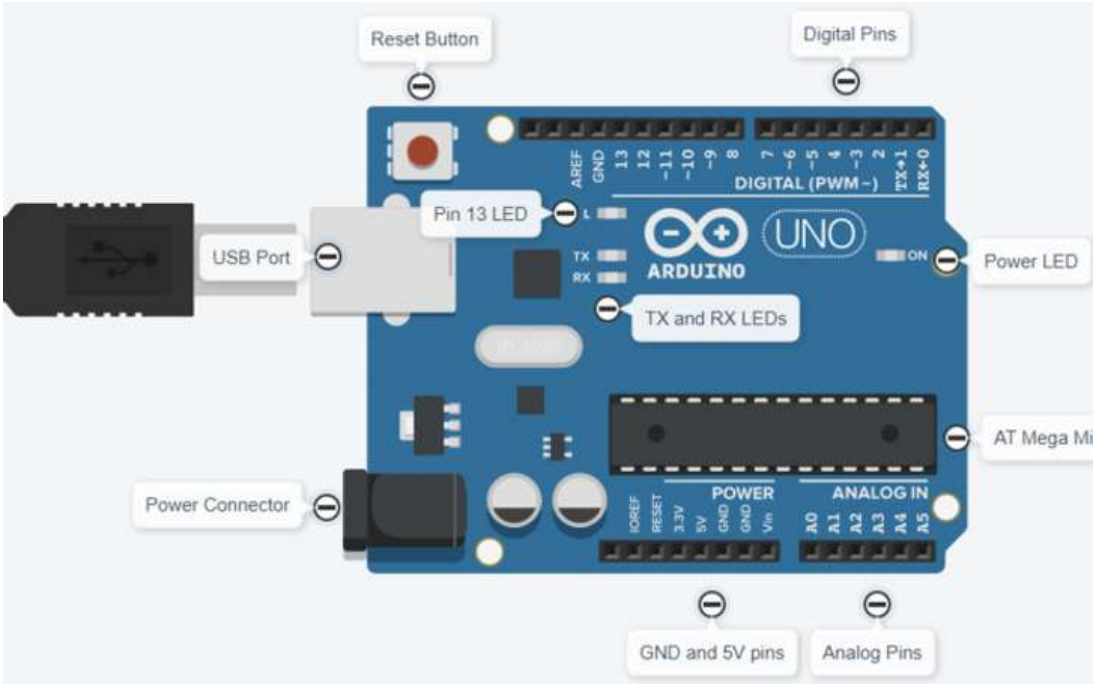
EXPERIMENT 1

AIM – To study about the ARDUINO and RASPBERRY-PI board architecture

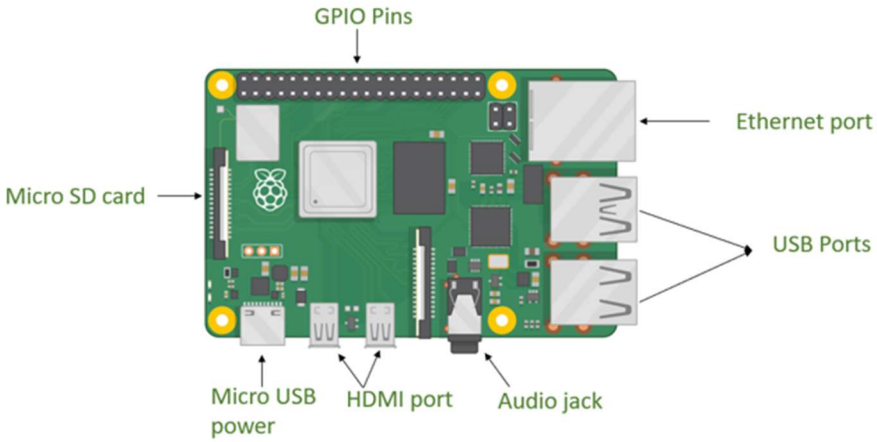
THEORY-



ARDUINO

The latest Microcontroller used in the Arduino board is **ATMEGA 328P.**

- **Analog Pins –** There are 6 pins connecting analogue input from A0 to A5. It receives input in the form of continuous waves sent by any external source.
- **Digital Pins –** Signifies serial input/output connections.
    - **D0, D1 -** It connects Arduino with other serial devices like computers or sensors.
    - **D2 to D13 –** These are used for data transfer digital Input/Output.
- **USB Jack –** It is used to connect Arduino board to PC.
- **Microcontroller –** It is designed for specific tasks. It is the heart and brain of the Arduino board. To program the microcontroller, we have to provide coding from your computers.
- **Test LEDs –** Connected at D13 pin to check the Arduino board.
- **Receivers and Transmitters –** Rx blinks when the Arduino board receives information whereas the Tx blinks when the Arduino board sends information.
- **Power Jack –** Used for power supply.
- **ICSP (In-Circuit Serial Programming) –** Used to modify the Arduino firmware.



RASPBERRY PI BOARD ARCHITECTURE

- **HDMI Port**: Transmits high-definition video and audio signals to external displays, such as monitors or TVs. It supports both HDMI 1.3 and 1.4 standards for various resolutions and refresh rates.
- **USB Ports**: Allow connection to peripherals like keyboards, mice, or external storage devices. These can be USB 2.0 or 3.0, depending on the model, for data transfer.
- **RJ45 Ethernet Port**: Provides wired internet connectivity. It connects to a local network, enabling fast and reliable internet access via a network cable.
- **GPIO Pins**: General-purpose input/output pins used for controlling external devices (e.g., sensors, LEDs). They can be configured as input or output for various electronic projects.
- **Audio Jack**: A 3.5mm stereo output that allows you to connect headphones or speakers for audio playback from the Raspberry Pi.
- **CSI Camera Connector**: A ribbon cable interface used to connect a camera module for capturing images and videos, commonly used for projects requiring vision.
- **DSI Display Connector**: Used for connecting official Raspberry Pi displays. It supports high-speed data transfer for video output to LCD screens.
- **Power Input (Micro-USB / USB-C)**: Supplies power to the Raspberry Pi. Depending on the model, it can be a Micro-USB (older models) or USB-C (newer models) port.
- **SD Card Slot**: Holds the microSD card, which acts as the primary storage for the operating system, software, and files. It is essential for booting the Raspberry Pi.
- **Broadcom SoC (System on Chip)**: The main processing unit of the Raspberry Pi, integrating the CPU, GPU, RAM, and various peripherals into a single chip for efficient computing.
- **LED Indicators**: Small lights that provide status information, such as power, network activity, and SD card status, to help troubleshoot and monitor the device's operation.
- **Wi-Fi / Bluetooth**: Integrated wireless modules (in newer models) for connecting to local Wi-Fi networks or Bluetooth devices, enabling wireless communication without needing additional hardware.
- **Camera Interface (CSI)**: Allows for the connection of camera modules to capture images and video, supporting projects in surveillance, robotics, and computer vision.

EXPERIMENT 2

AIM – To interface LED/Buzzer with Arduino/Raspberry Pi and write a program to turn ON LED for 1 sec after every 2 seconds.

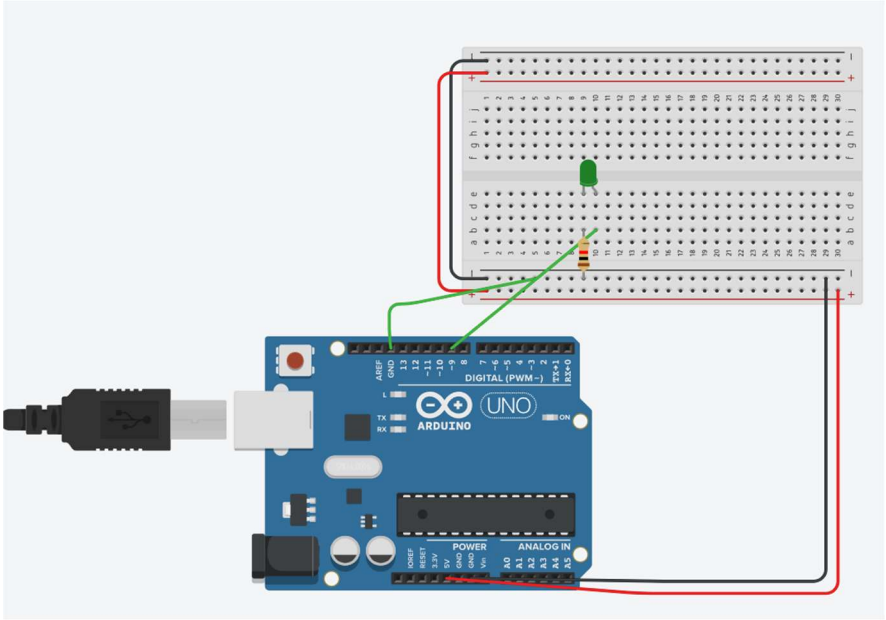
THEORY – The Arduino platform is widely used in electronics and programming due to its simplicity and versatility. This experiment aims to interface an LED (Light Emitting Diode) with an Arduino microcontroller. The circuit setup involves connecting the LED to designated digital output pins on the Arduino board. An LED emits light when current flows through it in the forward direction when activated by an electrical signal. This experiment is also a precursor to understanding event-driven programming and input/output (I/O) management in microcontroller applications.

Applications of LED/Buzzer Control Programs

Programs that control LEDs and buzzers with Arduino have a wide range of practical applications across various fields:

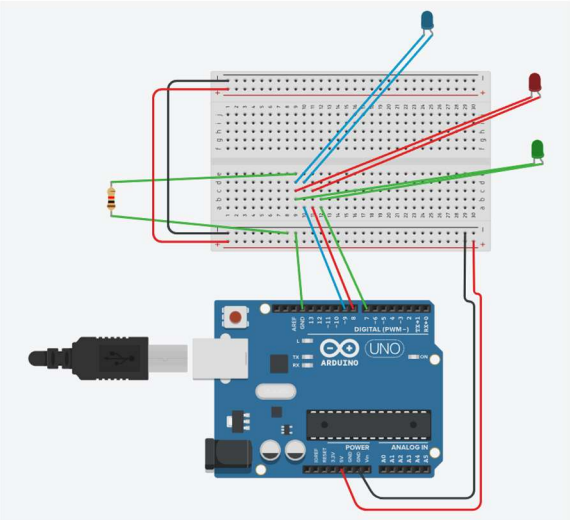
1. Indicators and Alerts: LED indicators can signal the status of devices, while buzzers provide audible alerts for alarms, notifications, or warnings in system monitoring.
2. Teaching Tools: These basic interfacing projects serve as foundational learning experiences in electronics and programming, making them ideal for classrooms, workshops, and beginner tutorials.
3. Robotics: In robotic systems, LEDs and buzzers can provide visual and auditory feedback, enhancing user interaction and communication between the robot and its environment.
4. Home Automation: In smart home setups, LED status lights can indicate device states (e.g., on/off) while buzzers can signal alarms or reminders for various tasks.
5. Game Development: LED and buzzer combinations can be used in simple games where visual and auditory cues enhance the gameplay experience.

TO SIMULATE AN LED TO BLINK ON AND OFF AFTER EVERY 2 SECONDS



```
// C++ code
//
void setup()
{
  pinMode(9, OUTPUT);
}

void loop()
{
  digitalWrite(9, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(9, LOW);
  delay(2000); // Wait for 2000 millisecond(s)
  digitalWrite(9, HIGH);
}
```

TO SIMULATE A TRAFFIC SIGNAL THAT TURNS ON AND OFF AFTER EVERY 2 SECONDS



```
// C++ code
//
void setup()
{
  pinMode(9, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(7, OUTPUT);
}

void loop()
{
  digitalWrite(9, HIGH);
  delay(500); // Wait for 500 millisecond(s)
  digitalWrite(9, LOW);
  delay(500); // Wait for 500 millisecond(s)
  digitalWrite(8, HIGH);
  delay(500); // Wait for 500 millisecond(s)
  digitalWrite(8, LOW);
  delay(500); // Wait for 500 millisecond(s)
  digitalWrite(7, HIGH);
  delay(500); // Wait for 500 millisecond(s)
  digitalWrite(7, LOW);
  delay(500); // Wait for 500 millisecond(s)
  digitalWrite(8, HIGH);
  delay(500); // Wait for 500 millisecond(s)
  digitalWrite(8, LOW);
  delay(500); // Wait for 500 millisecond(s)
  digitalWrite(9, HIGH);
  delay(500); // Wait for 500 millisecond(s)
  digitalWrite(9, LOW);
}
```

Functions Used in Code –

1. pinMode(pin, mode): Configures the specified pin to behave either as an input or an output. In this experiment, it sets the LED and buzzer pins as outputs.
2. digitalWrite(pin, value): Writes a HIGH or LOW value to a pin. Used to turn the LED and buzzer on (HIGH) or off (LOW).
3. delay(ms): Pauses the program for the duration specified in milliseconds. This function controls how long the LED stays on and when the loop iterates.
4. loop(): The main function that runs continuously after the setup() function. It is used for repeated actions like turning the LED on and off at specified intervals.

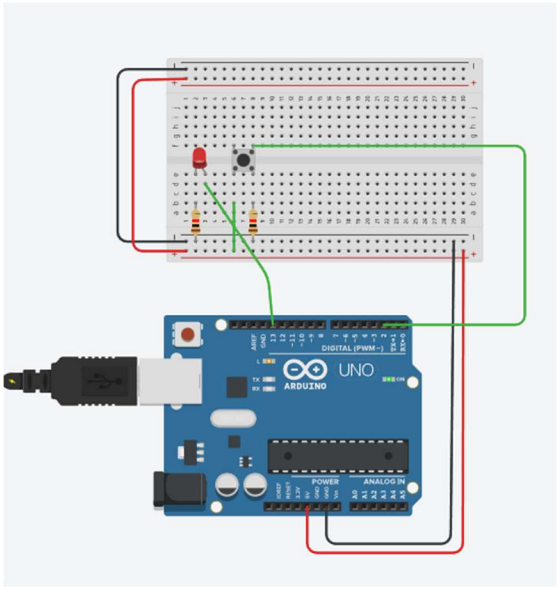
EXPERIMENT 3

AIM – To interface Push button/Digital sensor (IR/LDR) with Arduino/Raspberry Pi and write a program to turn ON LED when push button is pressed or at sensor detection.

THEORY – The Arduino and Raspberry Pi platforms are widely utilized in electronics and programming due to their accessibility and support for a broad range of sensors and modules. This experiment focuses on interfacing a push button or a digital sensor (such as IR or LDR) with an Arduino or Raspberry Pi to control an LED.

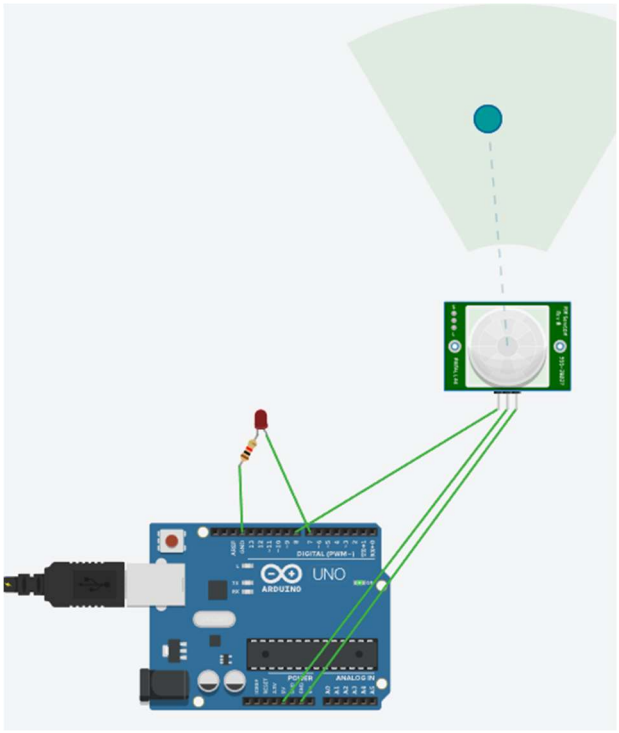
The circuit setup involves connecting the push button or digital sensor to a designated digital input pin and the LED to a digital output pin on the microcontroller or microprocessor. A push button or sensor detects input events—such as a physical press or changes in light intensity—and sends a corresponding signal to the board. The microcontroller processes this signal and activates the LED by enabling current flow in the forward direction.

TO SIMULATE A LED WHEN PUSH BUTTON IS PRESSED




```
// C++ code
//
int buttonState = 0;

void setup()
{
  pinMode(2, INPUT);
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  buttonState = digitalRead(2);
  if (buttonState == HIGH) {
    digitalWrite(LED_BUILTIN, HIGH);
  } else {
    digitalWrite(LED_BUILTIN, LOW);
  }
  delay(10); // Delay a little bit to improve
}
```

TO SIMULATE A LED AT SENSOR DETECTION




```
// C++ code
//
int PIRSensor = 0;

void setup()
{
  pinMode(8, INPUT);
  Serial.begin(9600);
  pinMode(7, OUTPUT);
}

void loop()
{
  PIRSensor = digitalRead(8);
  Serial.println(PIRSensor);
  if (PIRSensor == HIGH) {
    digitalWrite(7, HIGH);
  } else {
    digitalWrite(7, LOW);
  }
  delay(10); // Delay a little bit to improve
}
```

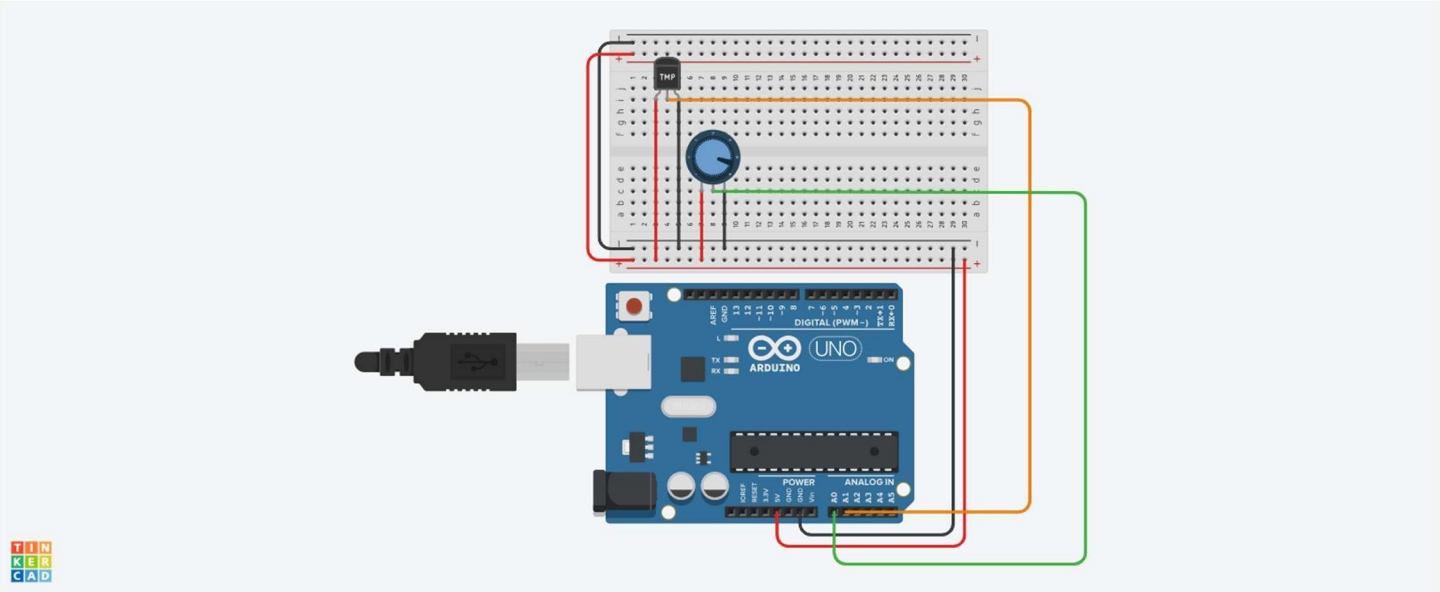
OUTPUT: The LED is turned on successfully, using both the Push Button and the Digital Sensor (IR/LDR).

EXPERIMENT-4

AIM-To interface DHT11 sensor with Arduino/Raspberry Pi and write a program to print temperature and humidity readings.

THEORY:-The **DHT11** is a basic and low-cost digital temperature and humidity sensor commonly used in microcontroller-based projects. It can measure both temperature and humidity, providing digital output that is easy to interface with microcontrollers like Arduino and Raspberry Pi.

The circuit is an Arduino-based temperature and humidity monitoring system using an LM35 temperature sensor to measure temperature and an optional humidity sensor on A1. The Arduino reads sensor data, converts it into °C and °F, and displays it on a 16x2 LCD and the Serial Monitor. A potentiometer adjusts the LCD contrast, and readings update every 5 seconds. Possible issues include an undefined analogIn variable (should be A0), random values from A1 if no humidity sensor is connected, and potential LCD contrast problems.

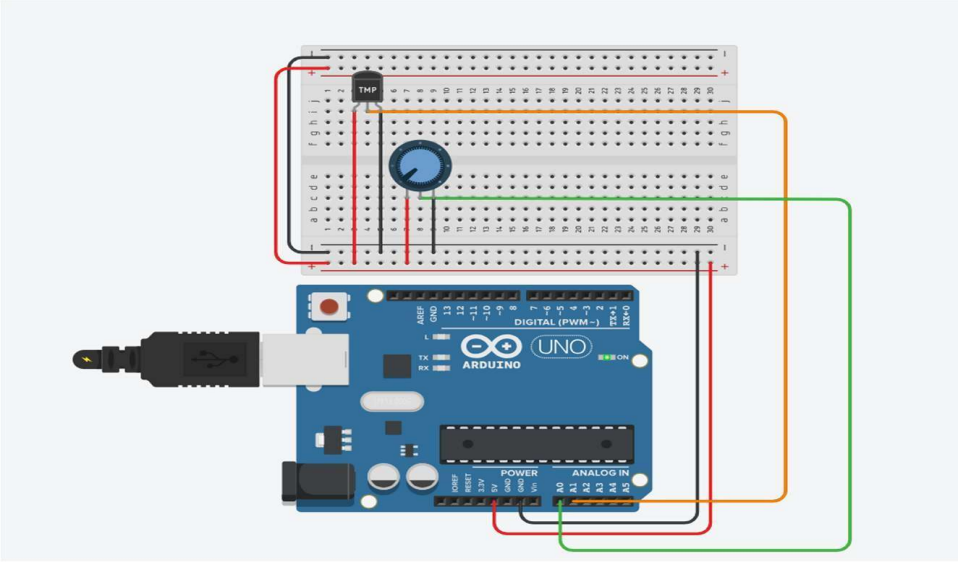


To stimulate DHT11 sensor with Arduino/Raspberry Pi

Program to print temperature and humidity readings:

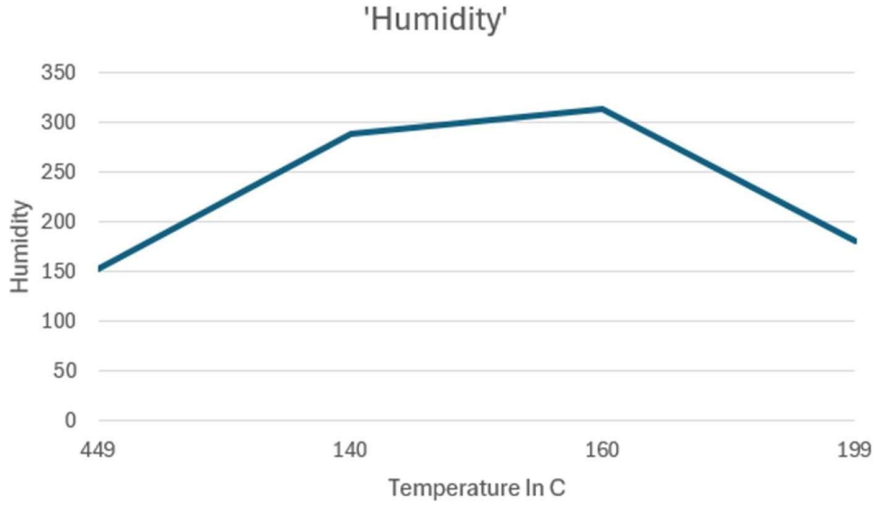
```
int analogIn = A0;

int humiditySensorOutput = 0;

int RawValue = 0;

float Voltage = 0;

float tempC = 0;

float tempF = 0;


void setup() {

  Serial.begin(9600);
```

```
  pinMode(A1, INPUT);}

void loop() {
  RawValue = analogRead(analogIn);
  Voltage = (RawValue / 1023.0) * 5000;
  tempC = (Voltage - 500) * 0.1;
  tempF = (tempC * 1.8) + 32;

  Serial.print("Raw Value = ");
  Serial.print(RawValue);
  Serial.print("\t MilliVolts = ");
  Serial.print(Voltage, 0);
  Serial.print("\t Temperature in C = ");
  Serial.print(tempC, 1);
  Serial.print("\t Temperature in F = ");
  Serial.println(tempF, 1);

  humiditySensorOutput = analogRead(A1);
  Serial.print("Humidity = ");
  Serial.println(humiditySensorOutput);

  delay(5000);}
```



RESULTS:-

```
ts=4990  Temperature in C=449.0  Temperature in F=840.2
humidity=153
Rawvalue=1021    milliVolts=4990         Temperature in C=449.0  Temperature in F=840.2
humidity=153
Rawvalue=1021    milliVolts=4990         Temperature in C=449.0  Temperature in F=840.2
humidity=153
Rawvalue=1021    milliVolts=4990         Temperature in C=449.0  Temperature in F=840.2
humidity=153
Rawvalue=389     milliVolts=1901         Temperature in C=140.1  Temperature in F=284.2
humidity=153
Rawvalue=430     milliVolts=2102         Temperature in C=160.2  Temperature in F=320.3
humidity=153
Rawvalue=511     milliVolts=2498         Temperature in C=199.8  Temperature in F=391.6
humidity=288
Rawvalue=511     milliVolts=2498         Temperature in C=199.8  Temperature in F=391.6
humidity=313
Rawvalue=511     milliVolts=2498         Temperature in C=199.8  Temperature in F=391.6
humidity=180
Rawvalue=511     milliVolts=2498         Temperature in C=199.8  Temperature in F=391.6
humidity=180
Rawvalue=104     milliVolts=508  Temperature in C=0.8    Temperature in F=33.5
humidity=180
```



EXPERIMENT 5

AIM: To interface motor using relay with Arduino/Raspberry Pi and write a program to turn ON motor when push button is pressed.

THEORY:

A relay is an electrically operated switch that allows a low-voltage signal (from Arduino/Raspberry Pi) to control a high-voltage or high-current load (like a motor). It consists of an electromagnet (coil), a common (COM) terminal, a normally open (NO) terminal, and a normally closed (NC) terminal.

- Normally Open (NO): The circuit is open (motor OFF) by default; it closes (motor ON) when the relay is activated.
- Normally Closed (NC): The circuit is closed (motor ON) by default; it opens (motor OFF) when the relay is activated.

Working Principle:

I. When the push button is pressed, the Arduino/Raspberry Pi sends a HIGH signal to the relay module.
II. This energizes the relay coil, closing the NO contact and allowing current to flow to the motor, turning it ON.
III. When the button is released, the signal goes LOW, de-energizing the relay and turning the motor OFF.

Applications:

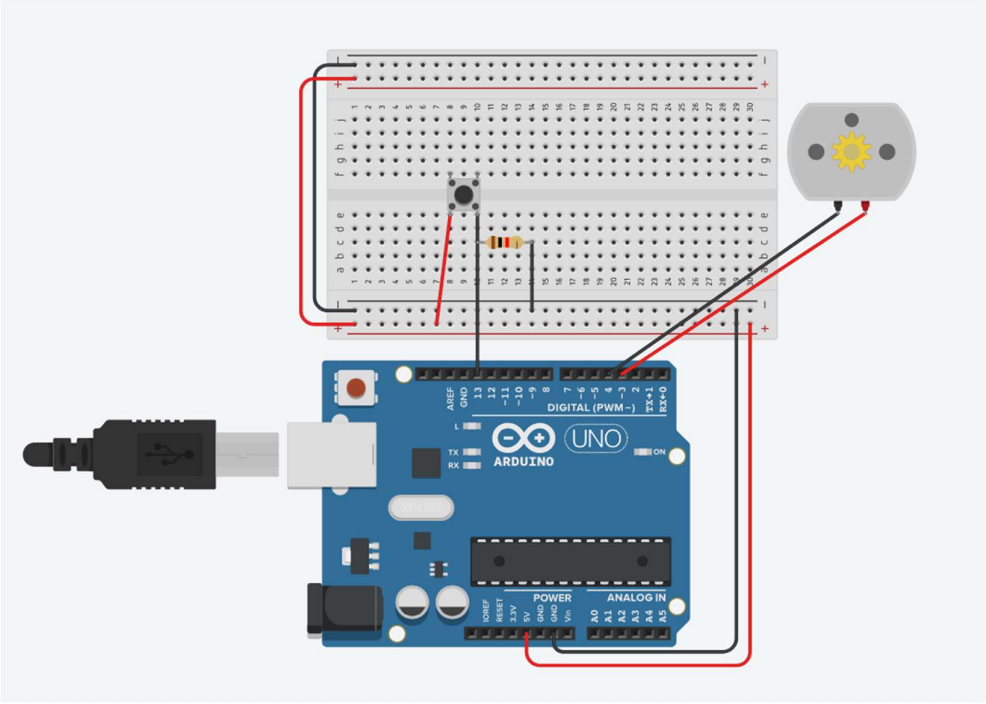
1. Home Automation Systems
2. Smart Irrigation
3. Remote Controlled Vehicles and Drones
4. Elevators and Escalators
5. Renewable Energy Systems

To Make the motor works without relay:

Code:

```
int pb = 13;

int m1 = 4;

 int m2 = 3;

void setup()

{

  pinMode(pb,INPUT);
```

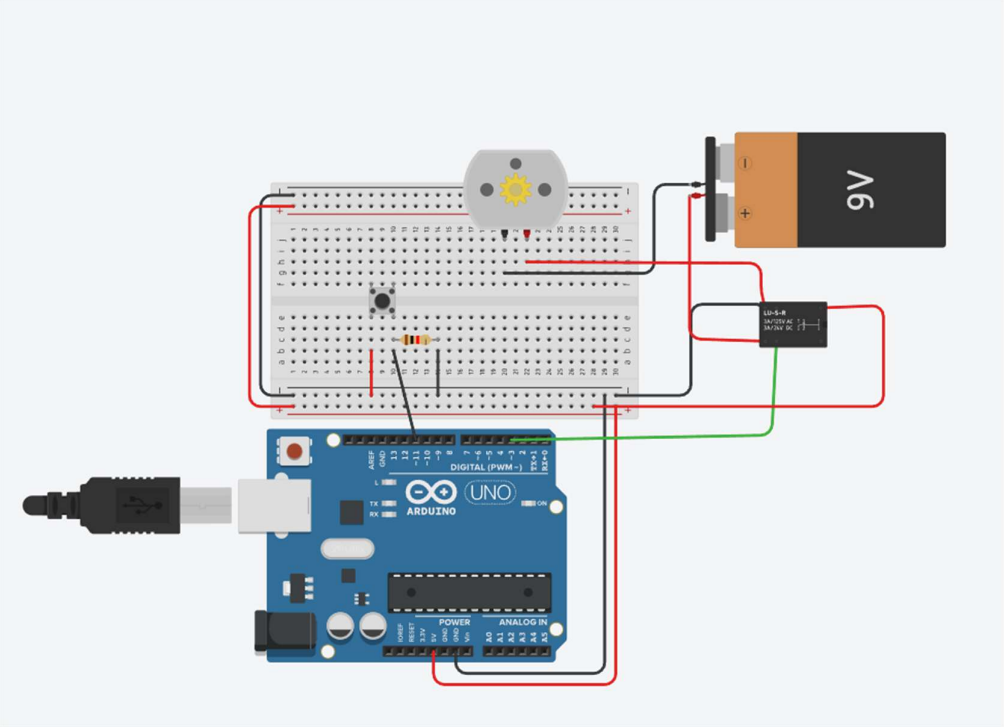
```
pinMode(m1,OUTPUT);

 pinMode(m2,OUTPUT);

    }

    void loop() {

      pb = digitalRead(pb);

    if (pb == HIGH) {

    digitalWrite(m1,HIGH);

    digitalWrite(m2,LOW);

     } else {

       digitalWrite(m1,LOW);

    digitalWrite(m2,HIGH);

     }

     delay(5000);

    }
```



To Make the motor work by using relay:

```
int pb = 11; int relay = 3;
```

```
void setup() {
 pinMode(pb,INPUT);
 pinMode(relay,OUTPUT);
}
void loop() {
  pb = digitalRead(pb);   if
(pb == HIGH) {
digitalWrite(relay,HIGH);
  }
  else {
    digitalWrite(relay,LOW);
  }
  delay(2000);
}
```



Result: We have successfully turned On the Motor when the push button is pressed and simulated it with and without relay.