Individual Lab Report #1

Sensors and Motors Lab

February 13, 2025

Yi Wu

Team B

Teammates: Gweneth Ge, Lance Liu, Josh Pen, Jet Situ

Table of Contents

Contents

1 Individual Progress

1.1 Sensors and Motor Lab

1.1.1 Motor Wiring

Powered by a 12V power supply, the 12V DC motor with encoder (251RPM) is connected to the Arduino UNO via the Solarbotics L298 Compact Motor Driver, as shown in Figure 1. I followed the wiring instructions[1] in Figure 2.

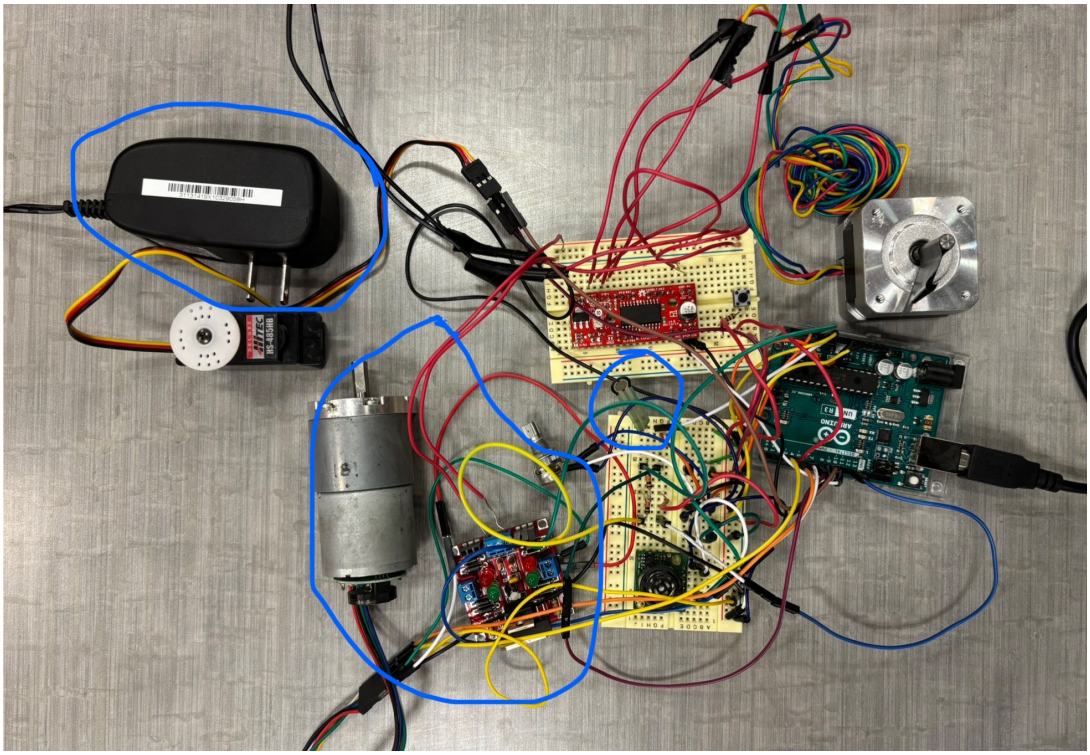

Figure 1: Connection diagram of the DC motor

The wires are connected according to the pin configuration shown in Figure 2.

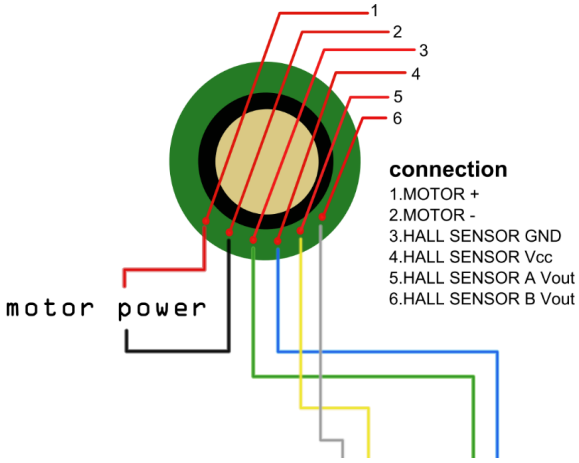

Figure 2: DC motor pin configuration

1.1.2 Motor Class

I developed a standard DC Motor class that can be reused by others. It provides motor on/off operations, encoder tick counting, rotation direction control, and PID velocity control, etc., as follows:

```
class Motor {
    ...
    void turn_on() {...
    void turn_off() {...
    void set_motor(){... //set rotating direction
    void handleEncoder() {... //count encoder pos
    void start_vel_pid() {...
    ...
}
```

Function handleEncoder() showcases the motor tick counting. Its working principle is based on encoding. Channel A (encoder0pinA) and Channel B (encoder0pinB) produce pulses that are phase-shifted relative to each other, enabling position counting through their states. The position counter (posi) is updated through an interrupt service routine triggered by changes in Channel A. The position increases when both A and B rise, and decreases when both fall. The complete function is:

```
void handleEncoder() {
    bool reading = digitalRead(enca);
    if (reading != buffer) {
        buffer = reading;
        bool b = digitalRead(encb);

        if (reading) {  // Rising edge of A
            if (b) posi--;
            else posi++;
        }
        else {          // Falling edge of A
            if (b) posi++;
            else posi--;
        }
    }
}
```

The rotation direction is determined by writing the states to pin IN1 and IN2, where (1,0) allows the motor to rotate clockwise, and (0,1) counterclockwise.

```

void set_motor(int dir, int pwm_val) {
    pwm_val = constrain(pwm_val, lower_limit,
       upper_limit);
    analogWrite(pwmpin, pwm_val);

    if (dir == 1 && motor_state == 1) {
        digitalWrite(in1, 1);
```

```
            digitalWrite(in2, 0);
        }
        else if (dir == -1 && motor_state == 1) {
            digitalWrite(in1, 0);
            digitalWrite(in2, 1);
        }
        else {
            digitalWrite(in1, 0);
            digitalWrite(in2, 0);
        }
    }
```

1.1.3 Sensor

I intend to use the 400 FSR (Force Sensing Resistor)[2] to control the speed of the DC motor. In triage drone applications, this can be used for real-time force feedback and collision avoidance.

The force sensor has a measurement range of 0.2 to 20 N. As shown in Figure 3 (left), the sensor needs to be connected to a measuring resistor (RM) of $10k\Omega$ to form a voltage divider circuit. This configuration converts the force-dependent resistance change of the FSR into a measurable voltage output (VOUT).

Based on the characteristic curves shown in Figure 3 (right), with our $10k\Omega$ measuring resistor, the output voltage exhibits a nonlinear relationship with applied force. To address this nonlinearity, I developed a piecewise linear approximation transfer function that divides the curve into four segments. The mathematical relationship between voltage and force is implemented as follows:

$$F(V) = \begin{cases} \frac{V}{2.0} \cdot 100 & \text{if } V < 2.0V \\ 100 + \frac{V-2.0}{0.5} \cdot 100 & \text{if } 2.0V \leq V < 2.5V \\ 200 + \frac{V-2.5}{0.5} \cdot 400 & \text{if } 2.5V \leq V < 3.0V \\ 600 + \frac{V-3.0}{0.3} \cdot 400 & \text{if } V \geq 3.0V \end{cases} \tag{1}$$

where V is the measured voltage and $F(V)$ is the force in grams. The final force value is converted to Newtons by multiplying by 9.81/1000. The force values can be calculated from the Arduino's analog input readings. The voltage can be obtained from the analog signal ($0 \sim 1023$) read from the Arduino via the following equation:

$$V_{out} = \frac{\text{signal value}}{1024} \cdot V_{cc} \tag{2}$$

1.1.4 PID Velocity Control

The velocity of the motor is calculated by the following equation:

$$velocity = \frac{pos - prepos}{\Delta t} \tag{3}$$

where $prepos$ is the previous position count, pos is the current position count, and Δt is the sampling time interval (in seconds).

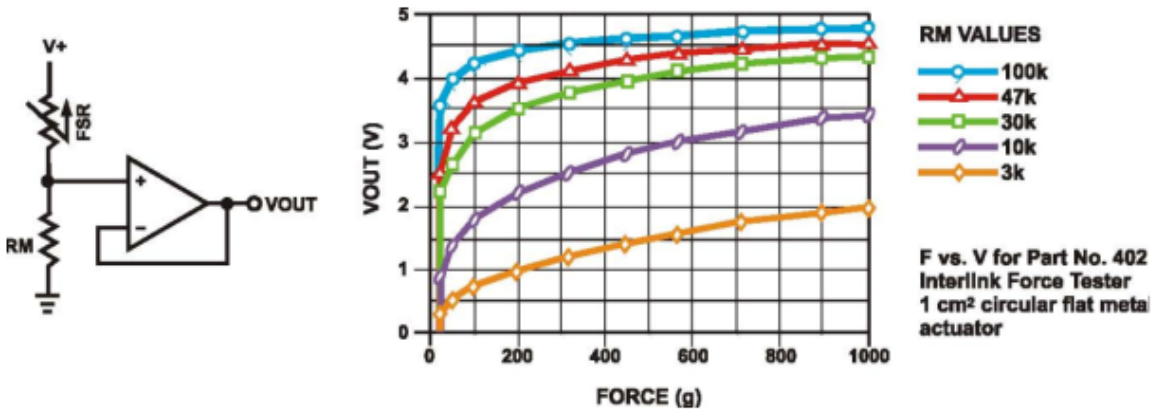

Figure 3: Force sensor specs

To establish a mapping between the force input (in N) and the target motor velocity (counts/second), we determined appropriate ranges for both parameters. Through experimental testing, we found that the practical finger force range is 0.2 to 20 N. The encoder resolution was measured at 5000 counts per revolution. Based on these measurements, we established a linear mapping from force ($0.2 \sim 20$N) to motor velocity ($200 \sim 5000$ counts/second), which is approximately $2.4 \sim 60$ RPM.

The PID control parameters were experimentally tuned to achieve good performance ($K_p = 0.3, K_d = 0.0, K_i = 0.5$). The integral term uses a decay factor of 0.99, so that it's $eintegral = 0.99 * eintegral + e * delta_t$. With these parameters, the system demonstrates rapid convergence to the target velocity with appropriate damping to prevent oscillations.

1.1.5 Velocity Control in Either Direction

In manual input mode, users can specify velocities between $-20 \sim 20$, which map to $-5000 \sim 5000$, encoder ticks, corresponding to approximately ± 1 rotation per second. Positive input causes clockwise rotation, and negative input for counterclockwise rotation. Unlike force sensor control, this enables bi-directional rotation.

1.2 MRSD Project

My work focuses on machine learning algorithm development and deployment. I collaborate with the human detection team at AirLab. While they focus on people detection and tracking algorithms, I work on human pose detection. Determining the direction that a person's chest and head are facing is critical, as we need to position the drone in front of people to monitor their vital signs by detecting chest movements and facial color changes for breathing and respiratory rate measurements.

So far, I have found suitable drone datasets for human detection, tested state-of-the-art human pose detection algorithms in both x86 and Jetson Orin docker environments, and wrapped the algorithm into a ROS2 node.

2 Challenges

2.1 Sensors and Motor Lab

The primary challenge for me was the hardware aspect, particularly wiring the circuits and working with electronic signals. I received great support from Josh with the wiring and circuit board assembly. Understanding the encoder's ticks/counts mechanism for position tracking also consumed a significant amount of my time. The last challenge involved PID parameter tuning, where I struggled with the controller performance due to a sign error in the pwr clipping function. After correcting the sign issue in the clipping function that constrains the external input u to $[-255, 255]$ for the motor PWM control signal, I gradually identified suitable values for K_p, K_i, and K_d.

2.2 MRSD Project

Finding the most suitable drone datasets and 3D human pose estimation algorithm required some effort. After conducting the literature review and screening around 20 datasets, I identified the Semantic Drone Dataset as the most suitable option.

Testing various 3D human pose detection algorithms on single RGB images was relatively straightforward. However, transforming the detected 3D bone structure into real-world frame coordinates proved more complex, requiring understanding of computer vision concepts like intrinsics and extrinsics matrix calculations, and some creative approaches which I am still working to fully solve.

Deploying the algorithms across x86 and Jetson Orin systems and wrapping them into ROS2 nodes was relatively straightforward. However, testing the state-of-the-art algorithm on the low-resolution DARPA videos from last year proved extremely challenging - no poses could be detected in these blurry videos captured from 6 meters away. We will probably need to annotate our own data and train the algorithm from scratch.

3 Team Work

3.1 Sensors and Motor Lab

Name	**Sensor**	**Motor**	**Contribution**
Jet Situ	GUI		GUI implementation with Arduino Integration.
Joshua Pen	Temperature Sensor	DC Stepper	Implemented a user-operated switch with debouncing. Integrated Temperature Sensor with DC Stepper.
Lance Liu	Ultrasonic Range Finder	DC Motor	Integrated Ultrasonic Sensor with Mean Filter and DC Motor. Designed PID controller for DC Motor Postion Control.
Gweneth Ge	Potentiometer	RC Servo	Integrated Potentiometer with RC Servo.
Yi Wu	Force Sensor	DC Motor	Designed PID controller for DC Motor Velocity Control.

Table 1: Team Members and Their Components

3.2 MRSD Project

Name	Contribution
Jet Situ	Assisted in rebuilding the drone for NDAA compliance during the final assembly phase, including mounting components to the provided attachment mounts. Soldered connection wires between the Orin and the Cube Blue, and connected the gimbal's UART control system to the Cube Blue. Validated and connected the gimbal's data connection to onboard networked protocol. Assisted in the development of deployed software on the Orin, and the design of the overall software architecture protocol. Contributed to setting up the ROS2 architecture for inter-system communication in the DTC. Obtained a Part 107 license for purposes of outdoor flight evaluation and testing.
Joshua Pen	Assisted in rebuilding the drone for NDAA compliance, replacing the DJI controller with Cube Blue ArduPilot. Soldered wires to connect various components during the drone rebuild. Designed attachment mounts for the Doodle Labs Radio antennas, Intel Realsense D435, and a separate mount for the NVIDIA Orin NX and Doodle Labs Smart Radio. Developed a mounting solution for the Hadron 640R Gimbal and designed extension legs for the drone. Procured propellers and shields. Contributed to the payload integration and configuration of the Hadron 640R with Cube Blue ArduPilot and NVIDIA Orin NX.
Lance Liu	Contributed to the rebuilding of the NDAA compliant drone. Established communication between the ground control station (GCS) and the drone via a sophisticated radio system. Executed iterative refining and testing on the embedded system. Integrated and validated the interaction pipeline between the GCS and the onboard Orin using MAVROS. Designed a behavior tree to manage decision during the challenge operation. Migrated the AirLab Docker environment to support the ARM architecture on the NVIDIA Orin, and integrated—while continuing to adapt—AirLab's codebase within our platform.
Gweneth Ge	Primarily contributed to the development of the first-version drone for submitting video documentation to DARPA, focusing on sourcing and integration of NDAA-compliant components. Helped the electrical integration, including configuring the Pixhawk ArduPilot and BlackCube on the DJI M100 frame, as well as powering and connecting the gimbal, cameras and radio systems. Additionally, supported overall project management and logistics.
Yi Wu	Collected human detection datasets specifically for drone applications and conducted a literature review of 3D Human Pose Estimation (HPE) algorithms. Collaborated with the AirLab human detection team to test the state-of-the-art HPE algorithm in their x86 and Jetson Orin docker environments, wrapping the algorithm as ROS2 nodes.

Table 2: Team Members and Their Contributions

4 Plans

4.1 MRSD Project

Name	Contribution
Jet Situ	Polygon covering waypoints generator, including sending entire waypoints in one go and flying to designated position at a certain altitude. Gimbal control protocol and sensor nodes development. Take off/landing planner, patients searching logic, task allocation planner, and visualization.
Joshua Pen	In my future role, I will focus on developing gimbal control protocols and sensor nodes, along with additional mechanical modifications. I will assist in sensor nodes development, including detection launch, visualization, and clicking interaction. Furthermore, I will handle project management and logistics.
Lance Liu	ROS2 network refinement. Data transmission of sensors including RGB, Thermal and gimbal. Behavior tree executive and management implementation including auto takeoff, land, safe landing, RTB, mapping, searching, and inspecting. Overall robot system bringing up.
Gweneth Ge	Primarily work on Inter-UAV collision logic and planner launch. Assist in sensor nodes development, detection launch, visualization and clicking interaction. Continue supporting on project management and logistics.
Yi Wu	Integrate the human pose estimation algorithm with the upstream person Re-Identification (ReID) algorithm. Test the algorithm performance on DARPA datasets, and prepare new annotated datasets if necessary for retraining purposes. Develop solutions for pose estimation algorithms using thermal camera data during nighttime conditions.

Table 3: Team Members and Their Contributions

5 References

[1]https://wiki.dfrobot.com/12V_DC_Motor_251rpm_w_Encoder__SKU__FIT0186_

[2]https://cdn2.hubspot.net/hubfs/3899023/Interlinkelectronics%20November2017/Docs/Datasheet_FSR.pdf

6 Appendix: Quiz

6.1 ADXL335

1. **What is the sensor's range?**

   Minimum is -3g to 3g, but typically could measure -3.6g to 3.6g. Here g represents the acceleration due to gravity, which is approximately 9.8 m/s^2 (or 9.8 N/kg) at Earth's surface.

2. **What is the sensor's dynamic range?**

   The dynamic range can be calculated as $|-3.6g| + |+3.6g| = 7.2g$

3. **What is the purpose of the capacitor CDC on the LHS of the functional block diagram on p. 1? How does it achieve this?**

   According to the datasheet, the CDC (Decoupling Capacitor) is placed close to the ADXL335 supply pins to adequately decouple the accelerometer from noise on the power supply. For most applications, a single 0.1 F capacitor is sufficient for power supply decoupling. It works by providing a local low-impedance energy storage element that can filter out high-frequency noise on the power lines and maintain a stable voltage for the chip. This is particularly important in applications where noise is present at the 50 kHz internal clock frequency (or any harmonic thereof) as power supply noise can cause errors in acceleration measurement.

4. **Write an equation for the sensor's transfer function.**

   The transfer function can be expressed as:

   $$V_{OUT} = Sensitivity \times (g) + Zero_{g\_Bias}$$

   where:

   - V_{OUT} is the output voltage (V)
   - Sensitivity is 300 mV/g (typical)
   - g is acceleration in units of gravity
   - $Zero_{g\_Bias}$ is 1.5V (typical)

5. **What is the largest expected nonlinearity error in g?**

   Calculation process:

   - Nonlinearity is specified as "±0.3% of full scale"
   - Full scale = Total range = 7.2g (from -3.6g to +3.6g)
   - Error = 0.3% of full scale
   - 0.3% of 7.2g = 0.003 × 7.2g = 0.0216g

   Therefore, the largest expected nonlinearity error is 0.0216g.

6. **What is the sensor's bandwidth for the X- and Y-axes?**

   Default bandwidth (without external filter) is 1600 Hz. Adjustable bandwidth range is 0.5 Hz to 1600 Hz.

7. **How much noise do you expect in the X- and Y-axis sensor signals when your measurement bandwidth is 25 Hz?**

   Given:

   - Noise density = 150 g/Hz rms
   - Bandwidth (BW) = 25 Hz
   - Formula: $Noise_{rms} = Noise_{Density} \times \sqrt{BW \times 1.6}$

   Calculate:

   $$Noise_{rms} = 150 \text{ g}/\sqrt{\text{Hz}} \times \sqrt{25 \text{ Hz} \times 1.6} = 948.75 \text{ g rms}$$

8. **If you didn't have the datasheet, how would you determine the RMS noise experimentally?**

   Experimental procedure:

   - **Setup:**
     - Mount ADXL335 rigidly to a stable surface
     - Orient one axis exactly parallel to gravity
     - Set desired bandwidth using appropriate filtering capacitors
     - Connect outputs to a data acquisition system
   - **Data Collection:**
     - Sample at rate $f_{sampling} \geq 4f_{bandwidth}$
     - Collect data for $\sim$ 1 minute
     - Record multiple datasets
   - **Analysis:**
     - Subtract mean value to remove DC offset
     - Calculate standard deviation
     - Verify across multiple datasets
   - **Validation:**
     - Test different orientations
     - Compare between axes
     - Perform FFT analysis

6.2 Signal Conditioning

6.2.1 Filtering

Moving Average Filter Problems:

1. **Time Lag/Phase Delay**

   - The filter introduces a time delay equal to half the window length
   - This can be critical in real-time applications or control systems
   - The longer the averaging window, the greater the delay

2. **Poor Step Response**
    - When there's a sudden change in the signal (step), the filter "smears" it over multiple samples
    - The output shows a gradual ramp instead of preserving the sharp transition
    - This can be problematic when detecting sudden movements or impacts

Median Filter Problems:

1. **Computational Complexity**
    - Requires sorting operations for each window position
    - More computationally intensive than moving average
    - Processing time increases with window size
2. **Detail Loss**
    - Can remove fine details or legitimate high-frequency components
    - May eliminate valid spikes or brief transients that are actually part of the signal
    - Particularly problematic with larger window sizes

6.2.2 Opamps

Only the Senario 1 can be calibrated, and the Scenario 2 can not be.

Scenario 1: Sensor range -1.5V to 1.0V

- **Input and Reference Voltage**
    - V_1 will be the input voltage (varies from -1.5V to 1.0V)
    - V_2 will be the reference voltage
- **Circuit Calibration** Circuit equation: $V_{\text{out}} = (R_f/R_i) * (V_1 - V_2) + V_s$

  Boundary conditions:
    - When $V_1 = -1.5\text{V}$, V_{out} should be 0V
    - When $V_1 = 1.0\text{V}$, V_{out} should be 5V
- **Solving:**
    - At $V_1 = -1.5\text{V}$: $0 = (R_f/R_i) * (-1.5 - V_2) + V_s$
    - At $V_1 = 1.0\text{V}$: $5 = (R_f/R_i) * (1.0 - V_2) + V_s$
- **After solving these simultaneously:**
    - $V_2 = -3\text{V}$
    - $R_f/R_i = 1$
    - $V_s = 1\text{V}$

Scenario 2: Sensor range -2.5V to 2.5V

- Analysis shows this cannot be calibrated:
  - When V_1 is input, R_f/R_i would need to be -1
  - When V_2 is input, R_f/R_i would be 0
  - These constraints make precise calibration impossible

6.3 Control

1. If you want to control a DC motor to go to a desired position, describe how to form a digital input for each of the PID (Proportional, Integral, Derivative) terms.

   Proportional (P) Term

   - Calculate error: $error = \text{desired\_position} - \text{current\_position}$
   - Digital input: $P\_\text{term} = K_p \times error$
   - Provides immediate response proportional to position deviation

   Integral (I) Term

   - Accumulate errors over time: $I\_\text{term} \leftarrow I\_\text{term} + K_i \times error \times \Delta t$
   - Eliminates steady-state positioning errors
   - Implement anti-windup mechanisms

   Derivative (D) Term

   - Calculate error rate: $D\_\text{term} = K_d \times \frac{error - \text{previous\_error}}{\Delta t}$
   - Provides damping and reduces overshoot

   PID Control Signal

   $$\text{control\_signal} = P\_\text{term} + I\_\text{term} + D\_\text{term} \tag{4}$$

2. If the system you want to control is sluggish, which PID term(s) will you use and why?

   Use proportional control KP if the system is sluggish, because higher Kp will produce a larger control signal for a given error so that it could decrease the rise time and make the motor move more aggressively.

3. After applying the control in the previous question, if the system still has significant steady-state error, which PID term(s) will you use and why?

   If the system still has significant steady-state error after applying proportional control, use the KD term, because the integral term accumulates errors over time and continuously generates a control signal proportional to the accumulated error, which can effectively eliminate steady-state offset in the system.

4. After applying the control in the previous question, if the system still has overshoot, which PID term(s) will you apply and why?

   If the system experiences overshoot after applying control, use the KD term because it helps reduce overshoot by providing damping to the system response. The D term calculates the rate of change of the error, which allows it to anticipate and counteract rapid changes, thereby stabilizing the system and minimizing overshoot.