

MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1 2025/2026

WEEK 3: SERIAL COMMUNICATION

DATE OF EXPERIMENT: 22 OCTOBER 2025

DATE OF SUBMISSION: 29 OCTOBER 2025

SECTION 1

GROUP 13

LECTURER: ZULKIFLI BIN ZAINAL ABIDIN

NO.	NAME	MATRIC
1	HARIZ IRFAN BIN MOHD ROZHAN	2318583
2	ABDULLAH HASAN BIN SIDEK	2318817
3	TAN YONG JIA	2319155
4	NUR QISTINA BINTI MOHD FAIZAL	2319512

**ABSTRACT**

In order to accomplish real-time data interchange and device control, this project studies the use of serial communication between an Arduino microcontroller and Python software. The Arduino is linked to a potentiometer in the first section so that analogue readings can be sent to a Python interface through a USB serial connection. An LED is controlled by the transmitted data and activates when the potentiometer value surpasses half of its maximum range. The experiment expands on this configuration in the second section by adding a servo motor that is managed by the potentiometer using the same communication channel. An LED gives input dependent on the location of the servo, and Python receives and uses Matplotlib to display the potentiometer and servo data in real time. Together, both tasks demonstrate the fundamentals of serial communication, sensor-based control, and real-time visualization, highlighting the integration of hardware and software for mechatronic applications.

## TABLE OF CONTENTS

ABSTRACT .....	1
TABLE OF CONTENTS .....	2
1.0 INTRODUCTION .....	3
TASK 1: .....	3
2.0.1 MATERIALS AND EQUIPMENTS.....	3
3.0.1 EXPERIMENT SETUP .....	4
4.0.1 METHODOLOGY .....	5
5.0.1 RESULT .....	9
TASK 2: .....	10
2.0.2 MATERIALS AND EQUIPMENTS.....	10
3.0.2 EXPERIMENTAL SETUP .....	10
4.0.2 METHODOLOGY .....	11
5.0.2 RESULT .....	17
6.0 DISCUSSION.....	18
7.0 CONCLUSION.....	19
8.0 RECOMMENDATIONS .....	20
10.0 REFERENCES.....	20
9.0 APPENDICES .....	21
10.0 ACKNOWLEDGEMENTS .....	22
11.0 STUDENTS DECLARATION .....	23

## **1.0 INTRODUCTION**

One of the most important techniques for data transfer between computers and embedded systems is serial communication, which allows for effective two-way communication with little wiring. Python serves as the monitoring and processing interface in this project, while the Arduino microcontroller serves as both a data source and a control unit. In the first assignment, the Arduino uses the serial port to communicate data to Python after reading analogue input from a potentiometer. Based on the location of the potentiometer, the Python program shows these measurements and automatically turns on an LED. Sensor-based decision-making and serial data transfer are introduced in this straightforward arrangement.

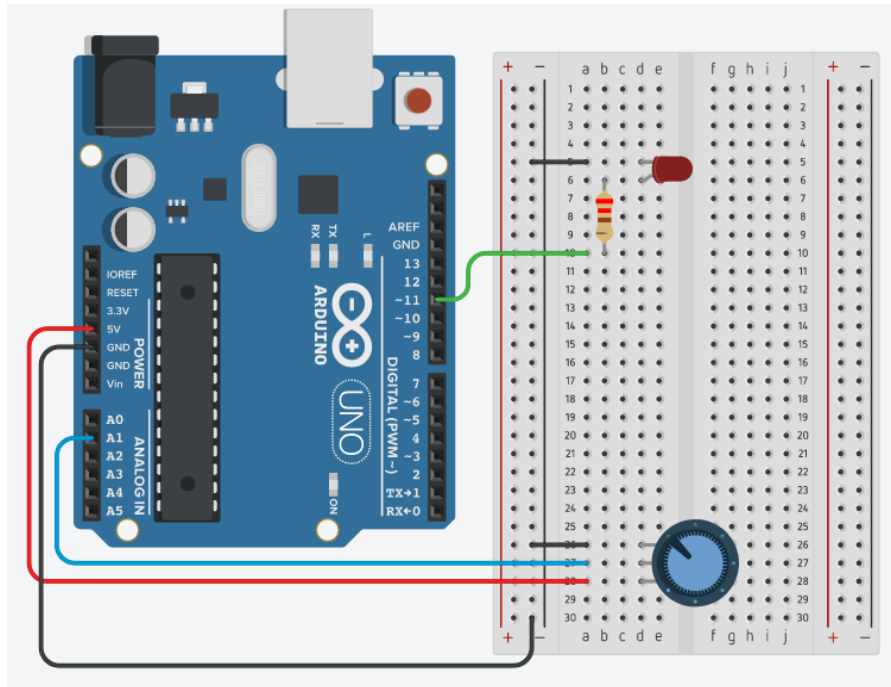
By adding a servo motor that reacts to potentiometer adjustments in real time, the second task develops on the previous one. The location data from the servo is passed back to Python so that Matplotlib may display it and show the movements of the system in real time. Additionally, a feedback system that improves user comprehension of the process is created by incorporating an LED indicator to reflect servo or sensor conditions. When combined, these activities offer practical expertise in real-time data collection, actuator control, hardware-software interface, and visualisation, which are all essential elements in modern automation and mechatronic systems.

## **TASK 1:**

### **2.0.1 MATERIALS AND EQUIPMENTS**

- Arduino board
- Potentiometer
- LED
- 220  $\Omega$  resistor
- Jumper wires
- Breadboard

### 3.0.1 EXPERIMENT SETUP



- Connect one leg of the potentiometer to 5V on the Arduino.
- Connect the other leg of the potentiometer to GND on the Arduino.
- Connect the middle leg (wiper) of the potentiometer to an analog input pin on the Arduino, such as A1.

### 4.0.1 METHODOLOGY

1. Connect the LED to pin 11 of the microcontroller.
2. Connect the potentiometer to analog pin A1 to read its variable resistance.
3. Ensure proper wiring:
  - One end of the potentiometer to VCC (5V)
  - The other end to GND
  - The middle terminal to A1

## Programming Logic

1. Begin serial communication at 9600 baud rate to monitor values.
2. Set pin 11 as the output for the LED.
3. Reading potentiometer input:
  - Continuously read the analog value from pin A1.
  - Convert the 0-1023 analog reading to a 0-255 range.
4. Adjust the LED brightness:
  - LED brightness can be adjusted using Pulse Width Modulation (PWM) that corresponds with the translated potentiometer value.
  - Pin 9 should get the PWM signal output.
5. Displaying data:
  - Print the brightness value to the Serial Monitor for real-time monitoring.
6. Adding delay for smoothness:
  - Introduce a 100ms delay to prevent rapid fluctuations and ensure smooth LED brightness transitions.

## Code used:

### Arduino Code:

```
void setup() {  
  
  Serial.begin(9600);  
  
  
  // Set the LED pin (12) as an output  
  
  pinMode(11, OUTPUT);  
  
}
```

```

void loop() {

    // Read the analog value from the potentiometer connected to pin A0

    int potValue = analogRead(A0);


    // Print the value to the Serial Monitor

    Serial.println(potValue);


    delay(100); // Wait for 100 milliseconds


    // Check if the potentiometer reading is above the threshold (500)
    if (potValue > 500) // The 'if' condition must be in parentheses ( )
    {
        // Turn the LED ON

        digitalWrite(11, HIGH);
    }
    else
    {
        // Turn the LED OFF

        digitalWrite(11, LOW);
    }
}

```

## Python Code:

```
PythonProject1 Version control
Current File
lab 3.py x
1
2 import serial
3 import time
4 import matplotlib
5 matplotlib.use('TkAgg') # or 'Qt5Agg' if you prefer
6 import matplotlib.pyplot as plt
7
8 import matplotlib.pyplot as plt
9
10 # --- Setup Serial Connection ---
11 ser = serial.Serial(port='COM3', baudrate=9600) # change COM3 to your port
12 time.sleep(2) # allow time for Arduino to reset
13
14 # --- Initialize plot ---
15 plt.ion() # turn on interactive mode
16 fig, ax = plt.subplots()
17 y_data = []
18 x_data = []
19 line, = ax.plot(x_data, y_data, 'b-', label="Potentiometer Value")
20
21 ax.set_xlabel("Samples")
22 ax.set_ylabel("Value")
23 ax.set_title("Live Potentiometer Data")
24 ax.legend()
25
26 # --- Read and Plot Data ---
27 sample = 0
28 try:
29     while True:
30         pot_value = ser.readline().decode().strip()
31
32 PythonProject1 > lab 3.py 1:1 CRLF UTF-8 4 spaces Python 3.8 (PythonProject1)
```

```
PythonProject1 Version control
Current File
lab 3.py x
22 ax.set_ylabel("Value")
23 ax.set_title("Live Potentiometer Data")
24 ax.legend()
25
26 # --- Read and Plot Data ---
27 sample = 0
28 try:
29     while True:
30         pot_value = ser.readline().decode().strip()
31         if pot_value.isdigit(): # ensure it's numeric
32             pot_value = int(pot_value)
33             y_data.append(pot_value)
34             x_data.append(sample)
35             sample += 1
36
37             line.set_xdata(x_data)
38             line.set_ydata(y_data)
39             ax.relim()
40             ax.autoscale_view()
41             plt.pause(0.01) # update plot
42 except KeyboardInterrupt:
43     print("Closing connection...")
44     ser.close()
45     plt.ioff()
46     plt.show()
47
PythonProject1 > lab 3.py 1:1 CRLF UTF-8 4 spaces Python 3.8 (PythonProject1)
```



## Control Algorithm

1. Start initialization:
  - Define pinMode as 11 (PWM output for LED).
  - Define potValue as A0 (analog value from the potentiometer is connected to pin A0).
  - Initialize serial communication at 9600 baud for monitoring.
2. Continuous loop execution:
  - Read the analog value from the potentiometer connected to pin A0 using `analogRead(A0)`.
  - Show the potentiometer reading on the Serial Monitor so that you can see it.
  - Evaluate the reading against the threshold value (500):
    - When the reading exceeds 500, the LED activates (`digitalWrite(11, HIGH);`). Otherwise, the LED switches OFF (`digitalWrite(11, LOW);`).
  - Implement a 100 ms pause between each reading for reliable performance.
3. Repeat Indefinitely:
  - The loop constantly modifies the LED brightness according to the live potentiometer input.

### 5.0.1 RESULT

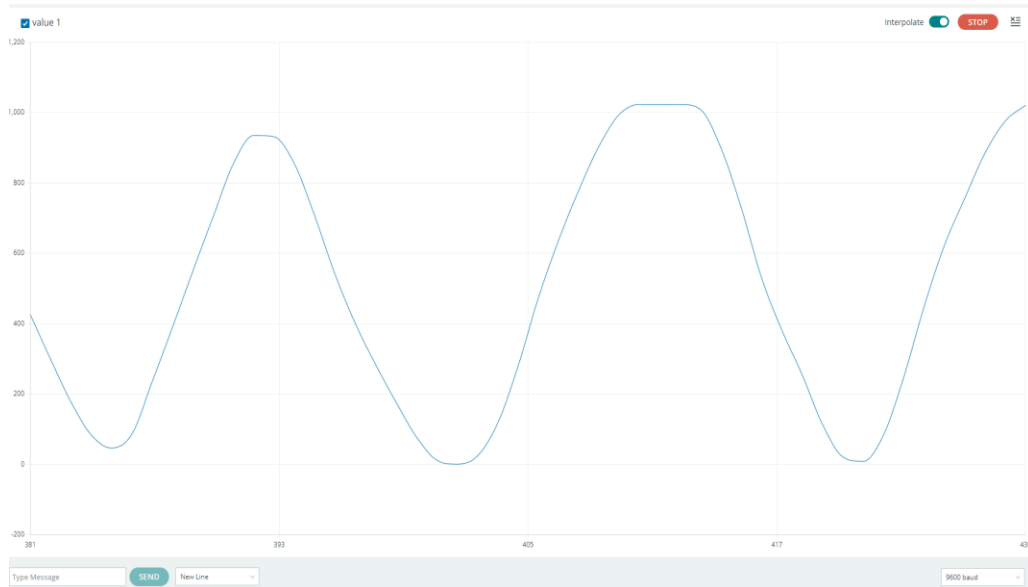
The experiment was carried out successfully to assess the LED's reaction to the signals from a potentiometer. Once the Arduino program was uploaded and run, the Serial Monitor showed ongoing readings from the potentiometer, varying from around 0 to 1023 as the knob of the potentiometer was turned.

When the readings surpassed the threshold value of 500, the LED linked to pin 11 illuminated. When the measurements fell under 500, the LED switched OFF. This showed the program's capability to evaluate the analog input value and manage a digital output accordingly.

The LED reacted instantly and aligned with the position of the potentiometer. The 100 ms delay guaranteed that the adjustments were seamless and steady, avoiding any flickering. The

serial output confirmed that the Arduino correctly interpreted and handled the potentiometer's changing resistance.

In general, the system operated as expected; the potentiometer successfully managed the LED state, verifying that the circuit and code logic were functioning properly.

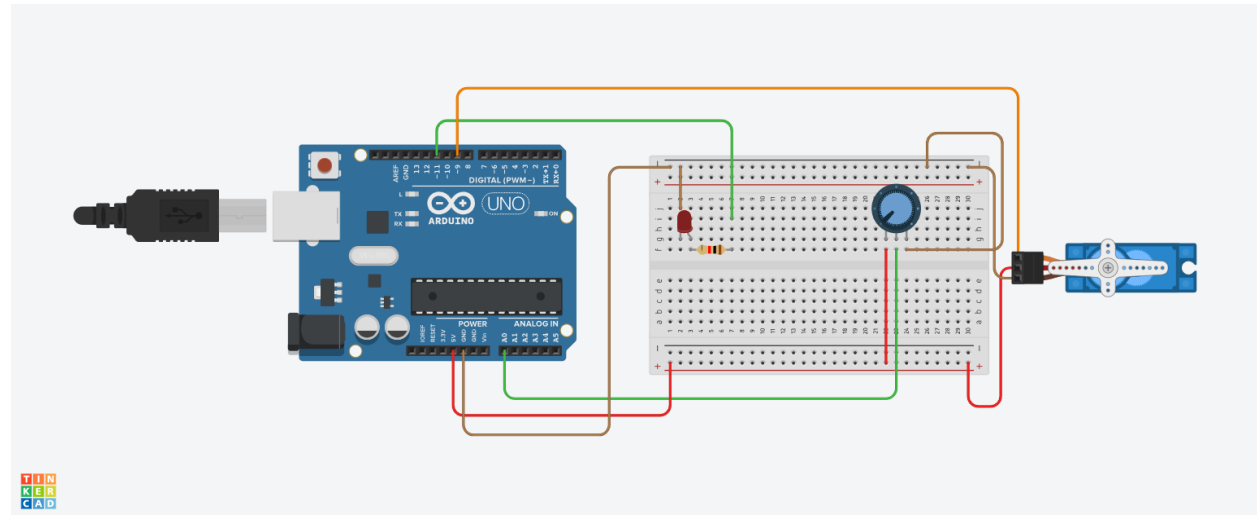


## TASK 2:

### 2.0.2 MATERIALS AND EQUIPMENTS

- Arduino board
- Potentiometer
- LED
- 220  $\Omega$  resistor
- Jumper wires
- Breadboard
- Servo motor

### 3.0.2 EXPERIMENTAL SETUP



- Connect the **signal wire** of the servo motor to **digital pin 9** on the Arduino.
- Connect the **VCC wire** of the servo motor to the **5V** pin on the Arduino.
- Connect the **GND wire** of the servo motor to the **GND** pin on the Arduino.
- Connect one leg of the potentiometer to the **5V** pin on the Arduino.
- Connect the other leg of the potentiometer to the **GND** pin on the Arduino.
- Connect the middle leg (wiper) of the potentiometer to an **analog input pin**, such as **A0** on the Arduino.

### 4.0.2 METHODOLOGY

1. Connect the **servo motor signal wire** to **digital pin 9** of the Arduino.
2. Connect the **LED anode (long leg)** to **digital pin 8** of the Arduino, and the **cathode (short leg)** to a **220  $\Omega$  resistor**, then connect the other end of the resistor to **GND**.
3. Connect the **potentiometer** to **analog pin A0** to read its variable resistance.
  - a. One end of the potentiometer to **VCC (5V)**
  - b. The other end to **GND**

- c. The middle terminal (wiper) to **A0**
- 4. Connect the **servo motor power (VCC)** to **5V** and **GND** to the **Arduino GND**.

## Programming Logic

1. Initialize the system:
  - Begin serial communication at **9600 baud rates** to monitor potentiometer readings.
  - Set **pin 11** as an **output** for the LED.
  - Attach the **servo motor** to **digital pin 9** for angle control.
2. Read potentiometer input:
  - Continuously read the **analog value** from **pin A0**, which ranges from **0 to 1023**.
  - Send this value to the **Serial Monitor** for real-time observation and debugging.
3. Map potentiometer readings to servo angle:
  - Use the `map()` function to scale the potentiometer input (0–1023) into a **servo angle** range of 0–180°.
4. Control the servo motor:
  - Write the **mapped angle** to the servo using `myServo.write()`.
  - The servo motor rotates proportionally to the potentiometer's position.
5. LED control logic:
  - If the potentiometer reading is greater than 500, turn the LED ON (digital HIGH).
  - Otherwise, turn the LED OFF (digital LOW).
  - This provides visual feedback of the potentiometer or servo state.
6. Add delay for stability:
  - Include a 100 ms delay in each loop to reduce servo jitter and prevent excessive data transmission.

#### 7. Python real-time plotting:

- Establish a serial connection with the Arduino at **9600 baud** using the **pyserial** library.
- Continuously read potentiometer data transmitted from Arduino.
- Use **Matplotlib** in interactive mode (plt.ion()) to display a **real-time plot** of the potentiometer values.
- Update the plot dynamically to visualize how the potentiometer, servo angle, and LED state change over time.

#### Code Used:

##### Arduino Code:

```
#include <Servo.h> // Include the Servo library to control the motor

// Define the pins

const int POT_PIN = A0; // Potentiometer connected to Analog Pin A0

const int LED_PIN = 11; // LED connected to Digital Pin 11

const int SERVO_PIN = 9; // Servo signal wire connected to Digital Pin 9 (often a PWM pin)

// Create a Servo object

Servo myServo;

void setup() {

  Serial.begin(9600);

  // Set the LED pin as an output
```

```

pinMode(LED_PIN, OUTPUT);

// Attach the servo object to the specified pin
myServo.attach(SERVO_PIN);
}

void loop() {
    // 1. Read the analog value from the potentiometer (0 to 1023)
    int potValue = analogRead(POT_PIN);

    // Print the value to the Serial Monitor (optional, but helpful for debugging)
    Serial.println(potValue);

    // 2. Map the potentiometer value (0-1023) to the servo angle range (0-180)
    // The map() function is essential for scaling the input to the output.
    int servoAngle = map(potValue, 0, 1023, 0, 180);

    // 3. Write the calculated angle to the servo motor
    myServo.write(servoAngle);

    // 4. LED Control Logic (Maintain the original requirement)
    if (potValue > 500)
    {

```

```

// Turn the LED ON if the potentiometer reading is above 500

digitalWrite(LED_PIN, HIGH);

}

else

{

// Turn the LED OFF otherwise

digitalWrite(LED_PIN, LOW);

}


// Small delay to prevent jitter and avoid sending data too fast

delay(100);

}

```

### **Python Code:**

```

import serial
import matplotlib
matplotlib.use('TkAgg') # Use interactive backend for PyCharm
import matplotlib.pyplot as plt

# --- Serial connection ---
ser = serial.Serial('COM5', 9600) # Change COM port to match your Arduino

# --- Live plot setup ---
plt.ion()
fig, ax = plt.subplots()
x_vals, y_vals = [], []

```

```

try:
    while True:
        line = ser.readline().decode().strip()
        if line.isdigit(): # Make sure we got a number
            pot_value = int(line)
            print("Potentiometer Value:", pot_value)
            x_vals.append(len(x_vals))
            y_vals.append(pot_value)

            ax.clear()
            ax.plot(x_vals, y_vals, color='green', linewidth=2)
            ax.set_xlabel("Sample Number")
            ax.set_ylabel("Potentiometer Value (0–1023)")
            ax.set_title("Real-Time Potentiometer, Servo, and LED Control")
            ax.grid(True)
            plt.pause(0.1)
except KeyboardInterrupt:
    print("\nStopped by user.")
finally:
    ser.close()
    plt.ioff()
    plt.show()
    print("Serial connection closed.")

```

## Control Algorithm

1. Start initialization:
  - Include the Servo library.
  - Define pins:
    - Servo → Pin 9



- LED → Pin 11
  - Potentiometer → A0
  - Begin serial communication at **9600 baud**.
  - Set the LED pin as **OUTPUT** and attach the servo to pin 9.
2. Continuous loop execution:
- Read the potentiometer value from A0.
  - Print the value to the Serial Monitor.
  - Map the potentiometer reading (0–1023) to a **servo angle (0–180°)**.
  - Move the servo to the mapped angle.
  - If the potentiometer value is above 500, turn the LED ON; otherwise, turn it **OFF**.
  - Add a 100 ms delay for smoother operation.
3. Repeat Indefinitely:
- The loop runs continuously, updating the **servo angle** and **LED state** based on the live potentiometer input.

## 5.0.2 RESULT

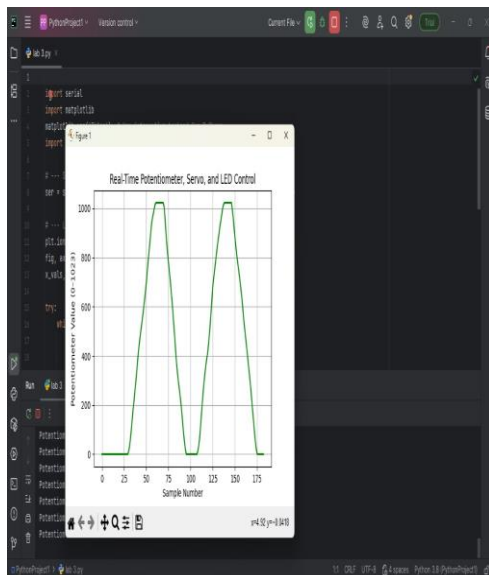
The experiment was carried out successfully to assess the LED's, servo motor's, and data visualization system's reactions to the signals from a potentiometer. Once the Arduino program was uploaded and run, the Serial Monitor showed ongoing readings from the potentiometer, varying from around 0 to 1023 as the knob of the potentiometer was turned.

When the readings surpassed the threshold value of 500, the LED linked to pin 11 illuminated. When the measurements fell under 500, the LED switched OFF. At the same time, the servo motor connected to the Arduino rotated smoothly in correspondence with the potentiometer's position, moving from 0° to 180° as the knob was turned from minimum to maximum. This demonstrated the system's ability to convert analog input values into both digital and positional outputs.

In addition, a Python script using the PySerial and Matplotlib libraries was used to create a real-time plot of the potentiometer readings. The script continuously received serial data from the Arduino and displayed a live graph that updated every 0.1 seconds. This visualization confirmed that the potentiometer output changed smoothly and consistently with the physical motion of the knob, providing clear feedback on the system's analog signal behavior.

The LED and servo reacted instantly and aligned with the position of the potentiometer. The 100 ms delay guaranteed that the adjustments were seamless and steady, avoiding flickering or jerky servo motion. The serial output and live plot both verified that the Arduino correctly interpreted and handled the potentiometer's changing resistance to control the LED and servo motor in real time.

In general, the system operated as expected; the potentiometer successfully managed both the LED state and the servo angle, and the live data plot validated the circuit and code logic were functioning properly.



## 6.0 DISCUSSION

The experiment successfully demonstrated the expected relationship between the potentiometer input and both the LED state and servo motor movement. Adjusting the potentiometer resulted in corresponding changes — the LED turned ON when the input exceeded the threshold value and OFF when below it, while the servo motor moved proportionally from 0° to 180° according to the potentiometer's position.

#### **Sources of Error and Limitations:**

- **Electrical Noise:** Minor fluctuations in ADC readings occasionally affected servo stability and LED responsiveness.
- **Hardware Tolerances:** Variations in potentiometer resistance and servo accuracy led to small inconsistencies in motion.
- **Non-Linear Response:** The servo's movement was not perfectly linear across the entire range due to mapping resolution and servo mechanics.
- **Limited Range:** The potentiometer did not always utilize the full analog input span effectively.
- **Serial Visualization Delay (Python):** The live plot introduced slight latency in data display compared to actual motion.

Despite these limitations, the experiment effectively showcased real-time control using PWM signals and analog-to-digital conversion. The system successfully integrated hardware control with live data visualization, verifying correct circuit operation and code logic.

## **7.0 CONCLUSION**

The experiment successfully demonstrated the integration of hardware and software through serial communication between the Arduino microcontroller and Python. The potentiometer effectively controlled both the LED and servo motor, illustrating the conversion of

analog input into digital and positional outputs. The LED accurately reflected threshold-based control, while the servo motor displayed proportional motion corresponding to potentiometer adjustments.

The use of Python with PySerial and Matplotlib enabled real-time visualization of potentiometer readings, enhancing understanding of analog signal variation and system response. Despite minor issues such as electrical noise and servo non-linearity, the system performed reliably and met the experiment's objectives. Overall, this experiment validated the principles of analog-to-digital conversion, pulse-width modulation (PWM) control, and real-time data monitoring in mechatronic system integration.

## 8.0 RECOMMENDATIONS

- **Improve Signal Stability:**

Use filtering techniques such as moving average or hardware capacitors to minimize electrical noise in analog readings.

- **Enhance Servo Precision:**

Implement calibration or feedback mechanisms to reduce mechanical inaccuracies and improve linearity across the 0°–180° range.

- **Optimize Visualization:**

Increase the Python plot refresh interval or buffer data points to enhance graph smoothness and reduce lag.

- **Expand System Functionality:**

Integrate additional sensors (e.g., temperature or light sensors) to further explore multi-sensor data handling and serial communication.

- **Implement Bi-Directional Communication:**

Allow Python to send commands back to Arduino for interactive control, strengthening understanding of full-duplex serial communication.

## 10.0 REFERENCES

Arduino Documentation. “*analogRead()* – Read the value from an analog pin.” Available at:

<https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/>

Arduino Documentation. “*Servo Library – Control a servo motor.*” Available at:

<https://www.arduino.cc/reference/en/libraries/servo/>

PySerial Documentation. “*pySerial: Python Serial Port Extension.*” Available at:

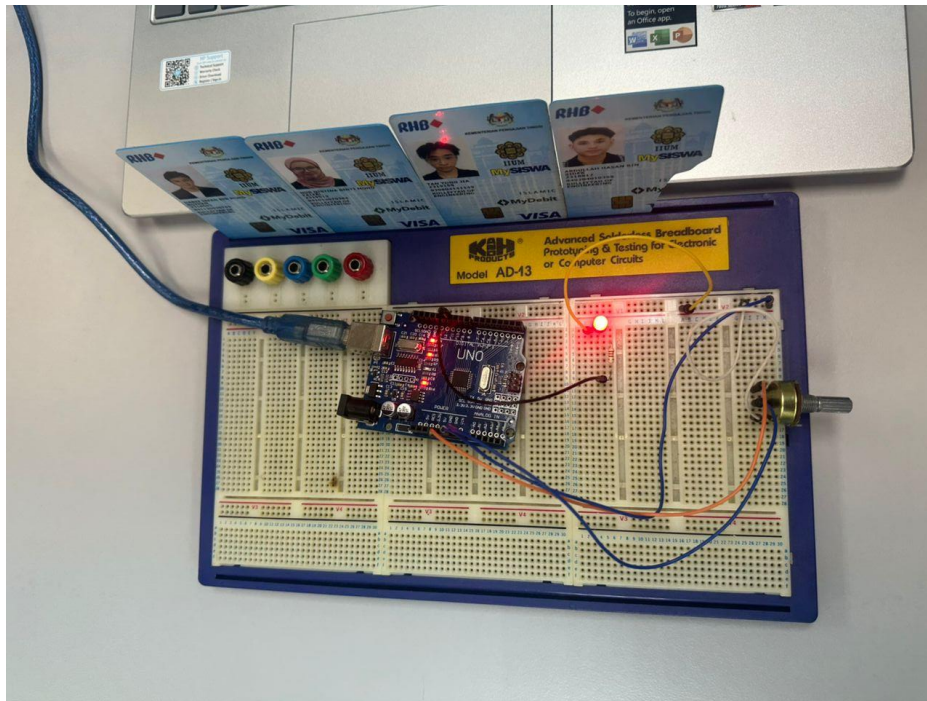
<https://pyserial.readthedocs.io/>

Matplotlib Documentation. “*Matplotlib: Visualization with Python.*” Available at:

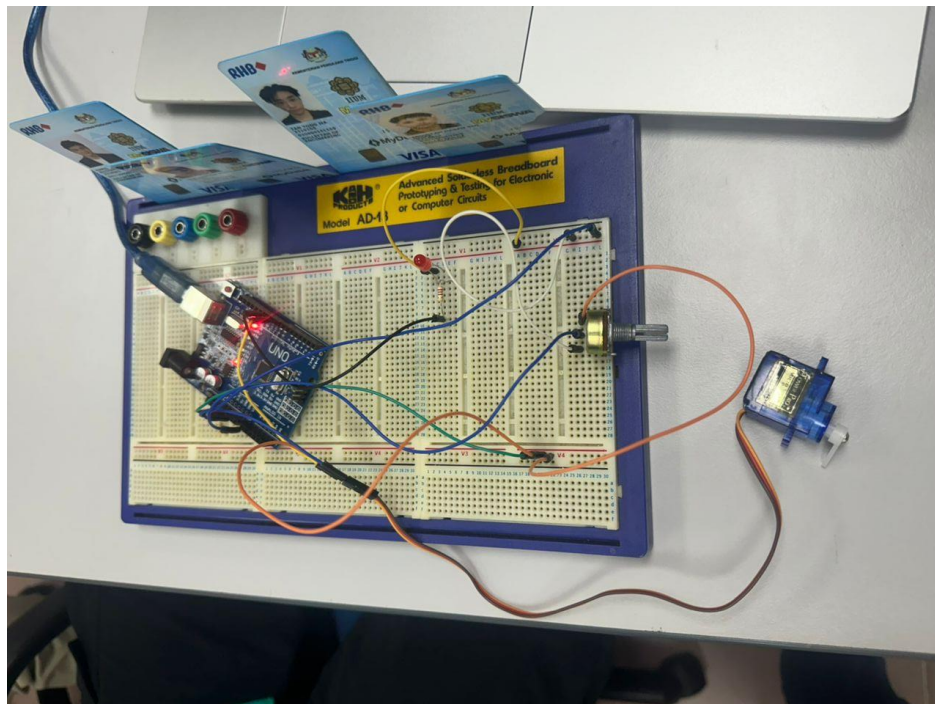
<https://matplotlib.org/stable/>

Zulkifli bin Zainal Abidin (2025). *Mechatronics System Integration (MCTA 3203) – Week 3: Serial Communication Laboratory Manual*, International Islamic University Malaysia.

## 9.0 APPENDICES



Experiment setup for Task 1



Experiment setup for Task 2


## 10.0 ACKNOWLEDGEMENTS

Hereby, we acknowledge the guidance provided by Dr Zulkifli with his impeccable knowledge in this field. We also thank Br. Harith with his assistance in correcting our work thus leading to a successful experiment. Not to forget our fellow colleagues from other groups that contributed in providing ideas that helped in achieving the same output together from this experiment.

## **11.0 STUDENTS DECLARATION**

### **Certificate of Originality and Authenticity**

We hereby certify that we are responsible for the work presented in this report. The content is our original work, except where proper references and acknowledgements are made. We confirm that no part of this report has been completed by anyone not listed as a contributor. We also certify that this report is the result of group collaboration and not the effort of a single individual. The level of contribution by each member is stated in this certificate. Furthermore, we have read and understood the entire report, and we agree that no further revisions are required. We collectively approve this final report for submission and confirm that it has been reviewed and verified by all group members.

Signature: 


Name: HARIZ IRFAN BIN MOHD ROZHAN

Matric Number: 2318583

Read [/]

Understand [/]

Agree [/]

Signature: 


Name: ABDULLAH HASAN BIN SIDEK

Matric Number: 2318817

Read [/]

Understand [/]

Agree [/]

Signature: 


Name: TAN YONG JIA

Matric Number: 2319155

Read [/]

Understand [/]

Agree [/]

Signature: 

Name: NUR QISTINA BINTI MOHD FAIZAL

Matric Number: 2319512

Read [/]

Understand [/]

Agree [/]