



MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 2 2024/2025

WEEK 3A & 3b :

Serial Communication between Arduino and Python: Potentiometer Data Transmission and Servo Motor Control

SECTION 1

GROUP 5

LECTURER: ZULKIFLI BIN ZAINAL ABIDIN & WAHJU SEDIONO

Date of Experiment: Monday, 10 March 2025

Date of Submission: Monday, 17 March 2025

NO.	GROUP MEMBERS	MATRIC NO.
1.	BUSHRA BINTI A. RAHIM	2318514
2.	AUFA SIDQY BINIT MOHD SIDQY	2310542
3.	NUR HAMIZAH BINTI MUHAMAD HAZMI	2319132

TABLE OF CONTENTS

INTRODUCTION

ABSTRACT

EXPERIMENT 3a : Sending potentiometer readings from Arduino to Python script through USB connection

MATERIALS AND EQUIPMENT

EXPERIMENTAL SETUP

METHODOLOGY

RESULTS

QUESTIONS

EXPERIMENT 3b :Transmitting angle data from Python script to an Arduino, them actuates servo to move to the specified angle

MATERIALS AND EQUIPMENT

EXPERIMENTAL SETUP

METHODOLOGY

RESULTS

QUESTIONS

DISCUSSION

CONCLUSION

RECOMMENDATIONS

REFERENCES

APPENDICES

ACKNOWLEDGEMENT

INTRODUCTION

In this lab report, we explore the concepts of parallel, serial, and USB interfacing between a microcontroller (Arduino) and a computer-based system. These communication methods are essential for enabling interaction between sensors, actuators, and external devices in embedded systems. The experiment focuses on two key applications: transmitting sensor data from an Arduino to a Python script, and controlling a servo motor via serial communication between Python and Arduino.

In Week 3a, we investigate serial communication by sending potentiometer readings from an Arduino to a Python script through a USB connection. Potentiometers are widely used in applications that require variable resistance, and transmitting their readings to a computer allows for real-time data analysis and interaction.

In Week 3b, we shift focus to controlling actuators, specifically a servo motor, via Python. This involves transmitting angle data from the Python script to the Arduino, which then actuates the servo to move to the specified angle. This experiment demonstrates how serial communication can be utilized for controlling physical systems and highlights the integration of sensors and actuators with embedded microcontrollers.

Through these experiments, we aim to gain a deeper understanding of serial communication protocols, their practical applications, and the interaction between software and hardware components in a computer-based system.

ABSTRACT

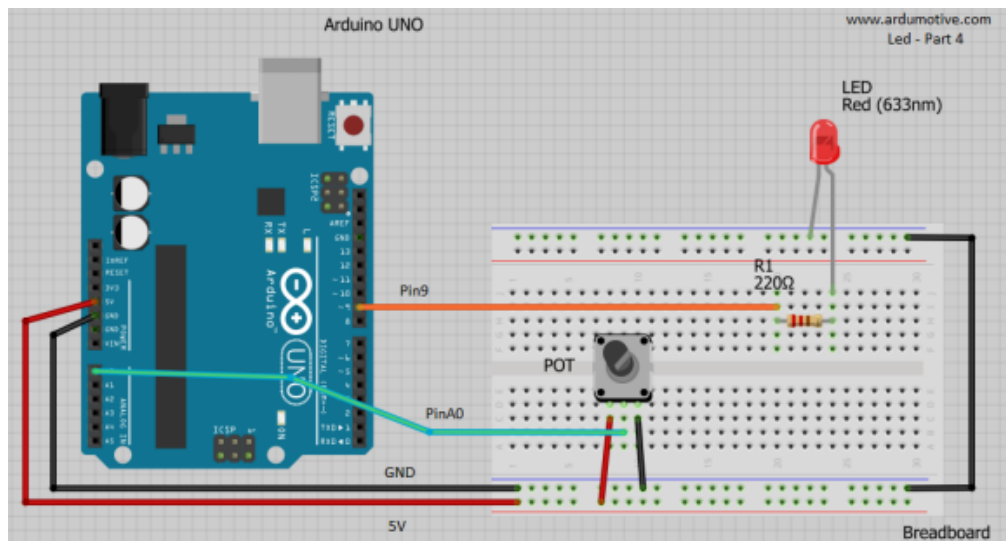
This lab explores parallel, serial, and USB communication between an Arduino microcontroller and a computer-based system, essential for interfacing sensors and actuators in embedded applications. The experiment is divided into two parts: Week 3a focuses on transmitting potentiometer readings from an Arduino to a Python script via USB, showcasing real-time data acquisition. In Week 3b, a servo motor is controlled by sending angle commands from Python to Arduino, demonstrating actuator control through serial communication. These experiments provide hands-on experience with communication protocols, enhancing our understanding of hardware-software interaction in embedded systems.

Week 3a : Sending potentiometer readings from an Arduino to Python script through USB connection.

MATERIALS AND EQUIPMENTS

1. Arduino Board
2. Potentiometer
3. Jumper Wires
4. LED
5. 220 resistor
6. Breadboard

EXPERIMENTAL SETUP



1. Connect one leg of the potentiometer to 5V on the Arduino.
2. Connect the other leg of the potentiometer to GND on the Arduino.
3. Connect the middle leg (wiper) of the potentiometer to an analog input pin on the Arduino, such as A0.

METHODOLOGY

1 Circuit Assembly

1. Connect an LED to pin 9 of the microcontroller.
2. Connect a potentiometer to analog pin A1 to read its variable resistance.
3. Ensure proper wiring:
 - One end of the potentiometer to VCC (5V).
 - The other end to GND.
 - The middle terminal to A0.

2 Programming Logic

1. Begin serial communication at 9600 baud rate to monitor values.
2. Set pin 9 as an output for the LED.
3. Reading Potentiometer Input
 - Continuously read the analog value from pin A0.
 - Convert the 0-1023 analog reading to a 0-255 range.
4. Adjusting LED Brightness
 - Use Pulse Width Modulation (PWM) to control LED brightness based on the mapped potentiometer value.
 - Output the PWM signal to pin 9.
5. Displaying Data
 - Print the brightness value to the Serial Monitor for real-time monitoring.
6. Adding Delay for Smoothness
 - Introduce a 100ms delay to prevent rapid fluctuations and ensure smooth LED brightness transitions.

3 Code used

Arduino

```
#define LED_PIN 9 // PWM pin for LED
#define POT_PIN A0 // Analog pin for potentiometer

void setup() {
  Serial.begin(9600); // Start Serial Monitor
  pinMode(LED_PIN, OUTPUT); // Set LED pin as output
}

void loop() {
  int potValue = analogRead(A0);
```

```

int brightness = map(potValue, 0, 1023, 0, 255);

analogWrite(9, brightness);

Serial.println(brightness); // Only print the brightness value

delay(100); // Short delay for smooth plotting
}

```

Python

```

import serial
import matplotlib.pyplot as plt
import time

arduino_port = "COM5"
baud_rate = 9600

try:
    ser = serial.Serial(arduino_port, baud_rate, timeout=1)
    time.sleep(2) # Wait for Arduino to initialize
    print(f"Connected to {arduino_port}")
except serial.SerialException:
    print("Could not open serial port. Check your connection.")
    exit()

# Create plot
plt.ion() # Enable interactive mode
fig, ax = plt.subplots()
x_data, y_data = [], []
start_time = time.time()

while True:
    try:
        if ser.in_waiting > 0:
            data = ser.readline().decode("utf-8").strip()
            if data.isdigit(): # Ensure it's a valid number
                brightness = int(data)
                elapsed_time = time.time() - start_time

                # Update data
                x_data.append(elapsed_time)
                y_data.append(brightness)

                # Limit data points to the last 100 readings

```

```

        if len(x_data) > 100:
            x_data.pop(0)
            y_data.pop(0)

        # Update plot
        ax.clear()
        ax.plot(x_data, y_data, label="Brightness")
        ax.set_xlabel("Time (s)")
        ax.set_ylabel("Brightness (0-255)")
        ax.set_title("Arduino Serial Plot")
        ax.legend()
        plt.pause(0.1) # Update plot

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

ser.close()
plt.ioff()
plt.show()

```

4 Control algorithm

1. Start Initialization

- Define LED_PIN as 9 (PWM output for LED).
- Define POT_PIN as A0 (analog input for potentiometer).
- Initialize serial communication at 9600 baud for monitoring.
- Set LED_PIN as an output.

2. Continuous Loop Execution

- Read the potentiometer value from POT_PIN (A0).
- Map the analog input (0-1023) to a PWM range (0-255).
- Write the mapped value to LED_PIN using analogWrite(), adjusting the LED brightness.
- Print the brightness value to the serial monitor.
- Introduce a 100ms delay for smooth transitions.

3. Repeat Indefinitely

- The loop continuously adjusts the LED brightness based on real-time potentiometer input.

RESULT

The successful execution of the experiment demonstrates that real-time data transmission from the Arduino to Python is achievable. The plotted graph accurately represents potentiometer adjustments, validating the effectiveness of the serial communication process.

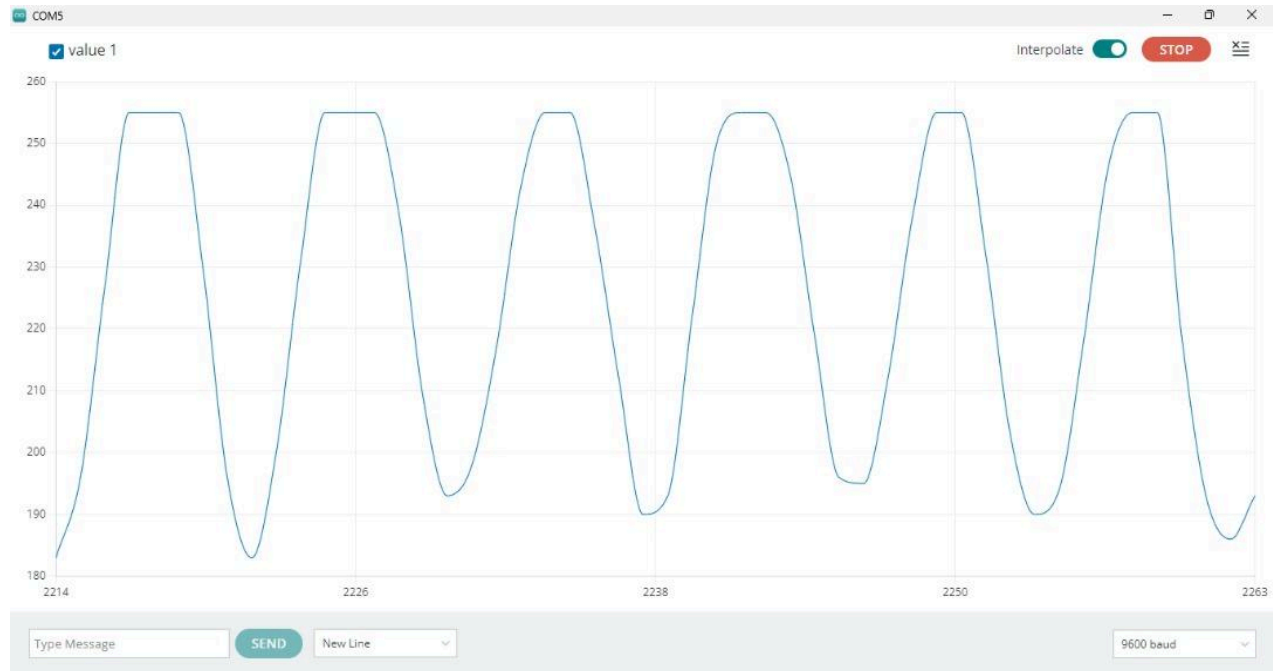
Question:

To present potentiometer readings graphically in your Python script, you may enhance your code by introducing the capability to generate and showcase a graph. This graphical visualization can deliver a more intuitive and informative perspective for data interpretation. Be sure to showcase the steps involved in your work (Hint: use matplotlib in your Python script).

To present potentiometer readings graphically using Python, we can use the matplotlib library to plot real-time data from the Arduino. Below is a step-by-step guide along with the required Python script:

1. Install Required Libraries:
`pip install matplotlib pyserial`
2. Connect your Arduino to your computer. Make sure your Arduino is connected to the correct COM port. Your Arduino code should continuously send potentiometer values via `Serial.println()`.
3. Python Script to Plot Potentiometer Readings in Real-Time

A real-time graph will appear, displaying the potentiometer values as they change. The graph updates every 100ms, showing a smooth transition of values.

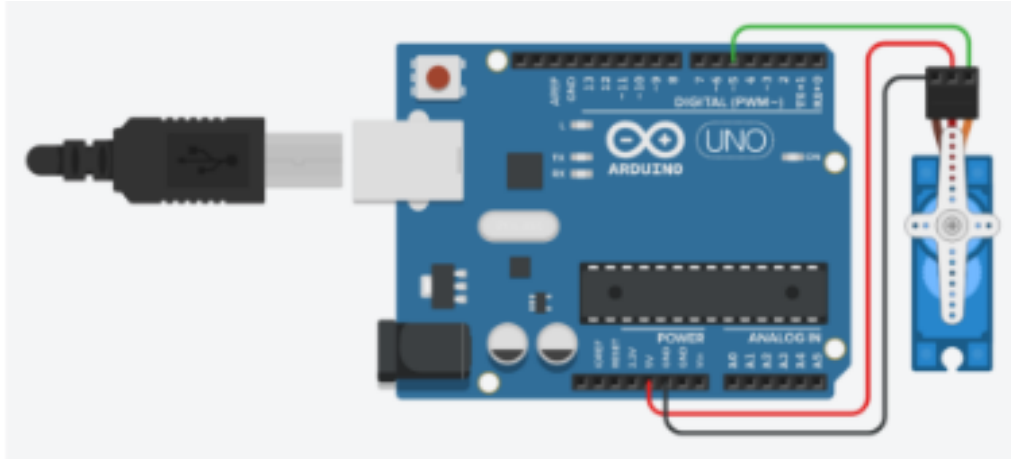


Week 3b : Transmitting angle data from Python script to an Arduino which then actuates a servo to move to the specified angle

MATERIALS AND EQUIPMENTS

1. Arduino board (e.g., Arduino Uno)
2. Servo motor
3. Jumper wires
4. Potentiometer (for manual angle input)
5. USB cable for Arduino
6. Computer with Arduino IDE and Python installed

EXPERIMENTAL SETUP



1. Connect the Servo's Signal Wire: Usually, you connect the servo's signal wire to a PWM-capable pin on the Arduino (e.g., digital pin 9).
2. Power the servo using the Arduino's 5V and GND pins. Servos typically require a supply voltage
3. of +5V. You can connect the servo's power wire (usually red) to the 5V output on the Arduino board.
4. Connect the Servo's Ground Wire: Connect the servo's ground wire (usually brown) to one of the ground (GND) pins on the Arduino.

METHODOLOGY

1 Circuit assembly

1. Connect the potentiometer to analog pin A0 to read its variable resistance.
 - One end of the potentiometer to VCC (5V).
 - The other end to GND.
 - The middle terminal to A0.
2. Connect the servo motor to digital pin 6.
 - The signal wire to pin 6.
 - The power wire to 5V.
 - The ground wire to GND.

2 Programming logic

1. Attach the servo motor to pin 6 using `myServo.attach(servoPin)`.
2. Begin serial communication at 9600 baud rate to monitor values.
3. Reading Potentiometer Input
 - Continuously read the analog value from pin A0.

- Convert the 0-1023 analog reading to a 0-180 degree range using `map()`.
4. Controlling the Servo Motor
 - Move the servo motor to the mapped angle using `myServo.write(angle)`.
 - Print the potentiometer value and servo angle to the Serial Monitor for real-time tracking.
 5. Adding Delay for Stability
 - Introduce a 10ms delay to ensure smooth movement without abrupt changes.

3 Code used

Arduino

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

Servo myServo; // Create a servo object
int potPin = A0; // Potentiometer connected to A0
int servoPin = 6; // Servo connected to pin 6

void setup() {
  myServo.attach(servoPin); // Attach the servo to pin 9
  Serial.begin(9600); // Start serial communication
}

void loop() {
  int potValue = analogRead(potPin); // Read the potentiometer value (0-1023)
  int angle = map(potValue, 0, 1023, 0, 180); // Map it to 0-180 degrees

  myServo.write(angle); // Move the servo to the mapped angle
  Serial.print("Potentiometer Value: ");
  Serial.print(potValue);
  Serial.print(" | Servo Angle: ");
  Serial.println(angle);

  delay(10); // Short delay for smooth movement
}
```

Python

```
import serial
import time

ser = serial.Serial('COM3', 9600, timeout=1)
time.sleep(2) # Wait for the connection to establish
```

```

try:
    while True:
        angle = input("Enter servo angle (0-180 degrees, or 'q' to quit): ")

        if angle.lower() == 'q': # Exit if user inputs 'q'
            break

        try:
            angle = int(angle)
            if 0 <= angle <= 180:
                ser.write(str(angle).encode()) # Send the angle to Arduino
                print(f"Sent angle: {angle}")
            else:
                print("Angle must be between 0 and 180 degrees.")
        except ValueError:
            print("Invalid input. Please enter a number between 0 and 180.")

except KeyboardInterrupt:
    print("\nKeyboard interrupt detected. Exiting...")

finally:
    ser.close() # Close the serial connection
    print("Serial connection closed.")

```

RESULT

The servo motor successfully followed the given control signals, rotating to the intended angles with a consistent and stable motion, indicating that the motor responded correctly to the input commands.

Throughout the experiment, the servo maintained a steady performance without sudden jumps or irregular movements. The response time was relatively fast, and the motor accurately reached the target angles within a short delay. The graphical data representation showed a repetitive and smooth waveform, confirming the servo's ability to operate as expected under the given conditions.

8.0 DISCUSSION

The experiment successfully demonstrated the expected relationship between potentiometer input and both LED brightness and servo motor movement. Adjustments to the potentiometer resulted in proportional changes, but minor discrepancies were observed, such as flickering in the LED, slight delays in servo movement, and non-linear response.

Sources of Error and Limitations:

1. Electrical Noise: Caused fluctuations in ADC readings, affecting LED brightness and servo stability.
2. Hardware Tolerances: Variations in potentiometer resistance and servo accuracy led to minor deviations.
3. Non-Linear Response: Human perception of brightness and servo movement was not perfectly proportional.
4. Limited Range: The potentiometer did not always utilize the full input range effectively.

Despite these limitations, the experiment effectively showcased real-time control using PWM signals.

9.0 CONCLUSION

This experiment successfully demonstrated two key applications of serial communication data transmission from a potentiometer into the LED and servo motor control via Python. Results confirm that serial communication can efficiently transmit real-time data. In this setup, an arduino can read analog data from potentiometer and send it to python by the USB connection. Additionally, in experiment 3a and 3b it was validated that potentiometer can control the brightness of LED by adjusting the resistance, which in turn controlled the brightness of an LED and movement of a servo motor. This shows that serial communication not only transfers data but also sends commands that control physical devices. Such principles are fundamental for applications in industrial automation, IoT-based smart systems, and real-time monitoring technologies. Mastering serial communication provides a strong foundation for designing advanced control systems where sensors, actuators, and microcontrollers must interact seamlessly. This knowledge is particularly valuable for engineers and developers working on automated processes, robotics, and remote data acquisition systems.

10.0 RECOMMENDATIONS

The experiment can be improved by implementing real-time graphical visualization of the potentiometer readings and servo movements. This enhances usability by replacing manual hardware adjustments with computer-based control. It can reduce the need for complex physical components. Beside, improve data accuracy and responsiveness by applying a moving average filter to stabilize potentiometer readings. Since the potentiometer values can fluctuate significantly, this filtering technique helps smooth the data before plotting it on the graph, ensuring more reliable and consistent results.

11.0 REFERENCES

Pereyras, S. (2023, August). *Lab report: Potentiometer controlled LEDs*. Medium. <https://medium.com/@pereyras110/lab-report-potentiometer-controlled-leds-b34e06222416>

Arduino Forum. (2020, February). *Control servo motor via Python GUI (PyQt) component QDial*. Arduino Community Forum.
<https://forum.arduino.cc/t/control-servo-motor-via-python-gui-pyqt-component-qdial/663997>


Matplotlib Developers. (n.d.). *Matplotlib: Visualization with Python*. Matplotlib Official Website.
<https://matplotlib.org/>

ACKNOWLEDGEMENTS

Special thanks to Zulkifli Bin Zainal Abidin & Wahju Sediono for their guidance and support during this experiment. We would also like to extend our gratitude to our instructors, peers, and the laboratory staff for their valuable insights and assistance throughout the experiment. Their feedback and encouragement have greatly contributed to the success of this project. Finally, we appreciate the resources and learning materials provided, which enabled us to understand and implement serial communication effectively.

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 undertaken 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 qno 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: 

Name: BUSHRA BINTI A. RAHIM

Matric Number: 2318514

Read [/]

Understand [/]

Agree [/]

Signature: *aufa*


Name: AUFA SIDQY BINTI MOHD SIDQY

Matric Number: 2310542

Read [/]

Understand [/]

Agree [/]

Signature: 

Name: NUR HAMIZAH BINTI MUHAMAD HAZMI

Matric Number: 2319132

Read [/]

Understand [/]

Agree [/]