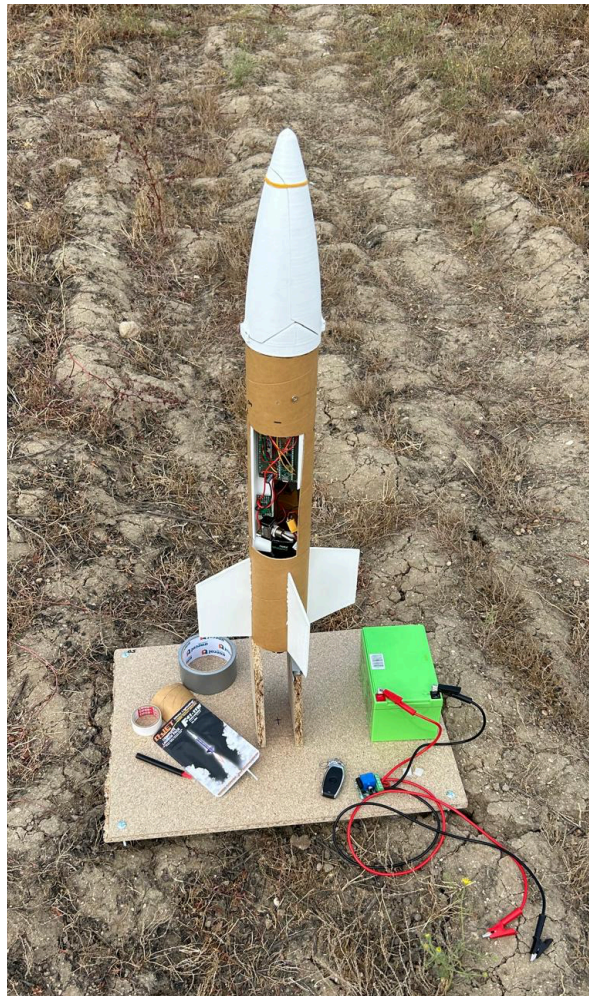


AETHER Rocket

Design Report

Design, Analysis & Flight Test



Ignacio Uriel Adrada

University of Michigan — Ann Arbor

nachouri@umich.edu

May 2026 – August 2026

Contents

- 1 Executive Summary
 - 1.1 System Overview
 - 1.2 Key Design Discriminators
- 2 References
- 3 Design Layout and Trade Studies
 - 3.1 Mission Requirements and Design Objectives
 - 3.2 Motor Selection Trade Study
 - 3.3 Risk and Risk Mitigation
 - 3.4 Overall Design Layout
- 4 Analysis
 - 4.1 Analysis Techniques
 - 4.1.1 Analytical Tools
 - 4.1.2 Simulation Models
 - 4.2 Flight Performance Analysis
 - 4.2.1 Stability Analysis (CG, CP, Static Margin)
 - 4.2.2 Trajectory Simulation
- 5 Electronics and Avionics
 - 5.1 Flight Computer Selection
 - 5.2 IMU Selection
 - 5.3 Barometer Selection
 - 5.4 Servo Selection
 - 5.5 SD card selection
 - 5.6 Power Architecture
 - 5.7 Wiring Diagram
 - 5.8 E-Match Circuit Igniter
 - 5.9 Launch Pad
- 6 Test and Integration
 - 6.1 Flight Test Results
 - 6.1.1 Flight 1
 - 6.1.2 Flight 2
- 7 Manufacturing
 - 7.1 Built Components
 - 7.2 Mass Budget
 - 7.3 Materials and components bought
- 8 Conclusion
- Appendix A — Flight Data Logs
- Appendix B — Code
 - B.1 FlightComputer_ApogeeDeploy.ino
 - B.2 ServoDropTest.ino (bench test)
 - B.3 rocket_sim.py (1D RK4 trajectory simulation)

List of Figures

Figure 1: OpenRocket model profile of the rocket
 Figure 2: CAD isometric view of the assembled rocket (open view inside airframe)
 Figure 3: Nose-cone split halves (open) — CAD
 Figure 4: Nose-cone deployment mechanism (open) — CAD
 Figure 5: Parachute bay + spring for deployment
 Figure 6: Lower Nose Cone half
 Figure 7: Metal U-Bracket
 Figure 8: Custom printed connectors
 Figure 9: Avionics sled (electronics tray) CAD
 Figure 10: Avionics sled (electronics tray) installed in rocket
 Figure 11: Main body phenolic tube CAD
 Figure 12: Fin CAD
 Figure 13: Motor-mount + adapter CAD
 Figure 14: Motor-mount assembly (exploded) CAD
 Figure 15: 1D Python simulation — altitude and velocity vs. time
 Figure 16: Static stability margin at liftoff: 1.19 calibers. At burnout: 1.45 calibers. Falling just short of the 1.5-cal target and lowest at liftoff
 Figure 17: F41W motor simulation of altitude vs time
 Figure 18: E12 motor simulation of altitude vs time
 Figure 19: F15 motor simulation of altitude vs time
 Figure 20: Teensy 4.1 Microcontroller
 Figure 21: MPU-6050 IMU
 Figure 22: MG90s micro servo
 Figure 23: Micro SD card used for data logging
 Figure 24: Wiring Diagram of Aether Avionics
 Figure 25: Wiring block diagram (recovery servos omitted)
 Figure 26: E-match circuit
 Figure 27: QIACHIP 433Mhz Wireless Remote Control Switch
 Figure 28: Launch Pad
 Figure 29: Flight 1 (LOG030) altitude vs time
 Figure 30: Flight 1 (LOG030) velocity vs time
 Figure 31: Flight 1 (LOG030) acceleration vs time
 Figure 32: Flight 1 (LOG030) body rates — pitch, roll and yaw vs. time
 Figure 33: Flight 2 (LOG045) altitude vs time
 Figure 34: Flight 2 (LOG045) velocity vs time

List of Tables

Table 1: Mission Requirements and Design Objectives
 Table 2: Motor Selection Trade Study
 Table 3: Risk and Risk Mitigation
 Table 4: Analytical Tools
 Table 5: Flight Computer Selection
 Table 6: IMU Selection
 Table 7: Barometer Selection
 Table 8: Servo Selection
 Table 9: Power Architecture
 Table 10: Flight performance summary (Flight 1 vs. Flight 2)
 Table 11: Built Components
 Table 12: Mass Budget

List of Acronyms

DOF Degrees of Freedom

IMU Inertial Measurement Unit

PID Proportional-Integral-Derivative

CG Center of Gravity

CP Center of Pressure

FDM Fused Deposition Modeling

PETG Polyethylene Terephthalate Glycol

PWM Pulse Width Modulation

I2C Inter-Integrated Circuit

SD Secure Digital (memory card)

CAD Computer-Aided Design

FEA Finite Element Analysis

T/W Thrust-to-Weight Ratio

NAR National Association of Rocketry

TRA Tripoli Rocketry Association

BOM Bill of Materials

LiPo Lithium-Ion Polymer Battery

1 Executive Summary

This report documents the design, analysis, fabrication, and flight testing of a rocket built as an independent summer engineering project. This document records every major design decision, trade study, analysis, and test result to provide a complete technical record of the project.

1.1 System Overview

AETHER is a single-stage, fin-stabilized rocket approximately 660 mm long, built around a 76 mm (3 in) phenolic airframe, with a pre-flight mass of around 930g. Primary flights used the Quest Q-Jet F41 motor (24 mm composite, ~46 N·s total impulse). The flight computer is a Teensy 4.1 that logs an MPU-6050 IMU and a BMP-388 barometer to microSD at ~100 Hz. Recovery is by a servo-actuated split nose cone that releases a parachute at barometrically detected apogee.

Across two flights the vehicle demonstrated powered flight, on-board data logging, and autonomous apogee-triggered deployment. Peak measured altitude was 54 m on Flight 1 (LOG030) against a ~95 m OpenRocket prediction; the shortfall is attributable to off-nominal, non-vertical flight caused primarily by the absence of a launch rail.

1.2 Key Design Discriminators

The design is distinguished by the following choices:

- **Teensy 4.1 flight computer** A 600 MHz Cortex-M7 with hardware floating point and an on-board microSD slot gives ample headroom for high-rate logging with no external SD interface needed.
- **Electronic (servo) parachute deployment.** A split nose cone driven by two MG90S servos releases the chute on a barometric apogee trigger, giving delay-independent deployment rather than relying on the motor ejection charge.
- **Barometric apogee detection with a boost lockout.** Deployment arms only after a set climb and is inhibited for a fixed coast time after launch, so boost-phase noise cannot cause an early deployment.
- **High-rate, self-documenting logging.** All sensor channels and flight-state flags are written to a CSV file at ~100 Hz. Data is later extracted to create graphs, data tables, etc.

2 References

- [1] OpenRocket. Open-source rocket simulation software. <https://openrocket.info>
- [2] Matweb. Material property data — PETG, 6061 aluminum. <https://www.matweb.com>
- [3] ThrustCurve.org. Motor performance data — Estes E12-0 and F15-0. <https://www.thrustcurve.org>
- [4] Teensy 4.1 Technical Reference. PJRC. <https://www.pjrc.com/teensy>
- [5] InvenSense. MPU-6050 Product Specification Rev 3.4.
- [6] NAR Model Rocket Safety Code. National Association of Rocketry. <https://www.nar.org>
- [7] Barrowman, J. S. The Theoretical Prediction of the Center of Pressure. 1967.
- [9] How I built a TVC Rocket. Nicola Gaiani. 2025. <https://www.youtube.com/watch?v=Ppn7o03hIPE>

3 Design Layout and Trade Studies

3.1 Mission Requirements and Design Objectives

The following requirements govern the design:

Table 1: Mission Requirements and Design Objectives

Requirement	Value	Rationale
Motor class	E–F (NFPA 1122 compliant)	No certification required; available commercially
Data logging	≥ 100 Hz to SD card	Required to analyze flight performance post-flight
Recovery system	Streamer, ≤ 5 m/s descent	Safety; prevents lawn-dart scenario
Budget	$\leq \$300$	Self-funded project constraint

3.2 Motor Selection Trade Study

Table 2: Motor Selection Trade Study

Motor	Class	Total Impulse (Ns)	Burn Time (s)	Peak Thrust (N)	Avg Thrust (N)	Notes
Estes E12-0	E	14.1	~1.2	~19	12	Short burn: less time to demonstrate control. Low apogee. Cheaper.
Estes F15-0	F	29.7	~2.0	~22	15	Longer burn, more control time, similar projects found online using F15-0
Q-Jet F41-11W	F	47.7	1.1	54.2	42.2	Not a lot of data found online. One of only available options to ship internationally.

Motor F41-11W was chosen as it was the only motor available to be shipped internationally to Spain, where most of the project (including assembling the whole rocket) took place.

3.3 Risk and Risk Mitigation

Table 3: Risk and Risk Mitigation

Risk	Consequence	Likelihood	Mitigation
Aerodynamic Instability (CG too far ahead or back)	Rocket pitches over, low apogee.	Medium	Ensure stability caliber is appropriate both in simulation and when built is complete.
Recovery fails to deploy	Several damage to rocket airframe and electronics	Medium	Include several safety deployment mechanisms in the code.
IMU noise too high in flight	PID diverges; unstable flight	Medium	Kalman filter; vibration-isolated mounting; bench characterization
Causing a wildfire when launching	Legal actions might be pursued	Medium	Find a safe launch site and ensure all legal standards are met.
Rocket is overweight	T/W < 4; marginal liftoff	Medium	Track mass weekly in Phase 1; budget 300g dry
Epoxy joint fails on fin	Fin separation; unstable	Low	30-min epoxy with fillets; post-cure proof test
Ejection charge damages electronics bay	Several damage to flight computer	Low	Design a motor mount that protects the electronics

3.4 Overall Design Layout

The vehicle is a single-stage rocket built around a $\approx 76\text{mm}$ -diameter airframe with an overall length of approximately 660mm. The design is organized into five major subassemblies, arranged nose-to-tail: the **nose cone / recovery module**, the **electronics (avionics) bay**, the **main body tube**, the **fins** and the **motor mount**.

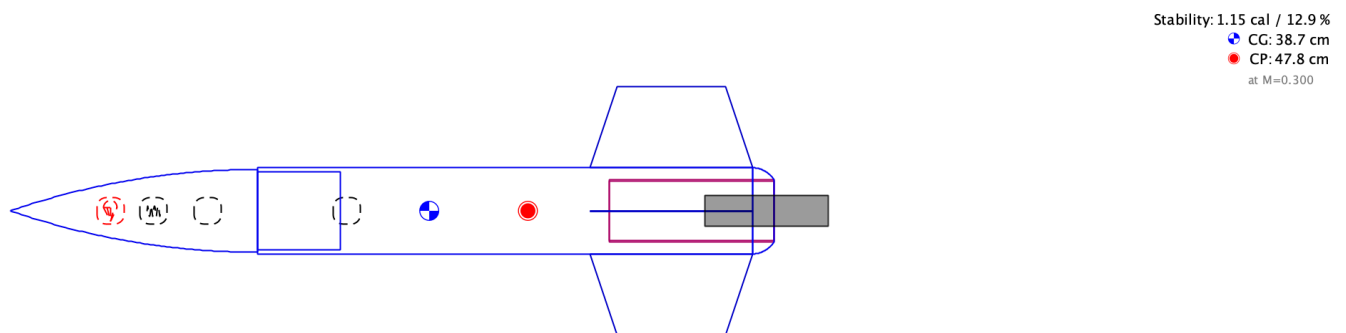


Figure 1: OpenRocket model profile of the rocket

The OpenRocket Assembly shows a stability margin of 1.15 cal, a little below the 1.5 objective but nevertheless in between the 1.0-2.0 range that is considered safe.

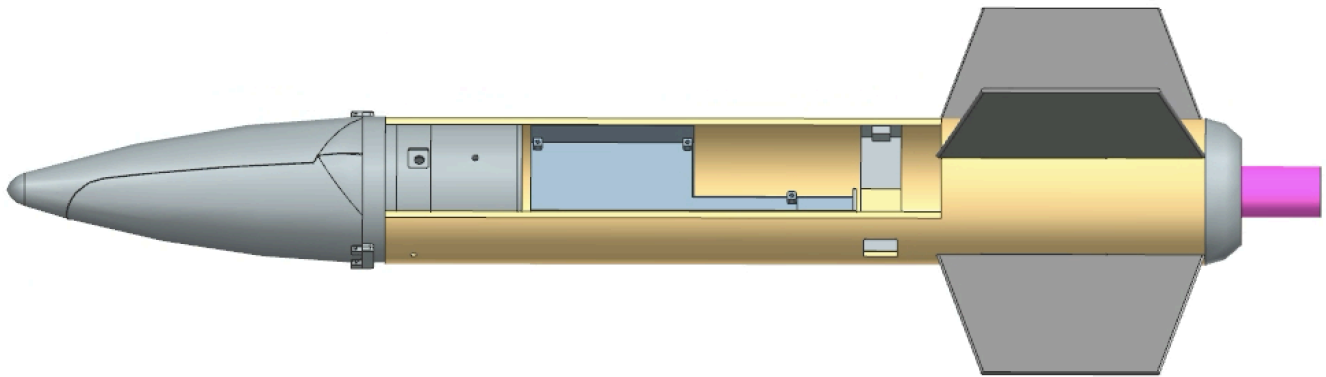


Figure 2: CAD isometric view of the assembled rocket (open view inside airframe)

The **nose cone and recovery module** is a 3D-printed ogive that splits into two hinged halves driven by 2 MG90s servos. The parachute is packed inside and sits on a parachute bay that pushes it out with the help of a small spring. At apogee the flight computer commands the servos to open the halves and release the chute. The nose interfaces to the airframe through a shoulder that seats into the forward body tube and is held closed by the servos until deployment. The parachute is connected to a shock cord that is tied to a metal u-bracket secured with two nuts to a hollow 3D printed part as shown in figure 9.

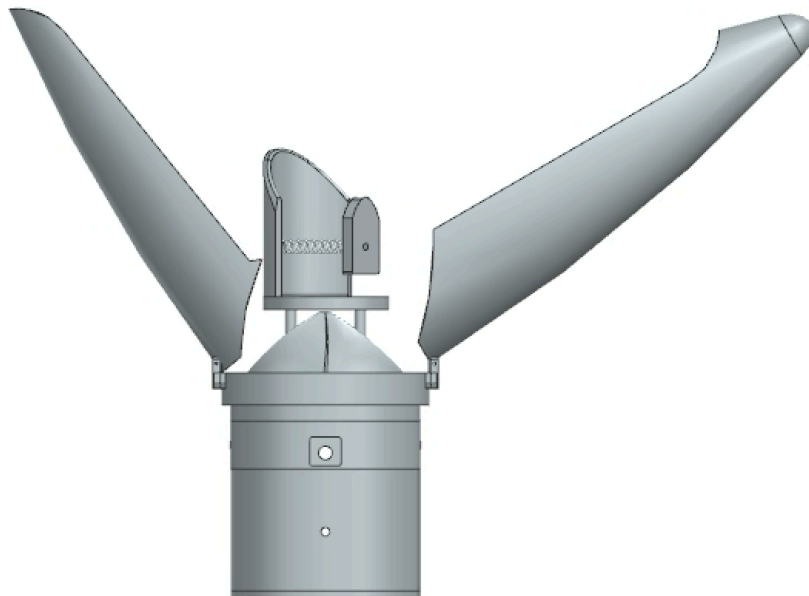


Figure 3: Nose-cone split halves (open) — CAD



Figure 4: Nose-cone deployment mechanism (open) — CAD

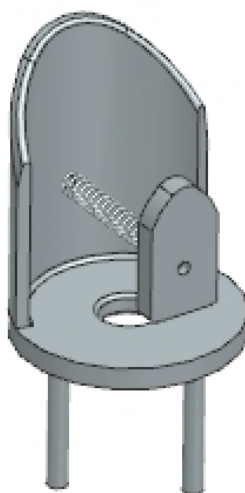


Figure 5: Parachute bay + spring for deployment

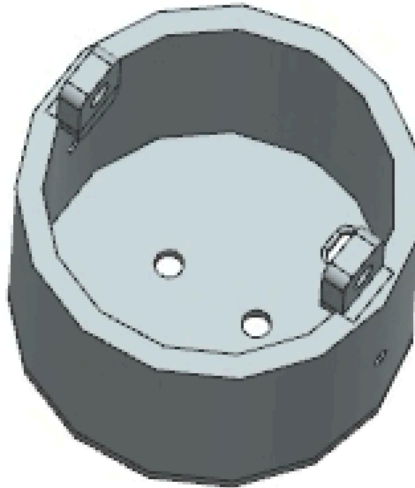


Figure 6: Lower Nose Cone half

Secured to airframe via 2 screws and connected to the upper nose cone half via custom made connectors (figure 10). Holes on the bottom for servo cables and attached to a U-Metal bracket (figure 11) that the shock cord is tied to.



Figure 7: Metal U-Bracket

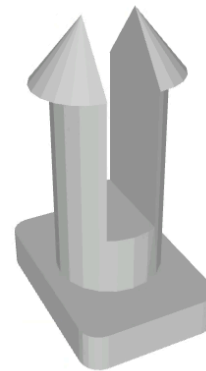


Figure 8: Custom printed connectors

Immediately aft is the **avionics bay**, which carries the Teensy 4.1 flight computer, the BMP388 barometer, MPU-6050 IMU, the microSD logging, the 7.4 V LiPo battery with its UBEC/BEC regulation and a small toggle switch that is used to turn on the flight computer. It is mounted on a removable sled secured to the airframe using 4 M3 screws and vented with static ports so the barometer reads true ambient pressure. A small hole was carved in the airframe for ease of access and removal of the battery right where the battery holder is. The bay interfaces upward to the nose-cone servos (deployment wiring) and to the recovery harness.

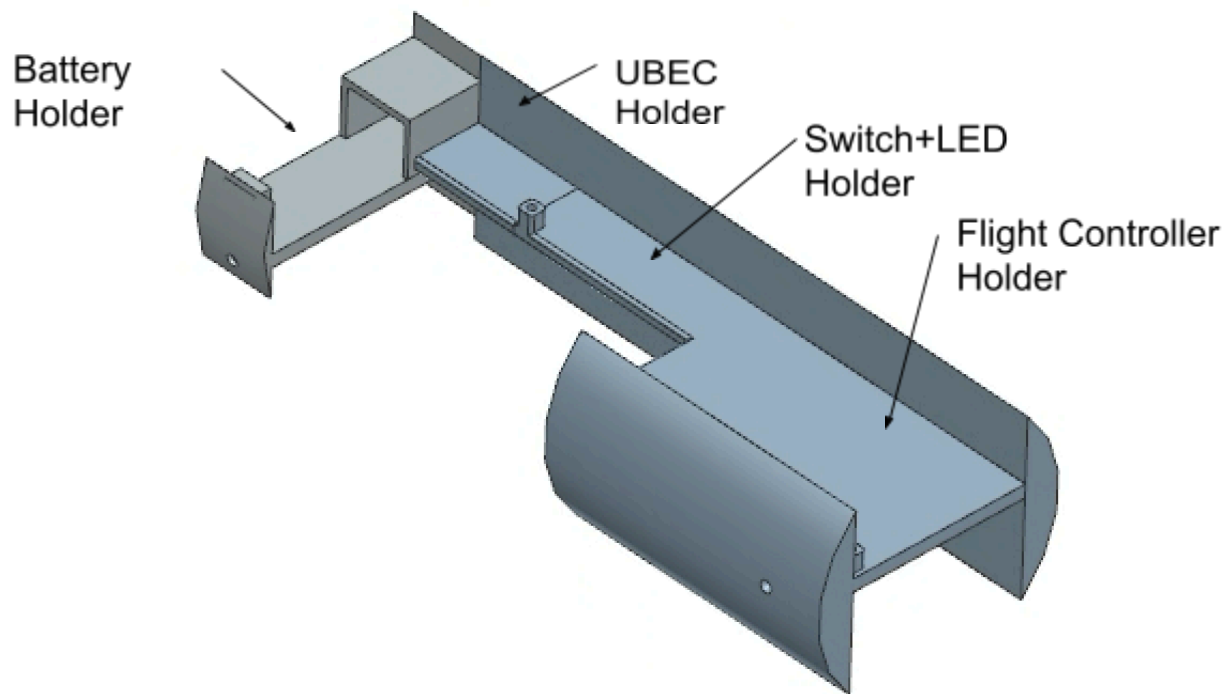


Figure 9: Avionics sled (electronics tray) CAD

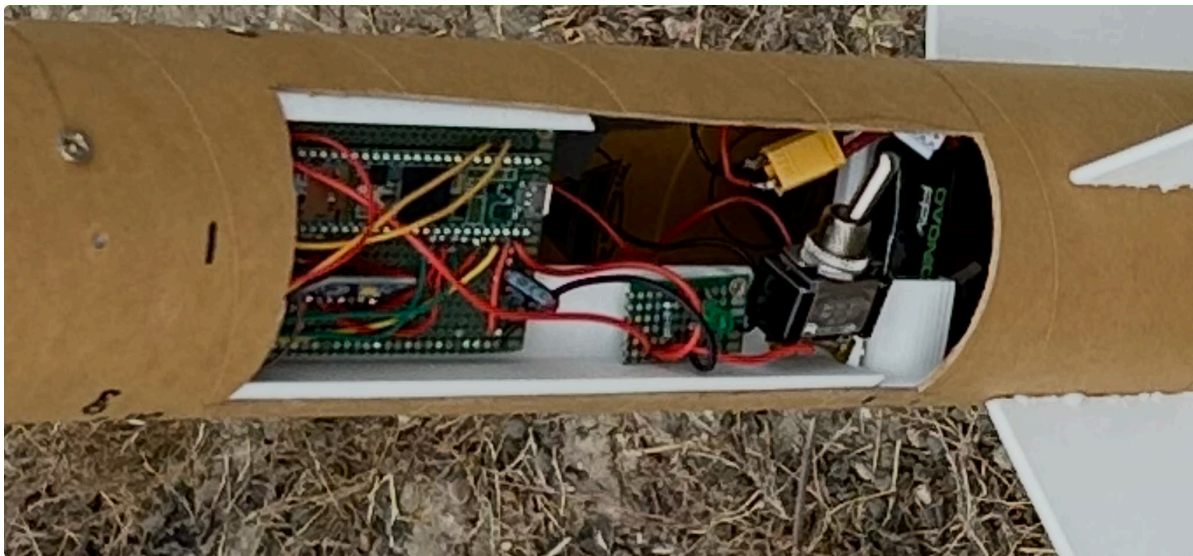


Figure 10: Avionics sled (electronics tray) installed in rocket

The **main body tube** is the primary structure, carrying aerodynamic and recovery loads between the nose and aft sections. It is made out of phenol primarily and has a length of 457.2 mm (18"). It has several screw holes to mount various components to it as well as 4 holes for the fins and a hole for ease of access and removal of the battery. All holes were made with a dremel and taking the necessary safety precautions, safety glasses, respirator, etc.



Figure 11: Main body phenolic tube CAD

The **fins** provides attitude stability. 4 fixed fins passively stabilize the vehicle by placing the center of pressure aft of the center of gravity. The dimensions are 15cm root chord, 10cm tip chord, 7.5cm height, 2.5cm sweep and 18° sweep angle, chosen to maximized apogee in the OpenRocket Simulation



Figure 12: Fin CAD

At the tail, the **motor mount** is a 24 mm tube retaining the Quest F41 motor and transferring thrust into the airframe through a thrust bulkhead attached to the airframe via 2 screws. The motor was coated in a thin layer of heat resistant tape for extra protection of the motor mount, which was printed using PETG plastic instead of PLA since it has a higher melting point.

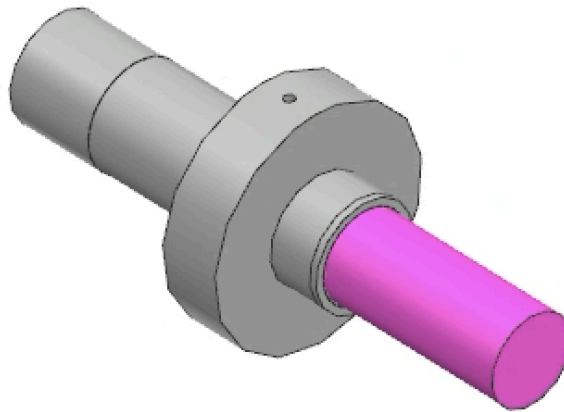


Figure 13: Motor-mount + adapter CAD

The motor mount assembly has 2 components. When the ejection charge goes off, the motor along with its adapter piece slide off the motor mount to avoid damaging it or the flight computer / avionics.

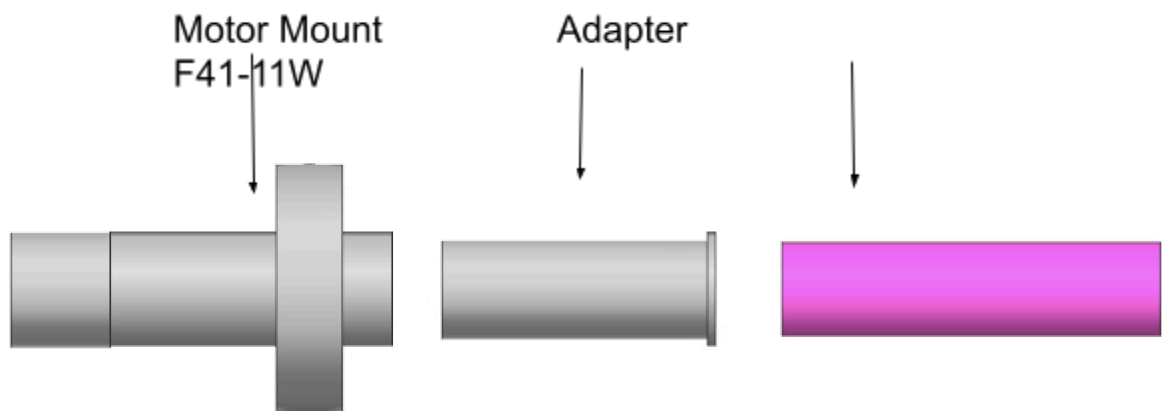


Figure 14: Motor-mount assembly (exploded) CAD

4 Analysis

4.1 Analysis Techniques

4.1.1 Analytical Tools

The following analysis tools were used throughout the design process:

Table 4: Analytical Tools

Tool	Purpose	Section
OpenRocket	Stability analysis, CG/CP prediction, trajectory simulation	3.4, 4.2.1, 4.2.2
Python	1D trajectory simulation, post-flight log parsing and plotting	4.1.2, 6.1.1, 6.1.2
NX Siemens	3D CAD, assembly, mass properties	3.4
Excel / Google Sheets	Budget, data logging	7.3, Appendix A
Arduino IDE	Write code for flight computer	Appendix B
EasyEDA	Wiring Diagrams and power architecture	5.6, 5.7

4.1.2 Simulation Models

The following is data, numerical and visual, of a python simulation treating the rocket as a 1D point-mass modeling thrust, drag and gravity. Thrust data imported from Quest F41W thrust curve available online. RK4 (classical 4th-order Runge-Kutta) numerical integration method was used with a fixed time step of $dt = 0.005$ s (5 ms).

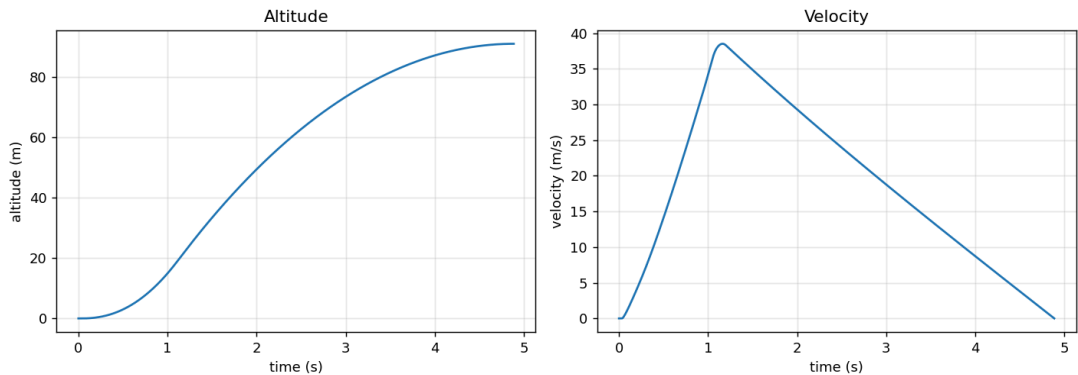


Figure 15: 1D Python simulation — altitude and velocity vs. time

Inputs:

Motor F41 data: Total impulse 46.3 N·s, Burn 1.20 s, Thrust Curve.
Coefficient of drag: 0.55 (arbitrary)
Liftoff mass: around 930 g
Airframe diameter: 0.076m

Outputs

Thrust/weight (peak): 5.7
Rail-exit velocity: 9.8 m/s (assuming 1.5m rail length).
Max velocity: 38.3 m/s
Apogee (ideal): 88.3 m at $t = 4.80$ s

4.2 Flight Performance Analysis

4.2.1 Stability Analysis (CG, CP, Static Margin)

The vehicle's static stability was assessed in OpenRocket by tracking the centre of gravity (CG) and centre of pressure (CP) throughout the flight. Static stability is quantified by the static margin, which is the CG-to-CP separation expressed in body calibers, where one caliber = one airframe diameter, here 76 mm, computed as $(CP_x - CG_x)/diameter$. A margin of at least 1.0 caliber is required for the fins to keep the vehicle pointed into the airflow, with 1.5–2.0 calibers considered good practice for a hobby rocket.

Figure 16 shows the CG and CP locations and the resulting static margin versus time. As propellant burns off the aft end, the CG migrates forward while the CP stays roughly fixed, so the margin grows over the burn: it is 1.19 calibers at liftoff, rising to 1.45 calibers at burnout. The margin stays above the 1.0-caliber minimum for the whole ascent, so the vehicle is statically stable, but only marginally. It never reaches the 1.5-caliber target and, critically, it is lowest at liftoff, exactly when the vehicle is slowest and the fins have the least aerodynamic authority.

This marginal liftoff stability is the most likely root cause of the off-nominal flights. Combined with the low rail-exit velocity (Section 4.1.2) and the absence of a launch rail (Section 5.9), a 1.19-caliber margin leaves little authority to correct an initial disturbance before the vehicle has built up speed. The flight data support this: both flights show a sustained pitch rate of roughly 150–190 °/s from the instant of liftoff (Section 6), the signature of a vehicle arcing over almost immediately rather than recovering to vertical.

To reach a robust ≥ 1.5 -caliber margin in the next iteration, the CG should be moved forward with nose ballast and/or the fin area increased to move the CP aft; the required ballast can be sized directly from the OpenRocket CG/CP values. Together with a proper launch rail, this is expected to eliminate the early pitch-over.

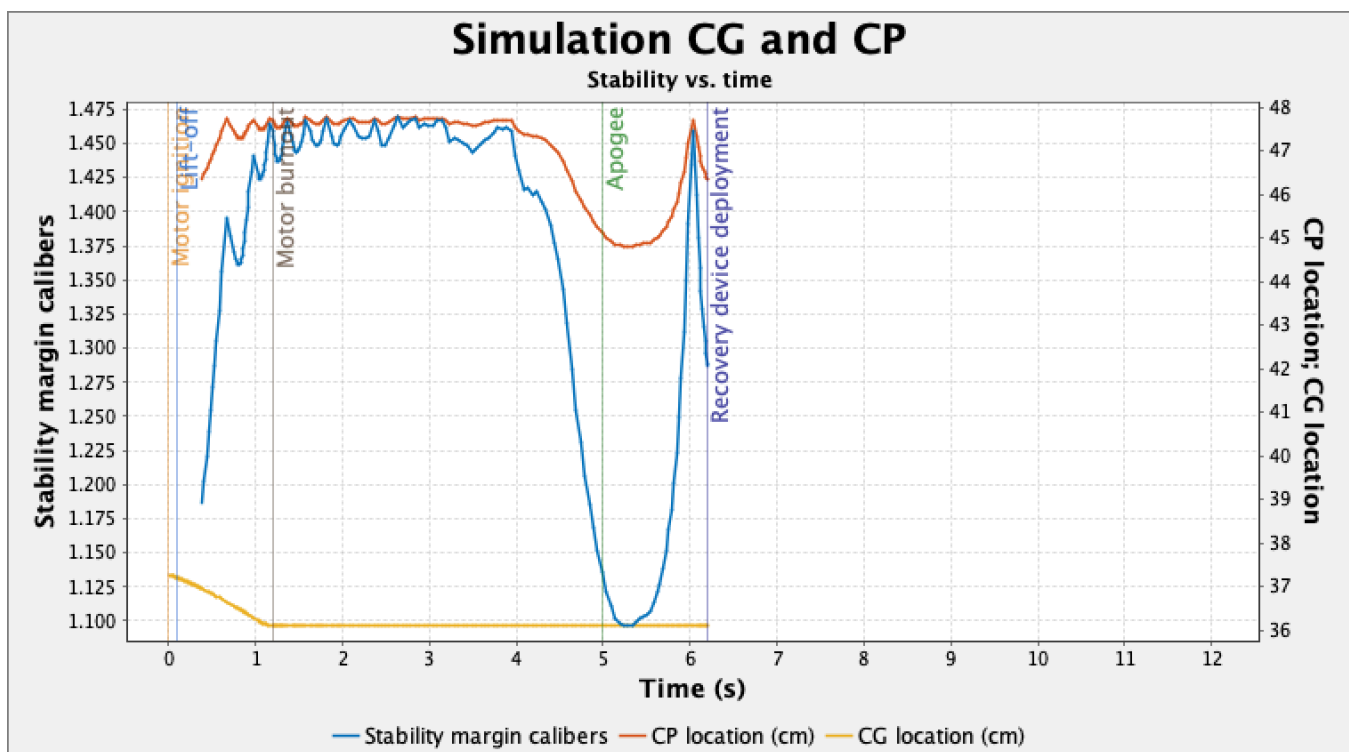


Figure 16: Static stability margin at liftoff: 1.19 calibers. At burnout: 1.45 calibers.

4.2.2 Trajectory Simulation

The predicted trajectory was evaluated with two independent models: the OpenRocket simulation (Section 4.1) and the in-house 1D point-mass Python model (Section 4.1.2). For the Quest F41, OpenRocket predicts a peak altitude of ~95 m and a peak velocity of ~40 m/s at a liftoff thrust-to-weight ratio of about 5.7; the Python model predicts ~88 m and ~38 m/s. The two agree to within roughly 8 %, the difference arising from differing drag and mass assumptions. Both assume vertical, stable flight and therefore represent an upper bound on performance. The measured Flight-1 apogee of 54 m (Section 6.3) is well below this, the shortfall reflecting energy lost to the off-axis, tumbling trajectory that resulted from the marginal 1.15-caliber liftoff margin and the absence of a launch rail. Parametrically, increasing either dry mass or drag coefficient reduces the predicted apogee; motor selection was ultimately governed by international availability (Section 3.2) rather than by optimising predicted altitude.

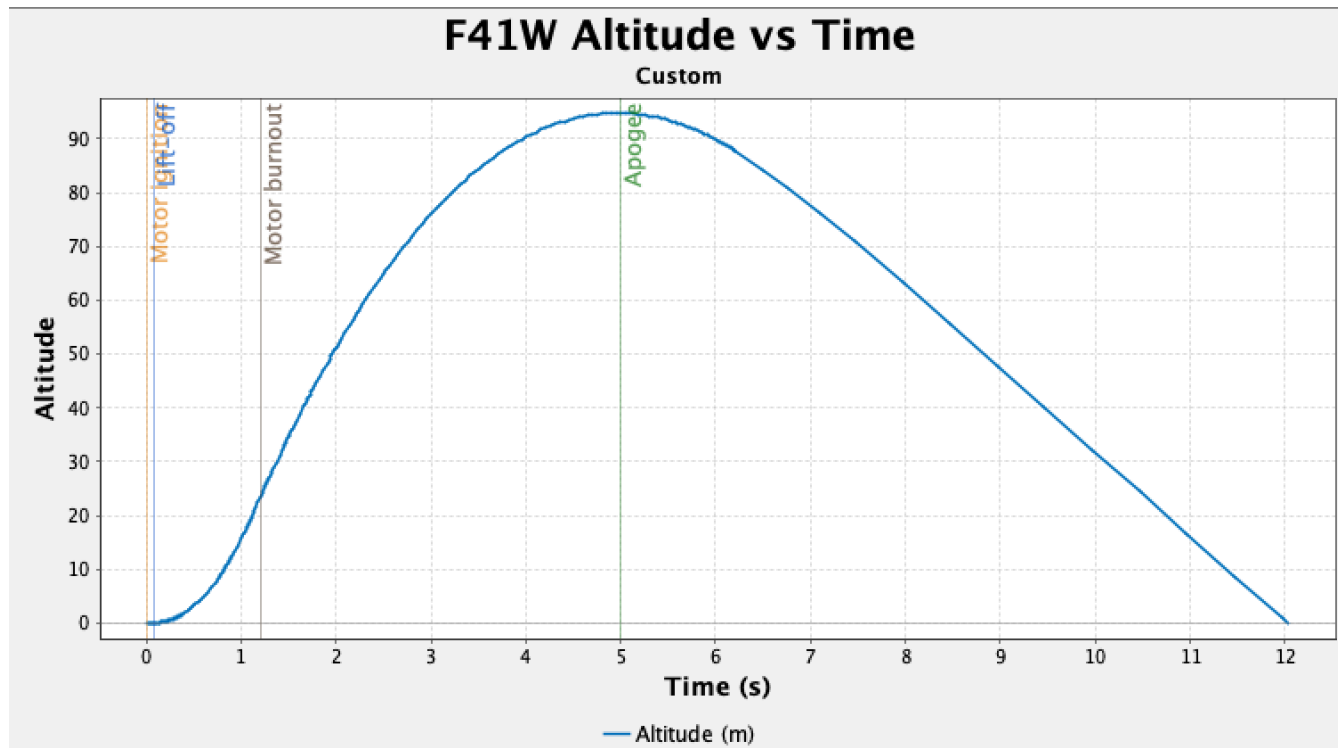


Figure 17: F41W motor simulation of altitude vs time

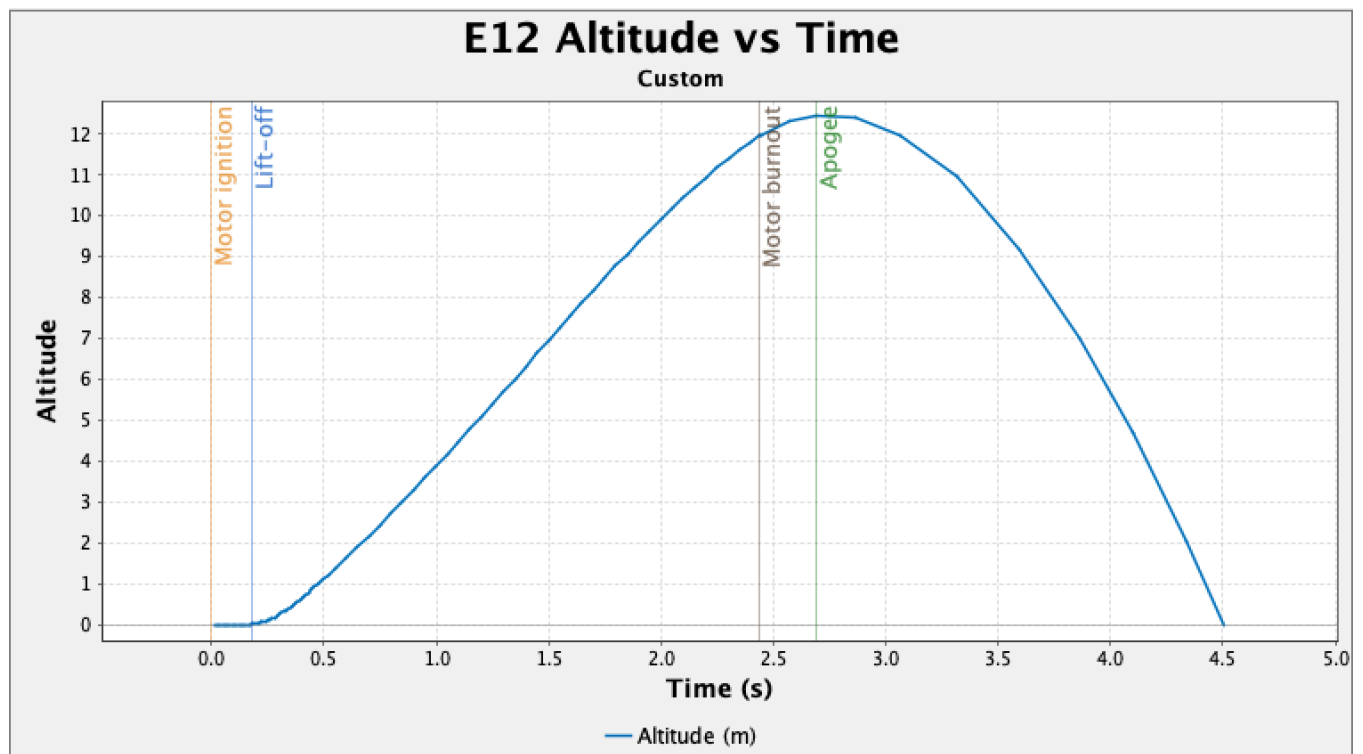


Figure 18: E12 motor simulation of altitude vs time

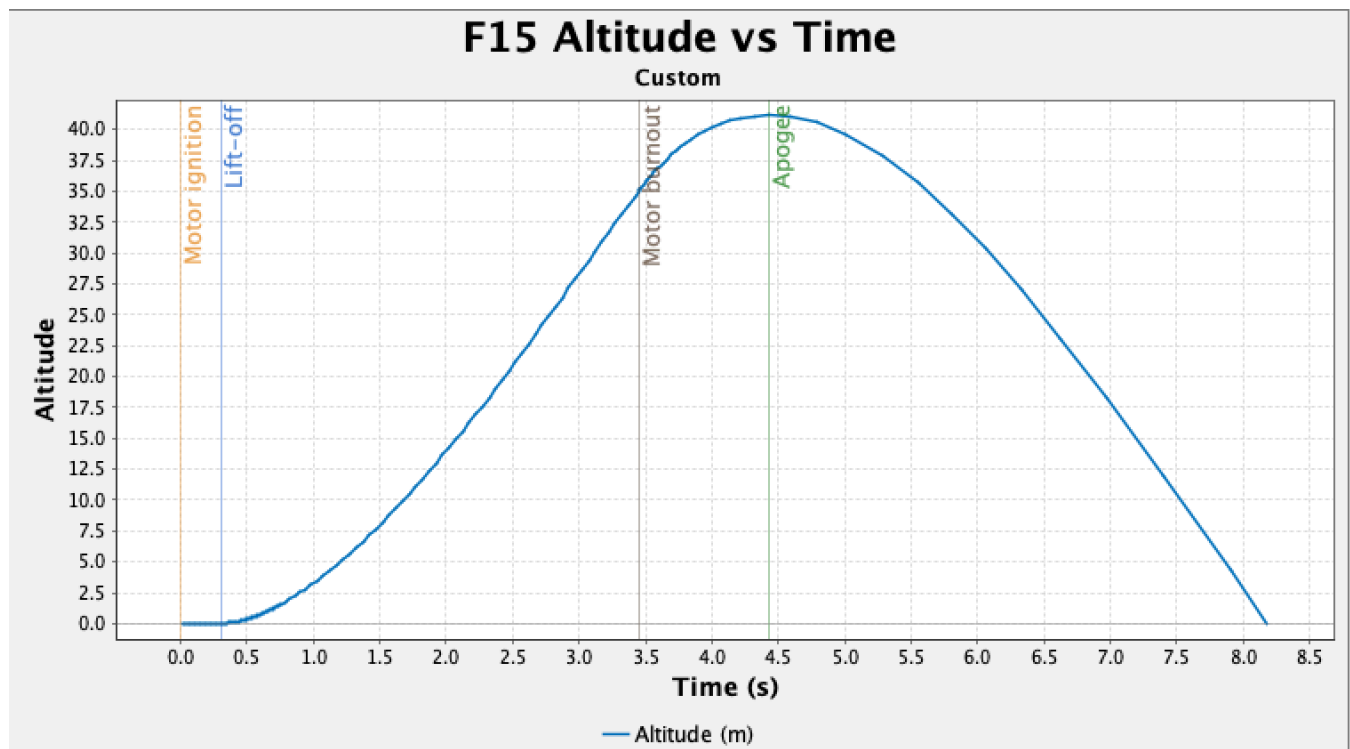


Figure 19: F15 motor simulation of altitude vs time

5 Electronics and Avionics

5.1 Flight Computer Selection

Table 5: Flight Computer Selection

Flight Computer	Raspberry Pi Pico	Teensy 4.1	Arduino Nano
Clock Speed	133 MHz	600 MHz	16 MHz
I2C Support	Two: 1 mbps	Three: 0.1-1 mbps	Two: 0.1-0.4 mbps
PWM output channels	16	35	6
SD card support	No SD card slot	SD card slot	No SD card slot
Cost (\$)	5.5	40	13



Figure 20: Teensy 4.1 Microcontroller

Teensy 4.1 was chosen over other alternatives despite the cost. 600 MHz ARM Cortex-M7 is advantageous over slower alternatives for a 200 Hz control loop. Moreover, Teensy 4.1 contains a micro SD card slot which is useful for data logging during flight.

5.2 IMU Selection

Table 6: IMU Selection

IMU Sensor	MPU-9250	MPU-6050	LSM6DSO
DOF	9	6	6
Sample Rate	1-4 kHz	1-8 kHz	0.0125-6.66 kHz
Noise Density (accelerometer)	300 $\mu\text{g}/\sqrt{\text{Hz}}$	400 $\mu\text{g}/\sqrt{\text{Hz}}$	70 $\mu\text{g}/\sqrt{\text{Hz}}$
Cost (\$)	20	12	15

MPU-6050 was chosen despite the higher noise density because of lower cost. In addition, 6 DOF, 3-accelerometer and 3-gyroscope, is enough for its application in this particular rocket.

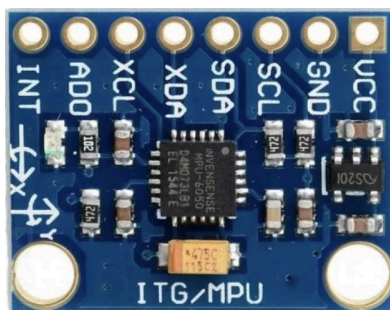


Figure 21: MPU-6050 IMU

5.3 Barometer Selection

Table 7: Barometer Selection

Barometer Sensor	BMP-280	BMP-388
Accuracy	$\pm 12 \text{ Pa } (\pm 1\text{m})$	$\pm 8 \text{ Pa } (\pm 0.5\text{m})$
Size (mm)	2.0 x 2.5	2.0 x 2.0
Pressure range (hPa)	300-1100	300-1250
Current drawn (μA)	0.1	2
Cost (\$)	5	10

BMP-388 was chosen over cheaper alternative BMP-280 for the higher accuracy and smaller size for just 5 dollars more.

5.4 Servo Selection

2 MG90s servos were used as part of the parachute deployment mechanism, opening the two nose cone halves and allowing the parachute to deploy.

Table 8: Servo Selection

Specs	
Operating Voltage (V)	4.8-6.0
Size (mm)	22.8 x 12.2 x 31
Max Torque (kg·cm)	2.2 at 4.8V
Weight (g)	13.4
Cost (\$)	14

The MG90s servos were chosen because their small size allows for fitting them inside the nose cone as depicted on (show pic of cad) and they operate at a voltage range that the teensy 4.1 can supply.



Figure 22: MG90s micro servo

5.5 SD card selection

A **SanDisk 32GB Ultra® microSDHC 120MB/s A1 Class 10 UHS-I** was selected as the micro sd card as it is pretty standard and inexpensive.



Figure 23: Micro SD card used for data logging

5.6 Power Architecture

The avionics are powered by a single 2-cell LiPo battery (7.4 V nominal, 550 mAh). The pack feeds a switching UBEC that steps 7.4 V down to a regulated 5 V rail for the servos, and also supplies the Teensy 4.1's **Vin** input; the Teensy's onboard linear regulator then derives the 3.3 V rail that powers the flight controller itself, the MPU-6050 IMU, the BMP-388, and the microSD interface. To keep the noisy, high-current servo load from disturbing the sensitive logging and sensor electronics, the servos are driven from a dedicated BEC output and share only a common ground with the flight computer. A $1\mu\text{F}$ capacitor was placed at the Teensy **GND/5V** pins to absorb transient current spikes affecting sd card logging.

The per-component draw and the resulting budget are summarized in Table 9.

Table 9: Power Architecture

Component	Rail (V)	Current (mA)	Power (W)
Teensy 4.1 (600Mhz)	5 → 3.3	120	0.600
MPU 6050-IMU	3.3	3.75	0.012
BMP-388	3.3	0.5 (peak)	0.0017
microSD (write burst)	3.3	150	0.495
MG90s servo x 2	5V	400 mA ea (800mA) (peak)	4.00
Peak Total (servos moving)	-	-	5.11
Quiescent (servos holding)	-	-	1.11

*Remember to cite data sheets for all components

Accounting for the UBEC efficiency (~85 %), the current drawn *from the 7.4 V battery* is:

- **Peak (both servos sweeping at deployment):** $5.11 \text{ W} / (7.4 \text{ V} \times 0.85) \approx \mathbf{0.81 \text{ A}}$ — but only for the ~1–2 s deployment sweep.
- **Quiescent (logging + holding, the entire rest of the flight):** $1.11 \text{ W} / (7.4 \text{ V} \times 0.85) \approx \mathbf{0.18 \text{ A}}$.

Endurance check. At the quiescent draw of ~0.18 A, a 550 mAh pack provides roughly $550 \text{ mAh} / 180 \text{ mA} \approx \mathbf{3 \text{ hours}}$ of runtime. The servo peak adds only ~0.82 A for a second or two ($< 0.5 \text{ mAh}$), and a complete mission — pad arming through descent — lasts on the order of a few minutes, consuming well under 5 % of pack capacity. The battery therefore provides ample flight time by a wide margin.

5.7 Wiring Diagram

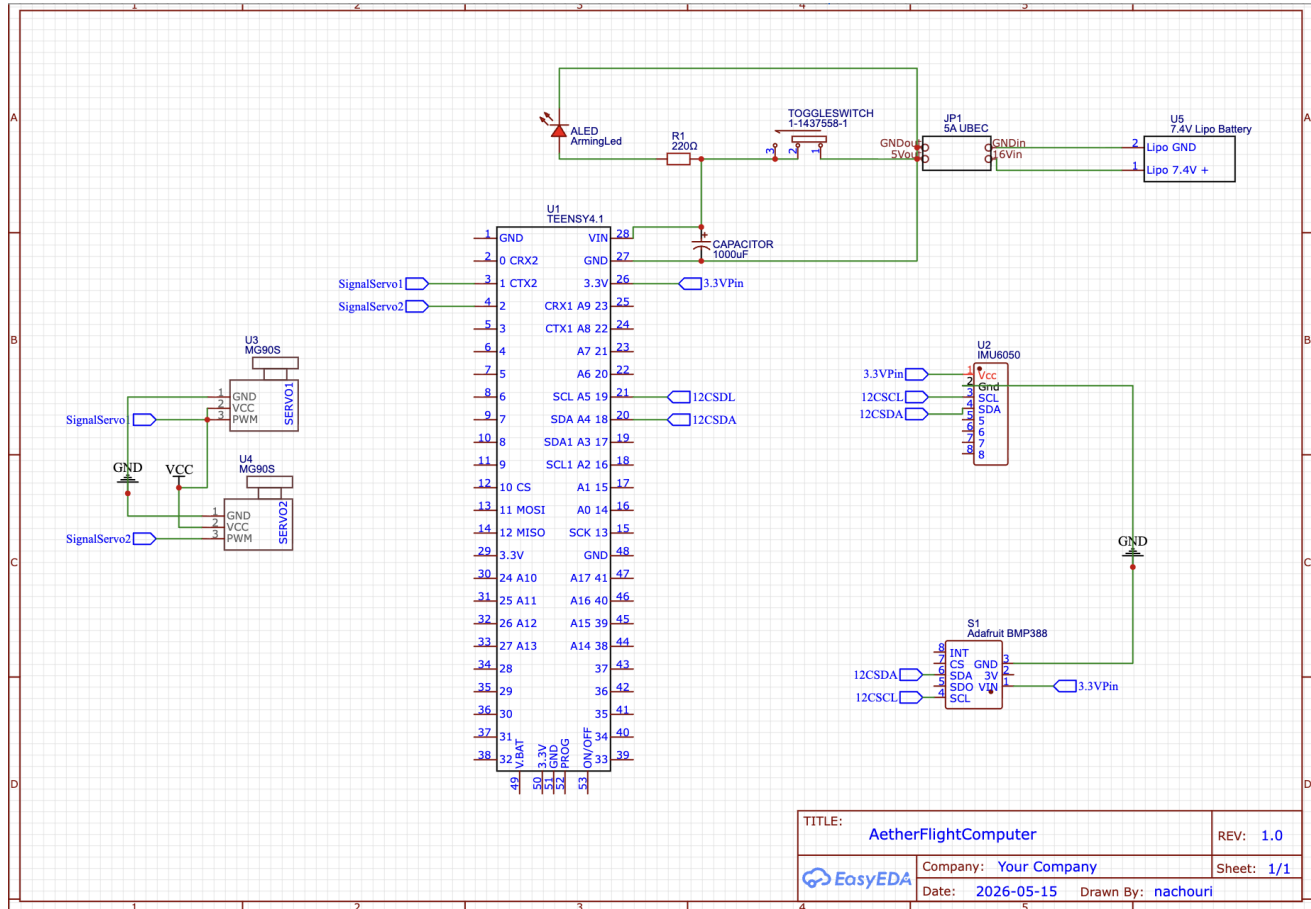


Figure 24: Wiring Diagram of Aether Avionics

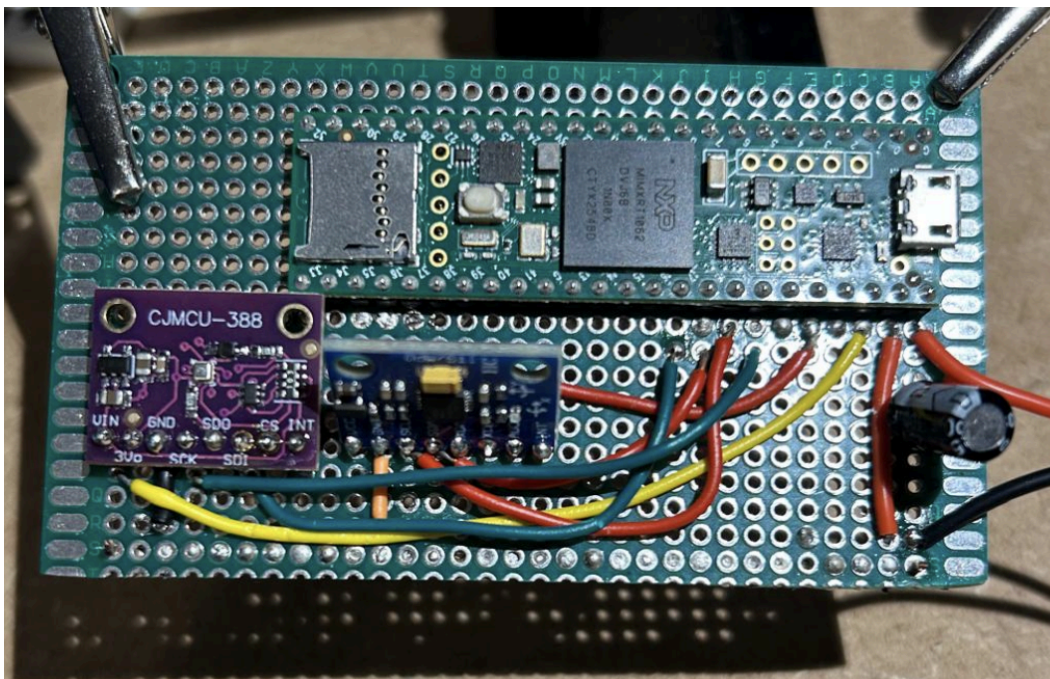


Figure 25: Wiring block diagram (recovery servos omitted)

5.8 E-Match Circuit Igniter

The ignition circuit was made with a 12V battery connected to a QIACHIP 433Mhz Wireless Remote Control Switch that was connected to the igniter using some alligator clips. A remote control was later used to start the ignition.

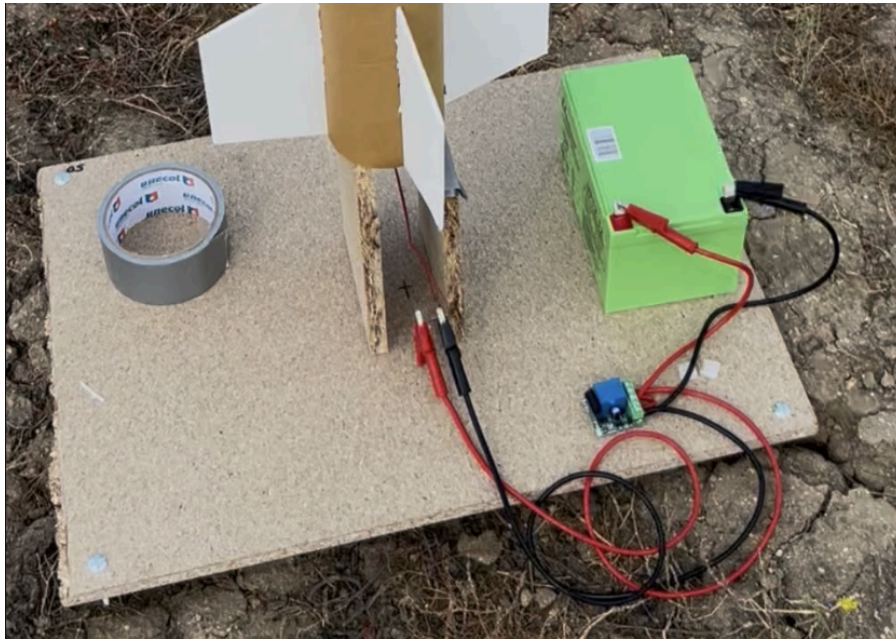


Figure 26: E-match circuit

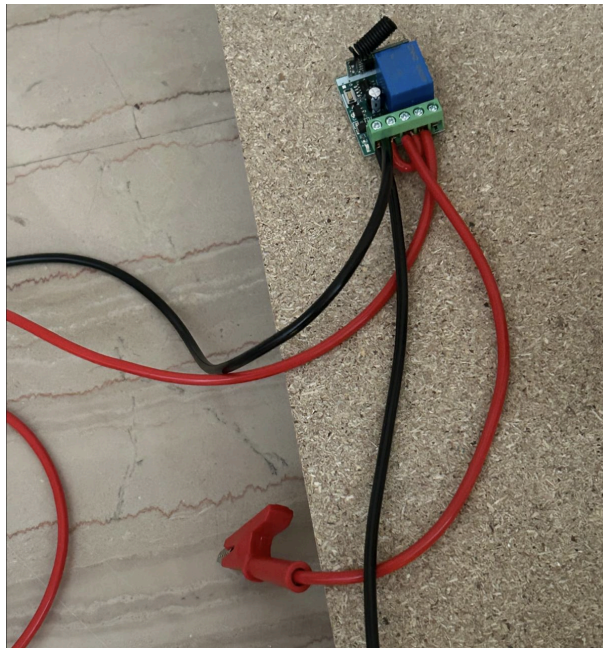


Figure 27: QIACHIP 433Mhz Wireless Remote Control Switch

5.9 Launch Pad

The launch pad was built using 1" birch wood, thick enough to ensure no fire danger and 4 screws as seen in figure X. No rail was implemented, which led to stability problems during both flights as mentioned previously.

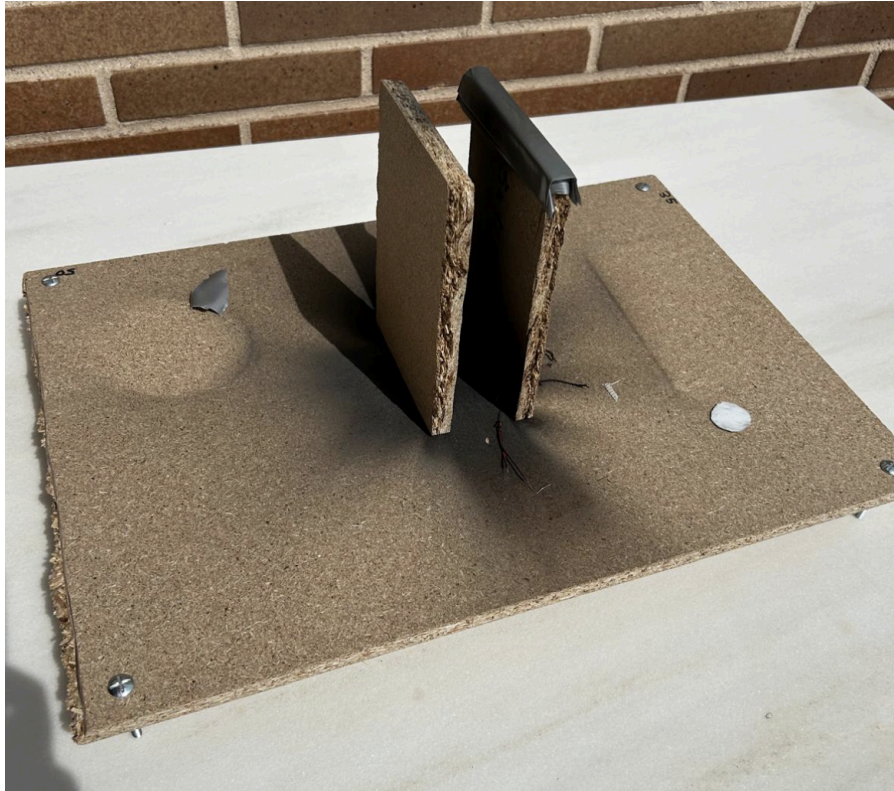


Figure 28: Launch Pad

6 Test and Integration

6.1 Flight Test Results

6.1.1 Flight 1

Date: 15 August 2026. Motor: Quest F41-11W. Pre-flight mass: 880 g. Peak altitude: 54 m (measured, LOG030). Peak velocity: 28.5 m/s (barometer-derived). Time to apogee: ~3 s after launch.

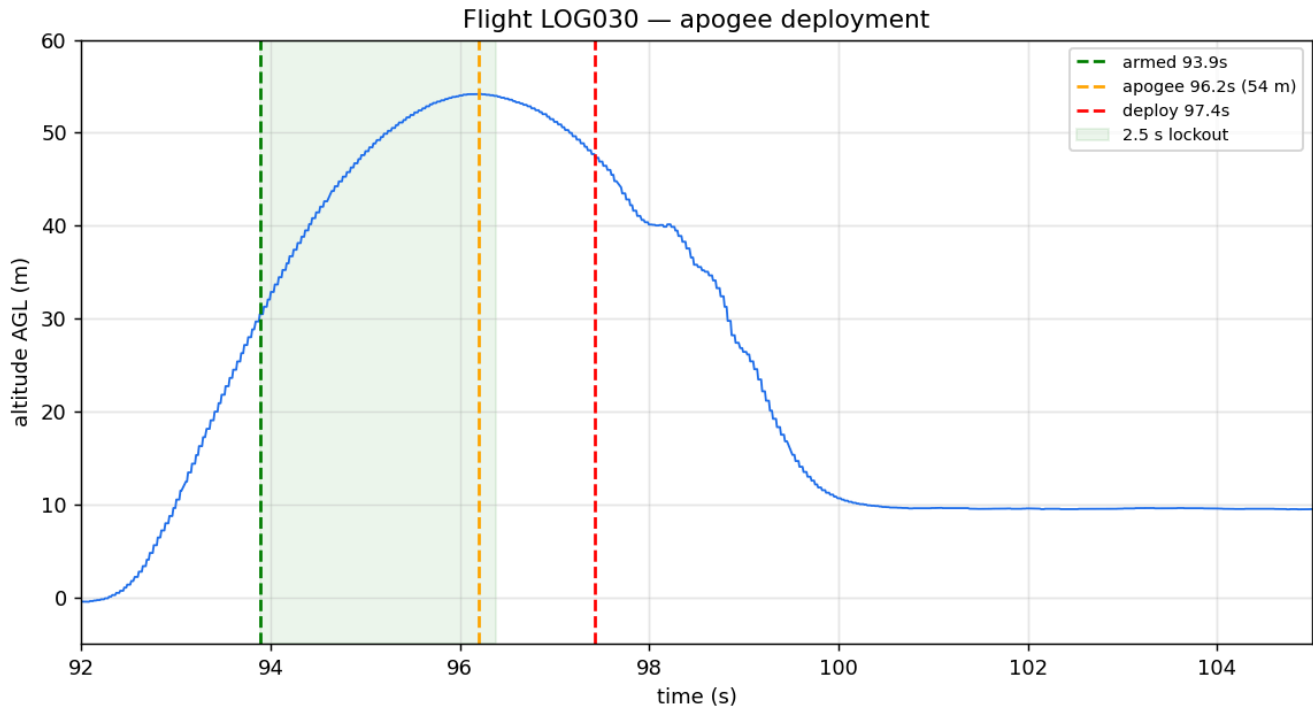


Figure 29: Flight 1 (LOG030) altitude vs time

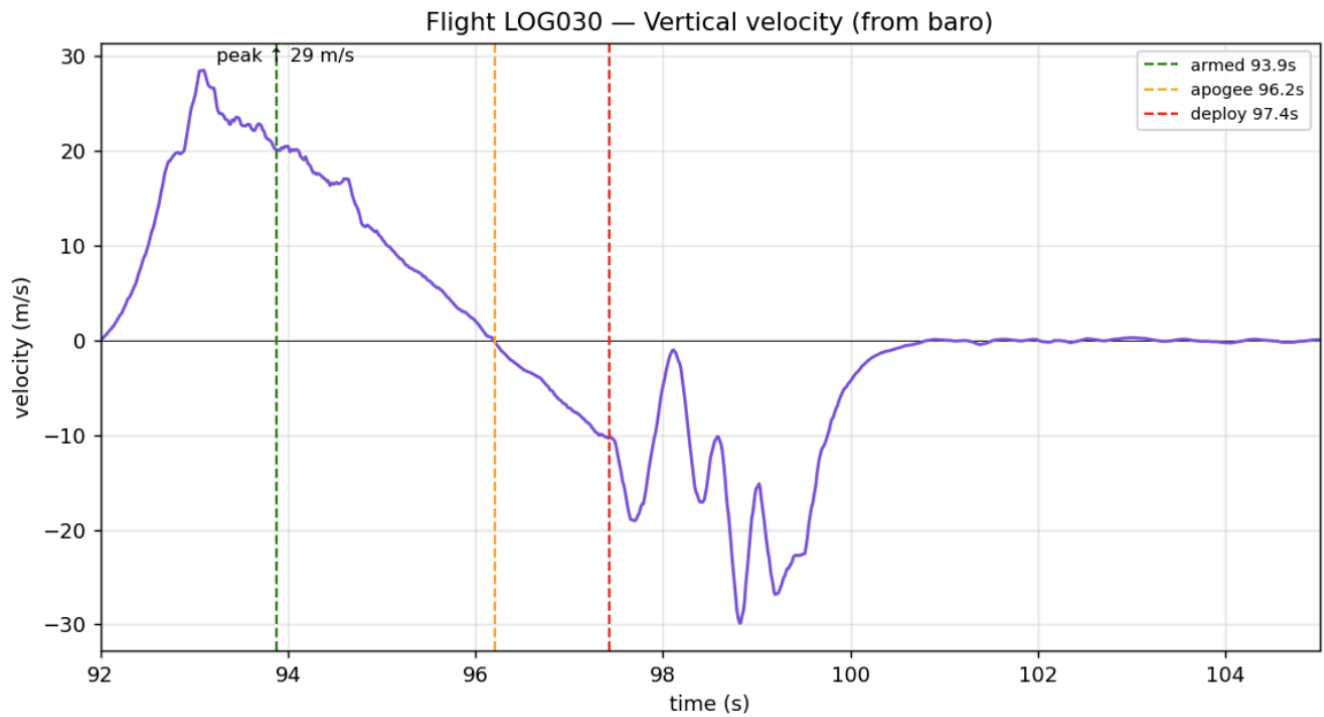


Figure 30: Flight 1 (LOG030) velocity vs time

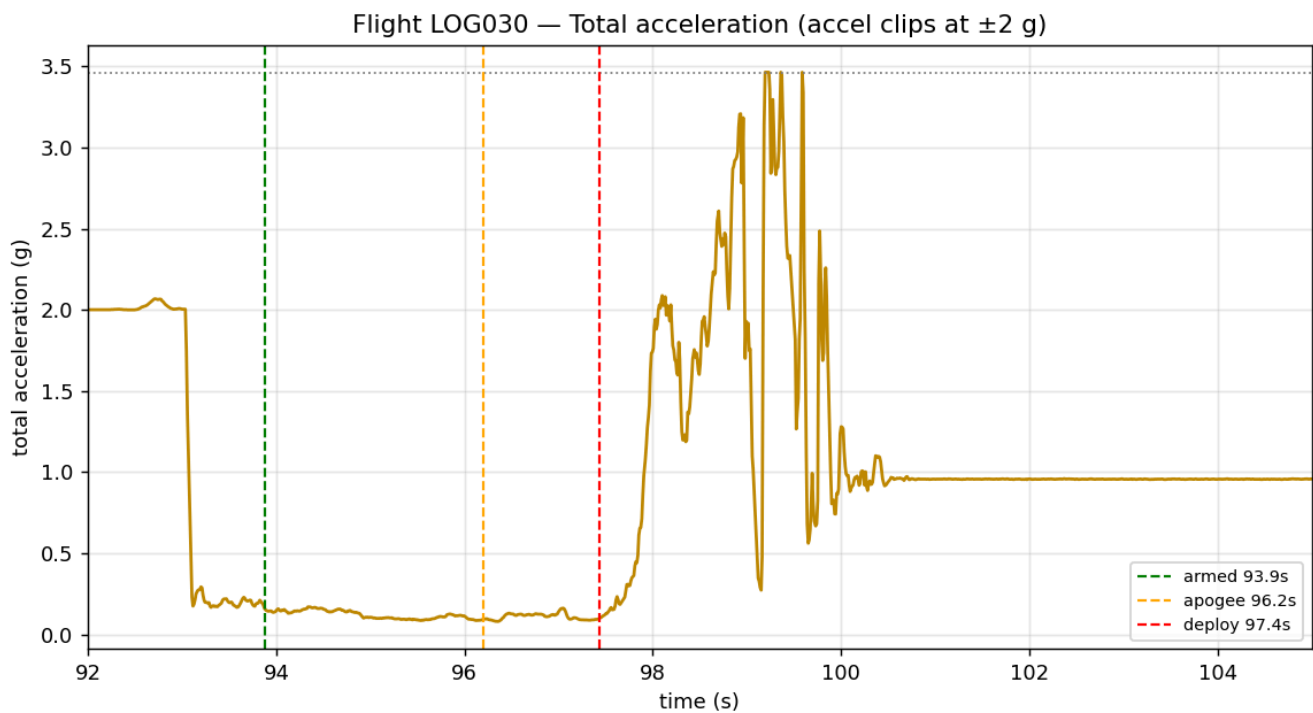


Figure 31: Flight 1 (LOG030) acceleration vs time

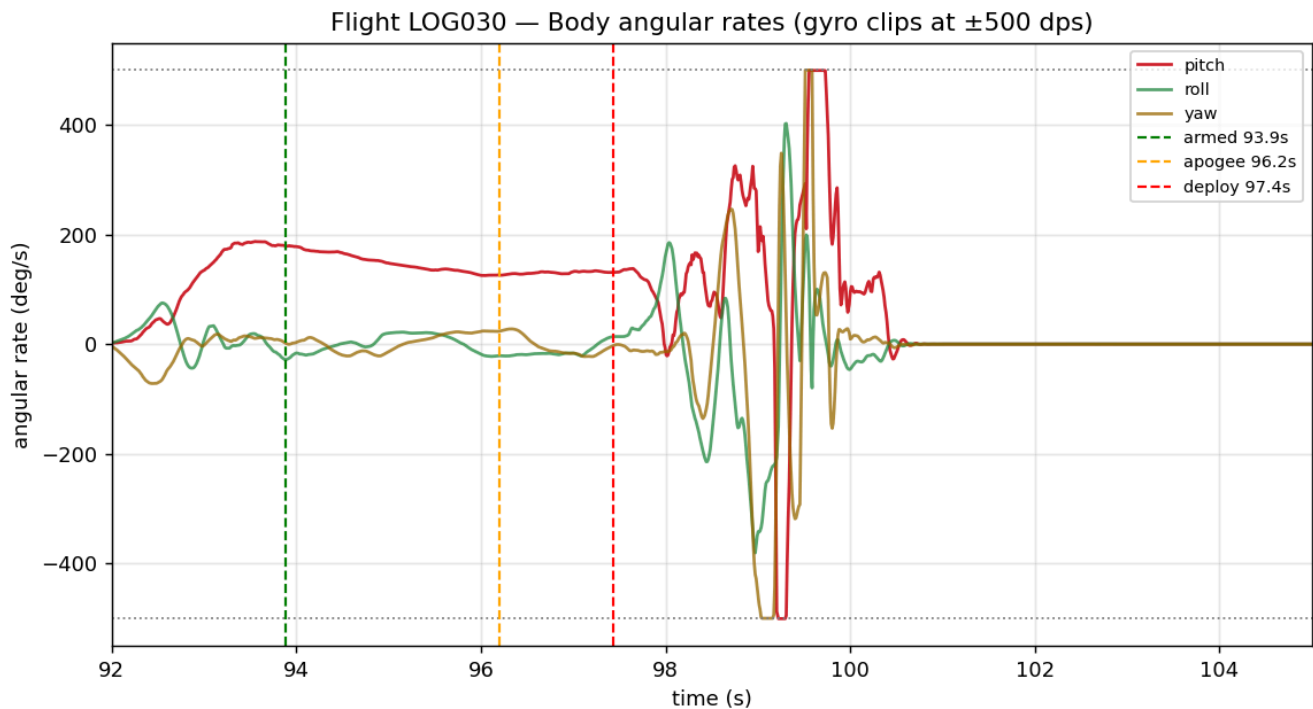


Figure 32: Flight 1 (LOG030) body rates — pitch, roll and yaw vs. time

Parachute deployment mechanism deployed too too close to the ground which caused moderate damage on the nose cone on this first flight. Nevertheless, the mechanism worked as intended, minor changes to the code should fix this problem (reduce lockout time when mechanism is armed so it can deploy right after apogee).

A second limitation was the accelerometer configuration. The MPU-6050 was set to its ± 2 g range, which saturated throughout powered flight: the logged total acceleration rails at 3.46 g ($= \sqrt{3} \times 2$ g) rather than the true value. From the F41 peak thrust (~ 54 N) and the liftoff mass, the actual boost acceleration was closer to 6 – 7 g. Because the accelerometer clipped, both the measured acceleration and any acceleration-derived velocity are lower bounds only; the reliable velocity estimate comes from differentiating the barometric altitude, which gives a peak of ~ 28.5 m/s on ascent. The accelerometer range has since been increased to ± 16 g.

Also, the lack of launch rail generated stability problems during flight as the rocket steered out of course slightly.

This is corroborated by the gyroscope record, which shows a sustained pitch rate of ~ 150 – 190 $^{\circ}$ /s from the moment of liftoff — the signature of the vehicle arcing over almost immediately, before the fins had enough airspeed to stabilise it.

6.1.2 Flight 2

Date: 25 August 2026. Motor: Quest F41-11W. Pre-flight mass: 910 g. Peak altitude: 40 m (measured, LOG045). Peak velocity: 21 m/s (barometer-derived).

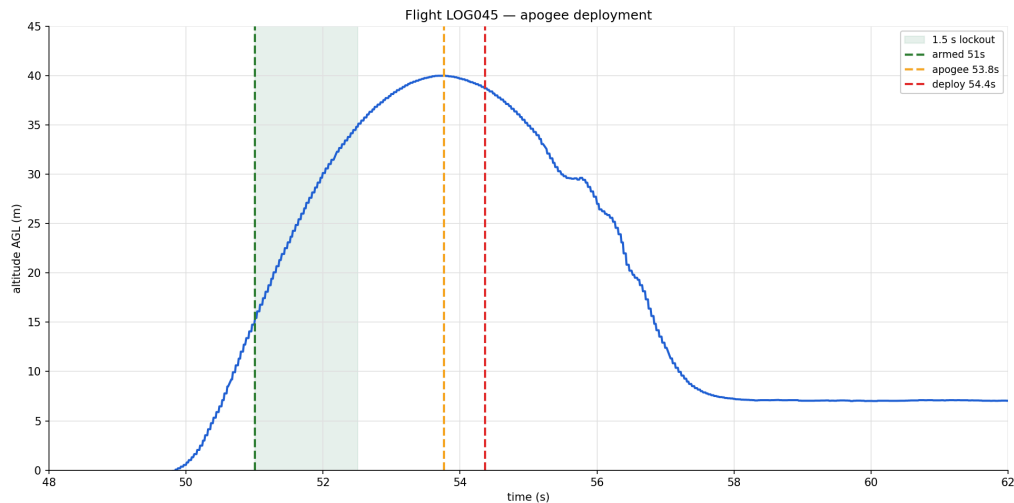


Figure 33: Flight 2 (LOG045) altitude vs time

Fixed issue with parachute deploying too late by reducing the lockout time from 2500ms to 1500ms

The main airframe suffered critical damage on this second flight. Time constraints made it difficult to fix the moderate damage caused by the parachute deployment issue on the first flight. The flight computer survived which made it possible to recover the data.

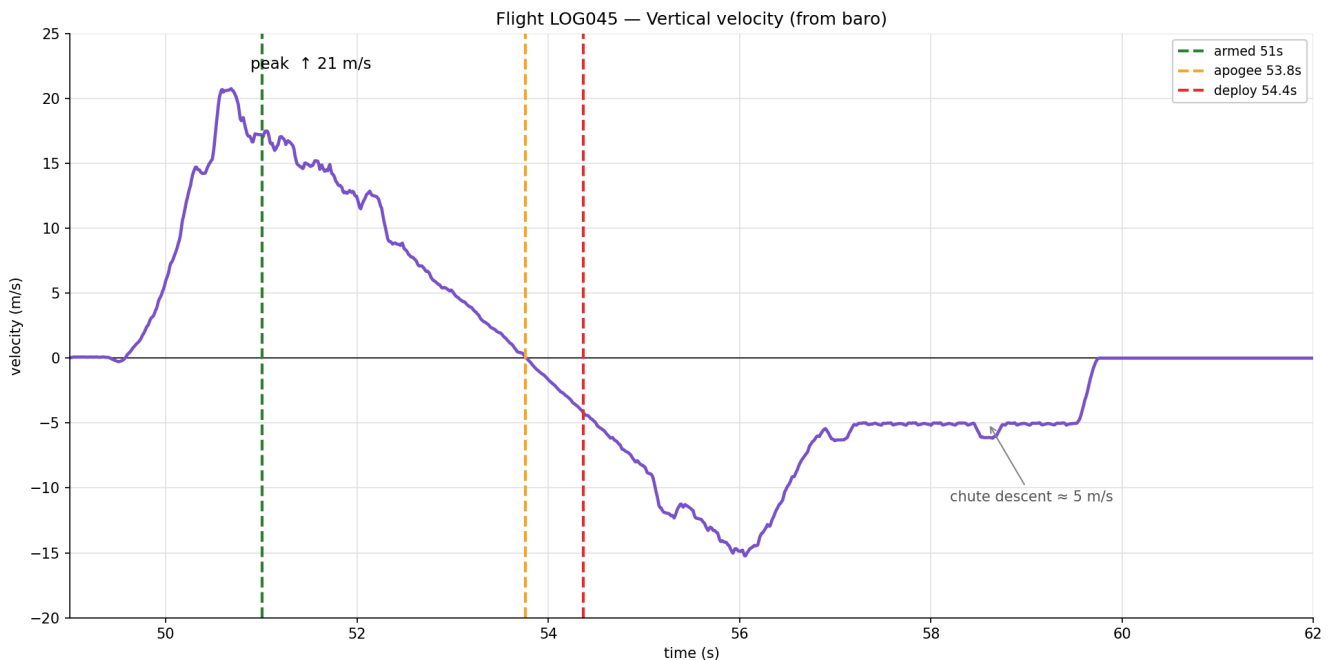


Figure 34: Flight 2 (LOG045) velocity vs time

The descent was also better controlled than on Flight 1, the parachute deploying nearer apogee after the lockout-timing change.

The sensor-range limitations from Flight 1 were only partially addressed. The accelerometer was still configured for ± 2 g and again saturated (total acceleration railed at 3.46 g), so the true boost acceleration is not captured. The gyroscope range had been set to ± 2000 $^{\circ}/s$, but the matching scale factor in firmware was not updated — the code still divided by the ± 500 $^{\circ}/s$ sensitivity (65.5 LSB/ $^{\circ}/s$) instead of 16.4 — so the logged angular rates still saturate at ± 500 $^{\circ}/s$, railing flat at that value during the post-apogee tumble (see the Flight-2 body-rate record). The true peak rotation rate therefore remains unknown, a lower bound of 500 $^{\circ}/s$. Both issues should be corrected for the next iteration using the ± 16 g accelerometer scale (2048 LSB/g) and the ± 2000 $^{\circ}/s$ gyroscope scale (16.4 LSB/ $^{\circ}/s$).

For this reason graphs for angular rate and acceleration for flight 2 (log045) have not been included in this report.

Table 10: Flight performance summary (Flight 1 vs. Flight 2)

Metric	Flight 1	Flight 2	Target
Peak Velocity (m/s)	29	21	40.1 (as per OpenRocketSim)
Peak Accel (g)	3.45	3.45	4.84 (as per OpenRocketSim)
Peak altitude (m)	54	40	95 (as per OpenRocket sim)
Post-flight damage	Moderate Damage to nose cone	Critical damage airframe	None

7 Manufacturing

7.1 Built Components

Table 11: Built Components

Component	Material	Justification
Motor Adapter	PETG (FDM)	Heat resistance > PLA; motor proximity; stiff under load
Nose cone	PLA	Complex geometry, easily printed; smooth ogive shape
Fins	PLA	Integrated fin geometry; bonded with 30-min epoxy
Parachute	PEVA	Zero Air Permeability (Zero Porosity): does not let any air through the fabric.
Avionics and Battery Bay	PLA	Custom 3D printed, PLA widely available.

7.2 Mass Budget

Table 12: Mass Budget

Component	Material	Weight (g)
Nose cone halves (x2)	PLA	70
Parachute bay	PLA	15
Spring	Steel	3
Nose cone top half	PLA	45
MG90s servos (x2)	-	27
Nose Cone Bottom Half	PLA	35
U-bracket	Steel	70
Parachute + shock cord	PEVA	60
Avionics + Battery Bay	PLA	65
Flight Computer	-	40
UBEC	-	8
Battery	-	40
Motor Mount	PETG	40
Fins (x4)	PLA	60
F41-11W motor	-	57
18" long 3" diameter airframe	Phenolic	255
Igniter	-	2
M3 Screws (x8)	Steel	8
Adhesive (flight 2)	Epoxy	30
	Total:	930

7.3 Materials and components bought

These are all materials and components bought and the total cost.

[TVC Rocket Tracker](#)

8 Conclusion

AETHER met its core objectives: a fin-stabilized rocket with an autonomous, barometrically triggered, servo-actuated recovery system was designed, built, and flown twice, logging full-rate flight data on both flights. The recovery mechanism functioned on both flights, and a firmware timing change between flights moved deployment closer to apogee.

The trajectory, however, did not go as designed: both flights flew off-vertical and well below the ~95 m prediction (Flight 1: 54 m, Flight 2: 40m) . The flight data identifying the cause, a sustained ~150–190 °/s pitch rate from liftoff shows the vehicle arced over almost immediately, which was caused by a marginal static margin (1.15 cal at liftoff, below the 1.5-cal target) and the absence of a launch rail.

The three most important lessons learned were:

- Guide the rocket off the pad — a launch rail giving ≥ 15 m/s exit velocity is essential; its absence was a primary cause of the instability.
- Size sensor ranges to the flight — the ± 2 g accelerometer and ± 500 °/s gyro both saturated, making the IMU data qualitative only; ± 16 g and ± 2000 °/s are required.
- Design in margin — ballast the nose to ≥ 1.5 –2 cal and tune the deployment lockout to the actual coast time so the chute fires at apogee.

Ideally, next iteration work will address all these issues as well as implement thrust vector control (TVC) to the design (basically a gimbaled mechanism implemented in the motor mount that would control the motor and make the fins obsolete in this design).

Appendix A — Flight Data Logs

Flight 1:

[LOG030.CSV](#)

Flight 2:

[LOG045.CSV](#)

Appendix B — Code

B.1 FlightComputer_ApogeeDeploy.ino

```

/*
  Flight Computer Logger + Apogee Deployment – Teensy 4.1
  -----
  Logs IMU + baro to microSD, and OPENS the nose-cone servos at APOGEE
  (when the barometric altitude has clearly started falling) to release
  the parachute.

  SAFETY / DESIGN:
  * Deployment ARMS only after the rocket climbs past LAUNCH_ALT_M AGL.
  * BOOST LOCKOUT: deployment is blocked until MIN_COAST_MS after launch,
    so high-speed baro noise during boost/early coast can't fire early.
  * Apogee is declared only after altitude falls APOGEE_DROP_M below the
    peak for APOGEE_CONFIRM consecutive fresh baro samples (noise reject).
  * One-shot: once fired, servos open once and stay open.
  * Backup timer deploys BACKUP_DEPLOY_MS after launch if baro fails.

  Servos: signal pins 3 & 1, powered from a SEPARATE BEC, grounds common.
  Servo angles: 20 = CLOSED (cone held), 140 = OPEN (releases).
*/

#include <Wire.h>
#include <SD.h>
#include <BMP388_DEV.h>
#include <Servo.h>

// — User config

#define KNOWN_ALTITUDE_M      740      // pad elevation, metres
#define CALIBRATION_SAMPLES  20
const float ALPHA = 0.98f;

// — Apogee deployment config

const float LAUNCH_ALT_M      = 30.0f; // climb this far AGL before arming
const float APOGEE_DROP_M     = 5.0f;  // fall this far below peak = apogee
const uint16_t APOGEE_CONFIRM = 5;      // consecutive fresh baro samples to
confirm
const uint32_t MIN_COAST_MS    = 1500;  // BOOST LOCKOUT: no deploy until this
long after

// arming. Tuned from LOG030 (apogee
~2.3 s after
// arming). Keep < your
time-from-arming-to-apogee.
const float GLITCH_MAX_M      = 100.0f; // reject baro samples that jump more
than this
// from the last good reading
(single-sample spikes)

```

```
const uint32_t BACKUP_DEPLOY_MS = 30000;    // failsafe deploy after launch (ms)
```

```
// — Servos
```

```
Servo servoLeft;
Servo servoRight;
const int PIN_LEFT      = 3;
const int PIN_RIGHT     = 1;
const int LEFT_CLOSED   = 20;    // physical CLOSED (cone held) on YOUR mechanism
const int LEFT_OPEN     = 140;   // physical OPEN (releases the cone)
const int RIGHT_CLOSED  = 20;
const int RIGHT_OPEN    = 140;
const uint32_t STEP_DELAY_MS = 10; // ms per 1-degree step when opening
int leftPos  = LEFT_CLOSED;
int rightPos = RIGHT_CLOSED;
```

```
// — Flight state
```

```
float  baselineAlt = KNOWN_ALTITUDE_M; // AGL zero reference (set at boot)
float  maxAlt      = 0;                // highest AGL seen
bool   armed       = false;
uint16_t dropCount = 0;
uint32_t launchTime = 0;
bool   deployTriggered = false;
bool   deployed      = false;
uint32_t deployStepTimer = 0;
float  lastGoodAg1 = 0;                // last accepted AG1 (for glitch
rejection)
bool   haveGoodAg1 = false;
```

```
// — SD logging
```

```
File  logfile;
char  filename[16];
uint32_t lastFlush = 0;
const uint32_t FLUSH_INTERVAL_MS = 250;
```

```
// — BMP388
```

```
BMP388_DEV bmp388;
float bmpTemp = 0, bmpPressure = 0, bmpAltitude = 0;
float seaLevelPressure = 1013.25;
```

```
// — MPU6050
```

```
float RateRoll, RatePitch, RateYaw;
float AccX, AccY, AccZ;
float AccAngleRoll, AccAnglePitch;
float RateCalRoll, RateCalPitch, RateCalYaw;
```

```
float AngleRoll = 0, AnglePitch = 0, AngleYaw = 0;
uint32_t LoopTimer;
```

```
//
```

```
void read_imu(void) {
    Wire.beginTransaction(0x68);
    Wire.write(0x3B);
    Wire.endTransmission();
    Wire.requestFrom(0x68, 14);

    int16_t ax = Wire.read() << 8 | Wire.read();
    int16_t ay = Wire.read() << 8 | Wire.read();
    int16_t az = Wire.read() << 8 | Wire.read();
    Wire.read(); Wire.read();
    int16_t gx = Wire.read() << 8 | Wire.read();
    int16_t gy = Wire.read() << 8 | Wire.read();
    int16_t gz = Wire.read() << 8 | Wire.read();

    AccX = (float)ax / 2048.0f;    // ±16 g range -> 2048 LSB/g (was 16384 for ±2 g)
    AccY = (float)ay / 2048.0f;
    AccZ = (float)az / 2048.0f;

    RateRoll  = (float)gx / 65.5f;
    RatePitch = (float)gy / 65.5f;
    RateYaw   = (float)gz / 65.5f;

    AccAngleRoll  = atan2(AccY, sqrt(AccX * AccX + AccZ * AccZ)) * 180.0f / PI;
    AccAnglePitch = -atan2(AccX, sqrt(AccY * AccY + AccZ * AccZ)) * 180.0f / PI;
}

void init_imu(void) {
    Wire.beginTransaction(0x68); Wire.write(0x6B); Wire.write(0x00);
    Wire.endTransmission();
    Wire.beginTransaction(0x68); Wire.write(0x1A); Wire.write(0x05);
    Wire.endTransmission();
    Wire.beginTransaction(0x68); Wire.write(0x1B); Wire.write(0x08);
    Wire.endTransmission();
    Wire.beginTransaction(0x68); Wire.write(0x1C); Wire.write(0x18);
    Wire.endTransmission(); // accel ±16 g
    delay(20);

    for (int i = 0; i < 2000; i++) {
        read_imu();
        RateCalRoll  += RateRoll;
        RateCalPitch += RatePitch;
        RateCalYaw   += RateYaw;
        delay(1);
    }
    RateCalRoll  /= 2000.0f;
    RateCalPitch /= 2000.0f;
```

```
RateCalYaw    /= 2000.0f;

read_imu();
AngleRoll    = AccAngleRoll;
AnglePitch   = AccAnglePitch;
}

// ——— Deployment: open both servos once, then hold open (non-blocking)
void triggerDeploy(void) {
    if (!deployTriggered) { deployTriggered = true; deployStepTimer = millis(); }
}

void updateDeploy(void) {
    if (!deployTriggered || deployed) return; // holding closed, or already open
    uint32_t now = millis();
    if (now - deployStepTimer >= STEP_DELAY_MS) {
        deployStepTimer = now;
        bool moving = false;
        // step each servo toward its OPEN angle, regardless of numeric direction
        if (leftPos != LEFT_OPEN) { leftPos += (leftPos < LEFT_OPEN) ? 1 : -1;
servoLeft.write(leftPos); moving = true; }
        if (rightPos != RIGHT_OPEN) { rightPos += (rightPos < RIGHT_OPEN) ? 1 : -1;
servoRight.write(rightPos); moving = true; }
        if (!moving) deployed = true; // both fully open
    }
}

// ——— Apogee detection – call ONLY on a fresh baro sample
void checkApogee(float agl) {
    if (agl > maxAlt) maxAlt = agl; // always track the true peak

    if (!armed && agl > LAUNCH_ALT_M) { // launch confirmed -> arm
        armed = true;
        launchTime = millis();
    }

    // BOOST LOCKOUT: only run apogee detection once we're armed AND past the
    // high-speed boost/early-coast window. Before that, ignore drops entirely so
    // baro noise at speed can't false-trigger.
    if (armed && !deployTriggered && (millis() - launchTime >= MIN_COAST_MS)) {
        if (agl < maxAlt - APOGEE_DROP_M) dropCount++; // falling below the peak
        else dropCount = 0; // still climbing / noise
        if (dropCount >= APOGEE_CONFIRM) triggerDeploy();
    }
}

//

void setup() {
```

```
delay(1500);

pinMode(LED_BUILTIN, OUTPUT);
Serial.begin(115200);
while (!Serial && millis() < 3000);

// Servos attach early and HOLD CLOSED through boost.
servoLeft.attach(PIN_LEFT);
servoRight.attach(PIN_RIGHT);
servoLeft.write(LEFT_CLOSED);
servoRight.write(RIGHT_CLOSED);
leftPos = LEFT_CLOSED; rightPos = RIGHT_CLOSED;

Wire.begin();
Wire.setClock(400000);
delay(250);

// — BMP388 —
Serial.print(F("BMP388 init... "));
if (!bmp388.begin(0x77)) {
    Serial.println(F("FAILED - halting."));
    while (1) { digitalWrite(LED_BUILTIN, HIGH); delay(100);
digitalWrite(LED_BUILTIN, LOW); delay(100); }
}
Serial.println(F("OK"));
bmp388.setTimeStandby(TIME_STANDBY_40MS);
bmp388.setTempOversampling(OVERSAMPLING_X2);
bmp388.setPresOversampling(OVERSAMPLING_X16);
bmp388.setIIRFilter(IIR_FILTER_8);
bmp388.startNormalConversion();

Serial.println(F("Calibrating BMP388 (hold still)..."));
float sum = 0; int n = 0;
while (n < CALIBRATION_SAMPLES) {
    if (bmp388.getMeasurements(bmpTemp, bmpPressure, bmpAltitude)) { sum +=
bmpPressure; n++; }
}
float avgP = sum / CALIBRATION_SAMPLES;
seaLevelPressure = avgP / pow(1.0 - (KNOWN_ALTITUDE_M / 44330.0), 5.255);
Serial.print(F(" Sea-level P0 = ")); Serial.print(seaLevelPressure, 2);
Serial.println(F(" hPa"));

// — MPU6050 —
Serial.println(F("Calibrating MPU6050 (hold still)..."));
init_imu();

// — SD card —
Serial.print(F("SD init... "));
bool sdOK = false;
for (int attempt = 0; attempt < 10; attempt++) {
    if (SD.begin(BUILTIN_SDCARD)) { sdOK = true; break; }
    Serial.print('.');
```



```
    delay(250);
  }
  if (!sdOK) {
    Serial.println(F(" FAILED - halting.));
    while (1) { digitalWrite(LED_BUILTIN, HIGH); delay(500);
digitalWrite(LED_BUILTIN, LOW); delay(500); }
  }
  Serial.println(F("OK"));

  int idx = 0;
  do { sprintf(filename, "LOG%03d.CSV", idx); idx++; } while (SD.exists(filename));
  logFile = SD.open(filename, FILE_WRITE);
  if (!logFile) { Serial.println(F("Could not open log file - halting.)); while
(1); }

  logFile.println(F("time_ms,roll_deg,pitch_deg,yaw_deg,rollrate_dps,pitchrate_dps,ya
wrate_dps,accX_g,accY_g,accZ_g,totalG,temp_C,pressure_hPa,altitude_m,agl_m,armed,de
ployed"));
  logFile.flush();

  // Capture ground level as the AGL zero-reference (average a few fresh samples).
  Serial.println(F("Setting ground reference...));
  float aglSum = 0; int m = 0;
  while (m < 10) {
    if (bmp388.getMeasurements(bmpTemp, bmpPressure, bmpAltitude)) {
      aglSum += 44330.0 * (1.0 - pow(bmpPressure / seaLevelPressure, 1.0 / 5.255));
      m++;
    }
  }
  baselineAlt = aglSum / 10.0f;
  Serial.print(F(" Ground alt = )); Serial.print(baselineAlt, 1);
  Serial.println(F(" m));

  Serial.print(F("Logging to )); Serial.println(filename);
  Serial.println(F("Armed on climb past LAUNCH_ALT_M. Waiting for launch...));
  digitalWrite(LED_BUILTIN, HIGH); // solid LED = ready/Logging

  LoopTimer = micros();
  lastFlush = millis();
}

//

void loop() {
  float dt = (micros() - LoopTimer) / 1000000.0f;
  LoopTimer = micros();

  // --- IMU ---
  read_imu();
  RateRoll -= RateCalRoll;
  RatePitch -= RateCalPitch;
```

```

RateYaw    -= RateCalYaw;
AngleRoll  = ALPHA * (AngleRoll  + RateRoll  * dt) + (1 - ALPHA) * AccAngleRoll;
AnglePitch = ALPHA * (AnglePitch + RatePitch * dt) + (1 - ALPHA) * AccAnglePitch;
AngleYaw   += RateYaw * dt;
float totalG = sqrt(AccX * AccX + AccY * AccY + AccZ * AccZ);

// --- BMP: fresh sample -> update altitude AND run apogee detection ---
//   Reject single-sample baro spikes (a glitch could otherwise false-trigger).
float agl = haveGoodAgl ? lastGoodAgl : 0.0f; // hold last good if no
fresh/valid sample
if (bmp388.getMeasurements(bmpTemp, bmpPressure, bmpAltitude)) {
    bmpAltitude = 44330.0 * (1.0 - pow(bmpPressure / seaLevelPressure, 1.0 /
5.255));
    float candidate = bmpAltitude - baselineAlt;
    if (!haveGoodAgl || fabs(candidate - lastGoodAgl) < GLITCH_MAX_M) {
        agl = candidate;                // accepted
        lastGoodAgl = candidate;
        haveGoodAgl = true;
        checkApogee(agl);
    }
    // else: glitch -> ignore, keep last good
agl
}

// --- Backup failsafe: deploy N ms after launch no matter what ---
if (armed && !deployTriggered && (millis() - launchTime > BACKUP_DEPLOY_MS)) {
    triggerDeploy();
}

// --- Advance servo opening (non-blocking) ---
updateDeploy();

// --- Log one row ---
uint32_t t = millis();
logFile.print(t);          logFile.print(',');
logFile.print(AngleRoll, 2); logFile.print(',');
logFile.print(AnglePitch, 2); logFile.print(',');
logFile.print(AngleYaw, 2); logFile.print(',');
logFile.print(RateRoll, 2); logFile.print(',');
logFile.print(RatePitch, 2); logFile.print(',');
logFile.print(RateYaw, 2); logFile.print(',');
logFile.print(AccX, 3); logFile.print(',');
logFile.print(AccY, 3); logFile.print(',');
logFile.print(AccZ, 3); logFile.print(',');
logFile.print(totalG, 3); logFile.print(',');
logFile.print(bmpTemp, 2); logFile.print(',');
logFile.print(bmpPressure, 2); logFile.print(',');
logFile.print(bmpAltitude, 2); logFile.print(',');
logFile.print(agl, 2); logFile.print(',');
logFile.print(armed ? 1 : 0); logFile.print(',');
logFile.println(deployed ? 1 : (deployTriggered ? 2 : 0)); // 0=closed 2=opening
1=open

```

```

    if (t - lastFlush >= FLUSH_INTERVAL_MS) { logFile.flush(); lastFlush = t; }

    // Heartbeat
    Serial.print(F("t=")); Serial.print(t);
    Serial.print(F(" agl=")); Serial.print(agl, 1);
    Serial.print(F(" max=")); Serial.print(maxAlt, 1);
    Serial.print(F(" armed=")); Serial.print(armed);
    Serial.print(F(" deploy=")); Serial.println(deployTriggered);

    delay(10);
}

/*
  BENCH TEST (Low thresholds) vs FLIGHT:
  - To bench-test by hand, temporarily set LAUNCH_ALT_M = 1.0, APOGEE_DROP_M =
    0.5,
    and MIN_COAST_MS = 0 so you can arm and trigger over small height changes.
  - RESTORE LAUNCH_ALT_M = 30, APOGEE_DROP_M = 5, MIN_COAST_MS = 1000 before
    flight.

  FLIGHT NOTES (updated from LOG030):
  * LOG030: armed at ~28 m, apogee ~2.3 s after arming (~51 m AGL), deployed ~7 m
    below peak. MIN_COAST_MS Lowered 2500 -> 1000 so deploy lands nearer apogee.
    If you change motors/mass, re-check MIN_COAST_MS stays <
    time-from-arming-to-apogee.
  * Accel now ±16 g (2048 LSB/g). LOG030 was clipped at ±2 g – boost data was
    useless.
  * Static/vent holes in the bay + seal the nose-cone gap so ram air can't
    corrupt baro.
  * Don't sit armed-waiting on the pad too long: baseline is set at boot, so baro
    drift
    over a long pad wait slowly shifts AGL (LOG030 drifted ~3 m over ~90 s).
  * Confirm servos hold closed under boost vibration/high-G.
*/

```

B.2 ServoDropTest.ino (bench test)

```

/*
  Parachute Ejection Bench Test – Teensy 4.1 + BMP388 + 2 servos
  -----
  Records the starting altitude, then OPENS both servos once as soon as the
  altitude has dropped DROP_TRIGGER_M below that baseline. Watch the live
  "drop =" value on Serial @ 115200 while you lower the rocket.

  BMP388 @ 0x77, SDA->18, SCL->19, 3.3 V
  Servos: signal pins 3 & 1, powered from a SEPARATE BEC, grounds common
*/

#include <Wire.h>
#include <BMP388_DEV.h>
#include <Servo.h>

// — Test config

```

```
const float    DROP_TRIGGER_M = 0.30f;    // open servos after altitude drops this
far.                                         // NOTE: 0.10 m (10 cm) is only ~0.012 hPa
                                           // the BMP388's reliable resolution, so it
- below                                   // trigger. 0.30 m is a dependable
                                           // bench-test value.
may never                                // readings averaged for the start
                                           // reference

// — Servos

—
Servo servoLeft;
Servo servoRight;
const int PIN_LEFT    = 3;
const int PIN_RIGHT   = 1;
const int LEFT_CLOSED = 20;    // physical CLOSED (nose held) on YOUR mechanism
const int LEFT_OPEN   = 140;   // physical OPEN (releases the cone)
const int RIGHT_CLOSED = 20;
const int RIGHT_OPEN  = 140;
const uint32_t STEP_DELAY_MS = 10;    // ms per 1-degree step
int leftPos  = LEFT_CLOSED;
int rightPos = RIGHT_CLOSED;

// — BMP388

—
BMP388_DEV bmp388;
float bmpTemp = 0, bmpPressure = 0, bmpAltitude = 0;
const float SEA_LEVEL_HPA = 1013.25f;    // nominal – only relative change matters
here

float baselineAlt = 0;
bool  deployTriggered = false;
bool  deployed       = false;
uint32_t deployStepTimer = 0;

float readAltitude() {
    return 44330.0f * (1.0f - pow(bmpPressure / SEA_LEVEL_HPA, 1.0f / 5.255f));
}

// Open both servos once, then hold open (non-blocking)
void updateDeploy() {
    if (!deployTriggered || deployed) return;
    uint32_t now = millis();
    if (now - deployStepTimer >= STEP_DELAY_MS) {
        deployStepTimer = now;
        bool moving = false;
        // step each servo toward its OPEN angle, regardless of numeric direction
```

```

    if (leftPos != LEFT_OPEN) { leftPos += (leftPos < LEFT_OPEN) ? 1 : -1;
servoLeft.write(leftPos); moving = true; }
    if (rightPos != RIGHT_OPEN) { rightPos += (rightPos < RIGHT_OPEN) ? 1 : -1;
servoRight.write(rightPos); moving = true; }
    if (!moving) { deployed = true; Serial.println(F(">>> Servos fully OPEN")); }
}
}

```

```

void setup() {
  delay(1000);
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(115200);
  while (!Serial && millis() < 3000);

  servoLeft.attach(PIN_LEFT);
  servoRight.attach(PIN_RIGHT);
  servoLeft.write(LEFT_CLOSED);
  servoRight.write(RIGHT_CLOSED);
  leftPos = LEFT_CLOSED; rightPos = RIGHT_CLOSED;

  Wire.begin();
  Wire.setClock(400000);
  delay(250);

  Serial.print(F("BMP388 init... "));
  if (!bmp388.begin(0x77)) {
    Serial.println(F("FAILED - halting."));
    while (1) { digitalWrite(LED_BUILTIN, HIGH); delay(100);
digitalWrite(LED_BUILTIN, LOW); delay(100); }
  }
  Serial.println(F("OK"));
  bmp388.setTimeStandby(TIME_STANDBY_40MS);
  bmp388.setTempOversampling(OVERSAMPLING_X2);
  bmp388.setPresOversampling(OVERSAMPLING_X16);
  bmp388.setIIRFilter(IIR_FILTER_8);
  bmp388.startNormalConversion();

  // Let the sensor + IIR filter settle BEFORE capturing the baseline, or the
  // reference is taken during the settling transient and reads tens of cm off.
  Serial.println(F("Warming up sensor (filter settling)..."));
  uint32_t warmEnd = millis() + 4000; // ~4 s of throwaway readings
  while (millis() < warmEnd) { bmp388.getMeasurements(bmpTemp, bmpPressure,
bmpAltitude); }

  Serial.println(F("Setting baseline - hold still..."));
  float sum = 0; int n = 0;
  while (n < BASELINE_SAMPLES) {
    if (bmp388.getMeasurements(bmpTemp, bmpPressure, bmpAltitude)) { sum +=
readAltitude(); n++; }
  }
  baselineAlt = sum / BASELINE_SAMPLES;
  Serial.print(F("Baseline set. Lower the rocket >"));

```

```

Serial.print(DROP_TRIGGER_M * 100.0f, 0);
Serial.println(F(" cm and HOLD to trigger.));

digitalWrite(LED_BUILTIN, HIGH);    // solid = armed, waiting for drop
}

void loop() {
  if (bmp388.getMeasurements(bmpTemp, bmpPressure, bmpAltitude)) {
    float alt = readAltitude();
    float drop = baselineAlt - alt;    // positive = it went DOWN

    if (!deployTriggered && drop >= DROP_TRIGGER_M) {
      deployTriggered = true;
      deployStepTimer = millis();
      Serial.print(F(">>> TRIGGERED at drop = "));
      Serial.print(drop * 100.0f, 1);
      Serial.println(F(" cm - opening servos"));
    }

    Serial.print(F("drop = "));
    Serial.print(drop * 100.0f, 1);
    Serial.print(F(" cm   armed="));
    Serial.print(deployTriggered ? 0 : 1);
    Serial.print(F(" open="));
    Serial.println(deployed ? 1 : 0);
  }

  // Diagnostic: solid LED = armed/waiting, blinking LED = trigger has fired.
  // Lets you see whether the TRIGGER fired independently of the servos.
  if (deployTriggered) digitalWrite(LED_BUILTIN, (millis() / 150) % 2);
  else                    digitalWrite(LED_BUILTIN, HIGH);

  updateDeploy();
  delay(20);
}

```

B.3 rocket_sim.py (1D RK4 trajectory simulation)

```

"""
1D point-mass rocket ascent simulation (RK4)
=====
Models a vertically launched rocket as a single point mass under three forces:
* Thrust T(t) -- imported from a RASP .eng thrust-curve file
* Gravity -m*g
* Drag      -0.5 * rho * Cd * A * v*|v|
Propellant mass is burned off in proportion to delivered impulse, so the mass
decreases during the burn. Equations of motion are integrated with classical
4th-order Runge-Kutta (RK4) at a fixed time step.

Outputs: rail-exit velocity, max velocity, and apogee, plus altitude/velocity
vs time plots. This is the "ideal straight-flight" model -- real apogee will be
lower if the rocket is not flying straight (see flight LOG030).
"""

```

```
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
```

```
# ----- RASP (.eng) thrust-curve parser
```

```
def load_eng(path):
    header, times, thrusts = None, [], []
    for line in open(path):
        s = line.strip()
        if not s or s.startswith(";"):
            continue
        p = s.split()
        if header is None:
            # first data line = header row
            header = dict(name=p[0], diam_mm=float(p[1]), len_mm=float(p[2]),
                          delays=p[3], prop_kg=float(p[4]), total_kg=float(p[5]),
                          maker=" ".join(p[6:]))
        else:
            times.append(float(p[0])); thrusts.append(float(p[1]))
    return header, np.array(times), np.array(thrusts)
```

```
class Motor:
```

```
    def __init__(self, path):
        self.h, self.t, self.F = load_eng(path)
        self.burn = self.t[-1]
        self.total_impulse = np.trapz(self.F, self.t) # N·s
        self.prop_kg = self.h["prop_kg"]
        # cumulative impulse vs time, for mass depletion
        self.cumimp = np.concatenate([[0.0],
                                       np.cumsum((self.F[1:] + self.F[:-1]) / 2 *
np.diff(self.t))])
```

```
    def thrust(self, t):
        return float(np.interp(t, self.t, self.F, left=0.0, right=0.0)) if 0 <= t
<= self.burn else 0.0
```

```
    def prop_remaining(self, t):
        if t <= 0:
            return self.prop_kg
        if t >= self.burn:
            return 0.0
        imp = np.interp(t, self.t, self.cumimp)
        return self.prop_kg * (1.0 - imp / self.total_impulse)
```

```
# ----- vehicle & environment
```

```
G = 9.80665
RHO = 1.20 # air density (kg/m³) ~ constant, low alt
```

```
class Rocket:
```

```
    def __init__(self, dry_mass, Cd, diameter_m, motor, rail_len=1.5):
```



```

        self.dry, self.Cd = dry_mass, Cd
        self.A = np.pi * (diameter_m / 2) ** 2
        self.motor, self.rail = motor, rail_len
    def mass(self, t):
        return self.dry + self.motor.prop_remaining(t)

def deriv(state, t, rk):
    y, v = state
    m = rk.mass(t)
    T = rk.motor.thrust(t)
    drag = 0.5 * RHO * rk.Cd * rk.A * v * abs(v)
    a = (T - drag) / m - G
    return np.array([v, a])

def simulate(rk, dt=0.005, tmax=60.0):
    state = np.array([0.0, 0.0]); t = 0.0
    T, Y, V = [0.0], [0.0], [0.0]
    rail_v = None
    while t < tmax:
        k1 = deriv(state, t, rk)
        k2 = deriv(state + .5*dt*k1, t + .5*dt, rk)
        k3 = deriv(state + .5*dt*k2, t + .5*dt, rk)
        k4 = deriv(state + dt*k3, t + dt, rk)
        state = state + dt/6*(k1 + 2*k2 + 2*k3 + k4)
        t += dt
        if state[0] < 0: # sitting on pad before liftoff
            state[:] = 0.0
        T.append(t); Y.append(state[0]); V.append(state[1])
        if rail_v is None and state[0] >= rk.rail:
            rail_v = state[1]
        if t > rk.motor.burn and state[1] <= 0: # reached apogee -> stop
            break
    return np.array(T), np.array(Y), np.array(V), rail_v

# ----- run it -----

if __name__ == "__main__":
    motor = Motor("Quest_F41W.eng")
    rk = Rocket(dry_mass=0.90, # kg, everything except propellant
               Cd=0.55, # drag coefficient (0.4-0.75 typical)
               diameter_m=0.076, # airframe diameter (m)
               motor=motor,
               rail_len=1.5) # launch rail length (m)

    T, Y, V, rail_v = simulate(rk)
    apogee = Y.max(); apogee_t = T[Y.argmax()]

    print(f"Motor {motor.h['name']}: total impulse {motor.total_impulse:.1f} N·s,
    burn {motor.burn:.2f} s")
    print(f"Liftoff mass: {rk.mass(0)*1000:.0f} g")
    print(f"Thrust/weight (peak): {motor.F.max()/(rk.mass(0)*G):.1f}")
    print(f"Rail-exit velocity: {rail_v:.1f} m/s (want >= ~15 m/s)")

```

```
print(f"Max velocity:      {V.max():.1f} m/s")
print(f"Apogee (ideal):   {apogee:.1f} m at t={apogee_t:.2f} s")

fig, ax = plt.subplots(1, 2, figsize=(11, 4))
ax[0].plot(T, Y); ax[0].set(xlabel="time (s)", ylabel="altitude (m)",
title="Altitude"); ax[0].grid(alpha=.3)
ax[1].plot(T, V); ax[1].set(xlabel="time (s)", ylabel="velocity (m/s)",
title="Velocity"); ax[1].grid(alpha=.3)
plt.tight_layout(); plt.savefig("sim_ascent.png", dpi=130)
print("saved sim_ascent.png")
```