



MECHATRONICS SYSTEM INTEGRATION (MCTA 3203)

SEMESTER 1, 2025/2026

WEEK 3A & 3B : SERIAL COMMUNICATION BETWEEN ARDUINO AND

PYTHON

SECTION 1

GROUP 20

LECTURER: DR. WAHJU SEDIONO

DATE OF EXPERIMENT: Monday, 20th October 2025

DATE OF SUBMISSION: Monday, 27th October 2025

PREPARED BY:

NAME	MATRIC NUMBER
ALI RIDHA BIN MUHAMMAD AMINUDIN	2316137
SITI NORATHIRAH BINTI RAZALLY	2319788
NUR ALYA SAKINAH BINTI MOHD MOKHTAR	2319444

TABLE OF CONTENTS

ABSTRACT	3
INTRODUCTION	4
EXPERIMENT 3A: READING POTENTIOMETER VALUES VIA SERIAL COMMUNICATION	5
1.0 MATERIALS AND EQUIPMENTS	5
2.0 EXPERIMENTAL SETUP	6
3.0 METHODOLOGY	7
3.1 Circuit Assembly	7
3.2 Programming Logic	7
3.3 Code Used	8
3.4 Control Algorithm	10
4.0 DATA COLLECTION	11
5.0 DATA ANALYSIS	13
6.0 RESULT	15
EXPERIMENT 3B: TRANSMITTING ANGLE DATA FROM PYTHON SCRIPT TO AN ARDUINO, THEM ACTUATES SERVO TO MOVE TO THE SPECIFIED ANGLE	16
1.0 MATERIALS AND EQUIPMENTS	16
2.0 EXPERIMENTAL SETUP	17
3.0 METHODOLOGY	19
3.1 Circuit Assembly	19
3.2 Programming Logic	19
3.3 Code Used	20
3.4 Control Algorithm	23
4.0 DATA COLLECTION	25
5.0 DATA ANALYSIS	27
6.0 RESULT	29
DISCUSSION	30
CONCLUSION	31
RECOMMENDATIONS	32
REFERENCES	33
APPENDICES	34
Certificate of Originality and Authenticity	37

ABSTRACT

These experiments show how to integrate serial communication between Python and Arduino to visualise sensor data and control a servo motor. In the first experiment, the Arduino's analogue input was used to read the potentiometer values, while the PySerial library was used to send the data to Python. In the second experiment, angle values were transmitted from Python to Arduino via the serial interface in order to control the position of a servo motor. By including LED indicators and live data graphing with Matplotlib for visualisation, the system was able to control servo movement in real time in response to potentiometer changes or user input. Through hands-on experience with communication protocols, these experiments enhance our understanding of hardware-software interaction in embedded systems.

INTRODUCTION

In this lab report, we explore the concept of serial communication between a microcontroller (Arduino) and a computer-based system using Python. Serial communication plays an important role in mechatronic applications, allowing data and commands to be exchanged between sensors, actuators, and software interfaces. This experiment focuses on two main applications: transmitting potentiometer readings from Arduino to Python, and controlling a servo motor through serial communication from Python to Arduino.

In the first part of the experiment, potentiometer readings are transmitted from the Arduino to the Python interface via a USB connection. This enables real-time monitoring of sensor data, demonstrating how analog signals can be converted, transmitted, and visualized on a computer.

In the second part, the focus shifts to controlling an actuator—a servo motor—using Python commands. The user inputs desired angle values in Python, which are then sent to the Arduino to adjust the servo position accordingly. This demonstrates the use of serial communication for motion control and interactive hardware–software integration.

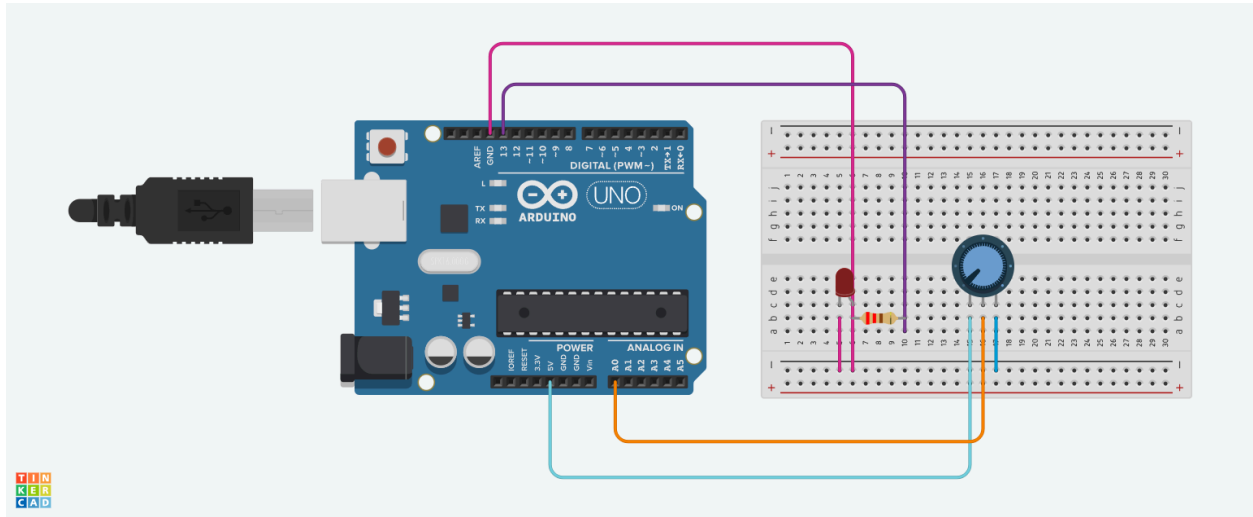
Through these experiments, we aim to gain a deeper understanding of how serial communication facilitates real-time data transmission and control in embedded systems, emphasizing the integration between sensors, actuators, and software platforms.

EXPERIMENT 3A: READING POTENTIOMETER VALUES VIA SERIAL COMMUNICATION

1.0 MATERIALS AND EQUIPMENTS

1. Arduino Board
2. Potentiometer
3. Jumper Wires
4. LED
5. 220 resistor
6. Breadboard

2.0 EXPERIMENTAL SETUP



1. The Arduino was connected to the computer via a USB cable.
2. The sketch was uploaded to the Arduino using the Arduino IDE.
3. The Python script was then executed on the computer.
4. As the potentiometer knob was turned, the potentiometer readings were displayed in the Python terminal.
5. These readings were used for various experiments, data logging, or control applications, depending on the project requirements.
6. The Serial Plotter was opened in the Arduino IDE by selecting Tools → Serial Plotter.
7. The correct COM port to which the Arduino was connected was selected.
8. The baud rate in the Serial Plotter was set to match the one used in the Arduino code (e.g., 9600).
9. As the potentiometer knob was turned, the Serial Plotter displayed the readings in real time, creating a graphical representation of the data and showing how the values changed with adjustment.
10. The Serial Plotter settings, such as graph range, labels, and colors, were customized as needed.
11. The circuit was then extended by adding an LED in series with a $220\ \Omega$ resistor, and the Python code was updated so that the LED turned ON automatically when the potentiometer value exceeded half of its maximum range, and OFF otherwise.

3.0 METHODOLOGY

3.1 Circuit Assembly

1. Connect an LED to pin 13 of the microcontroller.
2. Connect a potentiometer to analog pin A0 to read its variable resistance.
3. Ensure proper wiring:
 - One end of the potentiometer to VCC (5V)
 - The other end to GND
 - The middle terminal to A0

3.2 Programming Logic

1. Begin serial communication at 9600 baud rate to monitor values.
2. Set pin 13 as an output for the LED.
3. Reading potentiometer input;
 - Continuously read the analog value from pin A0
 - Convert the 0-1023 analog reading to a 0-255 range
4. Adjusting LED brightness;
 - Use pulse width modulation (PWM) to control LED brightness based on the mapped potentiometer value
 - Output the PWM signal to pin 13
5. Displaying data;
 - Print the brightness value to the Serial Monitor for real-time monitoring
6. Adding delay for smoothness
 - Introduce a 100ms delay to prevent rapid fluctuations and ensure smooth LED brightness transitions

3.3 Code Used

Arduino IDE code

```
#define LED_PIN 13      // PWM pin for LED
#define POT_PIN A0      // Analog pin for potentiometer

void setup() {
    Serial.begin(9600);    // Start Serial Monitor
    pinMode(LED_PIN, OUTPUT); // Set LED pin as output
}

void loop() {
    int potValue = analogRead(A0);
    int brightness = map(potValue, 0, 1023, 0, 255);

    analogWrite(13, brightness);

    Serial.println(brightness); // Only print the brightness value

    delay(100); // Short delay for smooth plotting
}
```

Python code

```
# task 1
import serial
import time

ser = serial.Serial('COM3', 9600)
time.sleep(2) # Give Arduino time to reset

try:
    while True:
        if ser.in_waiting > 0:
            pot_value = ser.readline().decode().strip()
            if pot_value.isdigit():
```



```
pot_value = int(pot_value)
print("Potentiometer Value:", pot_value)

if pot_value > 512:
    ser.write(b'ON\n')
else:
    ser.write(b'OFF\n')

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

3.4 Control Algorithm

1. Start initialization;
 - Define LED_PIN as 13 (PWM output for LED)
 - Define POT_PIN as A0 (analog input for potentiometer)
 - Initialize serial communication at 9600 baud for monitoring
 - Set LED_PIN as an output
2. Continuous loop execution;
 - Read the potentiometer value from POT_PIN (A0)
 - Map the analog input (0-1023) to a PWM range (0-255)
 - Write the mapped value to LED_PIN using analogWrite(), adjusting the LED brightness
 - Print the brightness value to the serial monitor
 - Introduce a 100ms delay for smooth transitions
3. Repeat indefinitely;
 - The loop continuously adjusts the LED brightness based on real-time potentiometer input

4.0 DATA COLLECTION

Data was collected to observe how the potentiometer readings changed when transmitted from the Arduino to Python through serial communication. The experiment also aimed to verify how the LED responded to these readings based on a set threshold. The components and instruments used for data acquisition included:

- Potentiometer: acted as an analog input sensor to provide variable voltage.
- LED: served as a visual indicator controlled by the potentiometer reading.
- Arduino: functioned as the microcontroller that read analog data and transmitted it to the computer.
- Python program (PySerial): received and displayed potentiometer readings in real time through the serial port.

The Arduino continuously reads the voltage from the potentiometer using the `analogRead()` function, producing analog-to-digital converter (ADC) values ranging from 0 to 1023. These values were then transmitted to the Python terminal using the `Serial.println()` function. As the potentiometer knob was rotated, the readings changed proportionally, showing how voltage input varied with rotation.

A threshold value of approximately 512 (half of 1023) was used in the Arduino program to control the LED. When the potentiometer reading exceeded 512, the LED turned ON; when below 512, it turned OFF.

Potentiometer Value (ADC)	Calculated Voltage (V)	Mapped Servo Angle (°)	LED Status
0	0.00	0°	OFF
256	1.25	45°	OFF
512	2.50	90°	ON
1023	5.00	180°	ON

The potentiometer readings were displayed in the Python terminal in real time. The results showed a smooth and linear change in the ADC values as the knob was rotated, confirming that the data transmission between the Arduino and Python was stable. The LED's response also matched the expected threshold behavior, turning ON only when the potentiometer output was above 512 (approximately 2.5 V).

This process demonstrated that analog sensor data could be successfully read, transmitted, and used for control applications through serial communication between the Arduino and Python.

5.0 DATA ANALYSIS

The objective of Experiment 3A was to transmit potentiometer readings from the Arduino to Python via serial communication and to control an LED based on those readings. The data collected consisted of analog-to-digital converter (ADC) values ranging from 0 to 1023, which represented the input voltage from the potentiometer. The ADC in the Arduino converts an input voltage between 0 V and 5 V into a 10-bit digital value using the formula:

$$V_{in} = \frac{ADC\ Value}{1023} \times 5V$$

For example, when the potentiometer reading was 512, the corresponding voltage at the analog pin was:

$$V_{in} = \frac{512}{1023} \times 5V = 2.50V$$

This means that when the potentiometer value was approximately 512 (half of the maximum range), the voltage output was around 2.5 V, which is half of the Arduino's reference voltage. This threshold value was used in the Arduino code to determine the LED's state. The LED turned ON when the potentiometer reading was greater than 512 (above 2.5 V), and OFF when below that value.

Statistically, the data showed a linear relationship between the potentiometer's rotational position and its corresponding ADC reading. As the potentiometer was turned clockwise, the readings increased steadily; when turned counterclockwise, they decreased proportionally. No major fluctuations or transmission delays were observed, indicating stable serial communication between the Arduino and Python interface.

The significance of these findings lies in demonstrating that analog sensor data can be reliably converted to digital signals, transmitted to a computer, and used for real-time control applications. This experiment successfully met its objective by showing how serial communication can integrate sensor input (potentiometer) and actuator output (LED) in a simple control system. The results also confirmed that the communication setup was effective, accurate, and consistent with expected electrical behavior.

6.0 RESULT

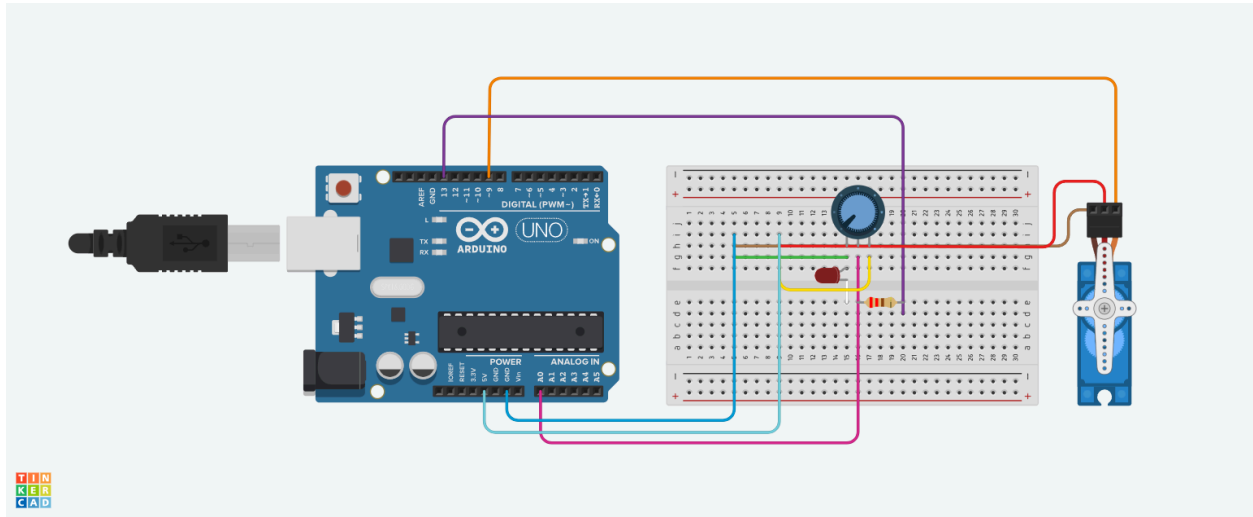
The successful execution of the experiment demonstrates that real-time data transmission from the Arduino to Python is achievable. The plotted graph accurately represents the potentiometer adjustments. The serial data transmission between the Arduino and Python remained stable throughout the experiment. The LED responded accurately by turning on when the potentiometer value exceeded half of its range and turning off otherwise. The real-time plot displayed consistent changes corresponding to the adjustment of the potentiometer knob. Thus, the results validate the effectiveness of the serial communication process.

EXPERIMENT 3B: TRANSMITTING ANGLE DATA FROM PYTHON SCRIPT TO AN ARDUINO, THEN ACTUATES SERVO TO MOVE TO THE SPECIFIED ANGLE

1.0 MATERIALS AND EQUIPMENTS

1. Arduino board (e.g., Arduino Mega 2560)
2. Servo motor
3. Jumper wires
4. Potentiometer (for manual angle input)
5. USB cable for Arduino
6. Computer with Arduino IDE and Python installed

2.0 EXPERIMENTAL SETUP



Circuit Connections:

1. The servo motor's signal pin was connected to digital pin 9 on the Arduino.
2. The Vcc pin of the servo motor was connected to 5V, and the GND pin was connected to GND on the Arduino.
3. The potentiometer was connected with one end to 5V, the other to GND, and the center (wiper) pin to analog input A0.
4. The LED was connected to digital pin 13 through a 220 Ω resistor, with its cathode connected to GND.
5. The Arduino was connected to the computer via a USB cable, providing both power and serial communication.

Arduino Setup:

1. The Arduino sketch was modified to read potentiometer values using the `analogRead()` function and to control the servo position using the `Servo` library.
2. The potentiometer readings were mapped to servo angles ranging from 0° to 180°.
3. The program also sent the servo position values to Python through serial communication for visualization purposes.

Python Setup:

1. A Python script was developed using the PySerial and Matplotlib libraries.
2. The script received the serial data from the Arduino, displayed the potentiometer and servo position values in real time, and plotted the readings using an interactive graph.
3. The serial port and baud rate were configured to match those used in the Arduino program (typically 9600 bps).

Operation:

1. When the Python script was executed, it established a serial connection with the Arduino.
2. As the potentiometer knob was rotated, the servo motor's position changed accordingly.
3. The LED provided visual feedback based on the potentiometer or servo angle.
4. The live graph in Python displayed the real-time potentiometer data, confirming successful two-way communication and visualization between the hardware and software components.

3.0 METHODOLOGY

3.1 Circuit Assembly

1. Connect the potentiometer to analog pin A0 to read its variable resistance;
 - One end of the potentiometer to VCC (5V)
 - The other end to GND
 - The middle terminal to A0
2. Connect the servo motor to digital pin 9
 - The signal wire to pin 9
 - The power wire to 5V
 - The ground wire to GND

3.2 Programming Logic

1. Attach the servo motor to pin 9 using `myServo.attach(servoPin)`.
2. Begin serial communication at 9600 baud rate to monitor values.
3. Reading potentiometer input;
 - Continuously read the analog value from pin A0
 - Convert the 0-1023 analog reading to a 0-180 degree range using `map()`
4. Controlling the servo motor;
 - Move the servo motor to the mapped angle using `myServo.write(angle)`
 - Print the potentiometer value and servo angle to the Serial Monitor for real-time tracking
5. Adding delay for stability;
 - Introduce a 10ms delay to ensure smooth movement without abrupt changes

3.3 Code Used

Arduino code

```
#include <Servo.h>

Servo myServo;
const int potPin = A0;
const int servoPin = 9;
const int ledPin = 13;

void setup() {
  Serial.begin(9600);
  myServo.attach(servoPin);
  pinMode(ledPin, OUTPUT);
}

void loop() {
  int potValue = analogRead(potPin);           // Read potentiometer
  (0-1023)
  int angle = map(potValue, 0, 1023, 0, 180);   // Map to servo angle
  myServo.write(angle);                        // Move servo

  Serial.println(potValue);                    // Send pot value to
Python
  // Check if there's serial input to control the LED
  if (Serial.available() > 0) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim(); // Remove whitespace/newlines

    if (cmd == "ON") {
      digitalWrite(ledPin, HIGH);
    } else if (cmd == "OFF") {
      digitalWrite(ledPin, LOW);
    }
  }
  delay(50);                                  // Small delay (~20 Hz
update rate)
}
```

Python code

```
import serial
import matplotlib
matplotlib.use('TkAgg') # Use interactive backend for PyCharm
import matplotlib.pyplot as plt
import time

# --- Serial connection ---
ser = serial.Serial('COM3', 9600) # Change COM port to match your
Arduino
time.sleep(2) # Wait for Arduino to reset

# --- Live plot setup ---
plt.ion()
fig, ax = plt.subplots()
x_vals, y_vals = [], []

try:
    while True:
        line = ser.readline().decode().strip()
        if line.isdigit(): # Make sure we got a number
            pot_value = int(line)
            print("Potentiometer Value:", pot_value)
            x_vals.append(len(x_vals))
            y_vals.append(pot_value)

        ax.clear()
        ax.plot(x_vals, y_vals, color='green', linewidth=2)
        ax.set_xlabel("Sample Number")
        ax.set_ylabel("Potentiometer Value (0-1023)")
```

```

        ax.set_title("Real-Time Potentiometer, Servo, and
LED Control")
        ax.grid(True)
        plt.pause(0.1)

        # Send LED control command to Arduino
        if pot_value > 500:
            ser.write(b'ON\n')    # Turn LED ON
        else:
            ser.write(b'OFF\n')  # Turn LED OFF

        time.sleep(0.021)  # Small delay to avoid flooding
        serial buffer

except KeyboardInterrupt:
    print("\nStopped by user.")
finally:
    ser.close()
    plt.ioff()
    plt.show()

    print("Serial connection closed.")

```

3.4 Control Algorithm

1. Initialize and setup;
 - Import necessary libraries (serial, matplotlib, time)
 - Configure Matplotlib to use an interactive display mode (TkAgg)
 - Establish serial communication with Arduino on COM3 at a baud rate of 9600
 - Wait for 2 seconds to allow Arduino to reset
 - Initialize variables for live plotting (x_vals, y_vals) and create a plot figure
2. Start continuous data reading loop;
 - Begin an infinite while loop to continuously read incoming data from the Arduino through the serial port
 - Read one line of data and decode it into a string
 - Check if the received line contains only digits (a valid potentiometer reading)
3. Process and display sensor data;
 - Convert the received string into an integer (pot_value)
 - Display the potentiometer value in the console
 - Append the new data point to the plotting lists (x_vals, y_vals).
 - Clear and update the plot in real time to show the latest sensor readings
4. Control led based on potentiometer value;
 - If the potentiometer value is greater than 500, send the command 'ON\n' to Arduino to turn the LED ON
 - Otherwise, send 'OFF\n' to turn the LED OFF
 - Add a short delay (0.021s) to prevent overloading the serial buffer

5. Exit and cleanup;

- When the user interrupts the program (e.g., by pressing Ctrl + C), stop the loop
- Close the serial connection safely
- Turn off interactive plotting and display the final graph window
- Print a confirmation message that the serial connection has been closed

4.0 DATA COLLECTION

Data was collected to study the relationship between the potentiometer input values and the servo motor angles using serial communication between the Arduino and Python. The main components used for data acquisition were:

- Potentiometer: acted as the analog sensor for position control.
- Servo motor: functioned as the actuator responding to the potentiometer input.
- LED: provided visual feedback by turning ON when the servo angle exceeded half its range.
- Arduino: served as the microcontroller to read analog input and control the servo.
- Python program (PySerial + Matplotlib): used for data collection, visualization, and real-time plotting.

The Arduino continuously read the potentiometer's analog signal using the `analogRead()` function, which produced digital values between 0 and 1023. These readings were mapped to servo motor angles ranging from 0° to 180°, and the corresponding data was transmitted to Python via the serial port. The Python script received these values using the *PySerial* library and displayed them in real time. The *Matplotlib* library plotted a live graph showing how the servo angle changed as the potentiometer was rotated.

Potentiometer Value (ADC)	Calculated Voltage (V)	Mapped Servo Angle (°)	LED Status
0	0.00	0°	OFF
256	1.25	45°	OFF
512	2.50	90°	ON
1023	5.00	180°	ON

The real-time plot generated by Python showed a smooth, linear trend between the potentiometer readings and the servo angles. As the potentiometer was turned clockwise, both the voltage and servo angle increased proportionally. The LED lit up when the potentiometer value exceeded approximately 512 (corresponding to about 2.5 V or 90°), indicating the system's response to the changing input.

This data collection process clearly demonstrated that the Arduino successfully read sensor input, transmitted data to Python for monitoring, and controlled the servo motor in real time. The Python visualization confirmed the stability of serial communication and the accuracy of the actuator's response to the sensor data.

5.0 DATA ANALYSIS

The objective of Experiment 3B was to control a servo motor using potentiometer input through serial communication between Arduino and Python, and to visualize the potentiometer data in real time. The potentiometer acted as an analog input device, producing values between 0 and 1023, which were converted into corresponding servo angles ranging from 0° to 180° using the Arduino `map()` function. The mapping relationship can be expressed as:

$$\text{Servo Angle} = \frac{\text{Potentiometer Value}}{1023} \times 180^\circ$$

For example:

- When the potentiometer value was 0, the servo angle was 0°.
 - When the potentiometer value was 512, the servo angle was approximately:

$$\frac{512}{1023} \times 180^\circ = 90^\circ$$

- When the potentiometer value was 1023, the servo angle was 180°.

This showed a direct linear relationship between the potentiometer's position and the servo's rotation angle. As the potentiometer knob was turned clockwise, the ADC reading increased, causing the servo to rotate toward the maximum angle. Turning it counterclockwise decreased the reading, moving the servo back toward 0°.

The Python program received the potentiometer data from the Arduino through serial communication and plotted the readings in real time using the *Matplotlib* library. The plotted data formed a smooth and continuous curve, confirming that the communication was stable and that the servo responded immediately to changes in input. The LED indicator provided an

additional visual cue and it turned ON when the potentiometer value (and corresponding servo angle) exceeded half the range (around 512 or 90°), and OFF when below this point.

Statistical observation of the plotted data indicated consistent readings with minimal fluctuations, showing that both the sensor input and servo response were reliable. Any minor variation in data could be attributed to electrical noise or slight delays in serial communication.

The significance of this data is that it confirms successful real-time control of an actuator (servo motor) through serial communication, using sensor feedback and Python visualization. The experiment demonstrated how analog signals from a sensor can be converted into digital data, transmitted, processed, and used to control a mechanical output. This fulfills the experiment's objectives by integrating hardware (Arduino and servo motor) with software (Python) to create a complete and interactive mechatronic control system.

6.0 RESULT

The servo motor successfully responded to control signals, rotating smoothly to the intended angles with stable and consistent motion. It performed reliably without sudden jumps or irregular movements, and the response time was fast, allowing the motor to reach target angles accurately. The real-time plot displayed smooth and repetitive waveforms that matched the potentiometer adjustments, confirming proper system performance. The LED indicator functioned as expected by turning on when the potentiometer value or servo position exceeded the threshold. Overall, the serial communication between the Arduino and Python remained stable, ensuring accurate and continuous data exchange.

DISCUSSION

The experiment successfully demonstrated the expected relationship between the potentiometer input and both the LED brightness and servo motor movement. Adjusting the potentiometer produced proportional changes in output; however, minor discrepancies were observed, such as flickering in the LED, slight delays in the servo's response, and a non-linear overall behavior.

Sources of Error and Limitations:

- **Electrical Noise:** Fluctuations in ADC readings caused variations in LED brightness and affected servo stability.
- **Hardware Tolerances:** Differences in potentiometer resistance and servo accuracy led to small deviations in output.
- **Non-Linear Response:** Human perception of brightness and servo motion was not perfectly proportional to the input signal.
- **Limited Range:** The potentiometer did not always utilize its full input range effectively.

Despite these limitations, the experiment successfully demonstrated real-time control using PWM signals and validated the effectiveness of serial communication between Arduino and Python.

CONCLUSION

This experiment successfully demonstrated two key applications of serial communication: data transmission from a potentiometer and control of an LED and a servo motor via Python. The results confirmed that serial communication can efficiently transmit real-time data. In this setup, the Arduino read analog signals from the potentiometer and sent the data to Python through a USB connection.

Furthermore, in Experiments 3A and 3B, it was validated that the potentiometer could control the LED brightness by adjusting resistance, which in turn influenced the LED intensity and the servo motor's movement. This demonstrates that serial communication not only transfers data but also transmits commands capable of controlling physical devices.

These principles are fundamental in industrial automation, IoT-based smart systems, and real-time monitoring applications. Mastering serial communication provides a strong foundation for designing advanced control systems where sensors, actuators, and microcontrollers must interact seamlessly. This knowledge is especially valuable for engineers and developers involved in automated processes, robotics, and remote data acquisition systems.

RECOMMENDATIONS

The experiment can be improved by implementing real-time graphical visualization for both potentiometer readings and servo movements. This would make the system more user-friendly by allowing easier control through the computer instead of manual adjustments. It can also reduce the need for complex physical components and make the setup more efficient. In addition, the accuracy and responsiveness of the data can be improved by applying a moving average filter to stabilize the potentiometer readings. Since the readings tend to fluctuate, this filtering technique would help smooth the data before plotting it on the graph, resulting in more reliable and consistent measurements.

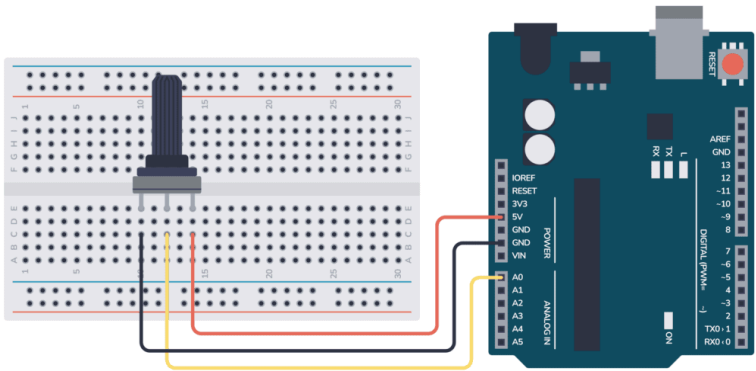
REFERENCES

Last Minute Engineers. (2019, October 20). *How Servo Motor Works & Interface It With Arduino*. Last Minute Engineers. <https://lastminuteengineers.com/servo-motor-arduino-tutorial/>

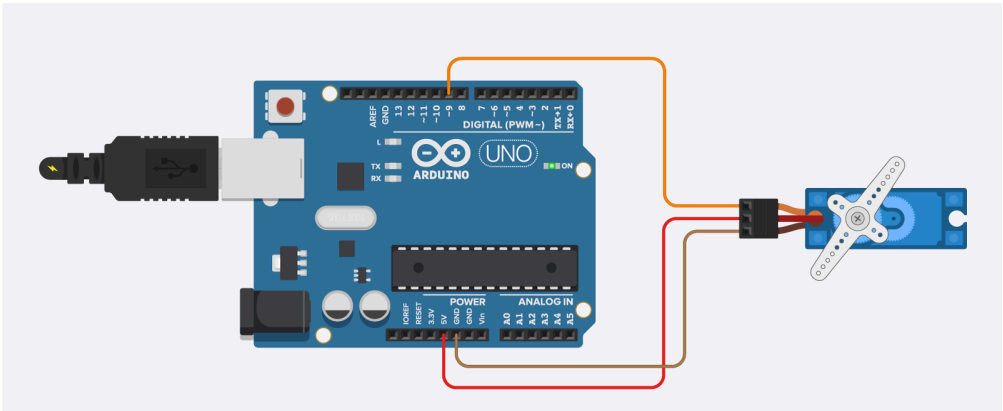
Instructables. (2015, July 10). *Arduino Servo Motors*. Instructables; Instructables. <https://www.instructables.com/Arduino-Servo-Motors/>

MertArduinoYouTube. (n.d.). *Arduino : How to Control Servo Motor With Potentiometer*. Instructables. <https://www.instructables.com/Arduino-How-to-Control-Servo-Motor-With-Potentiome/>

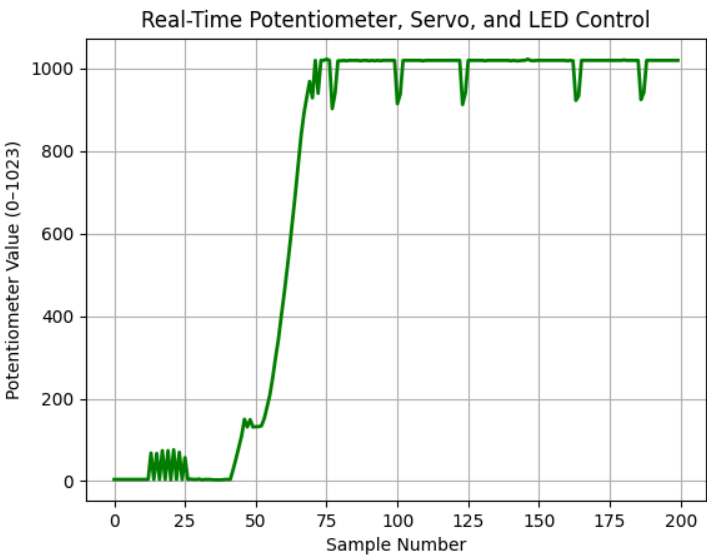
APPENDICES

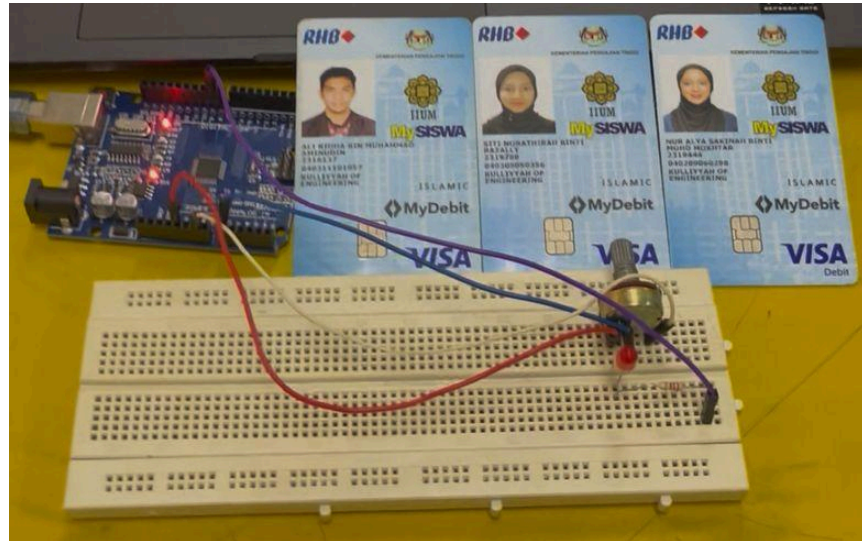


Wiring diagram for potentiometer

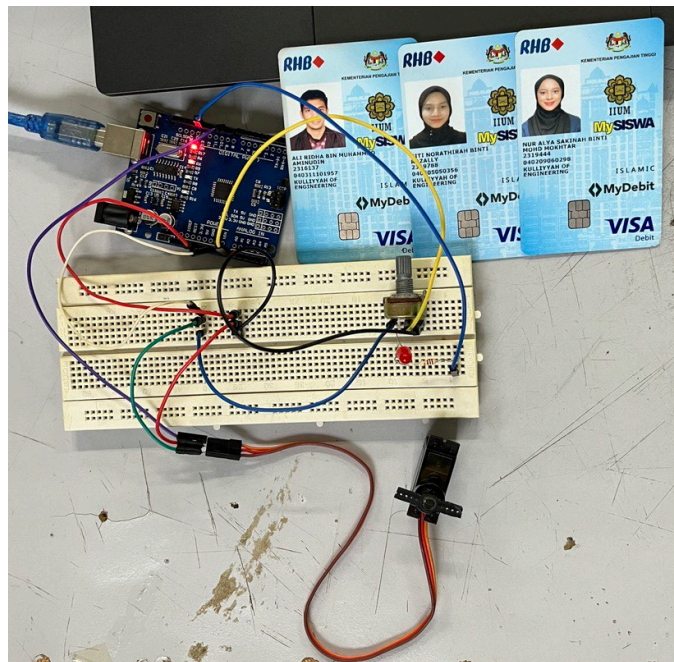


Wiring diagram for servo motor





Task 1




Task 2

ACKNOWLEDGEMENTS

Special thanks to Wahju Sediono for their guidance and support during this experiment. We would also like to extend our gratitude to our instructors, peers, and the laboratory staff for their valuable insights and assistance throughout the experiment. Their feedback and encouragement have greatly contributed to the success of this project. Finally, we appreciate the resources and learning materials provided, which enabled us to understand and implement serial communication effectively.

Certificate of Originality and Authenticity

We hereby certify that we are responsible for the work presented in this report. The content is our original work, except where proper references and acknowledgements are made. We confirm that no part of this report has been completed by anyone not listed as a contributor. We also certify that this report is the result of group collaboration and not the effort of a single individual. The level of contribution by each member is stated in this certificate. Furthermore, we have read and understood the entire report, and we agree that no further revisions are required. We collectively approve this final report for submission and confirm that it has been reviewed and verified by all group members.

Signature: 
Name: ALI RIDHA BIN MUHAMMAD AMINUDIN
Matric Number: 2316137

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

Signature: 
Name: SITI NORATHIRAH BINTI RAZALLY
Matric Number: 2319788

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

Signature: 
Name: NUR ALYA SAKINAH BINTI MOHD MOKHTAR
Matric Number: 2319444

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