Leadership & Excellence

Sri Eshwar College of Engineering

An Autonomous Institution

Accredited by NAAC with 'A' Grade

NBA NATIONAL BOARD of ACCREDITATION For CSE, ECE, EEE, MECH

Name _____

Roll No. _____

Reg No. _____

Class _____

RECORD NOTE BOOK

U23EC383 – Embedded Systems & IoT Laboratory

Sri Eshwar College of Engineering

An Autonomous Institution

Affiliated to Anna University

Kondampatti (Post), Vadasithur (Via), Coimbatore - 641 202.

Department of

.....

Record work of ..... Laboratory

Certified bonafide Record of work done by

Name : ..... Roll No. : .....

Class : ..... Branch : .....

Reg. No. : .....

Place : *HOD* *Faculty In charge*

Date :

University Register No. .....

Submitted for the University Practical Examination held on

.....

Internal Examiner *External Examiner*

Instructions for LABORATORY Classes

- Enter the Lab with observation, record notebook, calculator & necessary tools
- Enter the Lab with **CLOSED FOOTWEAR**
- Maintain Silence during the Lab Hours
- Boys should **"TUCK IN"** the Shirts
- Girls are instructed to wear **OVERCOATS**
- H**ANGING CHAINS, RINGS, WRIST WATCHES** and like items are likely to cause accidents and hence are to be **AVOIDED**
- **LONG HAIR** should be protected, let it not be loose especially near **ROTATING MACHINERY**
- Any machine / equipment **SHOULD NOT BE OPERATED** other than the prescribed one for the day
- Read and follow the work instructions inside the laboratory
- **POWER SUPPLY** to your table should be obtained only through the **LAB TECHNICIAN**
- Do not **LEAN** and do not be **CLOSE** to the rotating components
- Handle the equipments with care
- **TOOLS & GAUGE** sets are to be **RETURNED** before leaving the lab
- **ALL SUB - HEADING** and **DETAILS** should be neatly written on the right hand side page of the record
- For example

  Aim of the experiment

  Apparatus / Tools / Instruments required

  Procedure

  Model calculation

  Results / discussions / inference
- The sketches, block diagrams, circuits and flow charts are to be drawn on the left side blank page
- For example

  Neat Diagram

  Specifications / Design Details

  Tabulations

  Graph
- **GRAPHS / FIGURES** drawn on separate sheets in special cases, should be firmly pasted on to the record.
- Before writing the record, the student should get the corresponding observations signed by the **FACULTY - IN- CHARGE**
- The observation and record should be completed in all respects and submitted in the very **NEXT CLASS** itself.
- Begin every experiment on a **NEW PAGE**
- Write the **NAME OF EXPERIMENT** in Capital letters on the top of the page
- Experiment number with date should be written at the top left hand corner
- Be **PATIENT, STEADY, SYSTEMATIC** and **REGULAR**
- Follow the dress code for Laboratory classes
- Maintain punctuality

INDEX

S.No	Date	Name of the Experiment	Page No.	Aim, Circuits Requirements, (15)	Procedure & Code (30)	Output & Inference (20)	Viva (10)	Total (75)	Faculty sign
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									

..... *Laboratory*

Name of the Student :

Roll No :

Year :

Semester :

Description	*I Month*	*II Month*	*III Month*
Number of Experiments Proposed			
Number of Experiments done			
Reason for Deviation (if any)			
Remark by the Faculty Member			

VISION OF THE DEPARTMENT

To groom students into globally competent software professionals and meet the ever changing requirements of the industry.

MISSION OF THE DEPARTMENT

- Creating a quality academic environment with relevant IT infrastructure and empowering faculty and students with emerging technologies.
- Motivating staff and students to actively involve in lifelong learning and fostering research.
- Inculcating leadership and entrepreneurship skills in students.
- Generating opportunities for students to evolve as competent software professionals with societal consciousness.

PROGRAM OUTCOMES

PO1: Engineering Knowledge: Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop the solution of **complex engineering problems**.

PO2: Problem Analysis: Identify, formulate, review research literature and analyze **complex engineering problems** reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)

PO3: Design/Development of Solutions: Design creative solutions for **complex engineering problems** and design/develop systems/components/processes to meet identified needs with consideration for public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5)

PO4: Conduct Investigations of Complex Problems: Conduct investigations of **complex engineering problems** using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).

PO5: Engineering Tool Usage: Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve **complex engineering problems**. (WK2 and WK6)

PO6: The Engineer and The World: Analyze and evaluate societal and environmental aspects while solving **complex engineering problems** for its impact on sustainability with reference to the economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).

PO7: Ethics: Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)

PO8: **Individual and Collaborative Teamwork:** Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.

PO9: Communication: Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences

PO10: **Project Management and Finance:** Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.

PO11: Life-Long Learning: Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. (WK8)

Program Educational Objectives (PEO)

PEO1	To prepare graduates for a career in software engineering
PEO2	To prepare students for higher studies, research, entrepreneurial and leadership roles by imparting the quality of lifelong learning
PEO3	To enable students to apply innovative solutions for real-life problems in computer science domain.

Program Specific Outcome (PSO)

PSO1	Demonstrate knowledge in open source technologies
PSO2	Develop innovative solutions by adapting emerging technologies for industry oriented applications
PSO3	Implement SDLC principles for project/product development

Course Outcomes Cos:

U23EC383.1	Analyze and synthesize the architecture of Embedded Systems, delineating its constituent components.
U23EC383.2	Utilize conceptual understanding to apply the design process of Embedded Systems.
U23EC383.3	Apply advanced cognitive skills to conceptualize the architecture of IoT systems and develop basic applications employing embedded hardware.
U23EC383.4	Apply higher-order cognitive abilities to implement Cloud services in IoT applications..
U23EC383.5	Generate innovative solutions by integrating Node MCU and Raspberry Pi boards to design IoT applications.

CO & PO/PSO Mapping:

COURSE	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PSO 1	PSO 2	PSO 2
U23EC383.1	3	2	3	3	-	-	-	-	1	1	1	3	1	-
U23EC383.2	3	2	3	2	3	-	-	-	1	1	2	3	2	-
U23EC383.3	3	3	2	3	1	-	-	-	1	1	2	3	3	-
U23EC383.4	3	3	2	1	-	-	-	-	1	1	3	3	1	-
U23EC383.5	3	3	3	2	1	-	-	-	1	1	3	3	3	-

List of Experiments

1. Interfacing Arduino with Pot and Servo motor.

2. Interfacing Arduino with Stepper Motor.

3. Interfacing Arduino with Analog/Digital sensors.

4. Write a program to interface LED/Buzzer with Arduino and to turn ON LED for 1 second after every 2 seconds.

5. Write a program to interface Digital sensor PIR with Arduino and to turn ON LED when motion is detected.

6. Write a program to interface DHT22 sensor with Arduino and display temperature and humidity readings.

7. Write a program to interface the motor using relay with Raspberry Pi and turn ON the motor when the temperature is high.

8. Write a program to interface LCD with Raspberry Pi and print temperature and humidity readings on it.

9. Write a program to interface a flame/smoke sensor with Arduino/Raspberry Pi and give an alert message when flame/smoke is detected.

10. Implement any case study using Arduino/Raspberry Pi.

Safety Precautions:

- Do not touch any exposed wires. Work with a dry hand.
- Do not open up equipment casing.
- Do not unplug the cable while the power is switched on.
- Do not wear long, loose ties, scarves or other loose clothing. Make sure that long hairs are tied.

DATE :	Interfacing Arduino with Pot and Servo motor
EXP NO : 1	

Aim:

To control the position of a servo motor based on the analog input from a potentiometer by mapping the sensor values to servo rotation angles.

Components Required:

Arduino Board

Servo Motor

Potentiometer

Connecting wires

Breadboard

Theory:

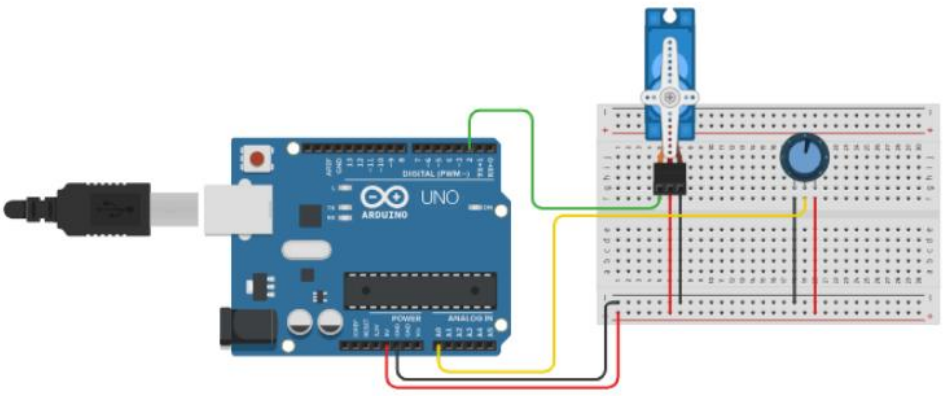
A servo motor rotates to a specific angle between 0° and 180° based on the PWM signal sent from the Arduino.

The potentiometer provides an analog voltage input between 0V and 5V, read by the Arduino as a value between 0 and 1023.

This analog value is mapped linearly to the servo's rotation range (0 to 180 degrees).

The servo motor moves to the corresponding angle based on the potentiometer position.

Circuit Diagram:



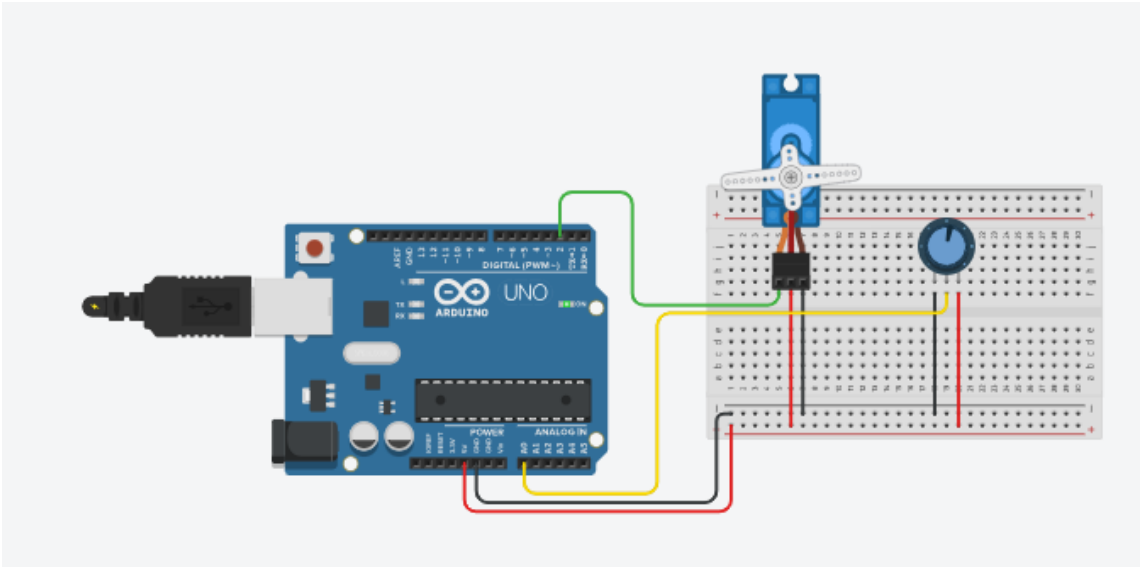
Procedure:

1. Connect the servo motor signal wire to digital pin 2 on the Arduino.
2. Connect the potentiometer middle pin to analog pin A0 on the Arduino, and the other two pins to 5V and GND.
3. Connect the servo motor's power and ground pins to 5V and GND respectively.
4. Upload the program to the Arduino.
5. Rotate the potentiometer knob; the Arduino reads the analog value.
6. The analog value is mapped to a rotation angle between 0° and 180°.
7. The servo motor shaft rotates to the mapped angle, mimicking the potentiometer position.
8. . The servo rotates proportionally as the potentiometer knob is turned.

Code:

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

int servoPin = 2;
int potPin = A0;
int value = 0;
int rot = 0;

void setup()
{
  pinMode(A0, INPUT);
  myservo.attach(2);
}
void loop()
{
  value=analogRead(A0);
  rot=map(value, 0, 1023, 0, 180);

  myservo.write(rot);
}
```

Output:



Result:

DATE :	Interfacing Arduino with Stepper Motor
EXP NO : 2	

Aim:

To control the speed of a stepper motor based on the analog input from a sensor (e.g., a potentiometer or light sensor) by varying the motor speed dynamically using the Arduino Stepper library.

Components Required:

Arduino Board

Stepper Motor (with 200 steps per revolution)

Stepper Motor Driver or ULN2003 driver board

Potentiometer or Analog Sensor (connected to analog pin A0)

Power supply suitable for the stepper motor

Connecting wires and breadboard

Theory:

A Stepper Motor moves in discrete steps, controlled by digital pulses.

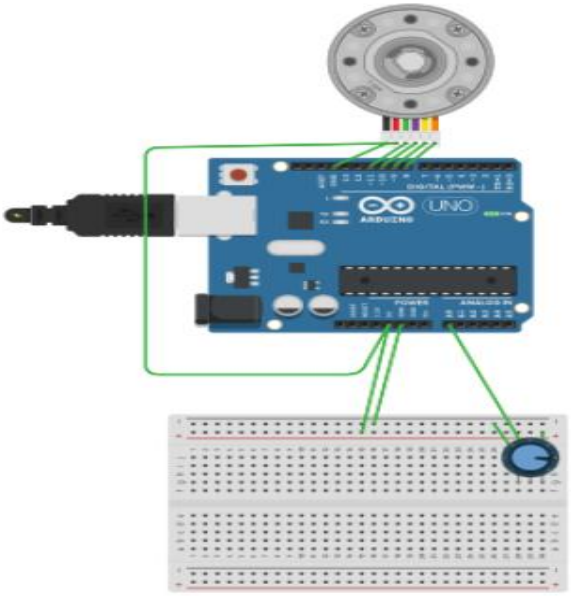
The Stepper library in Arduino helps manage the sequence of pulses required to rotate the motor in precise steps.

The analog sensor connected to pin A0 provides a varying voltage which is read as an analog value between 0 and 1023.

The sensor reading is mapped to a speed value (in RPM) to dynamically control the speed of the stepper motor.

The motor speed is updated continuously based on sensor input, allowing smooth speed control.

Circuit Diagram:



Procedure:

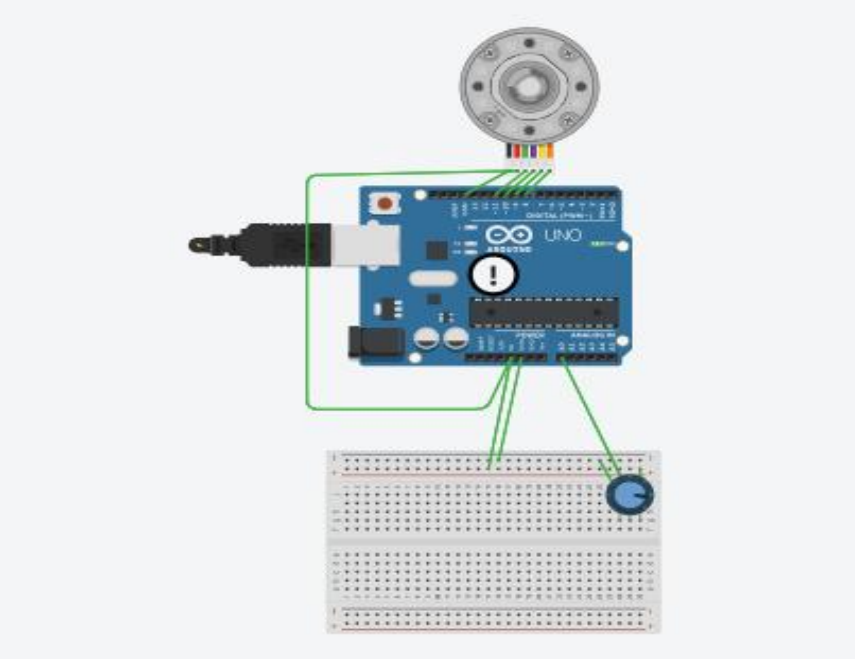
1. Connect the stepper motor to Arduino pins 8, 9, 10, and 11 through the driver module.Connect the analog sensor (e.g., potentiometer) to analog input pin A0.
2. Power the stepper motor with an appropriate external power supply if needed.
3. Upload the program to the Arduino board.
4. Rotate or adjust the sensor connected to A0 to change the analog reading.
5. The stepper motor speed will change accordingly; the higher the sensor value, the faster the motor spins.
6. Observe the motor rotating at speeds proportional to the sensor input.

Code:

```
#include <Stepper.h>

const int stepsPerRevolution = 200;

// Initialize the Stepper library on pins 8, 9, 10, and 11:
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);

void setup() {
  // Set the initial speed of the stepper motor:
  myStepper.setSpeed(0);
}
void loop() {
```

```
  int sensorReading = analogRead(A0);

  // Map the sensor reading to a speed between 0 and 100 RPM:
  int motorSpeed = map(sensorReading, 0, 1040, 0, 150);

  // Set the speed of the stepper motor:
  myStepper.setSpeed(motorSpeed);

  // Move the stepper motor by a certain number of steps:
  myStepper.step(stepsPerRevolution / 100);

  // Add a delay to allow the motor to complete its movement before reading again:
  delay(100);
}
```

Output:



Result:

DATE :	Interfacing Arduino with Analog/Digital sensors
EXP NO : 3	

Aim:

To control a relay based on the input from a PIR motion sensor and an LDR (Light Dependent Resistor) sensor such that the relay is activated only when motion is detected and the ambient light level is below a certain threshold.

Components Required:

Arduino Board (e.g., Arduino Uno)

PIR Motion Sensor

LDR (Light Dependent Resistor)

Relay Module

Resistor (typically 10kΩ for the LDR voltage divider)

Connecting wires and breadboard

Power supply for Arduino and relay

Theory:

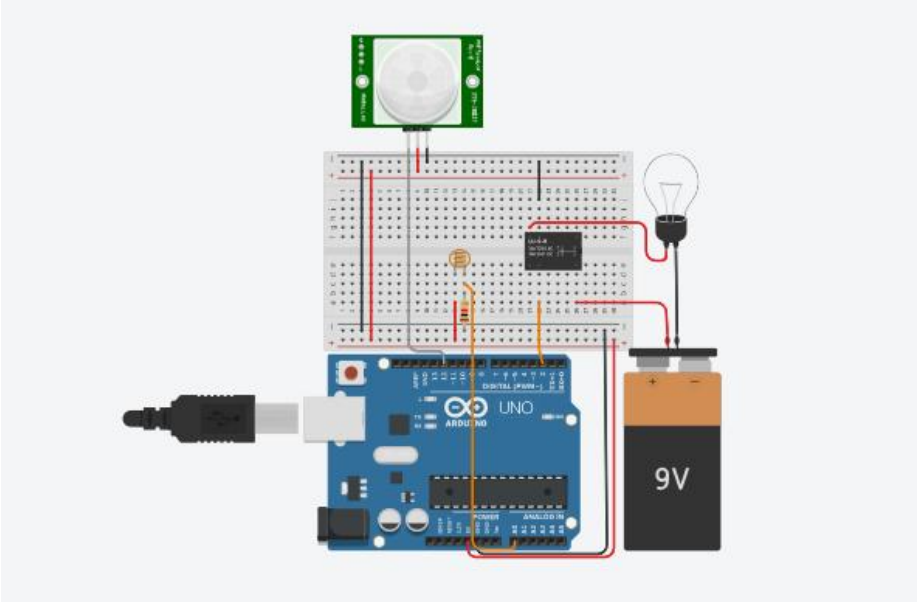
PIR Sensor: Detects motion based on changes in infrared radiation emitted by objects in its field of view.

LDR Sensor: Changes resistance based on light intensity; resistance decreases with increasing light.

Relay: An electrically operated switch that can control a high-power device using a low-power signal.

The program reads values from both sensors. If the light intensity is low (below a threshold), and motion is detected by the PIR sensor, the relay is switched ON for 5 seconds. This setup can be used in applications like automatic lighting control in dark environments where lights turn on only when someone is present.

Circuit Diagram:



Procedure:

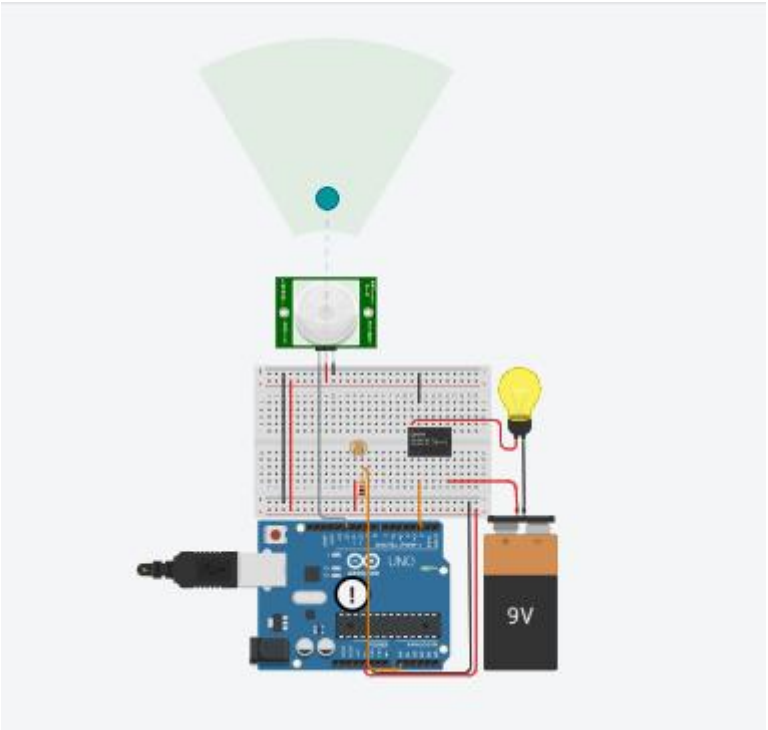
1. Connect the PIR sensor output to digital pin 12 of Arduino.
2. Connect the LDR sensor as a voltage divider circuit to analog pin A0 of Arduino (LDR connected in series with a resistor).
3. Connect the relay module input to digital pin 2 of Arduino.
4. Upload the provided code to the Arduino board.
5. Open the Serial Monitor to observe LDR sensor readings.
6. When the environment is dark (LDR value below threshold 500), and motion is detected by the PIR sensor, the relay activates, switching on the connected device for 5 seconds.
7. If the light level is high or no motion is detected, the relay remains OFF.

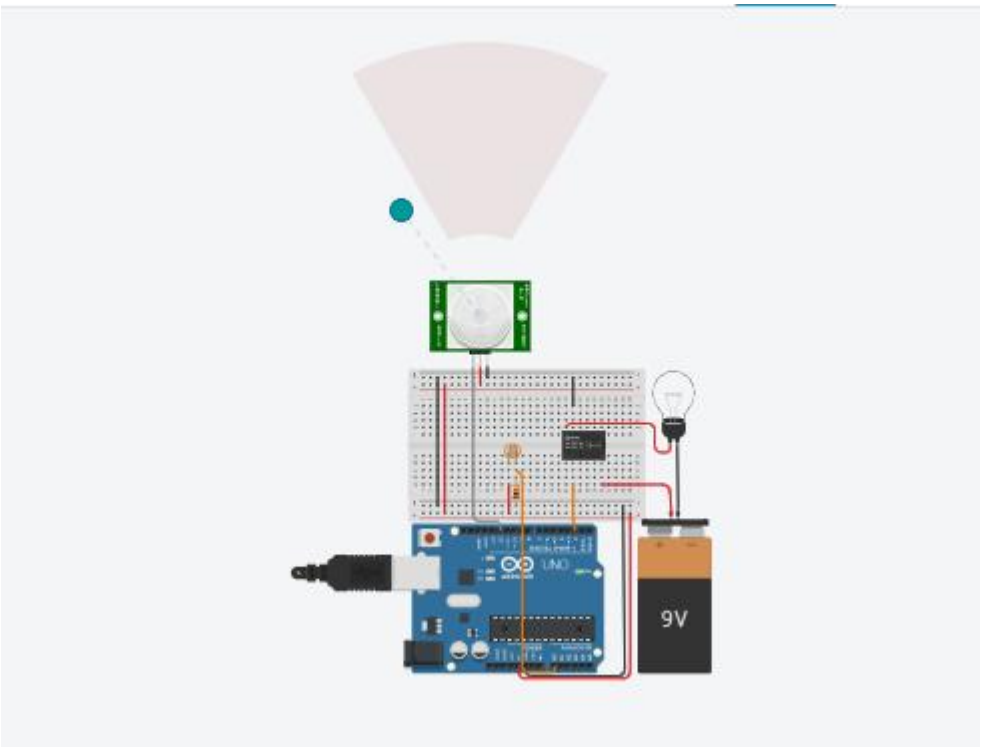
Code:

```
int pirSensor = 12;
int ldrSensor = A0;
int relay = 2;

int ldrValue, pirValue;
unsigned long currentMillis;
void setup()
{
  Serial.begin(9600);
```

```
  pinMode(pirSensor, INPUT);
  pinMode(ldrSensor, INPUT);
  pinMode(relay, OUTPUT);
}

void loop()
{
  ldrValue = analogRead(ldrSensor);
  pirValue = digitalRead(pirSensor);
  ldrValue = map(ldrValue, 6, 679, 0, 1023);
  ldrValue = constrain(ldrValue, 0, 1023);
  Serial.println(ldrValue);

  if(ldrValue < 500){
    if(pirValue == LOW){
                        digitalWrite(relay, LOW);
    } else{
      digitalWrite(relay,HIGH);
      delay(5000);
   }
  }
  else if(ldrSensor >= 500){
    if(pirValue == LOW){
                        digitalWrite(relay, LOW);
    } else{
      digitalWrite(relay,LOW);
    }
  }
}
```

Output:





Result:

DATE :	Interfacing LED/Buzzer with Arduino
EXP NO : 4	

Aim:

To interface an LED/Buzzer with Arduino and to turn ON the LED for 1 second after every 2 seconds.

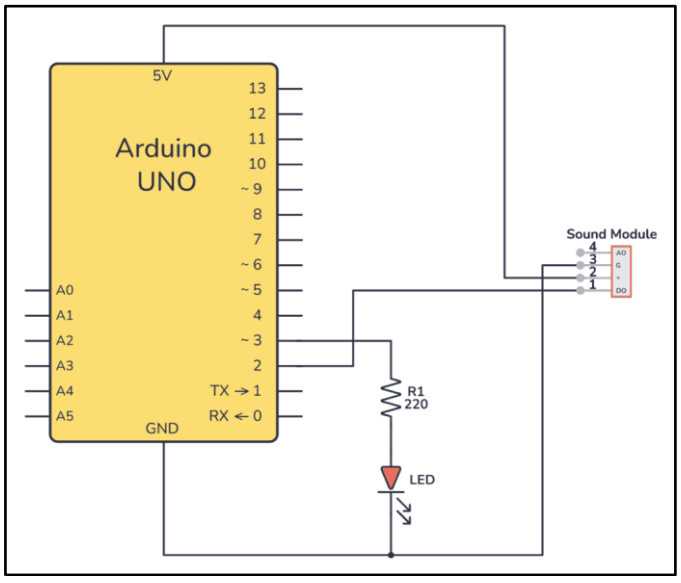
Components Required:

S.No	Component	Specification	Quantity
1	Arduino Uno	ATmega328P-based microcontroller board	1
2	LED	5mm Red LED (general purpose)	1
3	Resistor	220Ω	1
4	Buzzer	5V DC Piezo Buzzer	1
5	Jumper Wires	Male-to-Male	As required
6	Breadboard		1
7	USB Cable	Type A to B	1

Theory:

An LED (Light Emitting Diode) is an electronic component that emits light when current passes through it. A Buzzer converts electrical energy into sound through vibration of a diaphragm. The Arduino UNO controls the LED or buzzer using its digital output pins. By writing a simple program, we can switch the output pin HIGH (to turn ON LED) and LOW (to turn OFF LED) with a time delay. The delay() function in Arduino introduces a pause in milliseconds (1000 ms = 1 second).

Circuit Diagram:



Procedure:

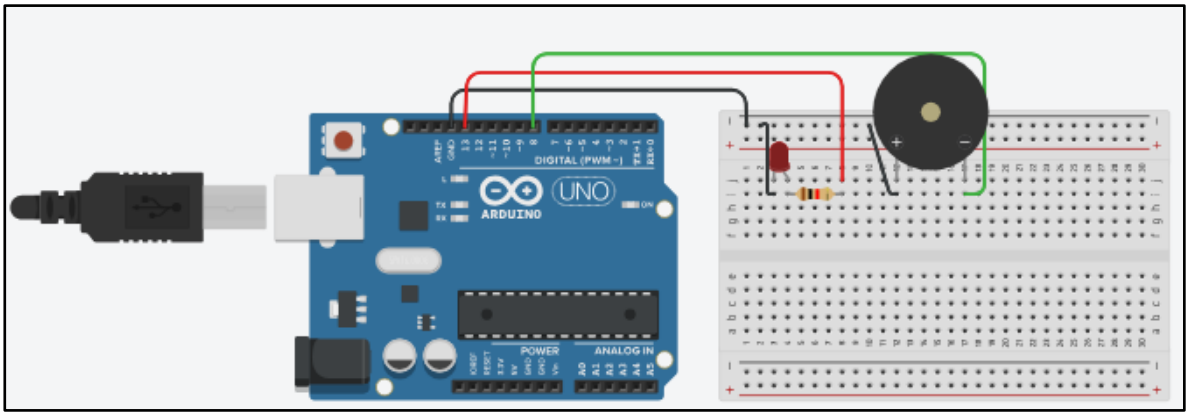
1. Start a new project: Open Tinkercad and create a new "Circuit."
2. Gather components: Add an Arduino Uno, a breadboard, one LED, one 220-ohm resistor, a passive piezo buzzer, and a few jumper wires to your workspace.
3. Place the components: Place the Arduino and breadboard on the workspace.
4. Place the LED on the breadboard.
5. Connect the longer leg (anode) of the LED to a digital pin on the Arduino (e.g., pin 13).
6. Connect the shorter leg (cathode) of the LED to one end of the 220-ohm resistor.
7. Connect the other end of the resistor to a ground (GND) pin on the Arduino.
8. Place the buzzer on the breadboard.
9. Connect the positive terminal of the buzzer to a different digital pin on the Arduino (e.g., pin 8).
10. Connect the negative terminal of the buzzer to a ground (GND) pin on the Arduino
11. Open the **code editor** in Tinkercad.
12. Select **Blocks + Text** or **Text** mode to enter the code below.
13. Click **Start Simulation** to observe the LED blinking (ON for 1 sec, OFF for 2 sec)

Code:

```
// Define the pins for the LED and buzzer
const int ledPin = 13; // Connect the LED to digital pin 13
const int buzzerPin = 8; // Connect the buzzer to digital pin 8

void setup()  {
  // Set the LED pin as an output
  pinMode(ledPin, OUTPUT);

  // Set the buzzer pin as an output
  pinMode(buzzerPin, OUTPUT);
}

void loop() {
  // Turn the LED on
  digitalWrite(ledPin, HIGH);

  // Turn the buzzer on (play a tone)
  tone(buzzerPin, 1000); // 1000 Hz tone

  // Wait for 1 second (1000 milliseconds)
  delay(1000);

  // Turn the LED off
  digitalWrite(ledPin, LOW);

  // Turn the buzzer off (stop the tone)
  noTone(buzzerPin);
```

```
    // Wait for 2 seconds (2000 milliseconds)
    delay(2000);
}
```

Output :



Observation:

- The LED turns **ON for 1 second** and then remains **OFF for 2 seconds**, repeating continuously.
- If a buzzer is used, it produces a **beep sound** for 1 second after every 2 seconds.

Result:

DATE :	Interfacing Digital sensor PIR with Arduino
EXP NO : 5	

Aim:

To interface a Digital PIR (Passive Infrared) Sensor with Arduino and to turn ON an LED when motion is detected**.**

Components Required:

S.No	Component	Specification	Quantity
1	Arduino Uno	ATmega328P-based microcontroller board	1
2	PIR Motion Sensor	Passive Infrared (PIR) Sensor	1
3	LED	5mm Red LED (general purpose)	1
4	Resistor	220Ω	1
5	Jumper Wires	Male-to-Male	As required
6	Breadboard	-	1

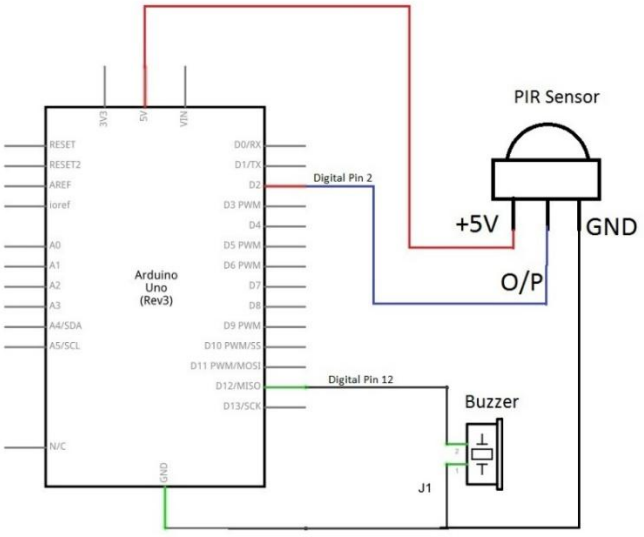
Theory:

The PIR sensor detects infrared radiation emitted by humans or animals. A PIR (Passive Infrared) sensor detects motion by measuring changes in infrared (heat) radiation in its surroundings. i) When a human or animal moves in front of it, the sensor outputs a HIGH (logic 1) signal. ii) When there is no motion, the output remains LOW (logic 0). This principle is widely used in security systems, automatic lighting, and intrusion detection.

The Arduino continuously monitors this digital output signal:

- If **HIGH**, Arduino turns **ON the LED**.
- If **LOW**, Arduino turns **OFF the LED**.

Circuit Diagram:



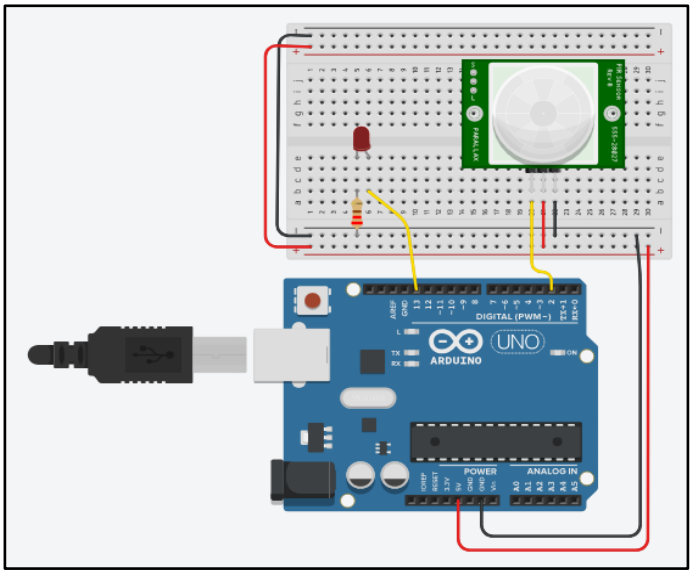
Procedure:

1. Start a new project: Open Tinkercad and create a new "Circuit."
2. Gather components: Add an Arduino Uno, a breadboard, one LED, one 220-ohm resistor, a passive piezo buzzer, and a few jumper wires to your workspace.
3. Place the components: Place the Arduino and breadboard on the workspace.
4. Place the LED on the breadboard.
5. Connect the longer leg (anode) of the LED to a digital pin on the Arduino (e.g., pin 13).
6. Connect the shorter leg (cathode) of the LED to one end of the 220-ohm resistor.
7. Connect the other end of the resistor to a ground (GND) pin on the Arduino.
8. Place the buzzer on the breadboard.
9. Connect the positive terminal of the buzzer to a different digital pin on the Arduino (e.g., pin 8).
10. Connect the negative terminal of the buzzer to a ground (GND) pin on the Arduino
11. Open the code editor in Tinkercad.
12. Select Blocks + Text or Text mode to enter the code below.
13. Click Start Simulation to observe the LED blinking (ON for 1 sec, OFF for 2 sec)

Code:

```
/*
  PIR Motion Sensor with LED in Tinkercad
  This program reads the state of a PIR motion sensor.
  When motion is detected, it turns on an LED.
*/

// Define the digital pins for the PIR sensor and the LED
const int pirPin = 2;   // Digital pin connected to the PIR sensor's OUT pin
const int ledPin = 13;  // Digital pin connected to the LED's anode (via a resistor)

void setup() {
  // Set the LED pin as an OUTPUT
  pinMode(ledPin, OUTPUT);
  // Set the PIR sensor pin as an INPUT
  pinMode(pirPin, INPUT);
  // Initialize the Serial Monitor for debugging
  Serial.begin(9600);
}

void loop() {
  // Read the state of the PIR sensor
  // It returns HIGH if motion is detected, otherwise LOW
  int pirState = digitalRead(pirPin);
  // Check if motion is detected
  if (pirState == HIGH) {
    // Motion detected, turn the LED on
    digitalWrite(ledPin, HIGH);
    Serial.println("Motion detected!");
  } else {
    // No motion, turn the LED off
    digitalWrite(ledPin, LOW);
  }
  // A small delay to avoid excessive serial output
  delay(100);
}
```

Output:



Observation:

1. When motion is detected, the PIR sensor output becomes HIGH and the LED turns ON.
2. When no motion is detected, the output becomes LOW and the LED turns OFF.

Result:

DATE :	Interfacing DHT22 Sensor with Arduino
EXP NO : 6	

Aim:

To interface a DHT22 temperature and humidity sensor with Arduino UNO and display the measured temperature and humidity readings on the serial monitor.

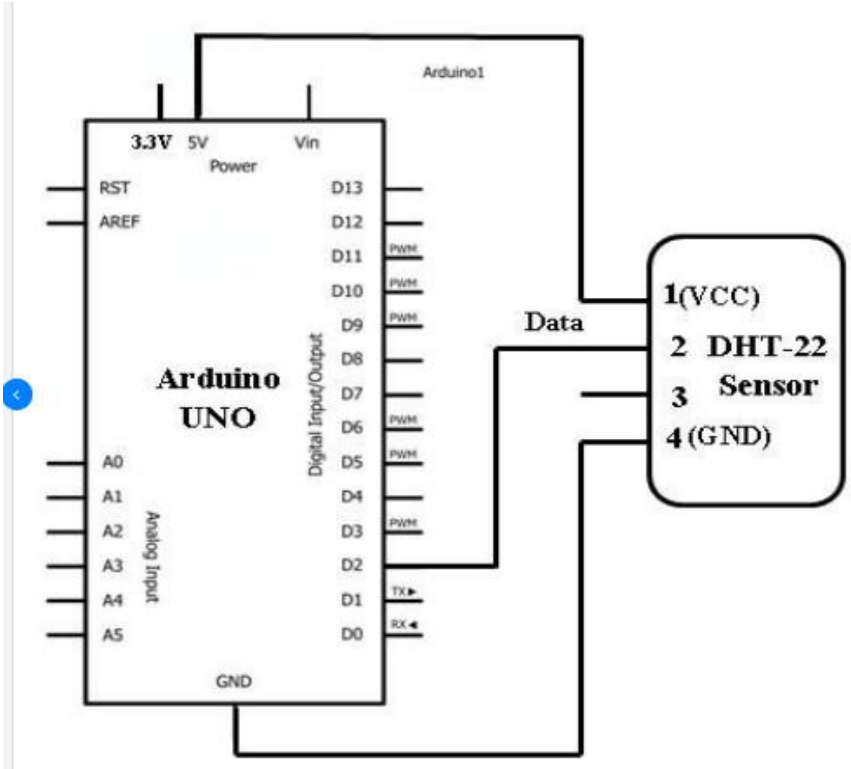
Components Required:

S.No	Component	Specification	Quantity
1	Arduino Uno	ATmega328P-based microcontroller board	1
2	DHT22 Sensor)	AM2302	1
3	Resistor	10kΩ	1
4	Jumper Wires	Male-to-Male	As required
5	Breadboard	-	1

Theory:

The DHT22 (also known as AM2302) is a digital sensor that measures both temperature and relative humidity.It uses a capacitive humidity sensor and a thermistor to measure environmental conditions, then transmits the data through a single-wire digital interface to the Arduino.The Arduino reads this digital signal using the DHT library, which converts raw data into temperature (°C) and humidity (%) values. These readings can be displayed on the Serial Monitor or an LCD display. The Arduino continuously monitors this digital output signal:

Circuit Diagram:



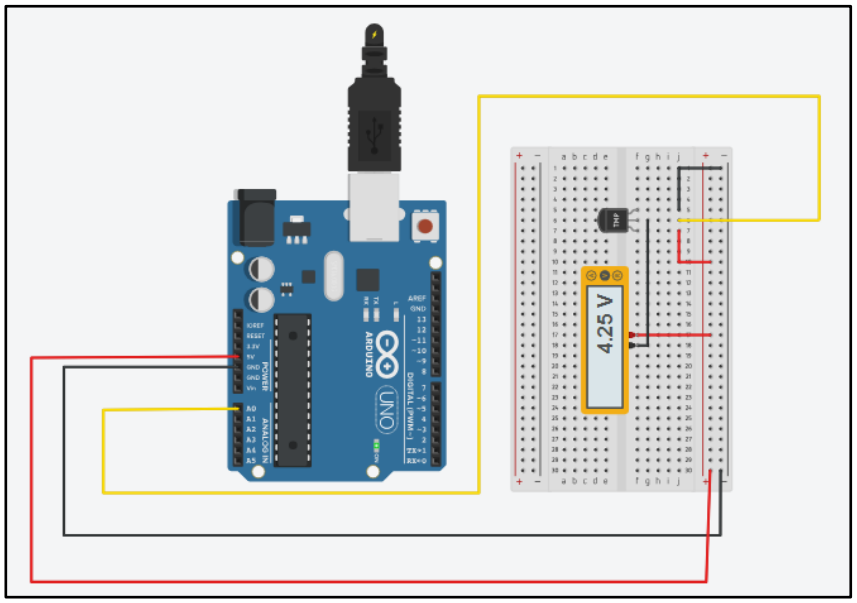
Procedure:

- Start a new project: Open Tinker cad and create a new "Circuit."
  Add the following components: Arduino UNO, DHT22 sensor,10kΩ resistor
- In the Code Editor, switch to Text Mode.
- Install and include the DHT library in the Arduino IDE (if using offline IDE).
- Go to Sketch → Include Library → Manage Libraries → Search "DHT sensor library" → Install.
- Copy and paste the below Arduino code.
- Start simulation and open the Serial Monitor (9600 baud).
- Observe the Temperature and Humidity readings displayed.

Code:

```
// Program: To interface DHT22 sensor with Arduino
// and display Temperature and Humidity readings

#include "DHT.h"       // Include DHT sensor library

#define DHTPIN 2       // DHT22 data pin connected to digital pin 2
#define DHTTYPE DHT22  // Define sensor type as DHT22

DHT dht(DHTPIN, DHTTYPE);  // Initialize DHT sensor

void setup() {
  Serial.begin(9600);       // Start serial communication
  Serial.println("DHT22 Sensor Reading...");
  dht.begin();              // Initialize DHT sensor
}

void loop() {
  float humidity = dht.readHumidity();       // Read humidity
  float temperature = dht.readTemperature(); // Read temperature in Celsius

  // Check if any reading failed
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT22 sensor!");
    return;
  }

  // Display readings
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C  |  Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");

  delay(2000);  // Wait for 2 seconds between readings
}
```

Output :



Result:

DATE :	**Interfacing Motor Using Relay with Raspberry Pi/ Arduino Based on Temperature**
EXP NO : 7	

Aim:

To interface a DC motor with a Raspberry Pi using a relay module and control the motor operation based on temperature sensor readings. When the temperature crosses a certain threshold (e.g., 30°C), the motor automatically turns ON; otherwise, it remains OFF.

Components Required:

S.No	Component	Quantity
1	Raspberry Pi (any model with GPIO, e.g. Pi 3/4)	1
2	5V Relay Module	1
3	Temperature Sensor (LM35 or DHT11/DHT22)	1
4	DC Motor (5V/12V)	1
5	Transistor (if using raw relay coil)	1
6	Diode (1N4007)	1
7	Jumper Wires	As required
8	Breadboard	1
9	Power Supply (5V/12V as per motor)	1

Theory:

Interfacing a motor with a Raspberry Pi or Arduino based on temperature involves a sensor providing data to the microcontroller, which then uses a relay as an electrically-controlled switch to manage power to the motor. The core theory is that since microcontrollers cannot directly handle the high voltage and current required by most motors, a relay acts as an intermediary, isolating the low-voltage control circuit of the microcontroller from the high-power circuit of the motor. In this system, the temperature sensor (such as LM35) measures the environmental or motor temperature, and the microcontroller's program reads this data. If the temperature reading crosses a predefined threshold, the microcontroller sends a low-power signal to the relay module, causing its internal electromagnetic switch to close or open the high-power circuit, thus turning the motor on or off. This allows for automated temperature regulation, such as activating a cooling fan when a motor gets too hot.

Circuit Connections/ Procedure:

🧩 **Connections using DHT11 (Digital Temperature Sensor)**

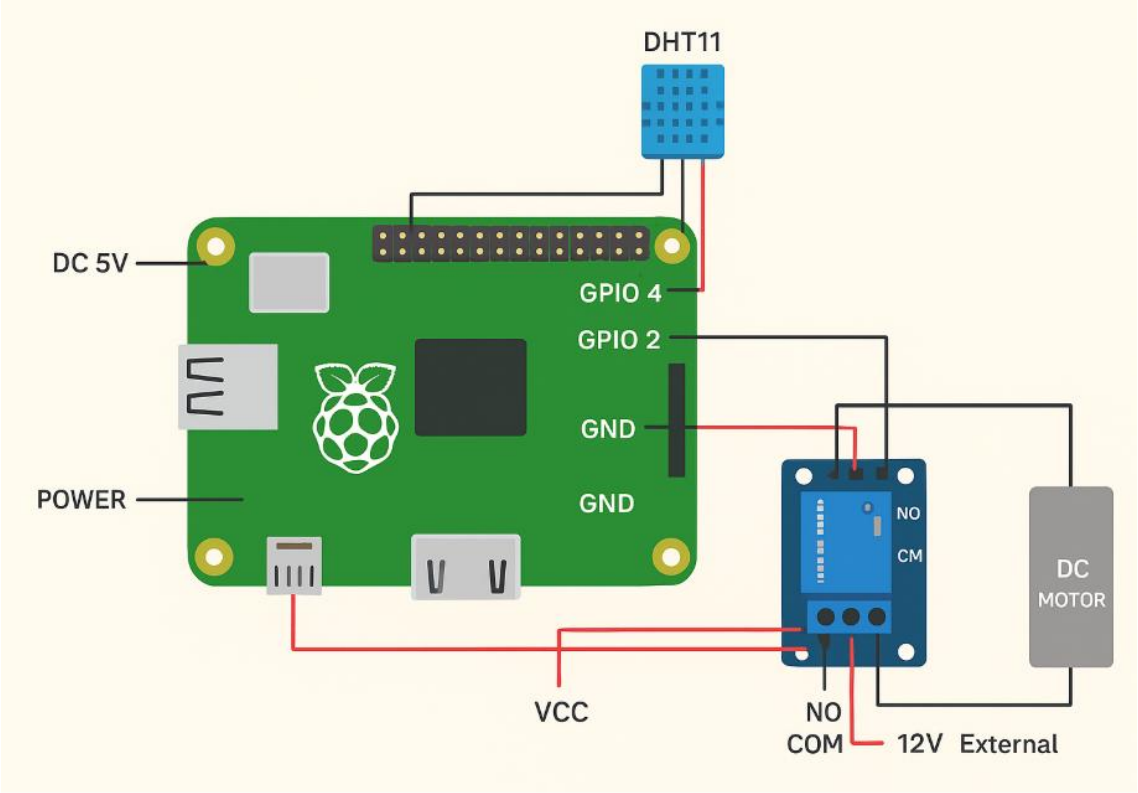
- DHT11 **VCC** → Raspberry Pi **5V**
- DHT11 **GND** → Raspberry Pi **GND**
- DHT11 **OUT** → Raspberry Pi **GPIO4**

🧩 **Relay and Motor Connections**

- Relay **VCC** → Raspberry Pi **5V**
- Relay **GND** → Raspberry Pi **GND**

- Relay **IN** → Raspberry Pi **GPIO17**
- Relay **COM** → Power supply positive (+)
- Relay **NO (Normally Open)** → One terminal of DC motor
- Other terminal of DC motor → Power supply negative (–)

When the relay activates, the circuit closes and powers the motor.



Python Code (Using DHT11 Sensor)

```
import RPi.GPIO as GPIO
import Adafruit_DHT
import time

# Set up sensor
DHT_SENSOR = Adafruit_DHT.DHT11
DHT_PIN = 4  # GPIO pin for DHT11 data

# Relay pin setup
RELAY_PIN = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(RELAY_PIN, GPIO.OUT)
GPIO.output(RELAY_PIN, GPIO.LOW)  # Start with motor OFF

# Temperature threshold (°C)
TEMP_THRESHOLD = 30.0

print("Temperature-based Motor Control System Started")
try:
    while True:
        humidity, temperature = Adafruit_DHT.read_retry(DHT_SENSOR, DHT_PIN)
```

```
        if temperature is not None:
            print(f"Temperature: {temperature:.1f}°C | Humidity: {humidity:.1f}%")

            if temperature > TEMP_THRESHOLD:
                GPIO.output(RELAY_PIN, GPIO.HIGH)
                print("Temperature high! Motor ON.")
            else:
                GPIO.output(RELAY_PIN, GPIO.LOW)
                print("Temperature normal. Motor OFF.")



        else:
            print("Sensor read error!")

        time.sleep(2)

except KeyboardInterrupt:
    print("Program stopped by user.")
finally:
    GPIO.output(RELAY_PIN, GPIO.LOW)
    GPIO.cleanup()
```

Working Principle

1. The **DHT11** sensor continuously measures ambient temperature.
2. When the temperature reading exceeds the **threshold (30°C)**:
    - The **relay module is triggered (GPIO17 → HIGH)**.
    - The relay switches ON the external **motor circuit**.
3. When the temperature falls below the threshold:
    - The **relay is deactivated**, turning the **motor OFF**.

Expected Output:

Temperature	Relay State	Motor Status
< 30°C	LOW	OFF
> 30°C	HIGH	ON

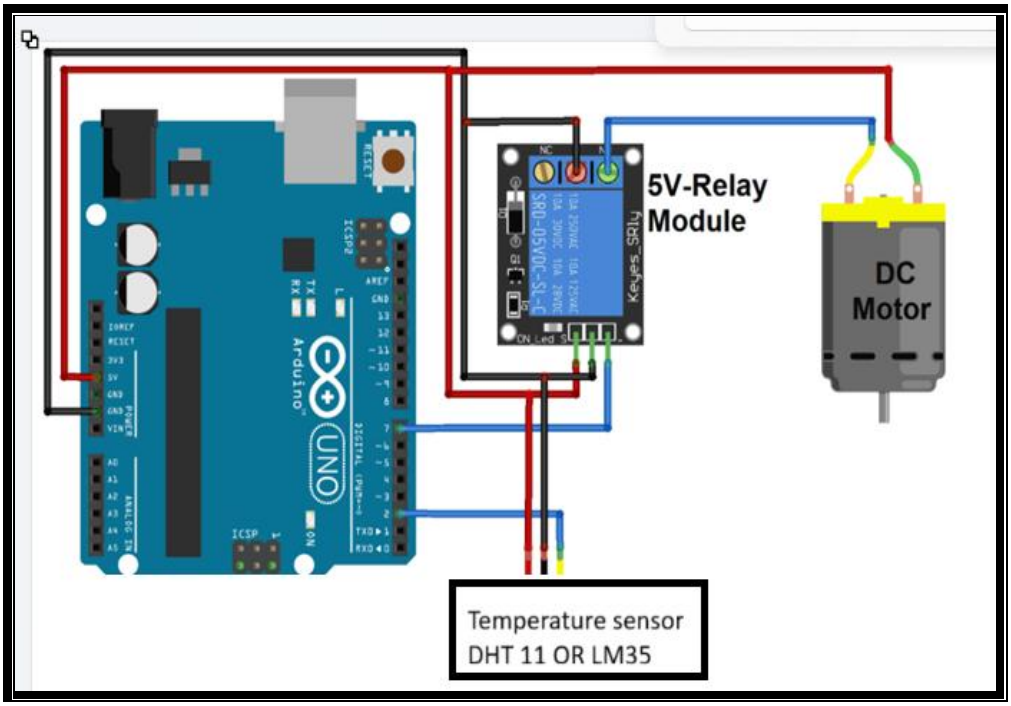
Using Arduino:

```
// Define pins
const int tempSensorPin = A0;  // LM35 connected to analog pin A0
const int relayPin = 7;        // Relay module connected to digital pin 7
const float thresholdTemp = 30.0; // Temperature threshold in Celsius

void setup() {
  pinMode(relayPin, OUTPUT);  // Set relay pin as output
  digitalWrite(relayPin, LOW); // Ensure motor is OFF initially
  Serial.begin(9600);          // Initialize serial communication for debugging
}

void loop() {
  int sensorValue = analogRead(tempSensorPin); // Read temperature sensor
value
  float voltage = sensorValue * (5.0 / 1023.0); // Convert to voltage (0-5V)
  float temperature = voltage * 100.0;          // Convert voltage to
temperature (LM35: 10mV/°C)

  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");

  if (temperature > thresholdTemp) {
    digitalWrite(relayPin, HIGH); // Turn ON motor via relay
    Serial.println("Motor ON");
  } else {
    digitalWrite(relayPin, LOW);  // Turn OFF motor via relay
    Serial.println("Motor OFF");
  }

  delay(1000); // Wait for 1 second before next reading
}
```

- **Temperature Sensor (LM35)**: The LM35 outputs a voltage proportional to the temperature. The formula used is:
- Temperature (°C) = Voltage (V) × 100 Temperature (°C)
- = Voltage (V) × 100
  - **Relay Module**: The relay acts as a switch to control the motor. When the relay pin is set to HIGH, the motor turns ON.
- Threshold Temperature: The motor turns ON when the temperature exceeds 30°C (modifiable via thresholdTemp).
-
  - **Components:**
    - Arduino Uno
    - LM35 Temperature Sensor
    - Relay Module
    - DC Motor
    - Power Supply (for motor)

Connections:

- LM35: Connect VCCto 5V, GNDto GND, and OUTto A0.
- Relay: Connect IN to pin 7, VCC to 5V, and GND to GND.
- Motor: Connect motor terminals to the relay's NO (Normally Open) and COM (Common) pins. Use an external power supply for the motor.



Result:

DATE :	Interfacing LCD with Raspberry Pi to Display Temperature and Humidity
EXP NO : 8	

Aim:

To interface a 16x2 LCD with Raspberry Pi and display real-time temperature and humidity readings from a DHT11/DHT22 sensor.

Components Required:

1. Raspberry Pi (any model with GPIO support)
2. DHT11 or DHT22 Temperature and Humidity Sensor
3. 16x2 LCD Display
4. I2C LCD Module (optional, for easier wiring)
5. Jumper Wires
6. Breadboard
7. Power Supply

Theory:

Interfacing an LCD with a Raspberry Pi to display temperature and humidity involves a three-part process: sensory data acquisition, microcontroller processing, and visual output. The project typically uses a DHT11 or DHT22 sensor, which captures environmental data by using a thermistor for temperature and a capacitive sensor for humidity, before transmitting the information as a digital signal over a single wire. The Raspberry Pi, running a Python script, acts as the central processor, reading this serial data from the sensor via one of its General Purpose Input/Output (GPIO) pins. To display the data, the Raspberry Pi communicates with an LCD screen, often connected via an I^2C module to minimize the number of GPIO pins needed for the connection. The Python program then takes the temperature and humidity values, formats them, and sends the text to the I^2C module, which in turn outputs the information to the character-based LCD screen for display.

Circuit Diagram:

- **Connect the DHT sensor:**
  - VCC to 3.3V
  - GND to GND
  - Data pin to GPIO pin (e.g., GPIO4)

- **Connect the LCD:**
  - If using an I2C module, connect SDA to GPIO2 and SCL to GPIO3.
  - If not using I2C, connect RS, E, D4-D7 pins to GPIO pins as per your setup.

Python Code:

```
import Adafruit_DHT
import I2C_LCD_driver
import time
# Initialize DHT sensor
DHT_SENSOR = Adafruit_DHT.DHT11  # Use DHT22 for DHT22 sensor
DHT_PIN = 4  # GPIO pin connected to DHT data pin
# Initialize LCD
lcd = I2C_LCD_driver.lcd()
try:
    while True:
        # Read temperature and humidity
        humidity, temperature = Adafruit_DHT.read_retry(DHT_SENSOR, DHT_PIN)
        if humidity is not None and temperature is not None:
            # Format and display on LCD
            lcd.lcd_display_string(f"Temp: {temperature:.1f}C", 1)
            lcd.lcd_display_string(f"Humidity: {humidity:.1f}%", 2)
        else:
            lcd.lcd_display_string("Sensor Error", 1)
        time.sleep(2)  # Update every 2 seconds

except KeyboardInterrupt:
    lcd.lcd_clear()
    print("Program stopped.")
```

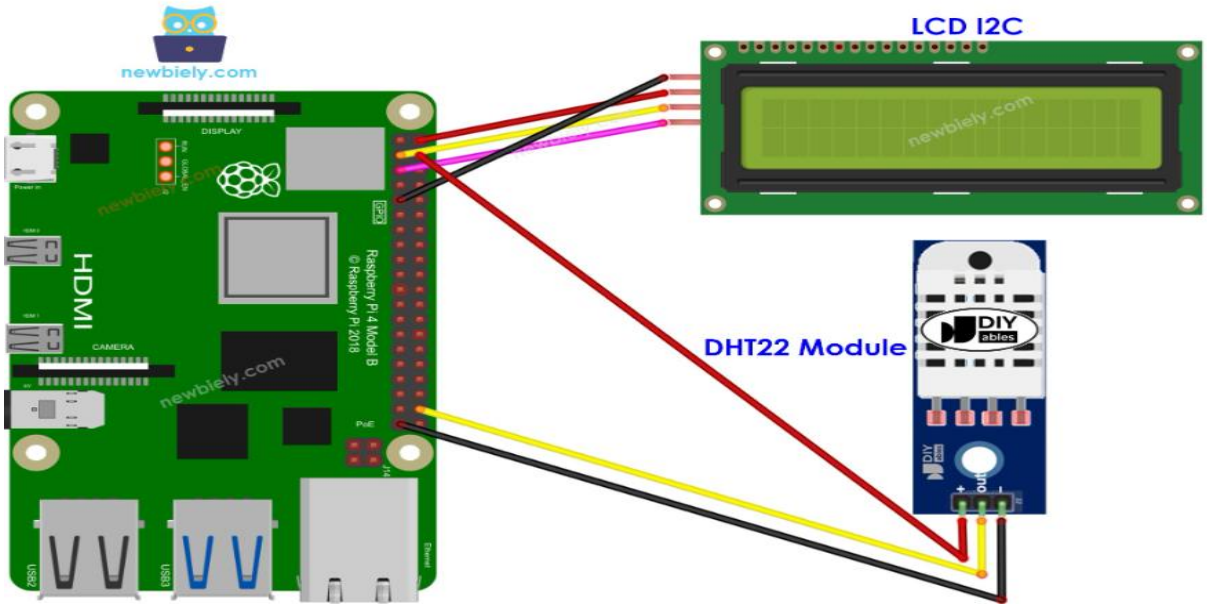
Procedure:

1. Setup Hardware: Assemble the circuit as per the diagram.
2. Install Libraries:
    - Install Adafruit_DHT library: pip install Adafruit_DHT
    - Install I2C_LCD_driver library (if using I2C): Follow the library's documentation.
3. Enable I2C on Raspberry Pi:
    - Run sudo raspi-config, navigate to "Interface Options," and enable I2C.
4. Run the Code:
    - Save the code as lcd_temp_humidity.py.
    - Execute using python3 lcd_temp_humidity.py.
5. Observe Output: The LCD will display temperature and humidity readings.

Expected Output:

The 16x2 LCD will display:

- Line 1: Temperature in Celsius (e.g., Temp: 25.3C)
- Line 2: Humidity in percentage (e.g., Humidity: 60.5%)



Result:

DATE :	**Write a program to interface a flame/smoke sensor with Arduino and give an alert Message when flame/smoke is detected.**
EXP NO : 9	

Aim:

To detect the temperature using smoke/flame sensor and activate the buzzer interfaced with Arduino.

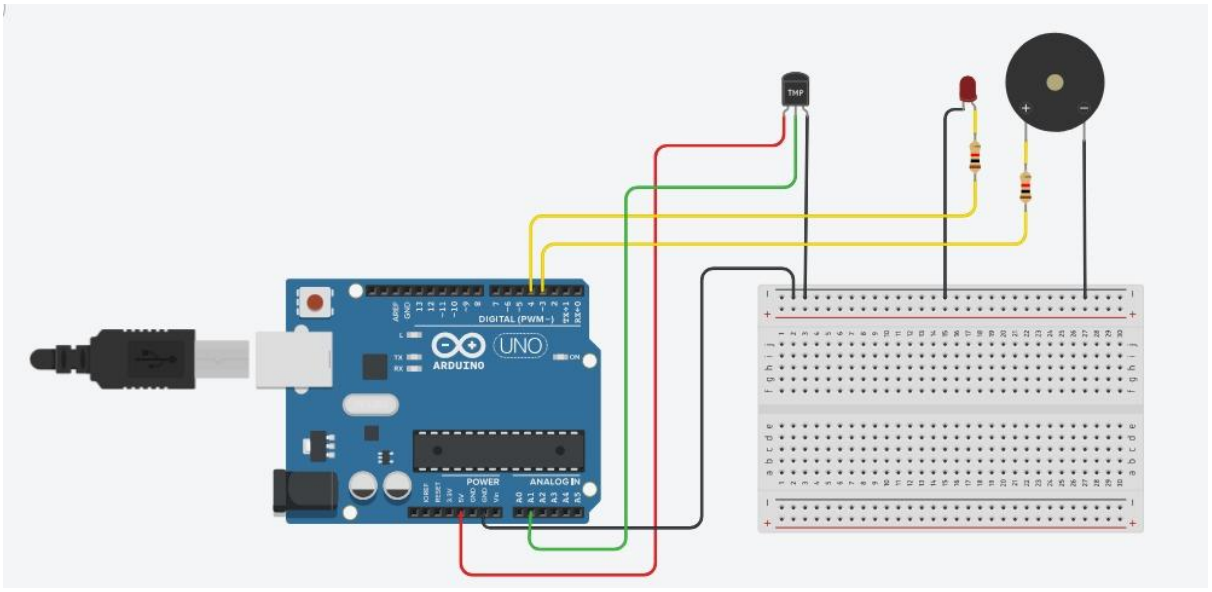
Components Required:

S.No	Component	Specification	Quantity
1	Arduino Uno	ATmega328P-based microcontroller board	1
2	Sensor	Flame Sensor	1
3	LED		1
4	Transistor	BC547	1
5	Jumper Wires	Male-to-Male	As required
6	Breadboard	–	1
7	USB Cable	Type A to B	1
8	Resistor	1KΩ	2
9	Buzzer	5v	1

Theory:

A flame/smoke sensor detects infrared light emitted by fire or changes in air composition due to smoke. It converts the sensed signal into an analog or digital output for the Arduino. When the sensor output crosses a threshold, it indicates flame or smoke presence. The Arduino processes this input and triggers an alert (buzzer or serial message). Such systems are useful for early fire detection and safety monitoring.

Circuit Diagram:



Procedure:

1.Connect the circuit as per the diagram using the breadboard and jumper wires.

2.Open Arduino IDE on your computer.

3.Select the board and port:

Board → *Arduino Uno*

Port → *COMx* (your connected port)

4.Write or paste the Arduino code given below.

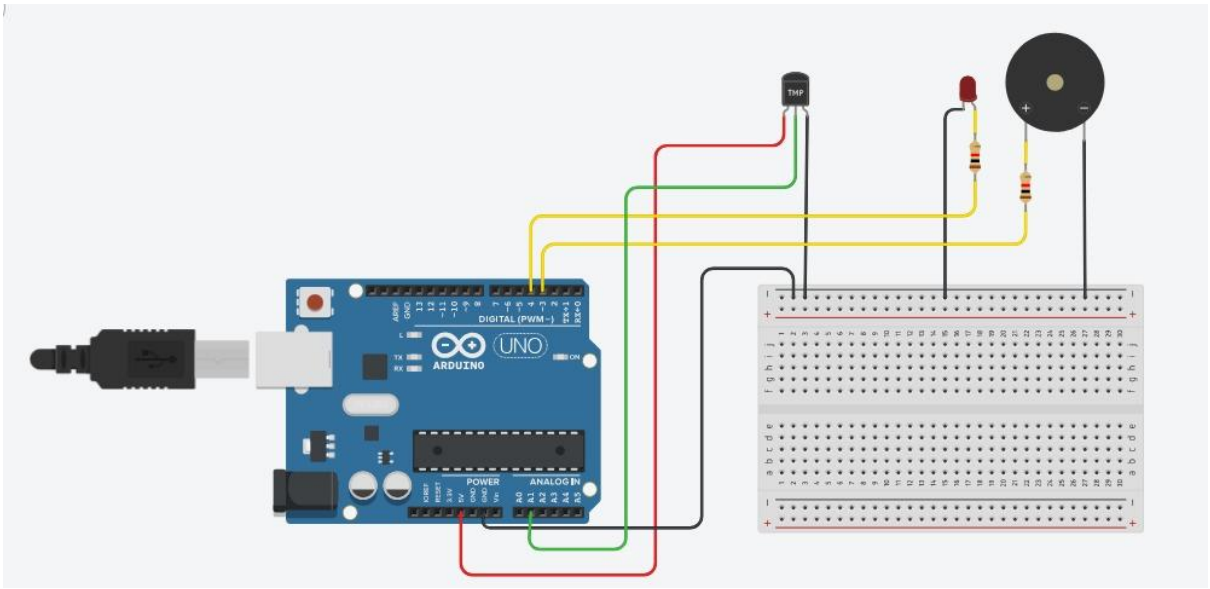
5.Verify and upload the code to the Arduino board.

6. Increase the temperature gradually and observe the flame detection using Buzzer.

Code:

```
float temp;
void setup()
{
Serial.begin(9600);
  pinMode(A1,INPUT);
  pinMode(4,OUTPUT);
  pinMode(3,OUTPUT);
}
void loop()
{
  temp=analogRead(A1);
  temp=((temp*5)/1024);
  temp=(temp-0.5)*100;
  Serial.print(temp);
  if(temp<=80)
  {
    Serial.println("Flame not detected");
    digitalWrite(4,LOW);
    digitalWrite(3,LOW);
  }
  else
  {
   Serial.println("Flame detected");
    digitalWrite(4,HIGH);
    digitalWrite(3,HIGH);
  }
  }
```

Output / Observation:



Result:

DATE :	Case study: Digital Range Finder using Arduino
EXP NO : 10	

Aim:

To find the intruder using digital range finder interfaced with Arduino.

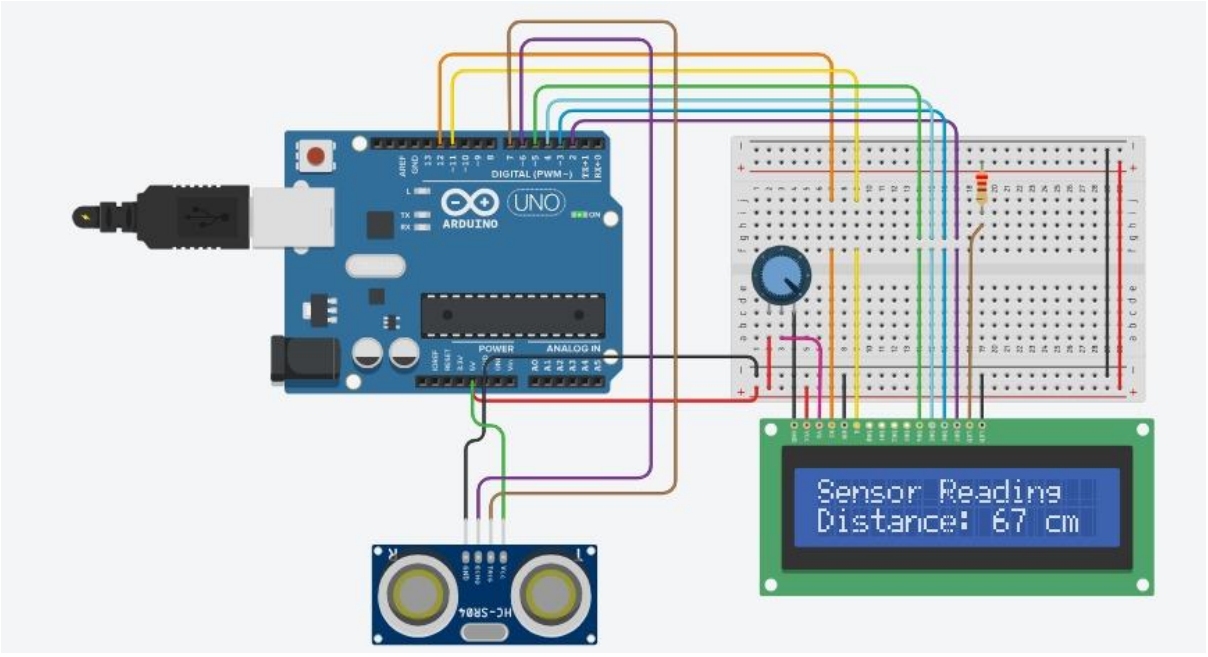
Components Required:

S.No	Component	Specification	Quantity
1	Arduino Uno	ATmega328P-based microcontroller board	1
2	Sensor	Ultrasonic Sensor	1
3	LCD		1
4	Jumper Wires	Male-to-Male	As required
5	Breadboard	–	1
6	USB Cable	Type A to B	1

Theory:

A digital range finder measures the distance between the sensor and an object using ultrasonic waves. The sensor emits a sound pulse and measures the time taken for the echo to return. The Arduino calculates distance using the formula: Distance = (Time × Speed of Sound) / 2. The measured distance is displayed on a serial monitor or digital display. Such systems are widely used in robotics, obstacle detection, and automation applications.

Circuit Diagram:



Procedure:

1.Connect the circuit as per the diagram using the breadboard and jumper wires.

2.Open Arduino IDE on your computer.

3.Select the board and port:

Board → *Arduino Uno*

Port → *COMx* (your connected port)

4.Write or paste the Arduino code given below.

5.Verify and upload the code to the Arduino board.

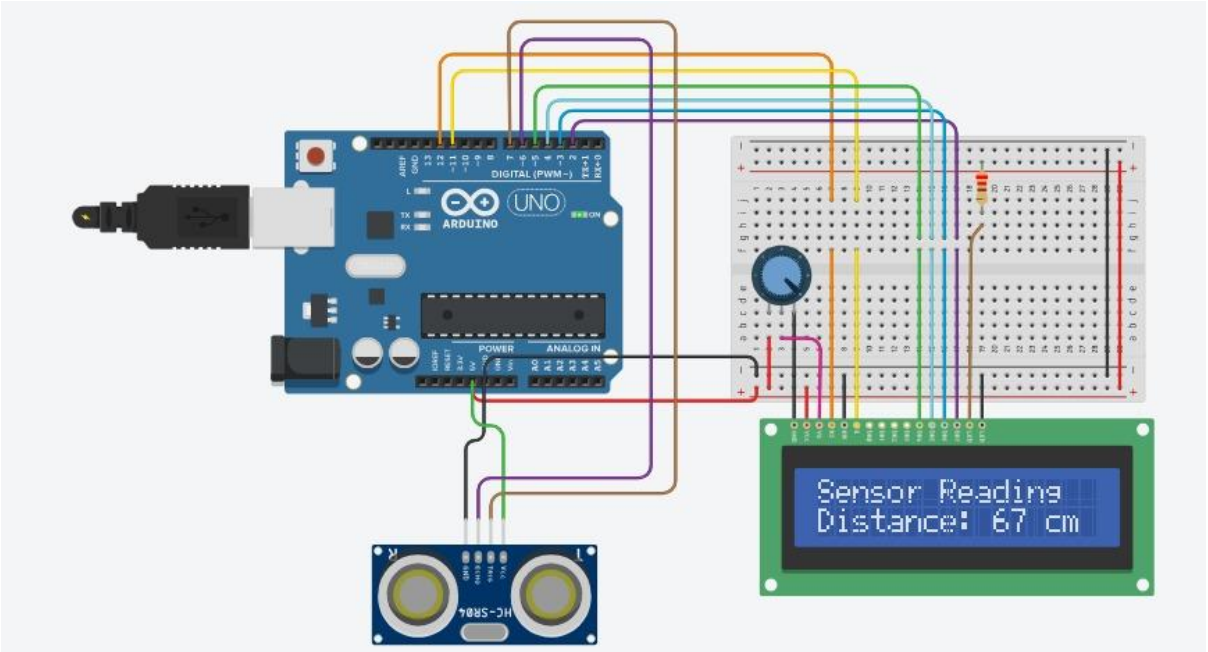
6. Need to check the intruder crosses the range digitally and alert the using LCD**.**

Code:

```
#include <LiquidCrystal.h>

// Variable for elapsed seconds (not used, but kept for future use)
int seconds = 0;

// Initialize LCD: (RS, E, D4, D5, D6, D7)
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// Define pins for Ultrasonic Sensor
int triggerPin = 7;
int echoPin = 6;

// Variables for time and distance
long duration;
int distance;

void setup()
{
  // Initialize LCD with 16 columns and 2 rows
  lcd.begin(16, 2);
  lcd.print("Sensor Reading");

  // Configure pin modes
  pinMode(triggerPin, OUTPUT);
  pinMode(echoPin, INPUT);
}

void loop()
{
  // Send trigger pulse
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
```

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

  // Read echo pulse duration (in microseconds)
  duration = pulseIn(echoPin, HIGH);

  // Calculate distance in cm (speed of sound = 0.0342 cm/µs)
  distance = duration * 0.0342 / 2;

  // Display distance on LCD
  lcd.setCursor(0, 1);
  lcd.print("Distance: ");
  lcd.print(distance);
  lcd.print(" cm   ");

  delay(200); // Small delay before next reading
}
```

Output / Observation:



Result: