Practical Robotics Projects with Arduino

(CSE 4571)

Lab Assignment No – 04

ULTRASONIC & IR SENSING

Submission Date: _____

Branch: CSE		**Section:**
Name	**Registration No.**	**Signature**

Department of Computer Science and Engineering
Institute of Technical Education and Research (Faculty of Engineering)
Siksha 'O' Anusandhan (Deemed to be University)
Bhubaneswar, Odisha-751030.

Aim:

Ultrasonic & IR Sensing – To Interface HC-SR04 and IR sensors with Arduino UNO for distance measurement and obstacle detection.

Objectives:

1) **To study the working principle of Ultrasonic (HC-SR04) and Infrared (IR) sensors.**
2) **To interface HC-SR04 ultrasonic sensor with Arduino UNO for distance measurement and display the output values on I2C LCD.**
3) **To interface IR sensor with Arduino Uno for obstacle detection using buzzer and LED.**
4) **To validate distance measurement of HC-SR04 ultrasonic sensor using LEDs as indicators.**

Answers to Pre-Lab Questions

A. Experiment-Specific

1. What is the working principle of the HC-SR04 ultrasonic sensor for distance measurement?
2. Mention the function of Trigger pin and Echo pin in the HC-SR04 sensor.
3. Write the formula to calculate distance from the ultrasonic sensor using the speed of sound.
4. What is the typical range of distance measured by an HC-SR04 ultrasonic sensor?
5. What is the working principle of an Infrared (IR) sensor used for obstacle detection?
6. Differentiate between active IR sensor and passive IR sensor with examples.
7. State any two applications of ultrasonic sensors in real life.
8. Why is it necessary to use pinMode() and digitalWrite() functions in Arduino while interfacing sensors?
9. How does an IR sensor distinguish between the presence and absence of an obstacle?
10. Mention one key advantage and one limitation of ultrasonic sensors compared to IR sensors?

Components/Equipment Required:

Sl. No.	Name of the Component / Equipment	Specification	Quantity
1)	Arduino UNO R3	16MHz	1
2)	Arduino UNO cable	USB Type A to Micro-B	1
3)	Ultrasonic sensor HC-SR04		1
4)	IR sensor module	Obstacle detection	1
5)	I2C LCD		1
6)	Buzzer		1
7)	Resistors (carbon type)	220Ω / 330Ω	2
8)	LED	Any colour of your choice	8
9)	Breadboard	840 Tie points	1
10)	Jumper Wire	----------	As per requirement

Objective 2

To interface HC-SR04 ultrasonic sensor with Arduino UNO for distance measurement and display the output values on I2C LCD.

Circuit / Schematic Diagram

Figure 1: *HC-SR04 Ultrasonic Sensor Pinout*

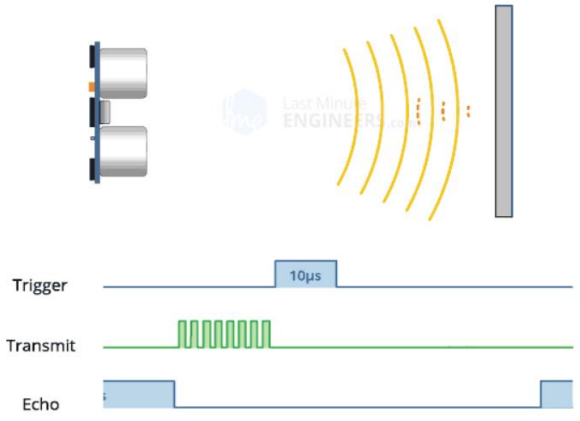


Figure 2: *Work of HC-SR04*

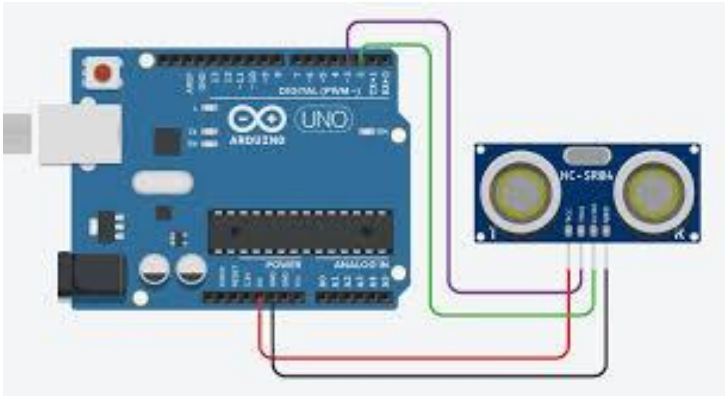


Figure 3: *Ultrasonic Distance measurement circuit using HC-SR04 & Arduino Uno*

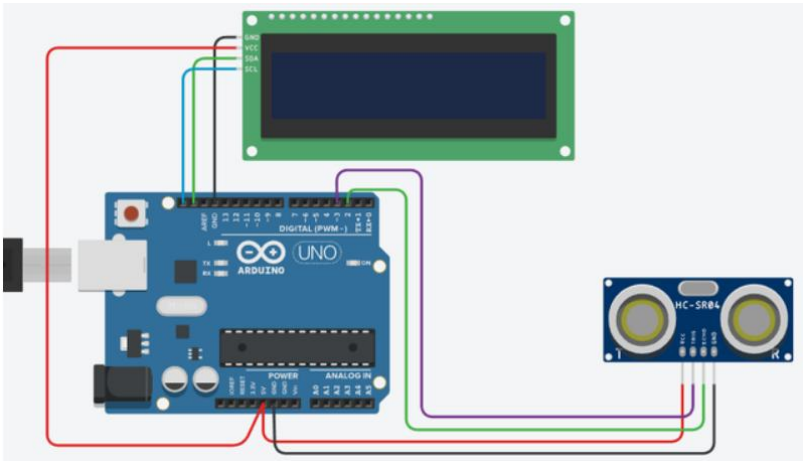


Figure 4: *Ultrasonic Distance Measurement Output in LCD Display*

Code

Write an Arduino program to interface HC-SR04 ultrasonic sensor with Arduino Uno for distance measurement and display the output value in I2C LCD.

```
#include <Wire.h>

#include <LiquidCrystal_I2C.h>

#define trigPin 9

#define echoPin 10

LiquidCrystal_I2C lcd(0x27, 16, 2);  // I2C address 0x27, adjust if needed

void setup() {

 lcd.init();

 lcd.backlight();

 pinMode(trigPin, OUTPUT);
```

```
  pinMode(echoPin, INPUT);
  lcd.setCursor(0, 0);
  lcd.print("Distance Meter");
  delay(1000);
  lcd.clear();
}

void loop() {
  long duration;
  float distance;
  // Send a 10µs pulse to trigger pin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
// Read echo pin and calculate distance
  duration = pulseIn(echoPin, HIGH);
  distance = duration * 0.0343 / 2; // cm
 // Handle out-of-range
  if (distance > 400 || distance < 2) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Out of range");
  } else {
    lcd.clear();
```

```
  lcd.setCursor(0, 0);
  lcd.print("Distance:");
  lcd.setCursor(0, 1);
  lcd.print(distance, 1);
  lcd.print(" cm");
 }
delay(500); }
```

Observation

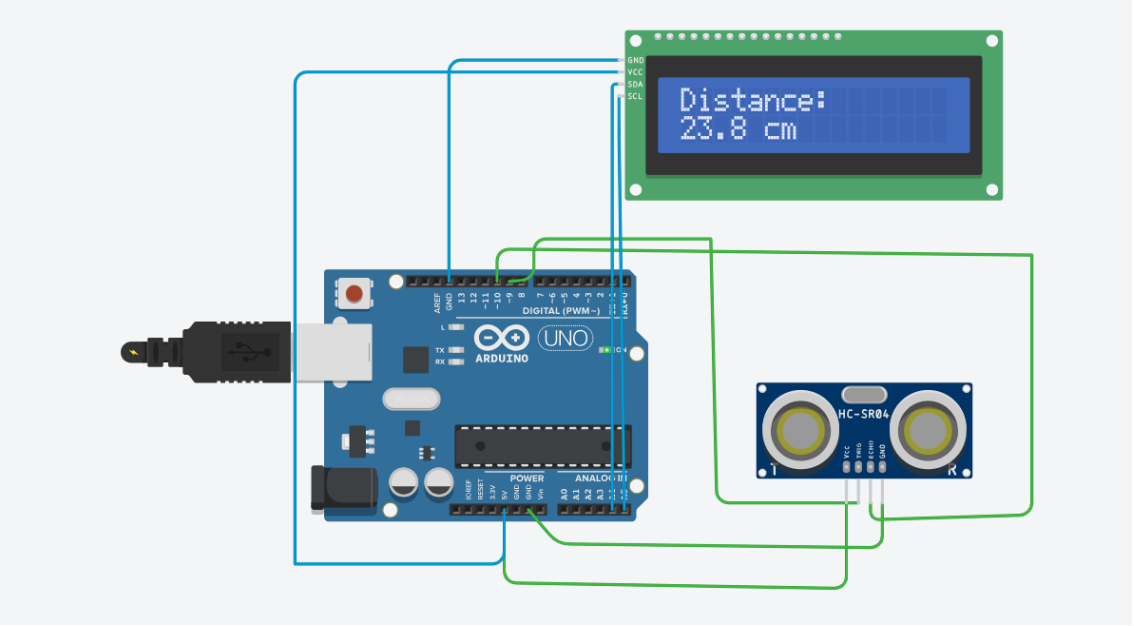


Figure 5: (Simuation based distance measurement using HC-SR04 ultrasonic sensor and output display in I2C LCD)

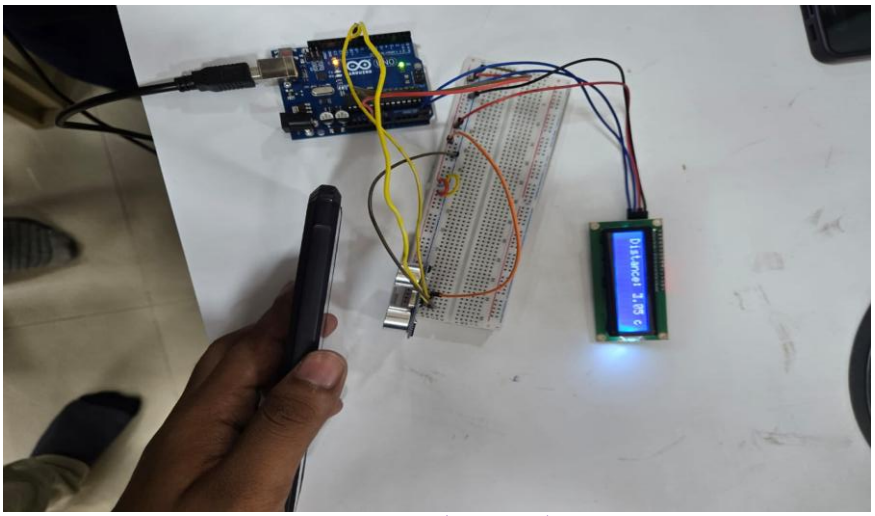


Figure 6: (Hardware Implementation based distance measurement using HC-SR04 ultrasonic sensor and output display in I2C LCD)

Objective 3

To interface IR sensor with Arduino Uno for obstacle detection using buzzer and LED.

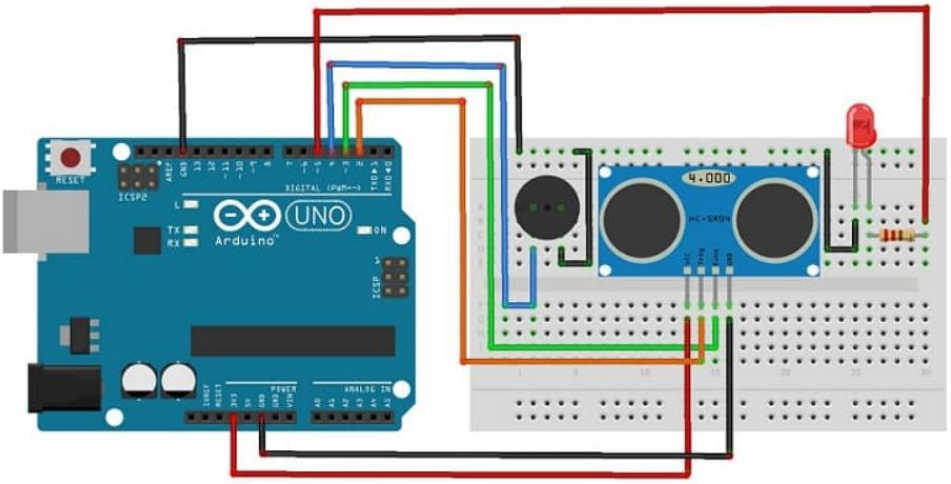


Figure 7: *Interface IR sensor with Arduino UNO for obstacle detection*

Code

Write an Arduino program to detect an obstacle in front, and turn ON the LED and buzzer.

```
#define TRIG_PIN    2
#define ECHO_PIN    3
#define LED_PIN     5
#define BUZZER_PIN  4

long duration;
int distance;

void setup() {
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);

  Serial.begin(9600);
}

void loop() {
  // Clear trigger
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);

  // Send 10 µs pulse
  digitalWrite(TRIG_PIN, HIGH);
```

```
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // Read echo
  duration = pulseIn(ECHO_PIN, HIGH);

  // Convert to cm
  distance = duration * 0.034 / 2;

  Serial.print("Duration: ");
  Serial.print(duration);
  Serial.print("\tDistance: ");
  Serial.print(distance);
  Serial.println(" cm");

  // Buzzer and led activation when distance < 30
  if (distance > 0 && distance <= 30) {
    digitalWrite(LED_PIN, HIGH);
    digitalWrite(BUZZER_PIN, HIGH);
  } else {
    digitalWrite(LED_PIN, LOW);
    digitalWrite(BUZZER_PIN, LOW);
  }

  delay(500);
}
```

Observation

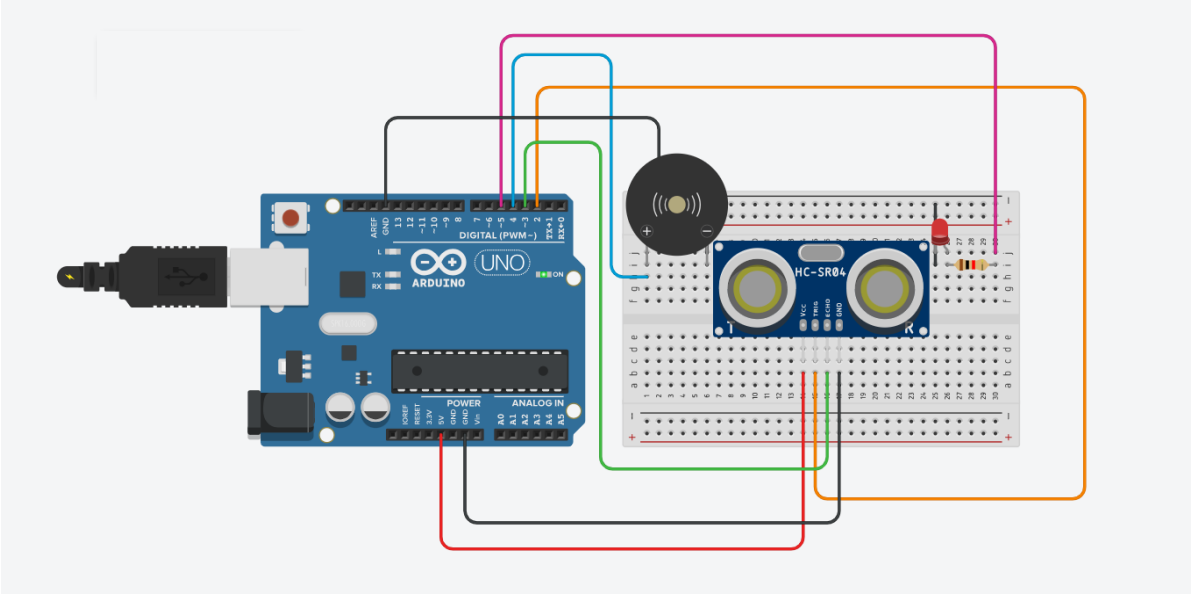


Figure 8: (*Simulation based interfacing IR sensor and buzzer with Arduino UNO for obstacle detection*)

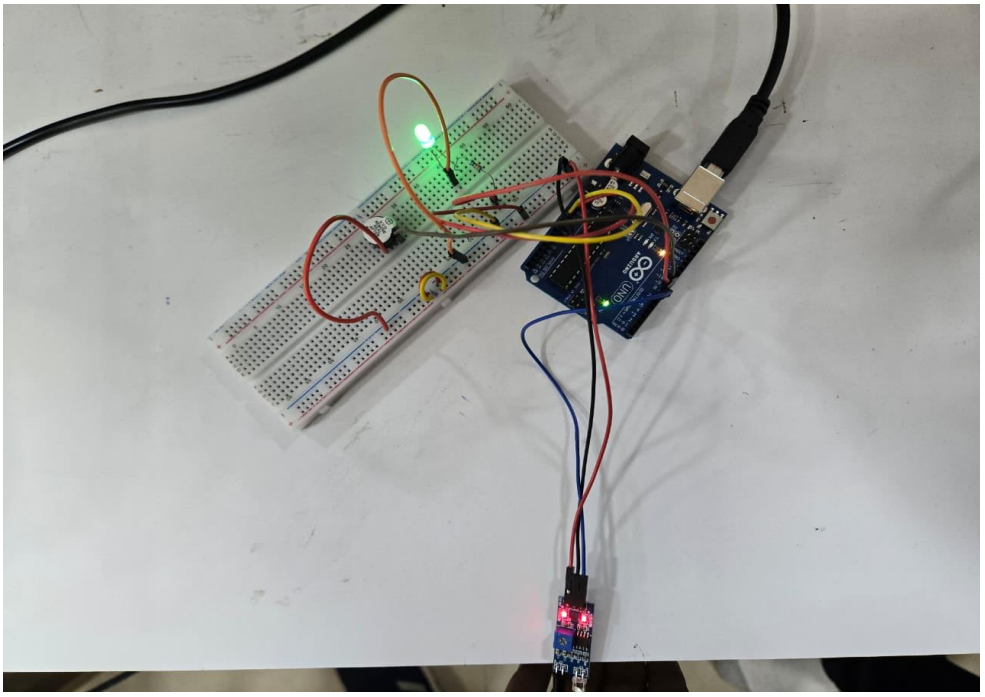


Figure 9: (*Hardware Implementation of interfacing IR sensor and buzzer with Arduino UNO for obstacle detection*)

Objective 4

To validate distance measurement of HC-SR04 ultrasonic sensor using LEDs as indicators.

Circuit / Schematic Diagram

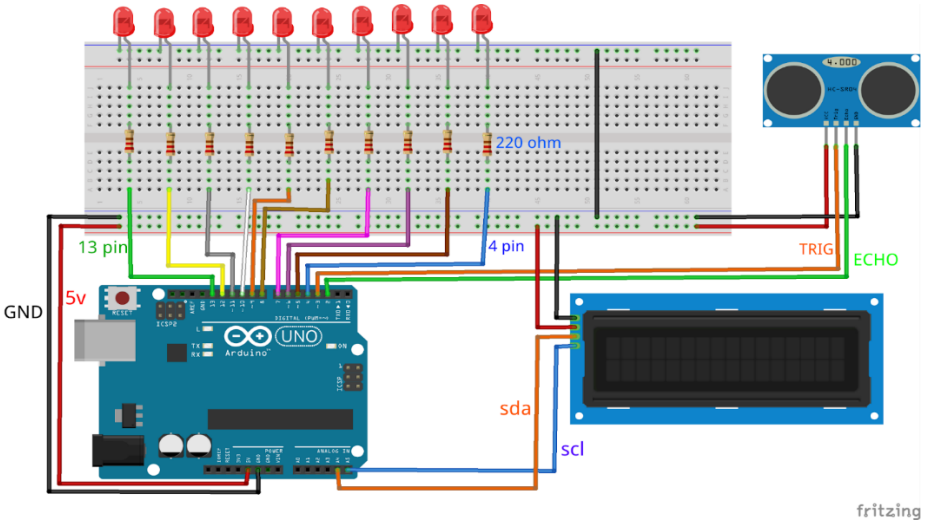


Figure 10: *HC-SR04 + Arduino Uno +LCD+LED*

Code

Write an Arduino program to integrate an ultrasonic sensor, an I2C LCD display, and LED indicators for real-time distance measurement and visualization.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#define TRIG_PIN 3
#define ECHO_PIN 2

// LEDs from pins 4–13
int leds[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
const int ledCount = 10;

// Initialize I2C LCD (usually address 0x27 or 0x3F)
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  Serial.begin(9600);

  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // setup all LEDs
  for (int i = 0; i < ledCount; i++) {
    pinMode(leds[i], OUTPUT);
  }

  // initialize LCD
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Distance Sensor");
  delay(1000);
}

void loop() {
  long duration;
  float distance;

  // Trigger pulse
  digitalWrite(TRIG_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);
```

```
  // Read echo pulse duration
  duration = pulseIn(ECHO_PIN, HIGH, 30000);

  // Convert to cm
  distance = duration * 0.034 / 2;

  // Safety check: ignore invalid readings
  if (distance == 0 || distance > 400) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Out of range   ");
    Serial.println("Out of range");
  } else {
    // Print distance
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Distance:");
    lcd.setCursor(10, 0);
    lcd.print(distance, 1);
    lcd.print("cm");

    Serial.print("Distance: ");
    Serial.print(distance);
    Serial.println(" cm");
  }

  // LED visualization (equal gaps from 4 cm to 320 cm)
  int numLEDs = map(distance, 4, 320, 0, ledCount);
  numLEDs = constrain(numLEDs, 0, ledCount);

  for (int i = 0; i < ledCount; i++) {
    if (i < numLEDs) {
      digitalWrite(leds[i], HIGH);
    } else {
      digitalWrite(leds[i], LOW);
    }
  }

  delay(300);
}
```

Observation

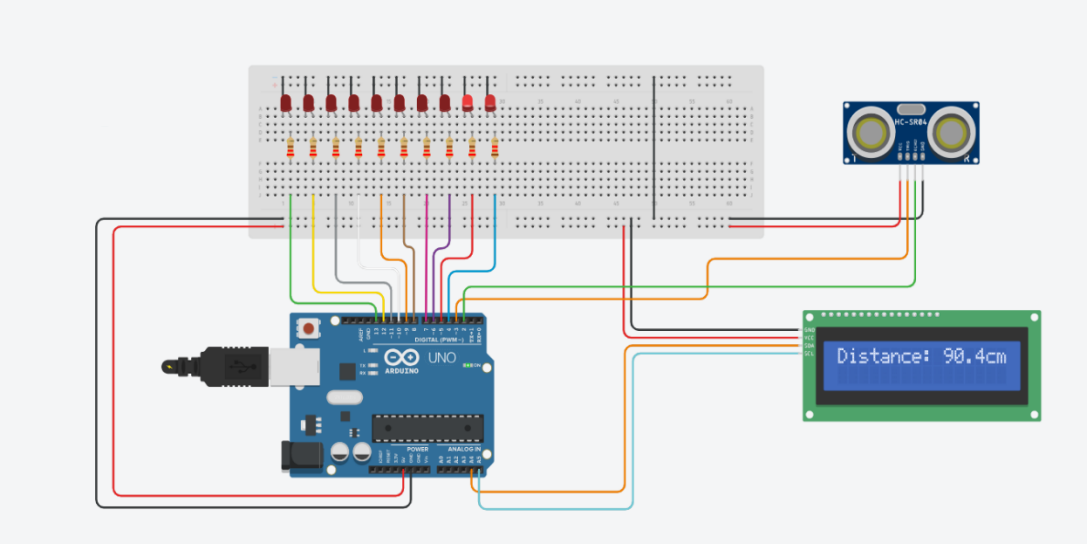


Figure 11: (*Software Implementation of interfacing ultrasonic sensor, an I2C LCD display, and LED indicators*)

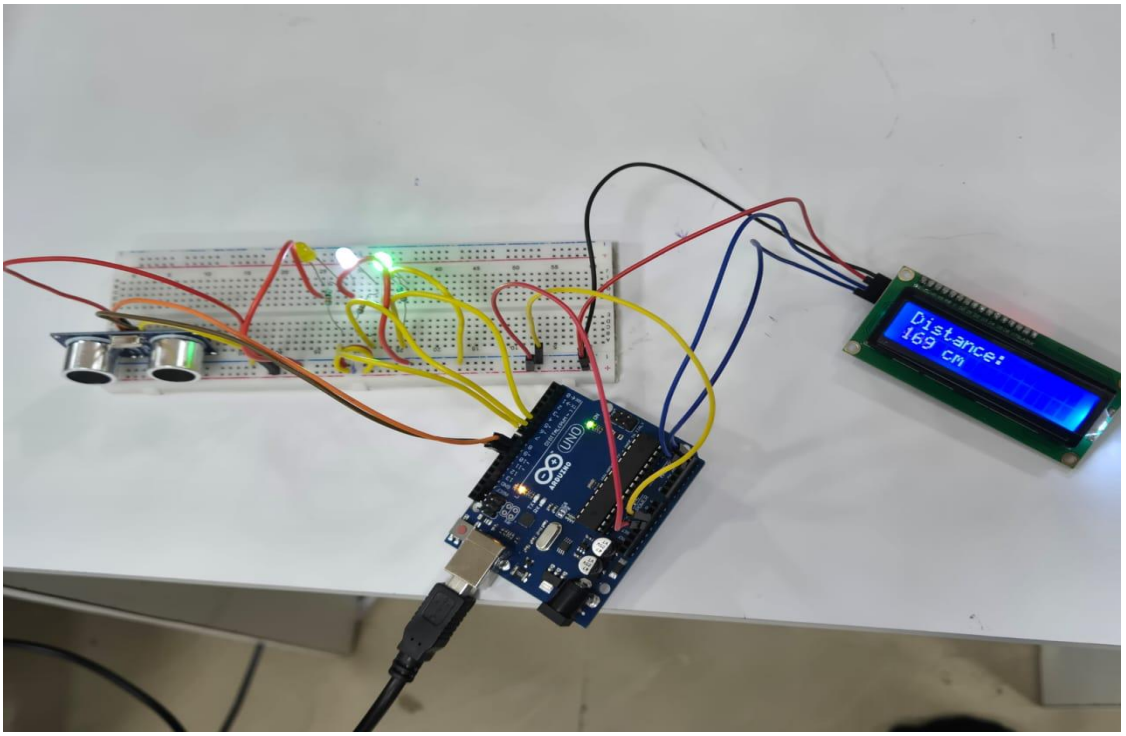


Figure 12: (*Hardware Implementation of interfacing ultrasonic sensor, an I2C LCD display, and LED indicators*)

Conclusion:

Precautions:

Post Experiment Questionnaire:

A. Experiment-Specific (LED Blinking & LED Patterns)

1. What happens if the Trigger pin of the HC-SR04 ultrasonic sensor is not given a proper 10 µs pulse?
2. If the Echo pulse duration is measured as 2 ms, calculate the distance of the obstacle. (Speed of sound = 343 m/s)
3. Why do ultrasonic sensors measure distance more accurately than IR sensors in outdoor conditions?
4. What would be the effect of bright sunlight on the IR sensor's performance?
5. How can you modify the Arduino code to turn ON an LED when an obstacle is detected within 15 cm using the HC-SR04 sensor?
6. If the ultrasonic sensor reads 200 cm, what should be the approximate Echo pulse duration?
7. Why is it necessary to divide the total sound travel time by 2 in ultrasonic distance measurement?
8. What logic output does the IR obstacle sensor provide when an object is detected?
9. Mention one limitation of using only IR sensors for obstacle detection in mobile robots.
10. Suggest a practical application where both ultrasonic and IR sensors are used together for better performance.?

(Signature of the Faculty)	**(Signature of the Student)**
	Name:
Date:	**Registration No.:**
	Branch:
	Section

Practical Robotics Projects with Arduino

(CSE 4571)

Lab Assignment No – 05

Temperature Monitoring

Submission Date: _____

Branch: CSE		**Section:**
Name	**Registration No.**	**Signature**

Department of Computer Science and Engineering
Institute of Technical Education and Research (Faculty of Engineering)
Siksha 'O' Anusandhan (Deemed to be University)
Bhubaneswar, Odisha-751030.

Aim:

Analog to Digital Converter - To interface an analog temperature sensor (LM35) and a digital temperature sensor (DHT11) with an Arduino Uno, utilizing the Arduino's ADC to read and convert the analog output from the LM35 and digital communication protocols to read data from the DHT11, thereby enabling accurate measurement and display of ambient temperature in degrees Celsius.

Objectives:

1) To set up and connect the LM35 analog temperature sensor to the Arduino, read the analog voltage using the ADC, and convert it into a temperature value in degrees Celsius through the Arduino program.
    1.1) Understand the working principle of the LM35 temperature sensor, which provides an output voltage proportional to temperature (10 mV/°C).
    1.2) Interface the LM35 sensor with the Arduino Uno by connecting its output pin to an analog input pin .
    1.3) Utilize the Arduino's 10-bit Analog-to-Digital Converter (ADC) to read the analog voltage using the analogRead() command.
    1.4) Convert the analog reading into temperature in degrees Celsius using the formula:

$$T(°C)=(AnalogValue\times 5.0\times 100)/1024$$

    1.5) Display the calculated temperature values on the Serial Monitor for observation and verification.
2) To connect the DHT11 digital temperature sensor to the Arduino, interface it via a digital input pin, and write a program to communicate with the sensor and read temperature data.
    2.1) Understand the digital communication protocol used by the DHT11 sensor for transmitting temperature and humidity data.
    2.2) Connect the DHT11 data pin to a digital input pin of the Arduino Uno.
    2.3) Implement the DHT library functions in the Arduino sketch to initialize and read digital temperature data.
    2.4) Display the measured temperature values from the DHT11 on the Serial Monitor for comparison and validation.
3) To display temperature readings from both LM35 and DHT11 sensors on the serial monitor or an LCD display.
    3.1) Develop an Arduino program capable of acquiring data from both analog (LM35) and digital (DHT11) sensors simultaneously.
    3.2) Format the serial output to clearly distinguish between LM35 and DHT11 temperature readings.
    3.3) Optionally interface a 16×2 or OLED display to present temperature readings without the need for a computer.
    3.4) Ensure accurate and real-time data display for both sensors under varying environmental conditions.
4) To compare temperature measurements from the LM35 and DHT11 sensors and observe any differences.

4.1) Record and analyze temperature data from both sensors under identical environmental conditions.

4.2) Observe variations in readings due to sensor characteristics, calibration, or response time.

4.3) Interpret differences to understand the performance, sensitivity, and accuracy of analog versus digital temperature sensors.

4.4) Conclude the experiment by summarizing the comparative behavior and reliability of both LM35 and DHT11 sensors.

Pre-Lab Questionnaire:

1) What is the function of the Analog-to-Digital Converter (ADC) in Arduino Uno?
2) What is the range of digital values that the analogRead() function can output in Arduino Uno?
3) What is the voltage range that can be read by the Arduino Uno's ADC?
4) How is the analog voltage from the LM35 sensor related to temperature in degrees Celsius?
5) Write the formula used to convert the ADC reading from the LM35 sensor into temperature.
6) What type of signal does the DHT11 sensor provide to the Arduino — analog or digital?
7) Which library is used in Arduino to read data from the DHT11 sensor?
8) What parameters can be measured using the DHT11 sensor?
9) Why might temperature readings from the LM35 and DHT11 sensors differ slightly?
10) How can temperature readings from both LM35 and DHT11 be displayed on the Serial Monitor or LCD?

Answers to Pre-Lab Questions

Components/Equipment Required:

Sl. No.	Name of the Component / Equipment	Specification	Quantity
1)	Arduino UNO R3	16MHz	1
2)	Arduino UNO Cable	USB Type A to Micro-B	1
3)	DHT11 Sensor	3–5 V, single-wire digital	1
4)	LM35 Sensor	10 mV/°C analog, −55 to 150°C	1
5)	16×2 I2C LCD	I2C backpack (PCF8574), 5 V	1
6)	Breadboard	≥ 400 tie-points	1
7)	Resistors (for LM35 wiring if needed)	10 kΩ (pull-down optional)	1
8)	Jumper Wire	M-M / M-F	As per requirement
9)	10 kΩ (pull-down optional)	5 V regulated	1

Objective 1

To set up and connect the LM35 analog temperature sensor to the Arduino, read the analog voltage using the ADC, and convert it into a temperature value in degrees Celsius through the Arduino program.

Code

```
int lm35Pin = A0;
float tempC;
void setup() {
  Serial.begin(9600);
}
void loop() {
  int analogValue = analogRead(lm35Pin);
  tempC = (analogValue * 5.0 * 100.0) / 1024.0;
  Serial.print("LM35 Temperature: ");
  Serial.print(tempC);
  Serial.println(" °C");
  delay(1000);
}
```

Circuit / Schematic Diagram

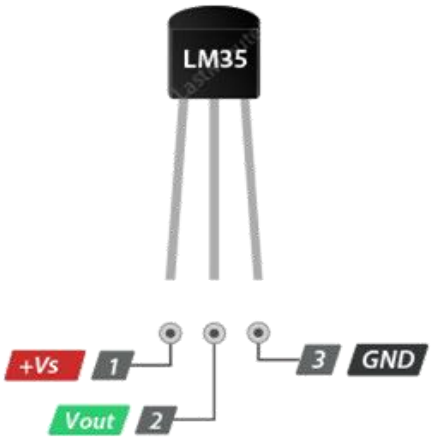


Figure 1: LM-35 Analog Temperature Senor Pinout

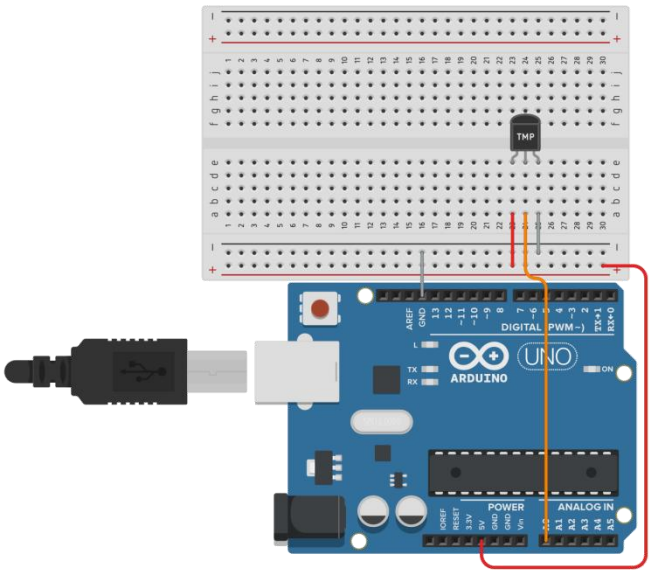


Figure 2: Analog Temperature Sensing Circusing

Observation :

S.No.	Analog Value (ADC Reading)	Calculated Voltage (V)	Temperature (°C)
1			
2			
3			
4			
5			

Figure 3: (Simulation based temperature measurement using LM-35 Analog Temperature Sensor and output display in Serial Monitor)

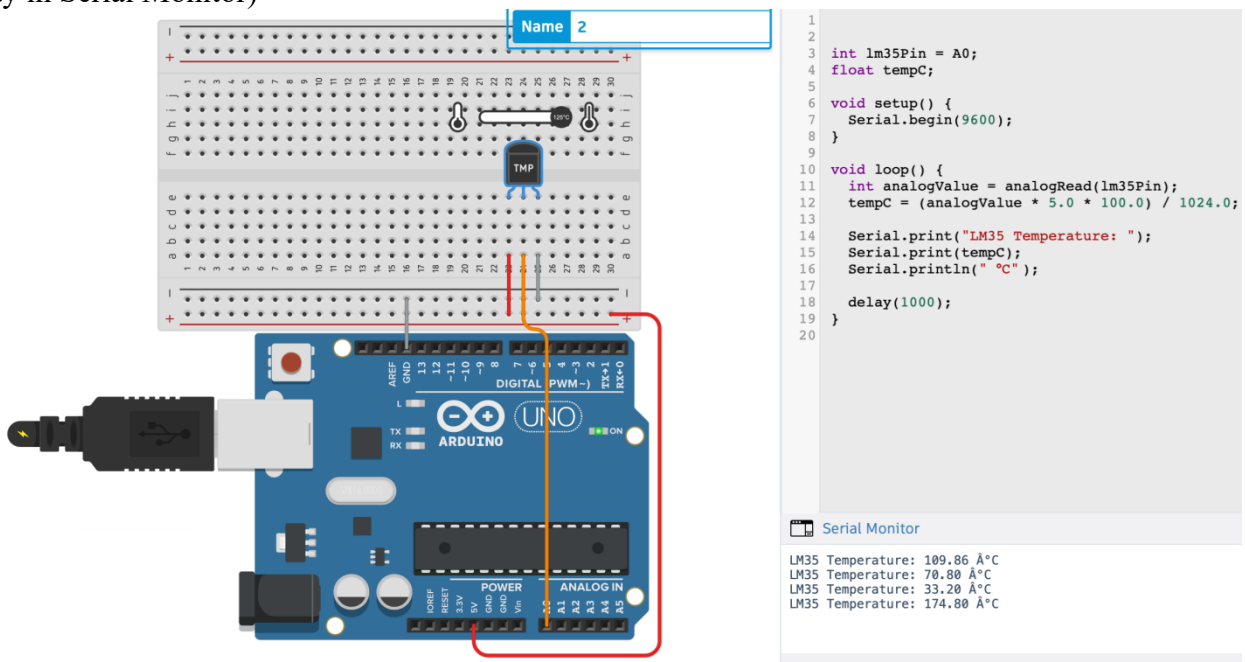
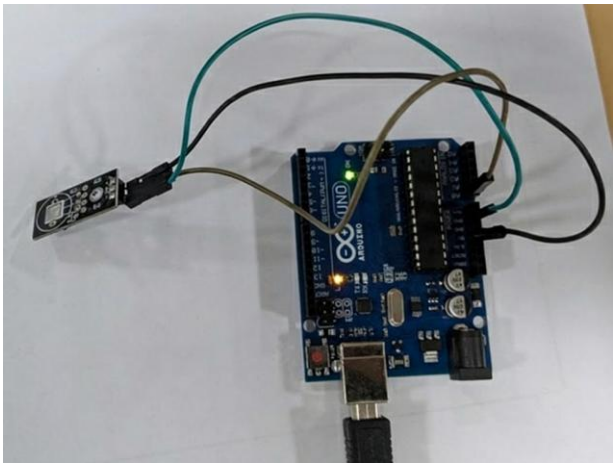


Figure 4: (Hardware Implementation based temperature measurement using LM-35 Analog Temperature Sensor and output display in Serial Monitor)



```
Analog Value (ADC): 50  Voltage (V): 0.24       Temperature (°C): 24.44
Analog Value (ADC): 43  Voltage (V): 0.21       Temperature (°C): 21.02
Analog Value (ADC): 55  Voltage (V): 0.27       Temperature (°C): 26.88
Analog Value (ADC): 48  Voltage (V): 0.23       Temperature (°C): 23.46
Analog Value (ADC): 38  Voltage (V): 0.19       Temperature (°C): 18.57
Analog Value (ADC): 61  Voltage (V): 0.30       Temperature (°C): 29.81
Analog Value (ADC): 44  Voltage (V): 0.22       Temperature (°C): 21.51
Analog Value (ADC): 48  Voltage (V): 0.23       Temperature (°C): 23.46
Analog Value (ADC): 45  Voltage (V): 0.22       Temperature (°C): 21.99
Analog Value (ADC): 40  Voltage (V): 0.20       Temperature (°C): 19.55
Analog Value (ADC): 26  Voltage (V): 0.13       Temperature (°C): 12.71
Analog Value (ADC): 26  Voltage (V): 0.13       Temperature (°C): 12.71
```

Objective 2

To connect the DHT11 digital temperature sensor to the Arduino, interface it via a digital input pin, and write a program to communicate with the sensor and read temperature data.

Circuit / Schematic Diagram

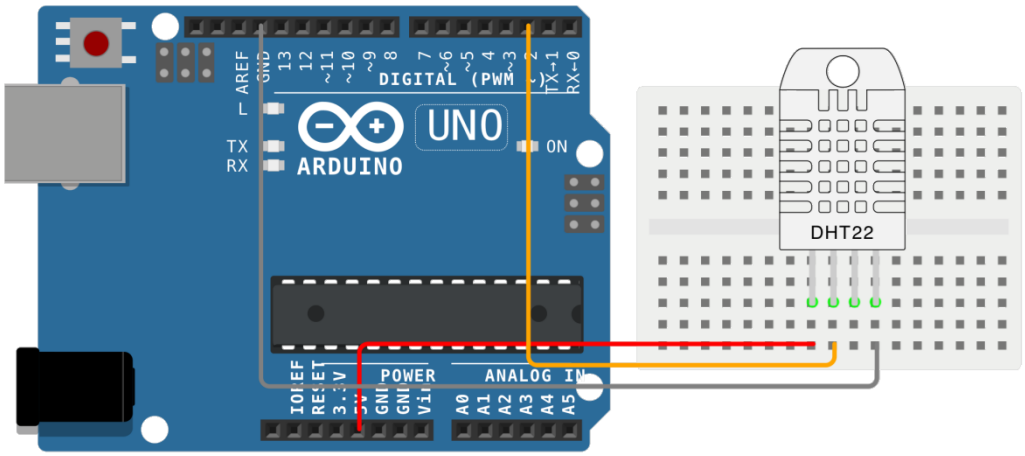


Figure 5: DHT11 digital temperature sensor with Arduino UNO to read temperature data.

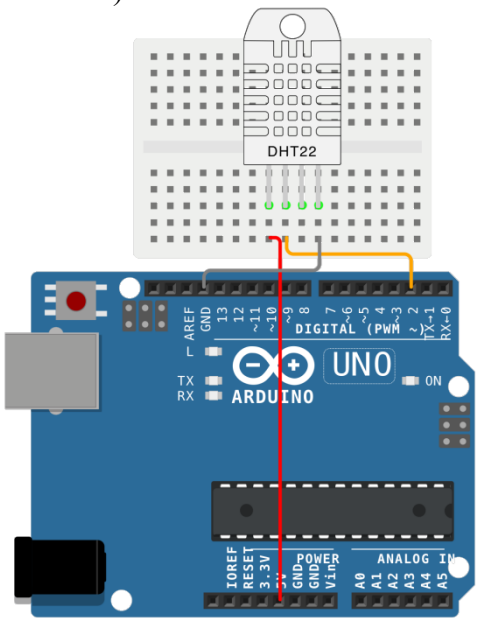
Code

```
#include "DHT.h"
#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  dht.begin();
}
void loop() {
  float temp = dht.readTemperature();
  if (isnan(temp)) {
    Serial.println("DHT22 read failed!");
    return;
  }
  Serial.print("DHT22 Temperature: ");
  Serial.print(temp);
  Serial.println(" °C");
  delay(1000);
}
```

Observation:

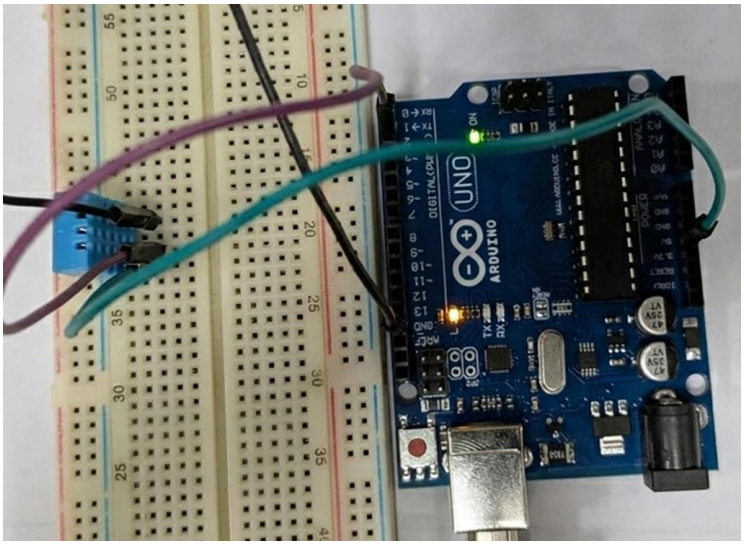
S.No.	Temperature (°C)	Humidity (%)	Conditions
1			Normal room condition
2			Slightly warmer
3			Moderate temperature
4			Slight cooling
5			Warmer environment

Figure 6: (Simulation based temperature measurement using DHT11 Digital Temperature Sensor and output display in Serial Monitor)




```
DHT22 Temperature: 53.50 °C
DHT22 Temperature: 28.40 °C
DHT22 Temperature: 28.40 °C
DHT22 Temperature: 74.10 °C
DHT22 Temperature: 74.10 °C
DHT22 Temperature: -40.00 °C
DHT22 Temperature: -40.00 °C
```

Figure 7: Hardware Implementation based temperature measurement using DHT11 Digital Temperature Sensor and output display in Serial Monitor



S.No.	Temperature (°C)	Humidity (%)
1	25.8	39.0
2	26.0	39.0
3	26.1	39.0
4	26.2	39.0
5	26.3	38.0
6	26.3	38.0
7	26.4	38.0
8	26.4	37.0
9	26.4	37.0
10	26.4	37.0

Objective 3

To display temperature readings from both LM35 and DHT11 sensors on the serial monitor or an LCD display.

Circuit / Schematic Diagram

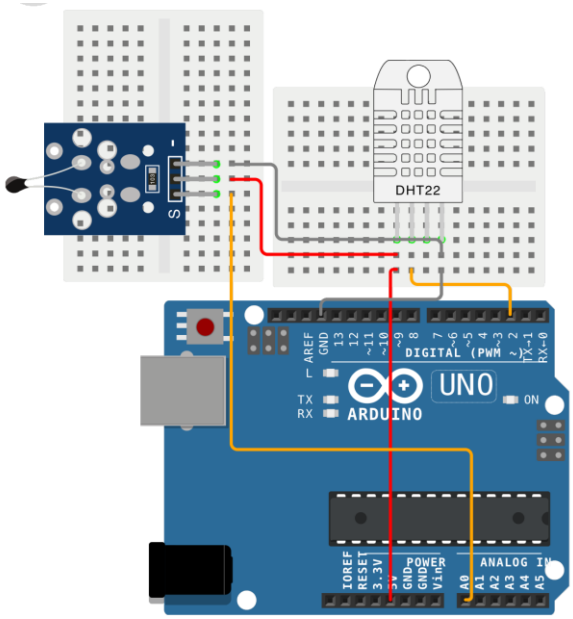


Figure 8: Temperature Sensing Circuit using LM-35 + DHT11 with Arduino Uno

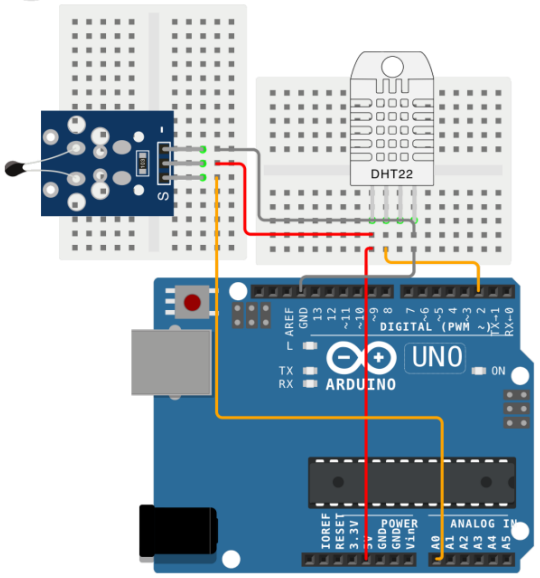
Code

```
#include "DHT.h"
#define DHTPIN 2
#define DHTTYPE DHT22
int lm35Pin = A0;
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  dht.begin();
}
void loop() {
  int analogValue = analogRead(lm35Pin);
  float tempLM35 = (analogValue * 5.0 * 100.0) /
      1024.0;
  float tempDHT = dht.readTemperature();
  Serial.println("-----------------------------");
  Serial.print("LM35 Temperature: ");
  Serial.print(tempLM35);
  Serial.println(" °C");
  Serial.print("DHT22 Temperature: ");
  Serial.print(tempDHT);
  Serial.println(" °C");
  Serial.println("-----------------------------");
  delay(2000);
      }
```

Observation

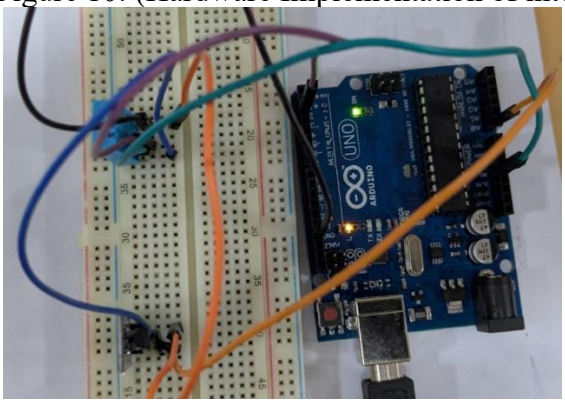
S.No.	LM35 Temperature (°C)	DHT11 Temperature (°C)	DHT11 Humidity (%)	Remarks / Notes
1				
2				
3				
4				
5				

Figure 9: (Software Implementation of interfacing LM-35 and DHT11 sensors)




```
------------------------------
LM35 Temperature: 255.37 °C
DHT22 Temperature: 24.00 °C
------------------------------
------------------------------
LM35 Temperature: 255.37 °C
DHT22 Temperature: 24.00 °C
------------------------------
```

Figure 10: (Hardware Implementation of interfacing LM-35 and DHT11 sensors)



LM35 Temperature (°C)	DHT11 Temperature (°C)	DHT11 Humidity (%)
138.3	26.5	32.0
41.1	26.5	32.0
49.4	26.5	32.0
65.5	26.5	32.0
58.7	26.5	32.0
46.4	26.5	33.0
51.8	26.4	33.0
65.5	26.4	33.0
67.0	26.4	33.0
41.1	26.4	33.0

Objective 4

To compare temperature measurements from the LM35 and DHT11 sensors and observe any differences.

Code

```
#include "DHT.h"
#define DHTPIN 2
#define DHTTYPE DHT22
int lm35Pin = A0;
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  dht.begin();
}
void loop() {
  float lm35Temp = (analogRead(lm35Pin) * 5.0 *
      100.0) / 1024.0;
  float dhtTemp = dht.readTemperature();
  if (isnan(dhtTemp)) {
    Serial.println("DHT22 failed to read!");
    delay(2000);
    return;
  }
  Serial.println("------------------------------");
  Serial.print("LM35 Temperature: ");
  Serial.print(lm35Temp);
  Serial.println(" °C");
  Serial.print("DHT22 Temperature: ");
  Serial.print(dhtTemp);
  Serial.println(" °C");
  float diff = lm35Temp - dhtTemp;
  Serial.print("Difference: ");
  Serial.print(diff);
  Serial.println(" °C");
  Serial.println("------------------------------");
  delay(2000);
}
```

Circuit / Schematic Diagram

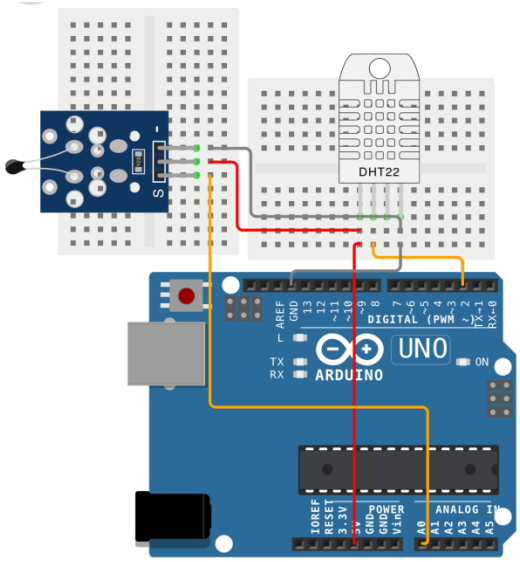


Figure 11: Temperature Sensing Circuit using LM-35 + DHT11 with Arduino Uno

Observation

S.No.	LM35 Temp (°C)	DHT11 Temp (°C)	Difference (°C)	Remarks
1				
2				
3				
4				
5				

Figure 9: (Software Implementation of interfacing LM-35 and DHT11 sensors for comparing the temperature data)

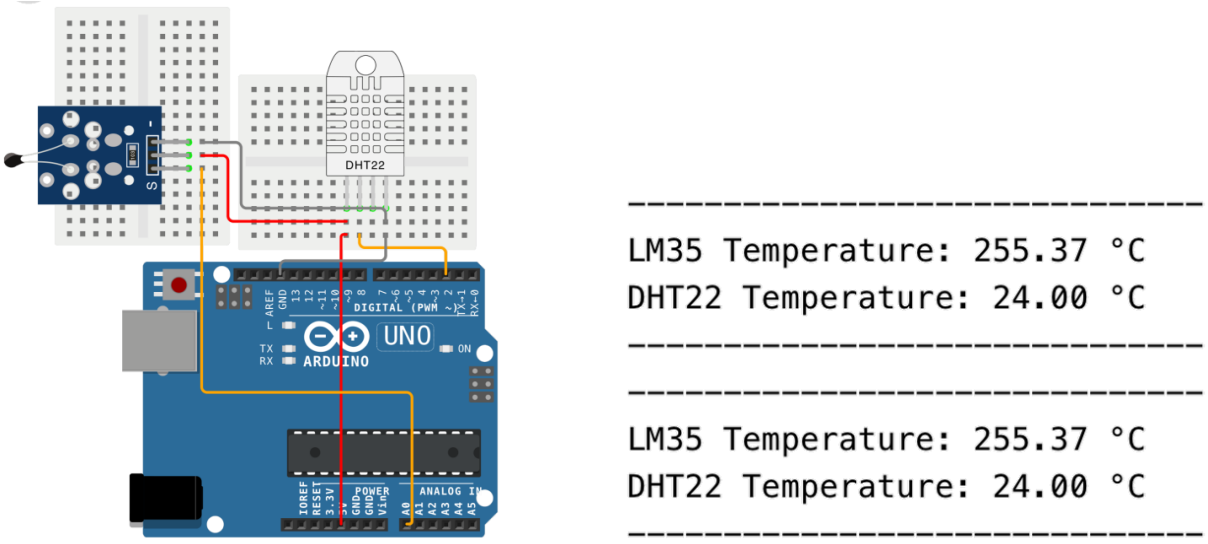
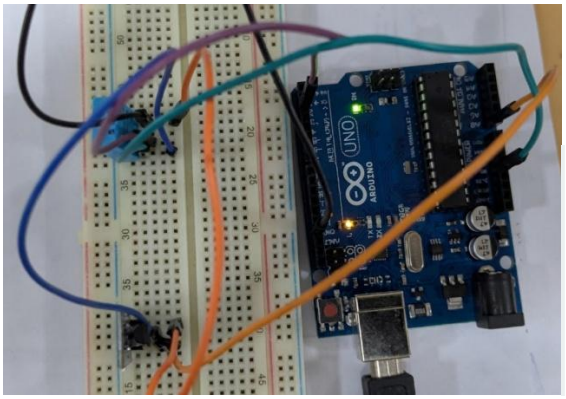


Figure 10: (Hardware Implementation of interfacing LM-35 and DHT11 sensors for comparing the temperature data)



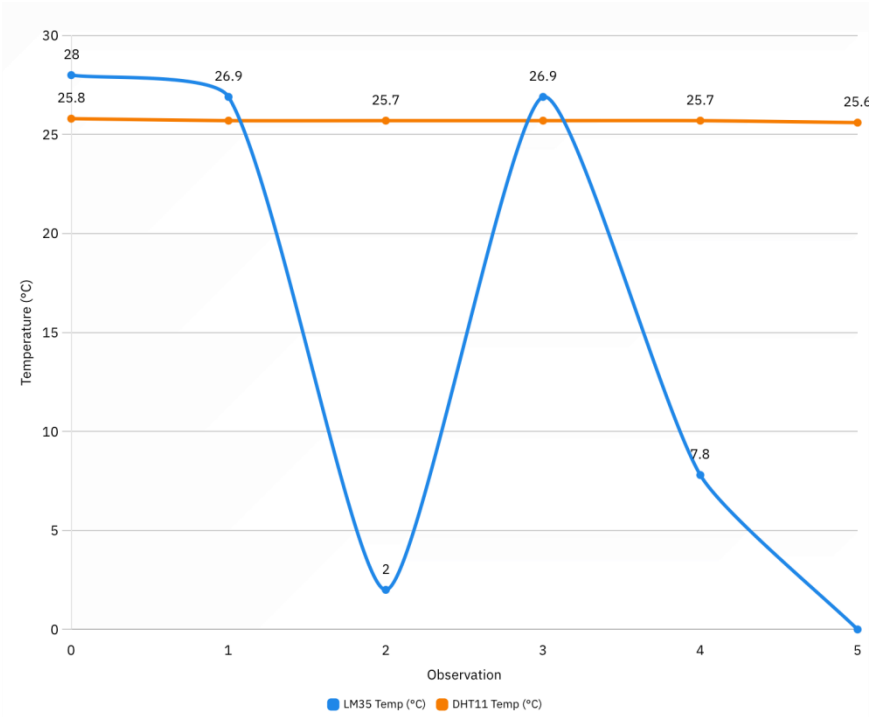
LM35 Temp (°C)	DHT11 Temp (°C)	Difference (°C)
132.0	25.8	106.2
26.9	25.7	1.2
2.0	25.7	-23.7
26.9	25.7	1.2
7.8	25.7	-17.9
0.0	25.6	-25.6

Graph

Instructions to Plot:

- X-axis: **Observation Number (1–5)**
- Y-axis: **Temperature (°C)**
- Plot two lines:
  - **LM35 Temp** (Analog)
  - **DHT11 Temp** (Digital)

Observation No	LM35 Temp	DHT11 Temp
1		
2		
3		
4		
5		



Conclusion

Precautions

Post Experiment Questionnaire:

1) What is the significance of resolution in an Analog-to-Digital Converter, and how does it affect measurement accuracy in Arduino?
2) How does the reference voltage (Vref) influence the output of the ADC in Arduino Uno?
3) Explain the concept of sampling in ADCs and why sampling rate is important in data acquisition systems.
4) What are the advantages and disadvantages of using analog sensors like LM35 compared to digital sensors like DHT11?
5) How does noise affect analog readings in micro-controller based systems, and what techniques can be used to reduce it?

Answers to Post-Lab Questions

(Signature of the Faculty)	**(Signature of the Student)**
	Name:
Date:	**Registration No.:**
	Branch:
	Section

Practical Robotics Projects with Arduino

(CSE 4571)

Lab Assignment No – 06

Servo Motor Control

Submission Date: 25/11/2025

Branch: CSE		**Section: 19**
Name	**Registration No.**	**Signature**

Department of Computer Science and Engineering
Institute of Technical Education and Research (Faculty of Engineering)
Siksha 'O' Anusandhan (Deemed to be University)
Bhubaneswar, Odisha-751030.

Aim:

Exploration and Implementation of ServoMotor Control Systems with Arduino Platform.

Objectives:

1. **Interface and control servomotors using Arduino, including using a potentiometer to adjust the position of the servo.**
   - **1.1. Demonstrate how to interface and control a servo motor with Arduino and configure the servo library.**
     - ✓ Introduce the concept of servo motors and their applications in embedded systems.
     - ✓ Explain how to connect a servo motor to an Arduino board using jumper wires.
     - ✓ Demonstrate how to install and configure the servo library in the Arduino IDE.
     - ✓ Write an Arduino sketch to Control a Servo Motor to Move Sequentially to Different Positions with Arduino and Serial Monitor.
   - **1.2. Control a servo motor to move back and forth between 0 and 180 degrees using the Arduino Servo library and for loop.**
     - ✓ Understanding the use of for loop to control the servo motor
     - ✓ Configuring the servo motor to move from 0 to 180 degrees using for loop
     - ✓ Configuring the servo motor to move from 180 to 0 degrees using for loop
     - ✓ Testing the servo motor and verifying its motion using serial monitor
   - **1.3. To control the position of a servo motor using values received from the serial monitor and verify its movement by observing the motor's motion through the use of serial communication.**
     - ✓ Understanding how to use the serial monitor to read input from the user.
     - ✓ Understanding how to parse the user input to obtain the desired servo position.
     - ✓ Understanding how to use the parsed input to control the servo motor.
     - ✓ Testing the program and verifying that the servo motor moves to the desired position in response to user input.
   - **1.4. Write an Arduino sketch to control the position of a servo motor using a potentiometer.**
     - ✓ Demonstrate how to connect a potentiometer to an Arduino board and read its analog input using the "analogRead" function.
     - ✓ Explain how to map the analog input value to the corresponding servo position value.
     - ✓ Write an Arduino sketch that reads the analog input from the potentiometer and generates PWM signals to control the position of the servo motor accordingly.
2. **Interface and control analog input devices such as an LDR and joystick with Arduino to collect data for use in servo motor control.**
   - **2.1. Interface an LDR (Light Dependent Resistor) with Arduino and use the collected data on environmental light conditions to control the position of a servo motor.**
     - ✓ Configure the LDR with Arduino.
     - ✓ Read the analog data from the LDR using the "analogRead" command.
     - ✓ Map the analog LDR data to the corresponding servo position using the "map" command.
     - ✓ Control the position of a servo motor based on the mapped LDR data.
     - ✓ Display the LDR value and corresponding servo position using the Serial monitor.
   - **2.2. Interface a Joystick with Arduino and Collect Data on its Position and Switch State**
     - ✓ Configure the pins for the joystick and switch as inputs.
     - ✓ Read the values of X and Y axis of the joystick and the switch.
     - ✓ Write an Arduino sketch to print the values of X and Y axis of the joystick and the switch state to the serial monitor.
   - **2.3. Interface a joystick with Arduino to control the position of a servo motor**
     - ✓ Connect the joystick to Arduino and calibrate the joystick inputs.
     - ✓ Map the joystick inputs to control the position of the servo motor.
     - ✓ Write an Arduino sketch to control the position of the servo motor using the joystick inputs.
   - **2.4. Implement advanced control techniques, such as using two servos to direct a laser beam, to demonstrate the versatility of servo motors in embedded systems.**
     - ✓ **Explain the principles of laser beam control using two servo motors and how it can be used in practical applications.** In this section, you would explain how two servo motors can be used to control the position of a laser beam. One servo motor would control the horizontal movement of the laser, while the other would control the vertical movement. This allows for precise targeting of the laser beam, which can be useful in a variety of practical applications such as laser cutting, engraving, and alignment.
     - ✓ **Demonstrate how to connect a laser diode to a servo motor and control its position using Arduino.** In this section, you would explain how to connect a laser diode to a servo motor and control its position using Arduino.

Pre-Lab Questionnaire:

1) What are motors and how do they work?
2) What are the components and mechanisms involved in the operation of a servomotor?
3) How do torque, speed, and precision relate to the unique features of servomotors?
4) What are some common applications for servomotors in embedded systems?
5) What is the role of feedback in motor control, and how can it be implemented with Arduino?
6) What is the difference between open-loop and closed-loop motor control, and how can Arduino be used to implement closed-loop control?
7) What is the purpose of the servo library in Arduino?
8) What are the applications of servo motors in embedded systems?
9) What is the role of pulse width modulation (PWM) in servo control?
10) How is the position of a servo motor related to the duty cycle of the PWM signal?
11) What is the purpose of mapping the analog input value to the corresponding servo position value?
12) What is the difference between digital and analog signals, and how are they used in servo control?

Answers to Pre-Lab Questions

Components/Equipment Required:

Sl. No.	Name of the Component / Equipment	Specification	Quantity
1	Arduino UNO R3	16MHz	1
2	Arduino UNO cable	USB Type A to B	1
3	Trimmer Potentiometer	10k, Preset	1
4	LDR/Photoresistor	5mm	1
5	Dual axis Joystick module		1
6	Servo Motor	SG90	2
7	Dual H-Bridge Motor Driver IC	L293D, DIP-16 Package	1
8	Toy fan blade	----------	1
9	Resistors (carbon type)	¼ watt (330Ω)	1
		¼ watt (4.7kΩ)	1
10	LED	Any two different colour of your choice	2
11	Buzzer	5V, small	1
12	Breadboard	840 Tie points	1
13	Digital Multimeter	----------	1
14	Jumper Wire	----------	As per requirement

Objective 1

Interface and control servomotors using Arduino, including using a potentiometer to adjust the position of the servo.

1. **Demonstrate how to interface and control a servo motor with Arduino and configure the servo library.**
   - ✓ Introduce the concept of servo motors and their applications in embedded systems.
   - ✓ Explain how to connect a servo motor to an Arduino board using jumper wires.
   - ✓ Demonstrate how to install and configure the servo library in the Arduino IDE.
   - ✓ Write an Arduino sketch to Control a Servo Motor to Move Sequentially to Different Positions with Arduino and Serial Monitor.
2. **Control a servo motor to move back and forth between 0 and 180 degrees using the Arduino Servo library and for loop.**
   - ✓ Understanding the use of for loop to control the servo motor
   - ✓ Configuring the servo motor to move from 0 to 180 degrees using for loop
   - ✓ Configuring the servo motor to move from 180 to 0 degrees using for loop
   - ✓ Testing the servo motor and verifying its motion using serial monitor
3. **To control the position of a servo motor using values received from the serial monitor and verify its movement by observing the motor's motion through the use of serial communication.**
   - ✓ Understanding how to use the serial monitor to read input from the user.

   - ✓ Understanding how to parse the user input to obtain the desired servo position.

- ✓ Understanding how to use the parsed input to control the servo motor.
- ✓ Testing the program and verifying that the servo motor moves to the desired position in response to user input.

4. **Write an Arduino sketch to control the position of a servo motor using a potentiometer.**
   - ✓ Demonstrate how to connect a potentiometer to an Arduino board and read its analog input using the "analogRead" function.
   - ✓ Explain how to map the analog input value to the corresponding servo position value.
   - ✓ Write an Arduino sketch that reads the analog input from the potentiometer and generates PWM signals to control the position of the servo motor accordingly.

Circuit / Schematic Diagram

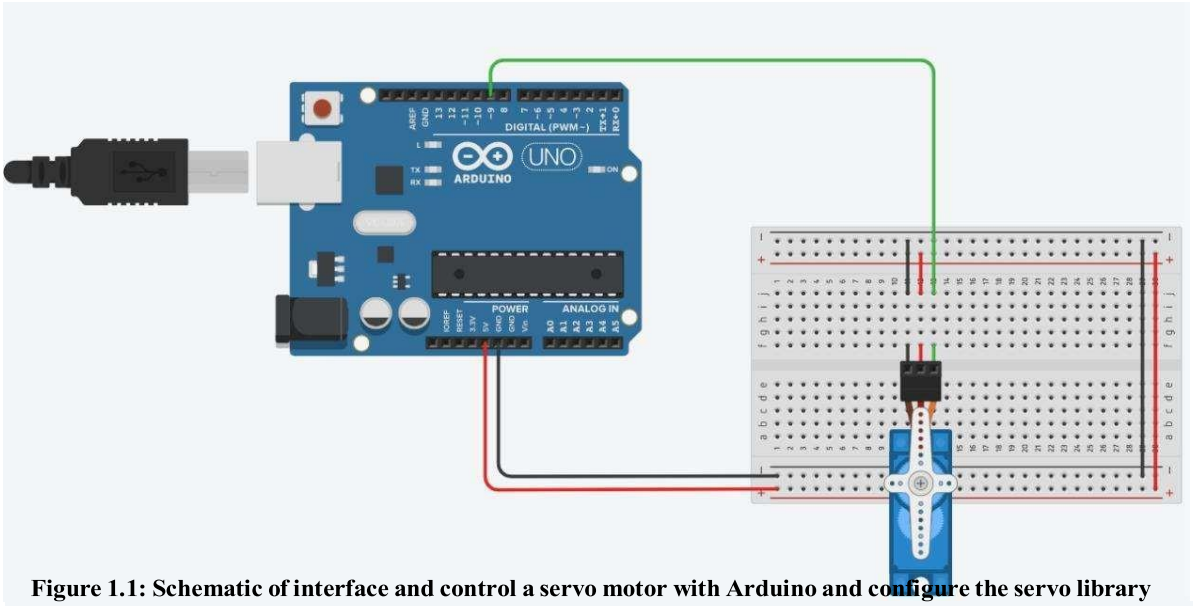


Figure 1.1: Schematic of interface and control a servo motor with Arduino and configure the servo library

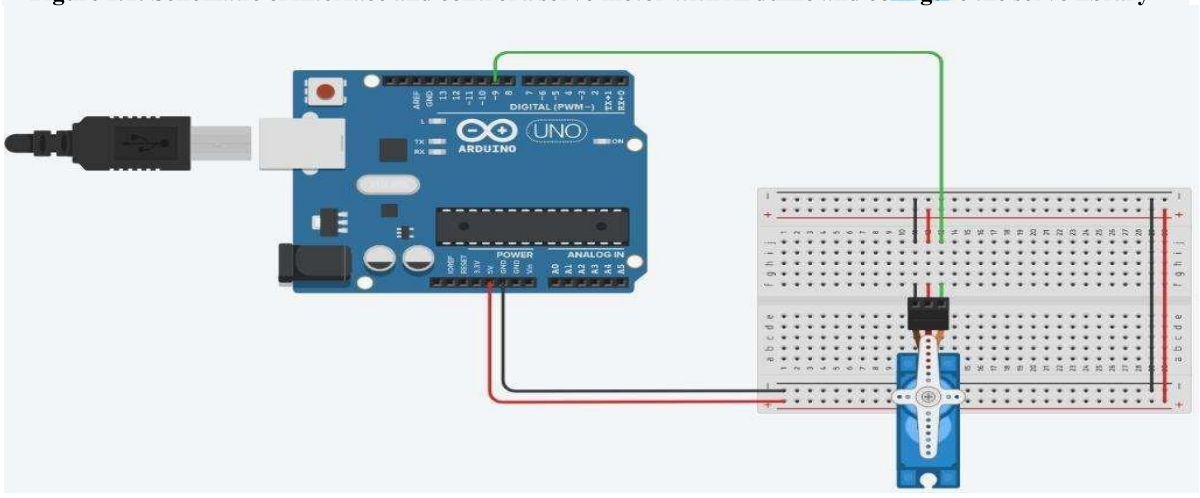


Figure 1.2: Schematic of control a servo motor to move back and forth between 0 and 180 degrees using the Arduino Servo library and for loop.

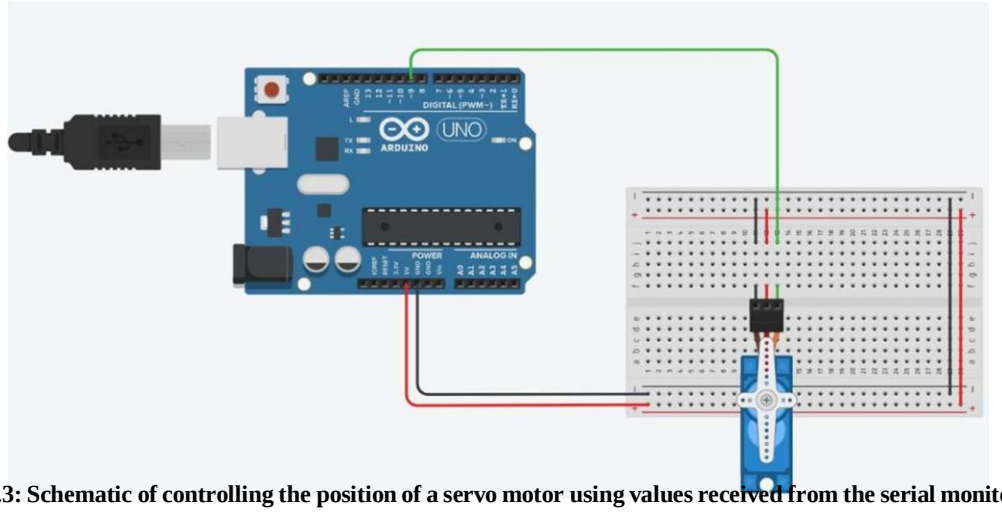


Figure 1.3: Schematic of controlling the position of a servo motor using values received from the serial monitor

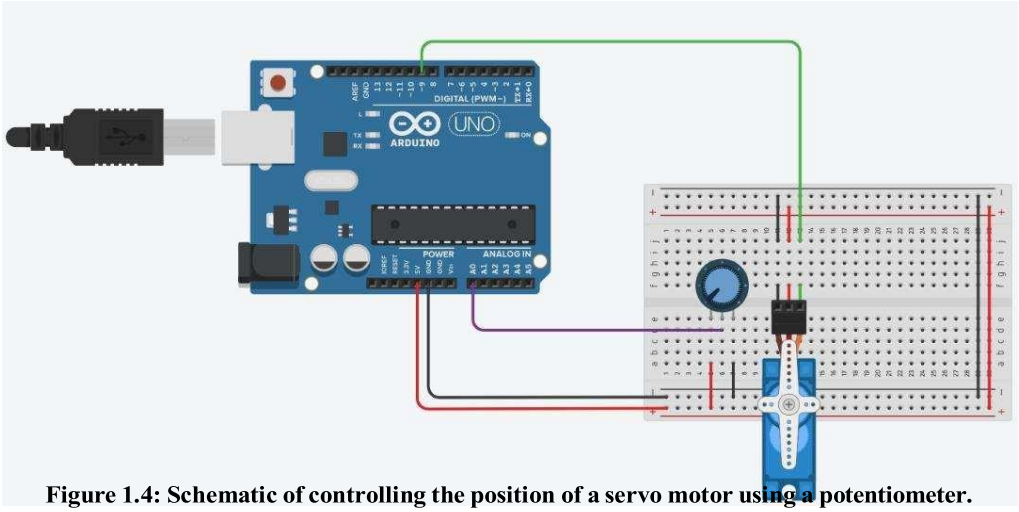


Figure 1.4: Schematic of controlling the position of a servo motor using a potentiometer.

Code

1.1 Interface and control a servo motor with Arduino and configure the servo library.

```
#include <Servo.h> Servo myServo; void
setup() {
  myServo.attach(9); // Attach servo signal wire to pin
  9 Serial.begin(9600);
  Serial.println("Servo Interface Test Started");
}
void loop() { // Move to 0 degrees
  myServo.write(0);
  Serial.println("Position: 0°");
  delay(1000); // Move to 90 degrees (middle)
  myServo.write(90);
  Serial.println("Position: 90°");
  delay(1000); // Move to 180 degrees
  myServo.write(180);
  Serial.println("Position: 180°");
  delay(1000);
}
```

1.2 Control a servo motor to move back and forth between 0 and 180 degrees using the Arduino Servo library and for loop.

```
#include <Servo.h>
Servo myServo;
void setup() {
myServo.attach(9);
Serial.begin(9600);
  Serial.println("Servo Sweep Test Started");
}
void loop() {
  for(int angle = 0; angle <= 180; angle++) {
    myServo.write(angle);
    Serial.print("Angle: ");
    Serial.println(angle);
    delay(10);
  }
  for(int angle = 180; angle >= 0; angle--) {
    myServo.write(angle);
    Serial.print("Angle: ");
    Serial.println(angle);
    delay(10);
  }
}
```

1.3 Control the position of a servo motor using values received from the serial monitor and verify its movement by observing the motor's motion through the use of serial communication.

```
#include <Servo.h>
Servo myServo;
int angle = 0;
void setup() {
  Serial.begin(9600);
  myServo.attach(9);
  Serial.println("Enter angle between 0 and 180:");
}
void loop() {
  if (Serial.available() > 0) {
    angle = Serial.parseInt();
    if (angle >= 0 && angle <= 180) {
      myServo.write(angle);
      Serial.print("Servo moved to: ");
      Serial.println(angle);
    }
    else {
      Serial.println("Invalid input! Enter a value between 0 and 180.");
    }
  }
}
```

1.4 Control the position of a servo motor using a potentiometer.

```
#include <Servo.h>
Servo myServo;
int potPin = A0;
int potValue = 0;
int angle = 0;
void setup() {
  myServo.attach(9);
```

```
  Serial.begin(9600);
  Serial.println("Potentiometer -> Servo Control Started");
}
void loop() {
  potValue = analogRead(potPin);
  angle = map(potValue, 0, 1023, 0, 180);
  myServo.write(angle);
  Serial.print("Potentiometer: ");
  Serial.print(potValue);
  Serial.print(" | Angle: ");
  Serial.println(angle);
  delay(15);
}
```

Observation

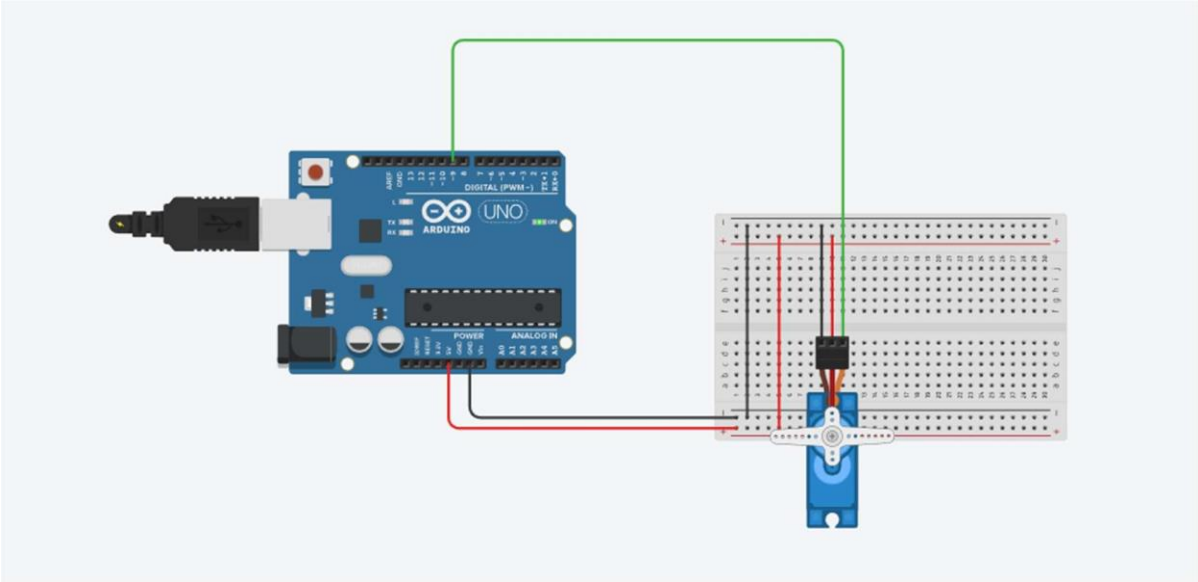


Figure 1.1.1: Simulation based interface and control a servo motor with Arduino and configure the servo library on Tinkercad.

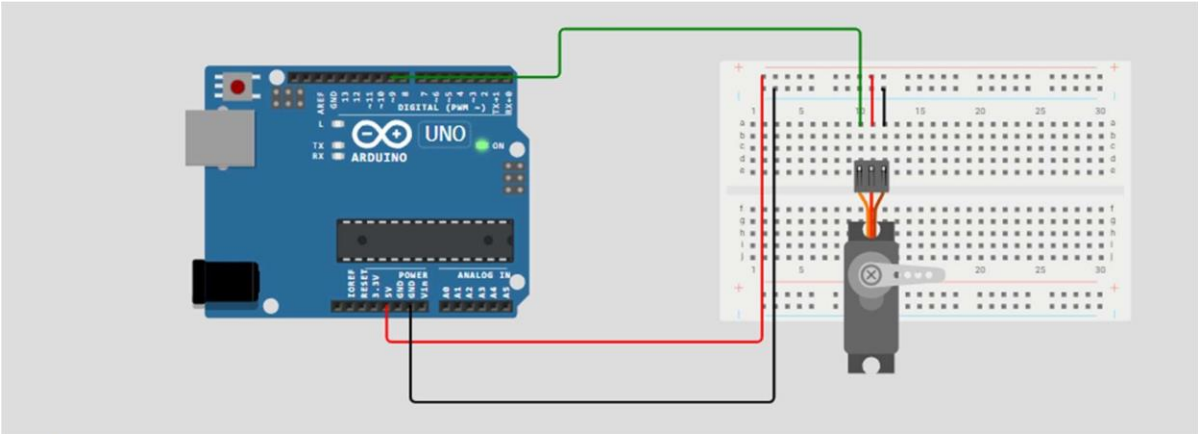


Figure 1.1.2: Simulation based interface and control a servo motor with Arduino and configure the servo library on Wokwi.

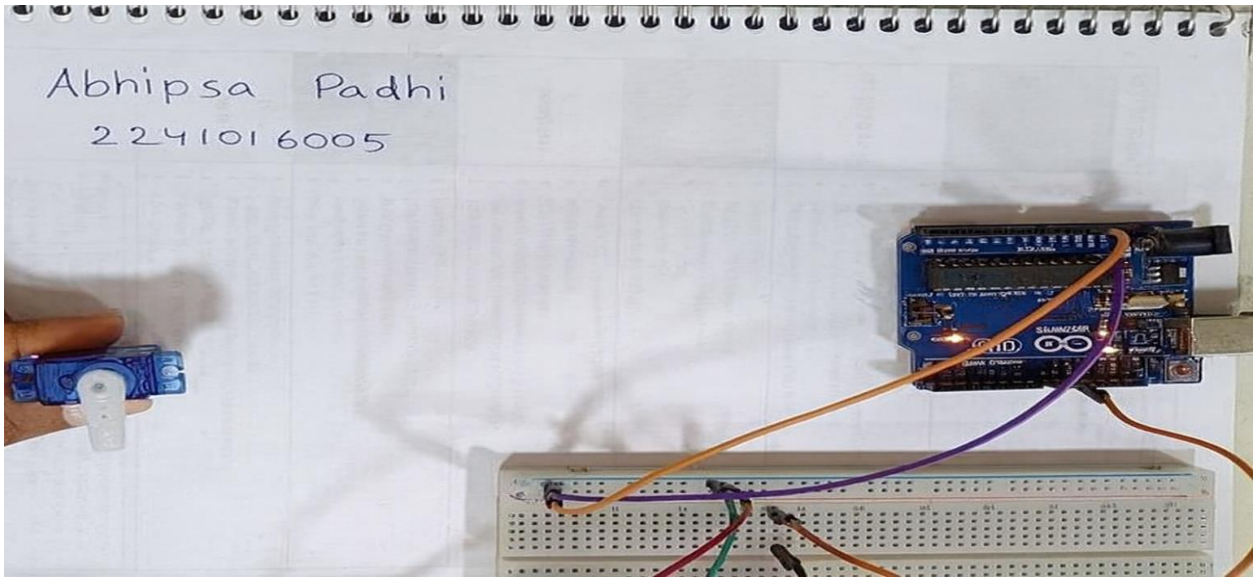


Figure 1.1.3: Hardware Implementation based interface and controlling a servo motor with Arduino and configure the servo library.

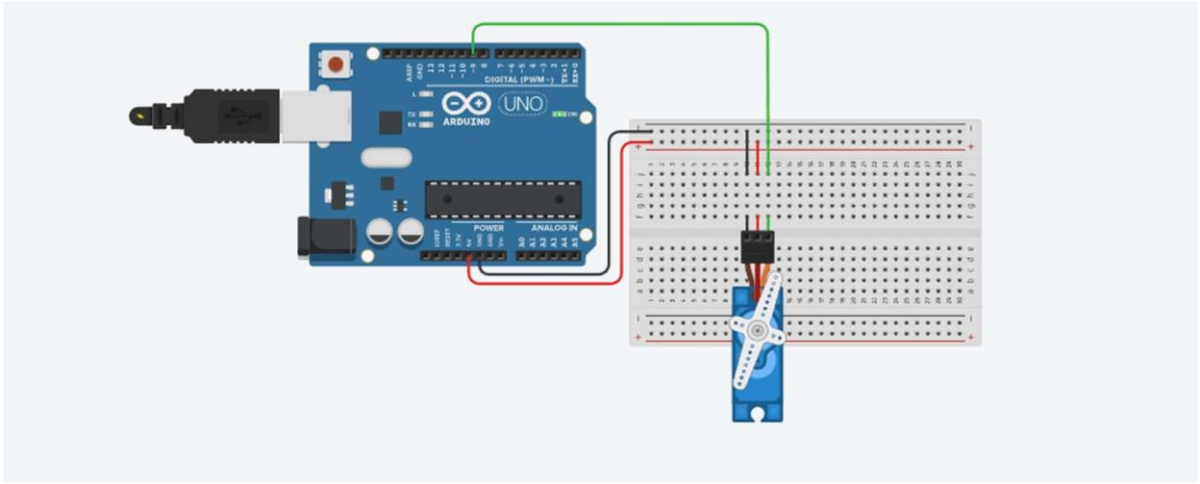


Figure 1.2.1: Simulation based controlling a servo motor to move back and forth between 0 and 180 degrees using the Arduino Servo library and for loop on Tinkercad

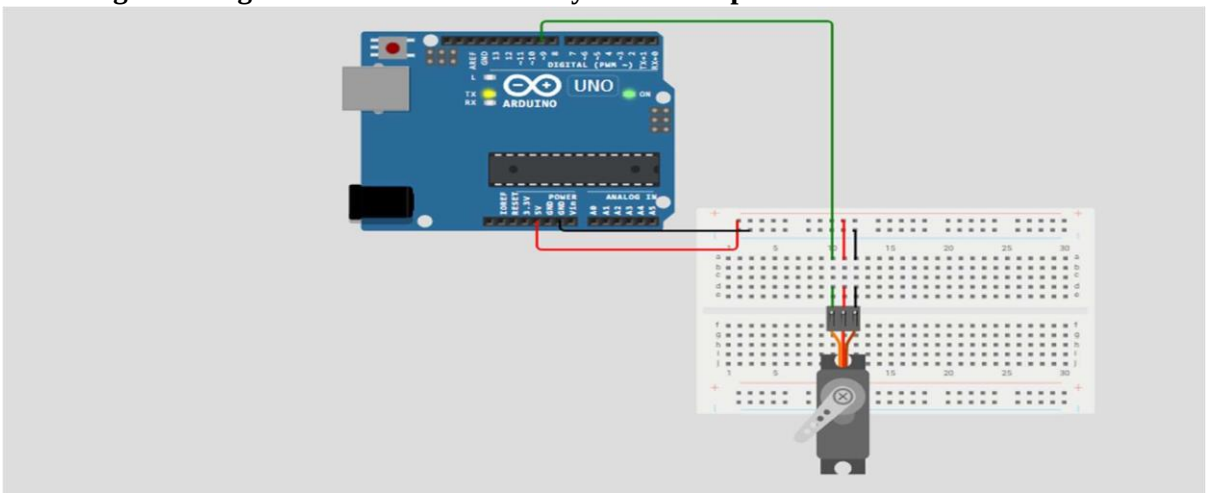


Figure 1.2.2: Simulation based controlling a servo motor to move back and forth between 0 and 180 degrees using the Arduino Servo library and for loop on Wokwi.

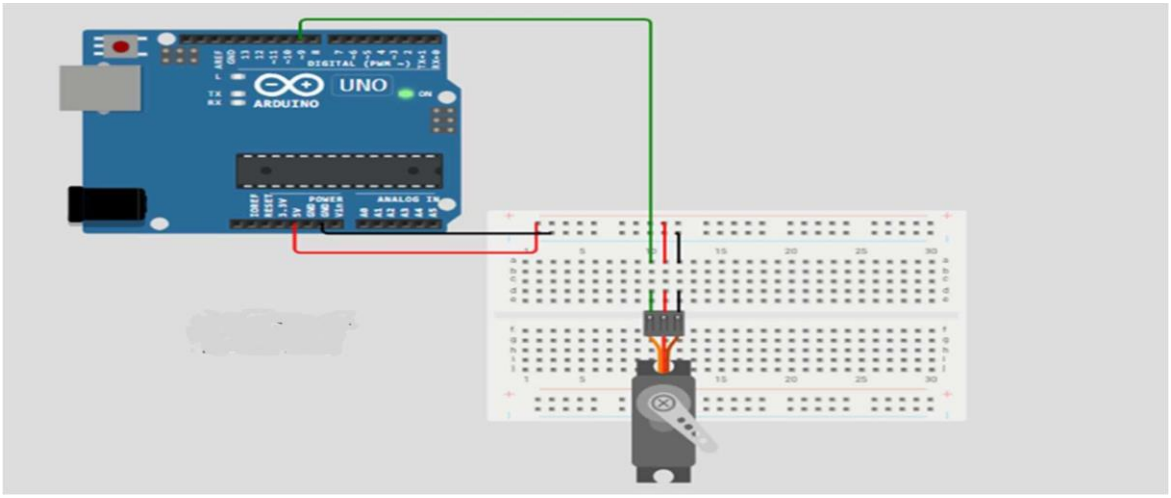


Figure 1.3.1: Simulation based controlling the position of a servo motor using values received from the serial monitor and verify its movement by observing the motor's motion through the use of serial communication on Tinkercad.

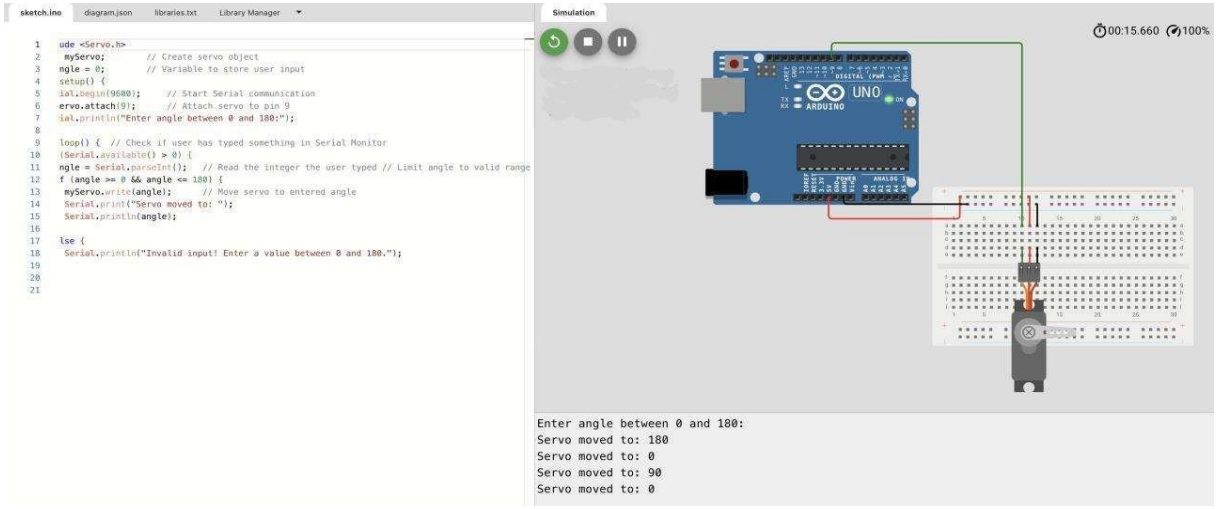


Figure 1.3.2: Simulation based controlling the position of a servo motor using values received from the serial monitor and verify its movement by observing the motor's motion through the use of serial communication on Wokwi.

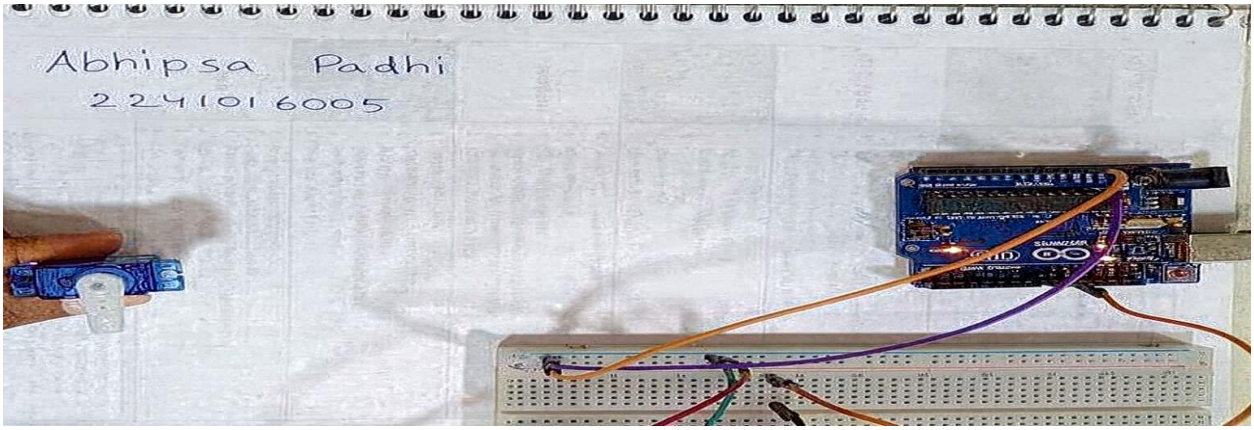


Figure 1.3.2: Hardware Implementation based controlling the position of a servo motor using values received from the serial monitor and verify its movement by observing the motor's motion through the use of serial communication.

Servo motor control:-To Connect a potentiometer to Arduino UNO to control servo motor position and display the angle on a Display device.

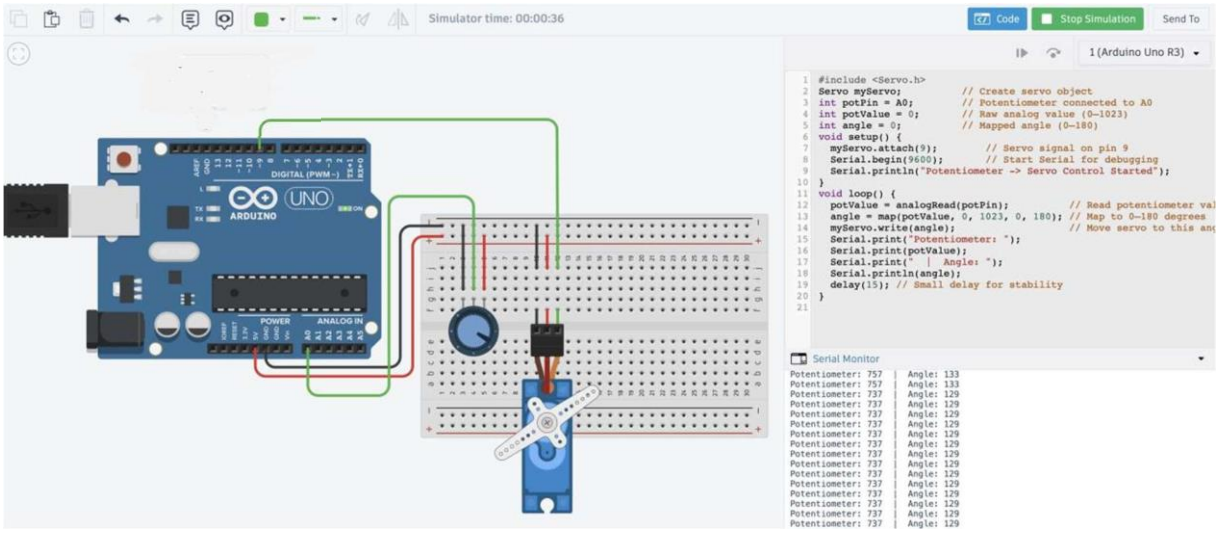


Figure 1.4.1: Simulation based controlling the position of a servo motor using a potentiometer on Tinkercad.

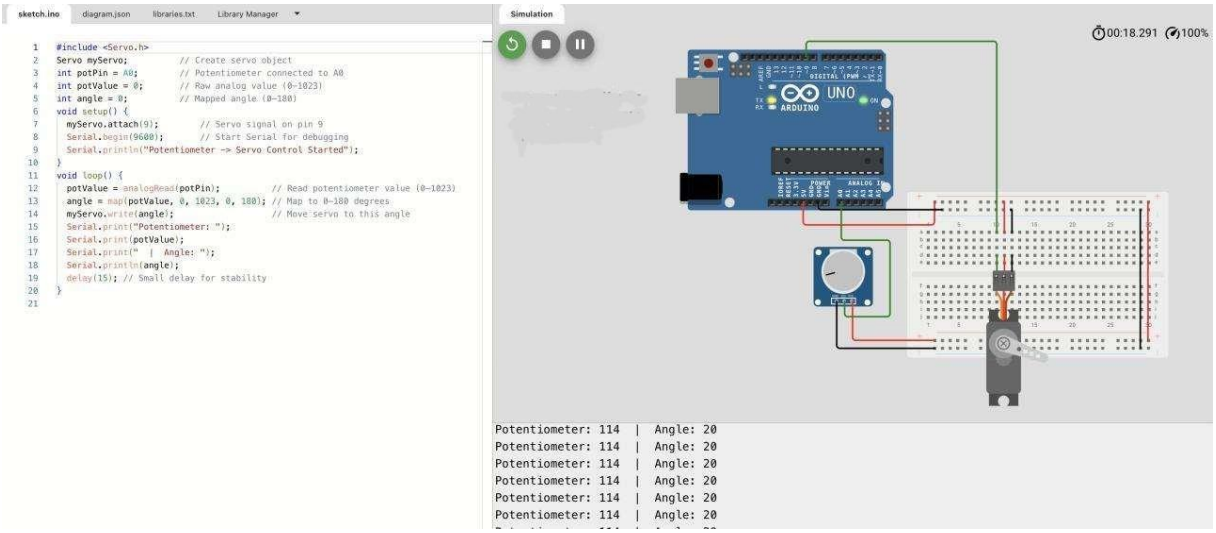
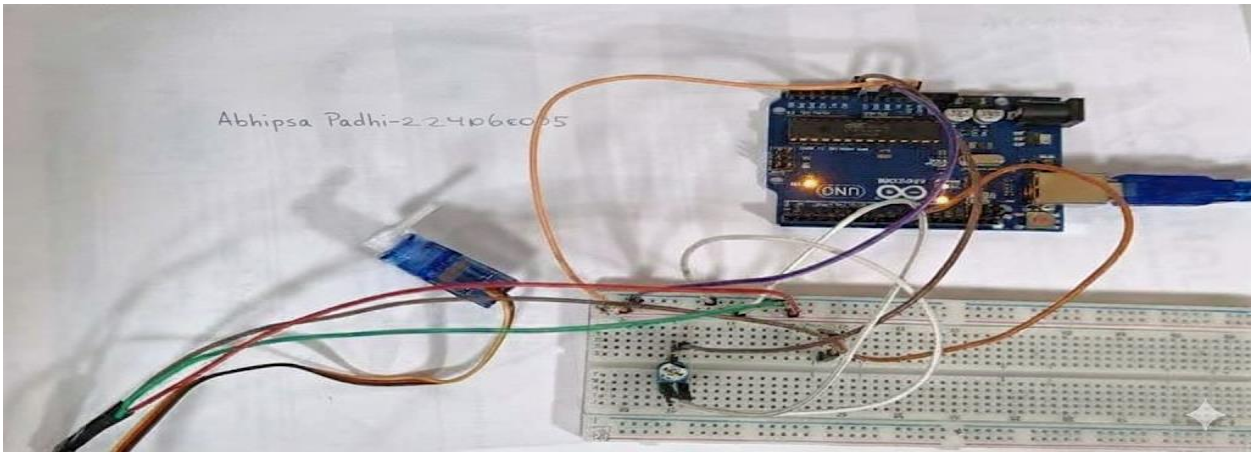


Figure 1.4.2: Simulation based controlling the position of a servo motor using a potentiometer on Wokwi.

Figure 1.4.2: Hardware Implementation based controlling the position of a servo motor using a potentiometer.



Objective 2

Interface and control analog input devices such as an LDR and joystick with Arduino to collect data for use in servo motor control.

1. **Interface an LDR (Light Dependent Resistor) with Arduino and use the collected data on environmental light conditions to control the position of a servo motor.**
   - ✓ Configure the LDR with Arduino.
   - ✓ Read the analog data from the LDR using the "analogRead" command.
   - ✓ Map the analog LDR data to the corresponding servo position using the "map" command.
   - ✓ Control the position of a servo motor based on the mapped LDR data.
   - ✓ Display the LDR value and corresponding servo position using the Serial monitor.
2. **Interface a Joystick with Arduino and Collect Data on its Position and Switch State**
   - ✓ Configure the pins for the joystick and switch as inputs.
   - ✓ Read the values of X and Y axis of the joystick and the switch.
   - ✓ Write an Arduino sketch to print the values of X and Y axis of the joystick and the switch state to the serial monitor.
3. **Interface a joystick with Arduino to control the position of a servo motor:**
   - ✓ Connect the joystick to Arduino and calibrate the joystick inputs.
   - ✓ Map the joystick inputs to control the position of the servo motor.
   - ✓ Write an Arduino sketch to control the position of the servo motor using the joystick inputs.
4. **Implement advanced control techniques, such as using two servos to direct a laser beam, to demonstrate the versatility of servo motors in embedded systems.**
   - ✓ **Explain the principles of laser beam control using two servo motors and how it can be used in practical applications.** In this section, you would explain how two servo motors can be used to control the position of a laser beam. One servo motor would control the horizontal movement of the laser, while the other would control the vertical movement. This allows for precise targeting of the laser beam, which can be useful in a variety of practical applications such as laser cutting, engraving, and alignment.
   - ✓ **Demonstrate how to connect a laser diode to a servo motor and control its position using Arduino.** In this section, you would explain how to connect a laser diode to a servo motor and control its position using Arduino. This would likely involve wiring the servo motors to the appropriate pins on the Arduino, connecting the laser diode to one of the servo motors, and configuring the servo motors to move the laser beam in the desired manner.
   - ✓ Write an Arduino sketch to control the position of a laser beam using two servo motors.

Circuit / Schematic Diagram

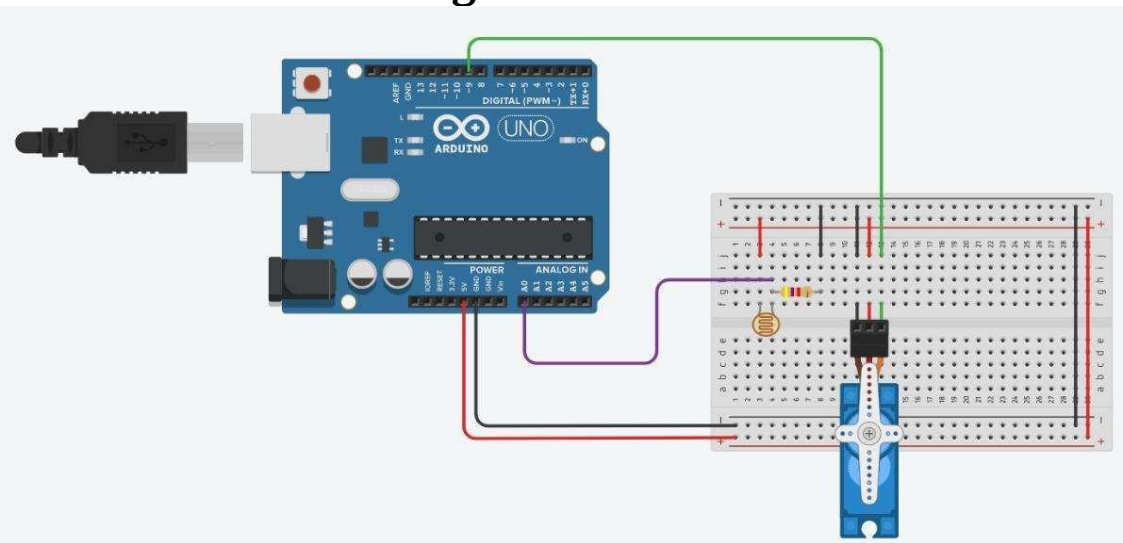


Figure 2.1: Schematic of interfacing an LDR (Light Dependent Resistor) with Arduino and use the collected data on environmental light conditions to control the position of a servo motor.

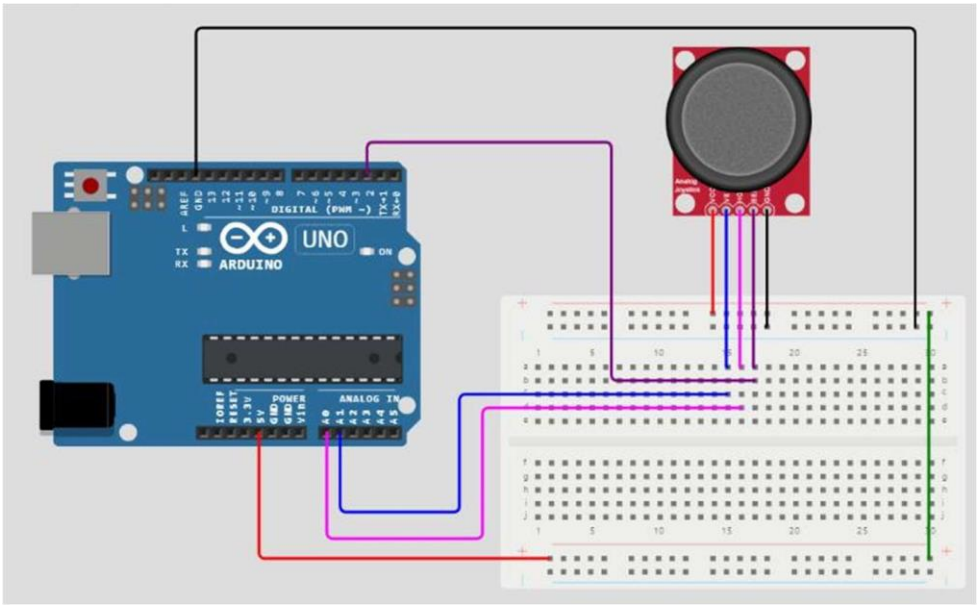


Figure 2.2: Schematic of interfacing a Joystick with Arduino and Collect Data on its Position and Switch State.

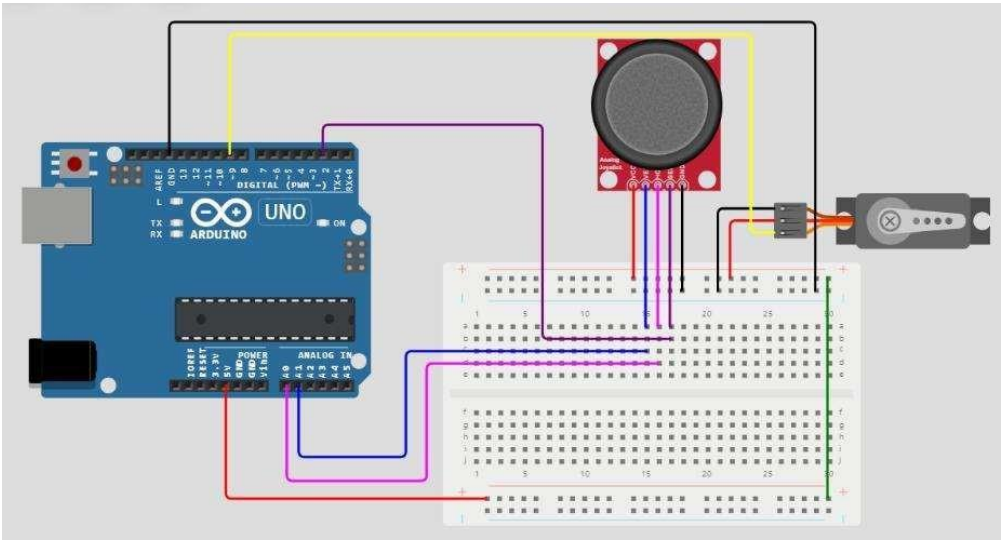


Figure 2.3: Schematic of interfacing a joystick with Arduino to control the position of a servo motor.

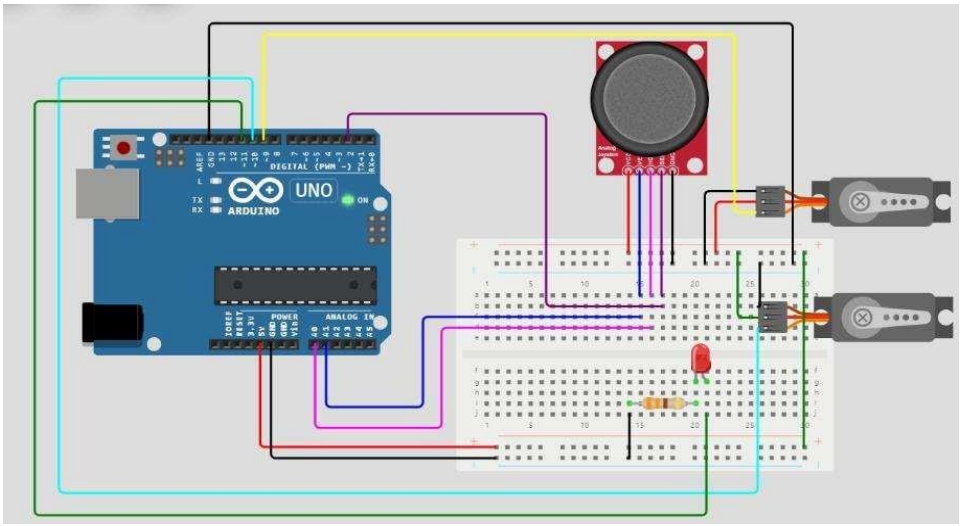


Figure 2.4: Schematic of implementing advanced control techniques, such as using two servos to direct a laser beam, to demonstrating the versatility of servo motors in embedded systems.

Code

2.1 Interfacing an LDR (Light Dependent Resistor) with Arduino and use the collected data on environmental light conditions to control the position of a servo motor Ans :

```
#include <Servo.h>
Servo myServo;
int ldrPin = A0;
int ldrValue = 0;
int angle = 0;
void setup() {
  Serial.begin(9600);
  myServo.attach(9);
  Serial.println("LDR -> Servo Control Started");
}
void loop() {
  ldrValue = analogRead(ldrPin);
  angle = map(ldrValue, 0, 1023, 0, 180);
  myServo.write(angle);
  Serial.print("LDR Value: ");
  Serial.print(ldrValue);
  Serial.print(" | Servo Angle: ");
  Serial.println(angle);
  delay(10);
```

2.2 Interfacing a Joystick with Arduino and Collect Data on its Position and Switch State. Ans :

```
int xPin = A0;
int yPin = A1;
int swPin = 2;
int xValue = 0;
int yValue = 0;
int swState = 0;
void setup() {
  Serial.begin(9600);
  pinMode(swPin, INPUT_PULLUP);
  Serial.println("Joystick Reading Started:");
}
void loop() {
  xValue = analogRead(xPin);
  yValue = analogRead(yPin);
  swState = digitalRead(swPin);
  Serial.print("X: ");
  Serial.print(xValue);
  Serial.print("  Y: ");
  Serial.print(yValue);
  Serial.print("  Switch: ");
  Serial.println(swState == LOW ? "Pressed" : "Not
  Pressed"); delay(150);
}
```

2.3 Interfacing a joystick with Arduino to control the position of a servo motor. Ans :

```
#include <Servo.h>
Servo myServo;
int joyX = A0;
int xValue = 0;
int angle = 0;
void setup() {
```

```
  Serial.begin(9600);
  myServo.attach(9);
  Serial.println("Joystick -> Servo Control Started");
}
void loop() {
  xValue = analogRead(joyX);
  angle = map(xValue, 0, 1023, 0, 180);
  myServo.write(angle);
  Serial.print("Joystick X: ");
  Serial.print(xValue);
  Serial.print(" | Servo Angle: ");
  Serial.println(angle);
  delay(10);
}
```

2.4 Implementing advanced control techniques, such as using two servos to direct a laser beam, to demonstrating the versatility of servo motors in embedded systems. Ans :

```
#include <Servo.h>
Servo servoX;
Servo servoY;
const int servoXPin = 9;
const int servoYPin = 10;
const int laserPin = 11;
int xMinAngle = 20; int xMaxAngle = 160; int yMinAngle = 40; int yMaxAngle = 120;
void setup() {
  Serial.begin(9600);
  servoX.attach(servoXPin); servoY.attach(servoYPin);
  pinMode(laserPin, OUTPUT);
  digitalWrite(laserPin, LOW); // Laser OFF at start
servoX.write((xMinAngle + xMaxAngle) / 2);
servoY.write((yMinAngle + yMaxAngle) / 2);
Serial.println("Two-Servo Laser Pan-Tilt Control Started");}
void loop() {
  digitalWrite(laserPin,  HIGH);
  for (int x = xMinAngle; x <= xMaxAngle; x += 10) {
    servoX.write(x);
    delay(150); // give time to move
    for (int y = yMinAngle; y <= yMaxAngle; y += 10) {
      servoY.write(y);
      delay(150);
      Serial.print("Laser at X: ");
      Serial.print(x);
      Serial.print(" deg, Y: ");
      Serial.print(y);
      Serial.println(" deg");   } }
  for (int x = xMaxAngle; x >= xMinAngle; x -= 10) {
    servoX.write(x);
    delay(150);
    for (int y = yMaxAngle; y >= yMinAngle; y -= 10) {
      servoY.write(y);
      delay(150);
      Serial.print("Laser at X: ");
      Serial.print(x);
      Serial.print(" deg, Y: ");
      Serial.print(y);
      Serial.println(" deg");   } }
  digitalWrite(laserPin, LOW);
  delay(1000);
}
```

Observation

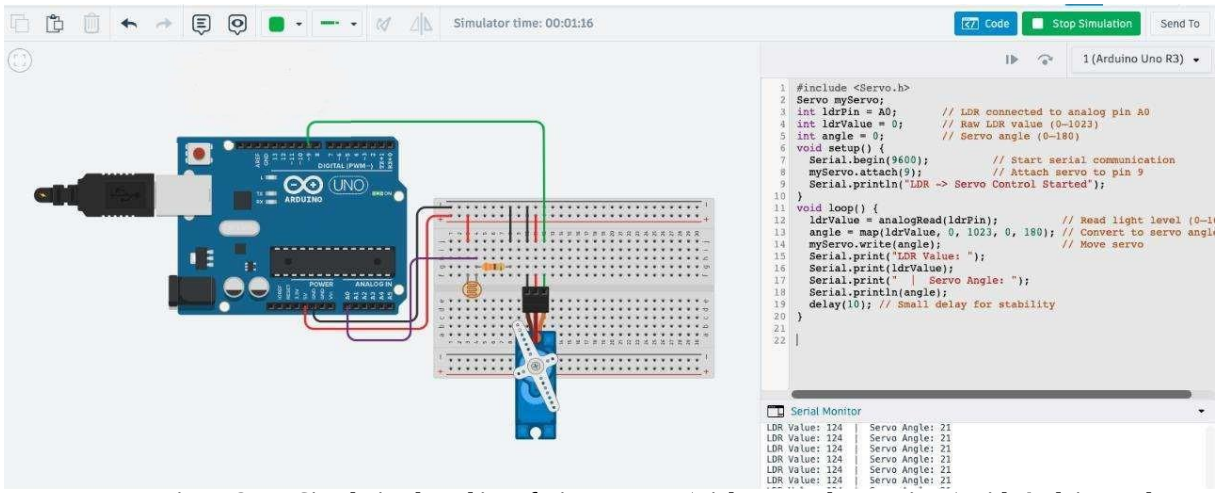


Figure 2.1.1: Simulation based interfacing an LDR (Light Dependent Resistor) with Arduino and use the collected data on environmental light conditions to control the position of a servo motor on Tinkercad.

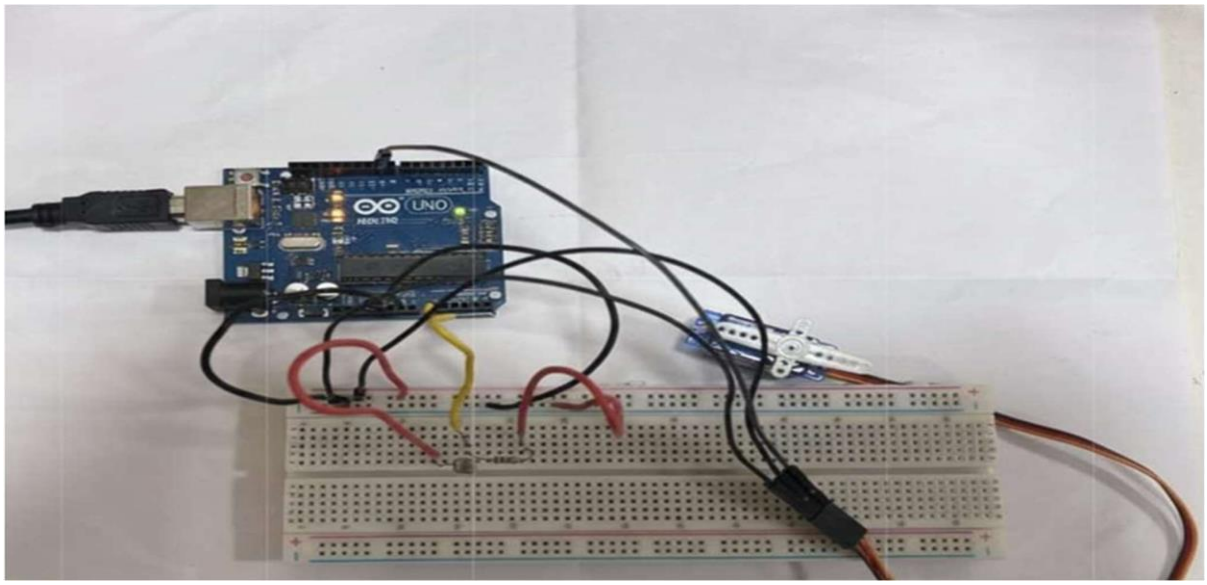


Figure 2.1.2: Hardware Implementation based interfacing an LDR (Light Dependent Resistor) with Arduino and use the collected data on environmental light conditions to control the position of a servo motor.

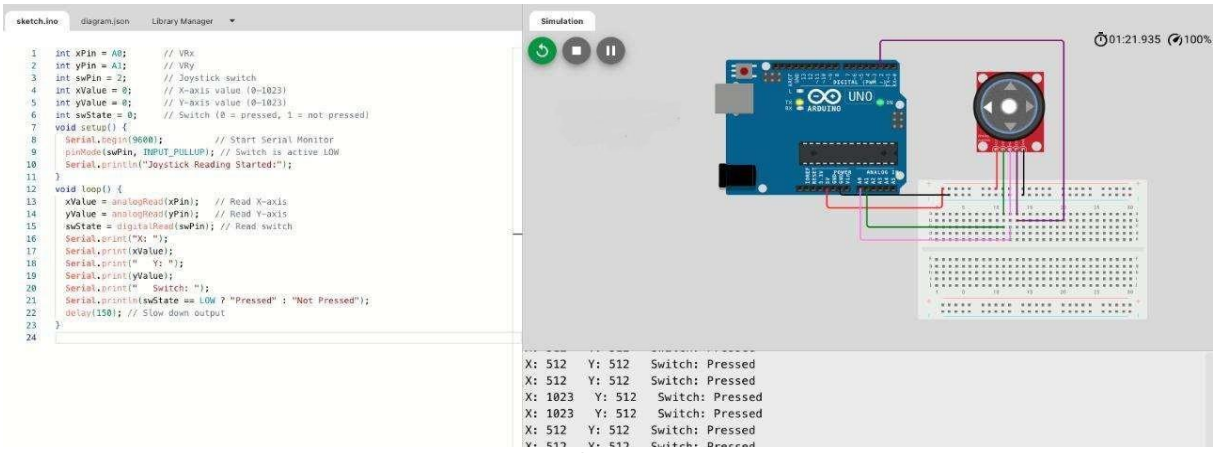


Figure 2.2.1: Simulation based interfacing a Joystick with Arduino and Collect Data on its Position and Switch State on Wokwi.

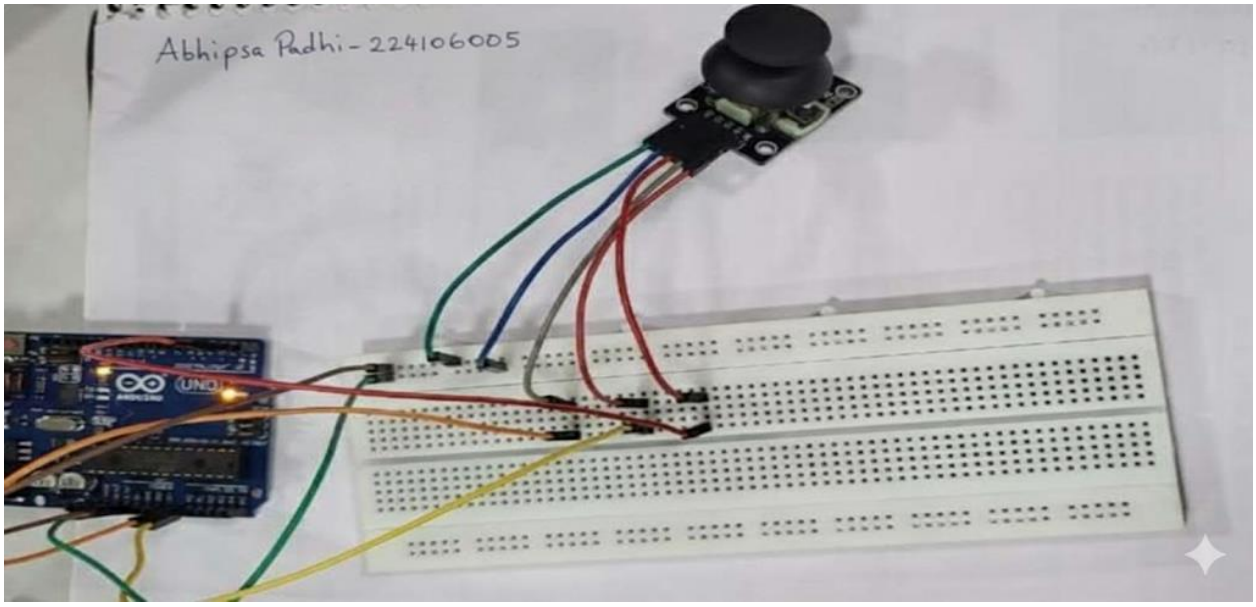


Figure 2.2.2: Hardware Implementation based interfacing a Joystick with Arduino and Collect Data on its Position and Switch State.

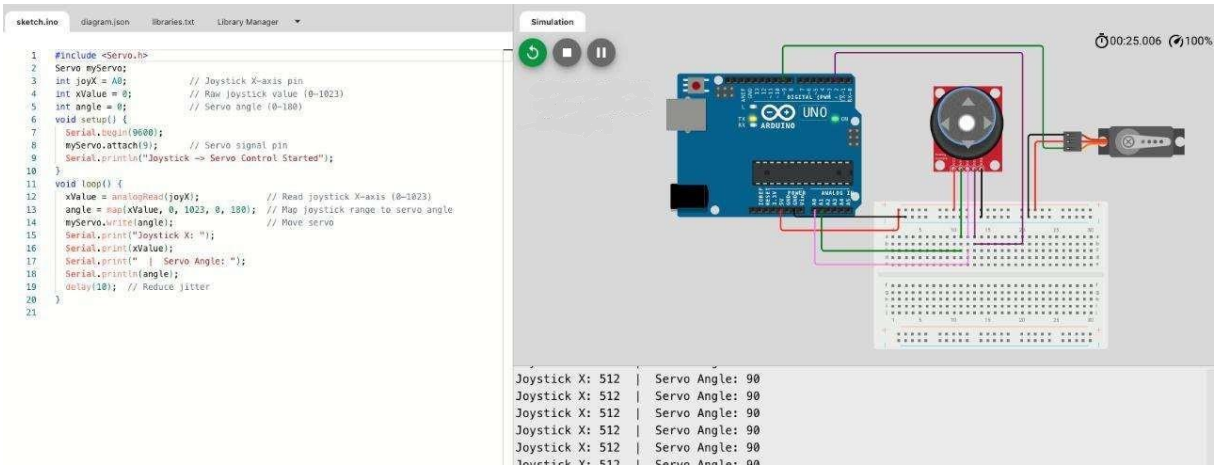
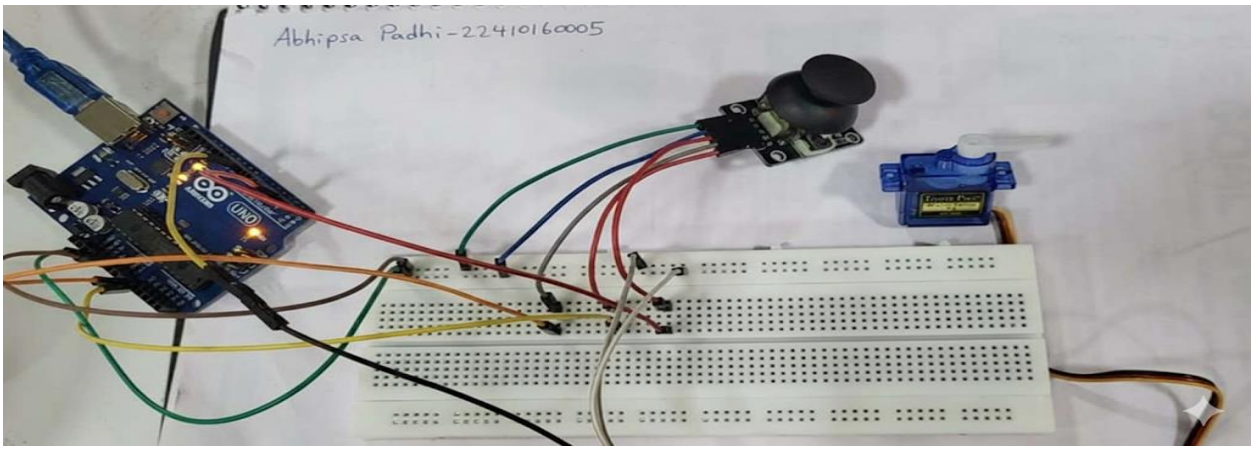


Figure 2.3.1: Simulation based interfacing a joystick with Arduino to control the position of a servo motor on Wokwi.

Figure 2.3.2: Hardware Implementation based interfacing a joystick with Arduino to control the position of a servo motor.



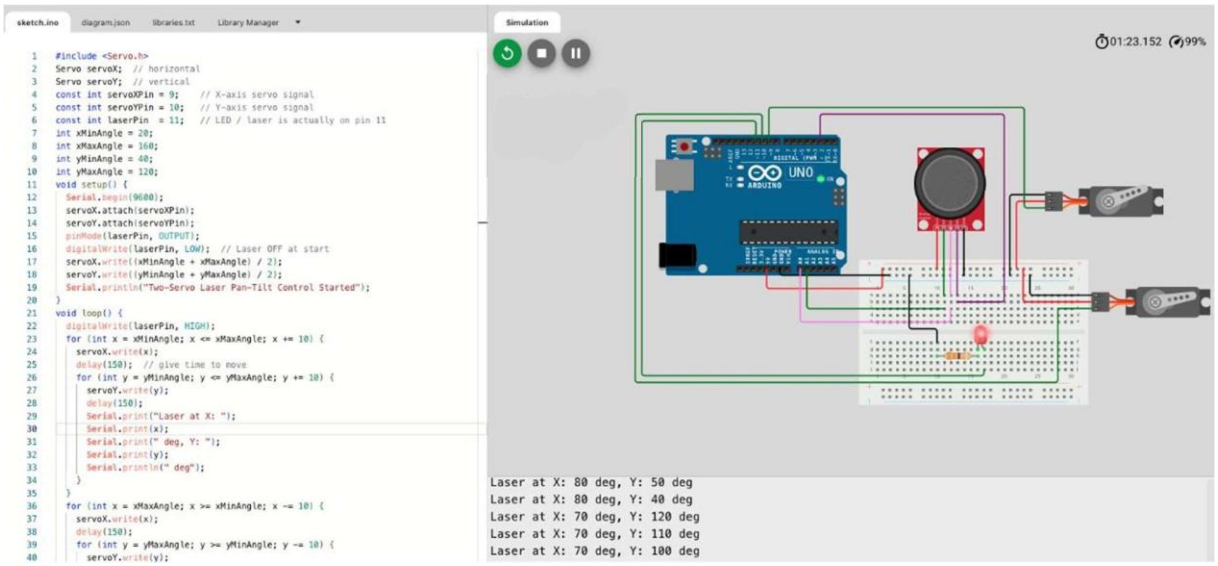
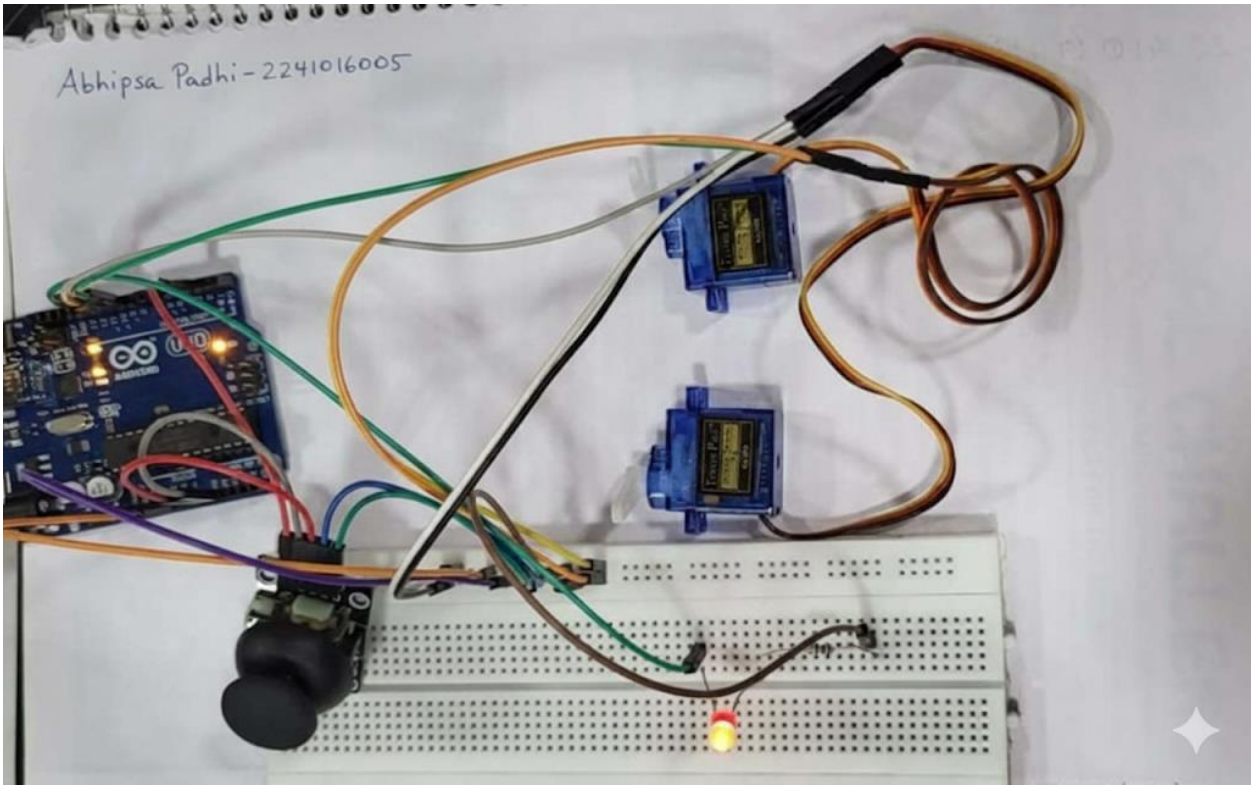


Figure 2.4.1: Simulation based implementing advanced control techniques, such as using two servos to direct a laser beam, to demonstrating the versatility of servo motors in embedded systems on Wokwi.

Figure 2.4.2: Hardware based implementing advanced control techniques, such as using two servos to direct a laser beam, to demonstrating the versatility of servo motors in embedded systems.



Conclusion:

Precautions:

Post Experiment Questionnaire:

1) What is laser beam control using two servo motors, and what are its practical applications?
2) How can advanced control techniques be used to demonstrate the versatility of servo motors in embedded systems?
3) What are the principles of using two servo motors to direct a laser beam, and how is this useful in practical applications?

Answers to Post-Lab Questions

(Signature of the Faculty) **(Signature of the Student)**

Name :

Date: **Registration No. :**

Branch : CSE

Section : 19

PRACTICAL ROBOTIC PROJECTS USING ARDUINO (CSE 4571)

Servo motor control:-To Connect a potentiometer to Arduino UNO to control servo motor position and display the angle on a Display device.

Practical Robotics Projects with Arduino

(CSE 4571)

Lab Assignment No – 07

NeoPixel Ring Programming

Submission Date: _____

Branch: CSE		**Section:**
Name	**Registration No.**	**Signature**

Department of Computer Science and Engineering
Institute of Technical Education and Research (Faculty of Engineering)
Siksha 'O' Anusandhan (Deemed to be University)
Bhubaneswar, Odisha-751030.

Aim:

NeoPixel Ring Programming – To Program a WS2812B NeoPixel LED Strip with Arduino UNO using the Adafruit library for multicolor and brightness effects.

Objectives:

1) **Gain familiarity with the WS2812B NeoPixel LED strip.**
2) **To write an Arduino code that controls individual pixel colours using setPixelColor().**
3) **Write an Arduino sketch to explore brightness control of a WS2812B NeoPixel 16 LEDs in a strip using the setBrightness() function. This function will control the global brightness based on user input from a push button.**
   - 3.1) **Write an Arduino sketch to control the 5 different global brightness levels based on user input from a push button.**
   - 3.2) **Write an Arduino sketch to control the brightness fades UP and DOWN smoothly each time the button is pressed, instead of jumping in steps. (1st press fade UP, 2nd press fade DOWN, 3rd fade UP again and so on).**
4) **To develop an Arduino sketch that generates static multi-colour patterns on the LED strip.**
5) **To develop Arduino code for multi-colour effects such as rainbow, fading, and chasing animations.**

Pre-Lab Questionnaire:

A. Experiment-Specific

1) What is the WS2812B LED commonly called in Adafruit terminology?
2) Which communication protocol is used by WS2812B LEDs?
3) Why is an external resistor (≈330–470 Ω) recommended on the data line of NeoPixels?
4) Why is a capacitor (≈1000 µF) recommended across the LED power supply?
5) What Arduino library is most commonly used to control WS2812B LEDs?
6) Which Arduino UNO pin is commonly used as the data pin for NeoPixels in example codes?
7) How many bytes are required to represent the color of one WS2812B LED?
8) Why must we call pixels.begin() in the setup function?
9) What power supply voltage is typically needed for WS2812B LEDs?
10) What is the typical maximum current consumption of one WS2812B LED at full brightness white?

Answers to Pre-Lab Questions

Components/Equipment Required:

Sl. No.	Name of the Component / Equipment	Specification	Quantity
1)	Arduino UNO R3	16MHz	1
2)	Arduino UNO cable	USB Type A to Micro-B	1
3)	WS2812B NeoPixel 16 LED strip	Any colour of your choice	1
4)	Breadboard	840 Tie points	1
5)	Jumper Wire	----------	As per requirement

Objective 2

To write an Arduino code that controls individual pixel colours using setPixelColor().

Circuit / Schematic Diagram

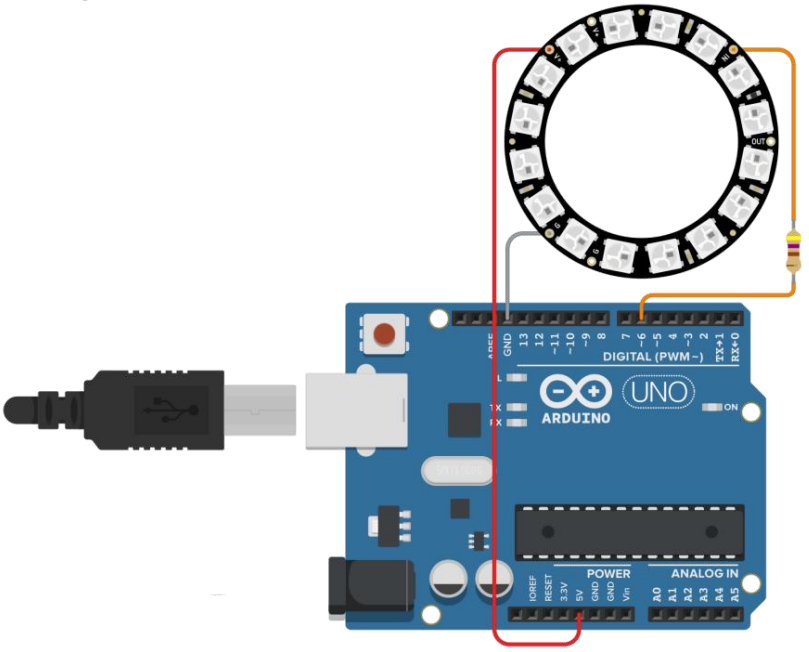


Figure 1: Controls of individual WS2812B NeoPixel colours

Code

Write an Arduino code that controls individual pixel colours using setPixelColor().

```
#include <Adafruit_NeoPixel.h>
#define PIN 6
#define NUMPIXELS 16
Adafruit_NeoPixel strip(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
void setup() {
  strip.begin();
  strip.show();
}
void loop() {
  strip.setPixelColor(0, strip.Color(255, 0, 0));
  strip.setPixelColor(1, strip.Color(0, 255, 0));
  strip.setPixelColor(2, strip.Color(0, 0, 255));
  strip.setPixelColor(3, strip.Color(255, 255, 0));
  strip.setPixelColor(4, strip.Color(0, 255, 255));
  strip.setPixelColor(5, strip.Color(255, 0, 255));
  strip.setPixelColor(6, strip.Color(255, 255, 255));
  strip.show();
  while (1);
}
```

Figure 2: (Simulation-based controls of individual WS2812B NeoPixel colours)

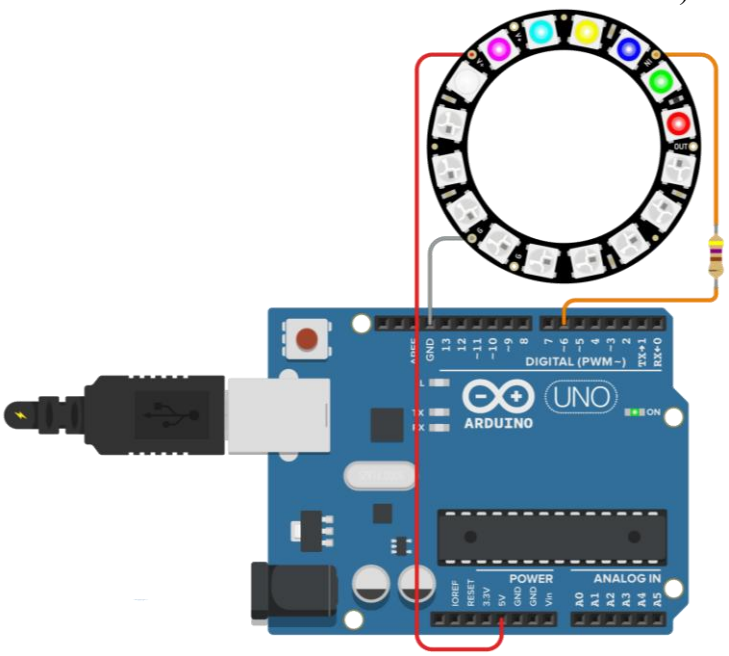
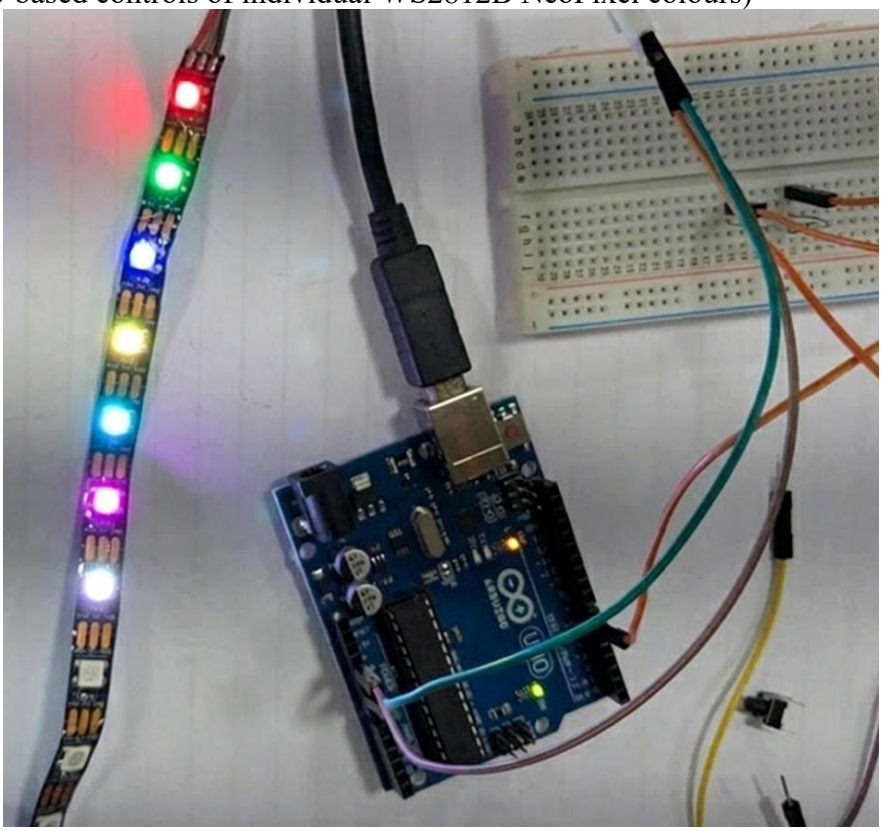


Figure 3: (Hardware-based controls of individual WS2812B NeoPixel colours)



Objective 3

Write an Arduino sketch to explore brightness control of a WS2812B NeoPixel 16 LEDs in a strip using the setBrightness() function. This function will control the global brightness based on user input from a push button.

Code

3.1) Write an Arduino sketch to control the 5 different global brightness levels based on user input from a push button.

```
#include <Adafruit_NeoPixel.h>
#define PIN 6
#define NUMPIXELS 16
#define BUTTON 2
Adafruit_NeoPixel strip(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
int brightnessLevels[] = {20, 60, 120, 180, 255};
int levelIndex = 0;
```

```
void setup() {
  pinMode(BUTTON, INPUT_PULLUP);
  strip.begin();
  strip.show();
  strip.setBrightness(brightnessLevels[levelIndex]);
}
void loop() {
  if (digitalRead(BUTTON) == LOW) {
    levelIndex = (levelIndex + 1) % 5;

strip.setBrightness(brightnessLevels[levelIndex]);
    colorWipe(strip.Color(0, 0, 255), 10);
    delay(300);
  }
}
void colorWipe(uint32_t color, int wait) {
  for (int i = 0; i < strip.numPixels(); i++) {
    strip.setPixelColor(i, color);
    strip.show();
    delay(wait);
  }
}
```

3.2) Write an Arduino sketch to control the brightness fades UP and DOWN smoothly each time the button is pressed, instead of jumping in steps. (1st press fade UP, 2nd press fade DOWN, 3rd fade UP again and so on).

```
#include <Adafruit_NeoPixel.h>
#define PIN 6
#define NUMPIXELS 16
#define BUTTON 2
Adafruit_NeoPixel strip(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
bool fadeUp = true;
bool lastButton = HIGH;
void setup() {
  pinMode(BUTTON, INPUT_PULLUP);
  strip.begin();
  strip.show();
}
void loop() {
  bool currentButton = digitalRead(BUTTON);
  if (lastButton == HIGH && currentButton == LOW) {
    fadeUp = !fadeUp;
    if (fadeUp) fadeBrightness(0, 255);
    else fadeBrightness(255, 0);
  }
  lastButton = currentButton;
}
void fadeBrightness(int start, int end) {
  int step = (start < end) ? 1 : -1;
  for (int b = start; b != end; b += step) {
    strip.setBrightness(b);
    colorShow(strip.Color(255, 100, 0));
    delay(5);
  }
}
void colorShow(uint32_t color) {
  for (int i = 0; i < strip.numPixels(); i++)
    strip.setPixelColor(i, color);
  strip.show();
}
```

Circuit / Schematic Diagram

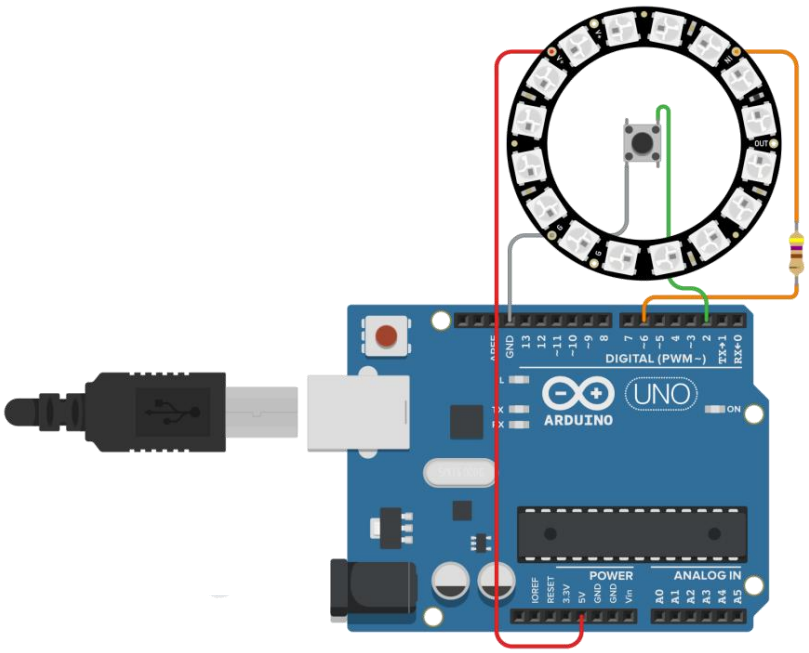


Figure 4: Brightness control of a WS2812B NeoPixel 16 LEDs in a strip

Figure 5: (Simulation-based brightness control of a WS2812B NeoPixel 16 LEDs in a strip)

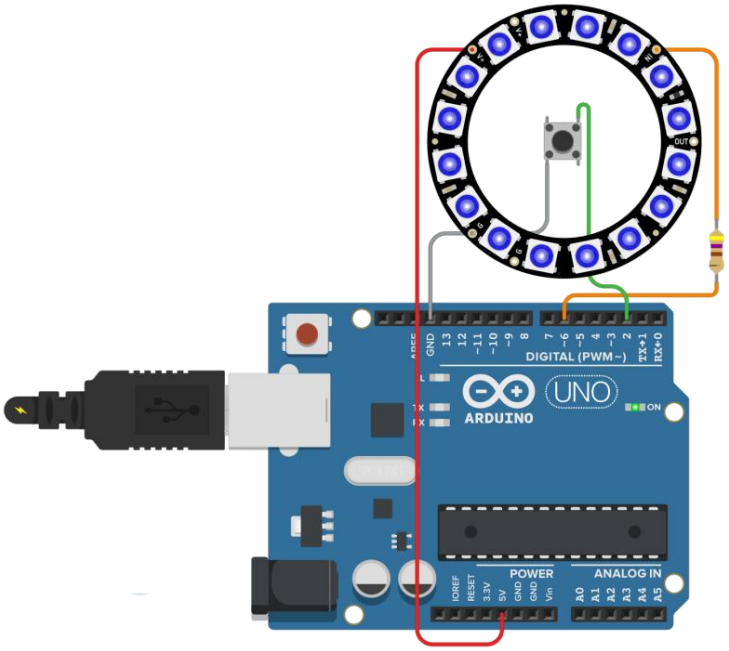
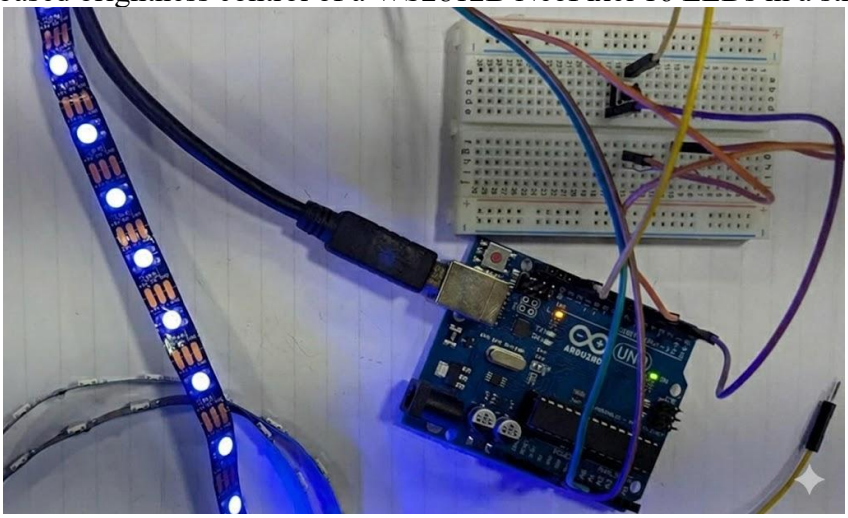


Figure 6: (Hardware-based brightness control of a WS2812B NeoPixel 16 LEDs in a strip)



Objective 4

To develop an Arduino sketch that generates static multi-colour patterns on the LED strip.

Code

Write an Arduino sketch that generates static multi-colour patterns on the LED strip

```
#include <Adafruit_NeoPixel.h>
#define PIN 6
#define NUMPIXELS 16
Adafruit_NeoPixel strip(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
void setup() {
  strip.begin();
  strip.show();
}
void loop() {
  for (int i = 0; i < strip.numPixels(); i++) {
    if (i % 3 == 0) strip.setPixelColor(i, strip.Color(255, 0, 0)); // Red
    else if (i % 3 == 1) strip.setPixelColor(i, strip.Color(0, 255, 0)); // Green
    else strip.setPixelColor(i, strip.Color(0, 0, 255));
  }
  strip.show();
  while (1);
}
```

Circuit / Schematic Diagram

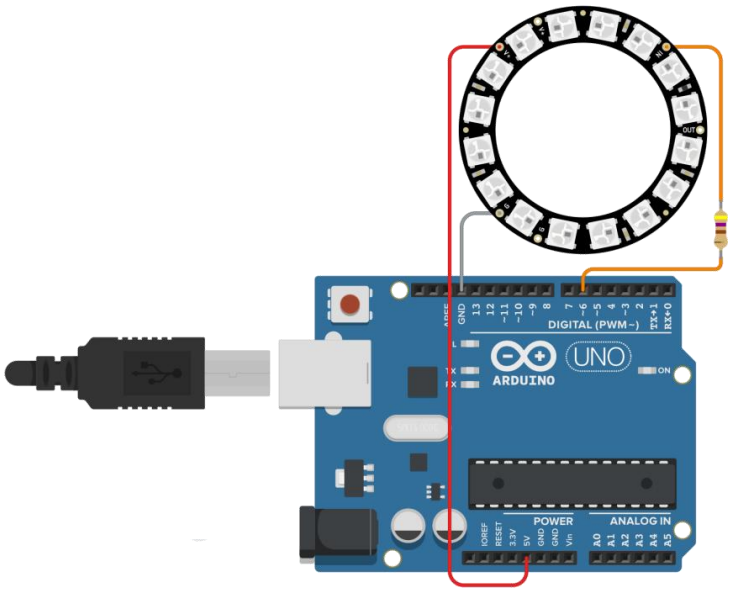


Figure 7: Static multi-colour patterns on the LED strip

Figure 8: (Simulation-based Static multi-colour patterns on the LED strip)

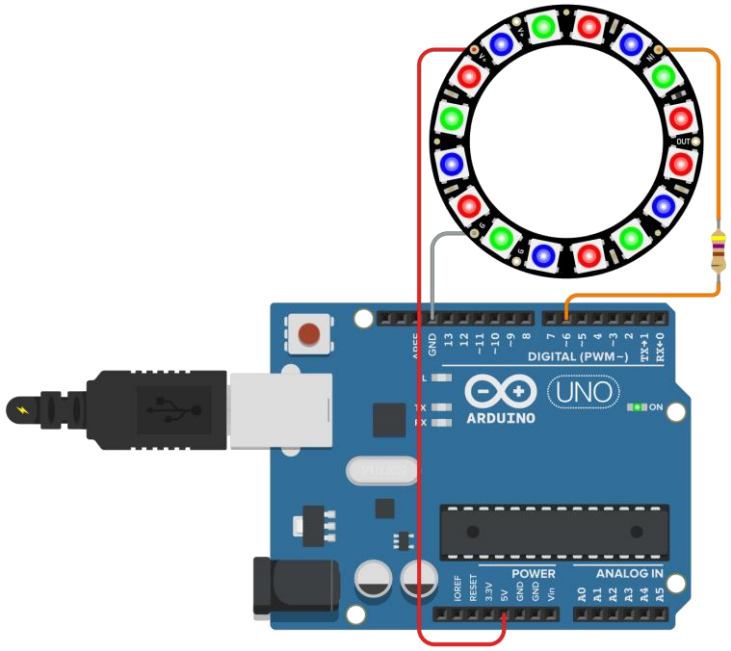
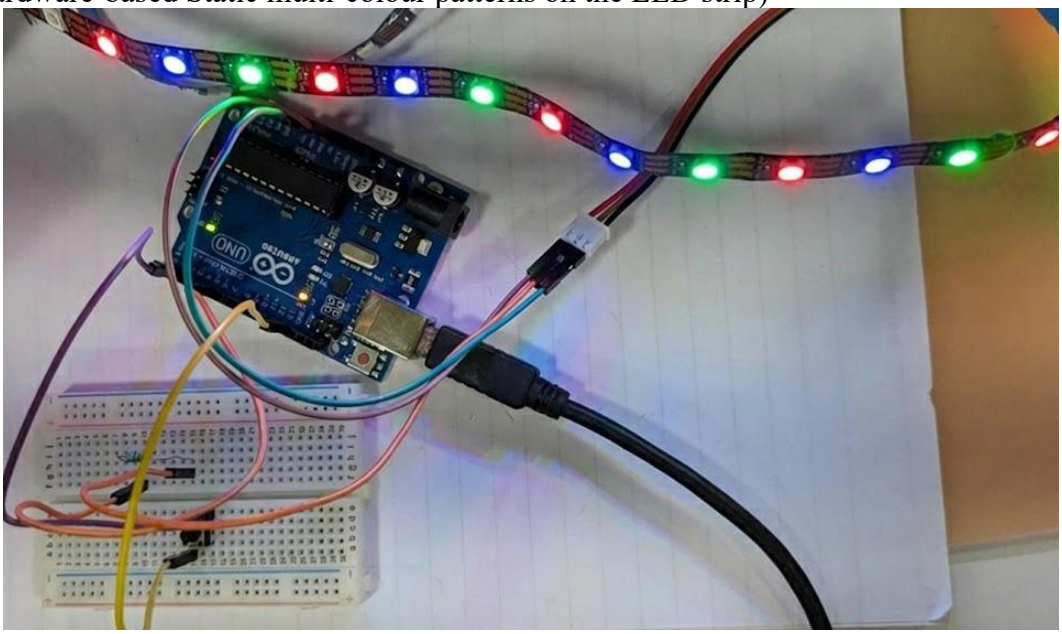


Figure 9: (Hardware-based Static multi-colour patterns on the LED strip)



Objective 5

To develop Arduino code for multi-colour effects, such as rainbow, fading, and chasing animations.

Code

Write an Arduino code for multi-colour effects, such as rainbow, fading, and chasing animations.

```
#include <Adafruit_NeoPixel.h>
#define PIN 6
#define NUMPIXELS 16
Adafruit_NeoPixel strip(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
void setup() {
  strip.begin();
  strip.show();
}
void loop() {
  rainbow(10);     // Rainbow animation
  colorFade(10);   // Smooth fade
  chase(strip.Color(255, 0, 0), 50);  // Red chasing
}
void rainbow(int wait) {
  for (long firstPixelHue = 0; firstPixelHue < 5 * 65536; firstPixelHue += 256) {
    for (int i = 0; i < strip.numPixels(); i++) {
      int pixelHue = firstPixelHue + (i * 65536L / strip.numPixels());
      strip.setPixelColor(i, strip.gamma32(strip.ColorHSV(pixelHue)));
    }
    strip.show();
    delay(wait);
  }
}
void colorFade(int wait) {
  for (int r = 0; r < 256; r++) {
    fillColor(r, 0, 255 - r);
    delay(wait);
  }
  for (int r = 255; r >= 0; r--) {
    fillColor(r, 0, 255 - r);
    delay(wait);
  }
}
void fillColor(int r, int g, int b) {
  for (int i = 0; i < strip.numPixels(); i++)
    strip.setPixelColor(i, r, g, b);
  strip.show();
}
void chase(uint32_t color, int wait) {
  for (int i = 0; i < strip.numPixels(); i++) {
    strip.clear();
    strip.setPixelColor(i, color);
    strip.show();
    delay(wait);
  }

}
```

Circuit / Schematic Diagram

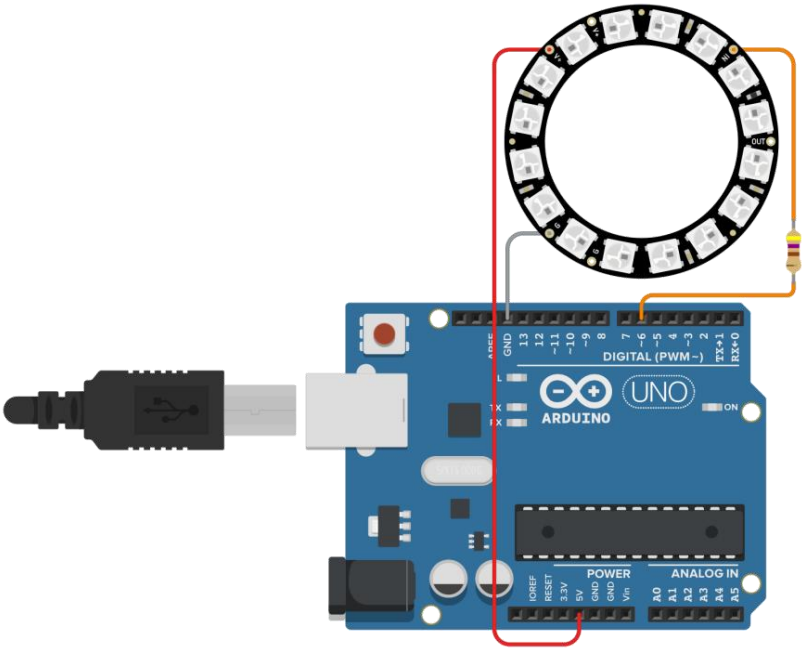


Figure 10: Multi-colour effects on the LED strip animation

Figure 11: (Simulation-based multi-colour effects on the LED strip animation)

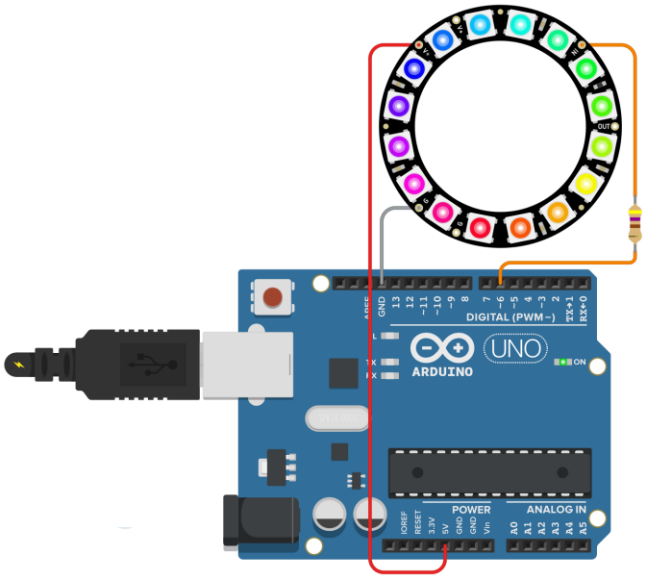
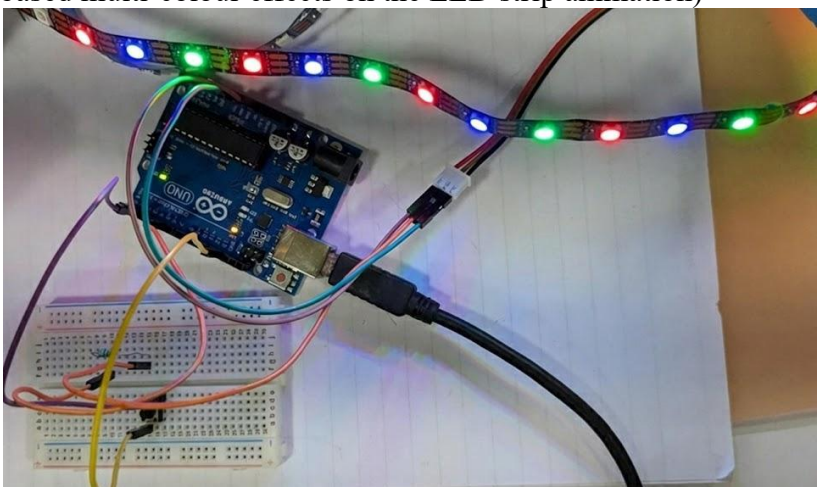


Figure 12: (Hardware-based multi-colour effects on the LED strip animation)



Conclusion

Precautions

Post Experiment Questionnaire:

1) What function is used to actually display the updated color values on the NeoPixel LEDs?
2) How do you set the brightness of all NeoPixels in the strip?
3) Which function is used to assign an RGB color value to a particular pixel?
4) What happens if you set pixel colors but forget to call pixels.show();?
5) What was observed when brightness was set to maximum for multiple LEDs?
6) How can you create a color-changing rainbow animation on NeoPixels?
7) What color order (RGB/GRB) did your NeoPixel strip use in the experiment?
8) What practical limitation did you observe when using Arduino UNO for NeoPixel control?

Answers to Post-Lab Questions

(Signature of the Faculty)	**(Signature of the Student)**
	Name:
Date:	**Registration No.:**
	Branch:
	Section

Practical Robotics Projects with Arduino

(CSE 4571)

Lab Assignment No – 08

Bluetooth Communication

Submission Date: _____

Branch: CSE		**Section:**
Name	**Registration No.**	**Signature**

Department of Computer Science and Engineering
Institute of Technical Education and Research (Faculty of Engineering)
Siksha 'O' Anusandhan (Deemed to be University)
Bhubaneswar, Odisha-751030.

Aim:

To interface an HC-05/HC-06 Bluetooth module with Arduino UNO for wireless serial communication between the Arduino and a Bluetooth-enabled device.

Objectives:

1) **To gain familiarity with the HC-05/HC-06 module and Arduino UNO serial communication.**
   - Learn the pin configuration (VCC, GND, TXD, RXD, EN/KEY) of the HC-05/HC-06 module.
   - Understand the role of baud rate, pairing code, and serial communication protocol.
2) **To establish a basic wireless link between Arduino UNO and a smartphone/PC.**
   - Connect the HC-05/HC-06 module to the Arduino UNO (using SoftwareSerial).
   - Write a simple Arduino sketch to send and receive data over Bluetooth.
   - Pair the module with a smartphone/PC using a Bluetooth terminal app and verify successful two-way communication.
3) **To implement bidirectional communication and LED control.**
   - Build an external circuit with an LED connected to Arduino digital pin 6.
   - Modify the Arduino program to turn the LED ON when the smartphone sends the character 1, and OFF when it sends 0.
   - Send acknowledgment messages back to the smartphone (e.g., “LED ON” / “LED OFF”).
4) **To Evaluate communication performance.**
   - Test and record the effective communication range of the Bluetooth module.
   - Measure response latency between sending a command (from phone) and Arduino execution.

Pre-Lab Questionnaire:

1) Name all the pins of the HC-05/HC-06 module and their functions.
2) Why is a voltage divider required between Arduino TX and HC RX?
3) What is the default baud rate of the HC-05/HC-06 module?
4) Explain the difference between HC-05 and HC-06 modules.
5) What is Software Serial in Arduino, and why is it used in this experiment?
6) What is the purpose of pairing a Bluetooth device before communication?
7) How does Arduino interpret commands sent from a smartphone?
8) How can you verify if the HC-05/HC-06 module is powered and ready for pairing?
9) List the factors that can affect Bluetooth communication range.
10) Explain the role of the Serial Monitor in this experiment.

Answers to Pre-Lab Questions

Components/Equipment Required:

Sl. No.	Name of the Component / Equipment	Specification	Quantity
1)	Arduino UNO R3	16MHz	1
2)	Arduino UNO cable	USB Type A to Micro-B	1
3)	HC-05 / HC-06 Bluetooth Module	Bluetooth 2.0 SPP, 3.3V TTL logic, default baud rate 9600 bps	1
4)	Resistors (carbon type)	220Ω / 2.2kΩ/ 1 kΩ	1 each
5)	LED	Any colour of your choice	1
6)	Breadboard	840 Tie points	1
7)	Smartphone with Bluetooth / PC with Bluetooth	Android/iOS device or Windows PC, for testing wireless serial communication	1
8)	Bluetooth Terminal App	Android/iOS app like "Serial Bluetooth Terminal" or "Bluetooth Terminal	1
9)	Jumper Wire	----------	As per requirement

Objective 1

To Gain familiarity with the HC-05/HC-06 module and Arduino UNO serial.

Circuit / Schematic Diagram

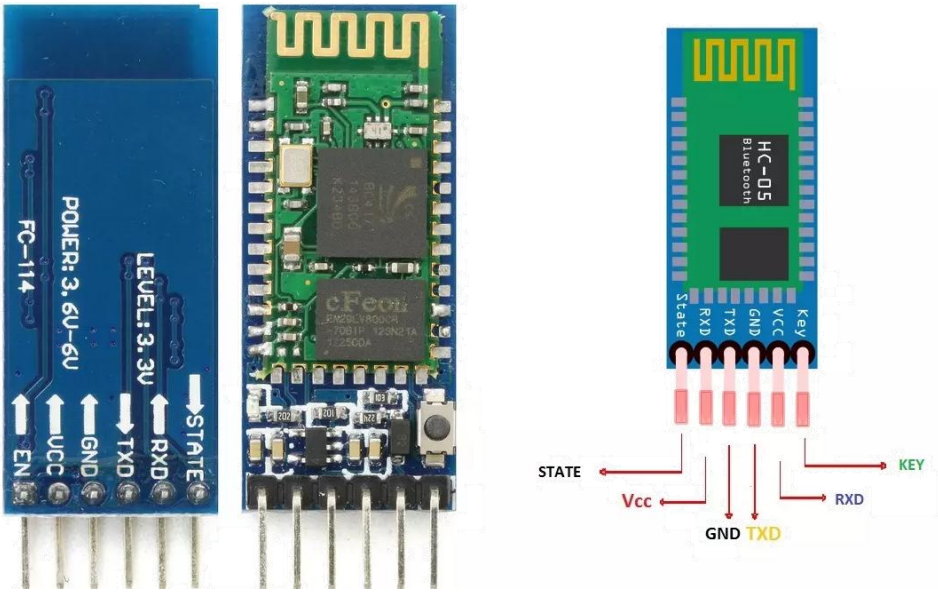


Figure 1: HC-05/HC-06 module Pinout

Objective 2

To establish a basic wireless link between Arduino UNO and a smartphone/PC.

Circuit / Schematic Diagram

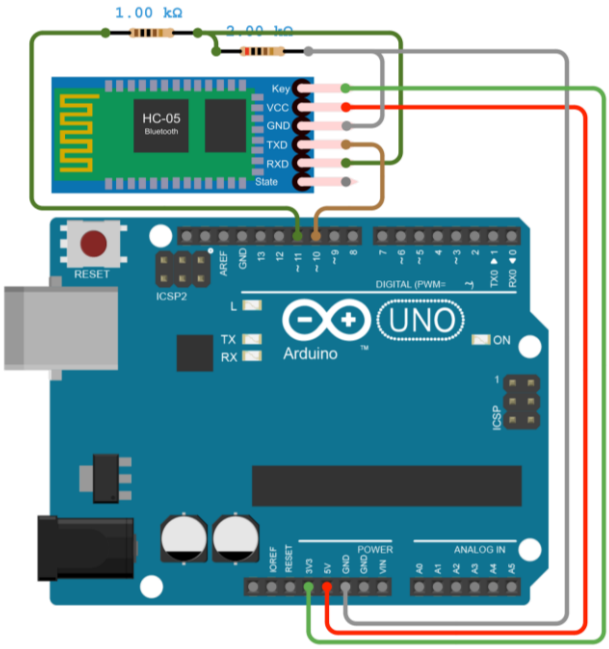


Figure 2: Interface between Arduino UNO and HC-05/HC-06 module

Code

```
#include <SoftwareSerial.h>
SoftwareSerial BT(10, 11); // RX, TX
void setup() {
  Serial.begin(9600);
  BT.begin(9600);
  Serial.println("Ready to communicate. Type on Serial Monitor or Bluetooth app.");
}
void loop() {
  if (Serial.available()) {
    char data = Serial.read();
    BT.write(data);
  }
  if (BT.available()) {
    char data = BT.read();
    Serial.write(data);
  }
}
```

Figure 3: Interface of Arduino UNO with HC-05/HC-06 module

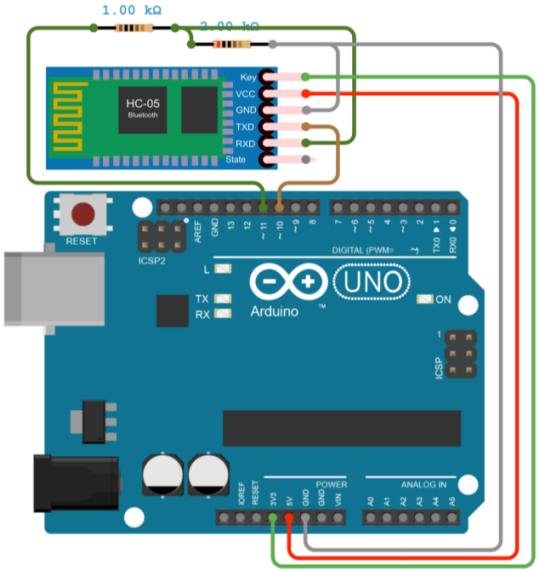
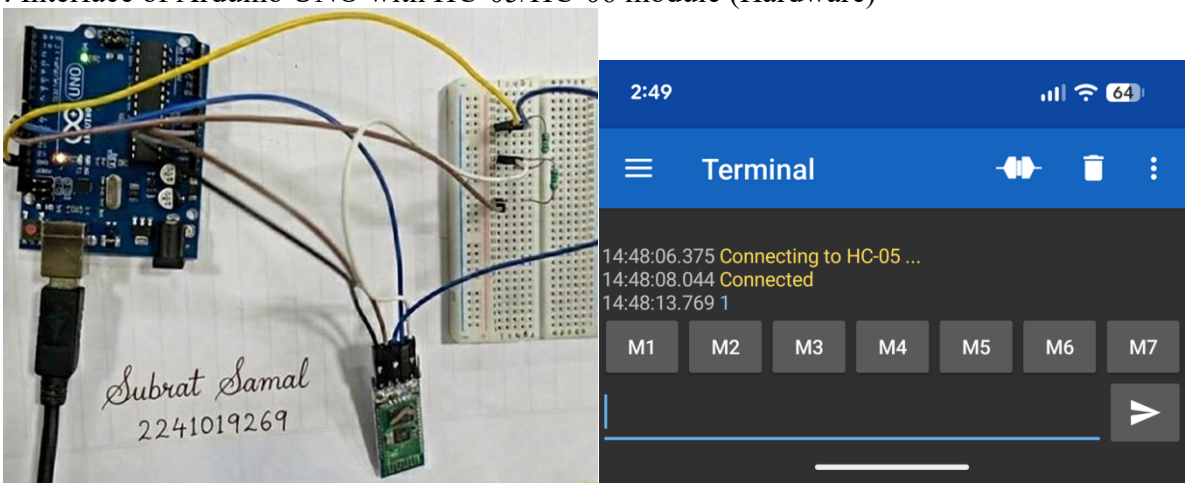


Figure 4: Interface of Arduino UNO with HC-05/HC-06 module (Hardware)



Objective 3

To implement bidirectional communication and LED control.

Circuit / Schematic Diagram

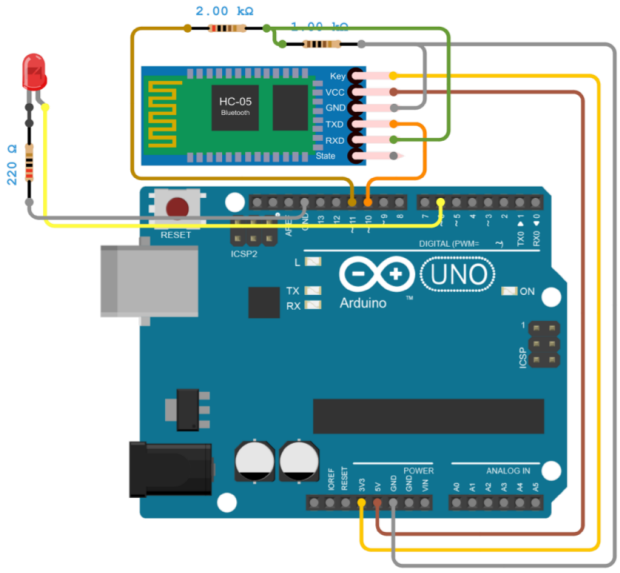


Figure 5: Interface between Arduino UNO and HC-05/HC-06 module

Code

```
#include <SoftwareSerial.h>
SoftwareSerial BT(10, 11); // RX, TX
int led = 6;
void setup() {
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  BT.begin(9600);
  Serial.println("Send '1' to turn ON LED, '0' to turn
OFF.");
}
void loop() {
  if (BT.available()) {
    char cmd = BT.read();
    Serial.print("Received: ");
    Serial.println(cmd);
    if (cmd == '1') {
      digitalWrite(led, HIGH);
      BT.println("LED ON");
      Serial.println("LED turned ON");
    }
    else if (cmd == '0') {
      digitalWrite(led, LOW);
      BT.println("LED OFF");
      Serial.println("LED turned OFF");
    }
    else {
      BT.println("Invalid Command");
    }
  }
}
```

Figure 6: Interface HC-06 and Arduino UNO for bidirectional communication and LED control and confirmation in serial monitor window

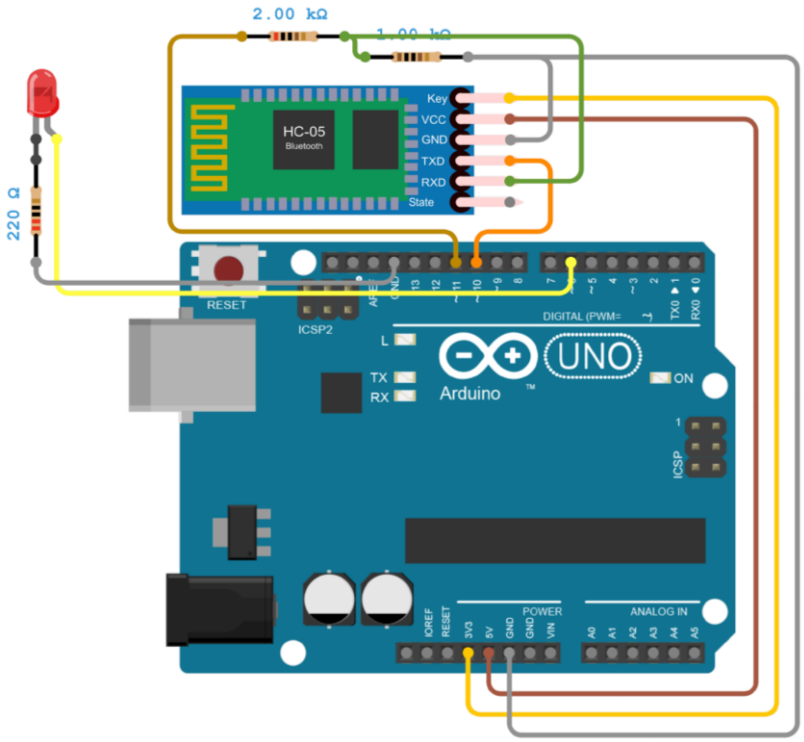
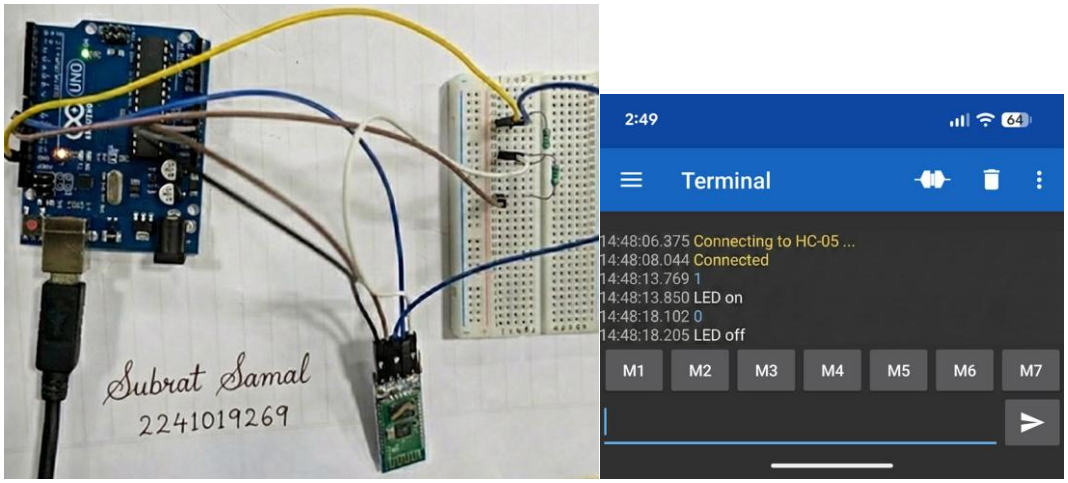


Figure 7: Interface HC-06 and Arduino UNO for bidirectional communication and LED control and confirmation in serial monitor window (Hardware)



Objective 4

To Evaluate communication performance.

Circuit / Schematic Diagram

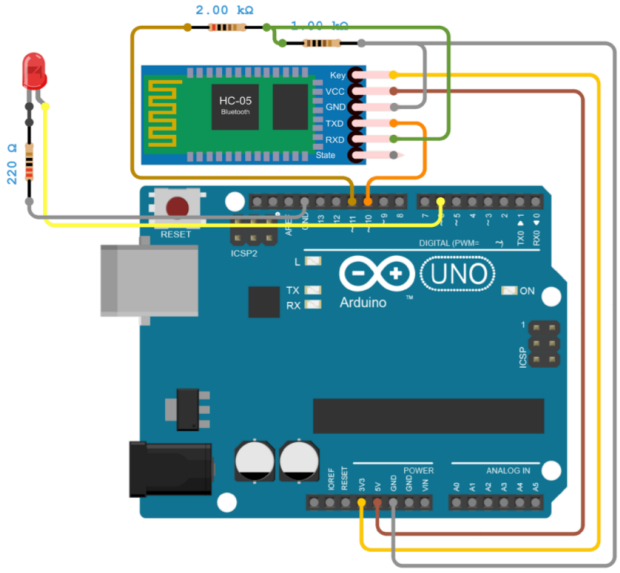
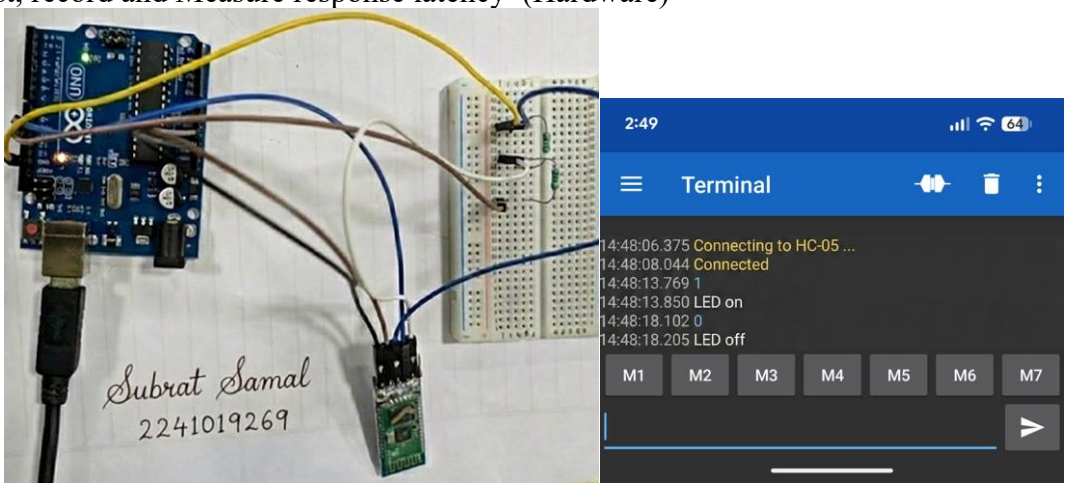


Figure 8: Establish connection with the Serial Monitor and Mobile Phone

Code

```
#include <SoftwareSerial.h>
SoftwareSerial BT(10, 11); // RX, TX
int led = 6;
void setup() {
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  BT.begin(9600);
  Serial.println("Send '1' to turn ON LED, '0' to turn
OFF.");
}
void loop() {
  if (BT.available()) {
    char cmd = BT.read();
    Serial.print("Received: ");
    Serial.println(cmd);
    if (cmd == '1') {
      digitalWrite(led, HIGH);
      BT.println("LED ON");
      Serial.println("LED turned ON");
    }
    else if (cmd == '0') {
      digitalWrite(led, LOW);
      BT.println("LED OFF");
      Serial.println("LED turned OFF");
    }
    else {
      BT.println("Invalid Command");
    }
  }
}
```

Figure 9: Test, record and Measure response latency (Hardware)



Conclusion

Precautions

Post Experiment Questionnaire:

1) How did you establish bidirectional communication between Arduino and a smartphone?
2) Describe the steps to control an LED using Bluetooth commands.
3) What was the maximum reliable communication range achieved during the experiment?
4) How did you implement forwarding between Serial Monitor and Bluetooth?
5) What challenges did you face while pairing the HC-05/HC-06 module?
6) Explain how the Arduino code processes different commands (1 = ON, 0 = OFF).
7) How would you modify the setup to control multiple LEDs with different commands?
8) Compare the observed LED behavior when using Serial Monitor vs. smartphone commands.

Answers to Post-Lab Questions

(Signature of the Faculty)	**(Signature of the Student)**
	Name:
Date:	**Registration No.:**
	Branch:
	Section