



(Company No.: 198301005860)

الجامعة الإسلامية العالمية ماليزيا
INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA
يُونِيسَيْتِي إِسْلَامِيَّةٌ إِنْتَارَابَغْسِيَا مَلَيْسِيَا

Garden of Knowledge and Virtue

MECHATRONICS SYSTEM INTEGRATION : MCTA 3203

LABORATORY REPORT

SECTION 1

TITLE : SERIAL COMMUNICATION BETWEEN ARDUINO AND PYTHON. VER 3 (WEEK 3)

LECTURER'S NAME: ASSOC. PROF. IR. TS. DR. ZULKIFLI BIN ZAINAL ABIDIN

DATE OF EXPERIMENT : 22 OCTOBER 2025

DATE OF SUBMISSION : 29 OCTOBER 2025

GROUP : 4

NO.	NAME	MATRIC NO.
1	MOHAMMAD FARISH ISKANDAR BIN MOHD SYAHIDIN	2311779
2	AHMAD HARITH IMRAN BIN MOHD YUSOF	2311581
3	ARIFA AQILAH BINTI ABDUL HALIM	2311530

ACKNOWLEDGEMENTS

We would like to express our heartfelt appreciation to Dr. Zulkifli Bin Zainal Abidin for his continuous guidance, support, and insightful explanations throughout this experiment. His expertise greatly helped us understand the concepts of interfacing and digital control using Arduino. We would also like to thank our lab assistant for providing technical assistance and ensuring that the experiment ran smoothly. Finally, we extend our gratitude to our classmates for their cooperation and teamwork during the lab session.

ABSTRACT

This experiment demonstrates the process of establishing a serial communication connection between an Arduino microcontroller and a Python program to enable real-time transmission of potentiometer readings. The Arduino continuously sends data through a USB connection, while a Python script uses the pyserial library to receive and process the incoming information.

The results confirm that stable, real-time communication can be achieved, serving as a proof of concept for serial data exchange between an Arduino and a computer. This setup effectively enables the use of live data within Python applications, showing how hardware readings can be integrated into software-based systems.

In summary, the experiment successfully established a functional serial communication link between the Arduino and Python for sharing potentiometer data. When both systems are connected, this configuration proves highly suitable for real-time data handling in Python. The outcomes provide essential groundwork for future developments and applications involving microcontroller-to-computer communication.

TABLE OF CONTENT

ACKNOWLEDGEMENTS.....	2
ABSTRACT	3
TABLE OF CONTENT	4
1.0 INTRODUCTION	5
2.0 MATERIAL AND EQUIPMENT	6
3.0 EXPERIMENTAL SETUP	7
4.0 METHODOLOGY	9
5.0 DATA COLLECTION	18
6.0 DATA ANALYSIS	21
7.0 RESULTS.....	25
8.0 DISCUSSION.....	27
9.0 CONCLUSION	29
10.0 RECOMMENDATIONS.....	30
11.0 REFERENCES.....	31
12.0 APPENDICES.....	32
13.0 STUDENT'S DECLARATION	36

1.0 INTRODUCTION

This experiment focuses on controlling a servo motor using an Arduino board through serial communication with Python. The purpose of the experiment is to establish a communication link between the Arduino and a computer, allowing the servo motor to respond to input commands sent from Python in real time. This setup demonstrates how data can be transmitted and interpreted between hardware and software for precise motion control applications.

In this experiment, serial communication serves as the medium for data exchange, enabling the Arduino to receive angle values and adjust the servo position accordingly. The servo motor, which converts electrical signals into controlled mechanical motion, operates based on pulse-width modulation (PWM). By sending varying pulse signals, the motor can rotate to specific angular positions within a range of 0° to 180° .

The objectives of this experiment are to understand the principles of serial communication between Arduino and Python, to learn how to interface and control a servo motor using software commands, and to explore real-time data visualization. It is expected that the servo motor will accurately follow the input angles provided via the Python interface, demonstrating effective serial data transfer and motion control.

2.0 MATERIAL AND EQUIPMENT

EXPERIMENT 1

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

EXPERIMENT 2

1. Arduino Board
2. Servo Motor
3. Jumper Wires
4. Potentiometer
5. USB cable for Arduino

3.0 EXPERIMENTAL SETUP

EXPERIMENT 1

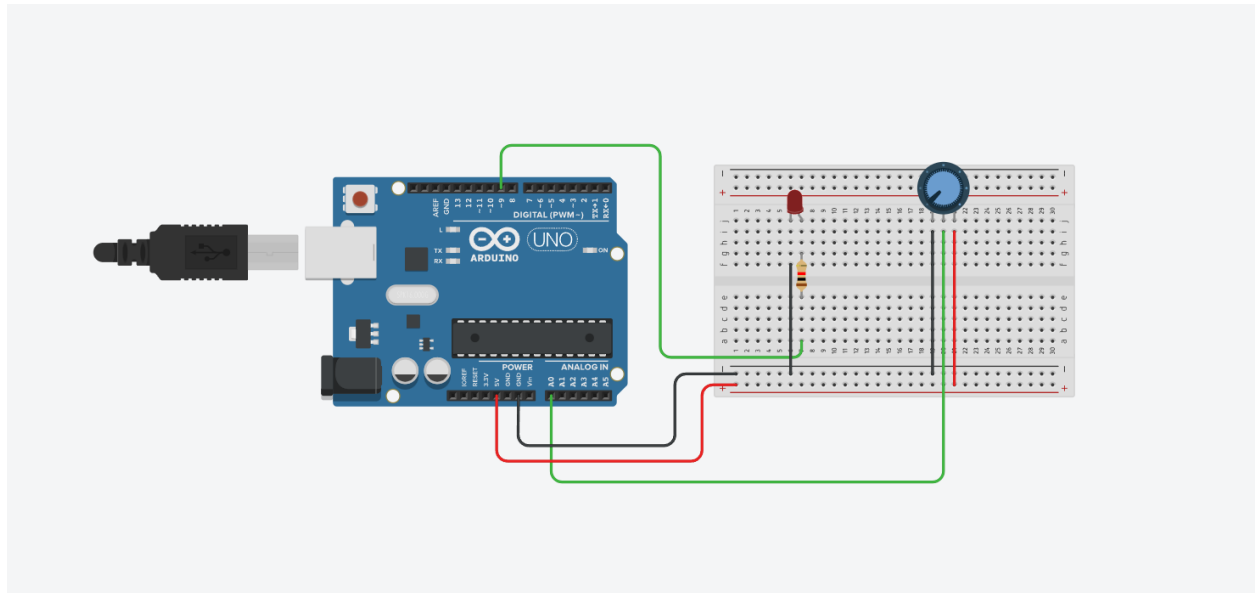


Figure A: Components connected to arduino uno in tinkercad

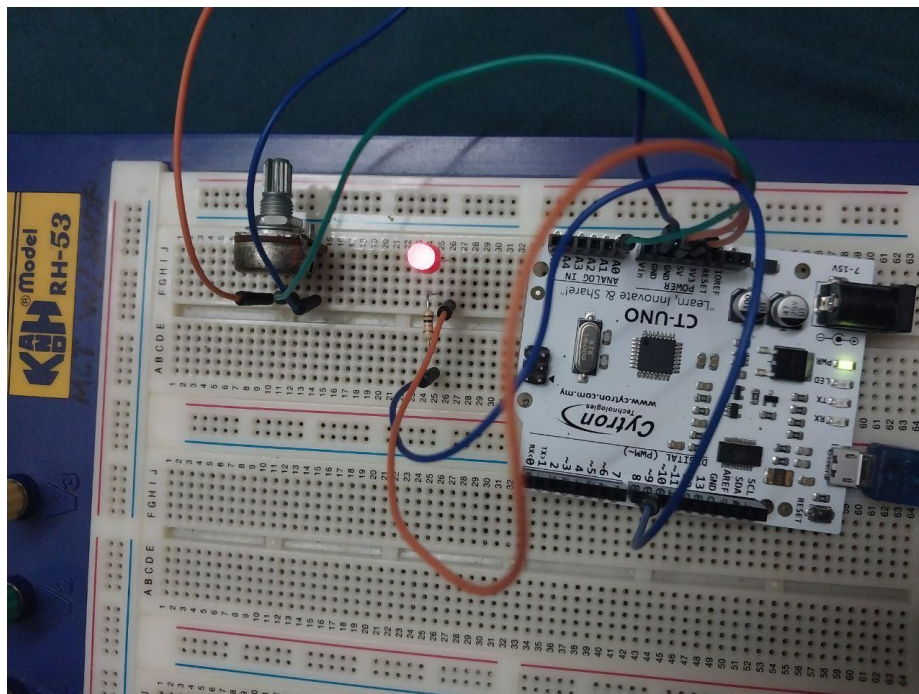


Figure B: Components connected to arduino uno physically

EXPERIMENT 2

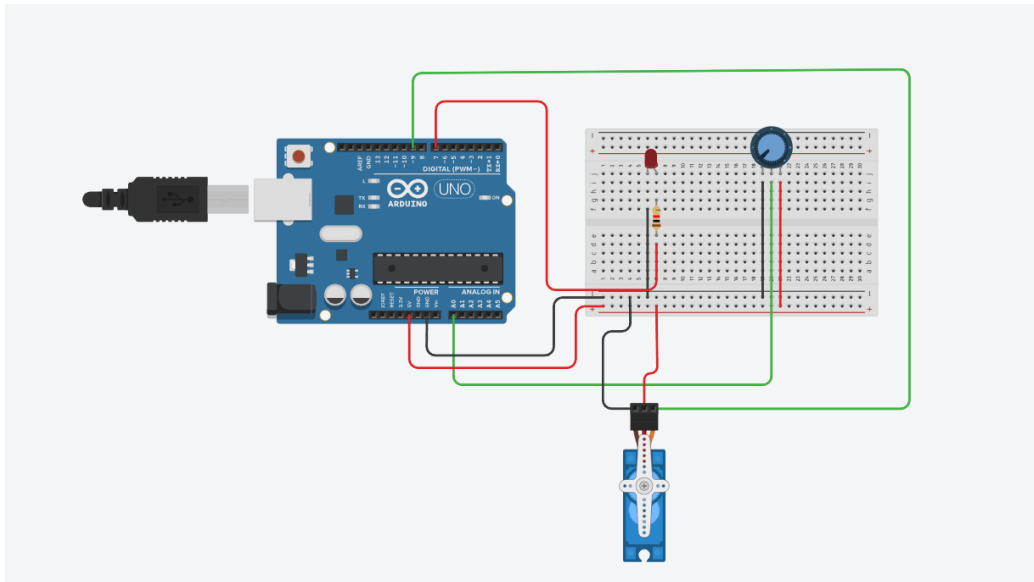


Figure C: Components connected to arduino uno in tinkercad

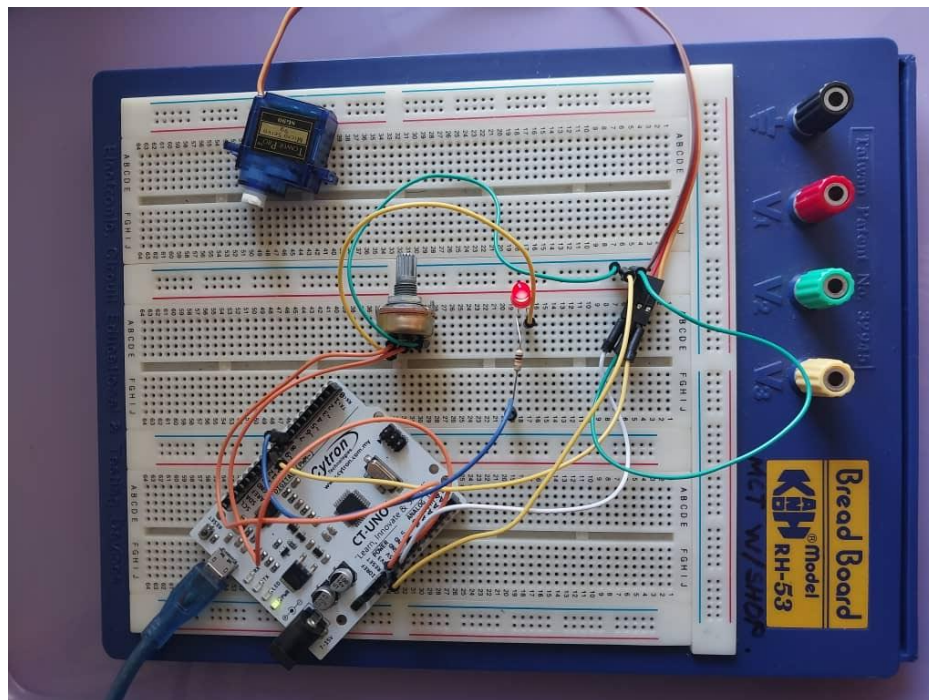


Figure D: Components connected to arduino uno physically

4.0 METHODOLOGY

EXPERIMENT 1

1. The circuit was built according to the instructions given in the lab module.
2. The Arduino code was uploaded to the Arduino Uno using the Arduino IDE.
3. The potentiometer was turned slowly, and the LED brightness changed based on the rotation.
4. The brightness value was displayed in the Serial Monitor and plotted in real time using Python
5. The process was repeated several times to observe consistent data transmission and LED response.

Circuit Assembly

- The LED was connected to pin 9 of the microcontroller.
- The potentiometer was connected to analog pin A0.
 - One leg connected to 5V
 - The other leg connected to GND
 - The middle leg connected to A0
- All components shared a common ground to ensure stable operation.

Programming Logic

- The Arduino code continuously reads the potentiometer value from pin A0.
- The value was sent to the computer using serial communication at 9600 baud rate.

- The Python program received the potentiometer value through the serial port.
- If the potentiometer value was more than 511, Python sent a command to turn ON the LED.
- If the potentiometer value was less than 511, Python sent a command to turn OFF the LED.
- The Arduino received the command and controlled the LED on pin 9 based on the signal sent from Python.
- A short delay was added to make the LED response smooth and stable.

Code Used

- Arduino

```
const int LEDPIN = 9;
char command;
void setup()
{
  pinMode(LEDPIN,OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  int potValue = analogRead(A0);
  command = Serial.read();
  Serial.println(potValue);
  if(command == 'f')
  {
    digitalWrite(LEDPIN,HIGH);
  }
}
```

```

else if(command == 's')
{
    digitalWrite(LEDPIN,LOW);
}
delay(1000);
}

```

- Python

```
import serial #get serial port from arduino
```

```
ser = serial.Serial('COM3', 9600,timeout=2) #get data from arduino
```

```
try:
```

```
    while True:
```

```
        value = ser.readline().decode('ascii').strip()
```

```
        potValue = int(value) #establish it as integer
```

```
        print("Potentiometer Value:", potValue)
```

```
        if potValue > 511: # turn on LED
```

```
            ser.write(b'f')
```

```
        elif potValue < 511: #turn off LED
```

```
            ser.write(b's')
```

```
        time.sleep(1)
```

```
except KeyboardInterrupt: # press Ctrl + c to stop
```

```
    ser.close()
```

```
    print("Serial connection closed.")
```

Control Algorithm

1. Start Initialization

- Define LEDPIN as 9 (output pin for LED).
- Start serial communication at 9600 baud rate.
- Set LEDPIN as an output.

2. Continuous Loop Execution

- Read the potentiometer value from analog pin A0.
- Send the potentiometer value to the computer using `Serial.println()`.
- Receive a command from Python through the serial port.
- If the command is 'f', turn the LED ON using `digitalWrite(LEDPIN, HIGH)`.
- If the command is 's', turn the LED OFF using `digitalWrite(LEDPIN, LOW)`.
- Add a short delay to make the LED response stable.

3. Repeat Indefinitely

- The loop keeps running, allowing the LED to turn ON and OFF based on the real-time potentiometer readings and the command from Python.

EXPERIMENT 2

1. The circuit was connected according to the given setup instructions.
2. The Arduino code was uploaded to the Arduino Uno using the Arduino IDE.
3. The potentiometer was turned slowly to control the servo motor angle.
4. The servo moved according to the input value and displayed the readings in the Serial Monitor.
5. The process was repeated several times to confirm that the servo rotated smoothly and correctly.

Circuit Assembly

- The servo signal wire was connected to pin 6 on the Arduino.
- The power wire (red) of the servo was connected to 5V, and the ground wire (black/brown) was connected to GND.
- The potentiometer was connected to analog pin A0:
 - One side to 5V,
 - One side to GND,
 - Middle pin to A0.
- The Arduino was connected to the computer using a USB cable for power and serial communication.

Programming Logic

- Attach the servo motor to pin 9 using `myservo.attach(servoPin)`.
- Set pin 7 as an output to control the LED.
- Begin serial communication at 9600 baud rate to send and receive data with Python.
- Wait for a command from Python through the serial port.
- When the command is 'f':
 - Read the potentiometer value from pin A0.
 - Convert the value from 0–1023 to 0–180 degrees using `map()`.
 - Move the servo motor to the mapped angle using `myservo.write(angle)`.

- Print both the potentiometer value and servo angle to the Serial Monitor for real-time display.
- Turn the LED ON if the potentiometer value is more than 339, otherwise turn it OFF.
- When the command is 's', stop the process and keep the Arduino in a halted state until it is reset.
- Add a short delay to make the servo movement and LED response stable.

Code Used

- Arduino

```
#include <Servo.h>

Servo myservo;
const int potpin = A0;
const int LEDPIN = 7;
const int servoPin = 9;
int potValue = 0;
int angle = 0;
char command;

void setup()
{
  pinMode(LEDPIN, OUTPUT);
  myservo.attach(servoPin);
  myservo.write(angle);
  delay(2000);
  Serial.begin(9600);
}
```

```

void loop()
{
  if(Serial.available()>0)
  {
    command = Serial.read(); //get info from python
    if(command == 'f') //if python return 'f'
    {
      potValue = analogRead(potpin);
      angle = map(potValue,0,1023,0,180);
      myservo.write(angle);
      Serial.print(potValue); //send potentiometer to python
      Serial.print(",");
      Serial.println(angle); //send angle to python in new line
      if(potValue>339)
      {
        digitalWrite(LEDPIN,HIGH);
      }
      else
      {
        digitalWrite(LEDPIN,LOW);
      }
      delay(100);
    }
    else if (command == 's') //if python return 's'
    {
      while(true) //endless looping until arduino reset
      {
        }
      }
    }
  }
}

```

```

}
}
}

```

- Python

```
import serial #using serial port
```

```
import matplotlib.pyplot as plt #to plot the graph
```

```
import time #for delay in coding
```

```
ser = serial.Serial('COM3', baudrate = 9600, timeout=2) #to get information from arduino
```

```
plt.ion() #real time plotting
```

```
fig, ax = plt.subplots() #function for plots
```

```
x_value, y_value = [], [] #storing the data into x and y value
```

```
try: #used to ignore error and execute command
```

```
    while True:
```

```
        ser.write(b'f') #send byte 'f' to arduino
```

```
        arduinoData = ser.readline().decode('ascii').strip()
```

```
        if arduinoData:
```

```
            data = arduinoData.split(',') #split data from ',' to two data
```

```
            if len(data) == 2:
```

```
                potValue = int(data[0]) #potentiometer data
```

```
                x_value.append(len(x_value)) #assign data to x value
```

```
                y_value.append(int(potValue)) #assign data to y value
```

```
                angle = int(data[1]) #servo data
```

```
                print(f"Potentiometer: {potValue} | Servo Position: {angle}") #print into 2 separate
```

```
data
```



```
ax.clear() #clear the plot
ax.plot(x_value,y_value) #plot
plt.title("Potentiometer Reading")
plt.xlabel("Integer")
plt.ylabel("Potentiometer Value")
plt.pause(0.1)
time.sleep(1) #allow the code to run smoother and slower
```

except KeyboardInterrupt: #this function will happen if 'Ctrl + c' is pressed

```
ser.write(b's') #send byte 's' to arduino
ser.close()
plt.ioff()
plt.show()
print("Serial connection closed.")
```

Control Algorithm

1. Initialization

- Attach the servo to pin 9.
- Set A0 for the potentiometer and pin 7 for the LED.
- Start serial communication at 9600 baud rate.

2. Loop Process

- Wait for a command from Python.
- If the command is 'f':
 - Read the potentiometer value from A0.
 - Convert it to an angle between 0–180°.
 - Move the servo to that angle.

- Turn LED ON if the value is above 339, otherwise turn it OFF.
- Send both the potentiometer and angle values to Python.

- If the command is 's', stop the program until reset.

3. Repeat

- Keep updating the servo and LED based on real-time potentiometer readings.

5.0 DATA COLLECTION

EXPERIMENT 1

This experiment involved a potentiometer and led. The arduino controls the system and the python in Virtual Studio Code acts as an input to put in user data to the system but through the arduino code. The task mentioned that the serial plotter in the Arduino IDE was used and additionally, the python code can't run at the same time the serial plotter was used. Below is the table showing the wiring details for each component. (See appendix A)

Component	Wiring Details
Potentiometer	The potentiometer can control the voltage from 0v to 5v using the middle pin (wiper) and the arduino converts the voltage into a digital number ranging from 0 until 1023 via the arduino Analog-to-Digital Converter. The wiper is also connected to the A0 pin.

Led light	The LED connects to the D9 pin and will only light up if the potentiometer changes to half of the maximum digital value (511).
-----------	--

Data Transmission (Arduino):

- The Arduino code reads the analog value (0-1023) from the potentiometer.
- This value is sent over the serial port using `Serial.println(potValue)`.

Data Reception (Python):

- The python script uses the pyserial library that can connect a serial connection to the arduino.
- The python reads the data from the arduino via a serial port.

EXPERIMENT 2

This experiment involved a potentiometer, led light and a servo. The task mentioned that the python script will receive servo angle and potentiometer value and display the value of potentiometer as a graph plot. Below is the table showing the wiring details for each component. (See appendix B)

Component	Wiring Details
Potentiometer	The potentiometer can control the voltage from 0v to 5v using the middle pin (wiper) and the arduino converts the voltage into a

	digital number ranging from 0 until 1023 via the arduino Analog-to-Digital Converter. The wiper is also connected to the A0 pin.
Led light	The LED connects to the D7 pin and will only light up if the potentiometer changes to half of the maximum digital value (511).
Servo motor	The servo motor controls the angle of the arm from 0 to 180 degrees by sending a specific type of electrical signal called Pulse Width Modulation (PWM) to the servo's signal pin. The width (duration) of the electrical pulse determines the angle of the servo's shaft. One of the wires is connected to a D9 pin.

Data Transmission (Arduino):

- The Arduino code reads the analog value (0-1023) from the potentiometer.
- This value was used to set the servo.
- The data for the potentiometer and servo was sent to python across the serial connection.

Data Reception (Python):

- The python script uses the pyserial library that can connect a serial connection to the arduino.
- The python reads the data from the arduino via a serial port.
- The python includes the matplotlib.pyplot library to plot the graph.

6.0 DATA ANALYSIS

EXPERIMENT 1

From the collected data in the experiment, we analyzed the function of the code from the task given. (See appendix A)

Arduino code:

Action	Description	Output
<code>int potValue = analogRead(A0)</code>	This line of coding was used to set the potentiometer value to read from the physical components.	Value can be set from 0 until 1023
<code>command = Serial.read()</code>	This set variable command to read coding from python script.	Either 'f' or 's' character

Python code:

Action	Description	Output
<code>ser.readline()</code>	Allow the python to read the incoming data from the system.	Set the value to variable 'value'
<code>ser.write(b'')</code>	Send data in byte to the system to run the command	Send 'f' and 's'

EXPERIMENT 2

From the collected data in the experiment, we analyzed the function of the code from the task given. (See appendix B)

Arduino code:

Action	Description	Output
<code>int potValue = analogRead (A0)</code>	This line of coding was used to set the potentiometer value to read from the physical components.	Value can be set from 0 until 1023
<code>command = Serial.read()</code>	This set variable command to read coding from python script.	Either 'f' or 's' character
<code>angle = map(potValue,0,1023,0,180)</code>	This function re-map the value from and to a range. Usually, it is	To receive the potential value and set

	used to connect the relationship between potentiometer and servo.	potentiometer value from 0 until 1023 and servo from 0 until 180
<pre>Serial.print(potValue) Serial.print(",") Serial.println(angle)</pre>	This three line of code allows arduino to send two data to python script with a comma in between.	Send potentiometer value and angle value to python.
<pre>while(true){}</pre>	This is an endless loop to be used for Keyboard interrupt.	KeyboardInterrupt in python.

Python code:

Action	Description	Output
<code>ser.readline()</code>	Allow the python to read the incoming data from the system.	Set the value to variable 'value'
<code>ser.write(b'')</code>	Send data in byte to the system to run the command	Send 'f' and 's'
<code>data = ArduinoData.split(',')</code>	Split the data from the comma	Split potentiometer value and angle value
<code>potValue = int(data[0])</code>	Set the potentiometer value and	Variable now assigned

angle = int(data[1])	angle value into an array.	with the current value
----------------------	----------------------------	------------------------

7.0 RESULTS

EXPERIMENT 1

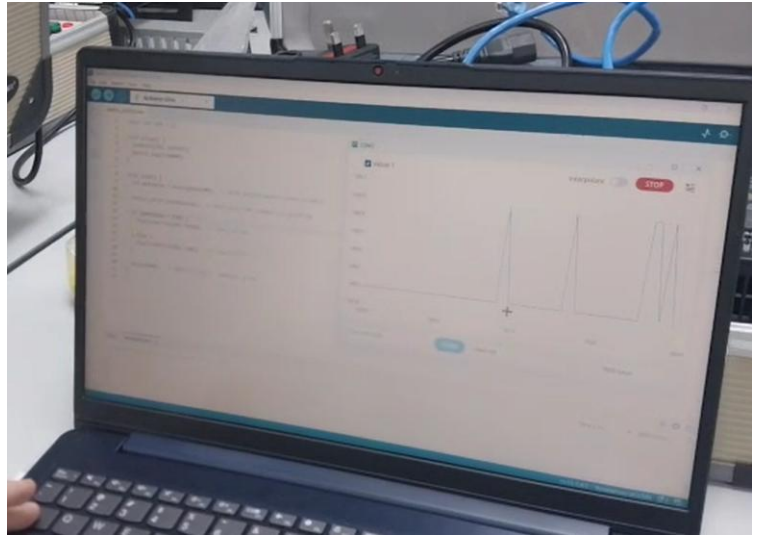
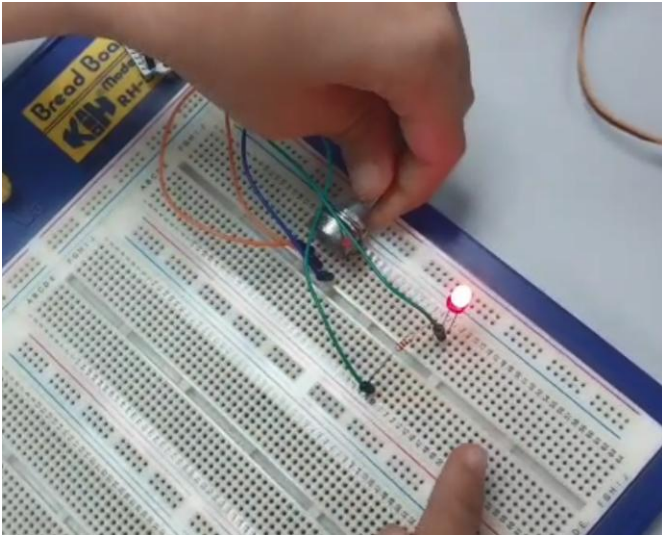


Figure E: Configuration of task 1

In this part of the experiment, the circuit was extended by adding an LED in series with a 220 Ω resistor. The Python code was updated so that the LED would automatically turn ON when the potentiometer value exceeded half of its maximum range and turn OFF when it was below that threshold. The LED responded accurately to the potentiometer's position, confirming that the serial communication and control logic were functioning correctly.

Using the Arduino IDE Serial Plotter, the potentiometer readings were visualized in real time as a smooth, continuous waveform. The graph clearly showed the variation in analog values from 0 to 1023 as the potentiometer knob was turned. This confirmed that the Arduino successfully read and transmitted the sensor data, and that the LED's switching behavior corresponded precisely to the plotted readings.

EXPERIMENT 2

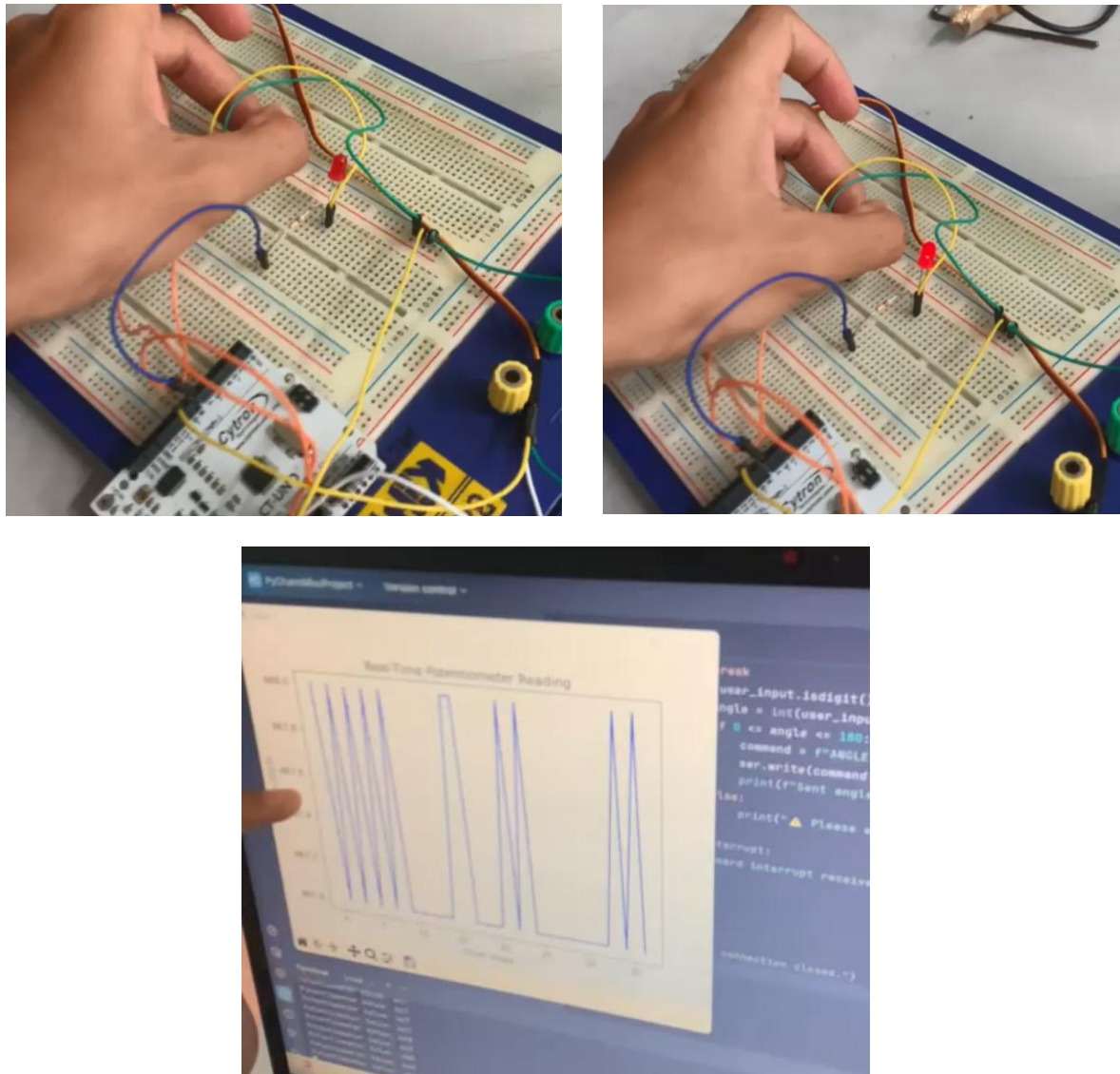


Figure F: Configuration of task 2

In this experiment, the Arduino and Python codes were successfully modified to control a servo motor using a potentiometer for real-time angle adjustment. As the potentiometer knob was

turned, the servo rotated smoothly between 0° and 180°, confirming proper mapping and serial communication between the two systems.

The LED indicator functioned correctly, lighting up when the potentiometer value exceeded a set threshold. The Python program displayed both potentiometer and servo angle values and allowed the process to stop via keyboard input. Using matplotlib, real-time graphs showed smooth changes in potentiometer readings, matching the servo's movement. Overall, the results confirmed accurate data transfer, responsive control, and effective visualization of servo behavior.

8.0 DISCUSSION

EXPERIMENT 1

The experiment successfully demonstrated the relationship between the potentiometer input and the LED brightness. As the potentiometer was turned, the LED brightness changed accordingly, and the data was sent to Python through serial communication. The results showed that Arduino and Python can communicate effectively in real time.

Sources of Error and Limitations:

1. **Electrical Noise:** Caused unstable LED brightness due to small fluctuations in the potentiometer readings.
2. **Hardware Tolerances:** Small variations in the potentiometer affected the accuracy of brightness changes.
3. **Loose Wiring:** Inconsistent connections caused unstable readings during testing.

Improvements:

1. Use a smoothing filter or average multiple readings to reduce noise.
2. Use a higher-quality potentiometer with stable resistance.
3. Secure all jumper wires firmly to ensure steady data flow.

EXPERIMENT 2

The experiment successfully showed how the potentiometer input controlled both the servo motor and the LED through serial communication with Python. The servo moved according to the potentiometer's position, while the LED turned ON or OFF depending on the value. The plotted data in Python confirmed that the readings were transmitted accurately.

Sources of Error and Limitations:

1. **Servo Motor Malfunction:** The servo stopped working due to possible overheating or overuse.
2. **Power Limitation:** The servo drew too much current from the Arduino, reducing stability.
3. **Servo Lag:** The servo movement was slightly delayed when the potentiometer changed quickly.

Improvements:

1. Use an external 5V power supply or give the servo rest intervals during testing.
2. Provide separate power to the servo for more consistent performance.

3. Use a faster or higher-quality servo motor for smoother response.

Both experiments successfully showed how Arduino and Python can work together through serial communication. The potentiometer was able to control the LED brightness and servo movement as expected. Although there were minor issues such as flickering, delays, and servo malfunction, the overall system worked well and achieved the experiment objectives.

9.0 CONCLUSION

This experiment successfully achieved its main objective, which was to control a servo motor using serial communication between Python and an Arduino board. By entering angle values in the Python interface, the servo motor responded by rotating accurately within the range of 0° to 180°, proving that the communication and control process worked as intended.

The results supported our initial hypothesis that the servo motor could be effectively controlled through serial data transmission from Python to Arduino. This demonstrates the capability of microcontrollers to interpret software commands and convert them into precise mechanical movements.

Through this experiment, we learned how software and hardware interact through serial communication. It also strengthened our understanding of data transmission, signal control, and system integration — concepts that can be applied to more advanced projects such as robotics and automation systems.

10.0 RECOMMENDATIONS

EXPERIMENT 1:

The experiment can be improved by adding a real-time graph in Python to make it easier to observe the changes in potentiometer readings and LED brightness. The accuracy of the data can also be improved by using a smoothing or averaging method to reduce flickering in the LED. Future students should check all circuit connections properly and use shorter delay times for faster and smoother responses.

EXPERIMENT 2:

For better performance, the servo motor should be powered using an external 5V supply to avoid power issues and prevent malfunction. A small delay or rest time can also help avoid overheating of the servo. The potentiometer readings can be made more stable by using a filter or taking multiple readings before sending data. Future students are advised to turn the potentiometer slowly and check the servo wiring carefully to prevent damage.

11.0 REFERENCES

- ansh2919. (2020, November 5). *Serial Communication between Python and Arduino*. Arduino Project Hub. <https://projecthub.arduino.cc/ansh2919/serial-communication-between-python-and-arduino-663756>
- Kite. (2020, March 5). *HOW TO USE Matplotlib in 4 MINUTES (2020 Python Tutorial)* [Video]. YouTube. <https://www.youtube.com/watch?v=iKGYbMD3NT8>
- WaveShapePlay. (2018, June 16). *Arduino and Python Serial Communication with PySerial Part 1* [Video]. YouTube. https://www.youtube.com/watch?v=D4VImL3G4_o
- WaveShapePlay. (2018, June 16). *Arduino and Python Serial Communication with PySerial Part 2 - Adding User Input* [Video]. YouTube. <https://www.youtube.com/watch?v=WV4U51TIRaQ>

12.0 APPENDICES

Appendix A

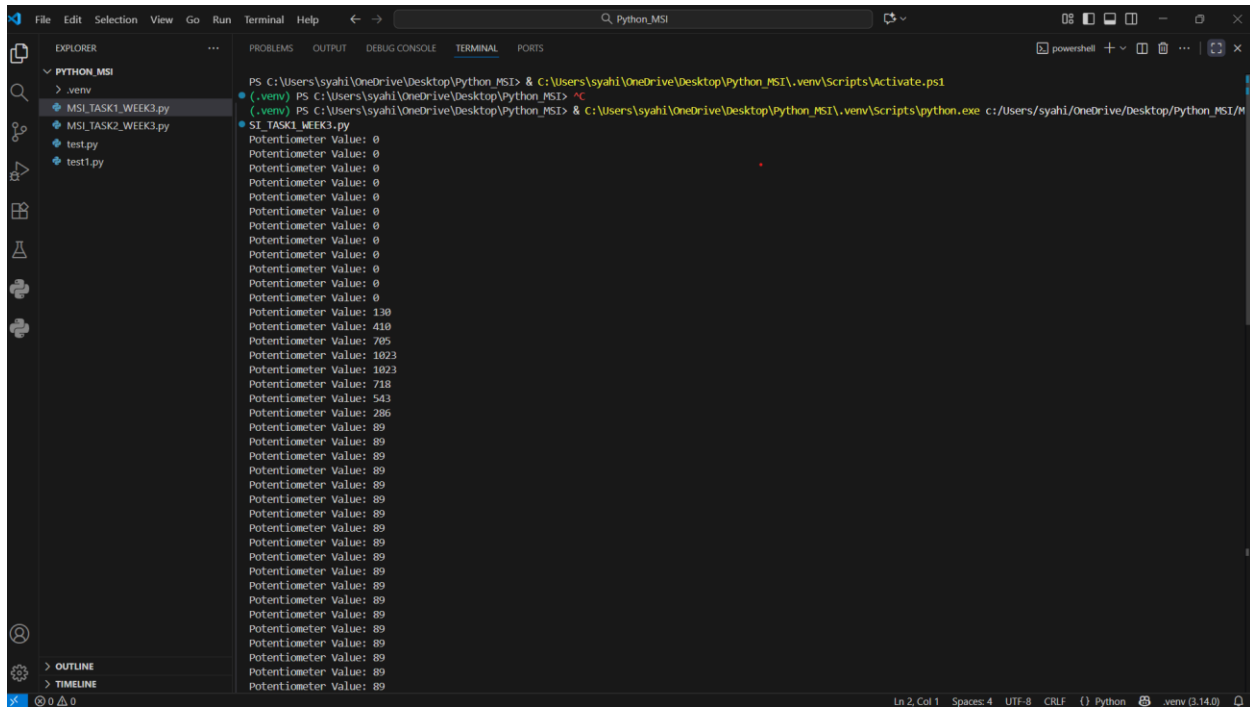


Figure G: Python terminal using VS code(Task 1)

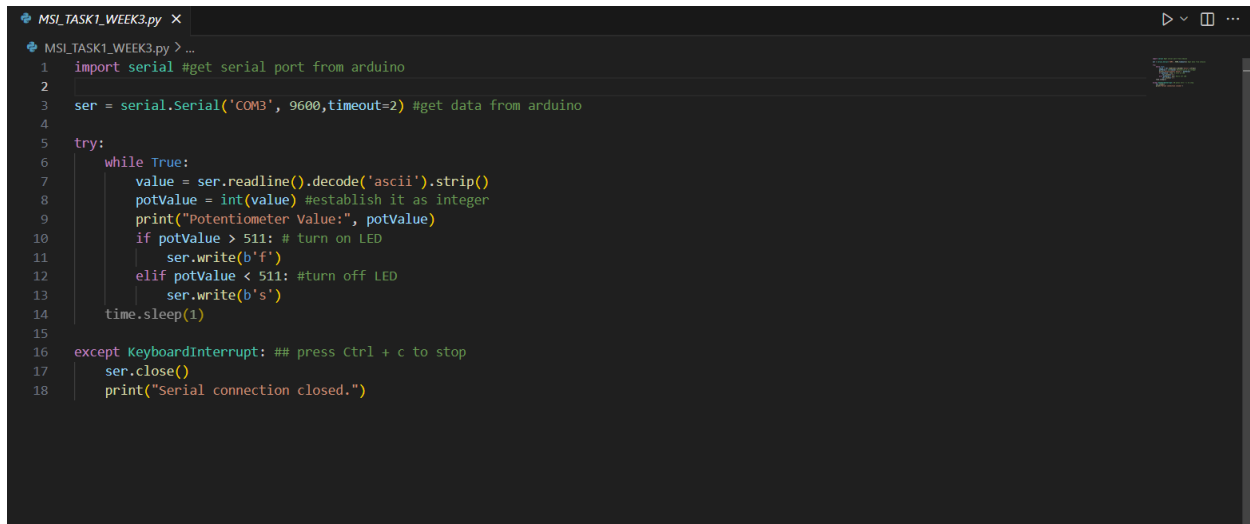


Figure H: Python code using VS code (Task 1)

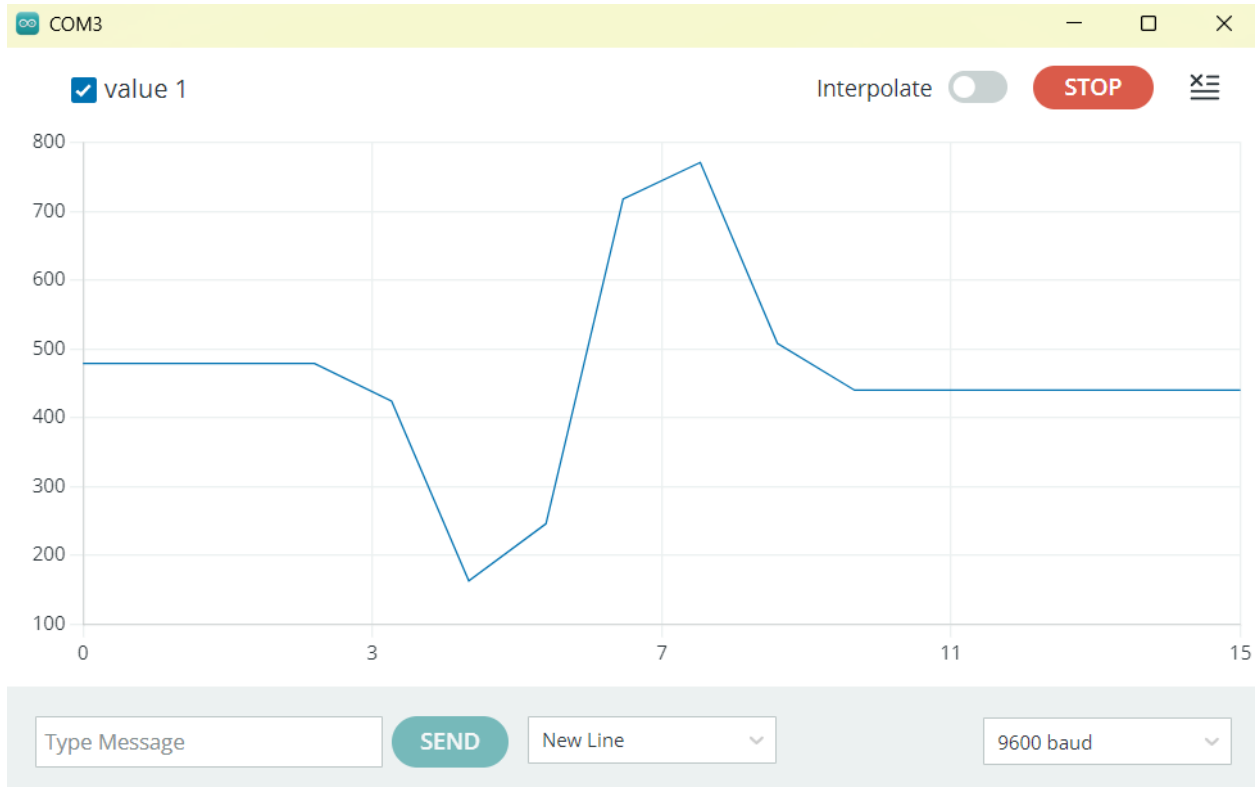


Figure I: Serial plotter using arduino IDE (Task 1)

```
CODE_WEEK_3_TASK_1_ARDUINO | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Arduino Uno
CODE_WEEK_3_TASK_1_ARDUINO.ino
1 const int LEDPIN = 9;
2 char command;
3 void setup()
4 {
5   pinMode(LEDPIN, OUTPUT);
6   Serial.begin(9600);
7 }
8 void loop()
9 {
10  int potValue = analogRead(A0);
11  command = Serial.read();
12  Serial.println(potValue);
13  if(command == 'f')
14  {
15    digitalWrite(LEDPIN, HIGH);
16  }
17  else if(command == 's')
18  {
19    digitalWrite(LEDPIN, LOW);
20  }
21  delay(1000);
22 }
23

Output
"C:\Users\syahi\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-gcc" -w -Os -g -fno-fuse-linker-plugin -Wl,--gc-sections -mmcu=atmega328p
"C:\Users\syahi\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-objcopy" -O ihex -j .eeprom --set-section-flags=.eeprom=alloc,load --no-char
"C:\Users\syahi\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-objcopy" -O ihex -R .eeprom "C:\Users\syahi\AppData\Local\arduino\sket
"C:\Users\syahi\AppData\Local\Arduino15\packages\arduino\tools\avr-gcc\7.3.0-atmel3.6.1-arduino7/bin/avr-size" -A "C:\Users\syahi\AppData\Local\arduino\sketches\A54D9DD1AD90E
Sketch uses 2208 bytes (6%) of program storage space. Maximum is 32256 bytes.
Global variables use 189 bytes (9%) of dynamic memory, leaving 1859 bytes for local variables. Maximum is 2048 bytes.
```

Figure J: Arduino code using arduino IDE (Task 1)

Appendix B

```

1 import serial #using serial port
2 import matplotlib.pyplot as plt #to plot the graph
3 import time #for delay in coding
4
5 ser = serial.Serial('COM3', baudrate = 9600, timeout=2) #to get information from arduino
6 plt.ion() #real time plotting
7 fig, ax = plt.subplots() #function for plots
8 x_value, y_value = [], [] #storing the data into x and y value
9
10 try: #used to ignore error and execute command
11     while True:
12         ser.write(b'f') #send byte 'f' to arduino
13         arduinoData = ser.readline().decode('ascii').strip()
14         if arduinoData:
15             data = arduinoData.split(',') #split data from ',' to two data
16             if len(data) == 2:
17                 potValue = int(data[0]) #potentiometer data
18                 x_value.append(len(x_value)) #assign data to x value
19                 y_value.append(int(potValue)) #assign data to y value
20                 angle = int(data[1]) #servo data
21
22                 print(f"Potentiometer: {potValue} | Servo Position: {angle}") #print into 2 separate data
23
24                 ax.clear() #clear the plot
25                 ax.plot(x_value, y_value) #plot
26                 plt.title("Potentiometer Reading")
27                 plt.xlabel("Integer")
28                 plt.ylabel("Potentiometer Value")
29                 plt.pause(0.1)
30             time.sleep(1) #allow the code to run smoother and slower
31
32 except KeyboardInterrupt: #this function will happen if 'Ctrl + c' is pressed
33     ser.write(b's') #send byte 's' to arduino
34     ser.close()
35     plt.ioff()
36     plt.show()
37     print("Serial connection closed.")

```

Figure K: Python code using VS code (Task 2)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Potentiometer: 857 | Servo Position: 150
Potentiometer: 856 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 856 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 856 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 856 | Servo Position: 150
Potentiometer: 857 | Servo Position: 150
Potentiometer: 856 | Servo Position: 150

```

Figure L: Python terminal using VS code (Task 2)

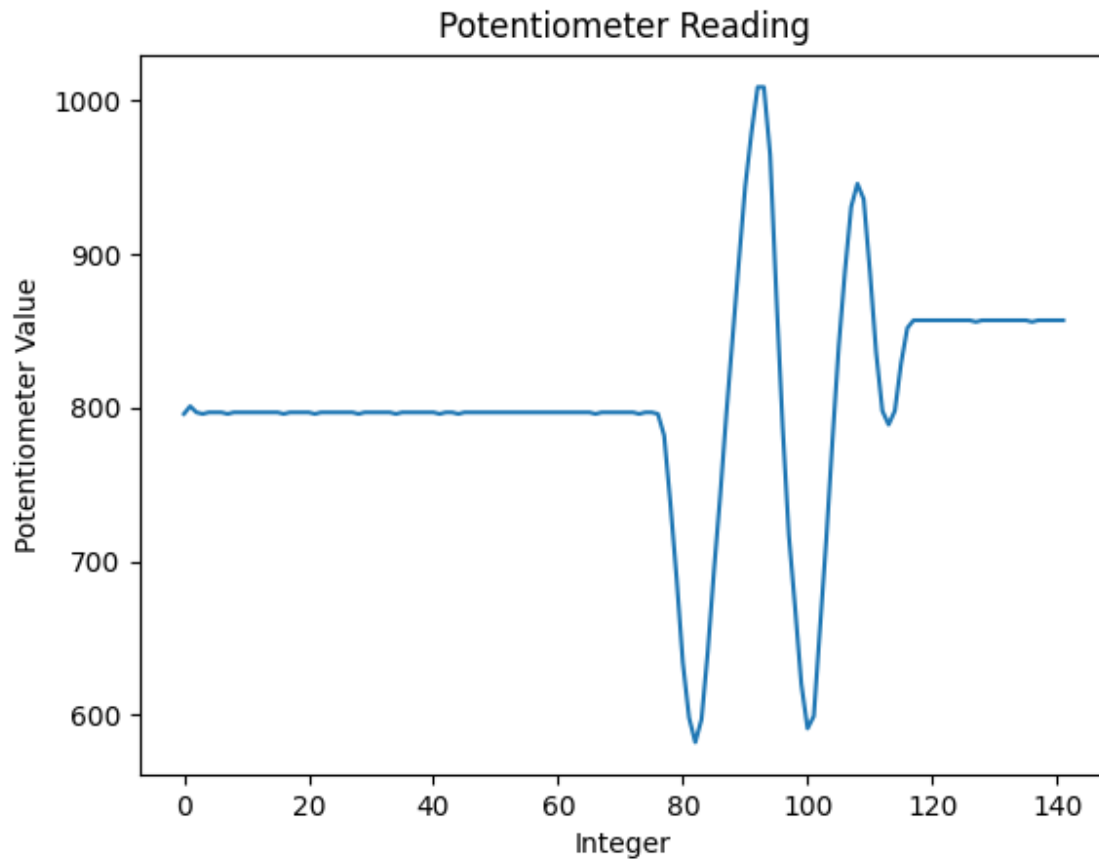


Figure M: Python plot graph using VS code (Task 2)

```
CODE_WEEK_3_ARDUINO.ino
1  #include <Servo.h>
2  Servo myservo;
3  const int potpin = A0;
4  const int LEDPIN = 7;
5  const int servopin = 9;
6  int potValue = 0;
7  int angle = 0;
8  char command;
9  void setup()
10 {
11   pinMode(LEDPIN, OUTPUT);
12   myservo.attach(servopin);
13   myservo.write(angle);
14   delay(2000);
15   Serial.begin(9600);
16 }
17
18 void loop()
19 {
20   if(Serial.available() > 0)
21   {
22     command = Serial.read(); //get info from python
23     if(command == 'f') //if python return 'f'
24     {
25       potValue = analogRead(potpin);
26       angle = map(potValue, 0, 1023, 0, 180);
27       myservo.write(angle);
28       Serial.print(potValue); //send potentiometer to python
29       Serial.print(",");
30       Serial.println(angle); //send angle to python in new line
31       if(potValue > 339)
32       {
33         digitalWrite(LEDPIN, HIGH);
34       }
35     }
36     else
37     {
38       digitalWrite(LEDPIN, LOW);
39       delay(100);
40     }
41     else if (command == 's') //if python return 's'
42     {
43       while(true) //endless looping until arduino reset
44       {
45       }
46     }
47   }
48 }
```

Figure N: Arduino code using arduino IDE (Task 2)


13.0 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: 	Read	/
Name: AHMAD HARITH IMRAN BIN MOHD YUSOF	Understand	/
Matric No.: 2311581	Agree	/

Contribution: Assembler, abstract, table of content, introduction, results and conclusion.

Signature: 	Read	/
Name: MOHAMMAD FARISH ISKANDAR BIN MOHD SYAHIDIN	Understand	/
Matric No.: 2311779	Agree	/
Contribution: Coder, experimental setup, data collection, data analysis, references, appendices.		

Signature: 	Read	/
Name: ARIFA AQILAH BINTI ABDUL HALIM	Understand	/
Matric No.: 2311530	Agree	/
Contribution: Assembler, material and equipment, methodology, discussion, recommendation.		