Autonomous Parafoil Glider

EECS 473
Fall 2025

Matteo Caporuscio, Maxwell Kenny, Thomas Tran, Shawn Trujillo, Anthony Young

Introduction and Overview

Project Goal and Motivation

The central objective of this project was to design, build, and validate an autonomous parafoil guidance system capable of navigating a payload to a designated GPS coordinate. This technology addresses the problem of recovering high-altitude payloads for the scientific community.

According to the World Meteorological Organization (WMO), approximately 900 to 1,000 radiosondes are launched daily worldwide, yet up to 80% are never recovered[1]. For higher-value assets, such as NASA's scientific balloon missions, recovery often involves costly and complex naval operations or expeditions into remote locations[2]. While commercial precision airdrop systems like JPADS exist, they are expensive and scaled for heavy military cargo, making them unusable for university research or small-scale atmospheric data collection.

Our project produced an Airborne Guidance Unit (AGU), a low-cost device designed to autonomously steer a 5kg payload during descent. By utilizing a custom-built Printed Circuit Board and a custom Guidance, Navigation, and Control (GNC) algorithm, we aimed to make precision landing technology accessible to everyone.

Demonstration and Achievement

At the COE Design Expo, the system was demonstrated through a video demonstration of payload drops as well as a "Hardware-in-the-Loop" simulation for the demo. While logistical constraints and safety regulations prevented a live high-altitude drop test at the BBB, we validated the system by feeding the AGU live GPS and IMU data, mimicking the actions of a moving parafoil on foot. The system successfully calculated necessary heading corrections and actuated the steering servos in real-time, demonstrating that the autonomous control loop was functional.

Moving forward to actual demonstration on December 6, 2025, our team successfully showcased the autonomous parafoil in action, guiding itself to its target coordinate. From a starting altitude of ~75 meters (~246 feet), the parafoil landed within 5 meters of the target coordinate, well within our target landing zone of 15m.

We achieved the majority of our Milestone 1 and 2 goals, including the development of a custom PCB, the implementation of our 7-state Extended Kalman Filter (EKF), and reliable long-range telemetry.

Description of Project

Design Specifications / Pathways

To recount the motivations put forth by our original project proposal, we aimed to design a device that autonomously guides a parafoil attached to a payload that can be used by the small-scale scientific community that requires recoverable equipment. In order to demonstrate that the system is viable for larger payloads (which, subsequently, means larger parafoils), we aimed to have a maximum weight of 5kg with a descent of 1.5m/s. As of writing, the physical specifications we have achieved are as follows: Our structure, including the housing and servo motors, has a total weight of just under 1kg. In our tests, we've reached a top velocity of 10 m/s, equating to a max travel distance of roughly 180 meters when dropped from our maximum height of 75 meters.

System Concept

The solution consists of two primary subsystems:

1. **The Airborne Guidance Unit:** A PCB-based system mounted to the payload. It houses the sensors (GPS, IMU) and the microcontroller that executes the flight algorithm. It actuates the parafoil's brake lines using high-torque servo motors to turn the glider.
2. **The Ground Control Station:** A laptop-based interface using Python connected to an XBee radio. It allows the user to upload target coordinates, define Geofence "no-fly" zones, deploy payload or emergency deadspin, and monitor real-time telemetry (altitude, heading, velocity) during the flight.

Design Constraints & Requirements

The project was designed under constraints to ensure feasibility for university-level applications.

- **Weight/Payload Capacity:** The system needed to support a payload of up to 5kg while minimizing its own weight. We successfully met this constraint, with our final structure weight being approximately 1kg, leaving room for scientific instruments.
- **Unit Cost:** To be accessible, the device needed to be significantly cheaper than military alternatives.
- **Descent Rate:** The target descent rate was 1.5m/s to ensure a soft landing for sensitive equipment.
- **Time:** The project had to be completed within a single semester. As detailed in the "Schedule Review," this was our tightest constraint.

System Architecture

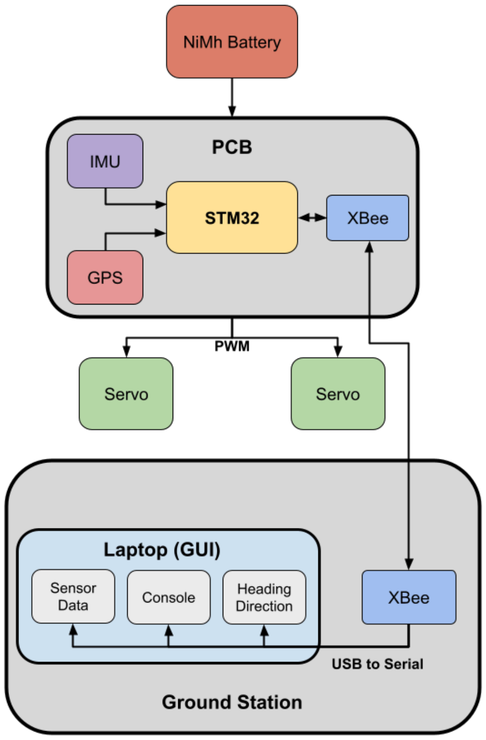


Figure 1: *System architecture and component diagram*

Device Overview:

Device	Task Summary
IMU	A 9-axis System in Package (SiP) which provides sensor data from an accelerometer, gyroscope, and magnetometer. This provides the high-frequency, drift-prone movement data for the EKF algorithm.
GPS	Provides highly accurate, but low-frequency position data to be fused with the IMU for the EKF. Also read using the UBX message format, allowing the collection of additional data like altitude.
Servos	Rotates paracord in order to change parafoil direction in response to STM32 PWM output.
XBee	Establishes communication between the PCB and USB plug-in breakout (for Ground Station) using the Zigbee protocol. Sensor data is periodically sent from the PCB with a FreeRTOS task.
STM32	The central processor that computes all our necessary tasks, such as sensor actuation, EKF algorithm, and packet transmission, through the use of FreeRTOS.

Table 1: *List of all devices and summaries of their tasks*

PCB Design & Hardware Architecture

The Airborne Guidance Unit (AGU) is built upon a custom 4-layer Printed Circuit Board (PCB) designed in Altium Designer. To ensure reliability during autonomous flight, the hardware architecture prioritizes signal integrity for the sensors and robust power distribution.

The PCB layer stackup uses a Ground-Signal-Power-Ground approach. By placing Ground planes on the top and bottom layers, we shield the sensitive inner signal traces from external electromagnetic interference. The two internal layers are fabricated with 2oz copper, doubling the standard thickness (1oz). This heavy copper design was chosen to handle the high current density required by the servo motors and to minimize the resistive heating and voltage drop across the internal power distribution network.

The power network was designed using a “Star Routing” topology. Rather than daisy-chaining power connections, the high-current traces for the servos and the sensitive logic traces (3.3V rail) radiate independently from the power source. This isolation prevents the voltage drops and ground bounce caused by the servos’ stall currents from resetting the microcontroller or adding noise to sensor measurements.

- **Regulation:** We utilized the LT1764AET-3.3, a Low-Dropout (LDO) regulator. Unlike switching regulators, which can introduce noise, the LDO provides a clean, low-noise 3.3V supply essential for the GPS module’s RF front-end performance.
- **Protection & Visualization:** The input stage uses a Schottky diode for reverse polarity protection. To aid in field diagnostics, we integrated status LEDs for every power rail. This allows for immediate visual confirmation of power health.

Recognizing the challenges of debugging embedded hardware in the field, we integrated specific hardware features for the system:

- **Hardware BOOT0 Switch:** A jumper switch was added to our PCB. This allows us to manually force the microcontroller into DFU (Device Firmware Update) mode, ensuring we can recover the device via UART.
- **Test Points:** Dedicated test points were placed on all sensor buses, such as SPI or UART, to allow us to attach logic analyzers and see if our modules were working properly.
- **Reset Switch:** A button was implemented on our PCB to allow us to reset the MCU when needed.

The GPS subsystem features a custom bias-tee circuit, comprising a 27nH inductor and a 100nF capacitor network. This injects DC power into the active antenna cable while isolating the RF signal path entering the NEO-M9N module to power the active antenna and optimize satellite reception.

Recognizing the difficulty in soldering and circuit design, and wanting to ensure that we could use our PCB regardless of the status of the surface mount sensors, we added additional 3-pin headers that could be used in tandem with two-pin jumpers to control where UART signals from the MCU were sent. We could control, through the placement of the jumpers, whether or not the

UART signals were directed to another set of header pins that facilitated off-board sensors, or to the onboard SMD sensors.

Firmware Architecture & RTOS Implementation

The Airborne Guidance Unit (AGU) operates on a firmware architecture built around the FreeRTOS real-time operating system on an STM32H7 microcontroller. To manage the asynchronous nature of our sensors and ensure autonomous flight control, we implemented a prioritized task scheduler. The system architecture is divided into three crucial threads, a High-Priority Sensor Acquisition task, a DMA-Driven Navigation Parsing task, and a Telemetry/Control task.

The most critical task, Startbno085Task, polls the IMU at a 100Hz interval. We built upon a custom SPI driver to handle the BNO085's Sensor Hub Transport Protocol (SHTP), which requires complex fragmentation and reassembly of the sensor reports[3]. This driver also manages the boot sequence, ensuring the IMU is calibrated before the flight control loop is enabled.

The GPS data is handled by the readGPS task, which utilizes Direct Memory Access (DMA) to capture high-speed UART streams from the NEO-M9N module without CPU intervention. Unlike standard NMEA parsing, which can be slow, we implemented a binary UBX protocol parser. This reduces overhead and allows us to extract precise velocity vectors and position covariance data required by the Extended Kalman Filter.

Lower priority tasks manage the XBee telemetry link and servo PWM generation. By separating these from the sensor loops, we make sure the heavy communication logging does not introduce jitter into the flight control algorithms.

State Estimation Algorithm (Extended Kalman Filter)

To achieve autonomous navigation, the system relies on a precise estimation of its position and heading. We implemented a 7-state Extended Kalman Filter (EKF) to fuse the high-frequency inertial data with the lower-frequency GPS updates.

State Vector Definition

The filter tracks a 7-dimensional state vector x, defined as:

$$x = [POS_N, POS_E, POS_D, VEL_N, VEL_E, VEL_D, \psi_{bias}]^T$$

The choice of these 7 states is driven by the physics of the parafoil and the sensors available.

- **Position:** Represents the payload's location in the NED (North-East-Down) coordinate frame. We use "Down" for altitude to align with standard aerospace conventions (where positive Z points towards the center of the Earth).

- **Velocity:** Tracks the speed of the payload in meters per second along the North, East, and Down axes. This is crucial since GPS updates are slow. The velocity state allows us to predict position between GPS logs.
- **Heading Bias:** This is the most critical state for accurate navigation. The system uses the BNO085's Game Rotation Vector (Accel + Gyro), which provides a stable relative heading but has no reference to magnetic North. This state estimates the offset between the arbitrary IMU frame and True North derived from the GPS ground track, effectively aligning the gyroscope to the real world during flight.

Prediction Step

Running at 100Hz, the prediction step performs inertial integration. The BNO085 reports acceleration in the body frame, which the filter rotates into the World (NED) frame using live quaternion. We perform Euler integration to update the position and velocity states.

During this step, we also increase the values in the Error Covariance Matrix (**P**). This mathematically represents the growing uncertainty in our position estimate the longer we go without a GPS correction (accumulating "Process Noise").

Sequential Update Optimization

When a valid GPS packet arrives, the filter performs a correction. A standard EKF update requires inverting the Innovation Covariance Matrix (**S)**. For a 7-state system, this involves inverting a 7 x 7 matrix, an operation that can be computationally expensive for a microcontroller and easy to fail when calculating with floating-point math.

To solve this, we implemented a Sequential Update method. This technique treats the GPS data not as one large block, but as independent scalar measurements (North, East, Down, V_N, V_E, V_D).

1. **Scalar Update:** We process the North Position first. The filter calculates a scalar Kalman Gain (a ratio determining how much to trust the GPS vs. the IMU).
2. **State Correction:** The states are updated immediately based on that single value.
3. **Iteration:** We repeat this for East, Down, and Velocity.

This reduced the complexity from matrix inversion to simple division, reducing the computational load and allowing complex estimation logic to run within the timing constraints of the flight control loop.

Servo Control Algorithm

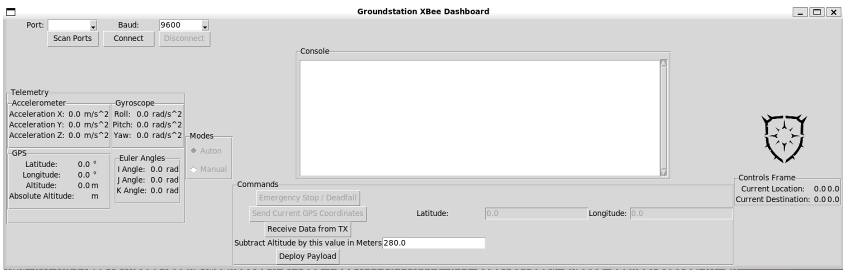
Alongside the Extended Kalman Filter, the control algorithm is running in its own task to update the two servo motors, which control the direction of the parafoil and guide it to its target. The servos on the AGU will rotate at the appropriate angle to pull the brake lines of the parafoil and control its heading direction.

This control task functions by taking in the fused sensor data from the EKF, which describes the current state, and comparing it to a target coordinate. By using trigonometry to calculate the angle of the target from the current position, then finding the difference from our current predicted heading, it calculates the amount the parafoil needs to turn. After finding this turn angle, the AGU knows which way and how much it needs to turn. A tunable, proportionate angle is applied to one of the servos to move the parafoil in that direction.

A +-20 degree deadzone is applied around the target angle to mitigate overshoot as the parafoil gets closer to the target. Also, within 160 and -160 from the target angle, the current servo command is locked until out of this zone to stop oscillation of the left and right servos.

Ground Station

The Ground Station (GS) is the part of our software that aims to communicate with the AGU by receiving sensor data and transmitting control input via the Xbee Boards. In this scenario, the GS runs through a laptop and is connected to an XBee Board breakout in order to use Serial Communication with the XBee that is soldered onto our PCB. At the onset of our project's development, the first draft of the GS would capture the sensor values on the PCB and send control inputs through our Graphical User Interface (GUI). These inputs would tell the AGU to either be in Manual Mode (manually turn the parafoil left and right) or Auton Mode (utilize the autonomous guidance algorithm). Under the hood, each input sends a character over Serial Communication to trigger a specific task in the AGU RTOS. Similarly, a buffer is periodically sent from the AGU to the GS containing the IMU and GPS data to be parsed and displayed.



Port:
Baud: 9600

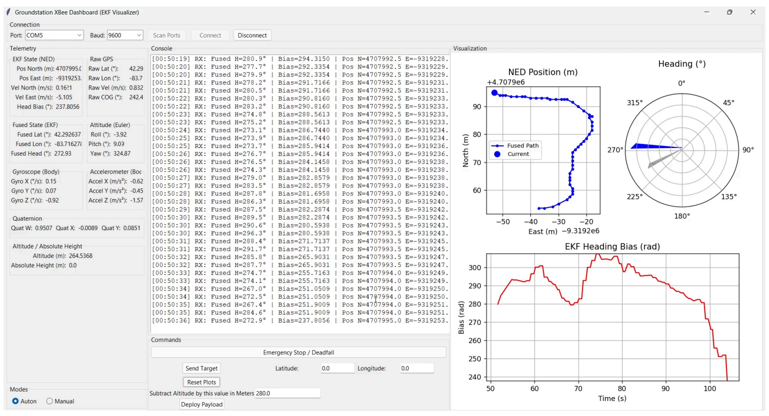
Telemetry
Accelerometer
Acceleration X: 0.0 m/s^2
Acceleration Y: 0.0 m/s^2
Acceleration Z: 0.0 m/s^2
Gyroscope
Roll: 0.0 rad/s^2
Pitch: 0.0 rad/s^2
Yaw: 0.0 rad/s^2
GPS
Latitude: 0.0 °
Longitude: 0.0 °
Altitude: 0.0 m
Absolute Altitude: m
Euler Angles
I Angle: 0.0 rad
J Angle: 0.0 rad
K Angle: 0.0 rad

Latitude: 0.0 Longitude: 0.0
Subtract Altitude by this value in Meters 280.0

Controls Frame
Current Location: 0.0 0.0
Current Destination: 0.0 0.0

Figure 2: *GUI for Groundstation Software to interact with XBee Breakout*

As the project progressed, several changes were made to the GUI in order to provide better interaction with the system. First, we needed a way to display the payload's flight path, as well as heading direction based on the fused data between the EKF algorithm and GPS. Second, we realized that we would be wise to offload some of the compute within the AGU RTOS where we can, and that this could be done when making additional data measurements. For example, To measure our payload's true height above the ground, we created an additional metric called Absolute Altitude that would be calculated within the GUI code.



EKF State (NED)
Pos North (m): 4707995.0
Pos East (m): -9319253.
Vel North (m/s): 0.1621
Vel East (m/s): -5.105
Head Bias (°): 237.8056

Raw GPS
Raw Lat (°): 42.29
Raw Lon (°): -83.7
Raw Vel (m/s): 0.832
Raw COG (°): 242.4

Fused State (EKF)
Fused Lat (°): 42.292637
Fused Lon (°): -83.71627
Fused Head (°): 272.93

Attitude (Euler)
Roll (°): -3.92
Pitch (°): 9.09
Yaw (°): 324.87

Gyroscope (Body)
Gyro X (°/s): 0.15
Gyro Y (°/s): 0.07
Gyro Z (°/s): -0.92

Accelerometer (Boc
Accel X (m/s²): -0.62
Accel Y (m/s²): -0.45
Accel Z (m/s²): -1.57

Quaternion
Quat W: 0.9507 Quat X: -0.0089 Quat Y: 0.0851

Altitude / Absolute Height
Altitude (m): 264.5368
Absolute Height (m): 0.0

```
[00:50:19] RX: Fused H=280.9° | Bias=294.3150 | Pos N=4707992.5 E=-9319228.
[00:50:20] RX: Fused H=277.7° | Bias=292.3354 | Pos N=4707992.5 E=-9319229.
[00:50:20] RX: Fused H=279.9° | Bias=292.3354 | Pos N=4707992.5 E=-9319229.
[00:50:21] RX: Fused H=278.2° | Bias=291.7166 | Pos N=4707992.5 E=-9319231.
[00:50:21] RX: Fused H=280.5° | Bias=291.7166 | Pos N=4707992.5 E=-9319231.
[00:50:22] RX: Fused H=280.3° | Bias=290.8160 | Pos N=4707992.5 E=-9319233.
[00:50:22] RX: Fused H=283.2° | Bias=290.8160 | Pos N=4707992.5 E=-9319233.
[00:50:23] RX: Fused H=274.8° | Bias=288.5613 | Pos N=4707992.5 E=-9319233.
[00:50:23] RX: Fused H=275.2° | Bias=288.5613 | Pos N=4707992.5 E=-9319233.
[00:50:24] RX: Fused H=273.1° | Bias=286.7440 | Pos N=4707993.0 E=-9319234.
[00:50:25] RX: Fused H=273.9° | Bias=286.7440 | Pos N=4707993.0 E=-9319234.
[00:50:25] RX: Fused H=273.7° | Bias=285.9414 | Pos N=4707993.0 E=-9319236.
[00:50:26] RX: Fused H=276.7° | Bias=285.9414 | Pos N=4707993.0 E=-9319236.
[00:50:26] RX: Fused H=276.5° | Bias=284.1458 | Pos N=4707993.0 E=-9319238.
[00:50:26] RX: Fused H=274.3° | Bias=284.1458 | Pos N=4707993.0 E=-9319238.
[00:50:27] RX: Fused H=279.0° | Bias=282.8579 | Pos N=4707993.0 E=-9319238.
[00:50:27] RX: Fused H=283.5° | Bias=282.8579 | Pos N=4707993.0 E=-9319238.
[00:50:28] RX: Fused H=287.8° | Bias=281.6958 | Pos N=4707993.0 E=-9319240.
[00:50:28] RX: Fused H=286.3° | Bias=281.6958 | Pos N=4707993.0 E=-9319240.
[00:50:29] RX: Fused H=287.5° | Bias=282.2874 | Pos N=4707993.5 E=-9319242.
[00:50:30] RX: Fused H=289.5° | Bias=282.2874 | Pos N=4707993.5 E=-9319242.
[00:50:30] RX: Fused H=290.6° | Bias=280.5938 | Pos N=4707993.5 E=-9319243.
[00:50:30] RX: Fused H=296.3° | Bias=280.5938 | Pos N=4707993.5 E=-9319243.
[00:50:31] RX: Fused H=288.4° | Bias=271.7137 | Pos N=4707993.5 E=-9319245.
[00:50:31] RX: Fused H=291.7° | Bias=271.7137 | Pos N=4707993.5 E=-9319245.
[00:50:32] RX: Fused H=285.8° | Bias=265.9031 | Pos N=4707993.5 E=-9319247.
[00:50:32] RX: Fused H=287.7° | Bias=265.9031 | Pos N=4707993.5 E=-9319247.
[00:50:33] RX: Fused H=274.7° | Bias=255.7163 | Pos N=4707994.0 E=-9319249.
[00:50:33] RX: Fused H=274.1° | Bias=255.7163 | Pos N=4707994.0 E=-9319249.
[00:50:34] RX: Fused H=267.0° | Bias=251.0509 | Pos N=4707994.0 E=-9319250.
[00:50:34] RX: Fused H=272.5° | Bias=251.0509 | Pos N=4707994.0 E=-9319250.
[00:50:35] RX: Fused H=267.4° | Bias=251.9009 | Pos N=4707994.0 E=-9319251.
[00:50:35] RX: Fused H=284.6° | Bias=251.9009 | Pos N=4707994.0 E=-9319251.
[00:50:36] RX: Fused H=272.9° | Bias=237.8056 | Pos N=4707995.0 E=-9319253.
```

Latitude: 0.0 Longitude: 0.0
Subtract Altitude by this value in Meters 280.0

Figure 3: *Updated GUI implementing Live Map and Heading for the system*

Milestones and Schedule

Milestones Overview

The milestones in our project were meant to serve as development checkpoints for major tasks in our project. These milestones each came with two major goals we were required to meet as well as some smaller goals that would be favorable to meet. Additionally, in our proposal, we outlined our general goals for the project relating to desired features and functionality in Appendix A. All tasks we proposed within our milestones and appendix are listed below:

Milestone 1 - Prototyping
Prototype is able to receive GPS information and control servos/motors accordingly.
Information is communicated between ground station and payload via XBee modules. Range is tested across campus. EECS -> Duderstadt is good. FRMCB -> Pierpont would be fantastic
Servo control driver is written, tested, and working (motors move on command)
GPS, motor, XBee interface modules are written and actively being tested
Devices have been integrated with at least one other device.
Milestone 2 - PCB Design
PCB assembled
Payload lands within our acceptable radius of ground station GPS coordinates. $$radius = \sqrt{(\frac{1}{10} \cdot height)^2 + (wind\ speed \cdot \sqrt{\frac{distance}{9.81}})^2}$$
Guidance algorithm is completely written and tested
Full device integration. All parts are able to work together through the PCB

Table 2: *Goals for Milestones One and Two from our proposal*

Design Criteria	Importance	Will/Expect/ Stretch	Final Status
Parafoil safely lands at desired landing coordinates (within a ~20m radius) starting from various heights reaching ~300m	Fundamental	Will	Complete
Allow for manual control of the parafoil from ground station	Important	Will	Complete
Base station can reliably change parafoil's destination coordinates	Important	Expect	Complete
Parafoil lands at closest possible point to unreachable destination coordinates	Important	Expect	Complete
Parafoil's location, direction, and speed monitored from base station	Optional	Expect	Complete
Allow for the parafoil to be automatically deployable when it reaches a certain altitude/speed	Optional	Stretch	Incomplete
Parafoil avoids collisions with large objects/structures	Optional	Stretch	Done but untested

Table 3: *General project goal list from proposal Appendix A*

Schedule Review

Our team initially adhered closely to the schedule proposed during the prototyping phase (Milestone 1). By the first milestone, we had successfully integrated the GPS, XBee, and IMU modules using breakout boards, demonstrating reliable scheduling and communication between the ground station and the payload. We met all primary deliverables for Milestone 1 on time, including the development of the servo control driver and the validation of wireless range across North Campus.

However, the project schedule faced roadblocks during the second half of the semester (Milestone 2). The primary cause of delay was the PCB design and manufacturing timeline. We chose to spend additional time refining the PCB layout to ensure we had signal integrity and proper power distribution, which resulted in the board arriving later than originally planned. This delay rippled into the integration phase, where the transition from prototype to the custom PCB was rushed. Debugging hardware issues, such as soldering defects and errors with our traces, consumed time that was originally allocated for field testing and tuning the guidance algorithm. Despite these setbacks, we mitigated risks by maintaining a parallel development path using the breakout board prototype. This allowed us to continue testing the firmware and EKF algorithm even while the PCB was in transit. As a result, while hardware integration was delayed, core functionality goals were being met.

We also encountered logistical challenges that further delayed our testing. Specifically, our team did not secure a drone for drop-testing in advance. We relied on informal agreements for a test platform, which ultimately fell through, leaving us without a way to perform high-altitude tests at the critical testing phase. Consequently, we were unable to validate the mechanical deployment of the parafoil in a realistic flight environment until very late into the semester.

Project Reflections and Lessons Learned

Reflecting on our collective experience throughout the project, there are several things that we wished to have done differently in order to have a more streamlined, successful result. These include how we would have changed our approach to **administrative, research, and development** tasks within the project. In short, we would have liked to tell our past selves to research *all* the constraints and capabilities of our components, and research better structural designs and manufacturing processes. Not all of these changes in approach would have been made without going through our project development process, but nevertheless, they would have prevented us from performing unnecessary work and losing time.

Administrative Tasks

One of the most important tasks that we needed to undertake was the task of requesting administrative permission to demonstrate our project through the university. To accomplish this, we contacted the building managers of North Campus buildings in order to get permission to potentially demonstrate our project in their main areas with a drop-test, so long as we did the test safely. There were two unexpected events that affected these original demonstration plans. Firstly, in order to demonstrate one of the most important features on the payload - our AGU's autonomous control system - we required a GPS signal, which has only worked outdoors in testing. Secondly, we have discovered in testing that our payload's parafoil only successfully deploys after falling a minimum distance of 20 to 40 meters. This distance is much taller than most of the buildings on North Campus, meaning a drop test in this area would not be safe for demonstration. As an alternative, we've pivoted to demonstrating our project outdoors at a location permitted to us by project supporter Kenny Farmer. In general, these unforeseen developments demonstrate the experimental nature of this project and could have been discovered earlier through better research on our component capabilities.

Researching Tasks

Research, as vital as it is, has presented itself as a double-edged sword in terms of pivoting our designs and plans when new developments are made. One prominent case of this is how we planned to calculate the altitude of our parafoil during flight. In our project proposal, our plan to tackle this issue was to use the BMP581 sensor component, which would measure temperature and air pressure in order to determine the height above Earth's surface. This method, we theorized, would also require tuning, especially as the semester progressed into colder outdoor temperatures. However, before we ordered this component, we learned that the GPS message would contain the Altitude of the receiver in the AGU. It would only need to be parsed and displayed on the GS in order to be read. This method was favorable, and even influenced us to remove the BMP581 sensor from our PCB design altogether. In retrospect, we would have preferred for easier methods like these to be discovered well in advance for our project, as they would have saved time in budgeting, development and design.

Development Tasks

Developing the physical system was a roller coaster of great successes and heartbreaking failures. The original structure for our system was a unique design employing a cylindrical main compartment that supported customizations through the use of addable layers. The entire design (lid, main compartment, and bottom) were held together through the mechanical strength of 3D printed threaded fastening. This worked well for holding all of our components together - however - mechanical challenges were encountered when we first attempted our first full system assembly including attaching it to a drone. The twisting of the lid onto the main compartment would result in twisting of the parafoil and attachment lines that was not considered during design. This necessitated a full structural redesign and reprint and essentially invalidated almost two months worth of work. This had a waterfall effect on our entire design process as the delay in having a working structure delayed our ability to drop our system and get data on how the parafoil would release, what the structure would do during flight, and how the structure would survive impact. Considering the twisting earlier in the project could have moved the starting of our testing campaign up two to three weeks.

Beyond just the structure, we faced significant difficulties when it came to bringing up our PCBs. Most of us had experience soldiering SMD components through the MESH lab tutorial - but that is the ideal environment, one that does not prepare you for the difficulties of soldering a 144 pin STM to a PCB. We struggled to ensure that pins that small did not bridge and cause shorts. Beyond just the STM, on several boards that we assembled, the GPS module would mysteriously short without any indication of why. Viewing it through a thermal camera with three times the expected current revealed no information about where the short could be occurring. In hindsight, we probably should have picked a smaller MCU as we only used about fifteen pins. This would have vastly improved our ability to manufacture the boards and likely would have caused less issues during assembly.

Team Member Contributions

Team Member	Contribution Description	Effort
Shawn Trujillo	- Developed, integrated and tested the servos with other sensors. - Worked on configuring the GPS to provide geofencing, heading, and ground speed information. Then tested to ensure information was accurate and controls reacted appropriately. (With Matteo) - Helped with writing FreeRTOS firmware and integration between components. (With Matteo) - Worked on and tested the control algorithm for servos to guide the parafoil to target coordinate both on foot and in the air. (With Matteo) - Located and acquired a drone that met the requirements for our project (able to lift more weight than itself). - Attended all drop tests and debugged code, structure, parafoil, and mother nature problems that came up.	20%
Matteo Caporuscio	- Developed and tested interface for Xbee wireless modules for communication between groundstation and MCU - Helped with writing FreeRTOS firmware and integration between components - Worked on control algorithm for servos to guide the parafoil to target coordinate - Integrated GUI with wireless commands to and from STM32 - Ported and debugged software from Nucleo Board to PCB - Worked on configuring firmware for Xbee and GPS modules on PCB	20%
Anthony Young	- Designed, developed and maintained Graphical User Interface (GUI) for the Ground Station. Also added in necessary features to interface with other components on the PCB, such as the XBee Radio. - Aided in administrative tasks such as contacting building managers, keeping meeting notes, and other general organization tasks. - Helped assemble additional PCBs for use in our project.	15%
Thomas Tran	- Led design lifecycle of the PCB in Altium Designer. This included component selection, schematic capture, and initial layout optimized for signal integrity and power isolation.	23%

	- Supported hand-soldering, board bring-up, and electrical verification of final hardware. - Developed custom SPI driver for the BNO085 IMU, managing SHTP protocol for high-frequency data acquisition. - Selected binary UBX protocol for the GPS to maximize throughput and wrote DMA-driven parser to extract precise velocity vectors. - Architected the FreeRTOS-based firmware and integration, designing prioritized task scheduler. Ported and tuned the 7-state Extended Kalman Filter, implementing a Sequential Update optimization to enable complex sensor fusion on the MCU.	
Maxwell Kenny	- Designed, developed, and tested multiple iterations of the structure to support repeated testing and reliably hold and release parafoil during drops. - Research and source several parafoils to ensure aerodynamic properties met the requirements of the project. Also responsible for researching and developing the folding technique used for reliable parafoil deployment. - Responsible for hand-soldering, board bring-up and supported electrical verification of final hardware. - Designed routing of the entire PCB to ensure signal and power integrity. - Designed and implemented a NEO-M9N GPS interface to read NMEA strings over I2C for quick access of Latitude, Longitude, and Altitude. - Handled ordering and reimbursement for the entire project. Also responsible for finding suitable testing locations for the duration of the project.	22%

Table 4: *Team member contributions*

Estimated Manufacturing Costs

Since this product fits a very niche market with the aim of being reusable, we decided a reasonable number to manufacture would be roughly 100 units for the initial hypothetical production. We used JLCPCB's quoting tool to get an estimate of what it would cost to have them manufacture and assemble all 100 PCBs for us. JLC has enough of all but one component to fully assemble the 100 boards. Unfortunately, the only missing component is the NEO-M9N GPS module - the most expensive component on the board by far. JLC only has 33 of them in stock, meaning that we would need to source the remaining 67 and add them to the board ourselves. The quoted cost for the manufacturing and assembly is $1366.30. Ordering from Digikey, we could get the remaining 67 GPS modules for $1313.87. This would mean to get 100 boards fully manufactured and assembled, it would cost $2,680.17 which translates to a cost of $26.81 for each PCB. Considering that this system would be piloting hardware that is sometimes worth tens of thousands of dollars safely to a retrievable location, we would argue that the price for these PCBs far exceeds our expectations. Even at the low end, a typical radiosonde (weather tracking device) costs roughly $200, ten times what the PCB for our device would cost.

Extending this to our full system, we must consider more than just the cost of the PCB. We must factor in the cost of the servos, the cost of the parafoil, and the cost of the structure in PLA. Each completed Airborne system requires 2 high torque servo motors. We used the Hitec HS-788HB servo motors, which are currently selling for $56.99 per unit. This adds an additional $11,398.00 to our running total. The 62” dual line sport parafoil that we used is currently $29.99 per unit adding an additional $5998.00 to our total. Finally, the structure uses 0.175m of PLA to fully print. Assuming that we would use the same filament that we used in our current design, it would cost roughly $172.40 to print out all 100 units.
Those additions bring the total cost of manufacturing 100 units to $20,248.57. If we were going to go commercial with this product, assuming we want a profit of a minimum of 30%, we would need to sell each unit at a cost of roughly $260. This is still easily worth it for larger scientific missions that have budgets in the thousands of dollars that run the risk of losing their entire systems or having to recover them without our system. Additionally, even though the initial cost of buying our autonomous parafoil would outweigh the cost of a single Radiosonde, the benefits of being able to reuse that single radiosonde rather than lose it or throw it away would immediately outweigh the steep investment after the first successful flight and recovery.

Budget & Purchase List

Since our proposal, the inventory and budget for our project has changed substantially. The components have been changed, and the number of replacement parts has been reduced or replaced several times in order to meet changing requirements with our system. As a result, the total cost accumulated for our project has changed significantly from what we previously estimated. Below is a complete breakdown of purchases with a short description of items in each order. A complete budget can be found here: Expenses
Total Budget: $250/person x 5 people = **$1250 budget**

Vendor	Description	Cost	Cost Shipping	Cost Total
Sparkfun	- XBee wireless kit - GPS Breakout board	$168.44	$0	$168.44
Adafruit	- IMU Breakout Board - High Torque Metal Gear Servo - Wifi Antenna for XBee	$118.99	$12.58	$131.57
ST	- Nucleo H7 Breakout Board	$29.23	$5.99	$35.22
Amazon	- High-speed servo motors - 1.4 m^2 Parafoil	$135.61	$0.00	$135.61

Digikey	- Passive components - Surface-mount components for PCB	$472.47	$21.2	$493.67
JLCPCB	- PCBs - Stencils	$129.4	$103.39	232.79
Total				$1,197.3

Table 5: *Budget*

References and Citations

Acknowledgements

We would like to extend a thank you to Infineon, the course sponsor, as well as the EECS 473 instructional staff, for their belief and invaluable support throughout the semester.

We would also like to extend our sincere gratitude to Dr. James Cutler and the Michigan Exploration Laboratory (MXL) for their mentorship and for providing access to research on previous autonomous guidance systems. Their documentation and design methodology for high-altitude payloads were instrumental to the architectural design of our AGU.

We also thank Jimmy Lowe (Vast) for his technical advice regarding testing and for sharing insights from his previous work on similar recovery systems. Additionally, we acknowledge Kenney Farmer at Belleville Airport for their cooperation regarding flight testing facilities, as well as Evelyn Alcomo for sewing the Drogue Chute that we used.

Interface Code

The following sections detail the low-level hardware abstraction layers (HAL) and API definitions used to interface the STM32H7 firmware with the Airborne Guidance Unit's peripherals.

System Configuration (main.h, FreeRTOSConfig.h)

Defines the global control logic states and RTOS priorities.

```
/* Global Control Modes */
typedef enum {
   MODE_GPS_AUTO,          // GPS-based navigation (default)
   MODE_MANUAL_FALLBACK,    // Manual control because no GPS
   MODE_MANUAL_OVERRIDE     // Manual override (stay manual even with GPS)
} control_mode_t;

/* Global State Variables */
```

```
extern volatile control_mode_t control_mode;
extern volatile uint8_t manual_servo_command;
extern volatile uint32_t last_manual_command_time;

/* RTOS Priority Configuration */
/* High priority for Sensor acquisition to ensure 100Hz stability */
#define configLIBRARY_MAX_SYSCALL_INTERRUPT_PRIORITY 5
#define configTICK_RATE_HZ ((TickType_t)1000)
#define configTOTAL_HEAP_SIZE                    ((size_t)16384*2)
```

IMU Sensor Interface (Sensor_Struct.h, Hardware_Init.h)

APIs for the BNO085 IMU, managing the Sensor Hub Transport Protocol (SHTP) over SPI.

```
/* Sensor Meta Structure - Holds handles and state for the IMU */
typedef struct sensor_meta {
  uint8_t number;
  SPI_HandleTypeDef *hspi;
  volatile bool has_reset;
  volatile uint8_t INTN_ready; // Interrupt ready flag

  // Data Containers
  quaternion_data quaternions;
  accelerometer_data accelerometer_data;
  gyroscope_data_t gyroscope_data;

  // Report Frequencies
  volatile uint16_t rotation_vector_report_frequency;
  volatile uint16_t accelerometer_report_frequency;
} sensor_meta;

/* Initialization Functions */
// Configures the SPI peripheral and resets the BNO085
uint8_t init_Hardware_BNO085(SPI_HandleTypeDef *hspi,
                             bno085_library_spi_config_struct config);

// Initializes Chip Select (CS) and Interrupt (INT) pins
void init_GPIO_IMU(struct sensor_meta *sensor);
```

GPS Interface (NEO_M9_UART.h)

API for the NEO-M9N GPS, using the binary UBX protocol for parsing.

```
/* UBX NAV-PVT Binary Message Structure */
/* Packed to align with raw UART buffer for DMA transfer */
```

```
#pragma pack(push, 1)
typedef struct {
    uint32_t iTOW;      // GPS time of week (ms)
    uint8_t  fixType;   // GNSSfix Type: 0..5
    uint8_t  numSV;     // Number of satellites
    int32_t  lon;       // Longitude (deg * 1e-7)
    int32_t  lat;       // Latitude (deg * 1e-7)
    int32_t  hMSL;      // Height above Mean Sea Level (mm)
    int32_t  gSpeed;    // Ground Speed (mm/s)
    int32_t  headMot;   // Heading of motion (deg * 1e-5)
} UBX_NAV_PVT_t;
#pragma pack(pop)

/* Main GPS Data Object */
typedef struct GPS_DATA {
        uint8_t newData;
        float Latitude;
        float Longitude;
        float Altitude;
        float Ground_Speed; // for EKF velocity updates
        uint8_t Fix_Type;
} GPS_DATA_t;

/* DMA-Driven Parsing Function */
void processGPS(const uint8_t* p_data, uint16_t length);

/* Initialization and Geofence Setup */
void M9N_Init(UART_HandleTypeDef *huart);
void config_geofence(UART_HandleTypeDef *huart, uint8_t numFences,
                uint8_t confLvl, int32_t lat_buf[],
                int32_t lon_buf[], uint32_t rad_buf[]);
```

Telemetry Interface (Xbee.h)

API for the XBee radio link used for ground station communication.

```
/* Initialization */
void xbee_init(UART_HandleTypeDef *huart);

/* Data Transmission */
/* Returns True if data is successfully sent */
bool xbee_send_data(uint8_t *payload, uint16_t length);

/* Data Reception */
/* Reads incoming telemetry or manual override commands */
```

```
bool xbee_receive_data(uint8_t *buffer, uint16_t max_len);

/* Command Parsing */
void handle_cmd(uint8_t *cmd, UART_HandleTypeDef *huart);
```

Actuation & Control (Servo.h, Control.h)

APIs for Servo Motor control (PWM Gen) and Navigation.

```
/* Servo Channel Mappings (TIM2) */
typedef enum {
        SERVO_L = TIM_CHANNEL_1,
        SERVO_R = TIM_CHANNEL_2
} SERVO_CHANNELS;

/* Microsecond-Level Control */
void servo_init_micro(TIM_HandleTypeDef *htim);
void ServoWriteUS(uint32_t channel, uint16_t pulse_us);

/* High-Level Abstractions */
void turnLeft(float angle);
void turnRight(float angle);
void deadSpin(); // Failsafe spiral descent

/* Navigation Math */
// Calculates bearing to target
float calc_angle(float target_long, float target_lat, float current_long, float current_lat);
// Checks if payload is within acceptance radius
bool withinTarget(float target_long, float target_lat, float current_long, float current_lat);
```

State Estimation (ekfAlgo.h)

Interface for the Extended Kalman Filter used for sensor fusion.

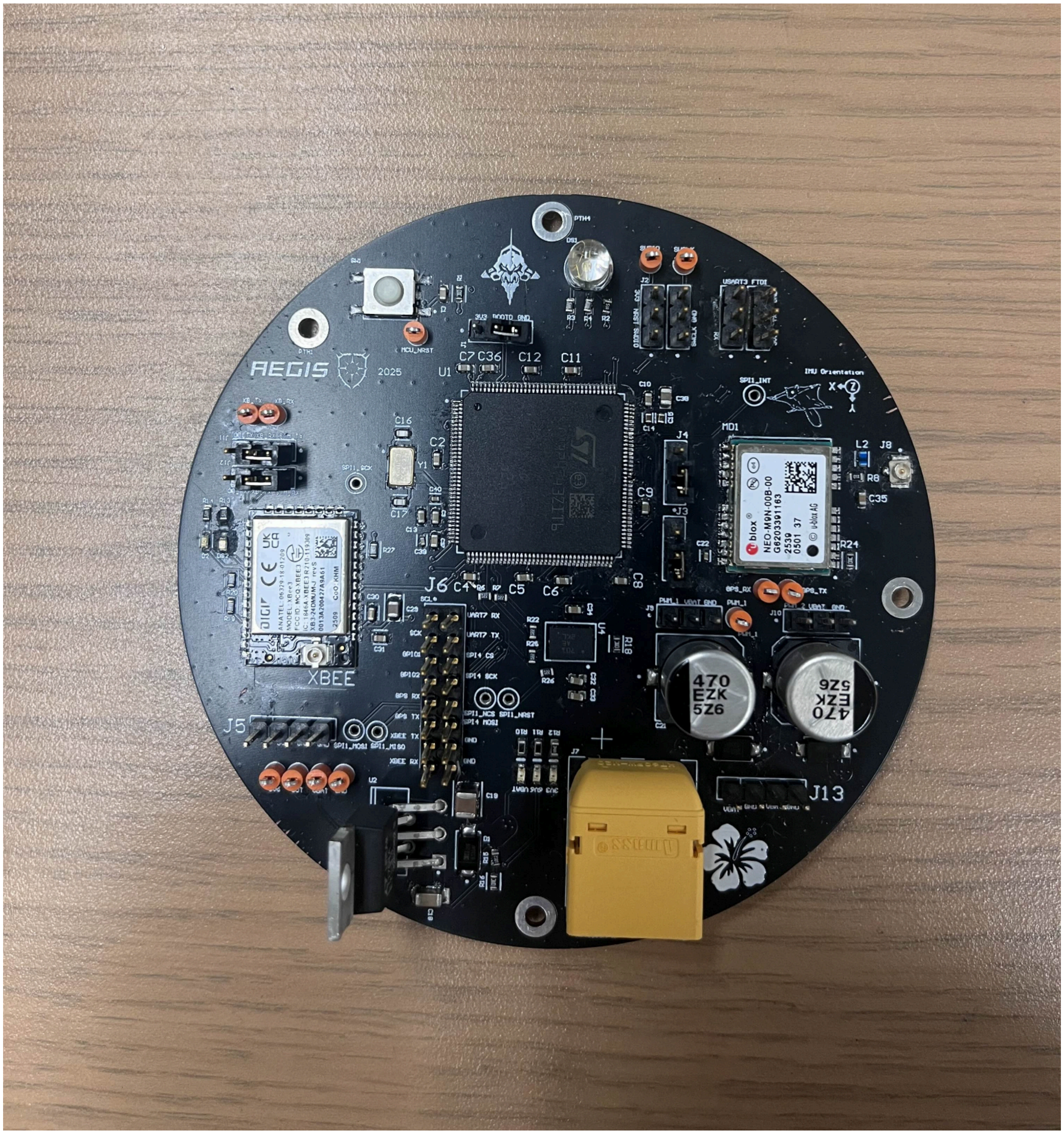
```
#define EKF_NUM_STATES 7        // [PosN, PosE, PosD, VelN, VelE, VelD, PsiBias]
#define EKF_NUM_MEASUREMENTS 7  // GPS Inputs

typedef struct {
   /* State estimates - NED Frame */
   float pos_n, pos_e, pos_d;
   float vel_n, vel_e, vel_d;
   float heading_bias;         // Magnetic North offset correction

   /* Covariance Matrices */
   float P[EKF_NUM_STATES][EKF_NUM_STATES]; // Error Covariance
```

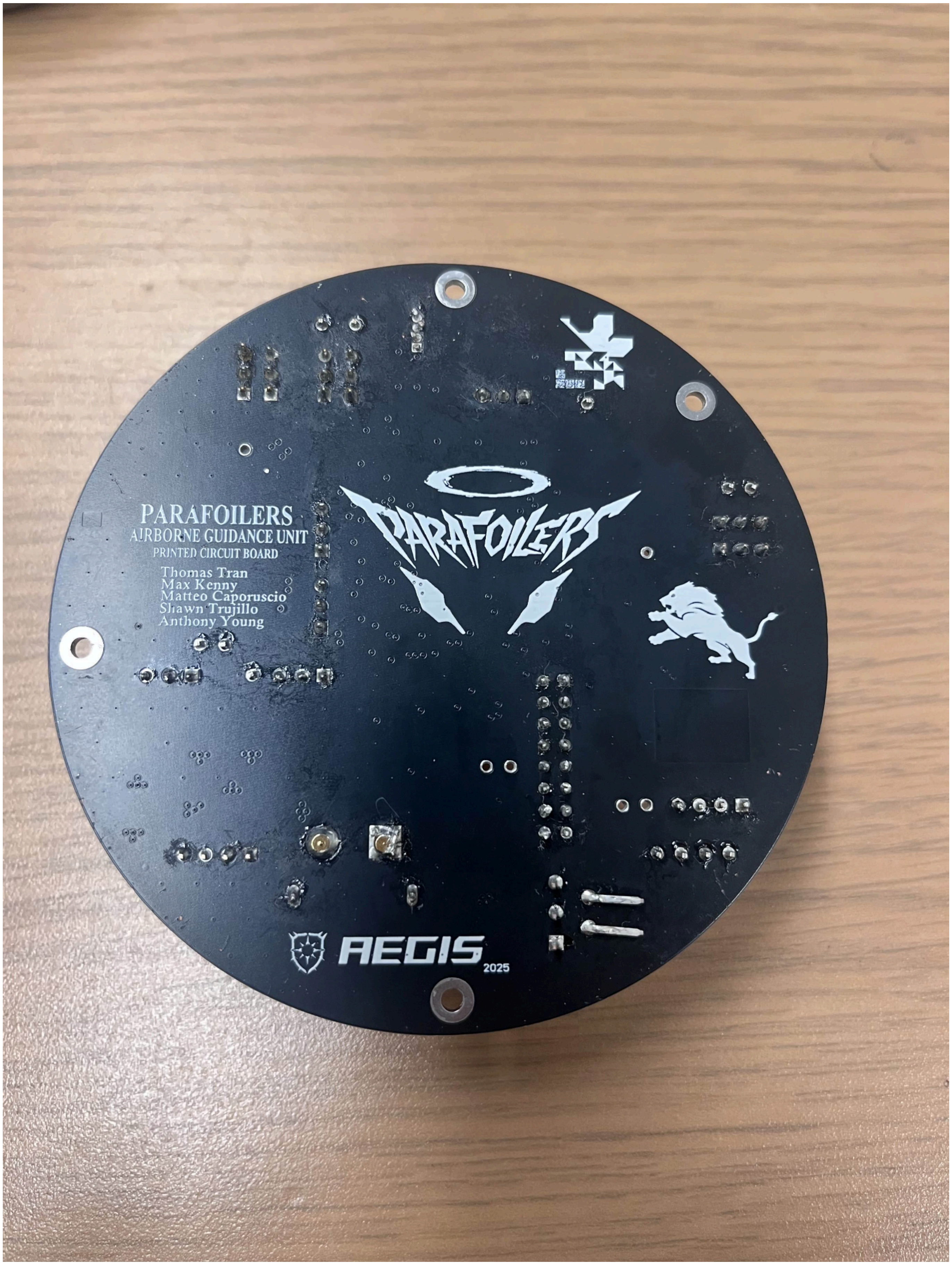
```
    float Q[EKF_NUM_STATES];                  // Process Noise
    float R[EKF_NUM_MEASUREMENTS];            // Measurement Noise

    uint8_t initialized;
} ParafoilEKF;

/* Prediction Step (100 Hz) */
/* Uses IMU acceleration and quaternion to predict state */
void ParafoilEKF_Predict(ParafoilEKF *ekf, float *quat, float *accel_body, float dt);

/* Update Step (~5 Hz) */
/* Corrects state estimate using GPS position and velocity */
int8_t ParafoilEKF_UpdateGPS(ParafoilEKF *ekf, double gps_lat, double gps_lon,
                             float gps_alt, float gps_speed,
                             float gps_heading, float *quat,
                             uint8_t valid_heading);
```

Github Repository

https://github.com/mjcapo/EECS473-Parafoilers.git

Printed Circuit Board Images




PARAFOILERS
AIRBORNE GUIDANCE UNIT
PRINTED CIRCUIT BOARD
Thomas Tran
Max Kenny
Matteo Caporuscio
Shawn Trujillo
Anthony Young
PARAFOILERS
AEGIS
2025

Drop Test Videos:

- No-Control Drops (No Manual or Autonomous Control)
- Manually Controlled Drops
- Autonomously Controlled Drops
- Demo Drop Video

References & Bibliography

[1] Radiosonde Recovery
https://www.weather.gov/media/key/KEY%20-%20Weather%20Balloon%20Poster.pdf
[2] NASA Funded Balloon Recovery
https://www.nasa.gov/science-research/heliophysics/nasa-funded-balloon-recovered-a-year-after-flight-over-antarctica/
[3] BNO085 SHTP Protocol Parsing
https://github.com/formove-ai/STM32_BNO085