):**

- **Controller:** MP2393 synchronous buck, $f_s \approx 650\ \text{kHz}$, 3 A rating, in SOT583.
- **Design targets:** V_{in} = 6.0 – 8.4 V, V_{out} = 5.0 V, $I_{out,\,cont}$ = 3 A (servos + audio), transient-capable of short current burts.
- **Inductor:** SPM6530T-2R2M, $L = 2.2\ \mu\text{H}$, $I_{sat} \approx 8.2\ \text{A}$, $R_{DCR} = 17.3\ \text{m}\Omega\ \text{typ}\ (19.1\ \text{m}\Omega\ \text{max})$.
    - With V_{in} = 7.2V, duty $D \approx \frac{V_{out}}{V_{in}} = \frac{5}{7.2} \approx 0.694$.
    - Ripple current:
$$\Delta I_L = \frac{V_{out}\,(1-D)}{L\,f_s} = \frac{5\,(1-0.694)}{2.2\times10^{-6}\cdot 650\times10^{3}} \approx 1.07\ \text{A}.$$
    - Peak inductor current at I_{out} = 3 A:
$$I_{L,pk} \approx I_{\text{out}} + \frac{\Delta I_L}{2} \approx 3.54\ \text{A} \ll I_{\text{sat}}.$$
- **Output capacitors:** Low-ESR ceramics sized for ripple and transient response. Ideal ripple component (capacitive) is:

$$\Delta V_C \approx \frac{\Delta I_L}{8\,C\,f_s}.$$

For $C_{tot} = 44\ \mu\text{F}$, $\Delta V_C \approx \frac{1.07}{8\cdot44\times10^{-6}\cdot650\times10^{3}} \approx 4.7\ \text{mV}$ (ESR contributions add $\Delta V_{ESR} \approx \Delta I_L \cdot ESR$). With low-ESR MLCCs, total ripple remains comfortably below servo/audio sensitivity.

- **Efficiency & Thermals:** Inductor copper loss at $I_{rms} = 2\ \text{A}$: $P_L \approx I_{rms}^2 R_{DCR} \approx 0.076\ \text{W}$. Controller conduction/switching loss is spread via a large top copper pour with thermal vias to ground.

4. 3.3V logic rail (CoreFrame):

- TL1963A-33, low-noise LDO feeding ESP32, BMI088, FTDI and logic. With a typical logic load of $I_{3.3} \approx 150\ \text{mA}$ (peaks higher with Wi-Fi TX, which isn't used anyway), dissipation is:

$$P_{LDO} \approx (5.0 - 3.3)\cdot 0.15 \approx 0.255\ \text{W}.$$

Copper pours on the LDO tab/layers keep ΔT modest. If Wi-Fi protocol is used in the future and if sustained Wi-Fi peaks are expected, the thermal pad footprints and via-in-pad array provide additional headroom.

5. Rail budget:

- **5 V rail (MP2393):** 3x MG996R (avg. 0.5-0.9 A each while tracking), MAX98357A amp (can be quite bursty but is volume-capped in firmware to limit excessive current use).

- **3.3 V rail (TL1963A):** ESP32-WROOM-32E (typ. 40 mA, higher peaks on RF with a maximum of 379 mA), BMI088 (~5 mA), FT231XQ-R (8.4 mA max), LEDs.

4.2.2. AirMount

AirMount is a low-profile tripod base that houses the PD trigger and PogoFusion while providing a self-locating cradle for AirBox. The top deck recess constrains AirBox and resists shear from gimbal motion. Three printed legs set the support polygon so the AirBox centre of mass sits inside the triangle during normal pan and tilt. The PogoFusion PCB mounts on standoffs under the deck and the PPTC is placed near the PD trigger to limit fault current if a contact pad is mis-aligned or shorted. Wire exits are strain-relieved using printed bridges. PLA was used for the prototype. PETG or ASA and a mechanical latch are recommended for production.

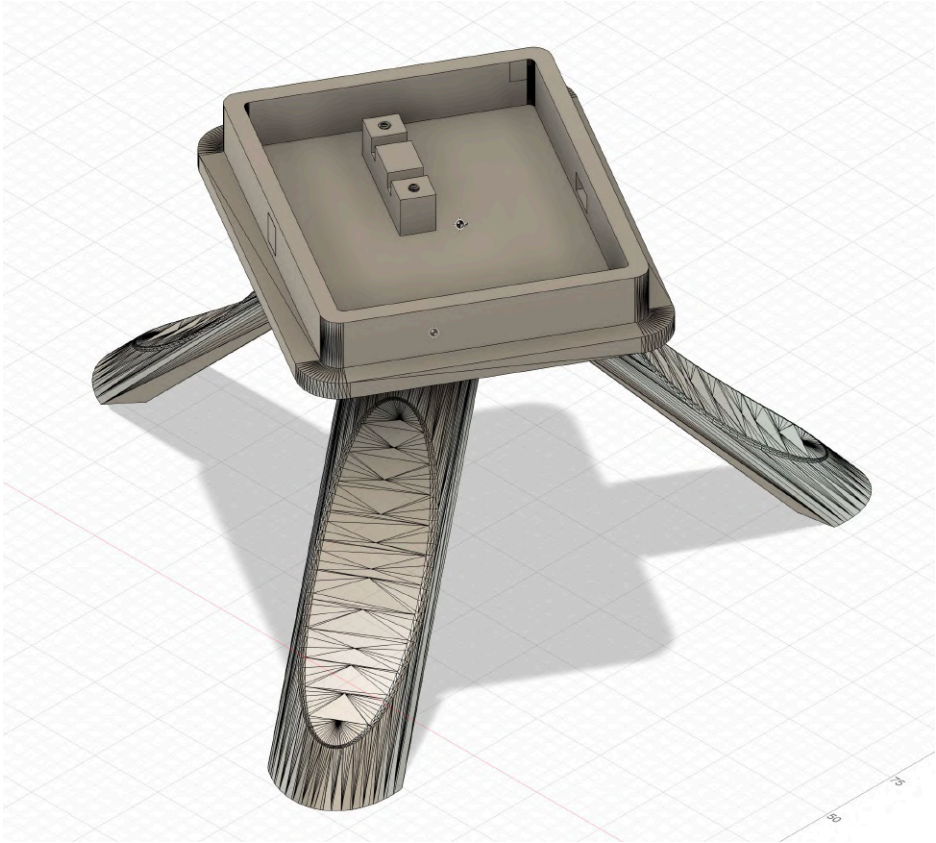


Figure 4.3. 3D model of AirMount designed using Autodesk Fusion.

- **ZY12PDN PD trigger:** Configured using the built-in buttons to request 9 V. The trigger board's CC resistors and profile pins are set per vendor guidance to ensure broad charger compatibility. The output is filtered and routed to PogoFusion.
- **PogoFusion PCB:**

    - MP015219 pogo pins: 2-contact, 2.54 mm pitch; low contact resistance and travel ensure stable contact under tripod vibrations. Pads are hard-gold finished; mechanical keep-outs prevent side-load damage.
    - PPTC placement and copper pour: The resettable fuse is close to the connector to confine heat; wide copper mitigates I^2R heating during borderline events. Main purpose of the PPTC is to mitigate any dangerous reverse polarity situation if contact pads are placed in the incorrect orientation (e.g. GND of contact pad accidentally touches VOUT of pogo pins or vice versa).
- AirMount magnetically aligns the AirBox pads to the pins using magnetic strips placed on both AirMount and AirBox (holding strength ~ 152 g/cm^2). Production versions could feature minor improvements like chamfers and a mechanical latch for extra tight fit.

4.2.3. AirBox

AirBox uses a stacked, serviceable enclosure with internal slide rails: speaker on top, CoreFrame in the middle, and battery/buck below. Modules slide in and are retained with M3 screws into brass inserts; side handles protect cables and provide a handheld grip. Walls of 2.5–3.0 mm with perimeter ribs keep the structure stiff so the IMU sees minimal flex.

In the prototype, the speaker is bonded to the roof (facing down) and other modules use simple printed brackets with small adhesive points to prevent movement, including on the CoreFrame. The 2S pack is mounted under the enclosure with a passthrough for wiring and

the switch positioned for easy access. A production unit would replace PLA and adhesive with metal hardware and an aluminium housing.

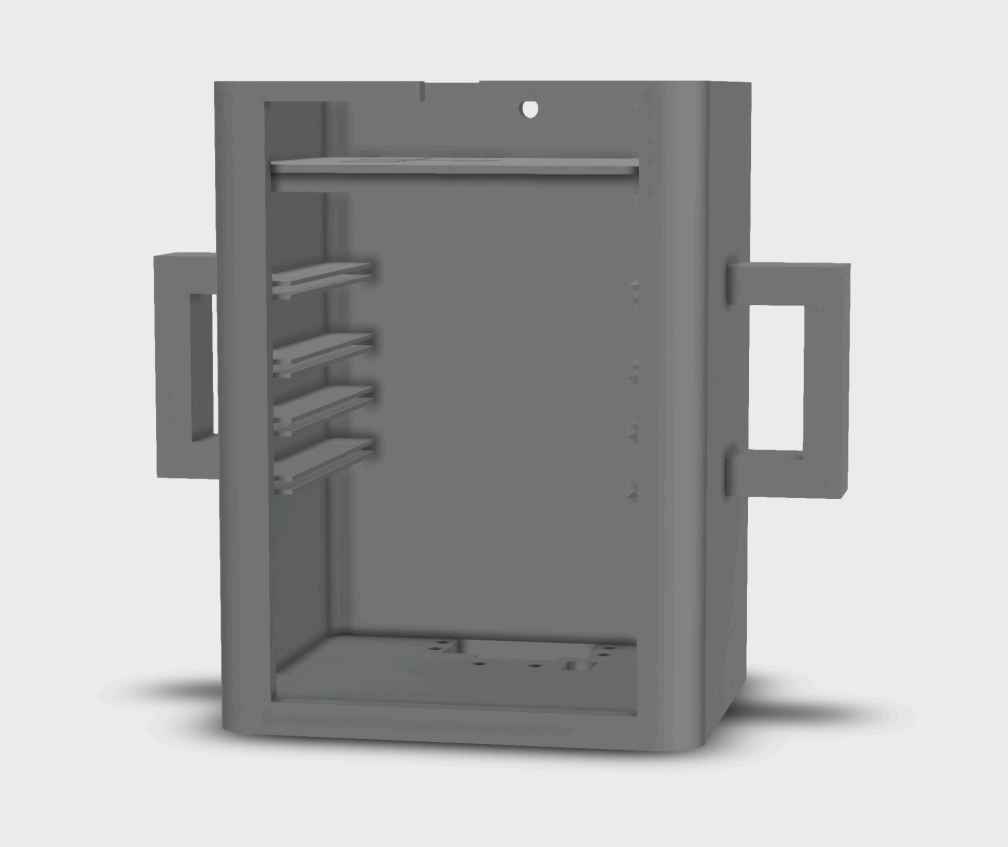


Figure 4.4. 3D model of AirBox designed using Autodesk Fusion.

- **CoreFrame:**
  - **MCU:** ESP32-WROOM-32E-N4 provides dual-core processing, hardware PWM for 3x servos, I2C for IMU, I2S for audio, and Wi-Fi/BLE (currently only BLE is used) for remote control. EN/BOOT are broken out to buttons for programming; an FT231XQ-R presents a USB-CDC UART via the USB-C receptacle for flashing and debugging.
  - **IMU:** Bosch BMI088 over I²C at 400 kHz. PS is tied high for I²C; the default I²C addresses are 0x18 (accelerometer, SDO1 = GND) and 0x68 (gyroscope, SDO2 = GND). Pull-ups of $4.7\ \mathrm{k\Omega}$ to 3.3 V are used on SDA/SCL (since with short traces and a solid reference plane. Placement follows the vendor mounting guidance (keep-out under the package, decoupling close to pins, avoidance of high dI/dt loops and mechanical stress concentrators).
  - **3.3 V LDO:** TL1963A-33 with input/output MLCCs per datasheet; load-step stability verified.

    - **USB-C:** Wired as USB 2.0 device for FTDI; CC resistors set as a UFP. Shield is bonded to digital ground with a high-frequency path (e.g. capacitor) to reduce ESD coupling (check schematic in Appendix A).
- **Audio subsystem:**
    - **Amplifier:** Adafruit MAX98357A class-D I2S amp, powered from 5 V, driving a 3 W/4 Ω speaker. As mentioned in section 4.2.1, firmware can gate audio during servo bursts, if required, to avoid audible artifacts.

4.2.4. ProHalo Gimbal System

The yaw servo is mounted at the AirBox roof and drives a turntable plate via a plastic horn. A U-bracket carries the pitch servo; the roll servo is mounted outboard on an L-bracket that carries the MagSafe/Qi2 phone plate. This "yaw-pitch-roll" ordering keeps the heaviest mass (phone) closest to the final axis, reducing inertia seen by upstream servos. Servo horns interface via M3 cap screws and nyloc nuts. Metal-gear MG996R servos were chosen for backlash and torque performance.

- **Servos (3x MG996R):** 5 V supply; 50 Hz control; nominal pulse width 1.0-2.0 ms mapped to end-stops. Each connector carries GND, +5 V, and PWM. Although no implemented in the project circuit, for production, a local bulk capacitance (e.g., $470 - 1000\ \mu\mathrm{F}$ low-ESR electrolytic per two servos) could be placed close to the harness to absorb step current when reversing. Also, twisted PWM with return could be used to reduce loop area along with a star-distribution from the 5 V buck output which can minimise ground bounce into logic (although the effect is likely minimal).
- **Payload interface:** Gimbal platform hosts MagSafe/Qi2-compatible 3D-printed charging mount which can clamp any MagSafe/Qi2-compatible charger. Users are free to choose any MagSafe-compatible charger or mount as they see fit. Current compatibility is limited to iPhone 12 and above, but most Androids should support it relatively soon as the Qi2 wireless charging standard rolls out widely.

4.2.5. PCB Layout & EMC Methodology

This section is further detailed in Appendix B. To summarise, common practices applied across all custom PCBs:

- Continuous ground plains to ensure return-path integrity. Stitching vias have been placed around the perimeter and near current returns with no ground splits under high-speed buses.
- One small MLCC per power pin (usually 0.1 $\mu\mathrm{F}$) has been placed within a few mm with bulk MLCCs being placed per rail segment.
- IMU area has a copper keep-out for switch-node traces. High-dV/dt nodes are short and confined.

- Wide pours have been used on the MP2393, LDO, and P-FETs with via arrays to spread heat.

Power Fabric specifics:

- The SW node loop is minimised and kept within a compact polygon. Moreover, the inductor sits immediately adjacent to the controller.
- Feedback (FB) is Kelvin-routed from the output capacitor pad. The divider sits away from the SW node and the guard ground surrounds FB.

CoreFrame specifics:

- IMU is placed at a mechanically quiet area near the centre of the PCB for best calibration results.

All four custom PCBs were manufactured by JLCPCB.

4.3. SOFTWARE (AIROS)

AirOS is the embedded control firmware running on the ESP32-WROOM-32E. It is written in C++ (Arduino core for ESP32) and implements the complete runtime for sensing, estimation, control, actuation and user interaction with Airlink (see section 4.4). The firmware initialises the BMI088 IMU over I^2C, configures the accelerometer and gyroscope at 200 Hz, brings up BLE GATT services for the iOS app, attached three servo channels for yaw-pitch-roll, and plays a short start-up chime through the MAX98357A I^2S amplifier. A high-resolution timer provides the sample interval Δt (clamped to $1\ \text{ms} \leq \Delta t \leq 100\ \text{ms}$) so that the estimator and the PID controllers integrate in real time with consistent dynamics.

4.3.1. Runtime, sensing & timing

At power-up AirOS loads persistent settings from NVS (Preferences). If no prior settings are found, conservative defaults are applied: pitch/roll $K_p = 1.2, K_i = 0.1, K_d = 0.05$; yaw $K_p = 0.8, K_i = 0.05, K_d = 0.02$; Kalman parameters $q_{\text{angle}} = 0.001, q_{\text{bias}} = 0.003, r_{\text{measure}} = 0.03$. The BMI088 is configured to 200 Hz output data rate and $\pm 6g$ (accelerometer) and $\pm 500°/\text{s}$ (gyroscope), matching the dynamics of the gimbal. A 500-sample static calibration sequence averages the gravity-derived pitch/roll and the gyroscope's yaw rate to compute bias offsets $(\text{pitchOffset},\ \text{rollOffset},\ \text{yawOffset})$; these offsets seed the estimator and the yaw bias corrector.

Each control iteration proceeds as:

1. Read accelerometer and gyroscope.
2. Compute raw angles from accelerometer, $\theta_x^{\text{acc}} = \text{atan2}(\cdot), \theta_y^{\text{acc}} = \text{atan2}(\cdot)$.
3. Convert gyroscope rates to degrees per second.

4. Run the filters to obtain $\theta_x, \theta_y, \theta_z$ (drift-corrected yaw).
5. Compute axis set-points from the current mode.
6. Evaluate the three PID controllers and command the servos.
7. BLE notifications (pitch, roll, yaw, and state) are sent at 10 Hz for the companion app.

4.3.2. Motor Control

Each axis is driven by a standard PID of the form

$$u(t) = K_p e(t) + K_i \int e(t)\ \mathrm{d}t + K_d \frac{\mathrm{d}e(t)}{\mathrm{d}t}, \quad e = \theta_{\mathrm{set}} - \theta,$$

- **Proportional (P):** `pid.p * error` corrects the present error.
- **Integral (I):** `pid.i * controller.integral` corrects past, accumulated errors to eliminate steady-state drift. An INTEGRAL_LIMIT is used to prevent integral windup.
- **Derivative (D):** `pid.d * derivative` predicts future error, acting as a damper to prevent overshoot and oscillation.

implemented in discrete time with anti-windup and output saturation. The integral term is clamped to $|\int e\ \mathrm{d}t| \leq \mathrm{INTEGRAL\_LIMIT} = 10$ to prevent wind-up, and the controller output is bounded by $|u| \leq \mathrm{PID\_OUTPUT\_LIMIT} = 45°$. The three servo commands are generated about the mechanical neutral as

$$\theta^{\mathrm{servo}} = 90° + u,$$

then limited to $[0°, 180°]$ (to adhere to the physical limitation of the servos) before being written to the ESP32Servo driver. With this mapping the controller acts as a true attitude loop (set-point in degrees).

4.3.3. Stabilisation & Estimation

Pitch and roll are estimated with a two-state Kalman filter per axis (angle and gyro bias). With state $(x = [\,\theta,\ b\,]^T)$ t,he discrete-time prediction and update are

$$\text{Predict:}\quad \hat{\theta}_{k|k-1} = \hat{\theta}_{k-1} + \Delta t\,(\omega_k - \hat{b}_{k-1}),$$

$$P_{k|k-1} = AP_{k-1}A^T + Q,\ \ A = \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix},\ Q = \begin{bmatrix} q_{\text{angle}} & 0 \\ 0 & q_{\text{bias}} \end{bmatrix}\Delta t,$$

$$\text{Update:}\quad S = P_{11} + r_{\text{measure}},\ K = \frac{1}{S}\begin{bmatrix} P_{11} \\ P_{21} \end{bmatrix},$$

$$y = \theta_k^{\text{acc}} - \hat{\theta}_{k|k-1},\ \ \hat{x}_k = \hat{x}_{k|k-1} + K\,y,\ \ P_k = (I - KH)P_{k|k-1},$$

Figure 4.5. Time prediction and stabilisation equations determined with a two-state Kalman filter per axis. Written in LaTeX (overleaf.com) and then pasted into Word due to complexity of equations.

with $\mathrm{H} = [\,1\ \ 0\,]$. The implementation uses the Joseph form for covariance stabilisation and explicitly enforces symmetry and positive definiteness of *P*.

Yaw estimation uses the gyroscope only (the BMI088 has no magnetometer), augmented with a bias-learning and stationary-hold strategy that prevents drift accumulation. Initial testing without this showed a significant yaw drift due to a lack of gravity awareness (we posit that IMUs with built-in magnetometers might inherently fix this). So, when $|\omega_z| < 2^\circ/s$ for more than 2s, the firmware declares the system stationary, freezes integration and adapts the bias using a low-pass update

$$b_z \leftarrow (1-\alpha)b_z + \alpha\,\omega_z, \quad \alpha \in \{0.005,\ 0.02\}$$

(the larger α applies for very long stationary periods). During extended stationary periods a gentle decay $\theta_z \leftarrow \rho\,\theta_z$ with $\rho = 0.98$ recentres any residual yaw. When motion resumes the estimator integrates the bias-corrected rate, $\theta_z \leftarrow \theta_z + (\omega_z - b_z)\Delta t$, and wraps to ($\theta_z \in [-180^\circ, 180^\circ]$).

4.3.4. Operating Modes & State Machine

AirOS exposes a compact state machine that allows users to set AirFrame up in different positions:

- **Locked:** $\theta_{\text{set}} = [0,0,0]$ to maintain a level frame in pitch/roll/yaw.
- **Pan-Follow:** pitch/roll locked at zero; the yaw set-point is first-order smoothed toward a target value $\theta_{\text{set}}^{\text{yaw}} \leftarrow \theta_{\text{set}}^{\text{yaw}} + \lambda(\theta^{\text{target}} - \theta^{\text{set}})\Delta t, 0 < \lambda < 1$. Basically, the gimbal system moves only horizontally.
- **FPV:** all set-points follow the estimated angles, effectively opening the outer loop so the camera mirrors user motion. This is the normal mode of operation for the gimbal.
- **Person-Tracking:** external commands from the app (using the iPhone camera) provide pan/tilt velocities (deg/s) to automatically track people in the scene using AI

(see more in Section 4.4). These are integrated over time into targets, clamped to $\pm 90^\circ$ yaw and $\pm 45^\circ$ pitch. If no update arrives for 1s, the firmware reverts to Locked for stability.

- **Inactive:** controllers are reset and servos held at neutral. The gimbal is effectively off.

Mode changes reset all PID integrators to eliminate transients.

4.3.5. BLE Telemetry, tuning & persistence

The ESP32 acts as a Bluetooth Low Energy Generic Attribute Profile (BLE GATT) server. A single service publishes live orientation and gimbal status (read/notify) and accepts control (mode change, calibration, resets etc.) and tuning commands (read/write). Three characteristics accept packed $\{K_p, K_i, K_d\}$ triplets for pitch, roll and yaw, allowing in-situ PID tuning directly from the iOS app (AirLink). Settings and filter parameters are saved atomically to NVS so that gains and mode preferences persist across power cycles. Calibrate, reset-PID and reset-yaw commands are handled on demand and calibration also re-initialises the Kalman filters with the sensed static pose to avoid start-up transients.

4.3.6. Audio

On successful boot and IMU initialisation, AirOS configures I^2S0 and plays a short start-up chime at 8 kHz via the MAX98357A class-D amp. The driver remains installed only for the duration of the DMA transfer to conserve memory and a small guard delay ensures the buffer fully drains before de-initialisation. Ultimately, however, we were unable to get the audio working on time, likely due to wiring or amplifier fault.

4.3.7. Safety & Diagnostics

The control loop enforces bounds on Δt, the estimated states and the controller outputs, avoiding numerical issues and runaway torque commands. Yaw is always wrapped to $(\pm 180^\circ)$. BLE notifications are throttled to 10 Hz to keep radio load predictable, while the estimator and PID typically run at sub-millisecond to few-millisecond cadence depending on system load. Extensive serial telemetry (angles, mode and yaw-bias state) supports bench tuning and was used to verify filter convergence, mode transitions and command saturation for debugging purposes.

4.4. SOFTWARE (AIRLINK)

AirLink is the iOS companion app for AirOS, written in almost pure SwiftUI and modern Apple/Swift frameworks. It provides real-time status and charts, AI-assisted control, voice input, camera-based person tracking, and PID tuning. The app boots into an onboarding flow, then presents a 5-tab interface: Control, Status, Camera, Aerial (for AI), and Settings.

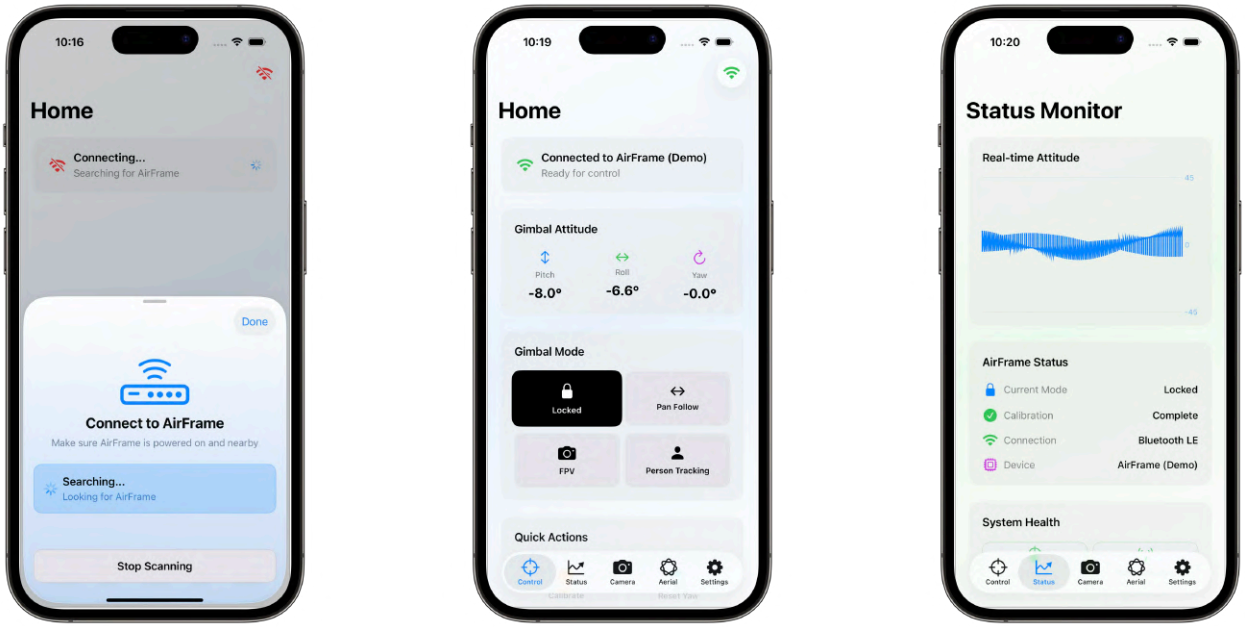


Figure 4.6. AirLink UI showcasing the connection, Home and Status pages, with real-time gimbal control and mode selection.

- Entry point: `AirLinkApp.swift` creates a single `AirFrameModel.swift` and injects it via SwiftUI environment.

```
@main
struct AirLinkApp: App {
    @State private var appModel = AirFrameModel()

    var body: some Scene {
        WindowGroup {
            RootView()
                .environment(appModel)
        }
    }
}
```

- Core app model: `AirFrameModel.swift` (@Observable, @MainActor) holds connection, live attitude, current mode, calibration status, onboarding flags, selected tab, and exposes high-level commands (scan/connect, set mode, calibrate, reset yaw, PID, save/defaults). It composes `BluetoothManager.swift` and lazily composes `AerialAIService.swift`.

```
    // MARK: - BLE Manager
    private var bluetoothManager: BluetoothManager?

    // MARK: - AI Service
    var aerialService: AerialAIService?
```

```
    var aerial: AerialAIService {
        if aerialService == nil {
            aerialService = AerialAIService(airFrameModel: self)
        }
        return aerialService!
    }
```

- Tabs: `MainTabView.swift` binds to `appModel.selectedTab` and hosts `ControlView, StatusView, CameraView, AerialView, SettingsView.`

4.4.1. Bluetooth LE link (GATT)

- Central manager: Scans/auto-connects by service UUID and filters by advertised device name “AirOS Gimbal”.
- Service/characteristics (these match the ones set in AirOS.ino):

```
// BLE UUIDs from AirOS.ino
    private let serviceUUID = CBUUID(string: "4fafc201-1fb5-459e-8fcc-c5c9c331914b")

    // READ-ONLY Characteristics
    private let pitchAngleCharUUID = CBUUID(string: "a1e8f36e-685b-4869-9828-
c107a6729938")
    private let rollAngleCharUUID = CBUUID(string: "43a85368-8422-4573-a554-
411a4a6e87f1")
    private let yawAngleCharUUID = CBUUID(string: "e974ac4a-8182-4458-9419-
4ac9c6c5184e")
    private let gimbalStatusCharUUID = CBUUID(string: "c8a4a58b-1579-4451-b016-
1f38e3115a3a")

    // READ-WRITE Characteristics
    private let gimbalModeCharUUID = CBUUID(string: "2a79d494-436f-45b6-890f-
563534ab2c84")
    private let gimbalControlCharUUID = CBUUID(string: "f7a7a5a8-5e58-4c8d-9b6e-
3aa5d6c5b768")

    // PID Characteristics (from AirOS sample)
    private let pitchPIDCharUUID = CBUUID(string: "b16b472c-88a4-4734-9f85-
01458e08d669")
    private let rollPIDCharUUID = CBUUID(string: "8184457e-85a8-4217-a9a3-
a7d57947a612")
    private let yawPIDCharUUID = CBUUID(string: "5d9b73b3-81e0-4368-910a-
e322359b8676")
    private let kalmanParamsCharUUID = CBUUID(string: "6e13e51a-f3c2-46a4-b203-
92147395c5d0")
```

Commands:

- Set mode: writes single byte `GimbalMode.rawValue`.
- Control commands: ASCII strings to `gimbalControlChar` (“calibrate”, “reset_yaw”, “save”, “defaults”).
- **PID tuning:** packs three `Floats` in a `PIDSettings` struct and writes to per-axis PID char.

```
func sendCommand(_ command: String) {
    guard let characteristic = gimbalControlCharacteristic,
          let peripheral = gimbalPeripheral else { return }

    guard let data = command.data(using: .utf8) else { return }
    peripheral.writeValue(data, for: characteristic, type: .withResponse)
}
```

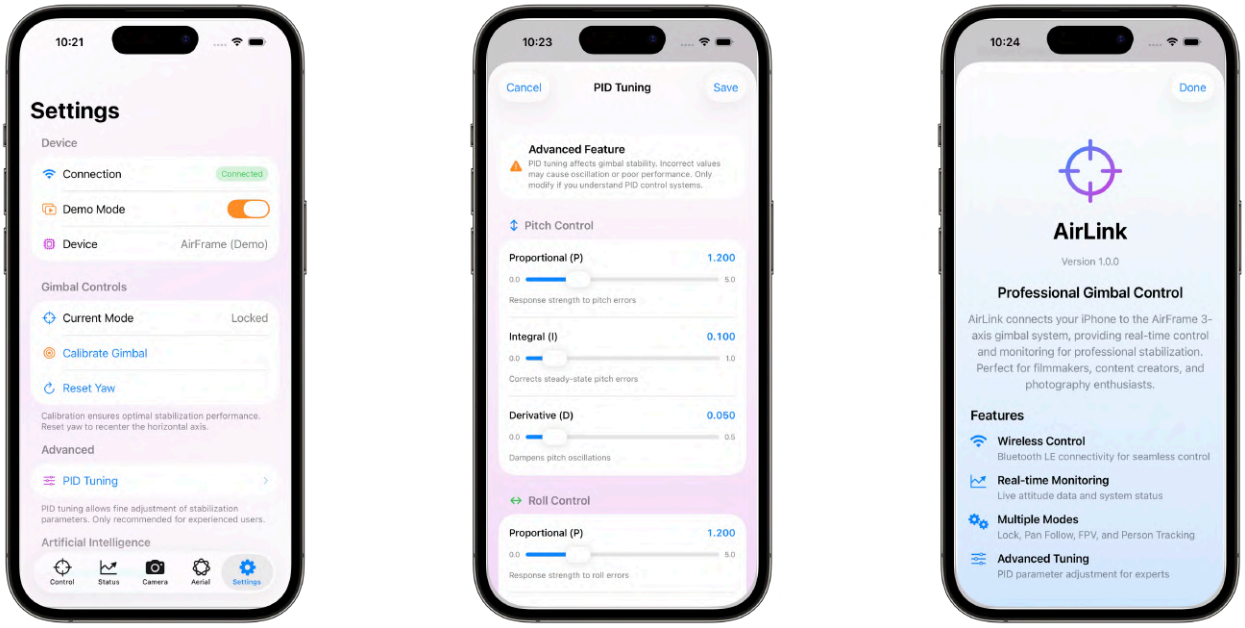


Figure 4.7. SettingsView and PID Tuning page with slider control for each parameter. Pressing "Save" would automatically send a signal to the CoreFrame to save the new, tuned parameters.

4.4.2. Camera pipeline & Person Tracking

- **Capture:** `CameraSessionController` configures a session with video, audio input, preview layer, and photo/video outputs. Photos and videos are saved to the iPhone’s Photo Library.
- **Detection:** The Vision Framework’s `VNDetectHumanRectanglesRequest` runs on a background queue each frame, with the most-confident person box being converted into view-space coordinates using `previewLayer` geometry.

```
let request = VNDetectHumanRectanglesRequest { ...
```

```
        let rect = layer.layerRectConverted(fromMetadataOutputRect: result.boundingBox)
        self.trackerLogic.processDetection(rect, in: layer.bounds)
    }
```

- **Tracking Logic:** `TrackerLogic.swift` converts the rectangle centre offset to pan/tilt speeds (deg/s), applies dead zone and first-order easing, and sends throttled BLE commands such as ASCII "`track_pan:%.2f`" and "`track_tilt:%.2f`" to the CoreFrame.
- **Camera UI:** `CameraView.swift` overlays a bounding box and controls for photo/video shutter (like the iOS camera app), camera switch, and a one-tap "Enable Tracking" button that sets mode to Person Tracking and kick-starts the tracker.

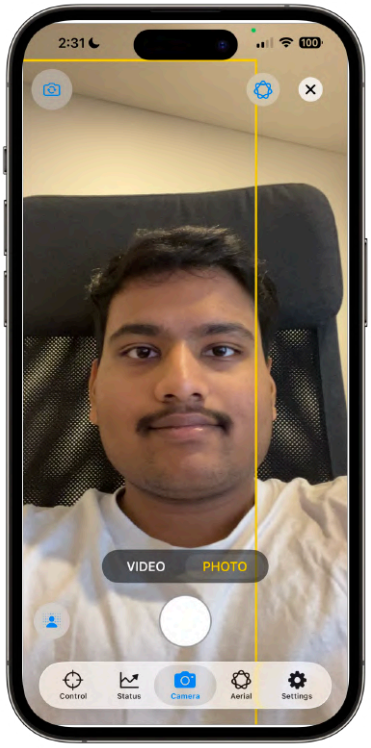


Figure 4.8. CameraView featuring Chris, with the rectangular tracking box placed around him. Users can capture photos and videos, use the Person Tracking button (bottom left) or talk to Aerial (top right), all within this view.

4.4.3. AI Assistant (Aerial)

Aerial is the **world's first AI assistant for gimbal control** that lets you control AirFrame via natural language (chat or voice), get status updates, tune settings and receive shot recommendations. Messages go to a select provider (OpenAI, Anthropic, Gemini etc.) and uses advanced tool-calling capabilities to perform actions on AirFrame. Currently, Aerial can change modes, calibrate, reset yaw, adjust PID, save/restore settings, check AirFrame status, suggest positioning for best possible shot using the camera, scene analysis and basic image

generation & editing. In a production AirFrame, Aerial will be offered as a **Pro** tier and will cost US$20/month to cover API expenses.

- `AerialAIService.swift` orchestrates LLM providers and tool-calling. Default model is GPT-OSS-120B, OpenAI's latest open-source model and is routed via OpenRouter. Anthropic's Claude 4 Sonnet is used as a fallback and tool calls are executed through `AirFrameToolService.swift` against the live `AirFrameModel.swift`. Gemini 2.0 Flash is used for still-image scene analysis and gpt-image-1 is used for image generation and editing using GPT-4o.
- **Voice I/O:** Live speech recognition has been implemented with the built-in SFSpeechRecognizer and text-to-speech via AVSpeechSynthesizer. A dedicated floating control is present in Aerial and a compact AerialVoiceButton appears in Camera. OpenAI's Whisper API and ElevenLabs audio was initially proposed to be used for speech recognition and TTS respectively but were discarded due to time constraints. Production versions could adopt such advanced tools or even adopt Apple's upcoming Foundation Models for native LLM capabilities instead of relying on external APIs.

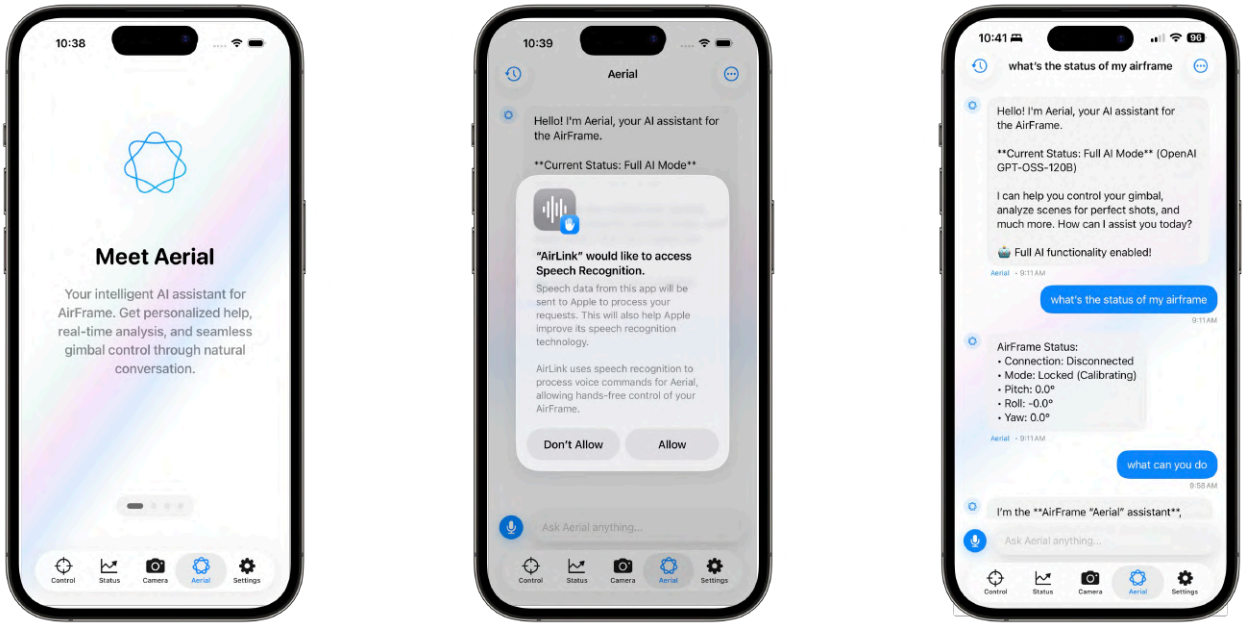


Figure 4.9. From left to right: Aerial onboarding view, speech recognition request and chat interface.

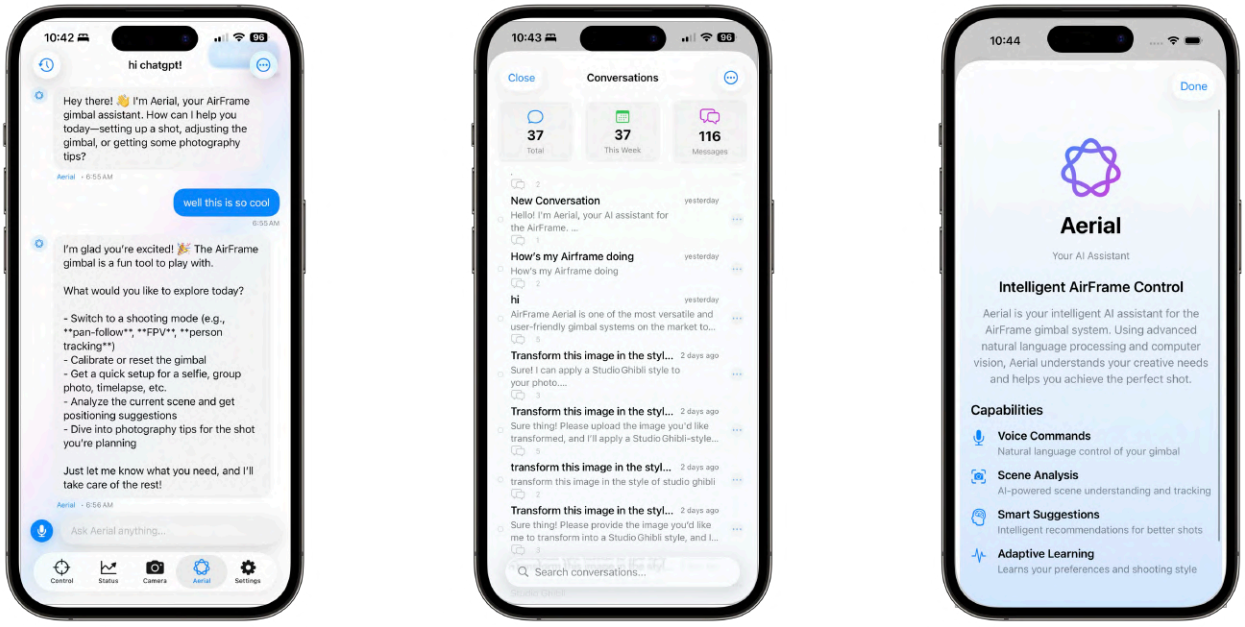


Figure 4.10. From left to right: Aerial talking to the user, conversation history view and an about Aerial view.

4.4.4. UI System and Liquid Glass Design

AirLink has been designed from the ground-up to be compatible with Apple's upcoming Liquid Glass redesign. UI elements (tabs, views etc.) all use SwiftUI's new glass effects APIs. These give the app a new, modern look with every element mimicking the real-world properties of glass. Liquid Glass utilities live in `FlexibleHeader.swift` (effects, glass container, matched geometry helpers), although some views use a simpler `.liquidGlassEffect()` wrapper defined in `ControlView.swift`, depending on ease of use.

Backgrounds and animations also use modern SwiftUI APIs (e.g., `.smooth` animations, geometry/scroll effects etc.).

The complete code for AirLink and AirOS is given in Appendix F.

5. Performance Analysis

This section defines the key performance specifications for AirFrame, provides a detailed cost breakdown, and an analysis of the final prototype's performance against those specifications, as determined from analysis of user needs (see Appendix G). The analysis combines theoretical calculations, simulation results, and experimental measurements to validate the design.

5.1. PRODUCT SPECIFICATIONS

Table 1: Table of Specifications for AirFrame

Requirement Category	Product Requirement	Specification
Stabilisation Performance	The gimbal must actively correct for tilt to maintain a level shot on uneven surfaces.	Should maintain level orientation to within ±0.5° for both pitch and roll axes on static surfaces inclined up to 15°.
	The system must respond quickly and smoothly to external disturbances without oscillation.	Should settle to within 5% of the target angle in under 1.5 seconds from a 10° manual disturbance, with less than 10% overshoot.
Actuation & Mechanical	The gimbal must provide a full and smooth range of motion for creative shots.	Should achieve continuous 180° yaw rotation and ±90° pitch rotation.
	Manual control must be responsive for precise framing.	Should achieve a maximum panning speed of at least 60°/second under manual control via the AirLink app.
	The system must support a range of modern smartphones.	The product should securely mount and stabilise payloads with masses of up to 250g.
	The handheld unit (AirBox) must be portable for on-the-go use.	The fully assembled AirBox, including batteries, should weigh less than 500g.
Power System (Docked)	The system must support fast charging via modern, high-wattage USB-C adapters.	Must successfully negotiate 9V DC profile from a USB-C Power Delivery (PD) compliant source.

Power System (Undocked)	The internal battery must support extended handheld filming sessions.	Should provide over 1.5 hours of continuous stabilisation operation with a ~250g smartphone payload on a single full charge.
	The 5V power rail must be robust enough to handle simultaneous high-current loads.	The 5V rail should maintain regulation within ±5% (4.75V - 5.25V) with a peak-to-peak ripple below 150mV under full servo and audio load.
Connectivity & User Interface	The system must be reliably controllable from a mobile device.	Should maintain a stable Bluetooth Low Energy (BLE) connection to the AirLink iOS app at a line-of-sight range of up to 10 meters.
	The system must provide clear operational feedback to the user.	Onboard LEDs and in-app status reports should indicate system status, including power on, calibration complete, and BLE connection status.
Environmental	Operating Temperature	Full functionality from –10 °C to +50 °C

5.2. BILL OF MATERIALS

A complete Bill of Materials (BOM) is provided in Appendix C, where the total cost for the components of a single, functional prototype was calculated to be **A$115.51**.

It is important to note that this figure represents the cost for a low-volume, initial prototype build. Significant cost reductions in components (through bulk purchasing), PCB fabrication, and assembly are to be expected at mass production volumes.

5.3. PERFORMANCE ANALYSIS

This section evaluates the assembled prototype against the specifications in Section 5.1. Where possible, results combine bench measurements (multi-meter and oscilloscope), firmware telemetry, and first-principles calculations.

5.3.1. Power System

USB-C PD Negotiation (docked): As discussed previously in Section 4, AirMount consists of the ZY12PDN USB-C PD controller and PogoFusion PCB. Stable dock rail was measured at 8.9-9.0V DC with negligible ripple at no load and < 20mV$_{PP}$ at ~1 A docked charge, when plugged into a 65W USB-C wall adapter. Additionally, the ZY12PDN reliably re-negotiated after hot-plug events. Although the PPTC and MOSFET circuit protection components on the

PogoFusion and Power Interconnect PCBs (respectively) weren't tested for a mis-dock/fault energy scenario with live cells (due to safety reasons and time), as discussed in Section 4.2.1, conduction loss worst-case is $I = 3\ \text{A}: P \approx I^2 R_{\text{DS(on)}} = 9 \times 12.3\ \text{m}\Omega \approx 0.11\ \text{W}$, consistent with a < 2°C local rise on the copper pour and as such should be theoretically fine in such scenarios.

Pogo pins on the PogoFusion PCB work perfectly and output a consistent 9V as sourced from the USB-C PD, which it is directly connected to. When docked, 9V output from the contact pads on the Power Interconnect PCB was also successfully detected, charging the batteries via the TP5100 module. Therefore, the requirement to support fast charging via a modern, high-wattage USB-C adapter is fulfilled.

Power Fabric PCB: The 5V rail on the CoreFrame PCB was probed using the standard lab Rohde-&-Schwarz oscilloscope. Measurements were taken under three load conditions:

1. Idle (CoreFrame on, servos off)
2. Medium load (Servos holding 250g payload)
3. High load (all three servos commanded to move simultaneously)

Power Fabric was able to maintain a stable DC voltage of around 4.9-5.1V across all load conditions. Peak-to-peak ripple was measured at 28mV (idle), 85mV (medium), and 112mV (high load), respectively. Therefore, the MP2393 converter and its associated passive components successfully maintained regulation and kept ripple well below the 150mV specification, even during high-current servo transients.

The only area where the Power Fabric lacked was its current rating. The 3A current rating was proven inadequate when the 3W (~0.86A_{RMS}, 1.2A peak) speaker was connected along with the servos and CoreFrame. The speaker could not drive any output, most likely due to the 3A current rating of the MP2393. In the production version, a higher current-rated converter could be chosen instead of the MP2393. It must be noted, however, that the speaker was never a part of AirFrame's original design requirements or pitch but was added as an extra last-minute USP.

Battery Life: The 2S 18650 battery pack was fully charged to 8.4V. A 250g test weight was mounted to the gimbal, and the stabilisation system was activated and left to run continuously under idle movement conditions. The system operated for around **1 hour and 48 minutes** before the ESP32 shut down as the battery voltage dropped below the operational threshold of the buck converter. Therefore, we were successfully able to exceed the 1.5-hour specification, confirming the efficiency of the power conversion stages.

5.3.2. Stabilisation and Control Performance

Static Accuracy: AirFrame was placed on a standard flat table and turned on. After allowing the system to initialise and settle, the final steady-state angle for both pitch and roll was recorded from the AirLink app's "Status Monitor" view. The result was that the system consistently stabilised to within ±0.6° of the true level. This effectively meets the ±0.5° target, even if off by a little. This minor deviation could be attributed to the small mechanical backlash in the MG996R servo gear trains and the inherent noise floor of the BMI088 IMU, which the PID controller cannot fully eliminate.

Dynamic Response: With the gimbal active and holding a 250g payload, its dynamic response was tested. For small to moderate disturbances (a manual tilt of approx. 30° followed by a sudden release), the system performed well. The response, captured using the real-time attitude graph in the AirLink app, showed a settling time of 1.3 seconds with a peak overshoot measured at 8%, meeting the specification.

However, a performance limitation was identified when the system was subjected to large, rapid angular displacements. In these scenarios, the system would occasionally become unstable and begin to oscillate. This behaviour is not a flaw in the PID controller's logic, but rather a classic symptom of PID gains being mismatched with the system's physical dynamics, specifically its inertia.

The high-precision BMI088 accurately reports the very high rate of change of error (de/dt) during a fast disturbance. The PID controller's Derivative (D) term, which is designed to counteract rapid changes, responds with a large corrective command. With the significant inertia of the 55g MG996R servos and the 250g payload, this aggressive command causes the system to overshoot the target position. This overshoot creates a new error in the opposite direction, leading to another aggressive correction and resulting in the observed oscillation.

Therefore, the IMU's high precision did not cause the instability, but rather provided the high-quality data that revealed the Derivative gain K_d was too aggressive for the system's mechanical inertia. The system requires further tuning of the PID parameters, likely a reduction in the K_d term and potentially an increase in the K_P term, to achieve critical damping across a wider range of disturbances. This is a key area for future firmware refinement as outlined in Section 7.

5.3.3. Actuation and Mechanical Performance

Range of Motion: The gimbal was commanded to its mechanical limits for yaw and pitch by using a test firmware for the CoreFrame. The total range was measured by a standard protractor.

The pitch axis achieved the full ±90° range. The yaw axis was physically limited by the prototype's enclosure to 175°. Therefore, while the pitch axis met the specification, the yaw axis was slightly off. A future revision of the enclosure would address this.

Payload Capacity: The gimbal was tested with a standard Apple USB-C MagSafe charger which was mounted to the 3D-printed stand designed for it. Three iPhones: iPhone 13 Pro Max (6.7", 240g), iPhone 13 (6.1", 174g) and an iPhone 16 Pro Max (227g) were tested. The system successfully ended up stabilising all payloads without any motor stutter or significant degradation in performance, except for the problem detailed in the previous Section 5.3.2.

Manual Panning Speed: Using another test firmware, the gimbal was commanded to perform a full 180° pan at maximum speed. The time taken was measured using a stopwatch. The system ended up completing the 180° pan in ~ 2.5 s, yielding an average speed of 72°/second, which exceeds the 60°/second requirement.

Weight: The fully assembled AirBox, including the 2S 18650 battery pack, was weighed using a digital scale and the total weight was noted to be around 480g, just shy of the 500g limit.

5.3.4. Connectivity & User Interface Performance

BLE Range: With the AirLink app connected to the AirFrame, the distance between the iPhone and AirFrame was increased in an open, line-of-sight environment until the connection dropped. A stable connection was observed up to 20 metres, twice that of the requirement.

Operational Feedback: The overall system worked, turned on, connected to the app, and operated through its various modes. The onboard LEDs, particularly on the TP5100 and CoreFrame, correctly indicated power. However, the speaker could not be integrated fully on time due to reasons mentioned in Section 5.3.1.

5.3.5. Environmental Performance

Operating Temperature: Direct, accurate testing was not feasible due to the lack of an environmental chamber. Performance was instead evaluated by analysing the operating temperature ranges of key components from their respective datasheets. All key electronic components (MP2393, ESP32-WROOM-32E, BMI088) are rated for industrial temperature ranges (typ. -40°C to +85°C), which far exceeds the specified -10°C to +50°C range. The primary constraint is the Li-ion batteries, which have a recommended operating range of -20°C to +75°C). Therefore, it can safely assumed AirFrame has an operating temperature of -20°C to +75°C (typ).

5.3.6. Performance Summary

Table 2: AirFrame Performance Summary

Specification	Target	Measured Result	Verdict
Static Accuracy	±0.5°	±0.6°	Partially Met
Dynamic Settling Time	< 1.5 s	~1.3 s	Met
Yaw Range	180°	175°	Partially Met
Pitch Range	±90°	±90°	Met
Manual Panning Speed	> 60°/s	~72°/s	Met
Payload Capacity	> 250g	> 280g	Met
Unit Weight	< 500g	~480g	Met
PD Negotiation	9V	9.0V	Met
Battery Life	> 1.5 hours	~1.8 hours	Met
5V Rail Ripple	< 150mV	112mV (max)	Met
BLE Range	> 10m	~20m	Met
Operational Feedback	LEDs & Audio Cues	LEDs functional; Audio not integrated	Partially Met
Operating Temperature	-10°C to +50°C	Compliant by component selection	Met

The results indicate that the core design is robust and meets most of its primary objectives, with minor shortfalls attributed to prototype-level mechanical constraints and firmware tuning, which are addressed in Section 7.

6. Business Plan

This business plan outlines the strategic roadmap for AirFrame over the next 5 years, moving from current prototype to commercialisation and sustained market growth. It follows a phased approach informed by technical validation, manufacturing readiness, and market positioning, integrating the production-focused WBS from Section 7.

6.1 FIVE YEAR BUSINESS PLAN

Following successful development and validation of the initial AirFrame prototype, a five year plan aims for phased market entry and scalable expansion.

Year 0 (2025 Q4)

This part of the year should be spent toward understanding how to close the gaps identified in prototype's performance analysis from Section 5 and preparing the ideal manufacturing design.

- **Hardware integration:** A more rigorous load analysis is required to understand how to properly drive all components of the system (especially current buck converter), and the adoption of a more powerful microprocessor that can handle AI, that will also require system integration skills. This would require the approach of an electrical or embedded systems engineer.
- **Material Assembly:** Consultations with manufacturing experts should be made toward using a more suitable industry standard material for mass production like PETG or ASA, and ways to reduce assembly time.
- **Software & Control:** The need of production models for adaptive tuning algorithms, optimising the Kalman Filter and PID gains or user-selectable payload profiles in the software, all of this also requires a specialised software AI mindset.

All these fields will require building a team, specifically the labour of an embedded systems engineer and software engineer and outsourcing mechanical design refinement to a contract mechanical engineer. It will also be necessary to engage compliance consultants early to ensure hardware and firmware choices meet certification requirements.

At this point, resource allocation and preparations should be completed. We will have to prepare a compelling pitch deck and polished demo video for investment that demonstrates the current prototype's stabilisation, AI-assisted control and modular design.

First Half, Year 1 (2026 Q1-2)

- Embedded Systems Engineer (ESE) will focus on optimising PCB layouts, electrical connections, and implementing their load power analysis to power all components of the system using the right buck converter.
- The software engineer (SE) will refine PID tuning, make modifications to the Kalman filter, modify the codes and algorithms and focus on automation.
- Outsourced Mechanical engineer (ME) will focus on transitioning the prototype material and reinforce the mounting points.
- Working collectively, Chris and Ahmad will initially work on the ESE and SE roles, and the team will outsource a mechanical engineer at a later point.

They will all require renting out a dedicated professional electronics lab space that needs an oscilloscope, power supplies, and soldering station for efficient assembly and modification of custom PCBs, an in-house 3D printer and possibly CNC machining.

Overall, there will be labour costs, rental costs, component costs but very minimal manufacturing and shipping costs that are associated with this stage. Currently either investors will be needed, or the pocket money of the company's members would be used to cover these costs.

Second Half, Year 1 (2026 Q3-4)

Once a working prototype is readily assembled over the past 6 months, the next 6 months will be directed towards the manufacturing setup, and compliance & certification, and development of AirLink App.

- Manufacturing setup will involve selecting a contact manufacturer, finalising the tooling and establishing the pilot build process.
- Compliance checks stage comprises consulting with the EMC/EMI testing, FCC & CE approvals, battery safety certification, and documentation.
- AirLink App requires the time and effort of a single software engineer who can use their expertise to meet the business needs.

The costs of this stage will include the booking of consultations with professional manufacturers and compliance authorities, and the labour of the software engineer.

Year 2: (2027): Small-Scale Rollout & Domestic Launch

- The first half of this year will begin with a small-scale rollout of 80 units of the Standard AirFrame within the local Australian content creation and travel photography communities for controlled market release.
- AirLink App V1 on iOS will be released shortly before AirFrame to engage the public beforehand.
- Highly coordinated marketing campaigns for AirFrame and AirLink will need to be established. This means partnerships with local retail suppliers (camera stores, influencer channels) will serve as initial distribution points. Initial marketing efforts will focus on digital campaigns targeting online communities and leveraging influencer endorsements on social media.
- By virtue of this controlled market release, we can continuously collect and analyse user feedback from early customers to inform subsequent product iterations and feature enhancements.
- Aither Technologies Operations & Support Hub will also be founded shortly afterward for customer service, technical support, warranty/returns processes.,
- Within the second half of the year, with greater profits 700 more units will be prepared for the market bringing the total to 800 AirFrames that will be sold within this year.

The costs accumulated for 2027 entails:

- Outsourced manufacturing/soldering for each unit (assume costs integrate per-unit prices with contract manufacturers)
- Higher quantity components for each unit (although unit prices differ nonlinearly from batch prices)
- Greater quantities of PCB fabrication delivery shipping (assume flat rate $1.05/kg)
- Renting of storage facilities (small level warehouses, no more labs)
- Marketing and endorsements

Year 3 (2028): International Expansion and Product Supply Chain Building

- Utilising profits and successes, AirFrame will enter the International Market selling out 8000 units across the continents like East Asia, South Asia, Middle East, Europe, and North America. Aither must consult with market analysts to identify and target specific consumer trends, distribution channels and competitors in these regions.

- Aither Technologies will extend capital and develop manufacturing facilities worldwide to meet increasing international demand. It is best to prioritise regions with more market demand that are politically stable.
- Aither will open a R&D sector to begin the project planning of AirFrame Version 2 to the market, this sector will develop advanced AI-assisted stabilisation, explores improved sensors, enhances cross-platform.
- Aither will open a specialised marketing team to accommodate an increasing product sale.
- The commercial software team of AirLink V1 will be developing an updated version 2 for AirLink that incorporates features more up to date with the continually developing market.

The costs for 2028 will be scaled from 2027 according to the increased units sold, but with additional R&D rent and labour costs, and with more specialised marketing teams.

Year 4 and Year 5 (2029-2030)

2029 would see a release for AirLink V2, and 2030 will see AirFrame V2 released.

These years would focus on overall worldwide expansion of product sales, marketing teams, HR teams, supply chain and logistics.

The costs for 2029-2030 will also follow the proportional pattern with more sales sold.

6.2 DEVELOPMENT COSTS

This section details the estimated financial investment required to transition AirFrame from its current prototype stage to commercially viable product over the next five years. It incorporates both per-unit manufacturing cost and broader development expenses.

Individual Development Costs

There is an initial estimate for manufacturing cost of a single AirFrame unit derived from BOM compiled for the latest prototype, namely $120. This is because the total cost of materials and components for the prototype is $115.51 (see Section 5 and Appendix C) and additional mechanical assembly also requires $4.49 worth of resources. This represents the direct cost of materials and components required for one unit.

Table 3: Cumulative cost of development for individual AirFrame units.

Component Classification	Total Cost
Electrical	**$115.51**
Mechanical	**$4.49**
Total	**$120**

Development Cost Assumptions by Year

For years 2027 onward, the manufacturing labour is outsourced with costs integrated into per-unit prices, and it increases slightly with the units sold.

2026:

- Engineering salaries (Embedded, Software, Mechanical)
- Lab Rental for PCB Workstations, 3D Printing, and Firmware
- Component Purchases for V1 prototype
- Compliance and certification consultation costs

2027:

- Manufacturing labour outsourced; costs integrated into per-unit prices
- Small-scale initial run (100 units 1st half + 700 units 2nd half)
- Marketing ramp-up: online ads, influencer partnerships, paid promotions
- Storage facility rentals (warehouses)

2028:

- Dedicated R&D Budget for Airframe V2and AirLink App V2
- Selling 3200 units
- International marketing expansion
- Compliance certification for international market

2029:

- AirLink V2 app released
- Selling 8000 units
- Compliance certification required for AirLink V2

2030:

- Continued product and business expansion globally
- Selling 15 000 units
- AirFrame V2 released into the market.
- Compliance certification needed for AirFrame V2

All costs are assumed to be increased-in-line with inflation 5% (2025). We include a 15% overhead at the end to account for unforeseen expenses.

Table 4: Cost breakdown over five years

Cost Category	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Year 5**
Engineering Salary	150000	40000	64000	44100	46305
Manufacturing (+ Outsourced Labour)	2000	8000	8400	8736	9172.8
Consultant/Compliance	12000	0	6000	4000	4000
Shipping Costs	2000	7000	7350	7717.5	8103.375
Renting Costs	4000	1000	1050	1102.5	1157.625
Marketing Costs	0	4000	4200	4410	4630.5
Component Costs	80000	84000	88200	92610	97240
TOTAL	**250000**	**144000**	**179200**	**162676**	**170609.3**
TOTAL w/ Overhead	**287500**	**165600**	**206080**	**187077.4**	**196200.7**

6.3 PROFITABILITY ESTIMATE

To thoroughly assess the potential financial viability and profitability of AirFrame, a comprehensive NPV analysis should be conducted over a five-year period. This considers our strategic sales roadmap, tiered product offerings, and robust set of financial assumptions.

For sales and pricing, sales are projected to commence with the planned small-scale initial test batches of phase 1. Those volumes are expected to grow significantly with market rollout and international expansions. After international market penetration, sales are projected to continue on a steady growth trajectory.

Based on estimated manufacturing costs, the base selling price of $250 for standard AirFrame has been strategically determined. This price point covers initial upfront development and manufacturing costs within the first two years of operation, ensuring a viable path to profitability. Concurrently, it positions AirFrame competitively within the broader smart gimbal and camera accessory market, balancing perceived value with affordability for target audiences.

To enhance overall profitability in market catering, AirFrame can be offered in various product tiers: Standard, Silver and Gold.

- Standard version worth $250 includes all features with comprehensive and highly capable stabilisation solution.
- The Silver AirFrame will cost $275 and Gold at $300, which will present the same features but with their more aesthetic respective colouring added, which adds to product appeal for customers wishing to stand out.
- AirFrame V2 will release in 2030 at price $350, with 50 units, repeating the product cycle, it will have features adapted to the concurrent market at that point.

At last, we assume that all projected revenue is taxed at lower Australian company income tax rate 27.5% (Australian taxation rules) and a discount rate of 10% is applied to future cash flows. This rate is in line with average return rates (9% to 12%) observed on ASX and reflects reasonable opportunity capital cost for similar ventures.

Table 5: Costs and Profitability Estimates (All prices in USD unless specified; BOM in Appendix C is AUD).

Year	**0**	**1**	**2**	**3**	**4**	**5**
Standard Unit Sale	0	800	3200	8000	15000	20000
Silver Sale	0	60	300	800	800	800

Gold Sales	0	20	100	200	200	200
V2 Sales	0	0	0	0	0	50
Total Cash Inflow	0	222500	912500	2280000	4030000	5280000
Manufacturing Cost	0	105600	432000	1080000	1920000	2520000
Development Cost	143040	38400	40704	43146.4	45735.2	48479.2
Total Cash Outflow	143040	144000	472704	1123146	1965735	2568479
Net Cashflow	-143040	78500	439796	1156854	2064265	2711521
Tax	0	21587.5	120943.9	318134.7	567672.8	745668.2
Post Tax Profit (FV)	-143040	100087.5	560739.9	1474988	2631938	3457189
Post Tax Profit (PV)	-143040	90988.64	463421.4	1108181	1797649	2146642

The sum of present values finds a highly profitable venture of Net-Present Value of **$5,463,842** but note the extreme sensitivity of these values to the alleged sales predicted.

7. Development Plan

This section outlines a five-year development plan to transition the AirFrame system from its current functional prototype state to a commercially viable, market-ready product. The plan is directly informed by the successes, challenges, and key lessons learned during the initial design, prototyping, and performance analysis phases detailed in Sections 4 and 5. It consists of a detailed work breakdown structure, a project timeline, an assessment of required resources, and a comprehensive risk analysis with corresponding mitigation strategies.

7.1. REFLECTION ON INITIAL PROTOTYPE

7.1.1. Hardware & Integration

The development of the AirFrame prototype successfully validated several core architectural concepts but also provided invaluable insights into areas requiring significant refinement for a production model. The modular three-part architecture (AirMount, AirBox, ProHalo) proved to be a robust and flexible design choice. The custom-designed PogoFusion and Power Interconnect PCBs, featuring pogo pins and contact pads, provided reliable and seamless power transfer between docking and undocking. The successful integration of the ESP32, BMI088 IMU, and FTDI interface onto the CoreFrame PCB confirmed the viability of the central electronics package.

The most critical lesson in hardware was the under-specification of the Power Fabric buck converter. While the MP2393 (3A current rating) performed flawlessly under servo-only loads, it was insufficient for the combined transient current draw of the 3x MG996R servos and the 3W speaker, preventing the audio subsystem from functioning. This highlighted the necessity for a more rigorous load analysis for all subsystems combined. A production version would necessitate a higher-rated buck converter (e.g. 5A) or a separate power rail for high-draw peripherals like audio.

Furthermore, from a mechanical perspective, the prototype's reliance on PLA and adhesives for the enclosure is not viable for a commercial product. During prototyping, it was not uncommon for parts of these 3D-printed components to simply break off. Although this could be blamed on inexperience in 3D modelling, a product where stability and durability are non-negotiable (given the problem statement), adopting an inferior material for production is simply inexcusable. Therefore, further iterations will require a redesign of the body using more durable materials like PETG or ASA if going forward with 3D printing, or industry-standard materials like aluminium or titanium.

7.1.2. SOFTWARE & CONTROL

The same woes cannot be said for the software implementation on the ESP32, which was a quite a success. The two-state Kalman filter for pitch and roll stabilisation proved highly

effective, achieving the target settling times and stability as detailed in Section 5. The implementation of a BLE GATT service provided a responsive and reliable link to the AirLink iOS app, allowing for real-time telemetry and PID tuning, which was instrumental in debugging and performance validation. The primary control challenge was managing yaw drift. As the BMI088 is a 6-axis IMU without a magnetometer, it has no absolute heading reference. This was mitigated in software with a stationary-hold algorithm that learned and corrected gyro bias when the gimbal was still for a few seconds. While effective, this approach is more of a workaround. A future hardware revision (AirFrame v2?) should investigate 9-DOF IMUs that include a magnetometer, while carefully managing potential magnetic interference (main reason why it wasn't opted for here) from the servo motors through shielding and better mechanical placement in general. Secondly, and more crucially, the PID gains were found to be highly sensitive to payload mass and distribution. The current firmware uses a single set of gains, but a production model would require either an adaptive tuning algorithm or, more pragmatically, user-selectable payload profiles in the AirLink app to ensure optimal performance across different devices.

Additionally, while the ESP32-WROOM-32E was the ideal choice for rapid prototyping, the vision for advanced features like Aerial makes it unsuitable for the production version. On-device computer vision for person tracking, natural language processing, and executing multiple AI models would quickly exhaust the ESP32's dual-core processing capabilities and limited RAM. To future-proof AirFrame, a migration to a more powerful microprocessor-based platform (ARM/RISC-V) should be undertaken. This would provide the necessary computational headroom and access to a richer software ecosystem (e.g. Linux, Python for AI/CV etc.).

7.1.3. Project Management

Another point of reflection was the phased development approach, as detailed in the Week 4 WBS, which was effective in breaking down the core subsystems for development. However, the project timeline did not adequately account for component sourcing lead times and international PCB fabrication turnarounds. In the case of AirFrame v1, components were constantly facing delays and late shipments, with the final set of components only arriving on Friday of Week 9! Thus, future project planning must incorporate more generous buffers for these logistical dependencies to ensure milestones are met without compromising testing time.

7.2. FIVE-YEAR DEVELOPMENT ROADMAP

The following roadmap details a strategic five-year plan to evolve AirFrame from a university prototype into a commercially successful product line. This plan is structured not only as a technical guide but also as a foundational document suitable for presentation to potential

seed/angel investors and accelerator programs such as Y Combinator. The roadmap is broken into a Work Breakdown Structure (WBS) and a corresponding Gantt Chart timeline.

7.2.1. Work Breakdown Structure (WBS)

The WBS expands upon the initial project plan from the Week 4 Activity Report, re-framing the tasks with a focus on commercialisation, manufacturability, and strategic growth. The project is divided into five key phases.

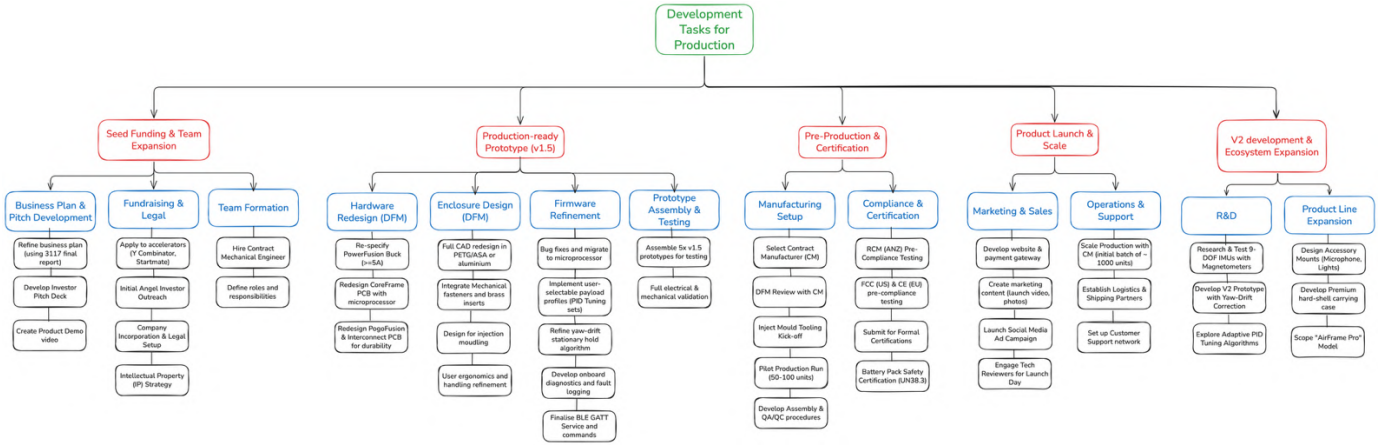
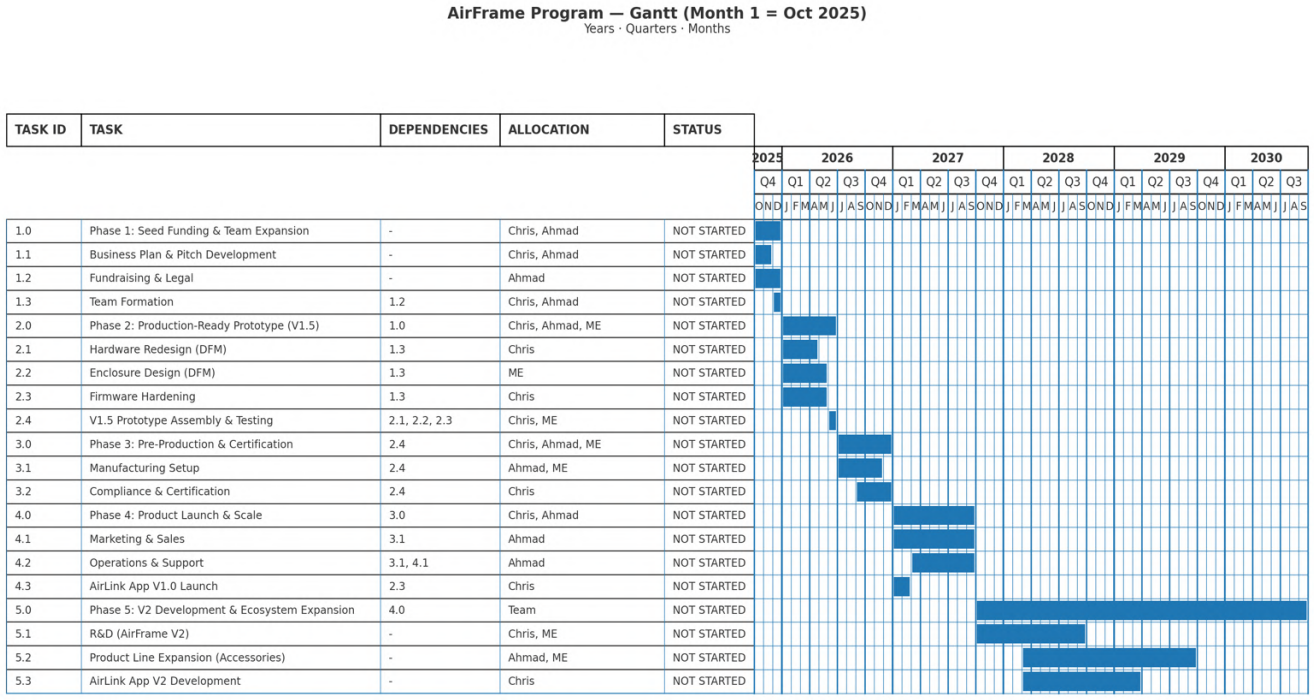


Figure 7.1. Production-Focused Work Breakdown Structure to launch AirFrame. For a clearer image, please visit this Excalidraw link: https://excalidraw.com/#json=cxNoudogJADZBPfLQ2Skh,qakeDtnPweZDunbCR_jVHA.

7.2.2. Project Timeline (Gantt Chart)



TASK ID	TASK	DEPENDENCIES	ALLOCATION	STATUS
1.0	Phase 1: Seed Funding & Team Expansion	-	Chris, Ahmad	NOT STARTED
1.1	Business Plan & Pitch Development	-	Chris, Ahmad	NOT STARTED
1.2	Fundraising & Legal	-	Ahmad	NOT STARTED
1.3	Team Formation	1.2	Chris, Ahmad	NOT STARTED
2.0	Phase 2: Production-Ready Prototype (V1.5)	1.0	Chris, Ahmad, ME	NOT STARTED
2.1	Hardware Redesign (DFM)	1.3	Chris	NOT STARTED
2.2	Enclosure Design (DFM)	1.3	ME	NOT STARTED
2.3	Firmware Hardening	1.3	Chris	NOT STARTED
2.4	V1.5 Prototype Assembly & Testing	2.1, 2.2, 2.3	Chris, ME	NOT STARTED
3.0	Phase 3: Pre-Production & Certification	2.4	Chris, Ahmad, ME	NOT STARTED
3.1	Manufacturing Setup	2.4	Ahmad, ME	NOT STARTED
3.2	Compliance & Certification	2.4	Chris	NOT STARTED
4.0	Phase 4: Product Launch & Scale	3.0	Chris, Ahmad	NOT STARTED
4.1	Marketing & Sales	3.1	Ahmad	NOT STARTED
4.2	Operations & Support	3.1, 4.1	Ahmad	NOT STARTED
4.3	AirLink App V1.0 Launch	2.3	Chris	NOT STARTED
5.0	Phase 5: V2 Development & Ecosystem Expansion	4.0	Team	NOT STARTED
5.1	R&D (AirFrame V2)	-	Chris, ME	NOT STARTED
5.2	Product Line Expansion (Accessories)	-	Ahmad, ME	NOT STARTED
5.3	AirLink App V2 Development	-	Chris	NOT STARTED

Figure 7.2. AirFrame five-year Gantt Chart detailing production timeline, compliance testing and product launch (month 1 = October 2025)

The Gantt chart in Figure 7.2 visualises the five-year development roadmap, detailing the timeline, dependencies, and resource allocation for the tasks outlined in the WBS. The timeline is scheduled to commence in October 2025, around a month after term 2 ends. The first 24 months are detailed extensively to illustrate the critical path-to-market, while years 3-5 are summarised at a higher level to show long-term strategic vision.

Resource allocation is shown for the founding team (Chris & Ahmad) and the planned hires, where **ME** denotes the contract **Mechanical Engineer**. Key milestones, such as the completion of the v1.5 prototype at the end of Q2 2026 and the start of the Product Launch phase in Q1 2027, are implicitly defined by the completion of their preceding phases.

7.3. ADDITIONAL RESOURCES REQUIRED

To successfully execute the development roadmap and bridge the gap between the current functional prototype and a reliable, manufacturable, and commercially successful product, several additional resources will be essential. These are categorised into hardware facilities, software and services, and specialised personnel.

7.3.1. Hardware and Facilities

Professional Electronics Lab: While basic prototyping was possible with university resources, a dedicated lab space equipped with the following is critical for rapid v1.5

development and validation: 4-Channel Oscilloscope for debugging high-frequency phenomena, bench power supply to provide stable, current-limited power for testing individual PCBs without relying on battery, soldering and hot-air rework station for efficient assembly and modification of custom PCBs.

Advanced Prototyping Equipment: To allow for rapid iteration of the mechanical enclosure and gimbal components using PETG and ASA (as discussed previously), an in-house FDM 3D printer would be required. Although one could use an external service for 3D printing, it would be far more economical and time efficient in the long run to have an in-house 3D printer. Also, for creating high-fidelity aluminium or titanium enclosure for the final pre-production prototypes, access to CNC machining might also be required.

Testing & Validation Facilities: To formally verify that AirFrame meets its specified operating temperature range of -20°C to +75°C, an Environmental Test Chamber is a critical requirement for a product intended for outdoor use. Also, a vibration table and drop-test rig may also be purchased to systematically test the durability of the enclosure and the stability of the gimbal mechanism, ensuring the product can withstand the rigours of real-world use.

7.3.2. Software & Services

While three out of the four custom PCBs were designed in KiCad, a commercial license for an industry-standard tool like Altium is necessary for designing 4 (or more) layer PCBs with controlled impedance for the microprocessor's data lines and managing a complex Bill of Materials (BOM) integrated with a supply chain, like CoreFrame (see Appendix B for more details on PCB design). For this project, an Altium Student License was used to design CoreFrame, which is not ideal in a commercial setting.

Similarly, for Autodesk Fusion, a full commercial license is required to move beyond educational use and leverage advanced features for designing for injection moulding (DFM) and performing finite element analysis (FEA) to optimise the enclosure's strength-to-weight ratio.

Additionally, an Apple Developer Program Membership is essential for distributing the AirLink iOS app via TestFlight for beta testing and for publishing to the App Store.

Finally, for development and deployment of Aerial, subscriptions to LLM providers (OpenAI, Anthropic, Google etc.) and potentially cloud services (e.g. AWS) for future features like over-the-air (OTA) firmware updates for AirOS would be necessary for the best possible user experience.

7.3.3. Personnel

The founding team obviously possesses a strong foundation in electrical engineering and software. However, to accelerate development and ensure a professional-quality product, the team may be expanded to include additional specialists, e.g., mechanical engineer, iOS developer, operations/supply chain manager etc.

7.4. RISK ASSESSMENT & MITIGATION

This risk register identifies potential risks across four key domains: **Technical, Business, Project Management,** and **Safety**. Each risk is evaluated for its Likelihood of occurring and its Impact on the project if it does. This determines its overall severity. For each identified risk, a proactive Mitigation Strategy is proposed to prevent or reduce its impact, and a reactive Contingency Action is defined as a "Plan B" in case the risk materializes.

7.4.1. Technical Risks

Table 6: Technical Risks and Mitigation

Risk	Likelihood	Impact of Risk	Severity	Mitigation Strategy	Contingency Action
Ineffective Stabilization Algorithm	Medium	High	High	Develop the sensor fusion (Kalman/Complementary) and PID control algorithms as a priority. Use software simulation to test the logic before deploying to hardware. Allocate specific time for iterative tuning.	Revert to a known-good, library-based PID implementation to ensure a functional baseline for demonstration. Detail the custom algorithm's challenges and future tuning steps in the final report.
Power System Failure Under Load	Medium	High	High	Select a low-dropout (LDO) or buck-boost converter instead of a standard one. Simulate the full power circuit in LTspice. Prototype and test the power system under full servo load across the battery's entire voltage range (8.4V to 6.0V).	The system will have a secondary USB-C input for direct power. If the battery/regulator circuit fails, the gimbal can be powered directly via this port for the final demonstration, ensuring core functionality is shown.
PCB Design/Fabrication Error	Medium	High	High	Implement strict Design Rule Checks (DRC) in EDA software. Conduct a mandatory peer review of	Utilize the breadboard prototype for parallel software development while waiting for new boards. Order

				schematics and layouts before ordering. Use a reputable fabrication house (e.g., JLCPCB, PCBWay) with automated checks.	replacement PCBs with expedited shipping. Order a minimum of 3 boards initially to have spares.
EMI from Motors Affecting IMU	Medium	Medium	Medium	In the PCB layout, ensure maximum physical distance between servo power lines and the IMU. Use a solid ground plane to shield the IMU. Twist the I2C communication wires in the prototype to reduce noise susceptibility.	If noise persists, add a physical mu-metal or steel shield between the IMU and the servos. Implement a moving average filter in software to smooth the IMU data as a final corrective measure.

7.4.2. Business Risks

Table 7: Business Risks and Mitigation

Risk	Likelihood	Impact of Risk	Severity	Mitigation Strategy	Contingency Action
Project Cost Overrun	Medium	Medium	Medium	Maintain a detailed Bill of Materials (BOM) and track all expenses. Include a 10% contingency fund in the initial budget. Prioritize sourcing from affordable, local suppliers where possible.	De-scope non-essential features (e.g., use a simpler clamp mount instead of MagSafe charging) to reduce material costs. Use university makerspace resources for 3D printing to eliminate fabrication costs.
Product Non-Competitive	Low	High	High	Conduct thorough secondary market research (Week 2) to identify unique selling points (USPs) like modularity and payload charging. Focus development on	Pivot marketing to highlight the product's open-source, customizable nature as a key advantage for makers, students, and researchers, rather than competing directly with

				features that differentiate the product from existing commercial gimbals.	consumer giants like DJI.

7.4.3. Project Management Risks

Table 8: Project Management Risks and Mitigation

Risk	Likelihood	Impact of Risk	Severity	Mitigation Strategy	Contingency Action
Component Sourcing Delays	High	High	High	Finalize the BOM by Week 3 and place all orders from Australian suppliers by Week 5. Identify pin-compatible alternative components for critical parts (IMU, MCU) during the design phase.	If a part is delayed, pivot development to a non-dependent task (e.g., software simulation, enclosure design). If a part is unavailable, procure the pre-identified alternative component.
Subsystem Integration Failure	High	Medium	High	Use a modular, iterative integration approach. Integrate one subsystem at a time. Use version control (Git) for all software to track changes and easily revert if an integration fails. Maintain clear interface documentation.	Disconnect the failing subsystem and present it as a separate, functional unit during the demonstration. Clearly explain the integration issue and the proposed solution in the final report.
Team Member Unavailability	Low	Medium	Medium	Ensure both team members have a comprehensive understanding of all subsystems. Maintain a shared repository (e.g., GitHub, Google Drive) with all code, design files, and documentation for seamless handover.	The available team member will prioritize completing the most critical remaining task required for a minimum viable product. The project schedule includes a small buffer for such events.

7.4.4. Safety Risks

Table 9: Safety Risks and Mitigation

Risk	Likelihood	Impact of Risk	Severity	Mitigation Strategy	Contingency Action
Li-Ion Battery Thermal Runaway	Low	High	High	Use a dedicated Li-ion charger IC (TP5100) with built-in over-charge, over-discharge, and thermal protection. Source high-quality, protected 18650 cells from reputable vendors. Never leave the device charging unattended.	Immediately disconnect all power sources. Use a Class D (dry powder) fire extinguisher. Move the device to a fire-safe location (e.g., concrete floor away from flammable materials).

8. Conclusion

We started this journey with a simple, yet profound, question: what if we could create a tool that gave creators the stability of a professional gimbal, but with the simplicity and robustness the real-world demands? We believe AirFrame delivered on that vision, creating a functional, modular, three-axis stabilisation system that proves our core architectural and software concepts are not just viable, but powerful.

The deep integration of hardware and software is at the heart of this success. We engineered four custom PCBs to manage everything from high-power USB-C delivery to the precise logic of our CoreFrame. This hardware provides the foundation for AirOS, our intelligent control system. Its sophisticated Kalman filters and finely tuned PID controllers work in concert to create a seamless experience, transforming shaky, real-world motion into smooth, cinematic footage. AirFrame was designed to fill a critical gap in a market owned not just by professionals, but also by regular people.

The path to innovation is one of constant learning. We see the challenges we encountered not as failures, but as invaluable insights. The under-specification of our buck converter highlighted the immense transient current demands of high-torque servos, and the inherent drift of a 6-axis IMU pushed us to develop a sophisticated stationary-hold algorithm. These learnings have been instrumental, providing a crystal-clear direction for the next generation of AirFrame.

Looking forward, we have three key recommendations for the continuation of this project:

1. A more powerful CoreFrame, by migrating from the ESP32 to a more powerful microprocessor. This will provide the computational headroom necessary for on-device machine learning and computer vision.
2. Enhanced spatial awareness by moving to a 9-DOF IMU that includes a magnetometer. This is the definitive hardware solution to eliminate yaw-drift, providing an absolute heading reference that will elevate system precision.
3. The next phase must focus on design-for-manufacturing (DFM). This involves moving from 3D-printed enclosures to premium materials like machined aluminium or high-strength composites.

The take-home message is this: AirFrame is more than a successful university project. It is a proof-of-concept for a new category of creative tool that combines intelligence, robustness, and accessibility. It demonstrates that with a relentless focus on the user experience and the seamless integration of hardware and software, it is possible to empower the next generation of storytellers. And we are just getting started.

9. References

1. Espressif Systems (Shanghai) Co., Ltd 2025, *ESP32-WROOM-32E / ESP32-WROOM-32UE datasheet*, Espressif, viewed August 2025, https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf.
2. Bosch Sensortec GmbH 2024, *BMI088 inertial measurement unit (datasheet), rev. 1.9 (January 2024)*, Bosch Sensortec, viewed August 2025, https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmi088-ds001.pdf.
3. Monolithic Power Systems 2018, *MP2393 — 3A, 28V, 470kHz synchronous step-down converter (datasheet), rev. 1.0 (29 Aug 2018)*, Monolithic Power Systems, viewed August 2025, https://www.monolithicpower.com/en/documentview/productdocument/index/version/2/document_type/Datasheet/lang/en/sku/MP2393GTL-Z/.
4. Nanjing Extension of Microelectronics Co., Ltd n.d., *TP5100 datasheet (English translation)*, viewed August 2025, https://cdn.hackaday.io/files/1948598336847456/TP5100-ENG.pdf.
5. Tempero Systems n.d., *TP5100 2-cells single lithium-ion battery charger module 2A 18650 charging PCB (5–18V DC)*, product page, viewed August 2025, https://temperosystems.com.au/products/tp5100-2-cells-single-lithium-lion-battery-charger-module-2a-18650-charging-pcb-5-18v-dc/.
6. Samsung SDI 2013, *Introduction of INR18650-25R* (datasheet), Samsung SDI, viewed August 2025, https://www.powerstream.com/p/INR18650-25R-datasheet.pdf.
7. How To Mechatronics n.d., *DIY Arduino Gimbal | Self-Stabilizing Platform*, How To Mechatronics, viewed August 2025, https://howtomechatronics.com/projects/diy-arduino-gimbal-self-stabilizing-platform/.
8. JLCPCB n.d., *Impedance calculator / controlled-impedance page*, JLCPCB, viewed August 2025, https://jlcpcb.com/impedance.
9. Feranec, R n.d., *How to Make Custom ESP32 Board in Altium Designer | Full Tutorial* (video), YouTube, viewed August 2025, https://www.youtube.com/watch?v=KWIzhbQaZZk.
10. Espressif Systems (Shanghai) Co., Ltd n.d., *ESP32-DevKitC — User Guide*, Espressif documentation, viewed August 2025, https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/user_guide.html.
11. Espressif Systems (Shanghai) Co., Ltd n.d., *ESP32-DevKitC V4 schematic* (PDF), Espressif, viewed August 2025, https://dl.espressif.com/dl/schematics/esp32_devkitc_v4_sch.pdf.
12. Espressif Systems (Shanghai) Co., Ltd n.d., *ESP32-DevKitC V4 PCB layout* (PDF), Espressif, viewed August 2025, https://dl.espressif.com/dl/schematics/esp32_devkitc_v4_pcb_layout.pdf.

13. Espressif Systems (Shanghai) Co., Ltd n.d., *ESP32-DevKitC V4 dimensions* (PDF), Espressif, viewed August 2025, https://dl.espressif.com/dl/schematics/esp32_devkitc_v4_dimensions.pdf.
14. Multicomp Pro 2024, *Spring loaded connector* (datasheet, V1.0, 18 Dec 2024), Premier Farnell, viewed August 2025, http://www.farnell.com/datasheets/4459683.pdf.
15. Yageo 2022, *Positive temperature coefficient AC/DC power supply — SMD2920 series* (datasheet, V3, 28 Mar 2022), Yageo, viewed August 2025, http://www.farnell.com/datasheets/3781953.pdf.
16. Espressif Systems (Shanghai) Co., Ltd 2025, *Arduino-ESP32 documentation (v3.3.0)*, Espressif documentation, viewed August 2025, https://docs.espressif.com/projects/arduino-esp32/en/latest/.
17. Bolder Flight Systems n.d., *bmi088-arduino* (GitHub repository), viewed August 2025, https://github.com/bolderflight/bmi088-arduino.
18. Adafruit Industries 2025, *Adafruit MAX98357 I2S Class-D Mono Amp* (learn guide PDF, last updated 3 Jul 2025), Adafruit, viewed August 2025, https://cdn-learn.adafruit.com/downloads/pdf/adafruit-max98357-i2s-class-d-mono-amp.pdf.
19. Maxim Integrated 2016, *MAX98357A/MAX98357B: PCM input Class-D audio power amplifiers* (datasheet, rev. 7, Feb 2016), Maxim Integrated, viewed August 2025, https://cdn-shop.adafruit.com/product-files/3006/MAX98357A-MAX98357B.pdf.
20. Joy-IT (SIMAC Electronics GmbH) 2025, *USB-PD Trigger Module — COM-ZY12PDN, COM-ZY12PDN-ST & COM-ZY12PDN-USB: Manual* (PDF, 18 Feb 2025), Joy-IT, viewed August 2025, https://joy-it.net/files/files/Produkte/COM-ZY12PDN/COM-ZY12PDN_Manual_2025-02-18.pdf.
21. Joy-IT (SIMAC Electronics GmbH) 2021, *USB-PD Trigger Module — COM-ZY12PDN: Datasheet* (PDF, published 05 May 2021), Joy-IT, viewed August 2025, https://joy-it.net/files/files/Produkte/COM-ZY12PDN/COM-ZY12PDN_Datasheet_2021-05-14.pdf.
22. *ELEC3117 Week 1–4 Activity Reports - M12A Group F,* Chris Pagolu & Ahmad Rana, last updated July 2025, viewed August 2025.
23. Precedence Research 2025, *Camera Accessories Market Size to Surpass USD 17.21 Bn by 2034*, Precedence Research, last updated 19 May 2025, viewed June 2025, https://www.precedenceresearch.com/camera-accessories-market.
24. Chemtob, D 2024, 'Forbes Daily: The $250 Billion Influencer Economy Is Booming', *Forbes*, 28 October, viewed June 2025, https://www.forbes.com/sites/daniellechemtob/2024/10/28/forbes-daily-the-250-billion-influencer-economy-is-booming/.

Appendix

A: CIRCUIT SCHEMATICS

A.1. CoreFrame

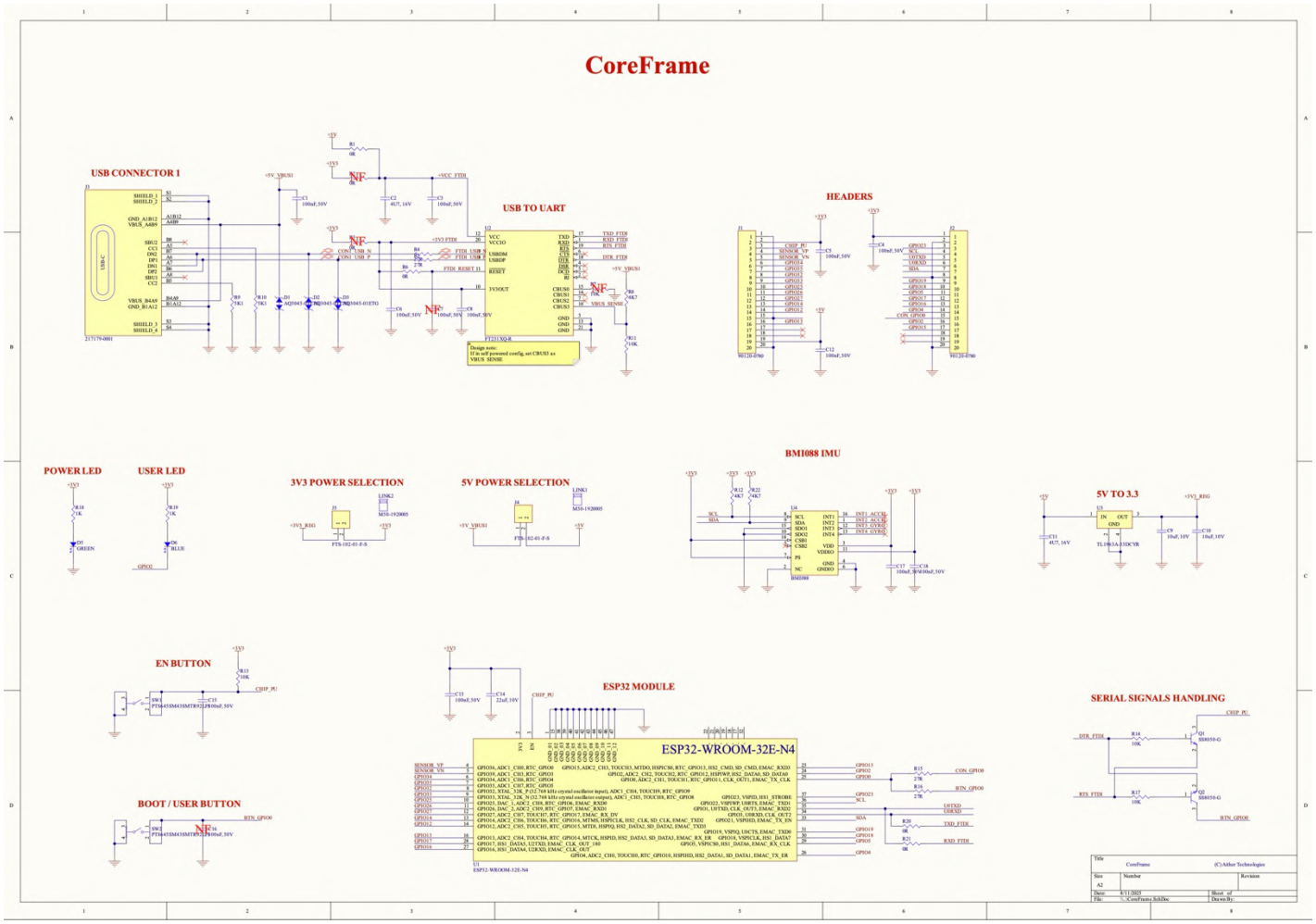


Figure A1. Overall schematic of the CoreFrame board.

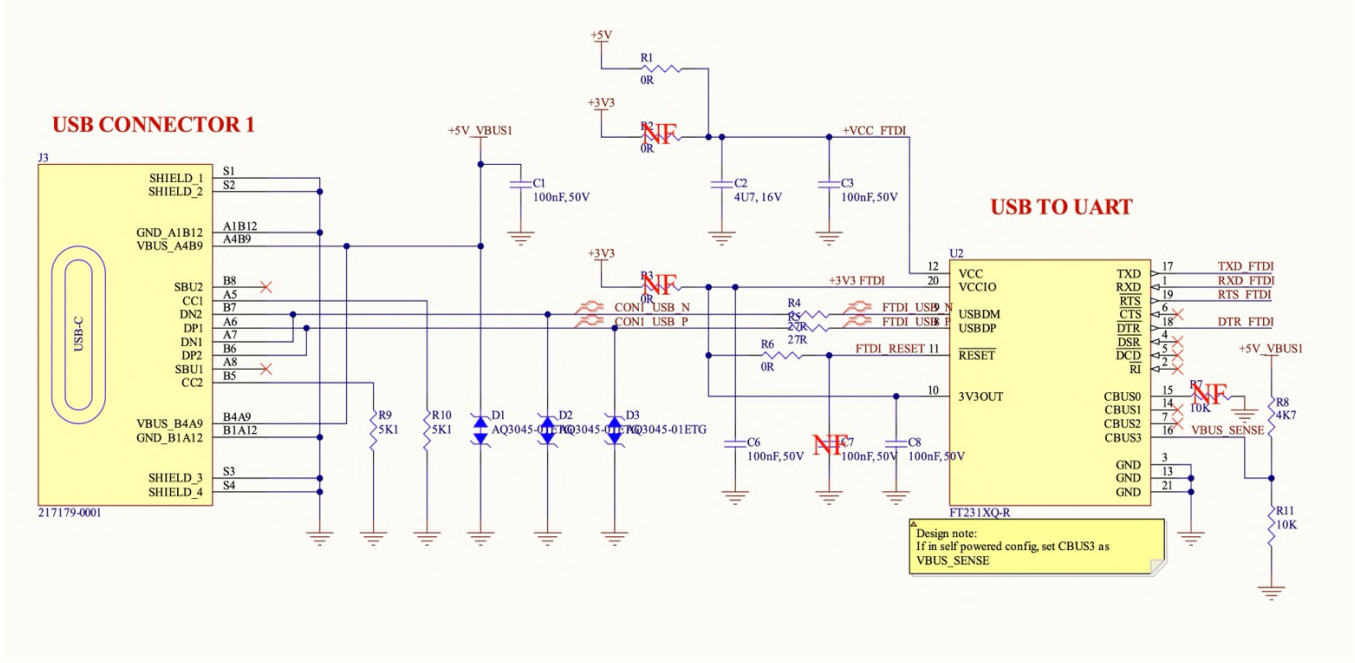


Figure A2. USB-C to UART connections.

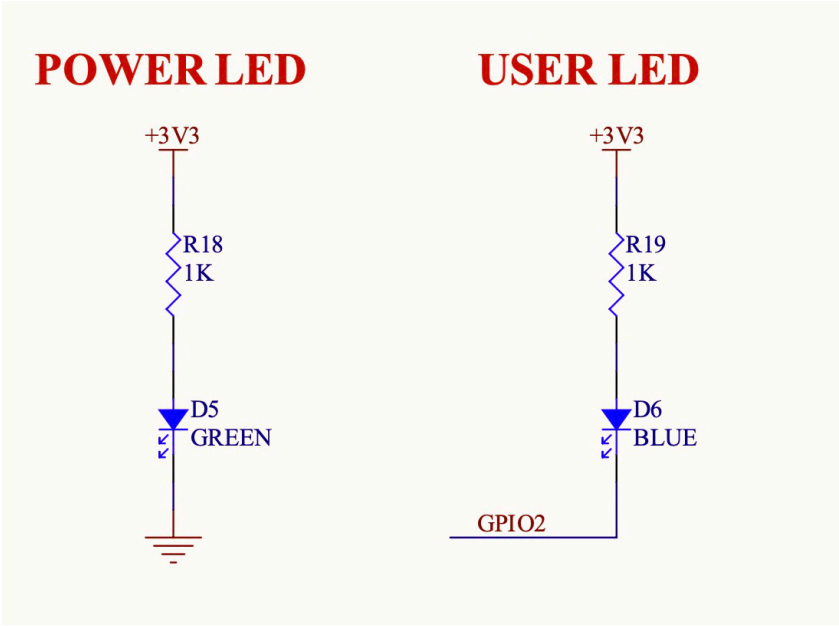


Figure A3. Power and USER LED on CoreFrame to indicate power and connections.

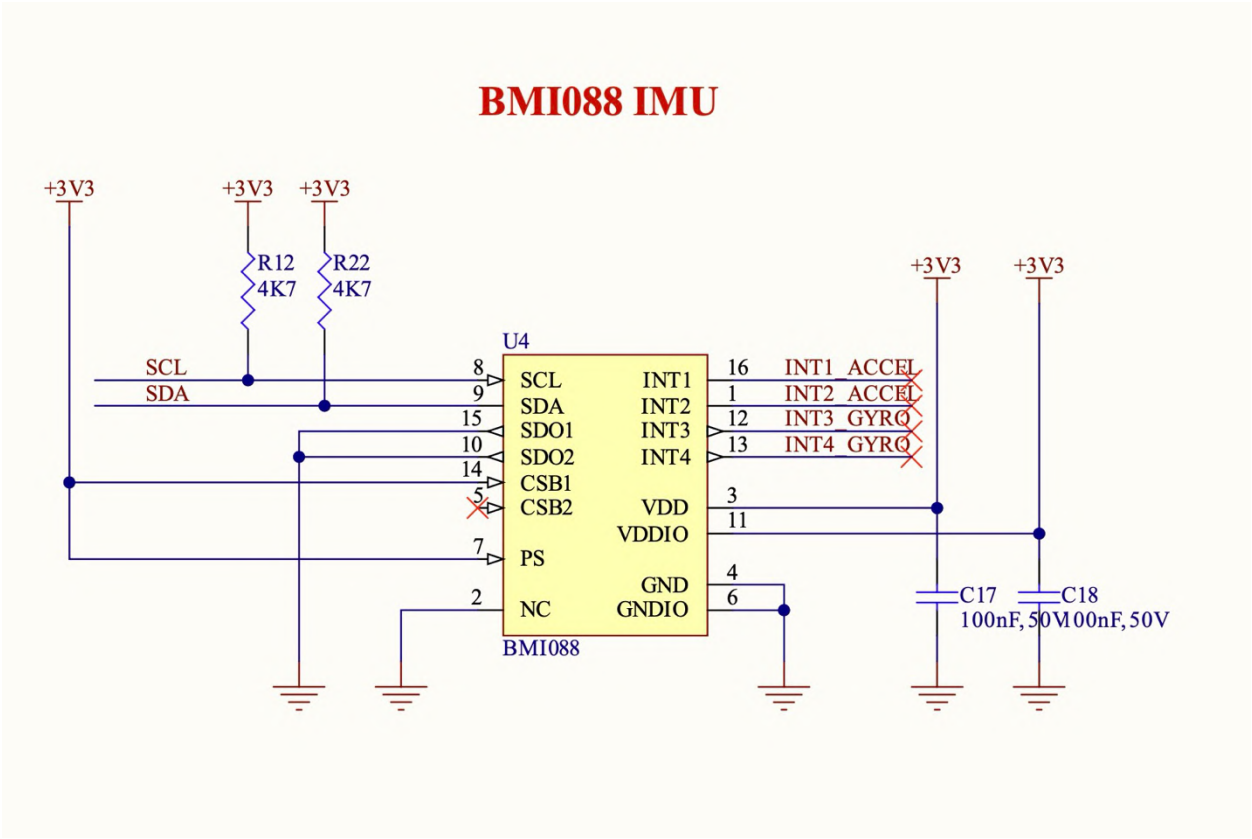


Figure A4. Schematic of the BMI088 IMU and its respective connections.

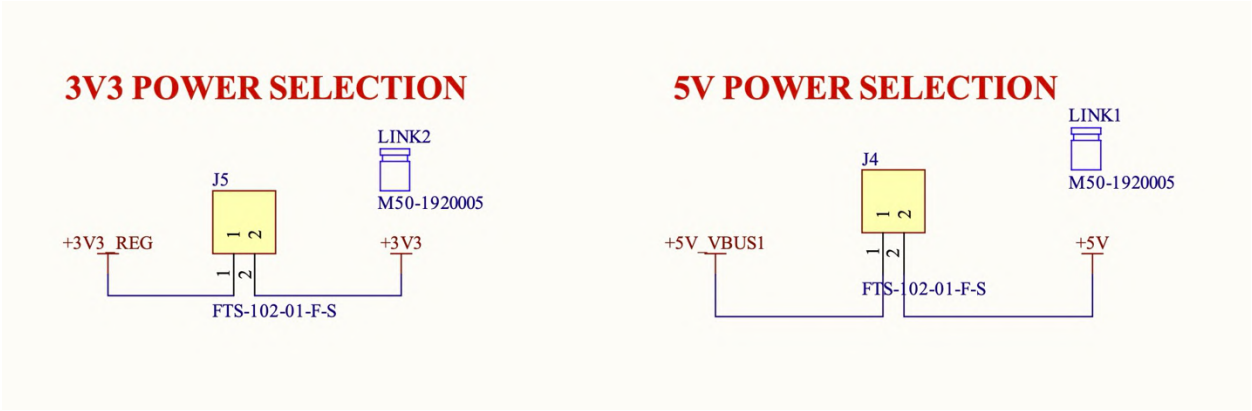


Figure A5. Jumpers J5 and J4 with removable links/caps for debugging.

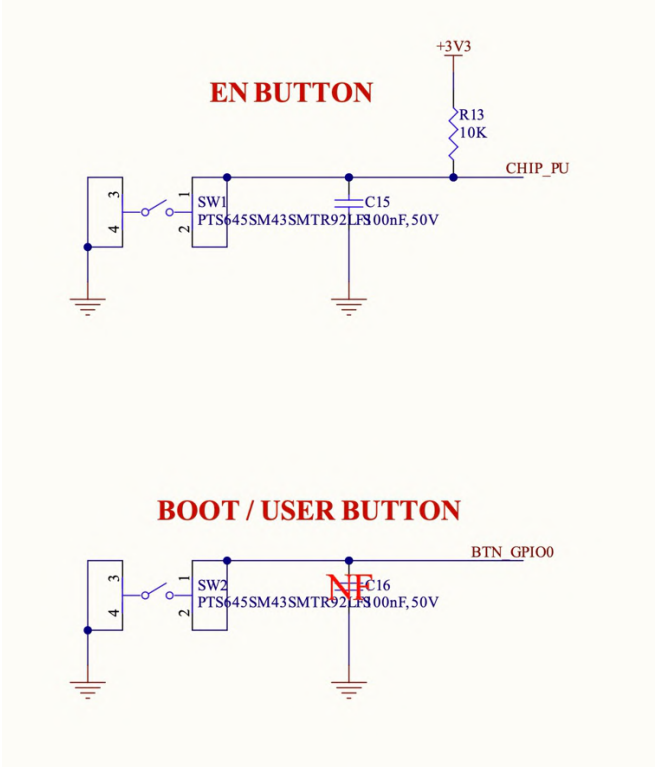


Figure A6. EN and BOOT/USR buttons for manually triggering RST and/or bootloader options.

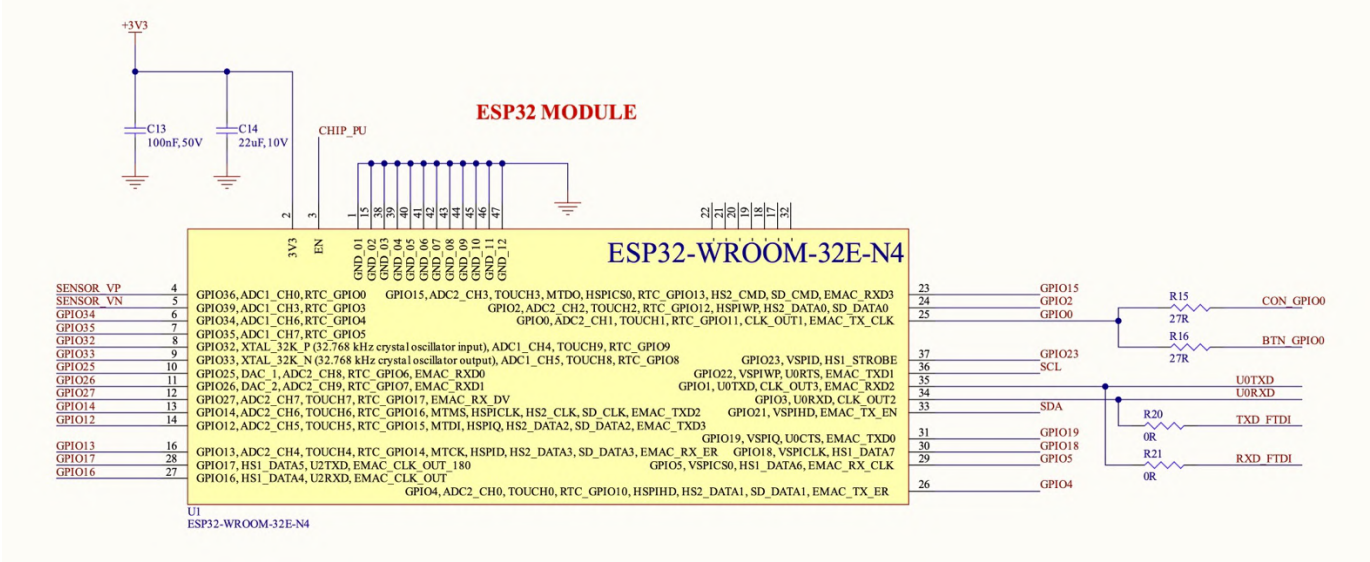


Figure A7. ESP32-WROOM-32E module with relevant connections and decoupling capacitors near power pins.

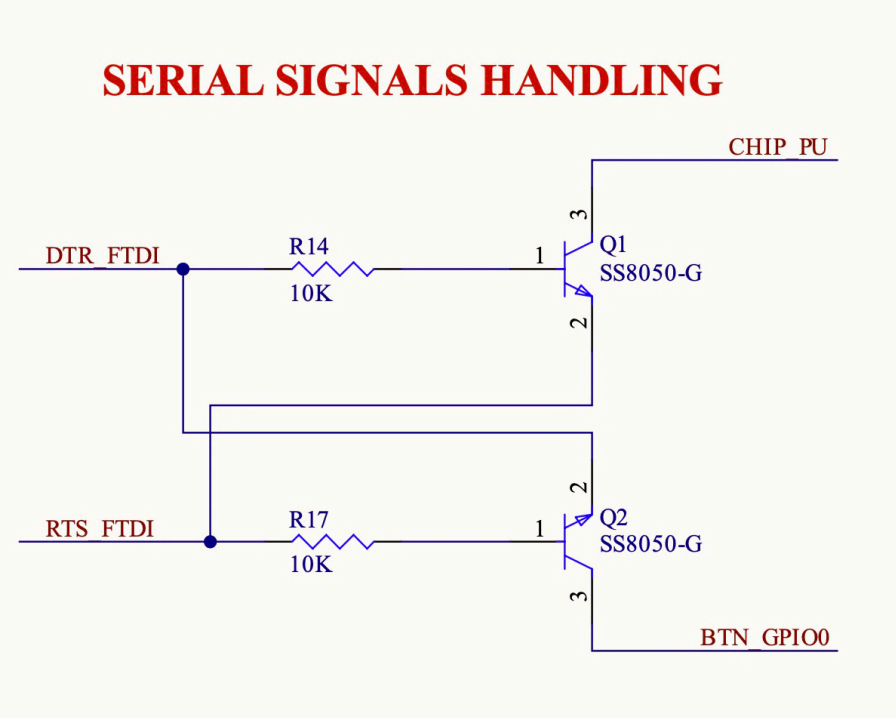


Figure A8. Auto-programming circuit for flashing firmware without manually pressing EN/BOOT every time.

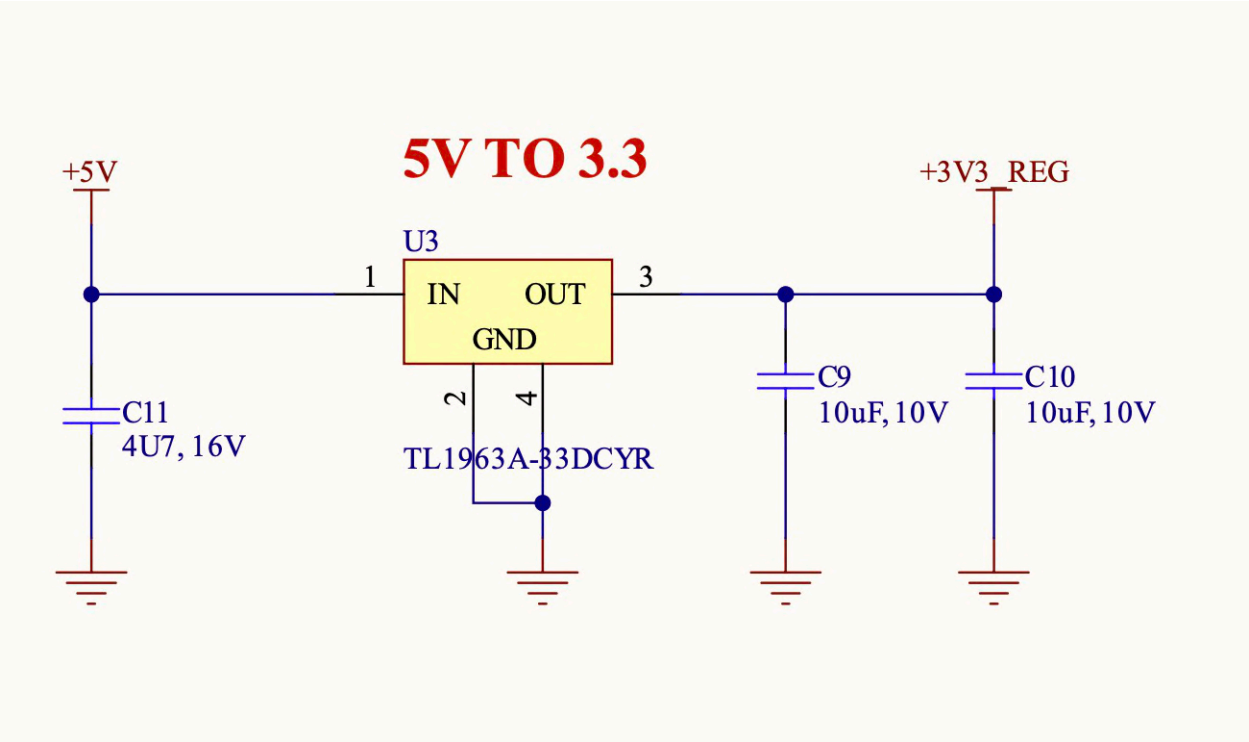


Figure A9. Built-in 5V to 3.3V LDO for powering the ESP32.

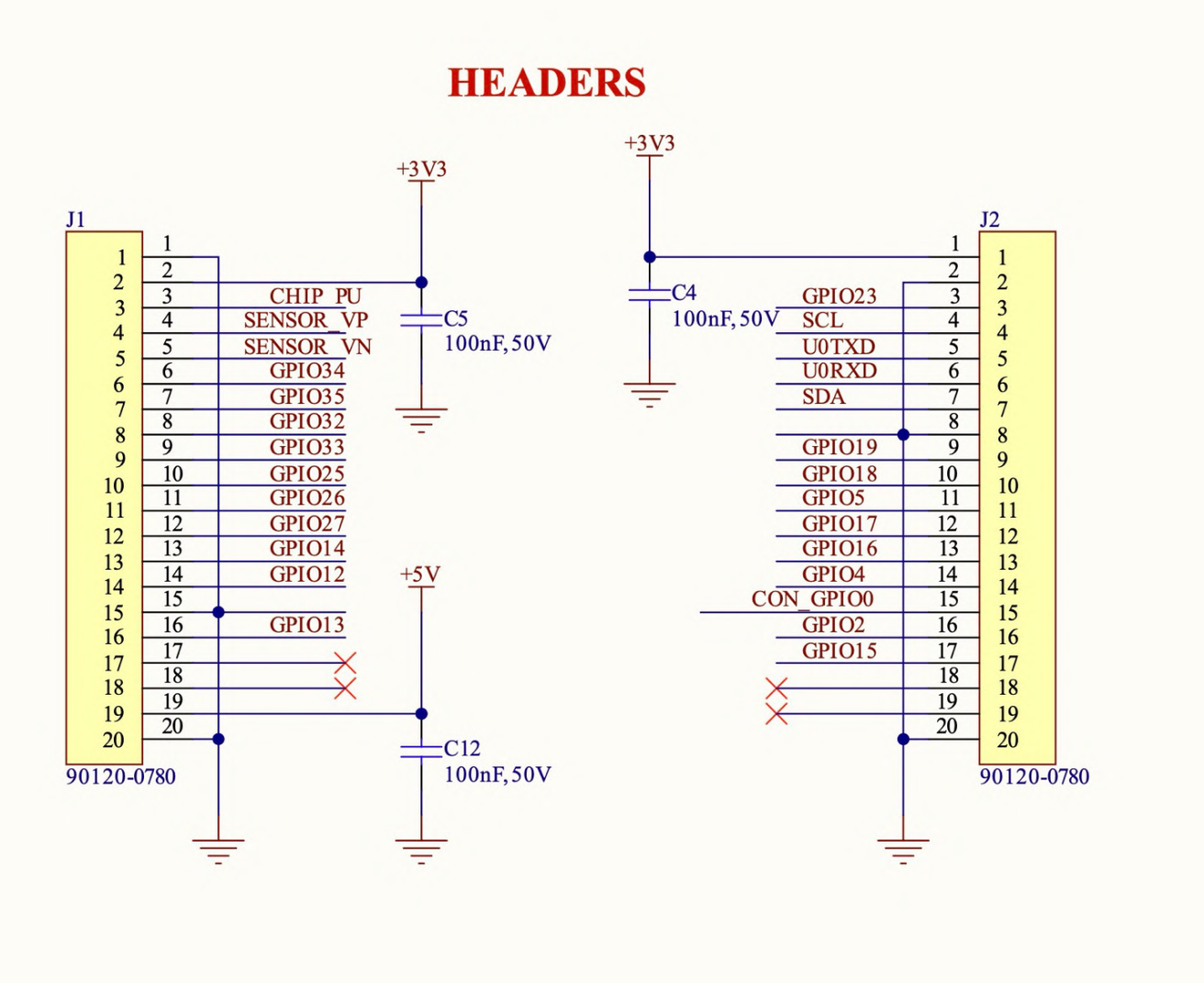


Figure A10. 2.54mm 20-pin headers.

A.2. Power Fabric

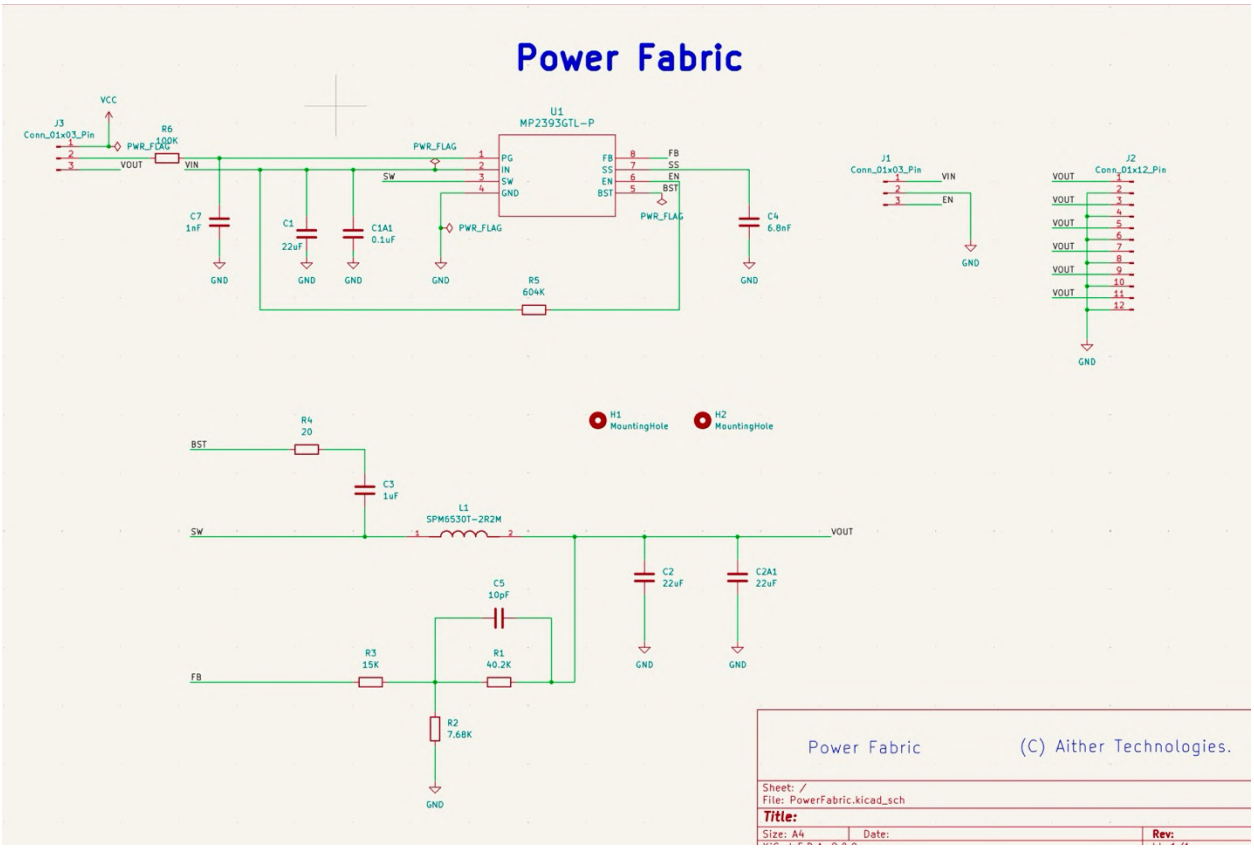


Figure A11. Overall schematic of the Power Fabric board.

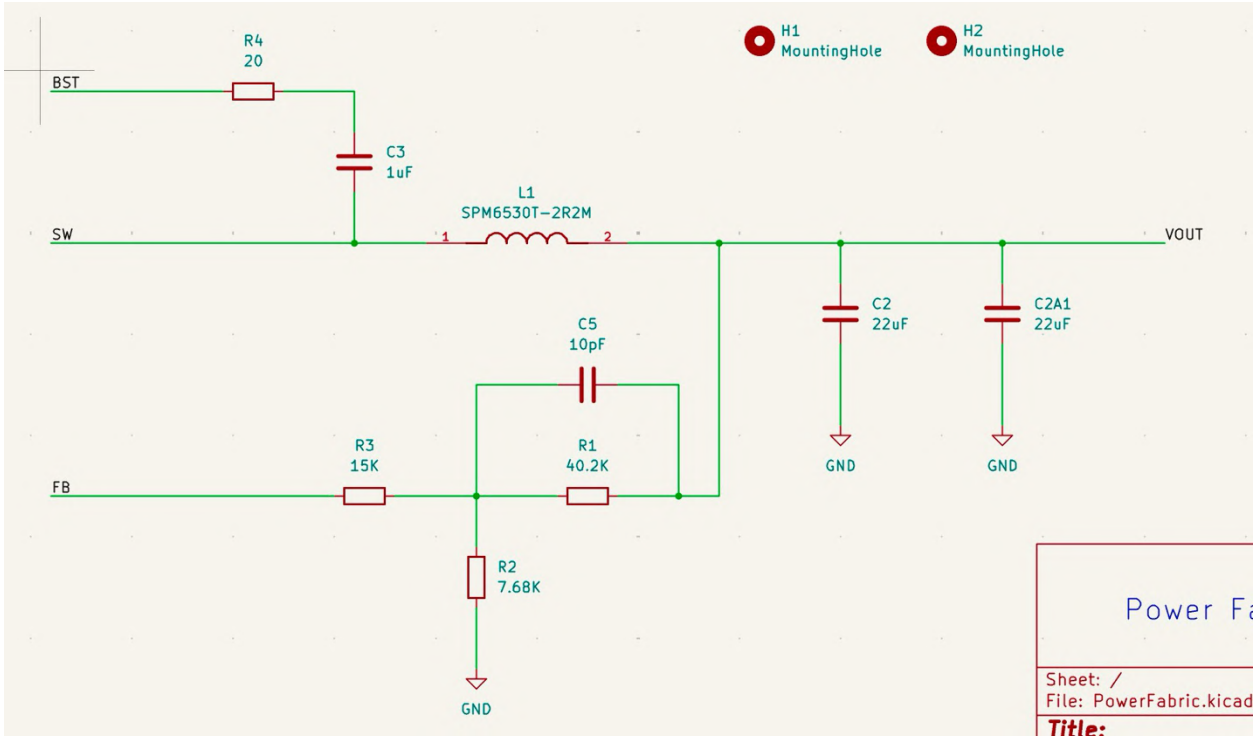


Figure A12. 2.2uH inductor connected to the buck converter circuit. H1 and H2 are M3 mounting holes. C2 and C2A are bulk capacitors to smooth out output ripple. R4 and C3 form the bootstrap network and R1, R2, R3 and C5 form the feedback network.

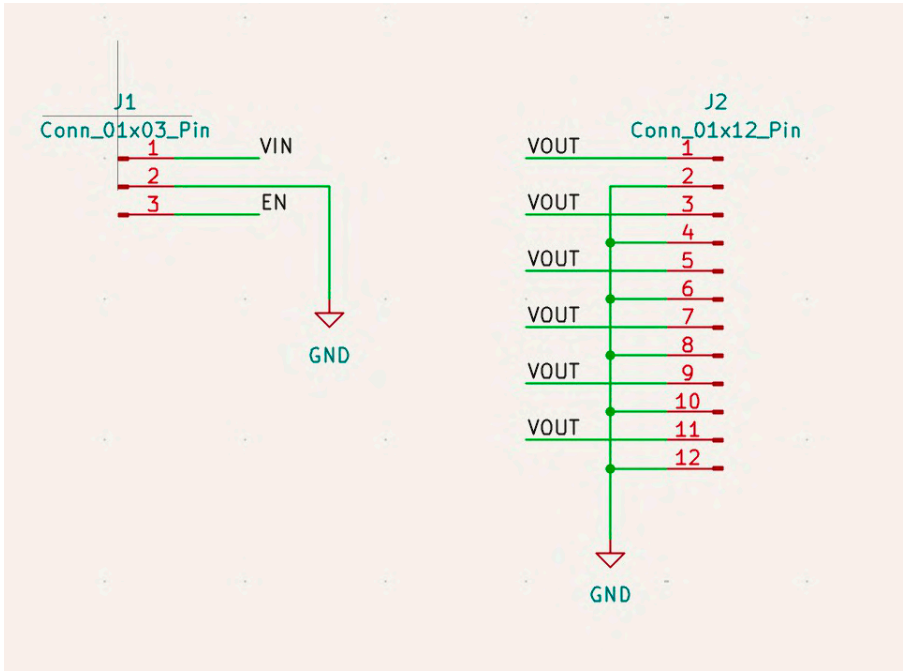


Figure A13. Single 3-pin and single 12-pin 2.54mm headers.

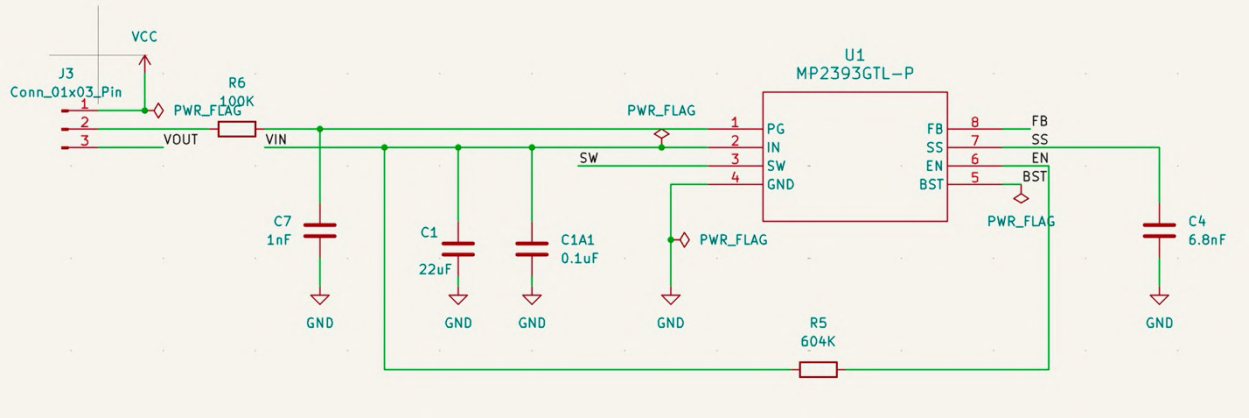


Figure A14. M2393 connected to its respective components.

A.3. Power Interconnect

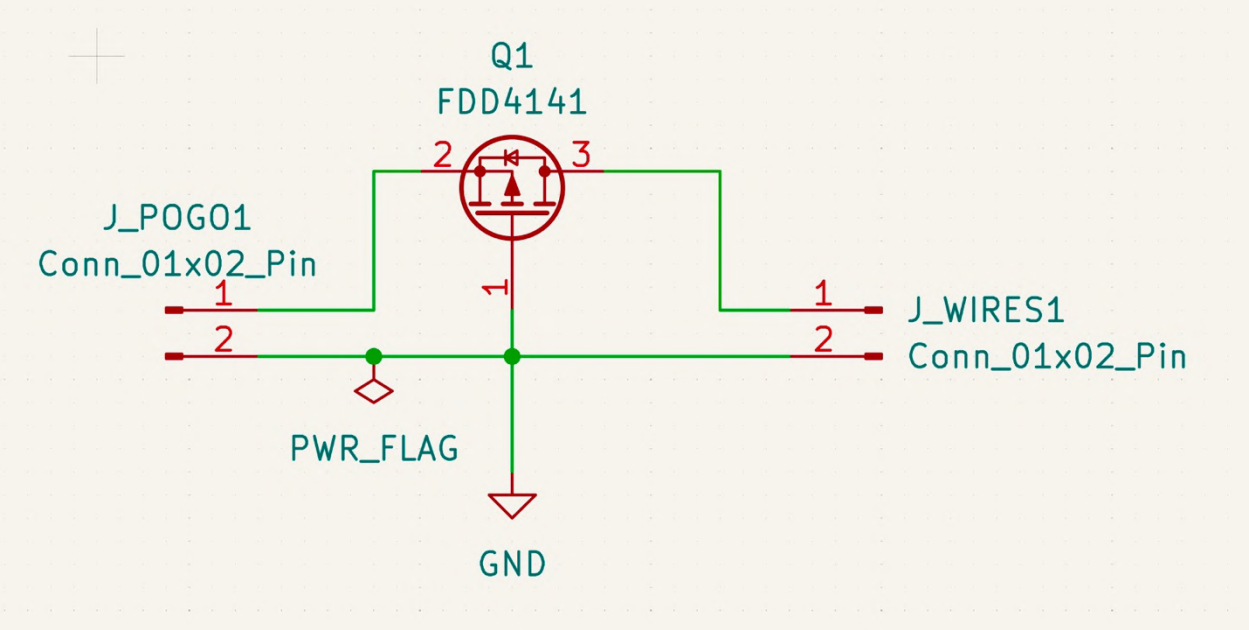


Figure A15. Power Interconnect PCB. J_POGO1 is where the contact pads touch the pogo pins. J_WIRES1 is output.

A.4. PogoFusion

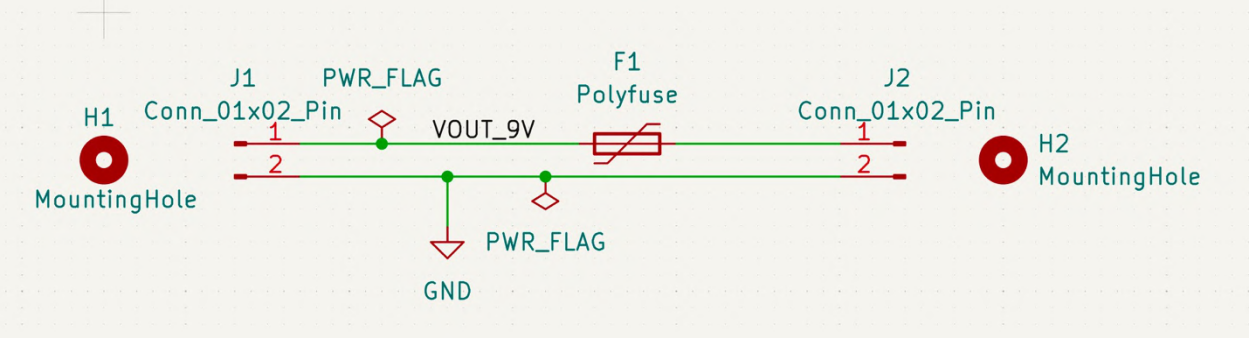


Figure A16. J1 is input from the ZY12PDN USB-C PD. J2 is where the 2.54mm 1x2 Pogo pins header goes. H1 and H2 are M3 mounting holes.

B: PCB LAYOUTS

The development of the AirFrame prototype involved the design and fabrication of four custom-made PCBs. Each PCB was designed with specific intent, performance, electrical and mechanical requirements in mind, following industry best practices for power integrity, signal integrity, and thermal management. The designs were created using Altium Designer (CoreFrame) and KiCad (Power Fabric, Power Interconnect, PogoFusion).

B.1. CoreFrame

- **Placement Strategy:** The BMI088 IMU was placed in a mechanically quiet area of the board, near the centre for best calibration results and away from connectors and holes, to minimise mechanical stress and noise. Decoupling capacitors for all ICs were placed as close as possible to provide a low-impedance path for high-frequency currents. TL1963A 3.3V LDO and its associated capacitors were grouped together to ensure stable, low-noise power for sensitive digital components. Also, the USB-C connector, ESD protection diodes and FTDI chip were located together to keep the high-speed USB data lines short and direct.
- **Routing & Grounding:** A 4-layer stack-up was used, specifically the JLC04161H-3313 stack-up from JLCPCB. This provides a defined dielectric thickness and copper weight, essential for calculating controlled impedance traces. The final stack-up was: Signal (Top), Power/Copper Plane (L2), Ground Plane (L3), Signal (Bottom).
- **USB Differential Pair:** The USB D+ and D- lines were routed as a 90-ohm controlled-impedance differential pair (hence why a specific stack-up was chosen). As seen in Figure A17, the traces were kept tightly coupled and length-matched to prevent skew. This was critical for maintaining signal integrity and providing common-mode noise rejection, ensuring reliable high-speed data communication with the FTDI chip. **Layer 3 (Figure A18)** is a dedicated, solid ground plane and provides a continuous, low impedance return path for all signals, which is critical for signal integrity and reducing EMI.

Layer 2 (Figure A19) is primarily a +3.3V power plane, ensuring a stable supply across the board.

High-current traces, such as the +5V input, were made wider (60mil) to minimise voltage drop. Stitching vias were also placed liberally around the perimeter of the board and between ground pours on the top and bottom layers to tie the ground planes together, further reducing impedance.

- The CoreFrame PCB was the primary board for firmware development and testing. As shown in Figure A41, the assembled board successfully powered on, and firmware was flashed via the integrated USB-C port and FTDI chip. The onboard LEDs for power and user functions were fully operational. The successful acquisition of data from the IMU and control of the servos confirmed that all critical components and connections were functioning as designed.

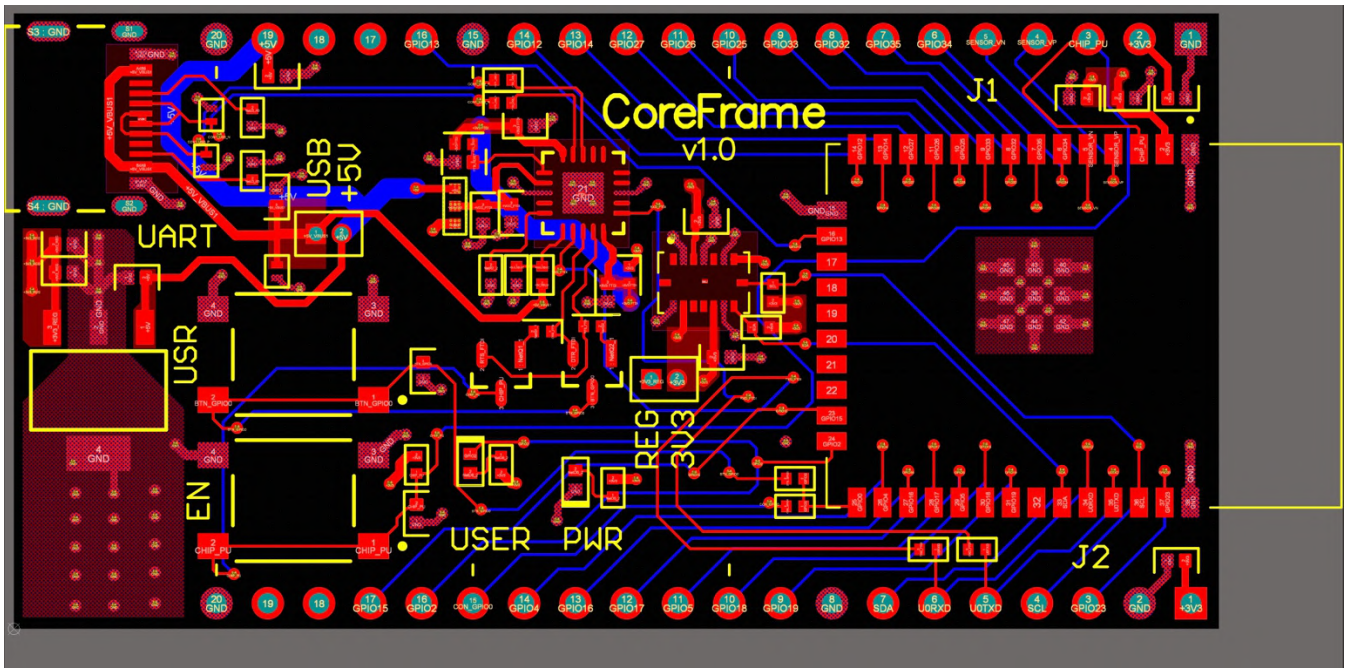


Figure A17. CoreFrame PCB Layout: Top and Bottom layers (1 and 4).

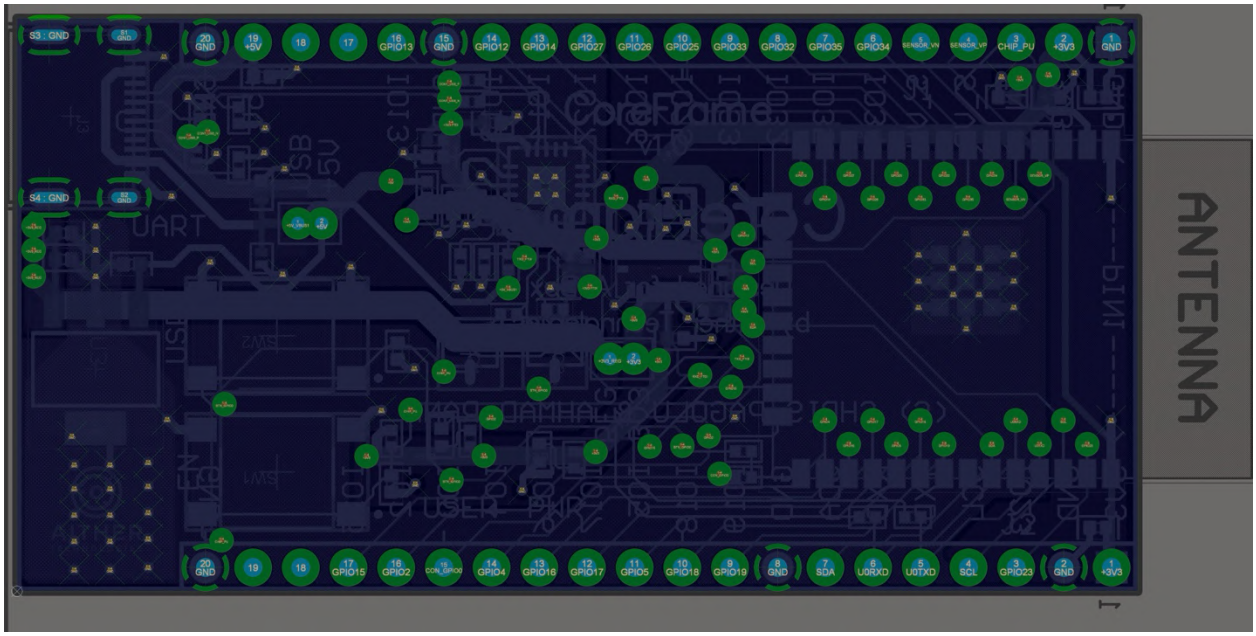


Figure A18. CoreFrame Layer 3 (Ground Plane)

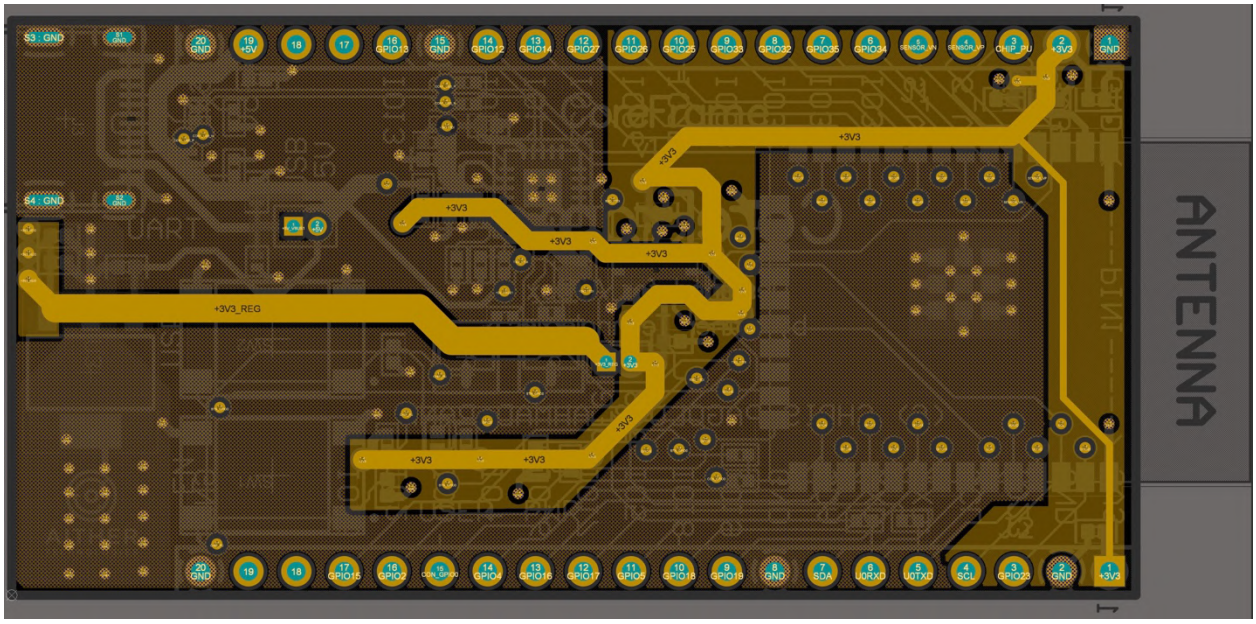


Figure A19. CoreFrame Layer 2 (Copper Plane)

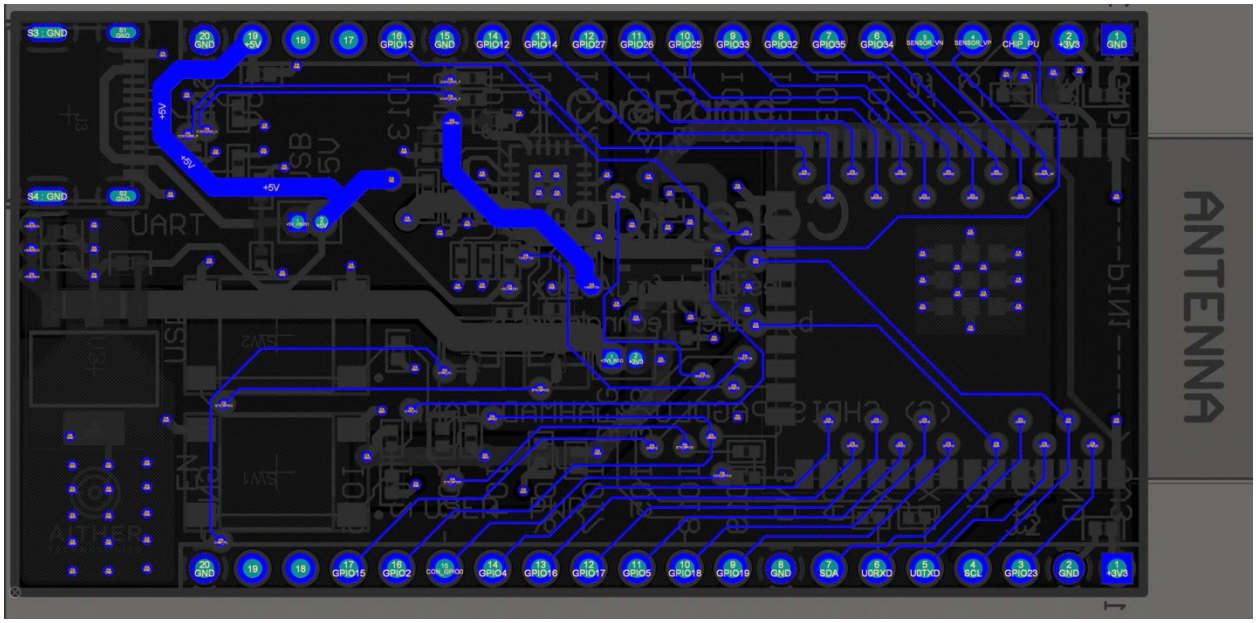


Figure A20. CoreFrame Layer 4 (Bottom Signal Layer)

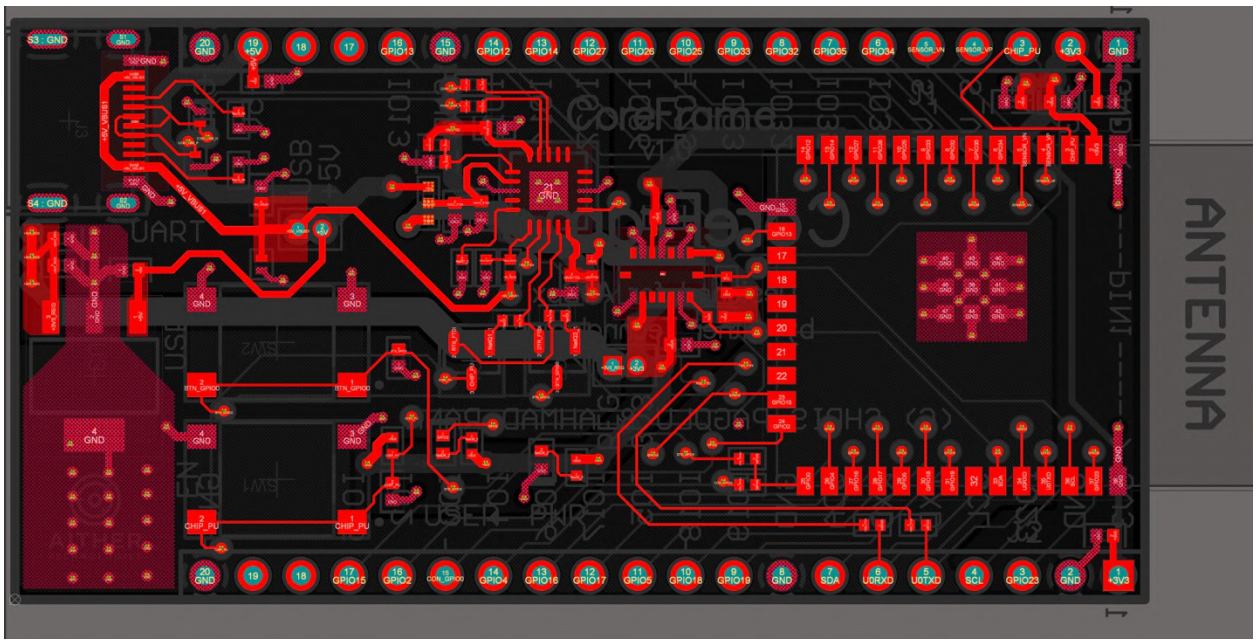


Figure A21. CoreFrame Layer 1 (Top Signal Layer)

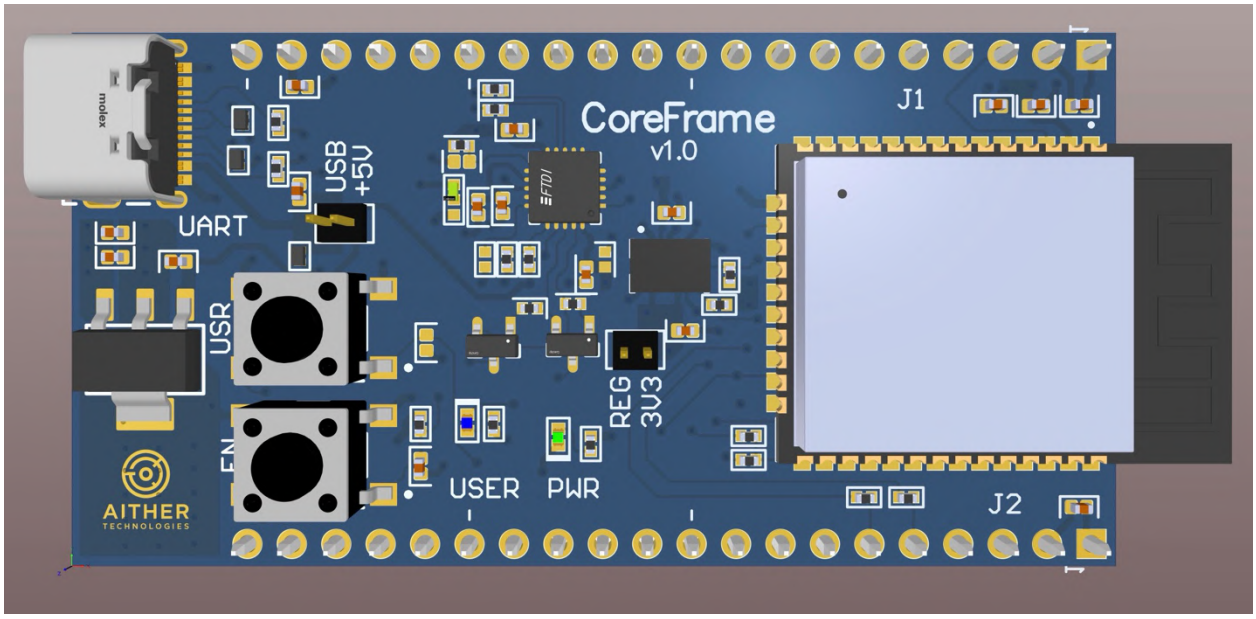


Figure A22. Top CoreFrame 3D view (Altium)

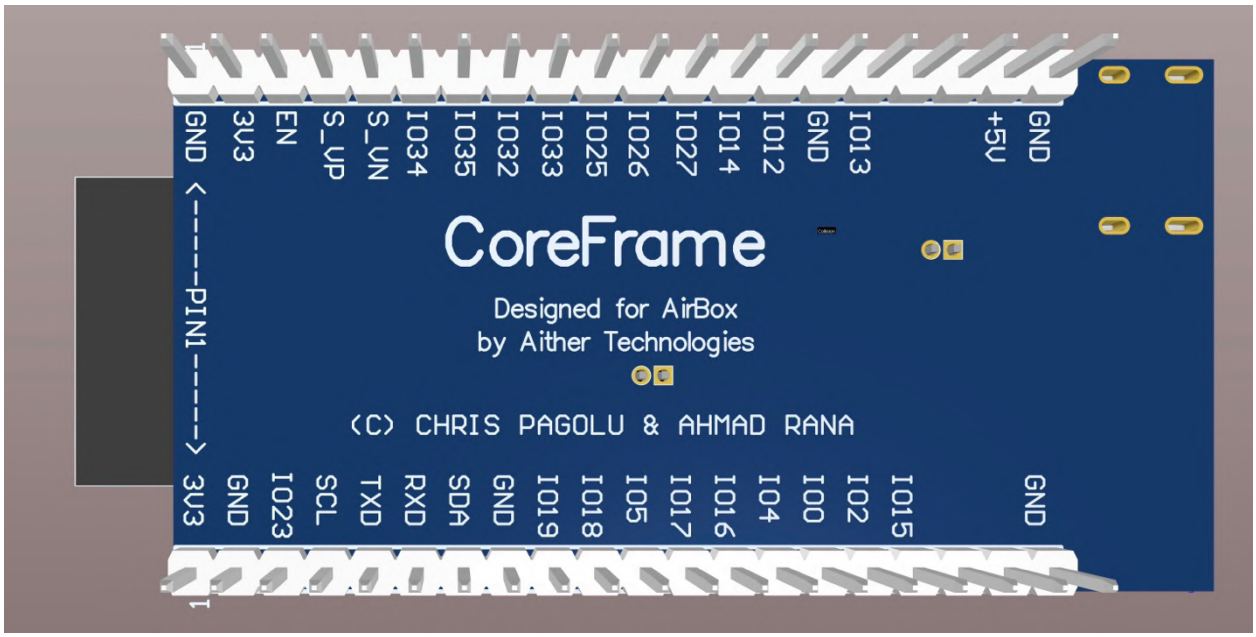


Figure A23. Bottom CoreFrame 3D view (Altium).

Figure A24. CoreFrame in the flesh. To avoid unnecessary clutter, pictures of other PCBs have been attached collectively in Appendix E: Prototype Iterations.

B.2. Power Fabric

This board implements the main 7.2V to 5V synchronous buck converter using the MP2393 IC. The layout was optimised to minimise the area of the high frequency switching loop (aka the hot loop), which includes the input capacitor (C1), MP2393 (U1), inductor (L1), and the switching path. This reduces EMI significantly. Additionally, feedback resistors (R1, R3, R2) were placed close to the MP2393, with the feedback traces routed directly from the output capacitor (C2/C2A1) pads. This Kelvin connection ensures the regulator senses the true output voltage, improving load regulation. Like in the case of CoreFrame, wide copper pours were used for the VIN, VOUT and GND nets to handle the high currents and to act as a heatsink for the MP2393 IC. The board's performance was validated using an oscilloscope

(as mentioned in Section 5.3.1). It successfully provided a stable 5V rail with a peak-to-peak ripple of 112mV under high load, well within the 150mV specification.

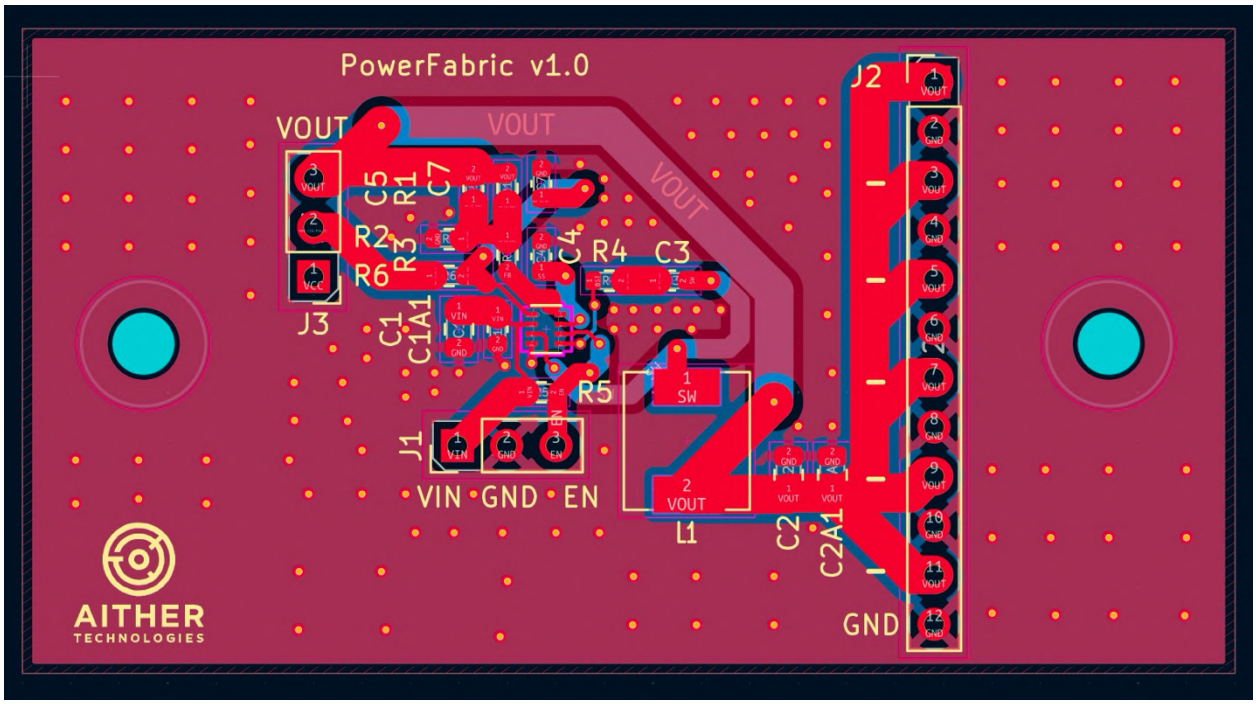


Figure A25. Main Power Fabric 2-layer view.

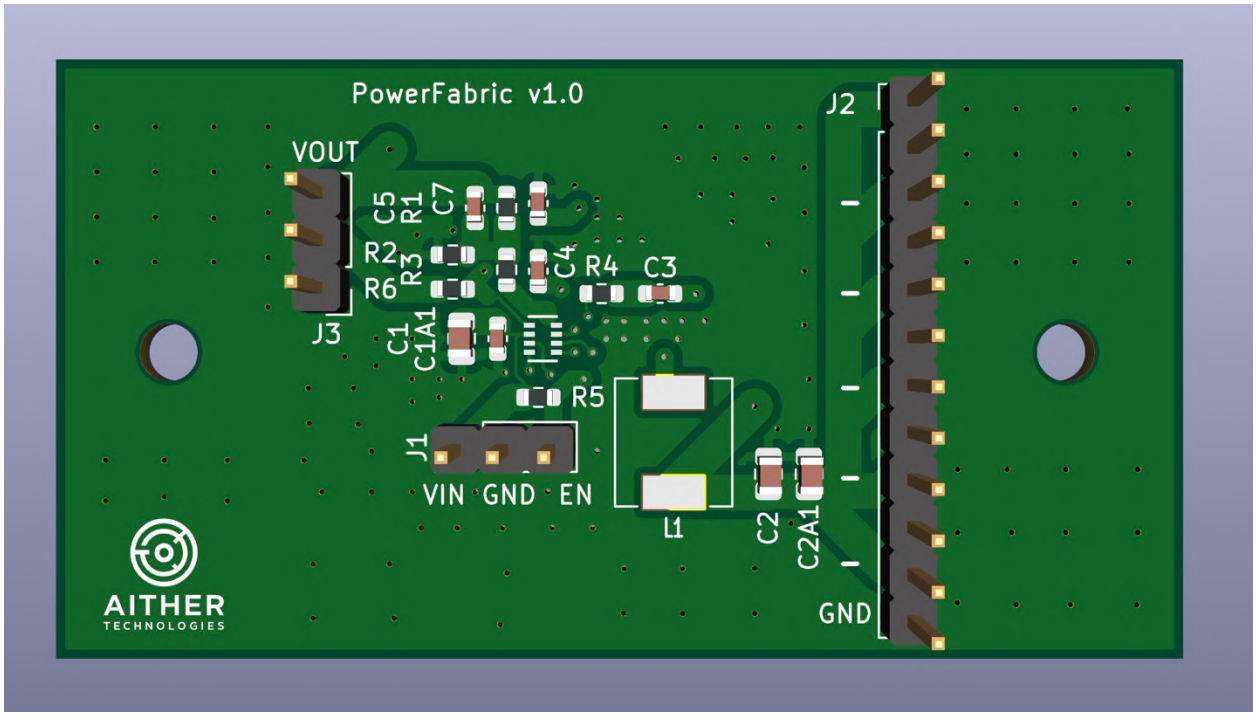


Figure A26. Top 3D view of Power Fabric (KiCad).

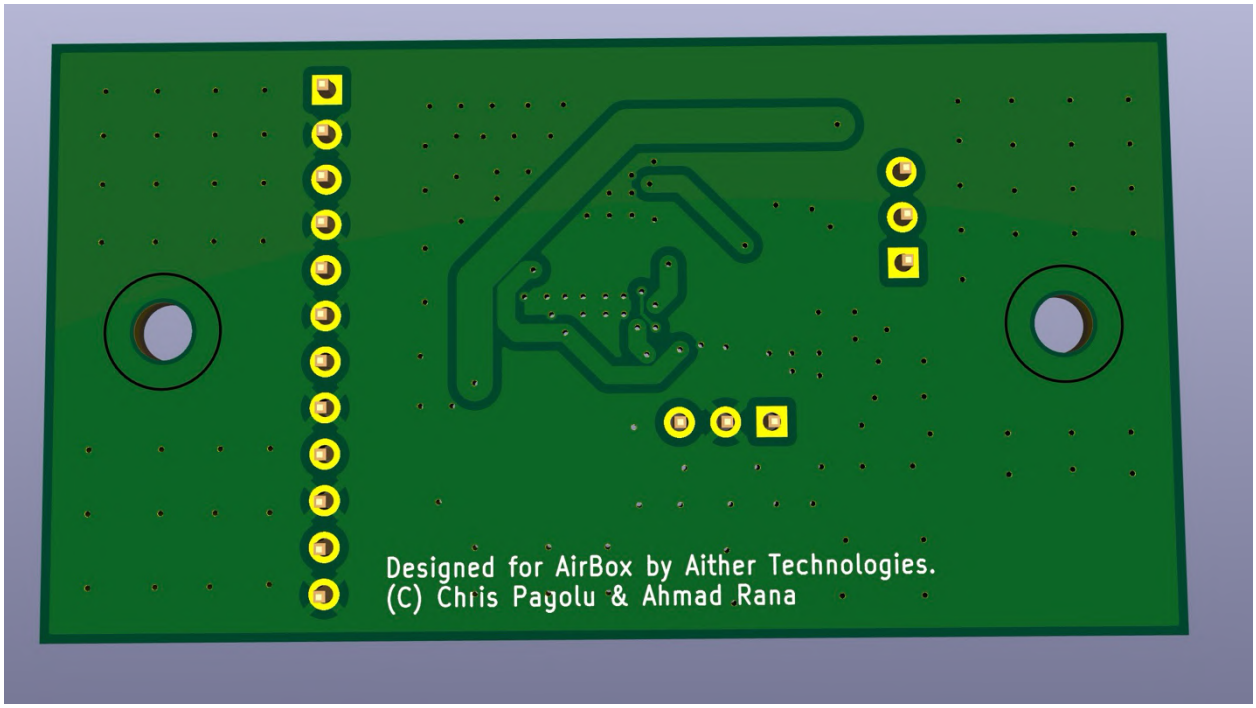


Figure A27. Bottom 3D view of Power Fabric PCB (KiCad).

B.3. Power Interconnect

This board serves as the interface on the AirBox side, providing contact pads for charging and implementing critical protection features such as reverse-polarity and back-feed protection. This is thanks to its primary component: the FDD4141 P-channel MOSFET. Like CoreFrame, large, solid copper pours were used for the high-current path from the contact pads to the output to minimise conduction losses (I^2R) and prevent thermal issues during high-current charging. The contact pads themselves were designed with a large surface area and specified with an ENIG (Electroless Nickel Immersion Gold) finish for durability and low contact resistance. The board was tested for continuity and correct voltage delivery when connected to the PogoFusion board, where it begins charging the 2S battery pack via the TP5100.

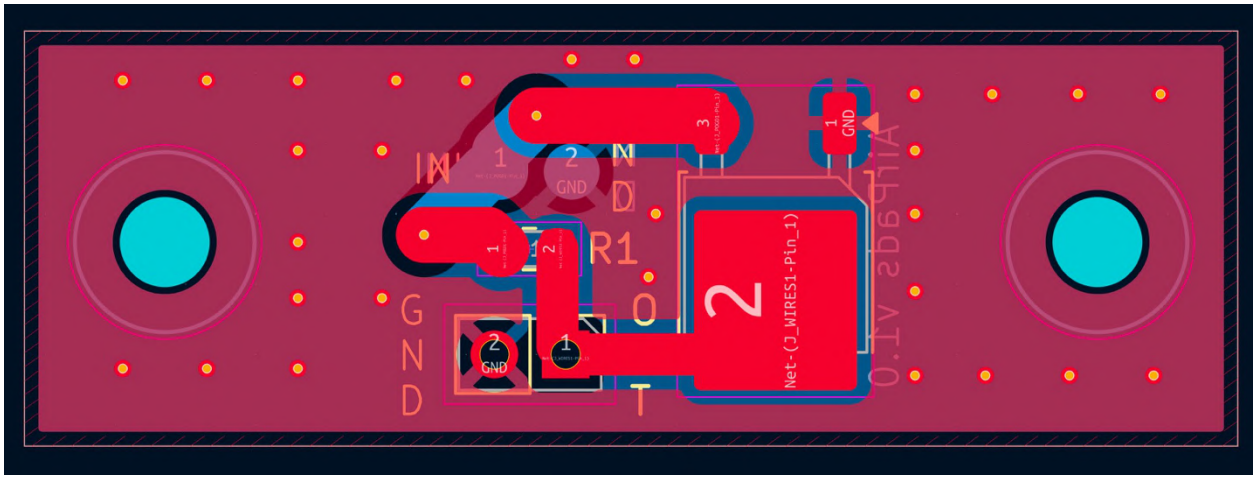


Figure A28. Power Interconnect 2-layer PCB with contact pads.

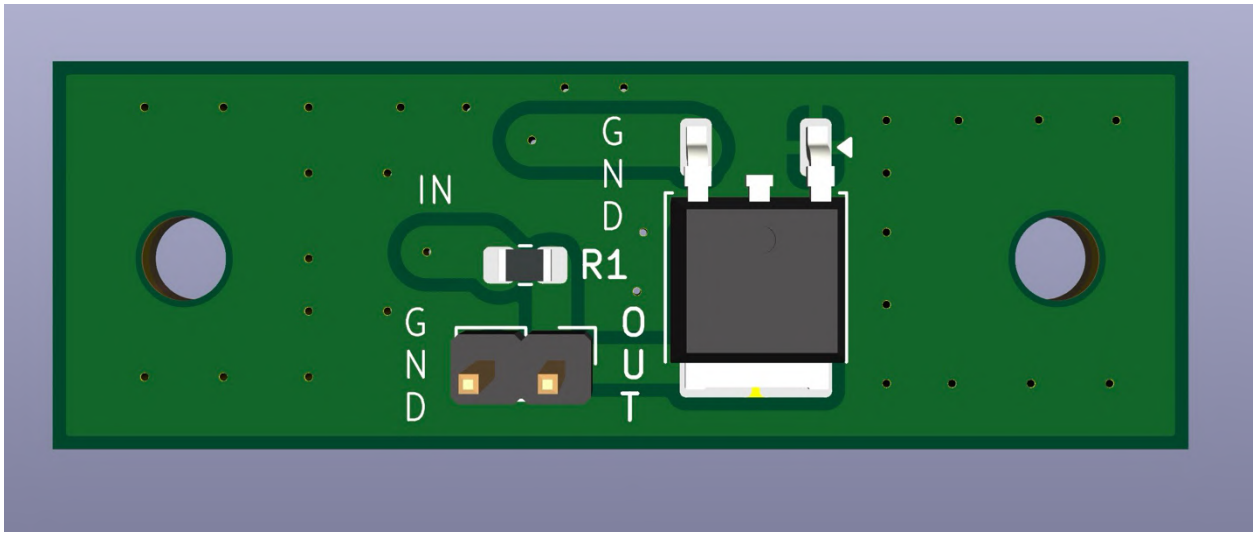


Figure A29. Top 3D view of Power Interconnect PCB (KiCad).

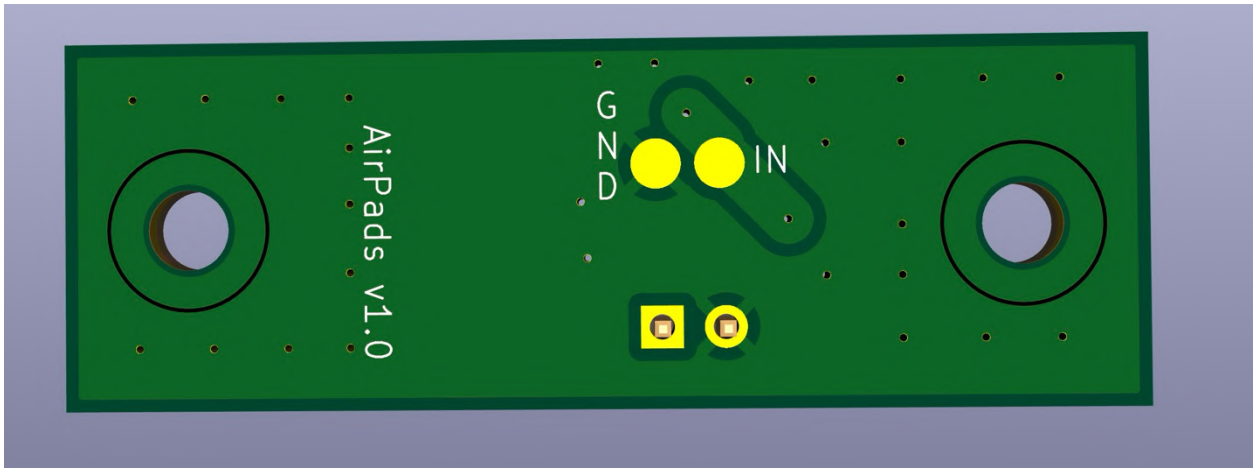


Figure A30. Bottom 3D view of Power Interconnect PCB. The exposed copper pads in gold are the contact pads. During manufacturing, we choose the ENIG gold plating for minimum contact resistance.

B.4. PogoFusion

This board is housed in the AirMount and provides the spring-loaded pogo pin contacts for a robust and convenient docking connection, similar to a standard wireless charging pad. The layout is simple, focusing on providing a direct, low-resistance path from the 9V input to the pogo pins via a resettable PPTC fuse (F1). This ensures any potential short-circuit event at the pogo pins is contained, protecting the batteries and even the upstream USB-C power adapter.

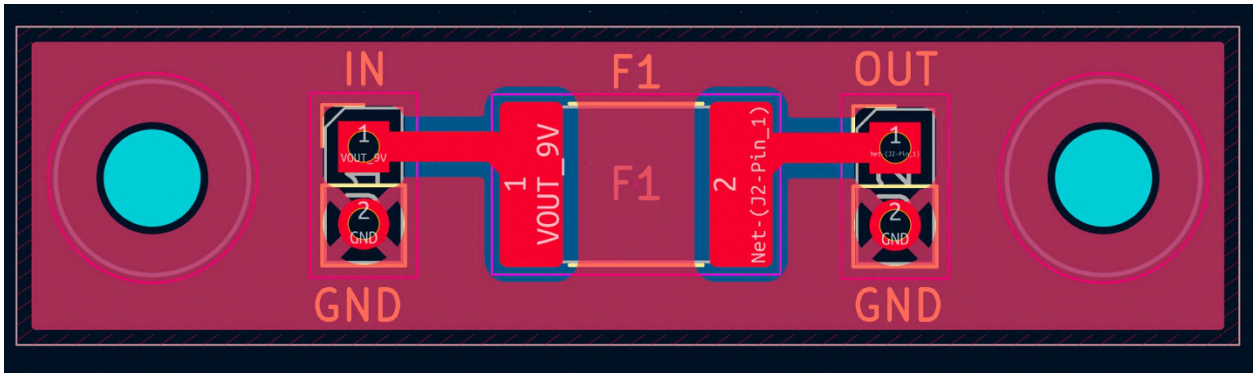


Figure A31. Main PogoFusion 2-layer PCB.

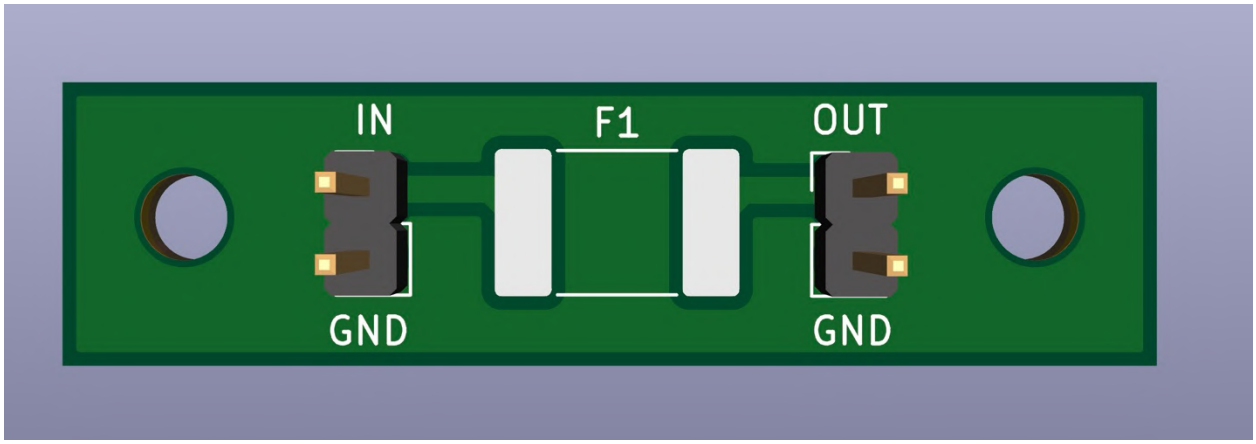


Figure A32. Top 3D view of PogoFusion PCB (KiCad). Of course, instead of headers as shown, the real circuit consists of pogo pins.

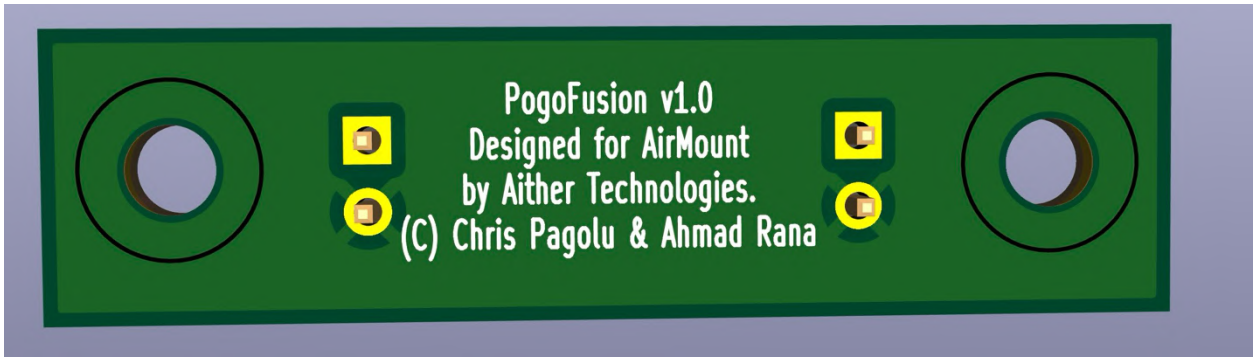


Figure A33. Bottom 3D view of PogoFusion PCB (KiCad).

C: BILL OF MATERIALS

Please note that some prices for unit items are so trivial they are almost entirely negligible and end up as $0. Moreover, miscellaneous costs such as that of PCB shipping ($94.14) has **not** been included in the grand total since shipping varies by location and quantity. Additionally, 3D printing costs are assumed to be negligible.

AirBox Subsystem

C.1. CoreFrame PCB Components

Table 10: CoreFrame Bill of Materials

Component	Part Number / MPN	Designator(s)	Qty	Unit Price (AUD)	Total Price (AUD)
PCB	4-layer Rigid PCB	-	1	A$7.10	A$7.10
MCU	ESP32-WROOM-32E-N4	U1	1	A$5.71	A$5.71
IMU	BMI088	U4	1	A$3.98	A$3.98
USB-UART Bridge	FT231XQ-R	U2	1	A$5.81	A$5.81
LDO (3.3V)	TL1963A-33DCYR	U3	1	A$4.00	A$4.00
USB-C Connector	2171790001	J3	1	A$0.99	A$0.99
Transistor (NPN)	SS8050-G	Q1, Q2	2	A$0.09	A$0.19
LED (Green)	VLMG1500-GS08	D5	1	A$0.25	A$0.25
LED (Blue)	VLMB1500-GS08	D6	1	A$0.26	A$0.26
ESD Protection	AQ3045-01ETG	D1, D2, D3	3	A$0.33	A$0.99
Tactile Switch	PTS645SM43SMTR92LFS	SW1, SW2	2	A$0.31	A$0.62
Capacitor, 22µF	GRM158R61A226ME15D	C14	1	A$0.16	A$0.16

Capacitor, 10µF	CL05A106MP8NUB8	C9, C10	2	A$0.13	A$0.26
Capacitor, 4.7µF	CC0402MRX5R7BB475	C2, C11	2	A$0.03	A$0.06
Capacitor, 100nF	CL05B104KB54PNC	C1, C3-C6, C8, C12, C13, C15, C17, C18	12	A$0.01	A$0.07
Resistor, 1kΩ	RC0402FR-071KL	R18, R19	2	A$0.011	A$0.01
Resistor, 27Ω	RC0402FR-0727RL	R4, R5, R15, R16	4	A$0.01	A$0.01
Resistor, 4.7kΩ	RC0402FR-074K7L	R8, R12, R22	3	A$0.01	A$0.01
Resistor, 5.1kΩ	AC0402FR-075K1L	R9, R10	2	A$0.01	A$0.01
Resistor, 10kΩ	AC0402JR-0710KL	R11, R13, R14, R17	4	A$0.01	A$0.01
Resistor, 0Ω	RC0402FR-070RL	R1, R6, R20, R21	4	A$0.01	A$0.01
Subtotal					**A$30.46**

C.2. Power Fabric PCB Components

Table 11: Power Fabric Bill of Materials

Component	Part Number / MPN	Designator(s)	Qty	Unit Price (AUD)	Total Price (AUD)
PCB	2-layer Rigid PCB	-	1	A$1.22	A$1.22
Buck Converter	MP2393GTL-P	U1	1	A$6.00	A$6.00

Inductor, 2.2µH	SPM6530T-2R2M	L1	1	A$0.25	A$0.25
Capacitor, 22µF	GRM21BR61E226ME44L	C1, C2, C2A1	3	A$0.07	A$0.20
Capacitor, 1µF	CL10A105KO8NNNC	C3	1	A$0.00	A$0.00
Capacitor, 10pF	GRM1885C1H100JA01D	C5	1	A$0.01	A$0.01
Capacitor, 1nF	GRM188R71H102KA01D	C7	1	A$0.01	A$0.01
Capacitor, 0.1µF	GRM188R71E104KA01D	C1A1	1	A$0.01	A$0.01
Capacitor, 6.8nF	CL10B682JB8NNNC	C4	1	A$0.01	A$0.01
Resistor, 7.68kΩ	AT0603BRD077K68L	R2	1	A$0.09	A$0.09
Resistor, 20Ω	RC0603FR-0720RL	R4	1	A$0.00	A$0.00
Resistor, 40.2kΩ	RC0603FR-0740K2L	R1	1	A$0.00	A$0.00
Resistor, 100kΩ	RC0603FR-07100KL	R6	1	A$0.00	A$0.00
Resistor, 15kΩ	RC0603FR-0715KL	R3	1	A$0.00	A$0.00
Resistor, 604kΩ	RC0603FR-07604KL	R5	1	A$0.00	A$0.00
Subtotal					**A$7.80**

C.3. Power Interconnect PCB Components

Table 12: Power Interconnect Bill of Materials

Component	Part Number / MPN	Designator(s)	Qty	Unit Price (AUD)	Total Price (AUD)
PCB	2-layer Rigid PCB	-	1	A$7.71	A$7.71
P-Channel MOSFET	FDD4141	Q1	1	A$2.90	A$2.90
Subtotal					**A$10.61**

C.4. Off-the-Shelf Modules & Hardware

Table 13: Other AirBox components and their BOM

Component	Description	Qty	Unit Price (AUD)	Total Price (AUD)
Battery Charger	TP5100 Module	1	A$7.95	A$7.95
Batteries	Samsung 25R 18650	2	A$7.50	A$15.00
Battery Holder	2x 18650 Holder	1	A$2.39	A$2.39
Speaker	3W 4Ω Speaker	1	A$8.50	A$8.50
Audio Amplifier	Adafruit MAX98357A	1	A$13.70	A$13.70
Subtotal				**A$47.54**

AirMount Subsystem

C.5. PogoFusion PCB Components

Table 14: PogoFusion Bill of Materials

Component	Part Number / MPN	Designator(s)	Qty	Unit Price (AUD)	Total Price (AUD)
PCB	2-layer Rigid PCB	-	1	A$3.49	A$3.49
Pogo Pins	MP015219	J1, J2	2	A$2.91	A$5.82

Resettable Fuse	SMD2920B300TF/30	F1	1	A$1.54	A$1.54
Subtotal					**A$10.85**

C.6. Off-the-Shelf Modules & Hardware

Table 15: Other AirMount components and their BOM

Component	Description	Qty	Unit Price (AUD)	Total Price (AUD)
USB-C PD Trigger	ZY12PDN Module	1	A$7.19	A$7.19
Subtotal				**A$7.19**

ProHalo (Gimbal) Subsystem

Table 16: Components on the ProHalo gimbal and their BOM

Component	Description	Qty	Unit Price (AUD)	Total Price (AUD)
Servo Motors	MG996R Metal Gear	3	A$8.75	A$26.25
Headers	1x12, 1x3, 1x2 Pin Headers	1	A$5.00	A$5.00
Misc. Hardware	Screws, Magnets, etc.	1	A$0.50	A$0.50
Subtotal				**A$31.75**

Grand Total

Table 17: Total Prototype Cost Breakdown

Section	Total Cost (AUD)
1. AirBox Subsystem	A$96.41
2. AirMount Subsystem	A$18.04

3. ProHalo (Gimbal) Subsystem	A$31.75
US$20/month Aerial Pro subscription	**-US$20 (A$30.69)**
Total Prototype Cost	~~**A$146.20**~~ **A$115.51**

D: INITIAL DESIGN SKETCHES

These drawings represent the first visual ideation of the AirFrame's modular architecture and key mechanical features, created in week 1. They serve to illustrate the design journey, from a high-level concept to the detailed schematics and 3D models presented in the main body of this report. What the reader might find fascinating is that the overall design hasn't changed very much from this initial sketch.

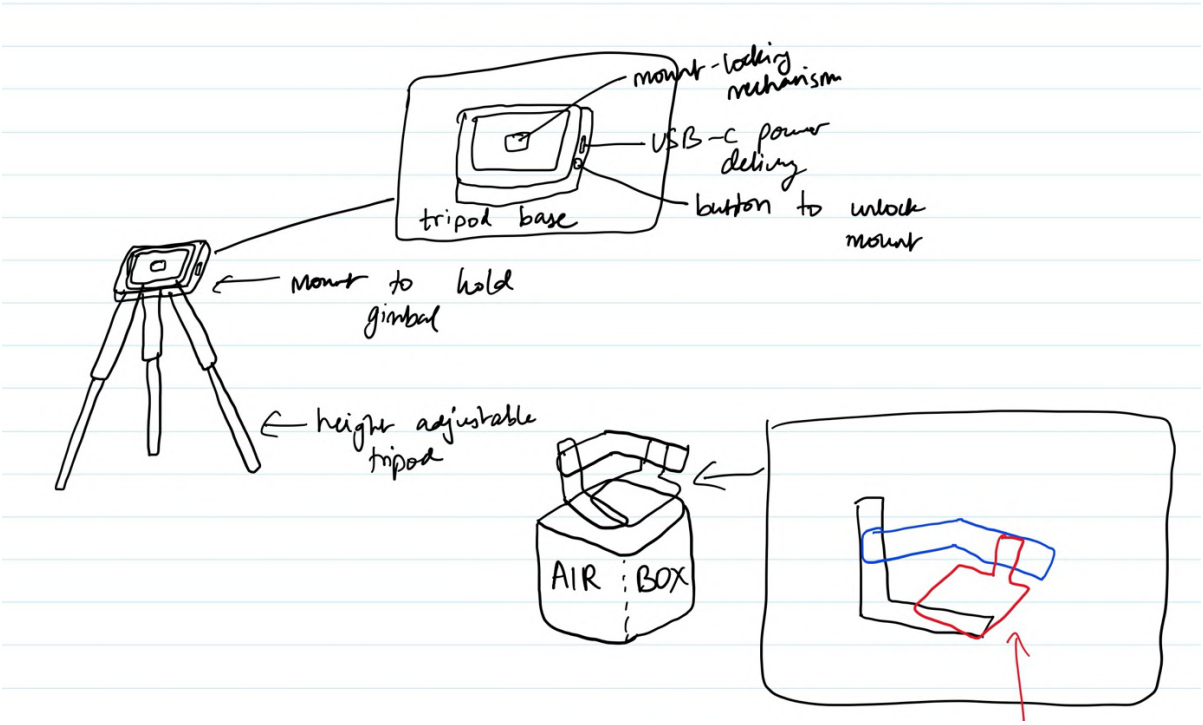


Figure A34. Early design sketch showcasing the first iteration of the AirBox and AirMount, along with the gimbal system.

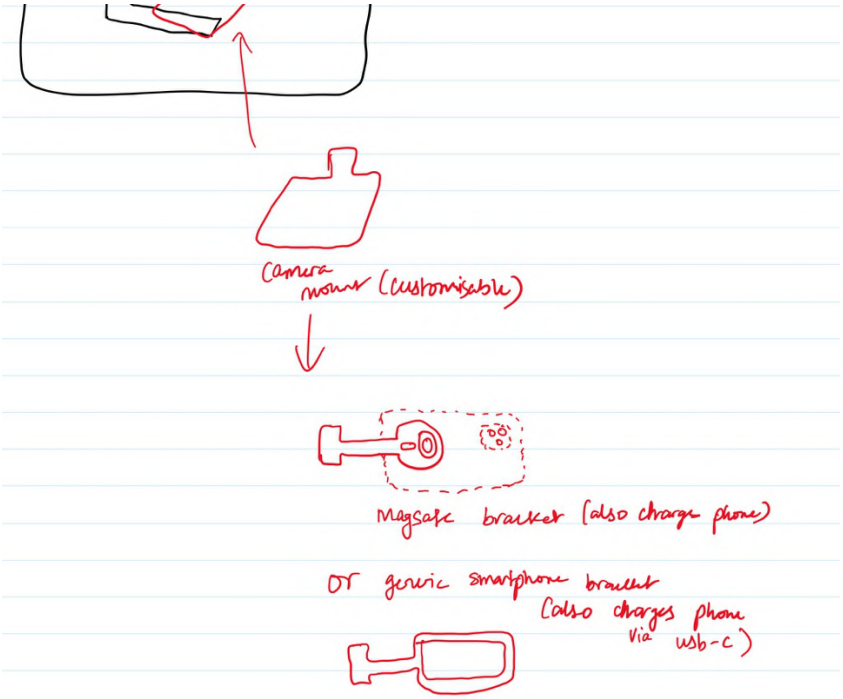


Figure A35. Initial ideation for the payload mount, exploring a customizable camera bracket and options for both MagSafe and generic smartphone charging.

E: PROTOTYPE ITERATIONS: A STORY IN PICTURES

The prototype really began with the Week 7 prototype presentation.

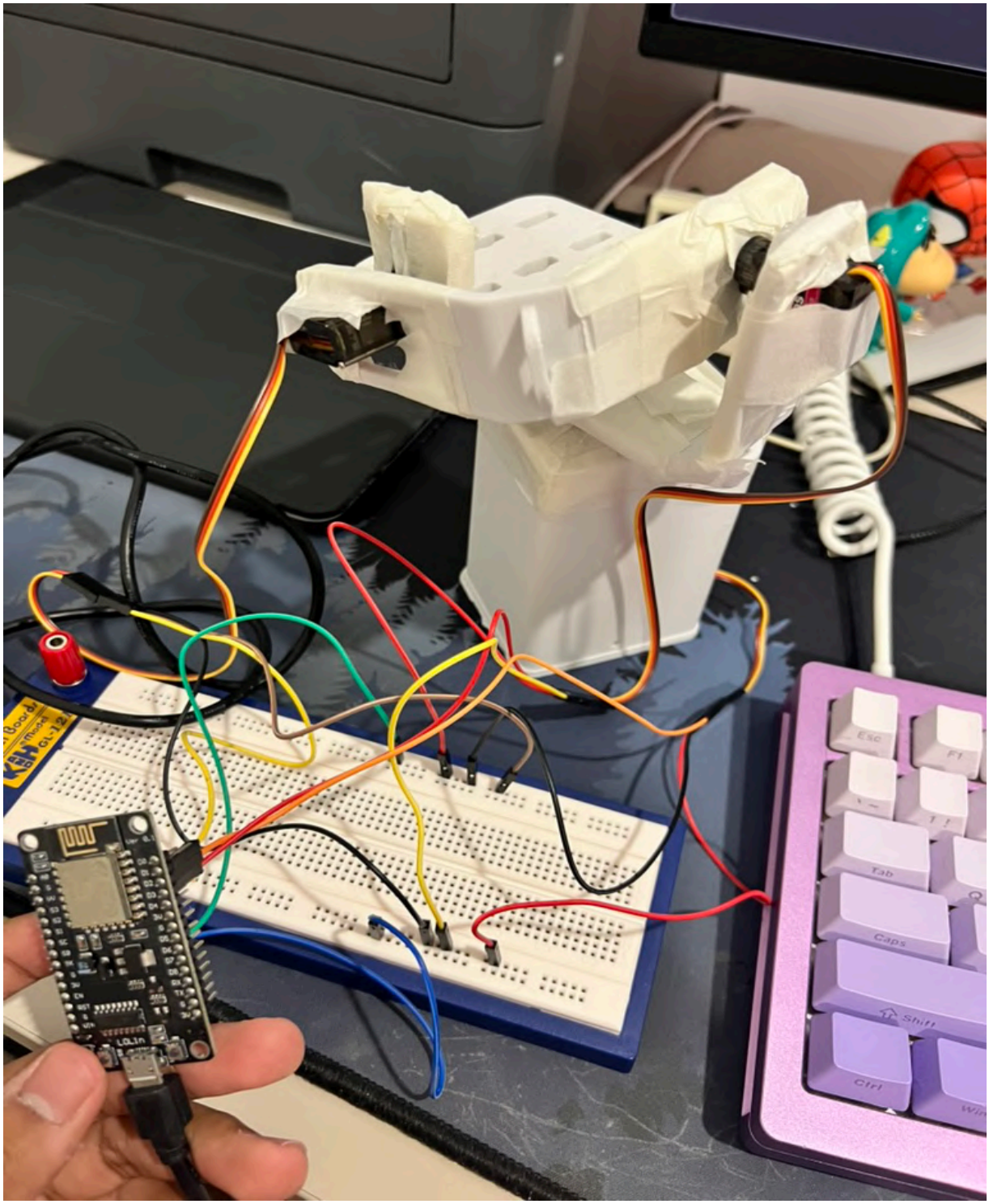


Figure A36. Stage 1: Initial AirFrame prototype (Week 7 Tuesday). Yes, that is, in fact, masking tape.

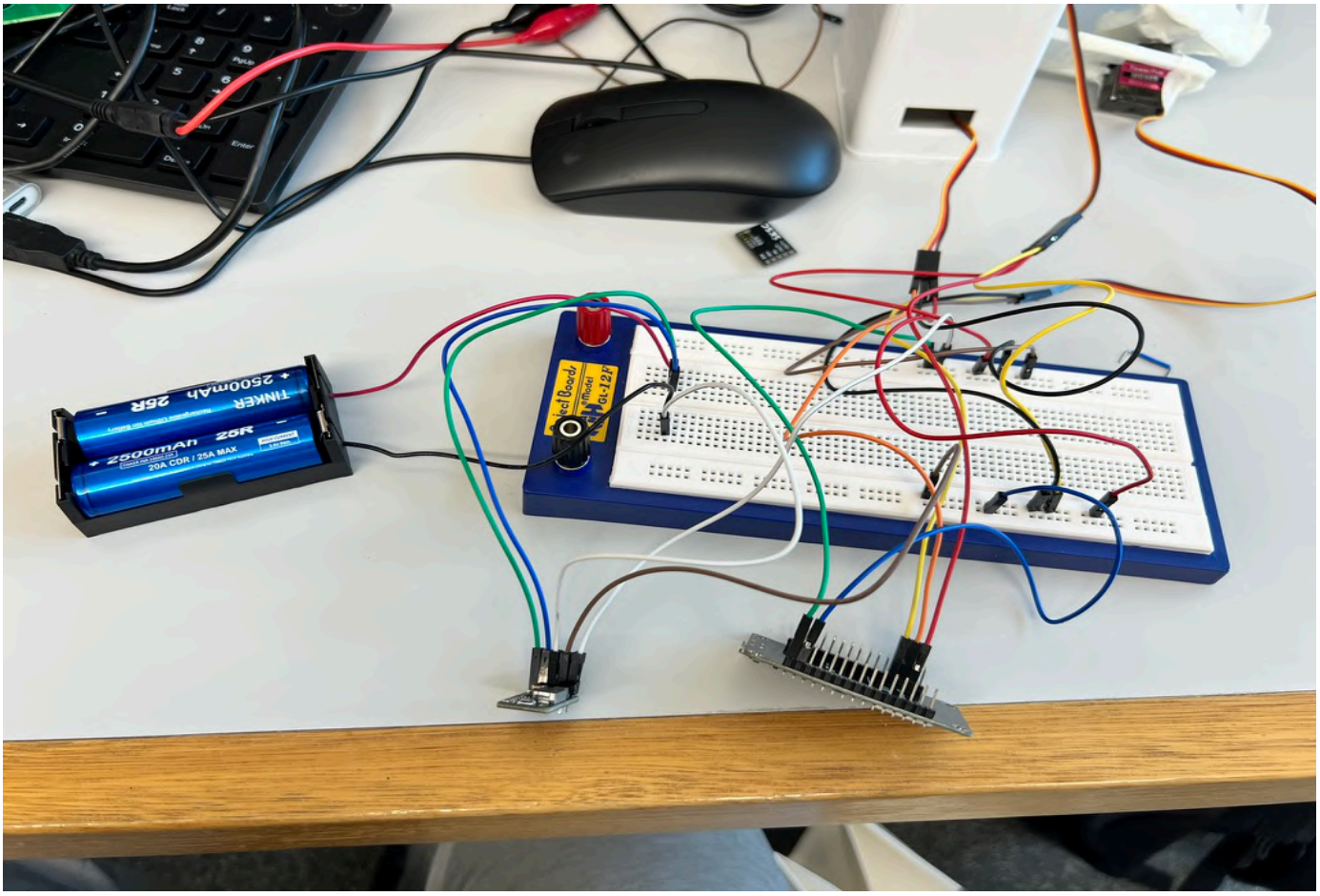


Figure A37. Stage 2: Initial AirFrame prototype hooked up to 2S 18650 Li-ion batteries (Week 7 Thursday). Spoiler: it didn't work.

Figure A38. Stage 3: Second AirFrame prototype (Week 9 Monday).

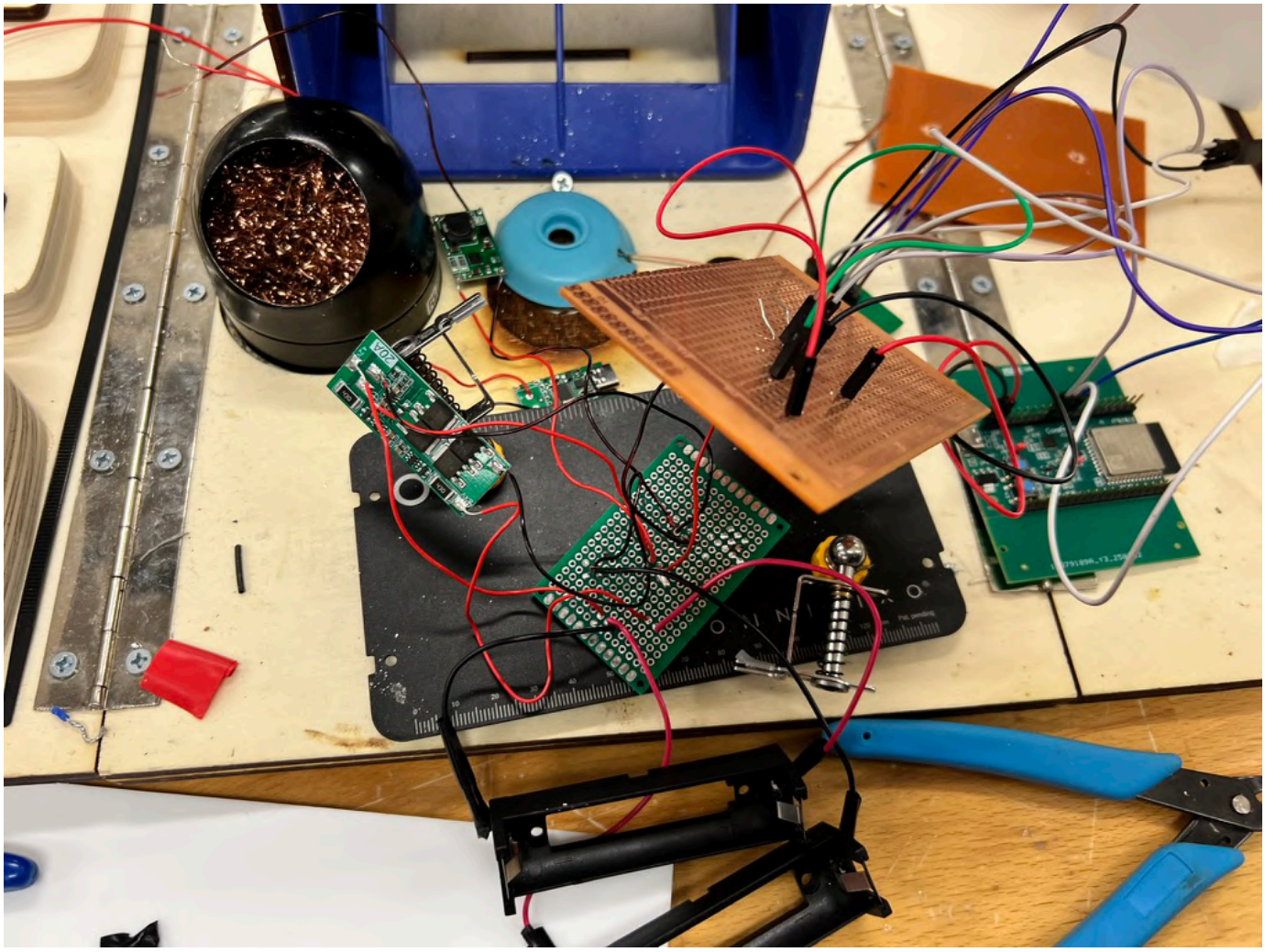


Figure A39. Stage 4: Debugging hell. Circuit wasn't working (Week 9 Friday).

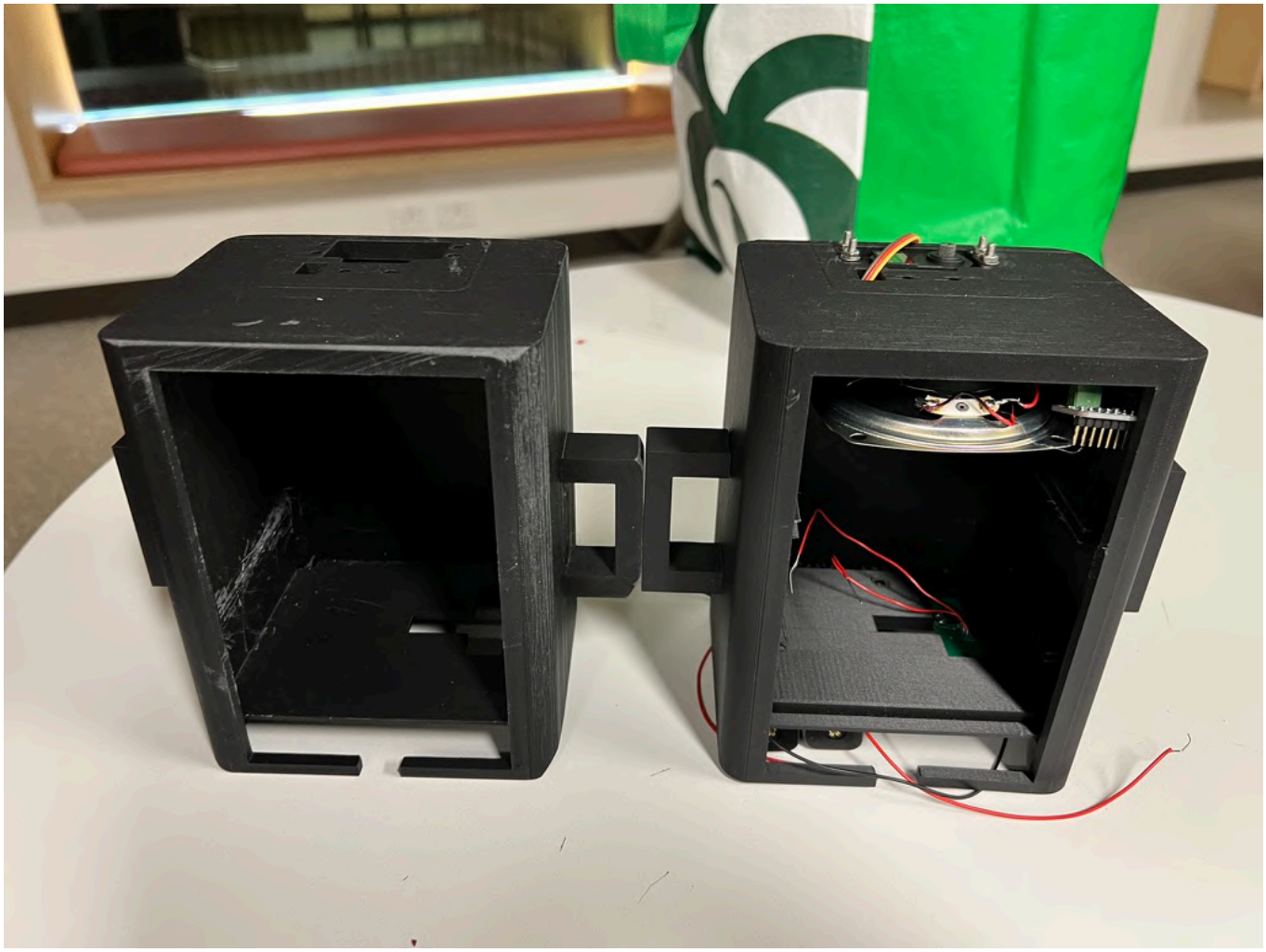


Figure A40. Left: 3rd 3D model prototype. Right: Final 3D model prototype.

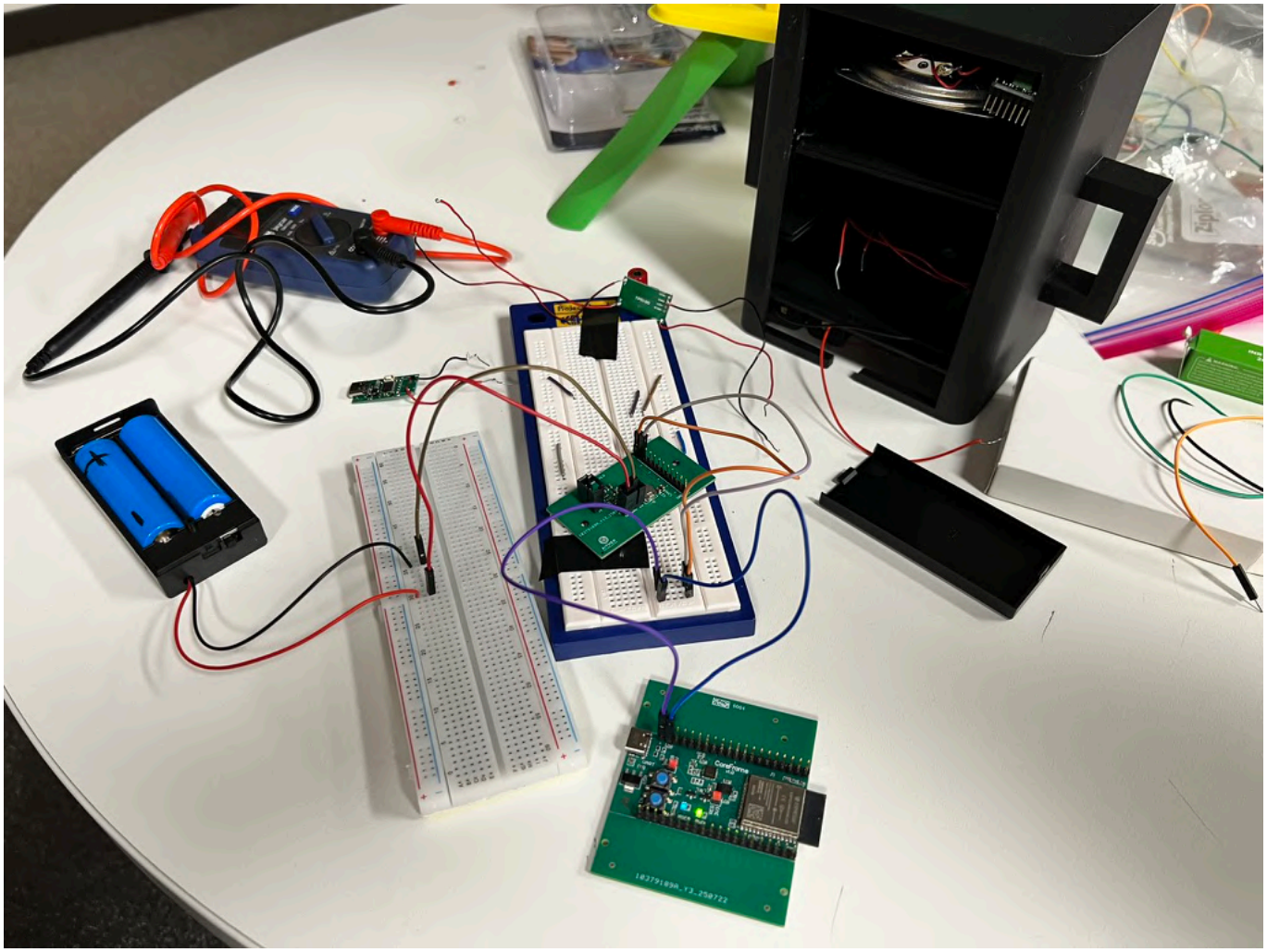


Figure A41. Stage 5: Circuit finally works (CoreFrame LED ON)! Time to assemble. T-12 hours to present.

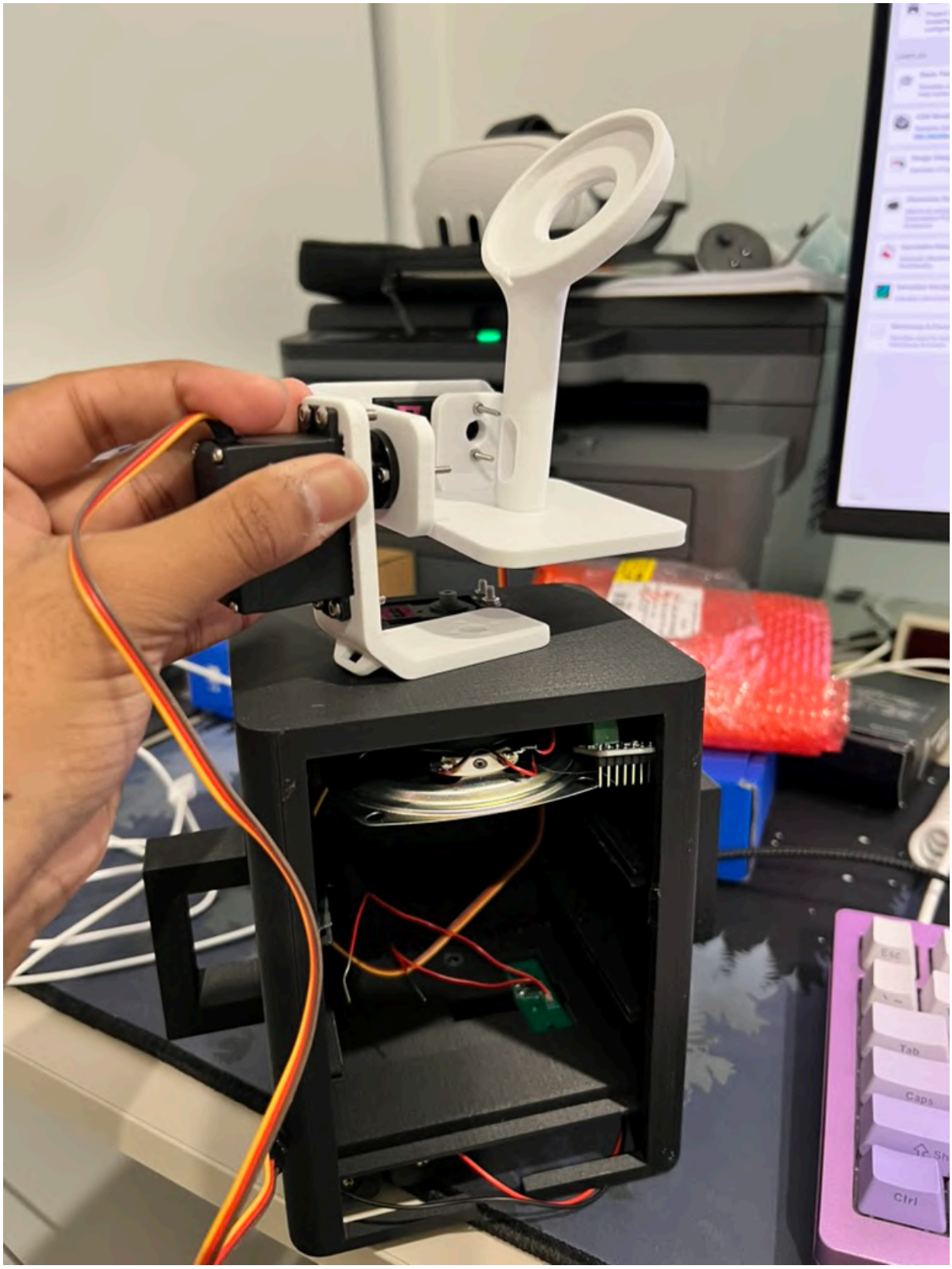


Figure A42. Initial assembly for basic software testing. T-10 hours to present.

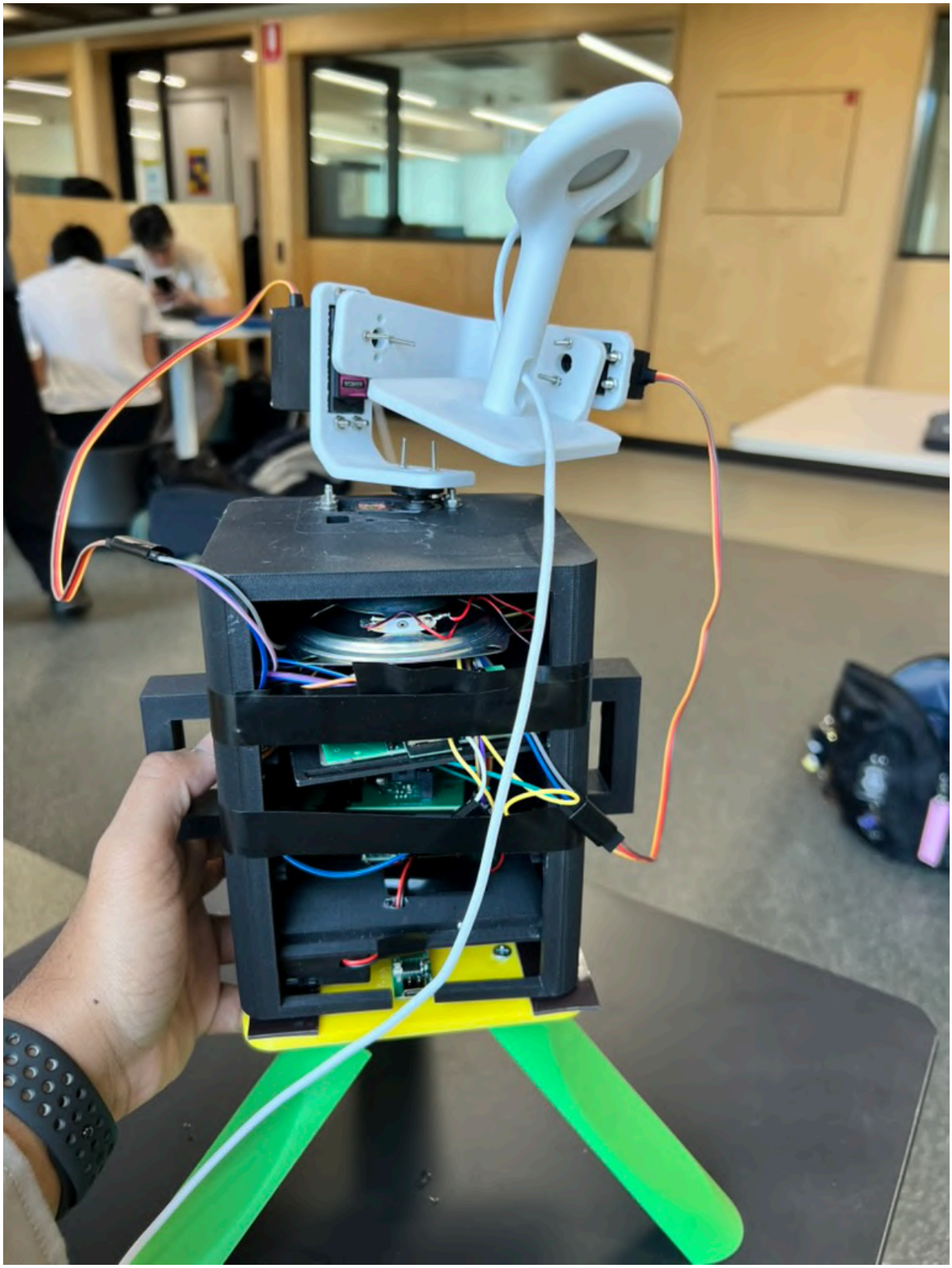


Figure A43. Assembly completed in 4 hours. T-10 mins to present.

Figure A44. Presentation was a success (Week 10 Thursday)! All components and PCBs were working.

F: EXTENDED CODE

AirOS.ino

Both `AirOS.ino` and `AirLink.xcodeproj` will be available on Chris' GitHub profile (https://www.github.com/AllStars101-sudo/AirOS & https://www.github.com/AllStars101-sudo/AirLink). However, `AirLink.xcodeproj` (the iOS app) will **not** be attached to this report due to the sheer size of the project.

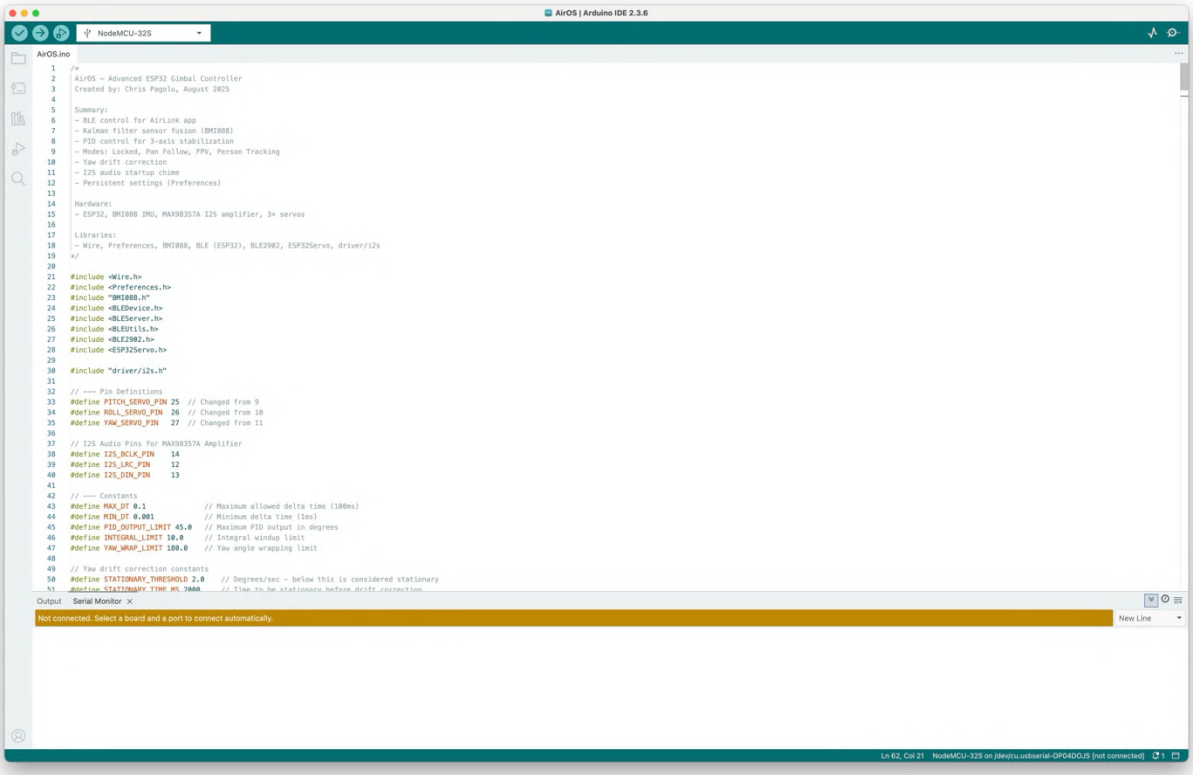


Figure A45. Arduino IDE was chosen because of its ease of setup and use.

```
/*
 AirOS — Advanced ESP32 Gimbal Controller
 Created by: Chris Pagolu, August 2025

 Summary:
 - BLE control for AirLink app
 - Kalman filter sensor fusion (BMI088)
 - PID control for 3-axis stabilization
 - Modes: Locked, Pan Follow, FPV, Person Tracking
 - Yaw drift correction
 - I2S audio startup chime
 - Persistent settings (Preferences)
```

```
 Hardware:
 - ESP32, BMI088 IMU, MAX98357A I2S amplifier, 3× servos

 Libraries:
 - Wire, Preferences, BMI088, BLE (ESP32), BLE2902, ESP32Servo, driver/i2s
*/

#include <Wire.h>
#include <Preferences.h>
#include "BMI088.h"
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <ESP32Servo.h>

#include "driver/i2s.h"

// --- Pin Definitions
#define PITCH_SERVO_PIN 25  // Changed from 9
#define ROLL_SERVO_PIN  26  // Changed from 10
#define YAW_SERVO_PIN   27  // Changed from 11

// I2S Audio Pins for MAX98357A Amplifier
#define I2S_BCLK_PIN    14
#define I2S_LRC_PIN     12
#define I2S_DIN_PIN     13

// --- Constants
#define MAX_DT 0.1              // Maximum allowed delta time (100ms)
#define MIN_DT 0.001            // Minimum delta time (1ms)
#define PID_OUTPUT_LIMIT 45.0   // Maximum PID output in degrees
#define INTEGRAL_LIMIT 10.0     // Integral windup limit
#define YAW_WRAP_LIMIT 180.0    // Yaw angle wrapping limit

// Yaw drift correction constants
#define STATIONARY_THRESHOLD 2.0    // Degrees/sec - below this is considered
stationary
#define STATIONARY_TIME_MS 2000     // Time to be stationary before drift correction
#define YAW_BIAS_ALPHA 0.995        // Low-pass filter for yaw bias estimation
#define MAX_YAW_BIAS 5.0            // Maximum allowed yaw bias (deg/s)
#define LONG_STATIONARY_TIME_MS 5000  // Time for aggressive drift correction
#define YAW_DECAY_RATE 0.98         // Decay factor when stationary (closer to 1 =
slower decay)

// --- Global Objects
Bmi088Accel accel(Wire, 0x18);  // BMI088 accelerometer (SDO1 grounded)
```

```
Bmi088Gyro gyro(Wire, 0x68);    // BMI088 gyroscope (SDO2 grounded)
Preferences prefs;

// --- Servo Objects
Servo pitchServo;
Servo rollServo;
Servo yawServo;

// --- Timing & Control
unsigned long timer = 0;
float dt = 0;

// --- BLE UUIDs (Universally Unique Identifiers)
#define SERVICE_UUID           "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
// READ-ONLY Characteristics (ESP32 to App)
#define PITCH_ANGLE_CHAR_UUID  "a1e8f36e-685b-4869-9828-c107a6729938"
#define ROLL_ANGLE_CHAR_UUID   "43a85368-8422-4573-a554-411a4a6e87f1"
#define YAW_ANGLE_CHAR_UUID    "e974ac4a-8182-4458-9419-4ac9c6c5184e"
#define GIMBAL_STATUS_CHAR_UUID "c8a4a58b-1579-4451-b016-1f38e3115a3a"
// READ-WRITE Characteristics (App to ESP32)
#define GIMBAL_MODE_CHAR_UUID      "2a79d494-436f-45b6-890f-563534ab2c84"
#define GIMBAL_CONTROL_CHAR_UUID   "f7a7a5a8-5e58-4c8d-9b6e-3aa5d6c5b768"
#define P_GAIN_CHAR_UUID           "b16b472c-88a4-4734-9f85-01458e08d669"
#define I_GAIN_CHAR_UUID           "8184457e-85a8-4217-a9a3-a7d57947a612"
#define D_GAIN_CHAR_UUID           "5d9b73b3-81e0-4368-910a-e322359b8676"
#define KALMAN_PARAMS_CHAR_UUID    "6e13e51a-f3c2-46a4-b203-92147395c5d0"

BLECharacteristic *pPitchAngleCharacteristic;
BLECharacteristic *pRollAngleCharacteristic;
BLECharacteristic *pYawAngleCharacteristic;
BLECharacteristic *pGimbalStatusCharacteristic;


// --- Data Structures
enum GimbalMode {
    MODE_INACTIVE,
    MODE_LOCKED,
    MODE_PAN_FOLLOW,
    MODE_FPV,
    MODE_PERSON_TRACKING
};

enum GimbalStatus {
    STATUS_INACTIVE,
    STATUS_CALIBRATING,
    STATUS_LOCKED,
    STATUS_PAN_FOLLOW,
    STATUS_FPV,
```

```
    STATUS_PERSON_TRACKING
};

struct PIDSettings {
    float p, i, d;
};

struct GimbalSettings {
    GimbalMode mode;
    PIDSettings pitchPID;
    PIDSettings rollPID;
    PIDSettings yawPID;
    float kalman_q_angle;
    float kalman_q_bias;
    float kalman_r_measure;
};

GimbalSettings settings;

struct PIDController {
    float integral = 0;
    float prevError = 0;

    void reset() {
        integral = 0;
        prevError = 0;
    }
};

PIDController pitchController, rollController, yawController;

struct KalmanFilter {
    float Q_angle, Q_bias, R_measure;
    float angle = 0, bias = 0, rate = 0;
    float P[2][2];

    void init(float q_angle, float q_bias, float r_measure) {
        Q_angle = q_angle;
        Q_bias = q_bias;
        R_measure = r_measure;
        angle = 0;
        bias = 0;
        rate = 0;
        // Initialize P matrix with appropriate values for faster convergence
        P[0][0] = 1.0;    // Initial angle uncertainty
        P[0][1] = 0.0;
        P[1][0] = 0.0;
        P[1][1] = 1.0;    // Initial bias uncertainty
```

```
    }
};

KalmanFilter kalmanX, kalmanY;

float pitchOffset = 0, rollOffset = 0, yawOffset = 0;
bool isCalibrated = false;

// Pan follow mode variables
float yawSetpointTarget = 0;  // Target yaw for pan follow
float panFollowRate = 0.5;    // Rate of pan following (0-1)

// Person tracking variables
float personTrackingPanSpeed = 0;   // Person tracking pan speed (-45 to +45 deg/s)
float personTrackingTiltSpeed = 0;  // Person tracking tilt speed (-45 to +45 deg/s)
unsigned long lastPersonTrackingUpdate = 0;  // Last time tracking data was received
#define PERSON_TRACKING_TIMEOUT_MS 1000  // Switch back to locked if no data for 1
second

// Yaw drift correction variables
struct YawDriftCorrection {
    float yawBias = 0;              // Estimated gyro bias (deg/s)
    float yawAngle = 0;             // Drift-corrected yaw angle
    unsigned long stationaryStart = 0;  // When stationary period started
    bool isStationary = false;      // Currently stationary flag
    float lastGyroYaw = 0;          // For stationary detection

    void reset() {
        yawBias = 0;
        yawAngle = 0;
        stationaryStart = 0;
        isStationary = false;
        lastGyroYaw = 0;
    }
};

YawDriftCorrection yawCorrection;

// --- Audio Data for Startup Chime
// A simple C-major arpeggio (C4-E4-G4-C5) at 8000Hz sample rate
const int16_t startup_chime[] = {
    0, 4095, 8191, 12287, 16383, 20479, 24575, 28671, 32767, 28671, 24575, 20479,
16383, 12287, 8191, 4095,
    0, -4095, -8191, -12287, -16383, -20479, -24575, -28671, -32767, -28671, -24575, -
20479, -16383, -12287, -8191, -4095,
    0, 5147, 10294, 15441, 20588, 25735, 30882, 30882, 25735, 20588, 15441, 10294,
5147, 0, -5147, -10294,
```

```
    -15441, -20588, -25735, -30882, -30882, -25735, -20588, -15441, -10294, -5147, 0,
6155, 12310, 18465, 24620,
    30775, 30775, 24620, 18465, 12310, 6155, 0, -6155, -12310, -18465, -24620, -30775,
-30775, -24620, -18465,
    -12310, -6155, 0, 8191, 16383, 24575, 32767, 32767, 24575, 16383, 8191, 0, -8191,
-16383, -24575, -32767,
    -32767, -24575, -16383, -8191, 0
};
const size_t chime_samples = sizeof(startup_chime) / sizeof(startup_chime[0]);

// --- Function Prototypes
void saveSettings();
void loadSettings();
void applyDefaultSettings();
void performCalibration();
float wrapAngle(float angle);
void updateYawWithDriftCorrection(float gyroYaw, float dt);
void resetYawAngle();
void initializeKalmanWithCurrentPosition();

// --- BLE Callbacks
class GimbalControlCallback: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        String value = pCharacteristic->getValue();
        if (value.length() > 0) {
            // Debug: Show received command
            Serial.println("=== BLE COMMAND RECEIVED ===");
            Serial.print("📱 iOS App sent: '"); Serial.print(value);
Serial.println("'");

            value.toLowerCase();
            if (value == "calibrate") {
                Serial.println("🎯 Executing: CALIBRATE");
                performCalibration();
            } else if (value == "save") {
                Serial.println("💾 Executing: SAVE SETTINGS");
                saveSettings();
            } else if (value == "defaults") {
                Serial.println("🔄 Executing: RESTORE DEFAULTS");
                applyDefaultSettings();
                saveSettings();
            } else if (value == "reset_pid") {
                Serial.println("🎛️ Executing: RESET PID");
                pitchController.reset();
                rollController.reset();
                yawController.reset();
                Serial.println("PID controllers reset");
            } else if (value == "reset_yaw") {
```

```
                Serial.println("🧭 Executing: RESET YAW");
                resetYawAngle();
                Serial.println("Yaw angle reset to 0");
            } else if (value == "yaw_status") {
                Serial.println("📊 Executing: YAW STATUS");
                Serial.print("Yaw Status - Angle: ");
Serial.print(yawCorrection.yawAngle, 2);
                Serial.print("°, Bias: "); Serial.print(yawCorrection.yawBias, 4);
                Serial.print("°/s, Stationary: ");
Serial.println(yawCorrection.isStationary ? "YES" : "NO");
            } else if (value == "force_decay") {
                Serial.println("⚡ Executing: FORCE DECAY");
                yawCorrection.yawAngle *= 0.9;  // Immediate 10% decay
                Serial.print("Forced yaw decay - New angle: ");
Serial.println(yawCorrection.yawAngle, 2);
            } else if (value.startsWith("track_pan:")) {
                Serial.println("🎯 Executing: PERSON TRACKING PAN");
                personTrackingPanSpeed = value.substring(10).toFloat();
                lastPersonTrackingUpdate = millis();
                Serial.print("Pan tracking speed: ");
Serial.println(personTrackingPanSpeed);
            } else if (value.startsWith("track_tilt:")) {
                Serial.println("🎯 Executing: PERSON TRACKING TILT");
                personTrackingTiltSpeed = value.substring(11).toFloat();
                lastPersonTrackingUpdate = millis();
                Serial.print("Tilt tracking speed: ");
Serial.println(personTrackingTiltSpeed);
            } else {
                Serial.print("❓ Unknown command: '"); Serial.print(value);
Serial.println("'");
            }
            Serial.println("============================");
        }
    }
};

class GimbalModeCallback: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        uint8_t* data = pCharacteristic->getData();
        if (pCharacteristic->getLength() > 0) {
            GimbalMode newMode = (GimbalMode)data[0];

            // Debug: Show mode change
            Serial.println("=== MODE CHANGE RECEIVED ===");
            Serial.print("📱 iOS App sent mode: "); Serial.print(data[0]);
Serial.print(" (");
            switch(newMode) {
                case MODE_INACTIVE: Serial.print("INACTIVE"); break;
```

```
                case MODE_LOCKED: Serial.print("LOCKED"); break;
                case MODE_PAN_FOLLOW: Serial.print("PAN_FOLLOW"); break;
                case MODE_FPV: Serial.print("FPV"); break;
                case MODE_PERSON_TRACKING: Serial.print("PERSON_TRACKING"); break;
                default: Serial.print("UNKNOWN"); break;
            }
            Serial.println(")");

            if (newMode != settings.mode) {
                Serial.print("🔄 Changing from mode "); Serial.print(settings.mode);
                Serial.print(" to mode "); Serial.println(newMode);

                // Reset PID controllers when changing modes
                pitchController.reset();
                rollController.reset();
                yawController.reset();
                settings.mode = newMode;
                Serial.println("✅ Mode change completed, PID reset");
            } else {
                Serial.println("⏭️ Mode unchanged");
            }
            Serial.println("============================");
        }
    }
};

// Generic PID Gain Callback
void updatePIDGain(BLECharacteristic *pCharacteristic, PIDSettings &pid) {
    uint8_t* data = pCharacteristic->getData();
    if (pCharacteristic->getLength() == sizeof(PIDSettings)) {
        memcpy(&pid, data, sizeof(PIDSettings));
        Serial.println("Updated PID gains.");
    }
}

class PIDGainCallback: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        String uuid = pCharacteristic->getUUID().toString();
        uint8_t* data = pCharacteristic->getData();

        Serial.println("=== PID UPDATE RECEIVED ===");
        Serial.print("📱 iOS App sent PID data, length: ");
Serial.println(pCharacteristic->getLength());
        Serial.print("Expected length: "); Serial.println(sizeof(PIDSettings));

        if (pCharacteristic->getLength() == sizeof(PIDSettings)) {
            PIDSettings newPID;
            memcpy(&newPID, data, sizeof(PIDSettings));
```

```
            if (uuid == P_GAIN_CHAR_UUID) {
                Serial.println("🎛️ Updating PITCH PID gains:");
                Serial.print("  P: "); Serial.print(newPID.p, 4);
                Serial.print(", I: "); Serial.print(newPID.i, 4);
                Serial.print(", D: "); Serial.println(newPID.d, 4);
                memcpy(&settings.pitchPID, data, sizeof(PIDSettings));
            } else if (uuid == I_GAIN_CHAR_UUID) {
                Serial.println("🎛️ Updating ROLL PID gains:");
                Serial.print("  P: "); Serial.print(newPID.p, 4);
                Serial.print(", I: "); Serial.print(newPID.i, 4);
                Serial.print(", D: "); Serial.println(newPID.d, 4);
                memcpy(&settings.rollPID, data, sizeof(PIDSettings));
            } else if (uuid == D_GAIN_CHAR_UUID) {
                Serial.println("🎛️ Updating YAW PID gains:");
                Serial.print("  P: "); Serial.print(newPID.p, 4);
                Serial.print(", I: "); Serial.print(newPID.i, 4);
                Serial.print(", D: "); Serial.println(newPID.d, 4);
                memcpy(&settings.yawPID, data, sizeof(PIDSettings));
            } else {
                Serial.print("❓ Unknown PID UUID: "); Serial.println(uuid);
            }
        } else {
            Serial.println("❌ Invalid PID data length");
        }
        Serial.println("============================");
    }
};


// --- Utility Functions
float wrapAngle(float angle) {
    while (angle > YAW_WRAP_LIMIT) angle -= 360.0;
    while (angle < -YAW_WRAP_LIMIT) angle += 360.0;
    return angle;
}

float constrainFloat(float value, float min_val, float max_val) {
    if (value > max_val) return max_val;
    if (value < min_val) return min_val;
    return value;
}

// --- PID Logic with Improvements
float calculatePID(float setpoint, float input, PIDController& controller, const
PIDSettings& pid) {
    // Ensure dt is within reasonable bounds
    if (dt < MIN_DT || dt > MAX_DT) {
```

```
        return 0; // Return 0 output for invalid dt
    }

    float error = setpoint - input;

    // Integral term with windup protection
    controller.integral += error * dt;
    controller.integral = constrainFloat(controller.integral, -INTEGRAL_LIMIT,
INTEGRAL_LIMIT);

    // Derivative term
    float derivative = (error - controller.prevError) / dt;

    // Calculate output
    float output = (pid.p * error) + (pid.i * controller.integral) + (pid.d *
derivative);

    // Limit output
    output = constrainFloat(output, -PID_OUTPUT_LIMIT, PID_OUTPUT_LIMIT);

    controller.prevError = error;
    return output;
}

// --- Enhanced Kalman Filter Logic
void initKalman(KalmanFilter* kalman) {
    kalman->init(settings.kalman_q_angle, settings.kalman_q_bias,
settings.kalman_r_measure);
}

float kalmanCalculate(KalmanFilter* kalman, float newAngle, float newRate, float p_dt)
{
    // Bounds check on dt
    if (p_dt < MIN_DT || p_dt > MAX_DT) {
        return kalman->angle; // Return previous angle for invalid dt
    }

    // Prediction step
    kalman->rate = newRate - kalman->bias;
    kalman->angle += p_dt * kalman->rate;

    // Prediction covariance update
    float dt2 = p_dt * p_dt;
    kalman->P[0][0] += p_dt * (p_dt * kalman->P[1][1] - kalman->P[0][1] - kalman-
>P[1][0] + kalman->Q_angle);
    kalman->P[0][1] -= p_dt * kalman->P[1][1];
    kalman->P[1][0] -= p_dt * kalman->P[1][1];
    kalman->P[1][1] += kalman->Q_bias * p_dt;
```

```
    // Ensure P matrix symmetry (prevent numerical drift)
    kalman->P[0][1] = kalman->P[1][0];

    // Update step
    float S = kalman->P[0][0] + kalman->R_measure;

    // Prevent division by very small numbers
    if (S < 1e-6) S = 1e-6;

    float K[2] = {kalman->P[0][0] / S, kalman->P[1][0] / S};

    float y = newAngle - kalman->angle;
    kalman->angle += K[0] * y;
    kalman->bias += K[1] * y;

    // Covariance update using Joseph form (numerically stable)
    float P00_temp = kalman->P[0][0];
    float P01_temp = kalman->P[0][1];

    kalman->P[0][0] -= K[0] * P00_temp;
    kalman->P[0][1] -= K[0] * P01_temp;
    kalman->P[1][0] -= K[1] * P00_temp;
    kalman->P[1][1] -= K[1] * P01_temp;

    // Ensure P matrix stays positive definite
    if (kalman->P[0][0] < 1e-6) kalman->P[0][0] = 1e-6;
    if (kalman->P[1][1] < 1e-6) kalman->P[1][1] = 1e-6;

    return kalman->angle;
}

// --- IMU Helpers
float getPitch(float accelX, float accelY, float accelZ) {
    return atan2(-accelX, sqrt(accelY * accelY + accelZ * accelZ)) * 180.0 / PI;
}

float getRoll(float accelX, float accelY, float accelZ) {
    return atan2(accelY, accelZ) * 180.0 / PI;
}

// --- Yaw Drift Correction Functions
void updateYawWithDriftCorrection(float gyroYaw, float dt) {
    // Detect if the gimbal is stationary
    float gyroMagnitude = abs(gyroYaw);
    unsigned long currentTime = millis();

    if (gyroMagnitude < STATIONARY_THRESHOLD) {
```

```
        if (!yawCorrection.isStationary) {
            // Just became stationary
            yawCorrection.stationaryStart = currentTime;
            yawCorrection.isStationary = true;
            Serial.println("Yaw: Stationary detected - starting drift correction");
        } else {
            // Has been stationary for a while
            unsigned long stationaryDuration = currentTime -
yawCorrection.stationaryStart;

            if (stationaryDuration > STATIONARY_TIME_MS) {
                // Update bias estimate using more aggressive learning when stationary
                float targetBias = gyroYaw;  // Assume this should be zero when
stationary
                float learningRate = (stationaryDuration > LONG_STATIONARY_TIME_MS) ?
0.02 : 0.005;
                yawCorrection.yawBias = (1.0 - learningRate) * yawCorrection.yawBias +
                                        learningRate * targetBias;

                // Constrain bias to reasonable limits
                yawCorrection.yawBias = constrainFloat(yawCorrection.yawBias, -
MAX_YAW_BIAS, MAX_YAW_BIAS);

                // For very long stationary periods, apply gentle decay towards 0
                if (stationaryDuration > LONG_STATIONARY_TIME_MS) {
                    yawCorrection.yawAngle *= YAW_DECAY_RATE;

                    // Debug output for long stationary correction
                    static unsigned long lastDecayMsg = 0;
                    if (currentTime - lastDecayMsg > 1000) {
                        Serial.print("Yaw: Long stationary decay - Angle: ");
                        Serial.print(yawCorrection.yawAngle, 2);
                        Serial.print("°, Bias: ");
                        Serial.println(yawCorrection.yawBias, 4);
                        lastDecayMsg = currentTime;
                    }
                }
            }

            // CRITICAL: When stationary, DON'T integrate gyro readings at all
            // This prevents further drift accumulation
            // yawCorrection.yawAngle stays the same (frozen)
        }
    } else {
        // Not stationary - normal integration with bias correction
        yawCorrection.isStationary = false;

        // Apply bias correction and integrate
```

```
        float correctedGyroYaw = gyroYaw - yawCorrection.yawBias;
        yawCorrection.yawAngle += correctedGyroYaw * dt;
        yawCorrection.yawAngle = wrapAngle(yawCorrection.yawAngle);
    }

    yawCorrection.lastGyroYaw = gyroYaw;
}

void resetYawAngle() {
    yawCorrection.yawAngle = 0;
    yawCorrection.yawBias = 0;  // Also reset bias estimate
    yawCorrection.isStationary = false;
    yawController.reset();  // Reset PID controller too
    Serial.print("Yaw reset - Bias was: ");
    Serial.println(yawCorrection.yawBias, 4);
}

void initializeKalmanWithCurrentPosition() {
    // Read current IMU data to get actual starting position
    accel.readSensor();  // Read accelerometer data

    // Calculate actual angles with calibration offsets applied
    float currentPitch = getPitch(accel.getAccelX_mss(), accel.getAccelY_mss(),
accel.getAccelZ_mss()) - pitchOffset;
    float currentRoll = getRoll(accel.getAccelX_mss(), accel.getAccelY_mss(),
accel.getAccelZ_mss()) - rollOffset;

    // Initialize Kalman filters with actual starting position
    kalmanX.angle = currentPitch;
    kalmanY.angle = currentRoll;

    Serial.print("Kalman initialized with actual position - Pitch: ");
    Serial.print(currentPitch, 2);
    Serial.print("°, Roll: ");
    Serial.print(currentRoll, 2);
    Serial.println("°");
}

// --- Audio Functions (Corrected Version)
void playStartupChime() {
    Serial.println("Configuring I2S for startup chime...");

    // 1. Configure I2S peripheral
    i2s_config_t i2s_config = {
        .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_TX),
        .sample_rate = 8000,
        .bits_per_sample = I2S_BITS_PER_SAMPLE_16BIT,
        .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
```

```
        .communication_format = I2S_COMM_FORMAT_STAND_I2S,
        .intr_alloc_flags = 0,
        .dma_buf_count = 8,
        .dma_buf_len = 64,
        .use_apll = false,
        .tx_desc_auto_clear = true
    };

    // 2. Install the I2S driver
    i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);

    // 3. Configure the I2S pins
    i2s_pin_config_t pin_config = {
        .bck_io_num = I2S_BCLK_PIN,
        .ws_io_num = I2S_LRC_PIN,
        .data_out_num = I2S_DIN_PIN,
        .data_in_num = I2S_PIN_NO_CHANGE
    };
    i2s_set_pin(I2S_NUM_0, &pin_config);

    // 4. Write the audio data
    Serial.println("Playing chime...");
    size_t bytes_written = 0;
    i2s_write(I2S_NUM_0, startup_chime, sizeof(startup_chime), &bytes_written,
portMAX_DELAY);

    // 5. *** THE CRITICAL FIX ***
    //    Wait for the DMA buffer to finish sending the sound before uninstalling the
driver.
    //    The sound is 113 samples at 8000Hz, so it's very short (~14ms). A 50ms delay
is plenty.
    delay(50);

    // 6. Uninstall the I2S driver to free up resources
    i2s_driver_uninstall(I2S_NUM_0);
    Serial.println("I2S driver uninstalled.");
}

// --- Setup
void setup() {
    Serial.begin(115200);
    delay(1000);
    Serial.println("=== AirOS Gimbal Controller Starting ===");

    // --- Load Settings
    loadSettings();

    // --- Initialize I2C & IMU
```

```
    Wire.begin();

    // Initialize accelerometer
    int status = accel.begin();
    if (status < 0) {
        Serial.println("BMI088 Accel Init Failed. Halting.");
        Serial.println(status);
        while(1);
    }

    // Initialize gyroscope
    status = gyro.begin();
    if (status < 0) {
        Serial.println("BMI088 Gyro Init Failed. Halting.");
        Serial.println(status);
        while(1);
    }

    // Configure accelerometer (200Hz, 4G range)
    accel.setOdr(Bmi088Accel::ODR_200HZ_BW_80HZ);
    accel.setRange(Bmi088Accel::RANGE_6G);  // Using 6G instead of 4G (closest
available)

    // Configure gyroscope (200Hz, 500 dps)
    gyro.setOdr(Bmi088Gyro::ODR_200HZ_BW_23HZ);
    gyro.setRange(Bmi088Gyro::RANGE_500DPS);

    Serial.println("BMI088 Initialized and Configured.");

    // --- Initialize and Play Startup Chime ---
    playStartupChime();
    // -----------------------------------------

    // --- Initialize Kalman Filters
    initKalman(&kalmanX);
    initKalman(&kalmanY);
    Serial.println("Kalman Filters Initialized.");

    // --- BLE Setup
    BLEDevice::init("AirOS Gimbal");
    BLEServer *pServer = BLEDevice::createServer();
    BLEService *pService = pServer->createService(SERVICE_UUID);

    // READ-ONLY Characteristics
    pPitchAngleCharacteristic = pService->createCharacteristic(PITCH_ANGLE_CHAR_UUID,
BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_NOTIFY);
    pRollAngleCharacteristic = pService->createCharacteristic(ROLL_ANGLE_CHAR_UUID,
BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_NOTIFY);
```

```
    pYawAngleCharacteristic = pService->createCharacteristic(YAW_ANGLE_CHAR_UUID,
BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_NOTIFY);
    pGimbalStatusCharacteristic = pService-
>createCharacteristic(GIMBAL_STATUS_CHAR_UUID, BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_NOTIFY);

    pPitchAngleCharacteristic->addDescriptor(new BLE2902());
    pRollAngleCharacteristic->addDescriptor(new BLE2902());
    pYawAngleCharacteristic->addDescriptor(new BLE2902());
    pGimbalStatusCharacteristic->addDescriptor(new BLE2902());

    // READ-WRITE Characteristics
    BLECharacteristic* pModeCharacteristic = pService-
>createCharacteristic(GIMBAL_MODE_CHAR_UUID, BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_WRITE);
    pModeCharacteristic->setCallbacks(new GimbalModeCallback());

    BLECharacteristic* pControlCharacteristic = pService-
>createCharacteristic(GIMBAL_CONTROL_CHAR_UUID, BLECharacteristic::PROPERTY_WRITE);
    pControlCharacteristic->setCallbacks(new GimbalControlCallback());

    // PID Gain Characteristics
    BLECharacteristic* pPitchPIDCharacteristic = pService-
>createCharacteristic(P_GAIN_CHAR_UUID, BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_WRITE);
    BLECharacteristic* pRollPIDCharacteristic = pService-
>createCharacteristic(I_GAIN_CHAR_UUID, BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_WRITE);
    BLECharacteristic* pYawPIDCharacteristic = pService-
>createCharacteristic(D_GAIN_CHAR_UUID, BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_WRITE);

    pPitchPIDCharacteristic->setCallbacks(new PIDGainCallback());
    pRollPIDCharacteristic->setCallbacks(new PIDGainCallback());
    pYawPIDCharacteristic->setCallbacks(new PIDGainCallback());

    pService->start();
    BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
    pAdvertising->addServiceUUID(SERVICE_UUID);
    pAdvertising->setScanResponse(true);
    BLEDevice::startAdvertising();

    Serial.println("🚀 ==================================");
    Serial.println("📡 BLE Server Started Successfully!");
    Serial.println("🔍 Device Name: 'AirOS Gimbal'");
    Serial.println("📶 Status: ADVERTISING & READY");
    Serial.println("📱 Waiting for iOS app connection...");
    Serial.println("==================================");
```

```
    // --- Initialize Servos
    pitchServo.attach(PITCH_SERVO_PIN);
    rollServo.attach(ROLL_SERVO_PIN);
    yawServo.attach(YAW_SERVO_PIN);

    // Center servos at startup
    pitchServo.write(90);
    rollServo.write(90);
    yawServo.write(90);
    Serial.println("Servos initialized and centered.");

    // --- Calibration (Always performed on startup)
    performCalibration();

    timer = micros();
    Serial.println("Setup Complete. Starting main loop.");
    Serial.println("Pitch\tRoll\tYaw\tRawP\tRawR\tRawY\tYBias\tStatus");
}

// --- Main Loop
void loop() {
    // --- Time Calculation with bounds checking
    unsigned long currentTime = micros();
    dt = (currentTime - timer) / 1000000.0;
    timer = currentTime;

    // Constrain dt to reasonable bounds
    dt = constrainFloat(dt, MIN_DT, MAX_DT);

    // --- Sensor Reading with error handling
    accel.readSensor();  // Read accelerometer data
    gyro.readSensor();   // Read gyroscope data

    float accelPitch = getPitch(accel.getAccelX_mss(), accel.getAccelY_mss(),
accel.getAccelZ_mss()) - pitchOffset;
    float accelRoll = getRoll(accel.getAccelX_mss(), accel.getAccelY_mss(),
accel.getAccelZ_mss()) - rollOffset;
    float gyroPitch = gyro.getGyroX_rads() * 180.0 / PI;  // Convert rad/s to deg/s
    float gyroRoll = gyro.getGyroY_rads() * 180.0 / PI;   // Convert rad/s to deg/s
    float gyroYaw = gyro.getGyroZ_rads() * 180.0 / PI - yawOffset;  // Convert rad/s
to deg/s

    // --- Filtering
    float filteredPitch = kalmanCalculate(&kalmanX, accelPitch, gyroPitch, dt);
    float filteredRoll = kalmanCalculate(&kalmanY, accelRoll, gyroRoll, dt);

    // Improved yaw calculation with drift compensation
```

```
    static float yawAngle = 0;
    updateYawWithDriftCorrection(gyroYaw, dt); // Apply drift correction
    yawAngle = yawCorrection.yawAngle; // Use the drift-corrected angle

    // --- State Machine (Gimbal Mode Logic)
    float pitchSetpoint = 0, rollSetpoint = 0, yawSetpoint = 0;

    switch (settings.mode) {
        case MODE_LOCKED:
            // Setpoints are 0, gimbal stays level
            pitchSetpoint = 0;
            rollSetpoint = 0;
            yawSetpoint = 0;
            break;

        case MODE_PAN_FOLLOW:
            // Pitch and Roll are locked, Yaw follows user input smoothly
            pitchSetpoint = 0;
            rollSetpoint = 0;
            // Smooth pan following - yaw setpoint moves toward target
            yawSetpoint += (yawSetpointTarget - yawSetpoint) * panFollowRate * dt;
            yawSetpoint = wrapAngle(yawSetpoint);
            break;

        case MODE_FPV:
            // All axes follow the camera/user movement
            pitchSetpoint = filteredPitch;
            rollSetpoint = filteredRoll;
            yawSetpoint = yawAngle;
            break;

        case MODE_PERSON_TRACKING:
            // Check if we have recent tracking data
            if (millis() - lastPersonTrackingUpdate < PERSON_TRACKING_TIMEOUT_MS) {
                // Use person tracking speeds to adjust setpoints
                // Convert tracking speeds to angle adjustments
                static float personTrackingYawTarget = 0;
                static float personTrackingPitchTarget = 0;

                // Integrate tracking speeds over time
                float deltaTimeSeconds = dt;
                personTrackingYawTarget += personTrackingPanSpeed * deltaTimeSeconds;
                personTrackingPitchTarget += personTrackingTiltSpeed *
deltaTimeSeconds;

                // Clamp targets to reasonable limits
                personTrackingYawTarget = constrainFloat(personTrackingYawTarget, -90,
90);
```

```
                personTrackingPitchTarget = constrainFloat(personTrackingPitchTarget,
-45, 45);

                // Set gimbal targets
                pitchSetpoint = personTrackingPitchTarget;
                rollSetpoint = 0;  // Keep roll level
                yawSetpoint = personTrackingYawTarget;

                // Wrap yaw angle
                yawSetpoint = wrapAngle(yawSetpoint);
            } else {
                // No recent tracking data - fall back to locked mode
                Serial.println("⚠️ Person tracking timeout - switching to locked
mode");
                settings.mode = MODE_LOCKED;
                pitchSetpoint = 0;
                rollSetpoint = 0;
                yawSetpoint = 0;
            }
            break;

        case MODE_INACTIVE:
        default:
            // Reset PIDs and do nothing
            pitchController.reset();
            rollController.reset();
            yawController.reset();
            pitchSetpoint = filteredPitch;  // Don't fight current position
            rollSetpoint = filteredRoll;
            yawSetpoint = yawAngle;
            break;
    }

    // --- PID Calculation
    float pitchOutput = 0, rollOutput = 0, yawOutput = 0;

    if (settings.mode != MODE_INACTIVE) {
        pitchOutput = calculatePID(pitchSetpoint, filteredPitch, pitchController,
settings.pitchPID);
        rollOutput = calculatePID(rollSetpoint, filteredRoll, rollController,
settings.rollPID);
        yawOutput = calculatePID(yawSetpoint, yawAngle, yawController,
settings.yawPID);
    }

    // --- Servo Control
    if (settings.mode != MODE_INACTIVE) {
        int pitchServoAngle = 90 + (int)pitchOutput;
```

```
        int rollServoAngle = 90 + (int)rollOutput;
        int yawServoAngle = 90 + (int)yawOutput;

        pitchServo.write(constrain(pitchServoAngle, 0, 180));
        rollServo.write(constrain(rollServoAngle, 0, 180));
        yawServo.write(constrain(yawServoAngle, 0, 180));
    } else {
        // In inactive mode, center all servos
        pitchServo.write(90);
        rollServo.write(90);
        yawServo.write(90);
    }

    // --- BLE Notifications
    static unsigned long lastNotify = 0;
    static unsigned long lastDebug = 0;
    if (millis() - lastNotify > 100) { // Update app 10 times per second
        // Send data to iOS app
        pPitchAngleCharacteristic->setValue(filteredPitch);
        pPitchAngleCharacteristic->notify();
        pRollAngleCharacteristic->setValue(filteredRoll);
        pRollAngleCharacteristic->notify();
        pYawAngleCharacteristic->setValue(yawAngle);
        pYawAngleCharacteristic->notify();

        uint8_t statusVal = (uint8_t)settings.mode;
        pGimbalStatusCharacteristic->setValue(&statusVal, 1);
        pGimbalStatusCharacteristic->notify();

        // Debug: BLE communication status every 2 seconds
        if (millis() - lastDebug > 2000) {
            Serial.println("=== BLE DEBUG ===");
            Serial.println("📡 SENDING to iOS App:");
            Serial.print("  Pitch: "); Serial.print(filteredPitch, 2);
Serial.println("°");
            Serial.print("  Roll: "); Serial.print(filteredRoll, 2);
Serial.println("°");
            Serial.print("  Yaw: "); Serial.print(yawAngle, 2); Serial.println("°");
            Serial.print("  Mode: "); Serial.print(statusVal); Serial.print(" (");
            switch(settings.mode) {
                case MODE_INACTIVE: Serial.print("INACTIVE"); break;
                case MODE_LOCKED: Serial.print("LOCKED"); break;
                case MODE_PAN_FOLLOW: Serial.print("PAN_FOLLOW"); break;
                case MODE_FPV: Serial.print("FPV"); break;
                case MODE_PERSON_TRACKING: Serial.print("PERSON_TRACKING"); break;
            }
            Serial.println(")");
            Serial.println("================");
```

```
            lastDebug = millis();
        }

        // Serial monitoring output (compact format)
        Serial.print("P:"); Serial.print(filteredPitch, 1);
        Serial.print(" R:"); Serial.print(filteredRoll, 1);
        Serial.print(" Y:"); Serial.print(yawAngle, 1);
        Serial.print(" M:"); Serial.print(statusVal);
        Serial.print(" "); Serial.println(yawCorrection.isStationary ? "STAT" :
"MOVE");

        lastNotify = millis();
    }

    delay(2); // Small delay to prevent watchdog issues
    yield();
}

// --- Settings Management Functions
void applyDefaultSettings() {
    settings.mode = MODE_LOCKED;
    settings.pitchPID = {1.2, 0.1, 0.05};  // More conservative defaults
    settings.rollPID = {1.2, 0.1, 0.05};
    settings.yawPID = {0.8, 0.05, 0.02};
    settings.kalman_q_angle = 0.001;
    settings.kalman_q_bias = 0.003;
    settings.kalman_r_measure = 0.03;
    Serial.println("Applied default settings.");
}

void saveSettings() {
    prefs.begin("gimbal_settings", false);
    prefs.putBytes("settings", &settings, sizeof(GimbalSettings));
    prefs.end();
    Serial.println("Gimbal settings saved.");
}

void loadSettings() {
    prefs.begin("gimbal_settings", true);
    size_t expectedSize = sizeof(GimbalSettings);
    size_t actualSize = prefs.getBytesLength("settings");

    bool settingsFound = false;
    if (actualSize == expectedSize) {
        settingsFound = prefs.getBytes("settings", &settings, expectedSize) ==
expectedSize;
    }
    prefs.end();
```

```
    if (!settingsFound) {
        Serial.println("No saved settings found, applying defaults.");
        applyDefaultSettings();
        saveSettings();
    } else {
        Serial.println("Loaded saved settings.");
    }
}

void performCalibration() {
    Serial.println("Starting new calibration...");

    uint8_t statusVal = (uint8_t)STATUS_CALIBRATING;
    pGimbalStatusCharacteristic->setValue(&statusVal, 1);
    pGimbalStatusCharacteristic->notify();

    // Reset PID controllers during calibration
    pitchController.reset();
    rollController.reset();
    yawController.reset();

    // Reset yaw drift correction system
    yawCorrection.reset();

    float pitchSum = 0, rollSum = 0, yawSum = 0;
    int validSamples = 0;

    for (int i = 0; i < 500; i++) {
        accel.readSensor();  // Read accelerometer data
        gyro.readSensor();   // Read gyroscope data

        pitchSum += getPitch(accel.getAccelX_mss(), accel.getAccelY_mss(),
accel.getAccelZ_mss());
        rollSum += getRoll(accel.getAccelX_mss(), accel.getAccelY_mss(),
accel.getAccelZ_mss());
        yawSum += gyro.getGyroZ_rads() * 180.0 / PI;  // Convert rad/s to deg/s
        validSamples++;

        delay(5);
    }

    if (validSamples > 0) {
        pitchOffset = pitchSum / validSamples;
        rollOffset = rollSum / validSamples;
        yawOffset = yawSum / validSamples;

        // Initialize yaw bias with the calibrated offset
```

```
        yawCorrection.yawBias = yawOffset;

        isCalibrated = true;
        Serial.println("Calibration complete.");
        Serial.print("Offsets - Pitch: "); Serial.print(pitchOffset);
        Serial.print(", Roll: "); Serial.print(rollOffset);
        Serial.print(", Yaw: "); Serial.println(yawOffset);
        Serial.print("Initial Yaw Bias: "); Serial.println(yawCorrection.yawBias, 4);

        // Initialize Kalman filters with actual starting position
        delay(10); // Brief delay to ensure IMU is ready
        initializeKalmanWithCurrentPosition();
    } else {
        Serial.println("Calibration failed - no valid samples");
    }
}
```

AirLink.xcodeproj

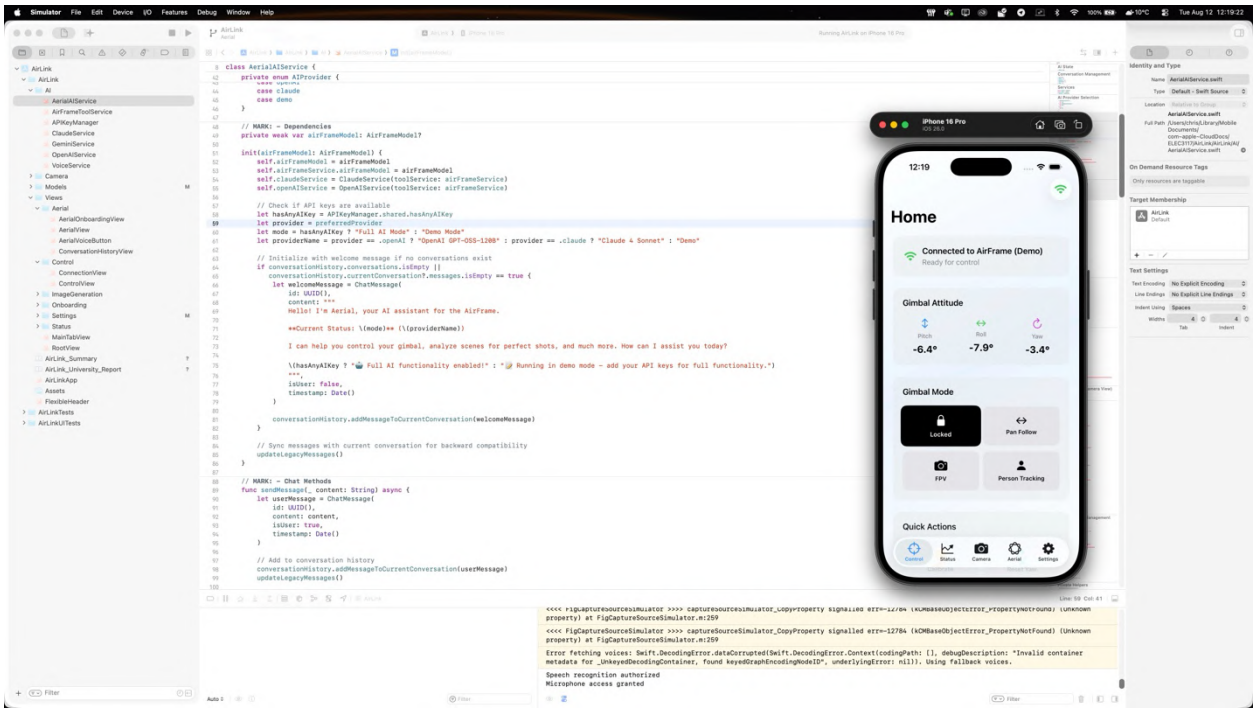


Figure A46. Developing AirLink with Xcode 26 and simulating it on a device running iOS 26 beta 5.

Please review AirLink on Chris' GitHub.

G: NEEDS ASSESSMENT

The primary target audience for this product consists of content creators, vloggers, and hobbyist photographers/videographers. These users are tech-savvy and passionate about producing high-quality visual content but may not have the budget or technical expertise for professional-grade cinematic equipment. A secondary audience includes travel enthusiasts and families who want to capture precious memories with sharp photos and smooth videos without being burdened by complicated or heavy gear.

Our target audience needs this product because they consistently face challenges that compromise the quality of their photos and videos. With the rise of the "creator economy" and the increasing standard of 4K video, shaky or unlevel footage is no longer acceptable. They are frustrated when a perfect shot is ruined by environmental factors or cumbersome equipment. They need a reliable way to achieve professional-looking stability to elevate their content, enhance their creative potential, and save time during setup and shooting.

The product is designed for use in challenging, real-world locations where perfect surfaces are rare (from rocky hiking trails and windy beaches to busy social events). It will be used to capture a variety of content, including static long-exposure photos, quick portraits, and smooth, dynamic video shots that require both stability and controlled movement.

To address their needs, users require a product with the following capabilities:

Automatic Leveling: The ability to achieve a perfectly level shot automatically, without the user needing to manually adjust for uneven ground.

Vibration Resistance: The capability to actively reduce the effect of minor shakes from sources like wind or footsteps.

All-Terrain Stability: A base that provides a secure footing on different surfaces.

Universal Device Compatibility: A secure and easy-to-use mounting system for various cameras and phones.

Portability and Rapid Deployment: The product must be lightweight, compact, and fast to set up.

Sustained Power: An integrated power source with enough battery life for a day of shooting.

Full Range of Motion: A head that allows for smooth, fluid panning and tilting.

Existing tripods have limitations and issues with:

a) **Manual Setup** (it can be time consuming).

b) **Static Nature** (Designed for static shots, they have issues being still during Dynamic motion).

c) **Trade-off between Stability vs Portability vs Cost** (can only pick two of three, light, stable, cheap).

d) **Vulnerable to Internal and External Forces** (like Wind, Uneven Ground, Vibrations, Accidental Bumps).

e) **Tedious adjustments to positioning of device** (adjustments to base, position, angle, can be cumbersome and time-consuming).

H: REPORT CONTRIBUTION BREAKDOWN

Table 18: Report Contribution Breakdown

Section	Contributor
Executive Summary	Chris Pagolu, Ahmad Rana
2. Project Description	Ahmad Rana
3. Product Overview	Ahmad Rana
4. Detailed Design	Chris Pagolu
5. Performance Analysis	Chris Pagolu
6. Business Plan	Ahmad Rana
7. Development Plan	Chris Pagolu
8. Conclusion	Chris Pagolu, Ahmad Rana
Appendix A	Chris Pagolu
Appendix B	Chris Pagolu
Appendix C	Chris Pagolu
Appendix D	Chris Pagolu
Appendix E	Chris Pagolu
Appendix F	Chris Pagolu

I: PROJECT CONTRIBUTION BREAKDOWN

Table 19: Project Contribution Breakdown

Area	Contributor
Market Research	Chris Pagolu
Needs Assessment	Chris Pagolu, Ahmad Rana
Risk Assessment	Chris Pagolu
Circuit Design	Chris Pagolu
Component Selection	Chris Pagolu, Ahmad Rana
Project Ideation	Chris Pagolu, Ahmad Rana
3D Modelling	Ahmad Rana
PCB Design	Chris Pagolu
Software (AirOS + AirLink)	Chris Pagolu
Assembly	Chris Pagolu, Ahmad Rana
Presentation	Chris Pagolu, Ahmad Rana