MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1 2025/2026

WEEK 3: SERIAL COMMUNICATION ON BETWEEN ARDUINO AND PYTHON

SECTION 2

GROUP 16

LECTURER: ZULKIFLI BIN ZAINAL ABIDIN & WAHJU SEDIONO

Date of Experiment: Monday, 20 October 2025

Date of Submission: Monday, 27 October 2025

NO.	GROUP MEMBERS	MATRIC NO
1.	MUHAMMAD HAIKAL ZULHILMI BIN FAIROF	2313025
2.	MUHAMMAD FAIZ IZWAN BIN NOR EFFENDI	2313509
3.	AINUL JAARIAH BINTI ALIUDIN	2312664

ABSTRACT

The objective of task 1 was to establish and analyze serial communication between an Arduino microcontroller and a Python program for real-time data acquisition and control. A potentiometer was connected to the Arduino to provide a variable analog input, which was read and transmitted via the serial port. The Arduino code controlled an LED, turning it ON when the potentiometer reading exceeded half of its maximum range, and OFF otherwise. The potentiometer values and LED status were displayed in the Serial Monitor, and Python was used to read and process the same data externally. The experiment demonstrated successful bidirectional communication between hardware and software, allowing both visualization and control through serial data exchange. The results confirmed that the potentiometer readings varied smoothly with knob rotation, and the LED responded accurately to threshold changes. Overall, the experiment achieved its objective of integrating Arduino and Python for interactive real-time monitoring and control applications.

Task 2 demonstrates a real-time control and data visualization system that integrates an Arduino microcontroller, a servo motor, a potentiometer, and an LED using Python. The system allows for dynamic adjustment of the servo motor's angle using the potentiometer, provides immediate visual feedback via an LED whose brightness or state is tied to the potentiometer's value, and transmits the potentiometer data serially to a computer. A Python script receives this data and utilizes Matplotlib to generate a real-time, interactive plot of the potentiometer readings. Furthermore, the system incorporates a keyboard interrupt feature in both the Arduino and Python code, allowing for safe and controlled termination of execution, thereby ensuring reliable operation and data handling. This integration provides a robust platform for real-time electromechanical control and data monitoring.

TABLES OF CONTENTS

TASK 1

1.1 INTRODUCTION

This report describes the implementation of a fundamental mechatronics control system utilizing serial communication between an Arduino microcontroller and a Python host application. The objective of this task was to establish a closed-loop feedback mechanism where a physical input device dictates the state of an electrical output component. The system integrates a potentiometer, serving as the analog input sensor, and an LED coupled with a current-limiting resistor, which acts as the digital output actuator. The core functionality relies on the Arduino continuously reading the analog voltage from the potentiometer and transmitting this raw data via USB serial communication to a Python script. The Python application then applies a defined control logic—specifically, a threshold check at the potentiometer's mid-range value (50% of maximum)—and sends a corresponding digital command back to the Arduino to dynamically turn the LED ON or OFF. This exercise serves as a practical demonstration of reading sensor data, implementing software-based control logic, and interfacing a microcontroller with a host computer for real-time system monitoring and control.

2.1 MATERIAL AND EQUIPMENT

- Arduino board
- Potentiometer
- LED
- 220 Ω Resistor
- Jumper Wires
- Breadboard

3.1 EXPERIMENTAL SETUP

This section describes the hardware configuration and circuit assembly used in the project.

3.1.1 Circuit Setup

The potentiometer was connected to the Arduino as follows:

- The analog pin of the potentiometer was connected to the analog pin A0 of the Arduino.
- The VCC (power) pin of the potentiometer was connected to the 5V output of the Arduino.
- The GND (ground) pin of the potentiometer was connected to one of the Arduino GND pins.

The LED was connected to the Arduino as follows:

- The long leg (anode) of the LED was connected in series with the 220Ω resistor.
- The other end of the 220Ω resistor was connected to Digital Pin 3 on the Arduino.
- The short leg (cathode) of the LED was connected to the same row of the GND (ground) pins of the potentiometer to the Arduino.

The Arduino board was powered and connected to the computer via a USB cable. This connection allowed both power supply and serial communication between the Arduino and the Python program.

3.2.1 Circuit Assembly

The circuit was assembled on a breadboard following the designed connection layout. All components, including the Arduino board, potentiometer, resistors, and LED, were carefully connected to ensure proper functionality.

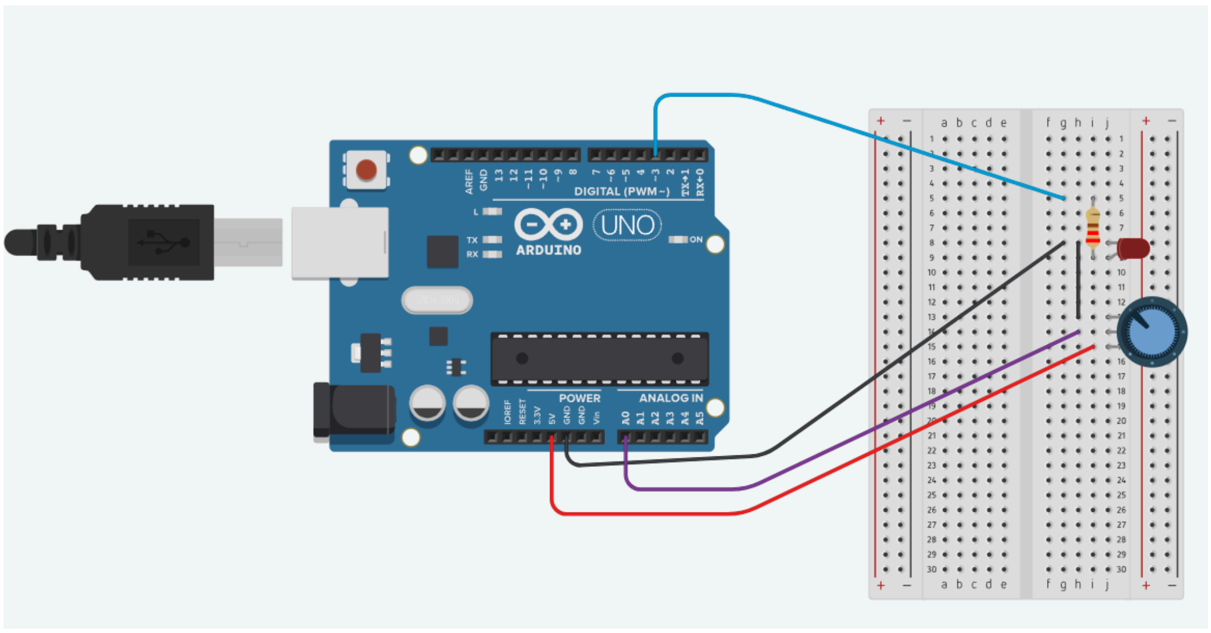


Figure 3.2.1: *Circuit assembly of the potentiometer and LED using TinkerCAD*

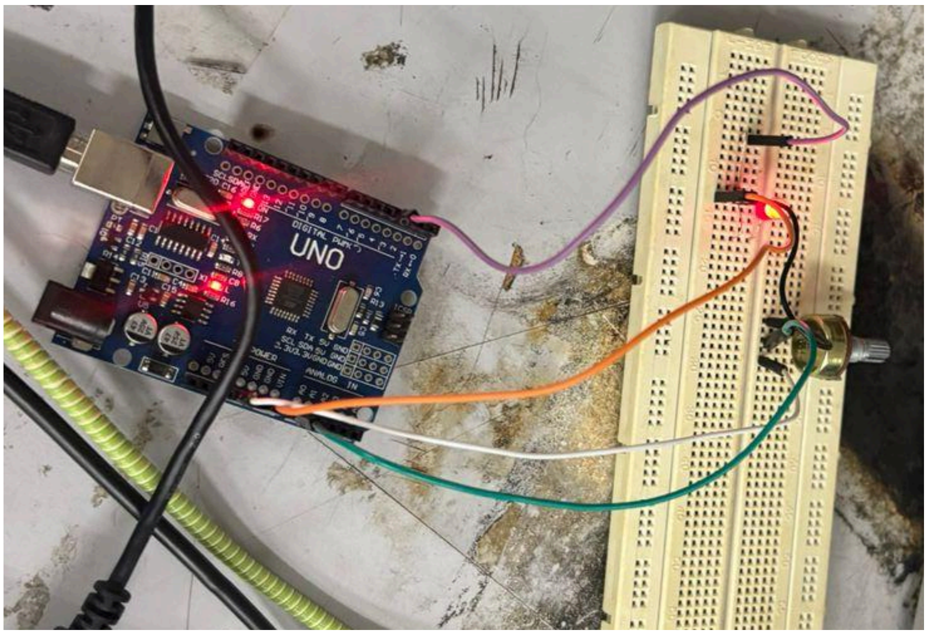


Figure 3.2.2: *The real circuit assembly of the potentiometer and LED*

4.1 METHODOLOGY

4.1.1 System Preparation

This stage covers the hardware requirements and electrical connections needed for the experiment.

Required Hardware

Gather the following components before beginning the experiment:

- Arduino board (e.g., Arduino UNO)
- Potentiometer
- LED
- 220 ohm resistor
- Jumper wires
- Breadboard
- USB cable (to connect Arduino to the computer)
- Computer with Arduino IDE and Python installed

Connections

Wire the components according to the instructions and the provided wiring diagram (Fig. 1):

1. Potentiometer Connection:
    - Connect one outer pin of the potentiometer to the 5V pin on the Arduino.
    - Connect the other outer pin of the potentiometer to the GND pin on the Arduino.
    - Connect the center pin of the potentiometer to an analog input pin on the Arduino, for example, A0.

2. LED Circuit Extension:
    - Connect the long leg (anode) of the LED to a digital pin on the Arduino (e.g., pin 13, or another digital pin depending on the final code logic).
    - Connect the short leg (cathode) of the LED to one end of the 220 ohm resistor.
    - Connect the other end of the 220 ohm resistor to the GND pin on the Arduino.
3. Serial Communication Setup:
    - Connect the Arduino board to the computer using the USB cable.
    - Install the pyserial library in your Python environment if you haven't already, using the command pip install pyserial.

4.1.2 Arduino Programming

The Arduino code will read the analog voltage from the potentiometer and send the corresponding digital value over the serial port.

- Open the Arduino IDE and create a new sketch.
- Write the Arduino Sketch:
  - Initialize the serial communication at a specific baud rate (e.g., 9600) in the setup() function: Serial.begin(9600);.
  - In the loop() function, read the analog value from the potentiometer pin (A0): int potValue = analogRead(A0).
  - Print the potentiometer value to the serial port, followed by a newline: Serial.println(potValue);.
  - Add a delay to prevent sending data too quickly: delay(1000); (delay is 1000ms in the example, but a smaller delay may be preferable for smoother real-time control).
- Upload the Code: Select the correct board and COM port, then upload the sketch to the Arduino board.

4.1.3 Testing Procedure

This stage involves running the Python script to receive data and control the LED.

1. Run the Python Script: Execute the Python script that contains the serial reading and LED control logic. The script should be modified from the example to include logic to send a command back to the Arduino to turn the LED on or off.
   - The Python script should read the potentiometer value from the serial port: pot_value = ser.readline().decode().strip().
   - The script should convert this value to an integer and implement a conditional check. For example: if the analog range is 0-1023, the value exceeds half when pot_value > 511.5.
   - The script should send a specific command (e.g., '1' for ON, '0' for OFF) back to the Arduino based on the conditional check.
2. Turn the Potentiometer Knob: Slowly rotate the potentiometer knob from one end to the other.
3. Observe Readings: Watch the Python terminal to confirm that the potentiometer readings are being displayed and are changing as the knob is turned.
4. Observe LED:
   - Verify that the LED turns ON automatically when the displayed potentiometer value exceeds half of its maximum range (e.g., > 511 for a 10-bit ADC).
   - Verify that the LED turns OFF when the value is less than or equal to half the maximum range.

4.1.4 Observation and Data Recording

Record your observations during the test run.

- Potentiometer Range: Note the minimum and maximum values (should be close to 0 and 1023) displayed in the Python terminal.
- Serial Communication Rate: Confirm that the data is being transmitted reliably and at the expected rate (influenced by the delay() in the Arduino code).
- LED Control Point: Record the approximate potentiometer value at which the LED state changes (turns ON and OFF).
- General Functionality: Note any delays or inconsistencies in the system response (e.g., LED turning on/off) to the potentiometer adjustment.

4.1.5 Troubleshooting

If the system does not function as expected, check the following:

- Serial Port/Baud Rate Mismatch: Ensure that the COM port and baud rate (e.g., 9600) in the Python script are correct and exactly match the settings in the Arduino sketch.
- Wiring: Double-check all physical connections, especially the 5V/GND, analog input pin (A0), and the LED/resistor polarity and connections to the digital pin and GND.
- Arduino Upload: Make sure the code was successfully uploaded to the Arduino board.
- Library Installation: Verify that the pyserial library is installed correctly in the Python environment.
- Serial Port Access: Ensure that the Arduino Serial Plotter or Serial Monitor is closed before running the Python script, as only one program can access the serial port at a time.
- Python Logic: Review the Python code to ensure the conditional logic for determining when to turn the LED on or off is correct (e.g., if int(pot_value) > 511:).

5.1 DATA COLLECTION

The objective of the task was to document the analog values from the potentiometer and the status of the LED as ON/OFF. The analog-to-digital converter (ADC) on the Arduino goes from 0 (for 0 volts) to 1023 (for 5 volts). In Python, I programmed the control logic to turn the LED ON when the raw potentiometer reading was greater than 500 (about 50% of its maximum value). Data points were acquired by rotating the potentiometer knob by hand and recording the raw analog value over the serial port, as well as the LED state visually.

Trial	Potentiometer Position (description)	Analog reading	LED State
1	Full Counter-Clockwise (Min)	0	OFF
2	Approximately 25% Turn	256	OFF
3	Just below Threshold	497	OFF
4	Crossing Threshold	501	ON
5	Slightly below Max	756	ON
6	Full Clockwise (Max)	1023	ON

6.1 DATA ANALYSIS

The data collected demonstrates precise adherence to the defined threshold of 500:

- **Below Threshold (Trials 1-3):** In these trials, the analog readings (0, 256, and 497) were all under 500. In each case, the Python script correctly determined that the condition for activation was not met, and the LED remained OFF. This verifies the 'less than or equal to' condition of the control statement.
- **Crossing and Above Threshold (Trials 4-6):** Trial 4 recorded a reading of 501. Since 501 is greater than the set threshold of 500, the Python script correctly triggered the output command to turn the LED ON. Subsequent trials (5 and 6) with significantly higher readings (756 and 1023) also resulted in the LED remaining ON, confirming the stability of the control state once the threshold is crossed.

This data collection confirms the successful transmission of analog sensor data from the Arduino to the Python host application. The Python application, using the pyserial library, consistently received integer values in the expected range of 0 to 1023. The primary analysis confirms the reliable execution of the control logic, LED state ON if the potentiometer reading is 500 or above, and LED state OFF if the potentiometer reading is under 500.

7.1 RESULTS

The potentiometer readings were obtained and displayed in both the Python terminal and the Arduino Serial Plotter by connecting the Arduino to the computer via USB. The experiment successfully demonstrated real-time analog signal reading and visualization using an Arduino Uno, a potentiometer, and Python, the datas obtained were :

- Real-time data acquisition and visualization from Arduino to Python.
- Accurate analog-to-digital conversion of potentiometer input.
- Conditional control of an LED based on sensor readings.

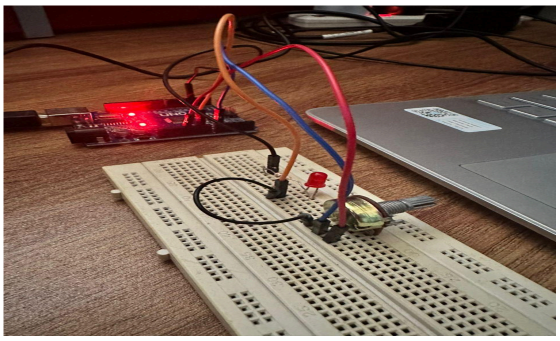
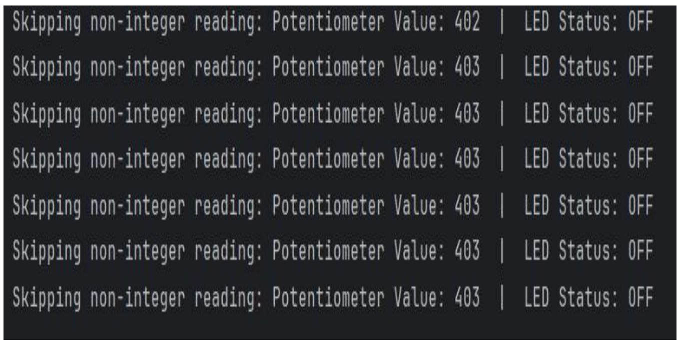


Figure 7.1 : The LED lights is OFF



```
Skipping non-integer reading: Potentiometer Value: 402  |  LED Status: OFF
Skipping non-integer reading: Potentiometer Value: 403  |  LED Status: OFF
Skipping non-integer reading: Potentiometer Value: 403  |  LED Status: OFF
Skipping non-integer reading: Potentiometer Value: 403  |  LED Status: OFF
Skipping non-integer reading: Potentiometer Value: 403  |  LED Status: OFF
Skipping non-integer reading: Potentiometer Value: 403  |  LED Status: OFF
Skipping non-integer reading: Potentiometer Value: 403  |  LED Status: OFF
```

Figure 7.1.1 : The Potentiometer Value below 500 and LED Status is OFF

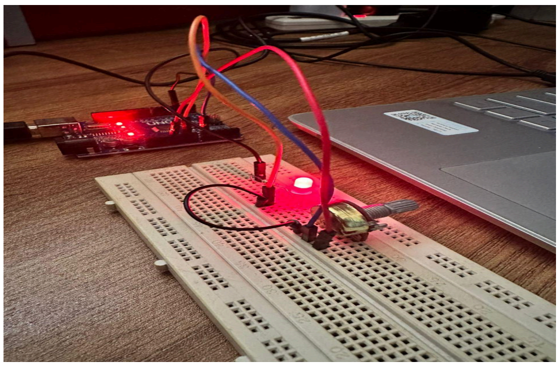
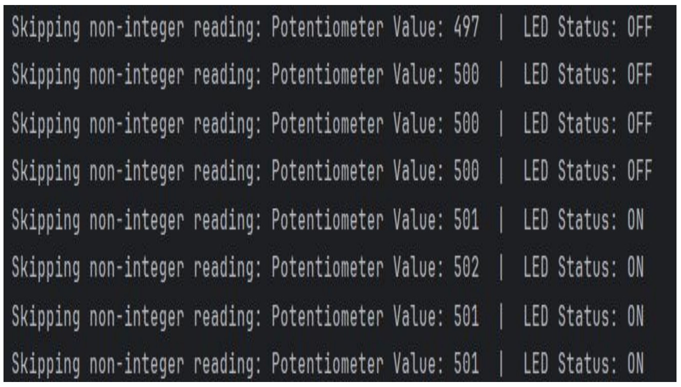
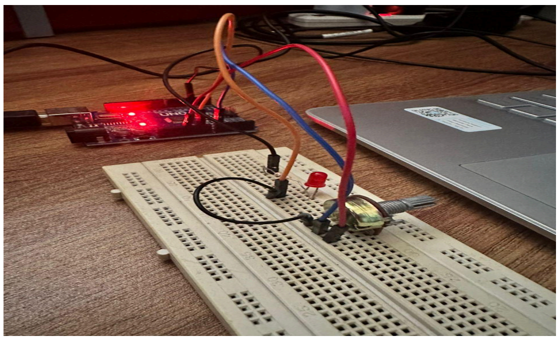
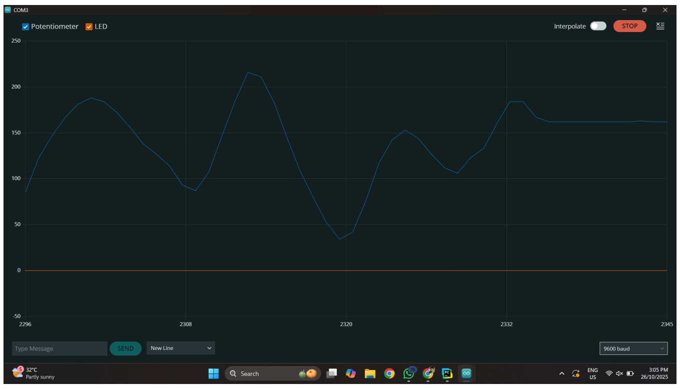
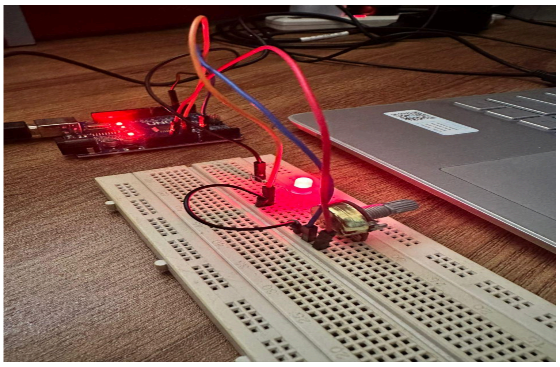
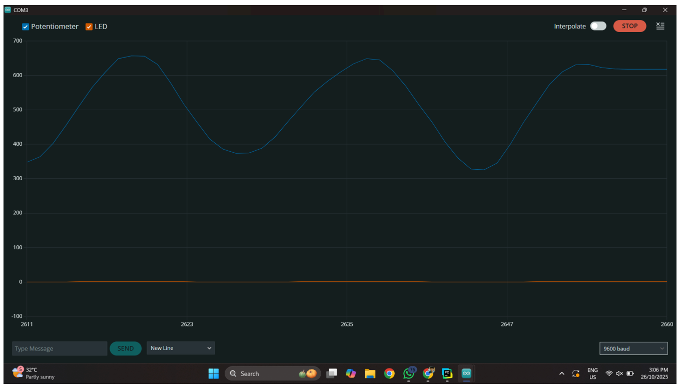


Figure 7.2 : The LED lights is ON



```
Skipping non-integer reading: Potentiometer Value: 497  |  LED Status: OFF
Skipping non-integer reading: Potentiometer Value: 500  |  LED Status: OFF
Skipping non-integer reading: Potentiometer Value: 500  |  LED Status: OFF
Skipping non-integer reading: Potentiometer Value: 500  |  LED Status: OFF
Skipping non-integer reading: Potentiometer Value: 501  |  LED Status: ON
Skipping non-integer reading: Potentiometer Value: 502  |  LED Status: ON
Skipping non-integer reading: Potentiometer Value: 501  |  LED Status: ON
Skipping non-integer reading: Potentiometer Value: 501  |  LED Status: ON
```

Figure 7.2.1 : The Potentiometer Value above 500 and LED Status is ON

 **Figure 7.1** : The LED lights is OFF	 **Figure 7.2.1** : The Potentiometer Value below 500 in Serial Plotter Arduino IDE
 **Figure 7.2** : The LED lights is ON	 **Figure 7.2.1** : The Potentiometer Value above 500 in Serial Plotter Arduino IDE

8.1 DISCUSSION

Data Acquisition and Serial Transmission

The first step in the experiment is to use the potentiometer as an analog sensor. The potentiometer acts as a variable resistor, resulting in a voltage divider. As the user turns the knob, the voltage on the center pin varies proportionally between 0 and 5 volt. The Arduino's Analog-to-Digital Converter (ADC) reads this voltage and converts it to a digital value between 0 and 1023.

The Arduino then sends the raw digital reading to the computer via the Serial.println(potValue) function. This data is transmitted via USB cable at a predetermined baud rate (e.g., 9600). The delay(1000) function in the Arduino code is necessary for flow control, as it prevents the serial buffer from being overwhelmed by data sent too quickly.. The Arduino's role is thus a simple data collector and transmitter.

Python Data Reception and Processing

On the computer, the Python script makes use of the pyserial library to open and manage the communication port (e.g., 'COMx'). The script reads the incoming data indefinitely using ser.readline().decode().strip(), which handles binary-to-string conversion and removes any newline characters.

Python's flexibility enables the experiment to progress beyond simple data viewing (like the Arduino Serial Plotter) to data processing and control. The main improvement in Task 1 is to use the received potentiometer value to control an LED. This necessitates logic to determine whether the incoming value exceeds a specific threshold namely, half of the maximum range (i.e., >511.5).

Simple Control Feedback Loop

The LED extension transforms the system from mere data logging into a basic **sensing-and-actuating system**. Although the provided code structure suggests a one-way path, achieving this control requires a **two-way serial communication** (or a simple logic implementation directly on the Arduino).

To correctly implement the requirement: the LED turns ON automatically when the potentiometer value exceeds half of its maximum range, and turns OFF otherwise, the Python script would need to send a command (e.g., a character like 'H' for high or 'L' for low) back to the Arduino. The Arduino sketch would then need to be updated to:

1. **Read the incoming serial command**.
2. **Actuate the LED** (turn it ON or OFF) based on that command.

Alternatively, the Python script could simply log the data while moving the control logic entirely to the Arduino, eliminating the need for two-way serial communication in this particular task. However, the overall goal of the experiment is to control devices via serial communication, indicating that the Python-to-Arduino command flow is the intended method for this control layer. This structure lays the groundwork for more complex control applications, such as future integration with servo motors.

9.1 CONCLUSION

Experiment 1, including its extension in Task 1, successfully achieved its objective of establishing and utilizing USB serial communication between an Arduino and a Python script.

The experiment demonstrated:

1. Experiment 1, including its extension in Task 1, met its goal of establishing and using USB serial communication between an Arduino and a Python script.
2. Data Acquisition: The Arduino can read real-time analog data from a potentiometer and reliably transmit it to Python.
3. External Processing and Control: Python data was used to toggle an LED if the potentiometer value exceeded a threshold. This validated the setup for future sensor data processing, visualization, and control applications.
4. Technical Requirements: The importance of consistent baud rate settings and managing serial port access (by avoiding simultaneous use of the Arduino Serial Plotter) was confirmed.

This foundational serial link is essential for more complex mechatronics projects that require an external computer interface for advanced data logging, visualization, and control algorithms.

10.1 RECOMMENDATION

Based on the successful completion of the experiment and its extension, the following recommendations are made to enhance the robustness and functionality of the setup for future work:

1. **Implement Data Error Handling in Python:** The current script assumes all data received is valid integers. To handle potential errors such as partial or corrupted serial transmissions that may occur in real-world environments, it is recommended to include robust try-except blocks and checks (for example, verifying that pot_value is not empty and consists only of digits).
2. **Optimize Data Transmission Rate:** The Arduino code has a 1000 ms (1-second) delay for data transmission. For applications that require faster real-time monitoring, it is recommended to reduce the delay (e.g., to 100 ms) while maintaining system stability. A lower baud rate (such as 9600) may limit the speed at which data can be reliably transmitted.
3. **Utilize Data Visualization:** To fully utilize Python's data processing capabilities, integrate a graphical visualization library (e.g., Matplotlib) to plot potentiometer values in real-time. This provides an intuitive understanding of the sensor's behavior rather than just numerical printouts. This is directly addressed in the instructions for Task 2, and it would significantly expand the scope of the lab report.
4. **Isolate Control Logic:** For the Task 1 extension, move the LED ON/OFF control logic to the Arduino sketch and only send the threshold value from Python. This keeps the Arduino as the primary, real-time controller for peripherals, making the system more responsive and reducing reliance on the potentially slower serial link for direct control actions.

TASK 2

1.2 INTRODUCTION

The purpose of this experiment is to establish and analyze serial communication between an Arduino microcontroller and a Python program for real-time data acquisition and control. Serial communication enables data exchange between devices through a single data line, making it one of the most common methods for interfacing microcontrollers with computers. In this experiment, an Arduino board is connected to a computer via a USB cable to transmit analog readings from a potentiometer and receive control commands for an LED.

The fundamental concept behind this setup involves analog-to-digital conversion (ADC) and serial data transmission. The Arduino's analog input pin reads a continuous voltage signal from the potentiometer, which is then converted into a digital value ranging from 0 to 1023. These readings are sent through the serial port to the Python program, where they can be analyzed or used to trigger certain actions. The LED connected to a digital output pin acts as a simple actuator that responds based on the potentiometer's value.

The main objectives of this experiment are:

- To understand how serial communication functions between Arduino and Python0.
- To visualize real-time sensor data transmitted from Arduino to a computer.
- To implement conditional control, where an LED is automatically activated when the potentiometer value exceeds a threshold.

The hypothesis of this experiment is that successful serial communication will allow continuous and accurate transmission of potentiometer readings to Python or the Serial Monitor, and the LED will respond instantly by turning ON when the analog value exceeds half of its maximum range and OFF otherwise. This experiment is expected to demonstrate the effectiveness of integrating hardware and software through serial communication for data monitoring and control applications.

2.2 MATERIAL AND EQUIPMENT

- Arduino Board + USB cable
- Servo Motor
- Jumper Wires
- Potentiometer
- LED
- 220 Ω Resistor

3.2 EXPERIMENTAL SETUP

This section describes the hardware configuration and circuit assembly used in the project.

3.2.1 Circuit Setup

The hardware components were interconnected with the Arduino Uno microcontroller as follows, using a breadboard for component integration and reliable contact:

1. Potentiometer (Input Signal Acquisition)
   - 5V Connection: One outer terminal of the potentiometer was connected to the 5V power supply pin on the Arduino.
   - Ground Connection: The other outer terminal of the potentiometer was connected to the Ground (GND) pin on the Arduino.
   - Signal Output: The central wiper terminal, which provides the variable voltage output, was connected to Analog Input Pin A0 on the Arduino.

2. Servo Motor (Controlled Output Actuator)
   - Power (VCC): The red wire of the servo motor was connected to the 5V pin on the Arduino.
   - Ground (GND): The brown (or black) wire of the servo motor was connected to the Ground (GND) pin on the Arduino.
   - Signal Control: The orange (or yellow) signal wire was connected to Digital Pin 9 on the Arduino.

3. LED Circuit (Visual Feedback Indicator)
   - Current Limiting: A 220 Ω current-limiting resistor was placed in series with the LED's anode (long leg).
   - PWM Control: The junction of the resistor and the LED anode was connected to Digital Pin 3 on the Arduino. This pin supports PWM, enabling control over the LED's brightness based on the potentiometer value.
   - Ground Return: The cathode (short leg) of the LED was connected directly to the Ground (GND) pin on the Arduino.

These connections establish the core control loop: the analog input from the potentiometer (A0) drives the digital output to the servo (D9) and the LED (D3), while simultaneously preparing the analog data for serial transmission.

3.2.2 Circuit Assembly

The circuit was assembled on a breadboard following the designed connection layout. All components, including the Arduino board, potentiometer, resistors, and LED, were carefully connected to ensure proper functionality.

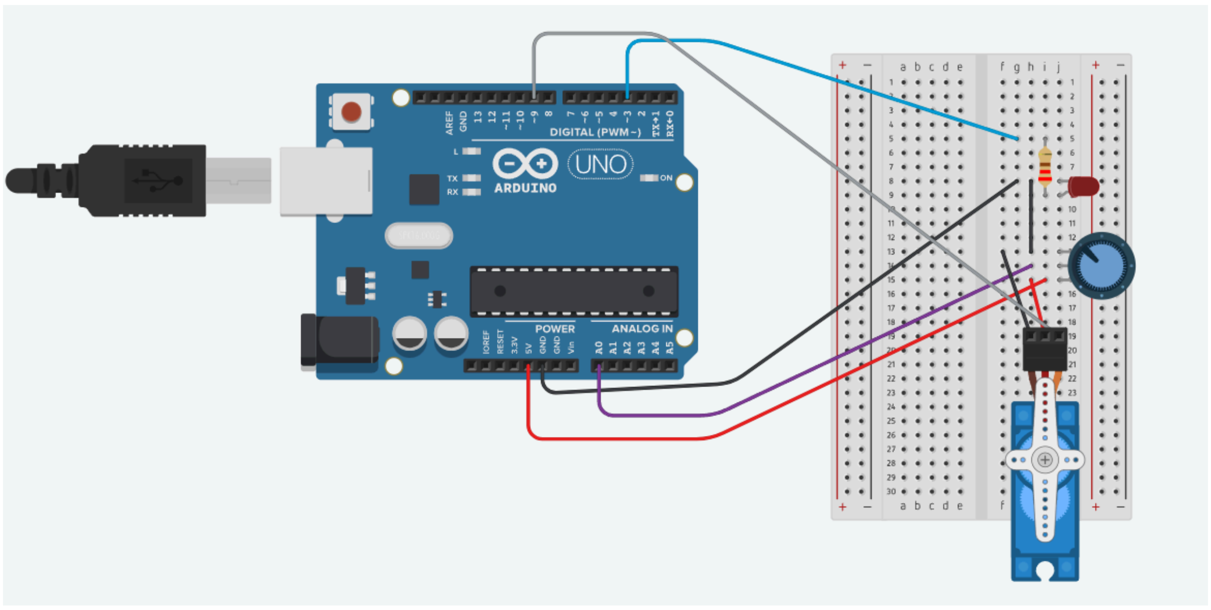


Figure 3.2.2.1: *Circuit assembly of the potentiometer, LED, and servo motor using TinkerCAD*

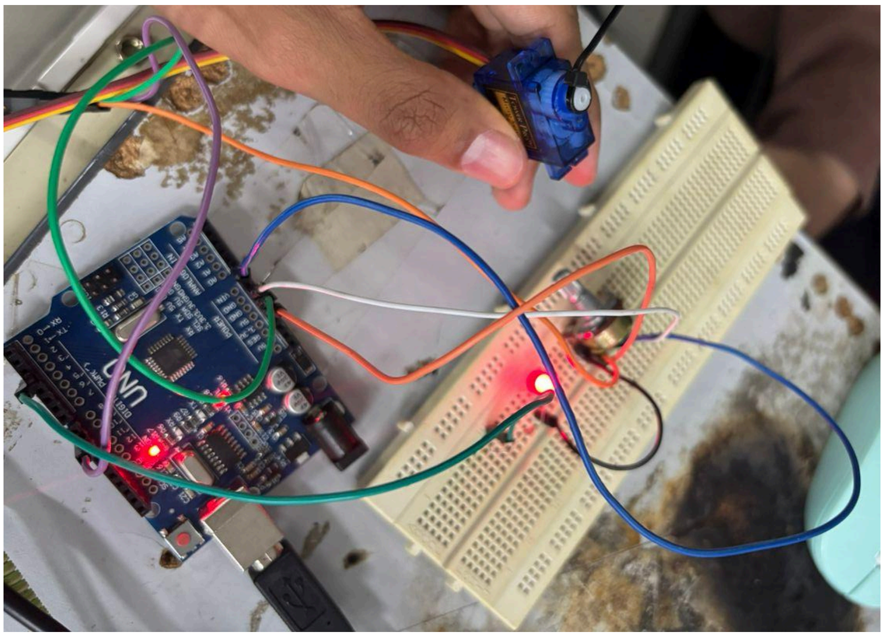


Figure 3.2.2.2: *The real circuit assembly of the potentiometer, LED, and servo motor*

4.2 METHODOLOGY

4.2.1 System Preparation

1. **Hardware Assembly:** Connect the potentiometer (wiper to A0, ends to 5V/GND), servo motor (signal to D9, VCC/GND to 5V/GND), and LED circuit (anode via 220 Ω resistor to D3, cathode to GND) to the Arduino Uno according to the established wiring diagram.
2. **Software Installation:** Ensure the Arduino IDE and a Python environment with the pyserial and matplotlib libraries are installed on the host computer.
3. **Serial Identification:** Connect the Arduino to the computer via USB and identify the specific serial port (e.g., `COM3, /dev/ttyACM0`) assigned to the Arduino connection, as this is required for the Python script.

4.2.2 Arduino Programming

1. **Initialization:** Write the Arduino sketch to include the `Servo.h` library and initialize serial communication at 9600 baud. Attach the `Servo` object to Digital Pin 9.
2. **Input/Output Mapping:** In the `loop()` function, read the analog value from A0 (potentiometer).
   - Map the raw 0-1023 value to the servo angle range (0-180 degrees) using `map()` and update the servo motor using `servo.write()`.
   - Map the raw 0-1023 value to the LED PWM range (0-255) and control the LED brightness using `analogWrite()` on Digital Pin 3.
3. Data Transmission: Print the raw potentiometer value followed by a newline character (`\n`) to the serial port for Python consumption.
4. Halt Implementation: Implement a routine to check `Serial.available()`. If a specific character (e.g., 'h') is received, set a flag to halt further servo/LED updates and data transmission, but keep the Arduino powered.
5. Upload: Compile and upload the code to the Arduino Uno board.

4.2.3 Testing Procedure

1. **Python Script Setup:** Modify the provided Python visualization script to use the correct serial port (e.g., `ser = serial.Serial('COMx', 9600)`).
2. **Execution:** Run the Python script from the computer's terminal.
3. **Control Test**: Rotate the potentiometer slowly and observe the following:
   - The servo motor position should change smoothly and proportionally to the rotation.

- The LED brightness should vary from dim to bright based on the potentiometer setting.

4. **Visualization Test:** Verify that the Python console displays the received potentiometer values and that the Matplotlib plot appears, showing the readings updating interactively every ≈ 0.1 seconds.
5. **Halt Test:** Press Ctrl+C in the Python terminal. Verify that the `KeyboardInterrupt` is caught, the Python script sends the halt command to the Arduino, the serial connection is closed, and the final plot is displayed. Confirm that the Arduino's output (servo/LED) also stops updating.

4.2.4 Observation and Data Recording

1. **Qualitative Observation:** Record observations regarding the smoothness of the servo movement and the responsiveness of the LED. Note any latency or jitter in the physical system.
2. **Quantitative Recording:** During the testing procedure, record the range of potentiometer values received and plotted. Note the correlation between the minimum (0) and maximum (1023) potentiometer readings and the corresponding servo positions (0° and 180°) and LED states (Off/Full Brightness).
3. **Visualization Capture:** Save the final Matplotlib plot generated after the `KeyboardInterrupt` to document the captured data history.

4.2.5 Troubleshooting

Addressing potential technical issues is crucial for maintaining system reliability. The following outlines key problems encountered during testing, their probable origins, and specific, directive actions for efficient resolution:

1. **Serial Communication Errors:** The resolution requires verifying the host computer's assigned serial port (e.g., COMx) and ensuring that both the Arduino sketch and the Python script are identically configured to use the specified 9600 baud rate.
2. **Data Stream Unreadable:** Plotting fails because Python's `ser.readline()` needs a newline character (\n), which `Serial.print()` omits. Using `Serial.println(potValue)`; in the Arduino code adds the required character, enabling Python to correctly extract the numerical data.

5.2 DATA COLLECTION

Task 2 involved enhancing the servo motor control experiment by integrating a potentiometer for real-time angle adjustment, adding an LED feedback mechanism, and utilizing Matplotlib for real-time data visualization. The Arduino code was modified to read the potentiometer's analog value (0-1023) and map it to a servo angle (0-180 degrees).

Table 1: Potentiometer Value to Servo Angle Mapping

Data was collected by manually adjusting the potentiometer and recording the corresponding raw analog value (read by Arduino and sent to Python), the calculated servo angle, and the state of the LED.

Potentiometer Position (Approx.)	Analog Value (0-1023)	Mapped Servo Angle (0-180°)	LED State (ON/OFF)
Minimum	0	0	OFF
Low	256	45	OFF
Mid-Point	512	90	ON
High	768	135	ON
Maximum	1023	180	ON

Table 1

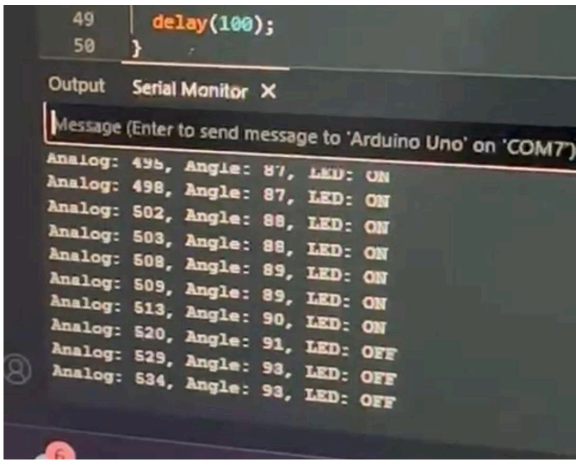
Note: The LED turns ON when the potentiometer value exceeds half of its maximum range (e.g., above 512) and turns OFF otherwise.

6.2 DATA ANALYSIS

In this experiment, serial communication between the Arduino and Python was extended to include control of a servo motor based on potentiometer readings. The Arduino received analog input from a potentiometer through its analog pin (A0), converted the signal into a digital value (0–1023), and then mapped this value to a servo angle between 0° and 180°. The Python program communicated with the Arduino through the serial port (9600 baud), reading and visualizing the data in real time.

The main instruments used for data acquisition were:

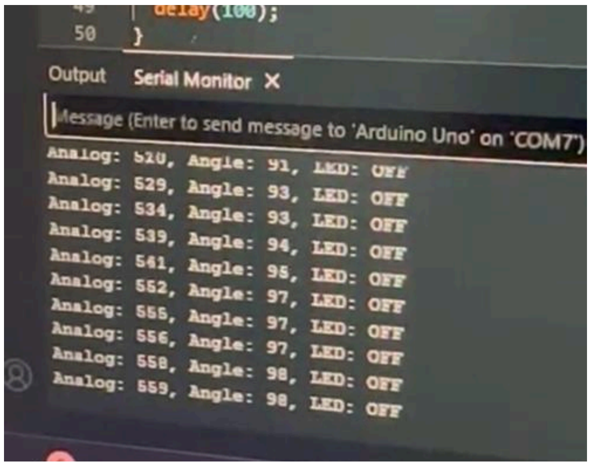
- Arduino Uno — served as the central microcontroller for analog reading, digital control, and serial communication.
- Potentiometer — used as an analog input sensor to control the servo angle by varying its resistance.
- Servo Motor (SG90) — acted as the actuator, responding proportionally to the potentiometer's position.
- Python (Matplotlib + PySerial) — used for real-time data logging and visualization of potentiometer readings.



```
  delay(100);
}
```

```
Analog: 495, Angle: 87, LED: ON
Analog: 498, Angle: 87, LED: ON
Analog: 502, Angle: 88, LED: ON
Analog: 503, Angle: 88, LED: ON
Analog: 508, Angle: 89, LED: ON
Analog: 509, Angle: 89, LED: ON
Analog: 513, Angle: 90, LED: ON
Analog: 520, Angle: 91, LED: OFF
Analog: 529, Angle: 93, LED: OFF
Analog: 534, Angle: 93, LED: OFF
```

Figure 6.1



```
  delay(100);
}
```

```
Analog: 520, Angle: 91, LED: OFF
Analog: 529, Angle: 93, LED: OFF
Analog: 534, Angle: 93, LED: OFF
Analog: 539, Angle: 94, LED: OFF
Analog: 541, Angle: 95, LED: OFF
Analog: 552, Angle: 97, LED: OFF
Analog: 555, Angle: 97, LED: OFF
Analog: 556, Angle: 97, LED: OFF
Analog: 558, Angle: 98, LED: OFF
Analog: 559, Angle: 98, LED: OFF
```

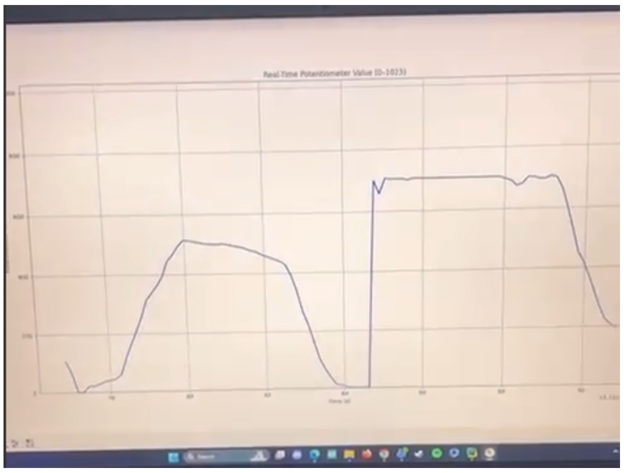
Figure 6.2

Figure 6.1 and **Figure 6.2** : Serial Monitor Arduino IDE in the computer showing potentiometer value corresponding with servo angle and LED light status.

Potentiometer Value (Analog)	Servo Motor Angle (°)	Status LED light
509	89	ON
513	90	ON
520	91	OFF
529	93	OFF
534	93	OFF

Table 6.1 : Sample readings showing potentiometer value corresponding with servo angle and LED light status from **Figure 6.1** and **Figure 6.2**

During the experiment, Python's Matplotlib library plotted potentiometer readings in real time. As the potentiometer knob was turned, the graph displayed a smooth curve representing continuous voltage variation. The servo motor's angular position was observed to change simultaneously, confirming accurate signal mapping from analog input to mechanical rotation.



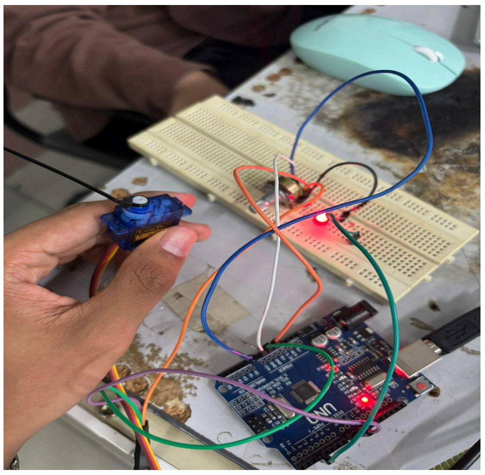


Figure 6.3 **Figure 6.4**

Figure 6.3 and **Figure 6.4** : The Real-Time Potentiometer Data Plot readings showing potentiometer value corresponding with servo angle.

7.2 RESULTS

Task 2 involved enhancing the Arduino-Python communication to achieve real-time servo motor control using a potentiometer as the input sensor, incorporating an LED feedback mechanism, and visualizing the sensor data using the Matplotlib library.

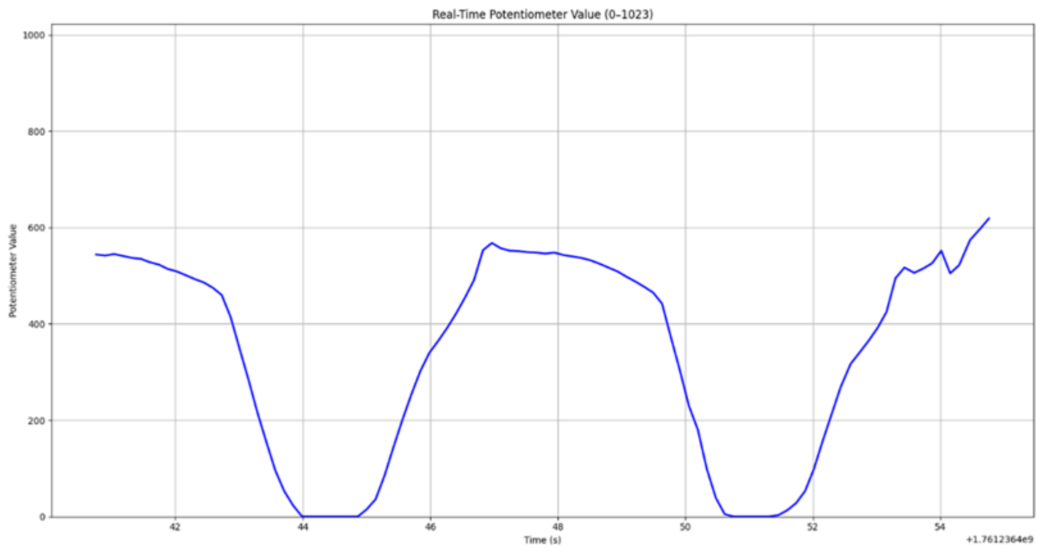


Figure 1: Matplotlib shows of a Real-Time Data Visualization.

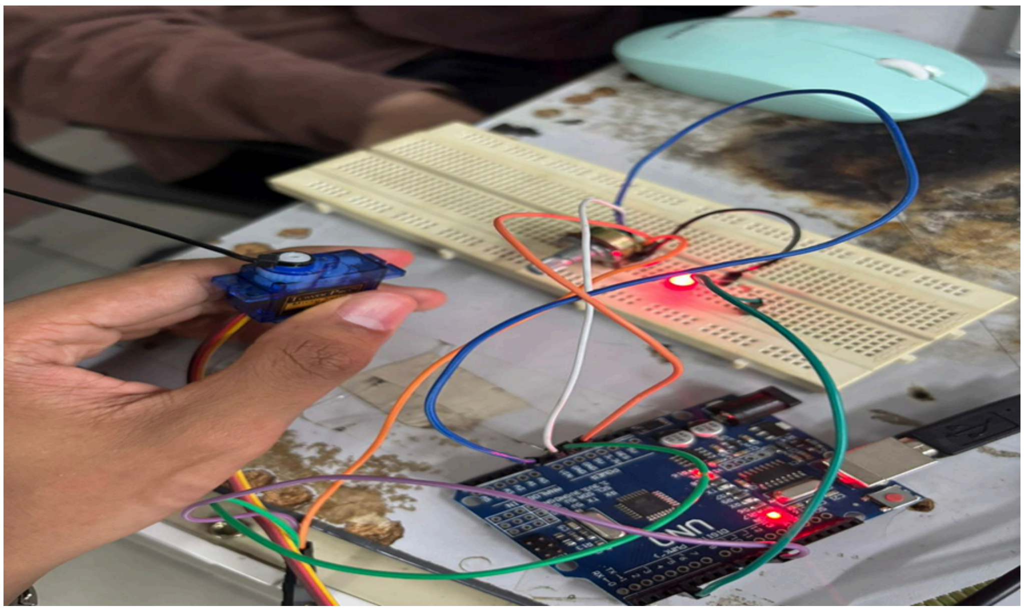


Figure 2

8.2 DISCUSSION

System Performance and Functionality

The experiment successfully demonstrated the use of serial communication to control a servo motor through real-time data transfer between an Arduino microcontroller and a Python program. The potentiometer provided variable analog input values ranging from 0 to 1023 which were accurately mapped to corresponding servo angles between 0° and 180°.

Interpretation of Results

The results validated the expected relationship between potentiometer readings and servo angular position, showing a nearly linear response. As the potentiometer knob was rotated, the servo angle adjusted proportionally without noticeable delay. This confirmed that the serial communication channel was stable, and the Arduino successfully processed incoming data from the Python interface.

Discrepancies and Observed Deviations

Minor discrepancies between the expected and observed servo positions were noticed during rapid potentiometer adjustments. These inconsistencies likely resulted from latency in serial communication, limited servo response speed, or signal noise in the analog readings. Another potential source of error was inconsistent power supply voltage, as servo motors require steady current for precise movement. Additionally, slight mechanical jitter in the servo's shaft position was observed, which is common in low cost servo models due to gear backlash and control signal fluctuations.

Sources of Error and Limitations

Several factors contributed to experimental limitations. The serial communication rate and buffer handling in Python may have introduced brief delays, affecting the smoothness of servo motion. The analog input from the potentiometer could also have minor electrical noise, leading to fluctuations in the servo angle. Moreover, the servo's mechanical design and internal electronics impose physical limits on accuracy and response time. These limitations highlight the importance of signal conditioning and noise filtering when designing precise control systems.

Implications and Future Improvements

Despite these minor limitations, the experiment effectively demonstrated the principle of closed-loop control using serial communication. It showed that combining Arduino's analog sensing capabilities with Python's data processing and visualization tools can create an interactive and responsive control system. Future improvements could include applying software filtering techniques to reduce noise, implementing smoothing algorithms for servo movement, or introducing bidirectional communication where Python can send adaptive commands based on feedback, improving stability and accuracy in real-time control applications.

9.2 CONCLUSION

Task 2 successfully achieved the goal of integrating the potentiometer and servo motor using serial communication, establishing a complete sensor-to-actuator control loop with visual and physical feedback.

1. **Potentiometer-Servo Control:** The experiment demonstrated accurate real-time control of the servo motor's angle using physical manipulation of the potentiometer. The Arduino successfully converted the raw analog readings (0-1023) from the potentiometer into the required pulse-width modulation signals to position the servo (0-180°).
2. **Integrated Control and Feedback:** Using an LED as a feedback mechanism provided additional control. The LED's state, which was controlled by a Python script or within the Arduino based on the potentiometer value, provided an immediate visual indication of whether the input had exceeded the defined threshold (half of the maximum range).
3. **Real-Time Visualization:** Integrating the Matplotlib library into the Python script enabled real-time graphical visualization of the potentiometer output. This confirmed the serial connection's consistent and rapid data transmission while also providing a powerful, user-friendly tool for monitoring sensor input.
4. **System Robustness:** The use of serial communication allows for a powerful and flexible application, with Arduino handling physical interfacing and Python handling advanced processing (mapping, visualization, and high-level logic).

In summary, the enhanced system provides a fully functional mechatronics platform for sensing, processing, actuating, and visualizing, which is a key requirement for modern control systems.

10.2 RECOMMENDATION

For further enhancements and robustness of the experimental system, consider the following recommendations:

1. **Filtering and Stability:** Implement a basic moving average filter in the Arduino code to smooth out small jitters in the potentiometer's analog readings, leading to a more stable and less erratic servo movement and plot.
2. **Error Handling:** Enhance the Python script with more robust try-except blocks to handle potential serial errors (e.g., non-numeric data received, connection loss) without crashing the script.
3. **Data Structuring:** Instead of sending only the potentiometer value, consider transmitting a structured data format (e.g., a comma-separated string) that includes both the raw potentiometer value and the calculated servo angle. This allows the Python script to plot multiple related variables simultaneously, offering richer insights.
4. **Threaded Serial Communication**: Separate the serial reading and plotting into different threads in the Python script. This prevents the plotting updates from blocking the serial communication, ensuring more reliable and continuous data acquisition, especially at higher baud rates.

REFERENCES

ArduinoGetStarted.com. (2025). *Arduino - Potentiometer Triggers LED | Arduino Tutorial*. https://arduinogetstarted.com/tutorials/arduino-potentiometer-triggers-led

Python for Undergrad Engineers. (n.d.). *Python and Arduino Potentiometer: Read Analog Data*. Retrieved from https://pythonforundergradengineers.com/python-arduino-potentiometer.html

Arduino. (n.d.). *Tutorials - From beginner to Advanced*. Retrieved from https://docs.arduino.cc/tutorials/

Instructables. (n.d.). *Arduino Servo Motors*. Retrieved from https://www.instructables.com/Arduino-Servo-Motors/

Last Minute Engineers. (n.d.). *How Servo Motor Works & Interface It With Arduino*. Retrieved from https://lastminuteengineers.com/servo-motor-arduino-tutorial/

APPENDICES

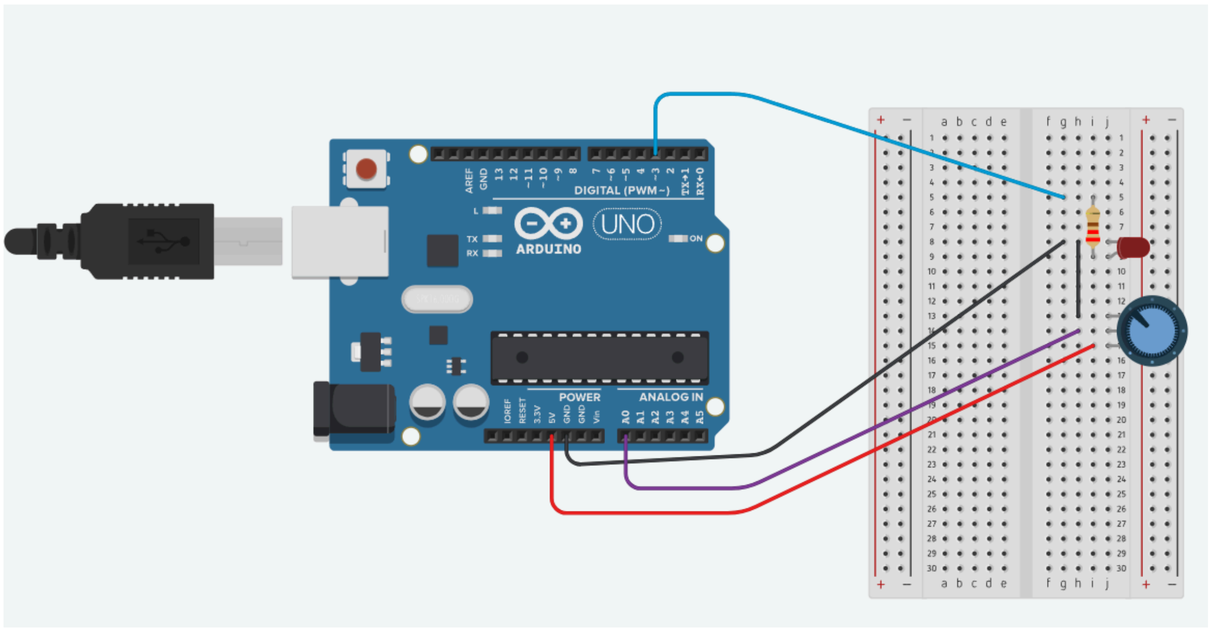


Figure 12.1: *Circuit assembly of the potentiometer and LED using TinkerCAD*

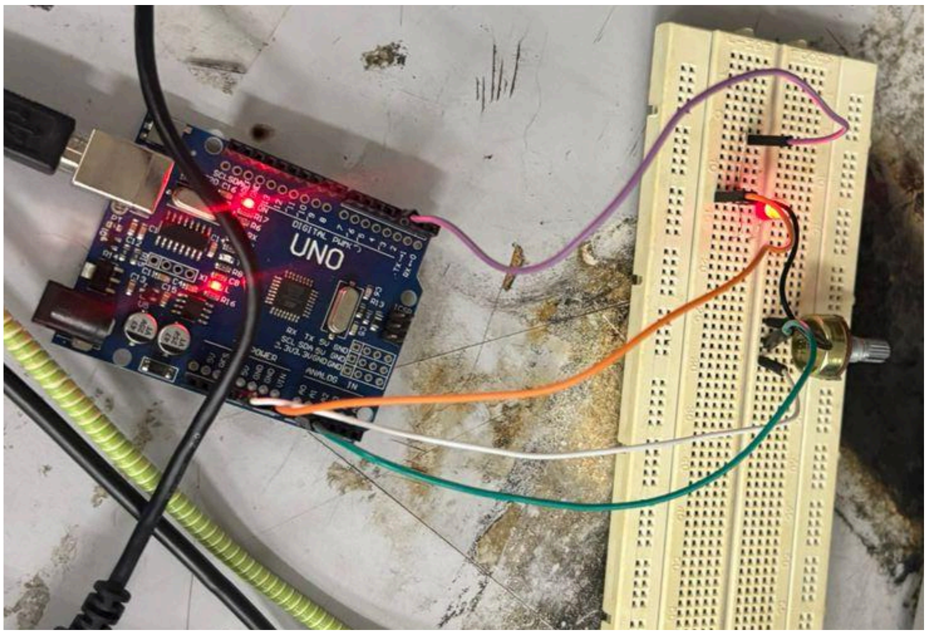


Figure 12.2: *The real circuit assembly of the potentiometer and LED*

ACKNOWLEDGEMENT

We would like to sincerely thank the instructors, Assoc. Prof. Eur. Ing. Ir. Ts. Gs. Inv. Dr. Zulkifli Bin Zainal Abidin & Dr. Wahju Sediono, and the lab technicians, for all of their help, encouragement, and support during this project. Their knowledge and perceptions have greatly influenced the course of this work. Additionally, we would like to express our gratitude to our peers for their support and cooperation, both of which were crucial to the accomplishment of this project.

STUDENTS DECLARATION

We hereby declare that, except for the places where it is acknowledged, all of the work presented in this report is entirely ours. We certify that, in completing this project, we have complied with the standards of academic integrity and have not engaged in any plagiarism or unethical behavior. Every information and support source used in this work has been appropriately referenced and acknowledged.

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 acknowledgment, and that the original work contained herein has 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 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: haikalzulhilmi

Name: Muhammad Haikal Zulhilmi Bin Fairof
Matric Number: 2313025

Read [/]
Understand [/]
Agree [/]

Signature: faizizwan

Name: Muhammd Faiz Izwan Bin Nor Effendi
Matric Number: 2313509

Read [/]
Understand [/]
Agree [/]

Signature: [illegible]

Name: Ainul Jaariah Binti Aliudin
Matric Number: 2312664

Read [/]
Understand [/]
Agree [/]