KULLIYYAH OF ENGINEERING (KOE)

MECHATRONICS SYSTEM INTEGRATION (MCTA3203)

SEMESTER 1, 24/25

SECTION 1

PROJECT REPORT WEEK 3

TITLE : SERIAL COMMUNICATION

PREPARED BY :

NO	NAME	MATRIC NO
1	ARIF EMRULLAH BIN TAJUL ARIFFIN	2215359
2	AZLIYANA SYAHIRAH BINTI AZAHARI	2210620
3	DAMIA MAISARAH BINTI ZAWAWI	2217830
4	HUDA BINTI AB RAHMAN AL-QARI	2226676
5	IKMAL HAKIM BIN ZAKI	2125625

DATE OF SUBMISSION : 30TH OCTOBER 2024

ABSTRACT

This experiment shows how to create a serial communication link between an Arduino microcontroller and Python software so that potentiometer readings may be transmitted in real-time. The Arduino is designed to send data over a USB connection indefinitely, while a Python script creates a serial interface with the pyserial package to accept this information. The findings prove stable, real-time communication and provide a proof of concept for serial data transfer between Arduino and PCs. This arrangement efficiently allows real-time data utilization in Python programs, laying the groundwork for future projects that combine microcontroller data with software-based systems.

To sum up, this experiment demonstrated how to successfully establish a serial communication link between an Arduino and Python to exchange data, specifically potentiometer readings. When both ends are linked, this communication setup is ideal for using real-time data in Python programs. The findings of this study set the groundwork for future applications and initiatives by providing a fundamental understanding of serial communication between microcontrollers and computers.

TABLE OF CONTENTS

INTRODUCTION

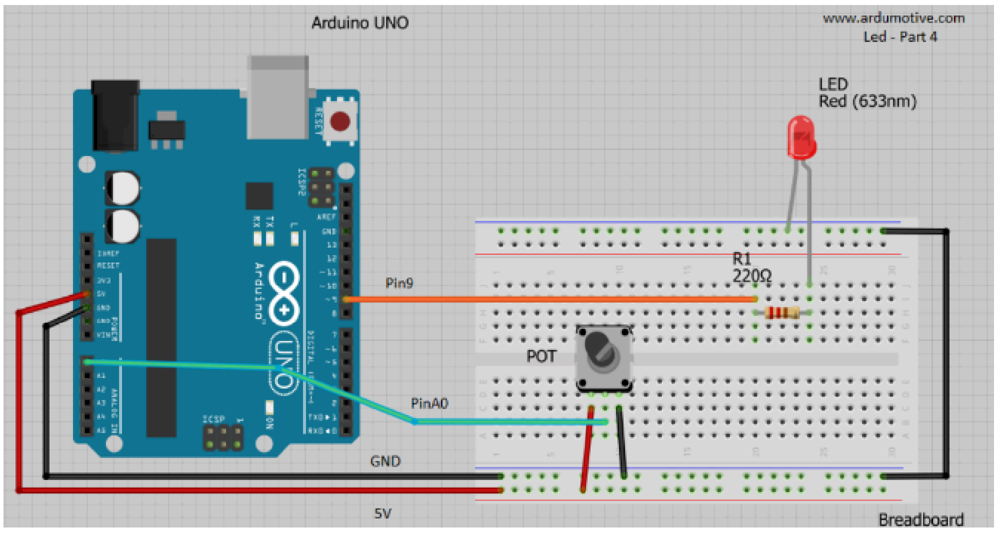
These experiments explore serial communication and control between Python and an Arduino. In Experiment 3a, we set up serial communication so that a potentiometer reading from an Arduino can be sent to a Python script over USB. This introduces how data can be shared between a computer and a microcontroller. Experiment 3b builds on this by using Python to control a servo motor connected to Arduino. Here, Python sends angle data to Arduino, which then moves the servo to the specified position. Together, these experiments lay the groundwork for basic interfacing and control applications.

PART A

MATERIALS AND EQUIPMENTS

1. Arduino Mega 2560 board
2. LED
3. Resistors 230 ohm
4. Potentiometers
5. Jumper wires
6. Breadboard

EXPERIMENTS SETUP



METHODOLOGY

1. The potentiometer was connected to Arduino Mega 2560 in following setup:
   - The VCC of the potentiometer was connected 5V pin of the Arduino Mega 2560
   - The output of the potentiometer was connected to analog pin 0 due to its value varies from 0 to 1k ohm
   - The ground of the potentiometer was connected to GND pin of the Arduino Mega 2560
2. The LED was connected to Arduino Mega 2560 in following order:
   - The cathode of the LED was connected to pin9 along with a resistor of 230 ohm to regulate the current to Arduino Mega 2560.
   - The anode of the LED was connected to a GND pin.

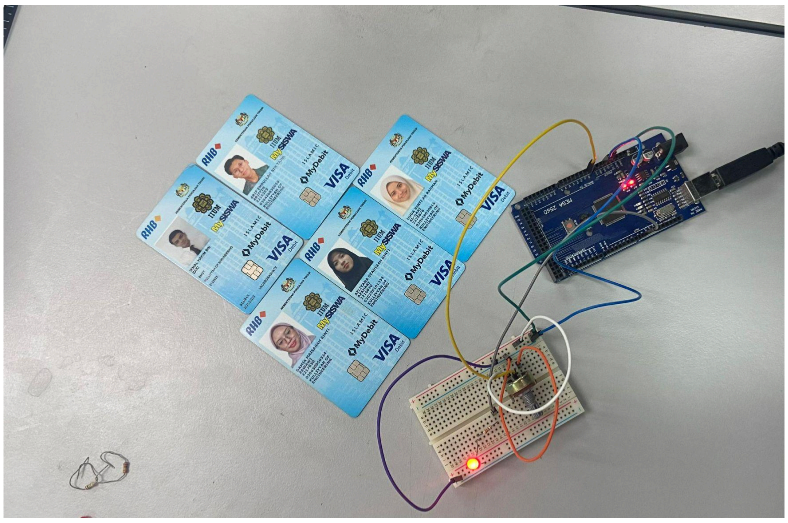
CODING

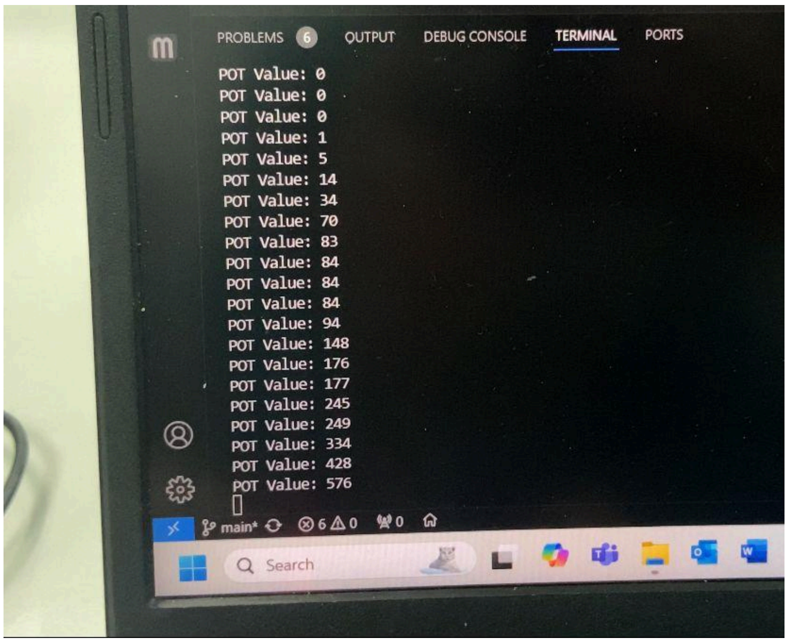
```
int potPin = A0;   // select the input pin for the potentiometer
int ledPin = 9;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from
the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(potPin);
  Serial.println(sensorValue);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

PROCEDURES

1. Build the circuit according to the circuit setup instructions.
2. Upload the provided Arduino code to Arduino Mega 2560.
3. Open the Serial Monitor in the Arduino IDE.
4. Rotate the tip of the potentiometer. As the potentiometer rotates the value increases and decreases, the led should blink slower and faster depending on the sensorValue.

RESULTS





```
POT Value: 0
POT Value: 0
POT Value: 0
POT Value: 1
POT Value: 5
POT Value: 14
POT Value: 34
POT Value: 70
POT Value: 83
POT Value: 84
POT Value: 84
POT Value: 84
POT Value: 94
POT Value: 148
POT Value: 176
POT Value: 177
POT Value: 245
POT Value: 249
POT Value: 334
POT Value: 428
POT Value: 576
```

We can see when the potentiometer has been rotated the output can be read from our PC by using Python. We can control the value of output by using the potentiometer by turning it up or down. Therefore, if the output value reach higher value, the LED blinked slower while lower value makes the LED blinked faster.

DISCUSSIONS

- **Software**

The code uploaded from Arduino IDE functions to control the blinking rate of the LED by adjusting the potentiometer value. A Python script is then used to effectively establish a serial connection using the pyserial library, capturing potentiometer readings from the Arduino using a baud rate of 9600.

- **Electrical**

The potentiometer's output was directly linked to the LED blink rate, showing that higher potentiometer values resulted in a faster blinking LED. Voltage fluctuations in the potentiometer readings caused slight variations in the LED's blink speed.

- **Hardware**

Turning the potentiometer knob in a consistent motion proved challenging, as reflected in the fluctuating slopes of the real-time graph. The graph exhibited an upward slope when turned clockwise and a downward slope when turned counterclockwise, though this was not perfectly linear due to intermittent stopping. A more refined potentiometer adjustment system would allow for smoother control, improving the accuracy and quality of the data visualized.

Questions:

Run your Python script, and it should display the potentiometer values as you turn the potentiometer knob. Make sure that you've selected the correct baud rate in both the Arduino code and the Python script. If your Arduino uses a different baud rate, update it in both places. This setup allows you to transmit the potentiometer readings from your Arduino to your Python script. You can then process or visualize this data in your Python application as needed.

Answers :

After running the Python script, we observed real-time potentiometer values displayed as we rotated the knob, confirming the functionality of the serial communication. Using a simple line graph with the matplotlib library, we illustrated how the potentiometer's position directly influenced its readings, demonstrating how real-time adjustments could be leveraged in applications requiring live data feedback.

```python
import serial
import matplotlib.pyplot as plt
import numpy as np
import time

ser = serial.Serial('COM3', 9600)

xpoints = []
ypoints = []

try:
   while True:
       pot_value = ser.readline().decode().strip()

       try:
           pot_value = int(pot_value) #conver to integer value
           print("POT Value:", pot_value)

           xpoints.append(time.time())
           ypoints.append(pot_value)

           if len(xpoints) > 100:
               xpoints.pop(0)
               ypoints.pop(0)
           # plotting
           plt.clf()
           plt.plot(xpoints, ypoints)
           plt.xlabel('Time (s)')
           plt.ylabel('Potentiometer Value')
           plt.title('Real-time Potentiometer Value')
           plt.pause(0.1)
       except ValueError:
           print('Invalid pot value received')

except KeyboardInterrupt:
   ser.close()
   print("Serial connection closed.")
except UnicodeDecodeError:
   print('Serial Error\n')
finally:
   ser.close()
   print("Serial connection closed.")
```

CONCLUSION

In these experiments, we successfully established serial communication and control between Python and Arduino. In Experiment 3a, we sent potentiometer readings from Arduino to Python, demonstrating simple data exchange over USB.

RECOMMENDATIONS

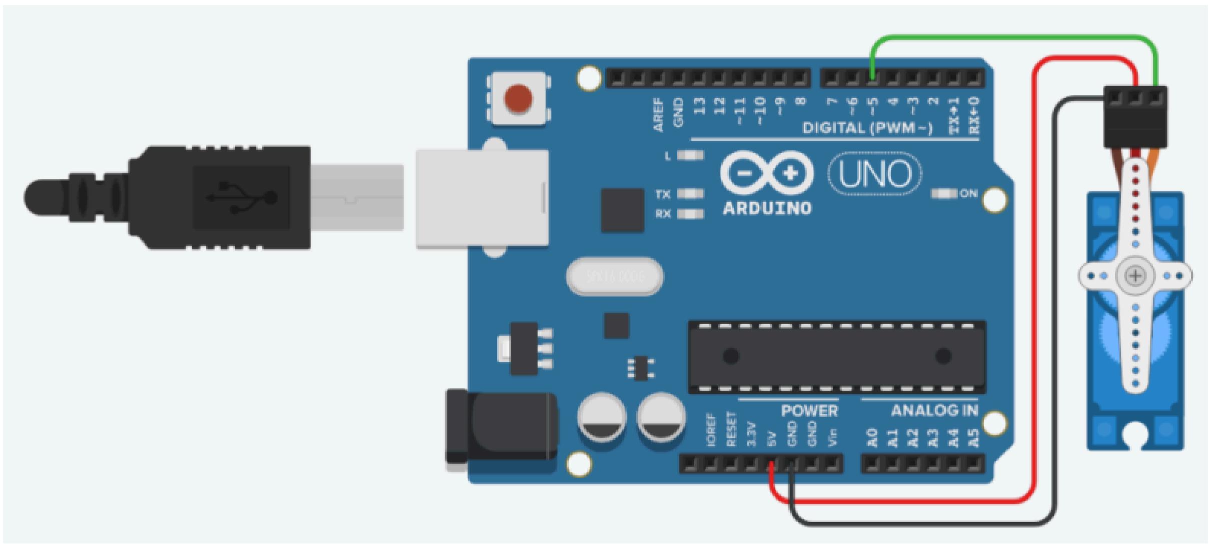
Future iterations of the experiment could incorporate more sensors and actuators to develop more complicated interactive systems, such as temperature or light sensors that modify LED brightness based on ambient conditions. Using numerous LEDs with different colours or intensity levels can broaden the experiment's scope, allowing for colour mixing and dynamic lighting effects. Incorporating wireless communication modules, such as Bluetooth or Wi-Fi, allows for remote control of the LED system and interaction with smartphones and IoT apps.

PART B

MATERIALS AND EQUIPMENTS

1. Arduino board (e.g., Arduino Uno)
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

EXPERIMENTS SETUP



METHODOLOGY

1. Servo motor was connected to Arduio Mega 2560 in following setup:
   a. Connect the servo's signal (orange) wire to digital pin 5 on the Arduino:
   b. Connect the servo's power (red) and ground (brown) wires to the 5V output and ground (GND) pins on the Arduino respectively.

CODING

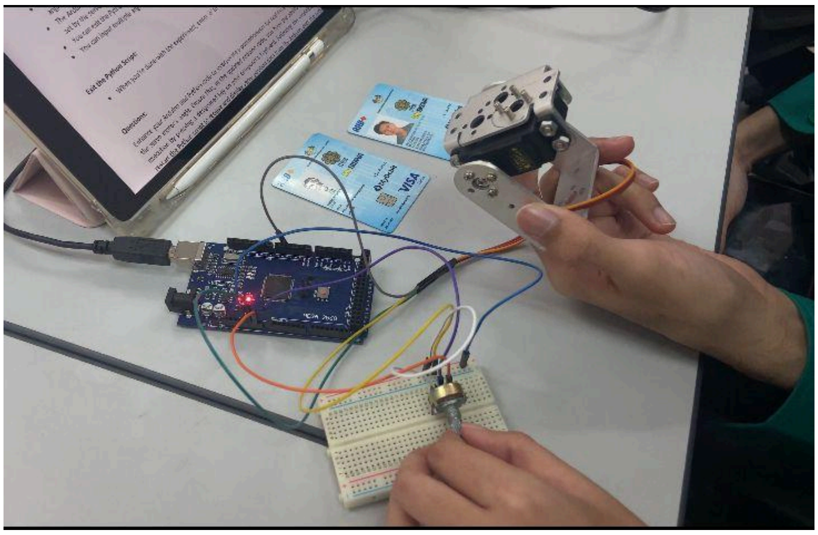
```
#include <Servo.h>

Servo myservo;
int potPin = A0;
int pos = 0;
char incomingByte;

void setup() {
  myservo.attach(5);
  Serial.begin(9600);
}

void loop() {

  if (Serial.available() > 0) {
    incomingByte = Serial.read();
    if (incomingByte == 'q') {
      while (true);
    }
  }

  int potValue = analogRead(potPin);
  pos = map(potValue, 0, 1023, 0, 180);
  myservo.write(pos);
  Serial.println(pos);

  delay(100);
}
```

PROCEDURES

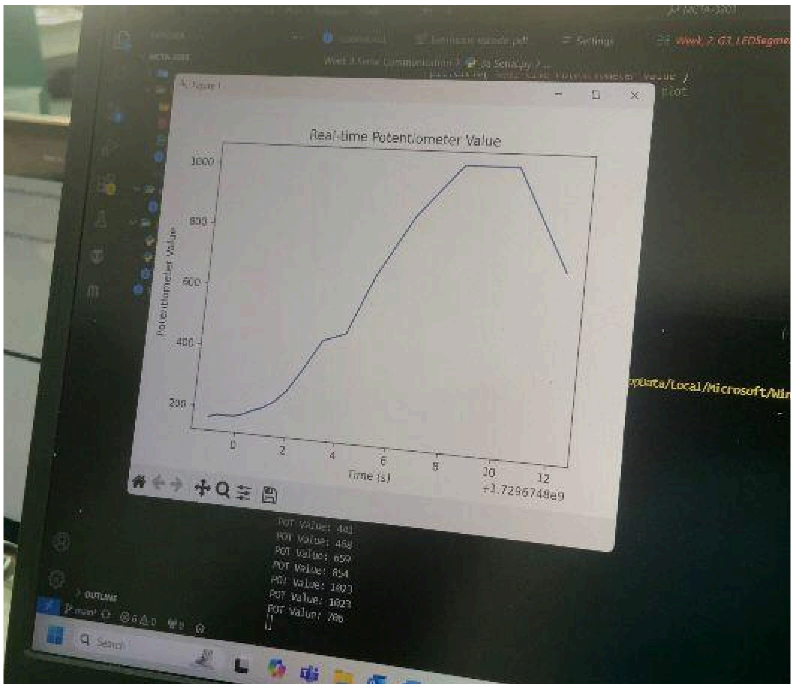
1) Build the circuit according to the circuit setup instructions.
2) Upload the provided Arduino code to Arduino Mega 2560.
3) Open the Serial Monitor in the Arduino IDE.

4) Rotate the tip of the potentiometer. As the potentiometer rotates the value increases and decreases, the servos should rotate on an angle between 0 and 180, depending on the “pos”.

RESULTS



In this experiment , we can see how to control the servo by using python.We have set it for the servo to rotate 0 to 180 degree. By writing the value of degrees in the python , we can observe the servo rotated according to the value of degrees that we have written in the python. Other than that if we press ‘q’ in the input the coding will be reset.



```
POT Value: 481
POT Value: 469
POT Value: 659
POT Value: 854
POT Value: 1023
POT Value: 1023
POT Value: 786
```

A real-time graph was constructed using the potentiometer values, and as shown in the video included in the week 3 file on github, the graph exhibited an upward slope when the knob was turned anticlockwise, but a downward incline when turned clockwise. The slope is not linear because the turning motion was not smooth and stopped intermittently. The second challenge was controlling the servo motor with a potentiometer. So, anytime the knob was turned, the servo continued to turn in the same direction, as seen clearly in the other video supplied.

DISCUSSIONS

- **Software**

The use of Python to control the servo motor was successful, with angle values sent over serial communication to the Arduino, which then positioned the servo accordingly. The addition of a key press in the Arduino script to halt operation added a convenient troubleshooting mechanism, as it allowed for easy program reset and interruption of the servo control if necessary. Moving forward, integrating a graphical user interface (GUI) in Python could make angle adjustments more intuitive and user-friendly.

- **Electrical**

The servo motor operated reliably within the 0-180 degree range, adjusting according to potentiometer input. However, when the potentiometer was rotated quickly, minor delays in servo response were observed. This could be mitigated by using a motor driver or refining the power supply, enhancing response times and alignment with real-time data.

- **Hardware**

Physical control of the servo motor using the potentiometer yielded predictable movements but revealed a few limitations. Due to quick or abrupt turns of the potentiometer, the servo occasionally struggled to match the real-time angle adjustments instantly. Replacing the current potentiometer with one that offers finer rotational control may allow smoother servo adjustments and improve system responsiveness.

Questions:

Enhance your Arduino and Python code to incorporate a potentiometer for real-time adjustments of the servo motor's angle. Ensure that, in the updated Arduino code, you have the ability to halt its execution by pressing a designated key on your computer's keyboard. Following the modification, restart the Python script to receive and display servo position data from the Arduino over the serial connection. While experimenting with the potentiometer, observe the corresponding changes in the servo motor's position.

Answers :

The potentiometer's position is continuously read and mapped to a servo angle, which is then sent to the Arduino, allowing live servo movement in response to the potentiometer's position. In the updated Python code, we added an option to halt execution by pressing 'q', which sends a command to the Arduino to stop. After any modifications, restarting the Python script re-establishes the connection and resumes the servo's real-time position feedback based on the potentiometer adjustments. This setup allowed for smooth control and precise servo positioning as verified by observing real-time data updates displayed in the Python terminal.

```
import serial
import time

ser = serial.Serial('COM3', 9600)
time.sleep(2)

try:
   while True:
       angle = input("Press Enter to read servo angle or 'q' to quit:
")
       if angle.lower() == 'q':
           ser.write(b'q')
           break
       if ser.in_waiting > 0:
           position = ser.readline().decode('utf-8').strip()
           print(f"Current Servo Angle: {position} degrees")
except KeyboardInterrupt:
   pass
finally:
    ser.close()
print("Serial connection closed.")
```

CONCLUSION

In Experiment 3b, we used Python to control a servo motor through Arduino, showing how serial communication can be applied to actuator control. These results matched our expectations and strengthened our understanding of basic data transfer and control techniques.

RECOMMENDATIONS

Replacing the potentiometer with one that has finer control or higher precision could result in smoother adjustments, minimizing the abrupt changes that impacted the servo's immediate responsiveness. Implementing a graphical user interface (GUI) in Python would allow for more intuitive control over the servo angle, making the setup more user-friendly and accessible for future applications. Lastly, testing with a variety of potentiometers and servo motors may help in identifying the optimal hardware combination for this setup, ensuring reliable performance in diverse scenarios.

ACKNOWLEDGEMENT

We would like to express our gratitude to Dr. Wahju Sediono, Dr. Ali Sophian, Dr. Zulkifli Bin Zainal Abidin, for providing the necessary resources and facilities to conduct this and support throughout the duration of the project. Additionally, we extend our appreciation to all individuals who contributed to the success of this lab report through their valuable insights and feedback.

STUDENT’S DECLARATION

Certificate of Originality and Authenticity

This is to certify that we are responsible for the work submitted in this report, that **the original work** is our own except as specified in the references and acknowledgement, and that the original work contained herein have not been untaken or done by unspecified sources or persons.

We hereby certify that this report has **not been done by only one individual** and **all of us have contributed to the report**. The length of contribution to the reports by each individual is noted within this certificate.

We also hereby certify that we have **read** and **understand** the content of the total report and that no further improvement on the reports is needed from any of the individual contributors to the report.

Signature: [illegible]	**Read**	/
Name: Arif Emrullah Bin Tajul Arifin	**Understand**	/
Matric No: 2215359	**Agree**	/

Signature: [illegible]	**Read**	/
Name: Azliyana Syahirah Binti Azahari	**Understand**	/

Matric No: 2210620	**Agree**	/

Signature: [illegible]	**Read**	/
Name: Damia Maisarah Binti Zawawi	**Understand**	/
Matric No: 2217830	**Agree**	/

Signature: [illegible]	**Read**	/
Name: Ikmal Hakim Bin Zaki	**Understand**	/
Matric No: 2125625	**Agree**	/

Signature: Huda	**Read**	/
Name: Huda binti Ab Rahman Al-Qari	**Understand**	/
Matric No: 2226676	**Agree**	/