Individual Lab Report #1

Sensors and Motors Lab

February 13, 2025

Joshua Pen

Team B

Teammates: Gweneth Ge, Lance Liu, Yi Wu, Jet Situ

Table of Contents

Contents

1 Individual Progress **1**
1.1 Sensors and Motor Lab 1
1.1.1 Joshua 1
1.2 MRSD Project 2

2 Challenges **2**
2.1 Sensors and Motor Lab 2
2.2 MRSD Project 2

3 Team Work **3**
3.1 Sensors and Motor Lab 3
3.2 MRSD Project 4

4 Plans **5**
4.1 MRSD Project 5

A Arduino Code **5**

B Sensors & Motor Control Quiz **6**

1 Individual Progress

1.1 Sensors and Motor Lab

1.1.1 Joshua

In this lab, my primary responsibilities included developing the code and constructing the circuit for the temperature sensor, DC stepper motor, and a push button. The push button was designed to function as a user-operated switch with debouncing to ensure reliable state transitions. Additionally, I was tasked with integrating all team members' circuits and code into the final demonstration platform.

I utilized a low-voltage temperature sensor, specifically the TMP36, interfaced with an Arduino. The Arduino's analog-to-digital converter (ADC) provides readings ranging from 0 to 1023, corresponding to input voltages between 0 and 5 volts. To convert the sensor's ADC reading to voltage, the following formula is used:

$$\text{Voltage (V)} = (\text{Sensor Reading} \div 1023) \times 5 \tag{1}$$

According to the TMP36 datasheet, the output voltage is linearly related to temperature, with a scale factor of 0.01 V per degree Celsius and a 0.5 V offset. This means that at 0°C, the output is 0.5 V, and it increases by 0.01 V for each degree rise in temperature. To convert the voltage reading to temperature, the formula is:

$$\text{Temperature (°C)} = (\text{Voltage (V)} - 0.5) \times 100.0 \tag{2}$$

By applying these formulas, I established a transfer function to convert the sensor's voltage output to degrees Celsius.

In this lab, I utilized the EasyDriver stepper motor driver to control both the direction and the number of steps of a stepper motor. The temperature sensor employed has a measurement range from -55°C to 125°C. I mapped the temperature readings to the stepper motor's movements, assigning positive temperatures to clockwise rotations and negative temperatures to counterclockwise rotations. To achieve this, I converted the temperature values into corresponding step counts, ensuring that each degree Celsius equated to a specific number of motor steps. Given that the stepper motor requires approximately 3,194.4 steps to complete a full 360-degree rotation, the conversion equation is:

$$\text{Steps} = (\text{Temperature (°C)} \div 360) \times 3194.4 \tag{3}$$

This formula enables precise mapping of the temperature reading in degrees Celsius to the corresponding angular degrees of rotation of the motor.

In my implementation, I enhanced the system's interactivity by adding serial communication capabilities. Users can input the command tmp into the serial terminal to prompt the stepper motor to rotate by an angle corresponding to the current temperature reading. Additionally, by entering a numerical value, the stepper motor will rotate by that specified number of degrees. This dual-input approach allows for both sensor-driven and manual control of the motor's position.

In my circuit design, I incorporated a push button to toggle the system between sensor feedback control and user inputs from our GUI. This state transition is managed within an interrupt handler. To address potential signal bouncing, the handler includes a debounce mechanism: if the button signal fluctuates within a 300ms window, it's identified as bounce noise, and the state change is disregarded.

1.2 MRSD Project

My initial role in the project focused on the mechanical components of the drone. With a January 15 deadline for a DARPA demo flight, the test included autonomous daytime takeoff, landing, waypoint navigation to the target patient, nighttime takeoff and landing, and emergency stop (e-stop) capabilities using QGroundControl. To meet these requirements, we began by rebuilding the DJI Matrice M100, ensuring all components were NDAA compliant by replacing the DJI controller with Cube Blue ArduPilot.

I 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. Additionally, I developed a mounting solution for the Hadron 640R Gimbal and designed extension legs for the drone. I also procured propellers and shields.

During test flights for the demo video, I assisted by 3D printing extra components and modifying mounts to address any damage that occurred. After completing the demo, I redesigned all mounts to ensure even payload distribution and refined the overall design. Lastly, I contributed to the payload integration and configuration of the Hadron 640R with Cube Blue ArduPilot and NVIDIA Orin NX.

2 Challenges

2.1 Sensors and Motor Lab

One challenge I encountered during the sensor and motor lab was locating the datasheet and determining the conversion for the temperature sensor from volts to degrees Celsius. The datasheet provided only a graph, so I estimated the equation to map the values. For instance, with sensors like the TMP36, the output voltage is linearly related to temperature, typically providing 0.01 V per degree Celsius with a 0.5 V offset. This means that at 0°C, the output is 0.5 V, and it increases by 0.01 V for each degree rise in temperature. To convert the voltage reading to temperature, the formula is:

$$\text{Temperature } (^\circ\text{C}) = (\text{Voltage (V)} - 0.5) \times 100.0 \tag{4}$$

However, without a clear equation in the datasheet, I had to approximate this relationship from the provided graph, which introduced potential inaccuracies in my temperature measurements.

Another challenge I encountered during this lab was that I couldn't control the DC stepper motor to rotate counterclockwise. Using a multimeter to measure the resistance between each pin, I discovered a short circuit in the Easydriver microcontroller. After replacing the microcontroller, I was able to resolve the issue.

2.2 MRSD Project

One of our biggest challenges as a team arose during flight testing for the DARPA demo video. The test included autonomous daytime takeoff, landing, waypoint navigation to the target patient, nighttime takeoff and landing, and emergency stop (e-stop) capabilities with QGroundControl.

During testing, we broke two sets of propellers, requiring us to order replacements before resuming. Unfortunately, both replacements also broke, forcing us to order six additional sets to complete the demo. To mitigate this issue, we decided to install propeller guards to reduce the risk of propeller damage during crashes.

We also encountered challenges with the payload integration and configuration of the Hadron 640R with Cube Blue ArduPilot and the NVIDIA Orin NX due to NDAA compliance restrictions. Since we were not allowed to use the HereLink device provided by the company because of its Wi-Fi capabilities, we opted to connect the gimbal via Ethernet to the NVIDIA Orin NX, enabling video streaming from the gimbal.

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. [4]

Table 2: Team Members and Their Contributions

Name	Contribution
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 3: Team Members and Their Contributions

4 Plans

4.1 MRSD Project

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.

A Arduino Code

```
// Pin definitions
const int stepPin = 8;     // STEP pin
const int dirPin = 10;      // DIR pin
const int TEMP_SENSOR_PIN = A3;     // Temperature sensor analog pin

// Motor control parameters
int stepDelay = 100;       // Pulse delay (microseconds)
bool isRunning = false;    // Motor running status

// Test pin definitions
const int outPin4 = 4;     // Test pin 4
const int outPin5 = 5;     // Test pin 5

void setup() {
    Serial.begin(9600);

    // Set pin modes
    pinMode(stepPin, OUTPUT);
    pinMode(dirPin, OUTPUT);

    // Set clockwise direction
    digitalWrite(dirPin, HIGH);  // HIGH for clockwise
}

void loop() {
    float temperature = readTemperature()*10;
    int temperature_int = (int)temperature;
    if (Serial.available() > 0) {
        String command = Serial.readStringUntil('\n');
        command.trim();
```

```
31
32          if (command == "tmp" || command == "TMP") {
33              isRunning = true;
34              Serial.println("Rotate␣according␣to␣temperature");
35          }
36          else if (isNumeric(command)) {
37              isRunning = true;
38              temperature_int = (int)(command.toFloat()/360*1585);
39              Serial.println("Rotate␣according␣to␣use-demand␣degrees");
40          }
41          else {
42              isRunning = false;
43              Serial.println("Enter␣tmp␣to␣rotate␣according␣to␣
                    temperature,␣or␣enter␣use-demand␣degrees␣to␣rotate␣
                    according␣to␣use-demand␣degrees");
44          }
45      }
46
47      if (isRunning) {
48          for (int i = 0; i < temperature_int; i++) {
49          digitalWrite(stepPin, HIGH);
50          delayMicroseconds(stepDelay);
51          digitalWrite(stepPin, LOW);
52          delayMicroseconds(stepDelay);
53          }
54          isRunning = false;
55      }
56
57  }
58
59  float readTemperature() {
60    int sensorValue = analogRead(TEMP_SENSOR_PIN);
61    Serial.print("sensorValue:␣");
62    Serial.println(sensorValue);
63    // Convert analog reading to temperature
64    // For LM35: (sensorValue * 5.0 * 100.0) / 1024.0
65    // For TMP36: ((sensorValue * 5.0 / 1024.0) - 0.5) * 100.0
66    float voltage = (sensorValue * 5.0) / 1024.0;
67    float temperature = (voltage - 0.5) * 100.0;  // For TMP36 sensor
68    return temperature;
69  }
70
71  boolean isNumeric(String str) {
72      if(str.length() == 0) return false;
73
74      for(char i = 0; i < str.length(); i++) {
75          if(!isDigit(str.charAt(i))) return false;
76      }
77      return true;
78  }
```

nguage=C++]

B Sensors & Motor Control Quiz

Sensors and Motor Control Lab Quiz

1. Reading a datasheet. Refer to the ADXL335 accelerometer datasheet (https://www.sparkfun.com/datasheets/Components/SMD/adxl335.pdf) to answer the below questions.
    - What is the sensor's range?

The minimum range is from -3g to 3g but typical specification says it's from -3.6g to 3.6g

    - What is the sensor's dynamic range?

For X and Y axis dynamic range = 20 log(3.6/(150 * 10^(-6)))= 87.6dB

For Z axis dynamic range = 20 log(3.6/(300 * 10^(-6)))= 81.6dB

    - What is the purpose of the capacitor C_{DC} on the LHS of the functional block diagram on p. 1? How does it achieve this?

The purpose of the capacitor is to decouple the accelerometer from noise on the power supply. It achieves this by bypassing the power supply.

    - Write an equation for the sensor's transfer function.

Vout=0.3a+1.5 since the sensitivity is 300mV/g and when there is 0g Vout is 1.5V.

    - What is the largest expected nonlinearity error in g?

The largest nonlinearity error in g is 0.3% of 7.2g so it is 0.003*7.2g=0.216g.

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

The sensor's bandwidth for the X and Y axes is 1600Hz.

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

The noise density is 150 ug/ (Hz)^0.5, so I expect 150 * 10^(-6) * (25)^0.5=0.75 * 10^(-3)g=0.00075g.

    - If you didn't have the datasheet, how would you determine the RMS noise experimentally? State any assumptions and list the steps you would take.

I would have the sensor held still so the true input signal is 0 and gather a time series of output measurements. I would take the sample average of these measurements to be the mean, and subtract it from each measurement to get the deviations. The RMS noise will be the square root of the average of the squares of the deviations.

2. Signal conditioning
    - Filtering
        - Name at least two problems you might have in using a moving average filter.

In a moving average filter if there is an outlier in the collected data it may skew the results of the overall reading from the sensor in the next few inputs depending on how many sensor readings are averaged together. Another problem is the window size of the filter may affect how well it is able to smooth out the sensor readings. Too small of a window size may not have enough effect on smoothing out the data while a window size too large can over smooth the data causing the data to lose important information that might be relevant.

        - Name at least two problems you might have in using a median filter.

One problem for the median filter is that some details in the data collected could be filtered out since it does not take account of the outlier data that pops up if the rest of the data has similar measurements.

So, if the outlier data is a measurement of a bird suddenly appearing in front of the sensor, that information will be lost. To compute a median filter, a sorting algorithm must be run causing the overall speed of computation to be slow. So, if the data is needed very fast the filter may cause an issue.

- Opamps
    - In the following questions, you want to calibrate a linear sensor using the circuit in Fig. 1 so that its output range is 0 to 5V. Identify in each case: 1) which of V1 and V2 will be the input voltage and which the reference voltage; 2) the values of the ratio Rf/Ri and the reference voltage. If the calibration can't be done with this circuit, explain why.
        - Your uncalibrated sensor has a range of -1.5 to 1.0V (-1.5V should give a 0V output and 1.0V should give a 5V output).

V2 is the input voltage and V1 is the reference voltage. Rf/Ri=1 and V1=-3V

- Your uncalibrated sensor has a range of -2.5 to 2.5V (-2.5V should give a 0V output and 2.5V should give a 5V output).

if V2 is the input voltage and V1 is the reference voltage. Rf/Ri=0 and there is no V1 that will make this work, since Vo = -Rf/Ri *V1 + (1+Rf/Ri)V2

if V2 is the reference voltage and V1 is the input voltage. Rf/Ri=-1 and this is not possible since resistance is positive. So the calibration can't be done.

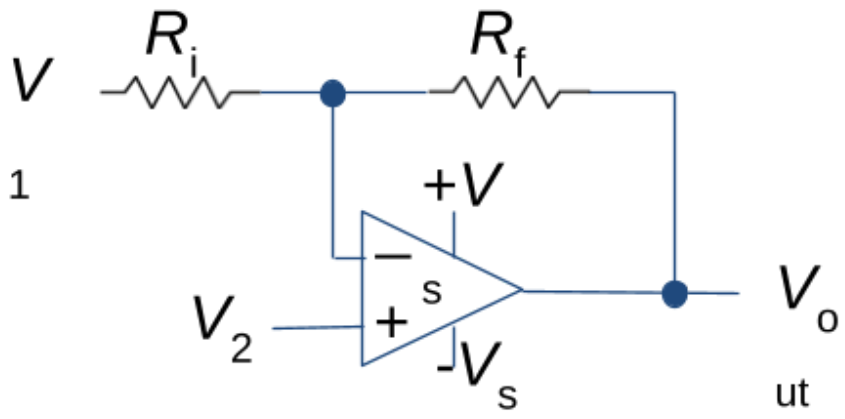

Fig. 1. Opamp gain and offset circuit

3. Control
    - 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.

For the proportional term I would multiply the P gain Kp by the error of the actual position minus the desired position.

For the integral term I would sum all the past errors in position and multiply by the sampling interval and multiply it by Ki the integral gain.

For the derivative term I would multiply the derivative gain Kd by the error in velocity or the derivative of the position error.

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

By using the proportional term I am able to make the system converge faster to the desired position.

- 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?

I will use the integral term to account for all the past errors and tune the system to the desired steady-state. This is because integral terms take account of all past errors and accumulate it and correct the error.

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

I will use the derivative term to act as a damper so that overshoot is less likely to occur. This is because the derivative term will look at the derivative of error and act as a pull back force.