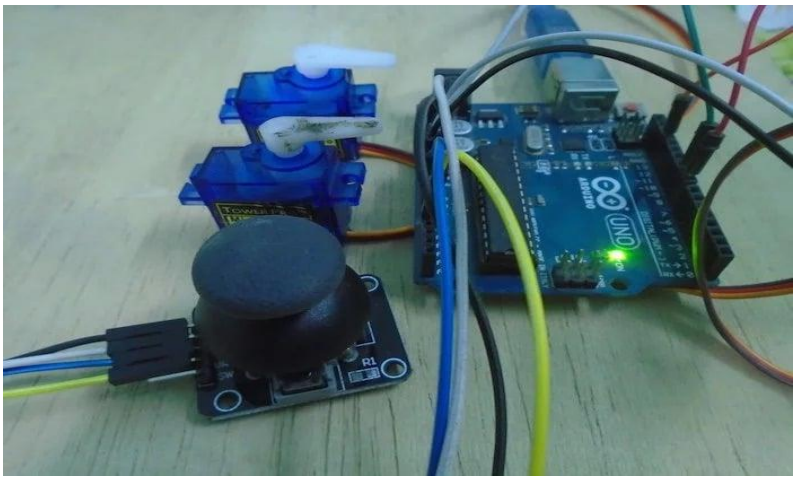
INDEX

LIST OF EXPERIMENTS

Experiment-1

Aim: Controlling actuators through Serial Monitor. Creating different led patterns and controlling them using push button switches. Controlling servo motor with the help of joystick.



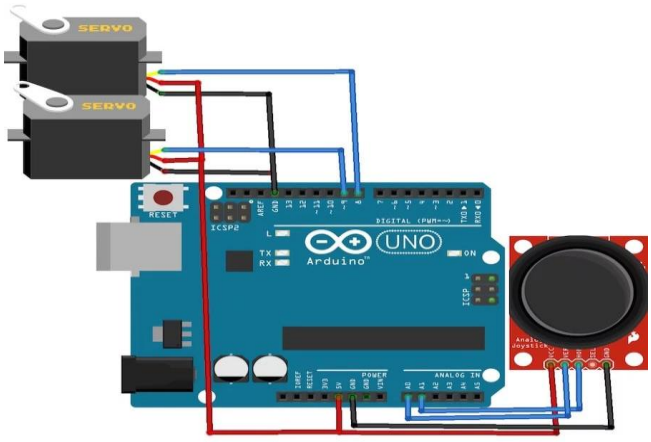
Circuit Diagram

The hardware part of this project is very easy to make. First, connect the joystick module with the Arduino. The connections for the joystick module and the Arduino are as follows:

- Connect the VCC on the joystick module with the 5V pin on the Arduino
- Connect the GND pin on the joystick module with the GND on the Arduino
- Connect the VER pin on the joystick module with the A0 on the Arduino
- Connect the HOR pin on the joystick module with the A1 on the Arduino

After that, connect the servo motors with the Arduino. The connections for servo motors with Arduino are as follows:

- Connect the black wire on both the servo motors with the GND on the Arduino
- Connect the red wire on both the servo motors with the 5V pin on the Arduino
- Connect the yellow wire on the first motor with pin 8 on the Arduino
- Connect the yellow wire on the second motor with pin 9 on the Arduino



Working Procedure

When the joystick module moves in the horizontal or in the vertical direction, it gives us values from 0 to 1023. So we can apply a condition in the code that if the value is less than 300 or greater than 700, then the servos will move.

When the joystick is moved in the horizontal direction, the first servo will move towards right or left and upon moving the joystick in the vertical direction, the second servo will move towards the right or left.

Arduino Code:

```
#include
Servo servo1;
Servo servo2;
int x_key = A1;
int y_key = A0;
int x_pos;
int y_pos;
int servo1_pin = 8;
int servo2_pin = 9;
int initial_position = 90;
int initial_position1 = 90;

void setup ( ) {
Serial.begin (9600) ;
servo1.attach (servo1_pin ) ;
servo2.attach (servo2_pin ) ;
servo1.write (initial_position);
servo2.write (initial_position1);
pinMode (x_key, INPUT) ;
pinMode (y_key, INPUT) ;
}

void loop ( ) {
x_pos = analogRead (x_key) ;
y_pos = analogRead (y_key) ;

if (x_pos<300){
if (initial_position< 10) { } else{ initial_position = initial_position - 20; servo1.write (
initial_position ) ; delay (100) ; } } if (x_pos> 700){
if (initial_position> 180)
{
}
else{
initial_position = initial_position + 20;
servo1.write ( initial_position ) ;
delay (100) ;
}
```

```
}

if (y_pos<300){
if (initial_position1 < 10) {} else{ initial_position1 = initial_position1 - 20; servo2.write
( initial_position1 ) ; delay (100) ; } } if (y_pos> 700){
if (initial_position1 > 180)
{
}
else{
initial_position1 = initial_position1 + 20;
servo2.write ( initial_position1 ) ;
delay (100) ;
}
}
}
```

Code Explanation

First of all, we included the library for the servo motor which will help us with making the code easier. Then, we initialized two variables, one for each of the two servo motors which will help us in using the library functions.

```
#include
Servo servo1;
Servo servo2;
```

Then, we initialized the pins where we have connected the vertical and horizontal pins on the joystick module and also the signal pins on the servos.

```
intx_key= A1;
inty_key= A0;
intx_pos;
inty_pos;
intservo1_pin=8;
intservo2_pin=9;
intinitial_position=90;
intinitial_position1=90;
```

Then we tell the Arduino where we have connected the servo pins and also moved the servo motors at the initial position, which is 90 degrees. After that, we declared both the vertical and horizontal pins on joystick module as the input pins.

```
servo1.attach (servo1_pin ) ;
servo2.attach (servo2_pin ) ;
servo1.write (initial_position);
servo2.write (initial_position1);
pinMode (x_key, INPUT) ;
pinMode (y_key, INPUT) ;
```

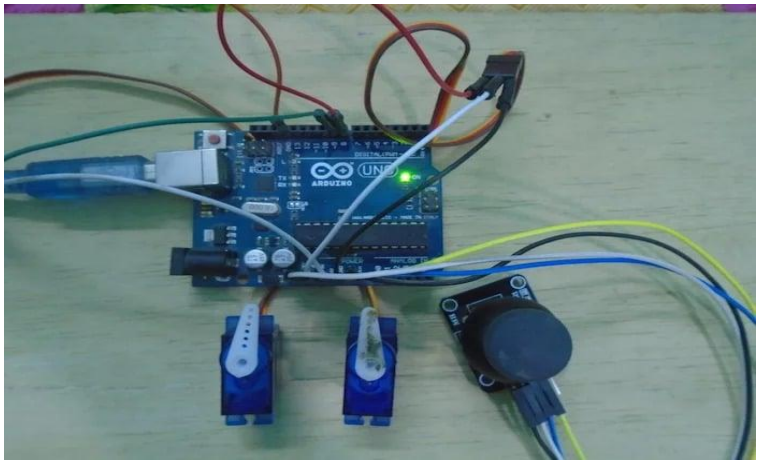
In the loop function, we read the values for the horizontal and the vertical position from the joystick module and saved these in the variables. Then we applied a condition that if the value for the horizontal position is less than 300, then the first servo will move towards the right.

```
x_pos = analogRead (x_key) ;
y_pos = analogRead (y_key) ;
if (x_pos<300){
if (initial_position< 10)
{
}
else{
initial_position = initial_position - 20;
servo1.write ( initial_position ) ;
delay (100) ;
}
}
```

If the value for the horizontal position is greater than 700, then the servo will move towards the left. Similarly for the vertical position of the joystick module, if the value is less than 300, then the second servo will move towards the left, and if the value is greater than 700, then the second servo will move towards the right.

```
if (x_pos>700){
if (initial_position> 180)
{
}
else{
initial_position = initial_position + 20;
servo1.write ( initial_position ) ;
delay (100) ;
}
}
```

Output:



Result: The above experiment is designed and executed successfully.

Experiment – 2

Aim: Calculate the distance to an object with the help of an ultrasonic sensor and display it on an LCD

Procedure:

In this experiment, we are going to interface Ultrasonic sensor HC-SR04 with Arduino and LCD Display. The ultrasonic sensor is used to measure the distance. It acts as a Sonar. It sends an ultrasonic wave of a certain frequency that comes back after hitting the object and calculates the time traveled by it. So let's learn about Distance Measurement Using Arduino & HC-SR04 Ultrasonic Sensor.

Components Required:

1. Arduino Uno Board
2. Ultrasonic Sensor HC-SR04
3. 16*2 LCD Display
4. Breadboard
5. Connecting Wires
6. 5V Power Supply

Ultrasonic Sensor HC-SR04:

Description:

The HC-SR04 ultrasonic sensor uses sonar to determine the distance to an object like bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package.

Distance Measurement Using Arduino & HC-SR04

From 2cm to 400 cm or 1" to 13 feet. Its operation is not affected by sunlight or black material like sharp rangefinders are (although acoustically soft materials like cloth can be difficult to detect). It comes complete with the ultrasonic transmitter and a receiver module.

The specifications of the ultrasonic distance sensor HC-SR04 are below:

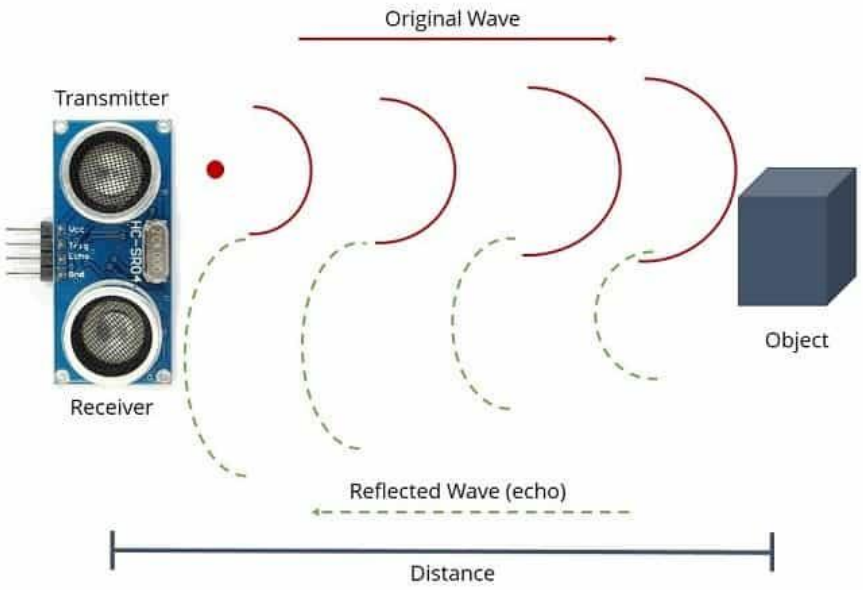
1. Minimum measuring range – 2 cm
2. Maximum measuring range : 400 cm or 4 meter
3. Accuracy : 3 mm
4. Operating Voltage: +5V
5. Operating Current: 15mA
6. Working Frequency: 40 KHz
7. Trigger Input signal: 10us pulse
8. Measuring angle: 15 degrees

Pins:

1. VCC: +5VDC
2. Trig: Trigger (INPUT)
3. Echo: Echo (OUTPUT)
4. GND: GND

Working Procedure:

Ultrasonic sensors emit short, high-frequency sound pulses at regular intervals. These propagate in the air at the velocity of sound. If they strike an object, then they are reflected back as echo signals to the sensor, which itself computes the distance to the target based on the time-span between emitting the signal and receiving the echo.

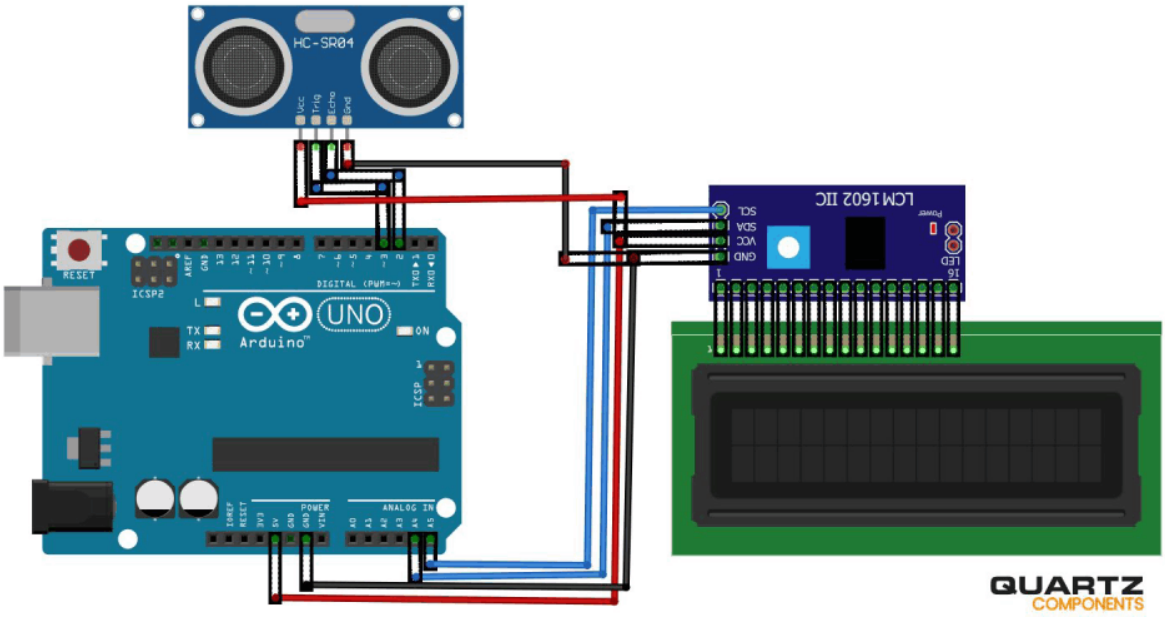


We will have to convert this time into cm to calculate the distance traveled. We will use the following equation to calculate the distance.

$$S = v * t$$

The ultrasonic wave is basically a sound wave that travels at a speed of 340 m/s (0.034 cm/s). The ultrasonic sensor is measuring the time it takes to hit the object and then come back but we need only time that it takes to hit the object. So, we will divide it by 2.

Circuit Diagram and Connections



Trig and Echo pins of the ultrasonic sensor are connected to digital pin 3 & 2 of Arduino. V_{CC} pin of the ultrasonic sensor is connected to the 5v pin of Arduino while the GND pin is connected to the GND of Arduino. SDA & SCL pin of the I2C module is connected to the A4 & A5 pin of Arduino while V_{CC} and GND pins are connected to the 5V & GND pin of Arduino.

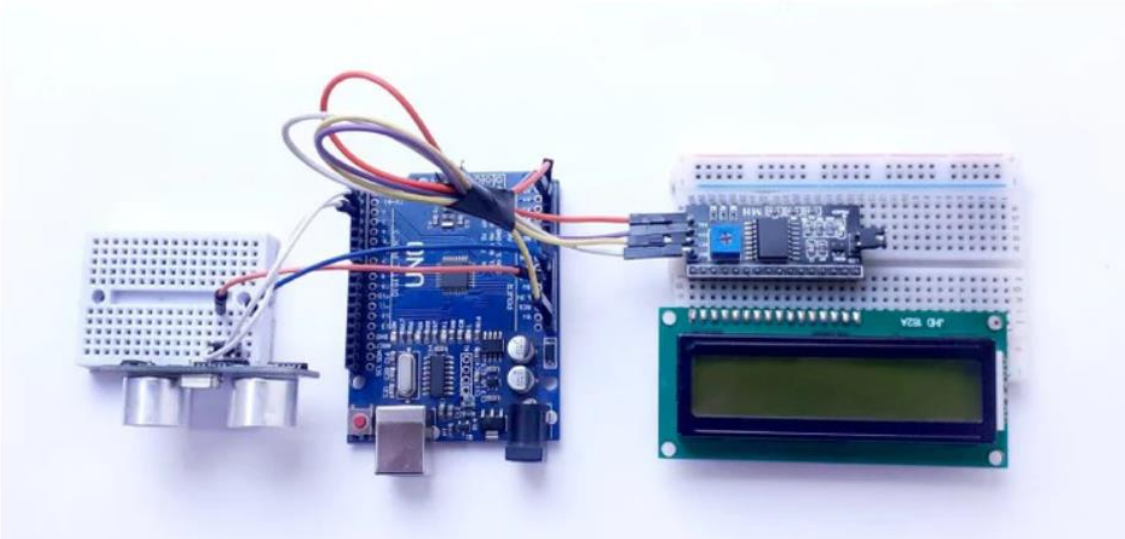
Copy this code then compile and upload it to your Arduino board.

Arduino Code:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
const int trigPin = 3;
const int echoPin = 2;
long duration;
int distance;

void setup() {

lcd.begin(); // Initializes the interface to the LCD display
lcd.backlight();
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
Serial.begin(9600);

}
```

```
void loop() {
lcd.clear();
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);
duration = pulseIn(echoPin, HIGH);
distance = duration * 0.0340 / 2;
Serial.println("Distance");
Serial.println(distance);
lcd.setCursor(0, 0);
lcd.print("Distance: ");
lcd.print(distance);
lcd.print("cm");
delay(1000);
}
```

Output:



Result: The above experiment is designed and executed successfully using Arduino board.

Experiment – 3(a)

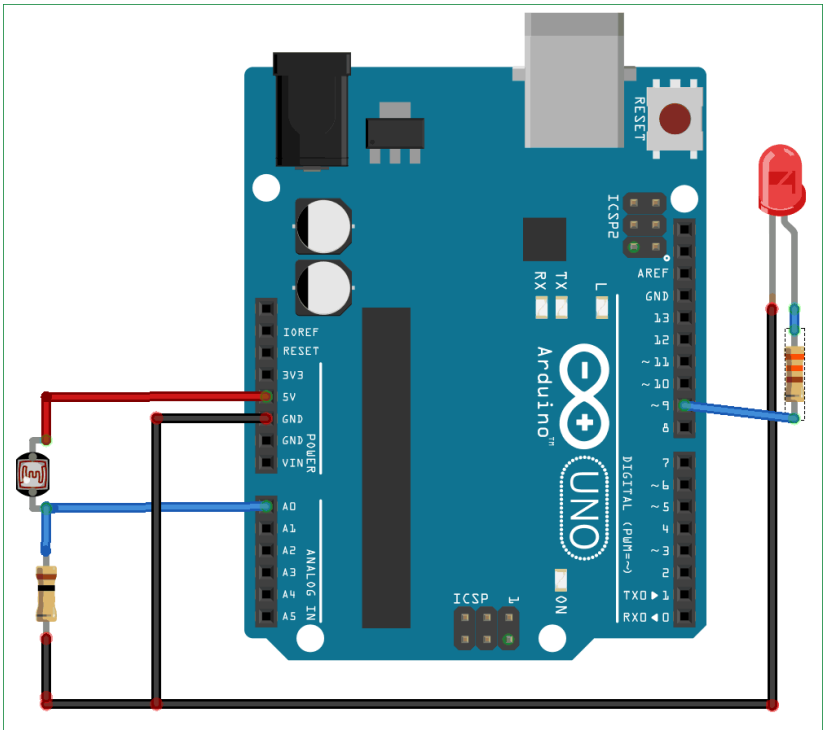
Aim: Controlling relay state based on ambient light levels using LDR sensor.

Procedure: In this circuit, we are making a Light Sensor using LDR with Arduino to control a bulb/CFL as per light condition of the room or outside area.

Components Required:

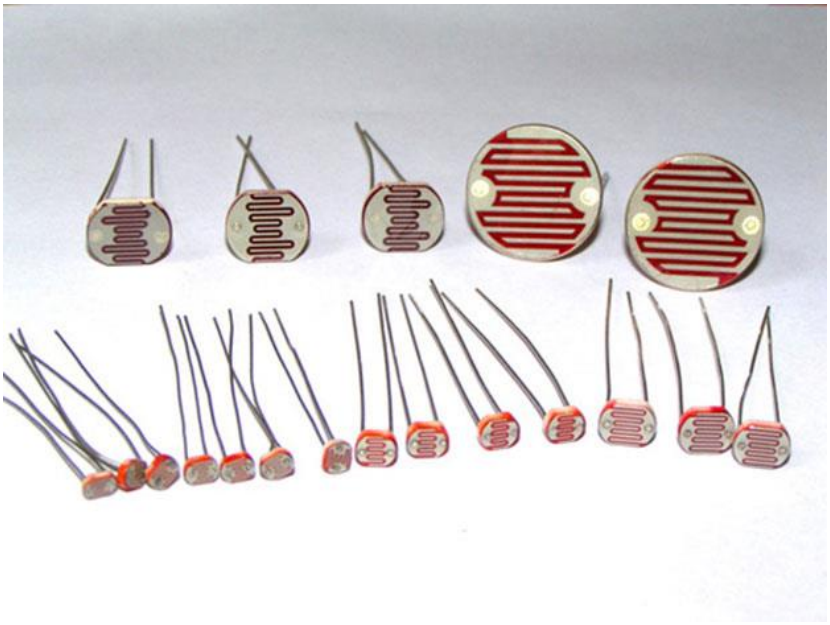
- Arduino UNO
- LDR (Light Dependent Resistor)
- Resistor (100k-1;330ohm-1)
- LED – 1
- Relay module – 5v
- Bulb/CFL
- Connecting wires
- Breadboard

Circuit Diagram and Connections



LDR

LDR is Light Dependent Resistor. LDRs are made from semiconductor materials to enable them to have their light-sensitive properties. There are many types but one material is popular and it is cadmium sulfide (CdS). These LDRs or PHOTO RESISTORS works on the principle of "Photo Conductivity". Now what this principle says is, whenever light falls on the surface of the LDR (in this case) the conductance of the element increases or in other words, the resistance of the LDR falls when the light falls on the surface of the LDR. This property of the decrease in resistance for the LDR is achieved because it is a property of semiconductor material used on the surface.



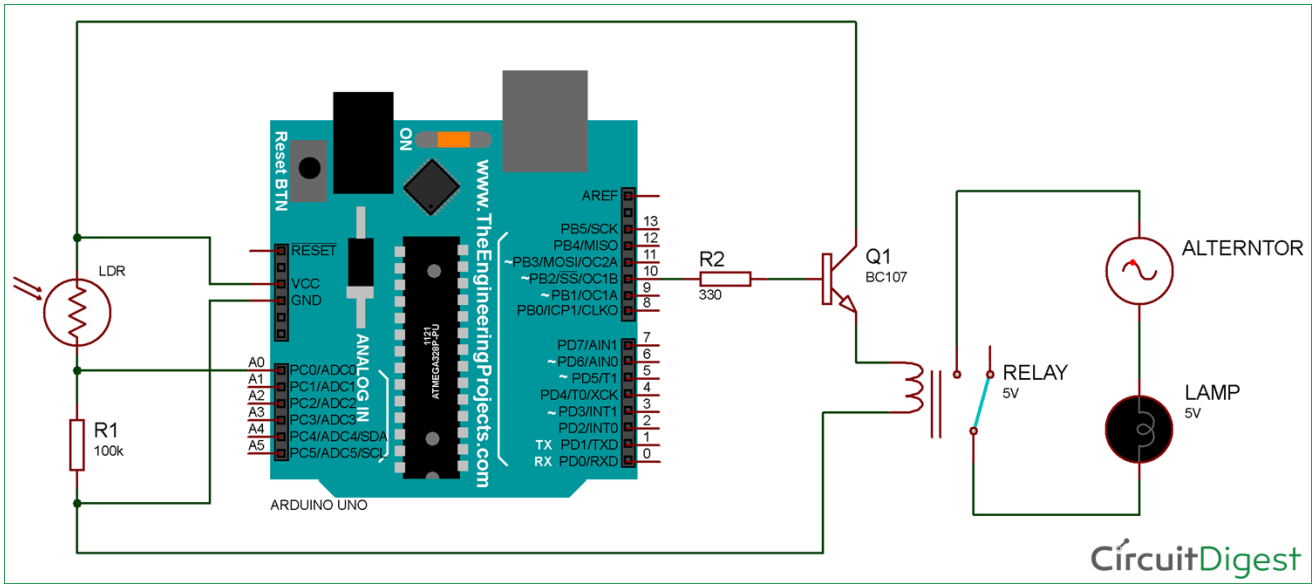
Working of LDR controlled LED using Arduino

As per the circuit diagram, we have made a voltage divider circuit using LDR and 100k resistor. The voltage divider output is feed to the analog pin of the Arduino. The analog Pin senses the voltage and gives some analog value to Arduino. The analog value changes according to the resistance of LDR. So, as the light falls on the LDR the resistance of it gets decreased and hence the voltage value increase.

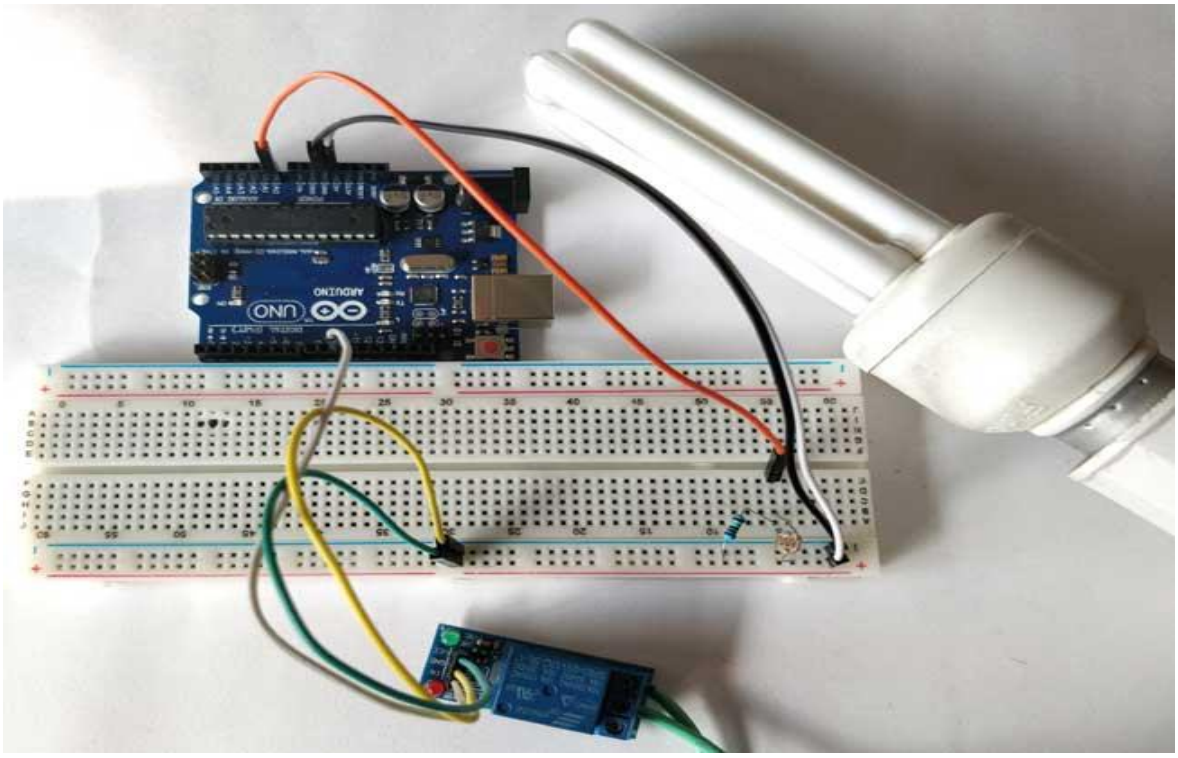
Intensity of light ↓ - Resistance↑ - Voltage at analog pin↓ - **Light turns ON**

As per the Arduino code, if the analog value falls below 700 we consider it as dark and the light turns ON. If the value comes above 700 we consider it as bright and the light turns OFF.

Controlling Relay using LDR with Arduino



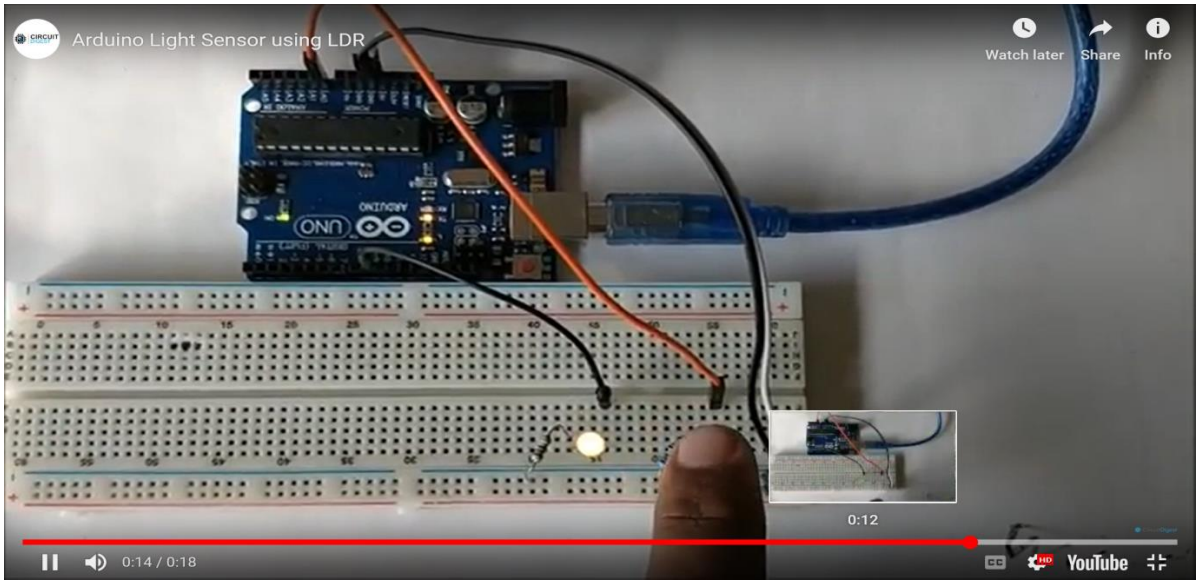
Instead of controlling an LED according to the brightness and darkness, we can control our home lights or any electrical equipment. All we have to do is connect a relay module and set the parameter to turn ON and OFF the any AC appliance according to the intensity of the light. If the value falls below 700, which means it Dark, then the relay operates and the lights turns ON. If the value is greater than 700, which means its day or bright, then the relay will not operate and the lights remain OFF.



Arduino Code

```
#define relay 10
int LED = 9;
int LDR = A0;
void setup()
{
Serial.begin(9600);
pinMode(LED, OUTPUT);
pinMode(relay, OUTPUT);
pinMode(LDR, INPUT);
}
void loop() {
int LDRValue = analogRead(LDR);
Serial.print("sensor = ");
Serial.print(LDRValue);
if (LDRValue<=700)
{
digitalWrite(LED, HIGH);
digitalWrite(relay, HIGH);
Serial.println("It's Dark Outside; Lights status: ON");
}
else
{
digitalWrite(LED, LOW);
digitalWrite(relay, LOW);
Serial.println("It's Bright Outside; Lights status: OFF");
}
}
```

Output:



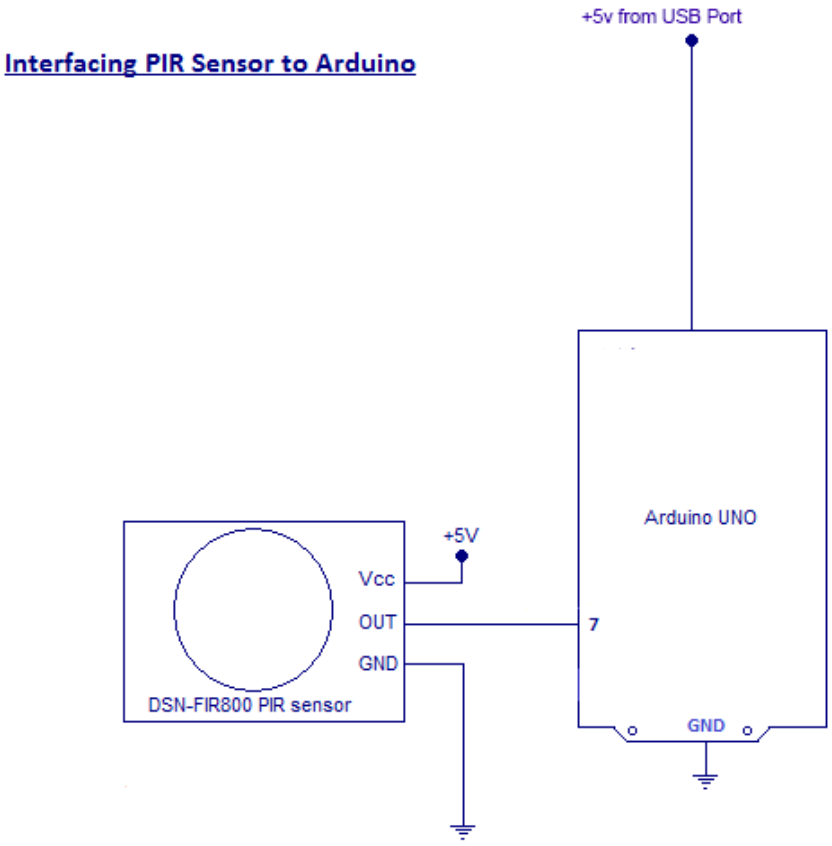
Result: The above experiment is designed and executed successfully using Arduino Board.

Experiment – 3(b)

Aim: Basic Burglar alarm security system with the help of PIR sensor and buzzer using Arduino

Procedure

A PIR sensor is generally known to the world as motion sensor or motion detector.A PIR sensor do not emit any kind of radiation for detection purposes but they just measure the infra red radiation emitted by other objects inside its field or range of measurement.
A PIR sensor module has only 3 pins – one is Vcc which is a +5 volts input, a ground pin and finally the digital output pin. Connect +5V from Arduino to Vcc of PIR sensor module, connect a GND from Arduino to ground of PIR sensor and finally connect the output pin (marked as 'out') to any digital pin of arduino. In our circuit diagram, we have connected it to pin 7 of arduino.



PIR Sensor – is the heart of this simple burglar alarm circuit using arduino. A PIR sensor – is basically a motion sensor or a motion detector which identifies any object

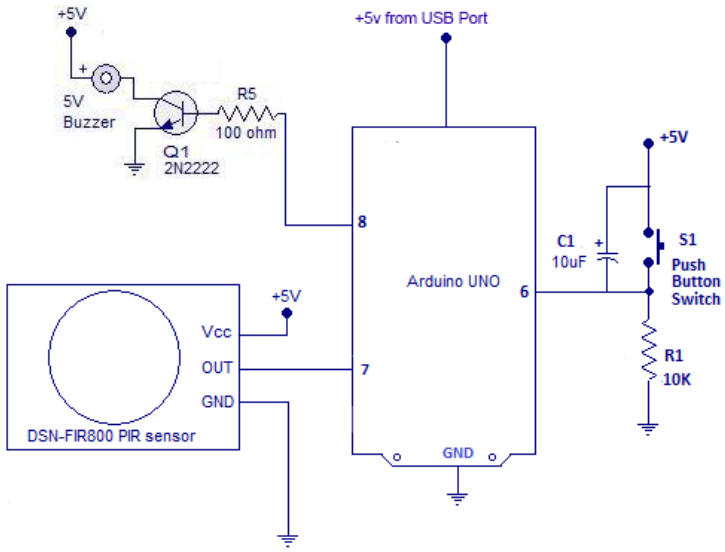
that moves inside its range of view. PIR sensor identifies infra red radiations emitted by any object under its radar range.

Buzzer – is used to create a sound alarm when ever a movement is identified inside the range of PIR sensor. A transistor 2N2222 is used to drive the buzzer. The maximum current that can be sourced or sinked from an arduino pin is 20mA (the total current being 200mA from different pins). But the buzzer will need more than just 20mA for its proper functioning. So how to give the necessary current required fir buzzer ? We use switching transistor 2N222 for this purpose. It can act as a switch and at the same time it provides the required current amplification.

A 2N2222 transistor with a gain of 100 can give up to 1A current at its output. Another purpose of using a transistor in between arduino pin and buzzer is isolation. A short circuit of the buzzer will destroy only the collector – emitter junction of transistor. Since there is isolation at the base region of transistor (base is connected to arduino), the destruction of collector-emitter junction will not affect base and hence our arduino will be safe from getting burned! The 100 ohms resistor at base is used to limit base current of transistor.

Switch – a push button switch is used to reset the burglar alarm once its activated. The capacitor is used for bypassing bouncing effects of a switch (debouncing capacitor).

Burglar Alarm using Arduino – Circuit Diagram



Connections

Arduino – Pin 7 – Output of PIR Sensor | Pin 6 – Push button switch | Pin 8 – Buzzer

Buzzer – + pin to Vcc (5 volts) | other pin to collector side of 2N2222

Transistor – 2N2222 – NPN – Collector to Buzzer | Emitter to Ground | Base to Arduino through 100 Ohm Resistor

Switch – One end of switch to +5V | Other end to Ground through a 10K current limiting resistor

PIR Sensor – has got 3 pins – Vcc to +5 volts | GND to Ground | OUT pin to Arduino pin 7

Note:-Wire all grounds together at a common point.

Arduino Code for Bulgar Alarm using PIR Sensor and Buzzer

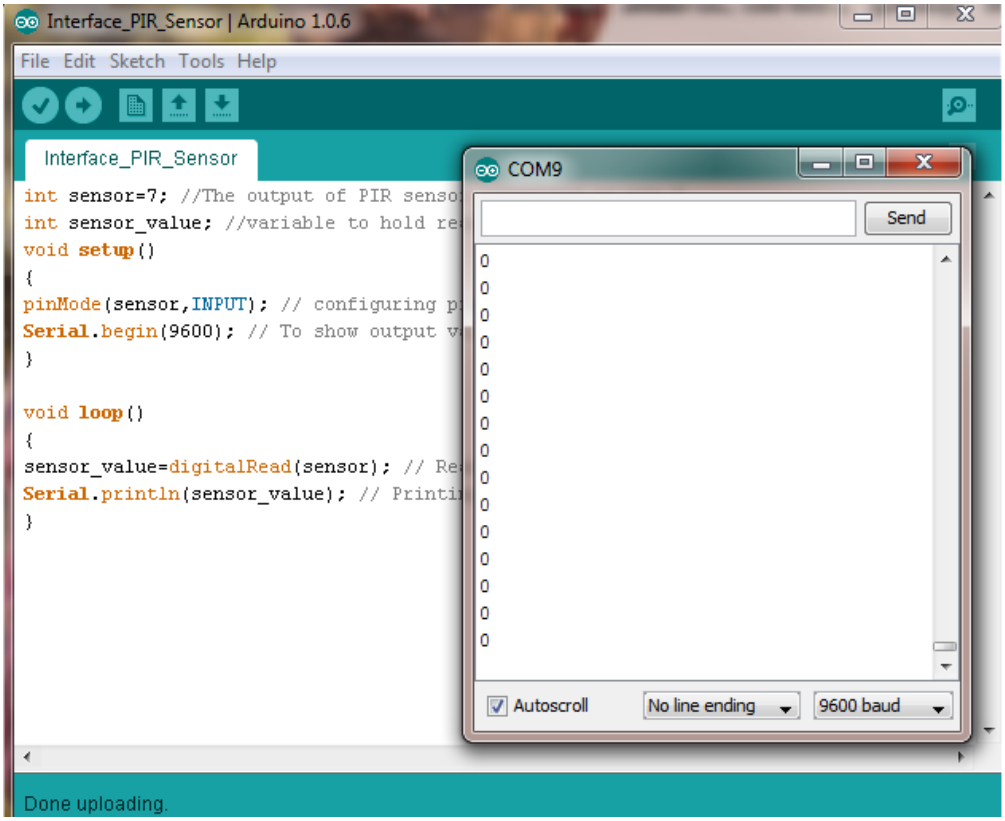
```
int sensor=7; //The output of PIR sensor connected to pin 7
int push_switch=6; // push button switch connected to pin 6
int buzzer=8; // buzzer connected at pin 8
int sensor_value; //variable to hold read sensor value
void setup()
{
pinMode(sensor,INPUT); // configuring pin 7 as Input
pinMode(push_switch,INPUT); // configuring pin 6 as Input
pinMode(buzzer,OUTPUT); // configuring pin 8 as OUTPUT
}
void loop()
{
sensor_value=digitalRead(sensor); // Reading sensor value from pin 7
if(sensor_value==HIGH) // Checking if PIR sensor sends a HIGH signal to Arduino
{
digitalWrite(buzzer,HIGH); // Activating the buzzer
}
if(digitalRead(push_switch==HIGH))// Checking if pushbutton was pressed
{
digitalWrite(buzzer,LOW); // turning OFF the buzzer
}}
```

Arduino Code for PIR Sensor Interfacing

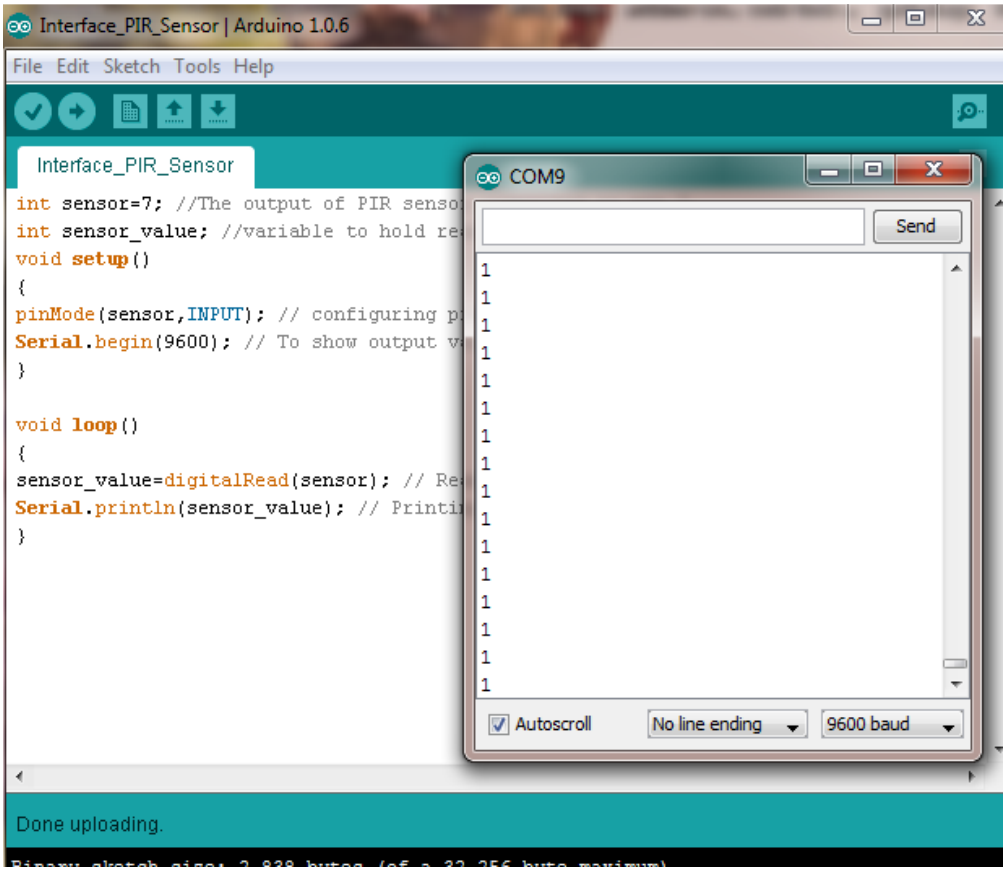
```
int sensor=7; //The output of PIR sensor connected to pin 7
int sensor_value; //variable to hold read sensor value
void setup()
{
pinMode(sensor,INPUT); // configuring pin 7 as Input
Serial.begin(9600); // To show output value of sensor in serial monitor
}

void loop()
{
sensor_value=digitalRead(sensor); // Reading sensor value from pin 7
Serial.println(sensor_value); // Printing output to serial monitor
}
```

Output:

when there is no motion inside range



Output of PIR Sensor in Arduino Serial Monitor – when there is motion detected in range



Result: The above experiment is designed and executed successfully.

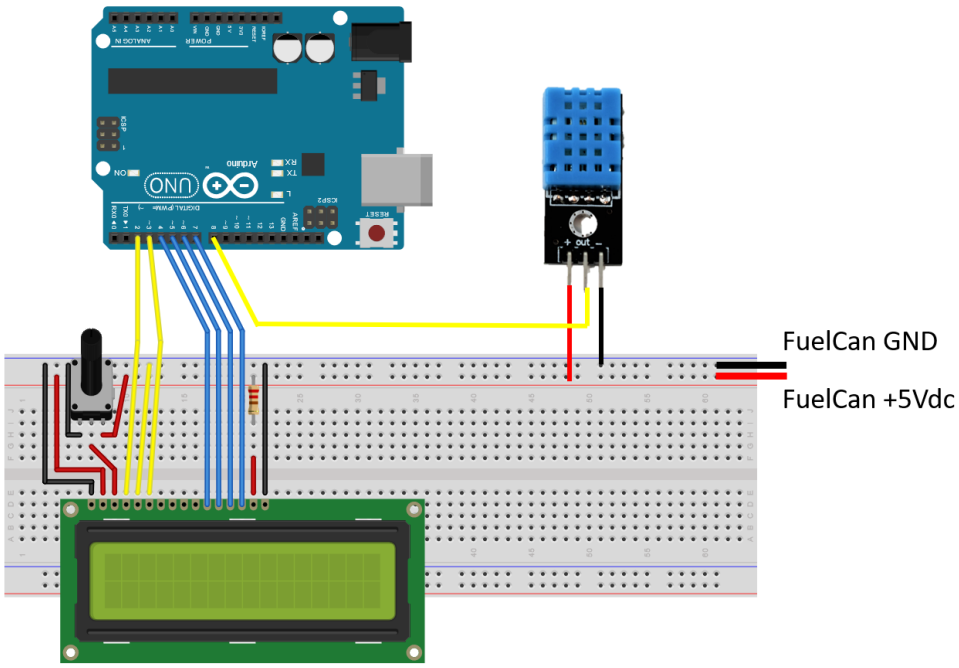
Experiment – 3(c)

Aim: Displaying humidity and temperature values on LCD

Procedure

Components Required:

- Arduino Uno Board
- 16 x 2 LCD Display
- Humidity Sensor
- Temperature Sensor



Interface 16x2 LCD to Arduino by connecting pins 4,6, and 11-14 of the LCD connected to Uno pins 2,3, and 4-7, respectively. Connect the LCD using a solderless breadboard.

DHT11 Temperature and Humidity Sensor

There are different types of DHT Sensors such as DHT11, DHT21, DHT22, DHT33, and DHT44. They all are used to measure temperature and humidity, but the difference lies mostly in their accuracy and sampling rate.

3 pin DHT11 Sensor

DHT11 Sensor mounted to Modulus with R/A Female Header.

Arduino Code

```
//Interface the DHT11 Temp & Humidity sensor and display humidity and
temperature
//in Celsius on a 16x2 character LCD

#include <dht.h>
#include   <LiquidCrystal.h>

dht DHT;
const int RS = 2, EN = 3, D4 = 4, D5 = 5, D6   = 6, D7 = 7;
LiquidCrystallcd(RS,EN,D4,D5,D6,D7);   //set Uno pins that are   connected to
LCD, 4-bit mode

void setup() {
lcd.begin(16,2);    //set   16 columns and 2 rows of 16x2 LCD

}

void loop() {
  int readDHT   = DHT.read11(8);    //grab 40-bit data packet from DHT sensor
lcd.setCursor(0,0);
lcd.print("Temp: ");
lcd.print(DHT.temperature);
  //lcd.print((char)223);          //used to display degree symbol on display
  //lcd.write(0xdf);              //another   way to display degree symbol
lcd.print("C");
lcd.setCursor(0,1);
```

```
lcd.print("Humidity: ");
lcd.print(DHT.humidity);
lcd.print("%");
delay(3000);

}
```

Result: The above experiment is designed and executed successfully.

Experiement – 4(a)

Aim: Controlling relay state based on input from IR sensors

Procedure

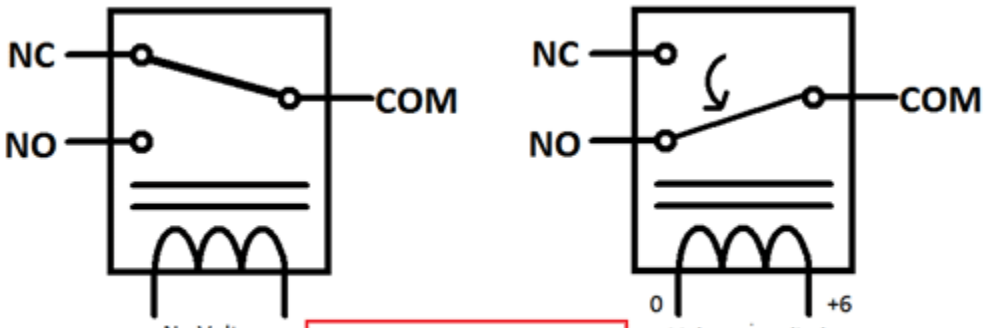
Whenever we need to connect any AC Appliance in our embedded circuits, we use a Relay.Use an NPN transistor to control relay.

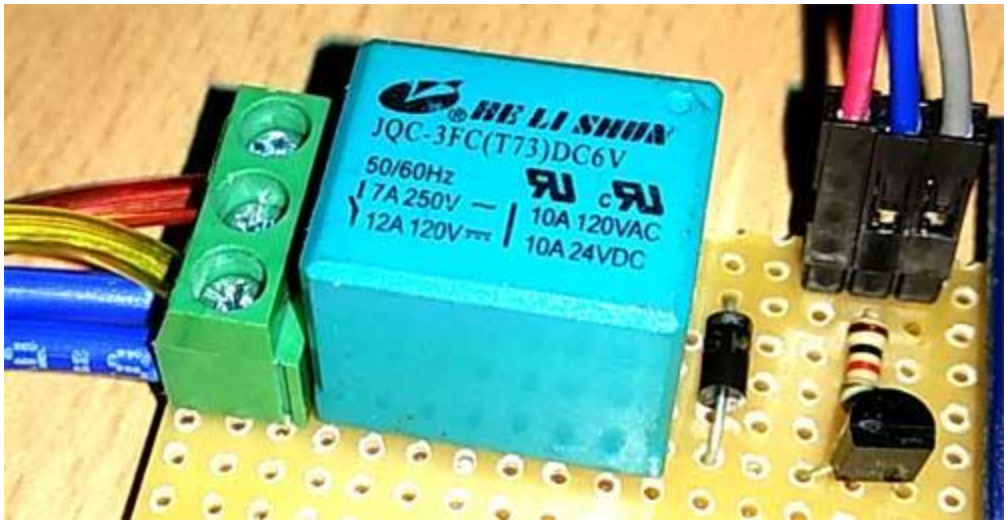
Components Required:

1. Arduino
2. 5v or 6v relay
3. AC appliance or Bulb
4. BC547 transistor
5. 1k resistor
6. Breadboard or PCB
7. Connecting jumper wire
8. Power supply
9. 1n4007 diode
10. Screw terminal or terminal block

Relay:

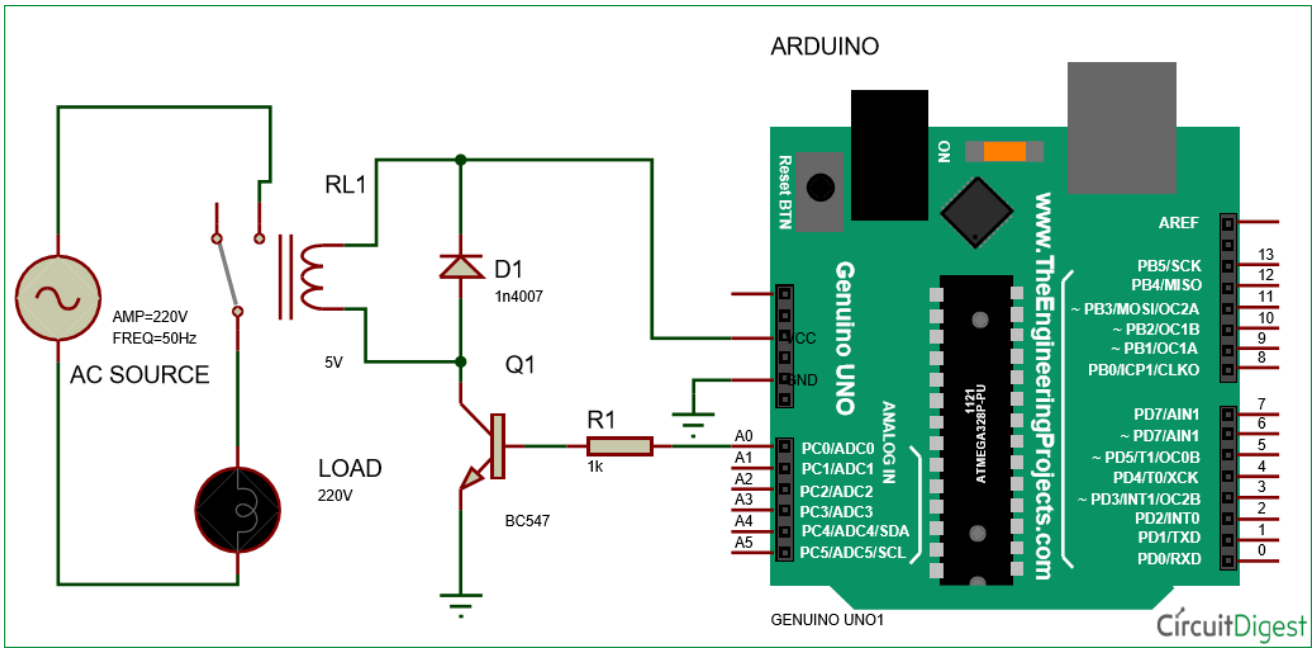
Relay is an electromagnetic switch, which is controlled by small current, and used to switch ON and OFF relatively much larger current. Means by applying small current we can switch ON the relay which allows much larger current to flow. A relay is a good example of controlling the AC (alternate current) devices, using a much smaller DC current. Commonly used Relay is **Single Pole Double Throw (SPDT) Relay**, it has five terminals as below:



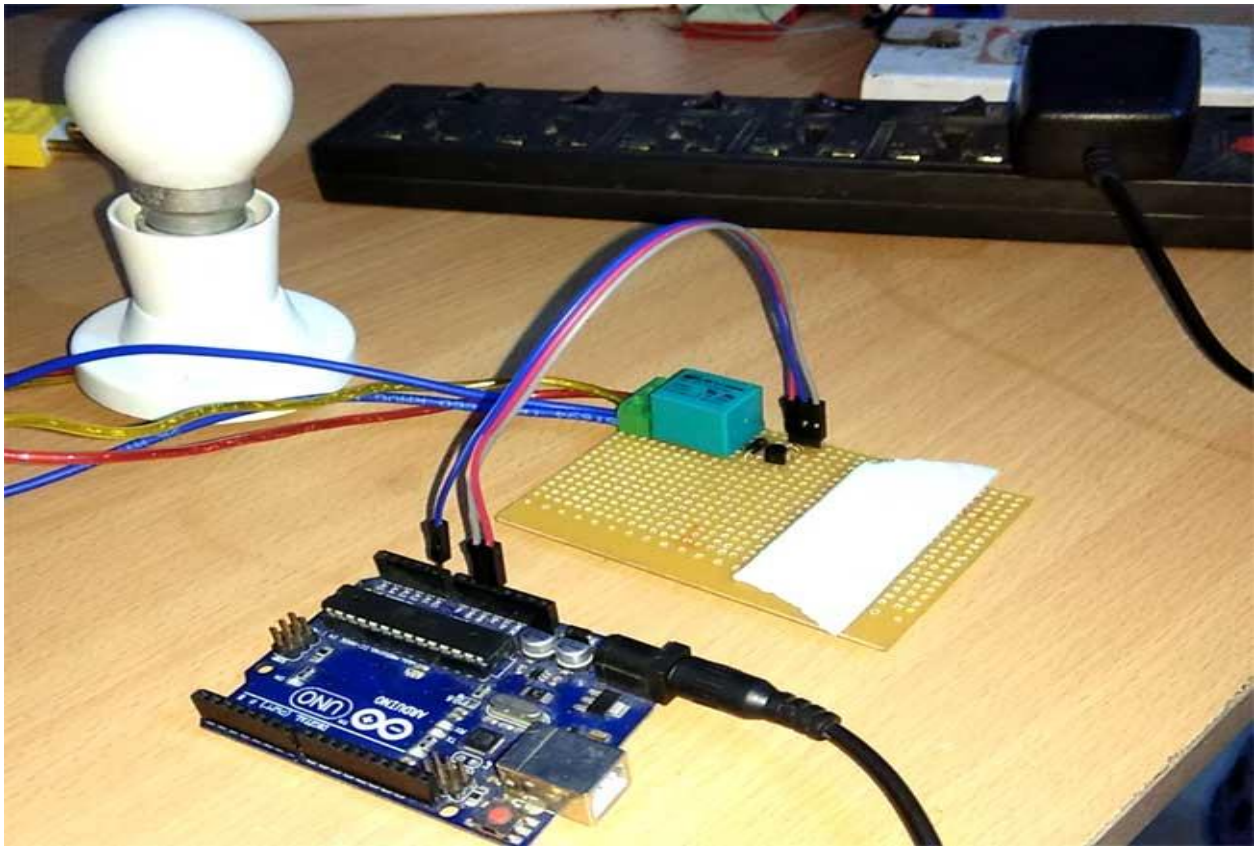


Here to **turn on the Relay with Arduino** we just need to make that Arduino Pin High (A0 in our case) where Relay module is connected.

Circuit Diagram and Working:



In this **Arduino Relay Control Circuit** we have used Arduino to control the relay via a BC547 transistor. We have connected transistor base to Arduino pin A0 through a 1k resistor. An AC bulb is used for demonstration. The 12v adaptor is used for powering the circuit.



Working is simple, we need to **make the RELAY Pin (PIN A0) high to make the Relay module ON** and **make the RELAY pin low to turn off the Relay Module**. The AC light will also turn on and off according to Relay.

We just programmed the Arduino to make the Relay Pin (A0) High and Low with a delay of 1 second:

```
void loop()

{

digitalWrite(relay, HIGH);
   delay(interval);
digitalWrite(relay, LOW);
   delay(interval);
}
// Arduino Relay Control Code

 #define relay A0
#define interval 1000

void setup() {
  pinMode(relay, OUTPUT);

}
```

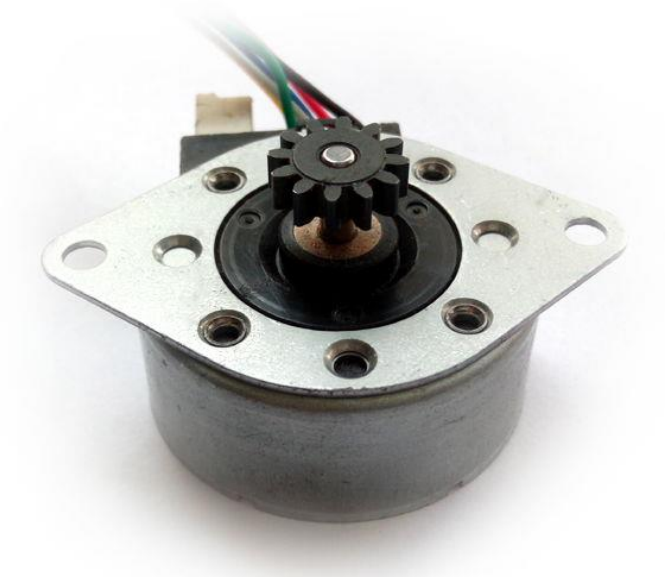
```
void loop()

{
  digitalWrite(relay, HIGH);
  delay(interval);
  digitalWrite(relay, LOW);
  delay(interval);
}
```

Result: The above experiment is designed and executed successfully.

Experiment – 4(b)

Aim: Interfacing stepper motor with R-Pi

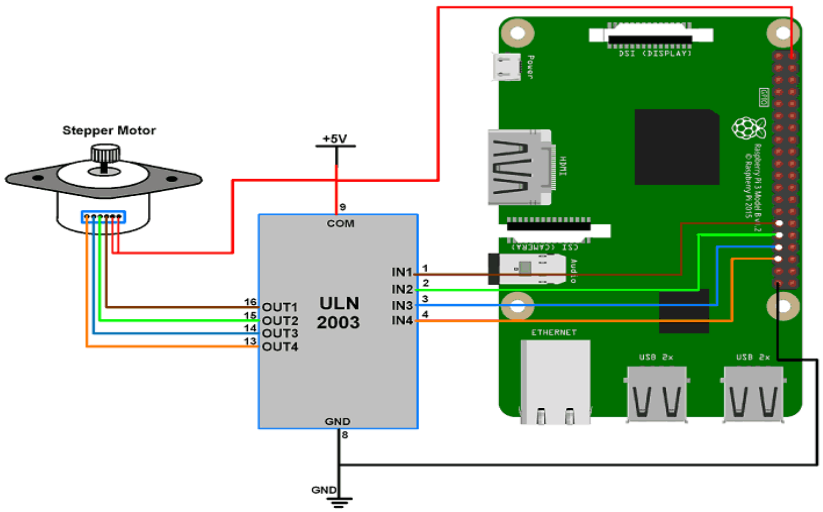
Procedure



Stepper Motor

- Stepper motor is a brushless DC motor that divides the full rotation angle of 360° into number of equal steps.
- The motor is rotated by applying certain sequence of control signals. The speed of rotation can be changed by changing the rate at which the control signals are applied.
- For more information about Stepper Motor, its control sequence and how to use it, refer the topic Stepper Motor in the sensors and modules section.
- Raspberry Pi's GPIOs can be used to control stepper motor rotation.

Connection Diagram of Stepper Motor with Raspberry Pi



Rotate Stepper Motor using Raspberry Pi

Let's rotate a Stepper Motor in clockwise and counter-clockwise directions alternately.

- Here, we are using six wire unipolar stepper motor. Only four wires are required to control this stepper motor. The two center tap wires of the stepper motor are connected to the 5V supply.
- ULN2003 driver is used to drive unipolar stepper motor.
- We will interface Stepper Motor with Raspberry Pi using Python language. In this program, we have used the keyboard key as input for selecting motor rotation direction (i.e. clockwise or anti-clockwise).

Stepper Motor Python Program for Raspberry Pi

```
'''
   Stepper Motor interfacing with Raspberry Pi
   http:///www.electronicwings.com
'''
importRPi.GPIOas GPIO
from time import sleep
import sys

#assign GPIO pins for motor
motor_channel = (29,31,33,35)
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
#for defining more than 1 GPIO channel as input/output use
GPIO.setup(motor_channel, GPIO.OUT)

motor_direction = input('select motor direction a=anticlockwise, c=clockwise: ')
whileTrue:
try:
if(motor_direction == 'c'):
print('motor running clockwise\n')
GPIO.output(motor_channel, (GPIO.HIGH,GPIO.LOW,GPIO.LOW,GPIO.HIGH))
sleep(0.02)
GPIO.output(motor_channel, (GPIO.HIGH,GPIO.HIGH,GPIO.LOW,GPIO.LOW))
sleep(0.02)
GPIO.output(motor_channel, (GPIO.LOW,GPIO.HIGH,GPIO.HIGH,GPIO.LOW))
sleep(0.02)
GPIO.output(motor_channel, (GPIO.LOW,GPIO.LOW,GPIO.HIGH,GPIO.HIGH))
sleep(0.02)

elif(motor_direction == 'a'):
print('motor running anti-clockwise\n')
GPIO.output(motor_channel, (GPIO.HIGH,GPIO.LOW,GPIO.LOW,GPIO.HIGH))
sleep(0.02)
GPIO.output(motor_channel, (GPIO.LOW,GPIO.LOW,GPIO.HIGH,GPIO.HIGH))
```

```
sleep(0.02)
GPIO.output(motor_channel, (GPIO.LOW,GPIO.HIGH,GPIO.HIGH,GPIO.LOW))
sleep(0.02)
GPIO.output(motor_channel, (GPIO.HIGH,GPIO.HIGH,GPIO.LOW,GPIO.LOW))
sleep(0.02)


#press ctrl+c for keyboard interrupt
exceptKeyboardInterrupt:
#query for setting motor direction or exit
motor_direction = input('select motor direction a=anticlockwise, c=clockwise or
q=exit: ')
#check for exit
if(motor_direction == 'q'):
print('motor stopped')
sys.exit(0)
```

Result: The above experiment is designed and executed successfully.

Experiment – 4(C)

Aim: Advanced burglar alarm security system with the help of PIR sensor, buzzer and keypad. (Alarm gets disabled if correct keypad password is entered)

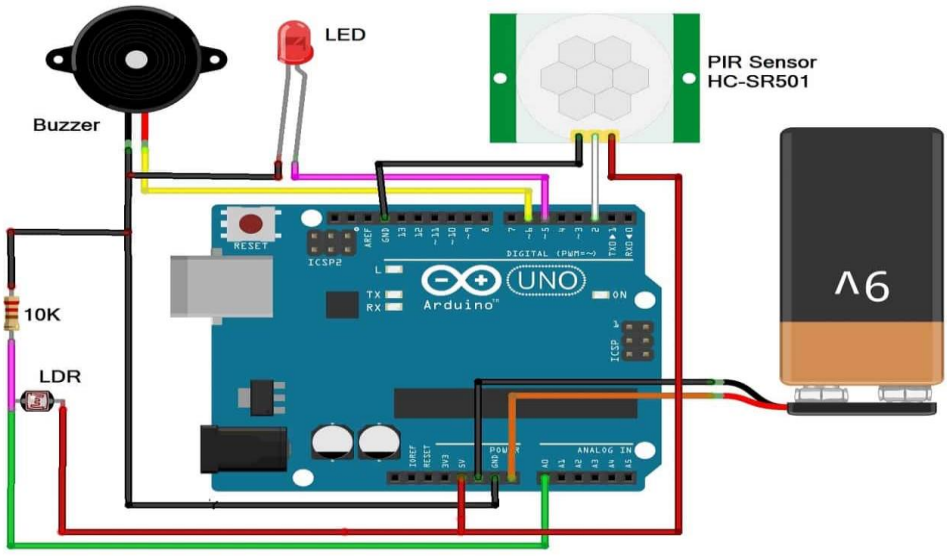
Procedure

Advanced Bulgar Alarm Using PIR Sensor & Arduino for Night time only. A night security light only turns on when it's dark and when movement is detected. The lamp & the buzzer turns on when it's dark & movement is detected. When there's light, the lamp is turned off, even when motion is detected.

Components Required:

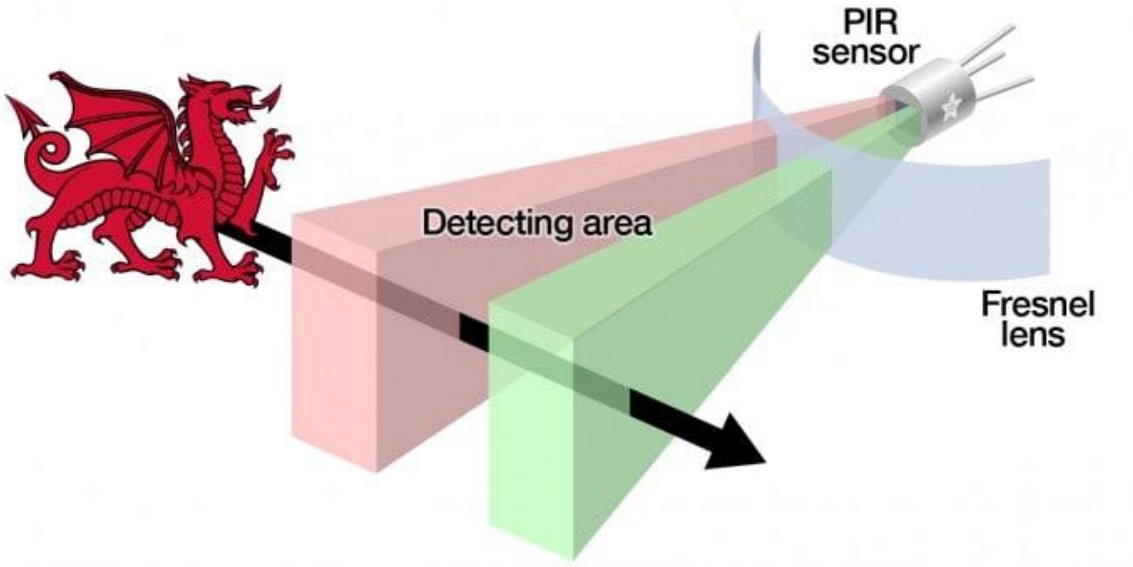
1. Arduino UNO Board
2. PIR Sensor HC-SR501
3. LDR
4. 10K Resistor
5. LED
6. Buzzer
7. 9V Battery

Circuit Diagram & Connections:



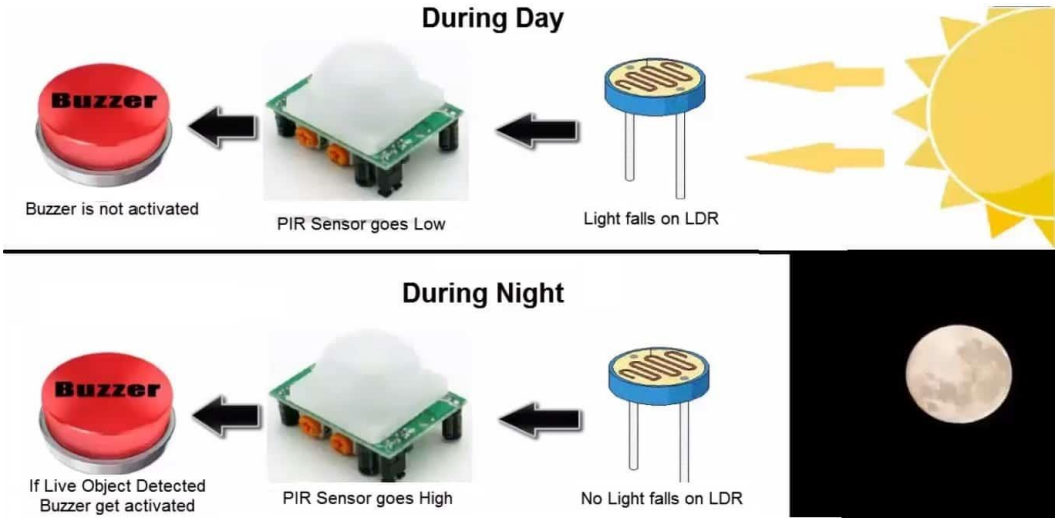
Working of PIR Sensor:

PIR has multiple variables that affect the sensor's input and output.



The PIR sensor itself has two slots in it, each slot is made of a special material that is sensitive to IR. The lens used here is not really doing much and so we see that the two slots can 'see' out past some distance (basically the sensitivity of the sensor). When the sensor is idle, both slots detect the same amount of IR, the ambient amount radiated from the room or walls or outdoors. When a warm body like a human or animal passes by, it first intercepts one-half of the PIR sensor, which causes a positive differential change between the two halves. When the warm body leaves the sensing area, the reverse happens, whereby the sensor generates a negative differential change. These change pulses are what is detected.

The circuit is in fact a dark activated switch that measures the ambient light level and will enable the system only when ambient light level is below a threshold value. Here an LDR (Light Dependent Resistor) is used to measure the light level. The alarm system is triggered when a "Logic High (H)" level signal is detected at its sensor input port.



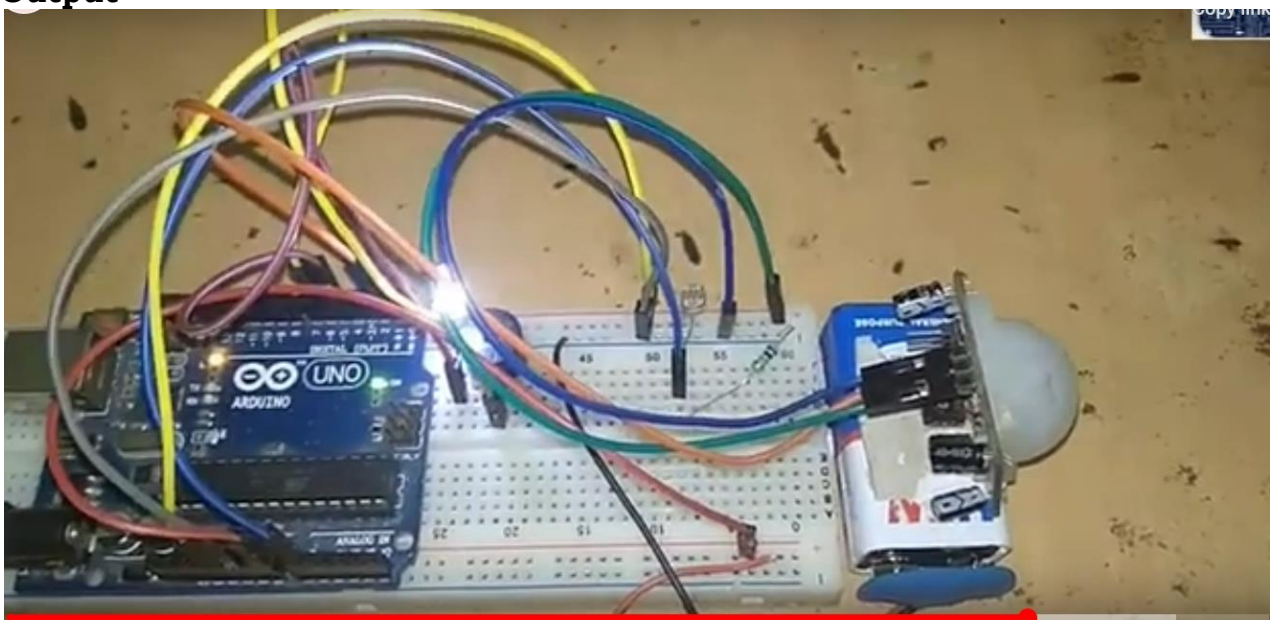
The best sensor you can use to detect an intrusion is the Passive Infrared (PIR) Sensor. The PIR Sensor detects the motion of a human body by the change in surrounding ambient temperature when a human body passes across, and effectively controls the switchingwhen it detects a moving target.

Copy the code and paste it to your Arduino IDE and then compile & finally, upload it to the Arduino Board.

Arduino Code

```
int Buzzer = 6; // choose the pin for the Buzzer
int inputPin = 2; // choose the input pin (for PIR sensor)
int pirState = LOW; // we start, assuming no motion detected
int val = 0; // variable for reading the pin status
void setup() {
pinMode(ledPin, OUTPUT); // declare LED as output
pinMode(Buzzer, OUTPUT); // declare Buzzer as output
pinMode(inputPin, INPUT); // declare sensor as input
Serial.begin(9600);
}
void loop(){
val = digitalRead(inputPin); // read input value
int value_ldr = analogRead(A0); // read LDR value
if((300>value_ldr) && ( val==HIGH) ){
if (val == HIGH) { // check if the input is HIGH
digitalWrite(ledPin, HIGH); // turn LED ON
digitalWrite(Buzzer, 1); // turn Buzzer ON
delay(5000);
if (pirState == LOW) {
// we have just turned on
Serial.println("Motion detected!");
// We only want to print on the output change, not state
pirState = HIGH;
}
} else {
digitalWrite(ledPin, LOW); // turn LED OFF
digitalWrite(Buzzer, 0); // turn Buzzer OFF
if (pirState == HIGH){
// we have just turned of
Serial.println("Motion ended!");
// We only want to print on the output change, not state
pirState = LOW;
}
}
}
}
```

Output



Result: The above experiment is designed and executed successfully.

Experiment – 4 (d)

Aim: Automated LED light control based on input from PIR (to detect if people are present) and LDR (ambient light level)

Procedure

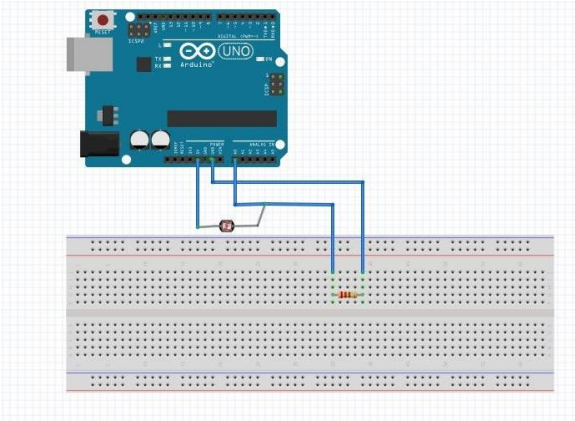
Hardware Supplies:
1. Arduino Uno (any other arduino board will be just fine as long as it provides an Analogical pin).
2. PIR Motion Sensor
3. LDR (Photoresistor)
4. 10 KOhms resistor
5. Relay module
6. Lamp
7. Breadboard (optional)

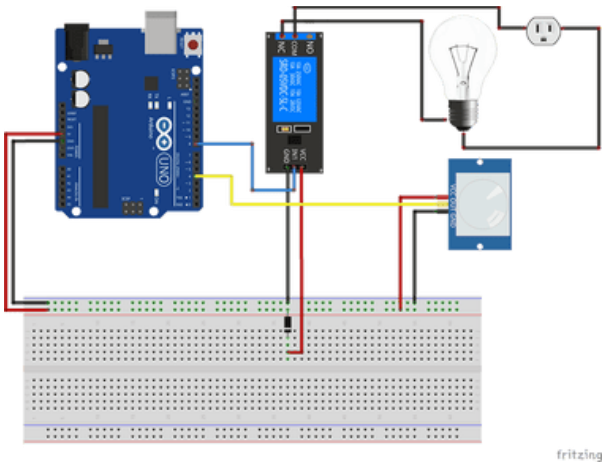
Software Supplies:
1. Arduino IDE

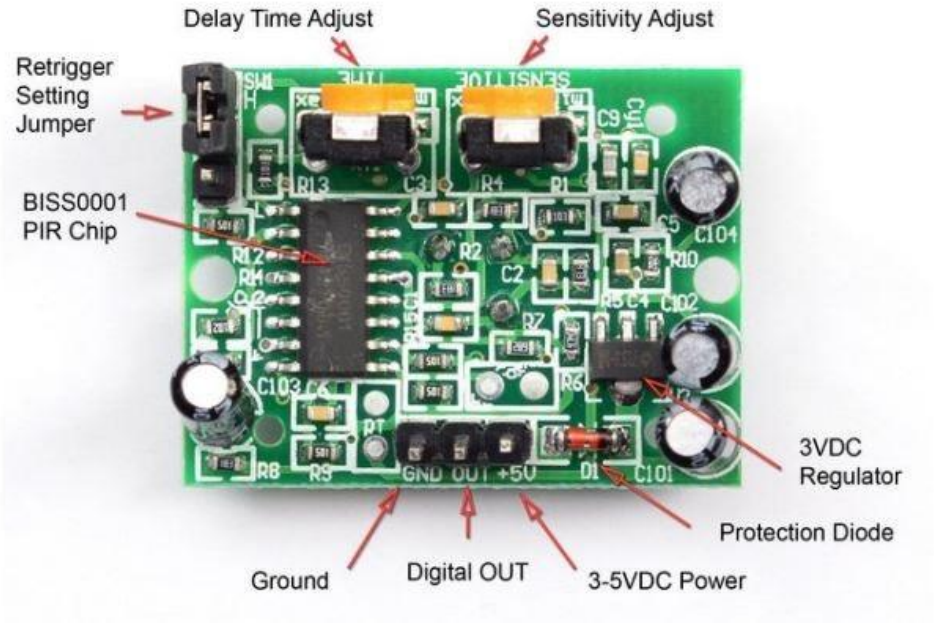
PIR sensor:
The PIR sensor stands for Passive Infrared sensor. It is a low cost sensor which can detect the presence of Human beings or animals. There are two important materials present in the sensor one is the pyroelectric crystal which can detect the heat signatures from a living organism (humans/animals) and the other is a Fresnel lenses which can widen the range of the sensor. Also the PIR sensor modules provide us some options to adjust the working of the sensor as shown in above image.

LDR resistor:
A photoresistor (or light-dependent resistor, LDR, or photo-conductive cell) is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity. a photoresistor is made of a high resistance semiconductor. In the dark, a photoresistor can have a resistance as high as several megohms (MΩ), while in the light, a photo resistor can have a resistance as low as a few hundred ohms.

Circuit Diagram and Connections







1. Arduino and LDR

The schematic is quite simple you should just follow the instructions bellow :

1.Connect one of the LDR leg to the VCC (5v of the Arduino).

2.Connect the other LDR leg to the A4 pin Of Arduino and also to the resistor

3.Connect the (empty) resistor to the GND of the Arduino.

Note: You can see all the connections in the picture above.

2. Arduino and PIR motion sensor

The PIR sensor has three pins:

1. VCC

2. GND

3. OUT

We have powered the PIR sensor using he 5V Rail of the Arduino. The output pin of the PIR Sensor is connected to the 8thdigital pin. Then, you have to wire the "GND" to the Arduino's "GND".

In this experiment, we use relay for controlling AC light because the Arduino cannot control high volt, but a relay can do this job, which is the sole design of it. so we are using relay as switch to control high power devices.

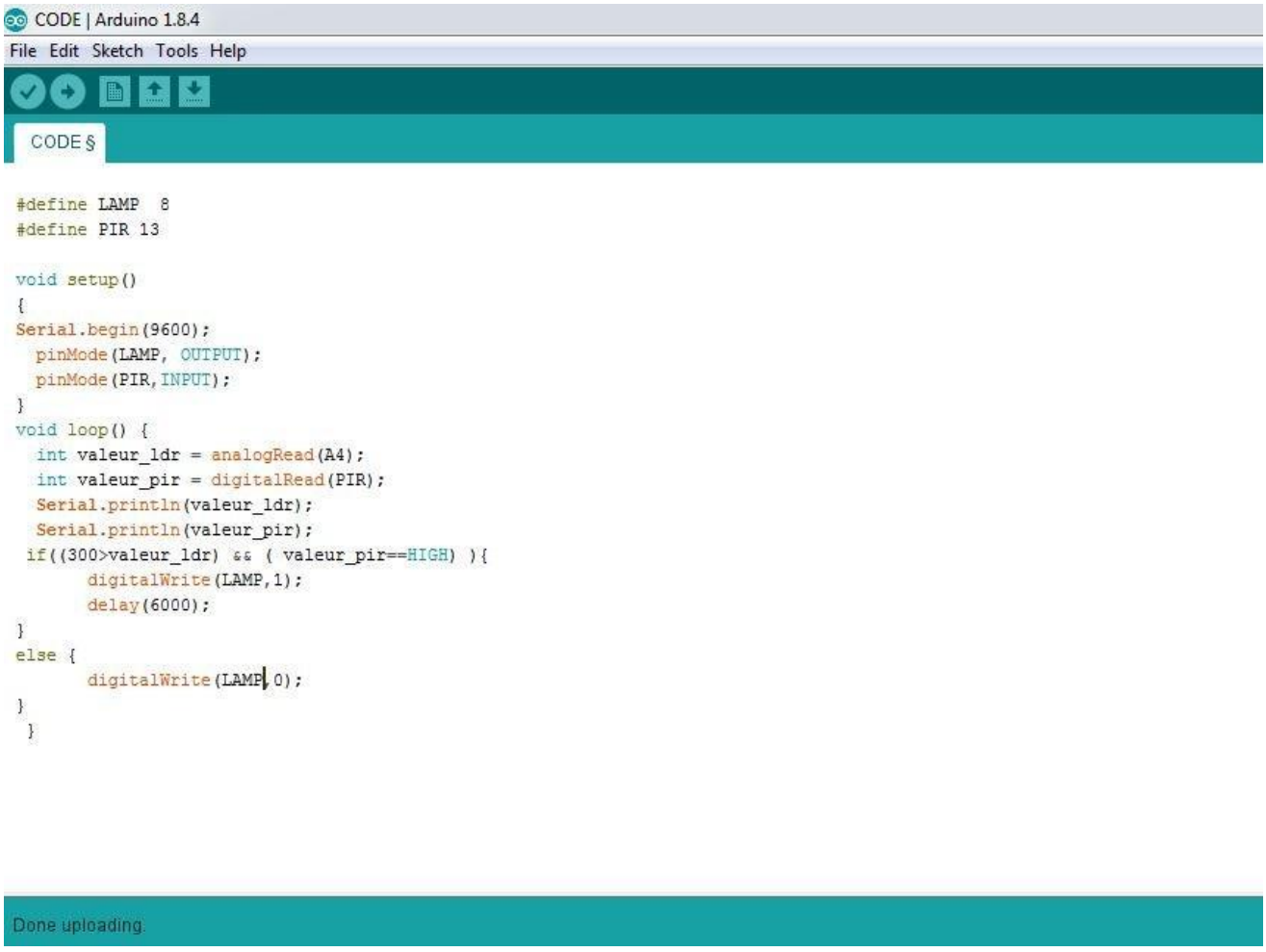
There are two ways to assembly the relay connection:

1. NC = Normally Closed Connection (Here we are using this one).

2. NO = Normally Open Connection.

So you have to wire the relays "OUT" to the 8th digital pin of the arduino uno.

Then the relay's "GND" to the arduino's "GND" and "VCC" to the "VCC".



Arudino Code

```
#define LAMP  8  // choose the pin for the RELAY
#define PIR 13   // choose the input pin (for PIR sensor)
void setup()
{
Serial.begin(9600);
pinMode(LAMP, OUTPUT); // declare lamp as output
pinMode(PIR,INPUT); // declare sensor as input
void loop()
{
  int value_ldr = analogRead(A4); // read LDR value
  int value_pir = digitalRead(PIR); // read input value
Serial.println(value_ldr);
Serial.println(value_pir);

 if((300>value_ldr) &&( value_pir==HIGH) ){
digitalWrite(LAMP,1);  // Turn ON the light
delay(6000);
}
else {
digitalWrite(LAMP,0); // Turn OFF the light
}
 }
```

Result : The above experiment is designed and executed successfully.

Experiment – 5

Aim: Upload humidity & temperature data to Thing Speak, periodically logging ambient light level to Thing Speak

Procedure

In this experiment, we are using the **DHT11 sensor for sending Temperature and Humidity data toThing speak using Arduino**. By this method, we can monitor our DHT11 sensor's temperature and humidity data over the internet using the ThingSpeak IoT server. And we can view the logged data and graph overtime on the Thingspeak website.

Here Arduino Uno reads the current temperature and humidity data from DHT11 and sends it to the ThingSpeak server for live monitoring from anywhere in the world. We previously used ThingSpeak with Raspberry Pi and ESP32 to upload the data on the cloud. **ThingSpeak** is an open data platform for monitoring your data online where you can set the data as private or public according to your choice. ThingSpeak takes a minimum of 15 seconds to update your readings. It's a great and very easy-to-use platform for building IoT projects.

Components Required

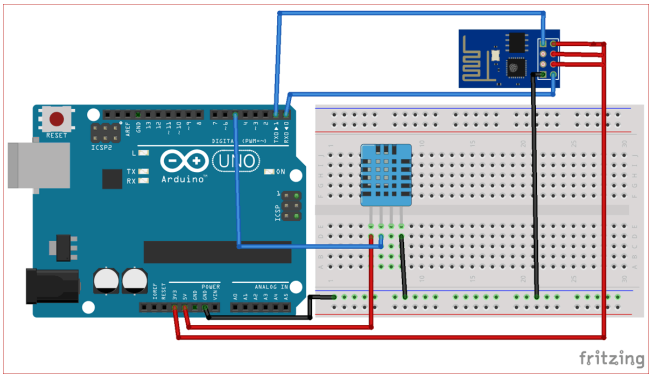
- Arduino Uno
- ESP8266 WiFi Module
- DHT11 Sensor
- Breadboard
- Jumper Wires

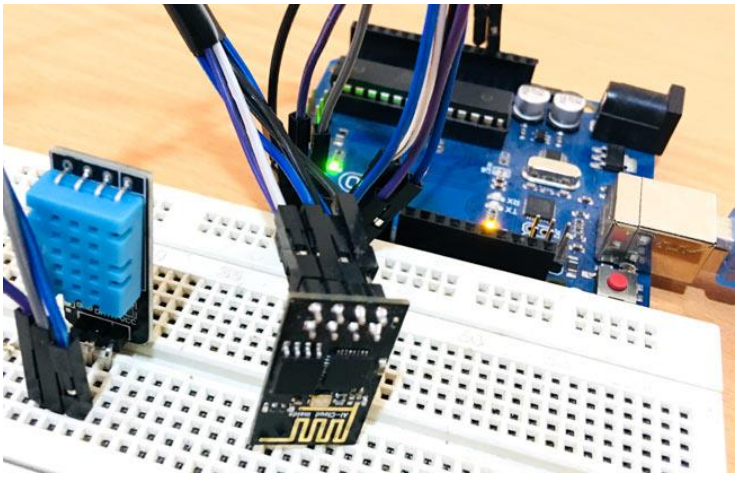
Software Required

Things Speak Account is needed i.e We have to create cloud server account.

Circuit Diagram and Connections

Temperature and Humidity Monitoring System Circuit Diagram





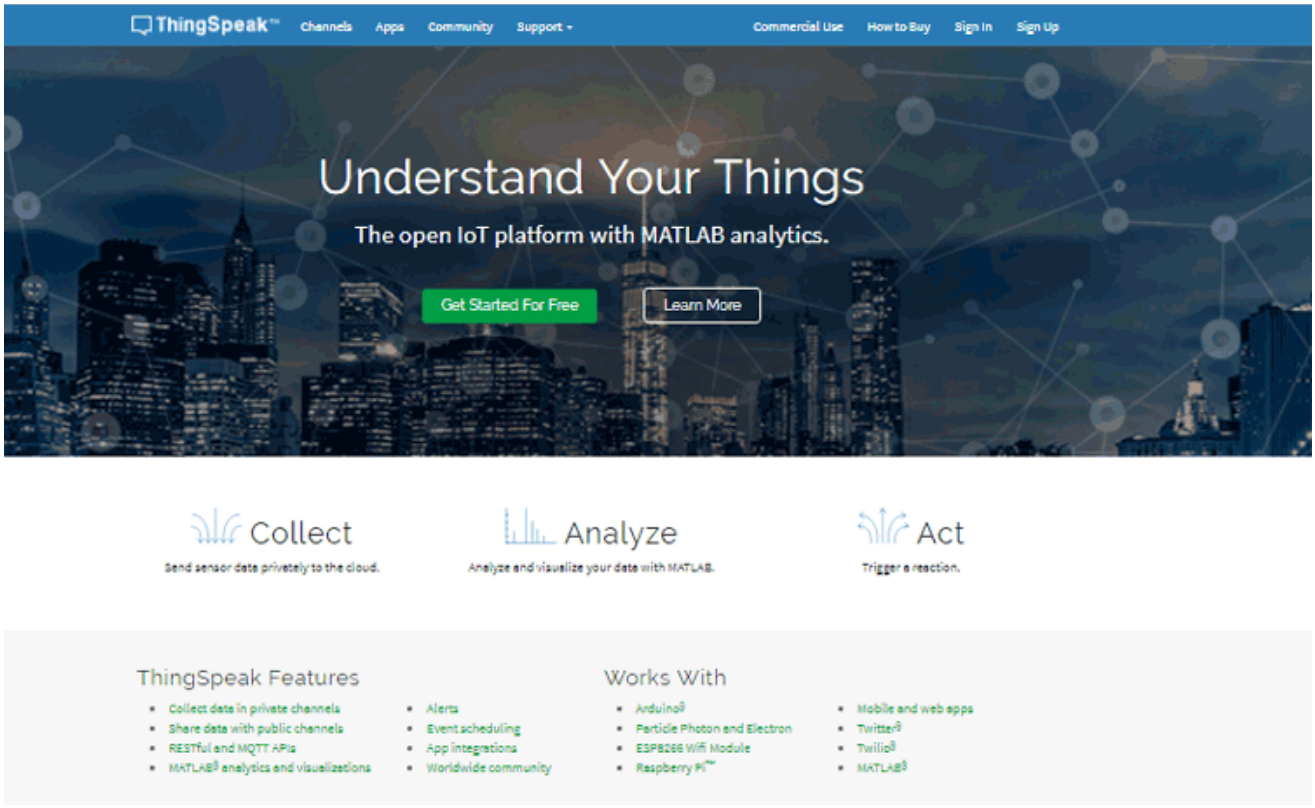
Connections are given in below table:

S.No.	Pin Name	Arduino Pin
1	ESP8266 VCC	3.3V
2	ESP8266 RST	3.3V
3	ESP8266 CH-PD	3.3V
4	ESP8266 RX	TX
5	ESP8266 TX	RX
6	ESP8266 GND	GND
7	DHT-11 VCC	5V
8	DHT-11 Data	5
9	DHT-11 GND	GND

Step 1: ThingSpeak Setup for Temperature and Humidity Monitoring

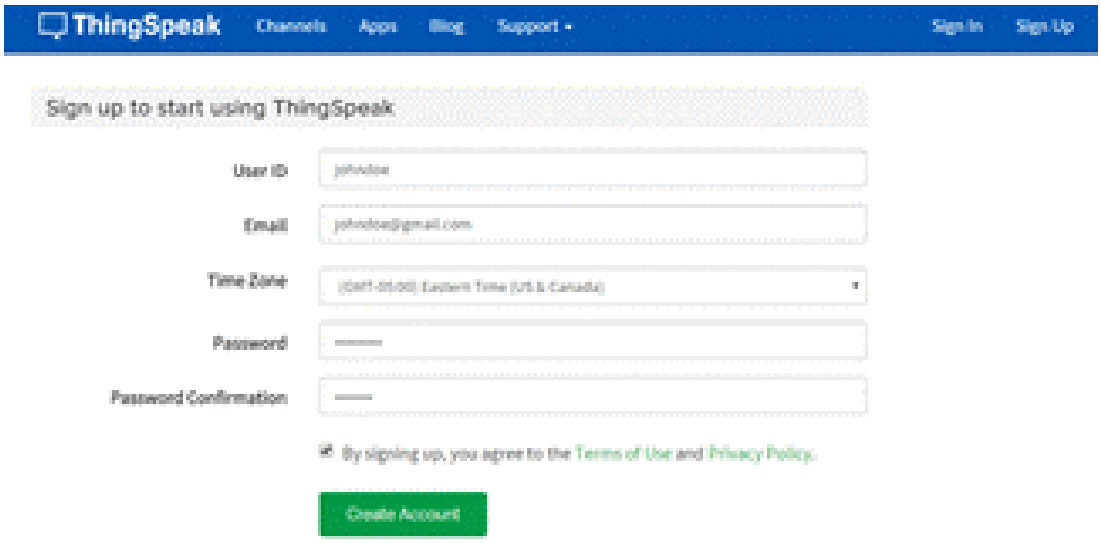
For creating your channel on Thingspeak, you first need to Sign up on Thingspeak. In case if you already have an account on Thingspeak, just sign in using your id and password.

For creating your account go to www.thinspeak.com



Click on Sing up if you don't have account and if you already have an account, then click on sign in.

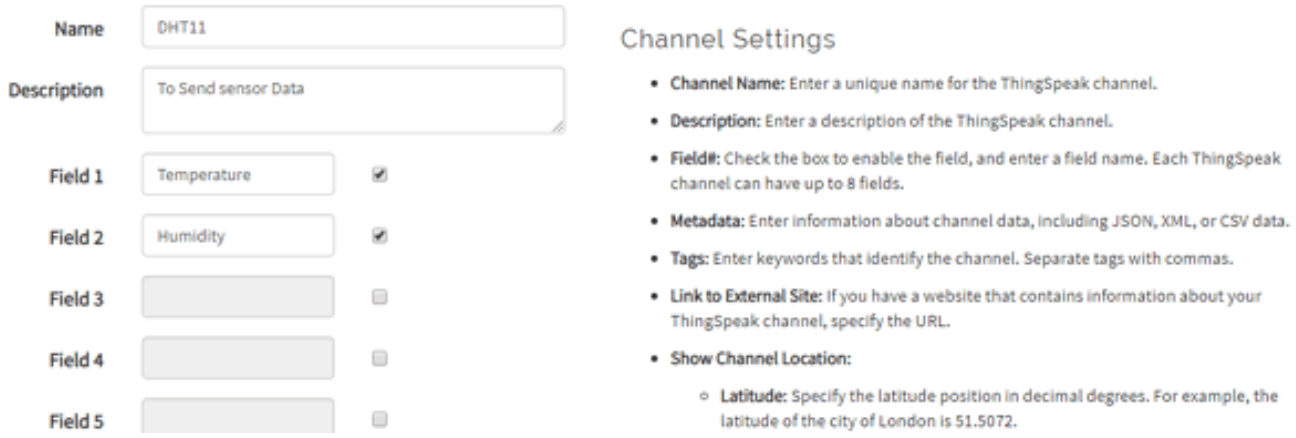
After clicking on signup, fill in your details.



After this, verify your E-mail id and click on continue.

Step 2: Create a Channel for Your Data

Once you Sign in after your account verification, Create a new channel by clicking "New Channel" button.



After clicking on "New Channel", enter the Name and Description of the data you want to upload on this channel. For example, I am sending my DHT11 sensor data, so I named it DHT11 data.

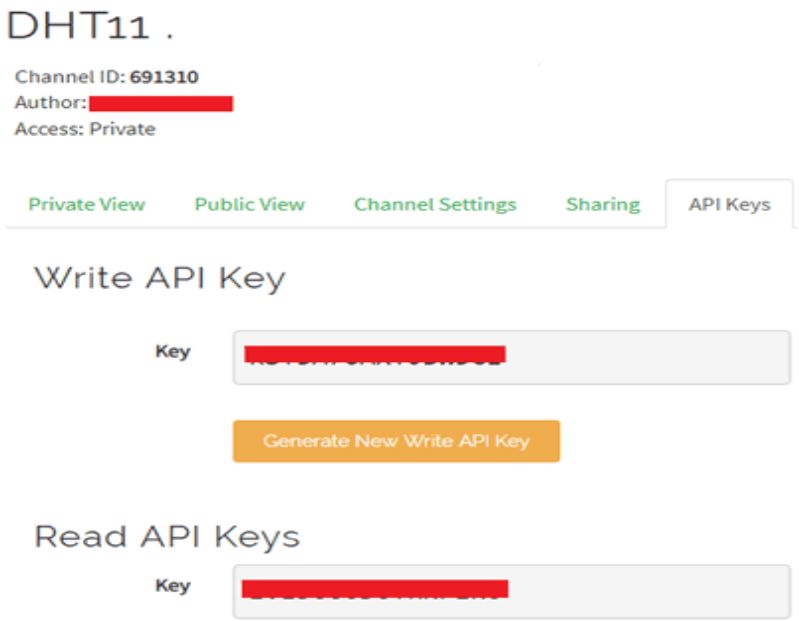
Enter the name of your data 'Temperature' in Field1 and 'Humidity' in Field2. If you want to use more Fields, you can check the box next to Field option and enter the name and description of your data.

After this, click on the save channel button to save your details.

Step 3: API Key

To send data to Thingspeak, we need a unique API key, which we will use later in our code to upload our sensor data to Thingspeak Website.

Click on "API Keys" button to get your unique API key for uploading your sensor data.



Now copy your "Write API Key".We will use this API key in our code.

Programming Arduino for Sending data to ThingSpeak

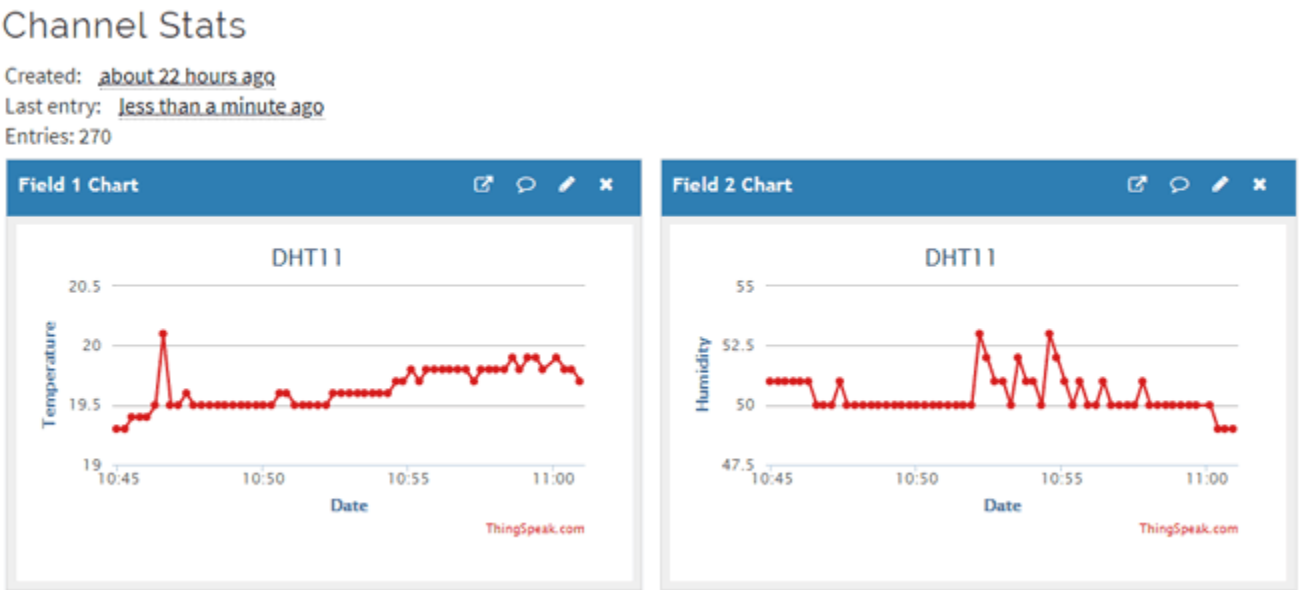
To program Arduino, open Arduino IDE and choose the correct board and port from the *'tool'* menu.

Output

Upload it in Arduino UNO. If you successfully upload your program, Serial monitor will look like this:

```
AT
AT+CWMODE=1
AT+CWJAP="CircuitLoop","circuitdigest101"
AT+CIPSTART="TCP","184.106.153.149",80
AT+CIPSEND=56
AT+CIPCLOSE
AT+CIPSTART="TCP","184.106.153.149",80
AT+CIPSEND=56
GET /update?key=9B6ILVOYMUSVOADA&field1=19.7&field2=50
AT+CIPSTART="TCP","184.106.153.149",80
AT+CIPSEND=56
GET /update?key=9B6ILVOYMUSVOADA&field1=19.7&field2=50
AT+CIPSTART="TCP","184.106.153.149",80
AT+CIPSEND=56
GET /update?key=9B6ILVOYMUSVOADA&field1=19.7&field2=50
AT+CIPSTART="TCP","184.106.153.149",80
AT+CIPSEND=56
GET /update?key=9B6ILVOYMUSVOADA&field1=19.7&field2=50
AT+CIPSTART="TCP","184.106.153.149",80
AT+CIPSEND=56
GET /update?key=9B6ILVOYMUSVOADA&field1=19.8&field2=50
AT+CIPSTART="TCP","184.106.153.149",80
AT+CIPSEND=56
```

After this navigate to your Thingspeak page and open your channel at Thingspeak and output will be shown as below:



Hence, we have successfully monitored Temperature and Humidity data over ThingSpeak using Arudino and ESP32.

Arduino Code

```
#include <stdlib.h>
#include <DHT.h>
#define DHTPIN 5         // DHT data pin connected to Arduino pin 5
#define DHTTYPE DHT11     // DHT11 (DHT Sensor Type )
DHT dht(DHTPIN, DHTTYPE); // Initialize the DHT sensor
#define SSID "WiFi Name"     // "WiFi Name"
#define PASS "WiFi Password"       // "Password"
#define IP "184.106.153.149"// thingspeak.com ip
String msg = "GET /update?key=Your API Key"; //change it with your key...
float temp;
int hum;
String tempC;
int error;
void setup()
{
  Serial.begin(115200); // use default 115200.
  Serial.println("AT");
  delay(5000);
  if(Serial.find("OK")){
    connectWiFi();
  }
}
void loop(){
   start:
  error=0;
  temp = dht.readTemperature();
  hum = dht.readHumidity();
  char buffer[10];
  tempC = dtostrf(temp, 4, 1, buffer);
  updateTemp();
  if (error==1){
    goto start;
  }

  delay(5000);
}
void updateTemp(){
  String cmd = "AT+CIPSTART=\"TCP\",\"";
  cmd += IP;
  cmd += "\",80";
  Serial.println(cmd);
  delay(2000);
  if(Serial.find("Error")){
    return;
  }
```

```
  cmd = msg ;
  cmd += "&field1=";
  cmd += tempC;
  cmd += "&field2=";
  cmd += String(hum);
  cmd += "\r\n";
  Serial.print("AT+CIPSEND=");
  Serial.println(cmd.length());
  if(Serial.find(">")){
    Serial.print(cmd);
  }
  else{
    Serial.println("AT+CIPCLOSE");
    //Resend...
    error=1;
  }
}
  booleanconnectWiFi(){
  Serial.println("AT+CWMODE=1");
  delay(2000);
  String cmd="AT+CWJAP=\"";
  cmd+=SSID;
  cmd+="\",\"";
  cmd+=PASS;
  cmd+="\"";
  Serial.println(cmd);
  delay(5000);
  if(Serial.find("OK")){
    return true;
  }else{
    return false;
  }
}
```

Result: The above experiment is designed and executed successfully.

Experiment – 6

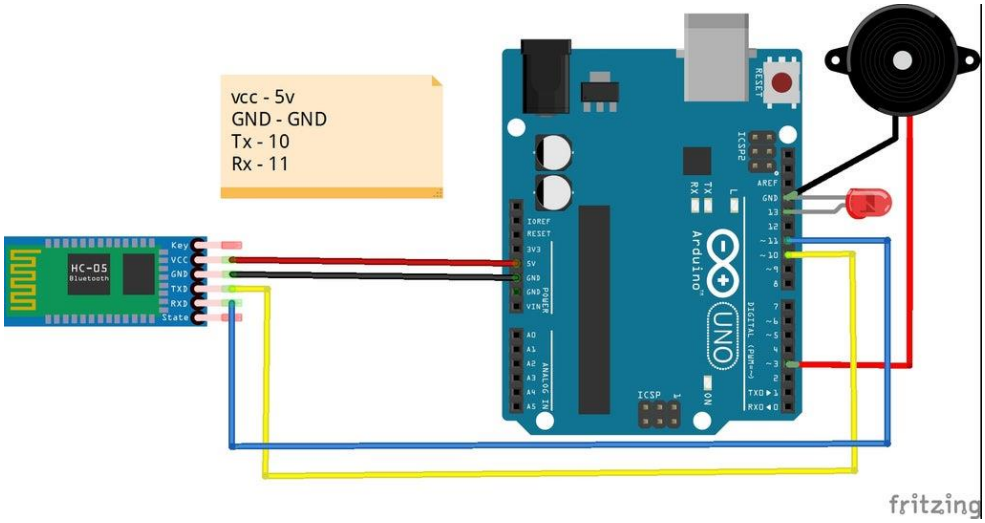
Aim: Controlling LEDs, relay & buzzer using Blynk app

Procedure

Step -1: Components Required

1. Arduino Uno X 1
2. HC-05 Bluetooth module X1
3. Jumper Wires X 6-10
4. Buzzer X 1
5. Led X 1

Step – 2 :Circuit Diagrami.e Setup



HC 05 Bluetooth module...

Module Ardduino

Vcc 5v

GND GND

Tx Digital pin 10.

Rx Digital pin 11

- Buzzer

module Arduino

+ve Digital pin 3

-veGnd

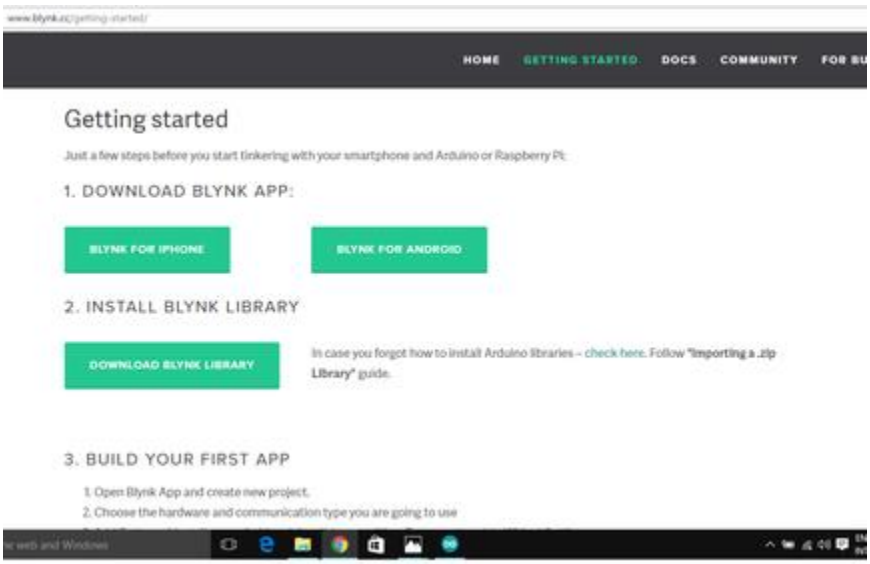
- LED

module Arduino

+ve Digital pin 13

-veGnd

Step – 3 Blink App



1. Go to Play Store from your android phone and download and install blynk app.
2. Open Blynk app.
3. Sign up/Register.
4. Now Click on Creat new project
5. Poject name - "Give Your project a name" (For e.g. i call it "BlueBuzz")
6. Choose device - "Arduino UNO"
7. Connection Type - "Bluetooth"

8. Now click "Create"
9. As soon as you click "Create", a mail called "dispatcher" is sent to you by blink.
10. Open mail and copy the "Auth token".
11. Now visit blynk website.
12. Now there you will find "Download blynk library". Now click on it follow the instructions and download blynk library.
13. Now extract the downloaded library and copy it to C:\Program Files (x86)\Arduino\libraries. (In the drive where you have installed arduino software, there you will find a folder called "libraries" not "lib" paste the library into "Library" folder.)
14. Now again visit blynk website.
15. Under "Flash" you will find "Sketch Builder", click on "Sketch Builder", On left side you will find...

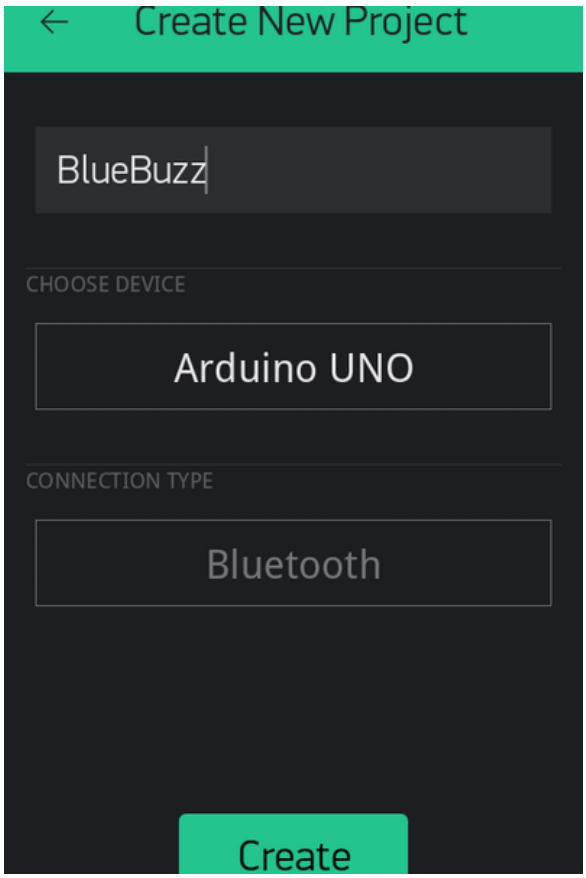
    -Board = Arduino

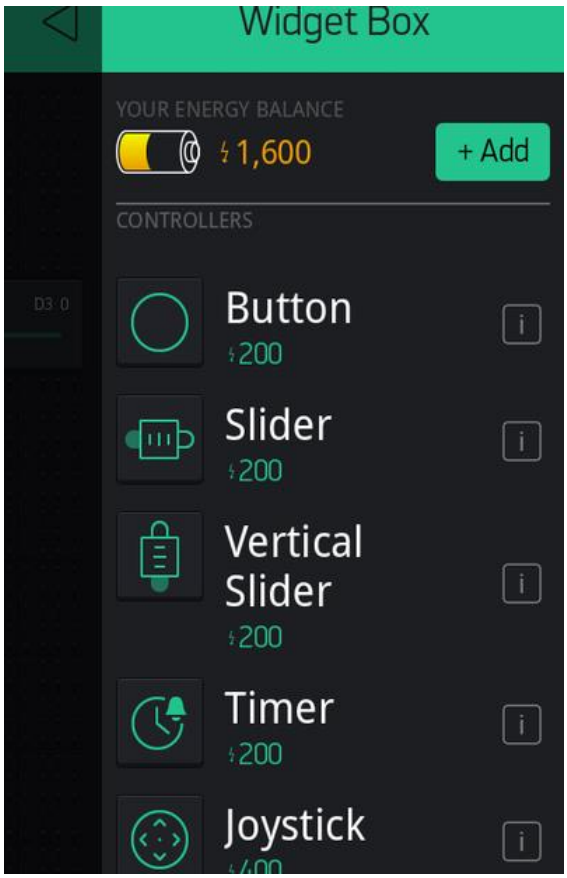
    -Connection = HC05/HC06

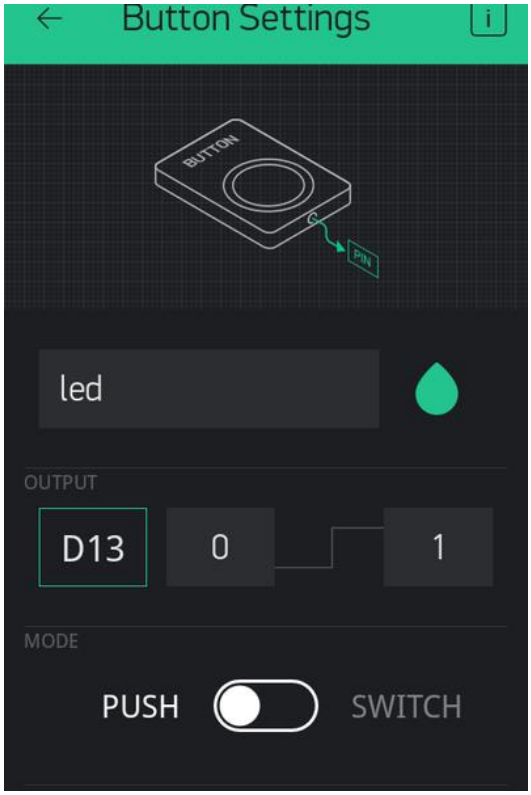
    -Example = GettingStarted/BlynkBlink

16. Now copy example and paste it to the arduino IDE.
17. Now paste the "Auth Token" (mailed by blynk) in the place of "Your Auth" and remove jumpers from pin 10 and 11 from arduino and upload the code to board.

Step 4: Creating Interface in App







- Click on "add widget" (+) and select a button.
- now click on button.
- Give name to button say "led".
- under OUTPUT tab...

- click pin and select the pin to which led is connected arduino, here it is digital pin 13, hence select digital and under pin D13. And Click continue.

- under MODE tab...

- select whether you want this button as "push button" or "Switch".
  - lick back.
  - Click on "add widget" (+) and select "Slider".
  - Click on "Slider".
- Name the slider say "buzzer"
- Under OUTPUT tab...

Select the pin no to which buzzer is connected to arduino, here it is digital pin D3. Click "continue".

- under SEND ON RELEASE tab...

set it to OFF

- click back.
  - Click on "add widget" (+) and select "Bluetooth".
  - Now close the app.
  - Now power your Arduino (you should see red light blinking on bluetooth module and make sure you have reconnected jumpers to pin 10 and 11)
  - Turn on bluetooth of your phone and search for "HC-05", now pair the device with default key "1234".
  - After successful pairing. Open Blynk app, select the project you have created select bluetooth.
  - tap on connect "bluetooth device" here you should find "HC 05" select it.
  - now you should see #HC-05 connected. and now hit back.
  - Now in rt most corner you should see "play" button adjacent to "Add widget", hit "play"
  - Now press led it should turn on led and move the slider accordingly buzzer should sound.

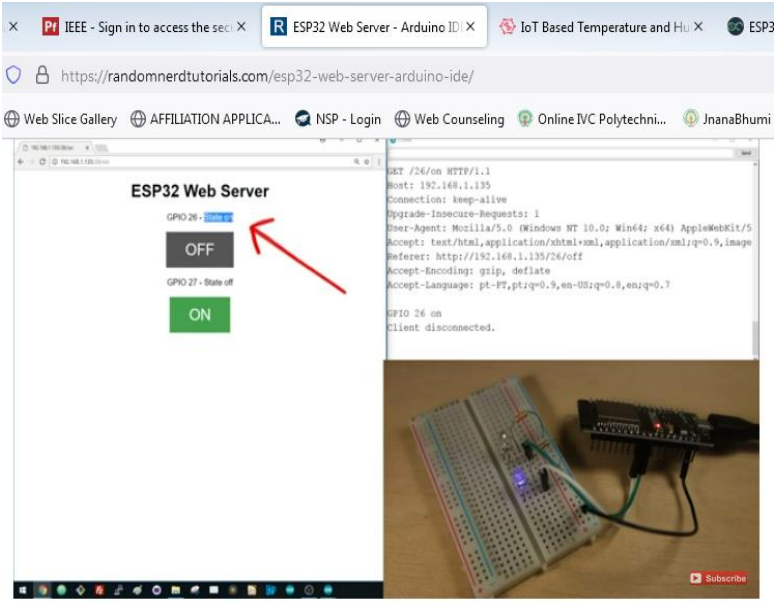
Experiment - 7

Aim: Introduction to HTTP. Hosting a basic server from the ESP32 to control various digital basedactuators (led, buzzer, relay) from a simple web page.

Procedure

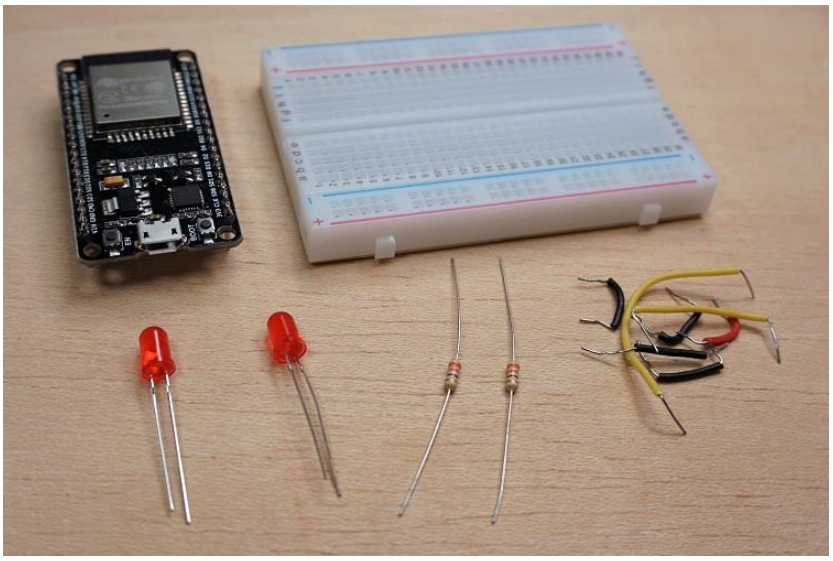
ESP32 Web Server – Arduino IDE

In this experiement you'll create a standalone web server with an ESP32 that controls outputs (two LEDs) using the Arduino IDE programming environment. The web server is mobile responsive and can be accessed with any device that as a browser on the local network. We'll show you how to create the web server and how the code works step-by-step.



- The web server you'll build controls two LEDs connected to the ESP32 GPIO 26 and GPIO 27;
- You can access the ESP32 web server by typing the ESP32 IP address on a browser in the local network;
- By clicking the buttons on your web server you can instantly change the state of each LED.

Parts Required

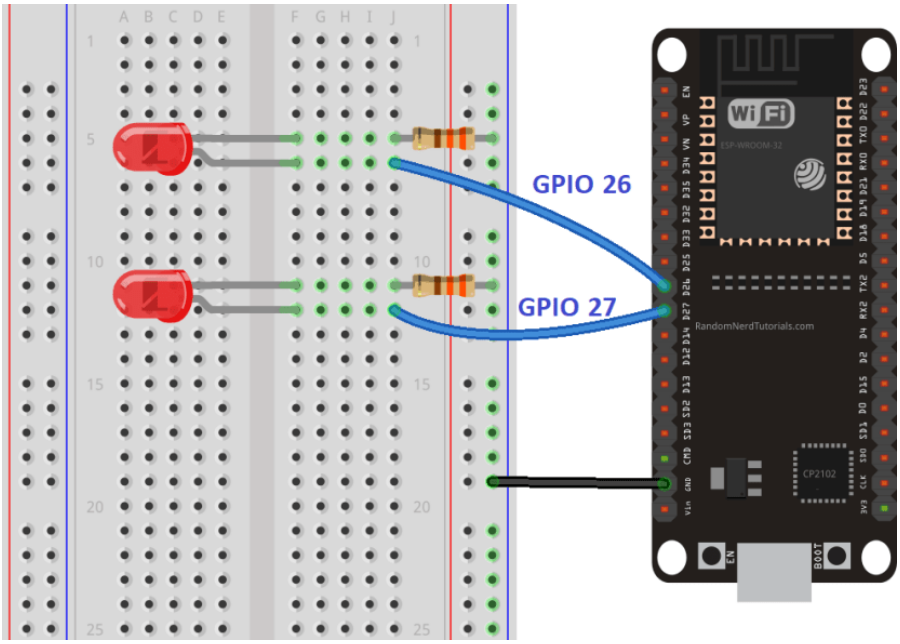


- ESP32 development board – read ESP32 Development Boards Review and Comparison
- 2x 5mm LED
- 2x 330 Ohm resistor
- Breadboard
- Jumper wires

Schematic

Start by building the circuit. Connect two LEDs to the ESP32 as shown in the following schematic diagram – one LED connected to GPIO 26, and the other to GPIO 27.

We're using the ESP32 DEVKIT DOIT board with 36 pins. Before assembling the circuit, make sure you check the pinout for the board you're using.



ESP32 Web Server Code

Copy the following code to your Arduino IDE, but don't upload it yet.

```
// Load Wi-Fi library

#include <WiFi.h>

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Set web server port number to 80
WiFiServerserver(80);

// Variable to store the HTTP request
String header;

// Auxiliar variables to store the current output state
String output26State = "off";
String output27State = "off";

// Assign output variables to GPIO pins
const int output26 = 26;
const int output27 = 27;

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;
```

```
void setup() {
Serial.begin(115200);
  // Initialize the output variables as outputs
pinMode(output26, OUTPUT);
pinMode(output27, OUTPUT);
  // Set outputs to LOW
digitalWrite(output26, LOW);
digitalWrite(output27, LOW);

  // Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
  }
  // Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();
}

void loop(){
WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                             // If a new client connects,
currentTime = millis();
previousTime = currentTime;
Serial.println("New Client.");          // print a message out in the serial port
    String currentLine = "";                // make a String to hold incoming data from the
client
    while (client.connected() &&currentTime - previousTime<= timeoutTime) {  // loop
while the client's connected
currentTime = millis();
      if (client.available()) {             // if there's bytes to read from the client,
        char c = client.read();             // read a byte, then
Serial.write(c);                    // print it out the serial monitor
        header += c;
        if (c == '\n') {                    // if the byte is a newline character
          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
            // and a content-type so the client knows what's coming, then a blank line:
client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
```

```
client.println("Connection: close");
client.println();

            // turns the GPIOs on and off
            if (header.indexOf("GET /26/on") >= 0) {
Serial.println("GPIO 26 on");
              output26State = "on";
digitalWrite(output26, HIGH);
            } else if (header.indexOf("GET /26/off") >= 0) {
Serial.println("GPIO 26 off");
              output26State = "off";
digitalWrite(output26, LOW);
            } else if (header.indexOf("GET /27/on") >= 0) {
Serial.println("GPIO 27 on");
              output27State = "on";
digitalWrite(output27, HIGH);
            } else if (header.indexOf("GET /27/off") >= 0) {
Serial.println("GPIO 27 off");
              output27State = "off";
digitalWrite(output27, LOW);
            }

            // Display the HTML web page
client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:,\">");
            // CSS to style the on/off buttons
            // Feel free to change the background-color and font-size attributes to fit
your preferences
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px
auto; text-align: center;}");
client.println(".button { background-color: #4CAF50; border: none; color: white;
padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
client.println(".button2 {background-color: #555555;}</style></head>");

            // Web Page Heading
client.println("<body><h1>ESP32 Web Server</h1>");

            // Display current state, and ON/OFF buttons for GPIO 26
client.println("<p>GPIO 26 - State " + output26State + "</p>");
            // If the output26State is off, it displays the ON button
            if (output26State=="off") {
client.println("<p><ahref=\"/26/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
client.println("<p><a href=\"/26/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }
```

```
            // Display current state, and ON/OFF buttons for GPIO 27
client.println("<p>GPIO 27 - State " + output27State + "</p>");
            // If the output27State is off, it displays the ON button
            if (output27State=="off") {
client.println("<p><a href=\"/27/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
client.println("<p><a href=\"/27/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }
client.println("</body></html>");

            // The HTTP response ends with another blank line
client.println();
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage return character,
currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
  }
}
```

Setting Your Network Credentials

You need to modify the following lines with your network credentials: SSID and password. The code is well commented on where you should make the changes.

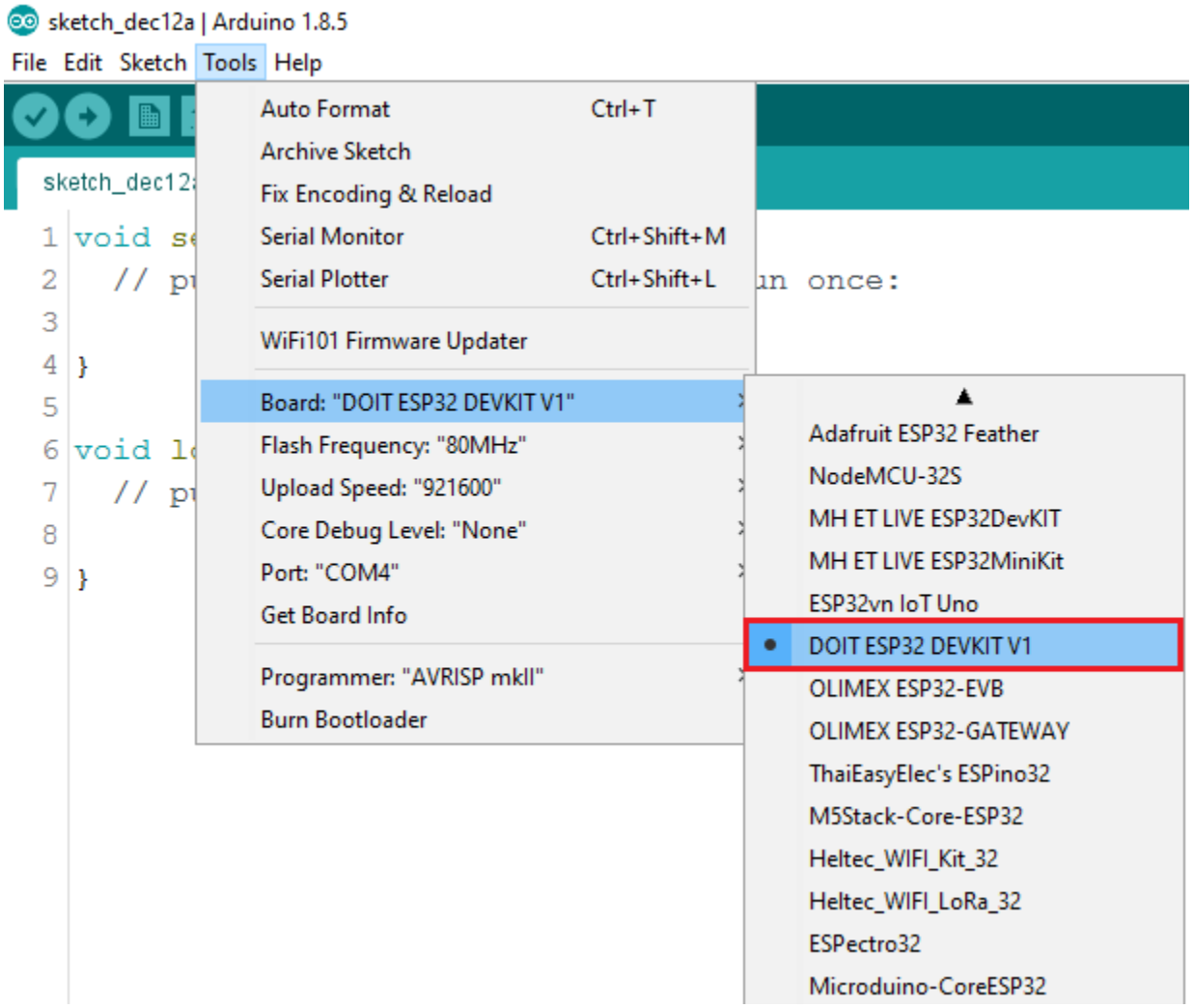
```
// Replace with your network credentials
constchar*ssid="REPLACE_WITH_YOUR_SSID";
constchar* password ="REPLACE_WITH_YOUR_PASSWORD";
```

Uploading the Code

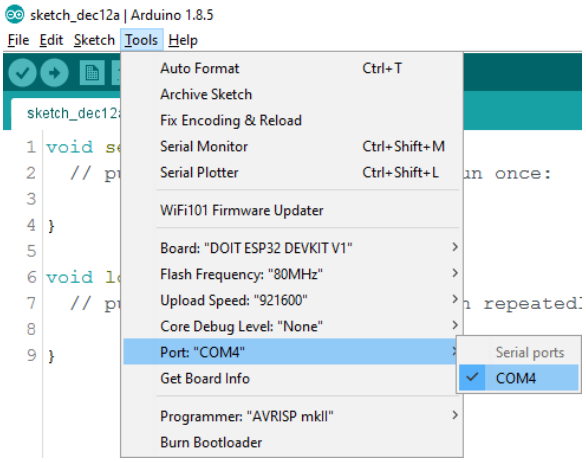
Now, you can upload the code and and the web server will work straight away. Follow the next steps to upload code to the ESP32:

1) Plug your ESP32 board in your computer;

2) In the Arduino IDE select your board in **Tools**>**Board** (in our case we're using the ESP32 DEVKIT DOIT board);

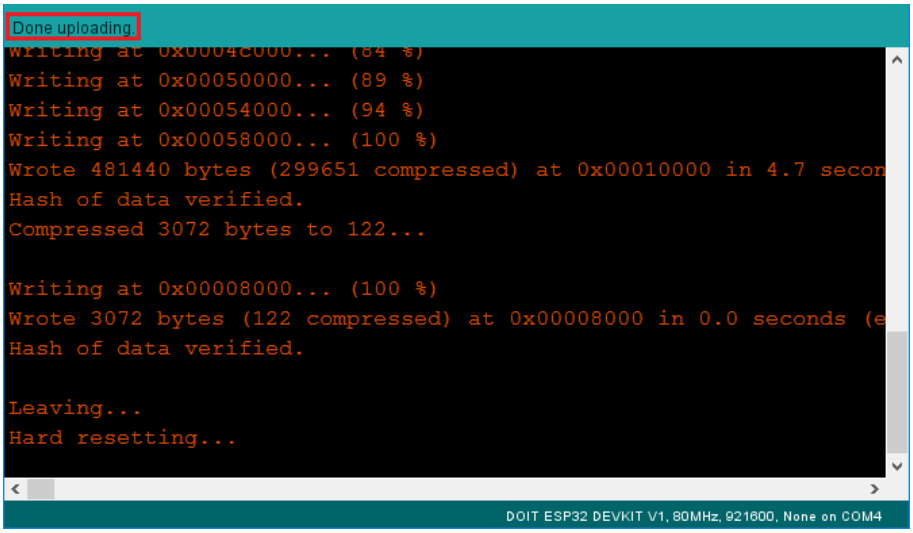


3) Select the COM port in **Tools**> **Port**.



4) Press the **Upload** button in the Arduino IDE and wait a few seconds while the code compiles and uploads to your board.

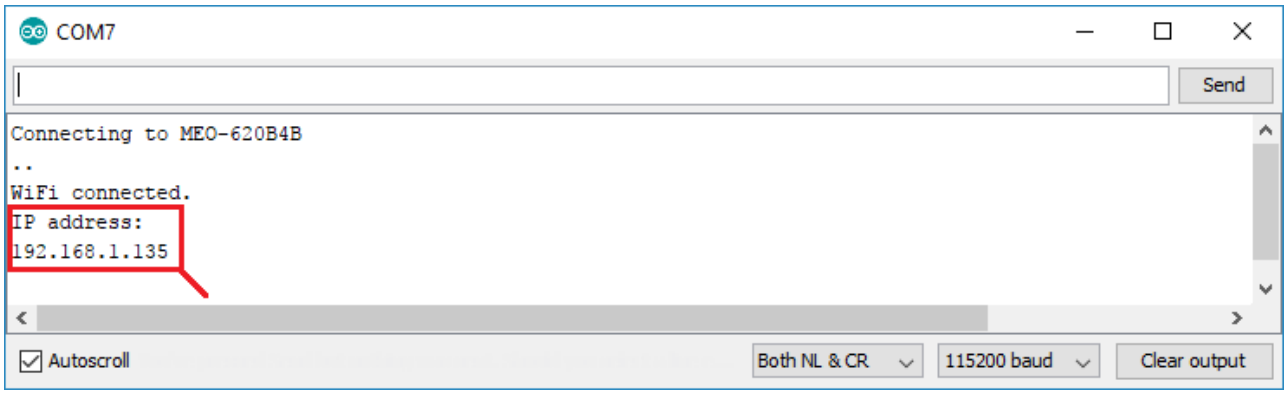
5) Wait for the "**Done uploading**" message.



Finding the ESP IP Address

After uploading the code, open the Serial Monitor at a baud rate of 115200.

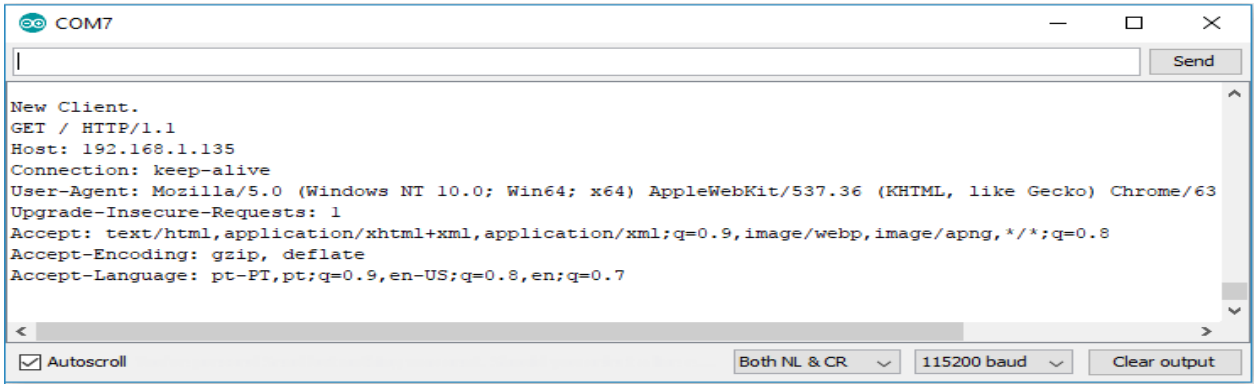
Press the ESP32 EN button (reset). The ESP32 connects to Wi-Fi, and outputs the ESP IP address on the Serial Monitor. Copy that IP address, because you need it to access the ESP32 web server.



Accessing the Web Server

To access the web server, open your browser, paste the ESP32 IP address, and you'll see the following page. In our case it is **192.168.1.135**.

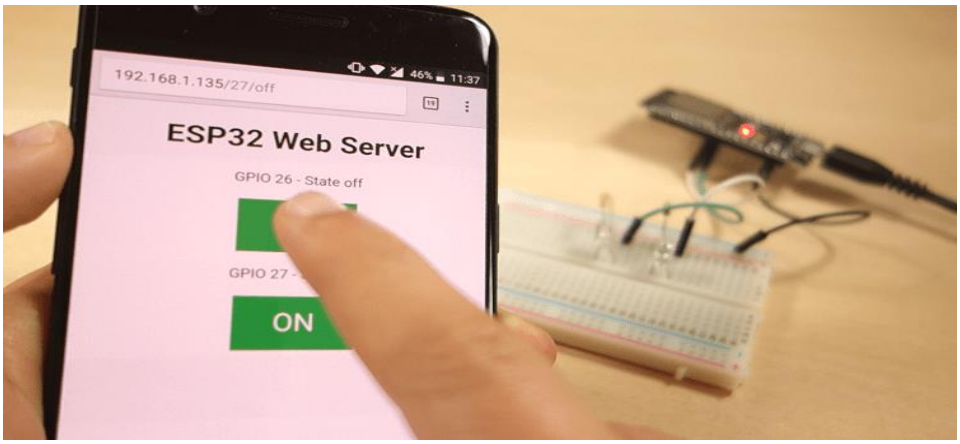
If you take a look at the Serial Monitor, you can see what's happening on the background. The ESP receives an HTTP request from a new client (in this case, your browser).



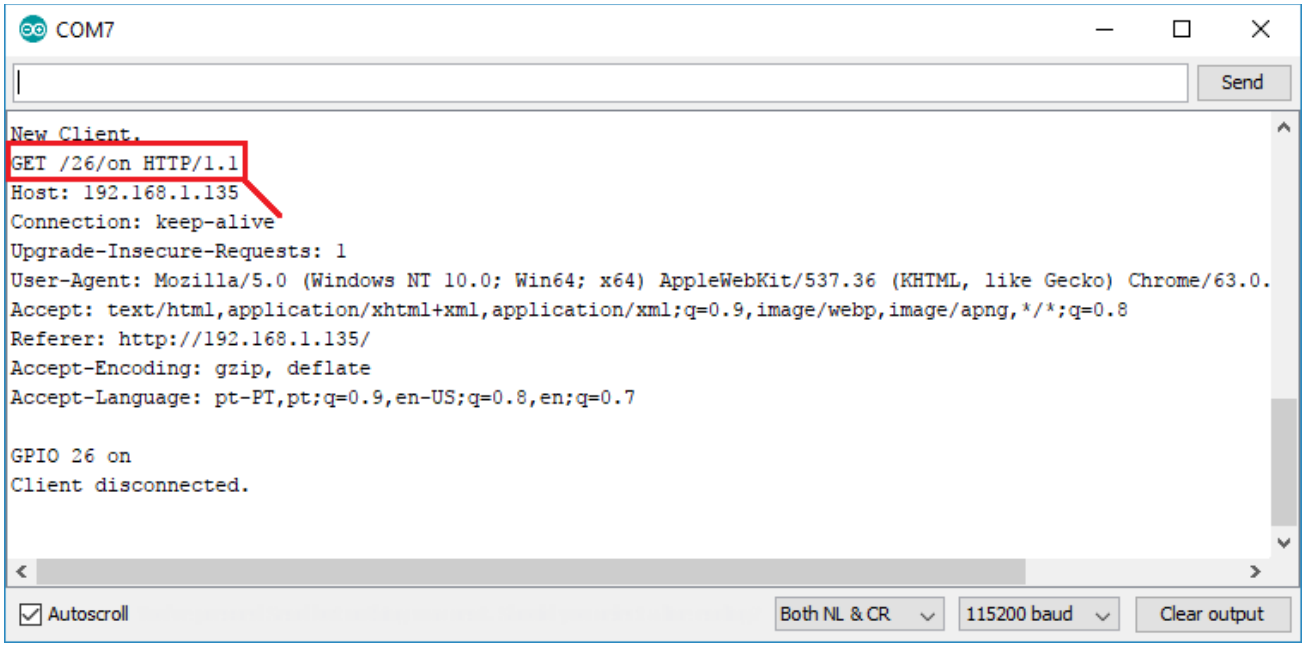
You can also see other information about the HTTP request.

Testing the Web Server

Now you can test if your web server is working properly. Click the buttons to control the LEDs.



At the same time, you can take a look at the Serial Monitor to see what's going on in the background. For example, when you click the button to turn GPIO 26 ON, ESP32 receives a request on the **/26/on** URL.



When the ESP32 receives that request, it turns the LED attached to GPIO 26 ON and updates its state on the web page.

The button for GPIO 27 works in a similar way. Test that it is working properly.

How the Code Works

In this section will take a closer look at the code to see how it works.

The first thing you need to do is to include the WiFi library.

```
#include <WiFi.h>
```

As mentioned previously, you need to insert your ssid and password in the following lines inside the double quotes.

```
constchar*ssid="";
constchar* password ="";
```

Then, you set your web server to port 80.

```
WiFiServerserver(80);
```

The following line creates a variable to store the header of the HTTP request:

```
String header;
```

Next, you create auxiliar variables to store the current state of your outputs. If you want to add more outputs and save its state, you need to create more variables.

```
String output26State ="off";
String output27State ="off";
```

You also need to assign a GPIO to each of your outputs. Here we are using GPIO 26 and GPIO 27. You can use any other suitable GPIOs.

```
constint output26 =26;
constint output27 =27;
```

setup()

Now, let's go into the setup(). First, we start a serial communication at a baud rate of 115200 for debugging purposes.

```
Serial.begin(115200);
```

You also define your GPIOs as OUTPUTs and set them to LOW.

```
// Initialize the output variables as outputs
pinMode(output26, OUTPUT);
pinMode(output27, OUTPUT);

// Set outputs to LOW
digitalWrite(output26, LOW);
digitalWrite(output27, LOW);
```

The following lines begin the Wi-Fi connection with WiFi.begin(ssid, password), wait for a successful connection and print the ESP IP address in the Serial Monitor.

```
// Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while(WiFi.status()!= WL_CONNECTED){
delay(500);
Serial.print(".");
```

```
}
// Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();
```

loop()

In the loop() we program what happens when a new client establishes a connection with the web server.

The ESP32 is always listening for incoming clients with the following line:

```
WiFiClient client =server.available();// Listen for incoming clients
```

When a request is received from a client, we'll save the incoming data. The while loop that follows will be running as long as the client stays connected. We don't recommend changing the following part of the code unless you know exactly what you are doing.

```
if(client){// If a new client connects,
Serial.println("New Client.");// print a message out in the serial port
  String currentLine="";// make a String to hold incoming data from the client
while(client.connected()){// loop while the client's connected
if(client.available()){// if there's bytes to read from the client,
char c =client.read();// read a byte, then
Serial.write(c);// print it out the serial monitor
      header += c;
if(c =='\n'){// if the byte is a newline character
// if the current line is blank, you got two newline characters in a row.
/ that's the end of the client HTTP request, so send a response:
if(currentLine.length()==0){
// HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
// and a content-type so the client knows what's coming, then a blank line:
client.println("HTTP/1.1 200 OK");
client.println("Content-type:text/html");
client.println("Connection: close");
client.println();
```

The next section of if and else statements checks which button was pressed in your web page, and controls the outputs accordingly. As we've seen previously, we make a request on different URLs depending on the button pressed.

```
// turns the GPIOs on and off
if(header.indexOf("GET /26/on")>=0){
Serial.println("GPIO 26 on");
  output26State ="on";
digitalWrite(output26, HIGH);
}elseif(header.indexOf("GET /26/off")>=0){
Serial.println("GPIO 26 off");
  output26State ="off";
digitalWrite(output26, LOW);
}elseif(header.indexOf("GET /27/on")>=0){
Serial.println("GPIO 27 on");
  output27State ="on";
digitalWrite(output27, HIGH);
}elseif(header.indexOf("GET /27/off")>=0){
Serial.println("GPIO 27 off");
```

```
  output27State ="off";
digitalWrite(output27, LOW);
}
```

For example, if you've press the GPIO 26 ON button, the ESP32 receives a request on the **/26/ON URL** (we can see that that information on the HTTP header on the Serial Monitor). So, we can check if the header contains the expression **GET /26/on**. If it contains, we change the output26state variable to ON, and the ESP32 turns the LED on.

This works similarly for the other buttons. So, if you want to add more outputs, you should modify this part of the code to include them.

Displaying the HTML web page

The next thing you need to do, is creating the web page. The ESP32 will be sending a response to your browser with some HTML code to build the web page.

The web page is sent to the client using this expressing client.println(). You should enter what you want to send to the client as an argument.

The first thing we should send is always the following line, that indicates that we are sending HTML.

```
<!DOCTYPE HTML><html>
```

Then, the following line makes the web page responsive in any web browser.

```
client.println("<head><meta name=\"viewport\" content=\"width=device-width,
initial-scale=1\">");
```

And the following is used to prevent requests on the favicon. – You don't need to worry about this line.

```
client.println("<link rel=\"icon\" href=\"data:,\">");
```

Styling the Web Page

Next, we have some CSS text to style the buttons and the web page appearance. We choose the Helvetica font, define the content to be displayed as a block and aligned at the center.

```
client.println("<style>html { font-family: Helvetica; display: inline-block;
margin: 0px auto; text-align: center;}");
```

We style our buttons with the #4CAF50 color, without border, text in white color, and with this padding: 16px 40px. We also set the text-decoration to none, define the font size, the margin, and the cursor to a pointer.

```
client.println(".button { background-color: #4CAF50; border: none; color: white;
padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor:
pointer;}");
```

We also define the style for a second button, with all the properties of the button we've defined earlier, but with a different color. This will be the style for the off button.

```
client.println(".button2 {background-color: #555555;}</style></head>");
```

Setting the Web Page First Heading

In the next line you can set the first heading of your web page. Here we have "**ESP32 Web Server**", but you can change this text to whatever you like.

```
// Web Page Heading
client.println("<h1>ESP32 Web Server</h1>");
```

Displaying the Buttons and Corresponding State

Then, you write a paragraph to display the GPIO 26 current state. As you can see we use the output26State variable, so that the state updates instantly when this variable changes.

```
client.println("<p>GPIO 26 - State "+ output26State +"</p>");
```

Then, we display the on or the off button, depending on the current state of the GPIO. If the current state of the GPIO is off, we show the ON button, if not, we display the OFF button.

```
if(output26State=="off"){
client.println("<p><a href=\"/26/on\"><button
class=\"button\">ON</button></a></p>");
}else{
client.println("<p><a href=\"/26/off\"><button class=\"button
button2\">OFF</button></a></p>");
}
```

We use the same procedure for GPIO 27.

Closing the Connection

Finally, when the response ends, we clear the header variable, and stop the connection with the client with client.stop().

```
// Clear the header variable
header ="";
// Close the connection
client.stop();
```

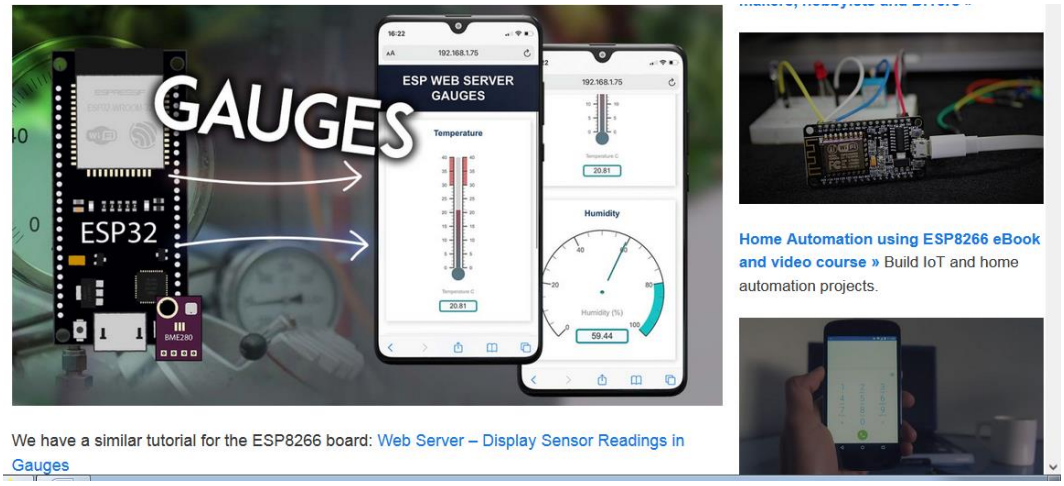
Result: The above experiment is designed and executed successfully.

Experiment – 8

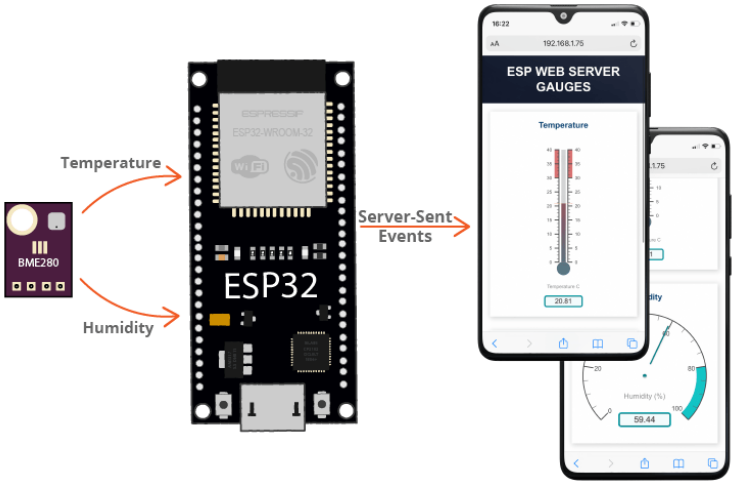
Aim: Displaying various sensor readings on a simple web page hosted on the ESP32.

Procedure

Build a web server with the ESP32 to display sensor readings in gauges. As an example, we'll display temperature and humidity from a BME280 sensor in two different gauges: linear and radial. You can easily modify the project to plot any other data. To build the gauges, we'll use the canvas-gauges JavaScript library.

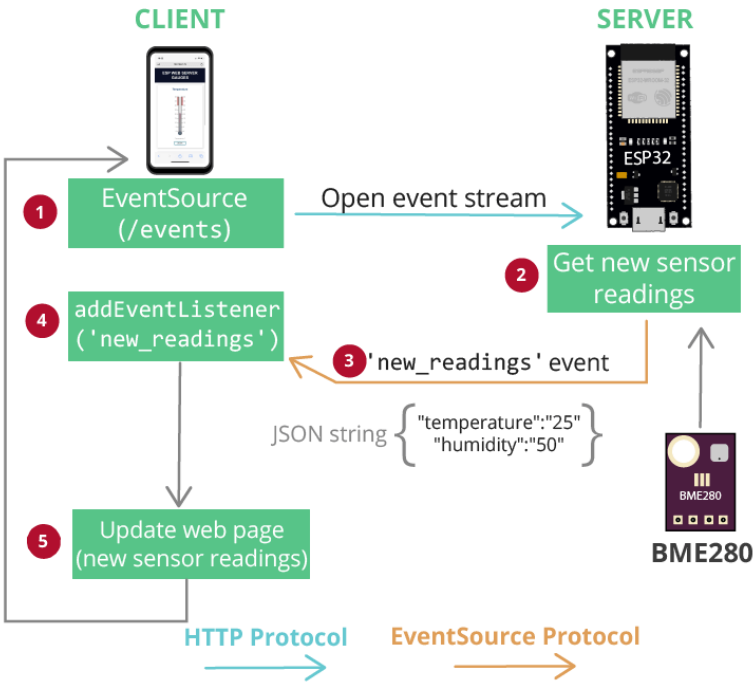


This experiment will build a web server with the ESP32 that displays temperature and humidity readings from a BME280 sensor. We'll create a linear gauge that looks like a thermometer to display the temperature, and a radial gauge to display the humidity.



Server-Sent Events

The readings are updated automatically on the web page using Server-Sent Events (SSE).



1. Install ESP32 Board in Arduino IDE

We'll program the ESP32 using Arduino IDE. So, you must have the ESP32 add-on installed. Follow the next tutorial if you haven't already:

- Installing ESP32 Board in Arduino IDE (Windows, Mac OS X, Linux)

If you want to use VS Code with the PlatformIO extension, follow the next tutorial instead to learn how to program the ESP32:

- Getting Started with VS Code and PlatformIO IDE for ESP32 and ESP8266 (Windows, Mac OS X, Linux Ubuntu)

2. Filesystem Uploader Plugin

To upload the HTML, CSS, and JavaScript files to the ESP32 flash memory (SPIFFS), we'll use a plugin for Arduino IDE: **SPIFFSFilesystem uploader**. Follow the next tutorial to install the filesystem uploader plugin:

- ESP32: Install SPIFFS FileSystem Uploader Plugin in Arduino IDE

If you're using VS Code with the PlatformIO extension, read the following tutorial to learn how to upload files to the filesystem:

- ESP32 with VS Code and PlatformIO: Upload Files to Filesystem (SPIFFS)

3. Installing Libraries

To build this experiment, you need to install the following libraries:

- Adafruit_BME280 (Arduino Library Manager)
- Adafruit_Sensor library (Arduino Library Manager)

- Arduino_JSON library by Arduino version 0.1.0 (Arduino Library Manager)
- ESPAsyncWebServer (.zip folder);
- AsyncTCP (.zip folder).

You can install the first three libraries using the Arduino Library Manager. Go to **Sketch** > **Include Library** > **Manage Libraries** and search for the libraries' names.

In your Arduino IDE, go to **Sketch** > **Include Library** > **Add .zip Library** and select the libraries you've just downloaded.

Installing Libraries (VS Code + PlatformIO)

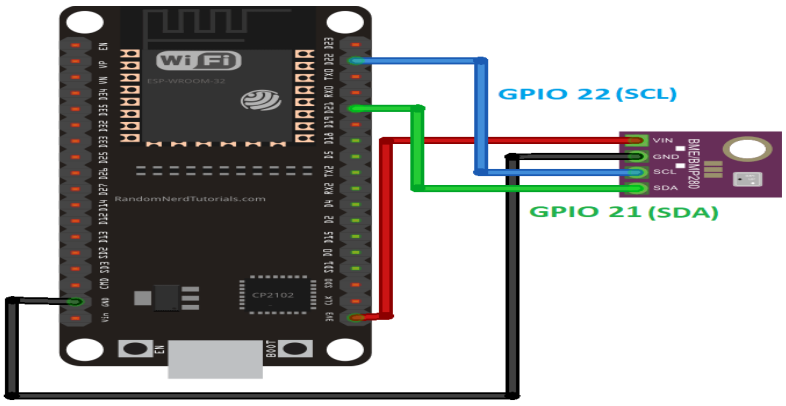
```
monitor_speed = 115200
lib_deps = ESP Async WebServer
arduino-libraries/Arduino_JSON @ 0.1.0
adafruit/Adafruit BME280 Library @ ^2.1.0
adafruit/Adafruit Unified Sensor @ ^1.1.4
```

Parts Required

- ESP32
- BME280 Sensor
- Jumper wires
- Breadboard

Schematic Diagram

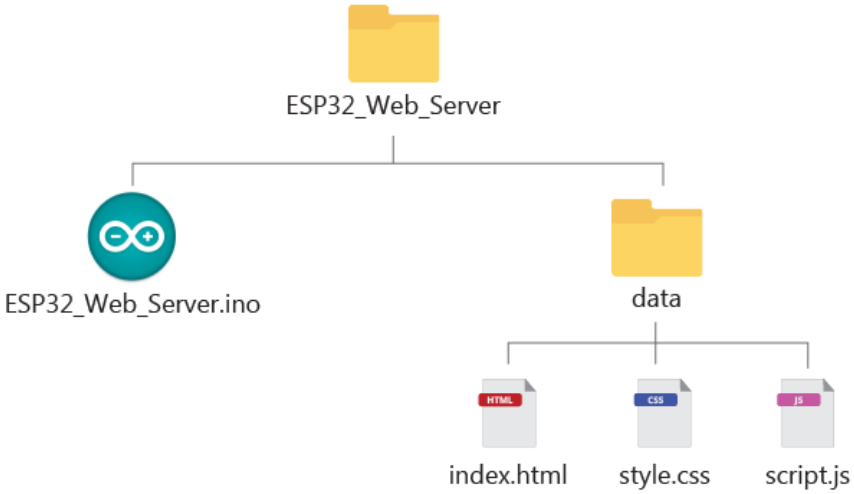
We'll send temperature and humidity readings from a BME280 sensor. We're going to use I2C communication with the BME280 sensor module. For that, wire the sensor to the default ESP32 SCL (GPIO 22) and SDA (GPIO 21) pins, as shown in the following schematic diagram.



Organizing Files:

- **Arduino sketch** that handles the web server;
- **index.html**: to define the content of the web page;
- **sytle.css**: to style the web page;

- **script.js**: to program the behavior of the web page—handle web server responses, events, create the gauges, etc.



Code

index.html file.

```
<!DOCTYPE html>
<html>
<head>
<title>ESP IOT DASHBOARD</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" type="image/png" href="favicon.png">
<link rel="stylesheet"
href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-
fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
crossorigin="anonymous">
<link rel="stylesheet" type="text/css" href="style.css">
<script src="http://cdn.rawgit.com/Mikhus/canvas-gauges/gh-
pages/download/2.1.7/all/gauge.min.js"></script>
</head>
<body>
<div class="topnav">
<h1>ESP WEB SERVER GAUGES</h1>
</div>
<div class="content">
<div class="card-grid">
<div class="card">
<p class="card-title">Temperature</p>
<canvas id="gauge-temperature"></canvas>
</div>
<div class="card">
<p class="card-title">Humidity</p>
```

```
<canvas id="gauge-humidity"></canvas>
</div>
</div>
</div>
<script src="script.js"></script>
</body>
</html>
```

CSS File

style.css file

```
html {
  font-family: Arial, Helvetica, sans-serif;
  display: inline-block;
  text-align: center;
}
h1 {
  font-size: 1.8rem;
  color: white;
}
p {
  font-size: 1.4rem;
}
.topnav {
  overflow: hidden;
  background-color: #0A1128;
}
body {
  margin: 0;
}
.content {
  padding: 5%;
}
.card-grid {
  max-width: 1200px;
  margin: 0 auto;
  display: grid;
  grid-gap: 2rem;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
}
.card {
  background-color: white;
  box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);
}
.card-title {
  font-size: 1.2rem;
```

```
            font-weight: bold;
            color: #034078
        }
```

JavaScript File (creating the gauges)

script.js file

```
    minValue: 0,
    startAngle: 90,
    ticksAngle: 180,
    maxValue: 40,
    colorValueBoxRect: "#049faa",
    colorValueBoxRectEnd: "#049faa",
    colorValueBoxBackground: "#f1fbfc",
    valueDec: 2,
    valueInt: 2,
    majorTicks: [
        "0",
        "5",
        "10",
        "15",
        "20",
        "25",
        "30",
        "35",
        "40"
     ],
    minorTicks: 4,
    strokeTicks: true,
     highlights: [
        {
            "from": 30,
            "to": 40,
            "color": "rgba(200, 50, 50, .75)"
        }
     ],
    colorPlate: "#fff",
    colorBarProgress: "#CC2936",
    colorBarProgressEnd: "#049faa",
    borderShadowWidth: 0,
     borders: false,
    needleType: "arrow",
    needleWidth: 2,
    needleCircleSize: 7,
    needleCircleOuter: true,
    needleCircleInner: false,
```

```
animationDuration: 1500,
animationRule: "linear",
barWidth: 10,
}).draw();

// Create Humidity Gauge
var gaugeHum = new RadialGauge({
renderTo: 'gauge-humidity',
  width: 300,
  height: 300,
  units: "Humidity (%)",
minValue: 0,
maxValue: 100,
colorValueBoxRect: "#049faa",
colorValueBoxRectEnd: "#049faa",
colorValueBoxBackground: "#f1fbfc",
valueInt: 2,
majorTicks: [
      "0",
      "20",
      "40",
      "60",
      "80",
      "100"

  ],
minorTicks: 4,
strokeTicks: true,
  highlights: [
      {
          "from": 80,
          "to": 100,
          "color": "#03C0C1"
      }
  ],
colorPlate: "#fff",
borderShadowWidth: 0,
  borders: false,
needleType: "line",
colorNeedle: "#007F80",
colorNeedleEnd: "#007F80",
needleWidth: 2,
needleCircleSize: 3,
colorNeedleCircleOuter: "#007F80",
needleCircleOuter: true,
needleCircleInner: false,
```

```
animationDuration: 1500,
animationRule: "linear"
}).draw();

// Function to get current readings on the webpage when it loads for the first
time
function getReadings(){
  var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
    if (this.readyState == 4 &&this.status == 200) {
      var myObj = JSON.parse(this.responseText);
      console.log(myObj);
      var temp = myObj.temperature;
      var hum = myObj.humidity;
gaugeTemp.value = temp;
gaugeHum.value = hum;
    }
  };
xhr.open("GET", "/readings", true);
xhr.send();
}

if (!!window.EventSource) {
  var source = new EventSource('/events');

source.addEventListener('open', function(e) {
console.log("Events Connected");
  }, false);

source.addEventListener('error', function(e) {
    if (e.target.readyState != EventSource.OPEN) {
console.log("Events Disconnected");
    }
  }, false);

source.addEventListener('message', function(e) {
console.log("message", e.data);
  }, false);

source.addEventListener('new_readings', function(e) {
console.log("new_readings", e.data);
    var myObj = JSON.parse(e.data);
    console.log(myObj);
gaugeTemp.value = myObj.temperature;
gaugeHum.value = myObj.humidity;
  }, false);
```

```
}
```

This code does:

- initializing the event source protocol;
- adding an event listener for the new_readings event;
- creating the gauges;
- getting the latest sensor readings from the new_readings event and display them in the corresponding gauges;
- making an HTTP GET request for the current sensor readings when you access the web page for the first time.

Get Readings

When you access the web page for the first time, we'll request the server to get the current sensor readings. Otherwise, we would have to wait for new sensor readings to arrive (via Server-Sent Events), which can take some time depending on the interval that you set on the server.

Add an event listener that calls the getReadings function when the web page loads.

```
// Get current sensor readings when the page loads
window.addEventListener('load', getReadings);
```

The window object represents an open window in a browser. The addEventListener() method sets up a function to be called when a certain event happens. In this case, we'll call the getReadings function when the page loads ('load') to get the current sensor readings.

Now, let's take a look at the getReadings function. Create a new XMLHttpRequest object. Then, send a GET request to the server on the /readings URL using the open() and send() methods.

```
function getReadings() {
  var xhr = new XMLHttpRequest();
xhr.open("GET", "/readings", true);
xhr.send();
}
```

When we send that request, the ESP will send a response with the required information. So, we need to handle what happens when we receive the response. We'll use the onreadystatechange property that defines a function to be executed when the readyState property changes. The readyState property holds the status of the XMLHttpRequest. The response of the request is ready when the readyState is 4, and the status is 200.

- readyState = 4 means that the request finished and the response is ready;
- status = 200 means "OK"

So, the request should look something like this:

```
function getStates(){
  var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
```

```
    if (this.readyState == 4 &&this.status == 200) {
      … DO WHATEVER YOU WANT WITH THE RESPONSE …
    }
  };
xhr.open("GET", "/states", true);
xhr.send();
}
```

The response sent by the ESP is the following text in JSON format (those are just arbitrary values).

```
{
  "temperature" : "25.02",
  "humidity" : "64.01",
}
```

We need to convert the JSON string into a JSON object using the parse() method. The result is saved on the myObj variable.

```
var myObj = JSON.parse(this.responseText);
```

The myObj variable is a JSON object that contains the temperature and humidity readings. We want to update the gauges values with the corresponding readings.

Updating the value of a gauge is straightforward. For example, our temperature gauge is called gaugeTemp (as we'll see later on), to update a value, we can simply call: gaugeTemp.value = NEW_VALUE. In our case, the new value is the temperature reading saved on the myObj JSON object.

```
gaugeTemp.value = myObj.temperature;
```

It is similar for the humidity (our humidity gauge is called gaugeHum).

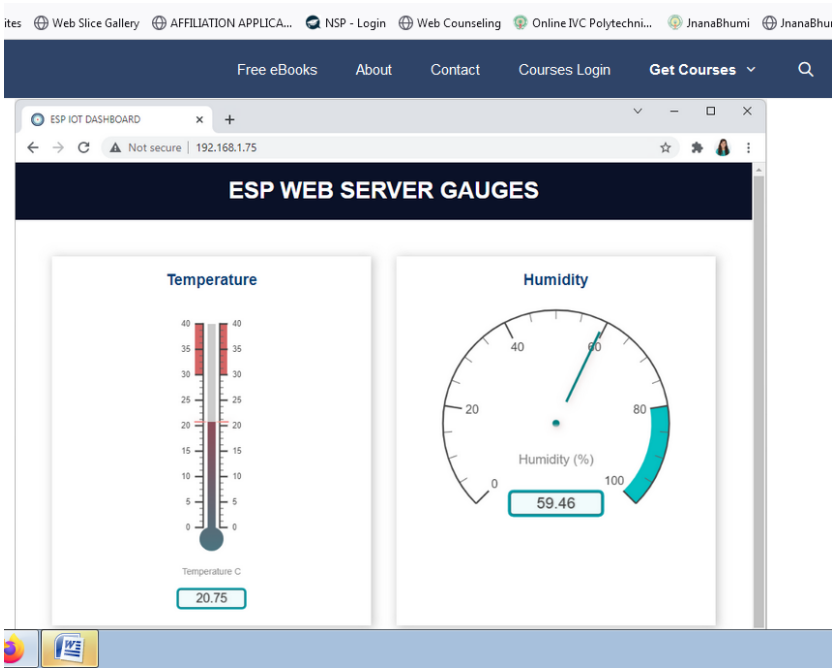
```
gaugeHum.value = myObj.humidity;
```

Here's the complete getReadings() function.

```
function getReadings(){
  var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
    if (this.readyState == 4 &&this.status == 200) {
      var myObj = JSON.parse(this.responseText);
      console.log(myObj);
      var temp = myObj.temperature;
      var hum = myObj.humidity;
gaugeTemp.value = temp;
gaugeHum.value = hum;
    }
  };
xhr.open("GET", "/readings", true);
xhr.send();
}
```

Reference: https://randomnerdtutorials.com/esp32-web-server-gauges/

Output:



Result: The above experiment is designed and executed successfully.

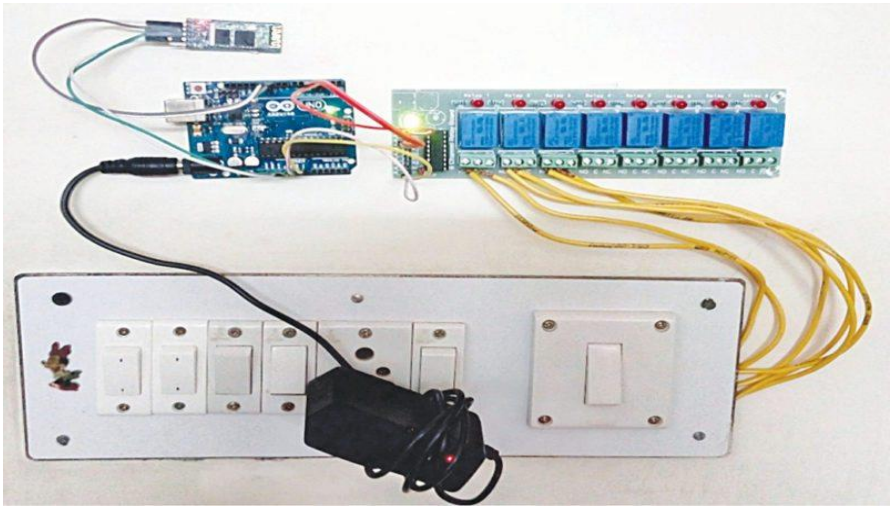
Experiment – 9

Aim: Controlling LEDs/Motors from an Android/Web app, Controlling AC Appliances from an android/web app with the help of relay.

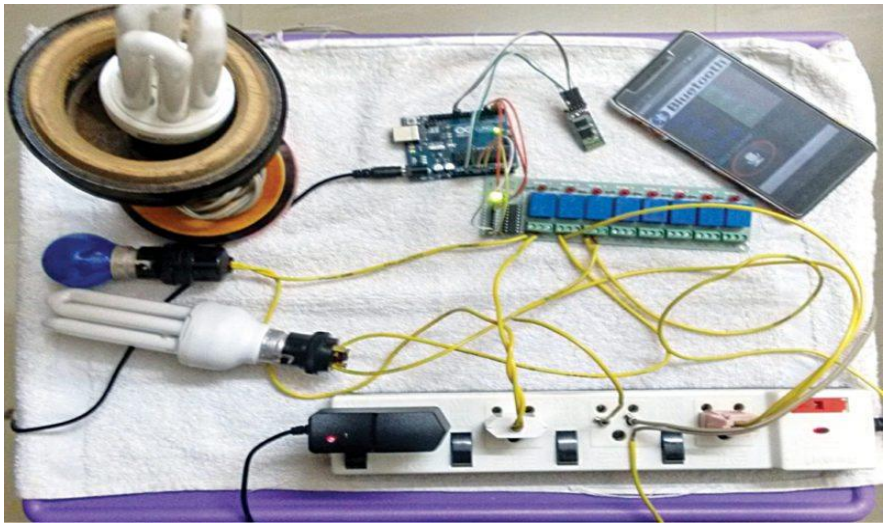
Procedure

Home automation: circuit and working

The home automation circuit is built around an Arduino Uno board, Bluetooth module HC-05 and a 3-channel relay board. The number of channels depends on the number of appliances you wish to control. Arduino Uno is powered with a 12V DC adaptor/power source. The relay module and Bluetooth module can be, in turn, powered using a board power supply of Arduino Uno. Author's prototype is shown in Fig. 1. Connection details for each appliance are shown in Fig. 2.



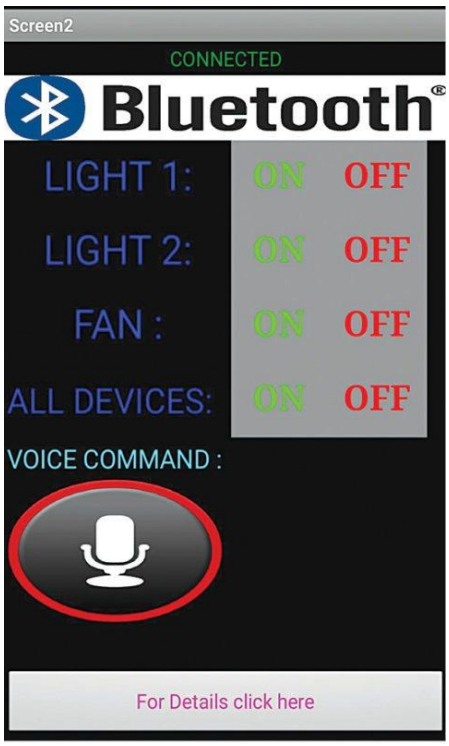
Prototype Model



Connections for the appliances

Bluetooth module

Bluetooth module used in this project is HC-05 (Fig. 4), which supports master and slave mode serial communication (9600-115200 bps) SPP and UART interface. Using these features it can communicate with other Bluetooth-enabled devices like mobile phones, tablets and laptops. The module runs on 3.3V to 5V power supply.



Control panel on Android smartphone

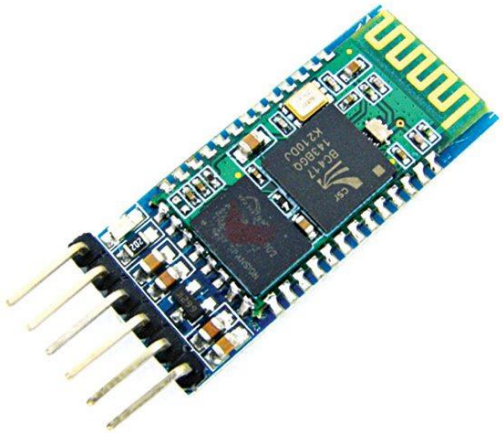
Relay module

A relay allows you to turn on or turn off a circuit using voltage and/or current much higher than what Arduino could handle. Relay provides complete isolation between the low-voltage circuit on Arduino side and the high-voltage side controlling the load. It gets activated using 5V from Arduino, which, in turn, controls electrical appliances like fans, lights and air-conditioners. An 8-channel relay module is shown in below.

Arduino Uno board

Arduino is an open source electronics prototyping platform based on flexible, easy-to-use hardware and software. It is intended for artists, designers, hobbyists and anyone interested in creating interactive objects or environments.

Arduino Uno is based on ATmega328 microcontroller (MCU). It consists of 14 digital input/output pins, six analogue inputs, a USB connection for programming the onboard MCU, a power jack, an ICSP header and a reset button. It is operated with a 16MHz crystal oscillator and contains everything needed to support the MCU. It is very easy to use as you simply need to connect it to a computer using a USB cable, or power it with an AC-to-DC adapter or battery to get started. The MCU onboard is programmed in Arduino programming language using Arduino IDE.

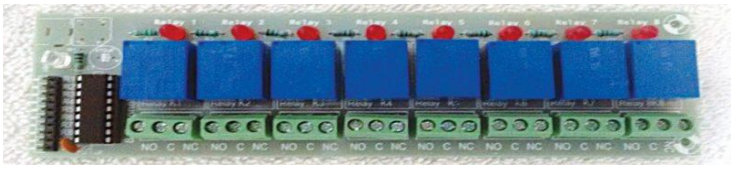


Bluetooth module

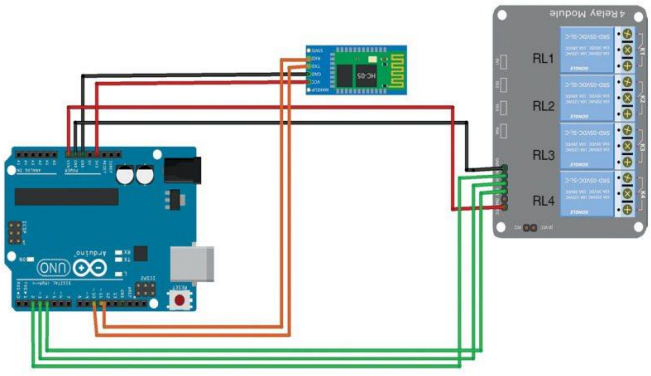
In this home automation project circuit, Pins 10 and 11 of Arduino are connected to pins T_{XD} and R_{XD} of the Bluetooth module, respectively, as shown in Fig. 6.

Pins Gnd and Vcc of the Bluetooth module are connected to Gnd and +3.3V of Arduino board respectively. Pins 2, 3 and 4 are connected to the three relays (RL1, RL2 and RL3) of the relay board. Pins Vin and Gnd of the relay board are connected to pins Vin and Gnd of Arduino board, respectively.

Note. Vin is usually used to give input power, but since we are supplying 12V to Arduino using an adaptor, we can use Vin pin on Arduino to power the 12V relay module.



An 8-channel relay module



Relay module connection

Software

The software program for the home automation project(homeautomation.ino) is written in Arduino programming language called Processing. Arduino Uno is programmed using Arduino IDE software that you can download from arduino.cc. MIT App Inventor software was used to create the Android app (.apk) for this project.

The app on your smartphone sends data when you click on buttons or feed voice commands via Bluetooth in the mobile to Bluetooth module HC-05 connected with Arduino board. Received data pin TXD of the HC-05 is connected to Arduino. Arduino Uno processes the received data and controls the relay board accordingly.

Procedure for installing the Android app (.apk) is as follows:
1. Download the app (homeautomation.apk).
2. Run .apk file. It will prompt you to complete the action. Click Package Installer and then Install.
3. You will also need a voice-recognition app on your Android smartphone. Most smartphones have this app preinstalled. If you do not have it, download one from Google Play Store.

Arduino Code

```
#include <SoftwareSerial.h>
SoftwareSerialBT(10, 11); //TX, RX  pins of arduinorespetively
String command;
void setup()
{
BT.begin(9600);
Serial.begin(9600);
pinMode(2, OUTPUT);
pinMode(3, OUTPUT);
pinMode(4, OUTPUT);
}

void loop() {
  while (BT.available()){  //Check if there is an available byte to read
delay(10); //Delay added to make thing stable
  char c = BT.read(); //Conduct a serial read
  command += c; //build the string.
  }
  if (command.length() > 0) {
Serial.println(command);
if(command == "light on") //this command will be given as an input to switch on
light1
  {
digitalWrite(2, HIGH);
  }
  else if(command == "light off") //this command will be given as an input to switch
off light1 simillarly other commands work
```

```
  {
digitalWrite(2, LOW);
  }
  else if (command == "lamp on")
  {
digitalWrite (3, HIGH);
  }
  else if ( command == "lamp off")
 {
digitalWrite (3, LOW);
 }
  else if (command == "fan on")
 {
digitalWrite (4, HIGH);
 }
  else if (command == "fan of")
 {
digitalWrite (4, LOW);
 }
  else if (command == "all on") //using this command you can switch on all devices
 {
digitalWrite (2, HIGH);
digitalWrite (3, HIGH);
digitalWrite (4, HIGH);
 }
  else if (command == "off")//using this command you can switch off all devices
 {
digitalWrite (2, LOW);
digitalWrite (3, LOW);
digitalWrite (4, LOW);
 }
command="";}} //Reset the variable
//you can add other command to control addition devices by adding an elseif
//and the additions commands you add in sketch can be given through voice
regonisation as i have created the app buttons only to control three devices
```

Construction and testing

Assemble the circuit as shown in the circuit diagram. Open Arduino IDE and compile the program (sketch). Upload the sketch (homeautomation.ino) to Arduino board. Switch on the power supply to Arduino by connecting it to 12V power source. Pair Bluetooth module with your Android phone. Type password '1234' (default password) of Bluetooth module.

Click Bluetooth Image on the app to connect it with the Bluetooth module. It automatically connects and displays as Connected in the app.

You are now ready to control the appliances using the app. You can either use on/off buttons or voice commands to control the appliances. You can control more electrical

appliances by increasing the number of channels in the relay. For instance, using an 8-channel relay, you can control up to eight devices. For this, you need to alter the source code by adding input commands and voice commands to control the devices

Result: The above experiment is designed and executed successfully.

Experiment – 10

Aim: Displaying humidity and temperature data on a web-based application

Procedure

Using Internet of Things (IOT), we can control any electronic equipment in homes and industries. Moreover, you can read a data from any sensor and analyse it graphically from anywhere in the world. Here, we can read temperature and humidity data from DHT11 sensor and upload it to a Thing Speak cloud using Arduino Uno and ESP8266-01 module. Arduino Uno is MCU, it fetch a data of humidity and temperature from DHT11 sensor and Process it and give it to a ESP8266 Module.ESP8266 is a WiFi module, it is one of the leading platform for Internet of Things. It can transfer a data to IOT cloud.

Hardware Requirements

- Arduino Uno
- ESP8266-01
- DHT11
- AMS1117-3.3V
- 9V battery

Software Requirements

- Arduino IDE

Circuit and Working

First make the connection as shown in fig: 1.1.The 2nd pin is of DHT11 is a data pin, it can send a temperature and humidity value to the 5th pin of Arduino Uno.1st and 4th pin of DHT11 is a Vcc and Gnd and 3rd pin is no connection. The Arduino Uno processes a temperature and humidity value and send it to a ESP8266 WiFi module. The Tx and Rx pin of ESP8266 is connected to the 2nd (Rx) and 3rd (Tx) of Arduino Uno. Make sure that input voltage of ESP8266 must be 3.3V, not a 5V (otherwise it would damage a device).For that, we are using AMS1117 Voltage regulator circuit. It can regulate a voltage from 9V to 3.3V and will give it to Vcc pin of ESP8266.The Ch_Pd is a chip enable pin of ESP8266 and should be pullup to 3.3V through 3.3KΩ resistor. For reset the module pull down the RST pin of ESP8266 to Gnd.ESP8266 have 2 GPIO pins GPIO 0 and GPIO 2.

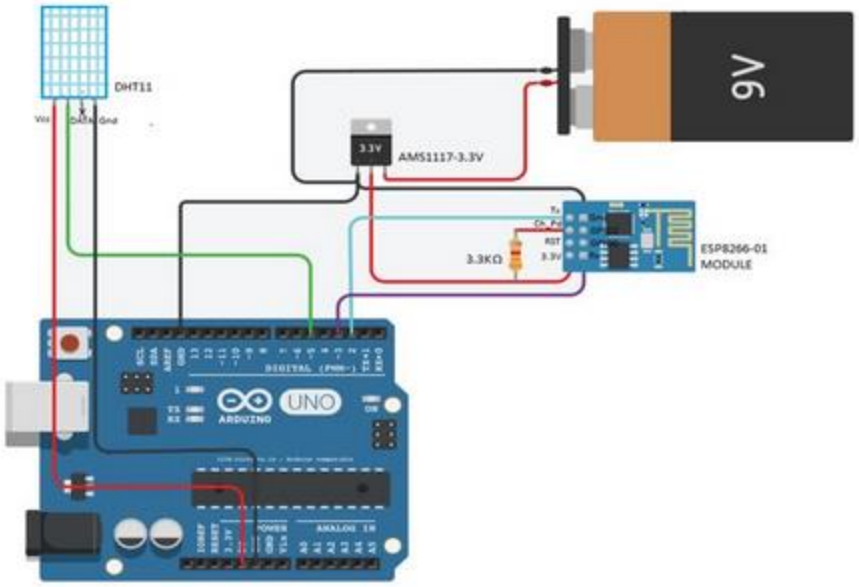


Fig 1.1: Circuit diagram for monitoring Humidity and Temperature in IOT cloud

Construction and Testing

ThingSpeak is an open source platform to store and retrieve a data for Internet of Things application. To use this, you need to register in Things Speak cloud and then login to your account. After create a new channel with temperature in one field and humidity in another field as shown in Fig: 1.2. Once you created a new channel, it will generate a two API keys, they are READ API keys and WRITE API keys. First, copy the WRITE API keys from Things Speak and paste it into the line (String apiKey = "OX9T8Y9OL9HD0UBP";) of the program. Next, replace the Host_Name and Password with your WiFi name and WiFi password in the two lines given below in the program. (String Host_Name = "Pantech" and String Password = "pantech123")

The Arduino program Uses DHT library, if it is not presented in your arduino IDE, select SketchàInclude libraryàManage librariesàInstall DHT Sensor library. Then compile the program and upload to a Arduino Uno through Arduino IDE. Ensure that WiFi modem and internet connection in your Smartphone or PC are working properly. After uploaded a program, the Temperature and Humidity data is uploaded on ThingSpeak platform. You can see it graphically in the private view window of your channel as shown in Fig: 1.3. And you can able to see the uploaded data from serial port of Arduino IDE.

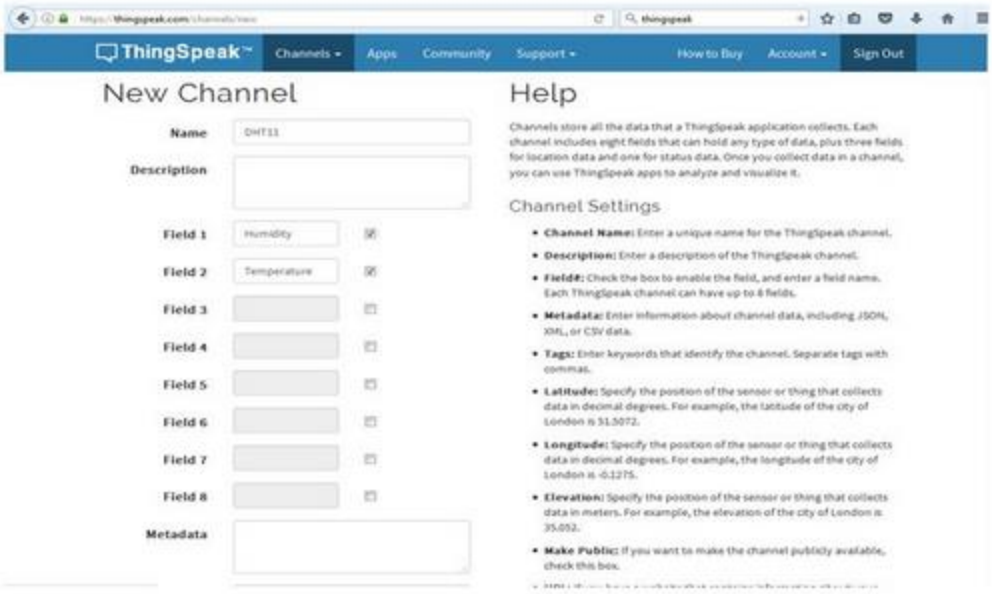


Fig: 1.2: Creating new channel on ThingSpeak cloud

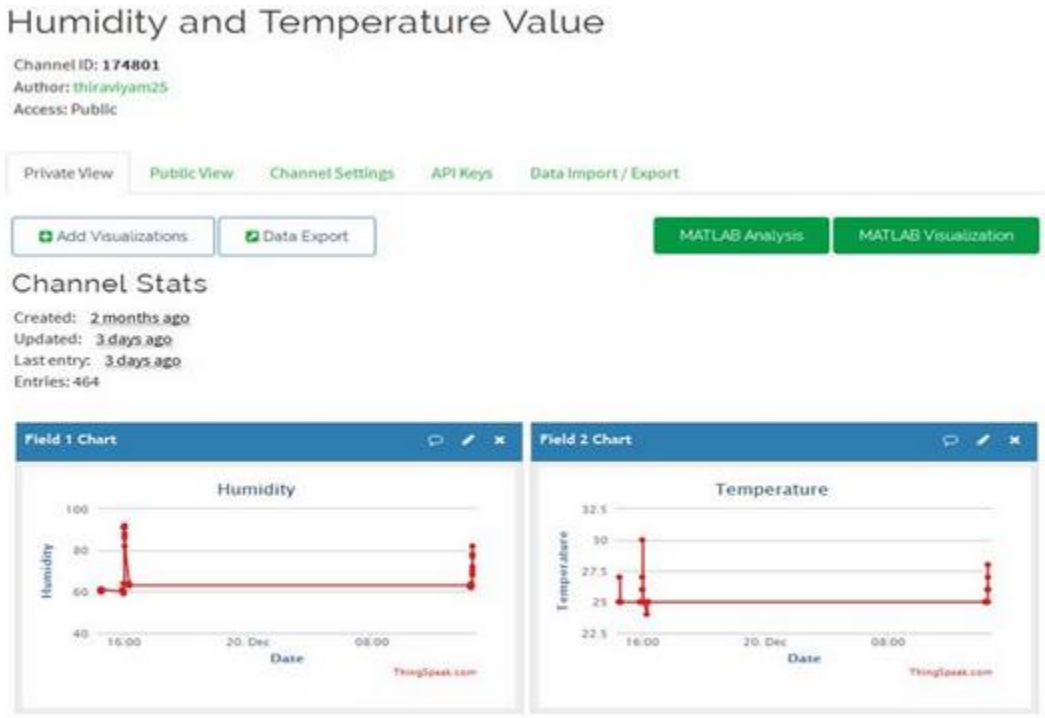


Fig: 1.3: Graphical representation of Humidity and Temperature data

Arduino Code

```
#include "DHT.h"
#include
#define DHTPIN 5                    // Digital Pin 5

#define DHTTYPE DHT11               // We are Using DHT11

String apiKey = "OX9T8Y9OL9HD0UBP";    // Edit this API key according to your
Account
String Host_Name = "Pantech";         // Edit Host_Name
String Password = "pantech123";          // Edit Password
SoftwareSerialser(2, 3);              // RX, TX
int i=1;
DHT dht(DHTPIN, DHTTYPE);             // Initialising Pin and Type of DHT
void setup() {
Serial.begin(115200);                  // enable software serial
ser.begin(115200);                    // reset ESP8266
ser.println("AT+RST");               // Resetting ESP8266
dht.begin();                        // Enabling DHT11
char inv ='"';
String cmd = "AT+CWJAP";
cmd+= "=";
cmd+= inv;
cmd+= Host_Name;
cmd+= inv;
cmd+= ",";
cmd+= inv;
cmd+= Password;
cmd+= inv;
ser.println(cmd);                   // Connecting ESP8266 to your WiFi Router
  }

// the loop

void loop() {
  int humidity =  dht.readHumidity();             // Reading Humidity Value
  int temperature = dht.readTemperature();        // Reading Temperature Value
  String state1=String(humidity);                 // Converting them to string
  String state2=String(temperature);              // as to send it through URL
  String cmd = "AT+CIPSTART=\"TCP\",\"";          // Establishing TCP connection
cmd += "184.106.153.149";                        // api.thingspeak.com
cmd += "\",80";                                  // port 80
ser.println(cmd);
Serial.println(cmd);
 if(ser.find("Error")){
Serial.println("AT+CIPSTART error");
    return;
  }
String getStr = "GET /update?api_key=";         // prepare GET string
```

```
getStr += apiKey;
getStr +="&field1=";
getStr += String(state1);                    // Humidity Data
getStr +="&field2=";
getStr += String(state2);                    // Temperature Data
getStr += "\r\n\r\n";
cmd = "AT+CIPSEND=";
cmd += String(getStr.length());              // Total Length of data
ser.println(cmd);

Serial.println(cmd);
if(ser.find(">")){
ser.print(getStr);
Serial.print(getStr);
  }
else{
ser.println("AT+CIPCLOSE");                  // closing connection
    // alert user
Serial.println("AT+CIPCLOSE");
  }
delay(1000);                                 // Update after every 15 seconds

}
```

Result: The above experiment is designed and executed successully

Table of Contents

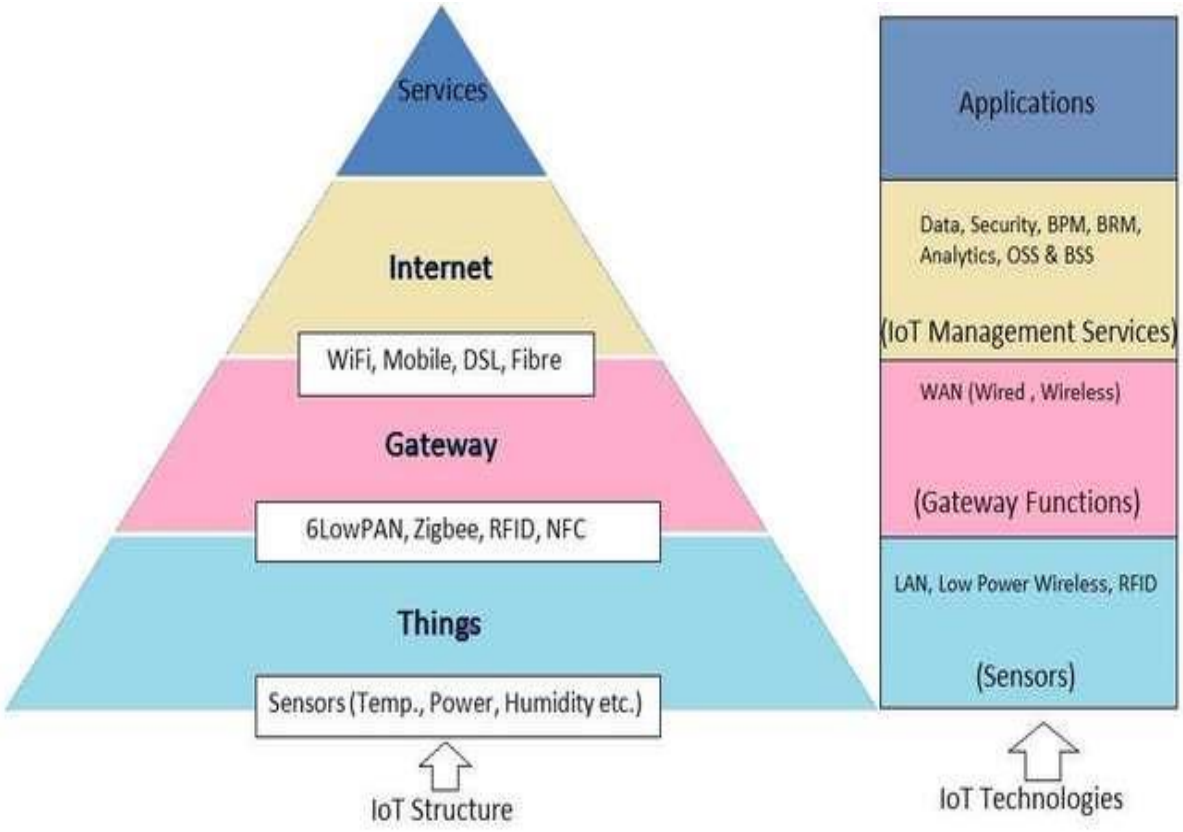
Internet of Things (IOT)

Introduction: IOT stands for "Internet of Things". The IOT is a name for the vast collection of "things" that are being networked together in the home and workplace (up to 20 billion by 2020 according to Gardner, a technology consulting firm).

Characteristics of the IOT

Networking	These IOT devices talk to one another (M2M communication) or to servers located in the local network or on the Internet. Being on the network allows the device the common ability to consume and produce data.
Sensing	IOT devices sense something about their environment.
Actuators	IOT devices that do something. Lock doors, beep, turn lights on, or turn the TV on

Communications in IoT



Communications are important to IOT projects. In fact, communications are core to the whole genre. There is a trade-off for IOT devices. The more complex the protocols and higher the data rates, the more powerful processor needed and the more electrical power the IOT device will consume.

TCP/IP base communications (think web servers; HTTP-based commutation (like REST servers); streams of data; UDP) provide the most flexibility and functionality at a cost of processor and electrical power.

Low-power Bluetooth and Zigbee types of connections allow much lower power for connections with the corresponding decrease in bandwidth and

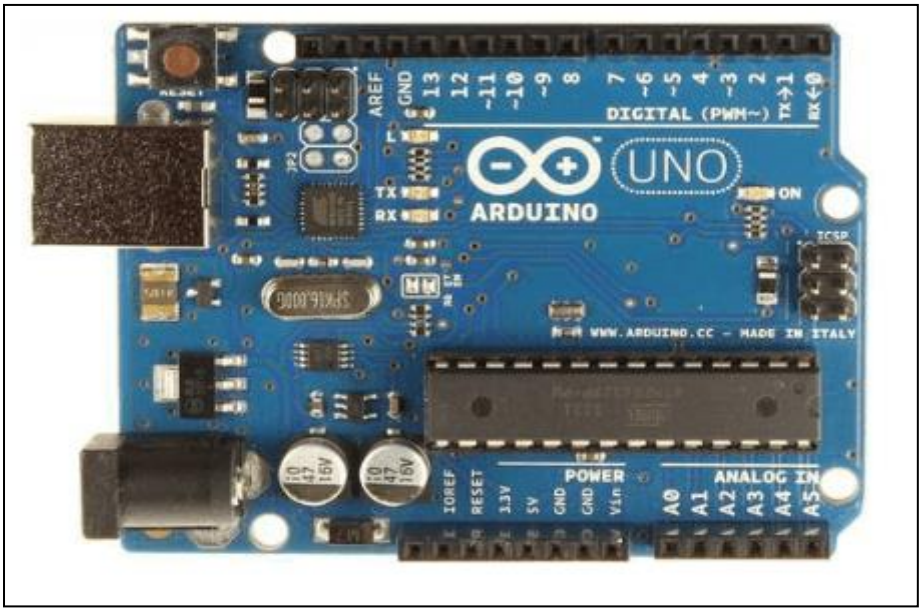
functionality. IOT projects can be all over the map with requirements for communication flexibility and data bandwidth requirements.

Arduino in IoT

In IoT applications the Arduino is used to collect the data from the sensors/devices to send it to the internet and receives data for purpose of control of actuators.

Arduino Uno

Introduction: The Arduino Uno is an open-source microcontroller board based on the Microchip ATmega328P microcontroller and developed by Arduino.cc. The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards (shields) and other circuits. The board has 14 digital I/O pins (six capable of PWM output), 6 analog I/O pins, and is programmable with the Arduino IDE (Integrated Development Environment), via a type B USB cable. It can be powered by the USB cable or by an external 9-volt battery, though it accepts voltages between 7 and 20 volts. The word "uno" means "one" in Italian and was chosen to mark the initial release of Arduino Software.



Features of the Arduino

1. Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.

2. The board functions can be controlled by sending a set of instructions to the microcontroller on the board via Arduino IDE.
3. Arduino IDE uses a simplified version of C++, making it easier to learn to program.
4. Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

Arduino IDE

(Integrated Development Environment)

Introduction: The Arduino Software (IDE) is easy-to-use and is based on the Processing programming environment. The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++. The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

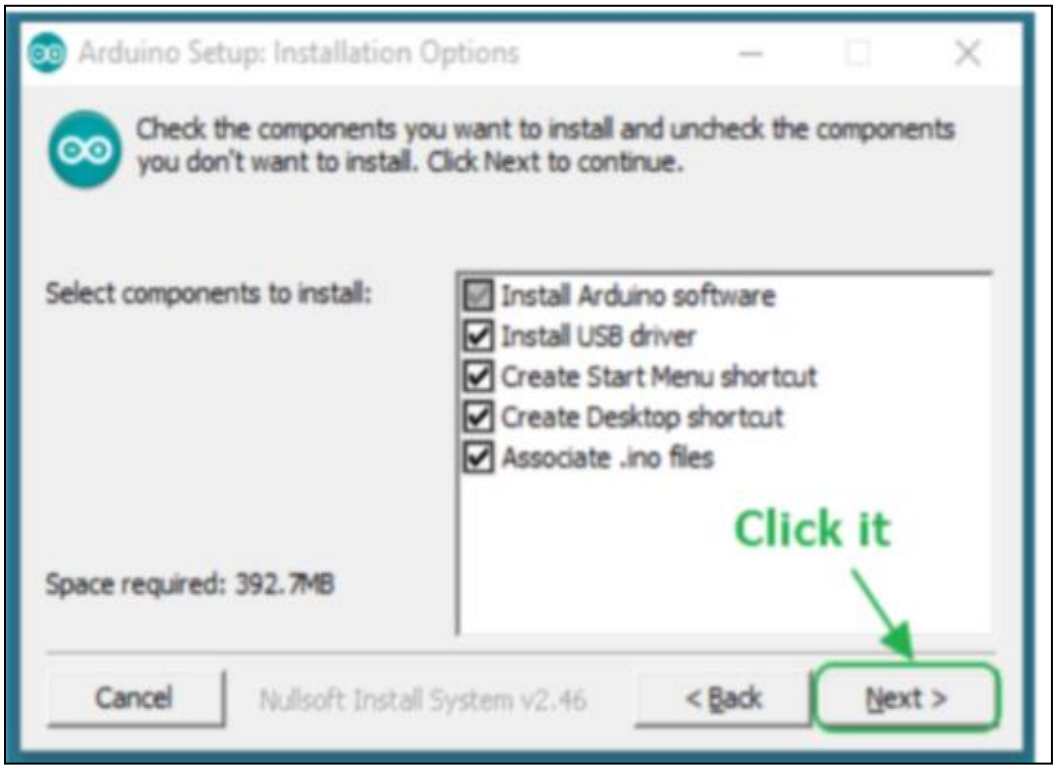
The Arduino Software (IDE) – contains:

- A text editor for writing code
- A message area
- A text consoles
- A toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

Installation of Arduino Software (IDE)

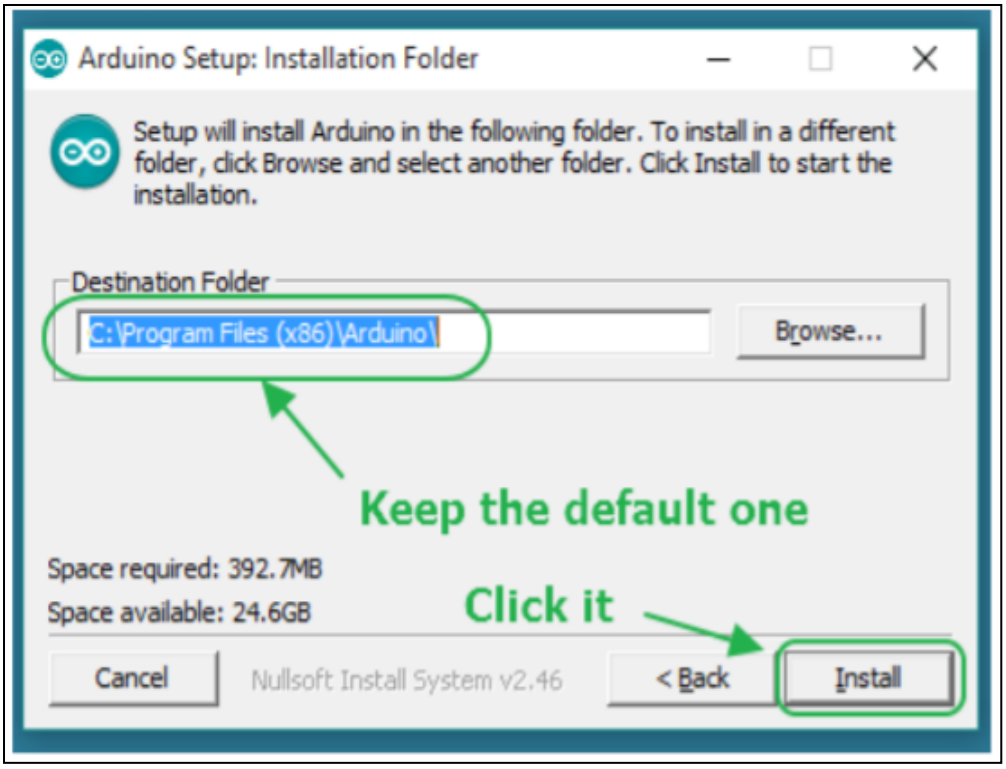
Step1: Downloading

- To install the Arduino software, download this page: http://arduino.cc/en/Main/Software and proceed with the installation by allowing the driver installation process.



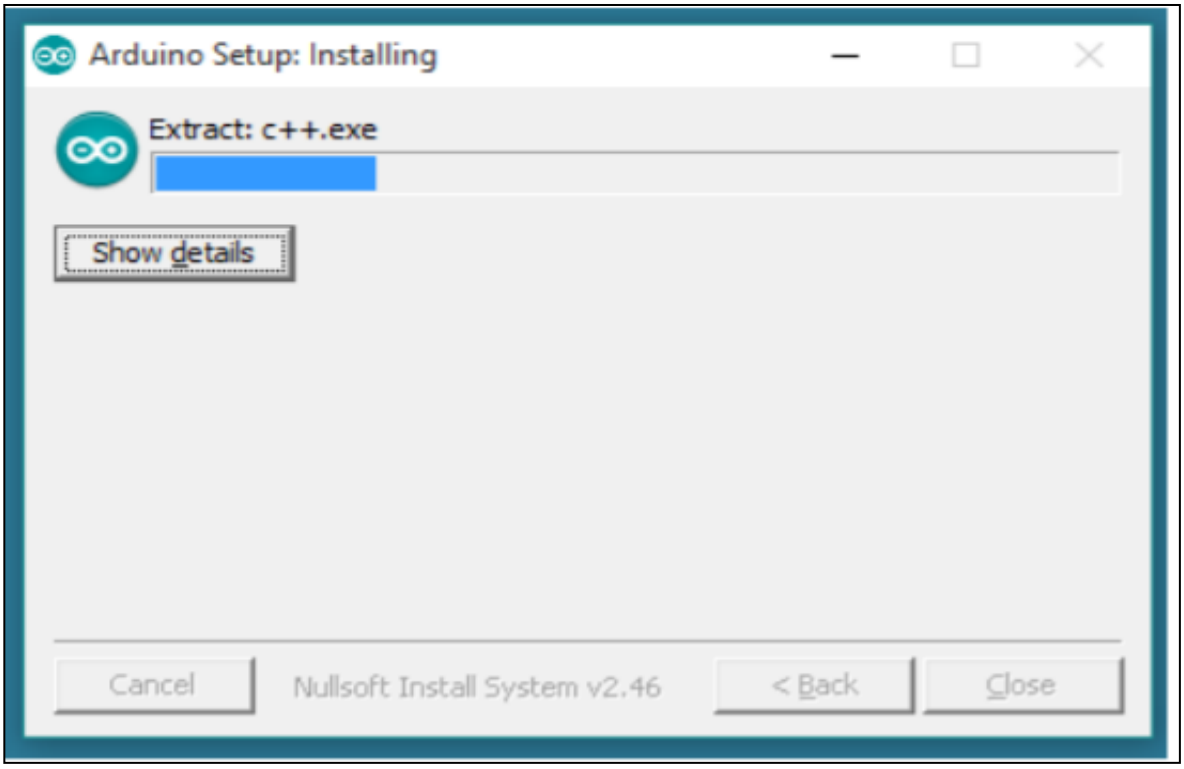
Step 2: Directory Installation

- Choose the installation directory.



Step 3: Extraction of Files

- The process will extract and install all the required files to execute properly the Arduino Software (IDE)

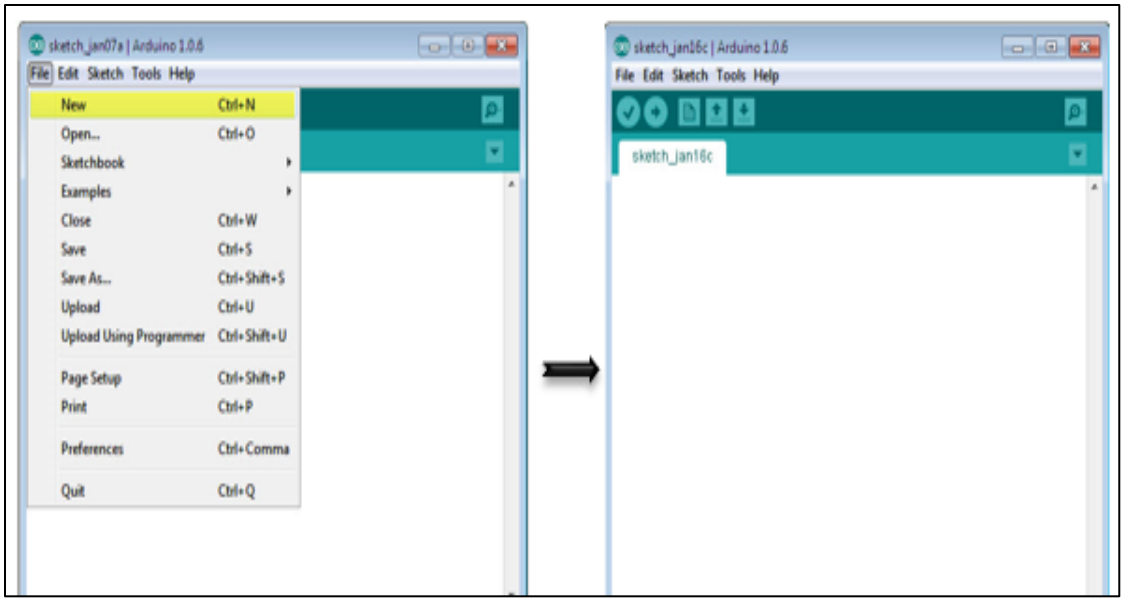


Step 4: Connecting the board

- The USB connection with the PC is necessary to program the board and not just to power it up. The Uno and Mega automatically draw power from either the USB or an external power supply. Connect the board to the computer using the USB cable. The green power LED (labelled PWR) should go on.

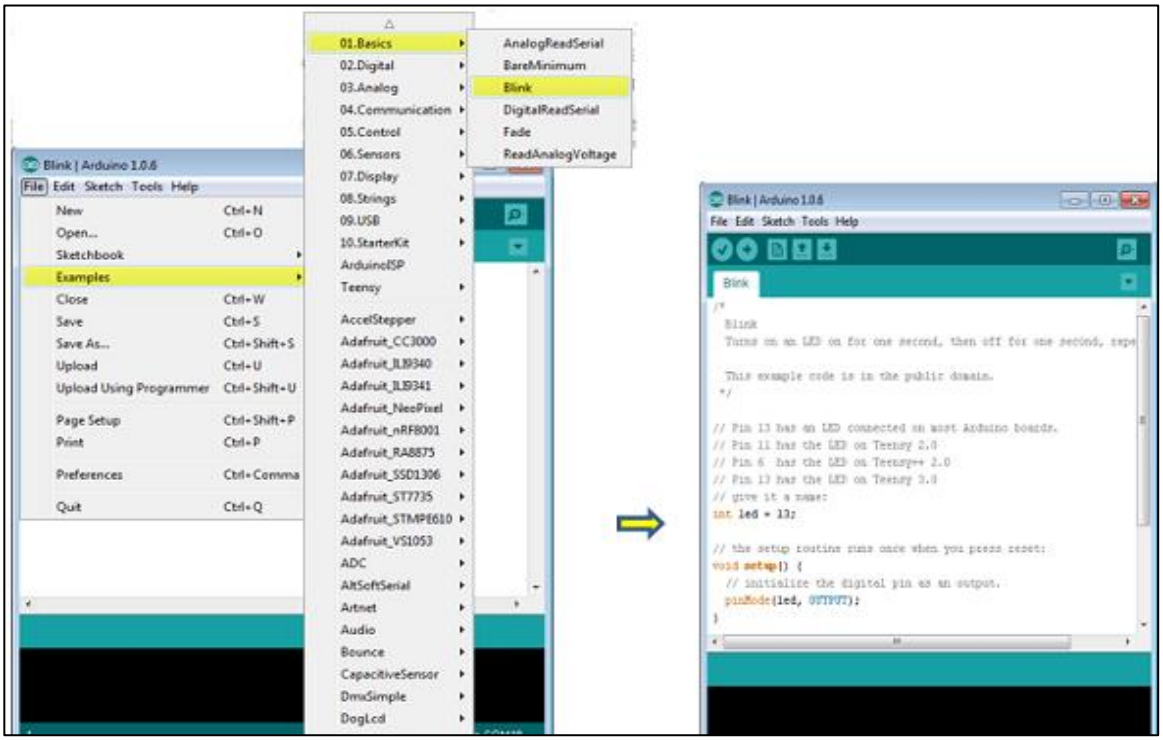
Step 5: Working on the new project

- Open the Arduino IDE software on your computer. Coding in the Arduino language will control your circuit.
- Open a new sketch File by clicking on New.



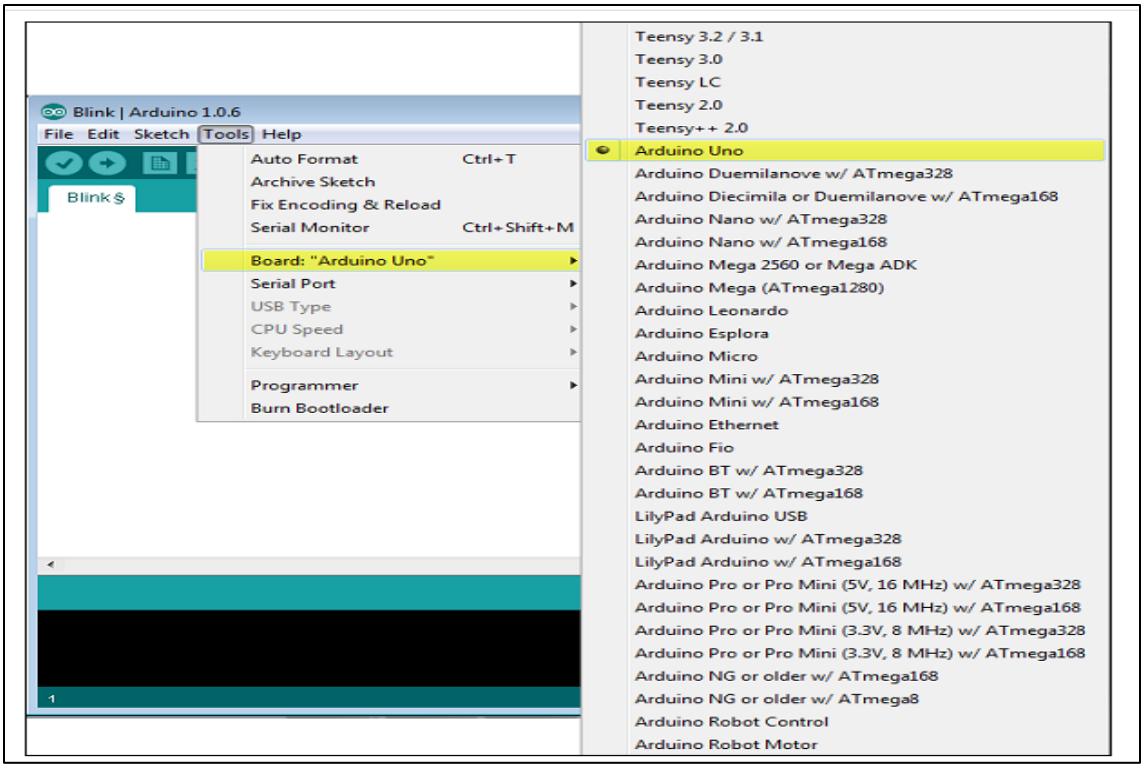
Step 6: Working on an existing project

- To open an existing project example, select File → Example → Basics → Blink.



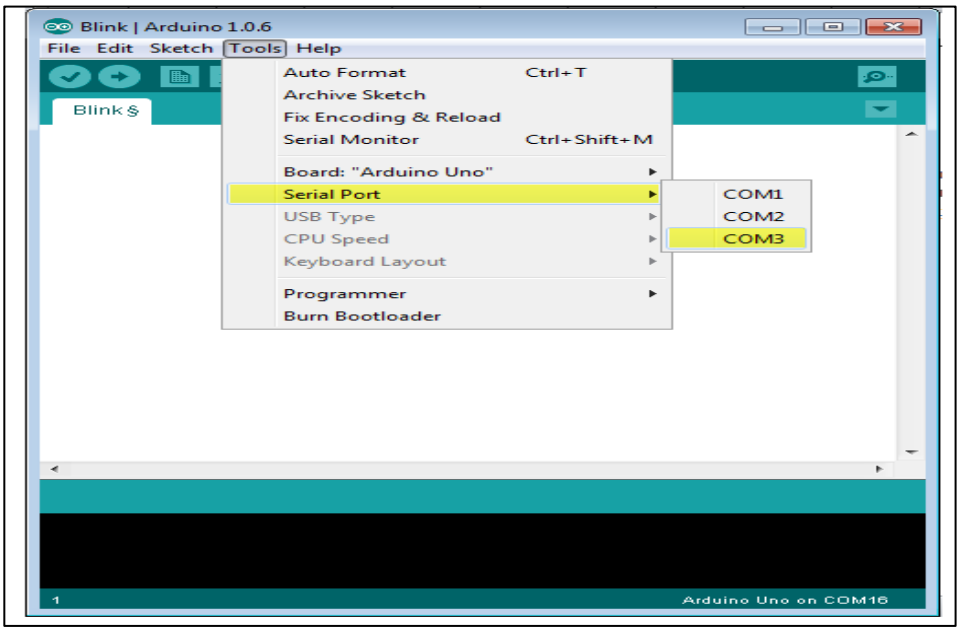
Step 7: **Select your Arduino board.**

- To avoid any error while uploading your program to the board, you must select the correct Arduino board name, which matches with the board connected to your computer.
- Go to Tools → Board and select your board.



Step 8: Select your serial port

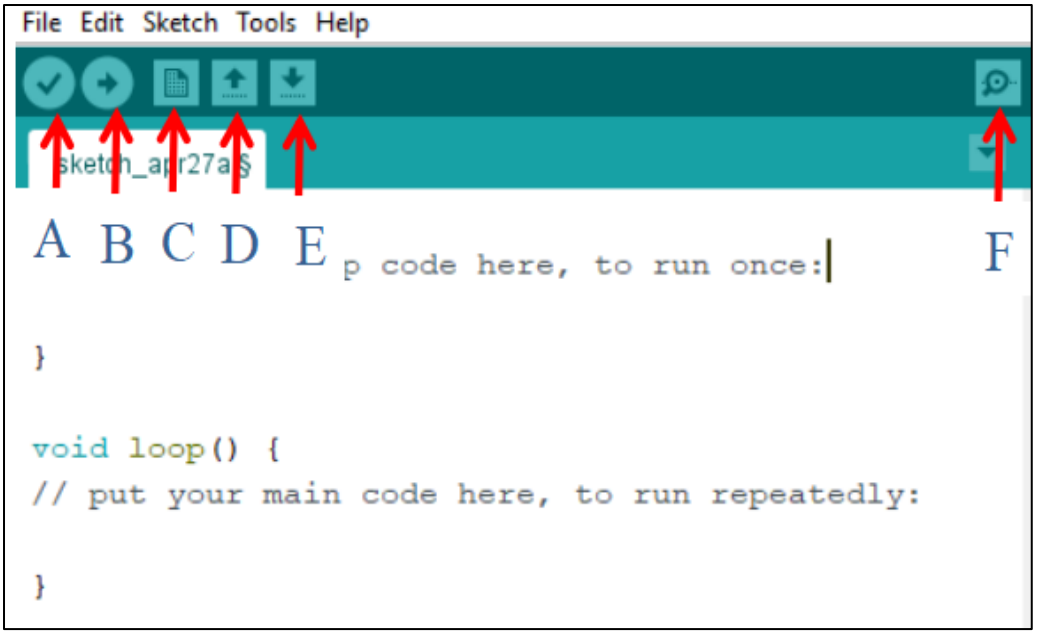
- Select the serial device of the Arduino board.
- Go to Tools → Serial Port menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports).
- To find out, you can disconnect your Arduino board and re-open the menu, the entry that disappears should be of the Arduino board. Reconnect the board and select that serial port.



Step 9: Upload the program to your board.

- Click the "Upload" button in the environment.
- Wait a few seconds; you will see the RX and TX LEDs on the board, flashing.
- If the upload is successful, the message "Done uploading" will appear in the status bar.

A	Verify
B	Upload
C	New
D	Open
E	Save
F	Serial Motor


File Edit Sketch Tools Help
A B C D E
F
p code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}

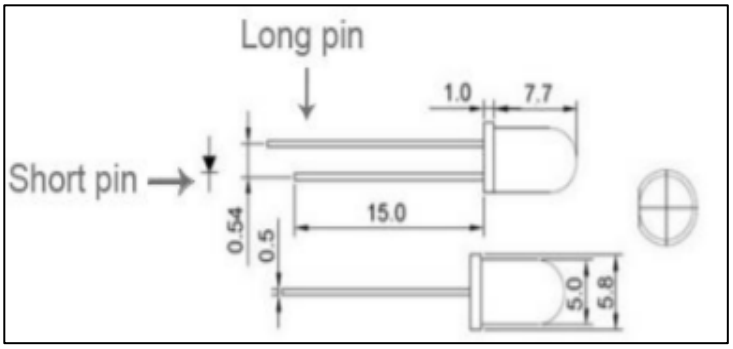

File Edit Sketch Tools Help
BareMinimum §
Sketch Name
Global Area
void setup() {
Setup Area
}
void loop() {
Loop Area
}

Practical 1

Controlling the Light Emitting Diode (LED) with a push button.

Introduction: Push-button is a very simple mechanism which is used to control electronic signal either by blocking it or allowing it to pass. This happens when mechanical pressure is applied to connect two points of the switch together. Push buttons or switches connect two points in a circuit when pressed. When the push-button is released, there is no connection between the two legs of the push-button. Here it turns on the built-in LED on pin 11 when the button is pressed. The LED stays ON as long as the button is being pressed.

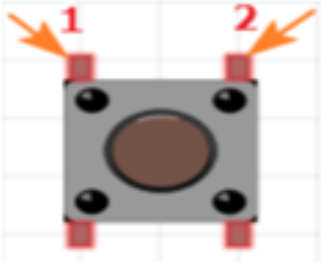
LED Specifications



Pin definition

Long pin	+5V
Short pin	GND

Push Button



Specifications:

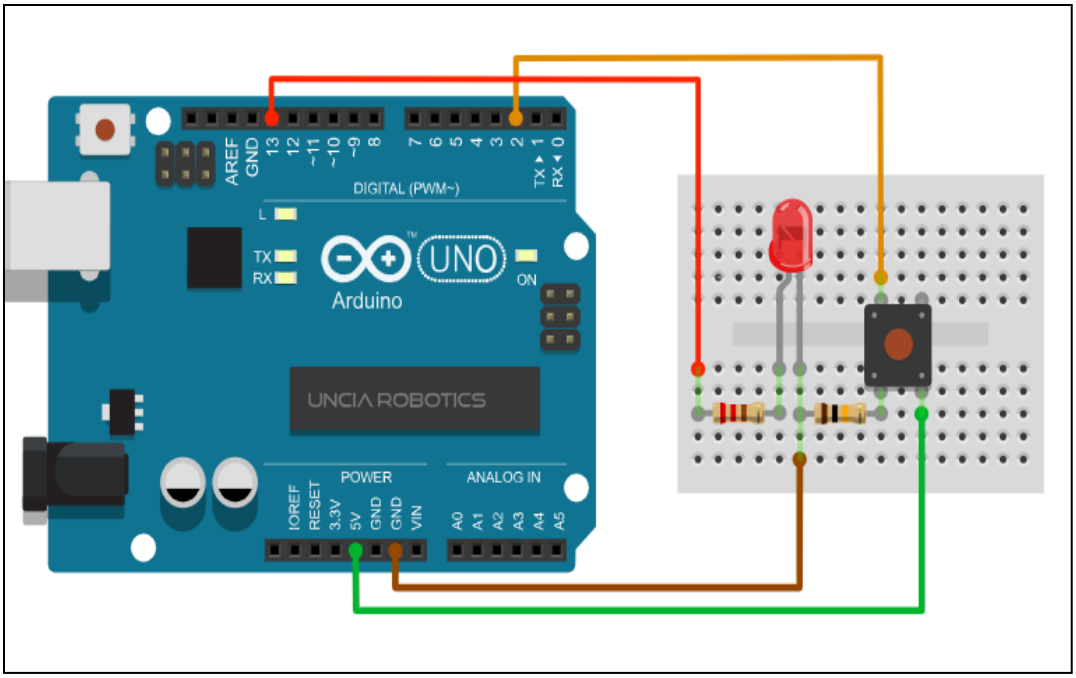
Size	6 x 6 x 5mm
Temperature	-30 ~ +70 Centigrade

Hardware Required:

Component Name	**Quantity**
Arduino UNO	1
LED	1
Push Button	1
220Ω resistor	1
10KΩ resistor	1
USB Cable	1

Breadboard	1
Jumper wires	Several

Connection Diagram:



Steps of working

1. Insert the push button into your breadboard and connect it to the digital pin 7(D7) which act as INPUT.
2. Insert the LED into the breadboard. Attach the positive leg (the longer leg) to digital pin 11 of the Arduino Uno, and the negative leg via the 220-ohm resistor to GND. The pin D11 is taken as OUTPUT.
3. The 10kΩ resistor used as PULL-UP resistor and 220 Ω resistors is used to limit the current through the LED.
4. Upload the code as given below.

5. Press the push-button to control the ON state of LED.

The Sketch

- This sketch works by setting pin D7 as for the push button as INPUT and pin 11 as an OUTPUT to power the LED.
- The initial state of the button is set to OFF.
- After that the run a loop that continually reads the state from the pushbutton and sends that value as voltage to the LED. The LED will be ON accordingly.

```
/**********Pressing Button LED*****/
const int buttonPin = 7; // choose the pin for the pushbutton
const int ledPin = 11;  // choose the pin for a LED
int buttonState = 0;   // variable for reading the pushbutton pin status
void setup()
{
  pinMode(ledPin, OUTPUT);  // declare LED as output
  pinMode(buttonPin, INPUT);    // declare pushbutton as input
}
void loop()
{
 buttonState = digitalRead(button Pin);  // read input value
 if (buttonState == HIGH)
{        // check if the input is HIGH (button pressed)
 digitalWrite(ledPin, HIGH);  // turn LED ON
```

```
 }
else
 {
  digitalWrite(ledPin, LOW);  // turn LED OFF}}
```

Observation Table:

Sr no.	Push button State	LED State
1		
2		

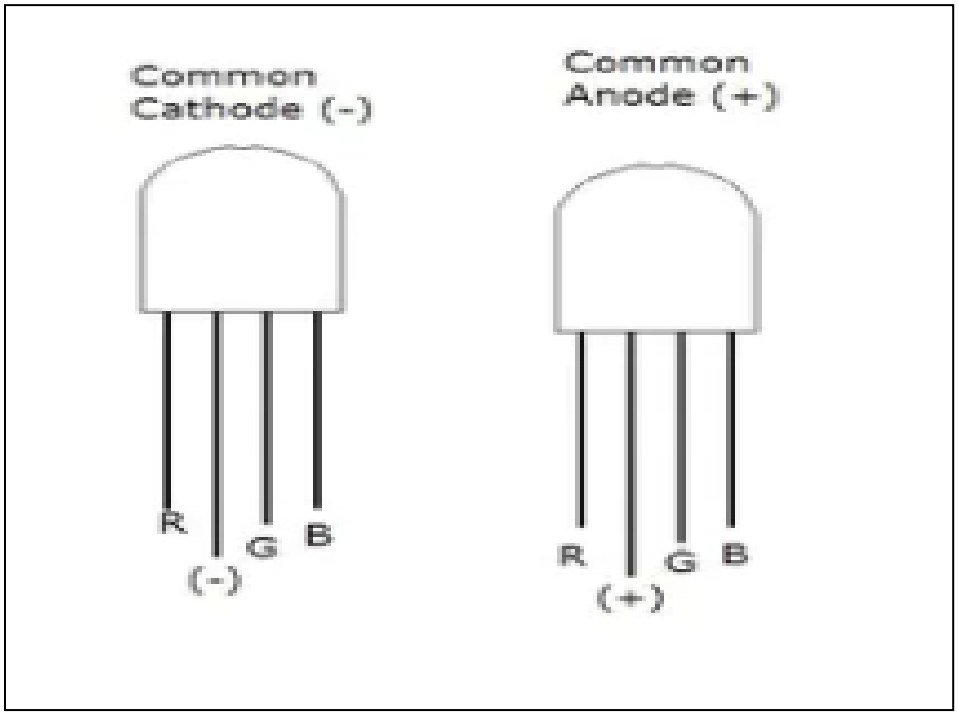
Precautions:

1. The pushbutton is square so it is important to set it appropriately on breadboard.
2. While making the connections make sure to use a pull-down resistor because directly connecting two points of a switch to the circuit will leave the input pin in floating condition and circuit may not work according to the program.
3. It is very important to set pinMode() as OUTPUT first before using digitalWrite() function on that pin.
4. If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim.

Practical 2

Interfacing the RGB LED with the Arduino

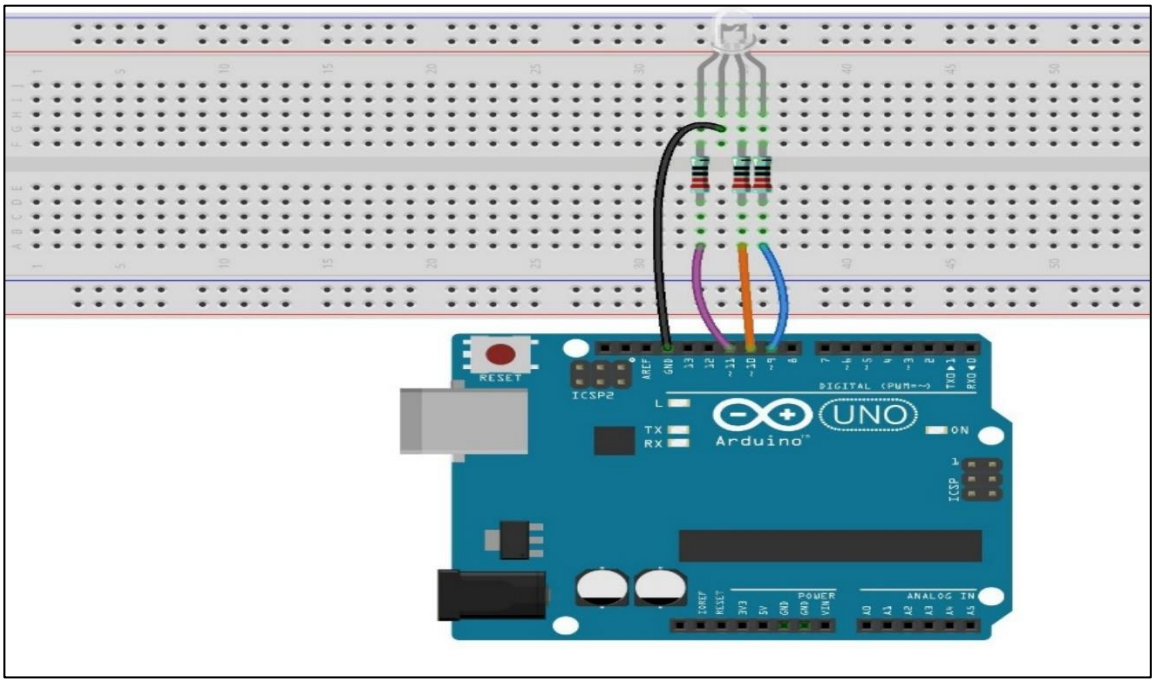
Introduction: There are actually two types of RGB LED's; the common cathode one and the common anode one. In the common cathode RGB led, the cathode of all the LED's is common and we give PWM signals to the anode of LED's while in the common anode RGB led, the anode of all the LED's is common and we give PWM signals to the cathode of LED's. Inside the RGB led, there are three more LED's. So, by changing the brightness of these LED's, we can obtain many other colors. To change brightness of RGB led, we can use the PWM pins of Arduino. The PWM pins will give signal different duty cycles to the RGB led to obtain different colors.



Hardware Required:

Component Name	Quantity
Arduino UNO	1
RGB LED	1
220Ω/330Ω resistor	3
USB Cable	1
Breadboard	1
Jumper wires	several

Connection Diagram:



Steps of working

1. Insert the RGB LED into your breadboard and connect its cathode pin to the GND of the Arduino.
2. Insert the LED into the breadboard. Attach Red pin to pin 8, Green pin to pin 9 and Blue pin to pin 10 of the Arduino via the 220-ohm resistor, and the negative leg to GND.
3. Upload the code as given below.
4. Observe the changes in the color of the RGB LED.

The Sketch

This sketch works by setting pinsD8, D9, D10 as for the different legs of RGB LED. After that the run a loop that continually reads the value from the pins and sends that value as voltage to the LED. The voltage value is between 0–5 volts, and the blinking of the LED will vary accordingly.

```
/************RGB LED Blink*******/
void setup() {
  // put your setup code here, to run once:
pinMode(8,OUTPUT);
pinMode(9,OUTPUT);
pinMode(10,OUTPUT);
}
void loop() {
  // put your main code here, to run repeatedly:
digitalWrite (8,HIGH);
digitalWrite (10,LOW);
delay(1000);
digitalWrite (9,HIGH);
```

```
digitalWrite (8,LOW);
delay(1000);
digitalWrite (10,HIGH);
digitalWrite (9,LOW);
delay(1000);
}
```

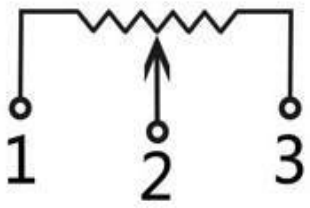
Observations:

Sr. No.	Time(ms)	Color of LED
1		
2		
3		

Practical 3

Controlling the LED blink rate with the potentiometer interfacing with Arduino

Introduction: A potentiometer is a variable resistor with a knob that allows altering the resistance of the potentiometer. The potentiometer manipulates a continuous analog signal, which represents physical measurements. The potentiometer is used with Arduino to control the blink rate of the LED. The potentiometer is an adjustable resistor, and its operating principle is shown in the following figure:

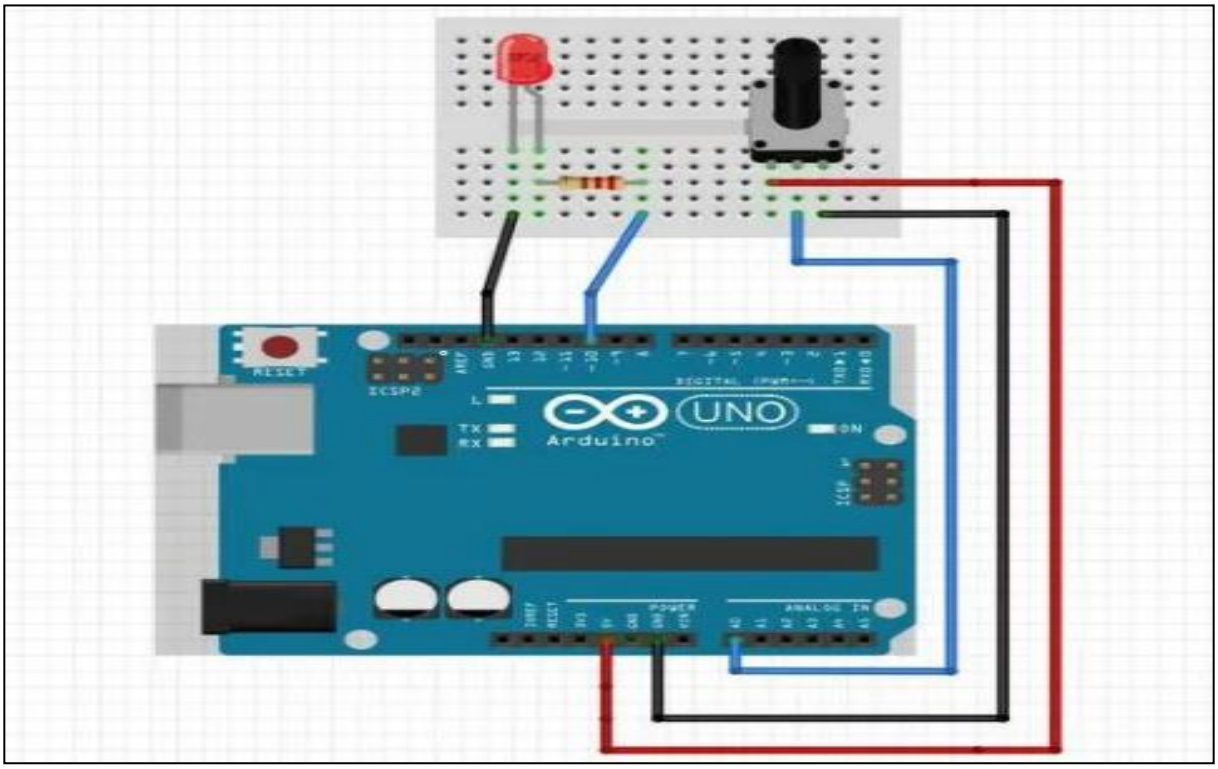


Hardware Required:

Component	Quantity
Arduino Uno	1
Bread board	1
220Ω current limiting resistor	1

5mm LED	1
10KΩ Potentiometer	1
Jumper Wires	Several
Supporting USB data cable	1

Working Diagram:



Steps of working

1. Insert the potentiometer into your breadboard and connect its center pin to the analog pin A2 and the remaining pin to GND on the breadboard.

2. Insert the LED into the breadboard. Attach the positive leg (the longer leg) to pin 13 of the Arduino via the 220-ohm resistor, and the negative leg to GND.
3. Upload the code as given below.
4. Turn the potentiometer to control the brightness of the LED and move the position of pin 2 by rotating the knob, changing the resistance value from pin 2 to both ends.
5. Observe the changes in the blinking rate of the LED.

The Sketch

This sketch works by setting pin A2 as for the potentiometer and pin 9 as an OUTPUT to power the LED. After that the run a loop that continually reads the value from the potentiometer and sends that value as voltage to the LED. The voltage value is between 0–5 volts, and the brightness of the LED will vary accordingly.

Observation Table:

Sr. no.	Voltage	Light Intensity
1		
2		
3		
4		
5		

Practical 4

Detection of the light using photo resistor

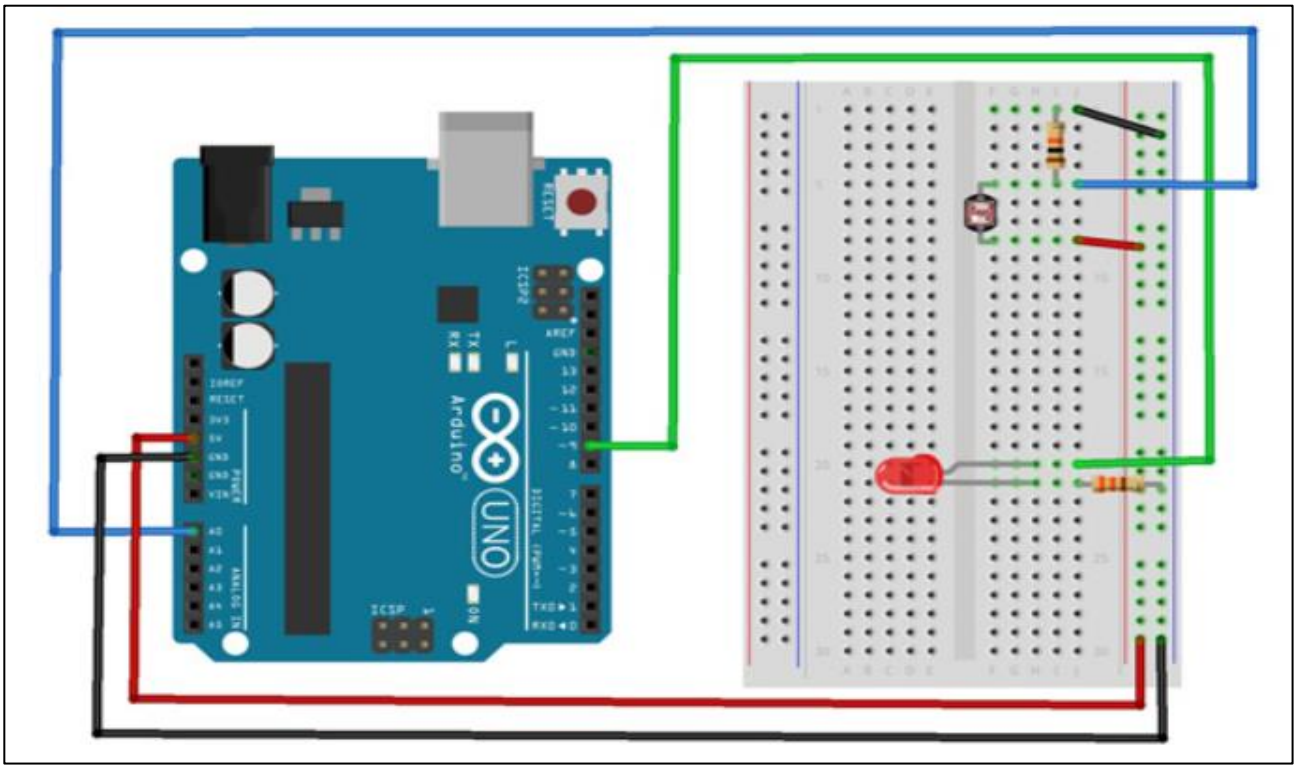
Introduction: A photo resistor or photocell is a light-controlled variable resistor made of a high resistance semiconductor. The resistance of a photo resistor decreases with increasing incident light intensity. A photo resistor can be applied in light-sensitive detector circuits, and light- and dark-activated switching circuits. It's also called light-dependent resistor (LDR).

Hardware Required:

Component Name	Quantity
Arduino UNO	1
LED	1
Photo Resistor	1
10KΩ Resistor	1
220Ω Resistor	1

USB Cable	1
Breadboard	1
Jumper wires	several

Connection diagram:



Steps of working

1. Insert the photo resistor into your breadboard and connect its pin to the analog pin A0 and the remaining pin to supply on the breadboard.
2. Insert the LED into the breadboard. Attach the positive leg (the longer leg) to pin 9 of the Arduino via the 220-ohm resistor, and the negative leg to GND.

3. Insert the 10K-ohm resistor
4. Upload the code
5. Turn the photo resistor to ON the LED
6. Observe the changes in the state of the LED.

The Sketch

This sketch works by setting pin A0 as for the photo sensor and pin 9 as an OUTPUT to power the LED. After that the run a loop that continually reads the value from the photo resistor and sends that value as voltage to the LED. The LED will vary accordingly.

```
/*****************Photo Resistor to LED*****/
const int sensorPin = A0; // choose the pin for the Photo resistor
const int ledPin = 9;   // choose the pin for a LED
int lightCal;    // variable for reading the initial state of photo sensor
int lightVal;    // variable for reading the current state photo sensor
void setup()
 {
pinMode(ledPin, OUTPUT);  // declare LED as output
lightCal = analogRead(sensorPin);
}
void loop() {
lightVal =analogRead(sensorPin); // read input value
if(lightVal < lightCal-50) { // check if the input is less than threshold
digitalWrite(9,HIGH);    // turn LED ON}
else { digitalWrite(9, LOW);  // turn LED OFF
  }
}
```

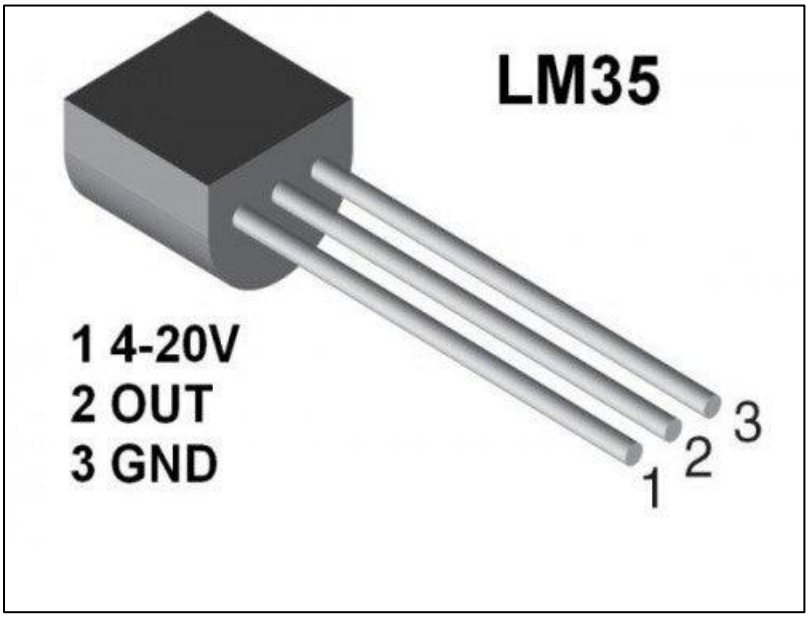
Observation Table:

Sr. no.	Light detected	LED state
1		
2		

Practical 5

Interfacing of temperature sensor LM35 with Arduino

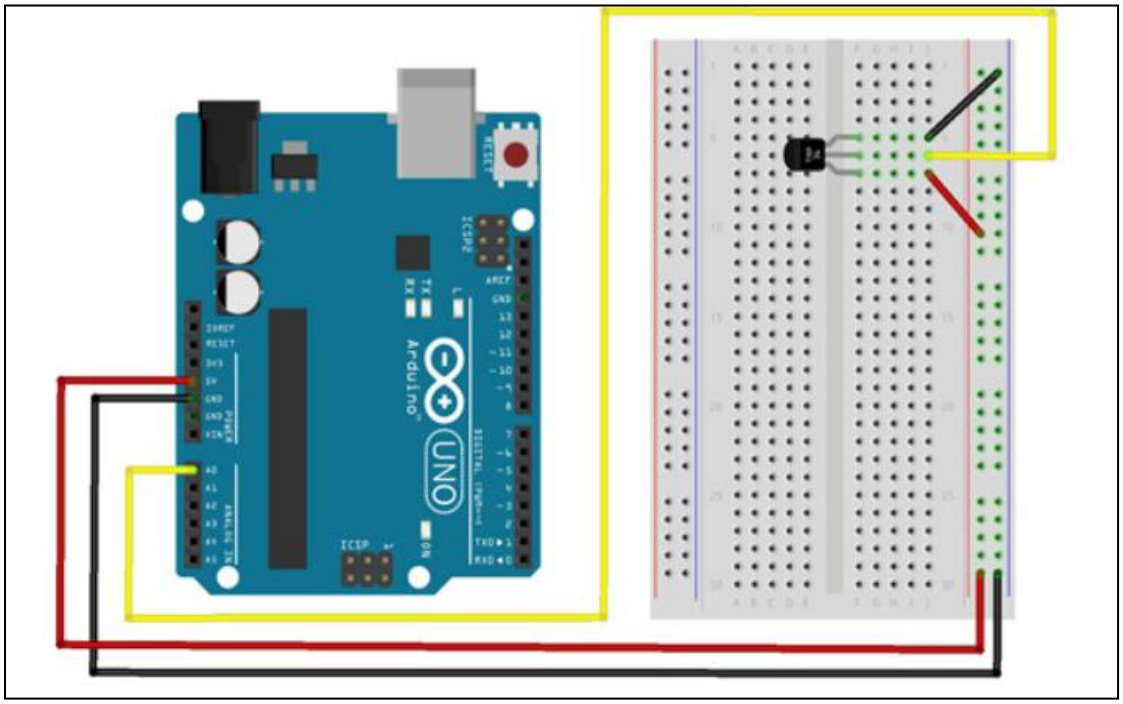
Introduction: The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly proportional to the Centigrade temperature. LM35 is three terminal linear temperature sensors from National semiconductors. It can measure temperature from -55 degree Celsius to +150 degree Celsius. The voltage output of the LM35 increases 10mV per degree Celsius rise in temperature. LM35 can be operated from a 5V supply and the stand by current is less than 60uA. The pin out of LM35 is shown in the figure below.



Hardware Required:

Component Name	Quantity
Arduino UNO	1
Lm35	1
USB Cable	1
Breadboard	1
Jumper wires	Several

Connection Diagram:



Steps of working

1. Insert the temperature sensor into your breadboard and connect its pin1 to the supply.
2. Connect its center pin to the analog pin A0 and the remaining pin3 to GND on the breadboard.
3. Upload the code as given below.
4. Vary the temperature and read the voltage changes.
5. Open the Arduino IDE's serial monitor to see the results.

The Sketch

This sketch works by setting pin A0 as for the temperature sensor. After that the run a loop that continually reads the value from the sensor and sends that value as voltage. The voltage value is between 0–5 volts, when temperature will vary accordingly.

```
/**********File name: LM 35 Temperature Sensor.ino Description: Lit LM35 Temperature Sensor, let Precision Temperature sensor***/
int LM35Pin=A0;
void setup()
{
Serial.begin(9600);}
void loop ()
{int val;
int data;
val = analogRead(LM35Pin);
data= (val*5)/10;
Serial.print("Temp:");
Serial.print(data);
```

```
Serial.println("C");
delay(500);
}
```

Observation Table:

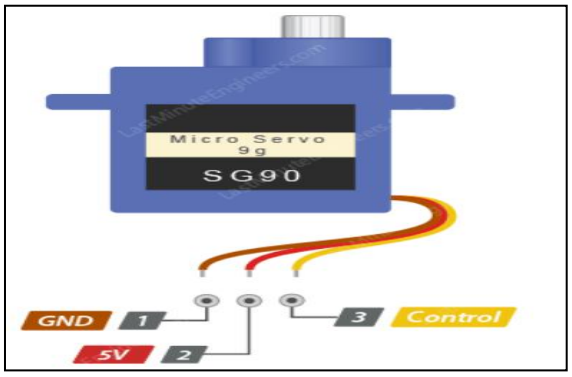
Sr. no.	Voltage	Temperature
1		
2		
3		
4		
5		

Practical 6

Interfacing Servo Motor with the Arduino

Introduction:

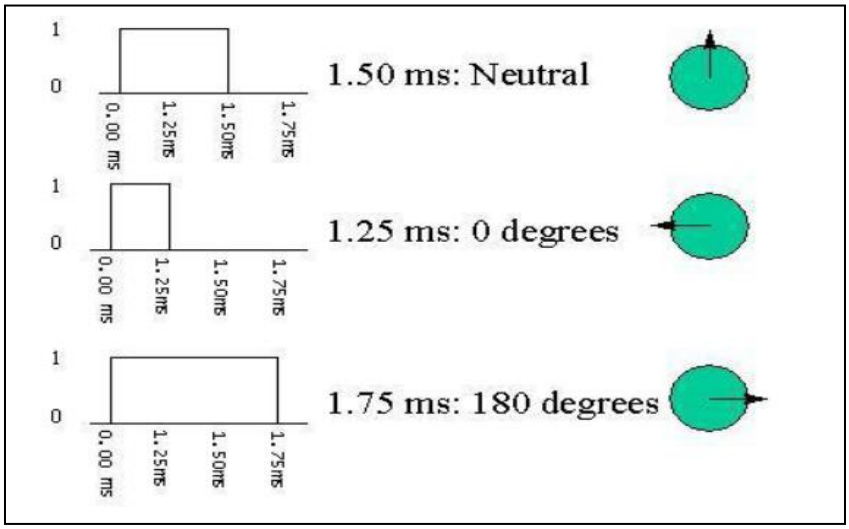
A Servo Motor is a small device that has an output shaft. This shaft can be positioned to specific angular positions by sending the servo a coded signal. As long as the coded signal exists on the input line, the servo will maintain the angular position of the shaft. If the coded signal changes, the angular position of the shaft changes. Servo motors have three terminals – power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino. The ground wire is typically black or brown as shown in figure:



Specifications:

GND	common ground for both the motor and logic.
5V	positive voltage that powers the servo.
Control	Input for the control system.

The control wire is used to communicate the angle. The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse Coded Modulation. The servo expects to see a pulse every 20 milliseconds (.02 seconds). The length of the pulse will determine how far the motor turns. A 1.5 millisecond pulse, for example, will make the motor turn to the 90-degree position (often called as the neutral position). If the pulse is shorter than 1.5 milliseconds, then the motor will turn the shaft closer to 0 degrees. If the pulse is longer than 1.5 milliseconds, the shaft turns closer to 180 degrees.

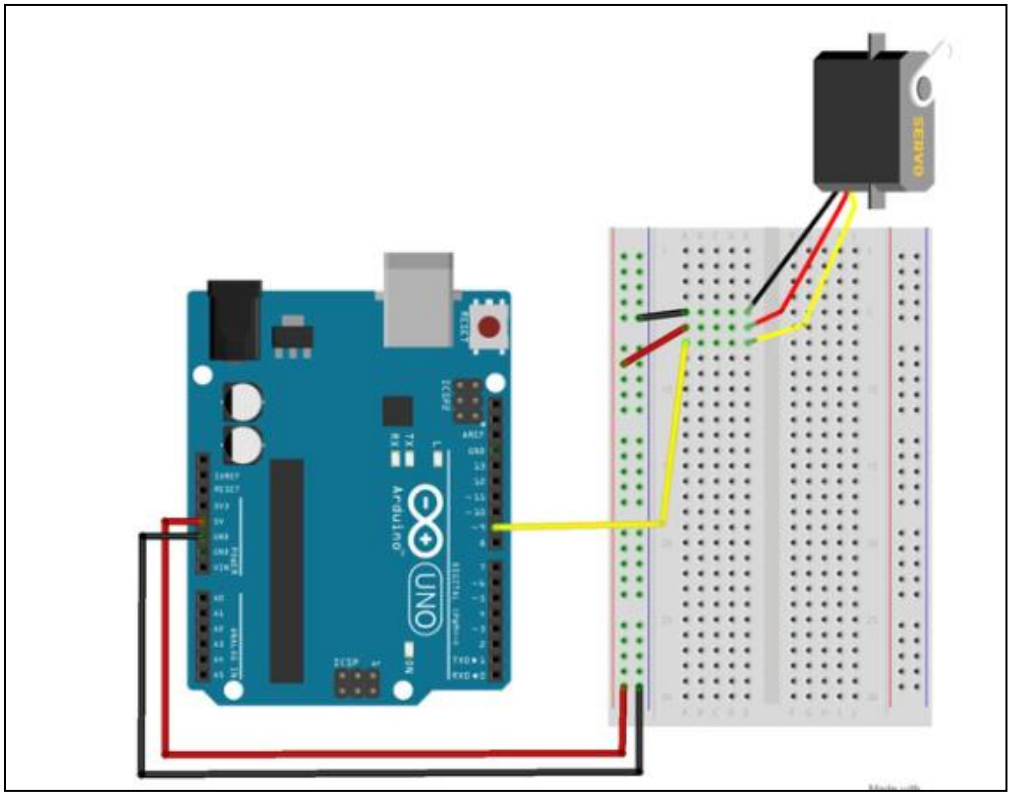


Hardware Required:

Component Name	Quantity
Arduino UNO	1
Servo motor	1
USB Cable	1

Breadboard	1
Jumper wires	several

Connection Diagram:



Steps of working

1. The servo motor has a female connector with three pins. The darkest or even black one is usually the ground. Connect this to the Arduino GND.
2. Connect the power cable that in all standards should be red to 5V on the Arduino.
3. Connect the remaining line on the servo connector to a digital pin on the Arduino.

4. Upload the code
5. Observe the position of the shaft.

The Sketch

This sketch works by setting pin D9 as for the control of servo motor. After that the run a loop that continually increment the value of the index of rotation angle and sends that value as voltage to the D9. The voltage value is between 0–5 volts, and the rotation angle of the servo motor will vary accordingly.

```
/******** Servo Motor Rotation******/
#include<Servo.h>
Servo myservo;
int pos=0;
void setup()
{
  // put your setup code here, to run once:
myservo.attach(7);}
void loop() {
  // put your main code here, to run repeatedly:
  for(pos=0;pos<=180;pos++)
  {
myservo.write(pos);
delay (15);
}
 delay (1000);
for (pos=180; pos>=0;pos--)
{
myservo.write (pos);
```

```
delay (15);
}
delay(1000);
}
```

Observation Table:

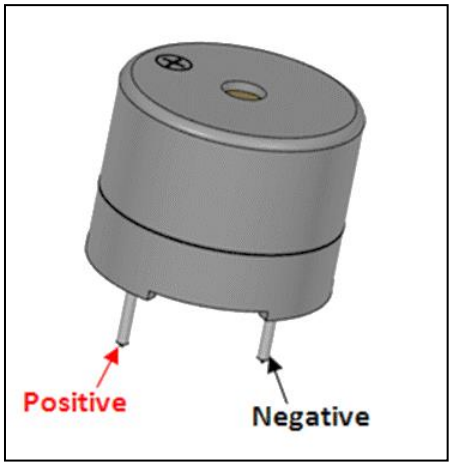
Sr. no.	Voltage	Position of Shaft
1		
2		
3		
4		
5		

Practical 7

Interfacing of the Active Buzzer with Arduino.

Introduction:

A piezo buzzer is a type of electronic device that's used to produce beeps and tones. The working principle of the device is piezoelectric effect. The main component of this device is a piezo crystal, which is a special material that changes shape when a voltage applied to it. The active buzzer will only generate sound when it will be electrified. It generates sound at only one frequency. This buzzer operates at an audible frequency of about 2 KHz.



Specifications:

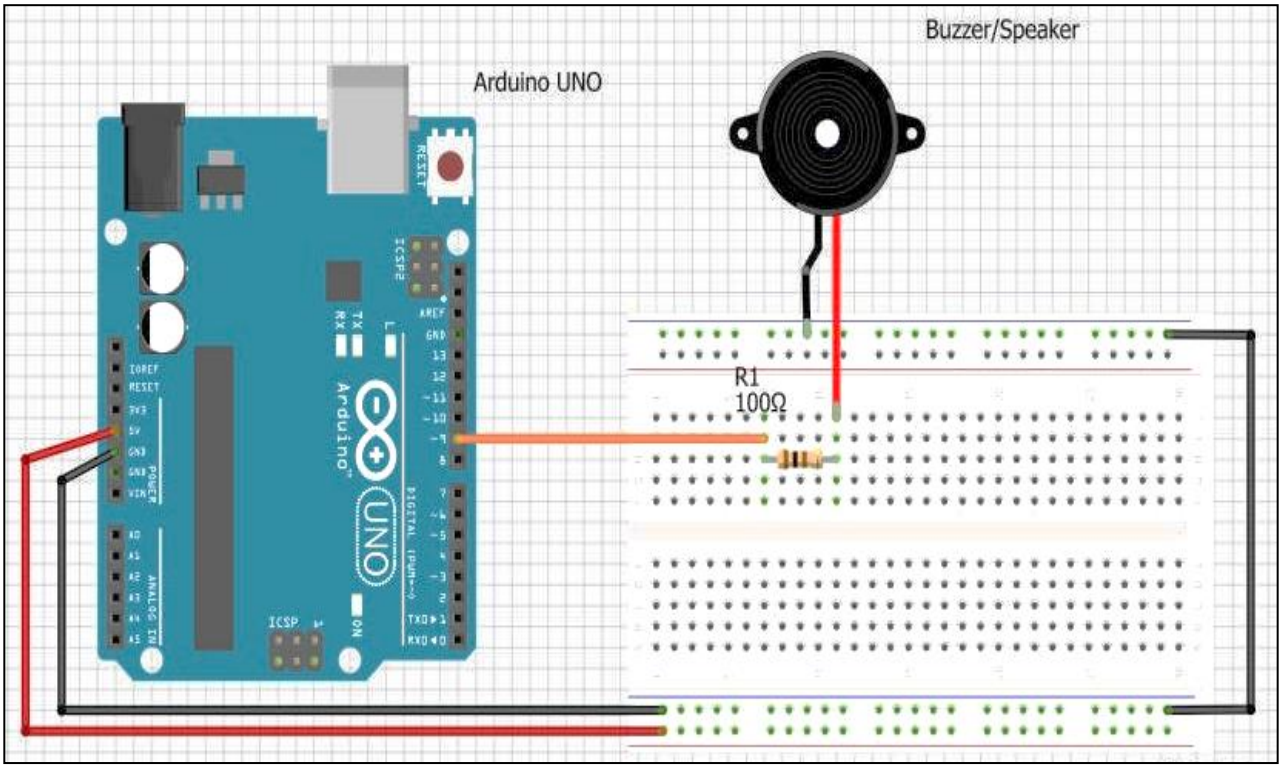
Specification	Range
VoltageRange	3.3-5V
Frequency	2KHz

Pin Name	Description
Positive	Identified by (+) symbol or longer terminal lead. Can be powered by 6V DC
Negative	Identified by short terminal lead. Typically connected to the ground of the circuit

Hardware Required:

Component Name	Quantity
Arduino UNO	1
Buzzer / piezo speaker	1
220-ohm resistors	1
USB Cable	1
Breadboard	1
Jumper wires	several

Connection Diagram:



Steps of working:

Connect the Supply wire (RED) of the buzzer to the Digital Pin 9 of the Arduino through a 100-ohm resistor.

Connect the Ground wire (BLACK) of the buzzer to any Ground Pin on the Arduino.

Upload the code

Observe the changes in the pitch and volume of the buzzer.

Sketch:

This sketch works by setting pin D9 as for the control the buzzer. After that the run a loop that continually sends that value as voltage high or low to the D9 using the function digitalWrite( ). The voltage value and the tone generated from the buzzer will vary accordingly.

```
/****Musical buzzer****/
int buzzer = 9;                    //the pin of the active buzzer
void setup()
{
pinMode (buzzer,OUTPUT);           //initialize the buzzer pin as an output
}
void loop(){
unsigned char i;
while(1){
//output a frequency
for(i=0;i<80;i++)
{
digitalWrite(buzzer,HIGH);
delay(1);                                  //wait for 1ms
digitalWrite(buzzer,LOW);
delay(1);                                  //wait for 1ms
}
//output another frequency
for(i=0;i<100;i++){
digitalWrite(buzzer,HIGH);
delay(2);                                  //wait for 2ms
```

```
digitalWrite(buzzer,LOW);
delay(2);                                   //wait for 2ms
}
}
}
```

Observation:

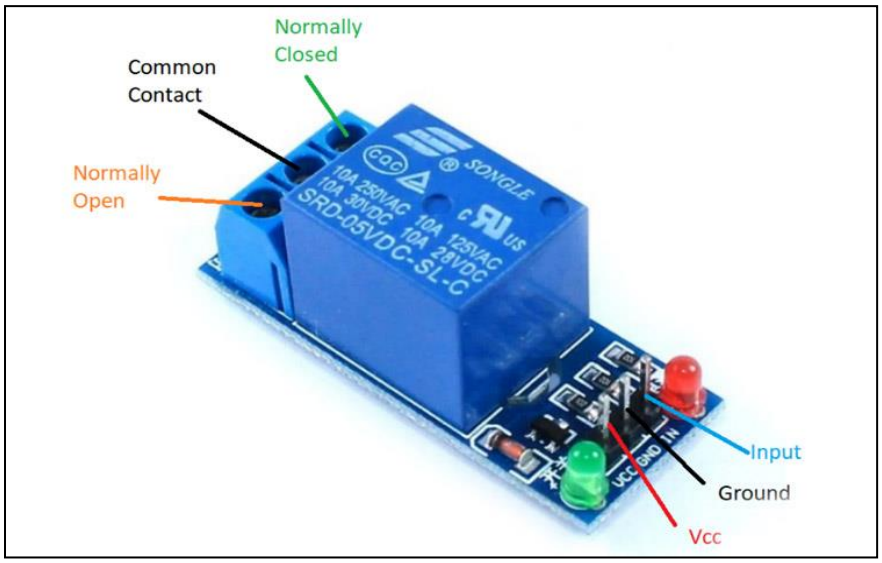
Sr. no.	Change the value	Frequency of tone
1		
2		
3		

Practical 8

Interfacing of the Relay with Arduino.

Introduction:

Relay is an electromagnetic switch, which is controlled by small current, and used to switch ON and OFF relatively much larger current. Means by applying small current we can switch ON the relay which allows much larger current to flow.

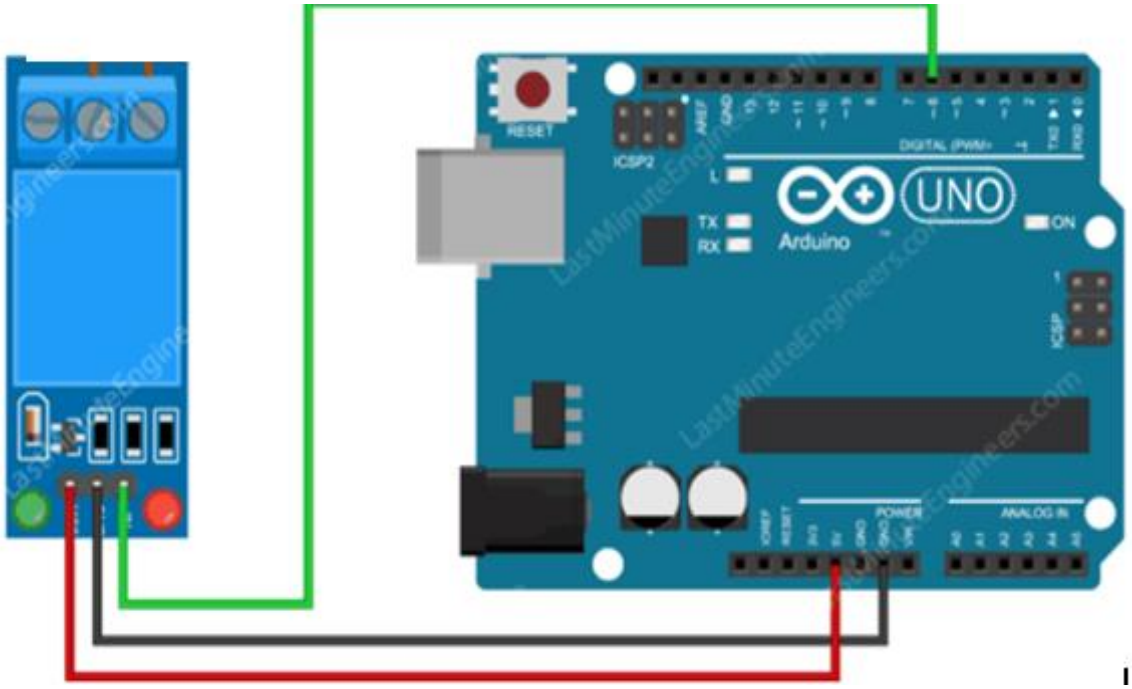


Hardware Required:

Component Name	Quantity
Arduino UNO	1
5V Relay	1

USB Cable	1
Breadboard	1
Jumper wires	several

Connection Diagram:



Steps of working:

1. The relay module connected with three pins. We will connect the relay module with Arduino in the normally open state. The black one of relay is usually the ground. Connect this to the Arduino GND.
2. Connect the red wire of relay module to 5V of the Arduino.
3. Connect the signal pin of relay module to a digital pin 6 of the Arduino.
4. Upload the code

5. Observe the clicking sound of the relay that states the ON and OFF constantly.

Sketch:

This sketch works by setting 5V supply pin of Arduino as for the control of relay module. After that the run a loop that continually sends that value as voltage to the D6 with the delay given.

```
// Arduino Relay Control Code
Int relayPin=6;
#define interval 2000
void setup() {
  pinMode(relayPin, OUTPUT);
}
void loop()
{
   digitalWrite(relayPin, HIGH);
   delay(interval);
   digitalWrite(relayPin, LOW);
   delay(interval);
}
```

Observations:

Sr. No.	Delay	Relay Status
1		
2		

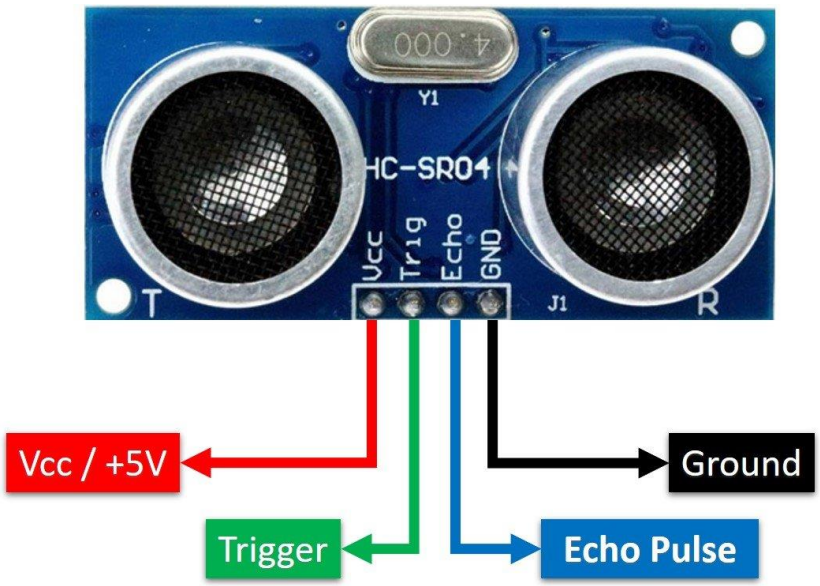
Practical 9

Building Intrusion Detection System with Arduino and Ultrasonic Sensor

Introduction:

An intrusion detection system (IDS) is a device or software application that monitors a network or systems for malicious activity

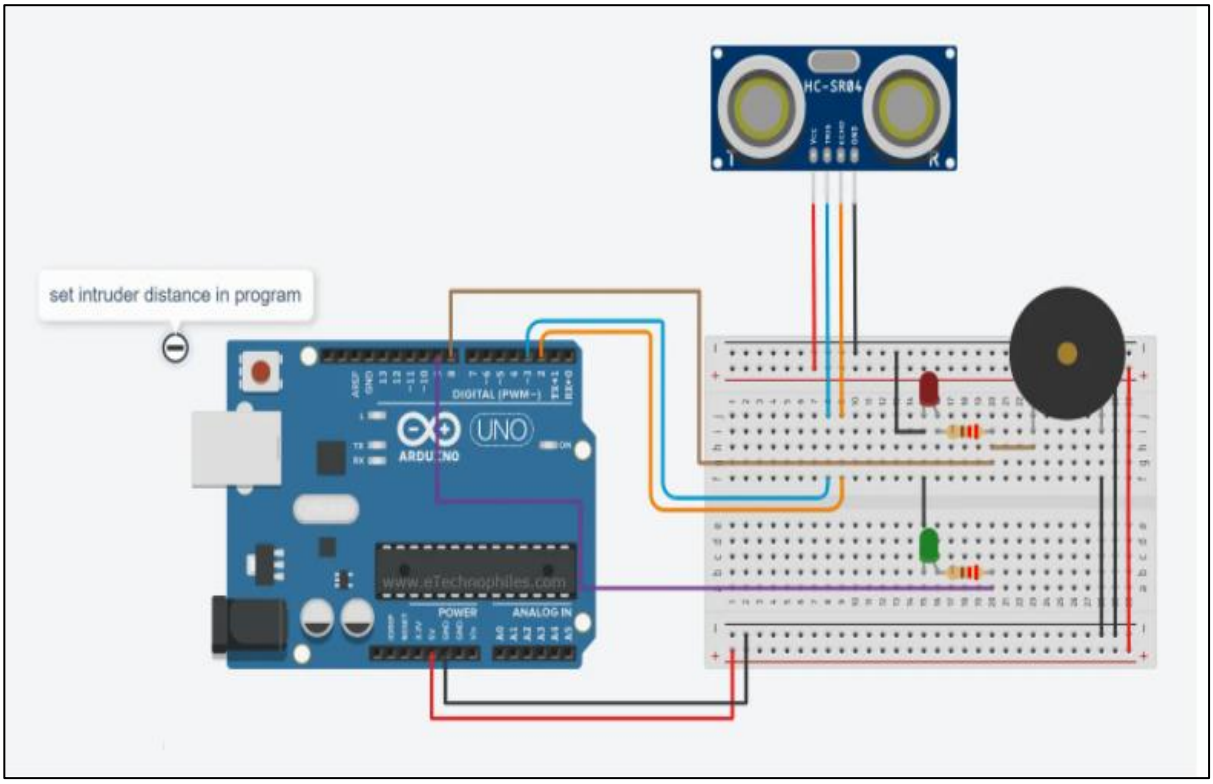
Ultrasonic Sensors: The HC-SR04 ultrasonic sensor uses SONAR to determine the distance of an object just like the bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package from 2 cm to 400 cm or 1" to 13 feet. It comes complete with ultrasonic transmitter and receiver module. The ultrasonic sensor uses the reflection of sound in obtaining the time between the wave sent and the wave received. It usually sent a wave at the transmission terminal and receives the reflected waves. The time taken is used together with the normal speed of sound in air (340ms-1) to determine the distance between the sensor and the obstacle. The Ultrasonic sensor is used here for the intruder detection. The sound via a buzzer occurs when an object comes near to the sensor. The distance to which the sensor will respond can be easily adjusted in the program.



Hardware Required:

Component Name	Quantity
Arduino UNO	1
Red LED	1
Green LED	1
HC-SR04 Ultrasonic Sensor	1
Buzzer	1
USB Cable	1
Breadboard	1
Jumper wires	several

Connection Diagram:



Steps of working

1. Insert the Ultrasonic sensor into your breadboard and connect its Echo pin to the digital pin 2 and the Trigger pin to digital pin 3 of the Arduino.
2. Insert the RED and Green LED into the breadboard. Attach the positive leg (the longer leg) of red LED to signal pin of the Buzzer via the 220-ohm resistor, and the negative leg to GND. The green LED is connected to digital pin 8 of the Arduino.
3. Upload the code.
4. Observe the LEDs and take some object in front of ultrasonic sensor.
5. Observe the changes in the LED and buzzer sound.

The Sketch

This sketch works by setting pin 2 as for the ultrasonic sensors and pin 8, pin9 & pin 10 as an OUTPUT to power the LEDs and buzzer. After that the run a loop that continually reads the value from the echo pin and sends that value as voltage to the LEDs. The color of the LED which glows will vary accordingly to the detection of object in the given range.

```
/**********Intrusion Detection******/
#define echo 2
 #define trig 3
 #define outA 8     // Red LED
 #define outB 9     // Green LED
 #define outC 10   // Buzzer
 float duration;      // time taken by the pulse to return back
 float distance;       // one way distance travelled by the pulse
const int intruderDistance = 10; // the minimum distance up to which the sensor is able to sense any object
 void setup() {
   pinMode(trig, OUTPUT);
   pinMode(echo, INPUT);
    pinMode(outA, OUTPUT);
   digitalWrite(outA, LOW);
   pinMode(outB, OUTPUT);
   digitalWrite(outB, LOW);
pinMode(outC, OUTPUT);
   digitalWrite(outC, LOW);
   Serial.begin(9600);
```

```
 }
void loop() {
time_Measurement();
distance = (float)duration * (0.0343) / 2;
                  // calculate the one way distance travelled by the pulse
Serial.println(distance);
alarm_condition();
 }
   void time_Measurement()
 {              // function to measure the time taken by the pulse to return back
   digitalWrite(trig, LOW);
   delayMicroseconds(2);
   digitalWrite(trig, HIGH);
   delayMicroseconds(10);
   digitalWrite(trig, LOW);
  duration = pulseIn(echo, HIGH);
 }
 void alarm_condition()
 {              //function to execute the output commands based on the sensor
input
   if(distance<=intruderDistance)
   {
digitalWrite(outA,HIGH);
 digitalWrite(outB,LOW);
 analogWrite(outC,200);}
   else
   {
```

```
digitalWrite(outA,LOW);
digitalWrite (outB, HIGH);
analogWrite (outC,0);
}
}
```

Observation Table:

Sr no.	Object Detected	LED	Buzzer
1			
2			

Practical 10

Directional Control of the DC motor using Arduino

Introduction:

A DC motor (Direct Current motor) is the most common type of motor. DC motors normally have just two leads, one positive and one negative. If you connect these two leads directly to a battery, the motor will rotate. If you switch the leads, the motor will rotate in the opposite direction.

Specification

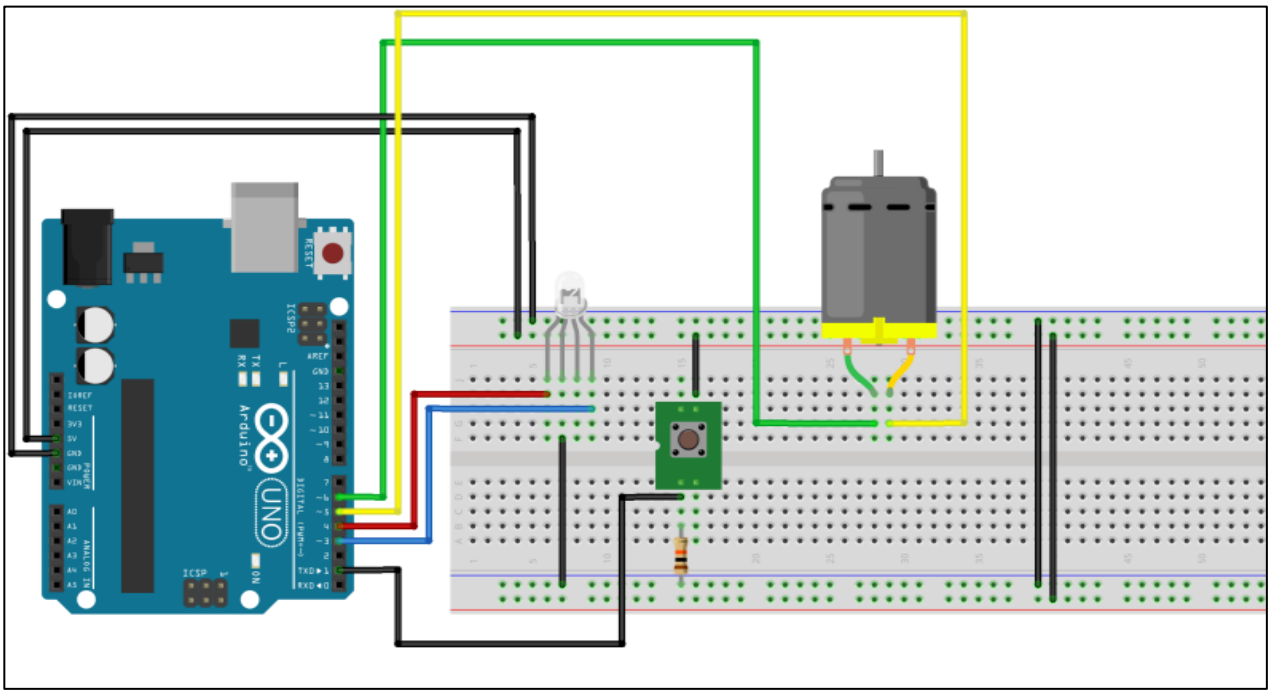
Pin	Description
GND	common ground for both the motor and logic
5V	positive voltage that powers the servo
Control	Input for the control system.

The control wire is used to communicate the angle. The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse Coded Modulation. The servo expects to see a pulse every 20 milliseconds (.02 seconds). The length of the pulse will determine how far the motor turns. A 1.5 millisecond pulse, for example, will make the motor turn to the 90-degree position (often called as the neutral position). If the pulse is shorter than 1.5 milliseconds, then the motor will turn the shaft closer to 0 degrees. If the pulse is longer than 1.5 milliseconds, the shaft turns closer to 180 degrees.

Hardware Required:

Component Name	Quantity
Arduino UNO	1
DC motor	1
RGB LED	1
Push button	1
10k-ohm resistor	1
USB Cable	1
Breadboard	1
Jumper wires	several

Connection diagram:



Steps of working

1. The servo motor has a female connector with three pins. The darkest or even black one is usually the ground. Connect this to the Arduino GND.
2. Connect the power cable that in all standards should be red to 5V on the Arduino.
3. Connect the remaining line on the servo connector to a digital pin on the Arduino.
4. Upload the code
5. Observe the position of the shaft.

The Sketch

This sketch works by setting pin A2 as for the potentiometer and pin 9 as an OUTPUT to power the LED. After that the run a loop that continually reads the value from the potentiometer and sends that value as voltage to the LED.

The voltage value is between 0–5 volts, and the brightness of the LED will vary accordingly.

```
/******** DC Motor Direction control by RGB******/
const int inputPin=1;
const int blue=3;
const int red=4;
const int motorPin1=5,motorPin2=6;
int dir=LOW;
int prevState=0,currentState=0;
void setup()
{
// put your setup code here, to run once:
pinMode(inputPin,INPUT);
pinMode(motorPin1,OUTPUT);
pinMode(motorPin2,OUTPUT);
pinMode(blue,OUTPUT);
pinMode(red,OUTPUT);
}
void loop()
{
// put your main code here, to run repeatedly:
currentState=digitalRead(inputPin);
if(currentState!=prevState)
{
  if(currentState==HIGH)
  { dir=!dir;
```

```
  }
}
prevState=currentState;
if(dir==HIGH)
{
  digitalWrite(motorPin1,HIGH);
  digitalWrite(motorPin2,HIGH);
  digitalWrite(blue,LOW);
digitalWrite(red,HIGH);
}
}
```

Observation Table:

Sr no.	Voltage	Position of Shaft
1		
2		
3		
4		
5		