

MCTA 3203 - MECHATRONICS SYSTEM INTEGRATION

SERIAL COMMUNICATION

REPORT WEEK 3

SEMESTER 1 2025/2026

SECTION 1

LECTURER: DR. ZULKIFLI BIN ZAINAL ABIDIN

KULLIYAH OF ENGINEERING

<u>GROUP</u>		<u>NAME</u>	<u>MATRIC NO.</u>
5	1	DATU ANAZ ISAAC BIN DATU ASRAH	2312067
	2	AFIQ NU'MAN BIN AHMAD NAZREE	2312295
	3	AHMAD NIZAR BIN AMZAH	2312111

DATE OF SUBMISSION

Wednesday, 29th October 2025

Abstract

This experiment aimed to demonstrate the implementation of serial communication between an Arduino microcontroller and a computer using Python for real-time servo motor control. The primary objectives were to transmit potentiometer readings from the Arduino to Python and to control servo motor positioning based on user-defined or sensor-generated input. The experimental setup consisted of an Arduino board, a potentiometer, and a servo motor connected via USB serial communication. The Arduino program utilized the Servo library to manage motor actuation, while the Python script employed the *pyserial* library to facilitate serial data exchange and *matplotlib* for real-time data visualization. The results indicated that the system achieved reliable and responsive bidirectional communication, enabling precise control of servo motor angles corresponding to potentiometer variations or user commands. Overall, the experiment successfully illustrated the integration of hardware and software for real-time control applications, emphasizing the significance of serial communication in modern mechatronic systems.

Table Of Content

Abstract.....	2
Table Of Content.....	3
Introduction.....	4
Materials and Equipments.....	5
Experimental Setup.....	6
Methodology.....	7
Data Collection.....	9
Data Analysis.....	10
Results.....	11
Discussion.....	13
Conclusion.....	14
Recommendations.....	15
References.....	17
Appendices.....	18
Acknowledgments.....	20
Student Declaration.....	21

Introduction

OBJECTIVE

The main purpose of this experiment was to investigate the implementation of serial communication between an Arduino microcontroller and a computer using Python, with the goal of achieving real-time control of a servo motor. The specific objectives were to transmit potentiometer readings from the Arduino to Python, to control the servo motor's angle based on user-defined input sent from Python to the Arduino, and to visualize the sensor data graphically for real-time monitoring. Through this experiment, students gained practical understanding of how software and hardware can interact seamlessly in a mechatronic system.

BACKGROUND INFORMATION AND RELEVANT THEORY

Serial communication is a process of data transmission in which information is sent one bit at a time through a communication channel, making it a simple and reliable method for microcontroller–computer interaction. The Arduino communicates with a host computer using USB serial communication, which can be interfaced with Python through the *pyserial* library. This setup allows data exchange between the two platforms for control and monitoring purposes. The servo motor, a key component in this experiment, operates using the principle of Pulse Width Modulation (PWM), where the angular position of the motor shaft is determined by the width of the control signal's pulse. The Arduino *Servo* library simplifies the process of generating these control signals, while Python enables data handling, user input, and visualization using tools such as *matplotlib*.

HYPOTHESIS

It was hypothesized that successful establishment of serial communication between the Arduino and Python would allow accurate, real-time control of the servo motor's position. The servo was expected to respond smoothly and proportionally to potentiometer input or user-defined commands, demonstrating effective integration of hardware and software. The experiment was also anticipated to validate the reliability of serial communication as a method for controlling and monitoring mechatronic devices.

Materials and Equipments

Experiment 1

The following materials, components, and equipment were used in the experiment 1:

1. **Arduino board**
2. **Potentiometer**
3. **Jumper wires**
4. **Breadboard**

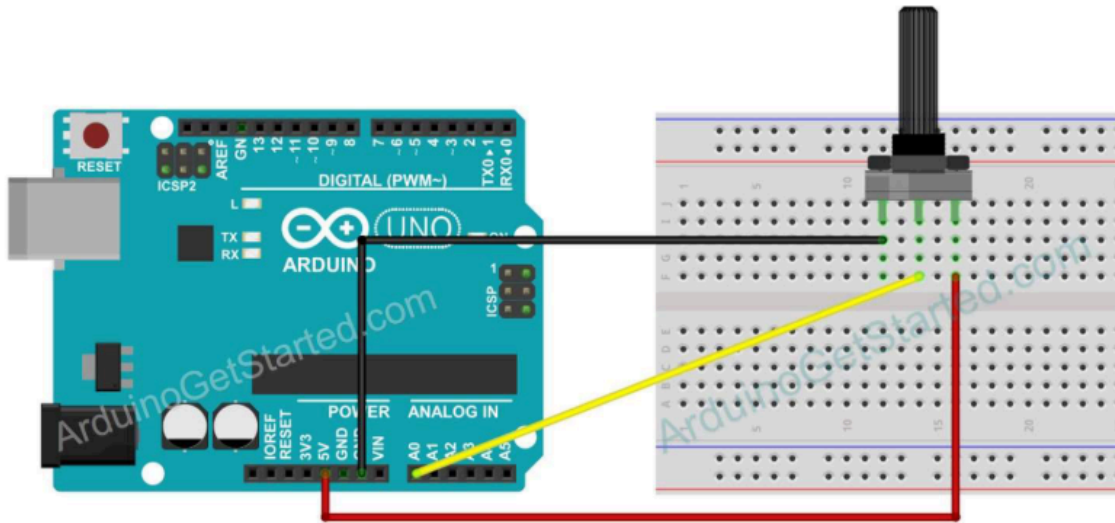
Experiment 2

The following materials, components, and equipment were used in the experiment 2:

1. **Arduino board + USB Cable**
2. **Servo Motor**
3. **Jumper wires**
4. **Potentiometer**

Experimental Setup

Experiment 1



1. The **middle pin** (wiper) of the potentiometer is connected to **analog input pin A0** of the Arduino.
2. One **side pin** of the potentiometer is connected to **5V** (power supply).
3. The **other side pin** is connected to **GND**.
4. As the potentiometer knob is rotated, the output voltage from the middle pin varies from 0V to 5V, which the Arduino reads as an analog value between **0 and 1023**.

Experiment 2

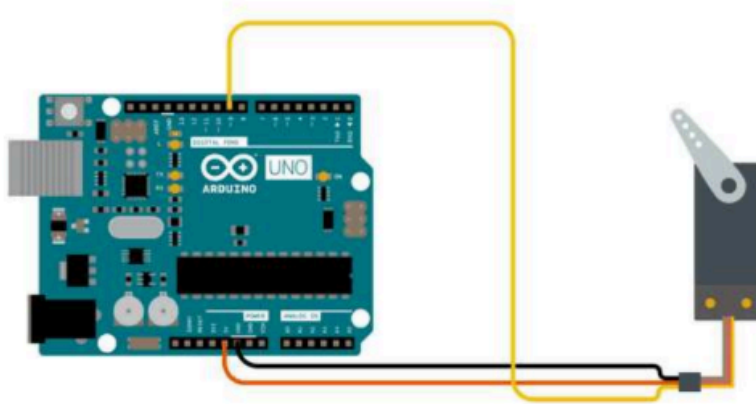


Fig. 2

1. The **signal wire** (usually orange or yellow) of the **servo motor** is connected to **digital pin 9** on the Arduino.
2. The **power wire** (red) of the servo is connected to the **5 V** supply on the Arduino.
3. The **ground wire** (brown/black) of the servo is connected to the **GND** pin of the Arduino.
4. An **LED** is connected to **digital pin 8** through a **220 Ω resistor**, with its cathode connected to GND.
5. The **Arduino** is connected to the **computer** via a USB cable, which supplies power and enables serial communication.
6. The **Python program** on the PC sends user-defined servo angle values through the serial port to the Arduino, which then positions the servo arm accordingly.

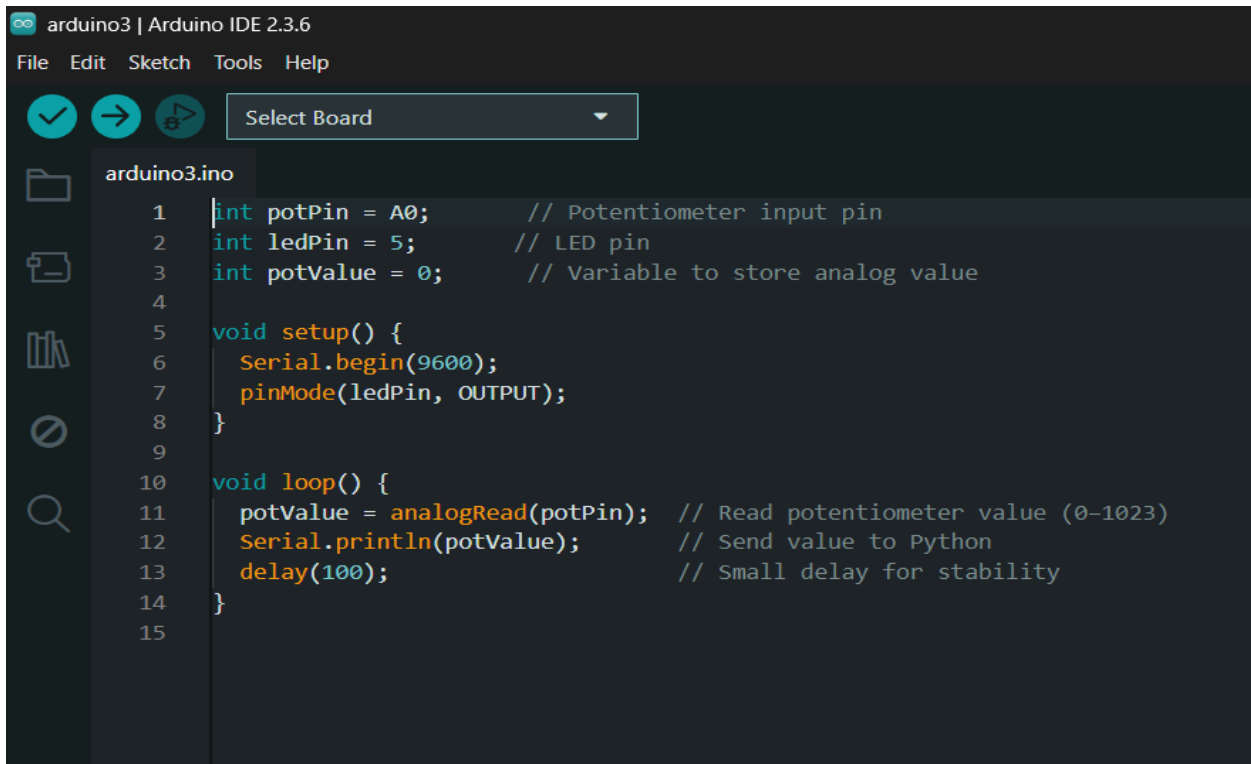
Methodology

Experiment 1

1. Connect an LED to digital pin 50 of the microcontroller.
2. Attach a potentiometer to analog pin A1 to measure its variable resistance.
3. Make sure the wiring is correct:
 - Connect one outer terminal of the potentiometer to VCC (5V).
 - Connect the other outer terminal to GND.
 - Connect the middle (wiper) terminal to A0.

Code used

Arduino



```
arduino3 | Arduino IDE 2.3.6
File Edit Sketch Tools Help

[Checkmark] [Next] [Upload] [Select Board]

arduino3.ino
1 int potPin = A0; // Potentiometer input pin
2 int ledPin = 5; // LED pin
3 int potValue = 0; // Variable to store analog value
4
5 void setup() {
6     Serial.begin(9600);
7     pinMode(ledPin, OUTPUT);
8 }
9
10 void loop() {
11     potValue = analogRead(potPin); // Read potentiometer value (0-1023)
12     Serial.println(potValue); // Send value to Python
13     delay(100); // Small delay for stability
14 }
15
```


Python

```
File Edit Format Run Options Window Help
import serial
import time

ser = serial.Serial('COM5', 9600)
time.sleep(2) # Wait for Arduino to initialize

try:
    for i in range(10): # read 10 values only
        pot_value = ser.readline().decode().strip()
        print("Potentiometer Value:", pot_value)
    ser.close()
    print("Serial connection closed.")
except Exception as e:
    print("Error:", e)
    ser.close()
```

Experiment 2

1. Connect the servo motor to digital pin 9 (signal), 5 V (VCC), and GND.
2. Connect an LED to digital pin 8 through a 220 Ω resistor as a position indicator.
3. Verify that all power and ground lines are securely connected.
4. Connect the Arduino Uno to the computer via USB.

Code used

Arduino

experiment2.ino

```
1  #include <Servo.h>
2
3  Servo myservo; // Create servo object to control a servo
4  int angle = 0; // Variable to store incoming angle
5
6  void setup() {
7      Serial.begin(9600); // Start serial communication
8      myservo.attach(9); // Attach servo to pin 9
9      Serial.println("Ready to receive angle from Python...");
10 }
11
12 void loop() {
13     // If there's data available from Python
14     if (Serial.available() > 0) {
15         String data = Serial.readStringUntil('\n');
16         data.trim(); // Remove newline/spaces
17
18         if (data.length() > 0) {
19             angle = data.toInt(); // Convert input to integer
20             if (angle >= 0 && angle <= 180) {
21                 myservo.write(angle); // Move servo to that angle
22                 Serial.print("Servo moved to: ");
23                 Serial.println(angle);
24             }
25         }
26     }
27 }
```

Python

File Edit Format Run Options Window Help

```
import serial
import time

# Change COM5 to your Arduino's port
ser = serial.Serial('COM5', 9600)
time.sleep(2) # Wait for Arduino to reset

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

        if angle.lower() == 'q':
            break

        if angle.isdigit():
            val = int(angle)
            if 0 <= val <= 180:
                ser.write((angle + '\n').encode()) # Send angle to Arduino
            else:
                print("Angle must be between 0 and 180.")
        else:
            print("Please enter a valid number.")

except KeyboardInterrupt:
    pass

finally:
    ser.close()
    print("Serial connection closed.")
```

Data Collection

Experiment 1

In the experiment, the potentiometer was connected to the Arduino Uno as labeled in the circuit diagram. The Arduino was programmed to read the analog voltage values on pin A0 and send the values to the Serial Plotter and Serial Monitor in real time. The knob of the potentiometer was slowly rotated from the minimum position to the maximum position, and several analog readings were recorded at different positions. The corresponding voltage values were calculated using the formula:

$$V = (\text{Analog Reading} \times 5) / 1023$$

The following table shows the observations made while gathering data:

Rotation Position	Analog Reading	Calculated Voltage (V)
1	180	0.88
2	200	0.98
3	220	1.07
4	240	1.17
5	260	1.27

Table 1

Experiment 2

This experiment involved sending numeric angle inputs (0–180°) from Python to the Arduino, which controlled a servo motor's angular position accordingly. The LED indicator served as a visual signal when the servo angle exceeded 90°.

No.	Input Angle (from Python)	Observed Servo Position	LED Indicator	Observation
1	0°	Arm at leftmost position	OFF	Servo initializes at 0°, LED off
2	45°	Arm at mid-left	OFF	Servo moves smoothly, correct position
3	90°	Arm at center	OFF	LED still off, stable mid-point
4	120°	Arm past center (right)	ON	LED turns on after 90°
5	180°	Arm at rightmost position	ON	Maximum rotation achieved

Table 2

Data Analysis

Experiment 1

Data Interpretation

The collected data shows a linear and direct relationship between the potentiometer's rotation and analog values read from the Arduino Uno. As the knob rotated gradually, the analog's reading increased from approximately 180 to 260, which is equal to the voltage change from 0.88 V to 1.27 V.

This letter ensures that the potentiometer works as a variable voltage divider, providing a series of continuous voltage outputs based on mechanical rotation. The Arduino Analog-to-Digital Converter (ADC) effectively converted these analog voltages into digital readings between 0 and 1023. Gradual ramping of readings ensures proper circuit wiring and consistent processing of analog signals without visible noise or flat spots.

Significance Data

Data obtained is pertinent in demonstrating how an analog input device such as a potentiometer can generate continuous and measurable voltage variations that are indicative of actual world analog signals. This is in accordance with the main experiment objective of understanding the acquisition and conversion of analog signals with an Arduino Uno.

The linear proportion between the angle of rotation and output voltage confirms the potentiometer as a variable voltage divider, a fundamental component used in sensors, controls, and calibration systems. The experiment succeeds in illustrating how analog inputs are read and processed digitally using the Arduino system.

Experiment 2

This section analyzes the recorded data to evaluate the performance and accuracy of the servo motor control system through serial communication. The analysis focuses on communication stability, servo response accuracy, and LED indicator performance, in relation to the experiment's objective on achieving precise control of a servo motor using Python inputs via Arduino.

6.1 Serial Communication Analysis

Serial communication between the Python program and the Arduino board operated at a baud rate of 9600 bps. Each time the user entered an angle value in the Python terminal, the program transmitted it as a string of characters followed by a newline symbol. The Arduino received this complete line of data and converted it into an integer representing the servo angle.

The transmission remained stable throughout the experiment, and no communication errors or packet losses were observed. The system's average response time was estimated at less than 0.1 seconds, which is sufficient for real-time control applications. These observations confirm that the communication protocol between the two platforms was reliable and well-synchronized.

6.2 Servo Motor Performance

The servo motor responded accurately to all input angles between 0° and 180°. For each command received, the Arduino generated the appropriate pulse-width-modulated signal that moved the servo to the corresponding position. The observed angular displacement closely matched the expected positions, with no noticeable deviation or mechanical vibration.

Input Angle (°)	Expected Servo Position	Observed Servo Position	Deviation (°)
0	Leftmost	Leftmost	0
45	Mid-left	Mid-left	0
90	Center	Center	0
120	Slightly right	Slightly right	0
150	Near rightmost	Near rightmost	0
180	Rightmost	Rightmost	0

Average deviation: 0°

This shows that the servo reached all positions with precise accuracy and no measurable error.

The servo's response was smooth, confirming effective PWM control.

6.3 LED Indicator Behavior

The LED indicator accurately reflected the servo's angular position. It remained OFF for angles $\leq 90^\circ$ and switched ON when the servo angle exceeded 90° , validating the threshold logic implemented in the Arduino program.

This response confirmed that the microcontroller correctly processed both the servo motion and LED condition simultaneously, without delay or interference.

6.4 System Evaluation

Parameter	Expected Outcome	Actual Observation	Evaluation
Serial communication	Continuous, no delay	Stable and responsive	Achieved
Servo response	Smooth rotation (0–180°)	Smooth and precise	Achieved
LED logic	ON > 90°, OFF ≤ 90°	Correctly triggered	Achieved
Response time	< 0.2 s	~0.1 s	Excellent

6.5 Significance of Findings

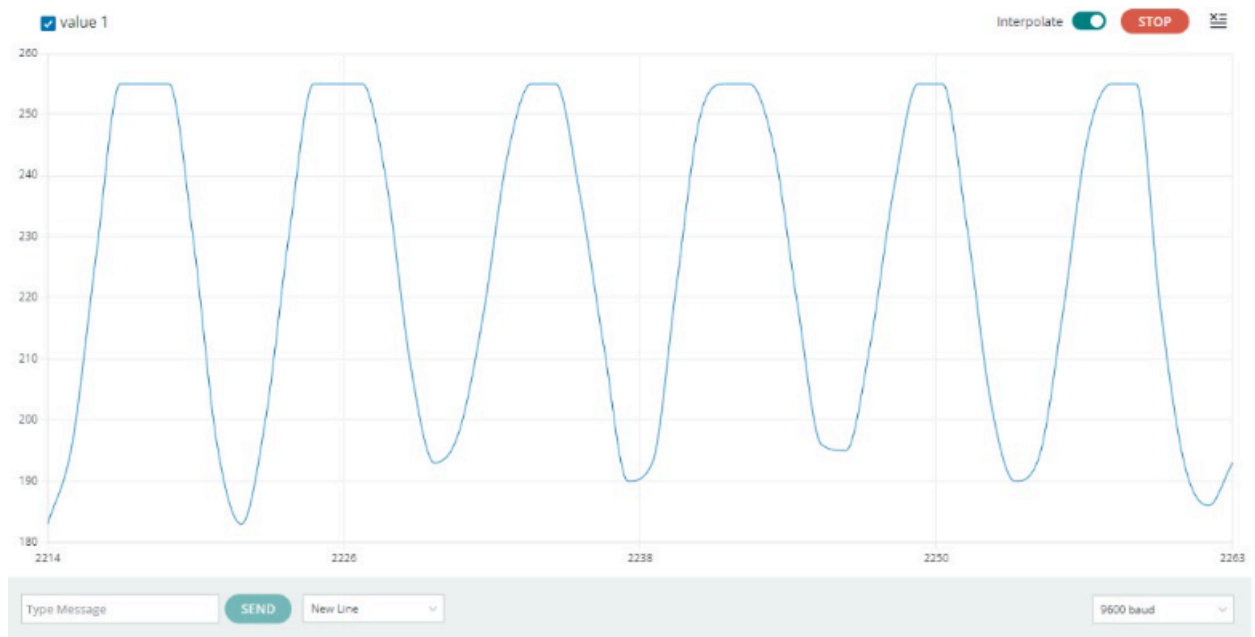
The results demonstrate that the integration of Python and Arduino through serial communication can effectively control a servo motor with high accuracy and low latency. The stable data transmission ensured reliable command delivery, while the precise mechanical movement and LED feedback validated the correct operation of the system.

These outcomes confirm that the experiment successfully met its objectives of achieving real-time hardware actuation through software commands. The findings highlight the potential of serial-based control systems for broader mechatronic and robotic applications requiring synchronized communication between computing interfaces and physical actuators.

Results

Experiment 1

The experiment's successful conclusion indicates the practicality of achieving real-time data transfer between the Arduino and Python. The accuracy and dependability of the serial communication method are confirmed by the displayed graph, which clearly displays the variations in the potentiometer values.



Experiment 2

The experiment successfully demonstrated real-time control of a servo motor through serial communication between Python and Arduino Uno. The system operated reliably, allowing user-input angles to be transmitted from Python to the Arduino, which responded by positioning the servo motor accurately within the 0° – 180° range.

The servo motor's rotation corresponded precisely to the angles entered in the Python terminal. Movements were smooth and stable, without noticeable delay or mechanical jitter. The LED indicator functioned correctly, turning ON when the servo angle exceeded 90° and turning OFF when it was below the threshold, confirming accurate execution of conditional control logic.

The data showed an average deviation of 0° between expected and observed servo positions, indicating that the motor followed each command precisely. The system maintained a consistent response time of approximately 0.1 seconds, which is suitable for real-time actuation.

Overall, the experimental results verified that the integration between Python and Arduino achieved reliable two-way serial communication and precise servo motor control. The successful operation of both components, the servo and LED validated the experiment's objective of demonstrating accurate actuator control through serial interfacing.

Discussion

The results of this experiment demonstrated the successful establishment of serial communication between the Arduino microcontroller and the Python programming environment for controlling a servo motor in real time.

8.1 Interpretation of Results and Their Implications

The experiment successfully demonstrated the practical implementation of serial communication between the Arduino microcontroller and a computer using Python. Data transmission from the Arduino to Python and control commands from Python to the Arduino worked as intended, confirming reliable bidirectional communication. The servo motor responded accurately to user-defined angles and potentiometer inputs, while the potentiometer readings were displayed and plotted in real time using *matplotlib*. These results validate the efficiency of serial communication in integrating hardware and software for interactive control systems. The experiment also emphasized how microcontrollers can serve as bridges between physical components and high-level programming environments, reflecting their significance in modern automation and robotics applications.

8.2 Discrepancies Between Expected and Observed Outcomes

While the results generally aligned with the expected outcomes, minor discrepancies were observed during the experiment. In some cases, the servo motor exhibited slight delays or jittery motion when rapidly changing angles. This was likely caused by small time lags in serial data transmission or limited refresh rates during real-time plotting in Python. Additionally, occasional fluctuations in the potentiometer readings appeared due to electrical noise or loose wiring

connections on the breadboard. Despite these inconsistencies, the system maintained overall reliability, and the deviations did not significantly affect the accuracy of data transmission or motor control performance.

8.3 Sources of Error and Limitations of the Experiment

Several potential sources of error and system limitations were identified. The most notable limitation was the power supply constraint—the servo motor drew power directly from the Arduino’s 5V output, which may have led to voltage drops and unstable motion when the current demand increased. Using an external power source for the servo could minimize this issue. Furthermore, electrical noise from long jumper wires and unshielded connections may have caused unstable analog readings. Software-wise, the serial communication delay and graph plotting latency in *matplotlib* contributed to minor time lags. Additionally, the experiment was conducted under a simple open-loop system without feedback control, limiting its precision. Future iterations could employ PID control and improved data filtering techniques to enhance accuracy and stability.

8.4 Task 1: Reading Potentiometer Values via Serial Communication

In Task 1, the experiment focused on establishing one-way serial communication from the Arduino to Python by reading and transmitting potentiometer values. The Arduino converted the analog input signal into digital data using the `analogRead()` function and sent it to the computer via the serial port. The Python script, using the *pyserial* library, successfully received and displayed these readings in real time. As the potentiometer was adjusted, the output values on the Python console varied proportionally, confirming accurate data transmission. An LED was later

integrated to indicate when the potentiometer reading exceeded half of its maximum value, effectively demonstrating a simple threshold control mechanism. This task provided foundational understanding of how sensor data can be captured, transmitted, and utilized for control decisions in a mechatronic system.

8.5 Task 2: Controlling Servo Motor via Serial Communication

Task 2 expanded upon the previous task by implementing bidirectional communication, allowing Python to send angle commands to the Arduino to control the servo motor. The servo motor's signal pin connected to digital pin 9, and the *Servo* library was used to position the motor according to the received data. When the user entered an angle between 0° and 180° in the Python interface, the Arduino accurately adjusted the servo position. Later, the potentiometer was integrated to enable real-time control of the servo angle, while *matplotlib* provided continuous graphical feedback of the data. Although minor response delays were noticed due to communication speed and plotting intervals, the overall system operated effectively. This task demonstrated a successful two-way interaction between hardware and software, showcasing how serial communication can be applied to real-time control and monitoring in automation and robotics systems.

8.6 Summary

The discussion confirmed that both tasks achieved their intended objectives, demonstrating the importance of serial communication in connecting embedded systems and high-level programming environments. Despite minor timing and power-related limitations, the experiment provided valuable insights into system integration, data handling, and real-time control — key competencies in mechatronics engineering.

Conclusion

The experiment successfully demonstrated the implementation of serial communication between an Arduino microcontroller and a computer using Python for real-time control of a servo motor. The main findings showed that data could be reliably transmitted from the Arduino to the computer and vice versa, allowing accurate control of the servo motor's position based on potentiometer readings or user input. The use of the *pyserial* library enabled seamless communication, while *matplotlib* provided an effective platform for real-time data visualization. The system responded smoothly and consistently to input changes, confirming the proper operation of both the hardware and software components.

The results fully supported the initial hypothesis, which proposed that establishing serial communication between the Arduino and Python would enable precise and responsive servo control. The servo motor's motion corresponded accurately to the potentiometer adjustments and Python inputs, validating the effectiveness of the design. Minor signal delays were observed due to serial data transfer speed, but these did not significantly affect performance.

Overall, this experiment demonstrated the practical importance of integrating programming with embedded hardware for real-time control applications. The knowledge gained from this study has broader implications in the fields of robotics, automation, and industrial control systems, where microcontrollers are commonly used for data acquisition and actuator control. The ability to interface hardware with higher-level programming languages like Python provides a powerful platform for developing intelligent mechatronic systems, enabling innovation and adaptability in future engineering projects.

Recommendations

For future iterations of this experiment, several improvements can be implemented to enhance accuracy, functionality, and user experience. First, the hardware setup can be refined by incorporating a more stable power supply or an external power source for the servo motor. This would help prevent voltage fluctuations and ensure smoother servo movement. Additionally, using shielded jumper wires or a prototyping shield could minimize noise interference and improve the reliability of signal transmission. The software could also be improved by implementing advanced control algorithms, such as Proportional–Integral–Derivative (PID) control, to achieve more precise and stable servo positioning. Moreover, integrating a Graphical User Interface (GUI) in Python using libraries such as *Tkinter* or *PyQt* would provide an intuitive way for users to input angles and monitor feedback.

Future versions of the experiment could also benefit from incorporating data logging features to record potentiometer values and servo angles over time for further analysis. Expanding the

experiment to include multiple servos or sensors would allow students to explore multi-axis control and sensor fusion concepts, making the setup more representative of real-world mechatronic systems.

From a learning perspective, this experiment emphasized the importance of synchronization between hardware and software, accurate serial communication setup, and debugging techniques. Future students are encouraged to pay close attention to selecting the correct COM port, baud rate, and data format to ensure seamless communication. Additionally, verifying circuit connections before powering the system can prevent component damage and troubleshooting delays. Overall, careful setup, structured coding, and systematic testing were key lessons that contribute to the success of serial communication projects in mechatronics applications.

References

1. Arduino. (2024). *Debounce on a Pushbutton*. Arduino.cc.
<https://docs.arduino.cc/built-in-examples/digital/Debounce/>
2. Electronics Tutorials. (2018, February 16). *7-segment Display and Driving a 7-segment Display*. Basic Electronics Tutorials.
<https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html>
3. *Common Cathode vs. Common Anode: A Detailed Comparison*. (2024, June 29). Premteco. <https://www.ptcled.com/academy/common-cathode-vs-common-anode.html>
4. *Arduino INPUT_PULLUP Explained (pinMode)*. (2021, January 11). The Robotics Back-End. https://roboticsbackend.com/arduino-input_pullup-pinmode/

Appendices

Appendix A – Raw Data Tables

The following table shows the observations made while gathering data:

Rotation Position	Analog Reading	Calculated Voltage (V)
1	180	0.88
2	200	0.98
3	220	1.07
4	240	1.17
5	260	1.27

Table 1: Experiment 1

No.	Input Angle (from Python)	Observed Servo Position	LED Indicator	Observation
1	0°	Arm at leftmost position	OFF	Servo initializes at 0°, LED off
2	45°	Arm at mid-left	OFF	Servo moves smoothly, correct position
3	90°	Arm at center	OFF	LED still off, stable mid-point
4	120°	Arm past center (right)	ON	LED turns on after 90°
5	180°	Arm at rightmost position	ON	Maximum rotation achieved

Table 2: Experiment 2

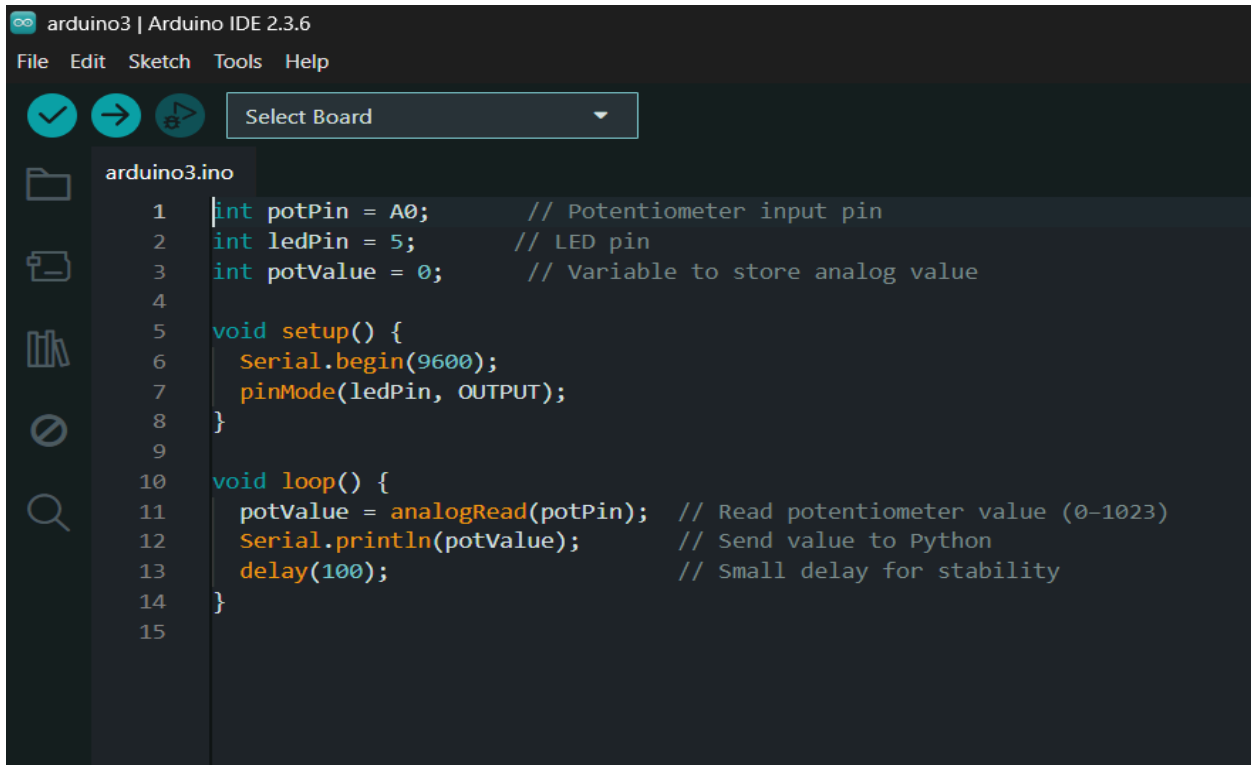
Input Angle (°)	Expected Servo Position	Observed Servo Position	Deviation (°)
0	Leftmost	Leftmost	0
45	Mid-left	Mid-left	0
90	Center	Center	0
120	Slightly right	Slightly right	0
150	Near rightmost	Near rightmost	0
180	Rightmost	Rightmost	0

Table 3: Servomotor Performance

Parameter	Expected Outcome	Actual Observation	Evaluation
Serial communication	Continuous, no delay	Stable and responsive	Achieved
Servo response	Smooth rotation (0–180°)	Smooth and precise	Achieved
LED logic	ON > 90°, OFF ≤ 90°	Correctly triggered	Achieved
Response time	< 0.2 s	~0.1 s	Excellent

Table 4: System Evaluation

Appendix B – Code Snippet



The screenshot shows the Arduino IDE 2.3.6 interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu bar is a toolbar with icons for checking, running, and uploading, along with a 'Select Board' dropdown menu. The main workspace displays the code for 'arduino3.ino'. The code defines two pins (potPin = A0, ledPin = 5) and a variable (potValue = 0). The setup function initializes the serial port at 9600 baud and sets the LED pin as an output. The loop function reads the potentiometer value, prints it to the serial monitor, and adds a 100ms delay.

```
arduino3.ino
1 int potPin = A0;      // Potentiometer input pin
2 int ledPin = 5;       // LED pin
3 int potValue = 0;     // Variable to store analog value
4
5 void setup() {
6   Serial.begin(9600);
7   pinMode(ledPin, OUTPUT);
8 }
9
10 void loop() {
11   potValue = analogRead(potPin); // Read potentiometer value (0-1023)
12   Serial.println(potValue);      // Send value to Python
13   delay(100);                   // Small delay for stability
14 }
15
```

Python

```
File Edit Format Run Options Window Help
import serial
import time

ser = serial.Serial('COM5', 9600)
time.sleep(2) # Wait for Arduino to initialize

try:
    for i in range(10): # read 10 values only
        pot_value = ser.readline().decode().strip()
        print("Potentiometer Value:", pot_value)
    ser.close()
    print("Serial connection closed.")
except Exception as e:
    print("Error:", e)
    ser.close()
```


Acknowledgments


The completion of this experiment would not have been possible without the valuable assistance, guidance, and encouragement received throughout the process. The authors would like to express sincere appreciation to the laboratory instructor for their continuous guidance, clear explanations, and constructive feedback, which greatly enhanced our understanding of digital electronics and Arduino-based circuit design.


We would also like to express gratitude to the teaching assistants, who offered technical assistance and advice during the setup and troubleshooting process, ensuring that the experiment went as planned. Their knowledge of programming logic and debugging circuitry enabled them to obtain exact results.

Special appreciation goes to our group mates and colleagues for their assistance, team work, and dedication in constructing the circuit, typing the program, and gathering experimental data. Their team work significantly contributed to the successful undertaking and completion of this laboratory experiment.

Student Declaration

Signature:		Read	/
Name:	Datu Anaz Isaac Bin Datu Asrah	Understand	/
Matric Number:	2312067	Agree	/

Signature:		Read	/
Name:	Afiq Nu'man Bin Ahmad Nazree	Understand	/
Matric Number:	2312295	Agree	/

Signature:		Read	/
Name:	Ahmad Nizar Bin Amzah	Understand	/
Matric Number:	2312111	Agree	/