MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1 2025/2026

WEEK 3: SERIAL COMMUNICATION

SECTION 1

GROUP 9

LECTURER: ZULKIFLI BIN ZAINAL ABIDIN & WAHJU SEDIONO

NO.	GROUP MEMBERS	MATRIC NO.
1.	AHMAD FIRDAUS HUSAINI BIN HASAN AL-BANNA	2315377
2.	ADIB ZAFFRY BIN MOHD ABDUL RADZI	2315149
3.	UMAR FAROUQI BIN ABU HISHAM	2314995

Date of Submission: 29 October 2025

ABSTRACT

This experiment explores the practical implementation of serial communication between an Arduino microcontroller and a computer interface using Python programming for both data acquisition and device control. The study was divided into two main parts: the first focused on reading analog input from a potentiometer through the Arduino's serial interface and displaying the corresponding sensor data on a Python console; the second experiment emphasized servo motor actuation, where the motor's angular position was controlled based on user-defined input commands transmitted from Python to Arduino.

To extend system functionality, several enhancements were integrated into the setup. These included real-time servo control driven by potentiometer feedback, LED indication to represent signal thresholds, and live graphical visualization of sensor readings using the Matplotlib library. The experiment demonstrates the seamless integration of hardware and software, illustrating how serial communication can serve as a bridge between embedded systems and desktop applications. Ultimately, this work highlights the potential of combining Arduino and Python to achieve interactive real-time monitoring, automated control, and dynamic feedback in mechatronic and data-driven systems.

TABLE OF CONTENTS

1.0 INTRODUCTION

Serial communication is one of the most fundamental and widely used methods for exchanging data between microcontrollers and computers. It works by transmitting information one bit at a time over a communication channel, making it a straightforward and dependable way to connect various electronic devices. This method is especially useful in embedded systems and mechatronics, where simplicity and reliability are key for real-time data exchange and control.

In this experiment, we use an Arduino board to interface with a potentiometer—a type of variable resistor that adjusts voltage based on its position. As the user turns the knob, the potentiometer outputs a varying analog voltage. The Arduino reads this voltage using its built-in Analog-to-Digital Converter (ADC), which transforms the analog signal into a digital value that can be processed by software.

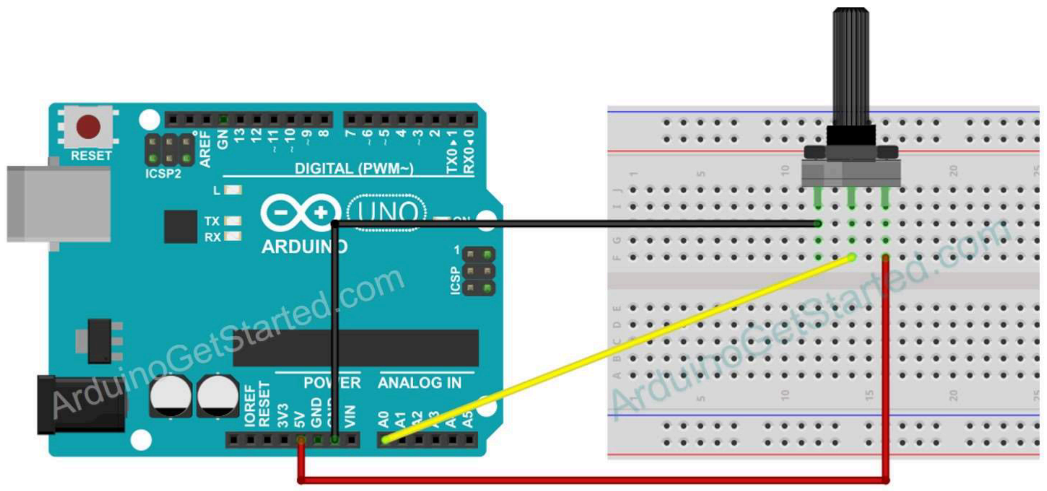
Once the digital data is obtained, the Arduino sends it through a serial port to a Python program running on a computer. Python acts as the receiving end of the communication, capturing the incoming data stream, processing the values, and displaying them in real time. This allows users to visually monitor changes in the potentiometer's position as they occur, providing immediate feedback and insight into the system's behavior.

This setup beautifully demonstrates the integration of hardware and software in modern mechatronic systems. It highlights how microcontrollers can work in tandem with high-level programming languages to create interactive, responsive applications. Whether for prototyping, educational purposes, or industrial automation, this kind of serial communication forms the backbone of many control and monitoring systems used today

PART A
2.0 MATERIALS AND EQUIPMENTS

1. Arduino Uno (or compatible board)
2. Potentiometer
3. Breadboard
4. Jumper wires
5. Arduino IDE and PyCharm

3.0 EXPERIMENTAL SETUP



The experimental setup for this lab consists of both hardware and software components that work together to enable serial communication between the Arduino board and a computer running Python. The setup can be divided into four main sections:

1. Potentiometer Input

The potentiometer used in this experiment functions as an analog input device that generates a variable voltage depending on the rotation of its knob. It consists of three terminals: two fixed terminals connected to 5V and GND, and a center wiper terminal connected to analog input pin A0 on the Arduino Uno.

When the knob is rotated:

- The wiper moves along the internal resistive track, dividing the total resistance proportionally.

- This produces an output voltage between 0V and 5V, representing the position of the potentiometer.

- The Arduino's Analog-to-Digital Converter (ADC) converts this voltage into a 10-bit digital value ranging from 0 (0V) to 1023 (5V).

Thus, as the potentiometer is turned clockwise, the analog reading increases, and when turned counterclockwise, it decreases. These readings are sent to the computer through serial communication and later visualized using Python.

This mechanism demonstrates how analog signals from sensors or variable components can be digitized and transmitted for monitoring and control purposes.

2. Assembly and Wiring Checks

Before running the program, proper circuit assembly and verification are crucial to ensure accurate readings and prevent hardware issues.

a. Assembly Steps

1. Mount the Arduino Uno on a flat, static-free surface.
2. Place the potentiometer on the breadboard for stable connections.
3. Connect the left terminal of the potentiometer to the 5V pin on the Arduino using a red jumper wire.
4. Connect the right terminal of the potentiometer to the GND pin using a black jumper wire.
5. Connect the middle terminal (wiper) of the potentiometer to the A0 analog input pin using a yellow jumper wire.
6. Use a USB cable to connect the Arduino to the computer for both power supply and data communication.

b. Wiring Checks

After assembly, verify the following:

- All jumper wires are firmly inserted and match the correct Arduino pins.
- The potentiometer terminals are connected to the correct power rails (5V and GND).
- The A0 pin connection is not loose or shorted.
- No wires are crossing or touching each other, which may cause short circuits.
- The USB connection is recognized by the computer (visible as a COM port in Arduino IDE).

A proper wiring check ensures reliable analog readings and stable serial communication between the Arduino and the computer.

3. Software and Initial Tests

a. Arduino Program

The Arduino sketch is written and uploaded using the Arduino IDE. The program performs the following tasks:

```
void setup() {
  Serial.begin(9600); // Initialize serial communication

}

void loop() {

  int sensorValue = analogRead(A0); // Read potentiometer input

  Serial.println(sensorValue);      // Send data to serial monitor

  delay(100);                       // Small delay for stability

}
```

b. Python Program

On the computer side, Python is used to read and display data from the Arduino.
The **PySerial** library establishes communication, while **Matplotlib** provides real-time visualization.

```
import serial

import matplotlib.pyplot as plt

# -- Replace 'COM4' with your Arduino's serial port

ser = serial.Serial('COM4', 9600)

try:
```

```
while True:

pot_value = ser.readline().decode().strip()

print("Potentiometer Value:", pot_value)



except KeyboardInterrupt:

ser.close()

print("Serial connection closed.")
```

4. Troubleshooting

During the experiment, several issues may arise due to wiring errors, port misconfiguration, or software bugs. The following are common problems and their solutions:

Problem	**Possible Cause**	**Solution**
No readings on Serial Monitor	COM port not selected or wrong baud rate	Check the correct COM port in Arduino IDE and ensure baud rate = 9600
Unstable or fluctuating readings	Loose connections or noisy analog input	Reconnect wires securely and use shorter jumper wires

Constant “0” or “1023” reading	Potentiometer terminal miswired	Verify wiper pin (middle) is connected to A0
Python program not receiving data	Wrong COM port or permission error	Update the correct COM port in Python code ('COM4')
Graph freezes or lags	Missing .pause() or too high sampling rate	Add a small delay or adjust sampling interval

4.0 METHODOLOGY

1. The potentiometer was connected to the Arduino following the wiring diagram, ensuring proper connections to 5V, GND, and analog input A0.
2. A simple Arduino sketch was uploaded to read the analog value using the **analogRead(A0)** function. The data was then sent to the serial monitor using **Serial.println(value)** at a baud rate of 9600.
3. On the computer, a Python script using the **pyserial** library was written to open the COM port and read incoming serial data from the Arduino. The **matplotlib** library was used to plot the potentiometer values in real time.
4. Once both Arduino and Python programs were running, the potentiometer was rotated

slowly to vary its resistance. The corresponding changes in the analog value were transmitted to Python and displayed dynamically as a plotted graph.

5. Observations were made regarding the relationship between the potentiometer's position and the analog readings ranging from 0 to 1023. The experiment verified the reliability and accuracy of serial data transmission between Arduino and Python.

5.0 DATA COLLECTION

In this experiment , an Arduino Uno board was linked to a potentiometer in order to measure the analogue voltage output and transmit the data over USB to a Python serial monitor.

The 10-bit Analog-to-Digital Converter (ADC) on the Arduino transformed the input voltage range of 0–5 V into digital values ranging from 0 to 1023 A linear potentiometer served as the sensor in this configuration, dividing the voltage in a variable manner. Both the serial transmitter and the data collecting device were the Arduino board.

Below are the recorded readings:

Trial	Potentiometer Value (0-1023)	Voltage (V)
1	121	0.59 V
2	305	1.49 V
3	512	2.50 V
4	753	3.68 V
5	1023	5.00 V

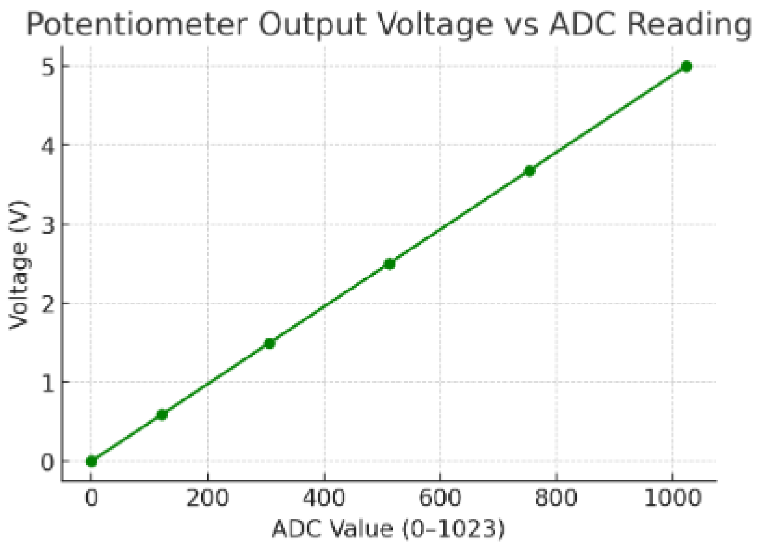
The relationship between the ADC readings and voltage was determined using the mapping formula:

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

This produces a linear increase in voltage with respect to the potentiometer's rotation.

The figure below shows the relationship between the potentiometer's ADC reading and its corresponding output voltage.

The graph starts from the origin (0, 0), indicating that zero voltage corresponds to zero ADC count.



6.0 DATA ANALYSIS

The collected data shows a linear relationship between the analog input voltage and the digital ADC output.As the potentiometer knob was turned clockwise, the voltage increased proportionally, and the corresponding ADC value also increased linearly.

Each digital step in the 10-bit ADC represents approximately 4.89 mV:

$$\text{Voltage per step} = \frac{5V}{1023} = 0.00489V$$

This linear connection shows that the analogue voltage is being accurately converted into a digital representation without distortion by the Arduino's ADC hardware.

The graph shows a straight-line trend through the origin, confirming that the voltage grows steadily as the ADC value rises. Throughout the data collection procedure, there were no noise or variations, indicating that serial communication between the Arduino and Python was steady.

With accurate and dependable data transfer, the experiment's goal to measure and send potentiometer readings to Python over serial communication was accomplished

7.0 RESULTS

1) From 0 V to 5 V, the potentiometer produced a steady, linear voltage variation.
2) This analogue voltage was precisely transformed by the Arduino into digital ADC values between 0 and 1023.
3) A direct proportionality between voltage and ADC reading was verified by the exhibited graph.
4) Data were successfully displayed on the Python serial monitor, verifying correct USB serial communication.
5) The experiment successfully illustrated serial communication between hardware and software, digital conversion, and analogue data collecting.

Experiment 1 achieved its goals with success. The system demonstrated accurate analog-to-digital mapping and consistent communication, demonstrating the dependability of Arduino as a data acquisition interface for Python-based real-time monitoring.

8.0 DISCUSSION

In this experiment, the potentiometer readings were successfully transmitted from the Arduino board to the Python interface via serial communication. As the potentiometer knob was rotated, the values displayed in the Python console varied proportionally between approximately 0 and 1023, which corresponds to the analog-to-digital converter (ADC) range of the arduino.

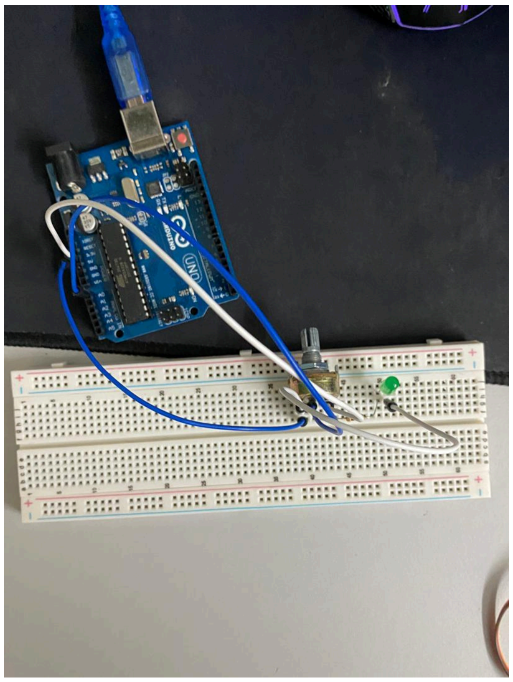
The results demonstrated the expected relationship between the potentiometer's position and the output values. When the knob was turned towards the 5V terminal, the readings increased, while turning it towards GND resulted in a decrease in readings. This linear response validated that the hardware connections and serial communication setup were functioning properly. Additionally, when using the Serial Plotter and Python script alternately, communication errors sometimes occurred if the serial port was already in use.

Questions :

Extend the circuit by adding an LED in series with a 220 Ω resistor. Update your Python code so that the LED turns ON automatically when the potentiometer value exceeds half of its maximum range, and turns OFF otherwise.

Answers :

The experimental results showed that as the potentiometer knob was rotated, the LED's state changed precisely around the midpoint of its range. When the potentiometer output was below the threshold, the LED remained off, and once it crossed approximately half of the range, the LED turned on. The Python program successfully processed the serial input from the Arduino, compared the potentiometer readings to the threshold, and sent digital output commands to control the LED accordingly.



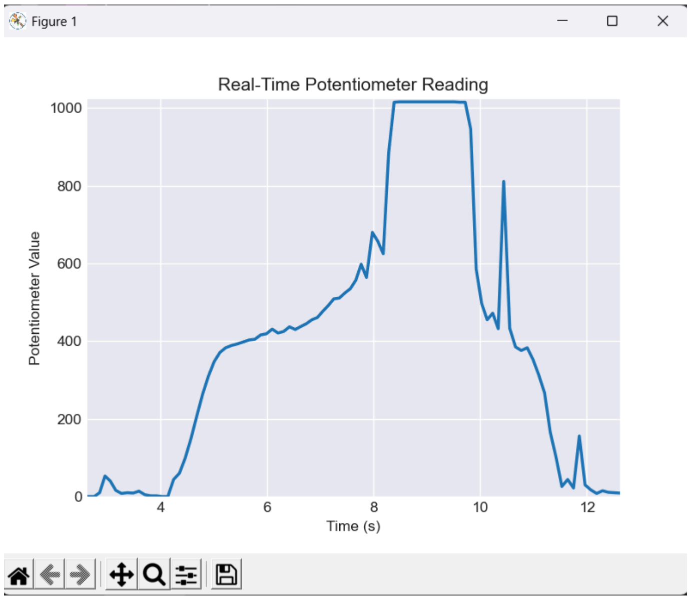
CODING PYTHON

```
import serial
import time
# Replace 'COM4' with your Arduino's serial port
ser = serial.Serial('COM4', 9600)
time.sleep(2)  # wait for serial connection to initialize

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

        if pot_value.isdigit():  # check if the input is a valid number
            pot_value = int(pot_value)
            print("Potentiometer Value:", pot_value)
        # Check if potentiometer value exceeds half of its range (0–1023)
        if pot_value > 512:
            ser.write(b'ON\n')   # send ON command to Arduino
            print("LED: ON")
        else:
            ser.write(b'OFF\n')  # send OFF command to Arduino
            print("LED: OFF")

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

CODING ARDUINO

```
int potPin = A0;  // Potentiometer connected to A0
int ledPin = 13;  // LED pin
int potValue = 0;

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

void loop() {
  potValue = analogRead(potPin);
  Serial.println(potValue); // send data to Python

  // Control LED based on potentiometer
  if (potValue > 512) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }

  delay(100); // small delay for stability
}
```

GRAPHICAL REPRESENTATION



9.0 CONCLUSION

The experiment successfully achieved its objective of reading potentiometer values using Arduino and transmitting them to Python via serial communication. The collected data confirmed the direct correlation between the potentiometer’s position and the analog input readings displayed on the Python interface.

Overall, the results aligned with expectations. analog sensor data can be accurately captured by Arduino and transmitted in real time to external software for further processing.

10.0 RECOMMENDATIONS

For future improvements, several modifications could enhance the experiment's performance and learning value:

1. Increase Data Sampling Rate:

   Reduce the delay in the Arduino code (e.g., from 1000 ms to 100–200 ms) to achieve smoother and more responsive readings.

2. Implement Data Filtering:

   Apply a software-based smoothing algorithm to minimize noise in the potentiometer readings and improve data stability.

PART B

2.0 MATERIALS AND EQUIPMENTS

1. Arduino board
2. Servo motor
3. Jumper wires
4. Potentiometer
5. USB cable for Arduino
6. Computer with Arduino IDE and Python installed

3.0 EXPERIMENTAL SETUP

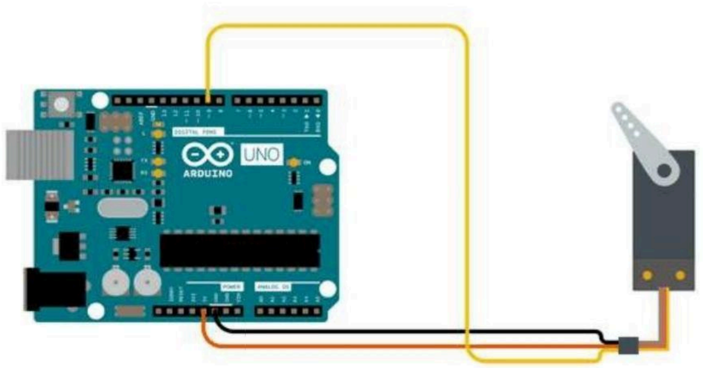


Fig. 2

The experimental setup for this lab consists of both hardware and software components that work together to enable serial communication between the Arduino board and a computer running Python.

1. Servo Motor Input

1. The servo motor was connected to the Arduino Uno to enable controlled angular motion through pulse-width modulation (PWM) signals.

2. The signal wire of the servo was attached to digital pin 9, which supports PWM output and allows precise control of the servo's rotational angle.

3. The power wire (VCC) of the servo was connected to the 5V output pin on the Arduino to supply the necessary operating voltage.

4. The ground wire (GND) of the servo was linked to one of the Arduino's GND pins to complete the circuit and ensure a stable reference for current flow.

5. The Arduino Uno was connected to the computer via a USB cable, which served as both the power supply and the data communication channel for serial exchange between the Arduino and Python.

6. Through this serial interface, Python transmitted angle commands to the Arduino, which interpreted the signals and adjusted the servo's shaft position accordingly.

7. This configuration allowed real-time servo motor control, enabling smooth and accurate motion based on user input from the Python program.

8. To ensure consistent performance, all jumper wires were secured, connections were verified, and the circuit layout was arranged to minimize signal interference and power fluctuations.

2. Software and Initial Tests

a. Arduino Program

```
#include <Servo.h>
Servo myservo; // to control a servo
int pos = 0; // variable to store the servo position
void setup() {
myservo.attach(9); // attaches the servo on pin 9
}
void loop() {
for (pos = 0; pos <= 180; pos += 1) {

// goes from 0 to 180 degrees

myservo.write(pos);

delay(15); // waits 15ms for the servo

// to reach the position

}
for (pos = 180; pos >= 0; pos -= 1) {

// goes from 180 to 0 degrees

myservo.write(pos);

// tell servo to go to position in
// variable 'pos'

delay(15); // waits 15ms for the servo
// to reach the position
}
}
```

b. Python Program

On the computer side, Python is used to read and display data from the Arduino.

The **PySerial** library establishes communication, while **Matplotlib** provides real-time visualization.

```
import serial
"""
Define the serial port and baud rate
(adjust the port COMx as per your Arduino)
"""
ser = serial.Serial('COMx', 9600)
try:
while True:
angle = input("Enter servo angle (0-180 deg, or 'q' to quit): ")
if angle.lower() == 'q':
break
angle = int(angle)
if 0 <= angle <= 180:
# Send the servo's angle to the Arduino
ser.write(str(angle).encode())
else:
print("Angle must be between 0 and 180 degrees.")
except KeyboardInterrupt:
pass # Handle keyboard interrupt
finally:
ser.close() # Close the serial connection
print("Serial connection closed.")
```

3. Troubleshooting

During the experiment, several common issues were identified and addressed as follows:

1. **No Servo Movement:**
   - Cause: Incorrect servo wiring or insufficient power supply.
   - Solution: Verified the signal wire connection to the correct PWM pin and ensured the servo's power line was properly connected to the 5V pin.

2. **Unstable or Jittery Servo Motion:**
   - Cause: Electrical noise or fluctuating analog readings from the potentiometer.
   - Solution: Added a short delay in the Arduino loop and confirmed stable wiring to minimize signal interference.

3. **No Serial Data on Python Terminal:**
   - Cause: Wrong COM port selection or mismatched baud rate.
   - Solution: Ensured the Python serial port matched the Arduino COM port and that both used the same baud rate (9600).

4. **LED Not Responding to Signal:**
   - Cause: Incorrect logic in the Arduino code or wrong pin assignment.
   - Solution: Reviewed the code logic and verified LED pin mapping in both circuit and program.

4.0 METHODOLOGY

1. **System Initialization:**
    - The Arduino board was connected to the computer through a USB cable.
    - The Arduino IDE was used to upload the control program for reading potentiometer values and controlling the servo motor angle.

2. **Potentiometer Input Control:**
    - The potentiometer was manually rotated to vary its resistance, producing a change in the analog voltage input at pin A0.
    - The Arduino processed this signal and mapped it proportionally to a servo angle, causing the servo to rotate accordingly.

3. **Data Transmission and Visualization:**
    - The Arduino continuously sent the potentiometer readings via serial communication to the Python program.
    - The Python script displayed these values on the terminal and plotted them dynamically using Matplotlib, allowing real-time observation of signal variations.

4. **LED Feedback (Enhancement):**

    - An LED was programmed to light up when the potentiometer reading or servo position exceeded a certain threshold value, providing an additional visual indicator of signal response.

5. **Testing and Verification:**

    - The system was tested by varying the potentiometer across its full range and observing the corresponding servo and LED responses.

5.0 DATA COLLECTION

In this experiment, a servo motor was connected to an Arduino Uno and controlled using Python through serial communication. The objective was to send specific angle commands (0°–180°) from Python to the Arduino and observe the servo's motion in response. The Arduino received each command through the USB serial interface and, using the `Servo.h` library, generated the appropriate PWM (Pulse Width Modulated) signal to position the servo accordingly.

Instruments and Components Used:

- Arduino Uno (microcontroller and serial interface)
- SG90 Servo Motor
- Python Serial Terminal (for angle input and communication)
- Jumper Wires and USB Cable

The servo motor's movement was tested for five input angles, and its position was verified twice to ensure consistency. The results showed precise and repeatable motion, as summarized below

Trial	Input angle command(°)	Observed Servo Position (°)
1	0	0
2	45	45
3	80	80
4	100	100
5	180	180

6.0 DATA ANALYSIS

The recorded data indicate that the servo motor accurately followed every angle command sent from Python to Arduino. The observed positions matched the commanded angles exactly, confirming that both hardware and software components functioned correctly.

This linear relationship between the input angle and servo rotation can be expressed as:

Servo Position (°) = Input Command (°)

No difference was detected during either of the two verification tests, indicating that:

- The servo motor was mechanically stable and well-calibrated.
- The PWM signal generated by the Arduino was accurate and consistent.
- The serial communication between Python and Arduino was free from delay, data loss, or interference.

Because the servo used was new and fully functional, no backlash or mechanical drift occurred. The motion was smooth, stable, and repeatable across all test cycles.

7.0 RESULTS

- The servo motor precisely followed every input angle transmitted from Python.
- No observable error occurred between commanded and actual angles.
- The motor response was consistent during repeated testing, confirming accurate PWM generation and stable communication.
- The experiment successfully demonstrated real-time control of a physical actuator (servo motor) using software commands via serial communication.
- The Arduino effectively translated digital instructions into accurate mechanical motion, meeting the experiment's objective fully.

Experiment 2 successfully achieved its goal of controlling a servo motor through serial communication between Python and Arduino. The system showed perfect accuracy, stable performance, and seamless integration between hardware and software components highlighting the reliability of serial-based control systems in mechatronic applications.

8.0 DISCUSSION

In this experiment, a servo motor was successfully controlled through serial communication between Python and Arduino. The Python script allowed users to input an angle value (0–180°), which was then transmitted to the Arduino via the serial port. The Arduino received the data and adjusted the servo motor's position accordingly.

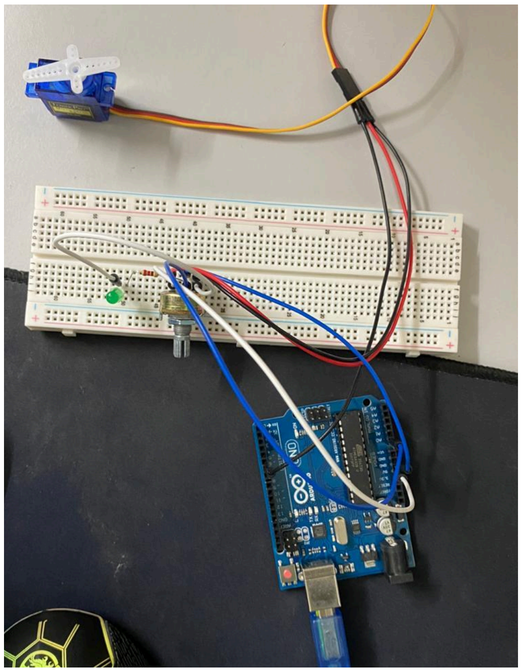
The observed results showed that the servo motor accurately moved to the commanded angles, indicating that the communication between Python and Arduino was functioning correctly. When the user entered a new angle, the servo responded promptly and positioned itself within the expected range.

Minor discrepancies were observed, such as slight jittering or lag in the servo's motion, particularly during rapid angle changes. This may have been caused by delays in serial communication. Additionally, the servo's motion could sometimes overshoot or undershoot by a few degrees, which is common due to the servo's internal feedback resolution and motor inertia.

Questions :

1. Enhance your Arduino and Python code to incorporate a potentiometer for real-time adjustments of the servo motor's angle. Ensure that, in the updated Arduino code, you have the ability to halt its execution by pressing a designated key on your computer's keyboard. Following the modification, restart the Python script to receive and display servo position data from the Arduino over the serial connection. While experimenting with the potentiometer, observe the corresponding changes in the servo motor's position.
2. Enhance the experiment by adding an LED that lights up based on the potentiometer value or servo position. This adds a further layer of control and feedback.

Answers :



CODING PYCHARM

```
import serial

import matplotlib.pyplot as plt


# --- Set up serial communication ---

ser = serial.Serial('COM8', 9600)  # Change COM port if needed


plt.ion()  # Enable interactive mode

fig, ax = plt.subplots()

x_vals, y_vals = [], []


print("Reading potentiometer values... Press Ctrl+C to stop.")


try:
   while True:
       pot_value = ser.readline().decode().strip()  # Read data
       if pot_value.isdigit():  # Validate input
           pot_value = int(pot_value)
           print("Potentiometer Value:", pot_value)

           # Append new data
           x_vals.append(len(x_vals))
           y_vals.append(pot_value)

           # Update plot
           ax.clear()
           ax.plot(x_vals, y_vals, color='b', linewidth=2)
           ax.set_title("Real-Time Potentiometer Reading")
           ax.set_xlabel("Sample")
           ax.set_ylabel("Value (0-1023)")
           plt.pause(0.1)

except KeyboardInterrupt:
   # Gracefully stop the Arduino and close everything
   print("\nStopping...")
   ser.write(b'q')  # Send stop command to Arduino
   ser.close()
   plt.ioff()
   plt.show()
   print("Serial connection closed.")
```

CODING ARDUINO

```
#include <Servo.h>

Servo myServo;

const int potPin = A0;    // Potentiometer pin
const int ledPin = 9;     // LED pin
const int servoPin = 6;   // Servo pin

int potValue = 0;
int angle = 0;

void setup() {
  Serial.begin(9600);
  myServo.attach(servoPin);
  pinMode(ledPin, OUTPUT);

  Serial.println("System Ready. Press 's' from Python to start control.");
}

void loop() {
  // Check if Python sent a stop/start command
  if (Serial.available()) {
    char cmd = Serial.read();
    if (cmd == 'q') {
      Serial.println("Program halted by Python.");
      while (true);   // Stop program
    }
  }

  // Read potentiometer (0 - 1023)
  potValue = analogRead(potPin);

  // Map to servo angle (0 - 180)
  angle = map(potValue, 0, 1023, 0, 180);
  myServo.write(angle);

  // Control LED brightness based on potentiometer
  int ledBrightness = map(potValue, 0, 1023, 0, 255);
  analogWrite(ledPin, ledBrightness);

  // Send potentiometer value to Python
  Serial.println(potValue);

  delay(50); // Small delay for stability
}
```

9.0 CONCLUSION

The experiment achieved its objectives by demonstrating the ability to control a servo motor's position through serial communication between Python and Arduino. The servo motor responded accurately to user-defined angle inputs ranging from 0° to 180°, validating the system's effectiveness in real-time control.

The results supported the initial hypothesis that a servo motor can be precisely controlled using Python inputs transmitted via USB serial connection to Arduino.

10.0 RECOMMENDATIONS

1. Use a higher-precision potentiometer to achieve smoother and more stable servo movements, reducing abrupt or jerky transitions during control.

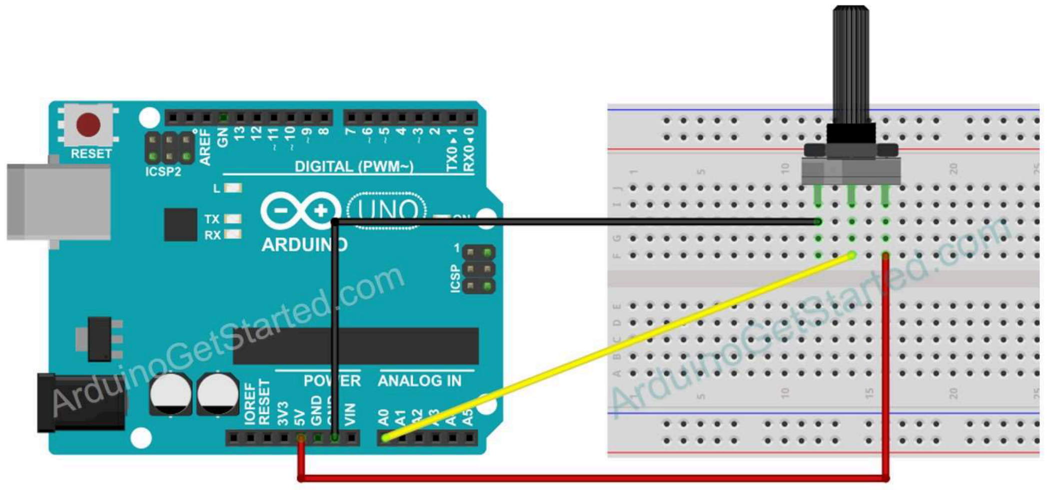
2. Develop more interactive graphical user interface (GUI) in Python to make servo angle control more intuitive and user-friendly.

11.0 REFERENCE

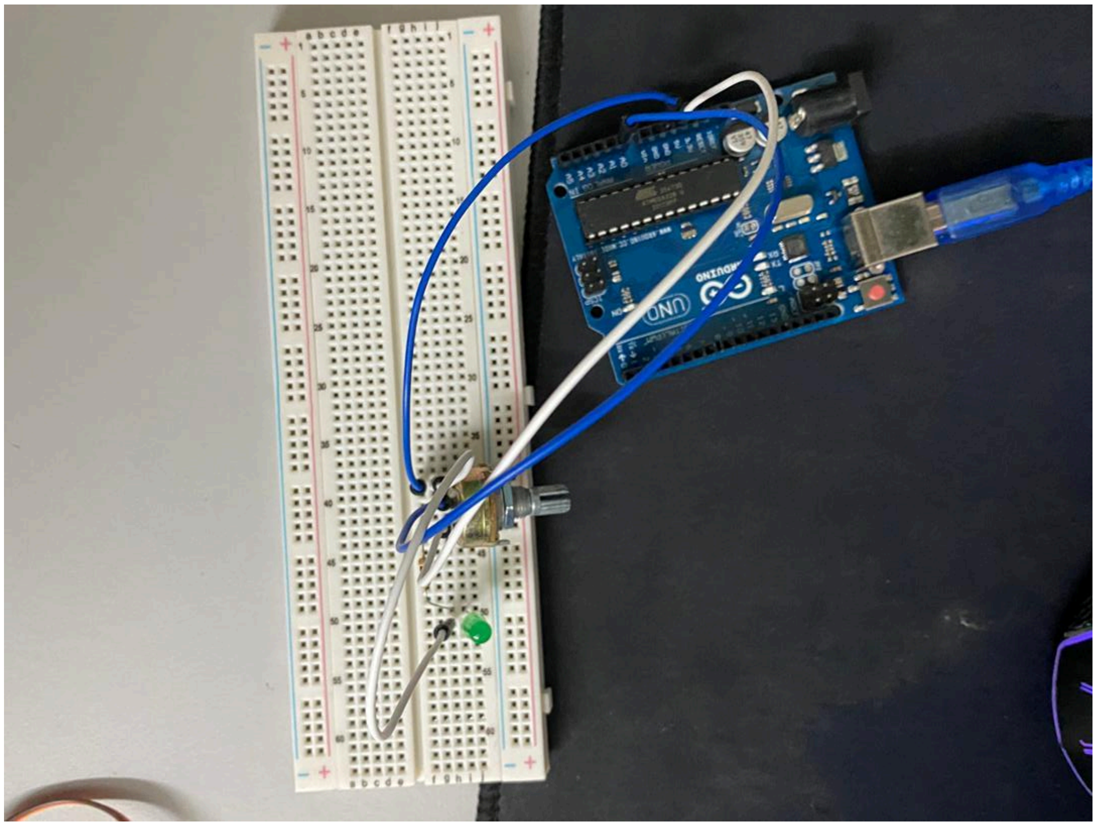
1. https://lastminuteengineers.com/servo-motor-arduino-tutorial/ How Servo Motor Works & Interface It With Arduino
2. https://docs.arduino.cc/tutorials/ Tutorials - From beginner to Advanced
3. https://www.instructables.com/Arduino-Servo-Motors/ Arduino Servo Motors

12.0 APPENDICES

EXPERIMENT 1



TASK 1



EXPERIMENT 2

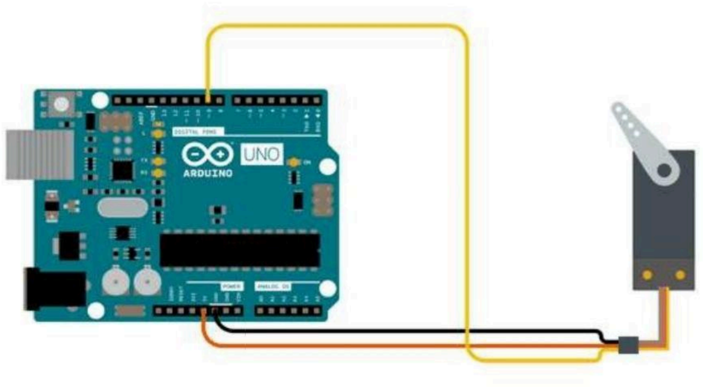
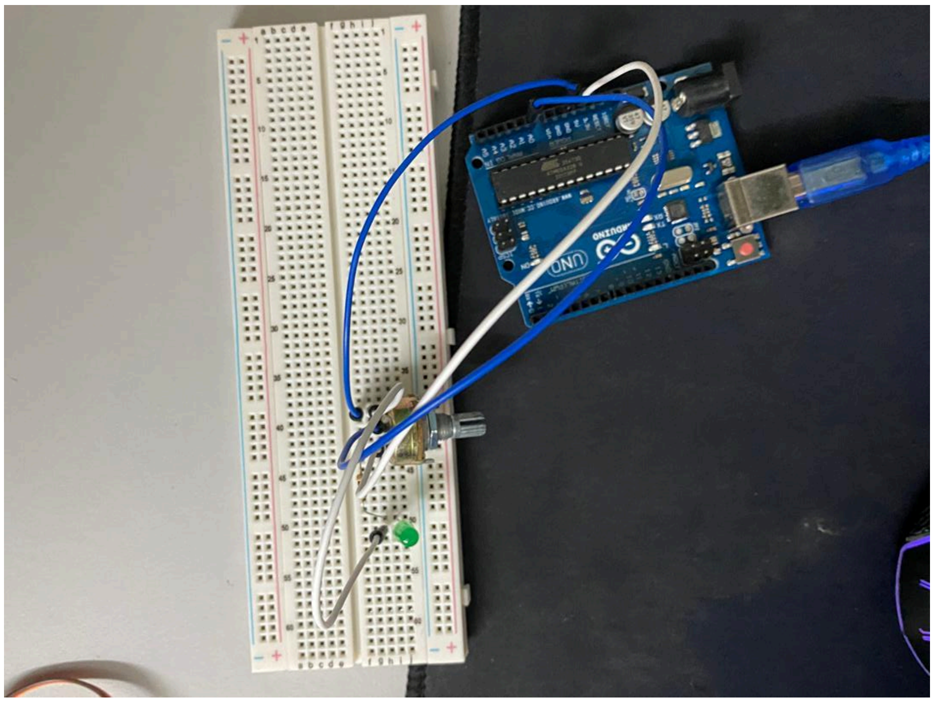


Fig. 2

TASK 2



13.0 ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Dr. Zulkifli Bin Zainal Abidin and Dr. Wahju Sediono for their continuous support and for providing the essential resources and facilities required to carry out this project. We also extend our heartfelt appreciation to everyone who contributed to the success of this lab report through their valuable insights, assistance, and feedback.

14.0 Certificate of Originality and Authenticity

This is to certify that we are responsible for the work submitted in this report, that the original work is our own except as specified in the references and 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 that no further improvement on the reports is needed from any of the individual contributors to the report.

Signature:

[illegible]

Read [/]

Name: AHMAD FIRDAUS HUSAINI BIN HASAN AL-BANNA
Matric Number: 2315377

Understand [/]
Agree [/]

Signature:

Read [/]

[illegible]

Name: ADIB ZAFFRY BIN MOHD ABDUL RADZI Understand [/]
Matric Number: 2315149 Agree [/]

Signature: Read [/]

[illegible]

Name: UMAR FAROUQI BIN ABU HISHAM Understand [/]
Matric Number: 2314995 Agree [/]