



## KULLIYAH OF ENGINEERING (KOE)

### MECHATRONICS SYSTEM INTEGRATION (MCTA3203)

#### SEMESTER 1, 25/26

#### SECTION 1

#### PROJECT REPORT WEEK 3

#### TITLE: SERIAL CONNECTION BETWEEN ARDUINO AND PYTHON

#### PREPARED BY :

NO	NAME	MATRIC NO
1	MUHAMMAD NAUFAL HAKIMI BIN IRWAN AFFANDI	2313041
2	MUHAMAD IQBAL BIN MD ISA	2313247
3	ADAM NABIL HANIFF BIN RUZAIMI	2313203

**DATE OF SUBMISSION: 29/10/2025**

## **ABSTRACT**

This experiment focuses on exploring the integration of Arduino-based hardware control with Python-based software visualization to achieve real-time servo motor movement and data monitoring. The system utilizes a potentiometer connected to the Arduino Uno as an analog input device, where variations in its resistance generate voltage signals that are converted into digital values ranging from 0 to 1023. These readings are then mapped to servo angles between 0° and 180°, allowing proportional control of the servo's position. Simultaneously, the Arduino transmits the potentiometer values through a serial connection to Python, where the data is processed and displayed in real time using the matplotlib library. This dual-interface approach demonstrates effective communication between embedded systems and computer-based applications. The experiment highlights key principles of mechatronics such as signal acquisition, data conversion, actuator control, and serial interfacing. The obtained results show that the servo responded smoothly to the potentiometer's rotation and that the real-time graphical plot accurately reflected changes in the input signal, confirming the successful synchronization between hardware operation and software visualization. Overall, the experiment provides a solid foundation for understanding how embedded systems interact with higher-level programming environments for automation and control applications.

## **TABLE OF CONTENT**

<b>EXPERIMENT 1.....</b>	<b>4</b>
<b>1.0 INTRODUCTION.....</b>	<b>4</b>
<b>2.0 MATERIALS AND EQUIPMENT.....</b>	<b>4</b>
<b>3.0 EXPERIMENTAL SETUP.....</b>	<b>4</b>
<b>4.0 METHODOLOGY.....</b>	<b>5</b>
<b>5.0 DATA COLLECTION.....</b>	<b>7</b>
<b>6.0 DATA ANALYSIS.....</b>	<b>8</b>
<b>7.0 RESULTS.....</b>	<b>8</b>
<b>8.0 DISCUSSION.....</b>	<b>9</b>
<b>9.0 CONCLUSION.....</b>	<b>10</b>
<b>10.0 RECOMMENDATIONS.....</b>	<b>10</b>
<b>EXPERIMENT 2.....</b>	<b>10</b>
<b>1.0 INTRODUCTION.....</b>	<b>10</b>
<b>2.0 MATERIALS AND EQUIPMENT.....</b>	<b>11</b>
<b>3.0 EXPERIMENTAL SETUP.....</b>	<b>11</b>
<b>4.0 METHODOLOGY.....</b>	<b>12</b>
<b>5.0 DATA COLLECTION.....</b>	<b>14</b>
<b>6.0 DATA ANALYSIS.....</b>	<b>15</b>
<b>7.0 RESULTS.....</b>	<b>15</b>

<b>8.0 DISCUSSION.....</b>	<b>16</b>
<b>9.0 CONCLUSION.....</b>	<b>17</b>
<b>10.0 RECOMMENDATIONS.....</b>	<b>18</b>
<b>11.0 REFERENCES.....</b>	<b>19</b>
<b>12.0 APPENDICES.....</b>	<b>20</b>
<b>13.0 ACKNOWLEDGEMENT.....</b>	<b>20</b>
<b>14.0 STUDENTS DECLARATION.....</b>	<b>21</b>

# **EXPERIMENT 1**

## **1.0 INTRODUCTION**

This experiment aims to understand how analog data from a potentiometer can be read and transmitted from an Arduino board to a computer using serial communication. By converting the analog voltage from the potentiometer into digital values, the Arduino sends real-time readings to Python through a USB serial connection. The data can be visualized using the Arduino Serial Plotter or processed further in Python for monitoring or control purposes. An LED is also added to provide visual feedback corresponding to the potentiometer's value.

## **2.0 MATERIALS AND EQUIPMENT**

The following is a complete list of equipments, components, and softwares required for the serial communication between Arduino and Python with the Arduino Uno project:

Required hardware:

1. Arduino Uno board
2. Potentiometer
3. Breadboard
4. Jumper wires
5. LED
6. 220 $\Omega$  resistor

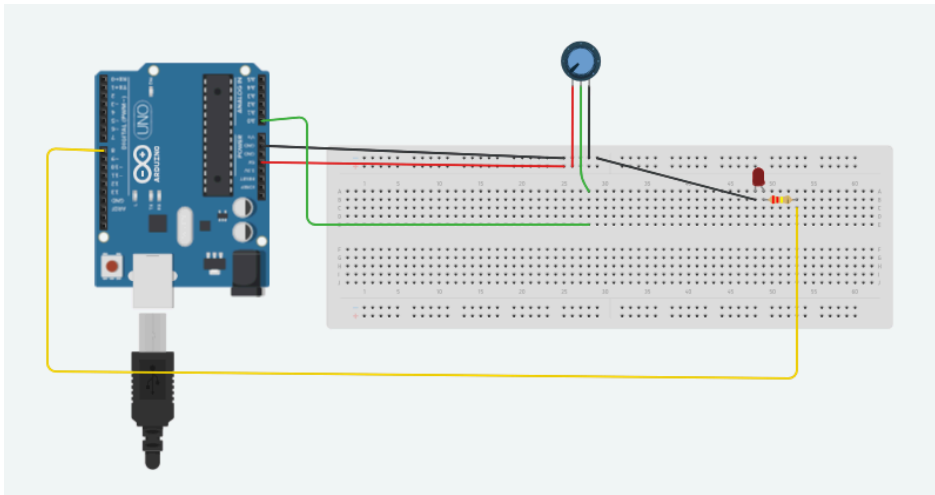
Required Software:

1. Python (PyCharm or IDLE) with pyserial and matplotlib installed
2. Arduino IDE

## **3.0 EXPERIMENTAL SETUP**

1. The experimental setup for task 1 is connected to the Arduino Uno as follows:

- Potentiometer middle pin  $\rightarrow$  Analog input A0
- Potentiometer side pins  $\rightarrow$  5V and GND
- LED anode (+)  $\rightarrow$  Digital pin D8 (through 220  $\Omega$  resistor)
- LED cathode (-)  $\rightarrow$  GND



## **4.0 METHODOLOGY**

### **Task 1:**

#### **1. Hardware Setup:**

- Assemble the circuit on a breadboard as per Figure 1.
- Ensure all components are connected securely to prevent loose contact or short circuits.

#### **2. Arduino Setup:**

- Connect the Arduino to the computer using a USB cable.
- Open the Arduino IDE and upload the provided sketch that reads the potentiometer value using `analogRead(A0)` and transmits the data via serial communication (`Serial.println()`).
- 

#### **3. Python Setup:**

- Open the Python IDE (e.g., PyCharm or IDLE).
- Run the Python script that reads and displays the potentiometer values through the serial port.

- Verify that the potentiometer readings appear in the terminal when the knob is turned.

```
import serial
import time

ser = serial.Serial('COM7', 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)

            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.")
```

---

#### 4. Real-Time Visualization (Arduino Serial Plotter):

- Ensure the Arduino Serial Plotter is not open when using Python, as both cannot access the serial port simultaneously.
- If using the Serial Plotter directly, close any Python serial connection first.
- Open Tools → Serial Plotter in the Arduino IDE, select the correct COM port, and set the baud rate to 9600.
- Observe real-time potentiometer readings displayed graphically on the plotter.

#### 5. LED Extension:

- Modify the Arduino and Python code so that the LED connected to pin D8 turns ON automatically when the potentiometer value exceeds 512 (half of 1023), and OFF otherwise.

- Re-upload the modified code and verify LED operation.

```
int potPin = A0;
int ledPin = 8;
int potValue = 0;
String command = "";

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

void loop() {

  potValue = analogRead(potPin);
  Serial.println(potValue);

  if (Serial.available() > 0) {
    command = Serial.readStringUntil('\n');
    command.trim();

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

## **5.0 DATA COLLECTION**

The potentiometer functioned as a variable voltage divider, providing an adjustable analog voltage between 0V and 5V depending on the knob's rotation. This analog voltage was read by the Arduino's analog input pin and converted into a digital value within the range of 0 to 1023, corresponding linearly to the input voltage. These readings were then transmitted to Python via serial communication for further visualization and analysis.

During this experiment, Python was programmed to continuously read potentiometer data transmitted from the Arduino through the serial communication port (COM7). The Python script displays these real-time readings in the terminal and visualises the relationship between the potentiometer's position and its corresponding digital values. Simultaneously, the Arduino Serial Plotter feature was used to illustrate a live graph that represented the analog input signal, which ranged from 0 to 1023. Below is the data obtained from the Arduino serial plotter:



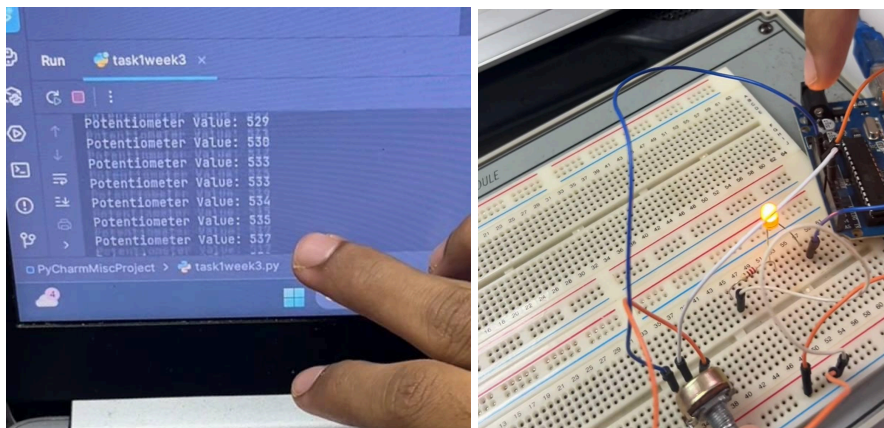
## **6.0 DATA ANALYSIS**

The analog input produces a 10-bit resolution output, meaning each digital value represents  $\sim 4.88 \text{ mV}$  ( $5 \text{ V} / 1024$ ). As the potentiometer rotates, the voltage output varies linearly, which is reflected in the serial data. The LED threshold (512) corresponds to roughly  $2.5 \text{ V}$ , marking the midpoint of the potentiometer range.

Potentiometer reading	Voltage (V)	LED state
0-511	0.0 - 2.49	OFF
512-1023	2.5-5.0	ON

## **7.0 RESULTS**

As the experiment was conducted, the Arduino successfully transmitted the potentiometer readings to the Python terminal in real time. The values displayed increased or decreased smoothly as the potentiometer knob was rotated, indicating accurate analog-to-digital conversion. The Serial Plotter provided a clear visual graph showing a linear change in data according to the potentiometer position. Additionally, the LED responded correctly by turning on when the potentiometer value exceeded approximately 512, and turning off when the value dropped below that threshold. This behavior confirmed the effective communication between hardware and software as well as the proper operation of the conditional control function.





## **8.0 DISCUSSION**

### **1. Software**

The program uploaded from the Arduino IDE was designed to control the LED's blinking rate by varying the potentiometer's analog input value. A Python script utilizing the *pyserial* library was then implemented to establish serial communication with the Arduino at a baud rate of 9600, allowing real-time collection and display of potentiometer readings in the Python terminal.

It was observed that the Arduino Serial Plotter must remain closed when the Python script is running, as both tools attempt to access the same serial port. When operated independently, the Serial Plotter provided a clear graphical visualization of potentiometer readings that changed smoothly as the knob was rotated. The Arduino and Python codes were later enhanced to enable the LED to automatically switch ON when the potentiometer reading exceeded half of its maximum range and turn OFF otherwise, effectively integrating input-based control through software.

### **2. Electrical**

The potentiometer's analog voltage output was directly associated with the LED's blink rate, where higher potentiometer readings corresponded to a faster blinking LED. Upon integrating the threshold control logic, the LED accurately responded to voltage levels illuminating when the input surpassed the midpoint threshold and switched off below it. Minor fluctuations in voltage readings occasionally caused slight inconsistencies in the LED's response time and blink speed. These variations were primarily attributed to electrical noise and the inherent resolution limitations of the Arduino's analog-to-digital converter (ADC).

### **3. Hardware**

The hardware setup consisted of an Arduino board, a 10 k $\Omega$  potentiometer, an LED connected through a 220  $\Omega$  resistor, jumper wires, and a breadboard. The potentiometer was connected to analog pin A0, while the LED was connected to digital pin 8.

During testing, achieving a consistent rotation speed of the potentiometer proved challenging, as reflected in the fluctuating slopes of the plotted data. The Serial Plotter graph showed an upward slope when turned clockwise and a downward slope when turned counterclockwise. However, due to manual inconsistencies, the response was not perfectly linear.

A more refined adjustment mechanism, such as a motorized potentiometer or smoother control knob, would likely improve accuracy and repeatability.

Nevertheless, the experiment successfully demonstrated the relationship between analog input variation and LED response, showcasing how potentiometer data can be used for real-time control applications and data visualization.

## **9.0 CONCLUSION**

During this experiment, we successfully demonstrated the interaction between analog input and digital output using an Arduino Uno, potentiometer, and LED. The system effectively converted variations in the potentiometer's resistance into corresponding changes in the LED's blinking rate. By incorporating Python through serial communication, we were able to monitor real-time potentiometer readings and visualize the data, confirming accurate synchronization between hardware and software components.

The experiment further showed that when the potentiometer value exceeded half of its maximum range, the LED automatically turned ON, and turned OFF otherwise, validating the programmed control logic. Minor variations in LED behavior were attributed to analog signal fluctuations and manual inconsistencies when adjusting the potentiometer.

Overall, this experiment reinforced the principles of analog-to-digital interfacing and demonstrated how microcontrollers can integrate sensor inputs to control electronic outputs. This understanding can be extended to a wide range of applications such as light dimmers, motor speed control, temperature regulation systems, and other automation projects in industrial or educational environments.

## **10.0 RECOMMENDATIONS**

For future improvements, it is recommended to ensure the COM port and baud-rate settings are correctly configured to prevent data transmission errors. A real-time plotting feature in Python could be integrated for better visualization and data logging. Additionally, applying signal filtering methods such as a moving average could help minimize noise in the analog readings, providing smoother and more stable results. The system could also be extended by adding multiple sensors to collect and compare different analog inputs simultaneously.

## **EXPERIMENT 2**

### **1.0 INTRODUCTION**

This experiment aims to explore the control of a servo motor using analog input from a potentiometer, integrated with an Arduino Uno and Python through serial communication. The potentiometer's analog voltage is converted into digital values by the Arduino, which are then used to adjust the servo motor's position within its 0°–180° operating range in real time. Through the serial connection, the Arduino continuously transmits servo position data to the computer, allowing Python to process and visualize these readings. To enhance system interactivity, an LED is incorporated to provide visual feedback based on the potentiometer value or servo angle, indicating specific threshold levels. Additionally, the Python program utilizes *matplotlib* to plot potentiometer data dynamically, offering a clear visualization of real-time changes.

### **2.0 MATERIALS AND EQUIPMENT**

The following is a complete list of equipments, components, and softwares required for the serial communication between Arduino and Python with the Arduino Uno project:

Required hardware:

1. Arduino Uno board
2. Servo motor
3. Potentiometer
4. Breadboard
5. Jumper wires
6. LED
7. 220Ω resistor

Required Software:

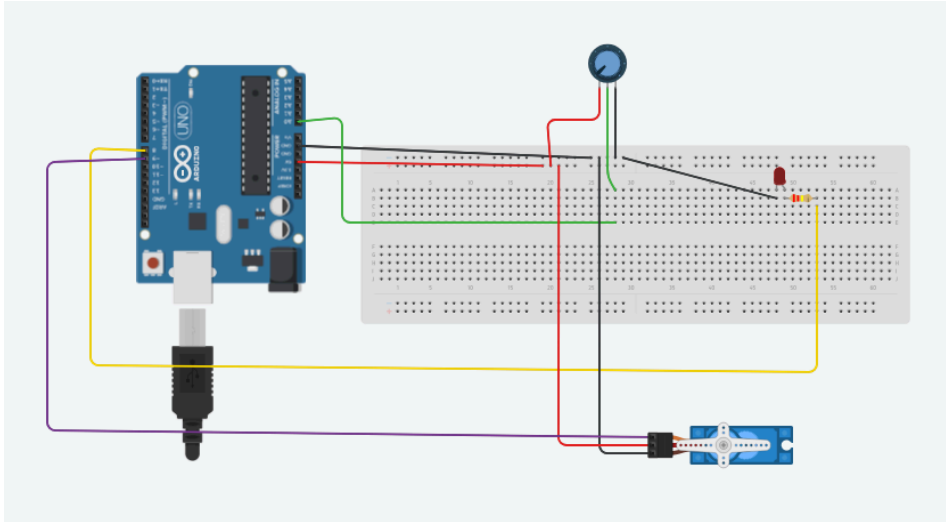
1. Python (PyCharm or IDLE) with pyserial and matplotlib installed
2. Arduino IDE

### **3.0 EXPERIMENTAL SETUP**

1. The experimental setup for task 2 is connected to the Arduino Uno as follows:

- Potentiometer middle pin → **A0** (Analog input)
- Potentiometer side pins → **5V** and **GND**
- Servo signal → **D9**

- Servo VCC → **5V**, Servo GND → **GND**
- LED anode (+) → **D8** (through 220  $\Omega$  resistor)
- LED cathode (-) → **GND**



## 4.0 METHODOLOGY

### Task 2:

#### 1. Arduino Configuration:

1. The potentiometer was connected to analog input A0.
2. The servo motor was connected to digital pin 9 and controlled using the Servo.h library.
3. The LED was connected to digital pin 8 to indicate when the potentiometer value exceeded 512.
4. The Arduino was programmed to:
  - Read the potentiometer input using `analogRead(A0)`.
  - Map the value (0–1023) to servo angles (0–180°).
  - Send potentiometer readings through the serial port to Python.
  - Light the LED when the potentiometer value was greater than 512.

5. A stop feature was implemented, allowing Python to send the character 'q' through the serial connection to halt the Arduino loop.

The coding is as shown below:

```
2
3  #include <Servo.h>
4
5  Servo myServo;
6  const int potPin = A0;
7  const int ledPin = 8; // LED on D8
8  int angle = 0;
9  char incomingChar;
10
11 void setup() {
12     Serial.begin(9600);
13     myServo.attach(9);
14     pinMode(ledPin, OUTPUT);
15     Serial.println("Arduino ready. Send 'q' to stop.");
16 }
17
18 void loop() {
19     // Check if Python sent any command
20     if (Serial.available() > 0) {
21         incomingChar = Serial.read();
22         if (incomingChar == 'q' || incomingChar == 'Q') {
23             Serial.println("Execution halted by Python.");
24             while (true); // Halt Arduino loop
25         }
26     }
27
28     // Read potentiometer (0-1023) and map to 0-180°
29     int potValue = analogRead(potPin);
30     angle = map(potValue, 0, 1023, 0, 180);
31
32     // Move servo
33     myServo.write(angle);
34
35     // LED control: ON if pot value > 512, else OFF
36     if (potValue > 512) {
37         digitalWrite(ledPin, HIGH);
38     } else {
39         digitalWrite(ledPin, LOW);
40     }
41
42     // Send angle + LED state to Python
43     Serial.print("Servo angle: ");
44     Serial.print(angle);
45     Serial.print(" | LED: ");
46     Serial.println((potValue > 512) ? "ON" : "OFF");
47
48     delay(200); // smooth update
49 }
50
```

## 2. Python Configuration:

1. The Python script established a serial connection to the Arduino using pyserial.
2. It continuously read incoming potentiometer data, displayed it in the terminal, and plotted it in real time using the matplotlib interactive mode.
3. The servo and LED responses were observed simultaneously while monitoring the plotted data.

The python The coding is as shown below:

```
42 import serial
43 import time
44 import keyboard # pip install keyboard
45
46 # Adjust this port number as needed
47 ser = serial.Serial('COM7', 9600, timeout=1)
48 time.sleep(2) # Wait for Arduino to reset
49
50 print("Connected to Arduino. Press 'q' to stop Arduino.")
51
52 try:
53     while True:
54         # Read data from Arduino
55         if ser.in_waiting > 0:
56             line = ser.readline().decode('utf-8').strip()
57             if line:
58                 print(line)
59
60         # If 'q' is pressed, send stop signal
61         if keyboard.is_pressed('q'):
62             ser.write(b'q')
63             print("Stop command sent to Arduino.")
64             break
65
66 except KeyboardInterrupt:
67     print("Program interrupted manually.")
68
69 finally:
70     ser.close()
71     print("Serial connection closed.")
```

## 3. Execution Steps:

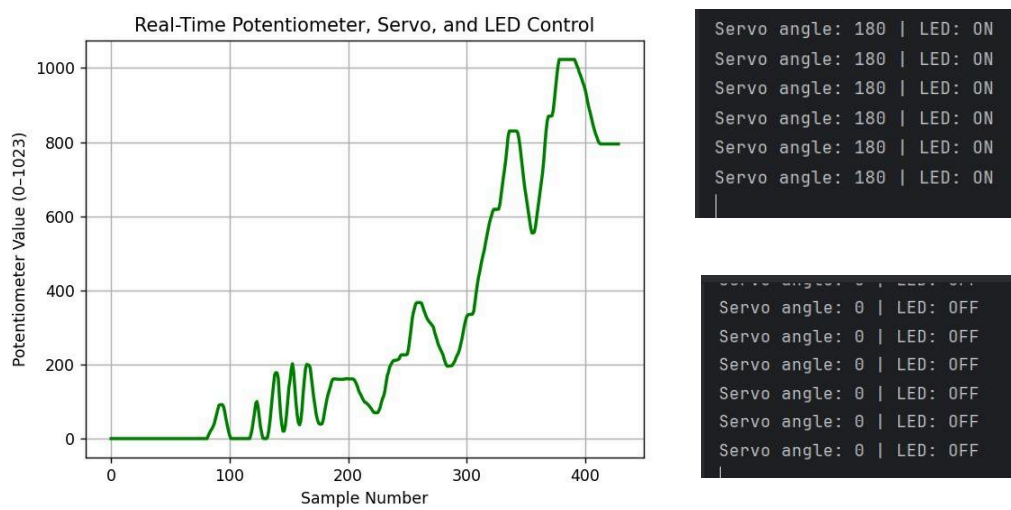
1. Upload the Arduino sketch to the Arduino Uno.
2. Run the Python script on the computer.
3. Rotate the potentiometer knob slowly to vary the servo's position.
4. Observe the servo motor movement, LED behavior, and the live potentiometer plot in Python.
5. Press the 'q' key in Python to stop the Arduino execution.

## **5.0 DATA COLLECTION**

In this experiment, the Arduino was programmed to read analog input from a potentiometer and map it to a corresponding servo angle between  $0^{\circ}$  and  $180^{\circ}$ . These readings were continuously sent to Python through serial communication (COM7) for real-time monitoring and visualization using the matplotlib library.

As the potentiometer was rotated, values ranged from 0 to 1023, representing voltages between 0V and 5V. The servo motor's position adjusted proportionally, while an LED turned ON whenever the potentiometer value exceeded 512 (around 2.5V), indicating the midpoint of the range.

Data were collected by rotating the potentiometer from minimum to maximum and back, while observing changes in servo position and LED status.



## 6.0 DATA ANALYSIS

Using the **map()** function in Arduino, potentiometer readings (0–1023) were scaled to servo angles (0–180°).

Servo response closely followed potentiometer movement, demonstrating proportional control.

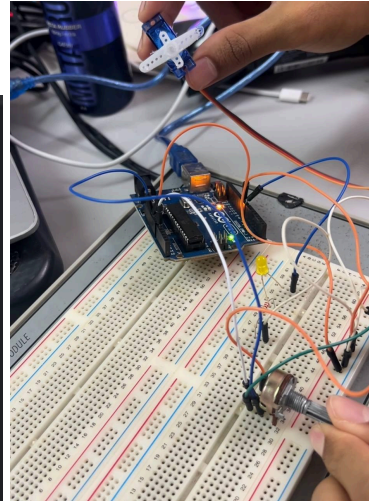
The LED threshold at 509 divides the potentiometer range roughly in half ( $\approx 2.49$  V), synchronizing visual indication with servo mid-position.

The Matplotlib graph showed smooth variation, confirming consistent data transmission and real-time plotting capability.

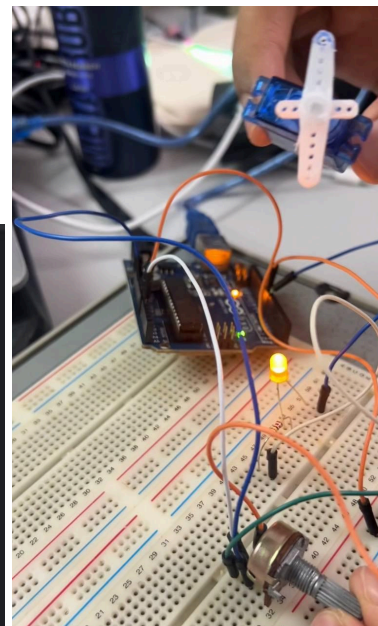
## **7.0 RESULTS**

During the experiment, the servo motor's rotation followed the potentiometer's position proportionally and smoothly. When the potentiometer value was low, the servo was positioned near  $0^\circ$ , and as the knob was turned clockwise, the servo angle increased up to  $180^\circ$ . The LED illuminated once the potentiometer passed the midpoint value of approximately 512, serving as an indicator of threshold crossing. The Python script displayed the potentiometer data in real time, and Matplotlib successfully plotted the changing values on a live graph. The data transmission remained stable throughout the experiment, showing effective synchronization between hardware control and software visualization.

```
Servo angle: 90 | LED: OFF  
Servo angle: 90 | LED: OFF  
Servo angle: 90 | LED: OFF  
Servo angle: 90 | LED: OFF  
Servo angle: 90 | LED: OFF  
Servo angle: 90 | LED: OFF
```



```
Servo angle: 91 | LED: ON  
Servo angle: 91 | LED: ON  
Servo angle: 91 | LED: ON  
Servo angle: 91 | LED: ON  
Servo angle: 91 | LED: ON  
Servo angle: 91 | LED: ON
```





## **8.0 DISCUSSION**

### **1. Software**

The integration of Python and Arduino to control the servo motor was successfully achieved, with real-time angle adjustments based on potentiometer readings. The Python script established a serial connection to the Arduino, continuously sending and receiving data at a baud rate of 9600. This allowed for smooth synchronization between the potentiometer's analog values and the servo motor's angular position. An additional feature was implemented to halt the servo's operation using a designated key press on the computer keyboard, providing an effective safety and debugging mechanism. Furthermore, an LED indicator was programmed to illuminate when the potentiometer value or equivalently, the servo's angle exceeded a specified threshold, adding a visual layer of feedback.

To enhance data visualization, the Python program was expanded to include *matplotlib* for real-time plotting of potentiometer values. This feature enabled users to observe angle fluctuations and servo responses graphically, offering clearer insights into the system's dynamic behavior. Future improvements may include adding a graphical user interface (GUI) for more intuitive servo angle adjustments and LED threshold configuration.

### **2. Electrical**

The servo motor performed reliably across its full 0°–180° range, accurately following the potentiometer's analog input. When the potentiometer was adjusted gradually, the servo responded smoothly; however, rapid adjustments occasionally produced minor lag due to the servo's mechanical limits and analog signal fluctuations. The LED feedback system effectively indicated threshold crossings, validating the correct implementation of the electrical control logic.

Performance could be further optimized by stabilizing the power supply and refining the analog-to-digital sampling interval in the Arduino code. These adjustments would reduce latency and voltage noise, ensuring better synchronization between potentiometer input, servo movement, and LED state changes.

### **3. Hardware**

The physical setup, consisting of the Arduino board, servo motor, potentiometer, LED, resistor, jumper wires, and breadboard, provided a clear demonstration of real-time hardware interaction. While the servo's response closely matched the potentiometer's motion, maintaining consistent manual control of the knob remained challenging. Quick or uneven rotations sometimes caused short delays in servo positioning and data plotting.

Replacing the potentiometer with one offering finer rotational precision could improve responsiveness and data accuracy. Overall, the experiment effectively showcased the integration of analog input, digital control, and visual feedback

through both LED indication and live data plotting, illustrating the capabilities of Arduino-Python communication in mechatronic systems.

## **9.0 CONCLUSION**

During this experiment, we successfully demonstrated real-time control of a servo motor using a potentiometer interfaced with an Arduino Uno and Python. The system accurately translated analog input from the potentiometer into corresponding servo motor positions within the  $0^{\circ}$ – $180^{\circ}$  range. Communication between Python and Arduino was established via serial connection, enabling smooth transmission of control signals and feedback data.

The implementation of a keyboard-controlled halt function provided a convenient method to stop the servo operation safely, while the addition of an LED indicator offered immediate visual feedback when the potentiometer or servo angle exceeded a set threshold. The integration of *matplotlib* for real-time graph plotting further enhanced the experiment by allowing live visualization of potentiometer readings, improving system monitoring and data interpretation.

Overall, the experiment effectively illustrated the synergy between hardware and software components in a control system. It reinforced key principles of analog-to-digital interfacing, feedback control, and real-time data visualization. This understanding can be extended to advanced automation systems such as robotic arms, sensor-based control mechanisms, and smart actuator systems used in industrial or educational applications.

## **10.0 RECOMMENDATIONS**

It is recommended to use a higher baud rate, such as 115200, to achieve smoother and faster real-time data plotting. Servo calibration can also be performed to ensure precise alignment between the potentiometer range and servo motion. Additionally, a button or keyboard interrupt function can be implemented to safely stop the servo operation when needed. Future improvements could include expanding the system to control multiple servos simultaneously or storing movement data for analysis and automation purposes. These enhancements would provide a more comprehensive understanding of servo behavior and improve system reliability.

## **11.0 REFERENCES**

LME Editorial Staff. (2025, July 31). *How servo motor works & Interface it with Arduino*. Last Minute Engineers.

<https://lastminuteengineers.com/servo-motor-arduino-tutorial/>

Instructables. (2017, October 6). *Arduino Servo Motors*. Instructables.

<https://www.instructables.com/Arduino-Servo-Motors/>

ArduinoGetStarted (n.d.). *Arduino - LED Matrix*. Arduino Get Started.

<https://arduinogetstarted.com/tutorials/arduino-led-matrix>

Scott Campbell (n.d.). *HOW TO SETUP LED MATRIX DISPLAYS ON THE ARDUINO*.

Retrieved October 20, 2023, from

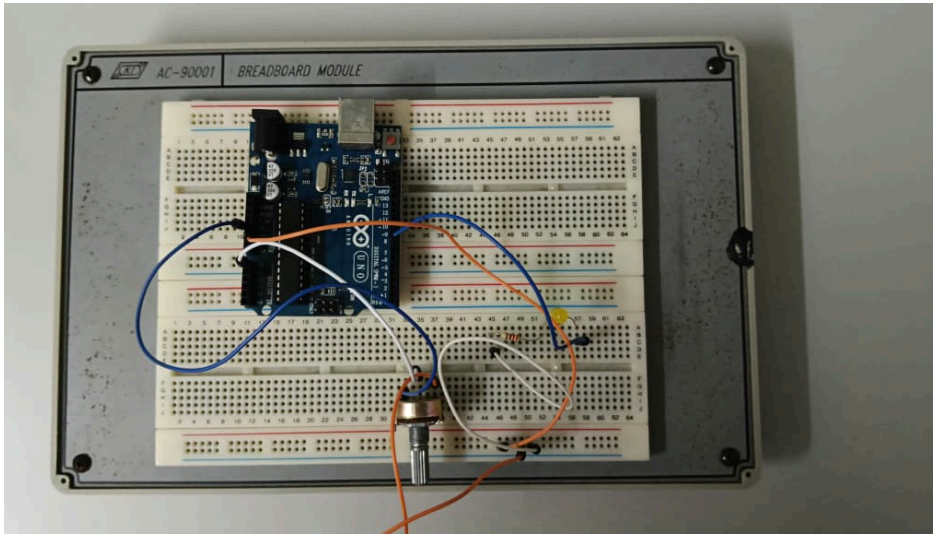
<https://www.circuitbasics.com/how-to-setup-an-led-matrix-on-the-arduino/>

Chris (n.d.). *5 Simple Ways to Reset Arduino*. Chip Wired. Retrieved October 24, 2023, from

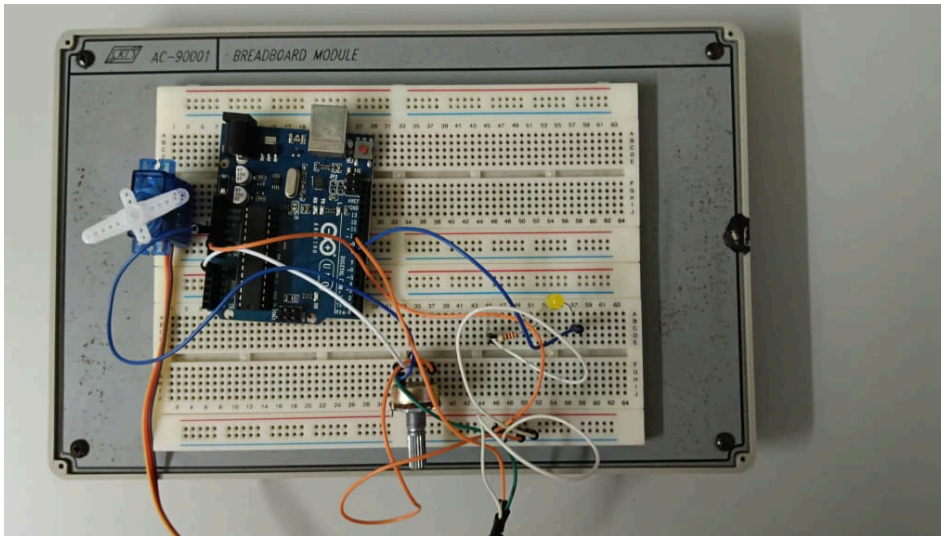
<https://chipwired.com/5-simple-ways-to-reset-arduino/>

## **12.0 APPENDICES**

### **Experiment 1**



### **Experiment 2**



## **13.0 ACKNOWLEDGEMENT**

We would like to express our gratitude to my lab instructor, Dr Zulkifli bin Zainal Abidin, for his guidance during the experiment. Special thanks to my groupmates and peers for their collaboration and help during circuit setup and troubleshooting.

## **14.0 STUDENTS 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 specific in references and acknowledgement, and that the original work contained herein have not been untaken or done by unspecified sources or a person.

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.

Signature: 	Read	/
Name: Adam Nabil Haniff bin Ruzaimi	Understand	/
Matric Number: 2313203	Agree	/
Contribution: Introduction, equipments and results		

Signature: 	Read	/
Name: Muhammad Naufal Hakimi bin Irwan Affandi	Understand	/
Matric Number: 2313041	Agree	/
Contribution: Procedure, results, discussion		

Signature: 	Read	/
Name: Muhamad Iqbal bin Md Isa	Understand	/
Matric Number: 2313247	Agree	/
Contribution: : Results, discussion, conclusion		