Project
ME 5245 Mechatronic Systems

PART A

Authors: Joshua Yoo, Tirth Patel

1 Objective

The objective of PART A was to physically construct the DC motor control circuit and develop the realtime software interface in Simulink. This involved wiring all hardware components, successfully acquiring sensor signals from the potentiometer and the motor's encoder, and processing the encoder data to calculate rotor displacement and speed.

2 System Hardware and Circuit Diagram

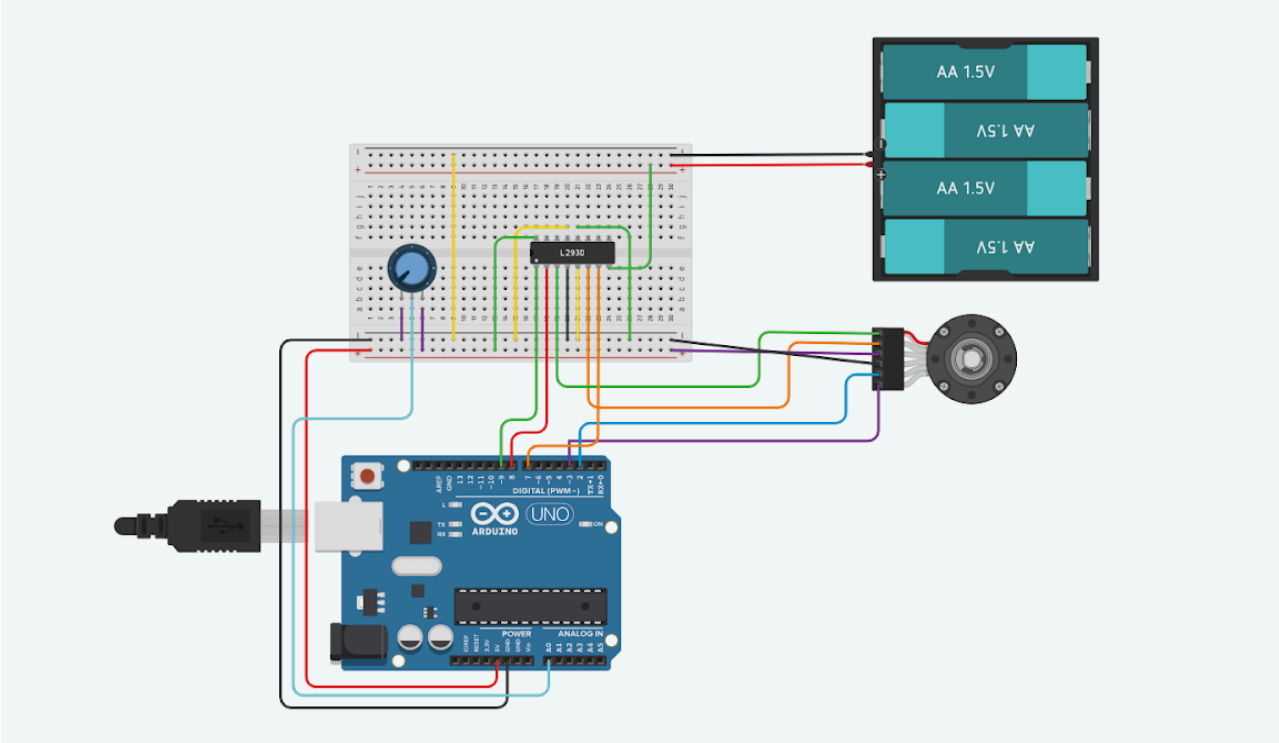

Fig 1: Complete Circuit Diagram.

The diagram shows the wiring for the Arduino Uno, L293D H-bridge, DC motor, encoder, potentiometer, and external battery pack. The circuit uses separate power rails for the 5V logic and the ~7.5V motor power, with a common ground connecting both systems.

3 Encoder Signal Acquisition and Processing

This section details the process of reading the encoder and interpreting its signals for displacement and speed.

3.1 Theoretical Resolution and Expected Position error

As required, the expected position error due to the encoder's resolution is calculated below.

The motor encoder has a resolution of **48 Counts Per Revolution (CPR)** on the motor shaft. The gearbox has a ratio of **172:1**

- **Total Counts per output shaft Revolution:**

  $$Total\ Counts\ =\ 48\ CPR\ \times\ 172\ =\ 8256\ Counts/Revolution$$

- **Angular Resolution per count:**

  $$Resolution\ =\ \frac{360^{\circ}}{8256\ Counts}\ \approx\ 0.0436^{\circ}/Count$$

- **Expected Quantization Error:** The maximum error is half the resolution.

  $$Expected\ Error\ =\ \pm\frac{0.0436}{2}\ =\ \pm\ 0.0218^{\circ}$$

3.2 Simulink Implementation

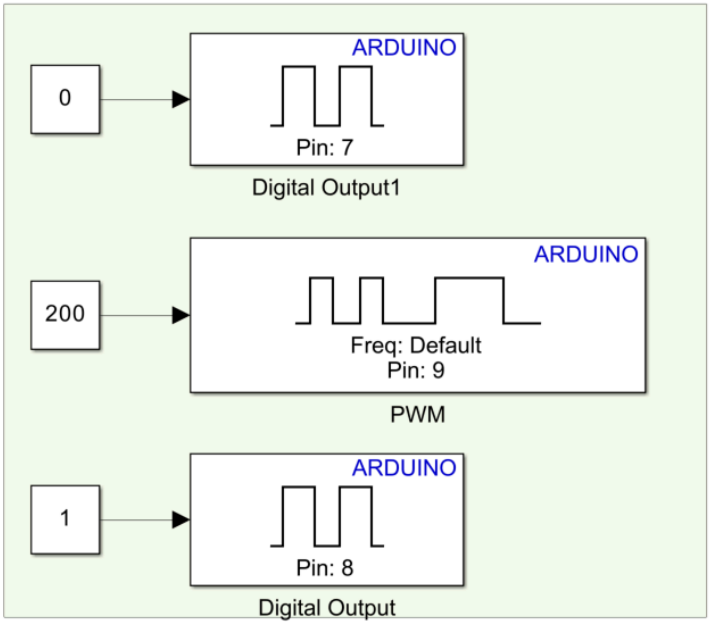

Figure 2: Simulink model for giving Motor Commands from Arduino to Encoder

A separate section to provide an open-loop command to spin the motor for testing purposes.

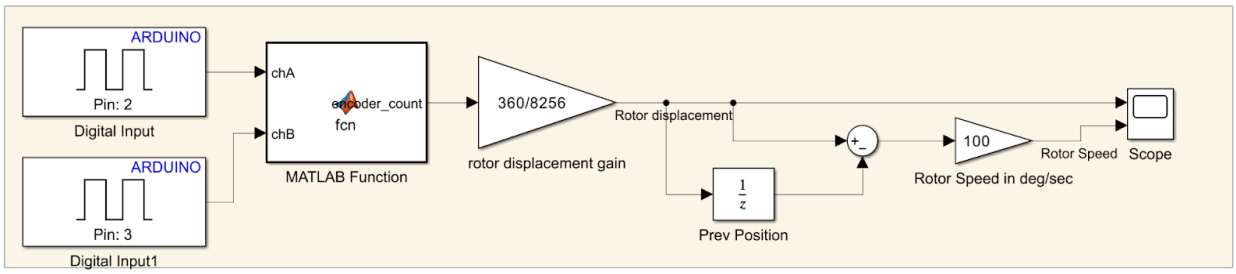

Figure 3: Simulink Model for Encoder Data Acquisition

The model reads digital inputs from pins 2 and 3, processes them in a MATLAB Function block to decode the quadrature signals, converts the raw count to displacement (degrees), and calculates speed (degrees/sec). The Unit Delay block stores the previous time step position and the sum block then gets the change in position, which is then fed into the gain block (1/0.01) to give us the rotor speed in degrees/sec.

3.3 Experimental Results

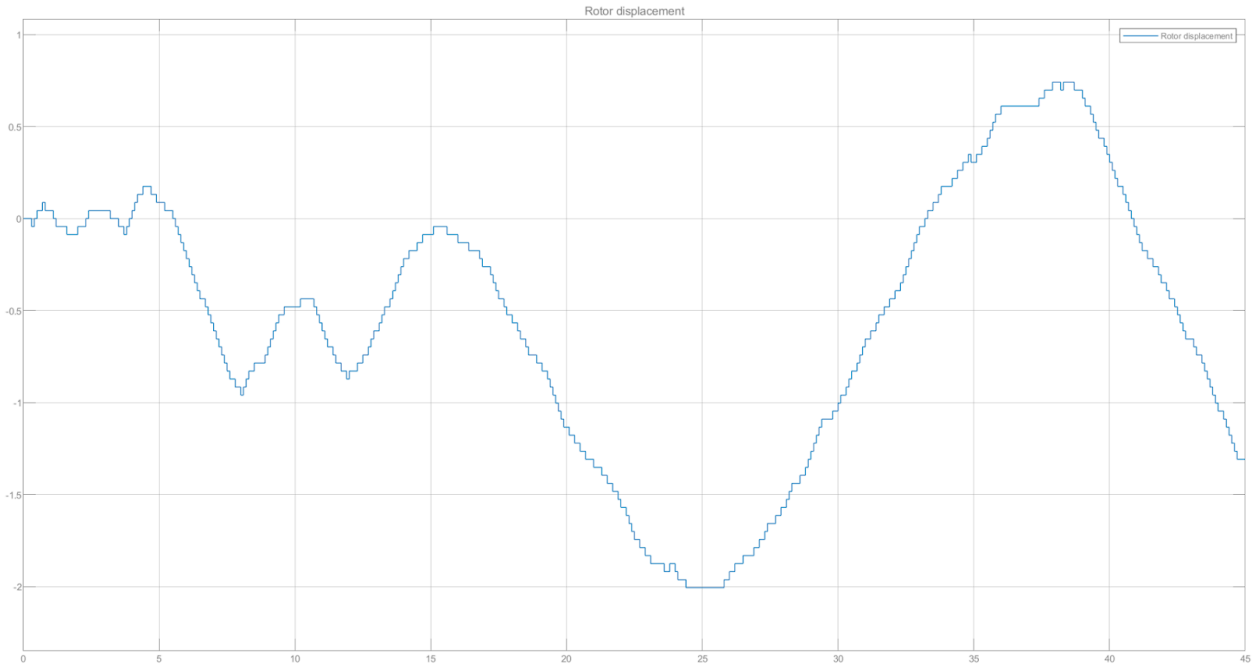

Figure 4: Measured Rotor Displacement vs. Time

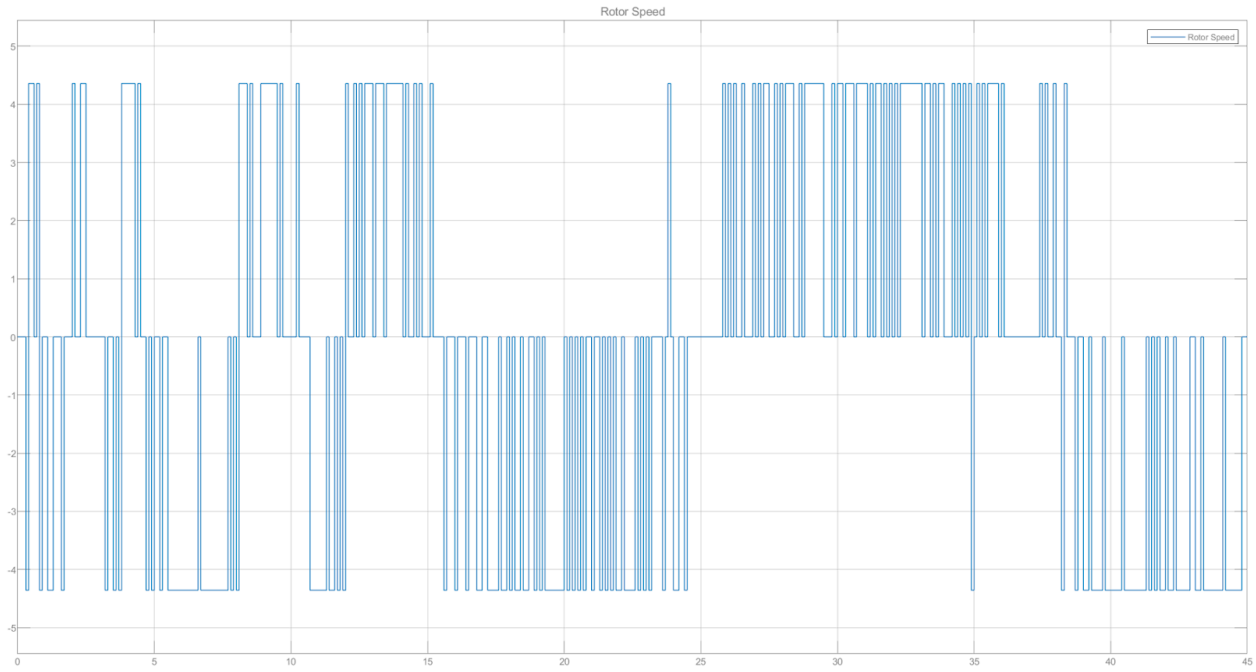

Figure 5: Measured Rotor Speed vs. Time

4 Potentiometer Signal Acquisition

This section demonstrates the successful acquisition of the potentiometer signal into simulink.

4.1 Simulink Implementation

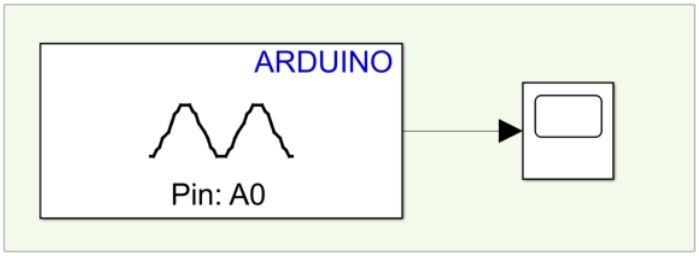

Fig 6: Simulink Model for Potentiometer Reading

An “Analog Input” block is configured to read from pin A0, and the scope displays the real-time value.

4.2 Experimental Results

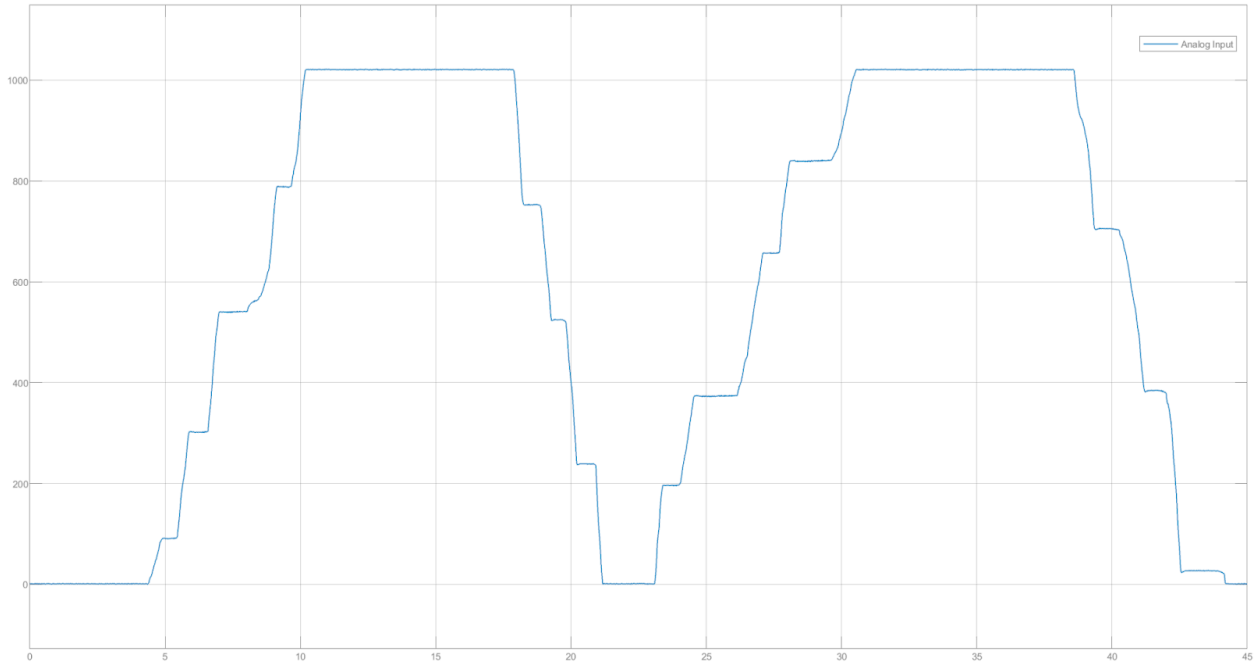

Fig 7: Live Potentiometer Signal

The plot shows how the voltage read by the Arduino varies as the resistance of the potentiometer changes. The potentiometer divides the 5 V supply into a variable output voltage (0–5 V), which the Arduino measures at its analog input pin. The Arduino's analog-to-digital converter (ADC) then converts this analog voltage into a 10-bit digital value that the microcontroller can process. This means the 0–5 V input range is mapped to integer values from 0 to 1023, where: $2^{10} = 1024$ bits. As the knob was manually turned, the signal varied smoothly between the minimum (0) and maximum (1023) values, confirming that the reference signal can be successfully read.

5 Conclusion

Part A of the project was successfully completed. The physical hardware was assembled and wired correctly. The Simulink real-time interface was developed to successfully read and interpret both the potentiometer reference signal and the encoder's feedback signal, including calculations for rotor displacement and speed. The system is now fully prepared for the open-loop experiments and system modeling required in Part B.

Appendix

A.1. MATLAB function block code for encoder decoding

```
function encoder_count = fcn(chA, chB)
persistent count last_state;
if isempty(count)
   count = 0;
   last_state = [0, 0];
end
current_state = double([chA, chB]);
% This lookup table determines the change in count (+1, -1, or 0)
% based on the transition from the last state to the current state.
% Rows are the last state [A,B]: 00, 01, 11, 10
% Columns are the current state [A,B]: 00, 01, 11, 10
state_table = [ 0, -1,  1,  0;  % Last state was 00
                1,  0,  0, -1;  % Last state was 01
               -1,  0,  0,  1;  % Last state was 11
                0,  1, -1,  0]; % Last state was 10
% convert the binary states to decimal indices for the table
last_idx = last_state(1) * 2 + last_state(2) + 1;
current_idx = current_state(1) * 2 + current_state(2) + 1;
% update the count based on the state transition
increment = state_table(last_idx, current_idx);
count = count + increment;
% save the current state for the next time step
last_state = current_state;
encoder_count = count;
```