LABORATORY REPORT

LAB 3: Parallel, Serial, and USB Interfacing

GROUP H

PROGRAMME: MECHATRONICS ENGINEERING

GROUP MEMBERS:	MATRIC NO:
1. SYEDA SAMIA	2123536
2. ATHIRAH HAZIQAH BINTI BADERULHISHAM	2210240
3. SITI HEIDI AMIRA BINTI AZRUL	2111970

DATE OF SUBMISSION:
Wednesday, 27th March 2024

ABSTRACT

Serial communication is one of the most common methods of transferring data between computers and microcontrollers. We establish serial communication between an Arduino and Python, using some codes on both the Python and the Arduino sides. In this example, we are going to transfer potentiometer values from an Arduino to the Python script via a USB connection and then plot a real time graph of the potentiometer values. Moreover, a servo motor is controlled using Python and a potentiometer by interfacing with an Arduino board that's connected to the servo.

Table of Contents

INTRODUCTION

The experiment conducted aimed to explore the practical application of Arduino microcontrollers in conjunction with basic electronic components, specifically a potentiometer and a resistor, to control the brightness of an LED. This experiment was designed to provide hands-on experience with circuit construction, and Arduino programming. The primary purpose of this experiment was to demonstrate how an Arduino board can be utilized to manipulate the brightness of an LED using a potentiometer, thereby illustrating the principles of analog voltage reading. The objectives of the experiments were to construct a simple circuit using an Arduino board, a potentiometer, a resistor, and an LED and to observe and analyze the relationship between the potentiometer position and the LED brightness. In electronic circuits, a potentiometer is a variable resistor with three terminals, often used to adjust the voltage in a circuit. By rotating the knob of the potentiometer, the resistance between its terminals changes, allowing for varying levels of voltage output. Last but not least, the expected outcome is as the potentiometer knob is turned clockwise or counterclockwise, the LED brightness will change accordingly.

PROCEDURE

Materials And Equipment:

- Arduino Board
- USB cable
- Potentiometer
- Servo motor
- Jumper Wires
- LED
- 220Ω resistor
- Breadboard
- Computer with Arduino IDE and Python installed

Experimental Setup for 3a:

1. Connect one leg of the potentiometer to 5V on the Arduino.
2. Connect the other leg of the potentiometer to GND on the Arduino.
3. Connect the middle leg (wiper) of the potentiometer to an analog input pin on the Arduino, such as A0.
4. Connect one leg of the LED to GND.
5. Connect the other leg to digital pin D7 using a resistor.

Experimental Setup for 3b:

1. Keep the setup from part 1 of the experiment.
2. Connect the Servo's Signal Wire to a digital pin (e.g., D9).
3. Connect the servo's power wire (usually red) to the 5V output on the Arduino board.
4. Connect the servo's ground wire (usually brown) to one of the ground (GND) pins on the Arduino.

Methodology for 3a:

1. Using the circuit setup shown in fig 1.1, we built the needed circuit.
2. An Arduino code was uploaded to the Arduino Mega.
3. Run the Python code on the computer.
4. When the potentiometer knob is turned, the values show up in the Python terminal.

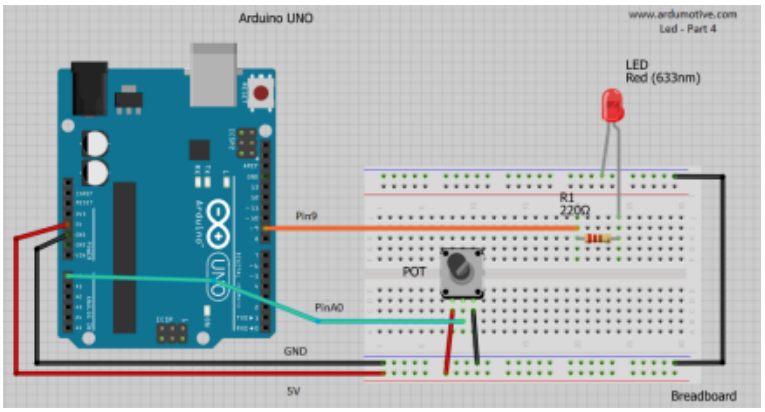


Fig 1.1

Arduino Code:

```
const int led = 7;


void setup() {
 Serial.begin(9600);
 pinMode(led,OUTPUT);
}
```

```
void loop() {
 int potValue = analogRead(A0);
 Serial.println(potValue);
 delay(1000); // add a delay to avoid sending data too fast
 if (potValue >= 300)
  {
   digitalWrite(led,HIGH); //LED turns on when readings are higher than
   300
  }
 else
  digitalWrite(led,LOW);
}
```

Python Script:

```
import serial
ser = serial.Serial('COM13', 9600)
try:
 while True:
  pot_value = ser.readline().decode().strip()
  print("Potentiometer Value:", pot_value)
except KeyboardInterrupt:
 ser.close()
 print("Serial connection closed.")
```

Methodology for 3b:

1. Using the circuit setup shown in fig 1.2, we connect the servo motor.
2. The arduino code was uploaded.
3. Run the Python script.

4. The user can input an angle in Python then the Arduino will make the servo set its position to the one inputted.

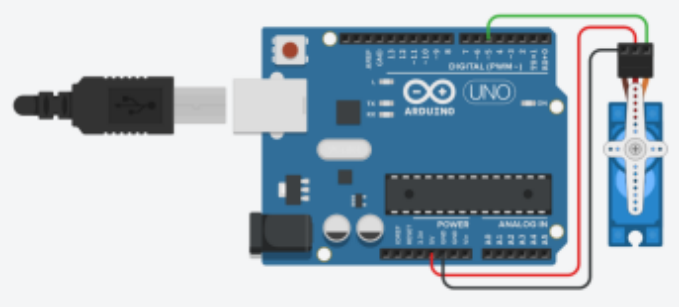


Fig 1.2

Arduino Code:

```
#include <Servo.h>


Servo servoMotor;


void setup() {
    Serial.begin(9600);
    servoMotor.attach(9);
}


void loop() {
    if (Serial.available() > 0) {
        int angle = Serial.parseInt(); // Read angle data from serial
   port
        servoMotor.write(angle); // Move servo to the specified angle
    }
}
```

Python Script:

```
import serial
import time
import sys

ser = serial.Serial('COM7', 9600)
def move_servo(angle):
   ser.write(str(angle).encode())  # Send angle data to Arduino
   time.sleep(0.1)  # Delay to allow time for Arduino to process

if __name__ == "__main__":
   print("Enter an angle (0-180) or 'q' to quit.")
   while True:
     try:
          user_input = input("Angle or 'q' to quit: ")
          if user_input.lower() == 'q':
                print("Exiting program.")
                sys.exit()
          angle = int(user_input)
          if 0 <= angle <= 180:
                move_servo(angle)
          else:
                print("Angle must be between 0 and 180.")
          except ValueError:
          print("Invalid input. Please enter an integer or 'q' to quit.")
```

RESULTS

A real time graph was generated using the values of the potentiometer and as seen in the video included in the week 3 file in github, the graph had an upward slope when the knob was turned counter clockwise, on the other hand it had a downward incline when turned to the clockwise. The slope is not linear since the turning motion done was not smooth and it kept stopping from time to time. For the

second task, the servo motor was controlled using the potentiometer. so whenever the knob was turning the servo kept on turning the same way, it is seen clearly in the other video uploaded.

The graph fig 2.1 below is part of the real time plot of the potentiometer:

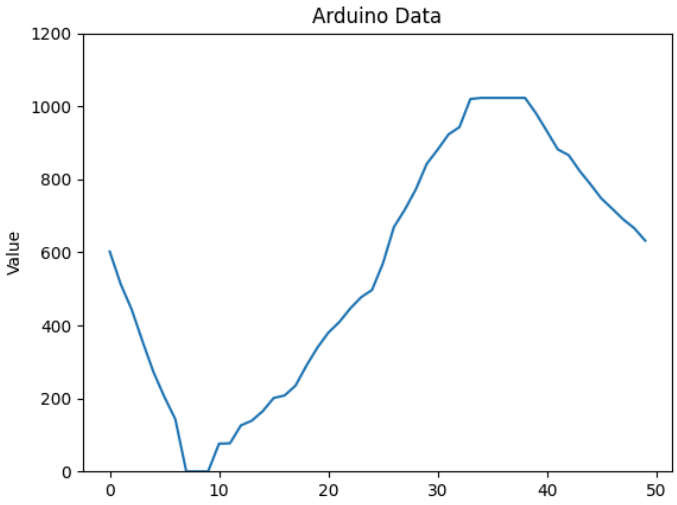


Fig 2.1

DISCUSSION

For task 3: to present the potentiometer readings graphically in Python we used the Python script below with the previous Arduino code:

```
import time
import serial
import matplotlib.pyplot as plt
import matplotlib.animation as animation
def animate(i, dataList, ser):
      ser.write(b'g')
      arduinoData_string = ser.readline().decode('ascii')
      #print(i)
      try:
            arduinoData_float = float(arduinoData_string)
            dataList.append(arduinoData_float)
```

```
        except:
            pass
        dataList = dataList[-50:]
        ax.clear()
        ax.plot(dataList)
        ax.set_ylim([0, 1200])
        ax.set_title("Arduino Data")
        ax.set_ylabel("Value")

dataList = []
fig = plt.figure()
ax = fig.add_subplot(111)
ser = serial.Serial("COM13", 9600)
time.sleep(2)
ani = animation.FuncAnimation(fig, animate, frames=100,
fargs=(dataList, ser), interval=100)
plt.show()
ser.close()
```

Explanation: dataList is a list to store received data points, then python sends the character 'g' to the Arduino by [ser.write(b'g')] to request data. Moreover, it reads a line from the serial port, assuming it represents a floating-point number then appends the received value to dataList, handling any exceptions. we limit dataList to contain only the last 50 data points with the line [dataList = dataList[-50:]], we clear the subplot using [ax.clear()]. dataList is plotted on the subplot and sets y-axis limits to [0, 1200] then sets the title and y-axis label for the plot.Furthermore, an empty list dataList is created to store received data whilst also creating a new figure fig and adds a subplot ax. It initializes the serial port communication with the Arduino and introduces a 2-second delay [time.sleep(2)] to allow the serial port to initialize properly. [ani = animation.FuncAnimation(fig, animate, frames=100, fargs=(dataList, ser), interval=100):] Creates an animation by repeatedly calling the animate function. fig: The figure to animate. animate: The function to call at each frame. frames: The number of frames to generate. fargs: Additional arguments to pass to the animate function (dataList and ser). interval: Delay between frames in milliseconds. [plt.show()] Displays the animated plot.

For task 3b: to incorporate a potentiometer for real-time adjustments of the servo motor's angle we used the Arduino and Python codes below:

```
#include <Servo.h>
```

```
Servo servoMotor;
int potPin = A0;
int potValue = 0;
int angle = 0;

void setup() {
    Serial.begin(9600);
    servoMotor.attach(9);
}

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

    // Check if any data is available from serial port
    if (Serial.available() > 0) {
        char receivedChar = Serial.read();  // Read the incoming byte
        if (receivedChar == 's') {
            // Stop servo motion if 's' is received
            servoMotor.write(90);  // Stop servo at the center position
        }
    }
    delay(50);
```

```
}
```

Explanation: potValue: Stores the value read from the potentiometer and angle: Stores the mapped angle value for controlling the servo motor. The arduino reads the analog value from the potentiometer (ranging from 0 to 1023) then maps to a servo angle (ranging from 0 to 180). This ensures that the potentiometer's full range corresponds to the servo's full range of motion. then, it writes mapped angle values to the servo to set its positions. Furthermore, if there is any data available from the serial port then it reads a single byte from the serial buffer. But if it receives the character ‘s’ then it would set the servo to the center position stopping it from moving. The delay is to help stabilize the readings.

```
import serial
import time
import sys
import keyboard

ser = serial.Serial('COM7', 9600)

def move_servo(angle):
	ser.write(str(angle).encode())
	time.sleep(0.1)

if __name__ == "__main__":
print("Use the potentiometer to adjust the servo angle.")
print("Press 'q' to quit, and 's' to stop the servo.")
while True:
	try:
		if keyboard.is_pressed('q'):
		print("Exiting program.")
		sys.exit()  # Exit the program
		if keyboard.is_pressed('s'):
		print("Stopping servo.")
		ser.write(b's')  # Send 's' to stop the servo
		time.sleep(0.1)  # Delay for stability
		pot_value = int(ser.readline().decode().strip())
		angle = int(map(pot_value, 0, 1023, 0, 180))
```

```
            move_servo(angle)
        except ValueError:
            print("Invalid input.")
```

Explanation: define move_servo function sends the desired angle to the Arduino via serial communication. [ser.write(str(angle).encode())] Sends the angle value as a string encoded in bytes to the Arduino. [time.sleep(0.1)] Introduces a small delay for stability. [if __name__ == "__main__":] Ensures that the following code block is executed only if the script is run directly (not imported as a module). print("Use the potentiometer to adjust the servo angle."): Instructs the user to adjust the servo angle using the potentiometer. [print("Press 'q' to quit, and 's' to stop the servo.")] Instructs the user on how to quit the program or stop the servo motor. [if keyboard.is_pressed('q'):] Checks if the 'q' key is pressed to exit the program. [if keyboard.is_pressed('s'):] Checks if the 's' key is pressed to stop the servo motor. [ser.write(b's')] Sends 's' to the Arduino to stop the servo motor. [pot_value = int(ser.readline().decode().strip())] Reads the potentiometer value from the serial port and converts it to an integer. [angle = int(map(pot_value, 0, 1023, 0, 180))] Maps the potentiometer value to a servo angle ranging from 0 to 180. [move_servo(angle)] Calls the move_servo function to move the servo to the calculated angle. [except ValueError:] Handles any value error that may occur during conversion of potentiometer value to an integer. [print("Invalid input.")] Prints a message indicating an invalid input.

CONCLUSION

Through the execution of this experiment, several key findings were observed, contributing to a deeper understanding of Arduino-based electronics and control systems. The experiment successfully demonstrated the feasibility of controlling LED brightness using a potentiometer and an Arduino microcontroller, validating the hypothesis put forth. The main findings and their significance are effective control of LED brightness. The experiment showcased the efficacy of using an Arduino board in conjunction with a potentiometer to dynamically adjust the brightness of an LED. By manipulating the potentiometer, participants were able to seamlessly modulate the LED's brightness, highlighting the versatility and precision of Arduino-based control systems. secondly, correlation between potentiometer position and LED brightness, As anticipated, there was a clear correlation between the position of the potentiometer knob and the resulting brightness of the LED. Turning the potentiometer clockwise or counterclockwise led to proportional changes in LED brightness, affirming the hypothesis and reinforcing basic principles of analog voltage control. In conclusion, the experiment not only supported the initial hypothesis but also yielded valuable insights into the broader applications and implications of Arduino-based control systems. By demonstrating the effective control of LED brightness through a simple yet versatile setup, the experiment highlighted the potential for Arduino microcontrollers in diverse fields, including robotics, automation, and interactive installations. Moving forward, further exploration and experimentation in these areas could yield innovative solutions to a wide range of technological challenges, underscoring the significance of hands-on learning experiences like the one presented in this experiment.

RECOMMENDATIONS

In future iterations of the experiment, enhancements could include integrating additional sensors or actuators to create more complex interactive systems, such as incorporating temperature or light sensors to trigger LED brightness adjustments based on environmental conditions. Moreover, introducing multiple LEDs with varying colors or intensity levels could expand the scope of the experiment, enabling exploration of color mixing and dynamic lighting effects. Additionally, incorporating wireless communication modules like Bluetooth or Wi-Fi could facilitate remote control of the LED system, offering opportunities for smartphone integration or IoT applications. Furthermore, providing supplementary resources such as online tutorials or documentation on circuit design, Arduino programming, and troubleshooting techniques would enhance participants' learning experience, fostering a deeper understanding of electronics principles and encouraging experimentation and innovation. Overall, future students could benefit from a more comprehensive exploration of Arduino-based projects, allowing for greater creativity and customization while reinforcing fundamental concepts in a hands-on learning environment.

REFERENCES

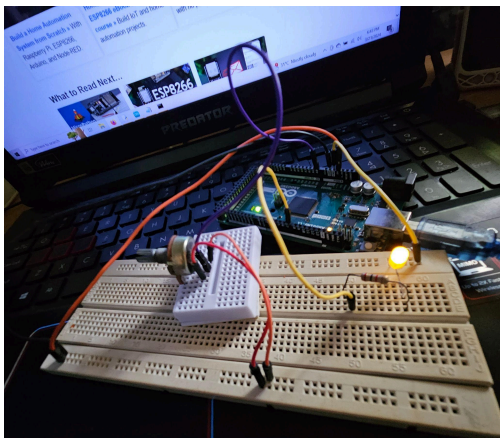
Arduino and Python Real Time Plot Animation | Lesson 1 Getting Started | PySerial MatPlotLib. (n.d.). Www.youtube.com. Retrieved March 22, 2024, from https://www.youtube.com/watch?v=z14l33paZGU

Arduino_RealTimePlot/Part1_GettingStarted/ArduinoRealTimePlot.py at master · WaveShapePlay/Arduino_RealTimePlot. (n.d.). GitHub. Retrieved March 22, 2024, from https://github.com/WaveShapePlay/Arduino_RealTimePlot/blob/master/Part1_GettingStarted/ArduinoRealTimePlot.py#L11

Simple Plot in Python using Matplotlib. (2020, April 12). GeeksforGeeks. https://www.geeksforgeeks.org/simple-plot-in-python-using-matplotlib/

Mansoori, J. (2019, January 23). Python3 and Arduino Communication [Review of Python3 and Arduino Communication]. Maker Pro. https://maker.pro/arduino/projects/python3-and-arduino-communication

APPENDICES



ACKNOWLEDGEMENT

I would like to express my sincere gratitude to the team for their invaluable guidance and support throughout the execution of this experiment. Their expertise and insightful feedback greatly enhanced my understanding of the principles underlying Arduino-based electronics. Additionally, for their assistance in troubleshooting technical challenges and providing constructive suggestions for improvement. Furthermore, I am thankful to my peers for their collaborative spirit and exchange of ideas, which enriched the overall learning experience and fostered a collaborative learning environment.

STUDENT'S DECLARATION

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 no further improvement on the reports is needed from any of the individual's 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.

Name: Syeda Samia Matric Number: 2123536 Contribution: Abstract, Results, Procedure, Discussion.	Read	✓
	Understand	✓
	Agree	✓
Name: Athirah Haziqah bt Baderulhisham Matric Number: 2210240 Contribution: Recommendations, Acknowledgement	Read	✓
	Understand	✓
	Agree	✓
Name: Siti Heidi Amira Binti Azrul Matric Number: 2111970 Contribution: Intro, Conclusion	Read	✓
	Understand	✓
	Agree	✓