Solution

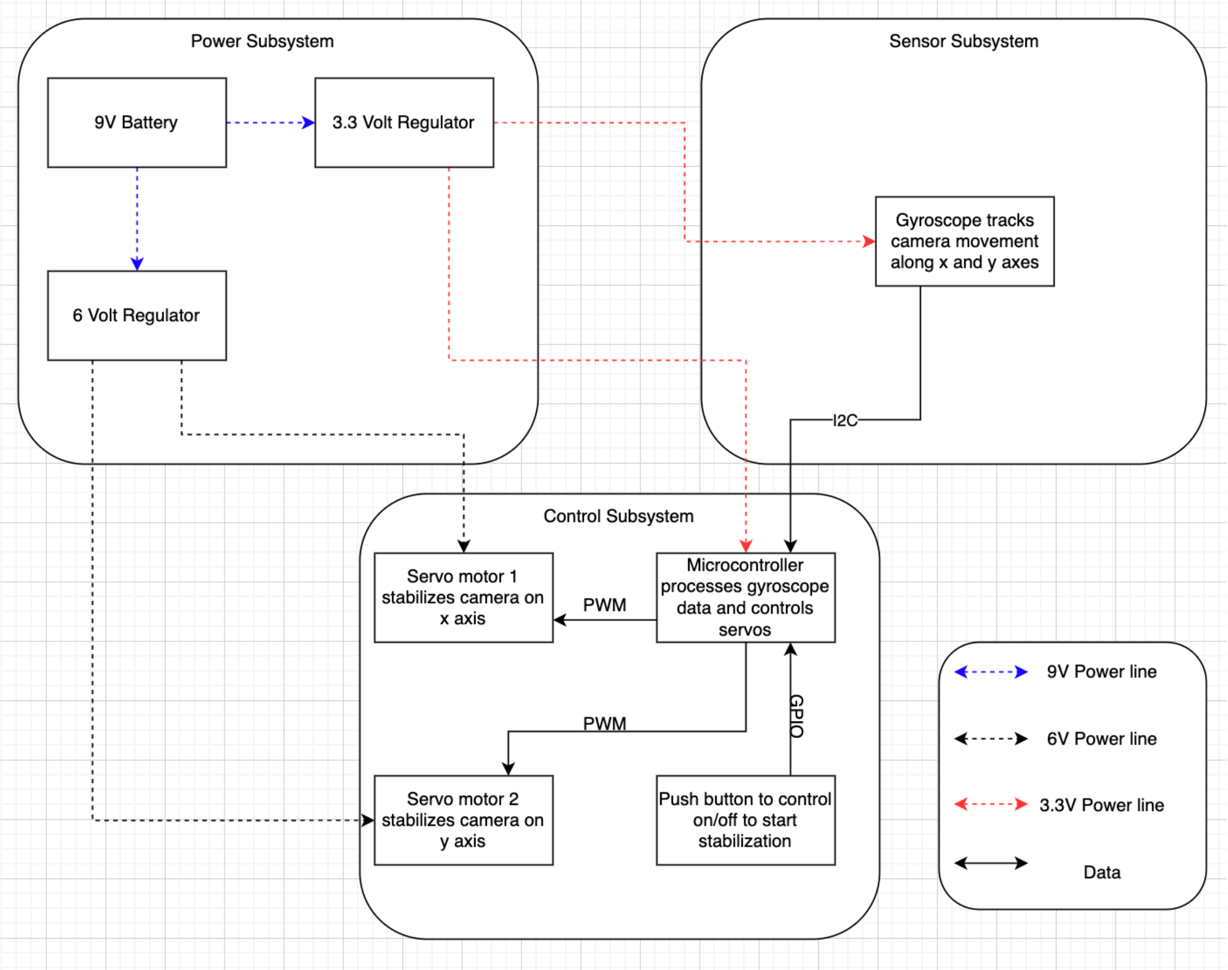
- Camera stabilization, countering the shift in pitch and roll, is the key to solving our problem.
- Gimbal maintains camera in its initial starting orientation, using motors to oppose shift in pitch and roll.
- 3D printed parts and servo motors allow for a more cost-effective solution.

High Level Requirements

1. Camera is stabilized on +x axis (Pitch)
    - minimum slew rate on the x axis of 100 degrees/second.
2. Camera is stabilized on +y axis (Roll)
    - minimum slew rate on the y axis of 100 degrees/second.
3. User interface (button) works
    - First button press turns on system and starts reading gyroscope orientation.
    - Second button press locks the camera orientation by saving gyroscope reading and turns the hold orientation mode on.
    - Third button press switches to normal gimbal mode.
    - Fourth button press turns the system off.

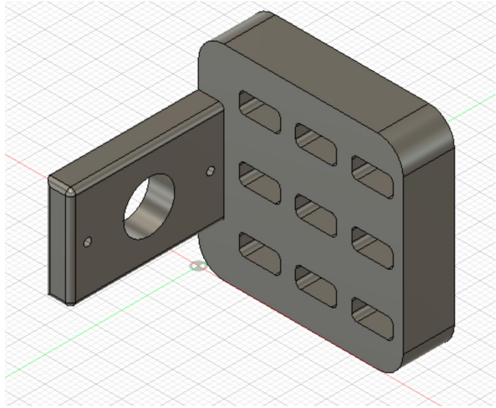
Design

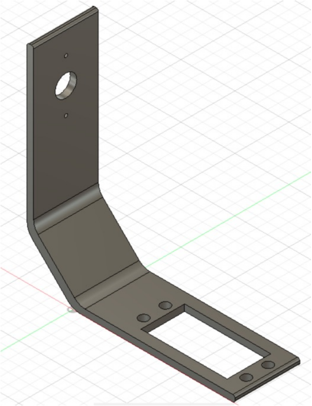
Block Diagram



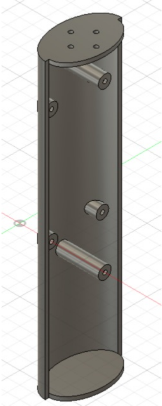
Design

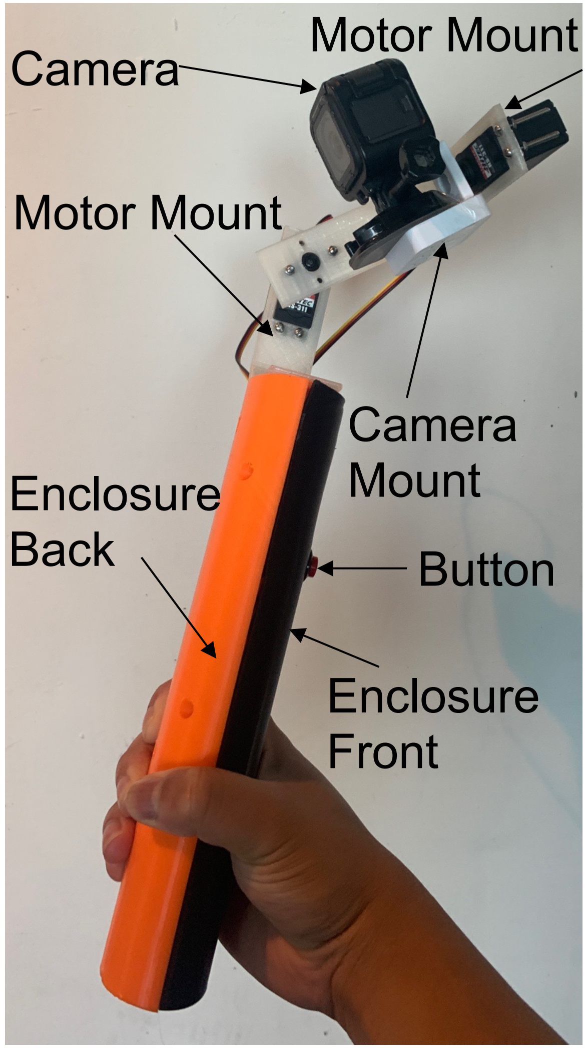
Enclosure


Camera Mount


Motor Mount


Enclosure Front


Enclosure Back



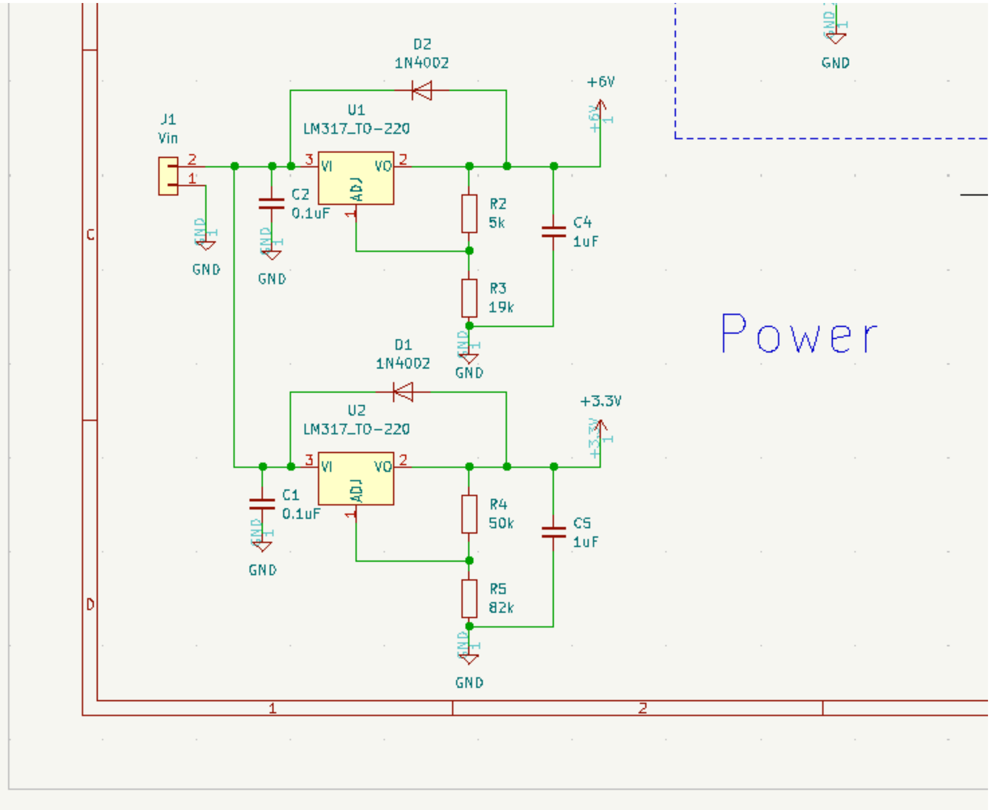
Problems

The Issue	The Cause	The Solution
Voltage Regulators were outputting voltages far too high (7 – 8 Volts)	Wiring issues in PCB with certain components and some miniscule resistors that were tough to solder	Switch to fixed voltage regulators and order resistors with larger package sizes
New parts orders did not arrive in time for the final demo	Parts order was not reviewed	Arduino Nano Dev board
Very jittery movement in orientation holding mode	Logic calculating altered positions were ineffecient	Use switch case statements instead of the constrain or if/else statements

Power Subsystem

Requirements:

- Step down the 9V battery through 2 voltage regulators to 6V for the servo motors and 3.3V for the microcontroller and gyroscope.



Verification:

- Battery outputs 9V.
- Resistors step down voltage to 6V and 3.3V.

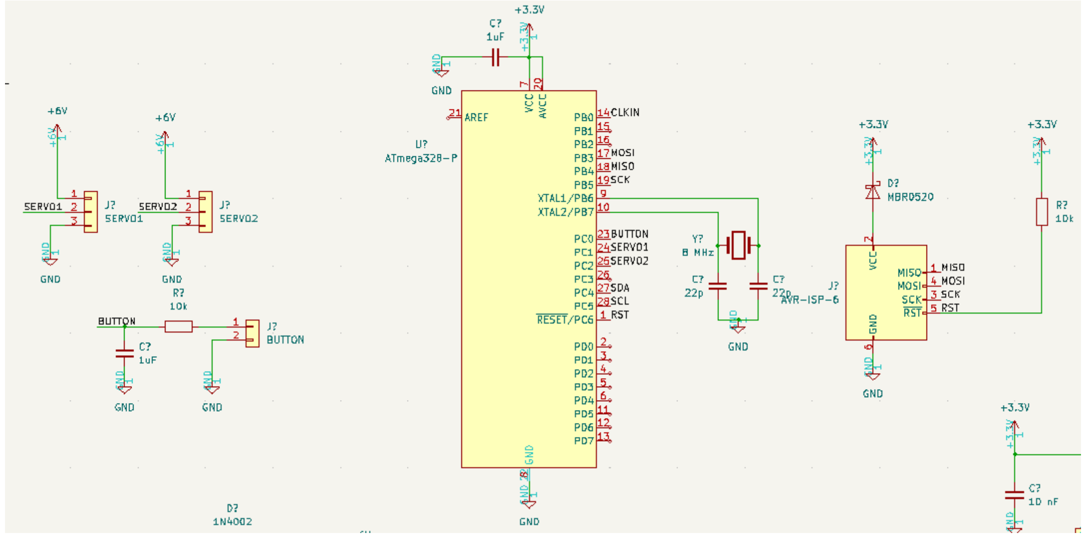
Control Subsystem

Requirements:

- Data Path: Gyro -> Microcontroller (modifies data) -> PWM signals to motor
- Motors must support a load of up to 180 grams (Camera/Phone)

Verification:

- Motor speed of at least 400 degrees/second
- Motors can support 189 g



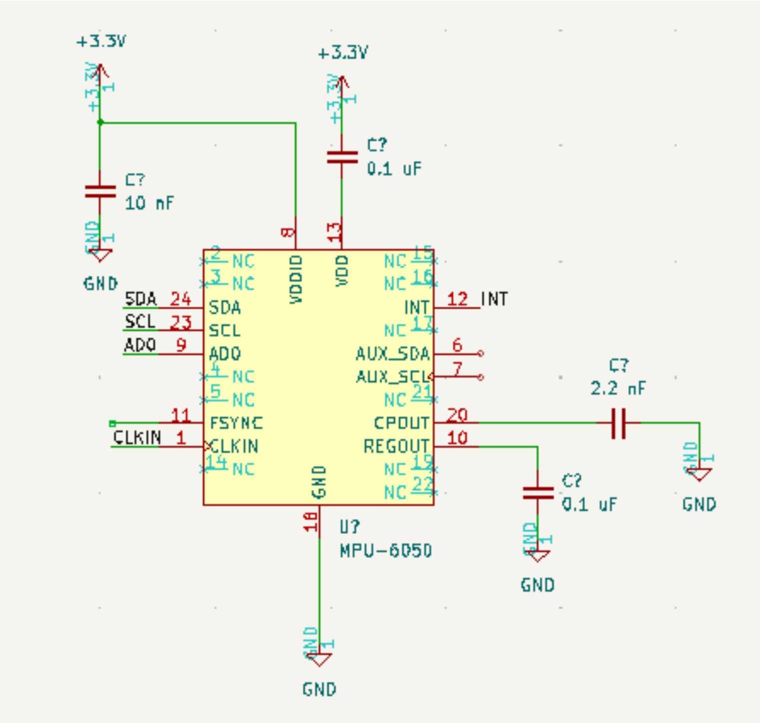
Sensor Subsystem

Requirements:

- Gyroscope readings must occur at a minimum rate of 1000 Hz

Verification:

- Obtains readings and computes PWM signals (extra) at around 1050 Hz



Results

Results

Overall Results

Control Subsystem

- Camera mount weight

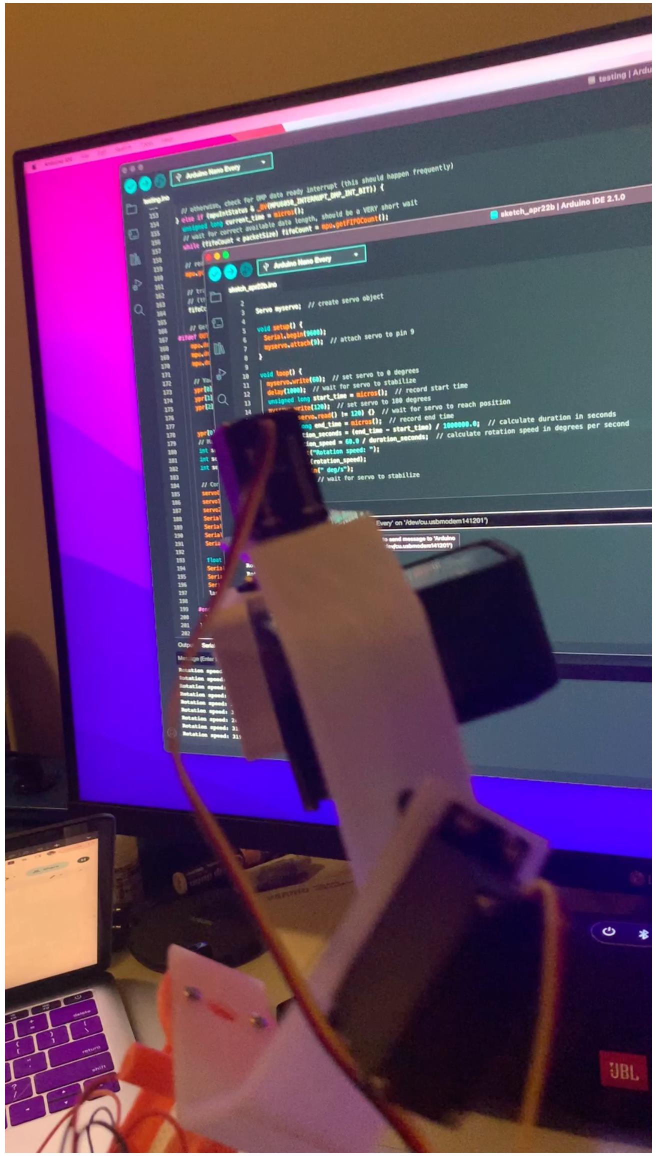
Results

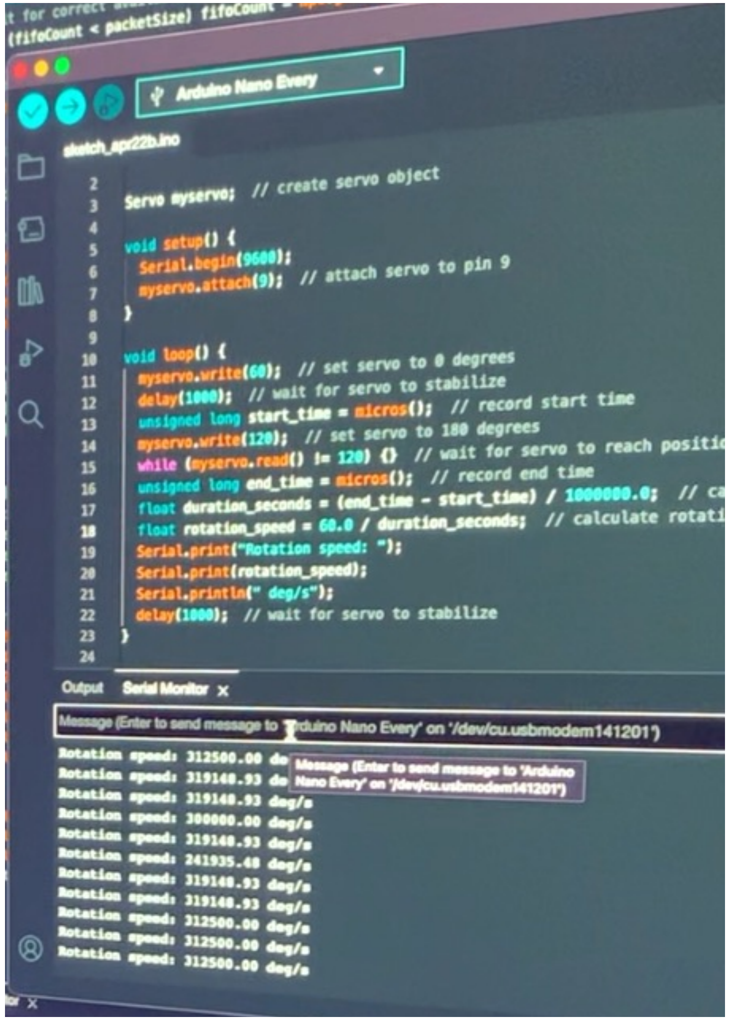
Control Subsystem

- Pitch motor speed test

```
Servo myservo;  // create servo object

void setup() {
  Serial.begin(9600);
  myservo.attach(6);  // attach servo to pin 6
}

void loop() {
  myservo.write(20);  // set servo to 20 degrees
  delay(1000);  // wait for servo to stabilize
  unsigned long start_time = micros();  // record start time
  myservo.write(120);  // set servo to 120 degrees
  while (myservo.read() != 120) {}  // wait for servo to reach position
  unsigned long end_time = micros();  // record end time
  float duration_seconds = (end_time - start_time) / 1000000.0;  // calculate duration in seconds
  float rotation_speed = 100.0 / duration_seconds;  // calculate rotation speed in degrees per second
  Serial.print("Rotation speed: ");
  Serial.print(rotation_speed);
  Serial.println(" deg/s");
  delay(1000);  // wait for servo to stabilize
}
```

Motor speed test code





Motor speed test readout

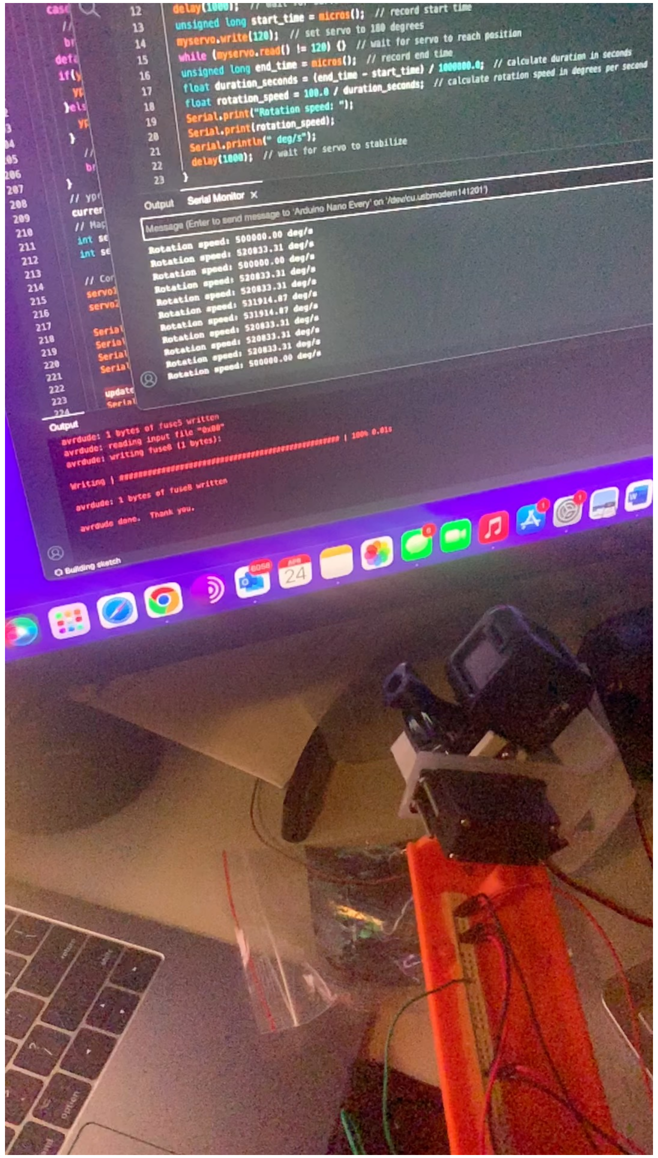
Results

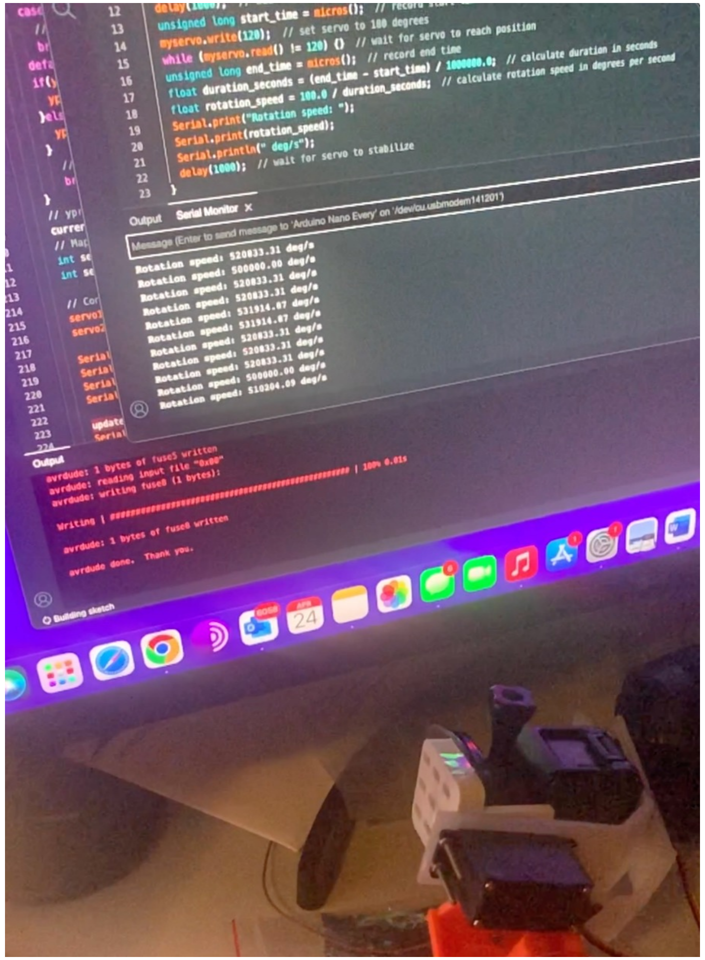
Control Subsystem

- Roll motor speed test

```
Servo myservo;  // create servo object

void setup() {
  Serial.begin(9600);
  myservo.attach(6);  // attach servo to pin 6
}

void loop() {
  myservo.write(20);  // set servo to 20 degrees
  delay(1000);  // wait for servo to stabilize
  unsigned long start_time = micros();  // record start time
  myservo.write(120);  // set servo to 120 degrees
  while (myservo.read() != 120) {}  // wait for servo to reach position
  unsigned long end_time = micros();  // record end time
  float duration_seconds = (end_time - start_time) / 1000000.0;  // calculate duration in seconds
  float rotation_speed = 100.0 / duration_seconds;  // calculate rotation speed in degrees per second
  Serial.print("Rotation speed: ");
  Serial.print(rotation_speed);
  Serial.println(" deg/s");
  delay(1000);  // wait for servo to stabilize
}
```

Motor speed test code





Motor speed test readout

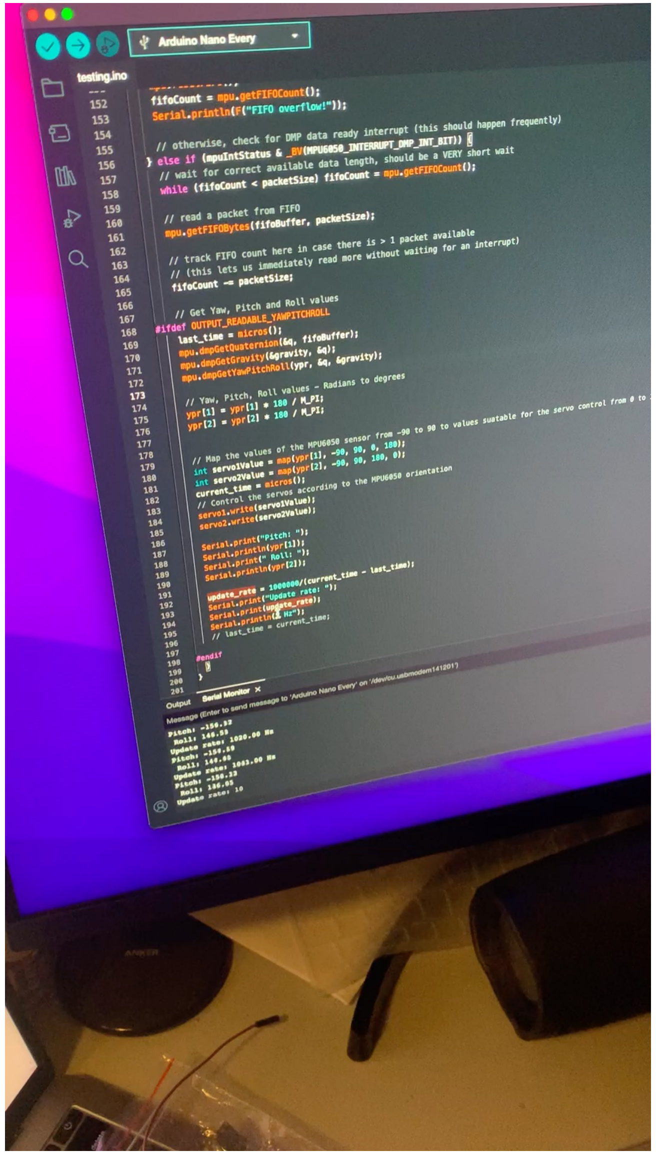
Results

Sensor Subsystem

- Roll motor speed test

```
    // Get Yaw, Pitch and Roll values
#ifdef OUTPUT_READABLE_YAWPITCHROLL
    last_time = micros();
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

    // Yaw, Pitch, Roll values - Radians to degrees
    ypr[1] = ypr[1] * 180 / M_PI;
    ypr[2] = ypr[2] * 180 / M_PI;

    // Map the values of the MPU6050 sensor from -90 to 90 to values suatable for the servo control from 0 to 180
    int servo1Value = map(ypr[1], -90, 90, 0, 180);
    int servo2Value = map(ypr[2], -90, 90, 180, 0);
    current_time = micros();

    // Control the servos according to the MPU6050 orientation
    servo1.write(servo1Value);
    servo2.write(servo2Value);

    Serial.print("Pitch: ");
    Serial.println(ypr[1]);
    Serial.print(" Roll: ");
    Serial.println(ypr[2]);

    update_rate = 1000000/(current_time - last_time);
    Serial.print("Update rate: ");
    Serial.print(update_rate);
    Serial.println(" Hz");
    // last_time = current_time;

#endif
  }
}
```

Sensor update rate test code



Conclusion

What we learned

- Test circuitry on breadboards before designing PCB to have a stronger initial draft
- Thorough research prior to purchasing and testing components
- Don't be afraid the escalate issues and ask for help

Conclusion

Next Steps

- Secure new parts to create a fully functional PCB with linear voltage regulators
- Reprint 3-D enclosures without defects for a cleaner final product
- Swap motors to increase torque and speed to improve overall performance of Gimbal
- Improve orientation lock mode to have no jittery movement