Undergraduate Research Project Deliverable

Implementation of a Hydrophone Winch System for a Solar-Powered Seaplane UAV

Graduate Student Advisor: Brian Epstein
Undergraduate Student: Joshua Li

Project Duration: Spring ‘26

1. Background

The AHAB project focuses on the design and operation of a solar-powered seaplane UAV intended for long-endurance marine mammal monitoring. A critical component of this mission is the ability to record underwater acoustic data to detect and identify marine species.

Figure 1. AHAB Seaplane

To achieve this, the aircraft must deploy a hydrophone (an underwater microphone designed to detect subaquatic sound) while resting on the water's surface. The hydrophone must be lowered to a specific depth to capture clear audio and then retracted before the aircraft takes off again. The current system requires a lightweight, reliable, and waterproof mechanism to automate this deployment and retrieval process.

2. Project Description

The undergraduate researcher will be responsible for the mechanical design, prototyping, and testing of a custom winch system for the AHAB aircraft. The system must house and deploy a specific sensor, the AudioMoth "Hydromoth," which is a low-cost, full-spectrum acoustic logger.

Figure 2. HydroMoth Hydrophone

A unique challenge of this project is the interface with the Hydromoth, which utilizes a magnetic reed switch for activation. The winch mechanism must be designed not only to raise and lower the sensor but also to trigger this magnetic switch to conserve battery life when the sensor is stowed. The primary tasks will include conceptual design, detailed CAD modeling, component selection (motors, gears, slip rings), and physical validation. The researcher must ensure the mechanism adheres to strict weight and integration constraints

Table 2. Specification Requirements

Requirement Category	Specification
Weight Budget	<400g
Mounting Location	Tail section of aircraft
Waterproofing	Mechanism must prevent water entry into fuselage (IP67 equivalent).
Sensor Integration	Compatible with Audiomoth Hydromoth form factor.
Actuation	Automated deployment and retraction of the tether.
Switching Mechanism	Ability to toggle the Hydromoth's magnetic on/off switch.

3. Design Evolution and Iterative Prototyping

Over the course of the semester, the winch system evolved through two primary design phases. The first phase focused on a more optimized solution, while the second phase prioritized functional validation (the minimum viable product). The switch in designs midway through was based on time considerations. However, both designs are potentially valid in tackling the problem statement, each with their own advantages and trade-offs.

3.1 Design 1: Integrated Hull Underbody Cutout (Weeks 1–11)

The initial design objective was to house the hydrophone flush within a recessed cavity in the aircraft's underbody to maximize aerodynamic efficiency.

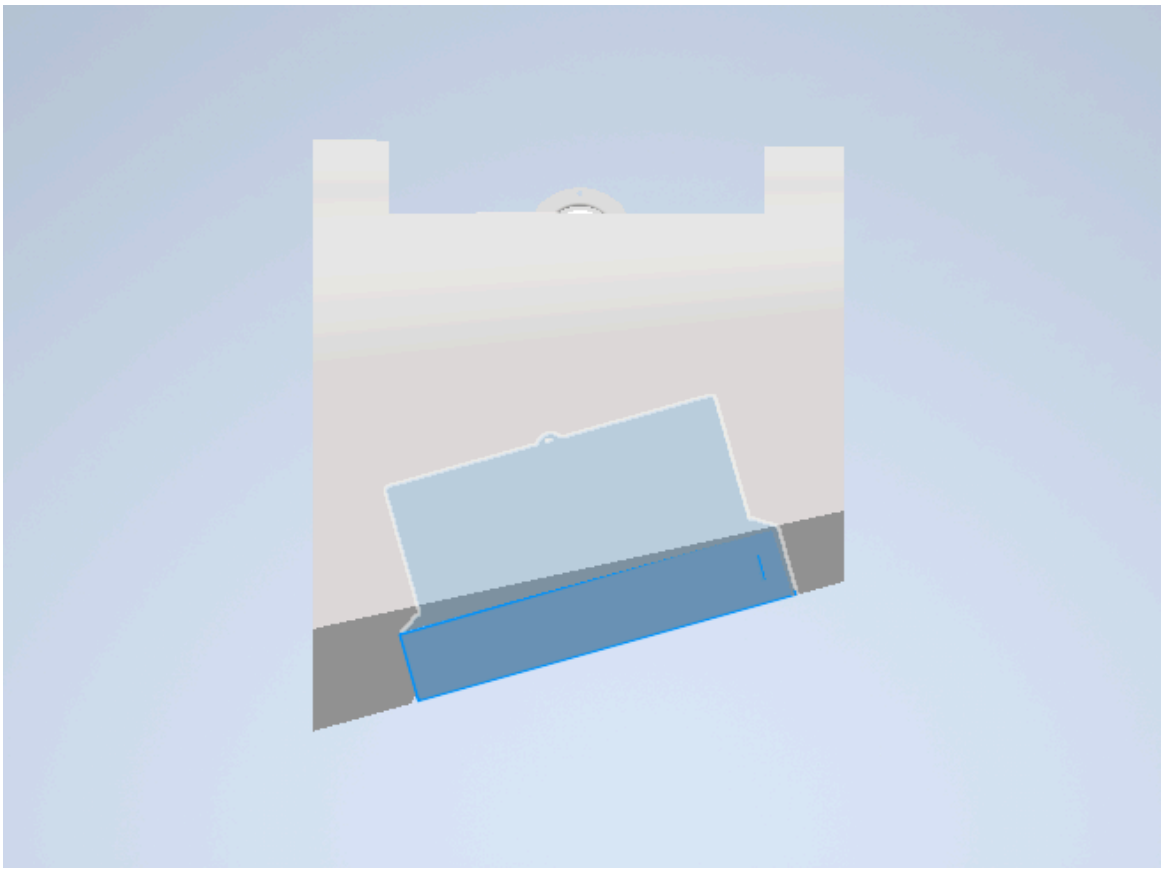

Figure 3. Integrated Hull Underbody Cutout Design Side View

In the figure above, the blue showcases the hydrophone enclosure that meshes exactly with the bottom geometry of the seaplane. A chamfer is utilized to autocenter the hydrophone enclosure once it is winched upward.

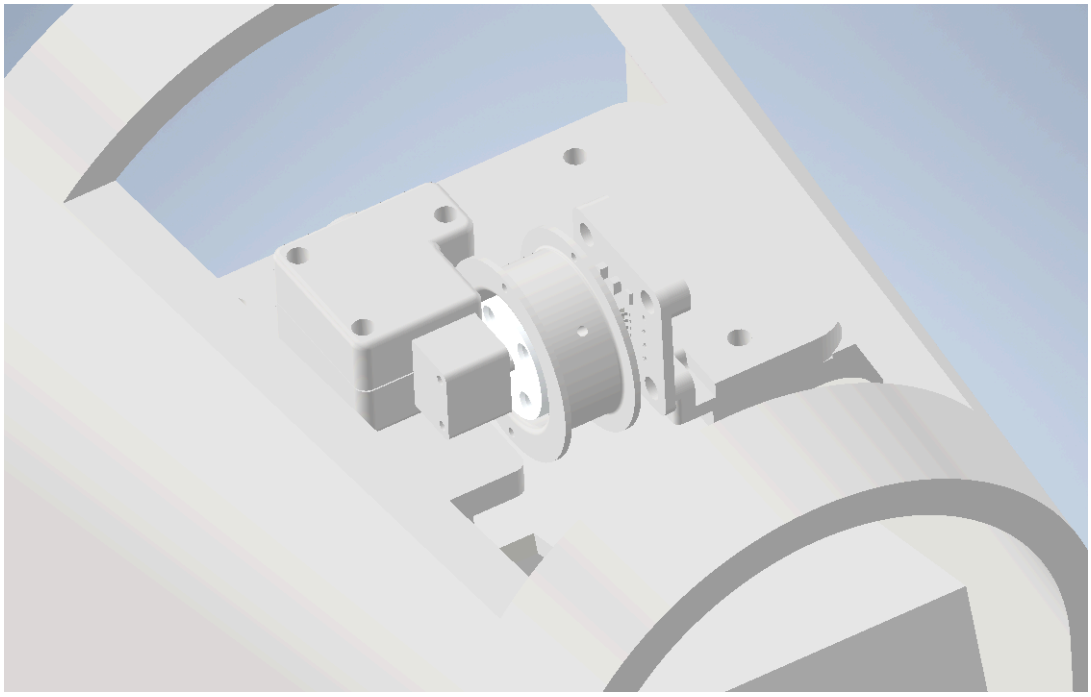

Figure 4. Winch Mechanism

In Figure 2, this showcases the winch mechanism that utilizes a 53RPM 12V worm gear motor. A worm gear motor was selected as it provides a self-locking mechanism. Due to the high friction between the worm and the gear, the system can support the weight of the hydrophone payload in a static position without requiring active motor torque. This ensures that the sensor remains securely retracted during flight without drawing constant current, significantly optimizing the aircraft's limited battery capacity for long-endurance monitoring. Additionally, the worm gear configuration allows for high torque output in a compact form factor, which was essential for adhering to the strict 400g overall weight budget.

The motor is mounted to a plate that also houses the encoder as seen to the right of the spool. This plate is then screwed into the hull that has press fits that house the heat set inserts. Since the plane will eventually be printed out of PLA Aero, a lightweight, foaming filament, press fits are required since heat set inserts cannot be used on that material. The configuration is shown in the figure below.

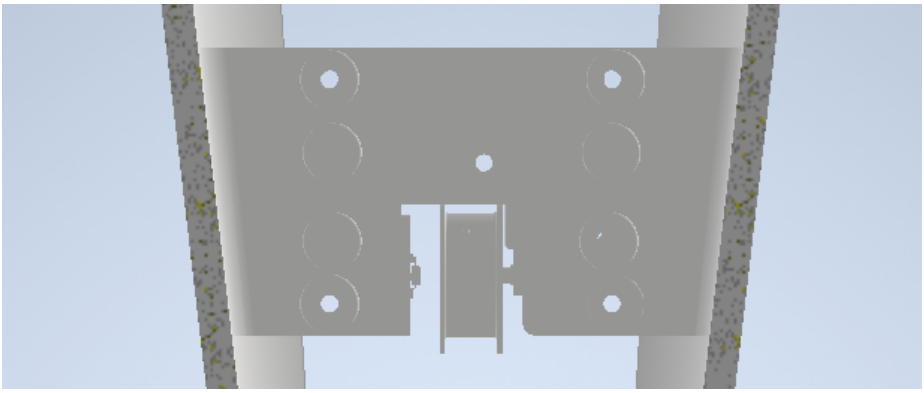

Figure 5. Press Fit Heat Set Inserts within Hull

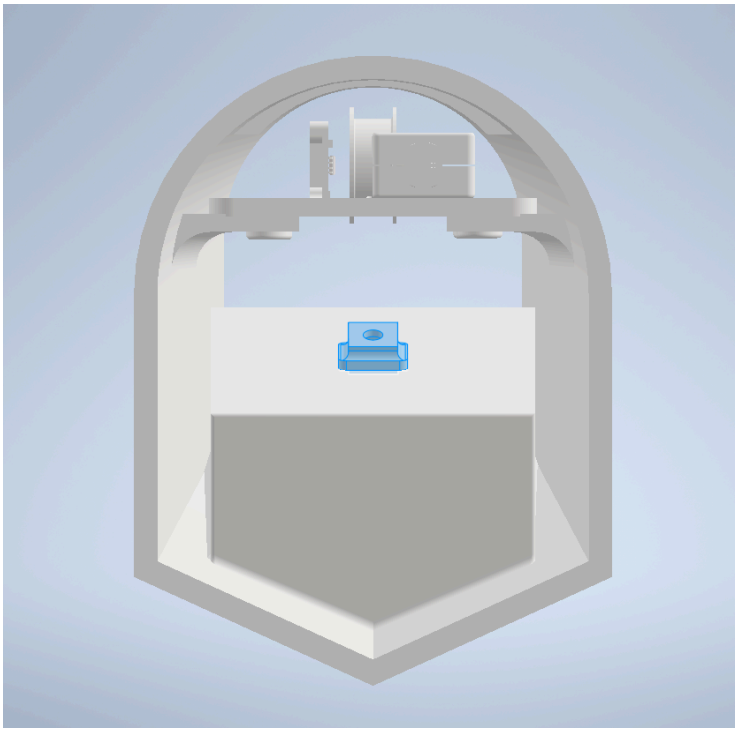

Figure 6. Waterproofing Chamber within Main Assembly

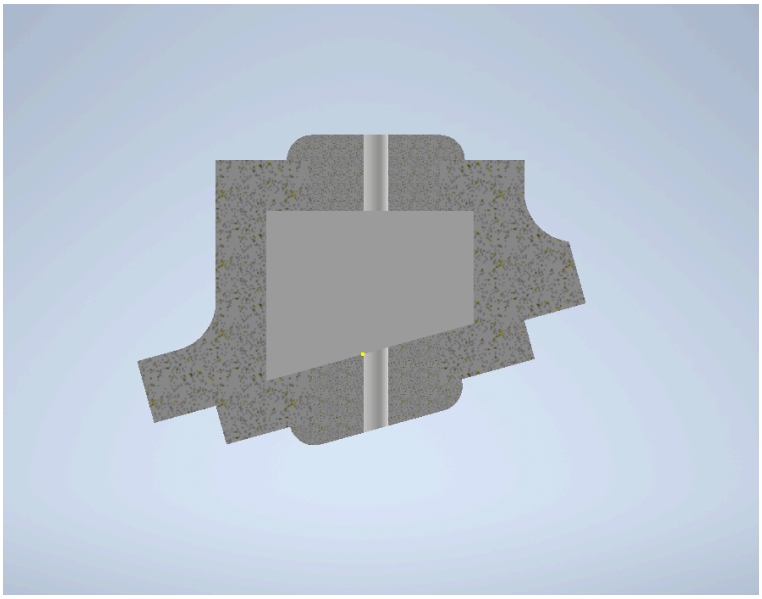

Figure 7. Waterproofing Assembly Section View

The two figures above showcase the waterproofing assembly. The waterproofing assembly is a chamber that is filled with marine grease, acting as a barrier against water transferring between the two sides. Additionally, the top and bottom of the chamber utilize two press fits that allow grease to be injected into the chamber using a syringe, then closed off with a 1 mm diameter hole that just barely allows the string to pass through.

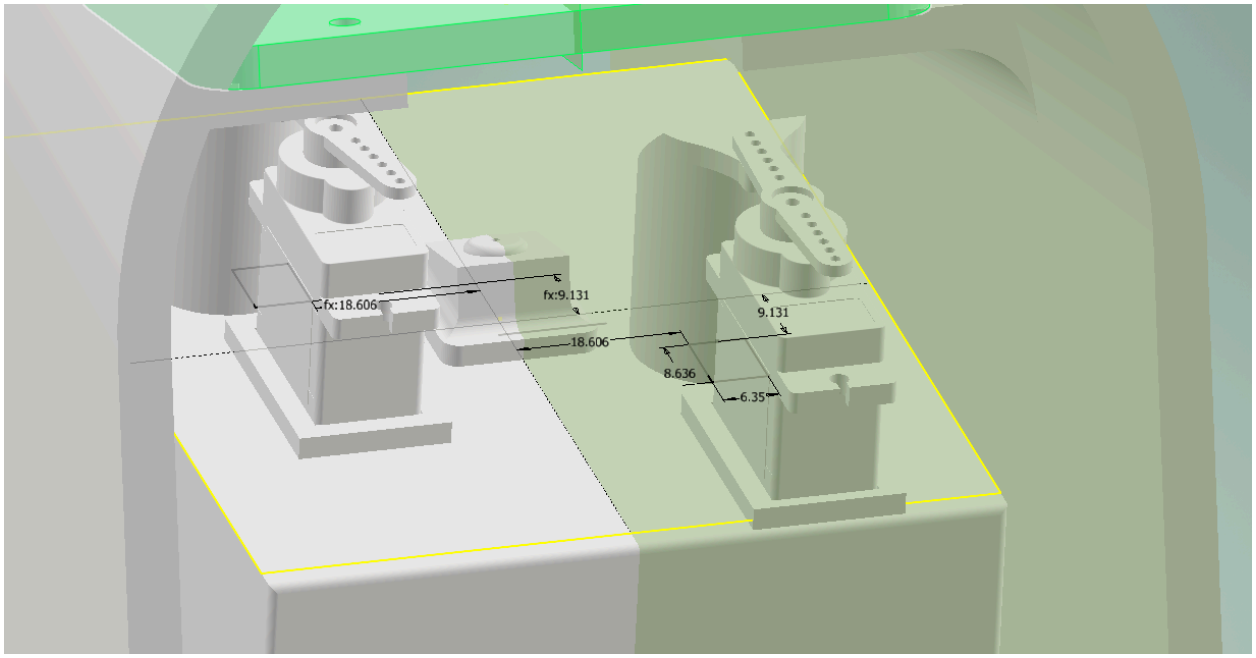

Figure 8. Initial Magnetic Switching Design

This was the first magnetic switching design that utilized two servos. However, this is unideal as it required twice the power input as opposed to using just one servo. This was a stepping stone in testing the implementation that later informed previous designs of this mechanism.

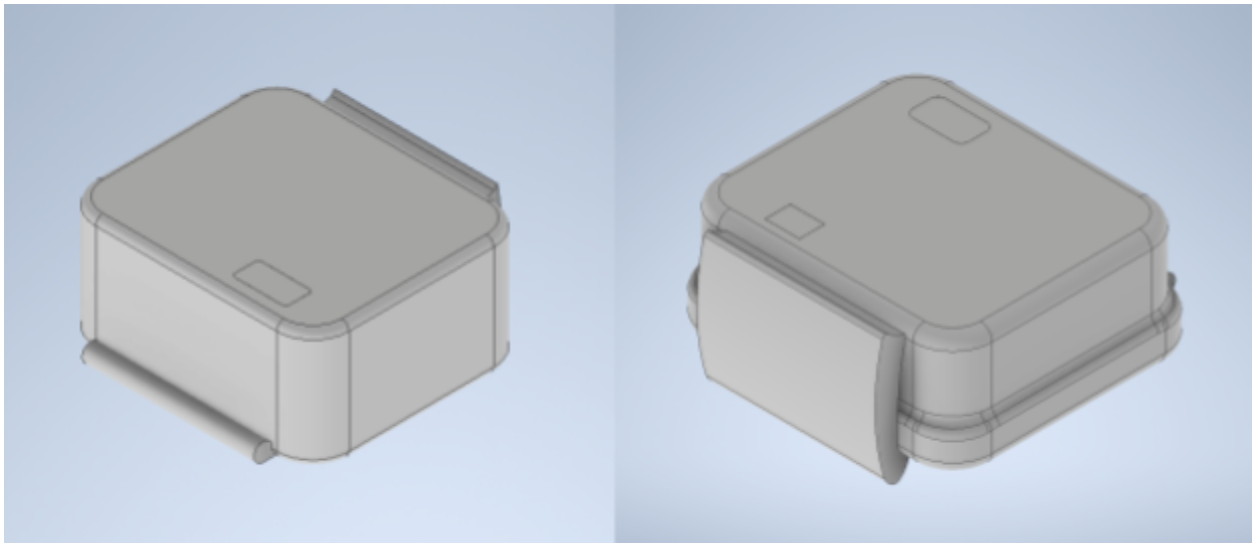

Figure 9. Hydrophone Design v1 and v2

During this design process, the hydrophone had to be modeled from scratch since there was no CAD for it. Going through two iterations, one for initial design and one with accurate dimensions, measured using a caliper. This allowed for more accurate designs in later iterations of the winch system.

Overall, this first design was the bulk of the design work that helped inform future design iterations of mechanisms of this system as this design underwent six major design iterations. Near the end of this design, there was a major design oversight that led to the transition. After creating the hydrophone enclosure, there was a realization that the enclosure actually floated instead of sinking below the water. Since the hydrophone is supposed to be submerged in the water to record, this was a major problem. Although there was a fix to this problem by reducing the volume of air in the enclosure, due to time remaining in the semester, a working minimum viable product was prioritized and the more optimized design could be revisited after the minimum viable product implementation was successful.

3.2 Design 2: Minimum Viable Product - Side-Mount Configuration (Weeks 11–15)

The goal of this design alternative was to create the minimum viable product. To achieve this, mounting the hydrophone to the side of the hull would be the simplest. Using experience from previous design iterations, this design was built out with much less time required.

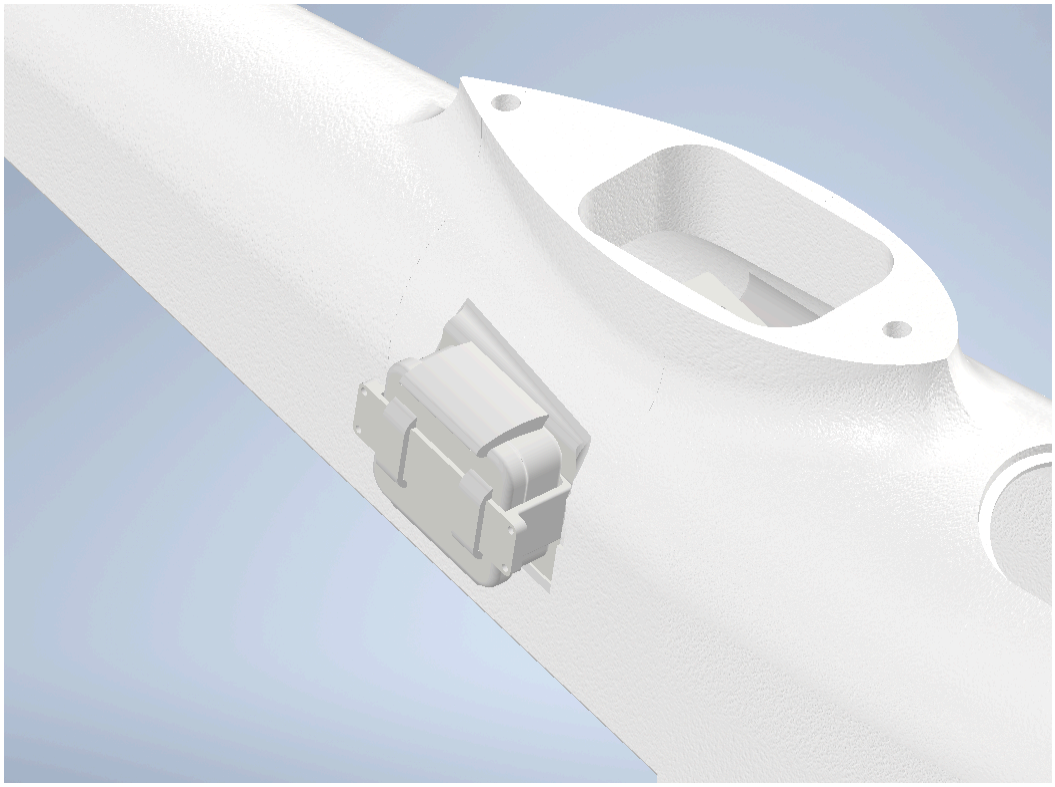

Figure 10. Side-Mount Hydrophone Design

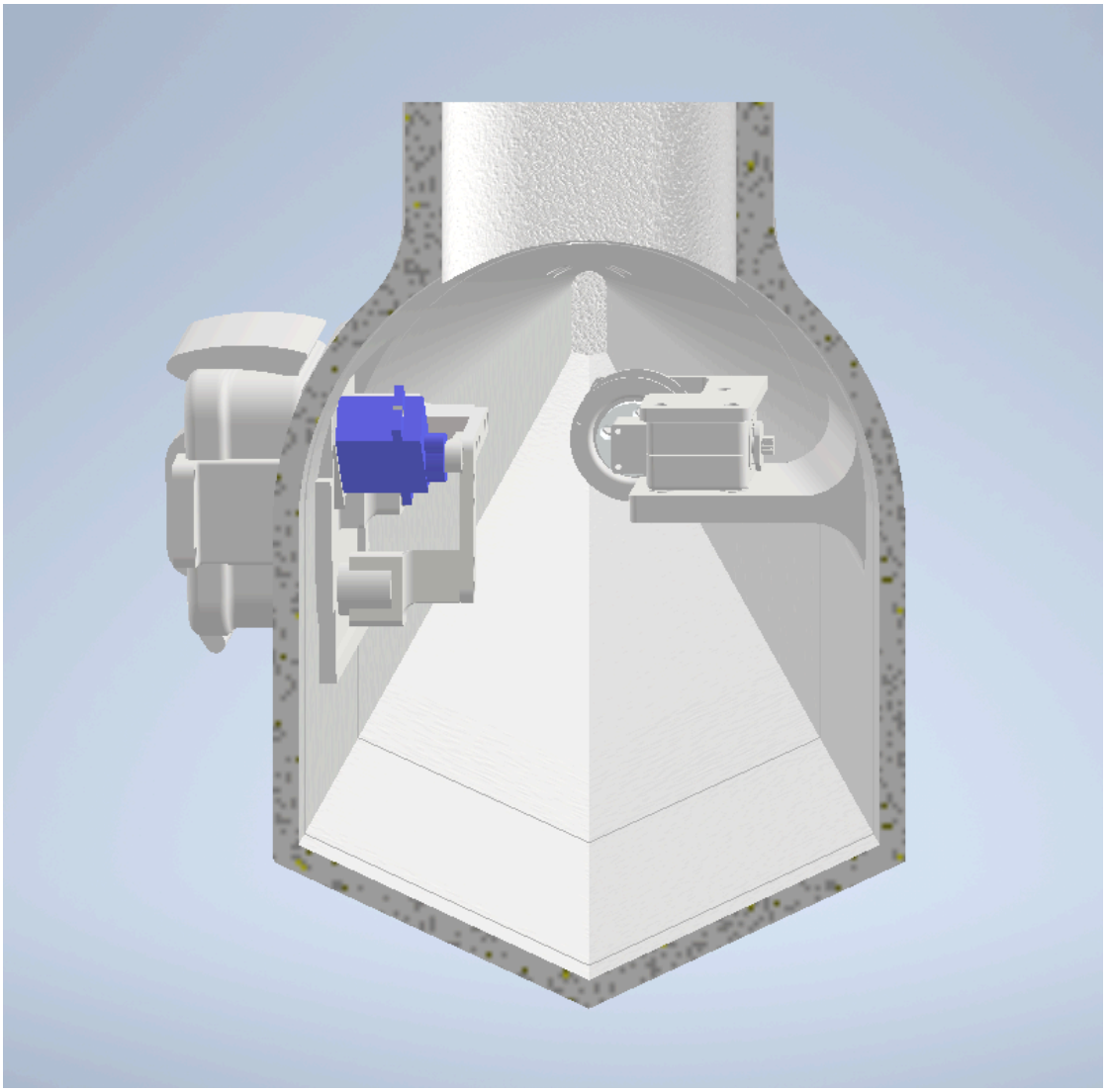

Figure 11. Section View of Side Mount Hydrophone Design

The two figures above showcase an overview of the minimum viable product design. The hydrophone is located in the midsection of the hull to avoid interfering with the water as it takes off since the sea plane sits an inch or two nominally within the water.

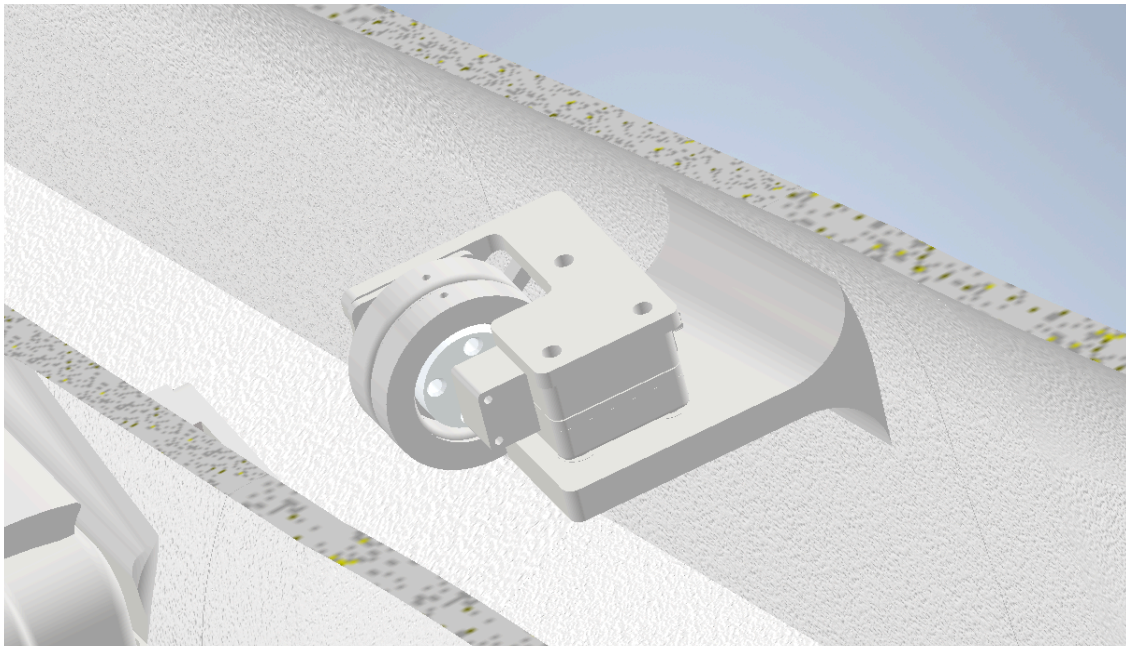

Figure 12. Close-up View of Winch Mechanism

This winch design is slightly different than the one used for design 1 due to difference in orientation; however, it employs the same design ideas using the press fit heat set insert into the hull. A separate piece was designed to mount the encoder stacked upon the motor mount. A spool cover was also added to the spool to prevent unspooling and tangling of the string during spooling and unspooling.

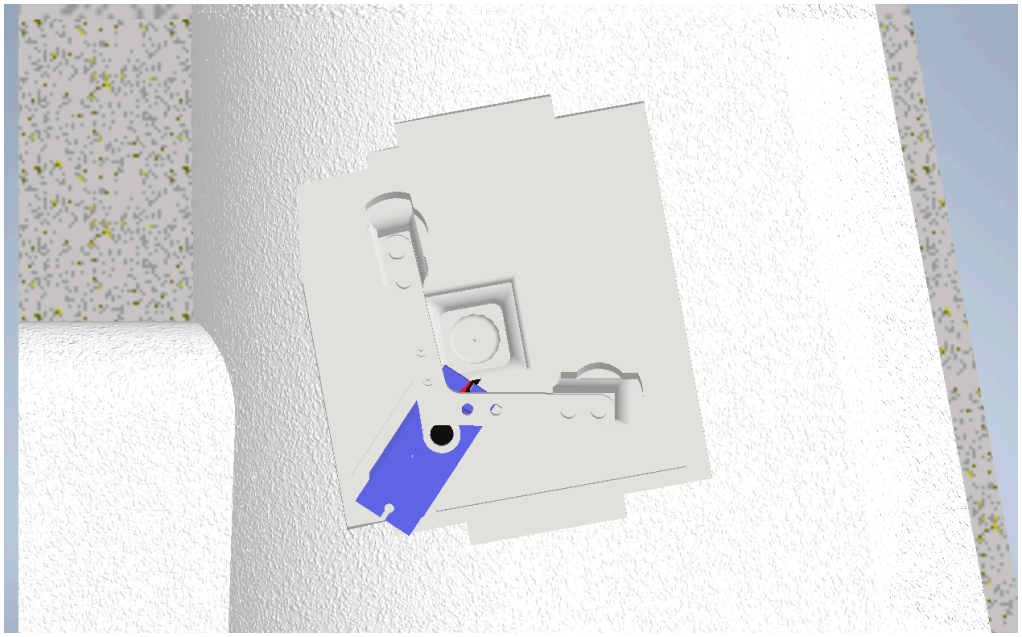

Figure 13. Close-Up View of Magnetic Switching Mechanism

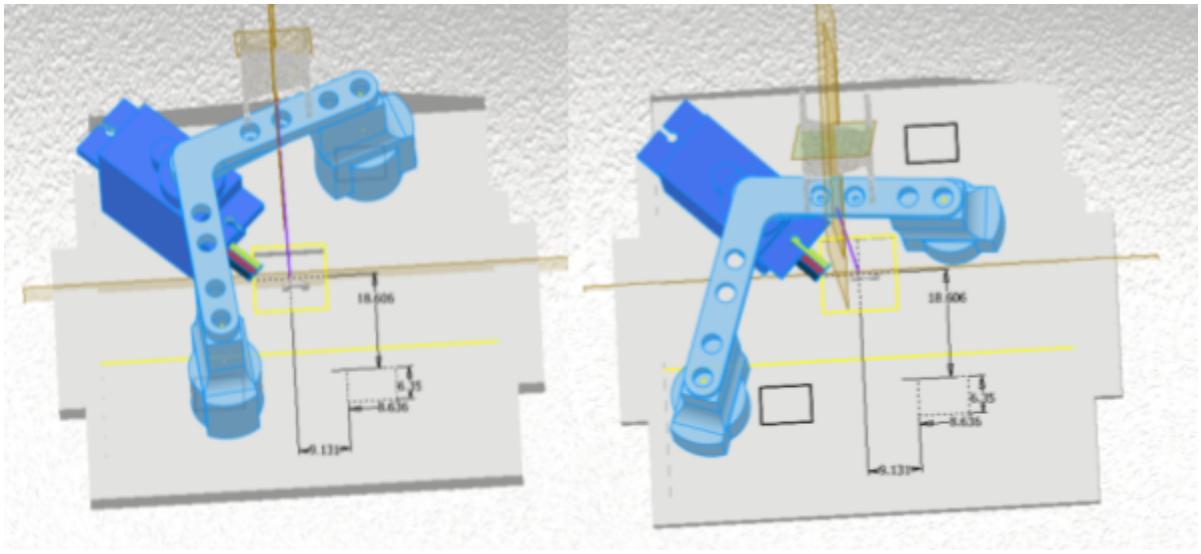

Figure 14. Magnetic Switching Arc

Figure 13 and 14 showcases the modified magnetic switching design that utilizes only one servo instead of the previous two. The length and angle of the arm was determined by modeling it within CAD first and then tested in the real-world.

After testing in real-life, it was found that the magnetic reed switch on the hydrophone is more sensitive than initially thought, so making the magnet not directly contact the hull surface but floating 0.1 in above may yield better results. Additionally, to trigger the start and stopping of recording, the magnet must pass and then retreat from the sensor to trigger the toggle effectively.

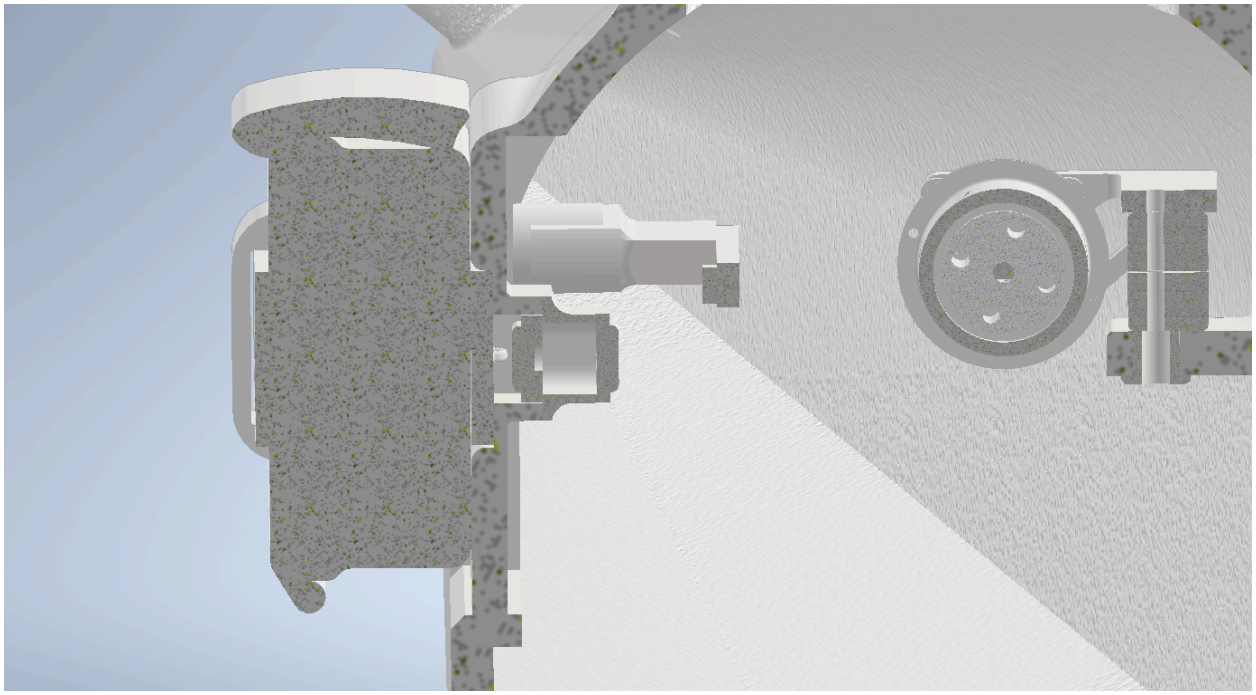

Figure 13. Section View of Waterproofing Chamber Design

This waterproofing chamber design is quite similar to the one used in design 1. The waterproofing chamber also uses marine grease and press fits to keep water out from within the hull.

4. Control Systems Implementation

The control architecture for the winch system is a closed-loop framework designed to automate the deployment and retrieval of the hydrophone payload with high precision. The system integrates an AS5600 12-bit magnetic rotary position sensor to provide continuous feedback to a custom-tuned PID control loop to ensure precise positioning at a target depth of 5 meters.

4.1 Encoder Feedback

Because the AS5600 encoder provides raw data in a range of 0 and 4095 counts, the code must account for the magnet's "rollover" as the spool completes multiple revolutions. The software handles this by calculating the delta between the current and last raw angles; if a transition across the 0/4096 boundary is detected, the system adds or subtracts the full count range to maintain an accurate total rotation count.

```
void updateRotationCount() {
  int currentRaw = encoder.rawAngle();
  int delta      = currentRaw - lastRawAngle;

  if (delta < -2048) delta += 4096;
  else if (delta > 2048) delta -= 4096;

  totalRotations += (float)delta / 4096.0 * -1.0;
  lastRawAngle = currentRaw;
}
```

Figure 14. updateRotationCount() method

4.2 PID Control Strategy

The primary control objective is to smoothly lower the hydrophone to depth without overshoot and retract it to a specific docking orientation. The PID control strategy manages the dynamics of the 250g payload by calculating three distinct terms. The Proportional (P) term provides the primary driving force, the Integral (I) term corrects steady-state error with an anti-windup clamp limited to ± 200.0, and the Derivative (D) term dampens oscillations to protect the internal gears from mechanical stress.

```
// ── P ──────────────────────────────────────────────────────
float P = kP * error;

// ── I (with anti-windup clamp) ─────────────────────────────
pidIntegral += error * dt;
pidIntegral  = constrain(pidIntegral, -INTEGRAL_CLAMP, INTEGRAL_CLAMP);
float I      = kI * pidIntegral;

// ── D ──────────────────────────────────────────────────────
float derivative = (error - lastError) / dt;
float D          = kD * derivative;
lastError        = error;

// ── Sum and apply deadband ─────────────────────────────────
float output = P + I + D;

// Enforce a minimum output so the motor actually moves
if (output > 0 && output < PWM_DEADBAND)  output =  PWM_DEADBAND;
if (output < 0 && output > -PWM_DEADBAND) output = -PWM_DEADBAND;

currentPWM = constrain((int)(PWM_STOP + output), PWM_MIN, PWM_MAX);
}
```

Figure 15. PID Implementation

4.3 PWM Mapping and Deadband Management

Physical actuation for the winch system is managed through an Electronic Speed Controller (ESC) utilizing Pulse Width Modulation (PWM) mapping. In this architecture, the ESC serves as the "brain," converting DC battery power into a signal that drives the motor, while the PWM signal dictates the specific speed and direction to the controller. A critical challenge addressed in this design is the high internal friction inherent to the N20 worm gear motor. To overcome this static friction, the control logic includes a specific deadband management system. This system ensures that the motor always receives a minimum PWM output of 80 whenever movement is commanded, providing the necessary "breakaway" power to initiate rotation before the PID loop takes over for fine-tuned adjustments.

The following snippet demonstrates how the software enforces this minimum output to maintain responsiveness despite mechanical resistance:

```
// Enforce a minimum output so the motor actually moves
if (output > 0 && output < PWM_DEADBAND)  output =  PWM_DEADBAND;
if (output < 0 && output > -PWM_DEADBAND) output = -PWM_DEADBAND;

currentPWM = constrain((int)(PWM_STOP + output), PWM_MIN, PWM_MAX);
}
```

Figure 16. Deadband Management Code

4.4 State Management and Telemetry

The system is rounded out by a finite state machine, tracking states like RAISED, LOWERED, and MOVING, and a telemetry system that provides real-time feedback every 250ms. This allows for detailed monitoring of the error margins and system stability during deployment.

```
void printTelemetry() {
  if (millis() - lastPrintTime < 250) return;
  lastPrintTime = millis();

  const char* stateStr = (currentState == RAISED)  ? "RAISED " :
                         (currentState == LOWERED) ? "LOWERED" : "MOVING ";

  Serial.print("State: ");    Serial.print(stateStr);
  Serial.print(" | Rot: ");   Serial.print(totalRotations, 2);
  Serial.print(" / ");        Serial.print(targetRotations, 2);
  Serial.print(" | Err: ");   Serial.print(targetRotations - totalRotations, 3);
  Serial.print(" | PWM: ");   Serial.print(currentPWM);
  Serial.print(" | Servo: "); Serial.print(targetServoPos);
  Serial.println(" deg");
}
```

Figure 17. printTelemetry() Method

5. Conclusion:

The AHAB winch system has reached the stage of a functional Minimum Viable Product (MVP), successfully integrating the mechanical design with closed-loop control systems. Through the iterative development of two distinct design configurations, the project has transitioned from a conceptual aerodynamic model to a system capable of reliable sensor deployment. This research has provided extensive experience in rapid prototyping and mechanical design, while the implementation of the AS5600 encoder and PID logic has allowed for the application of practical control theory. Specifically, the project facilitated a robust understanding of how Pulse Width Modulation (PWM) and Electronic Speed Controllers (ESC) interface to provide precise motor actuation.

5.1 Current Status and Accomplishments

At this stage, the winch system effectively manages the retrieval and deployment of the hydrophone within the established 400g weight budget. The software framework successfully monitors rotation counts, handles encoder rollovers, and maintains depth stability through real-time telemetry. Furthermore, the self-locking nature of the 150:1 worm gear motor has been validated as an energy-efficient solution for long-endurance solar-powered flight, as it requires zero power to hold the sensor in a stowed position.

5.2 Future Work and Optimization

To move the system toward a final flight-ready state, several optimizations and tweaks are required:

1) Environmental Testing: The newest iteration of the waterproofing chamber must undergo testing to ensure IP67 compliance under dynamic conditions.
2) Refinement of Magnetic Triggering: While the current servo-actuated magnet mechanism is functional, further geometric optimization of the magnet holder is needed to ensure a 100% success rate for toggling the HydroMoth sensor in varied orientations.
3) Aerodynamic Optimization: Following the successful validation of the MVP, the hydrophone enclosure can be redesigned to reduce drag, potentially revisiting the integrated "Hull Cutout" design once the buoyancy challenges are fully mitigated.
4) Control Loop Tuning: Further refinement of the PID constants may be explored to account for the fluid resistance encountered during different descent speeds.

Ultimately, this project establishes a robust technical foundation for autonomous acoustic monitoring in marine environments. By successfully bridging mechanical fabrication with closed-loop robotic control, this prototype demonstrates that a lightweight, energy-efficient winch system is feasible for long-endurance UAV operations. While further refinements will move the design closer to a flight-ready implementation, the current architecture serves as a critical proof-of-concept for low-cost, sustainable marine mammal surveillance.

6. Appendix

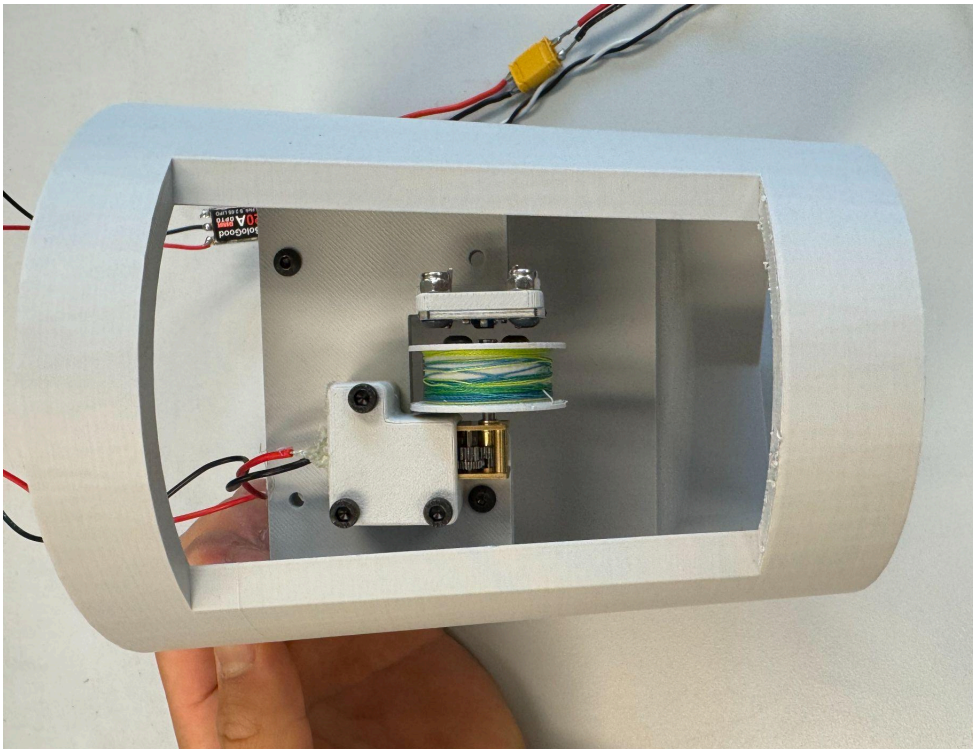

Figure A1. Design 1 Top View

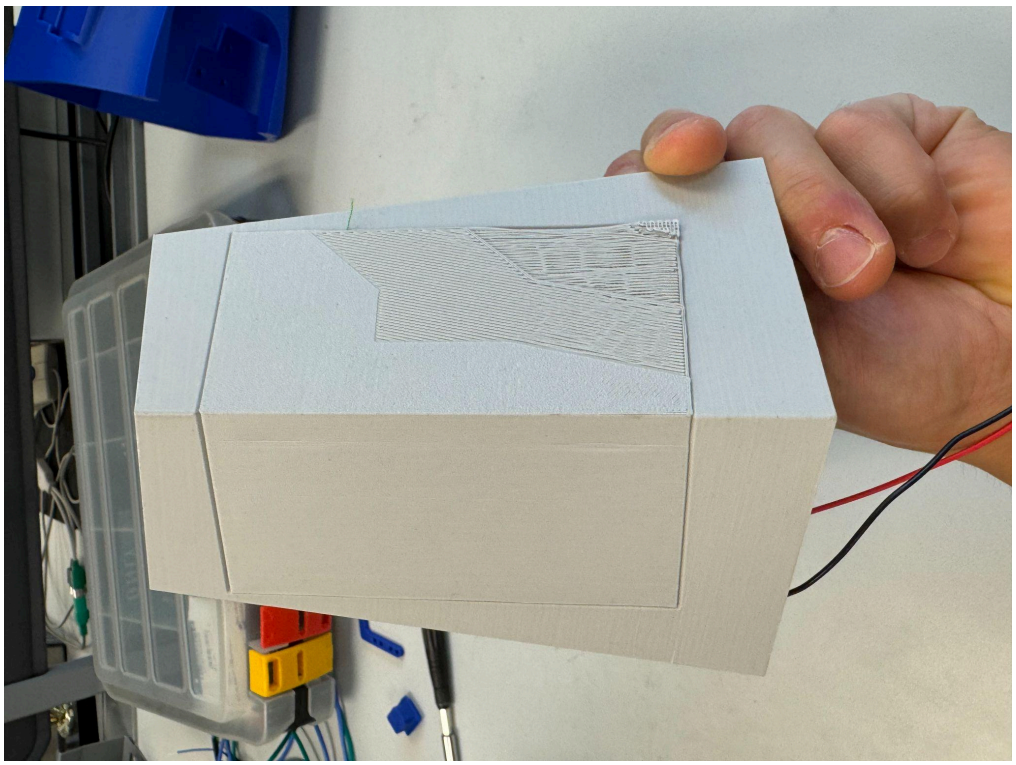

Figure A2. Design 1 BottomView

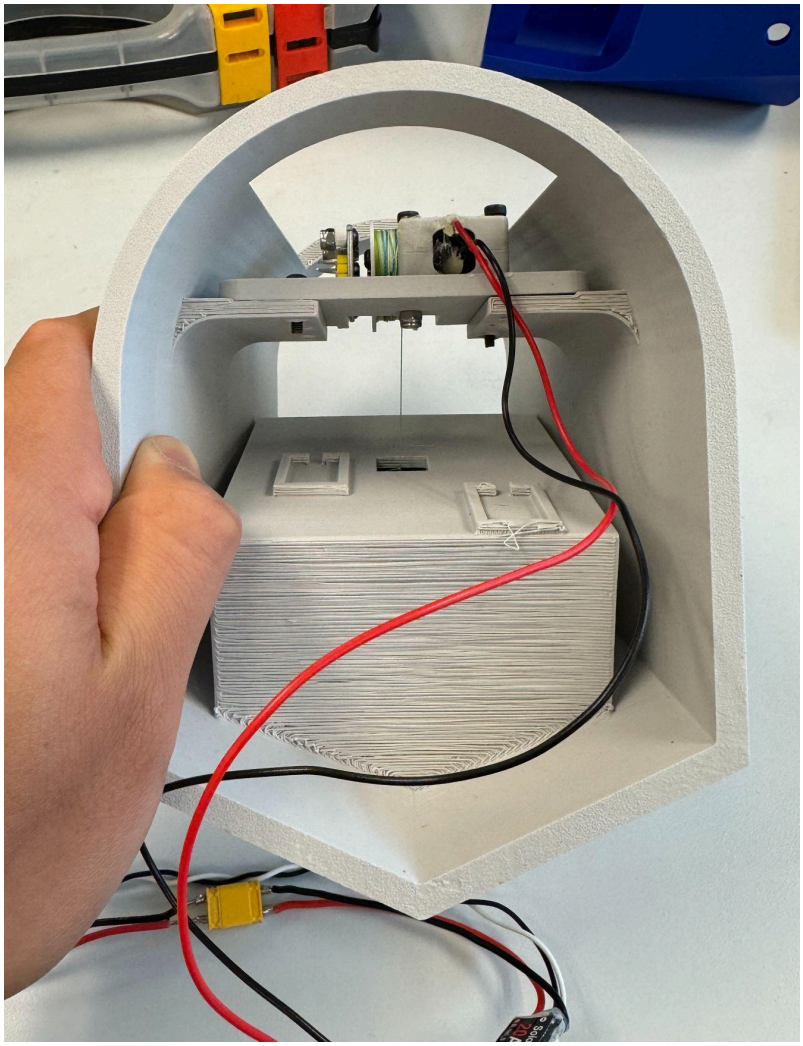

Figure A3. Design 1 Side View

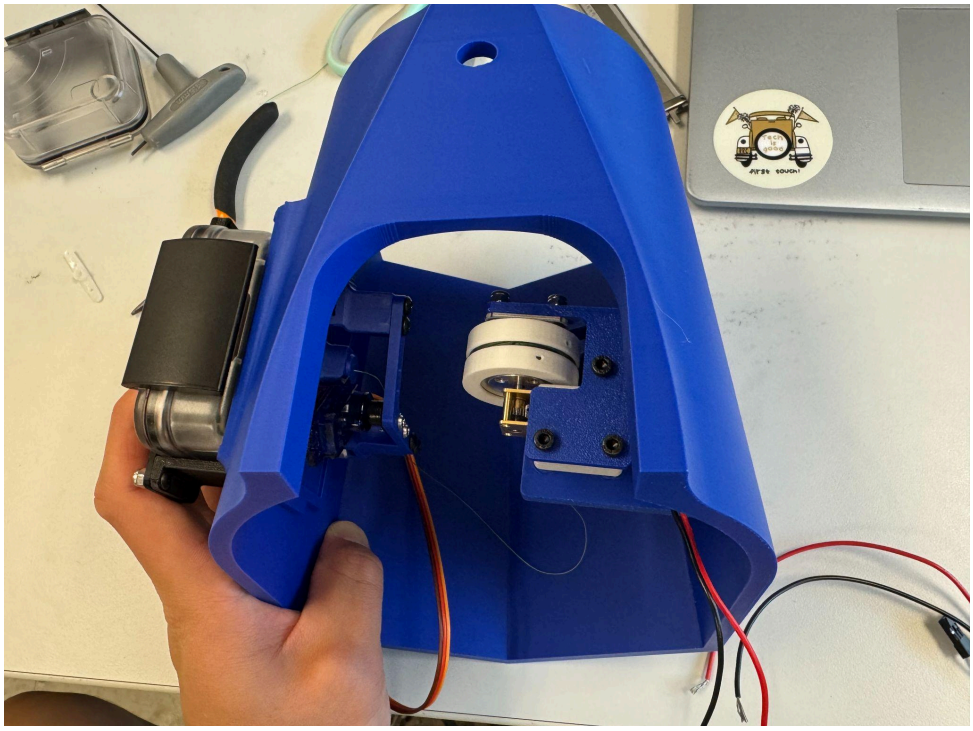

Figure A4. Design 2 (Minimum Viable Product) Top View

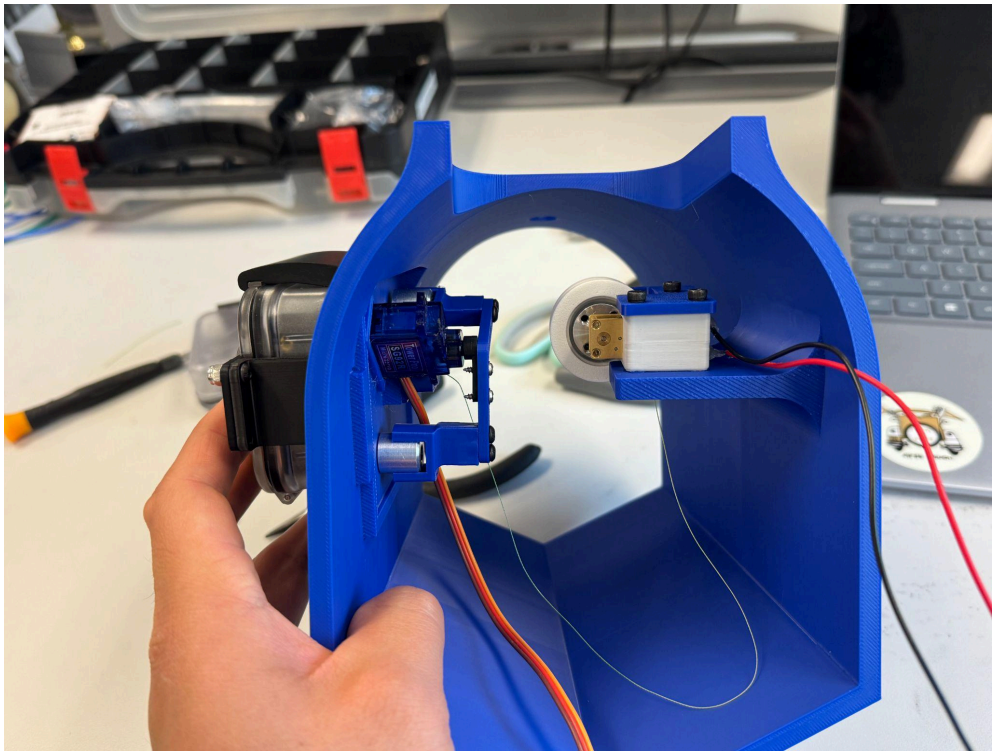

Figure A5. Design 2 (Minimum Viable Product) Side View

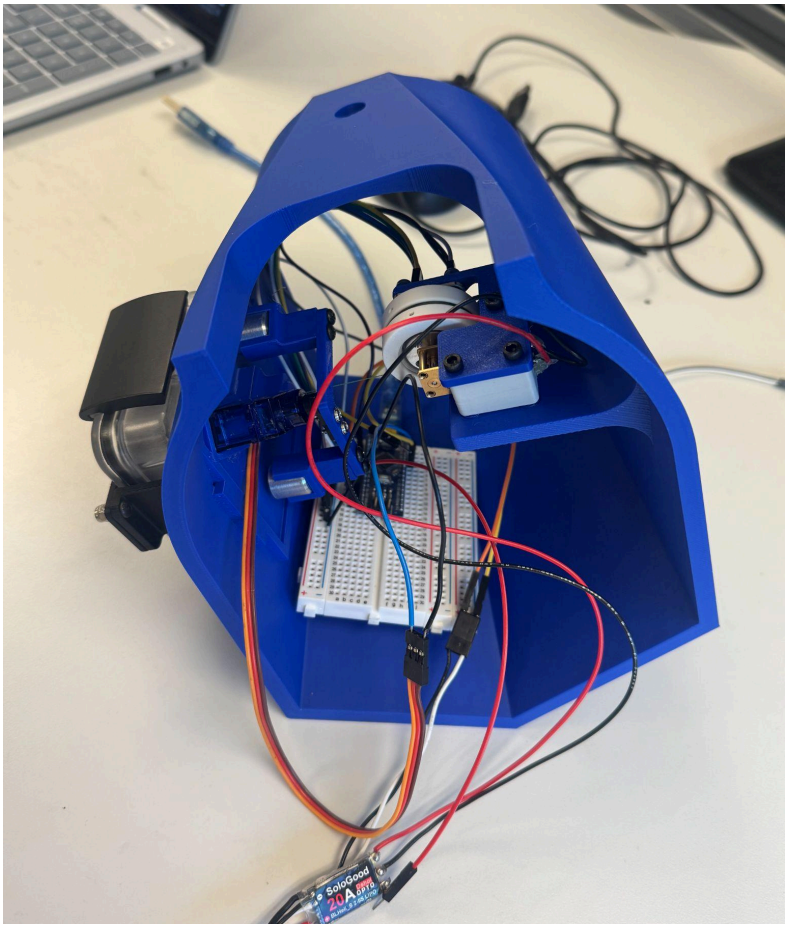

Figure A6. Design 2 (Minimum Viable Product) with Electrical Implementation

List A1. Full Code Implementation

```
#include <Servo.h>
#include <Wire.h>
#include "AS5600.h"

// ── Pin & Device Config ──────────────────────────────────────
Servo esc;
Servo auxServo;
AS5600 encoder;

const int ESC_PIN   = 9;
const int SERVO_PIN = 10;

// ── PWM Config ───────────────────────────────────────────────
const int PWM_STOP    = 1500;
const int PWM_MIN     = 1150;  // Max reverse power
const int PWM_MAX     = 1850;  // Max forward power
const int PWM_DEADBAND = 80;   // Min offset to actually move motor
int currentPWM        = PWM_STOP;

// ── Servo Positions ──────────────────────────────────────────
const int SERVO_NOMINAL = 80;
const int SERVO_ALT     = 60;
int targetServoPos      = SERVO_NOMINAL;

// ── PID Tuning Parameters ────────────────────────────────────
// Start with only kP, keep kI and kD at 0, then tune from there
float kP = 400.0;   // Proportional: main driving force
float kI = 0.0;     // Integral: corrects steady-state error
float kD = 50.0;    // Derivative: dampens oscillation

// ── PID State ────────────────────────────────────────────────
float pidIntegral       = 0.0;
float lastError         = 0.0;
uint32_t lastPidTime    = 0;

const float INTEGRAL_CLAMP = 200.0;  // Prevents integral windup

// ── State & Rotation Tracking ────────────────────────────────
enum SystemState { RAISED, LOWERED, MOVING };
SystemState currentState = RAISED;
```

```
float totalRotations  = 10.0;
int   lastRawAngle    = -1;
float targetRotations = 10.0;

const float TOLERANCE = 0.02;

uint32_t lastPrintTime = 0;

// ---------------------------------------------------------------
void setup() {
  Serial.begin(115200);
  Wire.begin();
  encoder.begin();

  esc.attach(ESC_PIN, 1000, 2000);
  auxServo.attach(SERVO_PIN);
  auxServo.write(targetServoPos);

  Serial.println("Arming ESC... Please wait.");
  esc.writeMicroseconds(1000);
  delay(2000);
  esc.writeMicroseconds(1500);
  delay(3000);

  lastRawAngle  = encoder.rawAngle();
  lastPidTime   = millis();

  Serial.println("System Ready!");
  Serial.println("Commands: U=Up  D=Down  S=Stop  T=Toggle Servo");
  Serial.println("          P###=set kP   I###=set kI   K###=set kD");
}

// ---------------------------------------------------------------
void loop() {
  updateRotationCount();
  checkSerialCommands();
  runPID();

  esc.writeMicroseconds(currentPWM);
  auxServo.write(targetServoPos);
  printTelemetry();
}
```

```
// ── PID Controller ──────────────────────────────────────────────
void runPID() {
  uint32_t now   = millis();
  float dt       = (now - lastPidTime) / 1000.0;  // seconds
  lastPidTime    = now;

  // Guard against dt = 0 or a huge first-frame jump
  if (dt <= 0.0 || dt > 0.5) return;

  float error = targetRotations - totalRotations;

  // ── Within tolerance: hold still ──────────────────────────────
  if (abs(error) <= TOLERANCE) {
    currentPWM   = PWM_STOP;
    pidIntegral  = 0.0;       // Reset integral when settled
    lastError    = 0.0;
    currentState = (targetRotations > 5.0) ? RAISED : LOWERED;
    return;
  }

  currentState = MOVING;

  // ── P ─────────────────────────────────────────────────────────
  float P = kP * error;

  // ── I (with anti-windup clamp) ────────────────────────────────
  pidIntegral += error * dt;
  pidIntegral  = constrain(pidIntegral, -INTEGRAL_CLAMP, INTEGRAL_CLAMP);
  float I      = kI * pidIntegral;

  // ── D ─────────────────────────────────────────────────────────
  float derivative = (error - lastError) / dt;
  float D          = kD * derivative;
  lastError        = error;

  // ── Sum and apply deadband ────────────────────────────────────
  float output = P + I + D;

  // Enforce a minimum output so the motor actually moves
  if (output > 0 && output < PWM_DEADBAND)  output =  PWM_DEADBAND;
  if (output < 0 && output > -PWM_DEADBAND) output = -PWM_DEADBAND;
```

```
  currentPWM = constrain((int)(PWM_STOP + output), PWM_MIN, PWM_MAX);
}

// ── Serial Commands ──────────────────────────────────────────────
void checkSerialCommands() {
  if (!Serial.available()) return;

  String input = Serial.readStringUntil('\n');
  input.trim();
  if (input.length() == 0) return;

  char cmd = input.charAt(0);

  if (cmd == 'U' || cmd == 'u') {
    targetRotations = 10.0;
    pidIntegral = 0.0;
    Serial.println("--> Winch: Moving to 10.0");
  }
  else if (cmd == 'D' || cmd == 'd') {
    targetRotations = 0.0;
    pidIntegral = 0.0;
    Serial.println("--> Winch: Moving to 0.0");
  }
  else if (cmd == 'S' || cmd == 's') {
    targetRotations = totalRotations;
    pidIntegral = 0.0;
    Serial.println("--> Winch: EMERGENCY STOP");
  }
  else if (cmd == 'T' || cmd == 't') {
    targetServoPos = (targetServoPos == SERVO_NOMINAL) ? SERVO_ALT :
SERVO_NOMINAL;
    Serial.print("--> Servo toggled to "); Serial.println(targetServoPos);
  }
  // Runtime PID tuning: send e.g. "P350" "I5" "K80"
  else if (cmd == 'P') { kP = input.substring(1).toFloat(); Serial.print("-->
kP = "); Serial.println(kP); }
  else if (cmd == 'I') { kI = input.substring(1).toFloat(); Serial.print("-->
kI = "); Serial.println(kI); }
  else if (cmd == 'K') { kK = input.substring(1).toFloat(); Serial.print("-->
kD = "); Serial.println(kD); }
}
```

```
// ── Encoder ──────────────────────────────────────────────────────────
void updateRotationCount() {
  int currentRaw = encoder.rawAngle();
  int delta      = currentRaw - lastRawAngle;

  if (delta < -2048) delta += 4096;
  else if (delta > 2048) delta -= 4096;

  totalRotations += (float)delta / 4096.0 * -1.0;
  lastRawAngle = currentRaw;
}

// ── Telemetry ────────────────────────────────────────────────────────
void printTelemetry() {
  if (millis() - lastPrintTime < 250) return;
  lastPrintTime = millis();

  const char* stateStr = (currentState == RAISED)  ? "RAISED " :
                         (currentState == LOWERED) ? "LOWERED" : "MOVING ";

  Serial.print("State: ");    Serial.print(stateStr);
  Serial.print(" | Rot: ");   Serial.print(totalRotations, 2);
  Serial.print(" / ");        Serial.print(targetRotations, 2);
  Serial.print(" | Err: ");   Serial.print(targetRotations - totalRotations,
3);
  Serial.print(" | PWM: ");   Serial.print(currentPWM);
  Serial.print(" | Servo: "); Serial.print(targetServoPos);
  Serial.println(" deg");
}
```