MECHATRONICS SYSTEM INTEGRATION MCTA3202

EXPERIMENT 3: TASK 1 & TASK 2

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

DATE: 20TH OCTOBER 2025
SECTION: 2
GROUP: 18
SEMESTER 1, 2025/2026

NO	NAME	MATRIC NO
1.	SHAMSUL HAKIMEE BIN SHAMSUL HAIREE	2315027
2.	QASHRINA AISYA BINTI MOHD NAZARUDIN	2315184
3.	NUREL HUMAIRAH BINTI MUHAMMAD FIRDAUS	2315680

DATE OF SUBMISSION:
Monday, 27TH October 2025

Abstract

This experiment explores the implementation of serial communication between an Arduino microcontroller and a Python interface for sensor data transmission and actuator control. In Task 1, a potentiometer was connected to the Arduino to generate analog readings, which were transmitted to a Python program using serial communication. The received data were displayed in real-time, and an LED was programmed to automatically turn ON when the potentiometer value exceeded half of its maximum range, providing a simple control feedback mechanism. As for Task 2, the experiment was extended to control a servo motor through serial communication. A potentiometer was used to adjust the servo angle in real-time, while Python handled both the input and visualization of the servo position. Additionally, an LED indicator was incorporated to reflect servo behavior, and real-time plotting was achieved using the matplotlib library. These experiments demonstrate fundamental concepts of bidirectional serial communication, data visualization, and hardware–software integration for mechatronic system applications.

TABLE OF CONTENTS

1.0 Introduction

This experiment aims to demonstrate the fundamentals of serial communication through the integration of an Arduino microcontroller with a Python-based interface. The main objectives are to transmit sensor data, such as potentiometer readings, from Arduino to Python, and to control hardware components like a servo motor and LED using commands from the Python program. Through this experiment, students will gain practical experience in establishing two-way communication between hardware and software, performing data visualization, and implementing real-time control in mechatronic systems.

Serial communication is an essential concept in embedded systems and modern electronics, as it enables data to be transferred one bit at a time between devices in a synchronized and efficient manner. In this setup, the Arduino serves as the transmitting and receiving unit for sensor and actuator signals, while Python functions as the high-level interface for monitoring and control. Components such as potentiometers, LEDs, and servo motors are used to illustrate key principles including analog-to-digital conversion, pulse-width modulation (PWM), and feedback-based control. Additionally, the *pyserial* and *matplotlib* libraries in Python are employed to handle data exchange and visualize sensor readings in real time.

The hypothesis states that, with correct circuit connections and successful program execution, the potentiometer values will be accurately transmitted to Python, the LED will activate based on defined threshold conditions, and the servo motor will adjust its angle according to the input from Python or the potentiometer. Successful execution of these functions will demonstrate a proper understanding of serial communication, microcontroller interfacing, and software-based control in a mechatronic system.

2.0 Experiment 1

2.1 Materials and Equipment

Materials and components that were used in the experiment:

1. Arduino Uno Mega Microcontroller
2. Potentiometer
3. 220 Ohm resistors (1x)
4. Jumper Wires
5. Breadboard
6. LED (1x)

2.2 Experiment setup

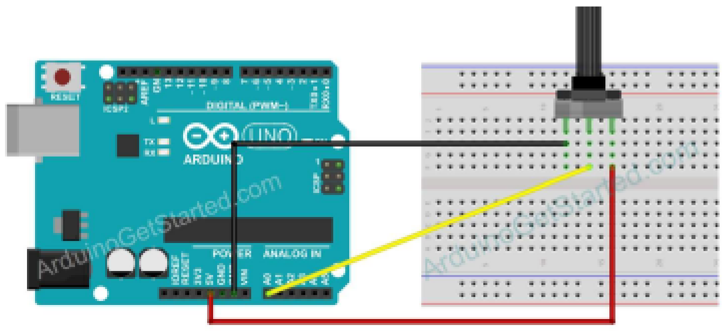


Fig. 1. Wiring Diagram

2.3 Methodology

1. Hardware Assembly

The experiment was conducted using an Arduino UNO, a potentiometer, a solderless breadboard, and connecting jumper wires. The components were assembled according to the wiring diagram (Fig. 1).

1. The potentiometer was mounted onto the breadboard.
2. One outer pin of the potentiometer was connected to the 5V pin on the Arduino UNO.
3. The other outer pin of the potentiometer was connected to the GND pin on the Arduino.
4. The center (wiper) pin of the potentiometer was connected to the analog input pin A0 on the Arduino.
5. The Arduino was connected to a host computer via a USB cable to provide power and establish a communication link.

2. Software and Programming

2.1 Arduino Sketch (Data Transmission)

An Arduino sketch was developed to read the analog sensor value and transmit it over the serial port.

1. In the `void setup()` function, serial communication was initialized at a baud rate of 9600 using the `Serial.begin(9600)` command.
2. In the `void loop()` function, the following steps were executed:
   - The analog voltage from the potentiometer on pin A0 was read using `int potValue = analogRead(A0);`. This converts the 0-5V input into an integer value between 0 and 1023.
   - This integer value was sent over the serial port as a string followed by a newline character using `Serial.println(potValue);`.
   - A `delay(1000);` was added at the end of the loop to limit the data transmission rate to approximately one reading per second, preventing the serial buffer from overflowing and making the output readable.
3. The completed sketch was compiled and uploaded to the Arduino UNO board.

2.2 Python Script (Data Reception)

A Python script was written on the host computer to receive, process, and display the data from the Arduino.

1. The `serial` library (`pyserial`) was imported to handle serial port communication.
2. A serial port object was created using `ser = serial.Serial('COMx', 9600)`, where 'COMx' was replaced with the Arduino's specific port (e.g., `COM3` or `/dev/ttyUSB0`) and the baud rate was set to 9600 to match the Arduino.
3. A `while True` loop was initiated to continuously listen for incoming data.
4. Inside a `try...except` block, the `ser.readline().decode().strip()` command was used to:
   - `readline()`: Read data from the serial buffer until a newline character (`\n`) was received.
   - `decode()`: Convert the received byte-string into a standard UTF-8 string.
   - `strip()`: Remove any leading or trailing whitespace.
5. The resulting clean string (`pot_value`) was printed to the console.
6. An `except KeyboardInterrupt` block was included to allow the loop to be terminated cleanly (e.g., by pressing Ctrl+C).

3. Data Collection Procedure

1. After the Arduino sketch was uploaded, the Arduino IDE's Serial Monitor was closed to free the COM port for the Python script.
2. The Python script was executed from the terminal or an IDE (like PyCharm).
3. The knob of the potentiometer was manually rotated from its minimum position to its maximum position and back.
4. The output in the Python console was observed to verify that the integer values (ranging from approximately 0 to 1023) were being received and displayed correctly, corresponding to the physical position of the potentiometer knob.

Code used:

Arduino

```
void setup() {
Serial.begin(9600);
}
void loop() {
int potValue = analogRead(A0);
Serial.println(potValue);
delay(1000); // add a delay to avoid sending data too fast
}
```

Python

```
import serial
# -- Replace 'COMx' with your Arduino's serial port
ser = serial.Serial('COMx', 9600)
try:
while True:
pot_value = ser.readline().decode().strip()
print("Potentiometer Value:", pot_value)
except KeyboardInterrupt:
ser.close()
print("Serial connection closed.")
```

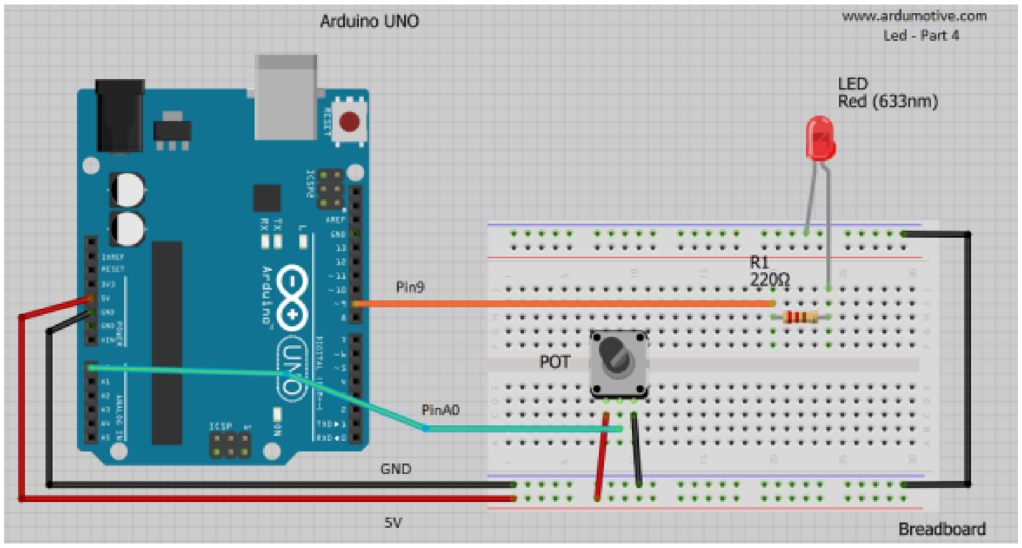
3.0 Task 1: Sending potentiometer readings from an Arduino to Python script through USB connection.

3.1 Materials and Equipment

Materials and components that were used in the experiment:

7. Arduino Uno Mega Microcontroller
8. Potentiometer
9. 220 Ohm resistors (1x)
10. Jumper Wires
11. Breadboard
12. LED (1x)

3.2 Experimental Setup



1. Connect one terminal of the potentiometer to the 5V power supply of the Arduino.
2. Connect the opposite terminal of the potentiometer to the Arduino GND.
3. Connect the middle pin of the potentiometer to the analog input pin A0 on the Arduino.
4. Attach an LED to digital pin 9 of the microcontroller.
5. Connect a second potentiometer to analog pin A1 to measure its variable resistance.
6. Verify proper wiring connections:
    - One outer terminal of each potentiometer goes to 5V (VCC).
    - The other terminal connects to GND.
    - The center terminal is connected to A0 or A1 respectively.

3.3 Methodology

Implementation and Testing

1. The circuit was built based on the schematic diagram.
2. The Arduino code was uploaded using the Arduino IDE, as this program was designed to read the potentiometer value using the analogRead pin (A0) and send data to the Python program through the serial port at a 9600 baud rate.
3. A Python script using the pyserial library was created to receive and display potentiometer values in real time.
4. During testing, the potentiometer knob was rotated, and the changes in readings were displayed in the Python terminal.
5. The LED responded according to the potentiometer position, confirming the correct operation of the threshold control.
6. The Arduino Serial Plotter was used to visualize the potentiometer readings, showing continuous and smooth variation.

Control Algorithm

The system was programmed using two separate components: an Arduino sketch uploaded using the Arduino IDE and a Python script running on a computer. The Arduino board's primary role was data acquisition to read the potentiometer, while the Python script was responsible for data processing and control logic for determining the LED state. Communication between the two was established using USB serial.

1. Initialize pins as the potentiometer was connected to pin A0 and the LED was connected to pin D7. As the GND and 5V pins were connected at their pins on the Arduino.
2. Define the pin connected to the LED as output.
3. Loop the program continuously.
4. Inside the loop, the analog value from the potentiometer (pin A0) was being read.
5. Reading Potentiometer Input by continuously read the analog value from pin A0 and converting the 0-1023 analog reading to a 0-255 range.
6. Adjusting LED Brightness
   -Use Pulse Width Modulation (PWM) to control LED brightness based on the mapped potentiometer value.
   -Output the PWM signal to pin 7.
7. The brightness value was being displayed to the serial monitor for real-time monitoring.

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

```python
import serial
import time

# --- CONFIGURATION ---
# !!! IMPORTANT !!!
# Update this to your Arduino's COM port (you found it was 'COM5')
COM_PORT = 'COM5'
BAUD_RATE = 9600
# ---------------------

print(f"Connecting to {COM_PORT}...")

try:
   # Use 'with' to automatically open and close the serial port
   with serial.Serial(COM_PORT, BAUD_RATE, timeout=1) as ser:

       print(f"Successfully connected to {ser.name}")
       time.sleep(2)  # Wait for the Arduino to reset

       print("Reading potentiometer data... (Press Ctrl+C to stop)")
       while True:
           # First, check if there's data to read
           if ser.in_waiting > 0:

               # Read a line from the Arduino
               pot_value_str = ser.readline().decode('utf-8').strip()

               # Make sure we got a non-empty string
               if pot_value_str:
                   try:
                       # Convert the value to an integer
                       pot_value = int(pot_value_str)
                       print(f"Potentiometer Value: {pot_value}")

                       # --- This is the Task 1 Logic ---
                       # Check if the value is in the upper half
                       if pot_value > 512:
                           # Send the '1' byte to turn the LED ON
                           ser.write(b'1')
                           print("--> LED ON command sent")
                       else:
                           # Send the '0' byte to turn the LED OFF
                           ser.write(b'0')
                           print("--> LED OFF command sent")

                   except ValueError:
                       # This catches any errors if the serial data was
incomplete
                       print(f"Received incomplete data: {pot_value_str}")

except serial.SerialException as e:
   # This 'except' block catches connection errors
   print(f"--- ERROR ---")
   print(f"Could not open port '{COM_PORT}': {e}")
   print("Please check the following:")
   print("1. Is your Arduino plugged in?")
   print(f"2. Is the COM_PORT variable set correctly to '{COM_PORT}'?")
   print("3. Is the Arduino IDE's Serial Monitor closed?")

except KeyboardInterrupt:
   # This 'except' block catches when you press Ctrl+C
   print("\nProgram stopped by user. Serial connection closed.")
```

3.4 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

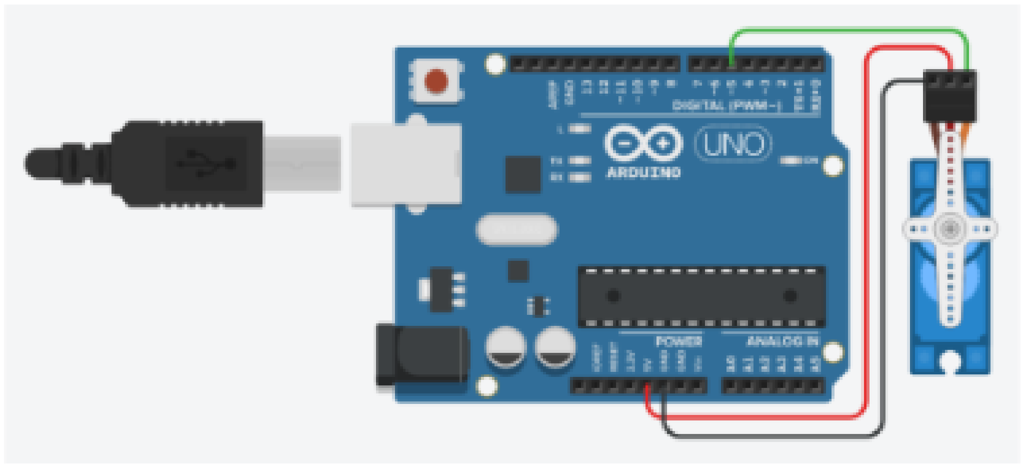
A real-time graph will appear, displaying the potentiometer values as they change. The graph updates every 100ms, showing a smooth transition of values.

4.0 Experiment 2

4.1 Materials and Equipment

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

4.2 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.

4.3 Methodology

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.

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.

Code Arduino

```
#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

}
}
```

Code Python Servo Motor

```python
import serial
import time

# --- CONFIGURATION ---
# !!! IMPORTANT !!!
# Update this to your Arduino's COM port (you found it was 'COM5')
COM_PORT = 'COM5'
BAUD_RATE = 9600
# ---------------------

# NOTE: This script does NOT need to know which pin the servo is on.
# It only sends the ANGLE. The Arduino code handles the pin.

print(f"Connecting to {COM_PORT}...")

try:
   # Use 'with' to automatically open and close the serial port
   with serial.Serial(COM_PORT, BAUD_RATE, timeout=1) as ser:

       print(f"Successfully connected to {ser.name}")
       time.sleep(2)  # Wait for the Arduino to reset

       print("Enter a servo angle (0-180) and press Enter.")
       print("Type 'q' and press Enter to quit.")

       while True:
           # Get user input from the keyboard
           angle_input = input("Enter servo angle: ")

           # Check if the user wants to quit
           if angle_input.lower() == 'q':
               print("Quitting program.")
               break  # Exit the 'while True' loop

           try:
               # Convert the input to an integer
               angle = int(angle_input)

               # Check if the angle is in the valid range
               if 0 <= angle <= 180:
                   # Send the angle to the Arduino
                   # We add a newline '\n' so Serial.parseInt() knows
                   # when the number is finished.
                   ser.write(f"{angle}\n".encode('utf-8'))
                   print(f"Sent angle: {angle}")
               else:
                   print("Error: Angle must be between 0 and 180.")
```

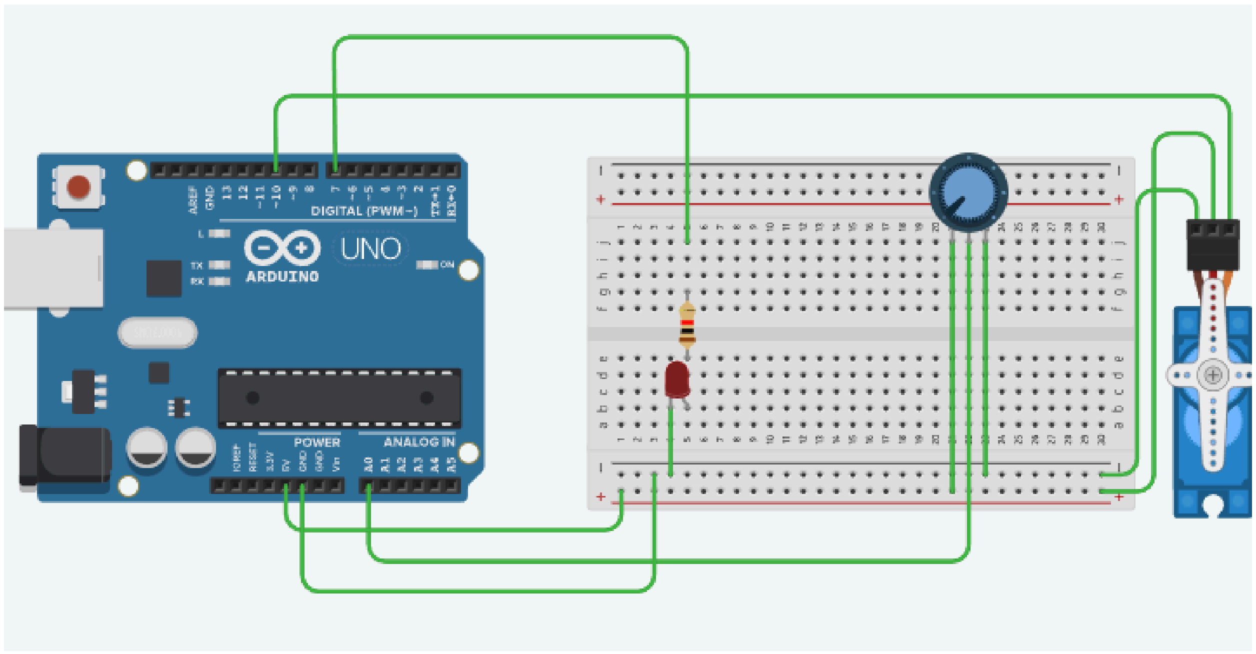
```python
            except ValueError:
                print("Error: Please enter a valid number or 'q'.")


except serial.SerialException as e:
    # This 'except' block catches connection errors
    print(f"--- ERROR ---")
    print(f"Could not open port '{COM_PORT}': {e}")
    print("Please check the following:")
    print("1. Is your Arduino plugged in?")
    print(f"2. Is the COM_PORT variable set correctly to '{COM_PORT}'?")
    print("3. Is the duino IDE's Serial Monitor closed?")

except KeyboardInterrupt:
    print("\nProgram stopped by user. Serial connection closed.")
```

5.0 TASK 2: Transmitting angle data from Python script to an Arduino which then actuates a servo to move to the specified angle

5.1 Materials and Equipment

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
7. LED (1x)
8. 220 Ohm resistors (1x)

5.2 Experimental Setup



5.3 Methodology

Connect the Potentiometer:

- The middle pin (wiper) of the potentiometer is connected to analog pin A0 on the Arduino UNO.
- One side pin of the potentiometer is connected to 5V on the Arduino to supply power.
- The other side pin of the potentiometer is connected to GND on the Arduino.

Connect the LED:

- The anode (long leg) of the LED is connected to digital pin 7 through a current-limiting resistor (220Ω–330Ω).
- The cathode (short leg) of the LED is connected to GND.

Connect the Servo Motor:

- The signal wire (usually orange or yellow) of the servo is connected to digital pin 9, which supports PWM control.
- The power wire (red) of the servo is connected to the 5V pin on the Arduino.
- The ground wire (brown or black) of the servo is connected to GND.

Power and Ground Connections:

- The breadboard power rail (+) is connected to Arduino 5V.
- The breadboard ground rail (−) is connected to Arduino GND.

Code Used:

Arduino

```
/*
  Task 2 - Arduino Code
  - Reads a potentiometer on pin A0.
  - Controls a servo motor on pin 9 based on the pot value.
  - Lights up an LED on pin 7 if the pot value is over 512.
  - Sends the raw pot value over serial for Python to read.
*/

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

// --- Define our hardware pins ---
const int POT_PIN = A0;   // Potentiometer wiper (analog in)
const int SERVO_PIN = 10;  // Servo signal (PWM)
const int LED_PIN = 7;     // LED anode (digital out)

// --- Create objects ---
Servo myServo; // Create a servo object

void setup() {
  // Initialize Serial communication at 9600 baud
  Serial.begin(9600);

  // Attach the servo to its pin
  myServo.attach(SERVO_PIN);

  // Set the LED pin as an output
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  // 1. Read the potentiometer value (range: 0 to 1023)
  int potValue = analogRead(POT_PIN);

  // 2. Map the pot value to a servo angle (range: 0 to 180)
  int servoAngle = map(potValue, 0, 1023, 0, 180);

  // 3. Write the angle to the servo
  myServo.write(servoAngle);

  // 4. Control the LED (Part 2 of Task)
  // Turn LED ON if pot value is in the upper half ( > 512)
  if (potValue > 512) {
    digitalWrite(LED_PIN, HIGH); // Turn LED ON
  } else {
    digitalWrite(LED_PIN, LOW);  // Turn LED OFF
  }

  // 5. Send the raw potentiometer value to Python
  // We send the raw potValue so the graph plots 0-1023
  Serial.println(potValue);

  // 6. Add a small delay
  // This prevents flooding the serial port and gives the
  // Python script time to plot. 50ms is a good starting point.
  delay(50);
}
```

Python

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

# --- CONFIGURATION ---
# !!! IMPORTANT !!!
# Update this to your Arduino's COM port.
# Find it in Arduino IDE > Tools > Port
# Examples: 'COM3' (Windows), '/dev/ttyACM0' (Linux/Mac)
COM_PORT = 'COM5'   # <--- CHANGE THIS
BAUD_RATE = 9600
# ---------------------

# --- Matplotlib Plot Setup ---
plt.ion()  # Turn on interactive mode for real-time plotting
fig, ax = plt.subplots()
fig.canvas.manager.set_window_title("Potentiometer Real-Time Plot")
```

```python
ax.set_xlabel("Time (samples)")
ax.set_ylabel("Potentiometer Value (0-1023)")
ax.set_ylim(0, 1050)  # Set Y-axis limit (0-1023, with a little extra room)

x_vals, y_vals = [], []
line, = ax.plot(x_vals, y_vals, 'r-')  # 'r-' is a red line


# ----------------------------


def run_plotter():
   print(f"Connecting to {COM_PORT} at {BAUD_RATE} baud...")

   try:
       # Connect to Serial port.
       # The 'timeout=2' means ser.readline() will wait up to 2 seconds
       # for a new line before timing out.
       with serial.Serial(COM_PORT, BAUD_RATE, timeout=2) as ser:
           print(f"Successfully connected to {ser.name}")
           time.sleep(2)  # Wait for Arduino to reset after connection
           ser.flushInput()  # Clear any old data in the buffer

           print("Plotting started. Turn the potentiometer.")
           print("Press Ctrl+C in this terminal to stop.")

           while True:
               try:
                   # Read a line from the serial port
                   # .decode('utf-8') converts bytes to string
                   # .strip() removes whitespace (like '\r\n')
                   serial_line = ser.readline().decode('utf-8').strip()

                   if serial_line:  # Check if we got any data
                       # Convert the string to an integer
                       pot_value = int(serial_line)

                       # --- Update Plot Data (Part 3) ---
                       y_vals.append(pot_value)
                       x_vals.append(len(y_vals))  # X-axis is just the sample
number

                       # Update the plot's data
                       line.set_data(x_vals, y_vals)

                       # Rescale the plot axes
                       ax.relim()
                       ax.autoscale_view()
```

```
                        # Redraw the plot
                        fig.canvas.draw()
                        fig.canvas.flush_events()

                        # Print to console (optional)
                        # print(f"Potentiometer Value: {pot_value}")

                except ValueError:
                    print(f"Warning: Could not convert to int. Got:
'{serial_line}'")
                except Exception as e:
                    print(f"An error occurred: {e}")

    except serial.SerialException as e:
        print(f"--- ERROR ---")
        print(f"Failed to connect to {COM_PORT}: {e}")
        print("Please check the following:")
        print("1. Is the Arduino plugged in?")
        print(f"2. Is the COM_PORT variable set correctly to '{COM_PORT}'?")
        print("3. Is the Arduino IDE's Serial Monitor closed?")

    except KeyboardInterrupt:
        print("\nPlotting stopped by user (Ctrl+C).")

    finally:
        plt.ioff()  # Turn off interactive mode
        print("Serial connection closed. Exiting.")
        if y_vals:  # If we have data, show the final static plot
            print("Showing final plot. Close the plot window to exit
completely.")
            plt.show()


# --- Run the main function ---
if __name__ == "__main__":
    run_plotter()
```

5.4 Result

The servo motor accurately responded to the control signals, rotating to the specified angles with smooth and stable motion. Throughout the experiment, it operated consistently. The response time was fast, allowing the motor to reach target positions with minimal delay. The graphical output produced a clear and repetitive waveform, confirming reliable servo performance under the test conditions.

6.0 Discussion

This experiment showed a clear relationship between the potentiometer input and the output of both the LED and the servo motor. Turning the potentiometer changed the LED brightness and the servo position proportionally. However, a few issues were observed, such as slight LED flickering and minor delays in servo movement.

Some sources of error and limitations included electrical noise, which caused unstable readings from the potentiometer, leading to LED flicker and servo jitter. The system showed a non-linear response, as changes in LED brightness and servo movement were not always proportional to the potentiometer's rotation. The limited input range of the potentiometer prevented it from generating the full range of analog values.

Even with these limitations, the experiment successfully demonstrated how real-time control can be achieved using PWM signals and a simple input device like a potentiometer.

7.0 Conclusion

This experiment successfully demonstrated two key applications of serial communication, which are transmitting potentiometer data and controlling external devices such as an LED and a servo motor using Python. The results confirmed that serial communication enables efficient real-time data transfer between the Arduino and a computer via USB. In this setup, the Arduino accurately read analog input from the potentiometer and sent the data to Python for processing. Furthermore, Task 1 and Task 2 illustrated how the potentiometer could control the LED brightness and servo motor position by varying resistance. These outcomes highlight that serial communication not only facilitates data transfer but also enables command execution for physical hardware components. Overall, this experiment reinforces the importance of serial communication in engineering applications, particularly within industrial automation, IoT-based smart systems, and real-time monitoring technologies. Gaining proficiency in serial data handling forms a strong foundation for developing advanced control systems where sensors, actuators, and microcontrollers must operate in coordination.

8.0 Recommendation

This experiment could be enhanced by improving real-time graphical visualization of potentiometer readings and corresponding servo motor movements. This feature would improve user interaction and system monitoring by minimizing manual hardware adjustments and enabling more intuitive computer-based control. Additionally, data accuracy and responsiveness can be improved by applying a moving average filter to the potentiometer input. Since raw potentiometer values often fluctuate, this filtering method smooths the signal before graphical display, resulting in more reliable output data.

9.0 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/

10. Acknowledgement

I would like to express my sincere gratitude to Dr. Wahju Sediono, Dr. Zulkifli Bin Zainal Abidin, my teaching assistant, and my peers for their invaluable guidance and support throughout the completion of this report. Their insights, feedback, and expertise greatly contributed to the depth and quality of this work. I truly appreciate their time, patience, and commitment to my academic growth.

11.0 Student's Declaration

Certificate of Originality and Authenticity

This is to certify that we are responsible for the work submitted in this report, that the original work is our own except as 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.

Name: Shamsul Hakimee Bin Shamsul Hairee	Read	[/]
Matric Number: 2315027	Understand	[/]
Signature: [illegible]	Agree	[/]

Name: Qashrina Aisya binti Mohd Nazarudin	Read	[/]
Matric Number: 2315184	Understand	[/]
Signature: [illegible]	Agree	[/]

Name: Nurel Humairah Binti Muhammad Firdaus	Read	[/]
Matric Number: 2315680	Understand	[/]
Signature: [illegible]	Agree	[/]