MECHATRONICS SYSTEM INTEGRATION

SECTION 2

SEMESTER 1 2025/2026

LECTURER:

Dr. Wahju Sediono

Dr Zulkifli bin Zainal

LAB REPORT WEEK 3: Serial Communication Potentiometer and Servo Motor

GROUP 14

NAME	MATRIC NUMBER
Ahmad Aidil Adha bin Rosehaizad	2118165
Ahmad Hafizi Bin Ishak	2111775
FIRZANAH BINTI ABDUL AZEEZ	2211430

TABLE OF CONTENTS

Introduction

Serial communication plays a crucial role in mechatronic systems, enabling seamless data transfer between microcontrollers and computers. In this experiment, we will demonstrate the usage of serial communication to control a servo motor and also potentiometer with an LED. By sending the data from a python script to the arduino for the demonstration for servo motor and vice versa for potentiometer, these experiments aim to highlight the practical application of parallel, serial, and USB interfacing in mechatronic system integration.

Abstract

This experiment explores the integration and data exchange between a microcontroller and a computer through serial communication, using an Arduino, a servo motor, and a potentiometer. The setup involves controlling a servo motor's position by sending angle commands from a Python script to the Arduino, which interprets the data to adjust the servo in real-time. Additionally, the potentiometer's analog readings are transmitted from the Arduino to the Python script, enabling visualization and monitoring of hardware sensor data. This combined approach demonstrates the fundamental principles of serial communication in embedded systems, highlights real-time control and feedback between hardware and software, and encourages further enhancements such as manual angle adjustment and system control via keyboard input, illustrating the versatility of microcontroller-computer interactions in mechatronic applications.

Experiment 3A: Serial Communication (PotentioMeter)

Materials and Equipment

ITEM	AMOUNT
LED	1
ARDUINO UNO MEGA	1
BREADBOARD	1
220 OHM RESISTANCE	7
JUMPER WIRES	9
POTENTIOMETER	1

Experimental Setup

1. Connect one leg of the potentiometer to 5V on the Arduino
2. Connect the other leg of the potentiometer to GNG 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

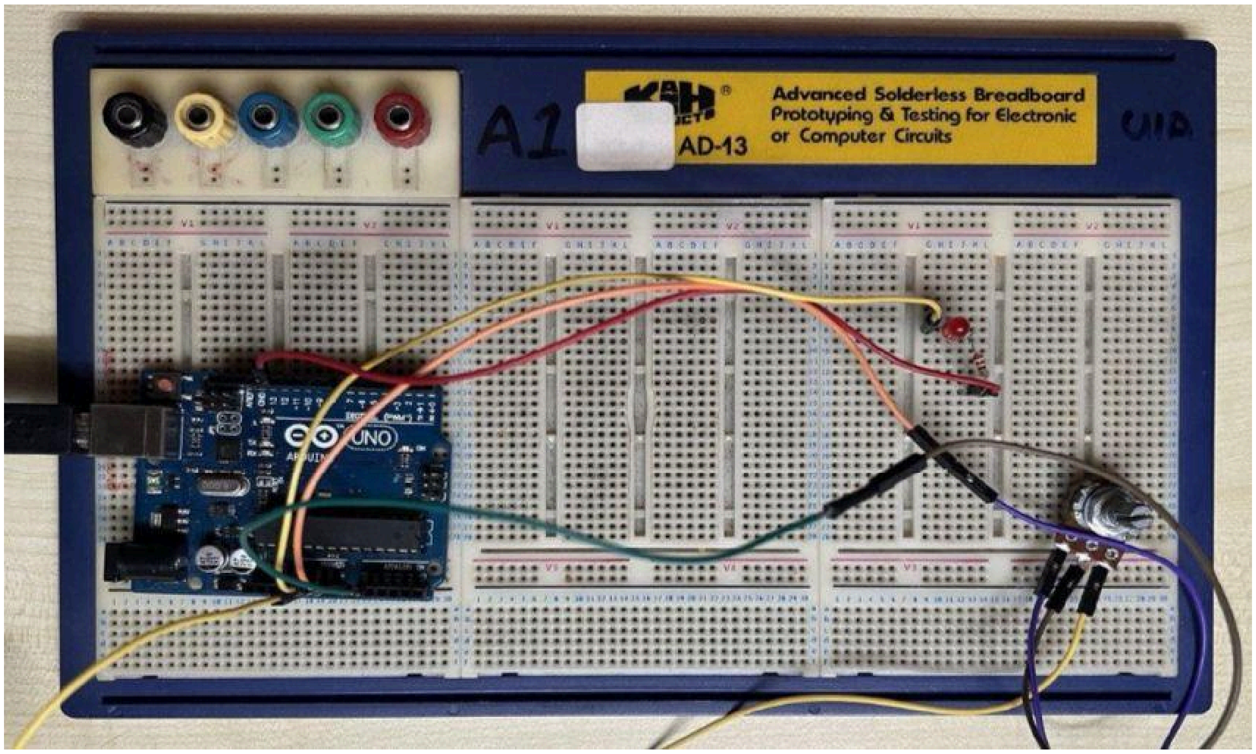
1. Setup for Arduino Board
    - Connect the potentiometer to the Arduino. Wire one end of the potentiometer to 5V, the other end to GND, and the centre pin to an analog input pin on the Arduino A0

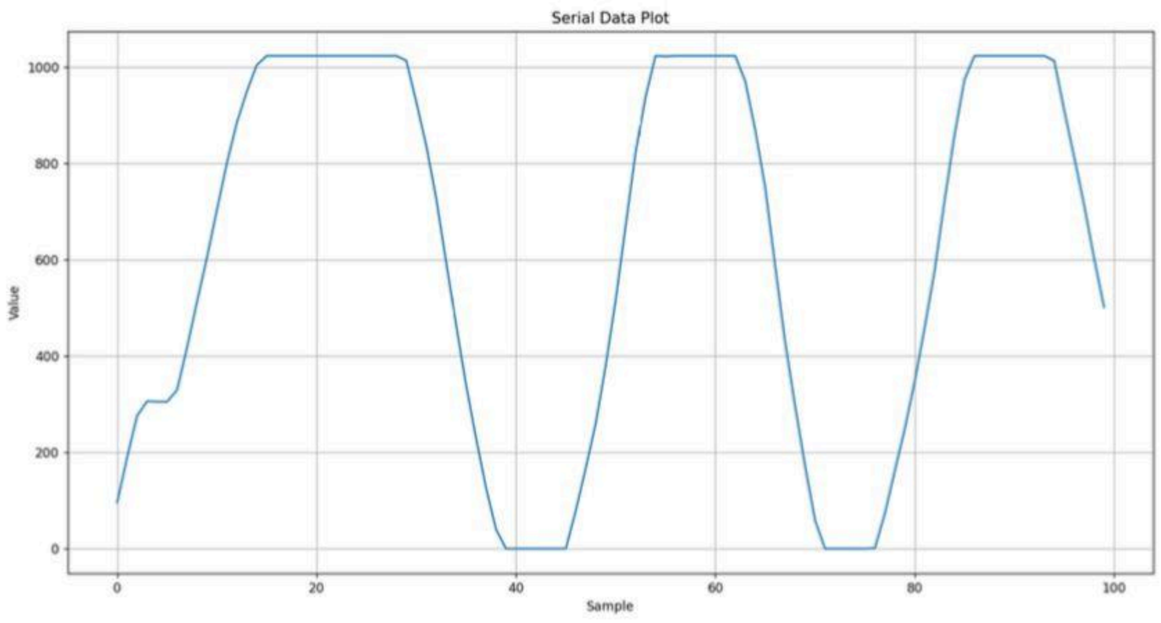
- Connect the LED to the Arduino, the negative part to GND, and the positive part to a resistor that is connected to pin 9
- Connect the Arduino to your computer via a USB cable

2. Software Programming Arduino IDE
   - Write a program in Arduino IDE to:
   - Define the pins that connected to the potentiometer and LED as potPin=A0 and ledPin = 9
   - Initialize serial communication at pin 9 to read the potentiometer value and send it over the serial port
   - Include a potentiometer function to control the LED using potentiometer
   - Upload the Arduino sketch to Arduino Uno
3. Arduino Execution
   - Control LED brightness using the potentiometer
   - Open Serial Plotter in Arduino IDE to monitor the data received from Arduino board using 9600 baud rate.
   - Record the graph of the serial Plotter
4. Python Execution
   - Write a program using Python to read the potentiometer data from the Arduino via the serial port
   - Turn the potentiometer knob and record the potentiometer values
   - Compare the values of the potentiometer between Python and Arduino IDE

Result

By building a simple circuit with a potentiometer to control an LED's brightness, the experiment's findings were obtained. The brightness of the LED is calculated once the value is read by the potentiometer. The potentiometer value is divided by four to determine the brightness. The function that regulates the LED's brightness then receives this value as an argument. The value was displayed on the serial monitor in order to debug the potentiometer. All things considered, the experiment effectively demonstrates how to control an LED's brightness using a potentiometer. The following diagrams are the connection of the LED and potentiometer circuit, the Serial plotter from Arduino IDE, and the Output Serial from the Python (Visual Studio COde) respectively.


Advanced Solderless Breadboard
Prototyping & Testing for Electronic or Computer Circuits
AD-13
A1
UNO
ARDUINO



Videos shall be provided in the Github link.

The Arduino and Python coding as such;

<u>Arduino</u>

```
int sensorPin = A0; // the
potentiometer is connected to analog pin 0
int ledPin = 13; // the LED is
connected to digital pin 13
int sensorV alue; // an integer
variable to store the potentiometer reading
void setup() { // this function
runs once when the sketch starts up
pinMode (ledPin, OUTPUT);
Serial.begin(9600);
}
void loop() { // this loop
runs repeatedly after setup() finishes
sensorV alue = analogRead(sensorPin); //
read the sensor
Serial.println(sensorV alue); // output
reading to the serial line
8
if (sensorV alue < 500){
```

```
digitalWrite(ledPin , LOW );} the LED off
else {
the LED on
digitalWrite(ledPin , HIGH );} // turn
// keep
delay (1000); // Pause in
milliseconds before next reading
}
```

We can illustrate the potentiometer readings with a graph by updating the existing code using Matplotlib. The code is as below:

Updated Arduino code

```
import serial
import matplotlib.pyplot as plt
from time import sleep
# Set your serial port and baud rate
serial_port = "COM3"
baud_rate = 9600
dt = 0.05
# Open the serial port
ser = serial.Serial(serial_port, baud_rate)
if ser.is_open:
print(f"Serial port {ser.name} is open.")
data = []
N = 100 # number of data
for k in range(N):
b = ser.readline()
strn = b.decode()
str1 = strn.rstrip()
flt = float(str1)
data.append(flt)
sleep(0.05)
# Close the serial port
ser.close()
if not ser.is_open:
print(f"Serial port {ser.name} is closed.")
# Plot the reading
plt.plot(data)
plt.xlabel('Time(seconds)')
plt.ylabel('Potentionmeter Reading')
plt.title('Potentionmeter Reading vs Time')
plt.grid(True)
plt.show()
else:
print(f"Failed to open serial port {serial_port}.")
```

Python

```
import serial
ser = serial.Serial('COM3', 9600)
try:
while True:
pot_value =
ser.readline().decode().strip()
print("Potentionmeter Value:",
pot_value)
except KeyboardInterrupt:
ser.close()
print("Serial connection closed.")
```

Discussion

In this lab session, we looked into the fundamentals of serial communication between an Arduino microcontroller and a Python script running on a PC. The goal was to send potentiometer readings from the Arduino to Python for analysis and display. This exercise demonstrated how computers and microcontrollers may work together to create interactive systems, which is an essential part of mechatronics and embedded systems design.

A potentiometer was connected to the analog input pin of the Arduino as part of the hardware configuration. Based on the position of the knob, the potentiometer functioned as a straightforward sensor, delivering varying voltage measurements. These values were read by the Arduino code, which then transmitted them at a predetermined baud rate across the serial connection. In order for the Arduino and Python script to transmit data accurately, the baud rate had to be set correctly.

To connect to the Arduino, the Python end used the pyserial library. After the script read and decoded the incoming data, the potentiometer values appeared in the terminal. This illustrated the capabilities of Python's hardware interface, especially when combined with modules that simplify communication protocols. Real-time data processing enables applications like automation, control systems, and data visualization.

One of the key lessons learned was the importance of managing serial communication effectively.

For instance, the Arduino Serial Plotter and Python script cannot both utilize the serial port simultaneously, emphasizing the need for careful resource management to avoid conflicts. However, by visualizing data in real time, the Serial Plotter aided in debugging and understanding sensor behavior.

All things considered, this experiment gave a good overview of how to combine software and hardware in mechatronic systems. It illustrated the usefulness of serial communication and showed how interactive systems might be made from basic parts like a potentiometer. The abilities acquired can be applied to increasingly challenging tasks, such managing motors or creating user interfaces, which makes Python plus Arduino a flexible platform for development and prototyping.

Recommendation

To enhance the performance and reliability of the potentiometer-controlled LED system, several improvements can be made across hardware, software, and user experience.

1. Hardware Improvements

-Component Replacement and Secure Connections: Ensure all components, especially jumper wires, are in good condition and properly connected to avoid performance issues.

-Verify Resistor Values: Use a 220-ohm resistor for current limiting to protect the LED and ensure consistent operation.

2. Software Improvements

-Optimize Code with Functions: Refactor the code to use functions for repetitive tasks like reading

potentiometer values and controlling the LED, making the code cleaner and easier to maintain.

-Use Serial Monitor for Debugging: Utilize the Serial Monitor for real-time debugging and monitoring of potentiometer data.

EXP 3B: Controlling Servo Motor via Serial Communication

MATERIALS AND EQUIPMENT

The following components are used for the experiment:

- Arduino board +USB cable
- Servo motor
- Jumper wires
- Potentiometer

EXPERIMENTAL SETUP

The following components and connections were used to carry out the experiment:

- The components used were an Arduino Uno, servo motor, potentiometer, jumper wires, and a USB cable.
- The servo motor's signal wire was connected to digital pin 9, Vcc to 5V, and GND to the Arduino ground.
- The potentiometer was connected with one terminal to 5V, the other to GND, and the middle terminal to A0 for analog input.
- The Arduino was linked to the computer via USB, allowing serial communication with the Python program.
- The Servo library was added in Arduino IDE (*Sketch → Include Library → Servo*).

METHODOLOGY

- The Arduino IDE programmed a sketch that read the angle data from the serial port and rotated the servo motor according to the input value.
- The Arduino IDE programmed a sketch that read the angle data from the serial port and rotated the servo motor according to the input value.

- A Python script was run to send servo angles ranging from 0° to 180° to the Arduino through serial communication.
- The user input an angle value into the Python command line, and the servo motor rotated to the respective position.
- In the long part of the experiment, the potentiometer worked in real time to adjust the servo angle, making smoother and continuous adjustments.
- An LED was also included to turn on or off based on the servo position or the potentiometer reading, as a visual indication.

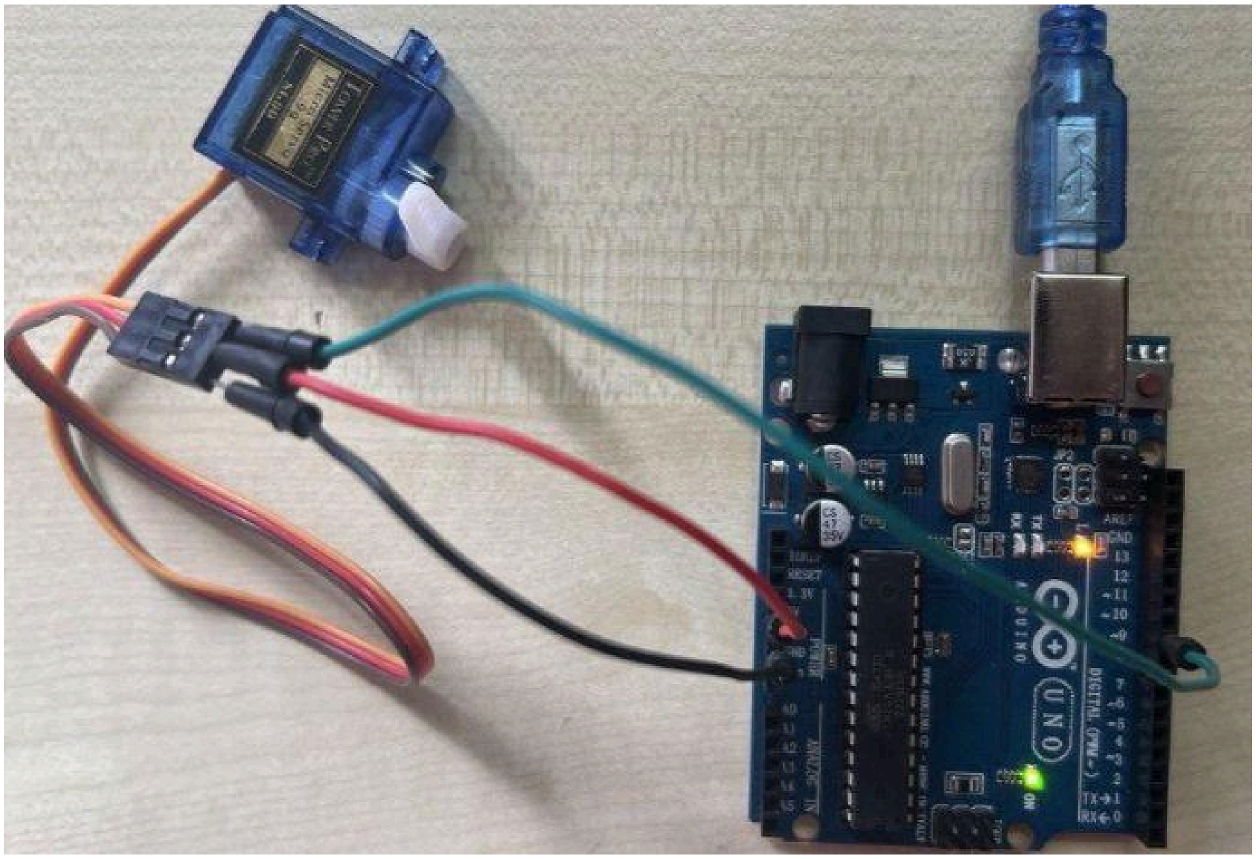
  The matplotlib library was used to create a real-time plot of potentiometer readings, to ensure live data transfer and reaction of the motor.
- The experiment stopped when the user entered 'q', which shut down the serial connection between the Arduino and the computer securely.

- A Python script was run to send servo angles ranging from 0° to 180° to the Arduino through serial communication.
- The user input an angle value into the Python command line, and the servo motor rotated to the respective position.

- In the long part of the experiment, the potentiometer worked in real time to adjust the servo angle, making smoother and continuous adjustments.
- An LED was also included to turn on or off based on the servo position or the potentiometer reading, as a visual indication.
- The matplotlib library was used to create a real-time plot of potentiometer readings, to ensure live data transfer and reaction of the motor.
- The experiment stopped when the user entered 'q', which shut down the serial connection between the Arduino and the computer seccure.

RESULT

The servo motor responded as expected to the angle values from the Python program through serial communication in the experiment. The system behaved as expected, and the results are :



Servo Movement Accuracy:

When an angle between 0° and 180° was entered in the Python terminal, the servo motor shifted to the corresponding position with satisfactory accuracy. The movement was continuous and stable, with hardly any delay between user input and the action of the servo.

Serial Communication Performance:

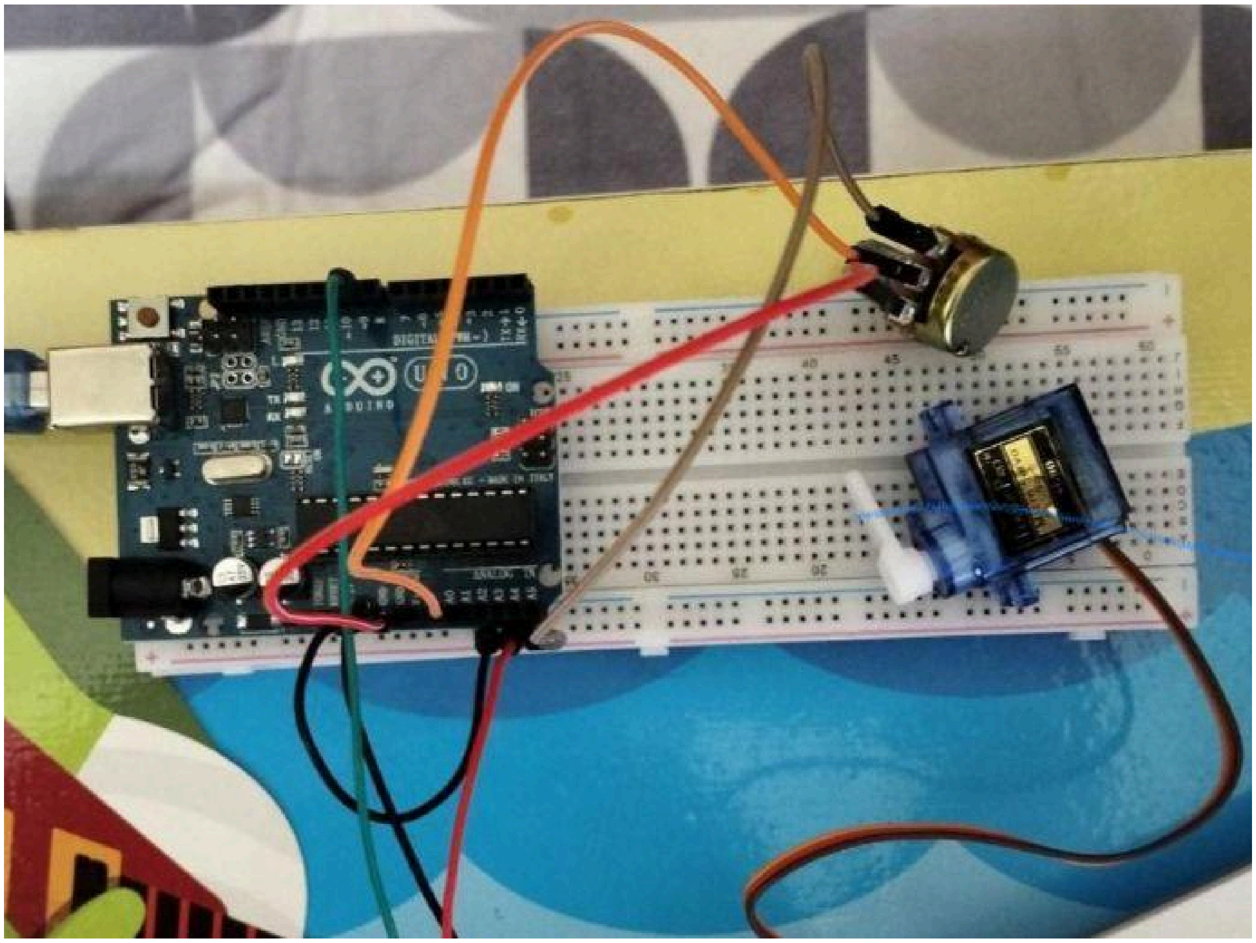
The serial communication between the Arduino and computer was reliable during the experiment. The serial port received valid numerical inputs successfully. Invalid or out-of-range values were ignored, with faultless operation.

Potentiometer Functionality:

As the potentiometer was added, the servo motor adjusted its position in real time based on the potentiometer's rotation. The serial monitor showed the alteration of the angle values, which reflected the successful transmission of data between hardware and software.

Program Termination:

Typing the command 'q' at the Python shell ended the program and closed the serial connection in a safe manner. This permitted no further data communication and graceful shutdown of the communication channel.



QUESTION

PYTHON CODE :

```
import serial

import time

ser = serial.Serial('COM8', 9600)

try:

while True:
```

```
angle = input("Enter servo angle

(0-180 degrees) or 's' to stop: ")

if angle.lower() == 's':

ser.write(b's') # Send 's' to stop

the servo

break

angle = int(angle)

if 0 <= angle <= 180:

ser.write(str(angle).encode()) #

Send the servo's angle to the Arduino

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.")
```

UPDATED ARDUINO CODE:

```
#include <Servo.h>
Servo myservo; // Create a servo object
int potPin = A0; // Analog pin for
potentiometer
int val; // Potentiometer value
int servoPos = 90; // Initial servo
position
boolean haltFlag = false; // Flag to
indicate if servo should be halted
void setup() {
myservo.attach(9); // Attach the servo
to pin 9
Serial.begin(9600); // Initialize serial
communication
}
void loop() {
if (!haltFlag) { // Check if the servo
should not be halted
```

```
val = analogRead(potPin);
// Read potentiometer value
servoPos = map(val, 0, 1023, 0, 180);
// Map potentiometer value to servo
angle
myservo.write(servoPos); // Set the
servo position
}
if (Serial.available() > 0) {
char key = Serial.read(); // Read a
character from serial input
if (key == 's') { // Check if 's' character
is received
haltFlag = !haltFlag; // Toggle the
haltFlag
if (haltFlag) {
myservo.write(90); // Halt the servo
at a specific position (90 degrees)
Serial.println("Servo halted.");
} else {
Serial.println("Servo resumed.");
}
}
}
}
```

DISCUSSION

The task clearly demonstrated how a servo motor can be controlled through serial communication between an Arduino Uno and Python. The servo responded smoothly and accurately to the entered angle values, with very little delay. The serial connection remained stable throughout the experiment, and invalid inputs were handled efficiently without affecting performance.

In the extended setup, the LED provided a simple visual cue for the motor’s position, while the potentiometer allowed real-time adjustment of the servo angle. Using Python’s matplotlib, the potentiometer data was plotted live, confirming proper data transfer between the computer and Arduino. Overall, the experiment successfully met its objectives, showing that hardware devices can be controlled easily and reliably through software-based serial communication.

RECOMMENDATION

The experiment worked well in showing how serial communication can be used to control a servo motor. However, a few changes could make the setup more practical and easier to use. Adding a potentiometer for real-time control and enabling two-way communication would make the system more responsive and accurate. Creating a simple GUI with buttons or sliders could also replace the command-line input and make the program more user-friendly.

It would be helpful to include safety features like angle limits or an emergency stop to protect the servo motor. Bluetooth or Wi-Fi could provide the ability for the system to be wireless and remote-controlled. The setup could also be expanded to be utilized for controlling multiple servos in synchrony for application in robotic arms. Adding the possibility for data logging and simple machine learning could help make the system more accurate and able to learn over time. After these are added, the experiment itself can become more reliable and valuable towards potential future uses in mechatronics or automation.

Conclusion

This experiment effectively demonstrated the practical use of serial communication to enable seamless interaction between a computer and a microcontroller for both sensor data acquisition and real-time hardware control. The successful transmission of potentiometer readings to a Python script illustrated efficient data monitoring, processing, and visualization, establishing a solid foundation for advanced applications in automation and embedded systems. Simultaneously, the use of Python to send angle commands to control a servo motor showcased the smooth integration of software and hardware in mechatronic systems, highlighting the importance of serial communication in precise, real-time control.

The experiment also emphasized the value of structured coding and graphical data representation for better understanding sensor behaviors and system responses. Incorporating manual control via a potentiometer and keyboard inputs further demonstrated the flexibility and interactivity of such systems, making them adaptable for practical scenarios like robotic actuation and industrial automation. Overall, these findings underscore the potential of serial communication to enhance human-machine interaction and advance the development of sophisticated mechatronic and embedded control systems.

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: Firzanah binti abdul azeez

Matric Number: 2211430

Signature: [illegible]

Read [✓]
Understand [✓]
Agree [✓]

Name: Ahmad Aidil Adha bin Rosehaizad

Matric Number: 2118165

Signature: [illegible]

Read [✓]
Understand [✓]
Agree [✓]

Name: Ahmad Hafizi Bin Ishak

Matric Number: 2111775

Read [✓]
Understand [✓]
Agree [✓]

Signature: Hafizi