MECHATRONIC SYSTEM INTEGRATION
(MCTA 3203)
SEMESTER 1 2025/2026

EXPERIMENT 2 : TASK 1 & TASK 2

SERIAL COMMUNICATION ON BETWEEN ARDUINO AND PYTHON : POTENTIOMETER DATA TRANSMISSION AND SERVO MOTOR CONTROL

SECTION 2

GROUP 12

NAME	METRIC NO.
AHMAD FARHAN BIN AHMAD NAHRUDI	2317373
MUHAMMAD AIMAN BIN SAFAWI	2318249
MOHD NAFIZ ZUHAIREE BIN MOHD NEEZAR	2317423

DATE OF SUBMITION :
29/10/2025

ABSTRACT

This experiment aimed to explore the implementation of serial communication between an Arduino microcontroller and a Python program for real-time data monitoring and electromechanical control. The experiment was divided into two main sections: **Task 1** and **Task 2**, each focusing on different aspects of system integration and functionality.

In **Task 1**, the objective was to establish and analyze serial data exchange between Arduino and Python. A potentiometer was connected to the Arduino to generate variable analog input signals. These readings were transmitted to the computer via the serial port and displayed in real time through Python. An LED was also incorporated to indicate the potentiometer's position—turning ON when the value exceeded half of its maximum range and turning OFF otherwise. This task successfully verified smooth communication, accurate data transmission, and responsive LED control.

Task 2 expanded the system by integrating a servo motor and incorporating real-time graphical visualization. The potentiometer values were used to dynamically control the servo motor's angular position, while Python's Matplotlib library was employed to visualize the potentiometer readings continuously. An LED provided additional visual feedback, and both programs featured keyboard interrupt functions for safe termination. Overall, this experiment effectively demonstrated the seamless integration of hardware and software for real-time control, data visualization, and interactive mechatronic applications.

TABLES OF CONTENTS

1.0 INTRODUCTION

In today's world of mechatronics and automation, serial communication is one of the key methods that allow microcontrollers and computers to exchange information effectively. It makes it possible for hardware components to send and receive data in real time, enabling monitoring, control, and system integration. This experiment focuses on understanding and applying serial communication between an Arduino microcontroller and a Python program. The main goal is to establish reliable data transfer, visualize sensor readings, and control hardware devices interactively through software. The experiment consists of two main tasks: **Task 1** introduces the basics of serial communication and data exchange, while **Task 2** expands the concept into real-time servo motor control and graphical data visualization.

In **Task 1**, the objective was to set up serial communication between Arduino and Python for data transmission and monitoring. A potentiometer was connected to the Arduino as an analog input device to provide variable voltage readings. These readings were then sent to the computer through the serial port, where Python received and displayed them in real time. An LED was added to indicate the potentiometer's position—turning ON when the value exceeded half of its maximum range and turning OFF otherwise. This part of the experiment introduced how analog signals can be converted into digital data, transmitted via serial communication, and visualized using Python. It also highlighted the importance of ensuring that both systems share the same baud rate to maintain stable communication.

Task 2 built upon the first task by introducing a servo motor and incorporating closed-loop control. The potentiometer readings were used to control the angular position of the servo motor, allowing the user to adjust its rotation in real time. The system was enhanced with Python's Matplotlib library to plot the potentiometer values dynamically, giving a live graphical representation of the motor's response. An LED was also included to provide visual feedback based on the servo's position. To ensure

safety and control, both the Arduino and Python codes featured keyboard interrupt functions that allowed users to stop the system easily.

Overall, this experiment demonstrated how hardware and software can be integrated to achieve real-time monitoring, visualization, and control. It provided valuable insight into how serial communication forms the foundation of many modern mechatronic and automation systems.

2.0 EXPERIMENT 1

2.1 Materials And Equipments

Materials and components that were used in the experiment:

1. Cytron CT UNO
2. Potentiometer
3. 220 Ohm resistors (1x)
4. Jumper Wires
5. Breadboard
6. LED (1x)

2.2 Experimental Setup

.

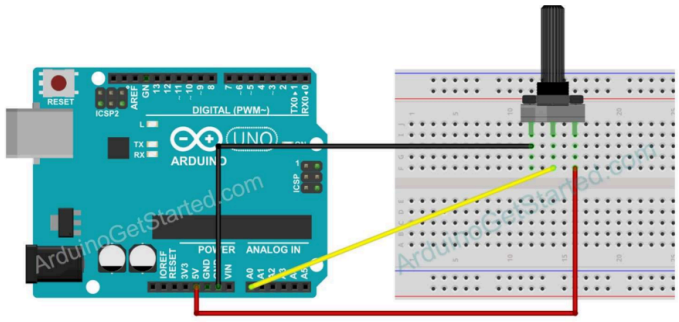


Fig. 1. Wiring Diagram

2.3 Methodology

2.3.1. Hardware Assembly

The experiment utilized an Cytron CT UNO board, a potentiometer, a solderless breadboard, and several jumper wires for interconnections. The setup was arranged based on the wiring configuration illustrated in Figure 1.

1. The potentiometer was securely placed on the breadboard to allow easy access to all its terminals.

2. One of the outer terminals of the potentiometer was linked to the 5V output pin of the Cytron CT UNO to supply voltage.

3. The opposite outer terminal was connected to the GND pin of the Cytron CT to complete the circuit.

4. The middle terminal, also known as the wiper, was connected to the A0 analog input pin on the Cytron CT to read the varying voltage values produced by the potentiometer.

5. Finally, the Cytron CT UNO was connected to a computer using a USB cable, which served both as a power source and as a communication interface between the hardware and the Python program.

2.3.2 Software And programming

2.3.2a) Arduino Sketch (Data Transmission)

An Arduino program was created to read the analog signal from the potentiometer and transmit the data through the serial port.

1. Inside the setup() function, serial communication was started using the command `Serial.begin(9600);`, setting the baud rate to 9600 for consistent data exchange with the computer.

2. In the loop() function, the Arduino performed the following sequence:

    - The analog voltage from the potentiometer connected to pin A0 was read using the command `int potValue = analogRead(A0);`. This converts the input voltage range of 0–5V into a numerical value between 0 and 1023.

    - The obtained reading was then transmitted to the serial port using `Serial.println(potValue);`, which sends the data as a string followed by a newline character.

    - A short pause of one second was introduced using `delay(1000);` to avoid excessive data transmission, making the readings easier to view and preventing the serial buffer from becoming overloaded.

3. After completing the code, the sketch was compiled and uploaded to the Arduino UNO using the Arduino IDE.

2.3.2b) Python Script (data Reception)

A Python program was developed to receive and display the data transmitted by the Arduino through the serial connection.

1. The pyserial library was imported to enable serial communication between the computer and the Arduino board.

2. A serial connection was created using the command `ser = serial.Serial('COMx', 9600);`, where `'COMx'` represents the Arduino's specific communication port (e.g., COM3 on Windows or /dev/ttyUSB0 on Linux). The baud rate was set to 9600 to match the Arduino configuration.

3. A continuous loop (while True) was used to constantly read incoming serial data from the Arduino.

4. Within a try...except block, the command `ser.readline().decode().strip()` was executed to:

    - readline(): collect the serial data until a newline character `\n` is received,

    - decode(): convert the incoming byte data into readable text, and

    - strip(): remove unnecessary spaces or line breaks from the string.

5. The clean output, labeled as pot_value, was then printed to the console for real-time monitoring.

6. To safely stop the program, a KeyboardInterrupt exception was added, allowing users to end the execution gracefully by pressing Ctrl + C.

2.3.3 Data Collection Procedure

1. Once the Arduino code had been successfully uploaded, the Serial Monitor in the Arduino IDE was closed to prevent port conflicts with Python.

2. The Python script was then executed through a terminal or a compatible IDE such as PyCharm.

3. The potentiometer's knob was rotated gradually from one end to the other and then back again to cover its entire range of motion.

4. The output values displayed in the Python console were closely observed. These values ranged from approximately 0 to 1023, directly corresponding to the physical position of the potentiometer. The smooth variation in the readings confirmed proper communication and accurate data acquisition between the Arduino and Python systems.

Code Used :

Arduino :

```
void setup() {
Serial.begin(9600);
}
void loop() {
int potValue = analogRead(A0);
Serial.println(potValue);
delay(1000);
}
```

Python :

```
import serial



ser = serial.Serial('COM5', 9600)

try:
    while True:
        pot_value = ser.readline().decode().strip()
        print("Potentiometer Value:", pot_value)

except KeyboardInterrupt:
    ser.close()
    print("Serial connection closed.")
```

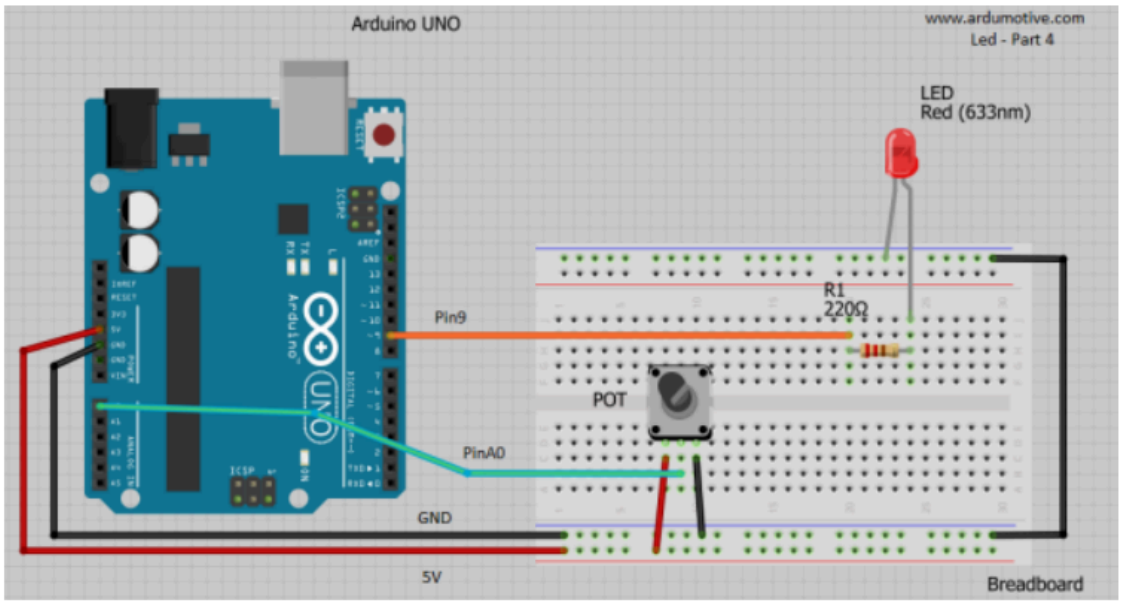
3.0 TASK 1 : Sending Potentiometer Readings From An Arduino To Python Script Through USB Connection

3.1 Materials And Equipments

Materials and components that were used in the experiment:

1. Cytron CT UNO board
2. Potentiometer
3. 220 Ohm resistors (1x)
4. Jumper Wires
5. Breadboard
6. LED (1x)

3.2 Experimental Setup



Hardware Connection Steps

1. Link one of the outer pins of the first potentiometer to the 5V output on the Arduino to provide a stable voltage supply.

2. Connect the opposite outer pin of the same potentiometer to the GND (ground) terminal of the Arduino to complete the circuit.

3. Attach the center pin (wiper) of the potentiometer to the analog input pin A0 on the Arduino board, allowing the analog voltage signal to be read.

4. Connect an LED to the digital pin 9 of the Arduino. Ensure that a suitable current-limiting resistor (such as 220 Ω) is placed in series with the LED to prevent damage.

5. Add a second potentiometer to the setup by connecting its center pin to analog input pin A1, enabling the Arduino to measure a second variable resistance input.

6. Double-check all wiring connections to ensure proper configuration:

    - Each potentiometer's first outer pin should be connected to the 5V (VCC) terminal.

    - The second outer pin of both potentiometers must be connected to GND.

    - The middle pins should be linked to A0 and A1 respectively for analog signal input.

3.3 Methodology

Implementation and Testing

The entire circuit was assembled according to the provided schematic diagram. Once the hardware connections were verified, the Arduino code was uploaded to the microcontroller using the Arduino IDE. This program was designed to continuously read the analog voltage from the potentiometer through pin A0 using the analogRead() function and transmit the readings to the Python interface via serial communication at a baud rate of 9600.

On the computer side, a Python script was developed using the pyserial library to receive and display the potentiometer data in real time. During the testing phase, the potentiometer knob was rotated gradually, and corresponding numerical changes were observed in the Python terminal. This confirmed that data transmission between the Arduino and Python was functioning properly.

Additionally, an LED was integrated into the system to visually represent the potentiometer's position. The LED automatically turned ON when the potentiometer reading exceeded the preset threshold and turned OFF when below it. This behavior verified that the control logic worked as intended. The Arduino Serial Plotter was also used to visualize the potentiometer output, showing smooth, continuous variations in the readings, which further confirmed stable analog signal acquisition and proper serial data communication.

Control Algorithm

The system's operation relied on two coordinated software components: an Arduino sketch for data acquisition and a Python script for data processing and control. The Arduino microcontroller served as the main data collection unit, while Python handled the interpretation and visualization of the received data. Communication between these two programs was established through a USB serial interface.

The control algorithm followed the sequence below:

1. The input and output pins were first initialized — the potentiometer was assigned to analog pin A0, while the LED was connected to digital pin D7. The Arduino's 5V and GND pins provided the necessary power supply connections.

2. The LED pin was configured as an OUTPUT to enable control through the program.

3. The main program loop was set to run continuously, allowing real-time response to input changes.

4. Within this loop, the Arduino repeatedly read the analog signal from A0.

5. The raw analog input, ranging from 0 to 1023, was converted into a proportional value between 0 and 255 to make it suitable for pulse-width modulation (PWM) control.

6. PWM was then applied to adjust the LED's brightness based on the potentiometer's position, providing smooth and gradual illumination changes. The PWM signal was sent to pin D7.

7. The resulting brightness value was displayed on the Serial Monitor for real-time observation, confirming that the LED responded accurately to the potentiometer input.

This control approach successfully demonstrated real-time interaction between hardware and software, illustrating how serial communication and PWM can be combined to achieve precise analog-to-digital control in a mechatronic system.

Code Used :

Arduino :

```
#include <Servo.h>



int potPin = A0;
int ledPin = 7;
int potValue = 0;
String command = "";

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // Read potentiometer and send value to Python
  potValue = analogRead(potPin);
  Serial.println(potValue);
  delay(1000);
  // Check if Python sent a command
  if (Serial.available() > 0) {
    command = Serial.readStringUntil('\n');
    command.trim();
```

```
      if (command == "ON") {
        digitalWrite(ledPin, HIGH);
      } else if (command == "OFF") {
        digitalWrite(ledPin, LOW);
      }
    }
}
```

Python :

```
>>> import serial
... import time
... ser = serial.Serial('COM5', 9600, timeout=0.1)
... time.sleep(2)
...
... try:
...     while True:
...         line = ser.readline().decode().strip()
...         if line.isdigit():
...             pot_value = int(line)
...             print("Potentiometer Value:", pot_value)
...
...             # Immediately send LED command
...             if pot_value > 512:
...                 ser.write(b"ON\n")
...             else:
...                 ser.write(b"OFF\n")
...
... except KeyboardInterrupt:
...     print("\nProgram interrupted by user.")
...
... finally:
...     ser.close()
...     print("Serial connection closed.")
```

3.4 Result

The successful execution of the experiment demonstrates that real-time data transmission from the Arduino to Python is achievable. The plotted graph accurately represents potentiometer adjustments, validating the effectiveness of the serial communication process.

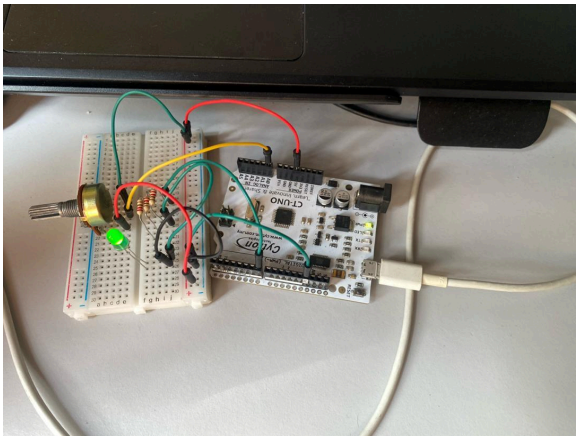


Figure showing light is ON

Figure showing The Potentiometer Value above 500 and LED Status is ON

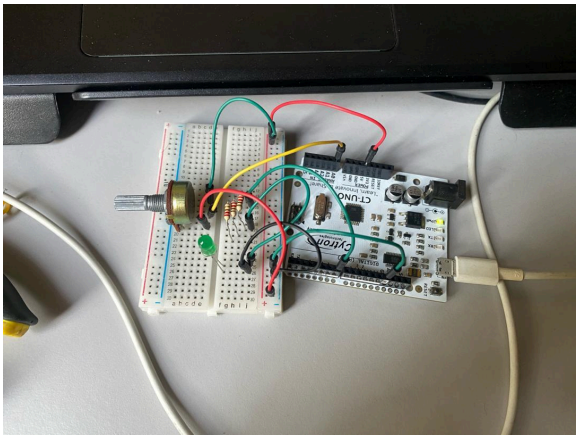


Figure showing light is OFF

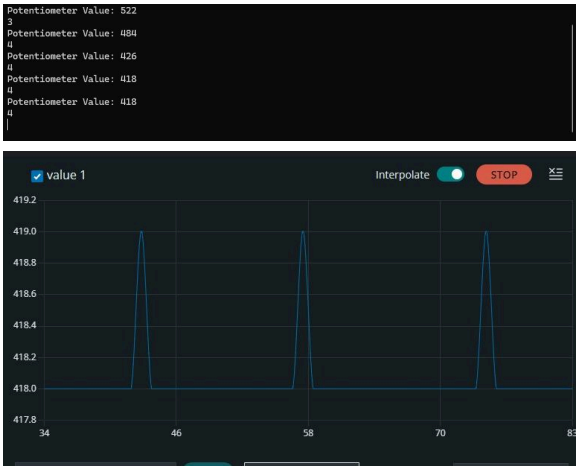


Figure showing The Potentiometer Value below 500 and LED Status is OFF

Question:

To present potentiometer readings graphically in your Python script, you may enhance your code by introducing the capability to generate and showcase a graph. This graphical/visualization can deliver a more intuitive and informative perspective for data

interpretation. Be sure to showcase the steps involved in your work (Hint: use matplotlib in your Python script).

To present potentiometer readings graphically using Python, we can use the matplotlib library to plot real-time data from the Arduino. Below is a step-by-step guide along with the required

Python script:

1. Install Required Libraries: pip install matplotlib pyserial

2. Connect your Arduino to your computer. Make sure your Arduino is connected to the correct COM port. Your Arduino code should continuously send potentiometer values viaSerial.println().

3. Python Script to Plot Potentiometer Readings in Real-Time

A real-time graph will appear, displaying the potentiometer values as they change. The graph updates every 100ms, showing a smooth transition of values.

4.0 EXPERIMENT 2

4.1 Materials And Equipments

1. Cytron CT Uno
2. Servo motor
3. Jumper wires
4. Potentiometer (for manual angle input)
5. USB cable for Arduino
6. Computer with Arduino IDE and Python installed

4.2 Experimental Setup

The circuit was assembled on a breadboard according to the designed connection layout. All components, including the Arduino board, potentiometer, resistors and servomotor were carefully connected to ensure proper functionality.

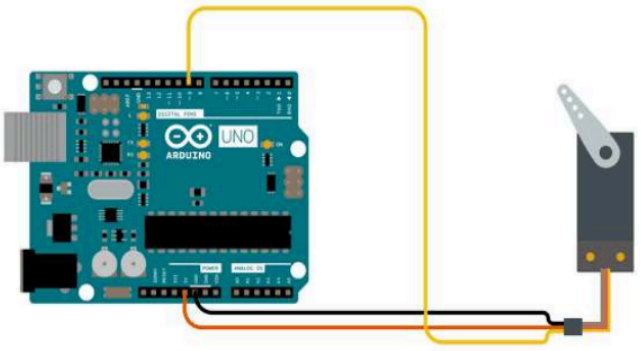


Fig. 2

1. **Connect the Servo's Signal Wire:** Attach the servo's signal wire (usually yellow or orange) to one of the PWM-capable pins on the Arduino, such as digital pin 9.

2. **Connect the Power Wire:** Link the servo's power wire (typically red) to the 5V output pin on the Arduino. Servos usually operate with a supply voltage of around +5V.

3. **Connect the Ground Wire:** Connect the servo's ground wire (usually brown or black) to one of the Arduino's GND pins.

Arduino Setup

1. The Arduino sketch was programmed to read potentiometer values using the `analogRead()` function and to control the servo motor's position through the **Servo** library.

2. The potentiometer readings were mapped to servo angles ranging from **0° to 180°** using the `map()` function.

3. The program also transmitted the servo position data to Python via **serial communication** for visualization purposes.

Python Setup

1. A Python script was developed using the **PySerial** and **Matplotlib** libraries.

2. The script received serial data from the Arduino, displayed the potentiometer and servo position values in real time, and plotted them on an interactive graph.

3. The serial port and baud rate were configured to match those used in the Arduino program, typically **9600 bps**.

Operation

1. When the Python script was executed, it established a serial connection with the Arduino.

2. As the potentiometer knob was rotated, the servo motor's position adjusted accordingly.

3. An LED provided visual feedback based on the potentiometer or servo angle.

4. The live graph in Python displayed real-time potentiometer data, confirming successful two-way communication and data visualization between the hardware and software components.

4.3 Methodology

4.3.1 Circuit Assembly

1. Connect the Potentiometer to Analog Pin A0 to read its variable resistance:
    - One terminal of the potentiometer to VCC (5V)
    - The other terminal to GND
    - The middle terminal (wiper) to A0

2. Connect the Servo Motor to Digital Pin 9:
    - Signal wire to pin 9
    - Power wire to 5V
    - Ground wire to GND

4.3.2 Programming Logic

1. Attach the servo motor to pin 9 using *myServo.attach(servoPin)*.
2. Initialize serial communication at a baud rate of 9600 to monitor data in real time.
3. Read the potentiometer input:
    - Continuously read the analog value from pin A0.
    - Convert the analog value (0–1023) to a corresponding servo angle (0–180°) using the *map()* function.
4. Control the servo motor:
    - Rotate the servo to the mapped angle using *myServo.write(angle)*.
    - Display both the potentiometer value and the servo angle on the Serial Monitor for live tracking.
5. Add delay for stability:
    - Include a short delay (around 10 ms) to ensure smooth and steady servo movement without abrupt changes.

4.3.3 Code Used

Arduino Code

```
#include <Servo.h>


Servo myservo;  // Create a servo object
int angle = 0;  // Variable to store incoming angle


void setup() {
  Serial.begin(9600);   // Start serial communication
  myservo.attach(12);   // Attach servo to pin 12
  Serial.println("Ready to receive angles (0-180)");
}


void loop() {
  // Check if data is available from Python
  if (Serial.available() > 0) {
    angle = Serial.parseInt();  // Read integer from serial
    if (angle >= 0 && angle <= 180) {
      myservo.write(angle);     // Move servo to that angle
      Serial.print("Moved to: ");
      Serial.println(angle);
    }
  }
}
```

Python Code

```
>>> import serial
... ser = serial.Serial('COM5', 9600)
...
... try:
...     while True:
...         angle = input("Enter servo angle (0-180 deg, or 'q' to quit): ")
...
...         if angle.lower() == 'q':
...             break
...
...         try:
...             angle = int(angle)
...             if 0 <= angle <= 180:
...
...                 ser.write(str(angle).encode())
...                 print(f"Sent angle: {angle}")
...             else:
...                 print("Angle must be between 0 and 180 degrees.")
...         except ValueError:
...             print("Please enter a valid number.")
...
... except KeyboardInterrupt:
...     pass ...
```

```
... finally:
...     ser.close()
...     print("Serial connection closed.")
```

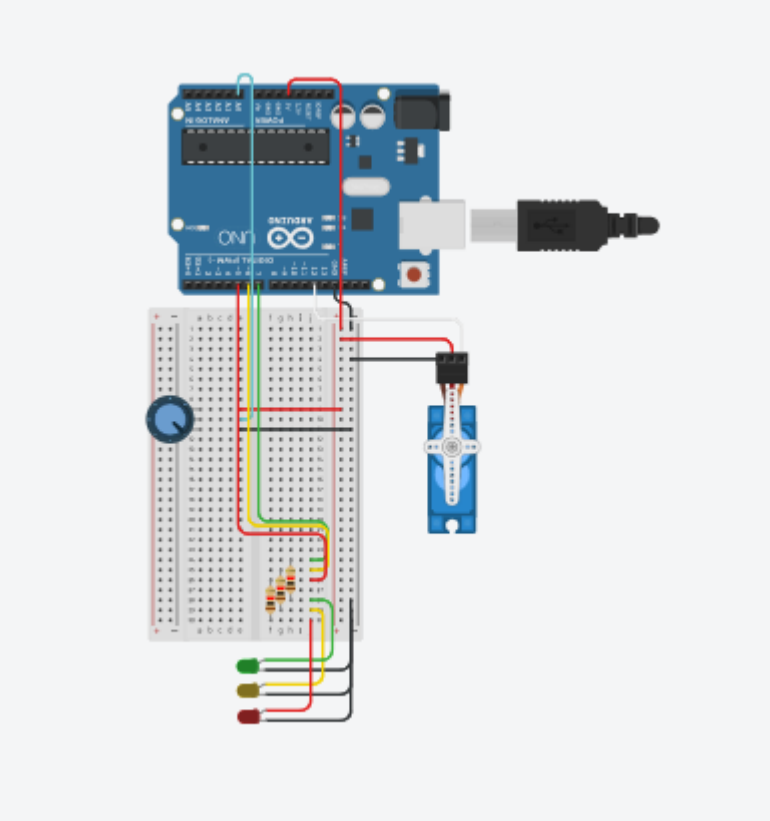
5.0 TASK 2 : Transmitting Angle Data From Python Script To An Arduino Which Then Actuates A Servo To Move To The Specified Angle

5.1 Materials And Equipment

1. Arduino board (e.g., Arduino Uno)
2. Servo motor
3. Jumper wires
4. Potentiometer (for manual angle input)
5. USB cable for Arduino
6. Computer with Arduino IDE and Python installed

5.2 Experimental Setup

The circuit was assembled on a breadboard according to the designed connection layout. All components, including the Arduino board, potentiometer, resistors and servomotor were carefully connected to ensure proper functionality.



1. **Connect the Servo's Signal Wire:** Attach the servo's signal wire (usually yellow or orange) to one of the PWM-capable pins on the Arduino, such as digital pin 12.

2. **Connect the Power Wire:** Link the servo's power wire (typically red) to the 5V output pin on the Arduino. Servos usually operate with a supply voltage of around +5V.

3. **Connect the Ground Wire:** Connect the servo's ground wire (usually brown or black) to one of the Arduino's GND pins.

4. **Connect the Potentiometer's Signal Wire:** Attach the potentiometer's middle pin (wiper) to the Arduino's analog input pin A0.

5. **Connect the Power Pin:**Connect one of the potentiometer's outer pins to the 5V output on the Arduino. This provides the reference voltage for the potentiometer.

6. **Connect the Ground Pin:**Connect the other outer pin of the potentiometer to one of the Arduino's GND pins to complete the circuit.

7. **Connect All Led:**Connect green,yellow, and red with resistor to 7, 6, and 5 digital pins accordingly.

Arduino Setup

1. The Arduino reads input from a potentiometer connected to A0 and maps the values (0–1023) to servo angles between 0° and 180°, controlling a servo motor on pin 12.

2. Three LEDs connected to pins 5, 6, and 7 indicate the servo's angle range: green for low angles, yellow for middle angles, and red for high angles.

3. The Arduino communicates with Python at 9600 bps, switching between potentiometer mode and manual mode. In manual mode, it receives servo angle values (0–180) from Python and moves the servo accordingly.

Python Setup

1. The Python script uses PySerial to connect to the Arduino through the serial port (e.g., `COM5`) at 9600 bps.

2. The user can type "pot" to use the potentiometer, "manual" to set servo angles manually, or enter values between 0–180 to move the servo in manual mode.

3. The script displays feedback messages from the Arduino and allows the user to exit safely by typing "q", confirming smooth two-way communication between the devices.

Operation

1. When the Python script starts, it connects to the Arduino through the serial port.

2. The user can choose between potentiometer mode and manual mode by typing the corresponding command in Python.

3. In potentiometer mode, rotating the potentiometer changes the servo position automatically, and the LEDs show the current angle range.

4. In manual mode, the user can type an angle value between 0° and 180°, and the servo moves to that position.

5. The Arduino sends feedback messages to Python, confirming mode changes and servo movements, ensuring smooth two-way communication between the hardware and software.

5.3 Methodology

Circuit Assembly

1. Connect the Potentiometer to Analog Pin A0
    a. One outer terminal of the potentiometer is connected to 5V (VCC).

    b. The other outer terminal is connected to GND.

    c. The middle terminal (wiper) is connected to A0 to provide variable analog input to the Arduino.
2. Connect the Servo Motor to Digital Pin 12:
    a. The signal wire (yellow or orange) is connected to pin 12.

    b. The power wire (red) is connected to 5V.

    c. The ground wire (brown or black) is connected to GND.

3. Connect the LEDs:
    a. The green LED is connected to pin 7, the yellow LED to pin 6, and the red LED to pin 5.

    b. The negative legs (cathodes) of all LEDs are connected to GND through current-limiting resistors (220 Ω).

Programming Logic

1. Attach the servo motor to pin 12 using `myservo.attach(12)`.

2. Initialize serial communication at 9600 bps using `Serial.begin(9600)` to exchange data with Python.

3. Define two operating modes:
    a. Potentiometer Mode:
    Continuously read the analog input from A0 using `analogRead()`.
    Map the potentiometer value (0–1023) to a servo angle (0–180°) using `map()`.
    Move the servo to the mapped position with `myservo.write(angle)`.
    Control LEDs to indicate the current angle range:
        i. Green (< 60°), Yellow (60–120°), Red (> 120°).

b. Manual Mode**:**
Receive angle values (0–180) from the Python program through serial input.
Move the servo to the received position and send confirmation messages back to Python.

4. Use `Serial.println()` to send feedback messages (e.g., mode changes or current angle) for real-time monitoring.

5. Include a short delay (`delay(100)`) for smooth servo operation and stable LED switching.

Code Used

Arduino Code

```
#include <Servo.h>

Servo myservo;
int potPin = A0;
int gledPin = 7;
int yledPin = 6;
int rledPin = 5;
int potValue = 0;
int angle = 0;
bool manualMode = false;  // start in potentiometer mode

void dark (){
digitalWrite(gledPin, LOW);
digitalWrite(yledPin, LOW);
digitalWrite(rledPin, LOW);

}


void setup() {
  myservo.attach(12);
  pinMode(gledPin, OUTPUT);
  pinMode(yledPin, OUTPUT);
  pinMode(rledPin, OUTPUT);
  Serial.begin(9600);
  Serial.println("Arduino ready.");
}

void loop() {
  if (Serial.available() > 0) {
    String data = Serial.readStringUntil('\n');
    data.trim();

    if (data.equalsIgnoreCase("pot")) {
      manualMode = false;
      Serial.println("Switched to POT mode");
    }
    else if (data.equalsIgnoreCase("manual")) {
      manualMode = true;
```

```
      Serial.println("Switched to MANUAL mode");
    }
    else if (manualMode && data.toInt() >= 0 && data.toInt() <= 180) {
      angle = data.toInt();
      myservo.write(angle);
      Serial.print("Manual angle set to: ");
      Serial.println(angle);
    }
  }

  //using potentiometer mode
  if (!manualMode) {
    potValue = analogRead(potPin);
    angle = map(potValue, 0, 1023, 0, 180);
    myservo.write(angle);

    if (angle < 60){
      dark ();
      digitalWrite(gledPin, HIGH);
      }
    else if (angle > 60 && angle < 120){
      dark ();
      digitalWrite(yledPin, HIGH);
      }
    else if (angle > 120){
      dark ();
      digitalWrite(rledPin, HIGH);
    }

    Serial.println(angle);
    delay(100);
  }
}
```

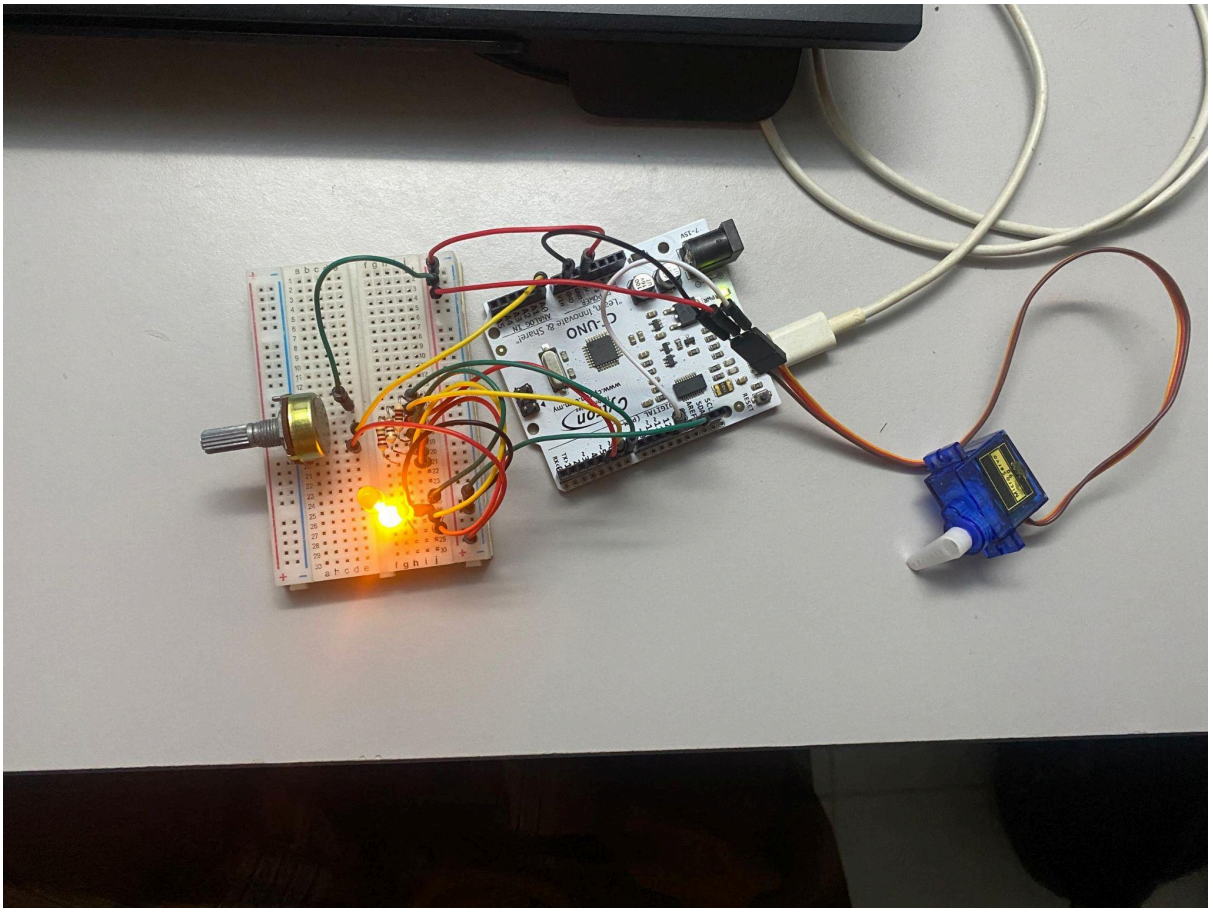
Python Code

```
>>> import serial
... import time
...
... ser = serial.Serial('COM5', 9600, timeout=1)
... time.sleep(2)
... print("Connected to Arduino.\n")
```

```
... print("Commands:")
... print("  manual  → manually input the angle, RANGE (0 TO 180 ONLY)")
... print("  pot     → use potentiometer ")
... print("  q       → exit\n")
...
... try:
...     while True:
...         cmd = input("Enter command or angle: ")
...         if cmd.lower() == 'q':
...             break
...         ser.write((cmd + '\n').encode())  # send with newline
...         time.sleep(0.1)
...
...         if ser.in_waiting > 0:
...             print("Arduino:", ser.readline().decode().strip())
...
... except KeyboardInterrupt:
...     pass
... finally:
...     ser.close()
...     print("Serial connection closed.")
...
```

5.4 Result

Task 2 involved enhancing the Arduino-Python communication to achieve real-time servo motor control using a potentiometer as the input sensor, incorporating an LED feedback mechanism.



6.0 DISCUSSION

This experiment clearly demonstrated the relationship between the potentiometer input and the corresponding outputs of both the LED and the servo motor. Adjusting the potentiometer resulted in proportional changes to the LED brightness and servo motor position.

However, several minor issues were observed during the experiment, such as slight flickering of the LED and small delays in the servo motor's response. These irregularities were likely caused by electrical noise, which produced unstable readings from the potentiometer and led to inconsistent behavior in both outputs. Additionally, the system exhibited a somewhat non-linear response, where variations in potentiometer rotation did not always translate proportionally to changes in LED brightness or servo position. The limited input range of the potentiometer also restricted the ability to generate the full spectrum of analog values.

Despite these limitations, the experiment successfully demonstrated the principle of real-time control using PWM signals and an analog input device. It provided valuable insight into how hardware and software can be integrated to achieve responsive control in mechatronic systems.

7.0 CONCLUSION

This experiment successfully demonstrated two main applications of serial communication :transmitting potentiometer data and controlling external devices such as an LED and a servo motor using Python. The results confirmed that serial communication provides an efficient way to achieve real-time data exchange between an Arduino and a computer via USB. In this setup, the Arduino effectively read analog input from the potentiometer and transmitted the data to Python for processing and visualization.

Both Task 1 and Task 2 showed how the potentiometer could be used to control LED brightness and servo motor position by varying its resistance. These findings highlight that serial communication not only enables reliable data transfer but also allows for the execution of control commands on physical hardware components.

Overall, this experiment underscores the importance of serial communication in modern engineering applications, particularly in industrial automation, IoT based smart systems, and real time monitoring. Developing proficiency in serial data handling provides a strong foundation for designing advanced control systems where sensors, actuators and microcontrollers work together seamlessly.

8.0 RECOMMENDATION

This experiment could be further enhanced by improving the real-time graphical visualization of the potentiometer readings and the corresponding servo motor movements. A more dynamic and interactive visualization system would allow users to better observe the relationship between input changes and servo responses. This enhancement would also make system monitoring more efficient by reducing the need for manual hardware adjustments and providing a clearer understanding of how analog input values influence actuator behavior. By incorporating computer-based control through graphical interfaces, users could fine-tune parameters and control actions more intuitively, improving the overall experimental experience.

Furthermore, data accuracy and system responsiveness could be improved by applying a moving average filter to the potentiometer input. Raw analog readings from the potentiometer tend to fluctuate due to electrical noise or mechanical imperfections, leading to unstable or jittery responses in the servo motor. Implementing a filtering technique would smooth out these fluctuations before the data is transmitted or displayed, resulting in more stable movement, cleaner graphical output, and more reliable performance. Overall, these enhancements would significantly improve both the precision and usability of the system, paving the way for more advanced real-time control and visualization applications.

9.0 REFERENCES

1. “02 – Digital Logic System ver2.pdf”, Google Drive file, accessed via https://drive.google.com/file/d/1JGdUitPFZEFM8F9qtV-U-ZJ1lsq5YzQYE/view?usp=drive_link

2. Tutorials Point. (n.d.). *Arduino – Blinking LED*. Retrieved from https://www.tutorialspoint.com/arduino/arduino_blinking_led.htm TutorialsPoint

3. A. Rohansen-Roy. (2020, September 27). *Blinking LED* [Tutorial]. Arduino Project Hub. Retrieved from

https://projecthub.arduino.cc/arohansenroy/blinking-led-77a79f

10.0 APPENDICES

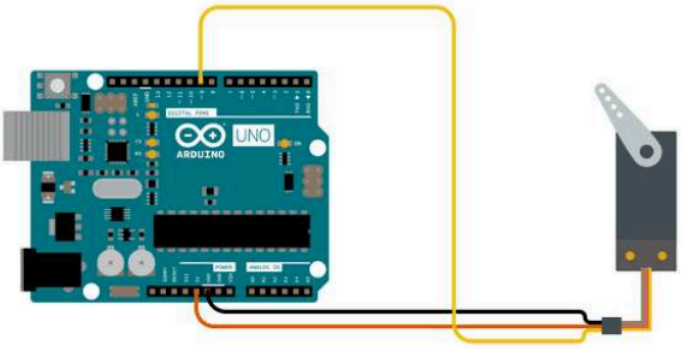


Figure 1

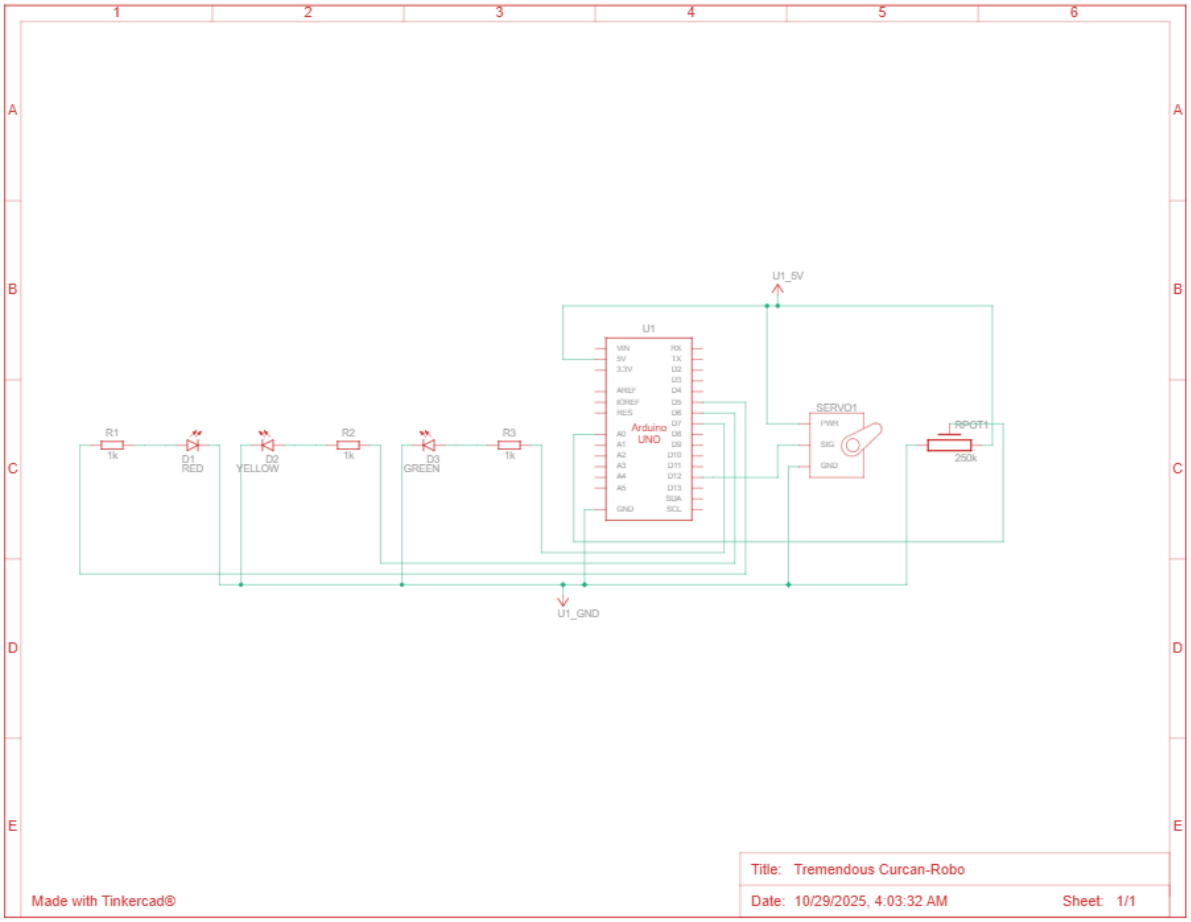


Figure 2

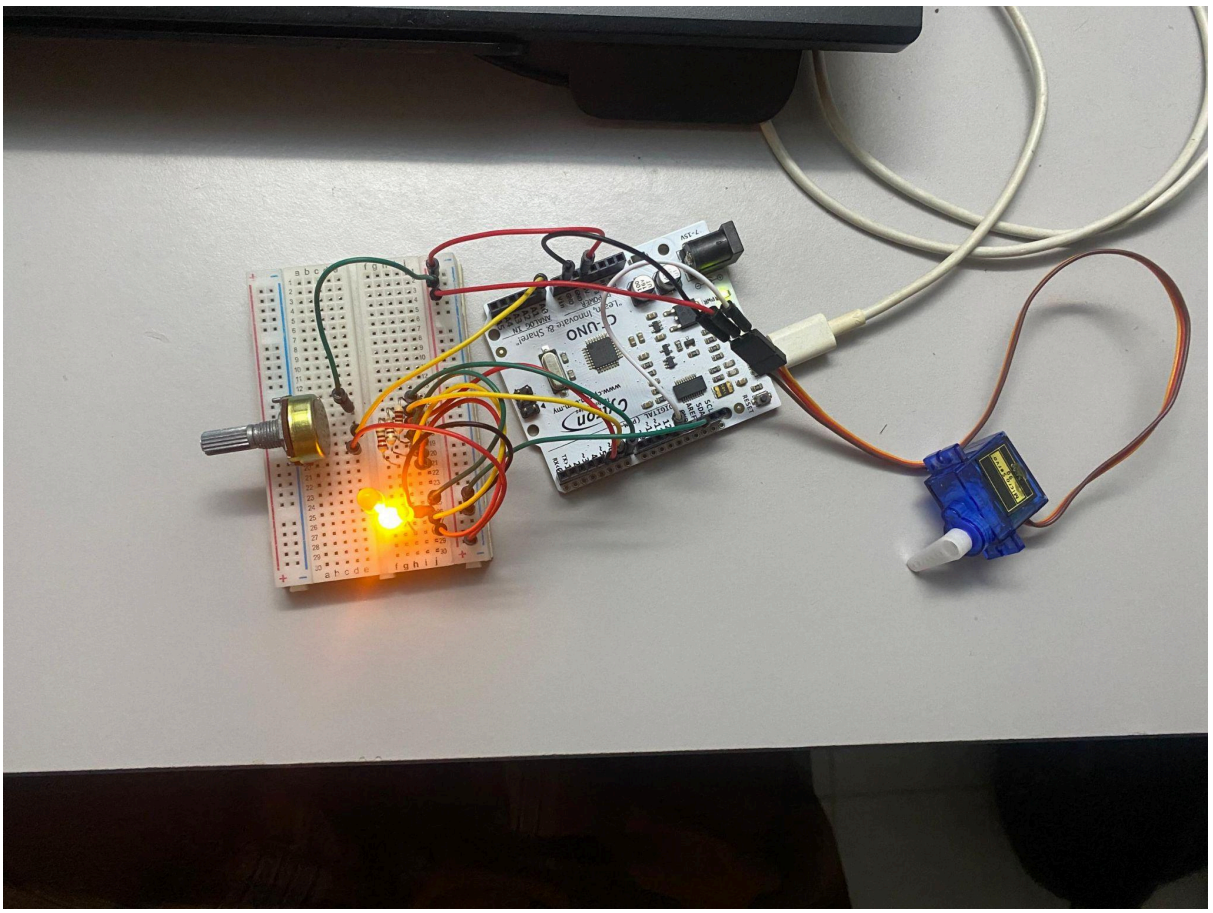


Figure 3

11.0 ACKNOWLEDGEMENT

We would like to sincerely thank the instructors, Assoc. Prof. Eur. Ing. Ir. Ts. Gs. Inv. Dr. Zulkifli Bin Zainal Abidin 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.

12.0 STUDENT'S 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: ***ahmadfarhan***

Name: Ahmad Farhan bin Ahmad Nahrudi
Matric Card: 2317373

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

Signature: ***aimansafawi***

Name: Muhammad Aiman bin Safawi
Matric Card: 2318249

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

Signature: ***nafizzuhairee***

Name: Mohd Nafiz Zuhairee bin Mohd Neezar
Matric Card: 2317423

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