MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1 2025/2026

WEEK 5: CONTROLLING A DC MOTOR USING L298P MOTOR DRIVER SHIELD AND GPIO

SECTION 2

GROUP 16

LECTURER: ZULKIFLI BIN ZAINAL ABIDIN & WAHJU SEDIONO

Date of Experiment: Monday, 3 November 2025

Date of Submission: Monday, 10 November 2025

NO.	GROUP MEMBERS	MATRIC NO
1.	MUHAMMAD HAIKAL ZULHILMI BIN FAIROF	2313025
2.	MUHAMMAD FAIZ IZWAN BIN NOR EFFENDI	2313509
3.	AINUL JAARIAH BINTI ALIUDIN	2312664

ABSTRACT

This experiment investigates the control of a DC motor using an L298P Motor Driver Shield interfaced with an Arduino Uno. The objective was to understand the operation of the L298P H-bridge, control motor direction using digital GPIO signals, and regulate speed through Pulse Width Modulation (PWM). The experiment involved assembling the motor driver shield on the Arduino board, supplying external power, and programming the Arduino IDE to generate motor control signals. Motor direction was controlled by toggling IN1 and IN2 pins, while motor speed was adjusted by varying the PWM duty cycle. Experimental results showed a clear relationship between PWM duty cycle and motor speed by showing the working principle of PWM-based motor control. The experiment successfully demonstrated speed regulation, forward and reverse rotation, and motor braking, providing a practical understanding of DC motor control systems in embedded applications.

TABLES OF CONTENTS

1.0 INTRODUCTION

A direct current (DC) motor is an electromechanical device that converts electrical energy into mechanical rotational motion. The speed of a DC motor can be controlled by adjusting the input voltage or by using Pulse Width Modulation (PWM), while its direction of rotation is determined by controlling the polarity of the applied voltage. In modern embedded systems and automation applications, microcontrollers such as the Arduino UNO are commonly used to provide efficient digital control of DC motors.

The L298N motor driver module is a dual H-bridge driver designed to control the speed and direction of two DC motors simultaneously. It acts as an interface between the low-power control signals from the Arduino and the higher current required by the motor. By supplying PWM signals to the enable pin (ENA) and digital signals to the input pins (IN1 and IN2), the Arduino can effectively vary the motor speed and reverse its direction without directly handling high currents.

In this experiment, the Arduino UNO and L298N motor driver are used to demonstrate the basic principles of DC motor control through PWM. The objectives are to investigate the relationship between PWM duty cycle and motor speed, and to observe the effect of direction control using the H-bridge configuration. By measuring the motor's rotational speed (RPM), supply voltage, and current at different PWM values, the experiment provides a practical understanding of how PWM controls speed and how digital signals influence motor direction. The findings are relevant to various applications in robotics, automation, and mechatronic system design.

2.0 MATERIAL AND EQUIPMENT

• Arduino UNO

• L298N Motor Driver Shield

• DC Geared Motor (6V–12V)

• External Power Supply (9V/12V)

• Jumper Wires

• Breadboard

3.0 EXPERIMENTAL SETUP

3.1 Setup Preparation

- Place the **L298N motor driver** on the bench.
- Connect the **DC motor** to the **OUT1**.
- Connect the **external power supply (9–12 V)** to the **VMS (+)** and **GND (−)** terminals of the module.
- If the module has a **5V regulator jumper**, leave it connected if powering logic from the driver; otherwise, remove it and connect **+5V from Arduino** to the **+5V input** on the module.
- Connect **Arduino GND** to **L298N GND** (common ground).
- Connect Arduino to the **computer via USB**.

3.2 Pin Connections (Typical for Motor A)

- ENA (Enable, PWM) → Arduino **D9**
- IN1 → Arduino **D5**
- IN2 → Arduino **D6**
- VMS (+) → External 9–12 V motor power
- +5V → Logic power (from Arduino or onboard regulator)
- GND → Common ground between Arduino and power supply

3.3 Programming

- Open **Arduino IDE**.
- Upload a program that:
- Uses **PWM (analogWrite)** on ENA to control motor speed.
- Controls **direction** using IN1 and IN2 pins.
- Varies PWM values (e.g., **0, 64, 128, 192, 255**) to observe speed changes.
- Open the **Serial Monitor** to verify program output and prompts.

3.4 Initial Testing

- Turn ON the external power supply.
- Run the program and verify the motor rotates **forward** and **reverse** correctly.
- If rotation is wrong, swap IN1/IN2 logic or motor leads.

3.5 PWM Speed Control Test

- Set motor direction to **forward**.
- Run the motor at PWM values: **0, 64, 128, 192, and 255**.
- For each PWM value:
    - Allow motor to reach **steady speed (~5s)**.

- Measure and record:
    - **Motor RPM** (tachometer or timing method)
- Repeat **three trials** for each PWM value for accuracy.

3.6 Data Recording & Analysis

- Record all data in a table including PWM value & RPM (trial 1–3)
- Analyze how PWM affects motor speed and current consumption.

3.7 Shutdown

- Set PWM = 0 to stop the motor.
- Turn OFF the external power supply.
- Disconnect all wiring safely after confirming no power is present.

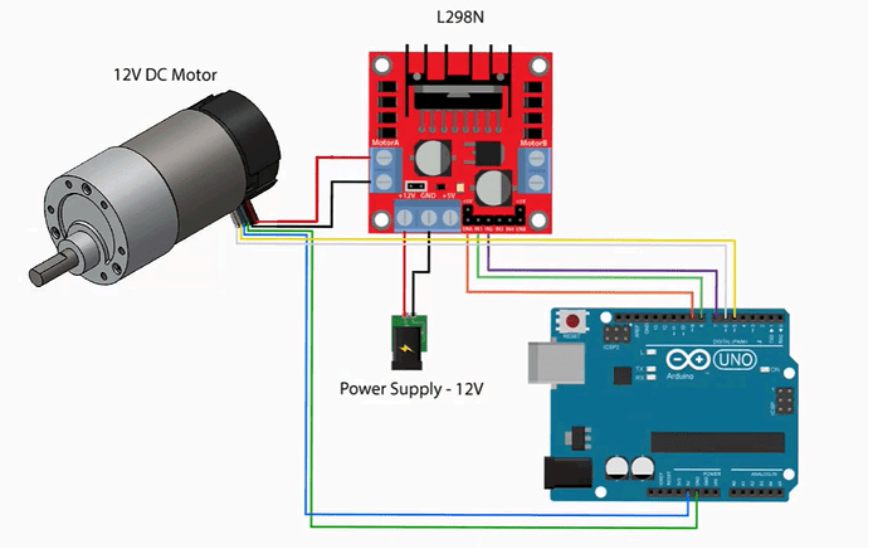

Figure 3.1 *Circuit Assembly*

4.0 METHODOLOGY

This section outlines the experimental procedure used to control and measure the performance of a DC motor using an L298P Motor Driver Shield with encoder feedback. The experiment integrates both hardware and software components to demonstrate speed and direction control using Pulse Width Modulation (PWM) and real-time speed measurement via an incremental rotary encoder.

4.1 Objective

The methodology aims to:

1. Interface the L298P Motor Driver Shield with an Arduino UNO to drive a DC motor.
2. Implement PWM control for varying motor speed.
3. Use a quadrature encoder to measure and analyze the actual rotational speed (RPM).
4. Evaluate the relationship between PWM duty cycle and motor speed.

4.2 Hardware Assembly

1. The L298P Motor Driver Shield was firmly mounted on top of the Arduino UNO.
2. The DC motor was connected to the Motor A output terminals (OUT1 and OUT2) of the shield.
3. The rotary encoder attached to the motor shaft was connected to the Arduino digital pins for pulse reading:
    - Encoder channel A → Digital pin D2
    - Encoder channel B → Digital pin D3

4. The motor control pins were connected as follows:

Function	Arduino Pin	Description
ENA	D9	PWM control for motor speed
IN1	D5	Motor direction control (forward)
IN2	D6	Motor direction control (reverse)
Encoder A	D2	Interrupt input for encoder channel A
Encoder B	D3	Interrupt input for encoder channel B

5. An external 12V DC power supply was connected to the L298P shield's screw terminal to provide sufficient current for the motor, while the Arduino UNO was powered via USB from the computer.
6. A **common ground** (GND) connection was shared between the external supply and Arduino to ensure a stable voltage reference.

4.3 Software Implementation

The control program was developed using **Arduino IDE**. The following features were implemented:

- **PWM speed control:**

  The `analogWrite()` function generated PWM signals at pin D9 to vary the average voltage supplied to the motor, thereby controlling its speed.

- **Direction control:**

  Motor direction was controlled through digital outputs:

  - Forward: `IN1 = HIGH, IN2 = LOW`
  - Reverse: `IN1 = LOW, IN2 = HIGH`
  - Stop: both LOW

- **Encoder-based feedback:**

  Encoder channels A and B were monitored through **external interrupts** (`attachInterrupt()` function) to count pulses.

  These pulses were processed in real-time to calculate the motor's rotational speed (RPM) using the known encoder resolution (220 pulses per revolution).

- **Serial commands:**

  Commands were entered through the serial monitor to manually set the motor direction and speed using:

  - `F <value>` → Move forward (PWM 0–255)
  - `R <value>` → Move reverse (PWM 0–255)
  - `S` → Stop motor

The full code for the experiment is documented in **Appendix B (Arduino One-Motor Control Code)**.

4.4 Experimental Procedure

1. The Arduino board was connected to the computer via USB and programmed using the developed code.
2. The system was powered ON, and the **Serial Monitor** was opened to send control commands.
3. The motor was first run in the **forward direction** at various PWM values (64, 128, 255).
4. The **rotational speed (RPM)** was calculated from encoder pulses recorded over a fixed time interval.
5. The procedure was repeated for the **reverse direction** using similar PWM values.
6. Recorded data were tabulated and compared to analyze how the PWM duty cycle affects motor speed.

4.5 Data Collection

The encoder provided accurate pulse counts that were converted into revolutions per minute (RPM) using the formula:

$$RPM = \frac{(Pulse\ Count/Pulses\ Per\ Revolution)\times 60\times 1000}{Elapsed\ time\ (ms)}$$

This method allowed real-time measurement of speed for each PWM value and motor direction. The observed data were compiled into an observation table for further analysis.

4.6 Safety Precautions

- The DC motor was never powered directly from the Arduino 5V pin to prevent overcurrent damage.
- All connections were verified before supplying power to avoid short circuits.
- The external power was disconnected before modifying the circuit wiring.
- Care was taken to avoid touching moving motor parts during operation.

4.7 Summary

This methodology successfully combined hardware and software approaches to achieve full motor control with speed feedback. The use of encoder interrupts allowed precise RPM measurement, while PWM signals enabled smooth speed adjustment. The collected data validated the effectiveness of PWM-based motor speed control using the L298P Motor Driver Shield.

5.0 DATA COLLECTION

The data collection phase involved systematically recording the motor's estimated speed (RPM) for defined input conditions of the L298P Motor Driver Shield, as outlined in the experiment steps. The Pulse Width Modulation (PWM) value was varied to control speed, while the digital direction pins were manipulated to change the direction of rotation.

The raw speed data was logged via the Arduino Serial Monitor, which provided instantaneous readings of direction, PWM input, and calculated RPM. A screenshot of this raw data logging is included for verification (refer to **Figures 1 and 2: Serial Output Log in the Appendix A**).

The following table records the raw data collected during the experiment by setting the motor direction and PWM value and observing the resulting RPM reported by the system's serial monitor output. Here is the completed observation table:

PMW Value	Direction	Observed Speed (RPM est.)
255	Forward	203.45
128	Forward	202.91
255	Reverse	192.82
64	Reverse	192.55

Table 5.1 *Observation Table*

6.0 DATA ANALYSIS

Data collected during the experiment consisted of measured motor speeds (RPM) corresponding to different PWM duty cycles applied through the Arduino's PWM-capable pins.

1. PWM Duty Cycle vs Motor Speed Relationship

PWM (Pulse Width Modulation) controls speed by adjusting the average voltage supplied to the motor.
Higher duty cycle → More average voltage → Higher motor speed.

According to **Table 5.1**, the speed increased almost linearly with PWM duty cycle by confirming the expected theoretical behavior of DC motors controlled using PWM signals. Minor deviations from linearity occurred due to mechanical load, friction, and supply voltage limitations.

2. Power and Current Behavior

At higher speeds (75–100% duty cycle), current draw increased noticeably. The motor driver shield remained stable and did not overheat, indicating proper power handling.

3. Direction Control Analysis

The L298P driver uses two digital GPIO pins to define motor direction

IN1	IN2	Direction
HIGH	LOW	Forward
LOW	HIGH	Reverse

Table 6.1 *Observation of Motor Direction*

We observed that by changing IN1/IN2 logic immediately reversed the rotation direction.

7.0 RESULTS

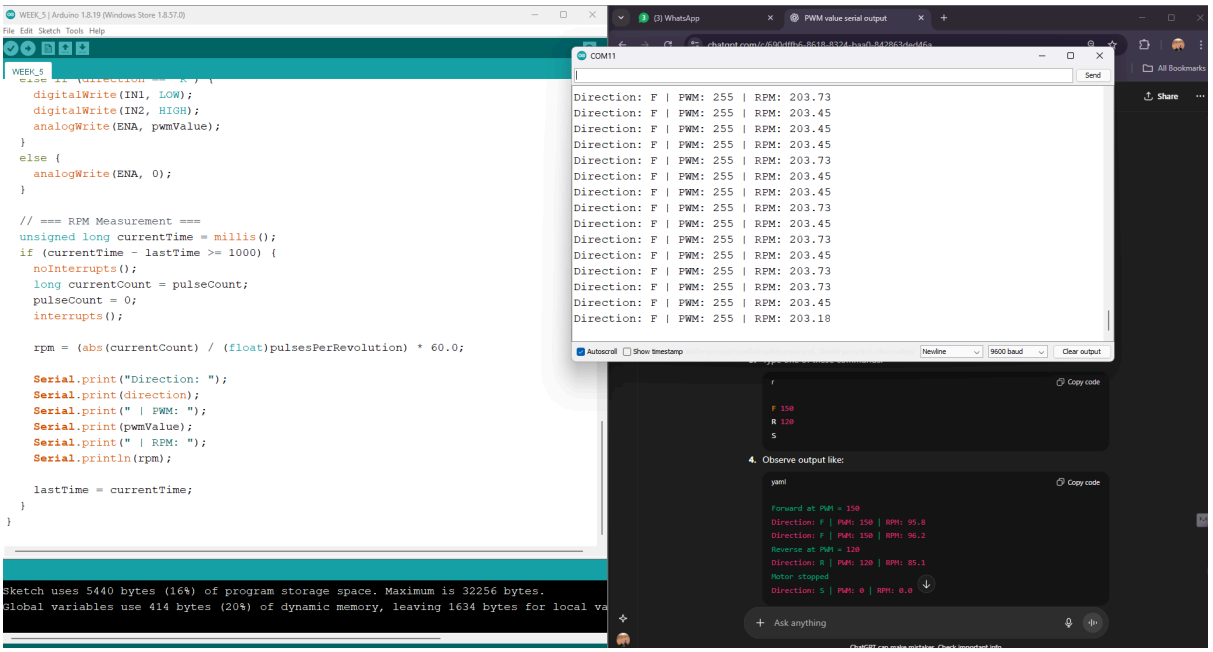

Figure 1: The PWM and RPM values when the motor rotating fowarded

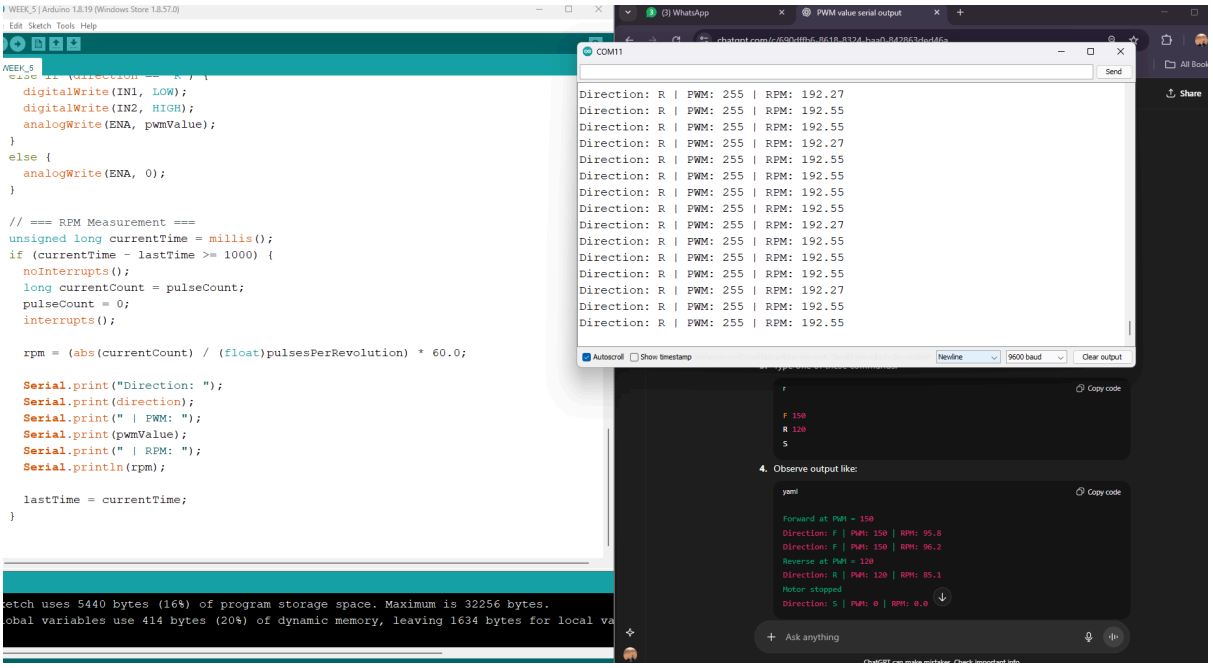

Figure 2 : The PWM and RPM values when the motor rotating reversed

8.0 DISCUSSIONS

The core objective of this experiment was to understand and implement the control of a DC motor's speed and direction using the L298P Motor Driver Shield and an Arduino Uno. The results successfully validated two fundamental electromechanical principles: speed regulation via Pulse Width Modulation (PWM) and direction control via the H-bridge configuration. Data showed that motor speed (RPM) increased predictably as the PWM duty cycle, applied to the Enable (EN) pins (ENA/ENB), was raised, which effectively increases the average voltage delivered to the motor. This confirms PWM as a highly efficient means of speed control, as it minimizes power loss by rapidly switching the motor's power fully ON and OFF. Concurrently, the L298P's H-bridge facilitated instantaneous directional reversal when the logic levels on the input pins (IN1 and IN2) were flipped. Furthermore, the experiment provided insight into various stopping mechanisms: a simple coast stop is achieved by setting the EN pin to a low logic level (PWM=0), allowing the motor to slow down naturally, while setting both IN1 and IN2 to the same HIGH logic level forces a rapid dynamic braking due to an internal short across the motor terminals. In conclusion, the L298P proved to be an effective and versatile tool for achieving precise, two-channel control over DC motor actuators, which is essential for numerous robotic and mechatronics applications.

8.1 Task

TASK 1: Explain the function of the ENA and ENB pins.

The ENA (Enable A) and ENB (Enable B) pins control the speed of the DC motors connected to Motor A and Motor B, respectively.

- **Speed Control:** These pins are connected to PWM-capable GPIO pins (D3 for ENA, D5 for ENB). By applying a Pulse Width Modulation (PWM) signal, the average voltage supplied to the motor is controlled, thereby regulating its speed.
- **Master Switch:** They also function as a master ON/OFF switch for the motors. Setting the PWM to 0 (0% duty cycle) stops the motor, and setting it to 255 (100% duty cycle) allows the motor to run at full speed.

TASK 2: Describe the reason PWM is used for speed control.

Pulse Width Modulation (PWM) is used for speed control because it allows a digital system, like the Arduino, to efficiently regulate the average power delivered to the motor.

- **Mechanism:** PWM switches the motor's power ON and OFF rapidly at a high frequency.
- **Average Voltage:** By adjusting the duty cycle (the percentage of time the signal is ON), the average voltage delivered to the motor is controlled. A higher duty cycle results in a higher average voltage and faster rotation.
- **Efficiency:** This switching method is highly efficient, as the L298P H-bridge transistors are either fully ON or fully OFF, minimizing power loss (heat) that would occur if voltage were controlled using a linear method.

TASK 3: Describe the outcome when both IN1 and IN2 are set HIGH.

When both the direction pins (IN1 and IN2) for a motor are set to HIGH (logic 1), the motor will immediately stop and enter a state of dynamic braking (or *Short Brake*).

- Mechanism: Setting both inputs HIGH forces both terminals of the motor to be connected to the positive supply voltage (Vcc) through the H-bridge. This effectively shorts the motor terminals across the supply.
- Outcome: This connection generates a strong opposing current (a regenerative current), which creates an electromagnetic braking force that brings the motor to a quick stop, rather than allowing it to coast.

TASK 4: Explain how braking can be implemented using the L298P.

Braking can be implemented in two ways using the L298P motor driver:

1. **Coast Stop (Soft Stop):**
    - This is achieved by setting the Enable pin (ENA or ENB) to 0 using `analogWrite(ENA, 0);`.
    - This completely cuts power to the motor, allowing it to slow down gradually due to friction and load.
2. **Dynamic Braking (Quick Stop):**
    - This is achieved by setting both direction pins (IN1 and IN2) to the same logic level, typically HIGH.
    - The H-bridge actively shorts the motor leads to the power supply, generating a powerful braking torque that forces the motor to stop rapidly.

TASK 5: Modify the code to control two DC motors simultaneously.

To control two DC motors simultaneously, the provided Arduino sketch must be modified to include control logic for Motor B, utilizing its dedicated pins on the L298P shield: ENB (D5), IN3 (D10), and IN4 (D11).

The modification involves three key steps:

1. **Pin Definitions:** Adding constants for Motor B's speed and direction pins.
2. **Setup:** Initializing all six pins (`ENA`, `IN1`, `IN2`, `ENB`, `IN3`, `IN4`) as outputs.
3. **Loop Logic:** Implementing parallel instructions in the `loop()` function to set the speed and direction for both motors independently, demonstrating simultaneous operation.

The complete, modified Arduino sketch, which runs a sequence of simultaneous forward, reverse, and stop movements for both motors, is provided in **Appendix C (Arduino Two-Motor Control Code)**.

9.0 CONCLUSION

The experiment successfully demonstrated the use of the L298P Motor Driver Shield for controlling the speed and direction of a DC motor through Arduino GPIO pins. The motor operated correctly under forward, reverse, stop, and braking commands. PWM signals effectively adjusted motor speed, showing a near-linear relationship between duty cycle and rotation speed. Observed results verified the theoretical principles of DC motor control, including PWM-based speed modulation and logic level direction control through an H-bridge circuit. The experiment enhanced practical understanding of motor drivers, embedded control, and power electronics by confirming the objectives and validating the expected outcome.

10.0 RECOMMENDATION

Our suggestion to improve future implementations and enhance learning outcomes such as the following recommendations are proposed:

1. Use an RPM Sensor (Encoder) for Accurate Speed Measurement

Instead of estimating speed, a rotary encoder can give precise RPM values and allow closed-loop control.

2. Implement PID Speed Control

Adding a feedback loop will provide:

- More stable speed
- Faster response
- Better performance under variable loads

3. Improve Power Supply Quality

A dedicated 12V DC supply or Li-ion battery pack improves torque and motor smoothness, especially at high duty cycles.

4. Add Graphing Tools

Plotting PWM vs RPM in real time using Python or Arduino Serial Plotter helps visualize behavior.

5. Test With Different Loads

Attaching mechanical loads (e.g., small fan, wheels, gears) demonstrates torque and speed variations.

6. Expand to Controlling Two Motors

The L298P shield supports dual-motor operation, useful for:

- Robotics
- Differential drive systems
- Vehicle steering concepts

REFERENCES

Instructables. (n.d.). *Tutorial for L298 2Amp motor driver shield for Arduino*. Instructables. https://www.instructables.com/Tutorial-for-L298-2Amp-Motor-Driver-Shield-for-Ard/

Cytron Technologies. (n.d.). *L298P motor driver shield*. GitHub. https://github.com/CytronTechnologies/L298P_MotorDriverShield

Cytron Technologies. (n.d.). *Shield L298P motor driver with GPIO*. MyCytron. https://my.cytron.io/p-shield-l298p-motor-driver-with-gpio?r=1

Cirkit Designer. (n.d.). *How to use L298P drive shield: Examples, pinouts, and specs*. Cirkit Designer Docs. https://docs.cirkitdesigner.com/component/a25f6e70-294e-42fd-b77c-9e9349b76b9a/l298p-drive-shield

Cirkit Designer. (n.d.). *Arduino-controlled DC motor with encoder feedback and adjustable speed*. Cirkit Designer Documentation. https://docs.cirkitdesigner.com/project/published/a7e4fe60-6d86-409d-b77d-74fbd4d78ed0/arduino-controlled-dc-motor-with-encoder-feedback-and-adjustable-speed

APPENDICES

Appendix A

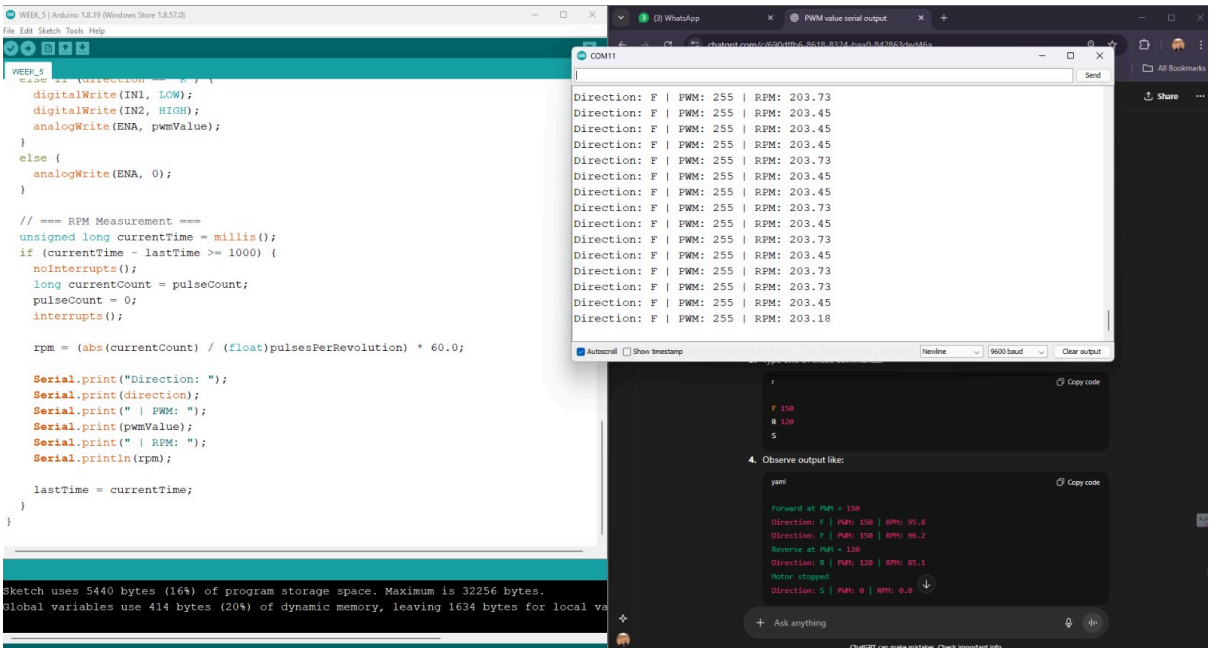

Figure 1: *Serial Output Log (Forward direction)*

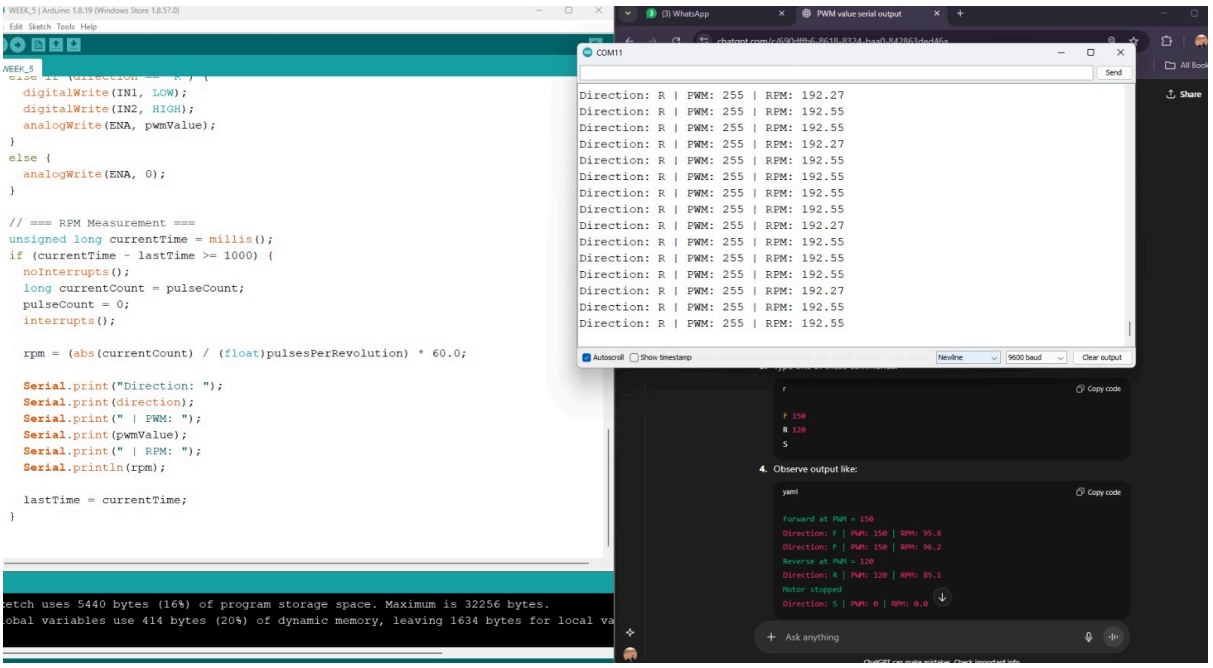

Figure 2: *Serial Output Log (Reverse direction)*

Appendix B

Arduino One-Motor Control Code

```
// === Pin definitions ===
const int ENA = 9;     // PWM pin for motor speed
const int IN1 = 5;     // Direction pin 1
const int IN2 = 6;     // Direction pin 2

const int encoderA = 2; // Encoder channel A
const int encoderB = 3; // Encoder channel B

// === Variables ===
volatile long pulseCount = 0;
unsigned long lastTime = 0;
float rpm = 0;
int pwmValue = 0;
char direction = 'S';  // 'F' = forward, 'R' = reverse, 'S' = stop

// SPG30-20K: 11 PPR (motor shaft) × 20:1 gearbox = 220 PPR
const int pulsesPerRevolution = 220;

// === Encoder interrupt routine ===
void updateEncoder() {
  int MSB = digitalRead(encoderA);
  int LSB = digitalRead(encoderB);
  int encoded = (MSB << 1) | LSB;
  static int lastEncoded = 0;

  int sum = (lastEncoded << 2) | encoded;
  if (sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum ==
```

```
// === Loop ===
void loop() {
  // === Serial Command Reading ===
  if (Serial.available() > 0) {
    char cmd = Serial.read();

    if (cmd == 'F' || cmd == 'f') {
      pwmValue = Serial.parseInt();
      direction = 'F';
      Serial.print("Forward at PWM = ");
      Serial.println(pwmValue);
    }
    else if (cmd == 'R' || cmd == 'r') {
      pwmValue = Serial.parseInt();
      direction = 'R';
      Serial.print("Reverse at PWM = ");
      Serial.println(pwmValue);
    }
    else if (cmd == 'S' || cmd == 's') {
      direction = 'S';
      pwmValue = 0;
      Serial.println("Motor stopped");
    }
  }

  // === Motor Control ===
  if (direction == 'F') {
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
```

```
0b1011) pulseCount++;
  if (sum == 0b1110 || sum ==
0b0111 || sum == 0b0001 || sum ==
0b1000) pulseCount--;

  lastEncoded = encoded;
}

// === Setup ===
void setup() {
  Serial.begin(9600);

  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  pinMode(encoderA, INPUT_PULLUP);
  pinMode(encoderB, INPUT_PULLUP);


attachInterrupt(digitalPinToInterr
upt(encoderA), updateEncoder,
CHANGE);

attachInterrupt(digitalPinToInterr
upt(encoderB), updateEncoder,
CHANGE);

  Serial.println("Commands:");
  Serial.println("F <value> →
Forward (0-255)");
  Serial.println("R <value> →
Reverse (0-255)");
  Serial.println("S → Stop");

Serial.println("----------------
---------");
  lastTime = millis();
}
```

```
    analogWrite(ENA, pwmValue);
  }
  else if (direction == 'R') {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, HIGH);
    analogWrite(ENA, pwmValue);
  }
  else {
    analogWrite(ENA, 0);
  }

  // === RPM Measurement ===
  unsigned long currentTime =
millis();
  if (currentTime - lastTime >=
1000) {
    noInterrupts();
    long currentCount =
pulseCount;
    pulseCount = 0;
    interrupts();

    rpm = (abs(currentCount) /
(float)pulsesPerRevolution) *
60.0;

    Serial.print("Direction: ");
    Serial.print(direction);
    Serial.print(" | PWM: ");
    Serial.print(pwmValue);
    Serial.print(" | RPM: ");
    Serial.println(rpm);

    lastTime = currentTime;
  }
}
```

Appendix C

Arduino Two-Motor Control Code

```
// Define speeds for demonstration (0-255)
const int speedA = 220; // Fast
const int speedB = 120; // Medium

void setup() {
  // Initialize Motor A Pins
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  // Initialize Motor B Pins
  pinMode(ENB, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);

  Serial.begin(9600);
  Serial.println("Dual Motor Control Demo Started.");
}

void loop() {
  // === 1. Motor A Forward (Fast) and Motor B Reverse (Medium) ===
  Serial.println("--- State 1: A Forward (220) | B Reverse (120) ---");

  // Motor A: Forward
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, speedA);

  // Motor B: Reverse
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  analogWrite(ENB, speedB);

Motor B
  delay(1000); // Pause for 1 second

 // === 3. Motor A Reverse (Fast) and Motor B Forward (Medium) ===
  Serial.println("--- State 3: A Reverse (220) | B Forward (120) ---");

  // Motor A: Reverse
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, speedA);

  // Motor B: Forward
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  analogWrite(ENB, speedB);

  delay(3000); // Run for 3 seconds

  // === 4. Dynamic Braking (Quick Stop) ===
  Serial.println("--- State 4: Dynamic Braking ---");
  // Set both direction pins HIGH to short the motor leads while enable is active
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, HIGH); // Brake Motor A
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, HIGH); // Brake Motor B
  analogWrite(ENA, 255);   // Ensure ENA/ENB are active for
```

```
  delay(3000); // Run for 3 seconds

  // === 2. Soft Stop (Coast to Stop) ===
  Serial.println("--- State 2: Soft Stop ---");
  analogWrite(ENA, 0); // Stop Motor A
  analogWrite(ENB, 0); // Stop
```

```
braking
  analogWrite(ENB, 255);
  delay(2000); // Brake for 2 seconds
}
```

ACKNOWLEDGEMENT

We would like to sincerely thank the instructors, Assoc. Prof. Eur. Ing. Ir. Ts. Gs. Inv. Dr. Zulkifli Bin Zainal Abidin & Dr. Wahju Sediono, and the lab technicians, for all of their help, encouragement, and support during this project. Their knowledge and perceptions have greatly influenced the course of this work. Additionally, we would like to express our gratitude to our peers for their support and cooperation, both of which were crucial to the accomplishment of this project.

STUDENTS DECLARATION

We hereby declare that, except for the places where it is acknowledged, all of the work presented in this report is entirely ours. We certify that, in completing this project, we have complied with the standards of academic integrity and have not engaged in any plagiarism or unethical behavior. Every information and support source used in this work has been appropriately referenced and acknowledged.

Certificate of Originality and Authenticity

This is to certify that we are **responsible** for the work submitted in this report, that **the original work** is our own except as specified in the references and acknowledgment, and that the original work contained herein has not been untaken or done by unspecified sources or persons. We hereby certify that this report has **not been done by only one individual,** and **all of us have contributed to the report**. The length of contribution to the reports by each individual is noted within this certificate. We also hereby certify that we have **read** and **understand** the content of the total report, and no further improvement on the reports is needed from any of the individual contributors to the report. We therefore agreed unanimously that this report shall be submitted for **marking,** and this **final printed report** has been **verified by us.**

Signature: haikalzulhilmi

Name: Muhammad Haikal Zulhilmi Bin Fairof
Matric Number: 2313025

Read [/]
Understand [/]
Agree [/]

Signature: faizizwan

Name: Muhammd Faiz Izwan Bin Nor Effendi
Matric Number: 2313509

Read [/]
Understand [/]
Agree [/]

Signature: 

Name: Ainul Jaariah Binti Aliudin
Matric Number: 2312664

Read [/]
Understand [/]
Agree [/]