LABORATORY REPORT

MECHATRONICS SYSTEM INTEGRATION

MCTA 3203

SEMESTER 2 2023/2024

WEEK 3

LAB: SERIAL COMMUNICATION

SUBMISSION DATE:

GROUP: 6

	NAME	MATRIC NUMBER
1.	MUHAMMAD AZIM ZUFAYRI BIN NORAZAM	2211973
2.	MUHAMMAD AMMAR FARIS BIN ARAFAS	2212645
3.	AHMAD ADAM BIN NORHELME	2219973
4.	AHMAD HAFIZULLAH BIN IBRAHIM	2216185

Table of Contents

EXPERIMENT 3A

Abstract

Serial communication between a Python script and an Arduino microcontroller is a widely used method for exchanging data between a computer and embedded systems. This experiment aims to transmit potentiometer readings from an Arduino to a Python script via a USB connection. To achieve this, code is implemented on both the Python and Arduino platforms to establish reliable serial communication and facilitate data exchange.

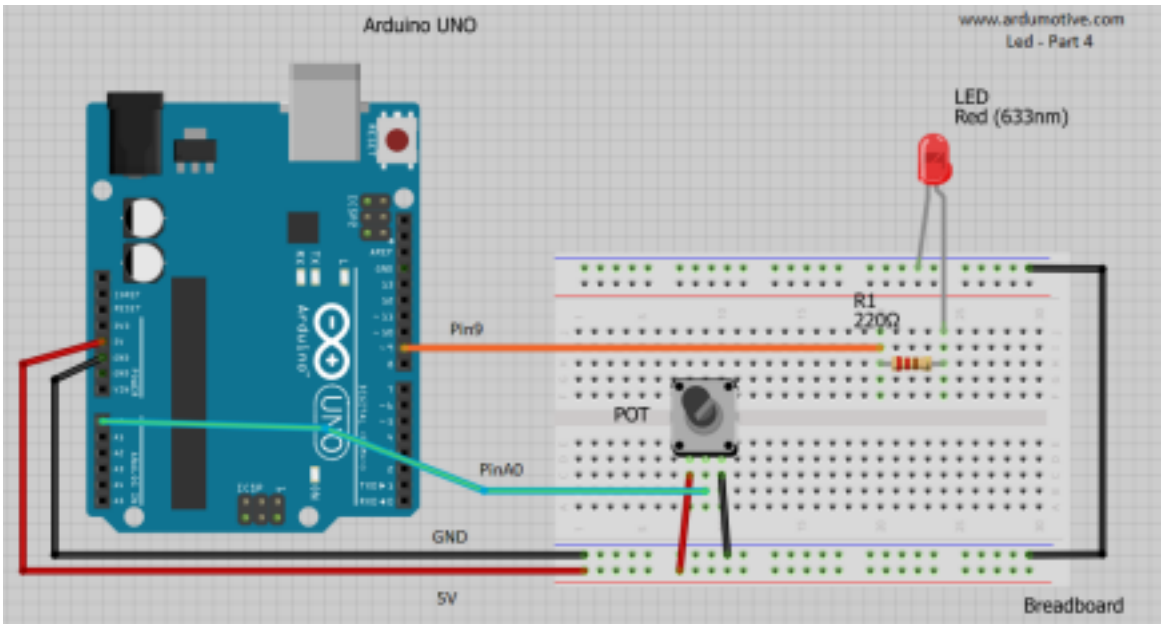
Introduction

Serial communication serves as a fundamental method for exchanging data between a computer and a microcontroller, such as an Arduino. By implementing code on both the Python and Arduino platforms, reliable serial communication can be established, enabling seamless data transmission. This experiment focuses on transmitting potentiometer readings from an Arduino to a Python script via a USB connection. Through this process, the practical application of serial communication in real-time data exchange is demonstrated, highlighting its significance in various embedded systems and IoT applications.

Materials and Equipment

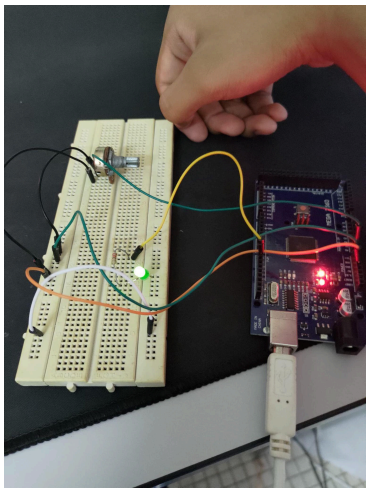
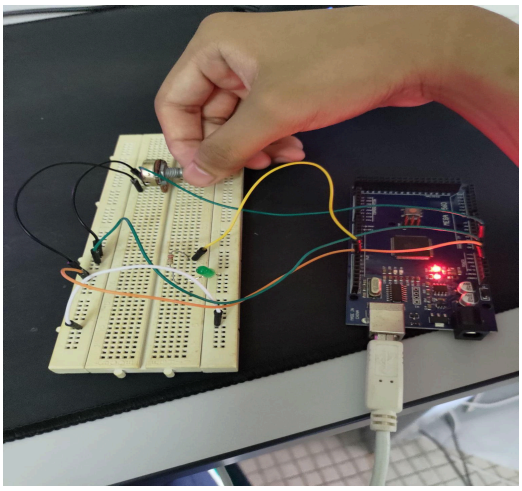
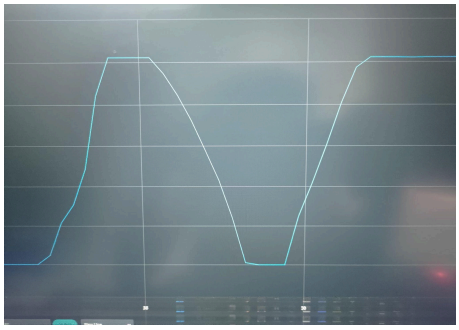
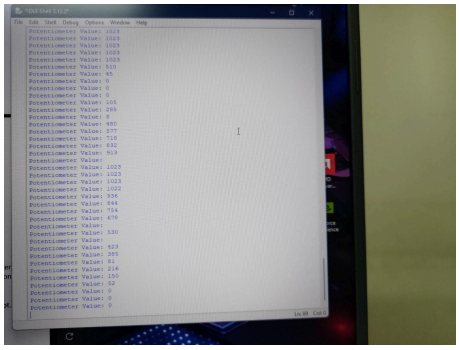
- Arduino Board
- Potentiometer
- Jumper Wires
- LED
- 220 resistor
- Breadboard

Methodology



1. Connect the Arduino to your computer via a USB cable.
2. Power on the Arduino (upload the sketch to your Arduino using the Arduino IDE)
3. Run the Python script on your computer.
4. As you turn the potentiometer knob, you should see the potentiometer readings displayed in the Python terminal.
5. You can use these readings for various experiments, data logging, or control applications, depending on your project requirements.
6. Open the Serial Plotter: In the Arduino IDE, go to "Tools" -> "Serial Plotter." 7. Select the Correct COM Port: In the Serial Plotter, select the correct COM port to which your Arduino is connected.
7. Set the Baud Rate: Ensure that the baud rate in the Serial Plotter matches the one set in your Arduino code (e.g., 9600).
8. Read Real-Time Data: As you turn the potentiometer knob, the Serial Plotter will display the potentiometer readings in real-time, creating a graphical representation of the data. You can see how the values change as you adjust the potentiometer.
9. Customize the Plotter: You can customize the Serial Plotter by adjusting settings such as the graph range, labels, and colors.

Results

Question

To present potentiometer readings graphically in your Python script, you may enhance your code by introducing the capability to generate and showcase a graph. This graphical visualization can deliver a more intuitive and informative perspective for data interpretation. Be sure to showcase the steps involved in your work (Hint: use matplotlib in your Python script).:

```
import serial
import matplotlib.pyplot as plt

# Open serial port (replace 'COMX' with your Arduino's port)
ser = serial.Serial('COMX', 9600)

# Initialize empty lists to store data
```

```
time_points = []
pot_values = []

# Set up the plot
plt.ion()  # Turn on interactive mode
fig, ax = plt.subplots()
line, = ax.plot(time_points, pot_values)

ax.legend()
ax.set_xlabel('Time')
ax.set_ylabel('Potentiometer Value')
ax.set_title('Real-time Potentiometer Readings')
plt.grid(True)

try:
    while True:
        # Read potentiometer value from Arduino
        data = ser.readline().decode().strip()
        if data:
            pot_val = int(data)
            print("Potentiometer Value:", pot_val)

            # Append data to lists
            pot_values.append(pot_val)
            time_points.append(len(pot_values))  # Use index as x-axis (time)

            # Update plot
            line.set_xdata(time_points)
            line.set_ydata(pot_values)
            ax.relim()
            ax.autoscale_view()
            fig.canvas.draw()
            fig.canvas.flush_events()

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

#This python code uses matplotlib module to display the data in a graphical way which is easier to understand.Python takes the values of the potentiometer from Arduino IDE and converts them

to the more understandable graphical form. It takes the time as the x-axis and the potentiometer values as the y-axis.Using the ‘line’ variable as an object, we create a subplot of 1 row and 1 column. The graph will show the updated version for every data cycle that Arduino sends to Python. This demonstrates the importance of data visualization in analysis and highlights the ease of using Python in this aspect.

Discussion

1. **Hardware**

This setup of hardware requires Arduino board, an LED, a potentiometer , a resistor(220Ω), and several jumpers. The anode part of the LED is connected to a digital pin(in our case Digital Pin 9) of the Arduino through the resistor.The resistor functions as a current limiter to reduce the risk of LED from any damage due to overcharging.The cathode part of the LED is connected to the ground. For the potentiometer, the middle leg(wiper) is the analog input pin for Arduino while the other two legs are for Ground and 5V. The potentiometer will give the information of the voltage variation to the Arduino by varying the resistance value of the potentiometer itself. The potentiometer utilizes the arrangement of the voltage divider, whereas the voltage of the middle leg(wiper) will be varying based on the positioning of the legs. When turning the knob, we should see the value either increasing or decreasing in the interface. Meanwhile, the LED acts as a visual indicator whether the value is increasing or decreasing making it easier for the observer to analyze the data by looking at the intensity of the light emitted. In this setup, the Arduino will utilize its built-in ADC to measure the analog voltage, converting the analogue value to a number between 0 to 1023 in digital value. By establishing a connection between Arduino and a computer via USB, the values can be sent and used by Python.

2. **Software**

ARDUINO CODE:

```
int Led_pin = 9;
int Pot_pin = A0;

void setup() {
pinMode(Led_pin,OUTPUT);
Serial.begin(9600);
}
void loop() {
int potValue = analogRead(Pot_pin);
```

```
int brightness = potValue/4 ;
analogWrite(Led_pin, brightness);
Serial.println(potValue);
delay(1000); // add a delay to avoid sending data too fast
}
```

PYTHON CODE:

```
# -- Replace 'COMX' with your Arduino's serial port
ser = serial.Serial('COMX', 9600)
try:
while True:
pot_value = ser.readline().decode().strip()
print("Potentiometer Value:", pot_value)
except KeyboardInterrupt:
ser.close()
print("Serial connection closed.")
```

These Arduino and Python codes both utilize a simple communication setup between Arduino board and Python. The Arduino reads the value of potentiometer from the analog pin, converts it to a digital value and at the same time gives a value of voltage to the LED. The Arduino then sent the value to Python over serial communication using the COM port of the computer. Meanwhile, Python takes the value from Arduino and prints the potentiometer value to the computer. It is important to note that the baud rates used in both code are the same(in this case '9600') as a mismatch can lead to communication and data errors.

- **Led_pin = 9;**: Specifies that the LED is connected to digital pin 9.
- **Pot_pin = A0;**: Assigns analog pin A0 to the variable Pot_pin, indicating that the potentiometer is connected to this pin.
- **pinMode(Led_pin, OUTPUT);**: Configures pin 9 as an output pin to control the LED.
- **Serial.begin(9600);**: Initializes serial communication at a baud rate of 9600 bits per second.
- **int potValue = analogRead(Pot_pin);**: Reads the analog value (0-1023) from the potentiometer connected to pin A0 and stores it in the variable potValue.
- **int brightness = potValue / 4;**: Calculates the brightness level for the LED by dividing the potentiometer value by 4. This assumes that the potentiometer value ranges from 0 to 1023 and adjusts it to a range suitable for analogWrite (0-255).

- **analogWrite(Led_pin, brightness);**: Sets the brightness of the LED connected to pin 9 based on the calculated brightness level.
- **Serial.println(potValue);**: Sends the potentiometer value over serial communication to the connected device (e.g., a computer) for monitoring or further processing.
- **delay(1000);**: Adds a delay of 1000 milliseconds (1 second) to slow down the loop execution, ensuring that the potentiometer value is sent over serial communication at a manageable rate.
- **ser = serial.Serial('COMX', 9600)**: This line initializes a serial connection using the serial.Serial class. Replace 'COMX' with the actual serial port your Arduino is connected to (e.g., 'COM3' on Windows). The baud rate is set to 9600, which should match the baud rate specified in Arduino code.
- **try: and except KeyboardInterrupt:**: These lines set up a try-except block to handle keyboard interrupts (Ctrl+C). When the user interrupts the program, the serial connection is closed, and a message is printed.
- **while True:**: This creates an infinite loop, continuously reading data from the serial port.
- **pot_value = ser.readline().decode().strip()**: This line reads a line of data from the serial port using ser.readline(). The data is then decoded from bytes to a string using decode() and stripped of leading/trailing whitespace characters using strip().
- **print("Potentiometer Value:", pot_value)**: This line prints the received potentiometer value to the console, along with a descriptive message.
- **ser.close()**: This line closes the serial connection when the user interrupts the program (Ctrl+C).

Conclusion

The project shows that it is certainly possible to establish a communication between two different systems, a microcontroller and a computer based system (in this case Arduino and Python). It demonstrates a simple yet effective way of exchanging data and displaying it entirely on different platforms as long as a communication is established.

Based on this project, it is shown that some codes are essential from both Python and Arduino to ensure that both systems can communicate. It can be understood that sometimes interchanging and communication between different systems may sometimes be essential and more efficient rather than relying on one system only, as everything has its own pros and cons.
From this particular project, it showcases the advantages of Arduino in handling simple external components, and Python's ability as well as versatility in handling and processing data.

Recommendations

1) It is highly recommended that a new breadboard is given for the students. We have faced and wasted unnecessary time finding fault in our setup when we finally tried moving the jumper wire to another breadboard hole.
2) A graph can be plotted in python to make the data easier to be analyzed.

EXPERIMENT 3B

Abstract

The goal of this investigation is to showcase the ability to control a servo motor using Python by connecting it to an Arduino board. The servo motor will be connected to the Arduino board, which will receive instructions from the Python program and regulate the servo motor accordingly. The Python program will transmit serial commands to the Arduino board, which will then interpret the commands and control the servo motor. This investigation will offer practical experience in programming microcontrollers and managing servo motors, which can be beneficial in various fields such as robotics, automation, and control systems. By the end of the investigation, the participant will have developed a functional Python program that can manipulate a servo motor via an Arduino board, and will have gained a deeper comprehension of control systems and microcontroller programming concepts.

Introduction

In this experiment, the goal is to develop a Python program that can control a servo motor through an Arduino board. The servo motor is a type of motor that can be precisely controlled to rotate to a specific angle, making it ideal for applications that require precise motion control.

The experiment involves interfacing the servo motor with an Arduino board, which will receive commands from the Python program and control the servo motor accordingly. The Python program will send serial commands to the Arduino board, which will interpret the commands and control the servo motor in response.

Materials and Equipment

Arduino board (e.g., Arduino Uno) X 1
Servo motor X 1
Jumper wires X 3
Potentiometer (for manual angle input) X 1
USB cable for Arduino X 1
Computer with Arduino IDE and Python installed X 1

Methodology

1. Set up the hardware: Connect the servo motor to the Arduino board using a servo cable and connect the Arduino board to the computer via a USB cable.

2. Install the necessary software: Install the Arduino IDE and Python environment on the computer.

3. Write the Arduino program: Create an Arduino program that receives serial commands from the Python program and controls the servo motor accordingly. The program should specify the servo motor's control pin and set it as an output. It should also define the pulse width and frequency for controlling the servo motor.

4. Write the Python program: Develop a Python program that sends serial commands to the Arduino board to control the servo motor. The program should import the necessary libraries for serial communication and define the servo motor's desired angle and position. It should use conditional statements and loops to control the servo motor's movement and position.

5. Test the experiment: Run the Python program and observe the servo motor's movement. Adjust the pulse width and frequency in the Arduino program as needed to achieve the desired servo motor movement.

6. Document the experiment: Record the hardware and software setup, the Arduino and Python programs, and the testing results for future reference.

By following this methodology, the participant can gain hands-on experience in programming microcontrollers and controlling servo motors, which can be useful in various applications such as robotics, automation, and control systems.

Question

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 can 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.

Arduino

```
#include <Servo.h>

Servo servo;
int servoPin = 9;
int angle;
int potPin = A0; // Analog pin for potentiometer
```

```
int potValue;

void setup() {
  servo.attach(servoPin);
  Serial.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    angle = Serial.read();
    servo.write(angle);
    delay(15);
  }

  potValue = analogRead(potPin);
  angle = map(potValue, 0, 1023, 0, 180); // Map potentiometer value to servo angle
  servo.write(angle);

  if (Serial.available() > 0) {
    char key = Serial.read();
    if (key == 'q') {
      while (Serial.available() > 0) Serial.read(); // Clear serial buffer
      servo.write(90); // Stop the servo at 90 degrees
    }
  }
}
```

Python

```
import serial
import time

ser = serial.Serial('COM3', 9600)  # Replace 'COM3' with the appropriate serial port

try:
    while True:
        angle = input("Enter servo angle (0-180 degrees) or 'q' to stop: ")
        if angle.lower() == 'q':
            ser.write(b'q')  # Send 'q' to stop the servo
            break
        angle = int(angle)
```

```
        if 0 <= angle <= 180:
            ser.write(str(angle).encode())
        else:
            print("Angle must be between 0 and 180 degrees.")
except KeyboardInterrupt:
    pass
finally:
    ser.close()
    print("Serial connection closed.")
```

Discussion

Arduino Adjustments;

1. Potentiometer input was added to read the current position using analogRead(A0).
2. The value of the potentiometer is mapped to the servo's angle range using map(potValue, 0, 1023, 0, 180).
3. The 'Q' key is added to stop the servo motor and halt code execution.

Python Adjustments;

1. The 'Q' key is added to stop the servo motor and halt code execution.
2. Servo position data is displayed over the serial connection to the user interface

Conclusion

This experiment makes use of Python and Arduino's ability to communicate with each other in real-time. This ensures its ability to receive angle adjustment via a potentiometer in real-time accurately. By following the steps, we can observe the changes that happen to the servo motor due to the changes in the potentiometer. This experiment showcases the huge potential of connecting software like Python and Arduino with the help of servo motors and other components to create a functioning feedback system. This will open up possibility for other various applications in the robotics and automation fields.

Recommendations

1. Add error handling in case of serial communication problems.
2. Add a feedback system to confirm successful angle movement from the Arduino.

Acknowledgements

We would like to express our sincere gratitude to the lab technician for their invaluable guidance, support, and encouragement throughout this project. Their expertise and insights have been instrumental in shaping the direction of this work. I also thank my fellow peers for their assistance and collaboration, which greatly contributed to the successful completion of this project.

Declaration

We hereby declare that the work presented in this report is entirely my own, except where otherwise acknowledged. I affirm that I have adhered to the principles of academic integrity and have not engaged in any form of plagiarism or unethical conduct in the completion of this project. All sources of information and assistance used in this work have been properly cited and acknowledged.