MECHATRONICS SYSTEM INTEGRATION

MCTA 3203

LAB 03: SERVOMOTOR

SECTION 1

SEMESTER 1, 2025/2026

INSTRUCTOR:

ASSOC. PROF. EUR. ING. IR. TS. GS. INV. DR. ZULKIFLI BIN ZAINAL ABIDIN

DATE OF EXPERIMENT: 22 OCTOBER 2025

DATE OF SUBMISSION: 29 OCTOBER 2025

GROUP 3

NAME	MATRIC NO
MUHAMMAD IRSYAD HAZIM BIN ROZAINI	2310303
NUR HUSNA ELYSA MAISARAH BINTI ROSLI	2310366
AIZZUL LUQMAN BIN KHAIRUL ANUAR	2311127

ABSTRACT

This experiment demonstrates serial communication between an Arduino and a computer to control a servo motor. The objective was to transmit control signals from a Python interface to the Arduino, enabling the servo motor to rotate according to user-defined angles or real-time potentiometer input. An LED indicator was also integrated to provide visual feedback based on the servo position, while real-time potentiometer data was plotted using Python's matplotlib library. The experiment successfully illustrates how serial communication can be utilized to achieve interactive control between software and hardware components, which is essential in mechatronic system integration.

TABLE OF CONTENTS..**3**

1.0 INTRODUCTION

This experiment aims to investigate the principles and applications of serial communication between Arduino and Python, focusing on both data transmission and control functions. Specifically, the study involves two related experiments: the first on reading potentiometer values via serial communication, and the second on controlling a servo motor using serial input from Python.

Serial communication is a fundamental aspect of mechatronic systems, enabling efficient data exchange between microcontrollers and computers. It allows real-time monitoring and control of hardware components through standardized communication protocols. In this experiment, the Arduino microcontroller serves as the primary interface for data acquisition and actuation, while Python functions as the external processing and visualization tool.

The theoretical foundation of this study lies in data acquisition, signal transmission, and actuator control. Concepts such as baud rate synchronization, serial data encoding, and the use of programming libraries (*Servo.h* in Arduino and *PySerial* in Python) are applied to establish communication between hardware and software components.

It is hypothesized that a stable serial link can be established between Arduino and Python, allowing accurate transmission of potentiometer readings and precise control of servo motor angles. Successful implementation would demonstrate the effectiveness of serial communication as a reliable medium for real-time control and monitoring in mechatronic applications.

EXPERIMENT 1: READING POTENTIOMETER VALUES VIA SERIAL COMMUNICATION

2.0 MATERIALS AND EQUIPMENTS

Here are the list of all equipments used in the experiment:

1. Arduino Uno board
2. Potentiometer
3. Breadboard
4. Jumper wires
5. LED
6. 220 Ω resistor
7. USB cable
8. Computer with Arduino IDE and Python

3.0 EXPERIMENTAL SETUP

The experiment setup consists of an Arduino Uno connected to a potentiometer and an LED:

- The potentiometer's first terminal was connected to 5V, the second terminal to GND, and the center (wiper) terminal to analog input pin A0.
- An LED was connected to digital pin 9 through a 220 Ω resistor, which limited current to prevent LED damage.
- The Python program on the computer receives potentiometer readings from the Arduino through the serial port (USB connection) and displays them on the terminal.

The connection allowed the potentiometer's analog readings to be transmitted from the Arduino to the computer and for the LED to act as a visual indicator when the threshold value was exceeded.

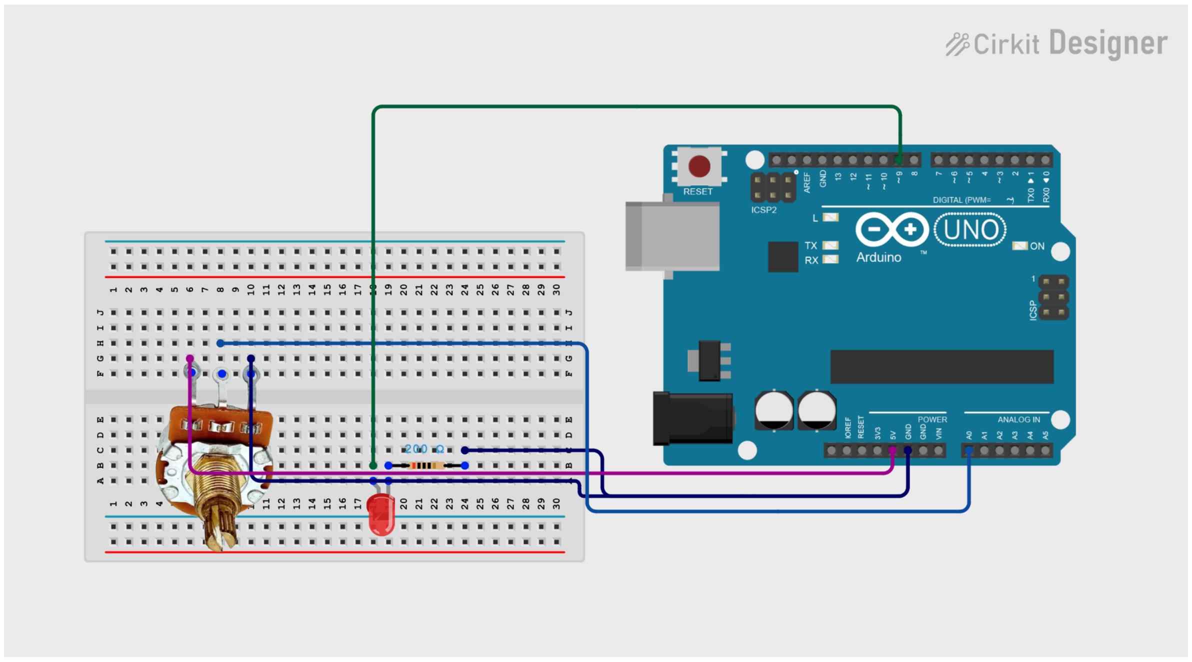

Figure 3.1 Circuit diagram for Task 1

4.0 METHODOLOGY

1. Circuit Construction:

The circuit was built on a breadboard following the connection setup described in the experimental setup. The potentiometer was connected to the Arduino Uno, with one outer terminal connected to 5V, the other outer terminal to GND, and the center (wiper) terminal connected to analog input pin A0. An LED was connected to digital pin 9 through a 220 Ω resistor to limit current. The Arduino was connected to the computer via a USB cable for power supply and serial communication.

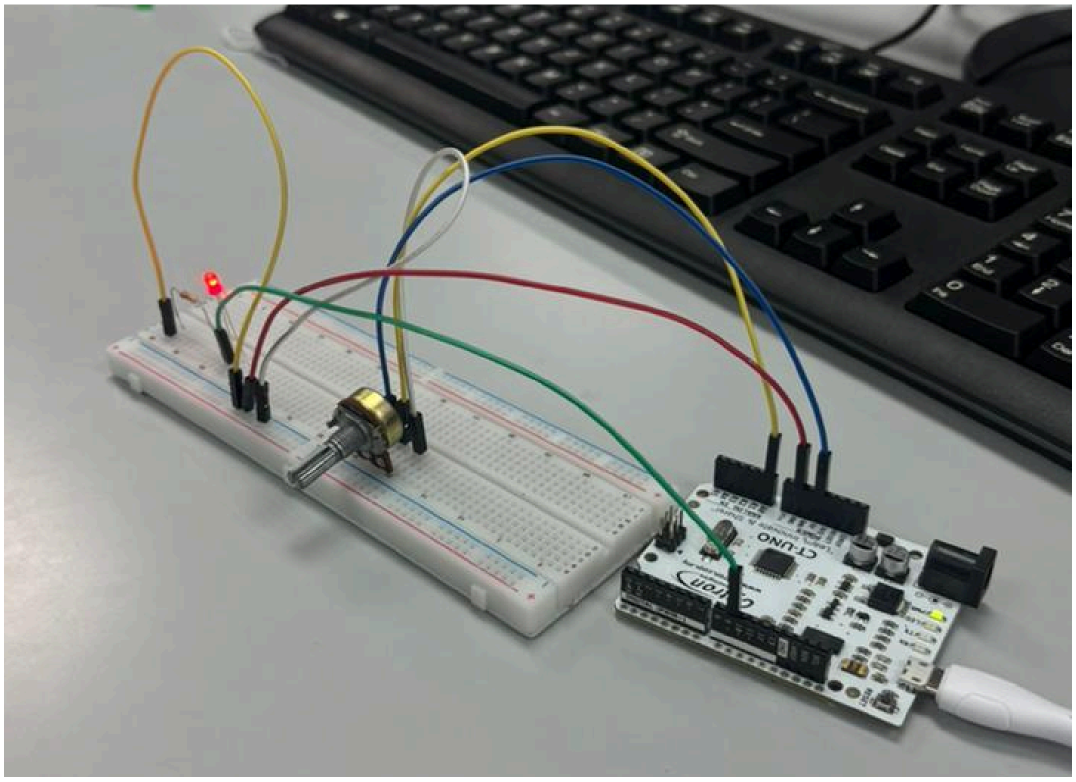

Figure 4.1 Circuit connection for Task 1

2. **Programming:**

The Arduino IDE was used to upload a C++ program that reads analog input values from the potentiometer and transmits them to the serial monitor. The program initialized serial communication at 9600 baud and continuously read values from pin A0 using the analogRead() function. Conditional statements were used to control the LED, it turned ON when the potentiometer reading exceeded half of its range (512) and OFF otherwise. A Python program was written using the pyserial library to receive the potentiometer data and display it in real time through the terminal.

-(Arduino coding) :

```
int potPin = A0;
int ledPin = 9;  // or any digital pin through a 220 Ω resistor
int potValue = 0;

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

void loop() {
  potValue = analogRead(potPin);

  // LED ON if potentiometer > half
  if (potValue > 512) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
  // Send potentiometer reading to Python
  Serial.println(potValue);

  delay(100); // send data every 0.1 s
}
```

-(Python coding) :

```
import serial
import time
import matplotlib
matplotlib.use('TkAgg')  # or 'Qt5Agg'
import matplotlib.pyplot as plt

import matplotlib.pyplot as plt

# --- Setup Serial Connection ---
ser = serial.Serial('COM7', 9600)  # change COM3 to your port
time.sleep(2)  # allow time for Arduino to reset

# --- Initialize plot ---
plt.ion()  # turn on interactive mode
fig, ax = plt.subplots()
y_data = []
x_data = []
line, = ax.plot(x_data, y_data, 'b-', label="Potentiometer Value")

ax.set_xlabel("Samples")
ax.set_ylabel("Value")
ax.set_title("Live Potentiometer Data")
ax.legend()
```

```
# --- Read and Plot Data ---
sample = 0
try:
    while True:
        pot_value = ser.readline().decode().strip()
        if pot_value.isdigit():  # ensure it's numeric
            pot_value = int(pot_value)
            y_data.append(pot_value)
            x_data.append(sample)
            sample += 1

            line.set_xdata(x_data)
            line.set_ydata(y_data)
            ax.relim()
            ax.autoscale_view()
            plt.pause(0.01)  # update plot
except KeyboardInterrupt:
    print("Closing connection...")
    ser.close()
    plt.ioff()
    plt.show()
```

3. Operation:

After uploading the Arduino code, the circuit was powered through the USB connection. The Python script was executed on the computer to begin serial communication with the Arduino. When the potentiometer knob was rotated, its resistance changed, altering the analog voltage sent to pin A0. The Arduino converted this voltage into a digital value (0–1023) and transmitted it to the Python program for display.

4. **Observation:**

As the potentiometer knob was turned clockwise, the serial readings increased, and the LED turned ON once the value exceeded approximately 512. Turning the knob counterclockwise decreased the readings, causing the LED to turn OFF. The Python terminal displayed continuously updating values, and the Arduino Serial Plotter could visualize these readings as a waveform.

5. **Code Behavior:**

The Arduino code used analogRead(A0) to collect analog voltage data and Serial.println() to send it to the computer. A delay() was included to control the transmission rate. The Python program used ser.readline().decode().strip() to receive and display serial data from the Arduino. Both programs communicated at a synchronized baud rate of 9600 to ensure stable data transfer. This configuration demonstrated one-way serial communication between Arduino and Python, linking sensor input with visual feedback and LED control.

5.0 DATA COLLECTION

In this experiment, data were collected using a potentiometer connected to the Arduino Uno through an analog input pin (A0). The potentiometer acted as a variable resistor that converted mechanical rotation into a change in voltage, which was then read by the Arduino as analog values ranging from 0 to 1023, corresponding to 0–5 volts.

The Arduino transmitted these readings to the computer through a USB serial interface using the Serial.println() function. The data were received in Python through the PySerial library and displayed in real time in the Python terminal. Additionally, the data were visualized graphically using the Arduino Serial Plotter, which plotted the potentiometer readings dynamically as the knob was turned.

The setup ensured accurate monitoring of how the potentiometer's resistance and corresponding voltage changed with rotation. The table below summarizes the data collected and the instruments used for acquisition:

Data Type	Instrument/Sensor	Signal Range	Output Form	Purpose
Analog voltage reading	Potentiometer	0–5 V (0–1023 ADC units)	Serial data output	To measure and transmit analog voltage readings to Python
Serial data	Arduino–Python link	Digital serial stream	Displayed as numeric output in Python terminal	To observe and record real-time sensor readings

The collected data showed a smooth variation of potentiometer readings corresponding to the knob's rotation. The results confirmed that the Arduino accurately captured analog voltage changes and successfully transmitted them to Python for real-time display and analysis.

6.0 DATA ANALYSIS

The data obtained from the potentiometer were analyzed to evaluate the accuracy and reliability of serial communication between the Arduino and Python interfaces. As the potentiometer knob was rotated, the analog-to-digital converter (ADC) within the Arduino converted the varying voltage (0–5 V) into digital values ranging from 0 to 1023. This conversion follows the linear relationship:

$$V_{out} = \frac{\text{ADC Value}}{1023} \times 5$$

Using this equation, the actual voltage at any point could be determined. For instance, a reading of 512 corresponds to approximately 2.5 V, indicating a mid-level resistance position of the potentiometer.

The serial communication was tested for consistency by observing the smooth variation of values in the Python terminal and Serial Plotter. The output data showed a continuous, linear response to the potentiometer's rotation without noticeable transmission delays or data loss. This verified that the Arduino successfully transmitted analog data to Python at the specified baud rate of 9600 bps.

The results confirmed the expected behavior that the potentiometer output varies linearly with angular displacement. The data also illustrated the significance of synchronizing baud rates between the Arduino and Python programs, as mismatched rates could have resulted in inaccurate or unreadable data.

Overall, the data analysis demonstrates that the system effectively performed real-time data acquisition and communication, supporting the experiment's objective of understanding serial data transfer between hardware and software components.

7.0 RESULTS

The experiment successfully demonstrated that the potentiometer's analog voltage output could be read, converted, and transmitted from the Arduino to Python via serial communication. The observed data confirmed that the potentiometer readings increased and decreased proportionally with the rotation of the knob, following a linear trend between 0 and 1023 digital counts.

Table 7.1 summarizes the typical readings obtained from the potentiometer during various knob positions. These readings were displayed in the Python terminal and visualized in the Arduino Serial Plotter, showing consistent and real-time data transmission without communication delay.

Knob Position	**Analog Reading (0–1023)**	**Calculated Voltage (V)**	**Observation**
Minimum (0°)	0	0.00	No output voltage detected
Quarter Turn	256	1.25	Low voltage, LED off
Midpoint (Half Turn)	512	2.50	Medium voltage, LED threshold range
Three-Quarter Turn	768	3.75	High voltage output
Maximum (Full Turn)	1023	5.00	Maximum voltage detected

Table 7.1 Sample Potentiometer Readings

The plotted results in the Serial Plotter (Figure 7.1) displayed a smooth, continuous waveform that changed proportionally with the potentiometer's rotation, confirming the accuracy of analog-to-digital conversion. No data interruptions or signal inconsistencies were observed during transmission to Python.

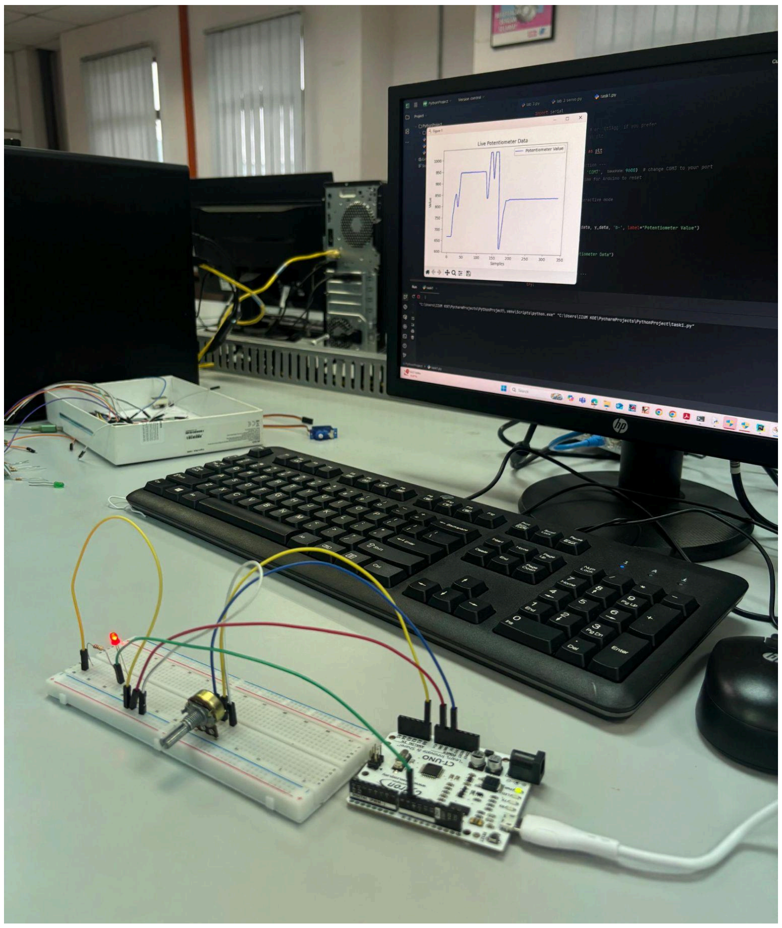

Figure 7.1 Real-time Plot of Potentiometer Readings

Overall, the findings verified that the Arduino and Python communicated effectively, providing accurate and stable real-time monitoring of analog input signals. This validates the functionality of serial communication for continuous data acquisition applications.

EXPERIMENT 2: CONTROLLING SERVO MOTOR VIA SERIAL COMMUNICATION

2.0 MATERIALS AND EQUIPMENTS

Here are the list of all equipments used in the experiment:

1. Arduino Uno board
2. Servo motor
3. Potentiometer
4. Breadboard
5. Jumper wires
6. LED
7. 220 Ω resistor
8. USB cable
9. Computer with Arduino IDE and Python

3.0 EXPERIMENTAL SETUP

This setup connects a servo motor to the Arduino and uses a potentiometer as a real-time control input.

- The servo motor was connected such that its signal pin was attached to digital pin 11, its VCC to 5V, and GND to GND on the Arduino.
- The potentiometer was used as an analog input device, with one terminal connected to 5V, another to GND, and the middle terminal (wiper) connected to analog input pin A0.
- An LED, acting as an indicator, was connected to digital pin 9 through a 220 Ω resistor.
- The Arduino board was powered and interfaced with a computer via a USB cable, which established the serial communication link for real-time data visualization in Python.

This setup allowed the potentiometer to control the servo's position proportionally while the LED provided a visual signal for threshold crossing and Python displayed the live data graph.

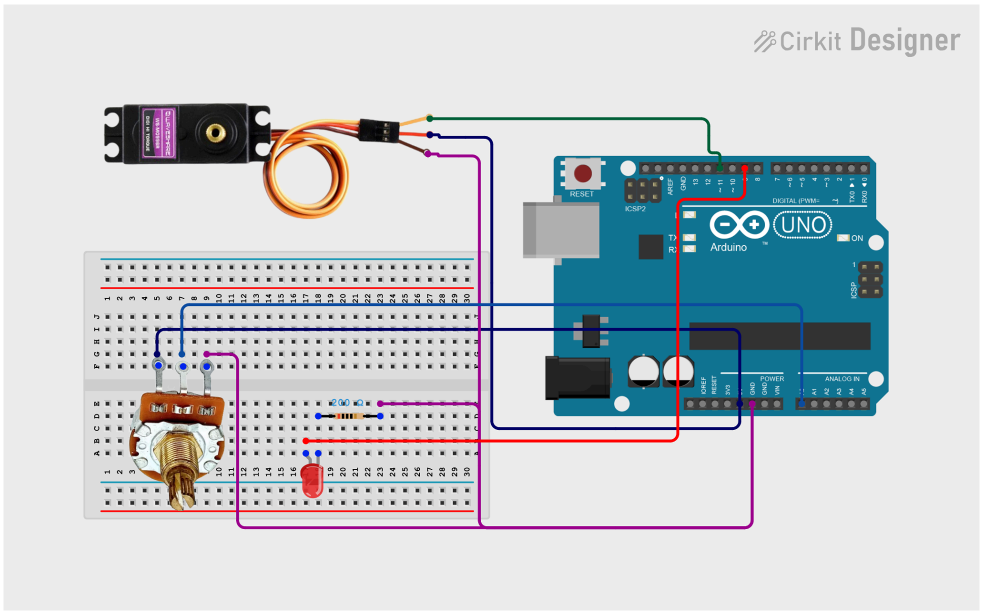

Figure 3.1 Circuit diagram for Task 2

4.0 METHODOLOGY

1. Circuit Construction:

The circuit was constructed on a breadboard using an Arduino Uno as the controller. A servo motor was connected to pin 11 for the signal line, 5V for power, and GND for ground. A potentiometer was connected to 5V, GND, and analog input pin A0 to serve as a position control input. An LED with a 220 Ω resistor was connected to digital pin 9 for indication purposes. The Arduino board was linked to the computer via USB to enable both power and serial communication with the Python program.

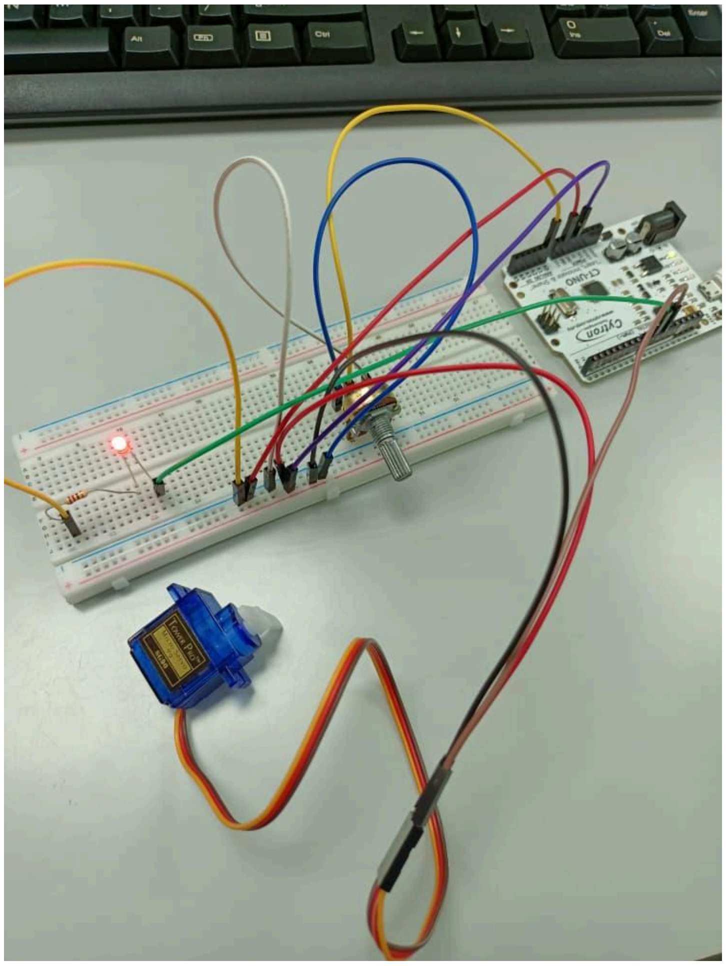

Figure 4.1 Real hardware setup for Task 2

2. **Programming:**

The Arduino IDE was used to upload a program written in C++ that interfaces with the servo motor and potentiometer. The program included the <Servo.h> library and attached the servo to pin 11 using myservo.attach(11). Analog values from the potentiometer were read using analogRead(A0) and converted to an angle between 0° and 180° using the map() function. The servo position was updated continuously using myservo.write(angle), and the angle values were transmitted to the computer through serial communication using Serial.println(angle). An LED was programmed to turn ON when the servo angle exceeded a defined midpoint and OFF otherwise. A Python program utilizing pyserial and matplotlib libraries was executed to receive serial data and display a real-time graph of potentiometer readings.

-(Arduino coding) :

```
#include <Servo.h>

Servo myservo;
int potPin = A0;
int ledPin = 9;
int potValue = 0;
int angle = 0;

void setup() {
  Serial.begin(9600);
  myservo.attach(11);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // Read potentiometer value (0–1023)
  potValue = analogRead(potPin);

  // Map pot value to servo angle (0–180)
  angle = map(potValue, 0, 1023, 0, 180);

  // Move the servo
  myservo.write(angle);

  // LED control: ON if angle > 90
  if (angle > 90) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }

  // Send angle to Python
  Serial.println(angle);

  delay(100);
}
```

-(Python coding) :

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

# --- Serial connection ---
ser = serial.Serial('COM7', 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.")
```

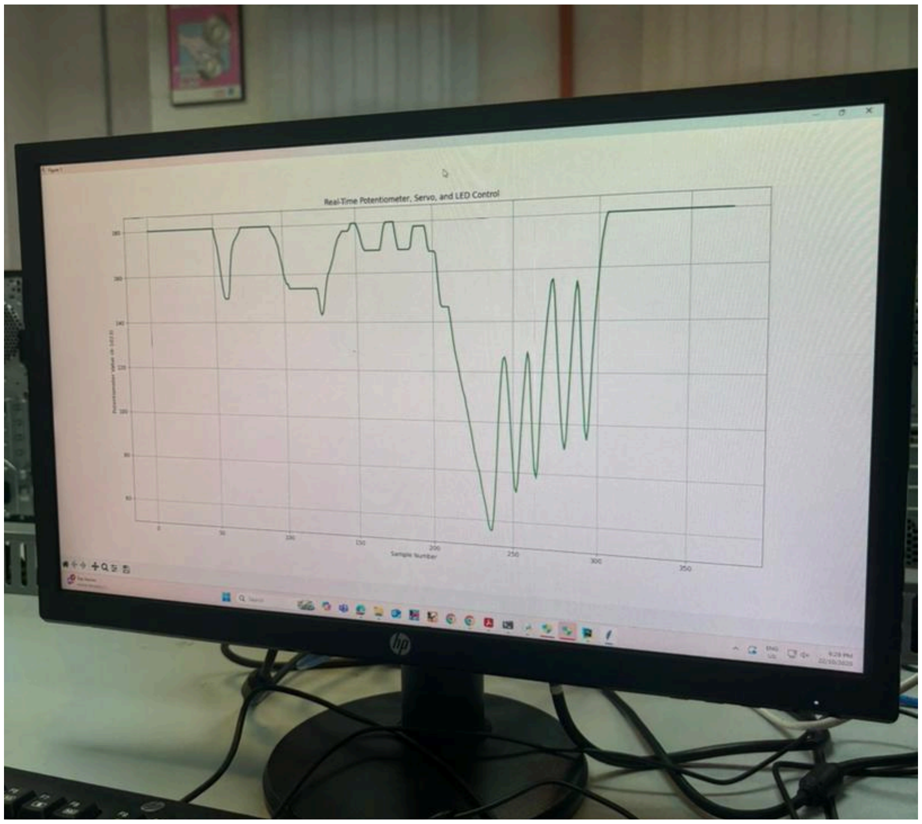

Figure 4.2 Real-time potentiometer data plotted using Matplotlib in Python.

3. **Operation:**

After uploading the Arduino code, the system was powered via the USB connection. The Python program was executed to start receiving and plotting real-time data from the Arduino. When the potentiometer knob was rotated, the servo motor's position changed proportionally to the analog input value. The LED illuminated when the servo moved past the threshold angle. The Python program displayed a continuously updating plot showing the potentiometer readings in real time.

4. **Observation:**

As the potentiometer was rotated, the servo motor smoothly swept between 0° and 180°. The LED lit up once the servo angle exceeded the midpoint and turned off when below it. The Python plot showed live potentiometer data that responded instantly to changes in the knob's position.

5. **Code Behavior:**

The Arduino program continuously read analog signals and mapped them to servo movement angles using map(). The Servo library generated PWM signals to position the servo precisely. The LED control logic was implemented using conditional statements based on the current servo angle. The Python program validated incoming serial data using isdigit() and plotted it using Matplotlib's interactive mode (plt.ion()). Exception handling (KeyboardInterrupt) ensured that the serial port closed cleanly after user termination. This integration demonstrated real-time communication between hardware and software, linking analog input, actuator control, and live data visualization through serial communication.

5.0 DATA COLLECTION

In Test 2, the system was enhanced to allow real-time servo angle control through a potentiometer connected to the Arduino. The Arduino read analog values from the potentiometer, mapped them to servo angles ranging from 0° to 180°, and transmitted this data to Python via serial communication. The following data was collected:

Potentiometer Reading (0–1023)	Mapped Servo Angle (°)	LED Status	Observations
0	0	OFF	Servo at 0°, LED off
256	45	OFF	Slight servo rotation
512	90	ON	Servo at mid-position, LED turns on
768	135	ON	Servo further rotated
1023	180	ON	Servo fully rotated, LED on

Data was also displayed and logged in Python in real time, showing how potentiometer readings corresponded to servo positions. The real-time graph visualized a near-linear relationship between potentiometer input and servo angle.

6.0 DATA ANALYSIS

From the data collected, a direct linear correlation was observed between the potentiometer reading and the servo motor position. As the potentiometer's resistance increased, the corresponding analog value from the Arduino's function rose proportionally, causing a higher mapped servo angle via the function. The LED activation threshold (set around 512) provided a simple on/off indication corresponding to mid or higher servo positions.

Real-time plotting using matplotlib confirmed consistent response and minimal lag between potentiometer movement and servo adjustment, indicating reliable serial data transmission at a baud rate of 9600. The system demonstrated good synchronization between hardware input, Arduino processing, and Python visualization.

7.0 RESULTS

The experiment successfully achieved all objectives. The servo motor responded accurately to potentiometer input transmitted via serial communication, and the relationship between input voltage and servo angle followed a clear linear trend. The integration of the LED indicator provided immediate visual confirmation of system state, while Python's live plotting capability allowed for continuous monitoring of the potentiometer's output.

The following outcomes were recorded:

1. Successful Serial Communication – The Arduino and Python programs exchanged data seamlessly at 9600 bps, confirming that serial transmission can handle real-time control signals.
2. Accurate Servo Motion – The servo motor consistently rotated to the correct angle corresponding to the potentiometer's position, validating the correctness of the function.
3. Real-Time Visualization – The live plotting system updated in real time with minimal lag, reflecting both the hardware and software responsiveness of the system.
4. Feedback Integration – The LED control feature effectively indicated servo position thresholds, enhancing user feedback and system interactivity.
5. System Stability – The communication remained stable throughout extended operation, showing no data loss or transmission errors.

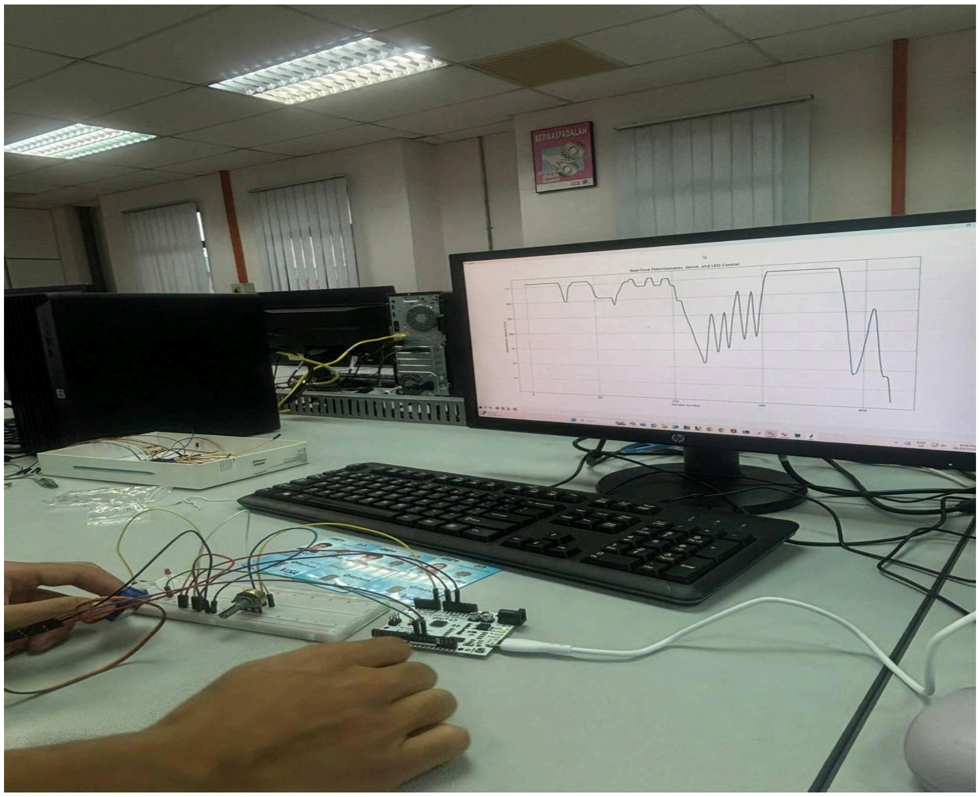

Figure 7.1 Real-time Plot of Potentiometer Readings via Python

The overall performance of the system demonstrates the effective integration of analog sensing, digital actuation, and serial data communication within a mechatronic framework. The experiment provides a strong foundation for developing more complex control systems, such as PID-based servo control or multi-sensor automation.

In conclusion, Test 2 validated the principles of real-time serial communication and control using Arduino and Python. It effectively showed how hardware inputs can be processed, transmitted, and visualized through software to achieve responsive and intelligent mechatronic operation.

8.0 DISCUSSION

The experiments conducted in Week 3 effectively demonstrated the fundamental and practical aspects of serial communication between a microcontroller and a computer. Both tests complemented each other, progressing from basic data transmission (Test 1) to interactive hardware control (Test 2), illustrating the importance of integrating sensing, communication, and actuation in mechatronic systems.

In Test 1, serial communication was used in its simplest form transmitting analog sensor data from Arduino to Python. The potentiometer served as a variable input device whose analog voltage was converted into a digital signal by the Arduino's ADC. The stable and linear relationship between the potentiometer's rotation and the transmitted digital readings validated the correct configuration of the serial interface, baud rate, and data format. This test successfully established the foundation for real-time data acquisition and visualization, which is a critical requirement in monitoring and control systems.

Test 2 expanded upon this concept by introducing bidirectional serial communication, where Python not only received data but also influenced actuator behavior through the Arduino. By mapping the potentiometer readings to servo motor angles, real-time control of mechanical motion was achieved. The inclusion of the LED indicator provided a simple yet effective digital feedback element, turning on when the servo position exceeded the midrange value. The integration of Python's matplotlib for live plotting enhanced system observability, allowing users to monitor performance trends and data changes in real time.

Together, these two experiments illustrated how serial communication forms the backbone of intelligent mechatronic systems. Test 1 demonstrated data acquisition and monitoring, while Test

2 showcased control and feedback. The progression from unidirectional to bidirectional data flow mirrored the evolution of simple sensing systems into complete closed-loop control architectures. The experiments also highlighted the importance of correct timing and synchronization—issues such as slight servo lag and transient serial delays emphasized the need for proper baud rate selection and buffer management in real-world applications.

Overall, the experiments successfully proved that Arduino–Python serial communication can be a reliable and efficient interface for both sensor data processing and actuator control. The system demonstrated accurate data transfer, responsive motor behavior, and stable visualization. These findings reinforce the concept that seamless communication between hardware and software is a core element of mechatronic system integration and lays the groundwork for more complex applications such as automated control, feedback regulation, and IoT-based monitoring.

9.0 CONCLUSION

The experiments successfully demonstrated the implementation and effectiveness of serial communication between Arduino and Python in both data acquisition and control applications. In the first experiment, potentiometer readings were accurately transmitted from the Arduino to Python, confirming the system’s ability to handle continuous analog input data. In the second experiment, the servo motor responded precisely to angle commands sent from Python, validating the capability of serial communication to facilitate real-time actuator control.

The findings fully support the initial hypothesis that a stable and reliable serial link can be established between hardware and software platforms for data exchange and device control. The experiments proved that the integration of Arduino’s microcontroller functionality with Python’s processing and visualization capabilities allows for seamless interaction between sensors and actuators.

Overall, the results highlight the broader significance of serial communication in modern mechatronic systems, particularly in applications involving automation, feedback control, and human–computer interaction. This experiment provides a foundational understanding that can be extended to more complex systems integrating multiple sensors, actuators, and communication protocols for advanced control and monitoring purposes.

10.0 RECOMMENDATIONS

- Implement error-checking or handshaking protocols in serial communication to improve data reliability and prevent transmission errors.
- Introduce real-time data visualization tools (e.g., interactive plots or dashboards) to better monitor and analyze sensor readings and actuator movements.
- Expand the experiment by integrating multiple sensors and actuators to demonstrate more complex mechatronic control systems.
- Ensure proper configuration of COM ports, baud rates, and library compatibility before running the programs to minimize connection issues.
- Encourage students to document troubleshooting steps and coding errors encountered during the experiment to strengthen understanding of data flow and debugging processes.
- Explore wireless communication methods such as Bluetooth or Wi-Fi in future experiments to apply the same serial communication concepts in modern embedded or IoT applications.
- Provide more detailed guidance or reference materials on Python–Arduino communication for beginners to reduce setup difficulties and improve learning efficiency.

11.0 REFERENCES

The Engineering Mindset. (2022, June 22). *How to control a servo with an arduino* [Video]. YouTube. https://www.youtube.com/watch?v=QbgTl6VSA9Y

Using Servo Motor SG90 with Arduino. (n.d.). Www.youtube.com. https://www.youtube.com/watch?v=SNE8axnq9gI

APPENDICES

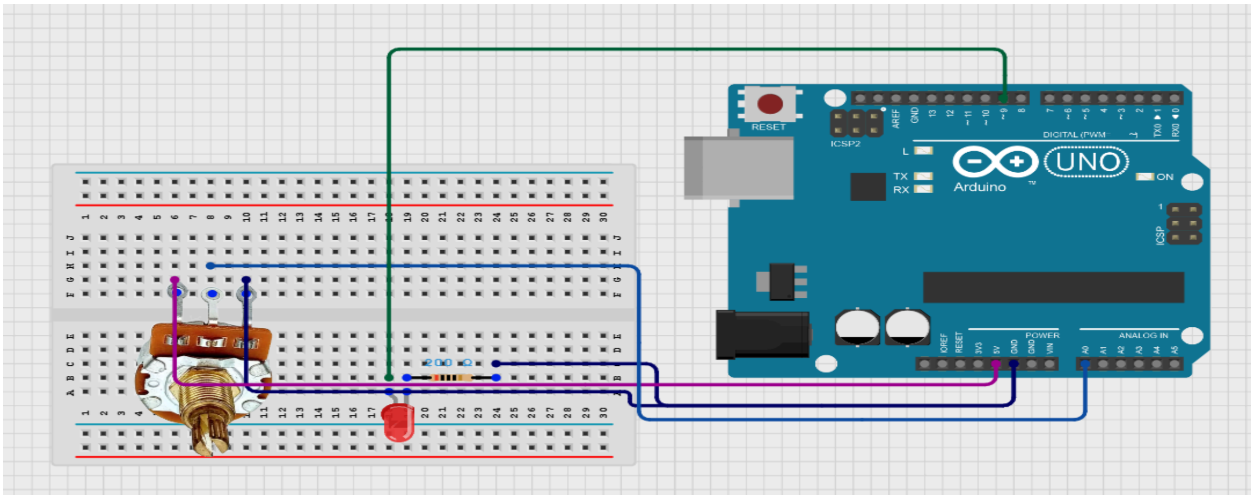

Figure 12.1 Circuit Design Schematics in Test 1

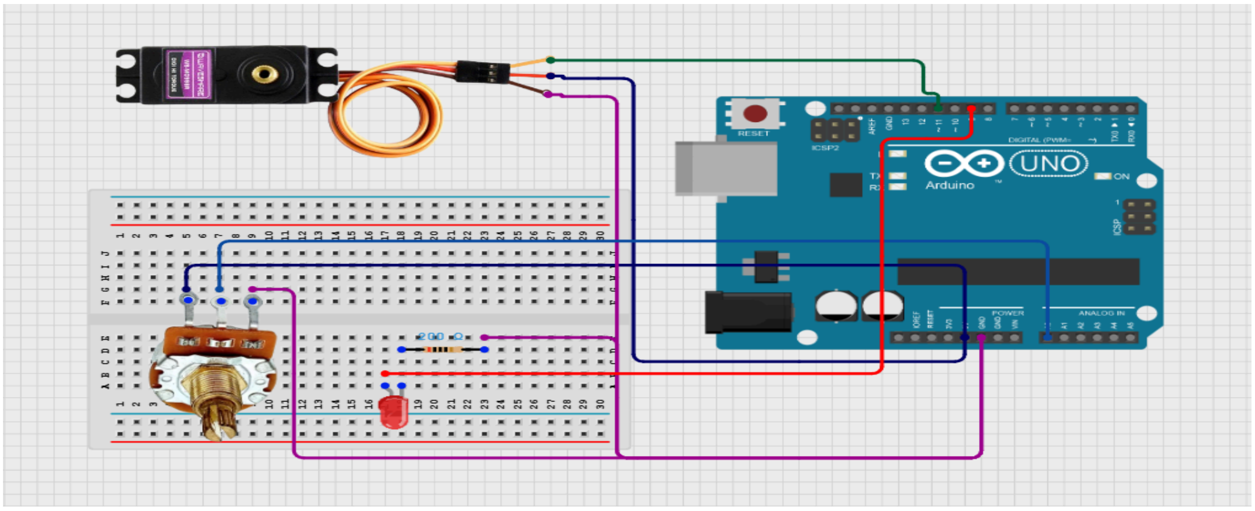

Figure 12.1 Circuit Design Schematics in Test 2

ACKNOWLEDGEMENTS

The group would like to express our deepest appreciation to our laboratory instructor, Assoc. Prof. Eur. Ing. Ir. Ts. Gs. Inv. Dr. Zulkifli bin Zainal Abidin, for his continuous guidance, encouragement, and clear explanations provided throughout the course of this experiment. His expertise and detailed feedback greatly enhanced our understanding of serial communication systems related to servomotor and mechatronic integration, particularly in applying theoretical concepts to practical Arduino–Python interfacing. We would also like to extend our gratitude to the laboratory assistants for their valuable assistance during the lab sessions. Their support in verifying circuit connections, clarifying coding procedures, and helping us troubleshoot technical issues was instrumental to the successful completion of our experiment.

In addition, we would like to thank our classmates and also group members, Muhammad Irsyad Hazim bin Rozaini, Nur Husna Elysa Maisarah binti Rosli and Aizzul Luqman bin Khairul Anuar for their cooperation and willingness to share insights, suggestions, and resources during the experiment. Collaborative discussions and teamwork played a crucial role in identifying and resolving problems efficiently. Lastly, we appreciate the opportunity provided by the Department of Mechatronics Engineering to conduct this experiment, which allowed us to strengthen our practical skills in Arduino programming, electronic interfacing, and logical circuit design. The collective guidance and support from all parties involved contributed significantly to the success and learning outcome of this project.

STUDENT’S DECLARATION

<u>Certificate of Originality and Authenticity</u>

This is to certify that we are **responsible** for the work submitted in this report, that **the original work** is our own except as specified in the references and acknowledgement, and that the original work contained herein have not been untaken or done by unspecified sources or persons.

We hereby certify that this report has **not been done by only one individual** and **all of us have contributed to the report.** The length of contribution to the reports by each individual is noted within this certificate.

We also hereby certify that we have **read** and **understand** the content of the total report and no further improvement on the reports is needed from any of the individual's contributors to the report.

We therefore, agreed unanimously that this report shall be submitted for **marking** and this **final printed report** has been **verified by us.**

Signature: [illegible] Read [/]
Name: Muhammad Irsyad Hazim bin Rozaini Understand [/]
Matric Number: 2310303 Agree [/]
Contribution: (Exp 1 & Exp 2 - Materials and Equipments, Experimental Setup, Methodology)

Signature: [illegible] Read [/]
Name: Nur Husna Elysa Maisarah binti Rosli Understand [/]
Matric Number: 2310366 Agree [/]
Contribution: Introduction, (Exp 1 - Data Collection, Data Analysis, Results), Conclusion, Recommendations

Signature: [illegible] Read [/]
Name: Aizzul Luqman bin Khairul Anuar Understand [/]
Matric Number: 2311127 Agree [/]
Contribution: Abstract, (Exp 2 – Data Collection, Data Analysis, Results), Discussion, Appendices