

**LABORATORY REPORT**

**MECHATRONICS SYSTEM INTEGRATION MCTA 3203**

**SEMESTER 2 2023/2024**

**WEEK 3**

**GROUP: 6**

|  |  |  |
| --- | --- | --- |
|  | **NAME** | **MATRIC NUMBER** |
| 1. | MUHAMMAD NABIL IMAN BIN ABD RAHMAN | 2313551 |
| 2. | MUHAMMAD NAZHAN BIN MOHAMED NADZRI | 2313703 |
| 3. | MUHAMMAD HAZIQ AJMAL BIN MD KAMAL | 2312821 |

**Table of Contents**

[Abstract 3](#_Toc193058084)

[Introduction 3](#_Toc193058085)

[Materials and Equipment 3](#_Toc193058086)

[Methodology 5](#_Toc193058087)

[Question 6](#_Toc193058088)

[Discussion 6](#_Toc193058089)

[Conclusion 7](#_Toc193058090)

[Recommendations 7](#_Toc193058091)

[Appendix 7](#_Toc193058092)

[Acknowledgement 9](#_Toc193058093)

[Declaration 9](#_Toc193058094)

# Abstract

# This experiment are divided into two parts which is 3a and 3b. This experiment shows how to

# create a serial communication link between an Arduino microcontroller and Python software so

# that potentiometer readings may be transmitted in real time. When both are linked, this experiment

# setup is ideal for using real-time data in Python programs. The findings of this experiment set the groundwork for future applications and providing a fundamental understanding of serial

# communication between microcontroller and computers.

# Introduction

The experiments demonstrate how Python communicates serially with Arduino for control purposes. Experiment 3a established serial communication between an Arduino potentiometer reading that USB transferred to a Python script.

Experiment 3b builds on this by using Python to control a servo motor connected to Arduino. Here, Python sends angle data to Arduino, which then moves the servo to the specified position.

Thus, data exchange is explained as the process which allows microcontrollers and computers to share information.

**Week 3a: Parallel, Serial and USB interfacing with microcontroller and computer based system (1): Sensors and actuators.**

# Materials and Equipment

• Arduino Board

• Potentiometer

• Jumper Wires

• LED

• 220Ω resistor

• Breadboard

**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. An example of the circuit setup is shown in Fig. 1.

# Methodology

1. Connect the Arduino to your computer via a USB cable.

2. Power on the Arduino (upload the sketch to your Arduino using the Arduino IDE)

3. Run the Python script on your computer.

4. As you turn the potentiometer knob, you should see the potentiometer readings displayed in the

Python terminal.

5. You can use these readings for various experiments, data logging, or control applications,

depending on your project requirements

Using the Arduino Serial Plotter (Please note that if you are working with Python to read data from your Arduino, it's important not to open the Arduino Serial Plotter simultaneously. The Arduino Serial Plotter is a dedicated tool for visualizing data received from the Arduino board and may interfere with Python's access to the serial port. If you intend to use Python to read and process data from the Arduino, ensure that the Serial Plotter is closed or not in use while running your Python script to maintain uninterrupted communication between Python and the Arduino.):

6. Open the Serial Plotter: In the Arduino IDE, go to "Tools" -> "Serial Plotter."

7. Select the Correct COM Port: In the Serial Plotter, select the correct COM port to which your

Arduino is connected.

8. Set the Baud Rate: Ensure that the baud rate in the Serial Plotter matches the one set in your

Arduino code (e.g., 9600).

9. Read Real-Time Data: As you turn the potentiometer knob, the Serial Plotter will display the

potentiometer readings in real-time, creating a graphical representation of the data. You can see

how the values change as you adjust the potentiometer.

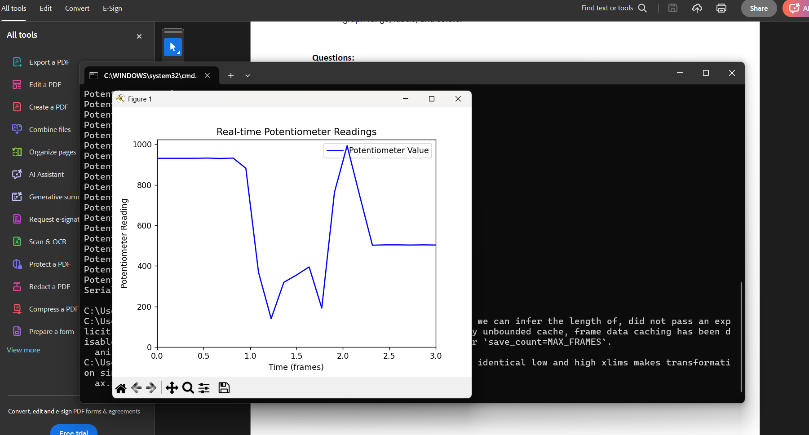
10. Customize the Plotter: You can customize the Serial Plotter by adjusting settings such as the

graph range, labels, and colors.

**Results**

A circuit board with wires and wires

AI-generated content may be incorrect.****

****

# 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).**

Through execution of the Python script we verified serial communication by watching real-time output of potentiometer data from knob motions. The matplotlib library generated a line graph which displayed that the potentiometer's position affected its measurements in real time thus enabling applications to use live data feedback.

# Discussion

1. **Hardware**

Connect the potentiometer to the Arduino. Wire one end of the potentiometer to 5V, the other end to GND, and the center pin to an analog input pin on the Arduino (e.g., A0). The attempt to maintain a steady motion with the potentiometer knob resulted in irregular slopes on the real-time graph. As the user turned the knob clockwise the graph rose and counterclockwise movement caused the slope to drop yet this behavior was not strictly linear.

The observed linearity failed due to stopped-instantaneous motions. Improved design features for the potentiometer would enable better control over experiments by producing smoother data visualization with higher accuracy levels.

1. **Software**

Arduino IDE is used to assemble the code for this project as it provides a robust environment for writing and uploading code to the Arduino. Arduino IDE functions to control the blinking rate of the LED by adjusting the potentiometer value.

Arduino Code:

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 is also use to read the data form the Arduino via the serial port. A Python script used effectively to establish a serial connection using the pyserial library, capturing potentiometer readings from the Arduino.

Install the pyserial library.

pip install pyserial

Python script:

import serial

# Replace 'COM10' with your Arduino's actual serial port (e.g., 'COM3', 'COM4' for Windows)

ser = serial.Serial('COM10', 9600, timeout=1)

try:

    while True:

        pot\_value = ser.readline().decode().strip()

        if pot\_value:

            print("Potentiometer Value:", pot\_value)

except KeyboardInterrupt:

    ser.close()

    print("Serial connection closed.")|  
  
  
code to plot  
  
import serial

import matplotlib.pyplot as plt

import matplotlib.animation as animation

# Set up the serial connection (Change 'COM3' to your port)

ser = serial.Serial('COM10', 9600, timeout=1)

# Store data

time\_values = []

pot\_values = []

max\_points = 100  # Maximum points to show on the graph

# Set up plot

fig, ax = plt.subplots()

ax.set\_ylim(0, 1023)  # Potentiometer range (0-1023 for Arduino)

ax.set\_xlim(0, max\_points)

line, = ax.plot([], [], 'b-', label="Potentiometer Value")

ax.legend()

# Function to update plot

def update(frame):

    try:

        data = ser.readline().decode('utf-8').strip()

        if data:

            pot\_value = int(data)

            time\_values.append(len(time\_values))

            pot\_values.append(pot\_value)

            # Keep only last 'max\_points' values

            if len(time\_values) > max\_points:

                time\_values.pop(0)

                pot\_values.pop(0)

            # Update plot

            line.set\_data(time\_values, pot\_values)

            ax.set\_xlim(time\_values[0], time\_values[-1])

    except:

        pass

    return line,

# Animate the plot

ani = animation.FuncAnimation(fig, update, interval=100, blit=True)

plt.xlabel("Time (frames)")

plt.ylabel("Potentiometer Reading")

plt.title("Real-time Potentiometer Readings")

plt.show()

# Close serial connection on exit

ser.close()

# Conclusion

The project aims to established serial communication and control between Python and Arduino. We sent potentiometer readings from Arduino to Python via USB connection, demonstrating simple data exchange over USB.

# Recommendations

# Future test runs should include additional sensing components and movable components to enable the advancement of sophisticated digital interactions through environmental condition-based LED brightness changes. Research becomes more advanced by implementing multiple LEDs with varying color combinations and lighting intensity levels. The addition of wireless communication components which includes Bluetooth or Wi-Fi enables users to operate the LED system remotely through mobile devices.

**Week 3b: Parallel, Serial and USB interfacing with microcontroller and computer based system (1): Sensors and actuators.**

# Materials and Equipment

* Arduino board (e.g., Arduino Uno)
* Servo motor
* Jumper wires
* Potentiometer (for manual angle input)
* USB cable for Arduino
* Computer with Arduino IDE and Python installed

**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 of +5V. You can connect the servo's power wire (usually red) to the 5V output on the Arduino board.
3. Connect the Servo's Ground Wire: Connect the servo's ground wire (usually brown) to one on the ground (GND) pins on the Arduino.

# Methodology

Install Arduino Libraries:

* Open the Arduino IDE.
* Go to "Sketch" > "Include Library" > "Servo" to install the Servo library if it's not already installed.

Write the Arduino Code:

* Open a new sketch in the Arduino IDE.
* Write the Arduino code that reads angle data from the serial port and moves the servo accordingly.

#include <Servo.h>

Servo servo;

int angle = 90;

void setup() {

servo.attach(9); // Attach the servo to dig. pin 9

}

void loop() {

servo.write(angle); // Set to the desired angle

delay(1000); // Wait for 1 second

angle = 180 - angle; // Reverse the angle (e.g., 90 to

90 degrees)

}

* Upload the code to your Arduino board.

Install Required Python Libraries:

* Open a terminal or command prompt.
* Install the pyserial library using pip, if you haven’t it:

pip install pyserial

Write the Python Script:

* Create a new Python script using a code editor (e.g., Visual Studio Code, PyCharm, or a text editor).

import serial

import time

"""

Define the serial port and baud rate

(adjust the port as per your Arduino)

"""

ser = serial.Serial('COM3', 9600)

try:

while True:

angle = input("Enter servo angle (0-180 degrees): ")

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.")

Run the Python Script:

* Save the above Python script with a .py extension.
* Run the Python script, which will prompt you to enter the servo angle. You can enter an angle between 0 and 180 degrees to control the servo.

Experiment with Angle Input:

* Input an angle between 0 and 180 degrees and press Enter. The Python script will send the angle to the Arduino over the serial port.
* The Arduino will move the servo to the specified angle, and the script will display the angle set by the servo.
* You can exit the Python script by entering 'q' and observing that the serial connection is closed.
* You can input multiple angles to see the servo move accordingly.

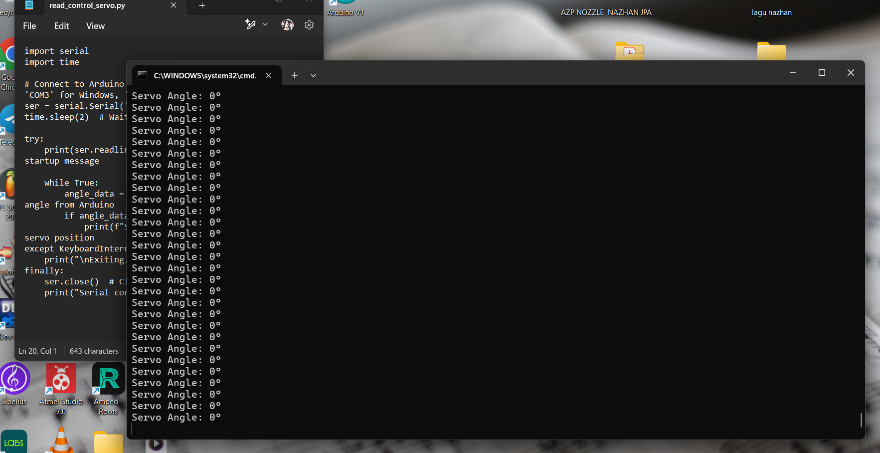
Exit the Python Script:

* When you're done with the experiment, enter 'q' to exit the Python script.

**Results**

A person holding a hand to a circuit board

AI-generated content may be incorrect.A person sitting at a table with wires

AI-generated content may be incorrect.****

# Question

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.

**“coding”**

**Code to control servo Arduino**

#include <Servo.h>

Servo servo;

int angle = 90; // Default angle

void setup() {

    Serial.begin(9600);  // Start serial communication

    servo.attach(9);     // Attach the servo to digital pin 9

    servo.write(angle);  // Move servo to default position

    Serial.println("Arduino Ready"); // Send confirmation to Python

}

void loop() {

    if (Serial.available() > 0) {  // If data is available

        String input = Serial.readStringUntil('\n'); // Read until newline

        input.trim(); // Remove any whitespace or newlines

        int receivedAngle = input.toInt(); // Convert to integer

        if (receivedAngle >= 0 && receivedAngle <= 180) { // Validate range

            servo.write(receivedAngle); // Move the servo

            Serial.print("Servo moved to: ");

            Serial.println(receivedAngle);

        } else {

            Serial.println("Invalid angle received");

        }

    }

}

**Code to read control servo**

#include <Servo.h>

Servo servo;

int potPin = A0;  // Potentiometer connected to A0

int angle = 0;    // Servo position

void setup() {

    Serial.begin(9600);  // Start serial communication

    servo.attach(9);     // Attach the servo to digital pin 9

    Serial.println("Arduino Ready");

}

void loop() {

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

    angle = map(potValue, 0, 1023, 0, 180);  // Convert to servo angle (0-180)

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

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

    delay(100);  // Small delay to stabilize readings

}

**Using phyton**

import serial

import time

# Define the serial port and baud rate (adjust 'COM10' to your Arduino port)

ser = serial.Serial('COM10', 9600, timeout=1)

time.sleep(2)  # Allow time for Arduino to reset

def send\_angle(angle):

    """Send the servo angle to Arduino"""

    ser.write(f"{angle}\n".encode())  # Send data with newline

    ser.flush()  # Ensure data is sent

    time.sleep(0.1)  # Give Arduino time to process

try:

    print(ser.readline().decode().strip())  # Read Arduino's startup message

    while True:

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

        if angle.lower() == 'q':

            break

        try:

            angle = int(angle)

            if 0 <= angle <= 180:

                send\_angle(angle)

                print(ser.readline().decode().strip())  # Read Arduino response

            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.")

**To control servo using phyton**

import serial

import time

# Connect to Arduino (Change 'COM10' to your port, e.g., 'COM3' for Windows, '/dev/ttyUSB0' for Linux)

ser = serial.Serial('COM10', 9600, timeout=1)

time.sleep(2)  # Wait for Arduino to initialize

try:

    print(ser.readline().decode().strip())  # Read Arduino startup message

    while True:

        angle\_data = ser.readline().decode().strip()  # Read angle from Arduino

        if angle\_data:

            print(f"Servo Angle: {angle\_data}°")  # Display servo position

except KeyboardInterrupt:

    print("\nExiting...")

finally:

    ser.close()  # Close serial connection

    print("Serial connection closed.")

# Discussion

1. **Hardware**

Physical control of the servo motor using the potentiometer yielded predictable movements but revealed a few limitations. Due to quick or abrupt turns of the potentiometer, the servo occasionally struggled to match the real-time angle adjustments instantly. Replacing the current potentiometer with one that offers finer rotational control may allow smoother servo adjustments and improve system responsiveness

1. **Software**

The use of Python to control the servo motor was successful, with angle values sent over serial communication to the Arduino, which then positioned the servo accordingly. The addition of a key press in the Arduino script to halt operation added a convenient troubleshooting mechanism, as it allowed for easy program reset and interruption of the servo control if necessary. Moving forward, integrating a graphical user interface (GUI) in Python could make angle adjustments more intuitive and user-friendly.

# Conclusion

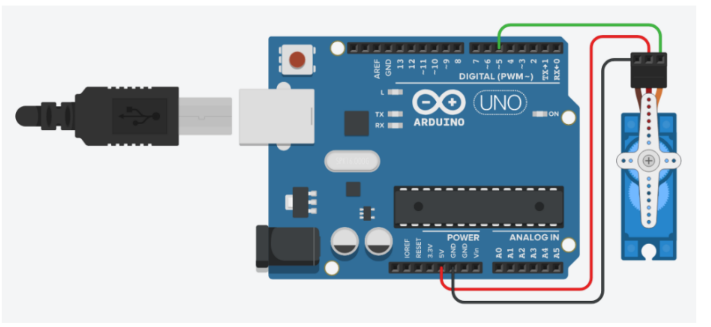
In Experiment 3b, we used Python to control a servo motor through Arduino, illustrating application of serial communication on actuator control. These results matched our expectations and strengthened our understanding of basic data transfer and control techniques.

# Recommendations

Replacing the potentiometer with finer control or higher precision could result in smoother

adjustments, minimizing the abrupt changes that impacted the servo’s responsiveness.

# Appendix

A diagram of a circuit board

AI-generated content may be incorrect.

3b

# Acknowledgement

I would like to express my sincere gratitude to the lab technician for their invaluable guidance, support, and encouragement throughout this project. Their expertise and insights have been instrumental in shaping the direction of this work. I would also like to extend my thanks to my fellow peers for their assistance and collaboration, which greatly contributed to the successful completion of this project.

# Declaration

We hereby declare that the work presented in this report is entirely my own, except where

otherwise acknowledged. I affirm that I have adhered to the principles of academic integrity and have not engaged in any form of plagiarism or unethical conduct in the completion of this project. All sources of information and assistance used in this work have been properly cited and acknowledged.