LAB REPORT 2

MECHATRONICS SYSTEM INTEGRATION

DR. ZULKIFLI BIN ZAINAL ABIDIN

SECTION 1

GROUP 6

MATRIC NUMBER	NAME
2312369	MUHAMMAD FAHIM BIN AHMAD NORISHAM
2312479	MUHAMMAD HAZIMUDDIN BIN FAIRUZ
2312451	MUHAMMAD ZAMARUL AZIM BIN ZULKHIBRI

DATE OF SUBMISSION

Wednesday, 29th October 2025

Abstract

This week experiments were focused to establish serial communication between an Arduino UNO and Python to explore real-time hardware and software (PyCharm) interaction. In addition to controlling the servomotor and LED by configuring the value of ADC by rotating the potentiometer. Two main tasks were conducted. In **Task 1**, potentiometer readings were transmitted from the Arduino to Python, where the LED was controlled based on the input value. In **Task 2**, a servo motor was controlled by the potentiometer through serial communication, and its angular position was plotted in real time using Python. The experiment provided hands-on experience in integrating electronic components with programming interfaces, enhancing understanding of mechatronic system communication and control.

Keywords: Arduino UNO, Python, Serial Communication, Potentiometer, LED Control, Servo Motor, Mechatronic System

Table of Contents

WEEK 3: SERIAL COMMUNICATION BETWEEN ARDUINO AND PYTHON

1. Introduction

This experiment explores the integration of hardware and software through serial communication between an Arduino UNO and Python. Serial communication allows data to be transferred between a microcontroller and a computer, enabling real-time control and monitoring of electronic components. The main hardware components used in this experiment include the Arduino UNO, **potentiometer**, **LED**, and **servo motor**, while the software components involve the Arduino IDE for coding the microcontroller and **PyCharm** for running Python scripts.

In Task 1, the Arduino reads analog voltage values from a potentiometer and transmits them to Python via serial communication. The LED is then controlled based on the received input level. The configuration of potentiometer is shown in Figure 1.0 below:

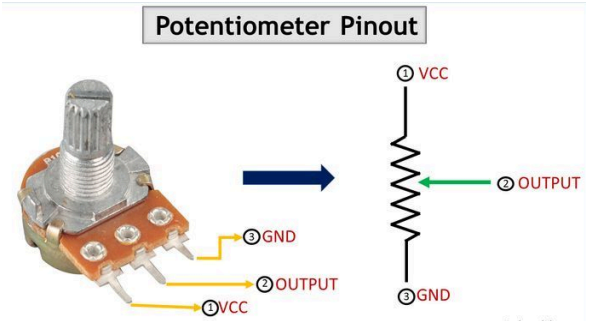

Figure 1.0 Shows the potentiometer pinout

In Task 2, a potentiometer serves as an input device to regulate the angular position of a servo motor. As the potentiometer is adjusted, the changing resistance alters the voltage output. This data is transmitted to a Python program that facilitates real-time plotting and visualization of the servo motor's position.

Our main objectives in this experiment were:

- To light-up LED by adjusting the potentiometer value from Arduino to Python via USB serial.
- Enable real-time adjustment of the servo motor’s angle through potentiometer input.
- Implement a feature in the Arduino–Python setup to stop the program execution via a designated keyboard key.
- Incorporate graphical visualization using *matplotlib* in Python to plot the potentiometer or servo position data in real time.

2. Materials and Equipments

Arduino UNO	1
Potentiometer	1
Light Emitting Diode, LED	1
220 Ω Resistor	1
Servo Motor (SG90)	1
Breadboard	1
Jumper Wires	

3. Experimental Setup

3.1 Task 1

The potentiometer was connected to the Arduino through a breadboard. One outer leg was connected to a common 5V rail, the other to common ground rail, GND, and the middle pin to analog input A0. Anode of the LED will connect to pin 7 through 220Ω resistor and the cathode will connect to the common ground rail, GND. The Arduino sends the analog readings to PyCharm using serial communication and the LED will turn ON based on the value of the potentiometer knob turn. The circuit setup will be shown below.

Circuit Description:

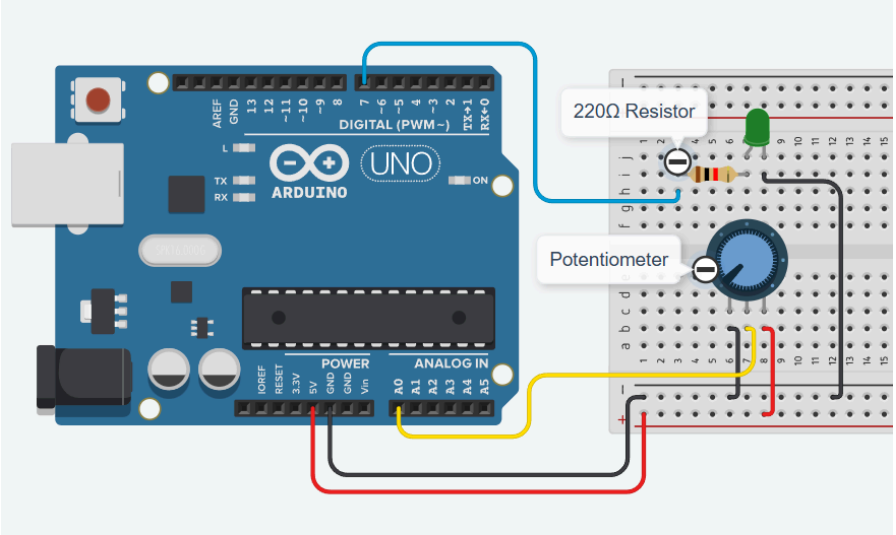

Figure 3.0 Shows the schematic diagram for Task 1 using Tinkercad

- When the potentiometer knob is turned and the value exceeds half of the maximum value, the LED will turn ON.

3.2 Task 2

The servo motor signal pin was connected to digital pin 9 of the Arduino, with power connected to 5V and ground to GND. The Arduino received angle values from Python and moved the servo accordingly. Then, LED was added and a potentiometer was used to adjust the servo angle in real time.

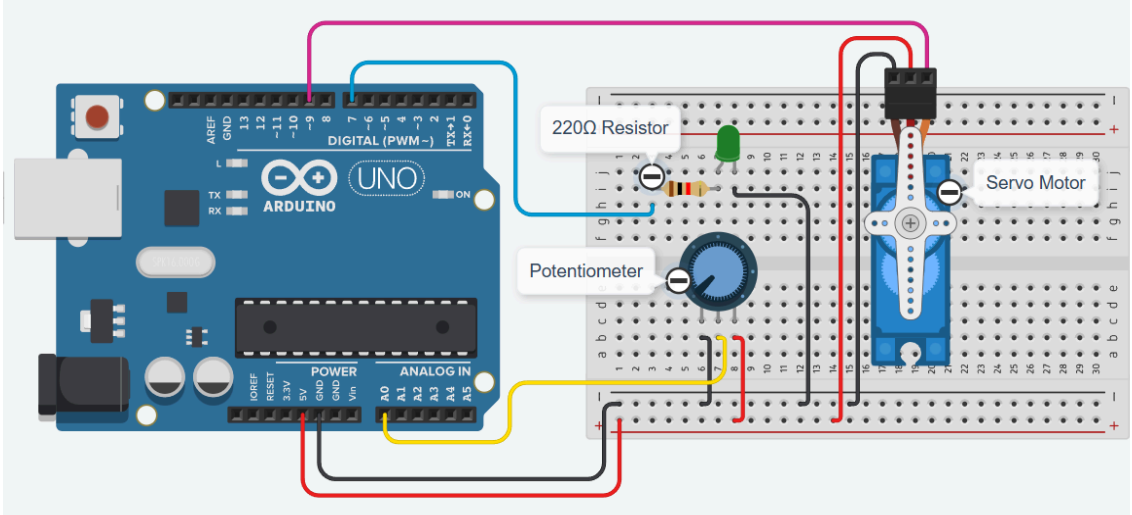

Figure 3.1 Shows the schematic diagram for Task 2 using Tinkercad

- The Arduino reads the potentiometer's analog input and adjusts the servo motor position accordingly.
- The servo angle data is sent from Arduino to Python via serial communication.
- An LED is added to provide visual feedback, lighting up or changing brightness based on the potentiometer value or servo position.
- A keyboard control feature is implemented to allow stopping the program execution safely.

4. Methodology

4.1 Task 1

4.1.1. Circuit Assembly

All **connections** were made on a breadboard based on **Figure 3.0**. The potentiometer was connected to the Arduino analog input pin (A0). An LED was connected to a digital pin 7 output pin through a 220 Ω resistor. The Arduino was powered via USB from the computer, providing 5 V logic for all components. Jumper wires were used to complete all connections.

4.1.2. Serial Communication Setup

The Arduino IDE was used to write and upload the code to the Arduino UNO. The program read analog input from the potentiometer, calculated voltage and resistance values, and formatted the data as "Voltage / ADC: value / R: resistance". This data was continuously sent to Python via the serial port (COM7). Python received the data, parsed it, and sent commands "LED_ON" or "LED_OFF" back to Arduino depending on whether the ADC value exceeded 512. Arduino updated the LED state based on the received command.

4.1.2.1 Code Upload to Arduino IDE

```
const int potPin = A0;
const int ledPin = 7;
const float Vin = 5.0;
const float R_total = 10000;

String command = "";

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
}
void loop() {
  int potValue = analogRead(potPin);
```

```
  float Vout = (potValue / 1023.0) * Vin;
  float R_wiper = (Vout / Vin) * R_total;

  // Send data to Python
  Serial.print(Vout);
  Serial.print("/ ADC: ");
  Serial.print(potValue);
  Serial.print("/ R: ");
  Serial.println(R_wiper);
  Serial.println(potValue);

  if (Serial.available() > 0) {
    command = Serial.readStringUntil('\n');
    command.trim();
    if (command == "LED_ON") {
      digitalWrite(ledPin, HIGH);
    }
    else if (command == "LED_OFF") {
      digitalWrite(ledPin, LOW);
    }
  }
  delay(500);
}
```

4.1.3 Serial Plotter

The Arduino Serial Plotter was used to graph the potentiometer values in real time. In the Arduino IDE, the correct COM port was selected, and the baud rate was set to match the value in the Arduino code which is 9600. As the potentiometer knob was turned, the Serial Plotter displayed the readings in real time, creating a graphical representation of the data. The graph settings, including range, labels, and colors, were adjusted as needed. The LED responded automatically as the potentiometer value crossed the threshold, providing immediate visual feedback. To open the Serial Plotter, in Arduino IDE, go to Tools then Serial Plotter. The Serial Plotter was kept closed whenever Python was reading data to avoid port conflicts.

4.1.4. Python Script

Python (PyCharm) was used to run the script for receiving potentiometer readings. The script displayed real-time values in the terminal. Python received the data, parsed it, and sent commands "LED_ON" or "LED_OFF" back to Arduino depending on whether the ADC value exceeded 512. Arduino updated the LED state based on the received command.

4.1.4.1 Code Upload to PyCharm

```
import serial
import time

ser = serial.Serial('COM7', 9600)
time.sleep(2)  # wait for Arduino to reset

print("Running... Press CTRL+C to stop.")

try:
   while True:
       line = ser.readline().decode().strip()

       if "/" in line:
           try:
               #Parse Arduino data
               data = line.split("/")
               voltage = float(data[0])
               adc_value = int(data[1].replace("ADC:",
"").strip())
               resistance = float(data[2].replace("R:",
"").strip())

               print(f"Voltage: {voltage:.2f} V | ADC:
{adc_value} | Resistance: {resistance:.2f} Ω")

               # LED Control Logic
               if adc_value > (1023 / 2):
                   ser.write(b"LED_ON\n")
               else:
                   ser.write(b"LED_OFF\n")
           except:
               print("Parsing Error:", line)

except KeyboardInterrupt:
   print("Stopped by user.")

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

4.2 Task 2

4.2.1 Circuit Assembly

All **connections** were made on a breadboard based on **Figure 3.1**. The potentiometer was additionally used to control a servo motor. The servo signal pin was connected to digital pin 9 on the Arduino, while VCC and GND were connected to 5 V and GND, respectively. The LED from Task 1 was reused for visual feedback based on potentiometer or servo position.

4.2.2. Code Upload to Arduino IDE

The Arduino IDE was used to write and upload enhanced code. The Arduino mapped potentiometer input to a servo angle between 0° and 180°, continuously sent the angle to Python, and updated the LED based on the potentiometer or servo position. A keyboard control feature was added to halt Arduino execution safely when prompted.

```
#include <Servo.h>
Servo myservo;

int potPin = A0;
int ledPin = 7;

bool stopControl = false;
int potValue, angle;

void setup() {
  Serial.begin(9600);
  myservo.attach(9);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  if (Serial.available() > 0) {
    char command = Serial.read();

    if (command == 'x') {
      stopControl = true;
      myservo.detach();
      digitalWrite(ledPin, LOW);
      Serial.println("STOP");}
  }
  if (!stopControl) {
    potValue = analogRead(potPin);
    angle = map(potValue, 0, 1023,
0, 180);

    myservo.write(angle);

    // LED control: ON bila lebih
dari 90°
    digitalWrite(ledPin, angle >
90 ? HIGH : LOW);

    // Hantar data ke Python
    Serial.println(angle);
    delay(50);}
}
```

4.2.3. Python Script for Servo and Visualization at PyCharm

Python received servo angle data over the serial connection and plotted it in real time using `matplotlib`. Interactive plotting allowed live observation of the servo motion while adjusting the potentiometer. The serial connection was closed safely after the experiment ended.

```
import serial
import matplotlib.pyplot as plt
import time

ser = serial.Serial('COM7', 9600)
time.sleep(2)

plt.ion()
fig, ax = plt.subplots()
x_vals, y_vals = [], []

try:
    while True:
        raw = ser.readline().decode().strip()

        if raw.isdigit():
            angle = int(raw)
            print("Servo Angle:", angle)

            x_vals.append(len(x_vals))
            y_vals.append(angle)

            ax.clear()
            ax.set_ylim(0, 180)
            ax.set_title("Real-Time Servo Angle Plot")
            ax.set_xlabel("Samples")
            ax.set_ylabel("Angle (°)")
            ax.plot(x_vals, y_vals)
            plt.pause(0.05)

except KeyboardInterrupt:
    print("Sending stop command to Arduino...")
    ser.write(b'x')  # Hantar stop signal ke Arduino
    time.sleep(1)

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

5. Data Collection

5.1 Task 1

The Arduino IDE continuously displays the Serial Plotter as the potentiometer knob is turned. The graph of the serial plotter is shown below:

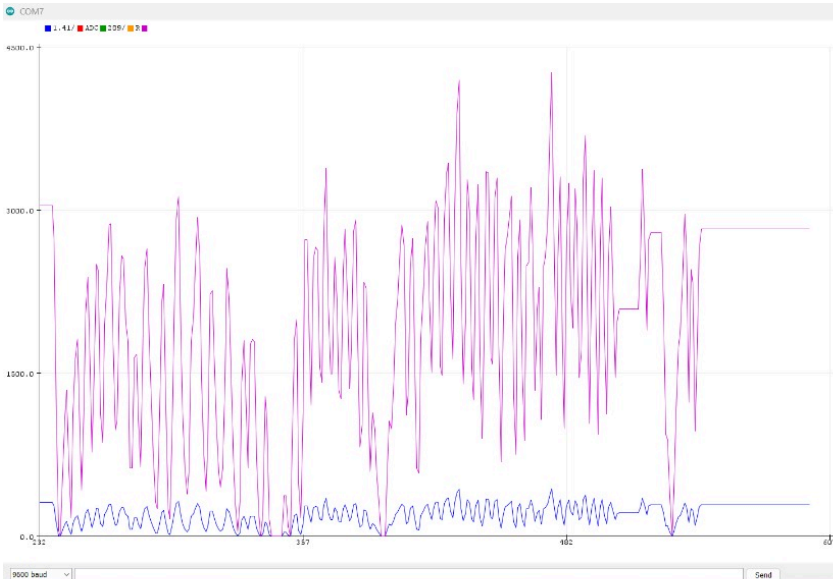

Figure 5.1.1 Shows the Serial Plotter

The PyCharm software continuously displays:

- Voltage (i n volts)
    - By using this formula: `Vout = (potValue / 1023.0) * Vin`
- ADC value (0–1023)
    - The LED will light up when the ADC value reaches half of its maximum value (512) that can be manipulated when turning the knob of potentiometer.
- Calculated resistance (Ω)
    - By using this formula: `R_wiper = (Vout / Vin) * R_total`

Example readings from the PyCharm terminal shown in table below:

Sample	Voltage (V)	ADC	Resistance (Ω)	LED State
1	1.20	245	2400	OFF
2	2.60	532	5200	ON
3	4.80	980	9500	ON

Table 5.1.1 Shows the output from the PyCharm and real-time observation.

5.2 Task 2

The PyCharm continuously displays the plotter by using `matplotlib` for graphical visualization as the potentiometer knob is turned. The graph of the serial plotter is shown below:

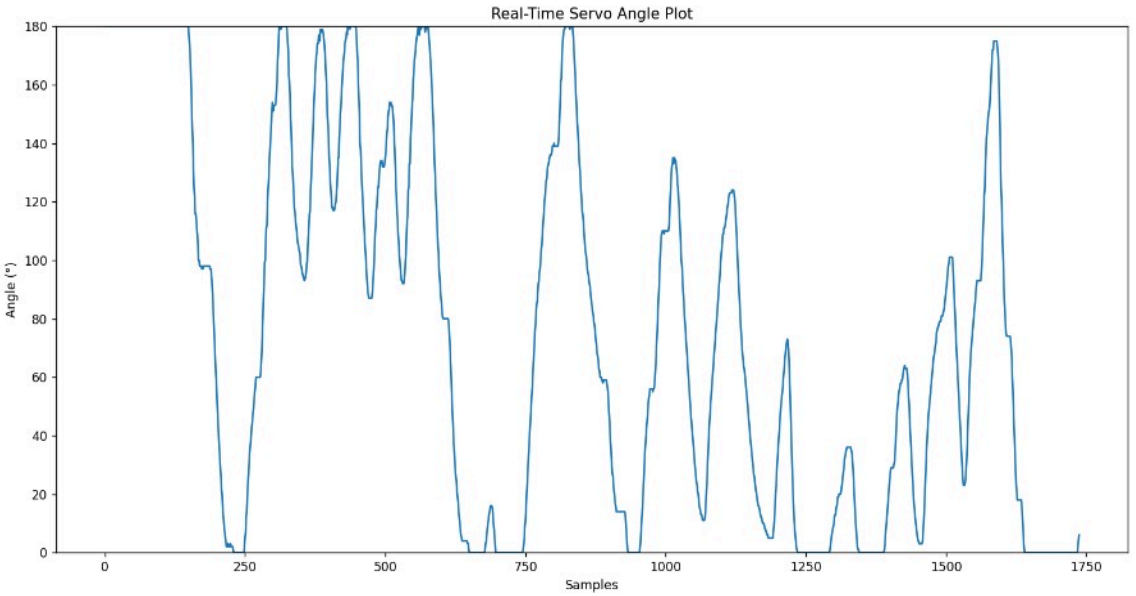

Figure 5.2.1 Shows the Graphical Visualization from PyCharm

Python receives servo angle values and plots them in real-time and the angle of the servo motor angle is shown in the output at PyCharm.

Example readings from the PyCharm terminal shown in table below:

Sample	Potentiometer Value	Servo Angle (°)	LED State
1	102	18	OFF
2	512	90	ON
3	845	148	ON

Table 5.2.1 Shows the output from the PyCharm and the real-time observation.

6. Data Analysis

6.1 Task 1

In Task 1, the potentiometer acted as an analog input device, generating voltage variations proportional to its rotational position. These voltage values were converted by the Arduino's ADC (Analog-to-Digital Converter) into numerical values ranging from 0 to 1023. The corresponding voltage output, Vout, was calculated using the equation:

$$Vout = (potValue / 1023.0) \times Vin$$

where Vin = 5V. The wiper resistance, Rwiper, was determined as:

$$Rwiper = Vout / Vin \times Rtotal \qquad (\text{with } Rtotal=10k\Omega.)$$

The collected data indicated that as the potentiometer was rotated clockwise, the ADC value and voltage output increased linearly. When the ADC reading exceeded approximately 512 (half of 1023), the LED turned ON, confirming the bidirectional communication between Arduino and PyCharm. The serial monitor and PyCharm outputs validated the correct data transmission and command response.

This behavior demonstrated the accurate implementation of threshold-based LED control, showing that serial communication allowed PyCharm to interpret sensor data and transmit control signals to Arduino efficiently.

6.2 Task 2

In Task 2, the potentiometer's analog input was mapped to the servo motor's angular position (0°–180°) using the Arduino map() function. The servo motor responded proportionally to potentiometer rotation, confirming a precise mapping relationship. The real-time graph generated in PyCharm using matplotlib displayed a graph of servo angle movement versus samples, indicating stable serial data transmission.

The LED control, synchronized with servo angle (ON when > 90°), visually verified correct conditional logic. The data collected from the terminal showed consistent correlation between potentiometer readings and servo angles. The addition of the keyboard stop command ('x') successfully terminated the program without data corruption, highlighting effective integration between hardware and software.

7. Results

7.1 Task 1

The results confirmed that the Arduino successfully read analog signals from the potentiometer and communicated them to PyCharm through serial communication. The LED responded immediately according to the potentiometer's position. When the potentiometer value exceeded half of its range (ADC > 512), the LED turned ON, and when below, it turned OFF.

The voltage, ADC, and resistance values displayed in PyCharm matched theoretical expectations based on the potentiometer's resistance division. The serial plotter graph showed a linear relationship between ADC value and resistance. The successful bidirectional control validated the efficiency of the serial communication setup and the output shown below.

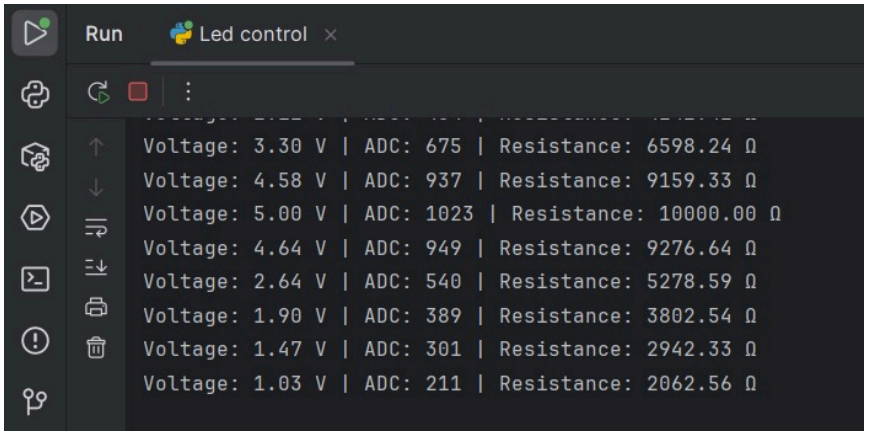

Figure 7.1.1 Shows the output of the PyCharm for Task 1

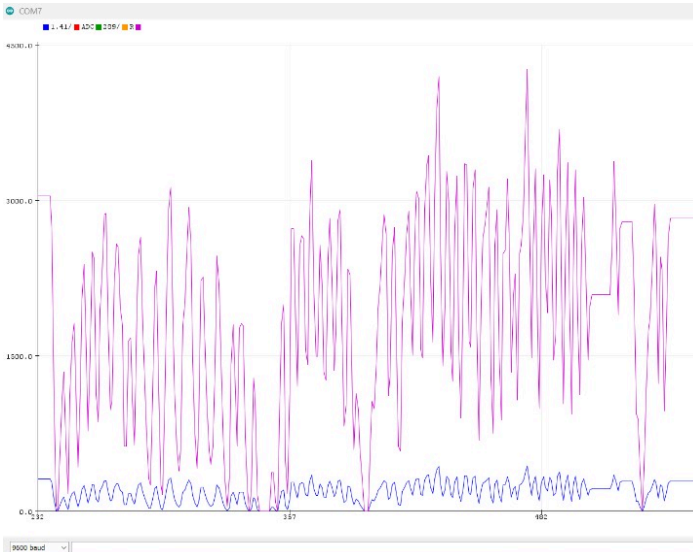

Figure 7.1.2 Shows the Serial Plotter from Arduino IDE

7.2 Task 2

In Task 2, the servo motor's angular motion corresponded precisely to the potentiometer adjustment. The live plot generated in PyCharm reflected smooth servo angle transitions and demonstrated accurate, real-time visualization. As the potentiometer value increased, the servo angle increased proportionally from 0° to 180°, and the LED illuminated when the angle exceeded 90°.

Both the Arduino and PyCharm terminals displayed synchronized readings, confirming correct serial data exchange. The implemented stop command halted the operation safely, demonstrating control reliability and program stability.

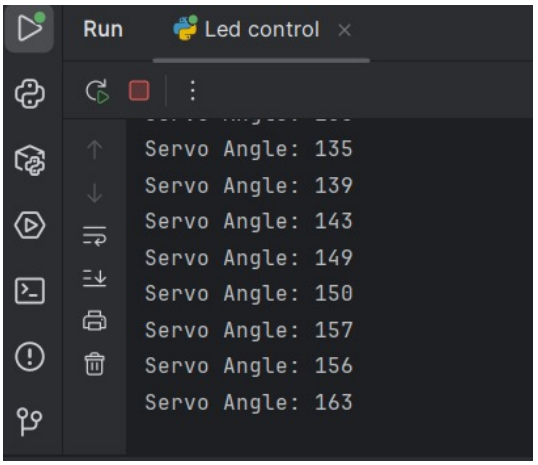

Figure 7.2.1 Shown the servo motor angle at output of PyCharm.

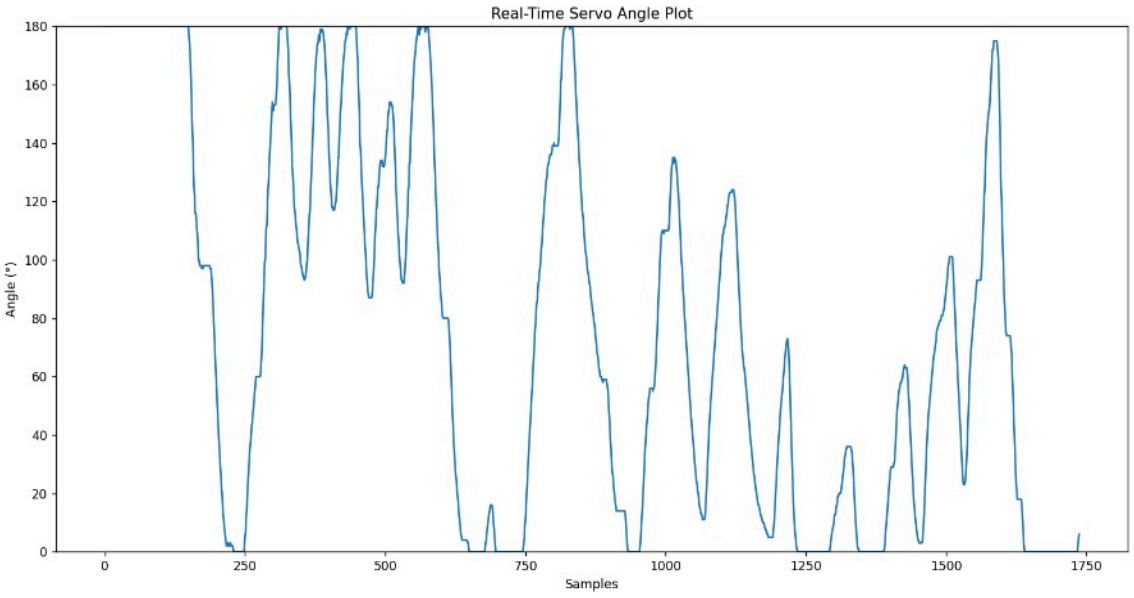

Figure 7.2.2 Shows the Graphical Visualization from PyCharm

8. Discussion

This experiment successfully established serial communication between Arduino and PyCharm and enabled real-time data transfer and hardware control. The two-way communication was demonstrated in Task 1, where Arduino transmitted sensor data and received control commands from PyCharm that will light up the LED, and in Task 2, where servo motor control and live visualization were implemented when the potentiometer is turned and changes the ADC value.

The results confirmed that **serial communication** can be effectively used for microcontrollers–computer interfacing. PyCharm served as a high-level interface for monitoring, control logic, and visualization, while Arduino handled low-level hardware execution.

Noise and timing delays were minimal due to consistent baud rate settings (9600 bps) and buffer handling. The linearity of potentiometer readings validated accurate ADC conversion. The servo's response was smooth and stable, reflecting proper PWM control and power regulation.

A key takeaway from this experiment is the significance of synchronized baud rates and closed serial ports to prevent connection errors. Additionally, using Python's matplotlib for visualization provided immediate feedback, enhancing understanding of real-time control in mechatronic systems.

9. Conclusion

This laboratory experiment successfully demonstrated the integration of hardware (Arduino UNO, potentiometer, LED, servo motor) with software (Python via PyCharm) through serial communication.

In **Task 1**, real-time data transmission between Arduino and Python was achieved, and the LED responded to potentiometer input via threshold logic.
In **Task 2**, servo motor control was established based on potentiometer position, accompanied by real-time plotting and safe program termination through keyboard input.

Overall, the experiment met all objectives:

- Achieved stable serial communication between Arduino and PyCharm.
- Implemented bidirectional data control (Arduino ↔ PyCharm).
- Demonstrated real-time visualization using matplotlib.
- Integrated hardware and software control in a synchronized system.

This exercise strengthened understanding of mechatronic system communication, data handling, and interactive control between physical devices and software platforms.

10. Recommendations

1. Improve Hardware Stability and Power Supply Management

In Task 2, the servo motor relied on the Arduino's 5V output, which may lead to voltage drops when the servo experiences load, potentially causing jitter or delayed response. Using an external regulated power supply or a servo driver module would ensure stable performance and prevent unintended resets or data fluctuations during operation.

2. Enhance Serial Communication Reliability

The communication between Arduino and Python sometimes exhibited minor delays and parsing errors due to continuous data streaming. Implementing structured data transmission (such as using delimiters consistently or sending data in fixed formats) and optimizing the baud rate would reduce communication lag and improve real-time control responsiveness.

3. Optimize Python coding for Better Performance and Control

The Python script continuously cleared and re-plotted the graph, which caused choppy visualization and increased processing load. Implementing more efficient plotting techniques (such as matplotlib animation or data buffering) would provide smoother real-time graphs. Additionally, improving the user interface, for example, adding buttons or control functions in PyCharm, since this would make the system more interactive and reduce dependence on keyboard interrupts.

11. References

Arduino. “Tutorials | Arduino Documentation.” *Docs.arduino.cc*, docs.arduino.cc/tutorials/.

Instructables. “Arduino Servo Motors.” *Instructables*, Instructables, 10 July 2015, www.instructables.com/Arduino-Servo-Motors/.

JetBrains. “PyCharm.” *JetBrains*, JetBrains, 2025, www.jetbrains.com/pycharm/.

Last Minute Engineers. “How Servo Motor Works & Interface It with Arduino.” *Last Minute Engineers*, 20 Oct. 2019, lastminuteengineers.com/servo-motor-arduino-tutorial/.

Python, Real. “Arduino with Python: How to Get Started – Real Python.” *Realpython.com*, realpython.com/arduino-python/.

“Serial Communication between Python and Arduino.” *Projecthub.arduino.cc*, projecthub.arduino.cc/ansh2919/serial-communication-between-python-and-arduino-663756.

Appendix A - Circuit Diagram

Task 1

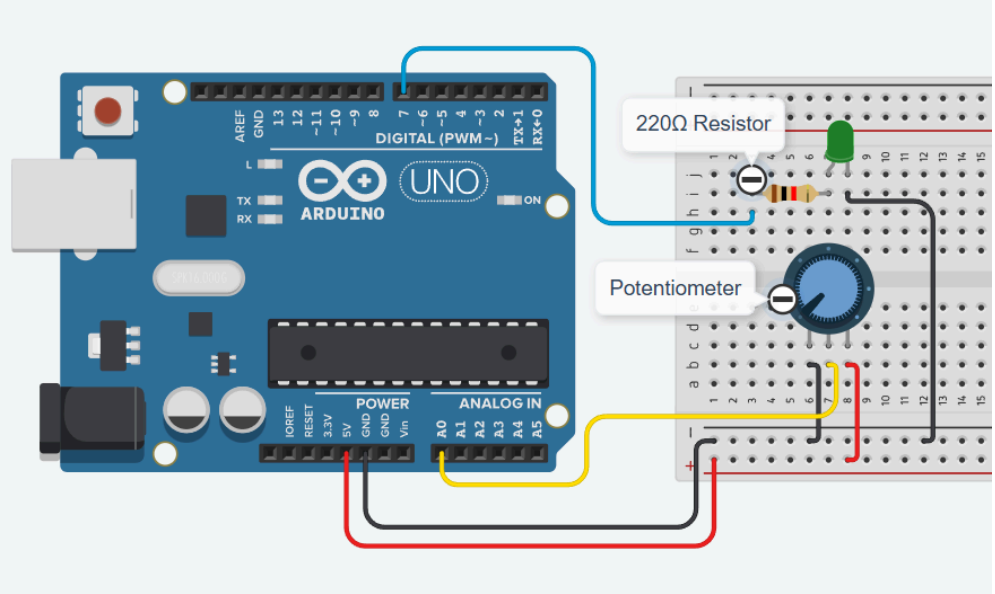

Figure A1 Shows the connection of Task 1 circuit configuration by using Tinkercad

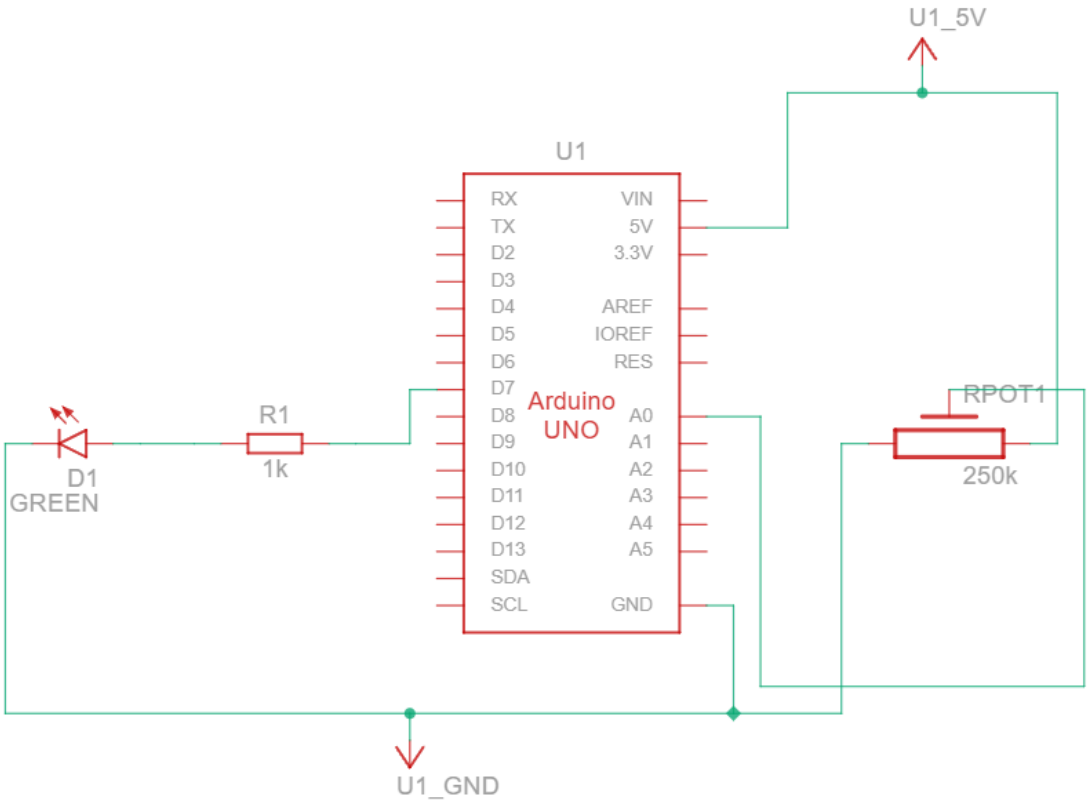

Figure A2 Shows the Task 1 circuit schematic diagram by using Tinkercad

Task 2

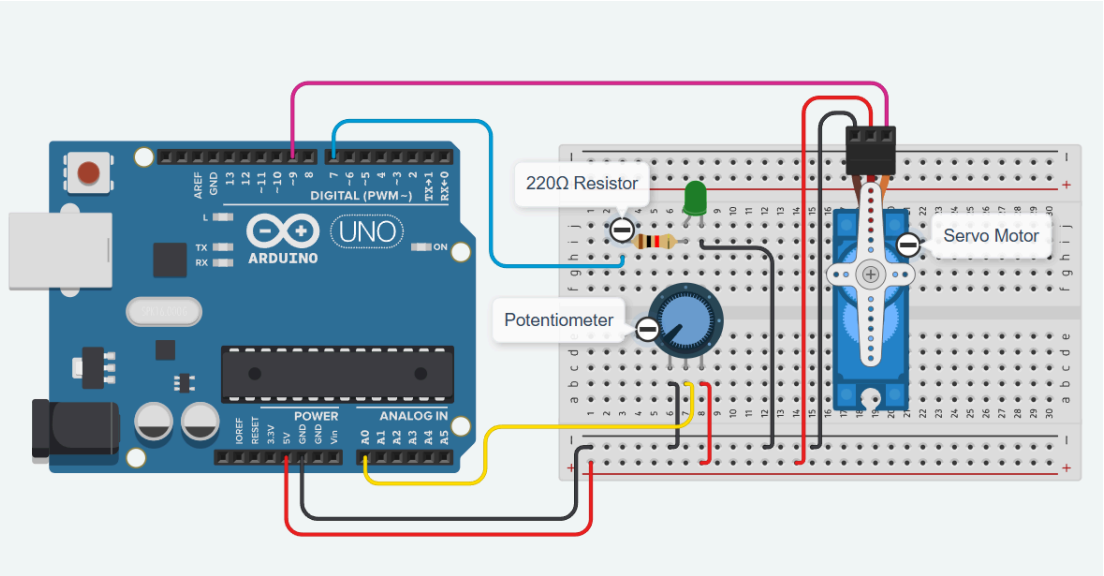

Figure A3 Shows the connection of Task 2 circuit configuration by using Tinkercad

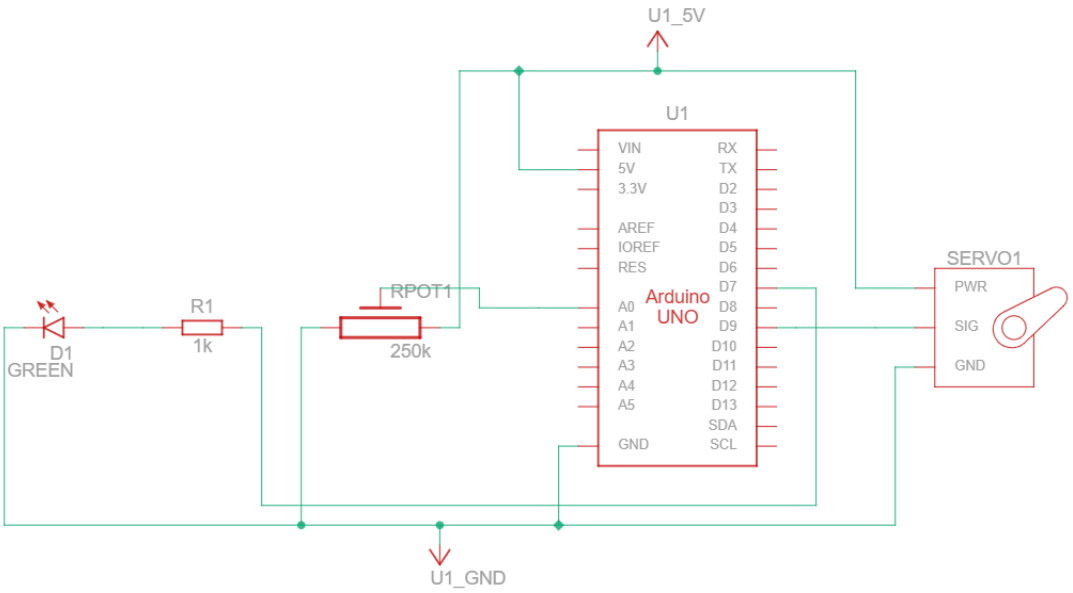

Figure A4 Shows the Task 2 circuit schematic diagram by using Tinkercad

Appendix B - Coding Snippets

Task 1

B1 Arduino IDE Coding

```
const int potPin = A0;
const int ledPin = 7;
const float Vin = 5.0;
const float R_total = 10000;
String command = "";

void setup() {
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
}
void loop() {
  int potValue = analogRead(potPin);
  float Vout = (potValue / 1023.0) * Vin;
  float R_wiper = (Vout / Vin) * R_total;

  // Send data to Python
  Serial.print(Vout);
  Serial.print("/ ADC: ");
  Serial.print(potValue);
  Serial.print("/ R: ");
  Serial.println(R_wiper);
  Serial.println(potValue);

  if (Serial.available() > 0) {
    command = Serial.readStringUntil('\n');
    command.trim();
    if (command == "LED_ON") {
      digitalWrite(ledPin, HIGH);
    }
    else if (command == "LED_OFF") {
      digitalWrite(ledPin, LOW);
    }
  }
  delay(500);
}
```

B2 PyCharm Coding

```
import serial
import time

ser = serial.Serial('COM7', 9600)
time.sleep(2)  # wait for Arduino to reset

print("Running... Press CTRL+C to stop.")

try:
   while True:
       line = ser.readline().decode().strip()

       if "/" in line:
           try:
                #Parse Arduino data
               data = line.split("/")
               voltage = float(data[0])
               adc_value = int(data[1].replace("ADC:",
"").strip())
               resistance = float(data[2].replace("R:",
"").strip())

               print(f"Voltage: {voltage:.2f} V | ADC:
{adc_value} | Resistance: {resistance:.2f} Ω")

               # LED Control Logic
               if adc_value > (1023 / 2):
                   ser.write(b"LED_ON\n")
               else:
                   ser.write(b"LED_OFF\n")
           except:
               print("Parsing Error:", line)

except KeyboardInterrupt:
   print("Stopped by user.")

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

Task 2

B3 Arduino IDE Coding

```
#include <Servo.h>
Servo myservo;

int potPin = A0;
int ledPin = 7;

bool stopControl = false;
int potValue, angle;

void setup() {
  Serial.begin(9600);
  myservo.attach(9);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  if (Serial.available() > 0) {
    char command = Serial.read();

    if (command == 'x') {
      stopControl = true;
      myservo.detach();
      digitalWrite(ledPin, LOW);
      Serial.println("STOP");}
  }
```

```
  if (!stopControl) {
    potValue = analogRead(potPin);
    angle = map(potValue, 0, 1023,
0, 180);

    myservo.write(angle);

    // LED control: ON bila lebih
dari 90°
    digitalWrite(ledPin, angle >
90 ? HIGH : LOW);

    // Hantar data ke Python
    Serial.println(angle);
    delay(50);}
}
```

B4 PyCharm Coding

```
import serial
import matplotlib.pyplot as plt
import time

ser = serial.Serial('COM7', 9600)
time.sleep(2)

plt.ion()
fig, ax = plt.subplots()
x_vals, y_vals = [], []

try:
   while True:
       raw = ser.readline().decode().strip()

       if raw.isdigit():
           angle = int(raw)
           print("Servo Angle:", angle)

           x_vals.append(len(x_vals))
           y_vals.append(angle)

           ax.clear()
           ax.set_ylim(0, 180)
           ax.set_title("Real-Time Servo Angle Plot")
           ax.set_xlabel("Samples")
           ax.set_ylabel("Angle (°)")
           ax.plot(x_vals, y_vals)
           plt.pause(0.05)

except KeyboardInterrupt:
   print("Sending stop command to Arduino...")
   ser.write(b'x')  # Hantar stop signal ke Arduino
   time.sleep(1)

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

Appendix C - Results

Task 1

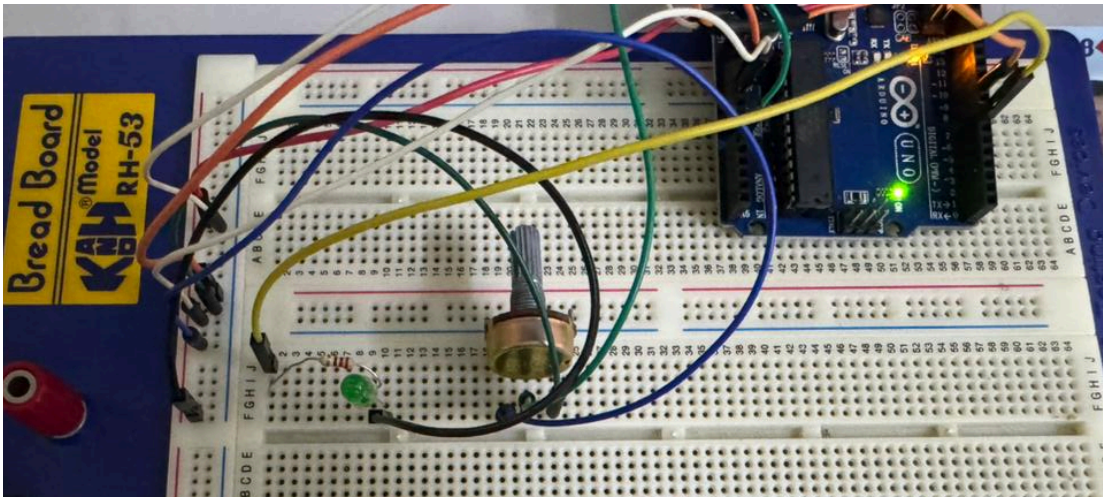

Figure C1 Shows the results before the ADC value below 512

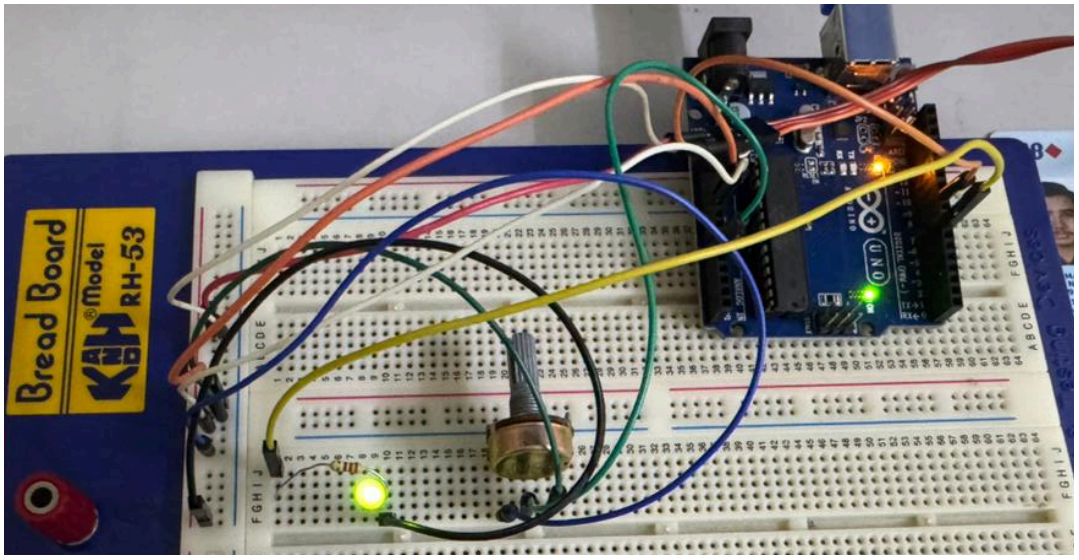

Figure C2 Shows the results after the ADC value above 512

Task 2

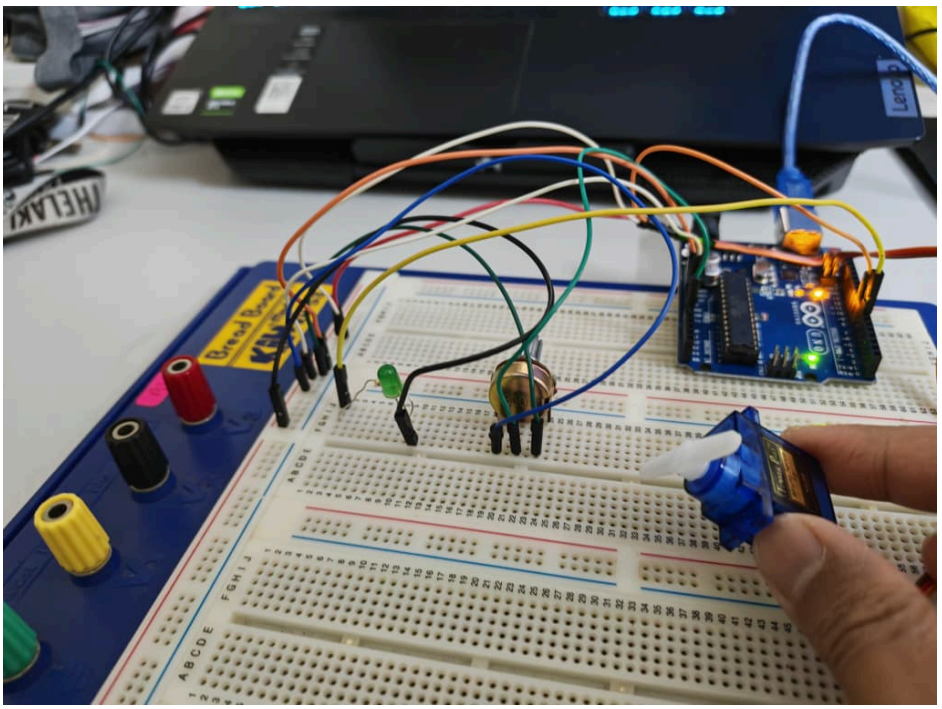

Figure C3 Shows the results before the ADC value below 512

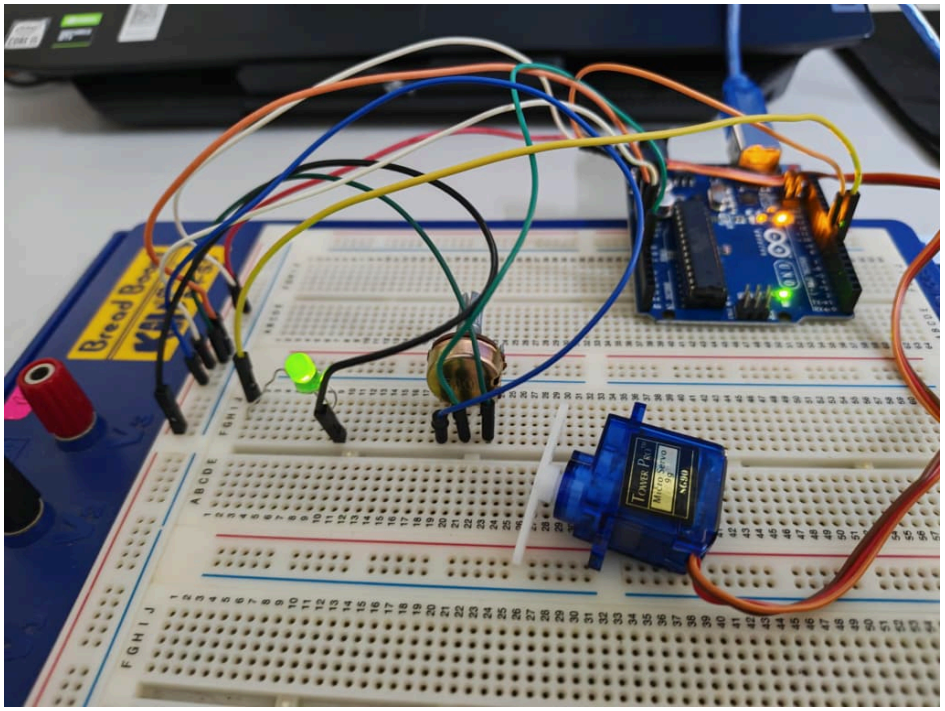

Figure C4 Shows the results after the ADC value above 512

Acknowledgments

We would like to express our sincere gratitude to our lecturer and lab instructor for their continuous guidance, clear explanations, and assistance with experimental setups throughout this experiment. Their detailed instructions, along with the necessary files provided, helped us understand how to properly set up the Arduino UNO microcontroller, PyCharm, and serial communication, and gave us the confidence to carry out the tasks effectively. We also extend our heartfelt appreciation to the lab assistants, whose patience and support in setting up the equipment, troubleshooting circuits, and clarifying procedural steps were instrumental in the smooth progress of the experiment.

Special thanks are also due to our team members, who contributed significantly to every stage of this project, including circuit assembly, coding, data collection, analysis, and documentation. Their collaboration, dedication, and teamwork ensured that each task was completed accurately and efficiently. We also sincerely appreciate our classmates and lab partners for their cooperation, ideas, and teamwork during the experiment. Together, we troubleshot connection issues, shared components, and tested the code to ensure everything functioned as expected.

Overall, the success of this experiment would not have been possible without the guidance and support of both our instructors and lab partners, and we are truly grateful for their contributions.

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	/
	Understand	/
Name: Muhammad Fahim Bin Ahmad Norisham	Agree	/
Matric Number: 2312369		
Signature: 	Read	/
Name: Muhammad Zamarul Azim Bin Zulkhibri	Understand	/
Matric Number: 2312451	Agree	/
Signature: 	Read	/
Name: Muhammad Hazimuddin Bin Fairuz	Understand	/
Matric Number: 2312479	Agree	/