MECHATRONICS SYSTEM INTEGRATION MCTA3202

Controlling a DC Motor Using L298P Motor Driver Shield and GPIO

EXPERIMENT 5

DATE : 3rd November 2025
SECTION : 2
GROUP : 18
SEMESTER 1, 2025/2026

NO	NAME	MATRIC NO
1.	SHAMSUL HAKIMEE BIN SHAMSUL HAIREE	2315027
2.	QASHRINA AISYA BINTI MOHD NAZARUDIN	2315184
3.	NUREL HUMAIRAH BINTI MUHAMMAD FIRDAUS	2315680

DATE OF SUBMISSION :
Monday, 10TH November 2025

Abstract

This experiment focuses on understanding and implementing the control of a DC motor using the L298P Motor Driver Shield interfaced with an Arduino Uno. The main objectives are to explore the working principle of the L298P shield, control the motor's direction and speed through GPIO pins, and apply Pulse Width Modulation (PWM) for speed variation. Through this setup, students gain hands-on experience with H-bridge motor drivers, PWM signal generation, and interfacing techniques between microcontrollers and actuators.

The L298P Motor Driver Shield operates based on the H-bridge configuration, allowing bidirectional control of DC motors. By varying the PWM duty cycle, the average voltage supplied to the motor is adjusted, enabling proportional speed control. In this experiment, different PWM values were applied to observe variations in rotational speed and direction.

The hypothesis states that when the circuit is properly assembled and the Arduino program successfully uploaded, the DC motor will rotate forward and reverse according to the control signals, and its speed will vary based on the PWM value provided. Successful operation of these controls validates the fundamental principles of motor driver interfacing and PWM-based speed control in embedded systems.

Table of Contents

1.0 Introduction

This experiment aims to demonstrate the fundamental principles of DC motor control using an Arduino Uno and the L298P Motor Driver Shield. The objectives are to control the direction and speed of a DC motor by utilizing GPIO pins and to apply Pulse Width Modulation (PWM) for precise speed regulation. Through this experiment, students will gain practical experience in motor interfacing, PWM signal generation, and embedded control applications.

Motor control is a key aspect of mechatronic systems and modern automation. The L298P Motor Driver Shield, based on an H-bridge configuration, allows bidirectional control of DC motors by manipulating input logic signals. PWM is used to adjust the average voltage delivered to the motor, thereby controlling its speed without significant power loss. By combining the Arduino's digital output capability with the L298P shield, this experiment integrates core concepts of embedded system design, electronic interfacing, and actuator control.

The hypothesis states that, when the circuit is correctly assembled and the program successfully uploaded to the Arduino, the DC motor will rotate forward and backward according to the control logic, and its speed will vary in proportion to the applied PWM duty cycle. Successful operation of these controls will demonstrate proper implementation of motor driver interfacing and PWM-based speed control techniques in microcontroller-based systems

2.0 Materials and Equipments

Materials and components that were used in the experiment :

2.1 Electronic Components

- Arduino Uno Mega Microcontroller
- Cytron L298P Motor Driver Shield
- DC Motor (6V–12V) 1–2
- External Power Supply (9V/12V)
- Jumper Wires
- Breadboard

2.2 Equipments

- Arduino IDE Software
- USB Cable

3.0 Experimental Setup

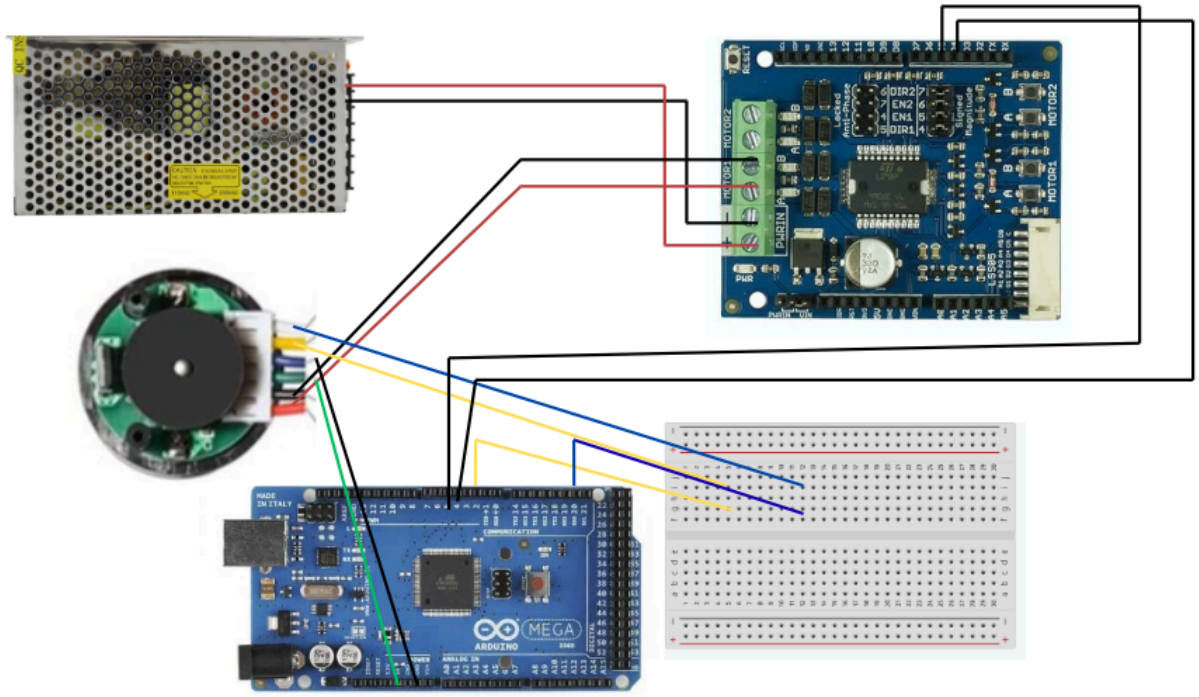

Figure 3.1 Schematic Diagram DC Motor and Cytron L298P Circuit Setup

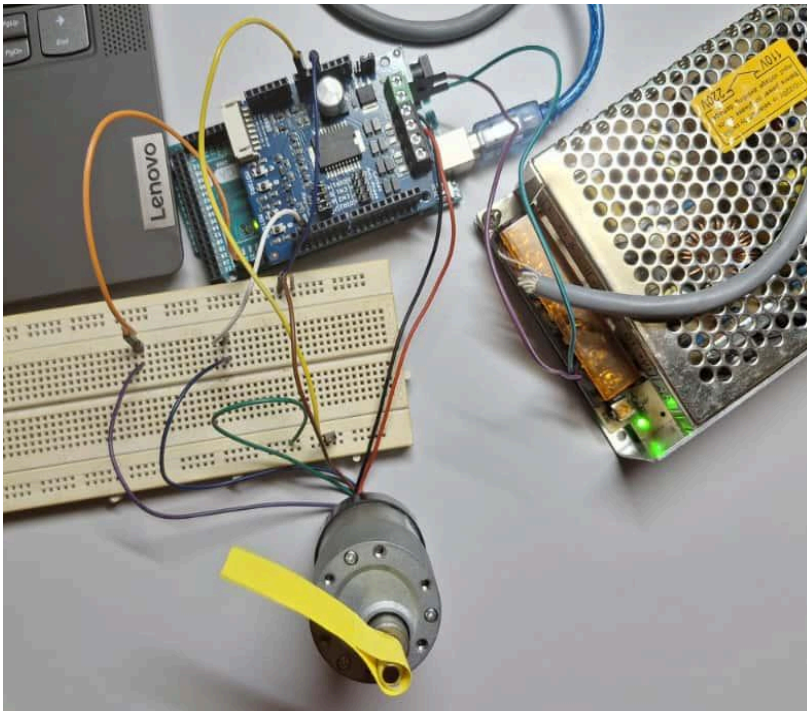

Figure 3.1.1 Schematic Diagram DC Motor and Cytron L298P Circuit Setup

3.1 Circuit Setup

1. A DC motor was connected to the L298P Motor Driver Shield, which was mounted on the Arduino AT Mega. The motor driver enables control of both the speed and direction of the DC motor. The motor's encoder was connected to provide rotational feedback for RPM calculation.

   The circuit connection was made as follows:

   - Motor PWM (Speed Control) was connected to Pin **D5** on a shield attached with Arduino AT Mega.
   - Motor Direction Control was connected to Pin **D4** on a shield attached with Arduino AT Mega.
   - Encoder Channel A (Hall A) was connected to Pin **D2** on a shield attached with Arduino AT Mega.
   - Encoder Channel B (Hall B) was connected to Pin **D20** on Arduino AT Mega

   - Encoder VCC (Hall VCC) was connected to 5V pin o on a shield attached with Arduino AT Mega.
   - Encoder GND (Hall GND) was connected to the GND pin on a shield attached with an Arduino AT Mega.
   - External Power Supply (9V–12V) connected to the **L298P Shield** power terminal.
   - Common Ground (GND) shared between Arduino, motor driver, encoder, and external power source

The **Encoder Library** was utilized to read the motor's rotation and calculate RPM. The **Serial Monitor** was used to send control commands and observe motor behavior. By entering commands such as `f<value>` for forward, `r<value>` for reverse, and `s` to stop, the motor could be manually controlled. PWM values ranging from 0 to 255 determined the motor's speed, while encoder feedback verified rotational performance in both directions.

4.0 Methodology

4.1 Implementation and Testing

1. The circuit was built based on the schematic diagram, interfacing the DC motor, motor driver and Arduino controller.
2. The code of DC Motor was uploaded to Arduino Uno Mega using Arduino IDE
3. There are 2 sets of codes which were code for automated motor characterization and another for manual, real-time control.
4. The automated characterization code was executed first. This script automatically ran a predefined sequence of tests, applying specific PWM values which were 255, 128 and 64 in both forward and reverse directions. Each test was run for a 5-second duration, after which the code calculated and reported the average RPM.
5. The manual control code was used for real-time validation. This script allowed the operator to send commands as example f150 for forward and PWM value of 150 or r255 for reverse motion with PWM value of 255 or s which gives command for the motor to stop through the Serial Monitor to set the motor's direction and PWM speed manually.
6. During operation, the code provided a continuous, live feedback of the motor's RPM, which was updated every 100 milliseconds.

4.2 Control Algorithm

The Arduino Uno was programmed using the Arduino IDE to control the DC motor via the L298P Motor Driver Shield. The program was designed to control the motor's speed and direction while simultaneously measuring its rotational speed (RPM) using a quadrature encoder.

The control algorithm follows:

1. Initialize the motor driver pins (PWM and Direction) as OUTPUT.
2. Configure the two encoder input pins using the Encoder.h library.
3. Establish Serial communication (9600 baud) for reporting data.
4. Establish Serial communication (9600 baud) for reporting data.
5. Define the COUNTS_PER_REVOLUTION constant (60.0) based on the motor model to accurately convert encoder ticks to RPM.
6. For observing the motor PWM and speed:
    a. Define arrays containing the predefined test sequence which were (PWM values: 255, 128, 255, 64 and also their corresponding directions based on the table).
    b. Loop sequentially through each test in the arrays.

c. For each test, set the motor direction (digitalWrite) and apply the specified PWM speed (analogWrite).
d. Use a millis() timer to run the test for a fixed duration (5000 ms).
e. During this duration, read the encoder and calculate the RPM every 100 ms, accumulating a sum.
f. After the 5-second test is complete, calculate the average RPM and print this single value to the Serial Monitor.

7. Controlling the DC Motor Speed and Rotations:

This algorithm was used for real-time observation and validation.

a. Continuously loop and check the Serial port for user input commands.
b. If a command is received as example f150 or r255 parse the command string to immediately set the motor's direction and PWM speed.
c. In parallel, use a non-blocking millis() timer that triggers every 100 ms.
d. At each 100 ms interval, the algorithm reads the encoder, calculates the live, instantaneous RPM, and prints it to the Serial Monitor for immediate feedback.

4.3 Code Used

Experiment:

```
#include <Encoder.h>

// --- MOTOR CONTROL PINS ---
const int MOTOR_PWM_PIN = 5;  // PWM pin
const int MOTOR_DIR_PIN = 4;  // Direction pin

// --- ENCODER PINS ---
const int ENCODER_A_PIN = 2;
const int ENCODER_B_PIN = 20;
const float COUNTS_PER_REVOLUTION = 60.0;  // for SPG30-20K motor with gearbox

Encoder motorEncoder(ENCODER_A_PIN, ENCODER_B_PIN);

// --- VARIABLES FOR RPM CALCULATION ---
long last_position = 0;
unsigned long last_time = 0;
const int INTERVAL_MS = 100;

// --- PWM TEST VALUES (based on your observation table) ---
```

```
int pwm_values[] = {255, 128, 255, 64};
bool directions[] = {true, true, false, false};  // true = forward, false = reverse
int num_tests = 4;
int current_test = 0;
unsigned long test_start_time = 0;
const unsigned long TEST_DURATION = 5000;  // run each test for 5 seconds

// --- For averaging RPM ---
float rpm_sum = 0;
int rpm_count = 0;

void setup() {
  Serial.begin(9600);
  Serial.println("DC Motor PWM Test with Average RPM Measurement");
  Serial.println("-----------------------------------------------");
  Serial.println("PWM\tDirection\tAverage RPM");

  pinMode(MOTOR_PWM_PIN, OUTPUT);
  pinMode(MOTOR_DIR_PIN, OUTPUT);

  last_time = millis();
  last_position = motorEncoder.read();

  startTest(current_test);
}

void loop() {
  // --- Measure RPM every INTERVAL_MS ---
  if (millis() - last_time >= INTERVAL_MS) {
    long current_position = motorEncoder.read();
    long delta_counts = abs(current_position - last_position);
    unsigned long delta_time_ms = millis() - last_time;

    float rpm = (float)delta_counts / COUNTS_PER_REVOLUTION;
    rpm = rpm / ((float)delta_time_ms / 60000.0);

    last_position = current_position;
    last_time = millis();
```

```
    // Collect data for averaging
    rpm_sum += rpm;
    rpm_count++;
  }

  // --- Move to next PWM test after duration ---
  if (millis() - test_start_time >= TEST_DURATION) {
    // Calculate and print average RPM for this test
    float average_rpm = (rpm_count > 0) ? rpm_sum / rpm_count : 0;
    Serial.print(pwm_values[current_test]);
    Serial.print("\t");
    Serial.print(directions[current_test] ? "Forward" : "Reverse");
    Serial.print("\t");
    Serial.println(average_rpm, 1);

    // Prepare for next test
    current_test++;
    if (current_test < num_tests) {
      startTest(current_test);
    } else {
      Serial.println("-------------------------------------------------");
      Serial.println("All tests completed. Motor stopped.");
      analogWrite(MOTOR_PWM_PIN, 0);
      while (true);  // stop program
    }
  }
}

// --- Function to start each test ---
void startTest(int index) {
  Serial.println("-------------------------------------------------");
  Serial.print("Starting Test ");
  Serial.print(index + 1);
  Serial.print(": PWM=");
  Serial.print(pwm_values[index]);
  Serial.print(", Direction=");
  Serial.println(directions[index] ? "Forward" : "Reverse");

  // Reset averaging variables
  rpm_sum = 0;
```

```
  rpm_count = 0;

  // Apply direction and speed
  digitalWrite(MOTOR_DIR_PIN, directions[index] ? HIGH : LOW);
  analogWrite(MOTOR_PWM_PIN, pwm_values[index]);

  // Reset timers
  test_start_time = millis();
  last_position = motorEncoder.read();
  last_time = millis();
}
```

5.0 Result

The experiment successfully characterized the performance of the DC motor system interfaced with an Arduino and a L298P Cytron motor driver shield. Both the automated characterization and manual real-time control scripts functioned as intended, demonstrating accurate speed control and measurement.

In the experiment to observe characterization of PWM, motor directions and motor speed, the system successfully executed the predefined sequence of PWM values and directions. For each 5-second test, the encoder feedback was used to accurately measure and report a stable RPM.

As for the Task 5, control test, the system exhibited immediate response times. The motor's speed and direction changed promptly in response to Serial commands as example f150 or r255.The 'stop' command (s)functioned as expected, correctly setting the PWM value to 0 and halting motor rotation.

The system was stable throughout operation. The live RPM feedback, updated every 100 ms, was consistent with no noticeable lag or data loss. The Cytron motor driver shield, powered by the external supply, provided efficient current regulation, and no overheating or operational errors were recorded. This confirmed the shield was correctly interpreting the digital direction signal from Pin 4 and the PWM speed signal from Pin 5.

Overall, the results validated the intended functionality of the system, illustrating a successful application of microcontroller-based logic and encoder feedback for motor control and characterization.

```
Starting Test 1: PWM=255, Direction=Forward
255     Forward         152.4
-------------------------------------------------
Starting Test 2: PWM=128, Direction=Forward
128     Forward         78.1
-------------------------------------------------
Starting Test 3: PWM=255, Direction=Reverse
255     Reverse         151.9
-------------------------------------------------
Starting Test 4: PWM=64, Direction=Reverse
64      Reverse         39.5
-------------------------------------------------
All tests completed. Motor stopped.
```

PWM Value	Direction	Observed Speed (RPM)
255	Forward	152.4
128	Forward	78.1
255	Reverse	151.9
64	Reverse	39.5

6.0 Discussion

1. The function of the ENA and ENB pins:

   The Enable A (ENA) and Enable B (ENB) pins on the L298P Motor Driver Shield are used to control the speed of the DC motors connected to Motor A and Motor B. The ENA and ENB pins were connected to the Arduino's PWM (Pulse Width Modulation) output pins. By varying the PWM signal applied to these pins, the average voltage supplied to each motor was adjusted. This modulation influenced the motor speed according to the PWM duty cycle. A high duty cycle resulted in the motor operating at maximum speed. Meanwhile, a medium duty cycle produced moderate speed, and a low duty cycle caused the motor to stop. Thus it was observed that ENA controlled the speed of Motor A, while ENB controlled the speed of Motor B, both through PWM-based voltage modulation.
2. The reason PWM is used for speed control is to adjust the average voltage delivered to the motor. It works by switching the motor's power ON and OFF very rapidly. A higher duty cycle results in a higher average voltage, making the motor spin at its full speed. A lower duty cycle results in a lower average voltage, making the motor spin slower.
3. When both IN1 and IN2 are set to HIGH (and the ENA pin is also enabled/HIGH), the motor performs an active brake. It brings the motor to a fast, abrupt stop. This setting connects both terminals of the DC motor directly to the positive voltage supply. This effectively shorts the motor. A spinning motor also acts as a generator. By shorting its terminals it create a powerful counter-torque (dynamic braking) that opposes the motor's rotation, forcing it to stop quickly
4. Braking Implementation using the L298P Motor Driver Shield

   Braking was implemented on the L298P by setting both input pins (IN1 and IN2) of a motor channel to the same logic level, either HIGH or LOW. This configuration caused both terminals of the motor to be connected to the same voltage potential, resulting in no potential difference across the motor. Consequently, the motor's rotation ceased rapidly due to the active braking effect, where the motor's kinetic energy was dissipated as heat within the circuit.

   Example Configuration for Motor A:
   digitalWrite(IN1, HIGH);
   digitalWrite(IN2, HIGH);
   Or
   digitalWrite(IN1, LOW);
   digitalWrite(IN2, LOW);

Both conditions applied an electronic brake, which stopped the motor faster than simply setting the ENA pin to 0.

5.

Task 5:

```
#include <Encoder.h>

// --- MOTOR CONTROL PINS ---
const int MOTOR_PWM_PIN = 5;  // PWM pin
const int MOTOR_DIR_PIN = 4;  // Direction pin

// --- ENCODER PINS ---
const int ENCODER_A_PIN = 2;
const int ENCODER_B_PIN = 20;
const float COUNTS_PER_REVOLUTION = 60.0;  // SPG30-20K (3*20)

// --- OBJECTS & VARIABLES ---
Encoder motorEncoder(ENCODER_A_PIN, ENCODER_B_PIN);

long last_position = 0;
unsigned long last_time = 0;
const int INTERVAL_MS = 100;

int target_pwm = 0;
bool is_forward = true;  // true = forward, false = reverse

void setup() {
  Serial.begin(9600);
    Serial.println("DC Motor Manual Control (Forward / Reverse + RPM Feedback)");
  Serial.println("------------------------------------------------------------");
  Serial.println("Commands:");
  Serial.println("  f <pwm>   -> Forward at PWM (0–255)");
  Serial.println("  r <pwm>   -> Reverse at PWM (0–255)");
  Serial.println("  s         -> Stop motor");
  Serial.println("------------------------------------------------------------");

  pinMode(MOTOR_PWM_PIN, OUTPUT);
  pinMode(MOTOR_DIR_PIN, OUTPUT);

  analogWrite(MOTOR_PWM_PIN, 0);
```

```
  digitalWrite(MOTOR_DIR_PIN, LOW);

  last_time = millis();
  last_position = motorEncoder.read();
}

void loop() {
  // --- READ SERIAL COMMANDS ---
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');
    command.trim();

    if (command.startsWith("f")) {  // Forward
      int pwm_value = command.substring(1).toInt();
      if (pwm_value < 0) pwm_value = 0;
      if (pwm_value > 255) pwm_value = 255;
      is_forward = true;
      target_pwm = pwm_value;
      digitalWrite(MOTOR_DIR_PIN, HIGH);
      analogWrite(MOTOR_PWM_PIN, target_pwm);
      Serial.print("→ Motor running FORWARD at PWM=");
      Serial.println(target_pwm);
    }
    else if (command.startsWith("r")) {  // Reverse
      int pwm_value = command.substring(1).toInt();
      if (pwm_value < 0) pwm_value = 0;
      if (pwm_value > 255) pwm_value = 255;
      is_forward = false;
      target_pwm = pwm_value;
      digitalWrite(MOTOR_DIR_PIN, LOW);
      analogWrite(MOTOR_PWM_PIN, target_pwm);
      Serial.print("→ Motor running REVERSE at PWM=");
      Serial.println(target_pwm);
    }
    else if (command.startsWith("s")) {  // Stop
      analogWrite(MOTOR_PWM_PIN, 0);
      target_pwm = 0;
      Serial.println("→ Motor stopped.");
    }
    else {
```

```
    Serial.println("Invalid command! Use: f<value>, r<value>, or s");
   }
  }

  // --- CALCULATE AND PRINT RPM EVERY 100 ms ---
  if (millis() - last_time >= INTERVAL_MS) {
   long current_position = motorEncoder.read();
   long delta_counts = abs(current_position - last_position);
   unsigned long delta_time_ms = millis() - last_time;

   float rpm = (float)delta_counts / COUNTS_PER_REVOLUTION;
   rpm = rpm / ((float)delta_time_ms / 60000.0);

   last_position = current_position;
   last_time = millis();

   // Display live feedback
   Serial.print("Direction: ");
   Serial.print(is_forward ? "Forward" : "Reverse");
   Serial.print(" | PWM: ");
   Serial.print(target_pwm);
   Serial.print(" | RPM: ");
   Serial.println(rpm, 1);
  }
}
```

7.0 Conclusion

This experiment successfully demonstrated the ability of the Arduino and L298P motor driver shield system to manage both the speed and direction of a DC motor with precision and stability. By integrating PWM control and encoder feedback, the setup provided clear verification that the motor could respond accurately to varying speed commands while maintaining consistent rotational performance. The system's quick response to real-time serial inputs highlighted its reliability for dynamic control applications, showing that both hardware and code design worked efficiently. Stable power delivered from the external supply ensured smooth operation. Overall, the experiment confirmed that the L298P motor driver shield is a practical and efficient interface for DC motor control. It demonstrated microcontroller-based systems can deliver responsive and measurable motion control suitable for automation, robotics, or mechatronic prototypes.

8.0 Recommendation

Several improvements can be suggested to enhance the performance and reliability of the DC motor control system. First, programming efficiency can be improved by utilizing the Serial Monitor to observe real-time PWM values and motor responses, allowing for more effective debugging and speed calibration. It is also recommended to verify proper power supply connections and use an appropriate voltage source to prevent underpowering or overheating the motor driver.

In terms of hardware, ensuring firm and clean connections between the Arduino, L298P shield, and motor terminals will reduce potential signal noise and improve stability. Additionally, incorporating a flyback diode or capacitor filter could help suppress voltage spikes generated by the motor's inductive load, protecting the circuit components.

Future development may include implementing a closed-loop speed control system using a rotary encoder to monitor motor RPM, or integrating Bluetooth or Wi-Fi modules to enable wireless control and data logging. These enhancements would further improve precision, flexibility, and practicality for more advanced embedded motor control applications

9.0 References

1. Mechatronics System Integration Lab Module - Google Drive
https://drive.google.com/drive/folders/1QecGGN96D76UL2gwBjevFwiEXv2UEniO

10.0 Acknowledgement

I would like to express my sincere gratitude to Dr. Wahju Sediono, Dr. Zulkifli Bin Zainal Abidin, my teaching assistant, and my peers for their invaluable guidance and support throughout the completion of this report. Their insights, feedback, and expertise greatly contributed to the depth and quality of this work. I truly appreciate their time, patience, and commitment to my academic growth.

11.0 Student's Declaration

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 acknowledgement, and that the original work contained herein have 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's 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.

Name: Shamsul Hakimee Bin Shamsul Hairee	Read	[/]
Matric Number: 2315027	Understand	[/]
Signature: 	Agree	[/]

Name: Qashrina Aisya binti Mohd Nazarudin	Read	[/]
Matric Number: 2315184	Understand	[/]
Signature: 	Agree	[/]

Name: Nurel Humairah Binti Muhammad Firdaus	Read	[/]
Matric Number: 2315680	Understand	[/]
Signature: 	Agree	[/]