REPORT 2 : PARALLEL, SERIAL AND USB INTERFACING WITH MICROCONTROLLER AND COMPUTER BASED SYSTEM (SENSORS AND ACTUATORS)

GROUP 4

MCTA 3203

SEMESTER 1 2024/2025

MECHATRONICS SYSTEM INTEGRATION

DATE OF SUBMISSION: 30 OCTOBER 2024

NO	NAME	MATRIC NUMBER
1.	IRDINA NABIHAH BINTI MOHD NAZRI	2214772
2.	KHALISAH AMANI BINTI YAHYA AZMI	2218184
3.	LUQMAN AZFAR BIN AZMI	2219857
4.	MOHAMAD NASRI BIN MOHAMAD NAZRI	2219879
5.	MUHAMAD NURHAKIMIE THAQIF BIN ABDULLAH	2213217

TABLE OF CONTENT

INTRODUCTION

This experiment was conducted to investigate how Python and Arduino interact to effect proper control of electronic modules using serial communication. In this project, our key emphasis is placed on two main subjects: first, we will build up a scheme of data exchange whereby an Arduino sends potentiometer readings to a Python script via a USB connection. The aspect will explain how analog input can be captured, processed, and used for monitoring and control purposes.

The second part of the experiment involves servo motor control with the help of Python. If we pass the angle data from a Python script over to Arduino, then we are in a position to actuate the servo motor for movement to certain angles. We can, therefore, illustrate how we can practically do servo control in robotics and automation applications.

In this lab, we can go further into serial communication protocols, the function of analog sensors like potentiometers, and how servo motors work. This will help know the theory behind how these components interact.

ABSTRACT

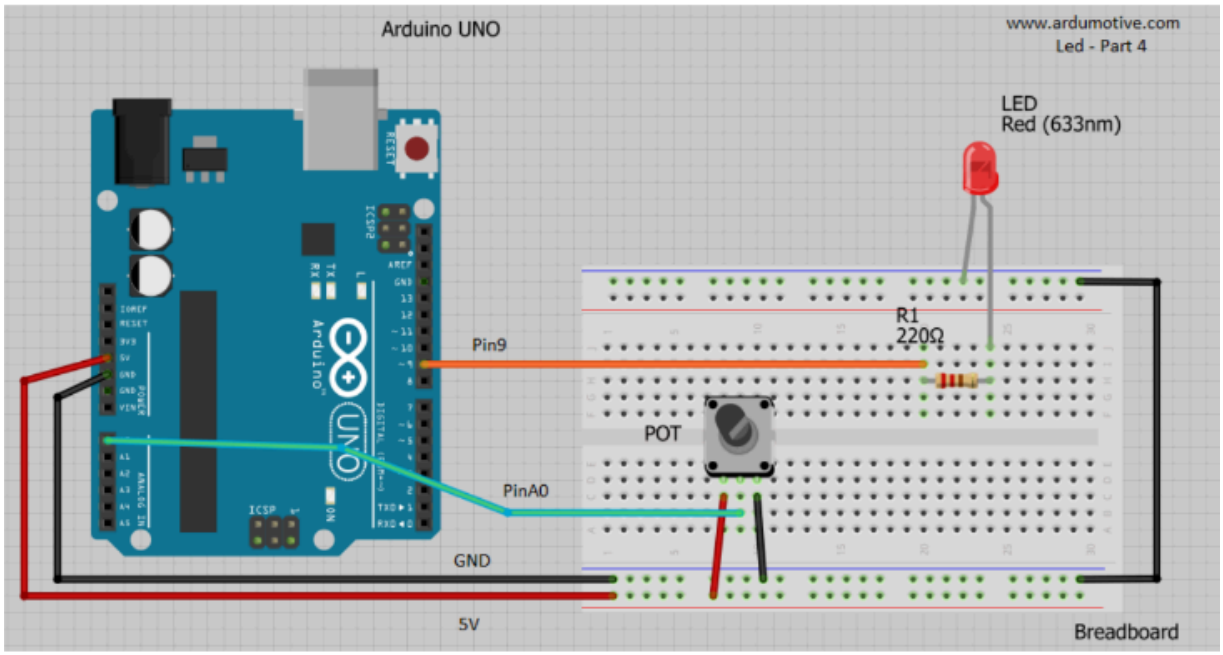
This experiment aims to demonstrate the interfacing of a potentiometer and a servo motor with an Arduino Uno, facilitated through serial communication with a Python script. The primary objective is to understand how to read analog inputs from a potentiometer and control the position of a servo motor based on angle data transmitted from Python. The methodology involves connecting the potentiometer to an analog input pin on the Arduino and setting up the servo motor on a PWM output pin. The Arduino code reads the potentiometer values and sends them via serial communication, while the Python script interprets this data to actuate the servo to specified angles.The experiment concludes with a successful demonstration of potentiometer readings influencing servo motor positioning.

EXPERIMENT 3A: <u>Sending potentiometer readings from an Arduino to a Python script through a USB connection.</u>

MATERIALS AND EQUIPMENT

- Arduino Uno board
- Potentiometer
- 220-ohm resistors
- LED
- Jumper wires
- Breadboard

EXPERIMENTAL SETUP



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.

METHODOLOGY

1. Hardware setup

-Connect the Potentiometer to the Arduino:

- Begin by identifying the three terminals of the potentiometer. The left leg will be connected to the 5V output pin on the Arduino. This connection provides the necessary power for the potentiometer to function correctly.
- Next, connect the right leg of the potentiometer to the GND (ground) pin on the Arduino. This ensures that the potentiometer has a complete circuit and can function properly by referencing the ground voltage.
- The middle leg of the potentiometer, known as the wiper, is where the variable voltage will be read. Connect this wiper to an analog input pin on the Arduino, such as A0. This pin will receive the voltage signal that changes as the potentiometer is adjusted, allowing the Arduino to capture real-time readings.

2. Software Programming and Execution

- Connecting the Arduino

- connect the Arduino Uno to your computer using a USB cable. This connection not only powers the Arduino but also allows for data transmission between the Arduino and the computer

- Uploading the Arduino Sketch

- initializing the pins connected to the potentiometer and any additional components that used in the experiment
- Implement a loop that reads the potentiometer value at regular intervals.
- After reading the potentiometer value, send this data to the serial port.

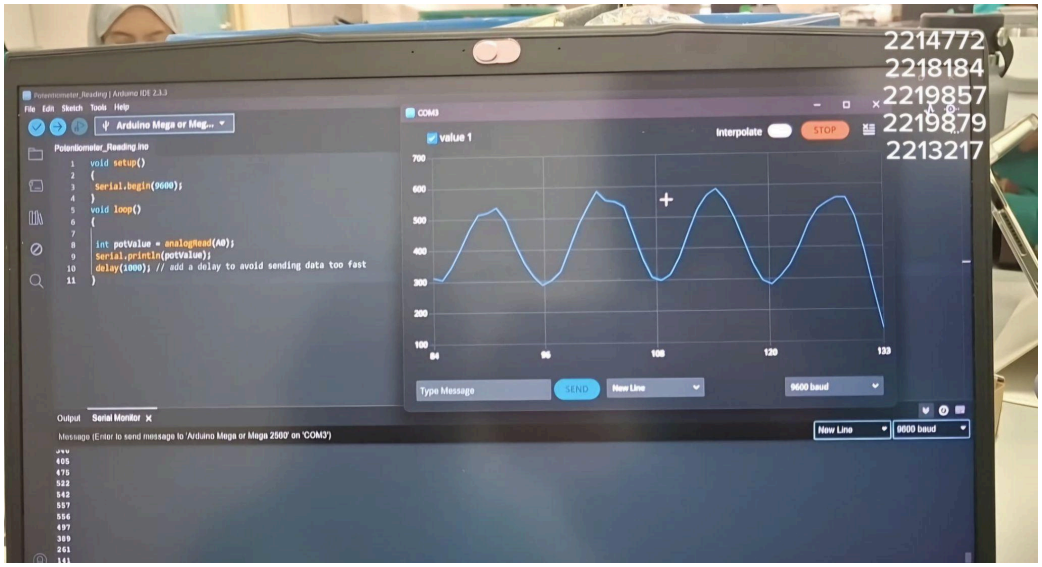
- Running the Python Script

- To executing the script, ensure that any necessary libraries, such as pyserial are installed
- After the Python script is running, observe that python makes a connection with the Arduino.
- As you turn the potentiometer knob, the Python terminal will display the corresponding potentiometer readings in real time.

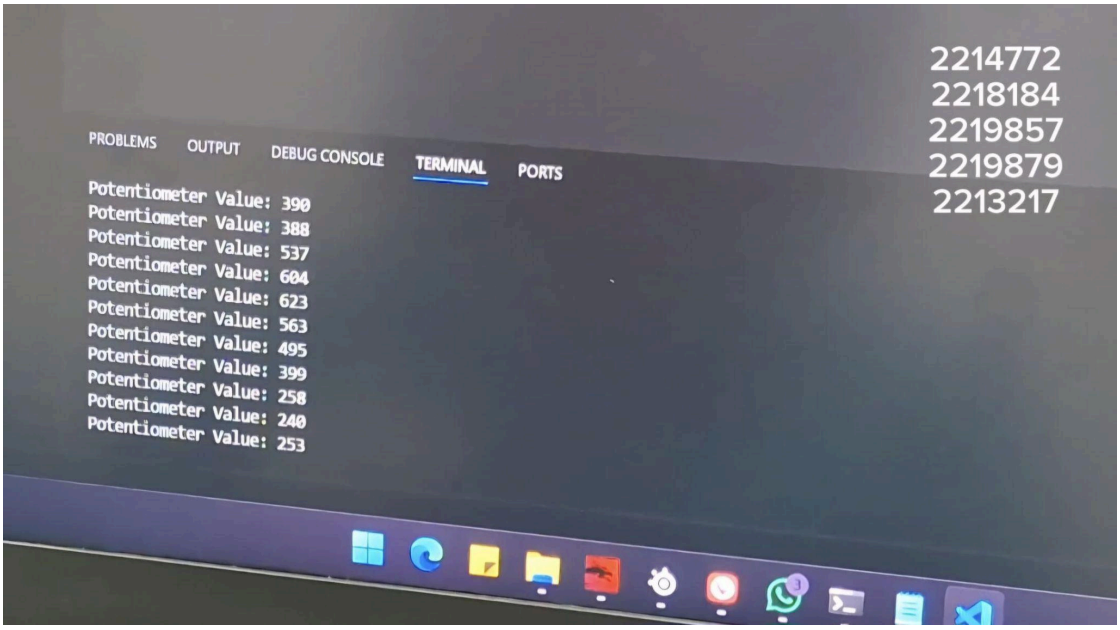
- Application of Readings

- The potentiometer readings can be utilized for various experimental purposes, data logging, or control applications, depending on project requirements.

RESULTS



Arduino Code



Potentiometer values in Python script

Video Link: https://github.com/nasrinazri/Group-4-lab-/tree/main/Week%203/A

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).**

Numpy provides efficient data structures for managing the arrays of time and angle data, which is essential for real-time processing. Numpy's array handling makes it quick to store and update numerical data points continuously.

Matplotlib is used to create a live plot of the potentiometer's angle data, giving users a real-time view of any adjustments. The pyplot module allows for flexible plot design, while FuncAnimation continuously updates the graph as new data arrives, creating a smooth scrolling effect.

The Serial library (from pyserial) manages the connection to the potentiometer, enabling Python to read angle data from the microcontroller over the specified serial port. Functions like Serial() and readline() handle setup and data retrieval.

Time is used to timestamp each reading, allowing for a time-based x-axis on the plot. This provides a time series where each angle point corresponds to a moment in time, giving users insight into how the angle changes over seconds.

Finally, Re (Regular Expressions) helps filter the incoming data lines to extract only the angle values, making data processing more efficient by ignoring unrelated data.

For Function The read_angle method captures potentiometer angle data from the serial input, parsing it from lines formatted like "Servo Angle: X." If the data is valid, the angle is

added to a list of recent values along with the current timestamp, maintaining only the latest max_points entries. The update method is called every 50 milliseconds to refresh the plot, giving a scrolling effect that tracks changes in real-time.

To run the visualization, the run method uses Matplotlib's FuncAnimation, updating the plot as new data arrives. A cleanup function ensures the serial port is properly closed when the program ends. The main function initializes PotentiometerVisualizer and starts the visualization, making it easy for users to observe live potentiometer angle changes.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import serial
from time import time
import re

class PotentiometerVisualizer:
    def __init__(self, port='COM6', baudrate=9600, max_points=100):
        try:
            # Set up serial connection
            self.serial = serial.Serial(
                port=port,
                baudrate=baudrate,
                timeout=0.1,
                bytesize=serial.EIGHTBITS,
                parity=serial.PARITY_NONE,
                stopbits=serial.STOPBITS_ONE
            )
            self.serial.reset_input_buffer()
            self.serial.reset_output_buffer()
        except serial.SerialException as e:
```

```
            print(f"Error opening serial port: {e}")
            raise

        # Initialize data
        self.max_points = max_points
        self.times = np.array([])  # Time data points
        self.values = np.array([])  # Potentiometer angle values

        # Create the plot
        self.fig, self.ax = plt.subplots(figsize=(10, 6))
        self.line, = self.ax.plot([], [], color='pink', label='Servo
Angle', linewidth=2)
        self.ax.set_title('Real-time Servo Angle Readings')
        self.ax.set_xlabel('Time (seconds)')
        self.ax.set_ylabel('Angle (degrees)')
        self.ax.grid(True)
        self.ax.legend()
        self.ax.set_ylim(0, 180)  # Angle range for potentiometer

        # Start time for tracking time offsets
        self.start_time = time()

    def read_angle(self):
        """Reads the potentiometer angle from the serial input."""
        try:
            if self.serial.in_waiting > 0:
                raw_data = self.serial.readline()
                line = raw_data.decode().strip()
                match = re.search(r'Servo Angle:\s*(\d+)', line)
                if match:
                    return float(match.group(1))
                else:
                    return None
        except (serial.SerialException, UnicodeDecodeError) as e:
            print(f"Serial/Decode error: {e}")
            return None

    def update(self, frame):
        """Update function to add new data points to the graph."""
```

```
        current_time = time() - self.start_time
        current_value = self.read_angle()

        if current_value is not None:
            self.times = np.append(self.times, current_time)
            self.values = np.append(self.values, current_value)

            if len(self.times) > self.max_points:
                self.times = self.times[-self.max_points:]
                self.values = self.values[-self.max_points:]

            # Update plot line data
            self.line.set_data(self.times, self.values)
            self.ax.set_xlim(max(0, self.times[-1] - 10),
self.times[-1] + 2)

        return self.line,

    def run(self):
        """Starts the real-time plot animation."""
        anim = FuncAnimation(self.fig, self.update, frames=None,
interval=50, blit=False, cache_frame_data=False)

        plt.show()

    def cleanup(self):
        """Closes the serial port upon completion."""
        if self.serial.is_open:
            self.serial.close()
            print("Serial connection closed")

def main():
    visualizer = PotentiometerVisualizer(port='COM6')
    try:
        visualizer.run()
    except KeyboardInterrupt:
        print("Visualization interrupted by user.")
    finally:
        visualizer.cleanup()
```

```
if __name__ == "__main__":
    main()
```

Video Demonstration:

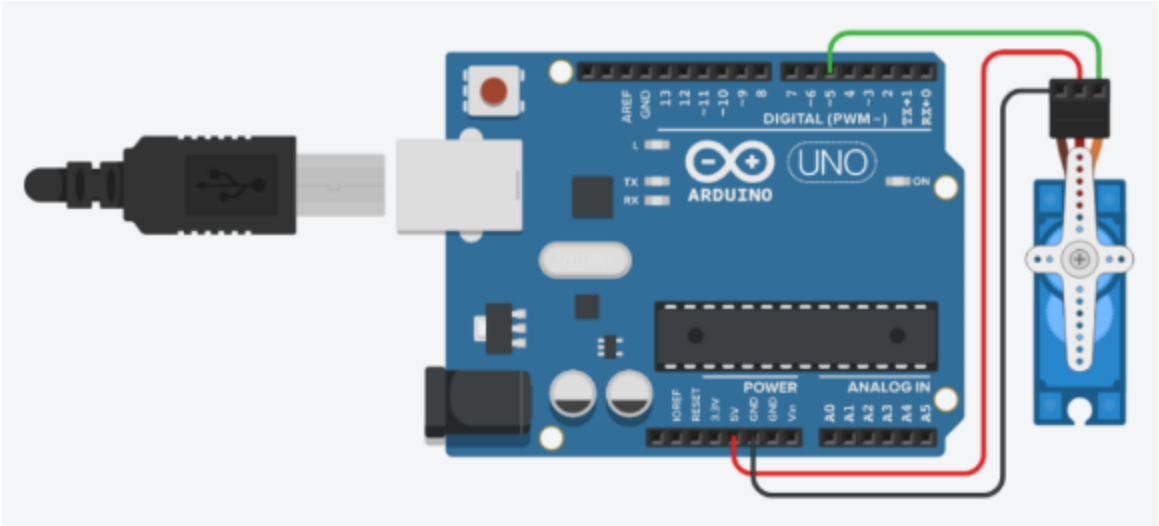
https://github.com/nasrinazri/Group-4-lab-/blob/main/Week%203/A/Potentiometer_Graph_Q3B/Q3A_Potentiometer_Graph.mp4

EXPERIMENT 3B: Transmitting angle data from a Python script to an Arduino, which then actuates a servo to move to the specified angle

MATERIALS AND EQUIPMENT

- Arduino Uno board
- Potentiometer
- Servo motor
- USB cable for Arduino
- Jumper wires
- Computer with Arduino IDE and Python installed

EXPERIMENTAL SETUP



METHODOLOGY

1. Hardware setup
    - Connect the the signal wire from the servo to an adruino digital pin
    - Connect the ground wire to the GND pin at the arduino and the servo power to the 5V pin.
    - Put the potentiometer in place by connecting the potentiometer's middle terminal to the analog input pin which is pin A0, the other terminal to the GND pin and one terminal to the 5V.
    -
2. Software setup and execution

    A. Connecting to the arduino

- connect the Arduino Uno to your computer using a USB cable. This connection not only powers the Arduino but also allows for data transmission between the Arduino and the computer
- Install the Servo library it may easier to code and control the servo movement.

B. Uploading the Arduino Sketch

- initializing the pins connected to the potentiometer and servo that were used in the experiment
- Reads angle data from the serial port to adjust the servo position as directed by the Python script.
- Reads the potentiometer value and maps it to an angle (0–180 degrees) to control the servo position in real-time.

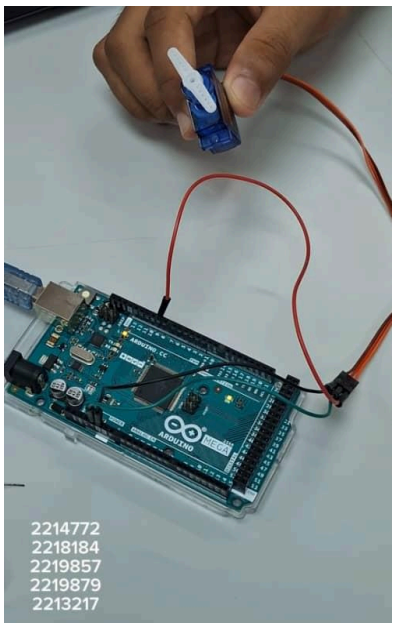
C. Running the python script

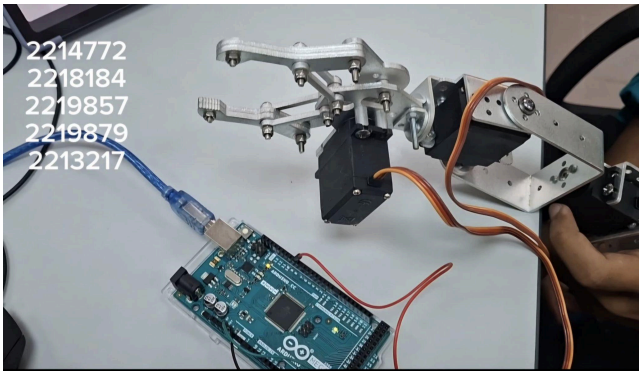
- Create a Python script that prompts the user to enter a desired angle, sends it to the Arduino through serial communication, and allows quitting the program.

D. Output of the execution of code

- Run the Python script, enter an angle (0–180 degrees) to control the servo, and terminate the program by entering 'q'.
- If no angle is entered through Python, the potentiometer’s position determines the servo’s angle, enabling dynamic movement based on potentiometer adjustments.

RESULTS





Video link : https://github.com/nasrinazri/Group-4-lab-/tree/main/Week%203/B

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.**

In the main loop, the code checks for incoming serial input. If the user sends('s' or 'S'), it sets a signal to stop further servo movement, moves the servo to a neutral 90-degree position, and enters an infinite loop, halting further execution. If no stop command is received, the potentiometer value (ranging from 0 to 1023) is read, mapped to an angle between 0 and 180 degrees, and then applied to the servo. The servo's angle is printed to the serial monitor for real-time feedback. A short delay helps stabilize the readings and movements.

```
#include <Servo.h>

Servo myServo;   // Create a servo object
int potPin = A0; // Pin where the potentiometer is connected
int potValue = 0; // Value read from the potentiometer
int angle = 0;    // Servo angle (0 - 180)
bool isRunning = true; // Flag to control program execution

void setup() {
  Serial.begin(9600); // Start serial communication
  myServo.attach(9);  // Attach the servo to pin 9
  Serial.println("Servo Control Started");
  Serial.println("Enter 's' to stop the program");
}

void loop() {
  if (Serial.available() > 0) {
    char input = Serial.read();
    if (input == 's' || input == 'S') {  // Accept both lower and uppercase 's'
```

```
      isRunning = false;
      myServo.write(90);  // Move servo to neutral position when stopping
      Serial.println("Program stopped. Reset Arduino to restart.");
      while(true) {  // Hold program in infinite loop
        delay(1000);
      }
    }
  }

  if (isRunning) {
    potValue = analogRead(potPin);            // Read the potentiometer value
(0-1023)
    angle = map(potValue, 0, 1023, 0, 180);   // Map the pot value to a range
for the servo (0-180)
    myServo.write(angle);                     // Set the servo angle

    // Print formatted output
    Serial.print("Servo Angle: ");
    Serial.println(angle);

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

Video Demonstration:

https://github.com/nasrinazri/Group-4-lab-/blob/main/Week%203/B/Servo_Potentiometer_Q3B/Q3B_Potentiometer_W_Servo.mp4

DISCUSSION

In this lab, we studied controlling a servo motor via serial communication between an Arduino and a computer. The experiment was divided into two phases. In the first, the Arduino received angle data from a Python script and used it to instruct the servo motor to rotate at the desired angle. A potentiometer for real-time servo angle control was introduced in the second section. Both approaches demonstrated different aspects of hardware control with microcontroller-based systems, emphasizing the difference between dynamic, sensor-based adjustments and programmed control inputs.

To enable the user to control the servo's position within a range of 0 to 180 degrees, we set up serial communication in the first experiment to transfer angle values from the Python script to the Arduino. This technique demonstrated how external data inputs from a computer interface can be used to manipulate hardware components by giving the servo precise, programmable control. Maintaining stable communication was crucial, and this was accomplished by balancing the baud rates of Python and Arduino, which allowed for seamless data transfer. This approach showed how automation could be programmed, but it was limited for applications that needed to react instantly to changes in the environment because it required constant user input.

In order to control the servo in real time without requiring extra computer inputs, the experiment's second section added a potentiometer as an analogue input. We could provide a continuous and instantaneous feedback loop by mapping the potentiometer's readings to angle values and then adjusting the servo's position in response to changes in the potentiometer's position. For situations like user-controlled robotic systems, where responsive adjustments are crucial, this method offered dynamic, manual control. Despite having real-time responsiveness, the potentiometer was not as programmable as Python, which made it more appropriate for manual control but less flexible for automated tasks that needed to be integrated with external commands.

CONCLUSION

In conclusion, This series of experiments demonstrates to us the principles of serial communication between a computer and a microcontroller using Python and Arduino. By interfacing a potentiometer and a servo motor with the Arduino, we explored two fundamental aspects of sensors and actuators in embedded systems.

In the first experiment, we were able to configure a potentiometer to send analog readings to a Python script using a USB connection. This setup allowed us to monitor real-time data, highlighting the ease with which analog inputs can be integrated into software applications for various purposes, such as data logging or control systems.

The second experiment done was the servo motor control, which was dependent on the angle input from a Python script to the Arduino. This communication highlighted a very important feature of PWM signals and how software can change physical elements. Changing the angle with a potentiometer allowed us to show exactly how input devices can be used to manipulate outputs in a controlled manner.

These experiments, in general, demonstrate the importance of serial communication in bridging the hardware-software gap. The projects, upon successful execution, prepare us for applications that are on the advanced level. This reinforces our understanding of programming in microcontrollers and also data exchange in embedded systems

RECOMMENDATION

To enhance the smoothness and effectiveness of the experiments, several recommendations can be implemented. For the potentiometer and analog readings experiment, introducing a calibration process at the start will ensure accurate readings, while employing graphical libraries like Matplotlib in Python can facilitate real-time data visualization. Adding error handling in the Python script will manage potential communication issues, and developing a simple user interface using Tkinter or PyQt can improve user interaction. In the servo motor control experiment, implementing a smoothing algorithm for the potentiometer input will yield more stable movements, and incorporating a feedback mechanism—such as an LED indicator—will confirm successful command execution. Adjusting PWM signal frequencies can optimize servo responsiveness, and conducting limit tests will prevent mechanical damage. By following these guidelines, error occurrence is much more minimal, allowing for an easier, more productive experimental experience and leading to successful outcomes and valuable learning experiences.

ACKNOWLEDGEMENTS

A special thanks goes out to Dr. Wahju Sediono and Dr. Zulkifli Bin Zainal Abidin, my teaching assistant, and my peers for their invaluable help and support in finishing this report. Their advice, feedback, and experience have greatly influenced the level of quality and understanding of this work. Their time, patience, and commitment to supporting my academic success are greatly appreciated.

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**.

Signature: 
Name: IRDINA NABIHAH BINTI MOHD NAZRI
Matric Number: 2214772

Read ☑
Understand ☑
Agree ☑

Signature: 
Name: KHALISAH AMANI BINTI YAHYA AZMI
Matric Number: 2218184

Read ☑
Understand ☑
Agree ☑

Signature: 
Name: LUQMAN AZFAR BIN AZMI
Matric Number: 2219857

Read ☑
Understand ☑
Agree ☑

Signature: 

Name: MOHAMAD NASRI BIN MOHAMAD NAZRI

Matric Number: 2219879

Read ☑

Understand ☑

Agree ☑

Signature: 

Name: MUHAMAS NURHAKIMIE THAQIF BIN ABDULLAH

Matric Number: 2213217

Read ☑

Understand ☑

Agree ☑