# Robotic Fish

Group 11: Seong Jae Ahn, Parjit Khakh, Kobey Cheung, Evan Kuo

**Opportunity:**
Much of the ocean is yet to be explored. Despite the many advances in technology, the ocean's unforgiving environment presents many challenges to exploration. Identifying this as a possible area that we could contribute to, we began researching underwater devices and came across prototypes of robotic fish developed by MIT and ETH Zurich. Inspired by these projects, we decided to construct our own fish with a unique tail propulsion mechanism, capable of surviving harsh salt-water environments for extended periods of time.

**High Level Strategy:**
Our design utilises a tail oscillating in a left-right motion to propel the fish forward. The dorsal fin on top of the fish provides stability, while the pectoral fins rotate to provide up-down rotation, allowing the fish to rotate to swim up or down.

Initially, we were targeting a fully waterproofed device capable of maintaining its position in the water when idling. However, due to time constraints and unexpected difficulties encountered in developing the tail mechanism, we were unable to test any form of underwater movement. Therefore, we relied on research papers to better understand the desired tail rotation speed that we would need. Our desired tail speed ended up being 60 RPM (1 Hz), and we were able to achieve that with a slight voltage buffer in case of any unexpected increases in torque.

**Photos:**

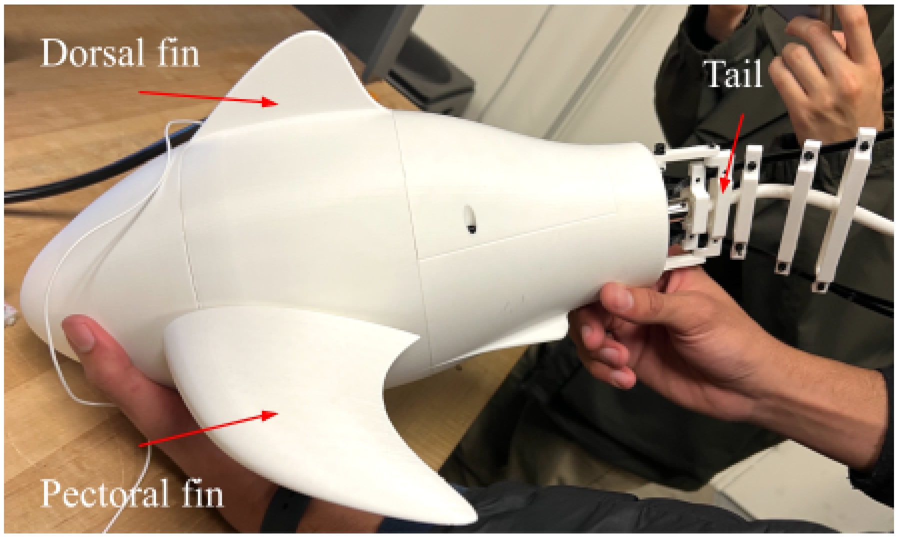


*Figure 1: Full Assembly*

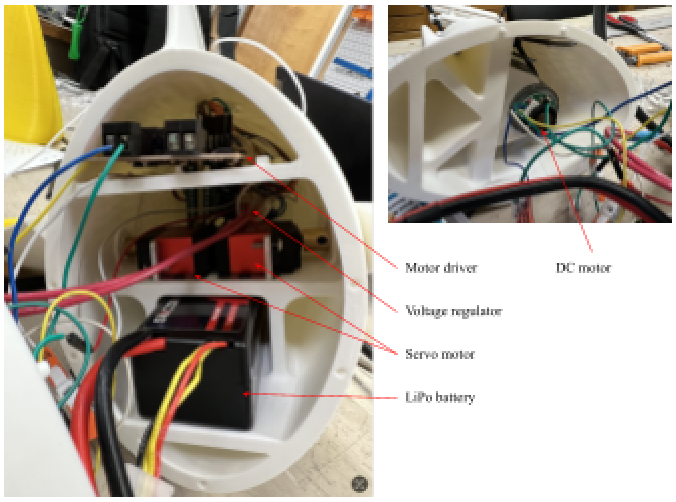


*Figure 2: Fish interior*

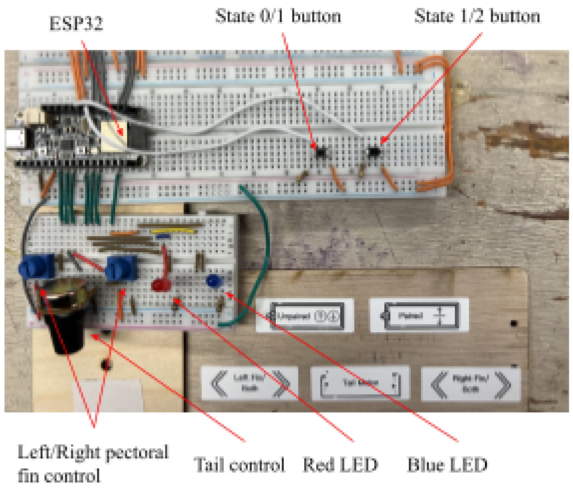


*Figure 3: Control board*

**Function-critical design and calculations:**
The function critical decision in selecting the proper motor was to ensure it was capable of supplying at least 60 RPM and the necessary torque for the tail motion. The calculations below show that even at a third of the maximum voltage level, the motor is capable of achieving our desired specifications.

<u>Torque for fish tail oscillation:</u>
Desired rotation speed: 1Hz → 60 RPM (source)
Stall torque: 18 kg cm
No load rotation: 251 RPM
Voltage input for 60 RPM: 4V (Max: 12V)
Assume linear relationship between stall torque and voltage: 18 (4/12) = 6 kg cm
Assume linear relationship between no load rotation and voltage: 251 (4/12) = 83.67 RPM
Linear approx. → RPM = -13.94 (Torque) + 83.67
Torque: (60 - 83.67)/(-13.94) = **1.7 kg cm → 0.17 Nm**

The motion and length of the tail results in a high radial load under its own weight and any reaction forces from the water being pushed to provide propulsion. Given that we are focussing on a table top test first, we will focus on the loads due to the tail's weight. Despite the spiral tail rod being catelivered, the calculations below show that our system with two bearings can adequately support the tail.

<u>Tail bearing radial load:</u>
Bearing 1 & 2:
Approx. shape as triangle: $r_{tail} = ⅔ L_{tail}$
$r_{12}$ is the average distance for bearings 1 and 2
$F_{B1+2} = W_{tail}(r_{tail})/r_{1+2} = 9.81(0.23)(⅔\ 0.228)/(0.05) = 6.86N$ (↑)
With two bearings used at close proximity, force is assumed to be roughly distributed in half. Each bearing has an approximate radial load of: **3.43N (↑)**

Given that we simply needed to actuate the pectoral fins up or down, we opted to minimize the complexity of the fin transmission and omitted bearings. This direct drive method is sufficient for our needs. The calculations below demonstrate that the loads on the servo shaft coupler are quite small.

Pectoral fin radial load on coupler:
Coupler:
$L_{rod} = 0.05m$
$r_{rod} = L_{rod} / 2$
$F_{radial,\,c} = [W_{fin}(L_{rod}) + W_{rod}(r_{rod}/2)]/r_c = 9.81[(0.03)(0.05) + 0.08(0.05/2)]/(0.025) = \mathbf{1.37N\ (\uparrow)}$

**Circuit:**

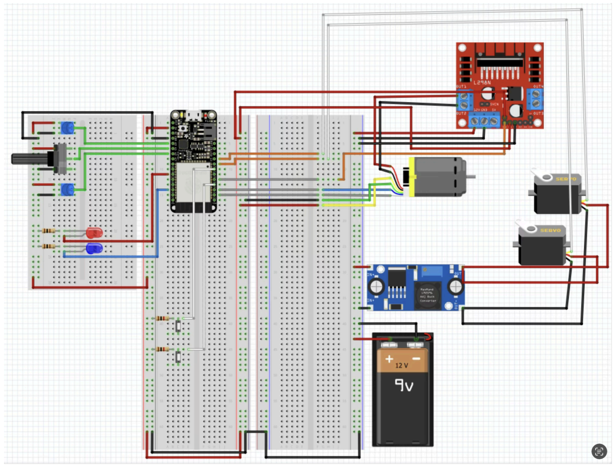
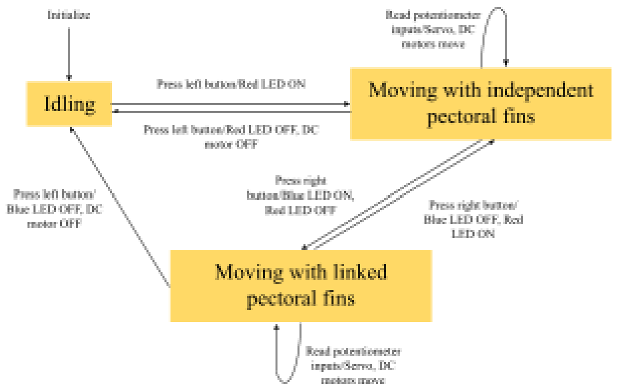


*Figure 4: Circuit Diagram*

**State Transition Diagram:**



Initialize
Idling
Press left button/Red LED ON
Moving with independent pectoral fins
Read potentiometer inputs/Servo, DC motors move
Press left button/Red LED OFF, DC motor OFF
Press right button/Blue LED ON, Red LED OFF
Press right button/ Blue LED OFF, Red LED ON
Press left button/ Blue LED OFF, DC motor OFF
Moving with linked pectoral fins
Read potentiometer inputs/Servo, DC motors move

*Figure 5: State Transition Diagram*

**Reflection:**
We maintained great communication, ensuring we were all aware of what everyone was working on at all times and helping each other when needed. Dividing up the project so that we were all working on parts that we were most comfortable with also helped speed up the design and construction process and allowed each of us to play to our strengths.

Addressing the hurdles encountered throughout the semester, we could have managed our time more efficiently. We were inefficient at printing and testing our design, putting too much focus on perfecting the model before printing. Thus, we neglected to conduct waterproof testing, which could have been completed in parallel. Once it came to the functionality exam, the lack of previous waterproof testing meant that we were unwilling to take the risk of putting our device in water right before such a critical moment, something that we were greatly disappointed by. Despite this setback, overall, we had great fun working together to complete this project and were happy with the results.

## Appendix A: Bill of Materials

| Item Name | Serial Number / SKU | Quantity | Vendor | Link to Item | | Purchase Total |
|---|---|---|---|---|---|---|
| | | | | | | |
| OVERTURE PLA Filament | B07PGZNM34 | 2 | Amazon | https://www.amazon.com/dp | | $ 18.99 |
| Servo | B0C5LWHTQ1 | 2 | Amazon | https://www.amazon.com/M | | $ 32.35 |
| Shaft Coupling | NA | 1 | Amazon | https://www.amazon.com/dp | | $ 10.93 |
| Flexible Coupling | 631105456623 | 1 | Amazon | https://www.amazon.com/gp/pro | | $ 11.04 |
| ESP32 | NA | 1 | Amazon | https://www.amazon.com/gp/pro | | $ 8.83 |
| zip tie | NA | 1 | Amazon | https://www.amazon.com/dp | | $ 7.72 |
| zip tie holder | NA | 1 | Amazon | https://www.amazon.com/dp | | $ 11.04 |
| 18AWG PVC copper wire | B0CGR6CC75 | 1 | Amazon | https://www.amazon.com/dp | | $ 69.98 |
| 18AWG wires | B089CRCXRK | 1 | Amazon | https://www.amazon.com/dp | | $ 11.99 |
| M3 screws and nuts | B0B51BFSWZ | 1 | Amazon | https://www.amazon.com/dp | | $ 7.99 |
| Servo to Shaft Coupler | 4010-0025-0250 | 1 | ServoCity | https://www.servocity.com/se | | $ 12.49 |
| Servo to shaft coupler | 4010-0025-0250 | 1 | ServoCity | https://www.servocity.com/se | | $ 14.17 |
| M3 Screw Kit | | 1 | Amazon | https://www.amazon.com/dp | | $ 7.45 |
| 8mm x 150mm Shaft | B0BNL57QGL | 1 | Amazon | https://www.amazon.com/dp | | $ 6.57 |
| 6mm Dia Round Aluminum Rod | B0D2VW39XX | 1 | Amazon | https://www.amazon.com/dp | | $ 6.03 |
| M8x12mm Dia. Steel Shim Pack | B0BT5THCCZ | 1 | Amazon | https://www.amazon.com/dp | | $ 11.47 |
| 8mm Shaft Lock Collar | B0D17CLMJM | 1 | Amazon | https://www.amazon.com/gp | | $ 7.99 |
| Belleville Washer | B0C3D3N7NG | 1 | Amazon | https://www.amazon.com/gp | | $ 9.99 |
| M3 Threaded Inserts | B0CNVXWQYZ | 1 | Amazon | https://www.amazon.com/gp | | $ 4.99 |
| DC Motor Driver | B06XGD5SCB | 1 | Amazon | https://www.amazon.com/dp | | $ 17.55 |
| RC Servo Mount Bracket | B0CD5X74PQ | 1 | Amazon | https://drive.google.com/file/ | | $ 13.50 |
| Metric Rotary Shaft Oil Seal | B011RB5RPS | 1 | Amazon | https://www.amazon.com/gp | | $ 8.22 |
| Pololu Metal Gearmotors » 37D Metal | 4752 | 1 | Polou | https://www.pololu.com/prod | | $ 63.44 |
| | | | | | **TOTAL:** | **$ 299.75** |

## Appendix B: CAD images of mechanical transmission elements

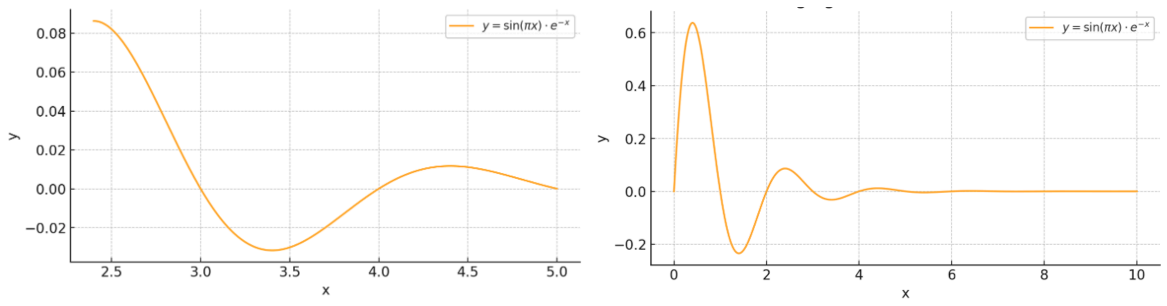


*Figure 6: Mathematical equation for the tail design*

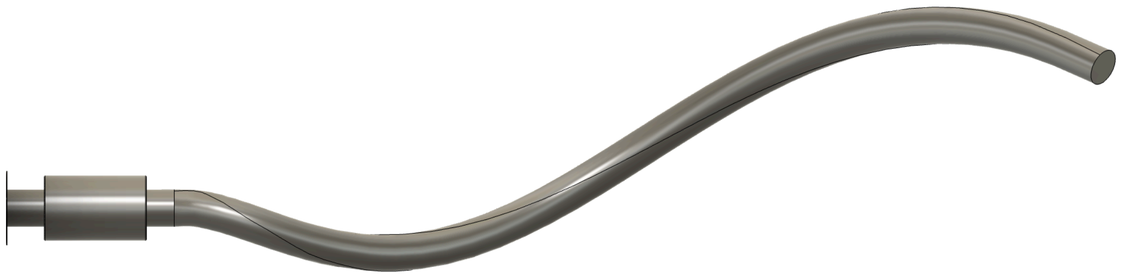


*Figure 7: CAD Based on Tail Equation*

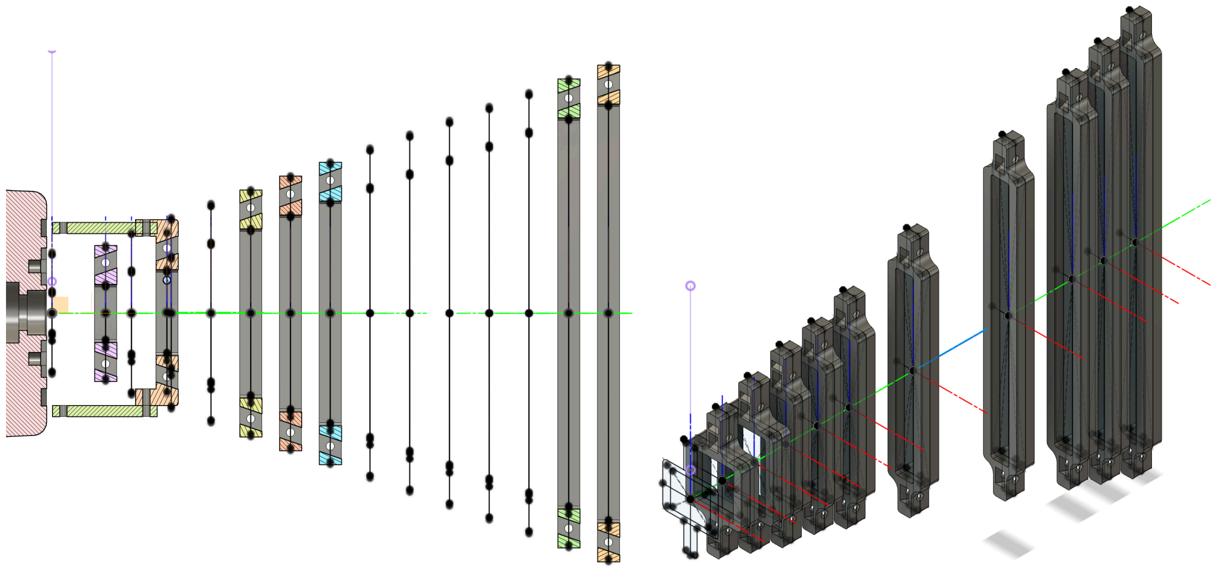


*Figure 8: Front View of the Tail, Rod Mounts and Isometric View of the Rod Mounts*

The vertical rods shown in Figure 8 accommodate a zip tie that passes through a slot on either end of each rod. The zip tie was drilled into and fixed to each rod using nuts and screws. Zip ties provide stiffness along their length but are flexible laterally, making them the ideal choice for our compliant mechanism.

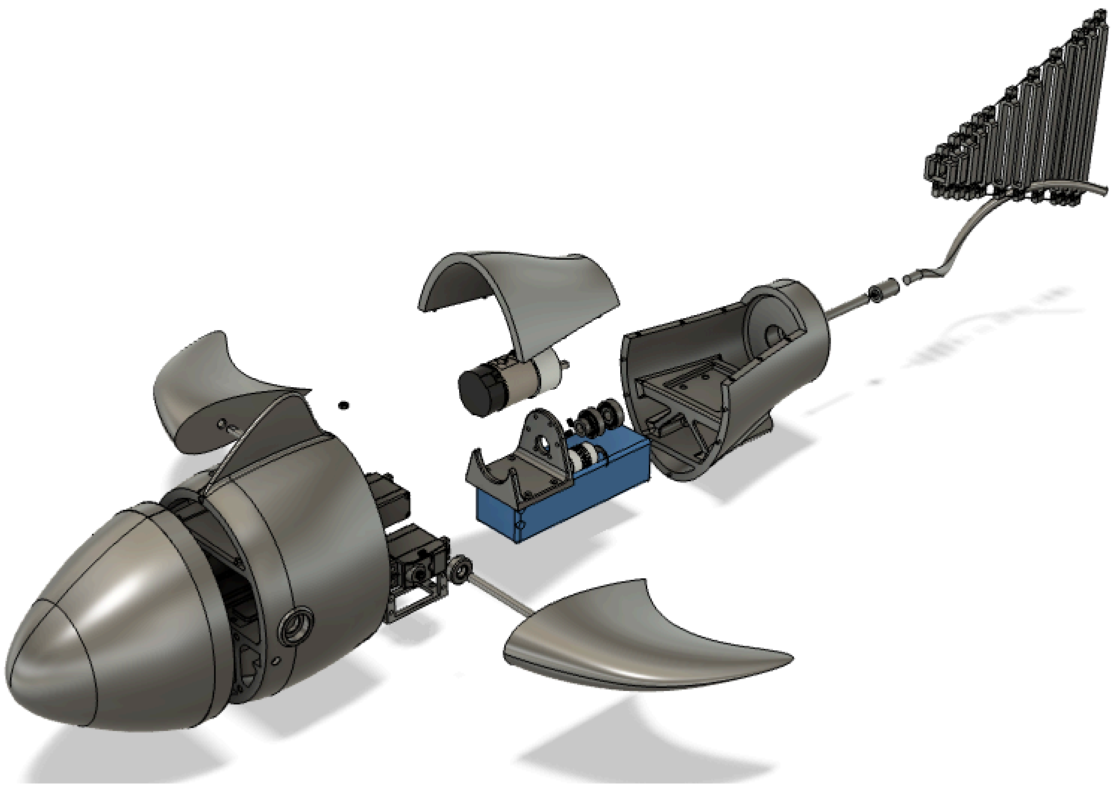


*Figure 9: CAD - Full Assembly*

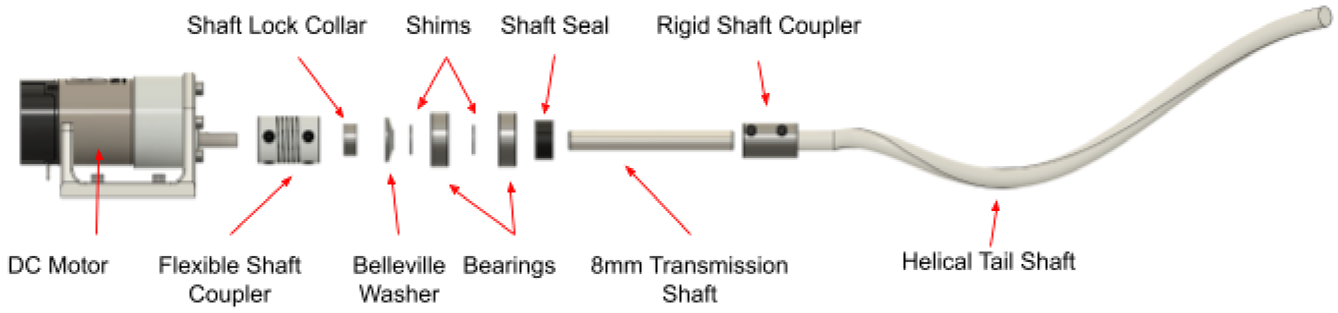


*Figure 10: Mechanical Transmission Elements*

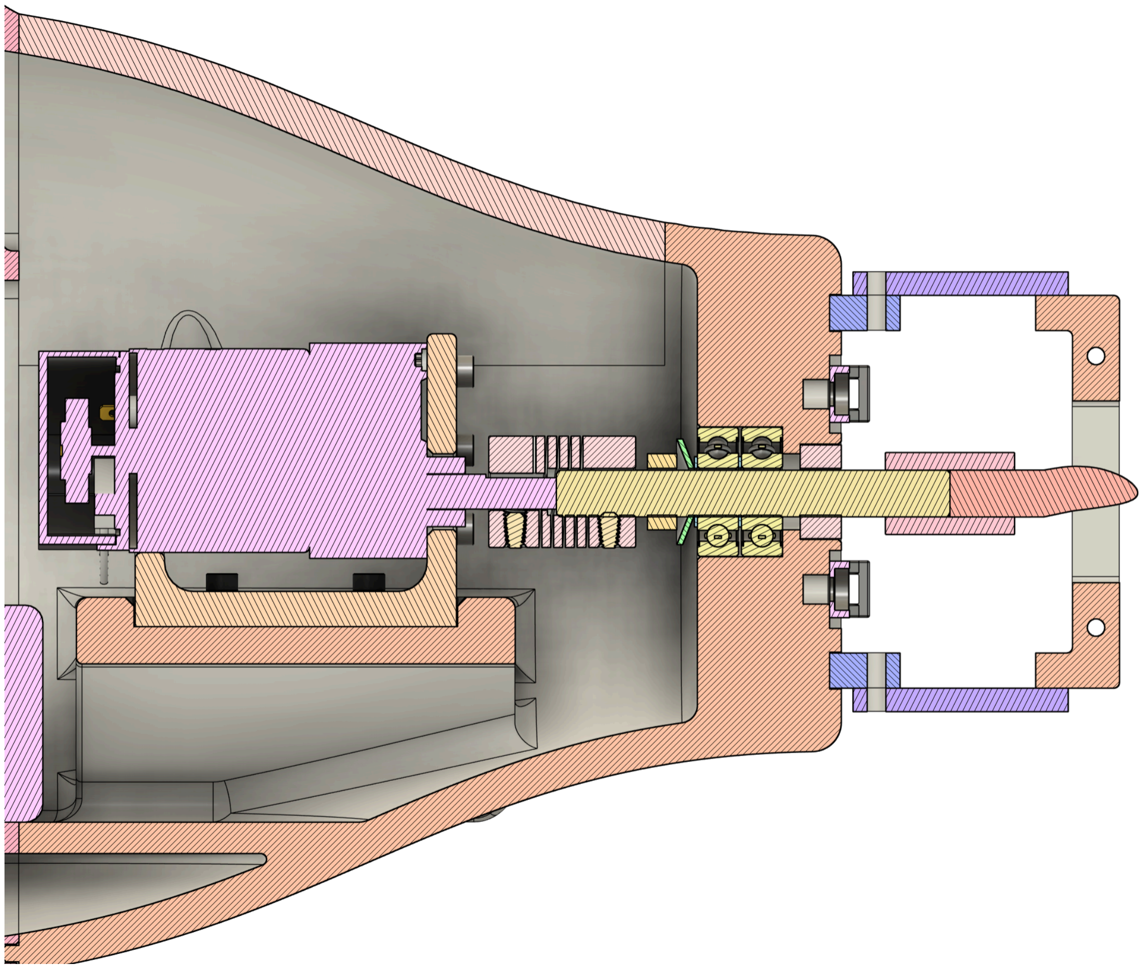


*Figure 11: Section View of Body - Transmission Elements and Linkage*

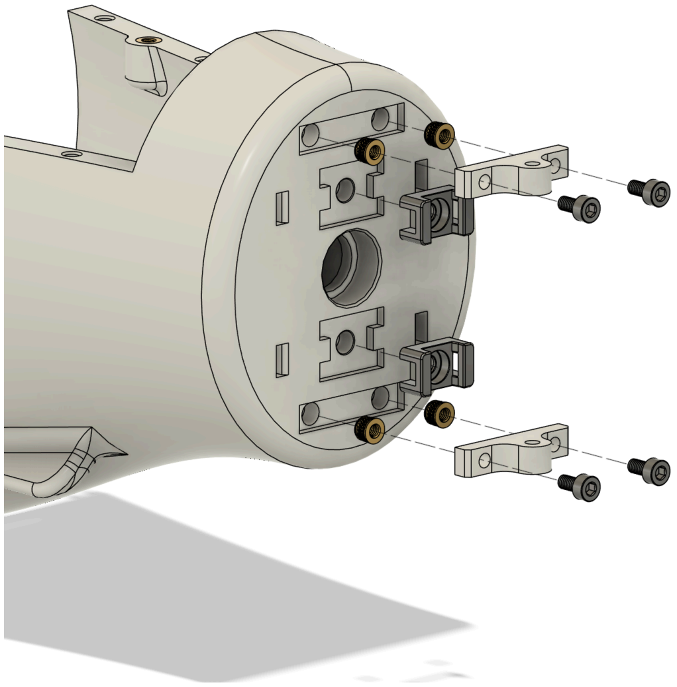
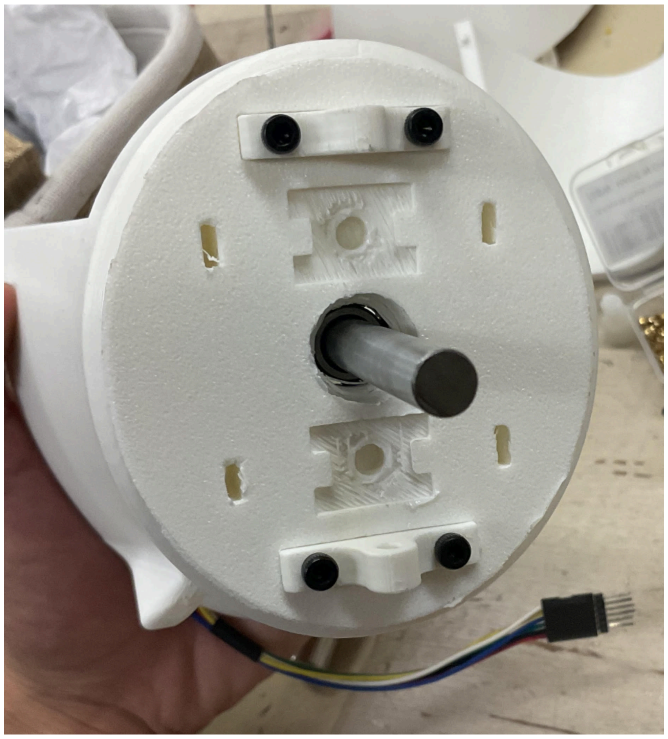



*Figure 12: Tail Mounting Sub-Assembly*

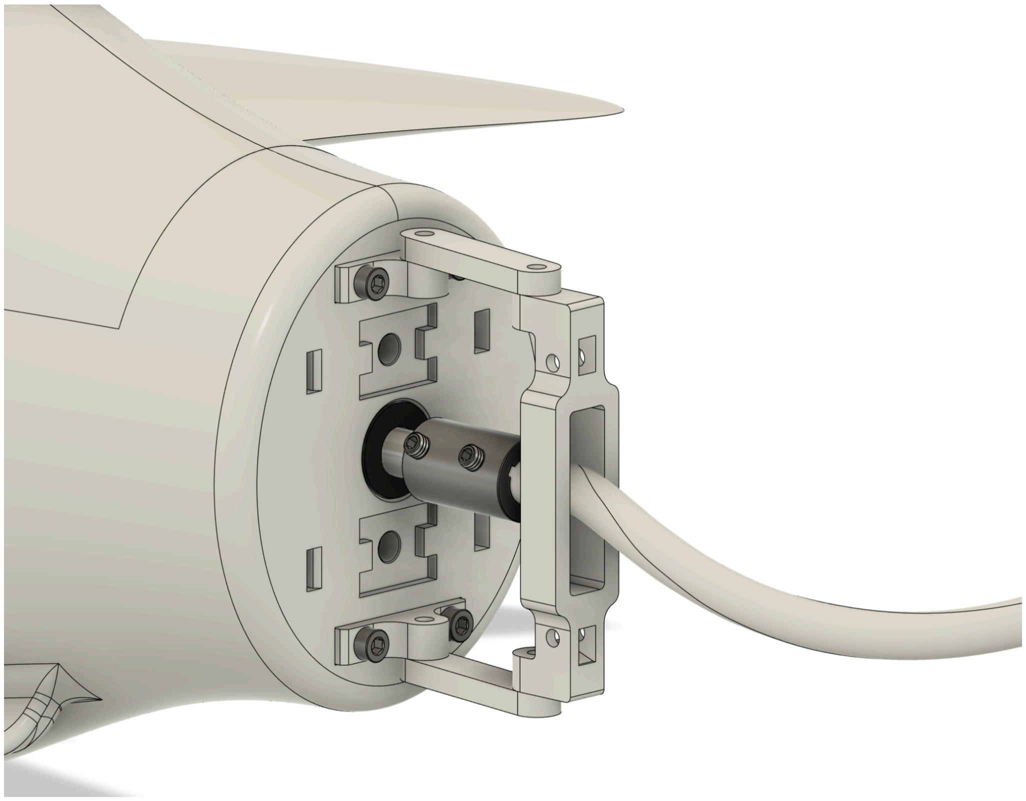


*Figure 13: Passive Linkage to Hold Tail Upright*

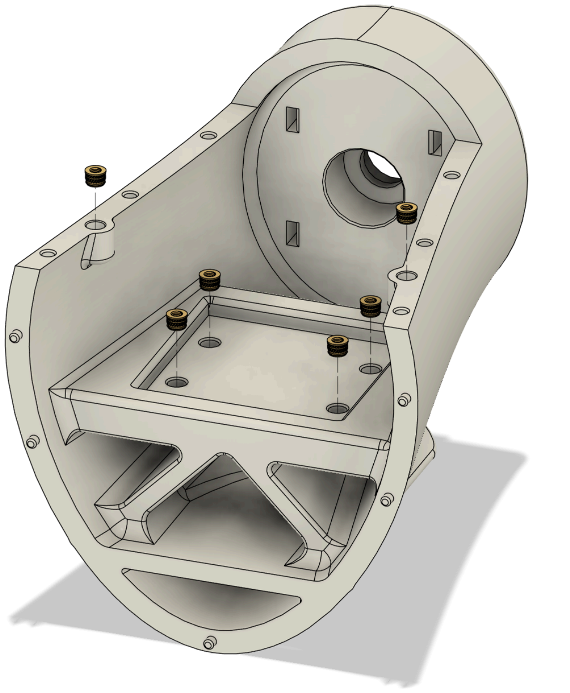
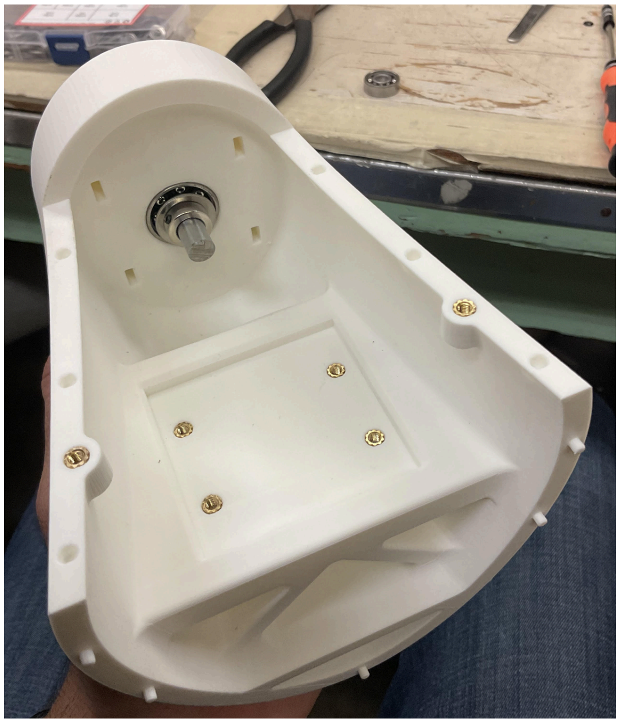



*Figure 14: Heat Set Inserts for Motor Mount*

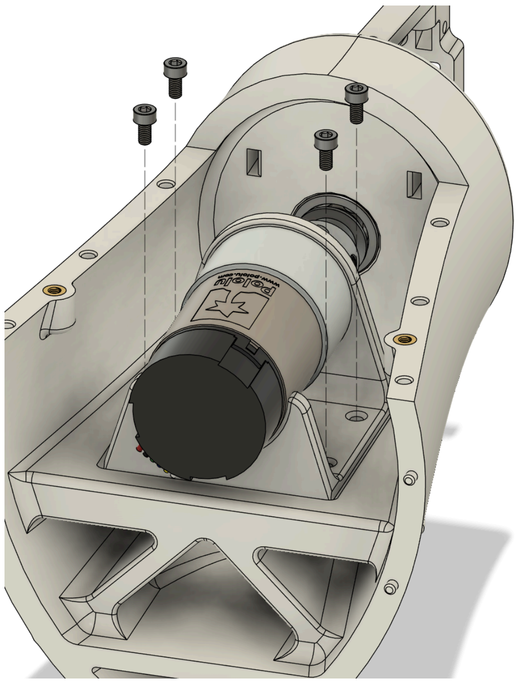
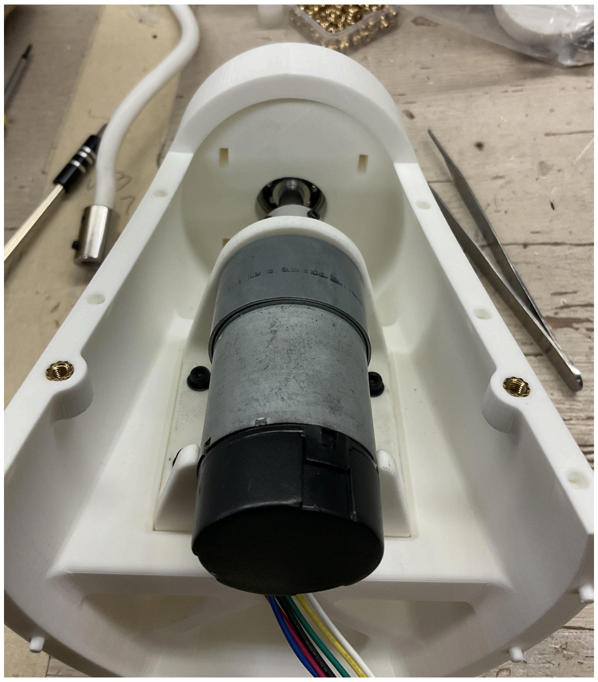



*Figure 15: Mounted DC Motor with M3 Screws*

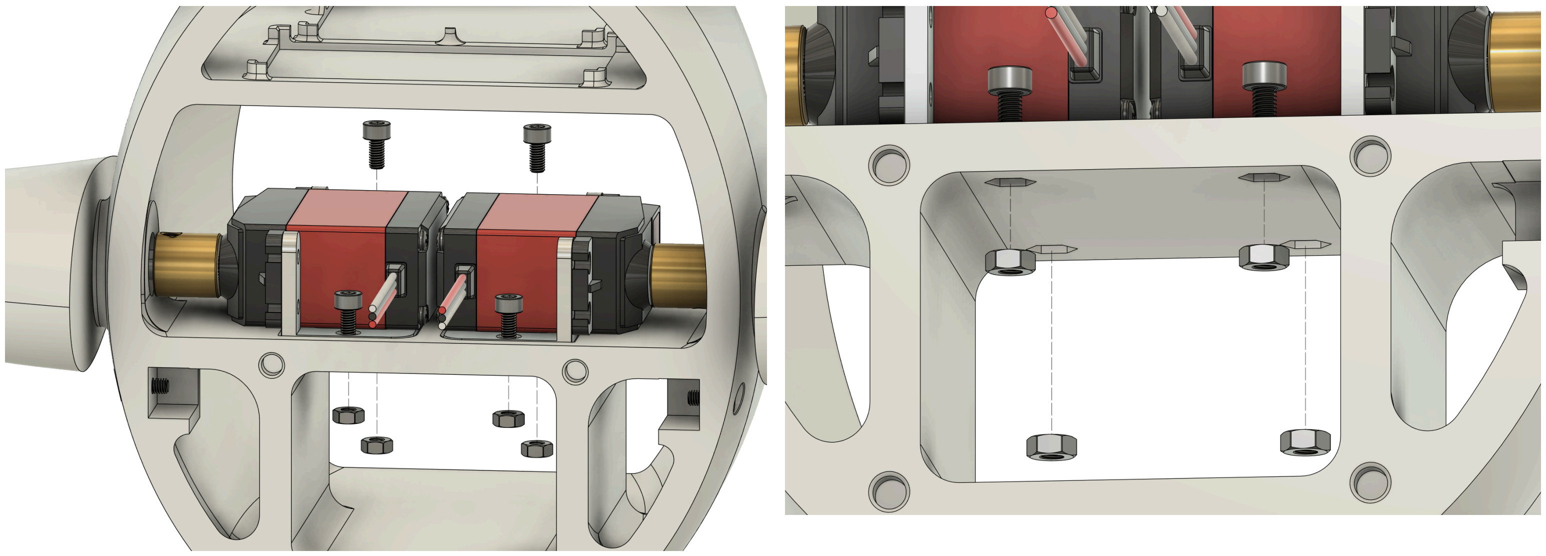


*Figure 16: Mounting Scheme for Servo Motors*

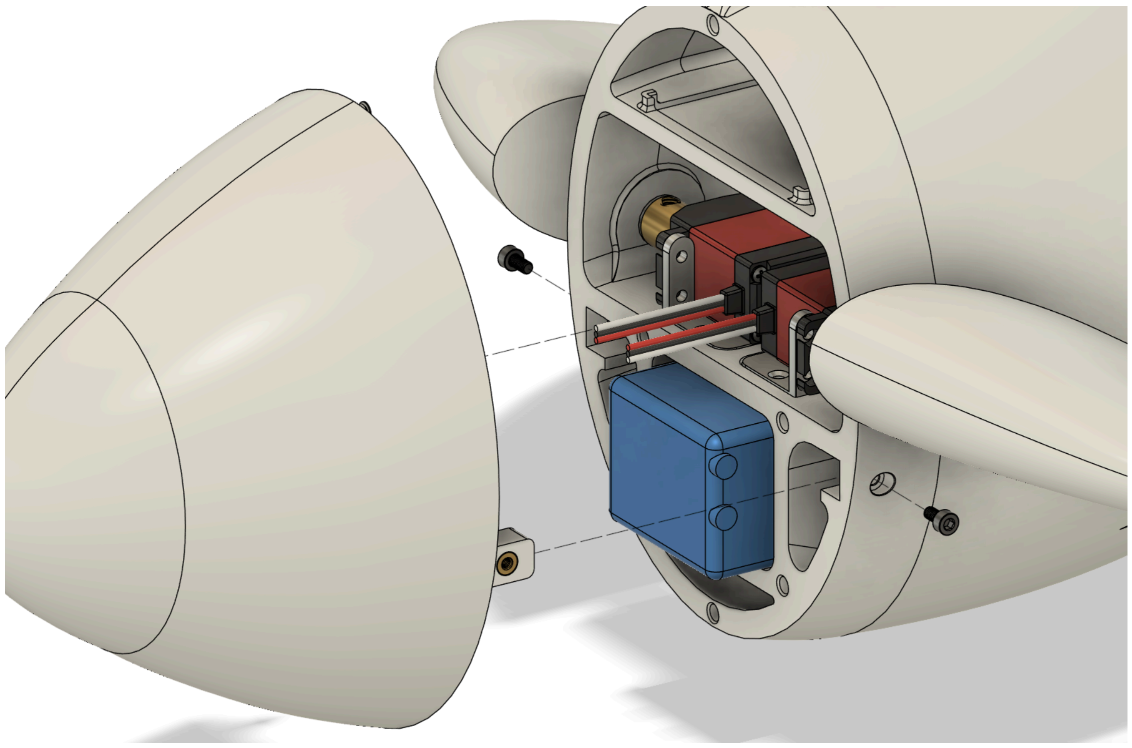


*Figure 17: Nose Cone Connection with M3 Screws*

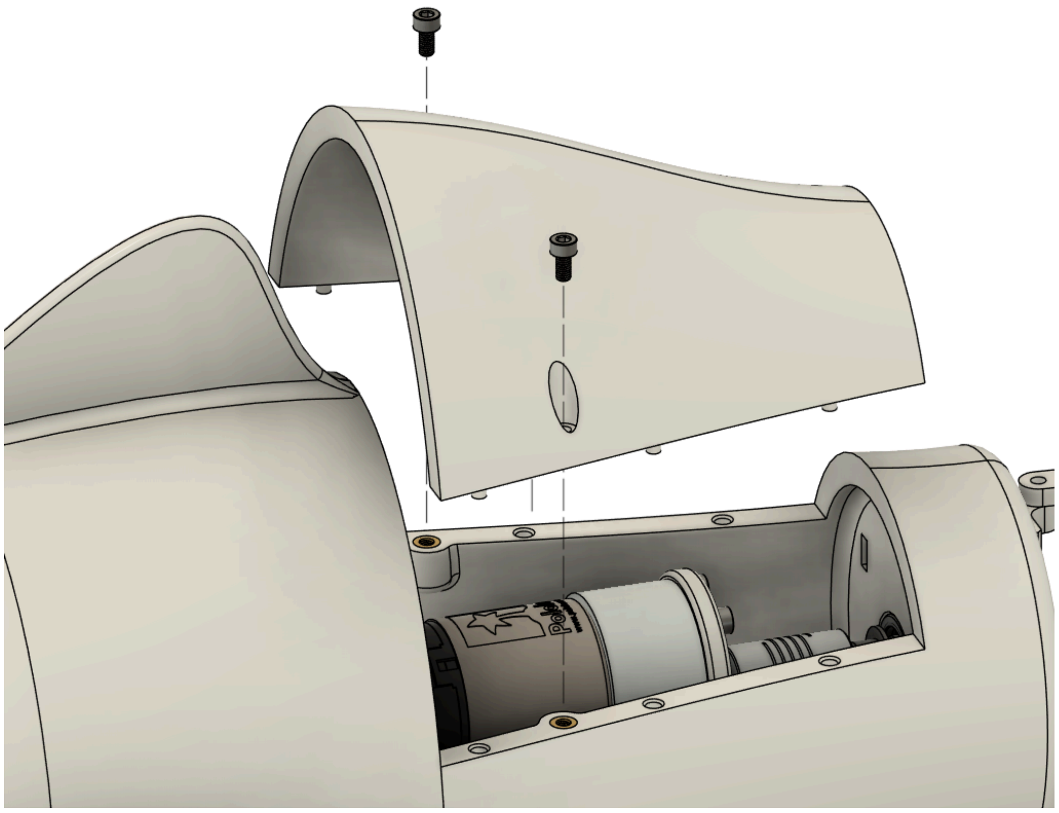


*Figure 18: Top Access Panel*

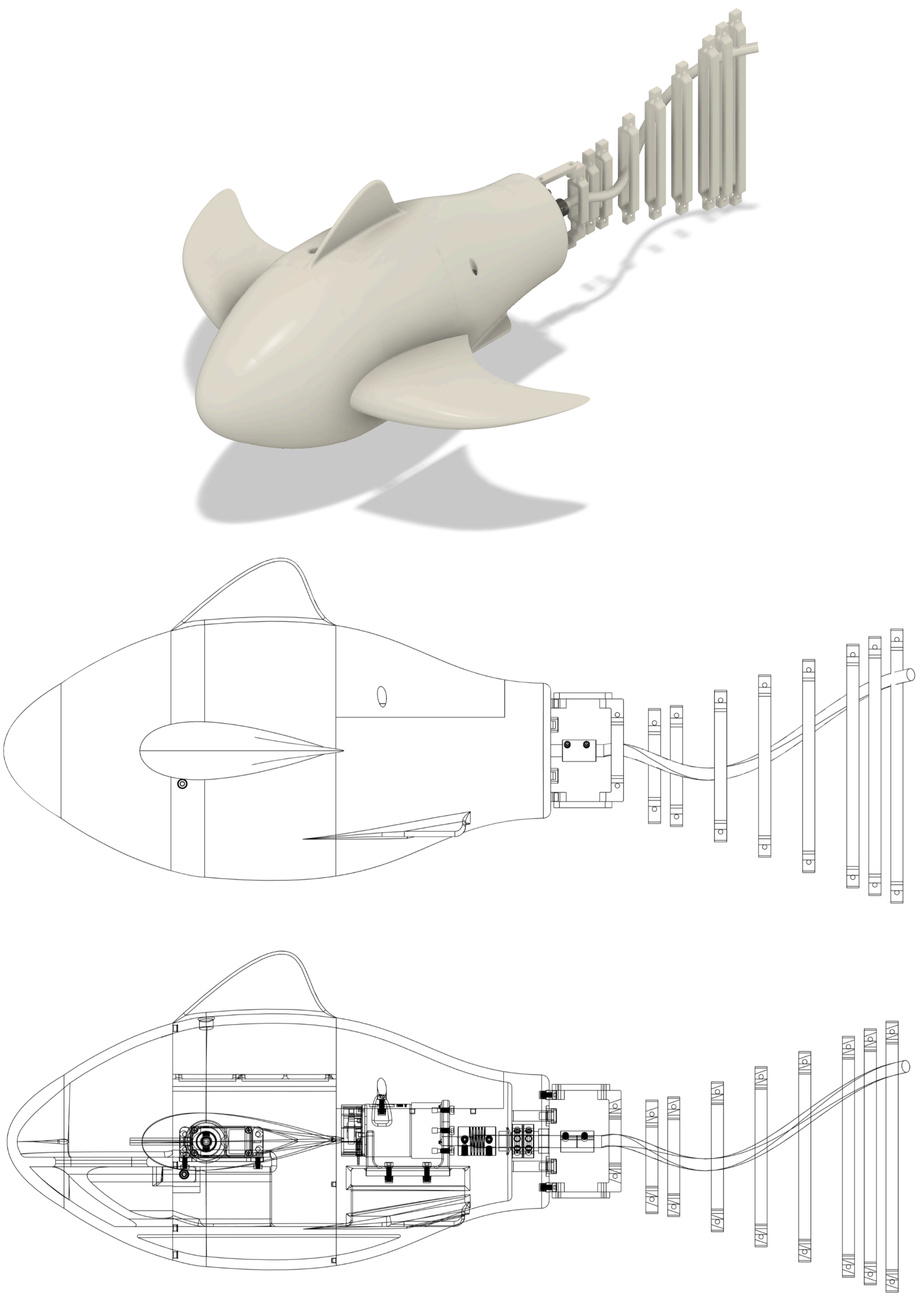
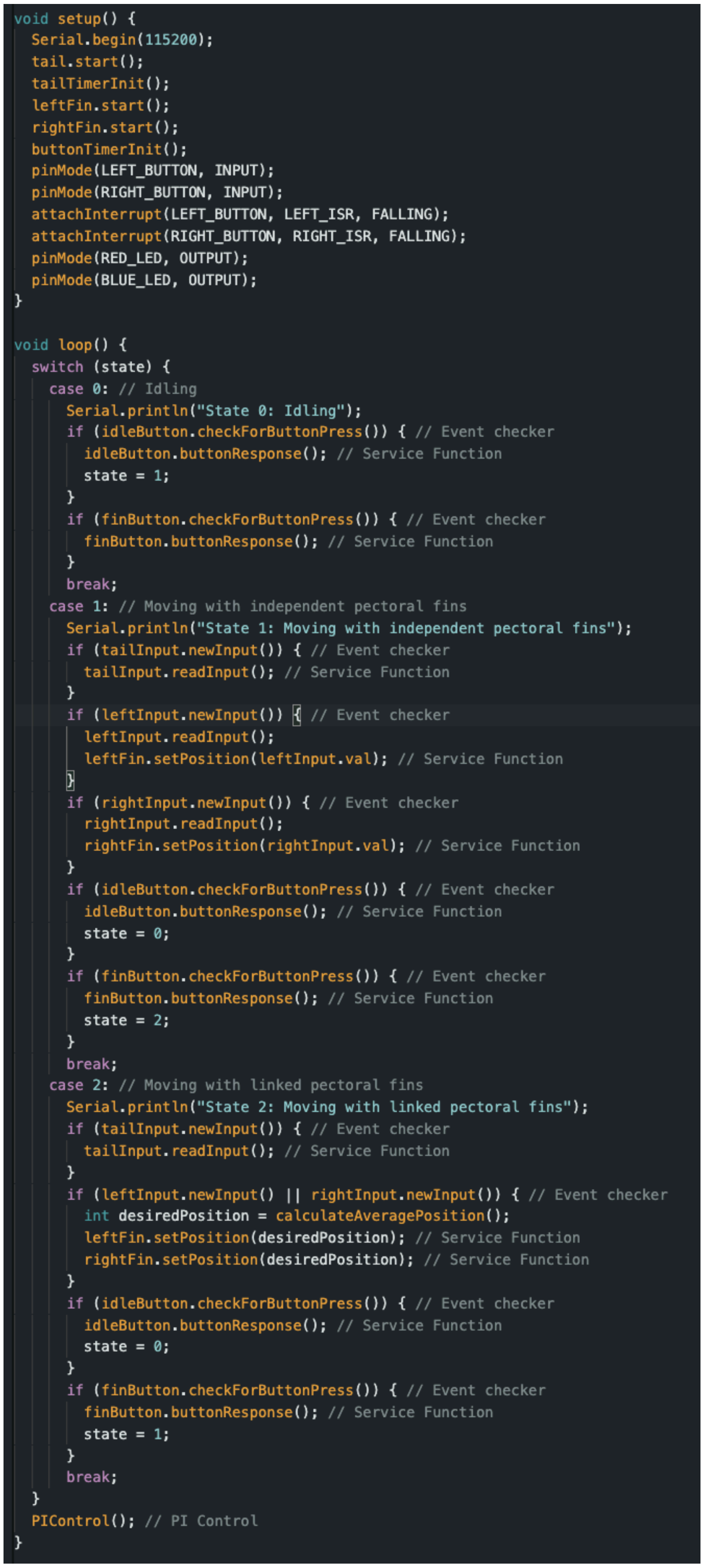
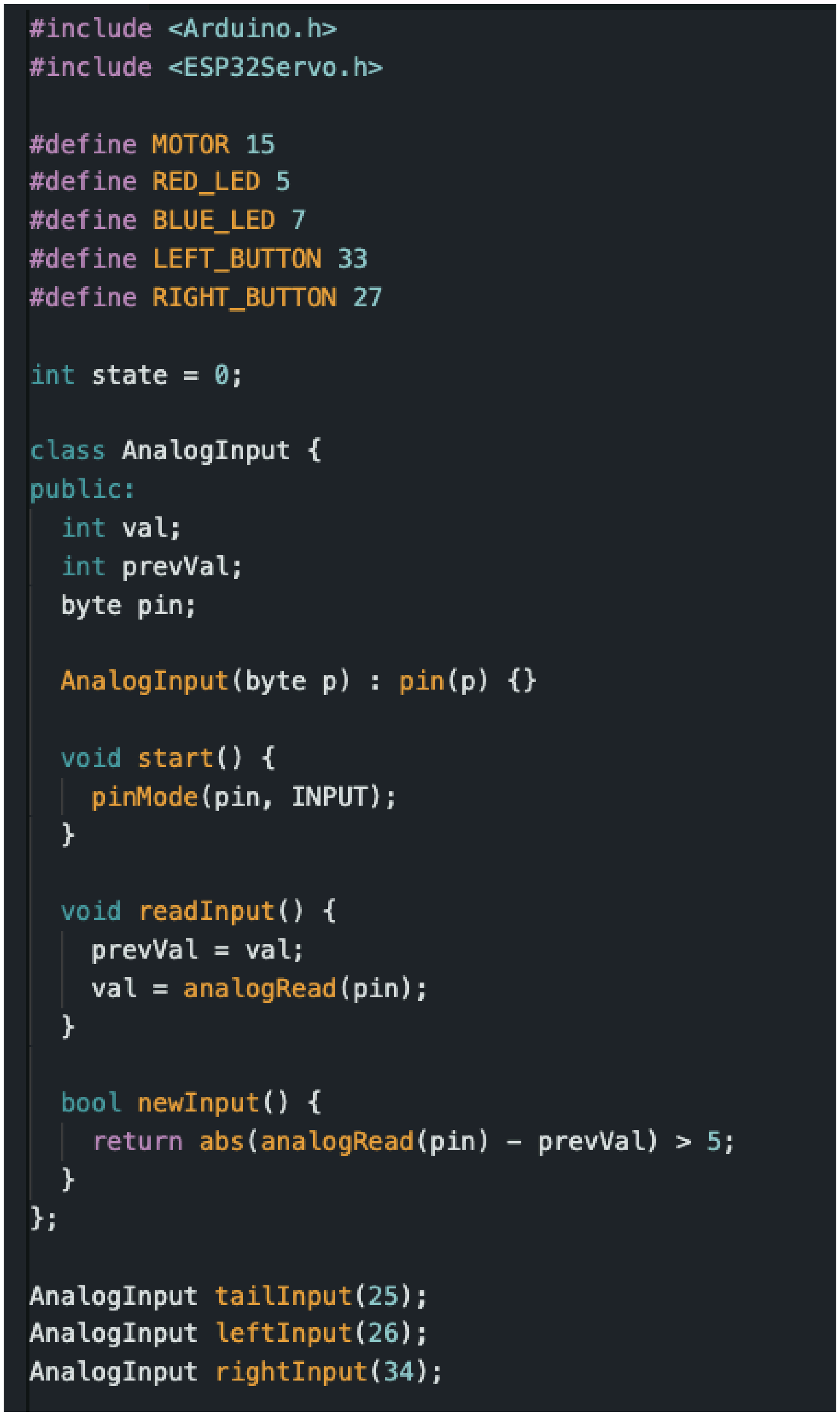


*Figure 18: Overview of Complete Design*

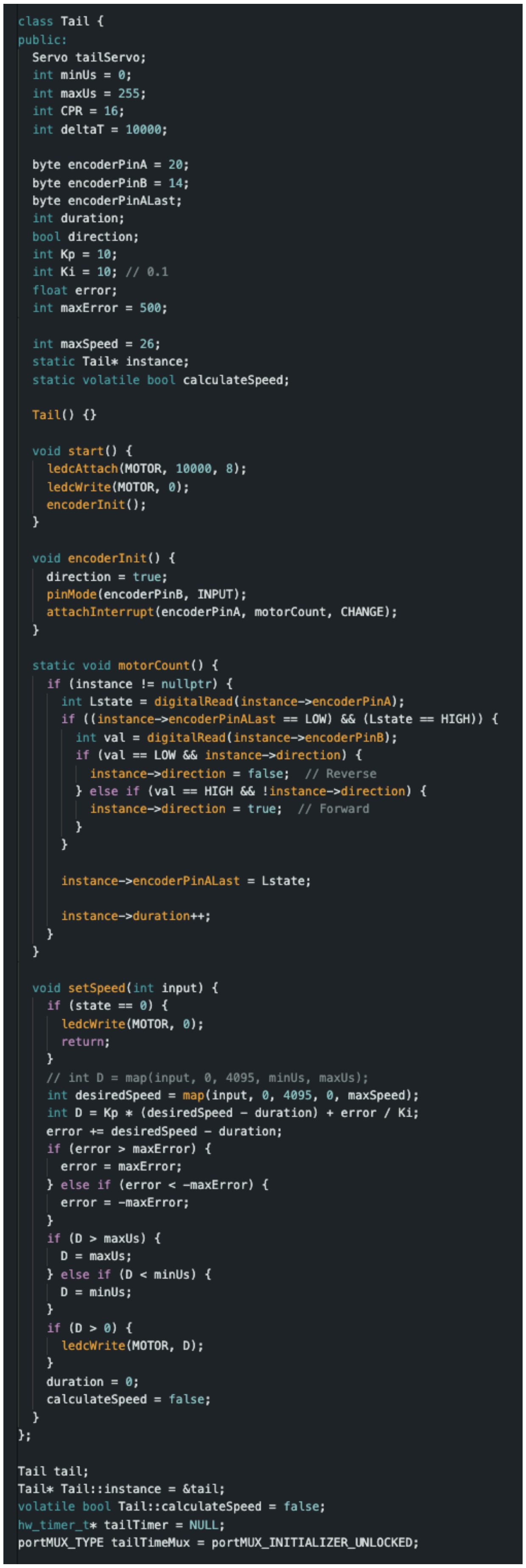
## Appendix C: Arduino IDE Code



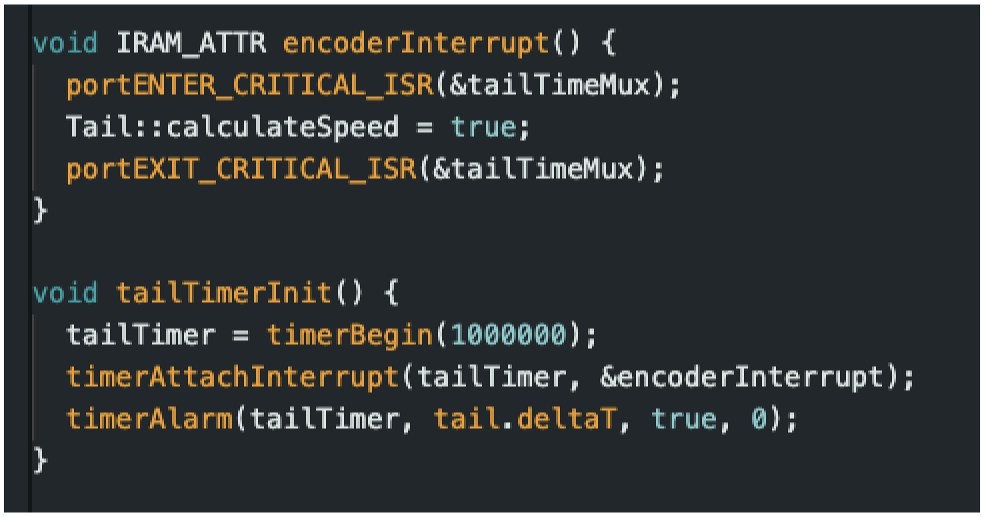
```
void setup() {
  Serial.begin(115200);
  tail.start();
  tailTimerInit();
  leftFin.start();
  rightFin.start();
  buttonTimerInit();
  pinMode(LEFT_BUTTON, INPUT);
  pinMode(RIGHT_BUTTON, INPUT);
  attachInterrupt(LEFT_BUTTON, LEFT_ISR, FALLING);
  attachInterrupt(RIGHT_BUTTON, RIGHT_ISR, FALLING);
  pinMode(RED_LED, OUTPUT);
  pinMode(BLUE_LED, OUTPUT);
}

void loop() {
  switch (state) {
    case 0: // Idling
      Serial.println("State 0: Idling");
      if (idleButton.checkForButtonPress()) { // Event checker
        idleButton.buttonResponse(); // Service Function
        state = 1;
      }
      if (finButton.checkForButtonPress()) { // Event checker
        finButton.buttonResponse(); // Service Function
      }
      break;
    case 1: // Moving with independent pectoral fins
      Serial.println("State 1: Moving with independent pectoral fins");
      if (tailInput.newInput()) { // Event checker
        tailInput.readInput(); // Service Function
      }
      if (leftInput.newInput()) { // Event checker
        leftInput.readInput();
        leftFin.setPosition(leftInput.val); // Service Function
      }
      if (rightInput.newInput()) { // Event checker
        rightInput.readInput();
        rightFin.setPosition(rightInput.val); // Service Function
      }
      if (idleButton.checkForButtonPress()) { // Event checker
        idleButton.buttonResponse(); // Service Function
        state = 0;
      }
      if (finButton.checkForButtonPress()) { // Event checker
        finButton.buttonResponse(); // Service Function
        state = 2;
      }
      break;
    case 2: // Moving with linked pectoral fins
      Serial.println("State 2: Moving with linked pectoral fins");
      if (tailInput.newInput()) { // Event checker
        tailInput.readInput(); // Service Function
      }
      if (leftInput.newInput() || rightInput.newInput()) { // Event checker
        int desiredPosition = calculateAveragePosition();
        leftFin.setPosition(desiredPosition); // Service Function
        rightFin.setPosition(desiredPosition); // Service Function
      }
      if (idleButton.checkForButtonPress()) { // Event checker
        idleButton.buttonResponse(); // Service Function
        state = 0;
      }
      if (finButton.checkForButtonPress()) { // Event checker
        finButton.buttonResponse(); // Service Function
        state = 1;
      }
      break;
  }
  PIControl(); // PI Control
}
```



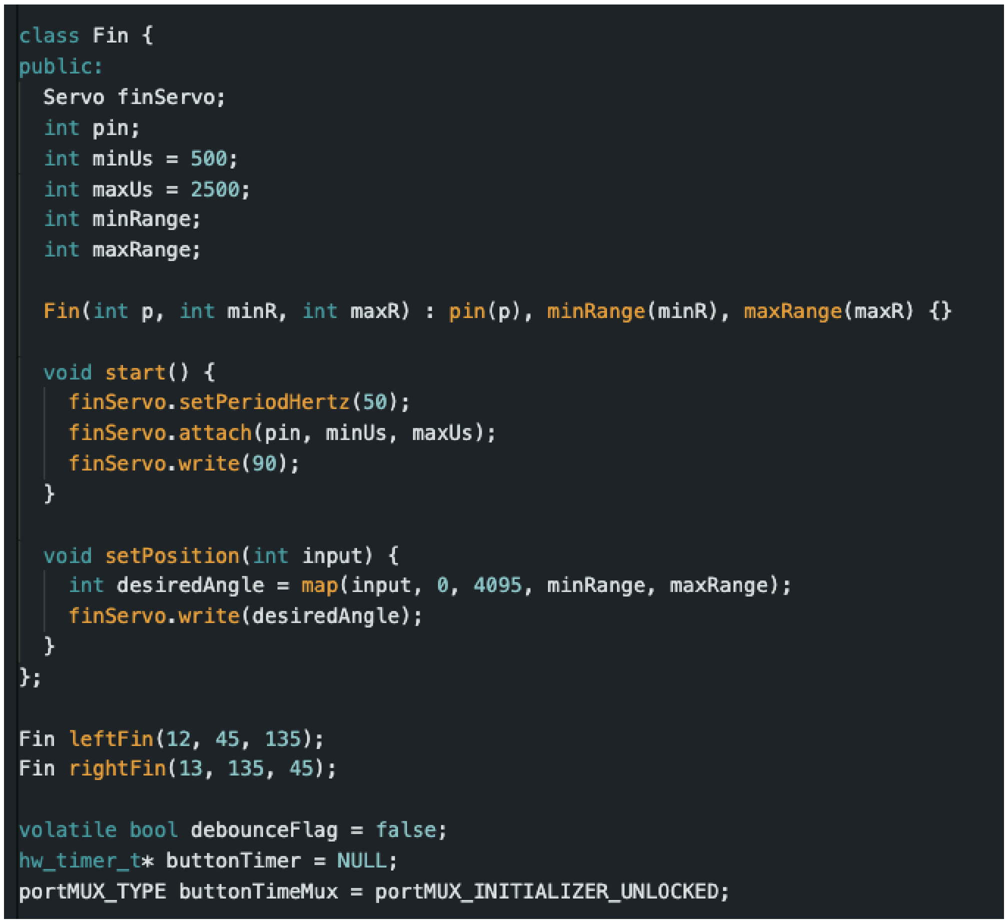
```
#include <Arduino.h>
#include <ESP32Servo.h>

#define MOTOR 15
#define RED_LED 5
#define BLUE_LED 7
#define LEFT_BUTTON 33
#define RIGHT_BUTTON 27

int state = 0;

class AnalogInput {
public:
  int val;
  int prevVal;
  byte pin;

  AnalogInput(byte p) : pin(p) {}

  void start() {
    pinMode(pin, INPUT);
  }

  void readInput() {
    prevVal = val;
    val = analogRead(pin);
  }

  bool newInput() {
    return abs(analogRead(pin) - prevVal) > 5;
  }
};

AnalogInput tailInput(25);
AnalogInput leftInput(26);
AnalogInput rightInput(34);
```



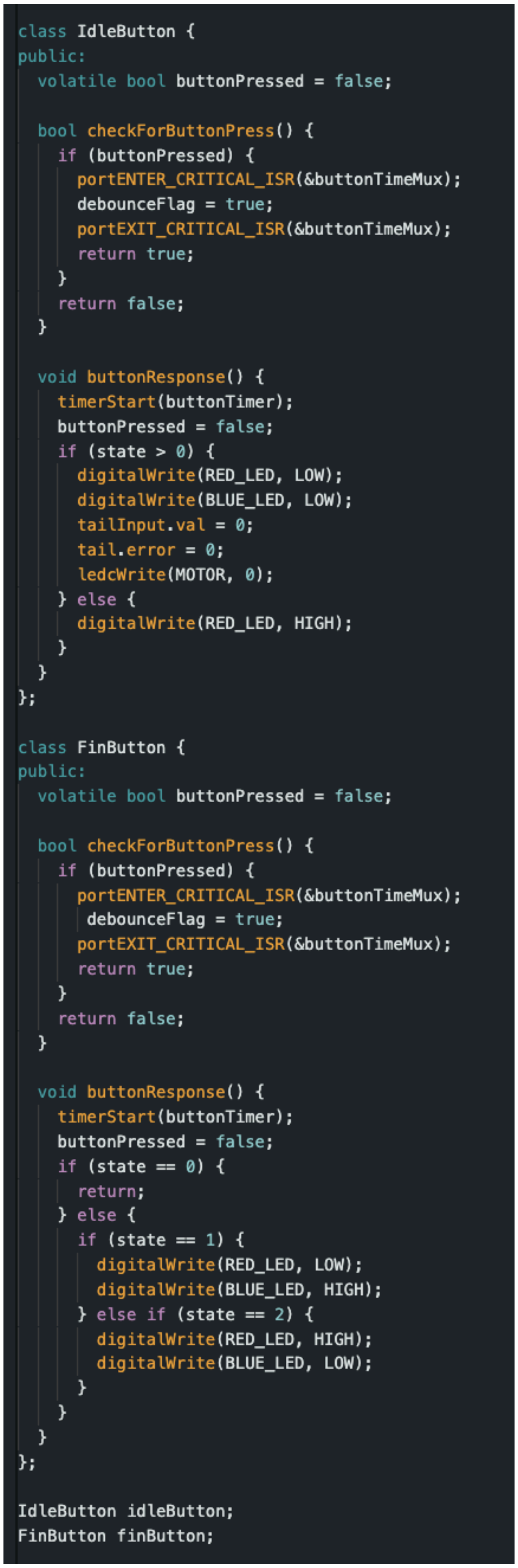
```cpp
class Tail {
public:
  Servo tailServo;
  int minUs = 0;
  int maxUs = 255;
  int CPR = 16;
  int deltaT = 10000;

  byte encoderPinA = 20;
  byte encoderPinB = 14;
  byte encoderPinALast;
  int duration;
  bool direction;
  int Kp = 10;
  int Ki = 10; // 0.1
  float error;
  int maxError = 500;

  int maxSpeed = 26;
  static Tail* instance;
  static volatile bool calculateSpeed;

  Tail() {}

  void start() {
    ledcAttach(MOTOR, 10000, 8);
    ledcWrite(MOTOR, 0);
    encoderInit();
  }

  void encoderInit() {
    direction = true;
    pinMode(encoderPinB, INPUT);
    attachInterrupt(encoderPinA, motorCount, CHANGE);
  }

  static void motorCount() {
    if (instance != nullptr) {
      int Lstate = digitalRead(instance->encoderPinA);
      if ((instance->encoderPinALast == LOW) && (Lstate == HIGH)) {
        int val = digitalRead(instance->encoderPinB);
        if (val == LOW && instance->direction) {
          instance->direction = false;  // Reverse
        } else if (val == HIGH && !instance->direction) {
          instance->direction = true;  // Forward
        }
      }

      instance->encoderPinALast = Lstate;

      instance->duration++;
    }
  }

  void setSpeed(int input) {
    if (state == 0) {
      ledcWrite(MOTOR, 0);
      return;
    }
    // int D = map(input, 0, 4095, minUs, maxUs);
    int desiredSpeed = map(input, 0, 4095, 0, maxSpeed);
    int D = Kp * (desiredSpeed - duration) + error / Ki;
    error += desiredSpeed - duration;
    if (error > maxError) {
      error = maxError;
    } else if (error < -maxError) {
      error = -maxError;
    }
    if (D > maxUs) {
      D = maxUs;
    } else if (D < minUs) {
      D = minUs;
    }
    if (D > 0) {
      ledcWrite(MOTOR, D);
    }
    duration = 0;
    calculateSpeed = false;
  }
};

Tail tail;
Tail* Tail::instance = &tail;
volatile bool Tail::calculateSpeed = false;
hw_timer_t* tailTimer = NULL;
portMUX_TYPE tailTimeMux = portMUX_INITIALIZER_UNLOCKED;
```



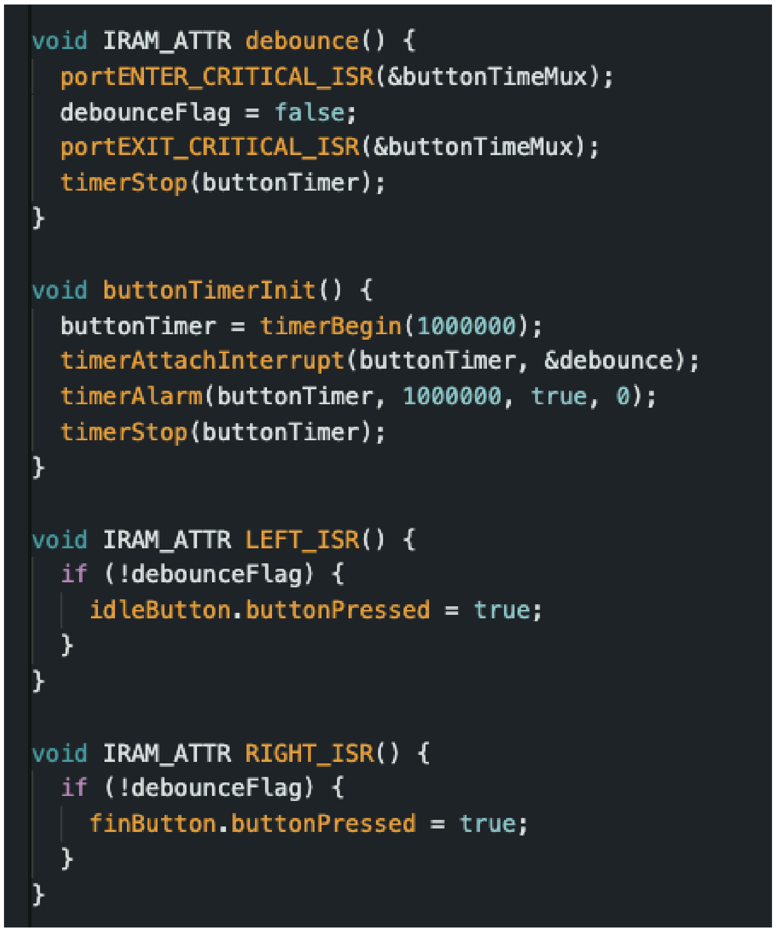
```
void IRAM_ATTR encoderInterrupt() {
  portENTER_CRITICAL_ISR(&tailTimeMux);
  Tail::calculateSpeed = true;
  portEXIT_CRITICAL_ISR(&tailTimeMux);
}

void tailTimerInit() {
  tailTimer = timerBegin(1000000);
  timerAttachInterrupt(tailTimer, &encoderInterrupt);
  timerAlarm(tailTimer, tail.deltaT, true, 0);
}
```



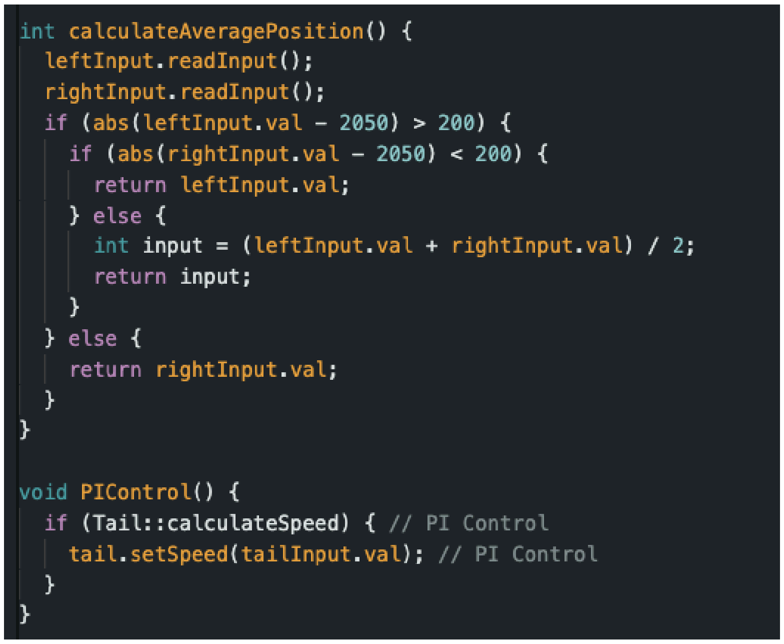
```
class Fin {
public:
  Servo finServo;
  int pin;
  int minUs = 500;
  int maxUs = 2500;
  int minRange;
  int maxRange;

  Fin(int p, int minR, int maxR) : pin(p), minRange(minR), maxRange(maxR) {}

  void start() {
    finServo.setPeriodHertz(50);
    finServo.attach(pin, minUs, maxUs);
    finServo.write(90);
  }

  void setPosition(int input) {
    int desiredAngle = map(input, 0, 4095, minRange, maxRange);
    finServo.write(desiredAngle);
  }
};

Fin leftFin(12, 45, 135);
Fin rightFin(13, 135, 45);

volatile bool debounceFlag = false;
hw_timer_t* buttonTimer = NULL;
portMUX_TYPE buttonTimeMux = portMUX_INITIALIZER_UNLOCKED;
```



```
class IdleButton {
public:
  volatile bool buttonPressed = false;

  bool checkForButtonPress() {
    if (buttonPressed) {
      portENTER_CRITICAL_ISR(&buttonTimeMux);
      debounceFlag = true;
      portEXIT_CRITICAL_ISR(&buttonTimeMux);
      return true;
    }
    return false;
  }

  void buttonResponse() {
    timerStart(buttonTimer);
    buttonPressed = false;
    if (state > 0) {
      digitalWrite(RED_LED, LOW);
      digitalWrite(BLUE_LED, LOW);
      tailInput.val = 0;
      tail.error = 0;
      ledcWrite(MOTOR, 0);
    } else {
      digitalWrite(RED_LED, HIGH);
    }
  }
};

class FinButton {
public:
  volatile bool buttonPressed = false;

  bool checkForButtonPress() {
    if (buttonPressed) {
      portENTER_CRITICAL_ISR(&buttonTimeMux);
       debounceFlag = true;
      portEXIT_CRITICAL_ISR(&buttonTimeMux);
      return true;
    }
    return false;
  }

  void buttonResponse() {
    timerStart(buttonTimer);
    buttonPressed = false;
    if (state == 0) {
      return;
    } else {
      if (state == 1) {
        digitalWrite(RED_LED, LOW);
        digitalWrite(BLUE_LED, HIGH);
      } else if (state == 2) {
        digitalWrite(RED_LED, HIGH);
        digitalWrite(BLUE_LED, LOW);
      }
    }
  }
};

IdleButton idleButton;
FinButton finButton;
```



```
void IRAM_ATTR debounce() {
  portENTER_CRITICAL_ISR(&buttonTimeMux);
  debounceFlag = false;
  portEXIT_CRITICAL_ISR(&buttonTimeMux);
  timerStop(buttonTimer);
}

void buttonTimerInit() {
  buttonTimer = timerBegin(1000000);
  timerAttachInterrupt(buttonTimer, &debounce);
  timerAlarm(buttonTimer, 1000000, true, 0);
  timerStop(buttonTimer);
}

void IRAM_ATTR LEFT_ISR() {
  if (!debounceFlag) {
    idleButton.buttonPressed = true;
  }
}

void IRAM_ATTR RIGHT_ISR() {
  if (!debounceFlag) {
    finButton.buttonPressed = true;
  }
}
```



```
int calculateAveragePosition() {
  leftInput.readInput();
  rightInput.readInput();
  if (abs(leftInput.val - 2050) > 200) {
    if (abs(rightInput.val - 2050) < 200) {
      return leftInput.val;
    } else {
      int input = (leftInput.val + rightInput.val) / 2;
      return input;
    }
  } else {
    return rightInput.val;
  }
}

void PIControl() {
  if (Tail::calculateSpeed) { // PI Control
    tail.setSpeed(tailInput.val); // PI Control
  }
}
```