Experiment 1: Traffic Light Simulation using Arduino Uno

Aim: To design and develop a smart Traffic light controller using Arduino Uno

Components Required:

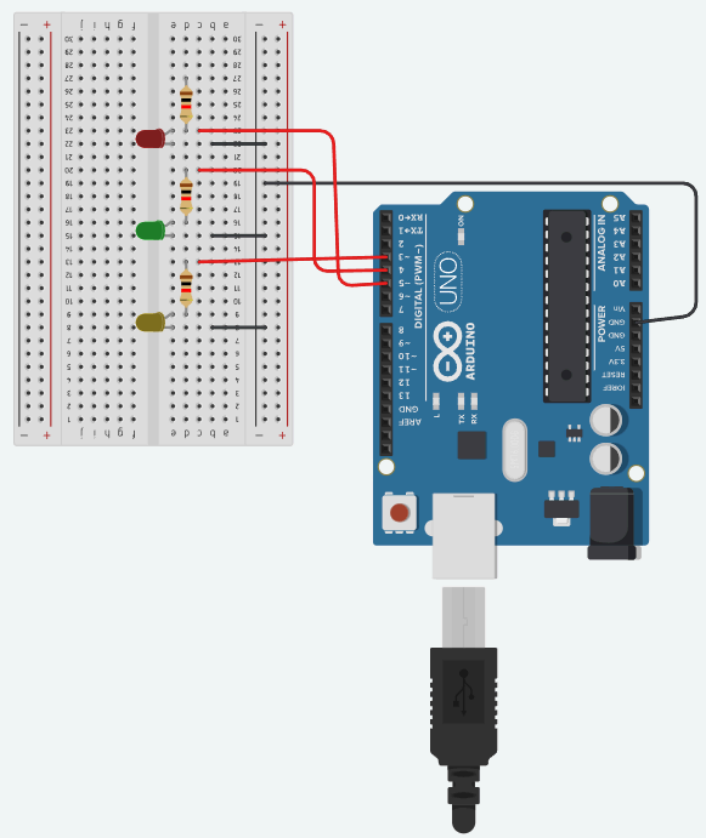
1. Arduino Uno Board
2. 3 LEDs
3. 3 Resistors
4. Jumper Wires

Procedure to Connect the Components:

1. Fix the LEDs on the Breadboard
2. Connect one end of the resistor to the +ve lead of the LED and another side to digital pin no. 3 on the Arduino board. Repeat the same for the other two LEDs and connect the other end of the Resistors to pin no. 4 and 5 respectively
3. Connect the -ve lead of the LEDs to the negative terminal on the Arduino board
4. Connect the -ve terminal of the Breadboard to the ground (GND) pin on Arduino board

Circuit Connection:

Initially all the LEDs are turned Off



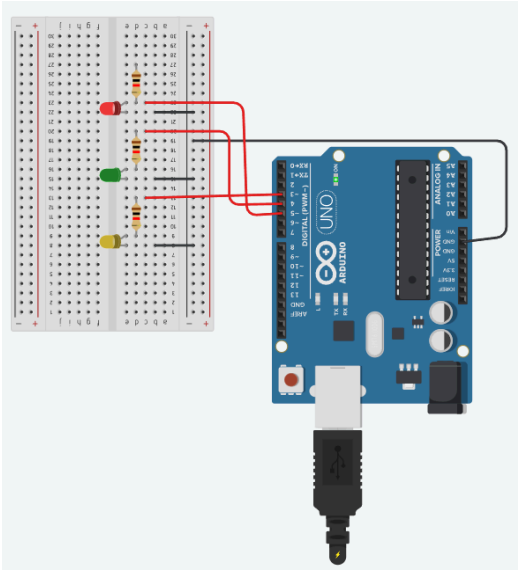
Code:

```
const int yled = 3;
const int gled = 4;
const int rled = 5;

void setup()
{
	//Set the pin Modes of LEDs to OUTPUT
	pinMode(yled, OUTPUT);
	pinMode(gled, OUTPUT);
	pinMode(rled, OUTPUT);

	//Initially Turn Off all the LEDs
	digitalWrite(yled, LOW);
	digitalWrite(gled, LOW);
	digitalWrite(rled, LOW);
}

void loop()
{
	//Turn on Red LED and Turn Off Green and Yellow LEDs
	digitalWrite(rled, HIGH);
	digitalWrite(gled, LOW);
	digitalWrite(yled, LOW);
	delay(5000); // Wait for 5000 millisecond(s)
	Serial.println("RED LED ON");

	//Turn on Yellow LED and Turn Off Red and Green LEDs
	digitalWrite(rled, LOW);
	digitalWrite(yled, HIGH);
	digitalWrite(gled, LOW);
	delay(5000); // Wait for 5000 millisecond(s)
	Serial.println("YELLOW LED ON");

	//Turn on Green LED and Turn Off Red and Yellow LEDs
	digitalWrite(rled, LOW);
	digitalWrite(yled, LOW);
	digitalWrite(gled, HIGH);
	delay(5000); // Wait for 5000 millisecond(s)
```

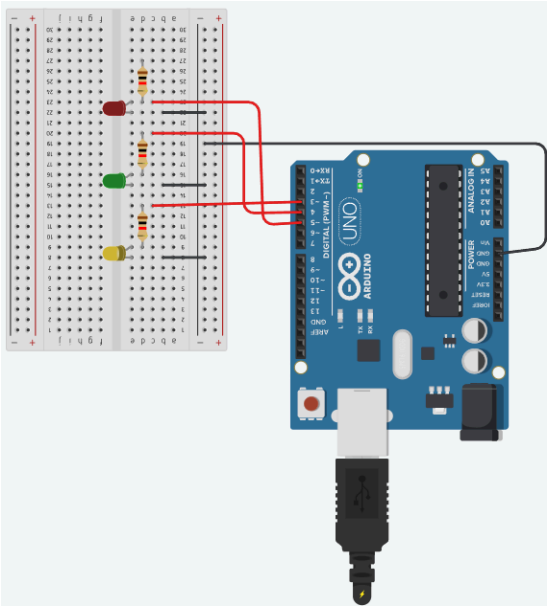
```
        Serial.println("GREEN LED ON");
}
```

Output:

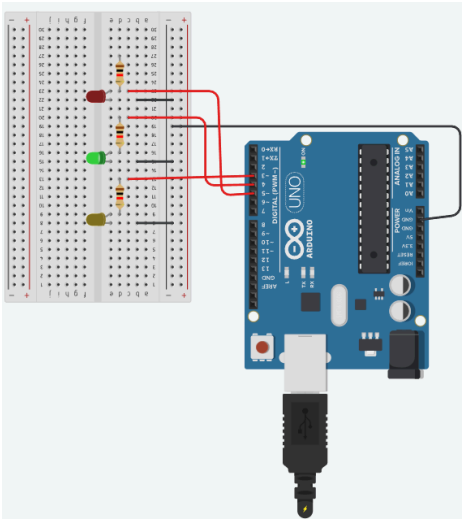
1. **Red LED glows for 5 seconds**



2. **Yellow LED glows for next 5 seconds**



3. **Green LED glows for next 5 seconds**



Result:

Thus the Simulation of Traffic Light has been done successfully using Arduino Uno.

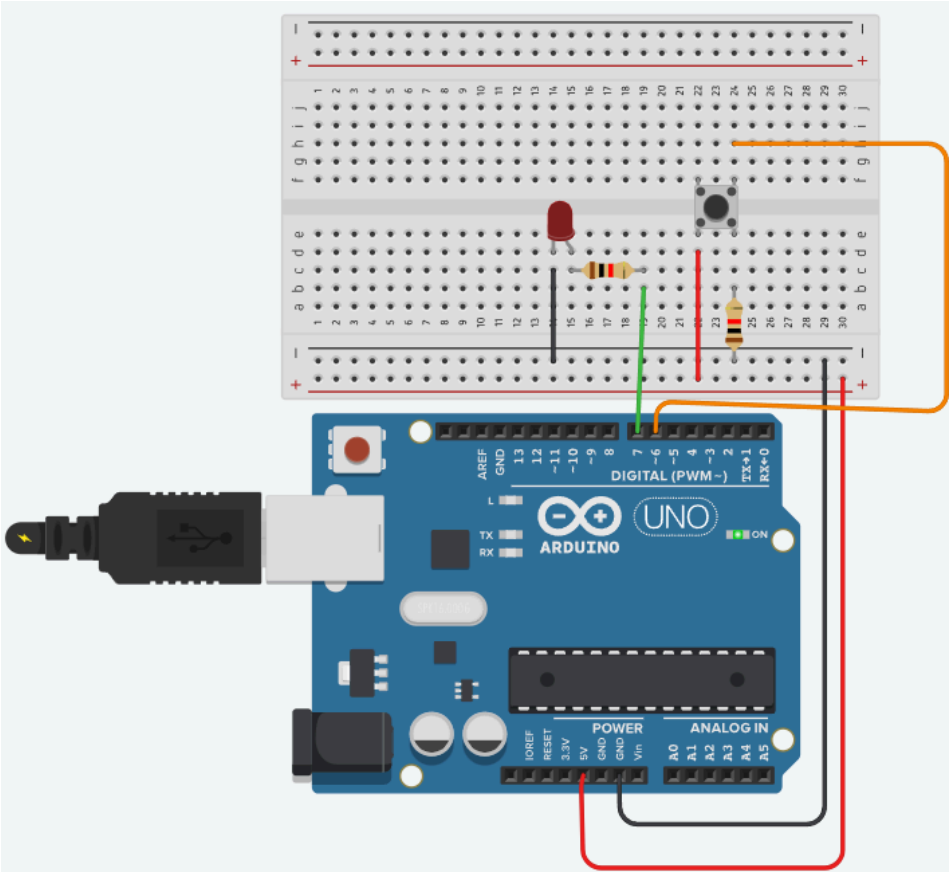
Experiment 2a: LED ON/OFF with Push Button

Aim: To control an LED using Push Button

Components Required:

1. An LED
2. A Push Button
3. 220 Ohms Resistor
4. 10 Kilo Ohms Resistor
5. Jumper Wires

Circuit Diagram:



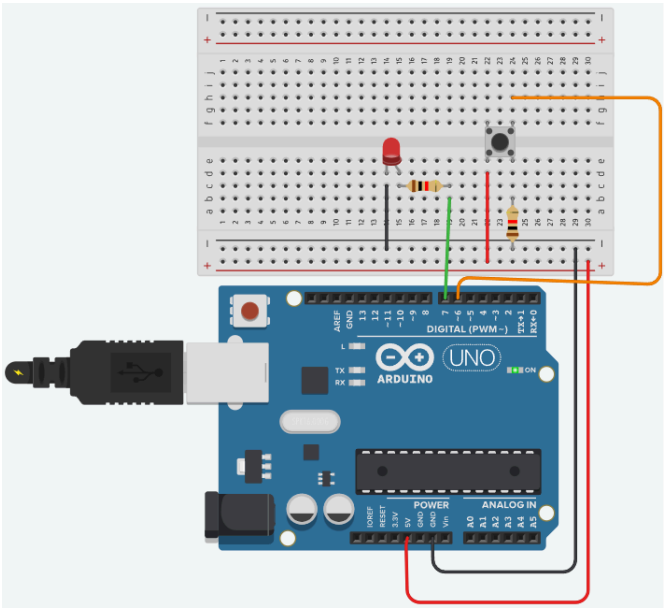
Code:

```
const int led = 7;
const int btn = 6;
void setup()
{
        pinMode(led, OUTPUT);
```

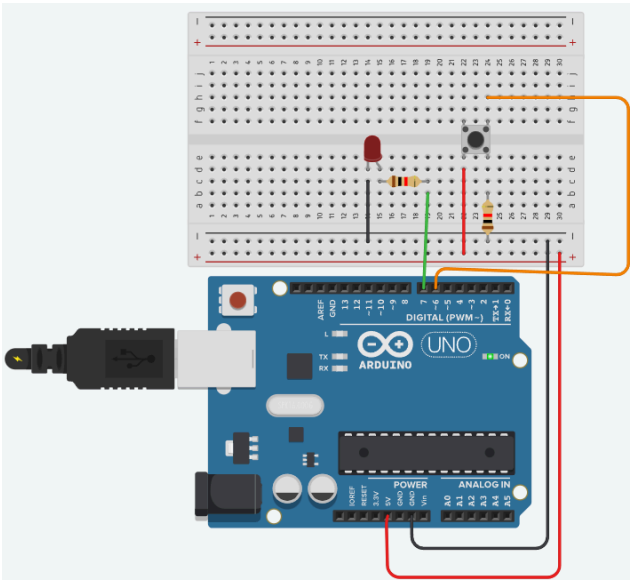
```
        pinMode(btn, INPUT);
        digitalWrite(led, LOW);
}


void loop()
{
        int state = digitalRead(btn);
        if(state == HIGH){
         digitalWrite(led,HIGH);
        }
        else{
         digitalWrite(led,LOW);
        }
 }
```

Output:

1. **After pressing the button**



2. **After releasing the button**



Result:

Thus an LED has been turned ON/OFF using Arduino Uno with a Push Button.

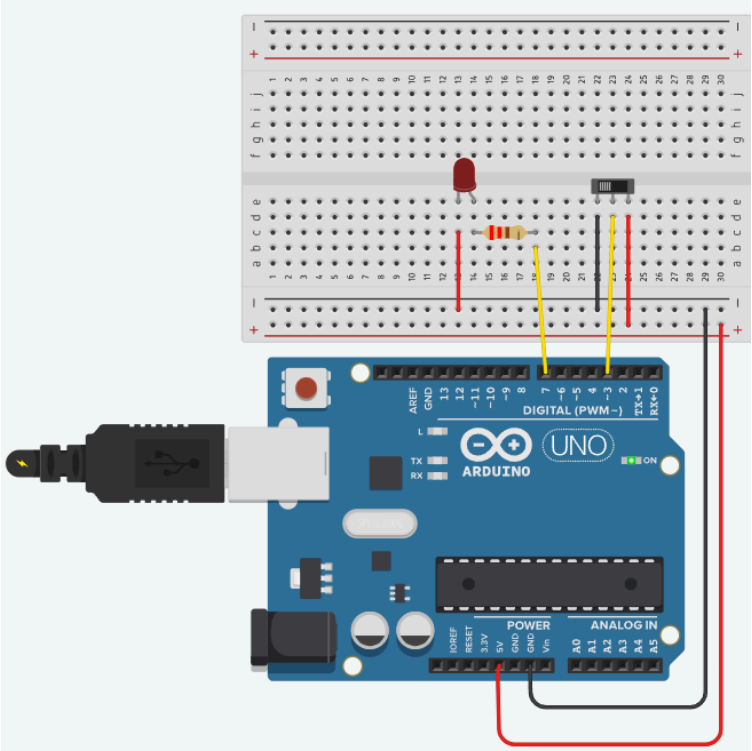
Experiment 2b: Turn ON/OFF an LED using Switch

Aim: To control an LED using Switch

Components Required:

1. An LED
2. A Switch
3. A 220 Ohms Resistor
4. Jumper Wires

Circuit Diagram:

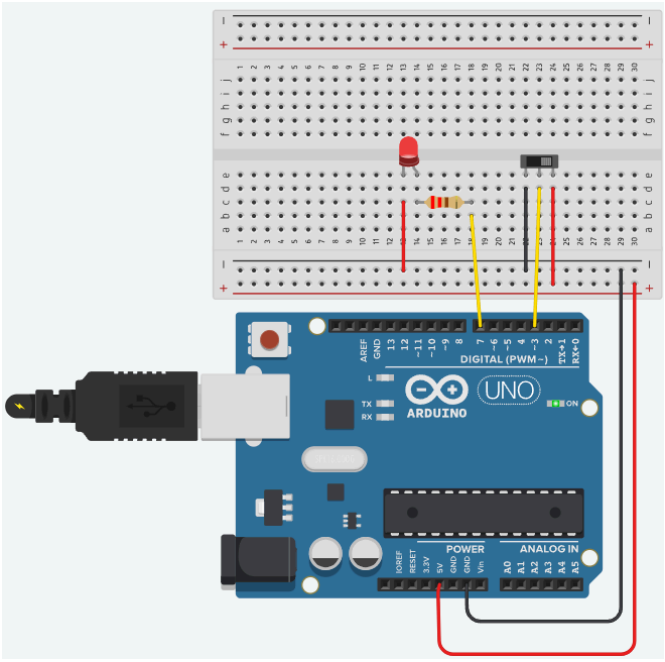
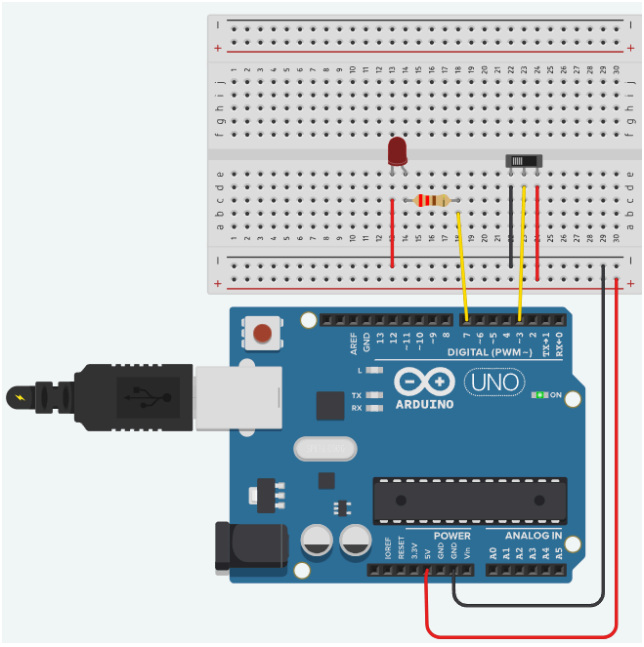


Code:

```
const int led = 7;
const int sw = 3;
void setup()
{
        pinMode(sw, INPUT);
        pinMode(led, OUTPUT);
}

void loop()
{
        if (digitalRead(sw) == 1) {
```

```
        digitalWrite(led, HIGH);
      } else {
        digitalWrite(led, LOW);
      }
  }
```

Output:

1. When the switch is in ON state, LED glows	2. When the switch is in OFF state, LED is not glowing
	

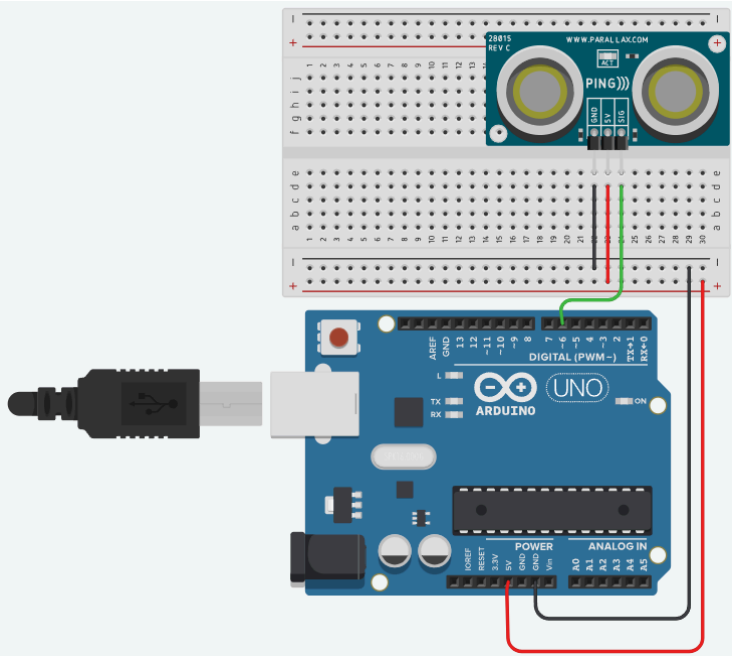
Result:

Thus an LED has been turned ON/OFF using Arduino Uno with a Switch.

Experiment 3: Object Detection using Ultrasonic Sensor and Arduino Uno

Aim: To develop an IoT-Based object detection system using Arduino Uno

a)

Components Required:	**Circuit Diagram:**
1. Breadboard 2. Arduino Uno Board 3. Ultrasonic Sensor 4. Jumper Wires	

Code:

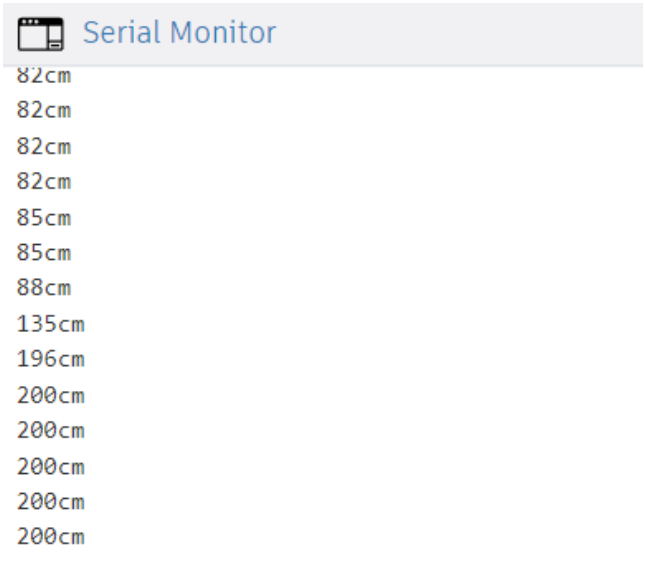
```
const int pingPin = 6;
const int led = 7;
void setup() {
        pinMode(led, OUTPUT);
        // initialize serial communication:
        Serial.begin(9600);
}

void loop() {
          // establish variables for duration of the ping, and the distance result
          // in inches and centimeters:
          long time, dist;
          // The PING is triggered by a HIGH pulse of 2 or more microseconds.
          // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
          pinMode(pingPin, OUTPUT);
          digitalWrite(pingPin, LOW);
          delayMicroseconds(2);
```

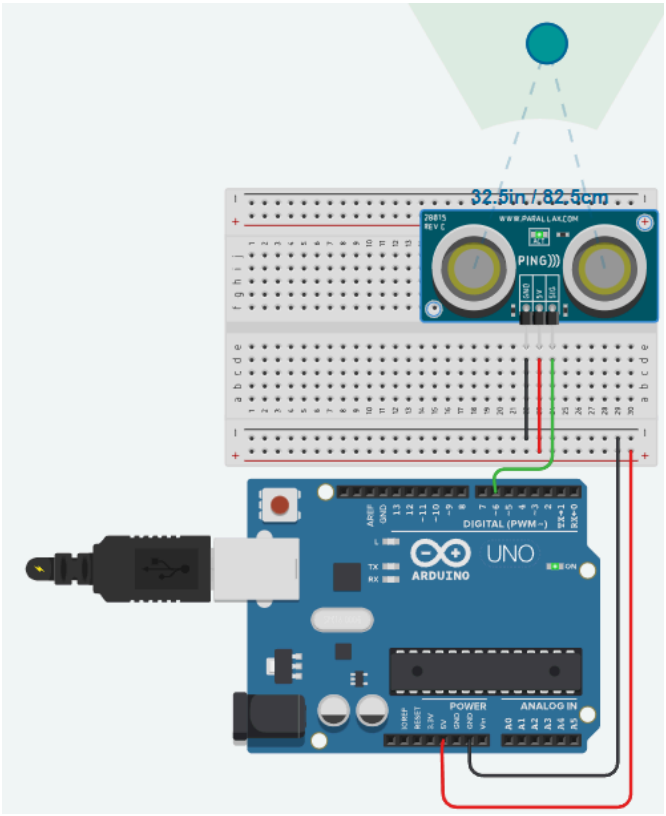
```
    digitalWrite(pingPin, HIGH);
    delayMicroseconds(5);
    digitalWrite(pingPin, LOW);

    // The same pin is used to read the signal from the PING))): a HIGH pulse
    // whose duration is the time (in microseconds) from the sending of the ping
    // to the reception of its echo off of an object.
    pinMode(pingPin, INPUT);
    time = pulseIn(pingPin, HIGH);

    // convert the time into a distance. 29 microseconds = 1 cm
    dist = (time/29)/2;
    Serial.print(dist);
    Serial.print("cm");
    Serial.println();

    delay(100);
}
```

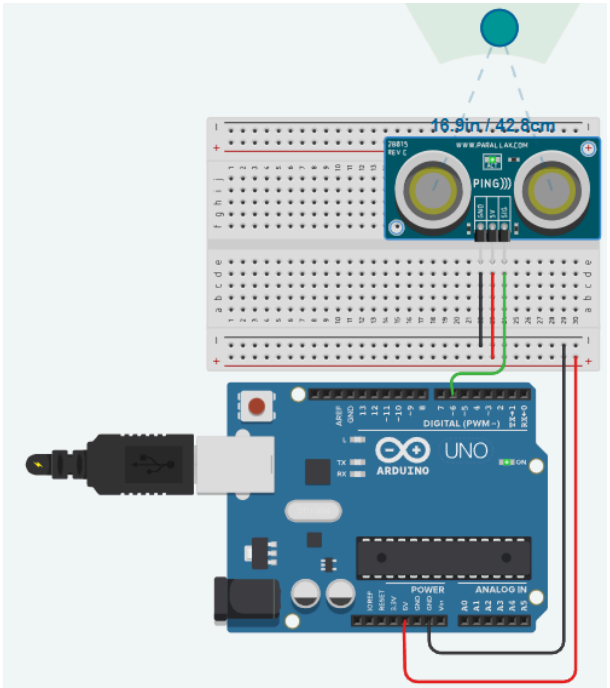
Output:

2. **When the switch is in ON state, LED glows**

2. When the switch is in OFF state, LED is not glowing



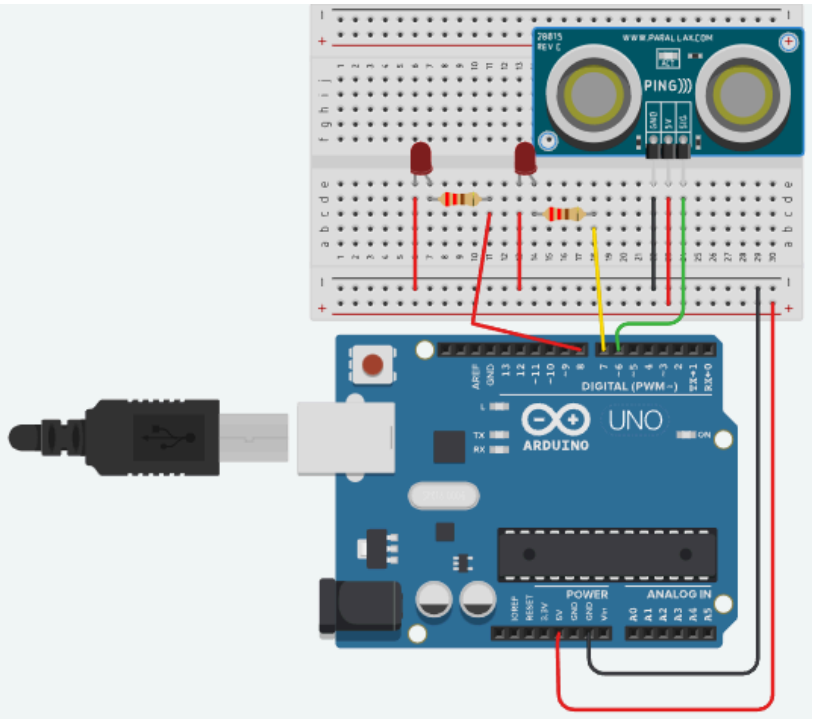




Serial Monitor

```
43cm
43cm
42cm
43cm
43cm
43cm
42cm
43cm
43cm
43cm
43cm
42cm
43cm
43cm
43cm
```

b)

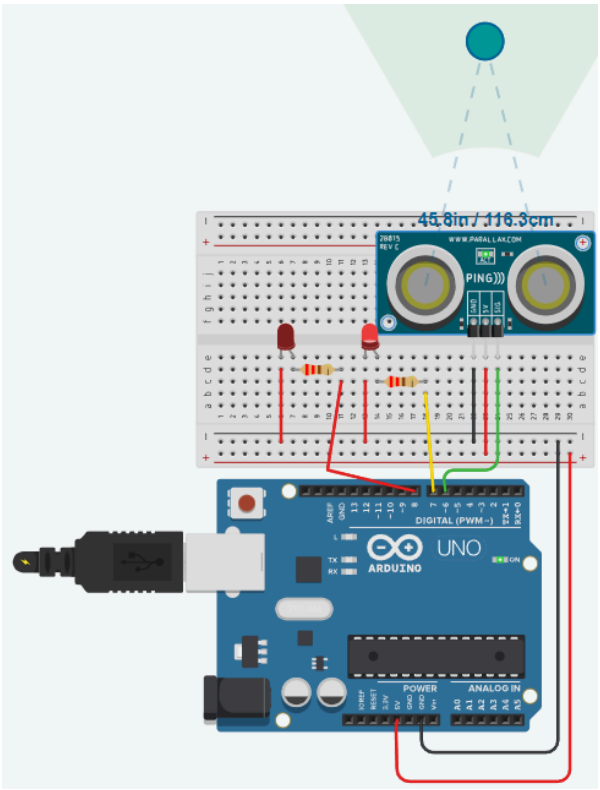
Components Required:	**Circuit Diagram:**
1. Breadboard 2. Arduino Uno Board 3. Ultrasonic Sensor 4. 2 LEDs 5. Jumper Wires	

Code:

```
const int pingPin = 6;
const int led1 = 7;
const int led2 = 8;
int brightness;
void setup() {
        // initialize serial communication:
        Serial.begin(9600);
        pinMode(led1, OUTPUT);
        pinMode(led2, OUTPUT);
        analogWrite(led1,0);
        analogWrite(led2,0);

}

void loop() {
   // establish variables for duration of the ping, and the distance result
   // in inches and centimeters:
```

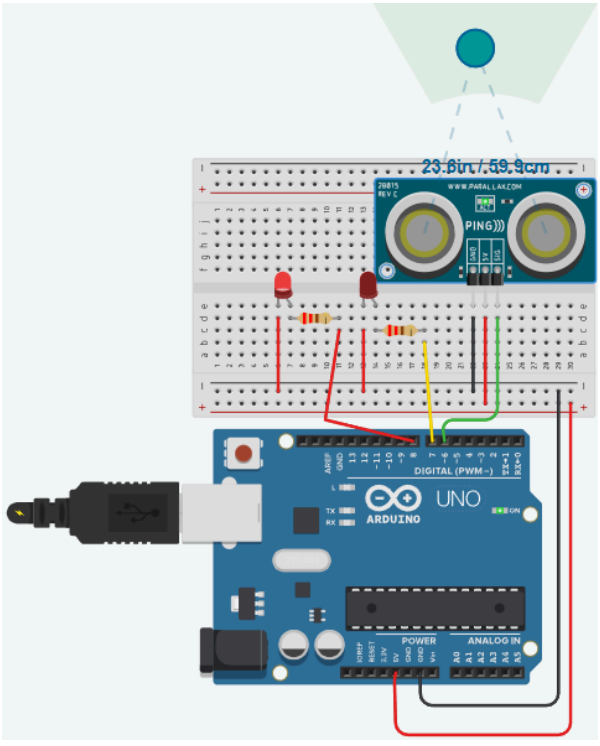
```
  long time, dist;
  // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);

  // The same pin is used to read the signal from the PING))): a HIGH pulse
  // whose duration is the time (in microseconds) from the sending of the ping
  // to the reception of its echo off of an object.
  pinMode(pingPin, INPUT);
  time = pulseIn(pingPin, HIGH);

  // convert the time into a distance

  dist = (time/29)/2;
  Serial.print(dist);
  Serial.print("cm");
  Serial.println();

  if(dist>100 && dist<=125){
      digitalWrite(led1,HIGH);
      digitalWrite(led2,LOW);
    }
    else{
      digitalWrite(led2,HIGH);
      digitalWrite(led1,LOW);
    }
    delay(100);
}
```

Output:



Serial Monitor

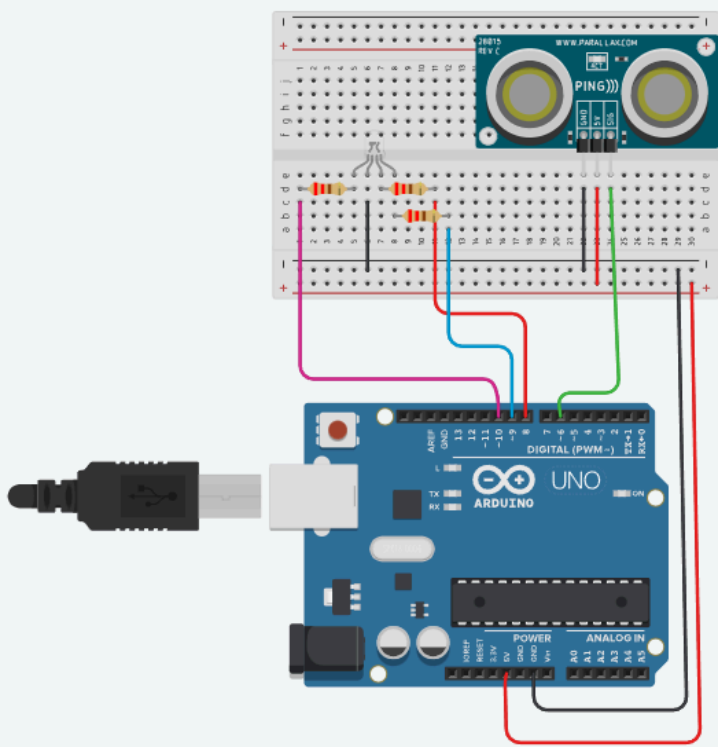
```
99cm
99cm
99cm
99cm
99cm
99cm
99cm
101cm
105cm
108cm
111cm
111cm
111cm
```



Serial Monitor

```
144cm
123cm
116cm
108cm
98cm
96cm
96cm
96cm
96cm
96cm
96cm
96cm
96cm
96cm
```

c)

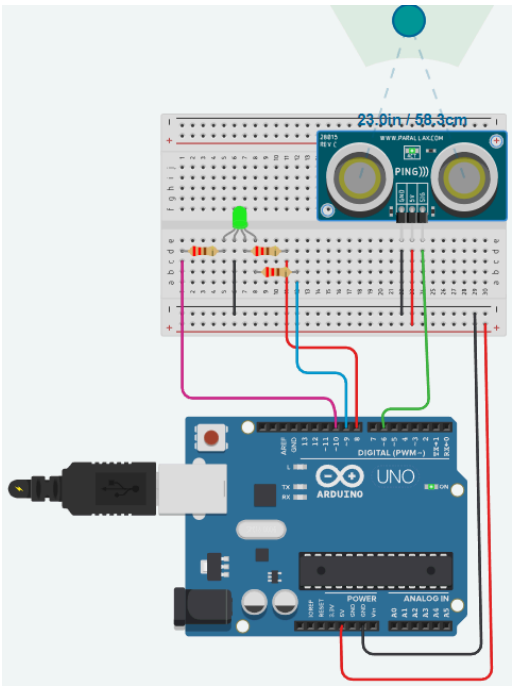
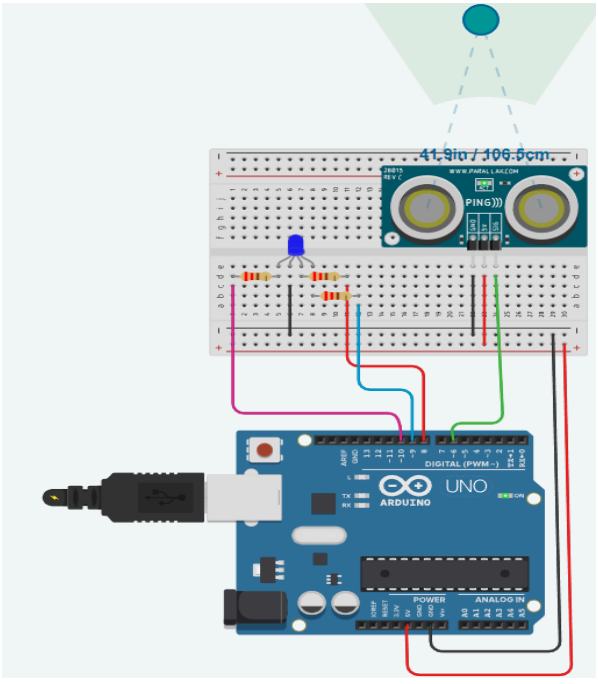
Components Required:	**Circuit Diagram:**
1. Breadboard 2. Arduino Uno Board 3. Ultrasonic Sensor (3 pins) 4. RGB LED 5. Jumper Wires	

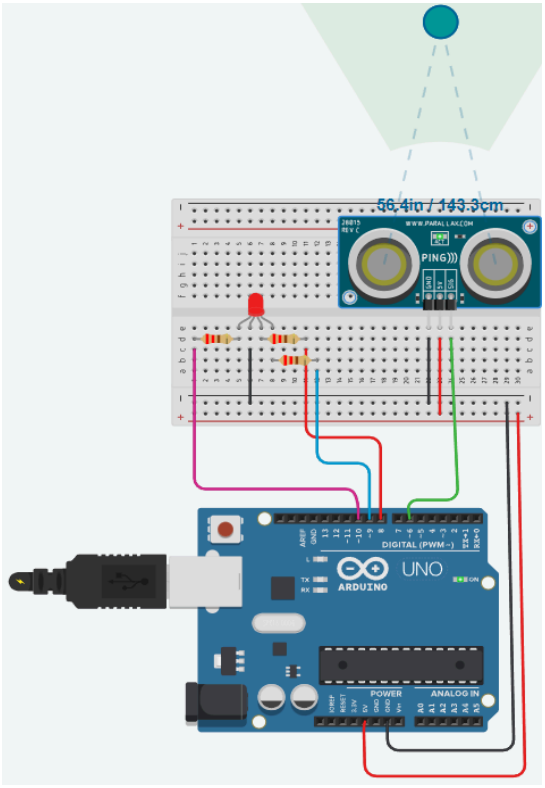
Code:

```
const int pingPin = 6;
const int b = 8;
const int g = 9;
const int r = 10;
int brightness;
void setup() {
        // initialize serial communication:
        Serial.begin(9600);
        pinMode(g, OUTPUT);
        pinMode(b, OUTPUT);
        pinMode(r, OUTPUT);
        digitalWrite(g,0);
        digitalWrite(b,0);
        digitalWrite(r,0);

}

void loop() {
```

```
  // establish variables for duration of the ping, and the distance result
  // in inches and centimeters:
  long time, dist;
  // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);

  // The same pin is used to read the signal from the PING))): a HIGH pulse
  // whose duration is the time (in microseconds) from the sending of the ping
  // to the reception of its echo off of an object.
  pinMode(pingPin, INPUT);
  time = pulseIn(pingPin, HIGH);

  // convert the time into a distance

  dist = (time/29)/2;
  Serial.print(dist);
  Serial.print("cm");
  Serial.println();

        if(dist<75){
    digitalWrite(g,HIGH);
    digitalWrite(b,LOW);
       digitalWrite(r,LOW);
   }
  else if(dist>75 && dist<=125){
    digitalWrite(b,HIGH);
    digitalWrite(g,LOW);
       digitalWrite(r,LOW);
   }
       else{
    digitalWrite(r,HIGH);
    digitalWrite(b,LOW);
       digitalWrite(g,LOW);
   }
   delay(100);
}
```

Output:

<table>
<tr>
<td> 
</td>
<td>Serial Monitor
58cm
58cm
58cm
58cm
58cm
58cm
58cm
58cm
58cm
58cm
58cm
58cm
58cm</td>
</tr>
<tr>
<td> 
</td>
<td>Serial Monitor
24cm
33cm
43cm
48cm
52cm
55cm
62cm
65cm
72cm
76cm
82cm
88cm
89cm</td>
</tr>
</table>



Serial Monitor

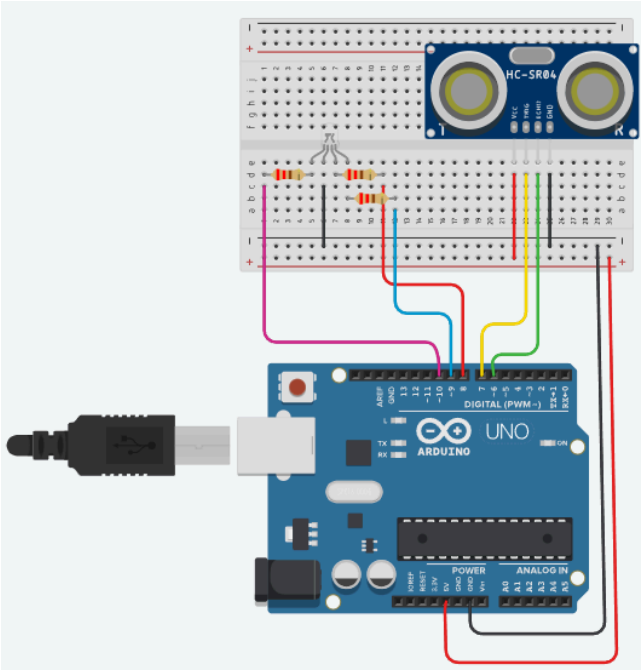
```
144cm
123cm
116cm
108cm
98cm
96cm
96cm
96cm
96cm
96cm
96cm
96cm
96cm
96cm
```

d)

Components Required:

6. Breadboard
7. Arduino Uno Board
8. Ultrasonic Sensor (4 pins)
9. RGB LED
10. Jumper Wires

Circuit Diagram:

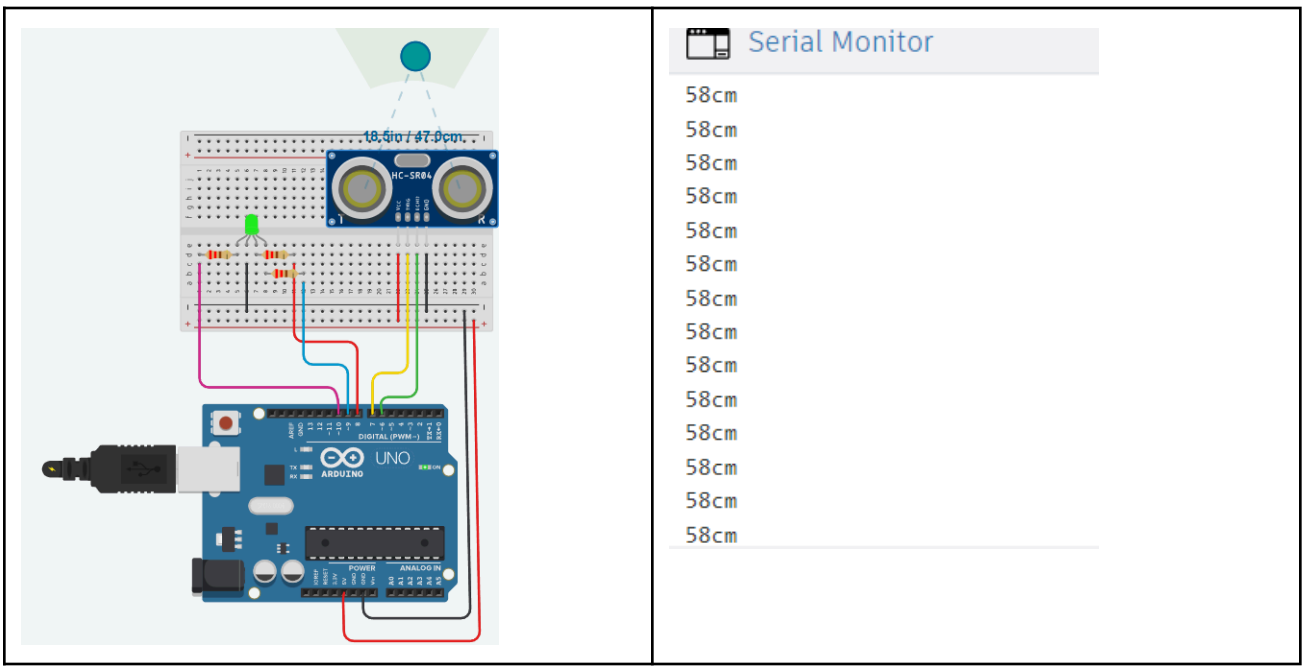


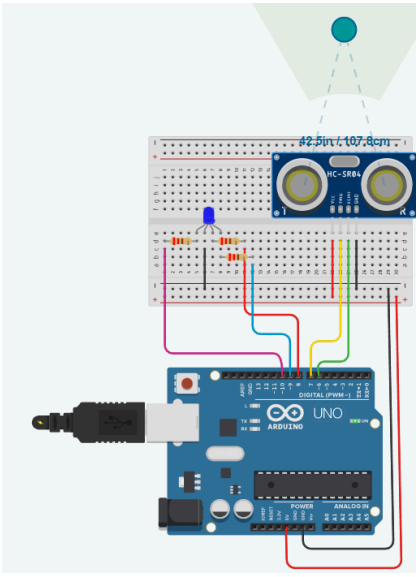
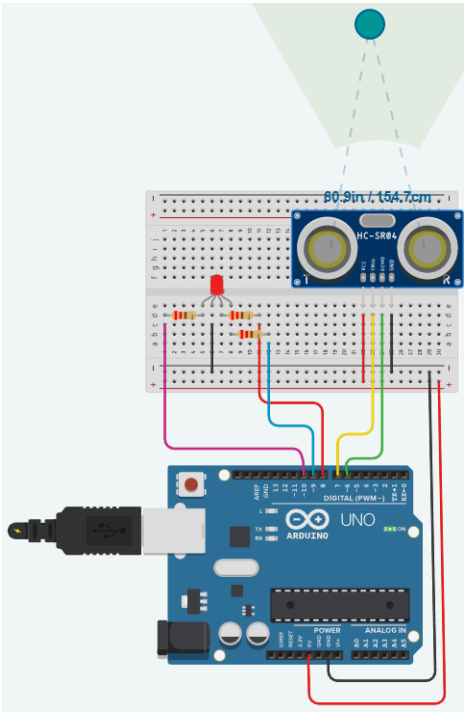
Code:

```
const int echoPin = 6;
const int trigPin = 7;
const int b = 8;
const int g = 9;
const int r = 10;
int brightness;
void setup() {
        // initialize serial communication:
        Serial.begin(9600);
        pinMode(trigPin, OUTPUT);
        pinMode(echoPin, INTPUT);
        pinMode(g, OUTPUT);
        pinMode(b, OUTPUT);
        pinMode(r, OUTPUT);
        digitalWrite(g,0);
        digitalWrite(b,0);
        digitalWrite(r,0);

}

void loop() {
   // establish variables for duration of the ping, and the distance result
   // in inches and centimeters:
   long time, dist;
   // The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
   // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:

   digitalWrite(trigPin, LOW);
   delayMicroseconds(2);
   digitalWrite(trigPin, HIGH);
   delayMicroseconds(5);
   digitalWrite(trigPin, LOW);

   // The same pin is used to read the signal from the PING))): a HIGH pulse
   // whose duration is the time (in microseconds) from the sending of the ping
   // to the reception of its echo off of an object.

   time = pulseIn(echoPin, HIGH);

   // convert the time into a distance
```

```
dist = (time/29)/2;
Serial.print(dist);
Serial.print("cm");
Serial.println();

      if(dist<75){
   digitalWrite(g,HIGH);
   digitalWrite(b,LOW);
      digitalWrite(r,LOW);
  }
else if(dist>75 && dist<=125){
   digitalWrite(b,HIGH);
   digitalWrite(g,LOW);
      digitalWrite(r,LOW);
  }
      else{
   digitalWrite(r,HIGH);
   digitalWrite(b,LOW);
      digitalWrite(g,LOW);
  }
  delay(100);
}
```

Output:



<table>
<tr>
<td> 
</td>
<td>Serial Monitor
24cm
33cm
43cm
48cm
52cm
55cm
62cm
65cm
72cm
76cm
82cm
88cm
89cm</td>
</tr>
<tr>
<td> 
</td>
<td>Serial Monitor
144cm
123cm
116cm
108cm
98cm
96cm
96cm
96cm
96cm
96cm
96cm
96cm
96cm
96cm</td>
</tr>
</table>

Result:

Thus object Detection using Ultrasonic Sensor and Arduino Uno has been implemented successfully.

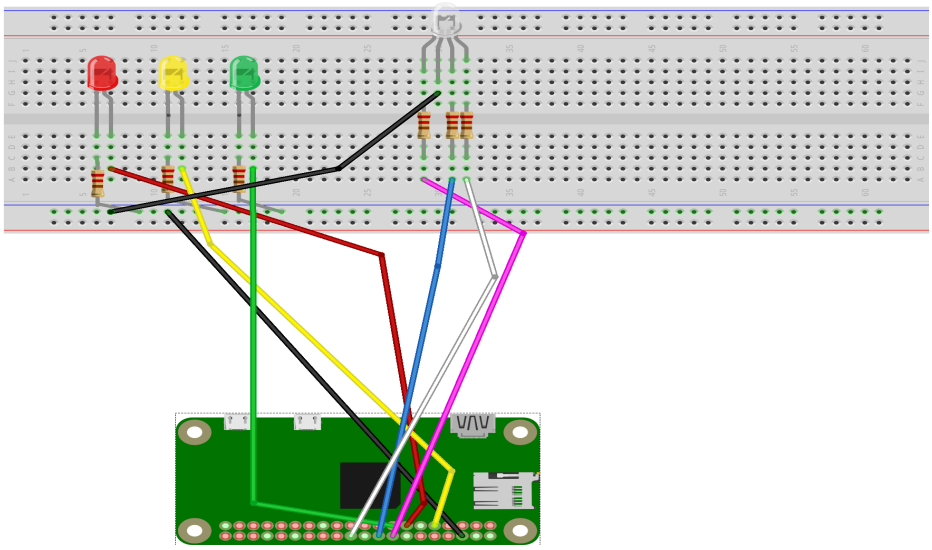
Experiment 4: Controlling traffic light from a Web Page using PHP with Raspberry Pi

Aim: To control the traffic lights from a web page using PHP and Arduino Uno

Components & Softwares Required

1. **Raspberry Pi** with GPIO pins connected to the traffic light.
2. **Web page interface** (HTML + PHP) to control the lights.
3. **PHP script** to interface between the web page and the Raspberry Pi GPIO.
4. **Traffic Light**: Use three LEDs (Red, Yellow, Green) connected to GPIO pins.
5. **Resistors**: Use current-limiting resistors (220Ω) for each LED.
6. **Wires**: Connect each LED's anode (long leg) to the respective GPIO pin and cathode (short leg) to ground via the resistor.

Circuit Diagram:



Program:

index.html

```
<html>
<head>
<meta name="viewport" content="width=device-width" />
<title>Control LED with Raspberry Pi using Apache Webserver </title>
</head>
   <style>
   body
   {
```

```
    background-color: rgb(212,250,252);
    font-family: 'Arial';

}
.red {
    background-color: red;
    width: 10em; height: 4em;
    font-size: 20px;
}
.yellow {
    background-color: yellow;
    width: 10em; height: 4em;
    font-size: 20px;
}
.green {
    background-color: green;
    width: 10em; height: 4em;
    font-size: 20px;
}
.blue{
    background-color: blue;
    width: 10em; height: 4em;
    font-size: 20px;
</style>


<body>
   <center><h1>Control LED from web using Apache Webserver</h1>
   <form method="get" action="led.php">
          <input class ="red" type="submit"  value="Turn Red LED On" name="ron">
    <input class=" red" type="submit"  value="Turn Red LED Off" name="roff">
    <br /> <br />
    <input class="yellow" type="submit" value="Turn Yellow LED On" name="yon">
    <input class="yellow" type="submit" value="Turn Yellow LED Off" name="yoff">
       <br /> <br />
    <input class="green" type="submit" value="Turn Green LED On" name="gon">
```

```
      <input class="green" type="submit" value="Turn Green LED Off" name="goff">
      <br /> <br />
      <h1>RGB LED</h1>
    <input class="red" type="submit" value="Red On" name="rgbron">
      <input class="red" type="submit" value="Red Off" name="rgbroff">
      <br /> <br />
   <input class="green" type="submit" value="Green On" name="rgbgon">
         <input class="green" type="submit" value="Green Off" name="rgbgoff">

   <br /> <br />
        <input class="blue" type="submit" value="Blue On" name="rgbbon">
          <input class="blue" type="submit" value="Blue Off" name="rgbboff">
</form>
</body>
</html>
```

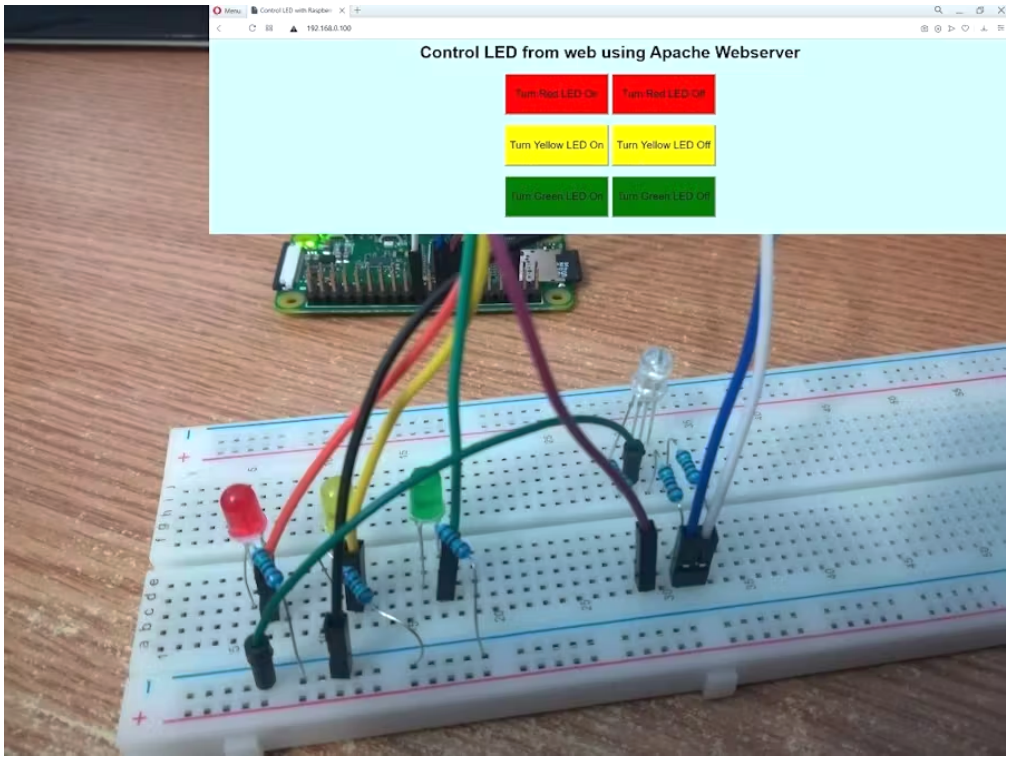
Led.php

```
<?php
   shell_exec("gpio -g mode 17 out");
   shell_exec("gpio -g mode 22 out");
   shell_exec("gpio -g mode 27 out");
   //RGB LED
   shell_exec("gpio -g mode 23 out");
   shell_exec("gpio -g mode 24 out");
   shell_exec("gpio -g mode 25 out");

 if(isset($_GET['roff']))
 {
     shell_exec("gpio -g write 17 0");
 }
 else if(isset($_GET['ron']))
 {
  shell_exec("gpio -g write 17 1");
  }
  else if(isset($_GET['yon']))
  {
```

```
    shell_exec("gpio -g write 27 1");
}
else if(isset($_GET['yoff']))
{
    shell_exec("gpio -g write 27 0");
}
else if(isset($_GET['gon']))
{
    shell_exec("gpio -g write 22 1");
}
else if(isset($_GET['goff']))
{
    shell_exec("gpio -g write 22 0");
}

// for RGB LED
else if(isset($_GET['rgbron']))
{
    shell_exec("gpio -g write 23 1");
}
else if(isset($_GET['rgbroff']))
{
    shell_exec("gpio -g write 23 0");
}
else if(isset($_GET['rgbgon']))
{
    shell_exec("gpio -g write 24 1");
}
else if(isset($_GET['rgbgoff']))
{
    shell_exec("gpio -g write 24 0");
}
else if(isset($_GET['rgbbon']))
{
    shell_exec("gpio -g write 25 1");
}
```

```
    else if(isset($_GET['rgbboff']))
    {
        shell_exec("gpio -g write 25 0");
    }

?>
```

Output:



Result:

Thus a web page has been developed to control the traffic lights using Raspberry Pi and PHP

Experiment 5: Sensing and sending sensor value via SMS

Aim: To sense a sensor value and send it as an SMS

```
#include <SoftwareSerial.h>
// Include the MG2639 Cellular Shield library
#include <SFE_MG2639_CellShield.h>
const char EOM_CHAR = '~';

char myPhone[15];

int moistureSensor = 0;
void setup()
{
        Serial.begin(9600);

        int beginStatus = cell.begin();
        if (beginStatus <= 0)
        {
          Serial.println(F("Unable to communicate with shield. Looping"));
          while(1)
            ;
  }
  delay(2000);

        sms.setMode(SMS_TEXT_MODE);
}

void loop()
{
        int sensorValue = analogRead(moistureSensor);
        Serial.println(sensorValue);

        if (Serial.available())
        {

          if (sensorValue >= 645)
          {
            writeSMS();
          }
        }
}
```

```
void writeSMS()
{


        Serial.print(F("Sending a message to "));
        Serial.println("5128254963");

        sms.start("5128254963");
        if (Serial.available())
        {
        sms.write("water.");

        }

        Serial.println();
        Serial.println("Sending the message.");
        Serial.println();
         sms.send();
}
```

Result:

Thus the sensor value has been sent as SMS to the user's mobile using Arduino Uno and embedded C.

Experiment 6: Sending images and video via Gmail using Raspberry Pi

a) Sending image via Gmail

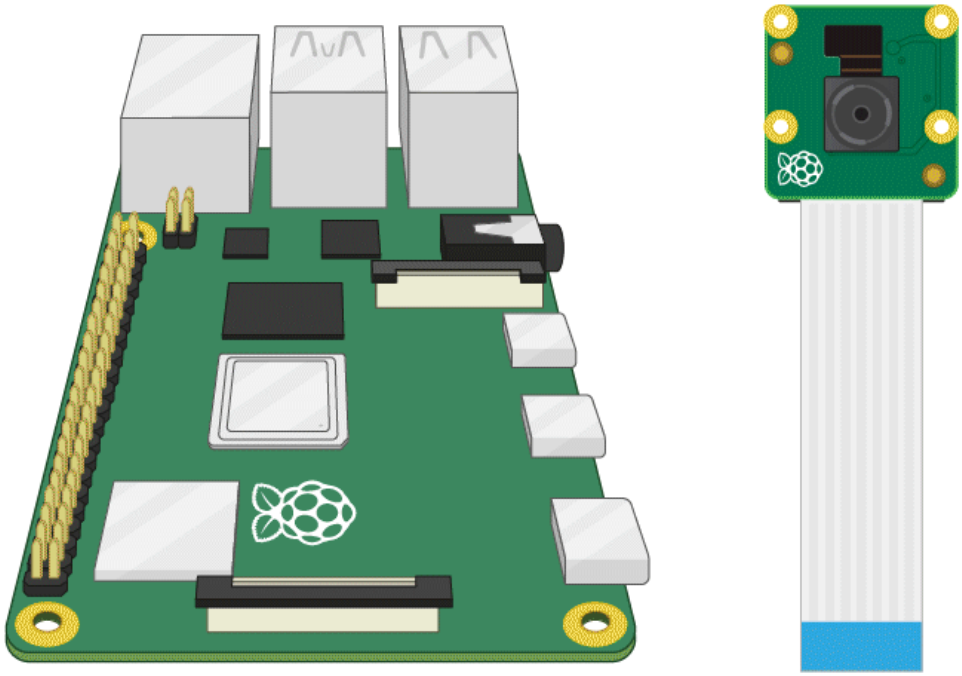
Aim: To capture image via camera module and send the image as attachment to an Email ID

Components Required:

Raspberry Pi

Raspberry Pi Camera Module

Circuit Diagram:



Procedure:

1. Open a terminal and type the following commands one at a time to install picamzero:

   sudo apt update
   sudo apt install python3-picamzero

2. Type and Execute the following code to capture the image from the capture and store it in a directory:

   from picamzero import Camera import os home_dir = os.environ['HOME'] #set the location of your home directory cam = Camera() cam.start_preview()

   cam.take_photo(f"{home_dir}/Desktop/new_image.jpg") #save the image to your desktop

   cam.stop_preview()

3. Follow the steps given below to set up your Gmail account for sending emails using Python:

Step 2:Enable Less Secure Apps (Optional but not recommended)

1. Go to your Google Account settings: Google Account Security
2. Scroll down to **Less secure app access**.
3. Turn **Allow less secure apps ON**.

Step 2: Use App Passwords (Recommended)

1. **Enable 2-Step Verification:**
    - Go to Google Account Security
    - Under **Signing into Google**, select **2-Step Verification** and follow the steps to enable it.
2. **Generate App Password:**
    - Once 2-Step Verification is enabled, go back to the **Security** section.
    - Find **App Passwords** (it might be under **Signing in to Google**).
    - Choose **Mail** as the app and **Other (Custom name)** — give it a name like *Python Email Script*.
    - Click **Generate**.
    - Copy the 16-character app password

4. Next, type and Execute the following code to send the captured image to mail:

Program:

```
subject='Security alert!!'
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.image import MIMEImage


# Sender and recipient email addresses
sender_email = "geetha.cse@kongu.edu"
sender_password = "nhyrwngpwzrpchvu"
recipient_email = "kalaivanip.cse@kongu.edu"
```

```
# Create a multipart message
message = MIMEMultipart()
message['From'] = sender_email
message['To'] = recipient_email

# Read and attach the image
with open('/content/es.jpg', 'rb') as f:
    image_part = MIMEImage(f.read(), name='/content/es.jpg')
    message.attach(image_part)

# Connect to Gmail's SMTP server securely using SSL
with smtplib.SMTP_SSL('smtp.gmail.com', 465) as server:
    try:
        # Login to the sender's email account
        server.login(sender_email, sender_password)

        # Send the email
        server.sendmail(sender_email, recipient_email, message.as_string())
        print("Email sent successfully!")
    except Exception as e:
        print("An error occurred:", e)
```

b) Sending Video via Gmail

```
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders
```

```
def send_email(sender_email, sender_password, recipient_email, subject, body, file_path):
    # Create a multipart message
    message = MIMEMultipart()
    message['Subject'] = subject
    message['From'] = sender_email
    message['To'] = recipient_email

    # Add text part
    html_part = MIMEText(body, 'plain')
    message.attach(html_part)

    # Attach video file
    with open(file_path, 'rb') as f:
        attachment = MIMEBase('application', 'octet-stream')
        attachment.set_payload(f.read())
        encoders.encode_base64(attachment)
        attachment.add_header('Content-Disposition', f'attachment;
            filename="{file_path}"')
        message.attach(attachment)

    # Connect to Gmail's SMTP server securely using SSL
    with smtplib.SMTP_SSL('smtp.gmail.com', 465) as server:
        try:
            # Login to the sender's email account
            server.login(sender_email, sender_password)

            # Send the email
            server.sendmail(sender_email, recipient_email, message.as_string())
            print("Email sent successfully!")
        except Exception as e:
            print("An error occurred:", e)

# Usage
sender_email = "suchetan.21cse@kongu.edu"
sender_password = "nhyrwngpwzrpchvu"
```

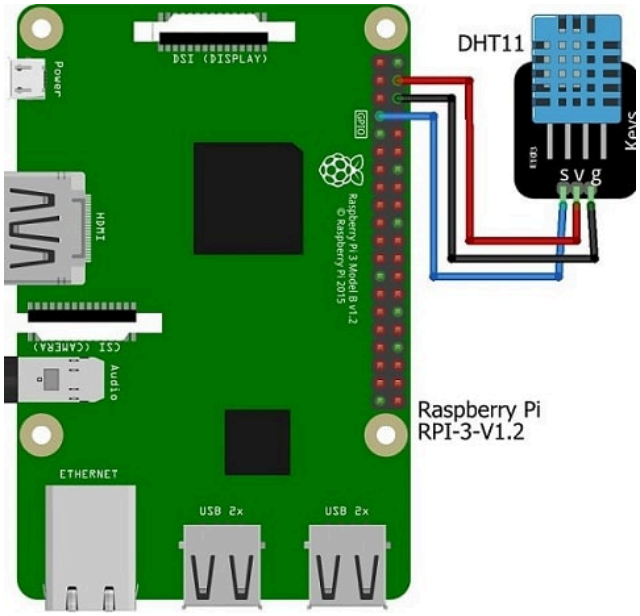
```
recipient_email = "subigan.21cse@kongu.edu"
subject = "Video Email"
body = "Someone is trying to unlock the door."
file_path = "/Home_Security/video.mp4"

send_email(sender_email, sender_password, recipient_email, subject, body, file_path)
```

Result:

Thus an image and video have been sent to the user's Gmail account as a security alert.

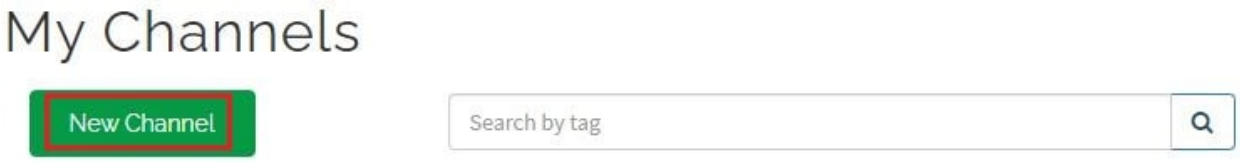
Experiment 7: Measuring sensor value and uploading the content onto ThingSpeak cloud for analysis

Aim: To read temperature and humidity from DHT11 sensor and send the data to ThingSpeak cloud channel

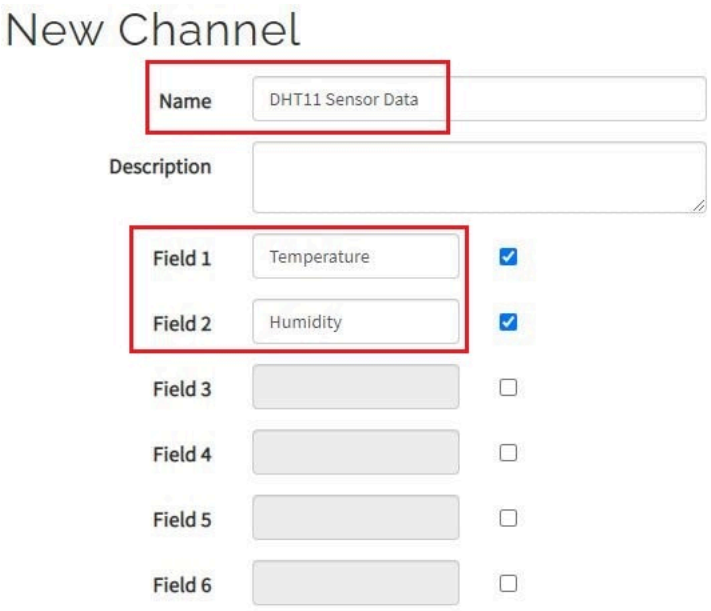
Components Required:	**Circuit Diagram:**
1. Breadboard 2. Raspberry Pi 3. Ultrasonic Sensor 4. Jumper Wires	

Setting up a Channel in ThingSpeak:

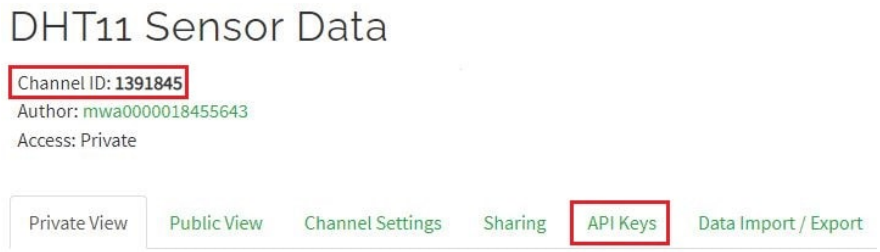
1. Create an account in ThingSpeak. After creating the account login, click on New Channel to create a channel.



2. Define the channels by entering the a proper name and description. For the Field 1 and Field 2 we have named Temperature and humidity.

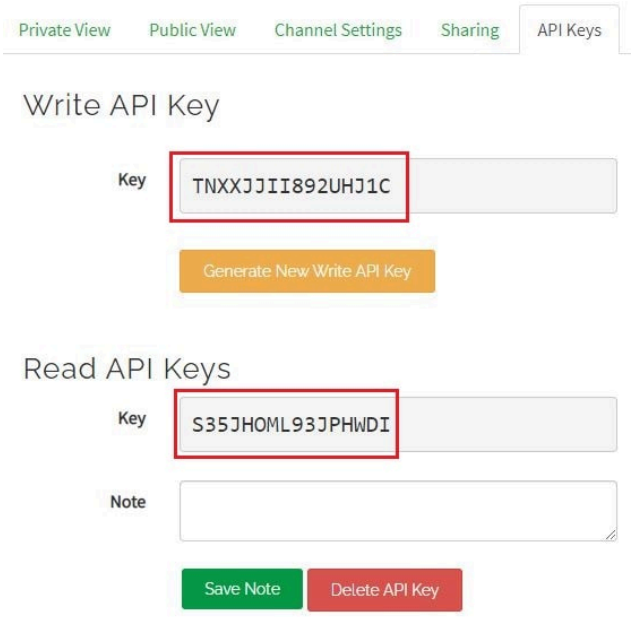


3. Save the channel. Now, you will be automatically redirected to the "Private View" tab. Here, we can find the "Channel ID". Below You will also see API Keys option.



4. Click on the "API Keys" tab. The two values of "Write API key" and "Read API key" are required to write or retrieve data. Copy these keys and keep it safe.

Installing Required Libraries



5. Installing Required Libraries

   For installing the basic updates run these commands in a terminal window on your Raspberry Pi

   ```
   sudo apt - get update
   sudo apt - get install build - essential python - dev python - openssl git
   ```

6. Install the library to read the DHT11 or DHT22 sensors. Below library will work for both the sensor types

   ```
   git clone https://github.com/adafruit/Adafruit_Python_DHT.git && cd
   Adafruit_Python_DHT
   ```

7. Installing Raspberry Pi Thingspeak Library

   ```
   sudo pip install thingspeak
   ```

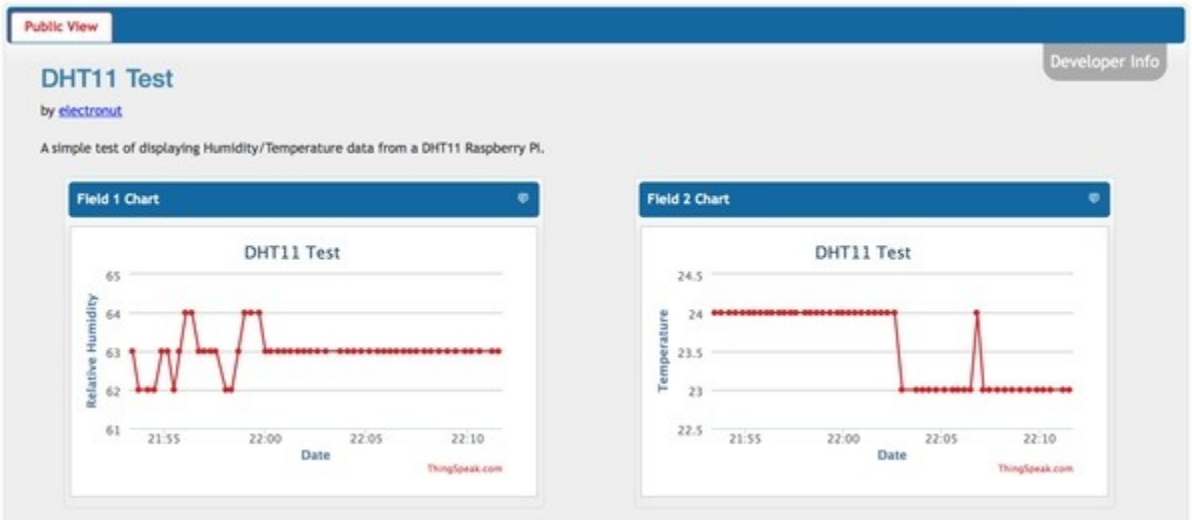
The python script will read the DHT11 temperature and humidity every 15 seconds and send it to our channel.

Program:

```
#Import necessary libraries
import RPi.GPIO as GPIO
import time
import sys

import urllib2

#Set GPIO pin mode and pin numbering
GPIO.setmode(GPIO.BCM)

import sys
import RPi.GPIO as GPIO
from time import sleep
import Adafruit_DHT
import urllib2

def getSensorData():
   RH, T = Adafruit_DHT.read_retry(Adafruit_DHT.DHT11, 23)
   # return dict
   return (str(RH), str(T))

# main() function
```

```
def main():
    # use sys.argv if needed
    if len(sys.argv) < 2:
        print('Usage: python tstest.py PRIVATE_KEY')
        exit(0)
    print 'starting...'

#Write the sensor data to ThingSpeak Channel Field
baseURL = 'https://api.thingspeak.com/update?api_key=%s' % sys.argv[1]

    while True:
        try:
            RH, T = getSensorData()
            f = urllib2.urlopen(baseURL +
                                "&field1=%s&field2=%s" % (RH, T))
            print f.read()
            f.close()
            sleep(15)
        except:
            print 'exiting.'
            break

# call main
if __name__ == '__main__':
    main()
```

Output:

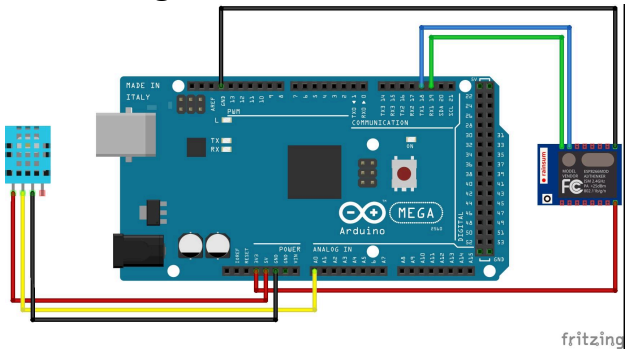


Result:

Thus a Python code is executed to sense the sensor data to Thing Speak channel

Experiment 8: IoT Based temperature monitoring over Blynk App.

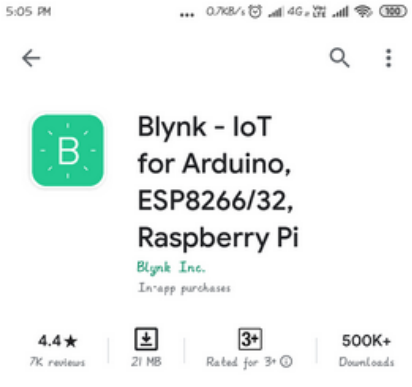
Aim: To develop an IoT-based temperature monitoring system using Arduino Uno and Blynk App

Components Required: • ESP8266 NodeMCU • DHT-11 sensor • Breadboard • Jumper Wires **Softwares Required:** • Arduino IDE • Blynk Appilication	**Circuit Diagram:** 

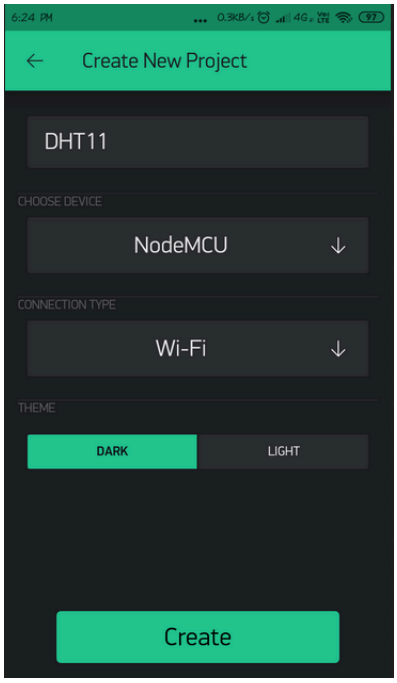
Procedure:

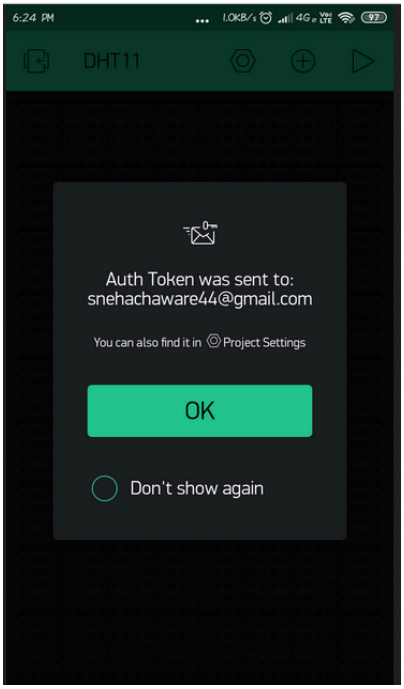
<u>Installation and Configuration of Blynk App:</u>

1. Go to play store app store and type blynk, you will find the green color icon for the blynk app, then install it on your device.

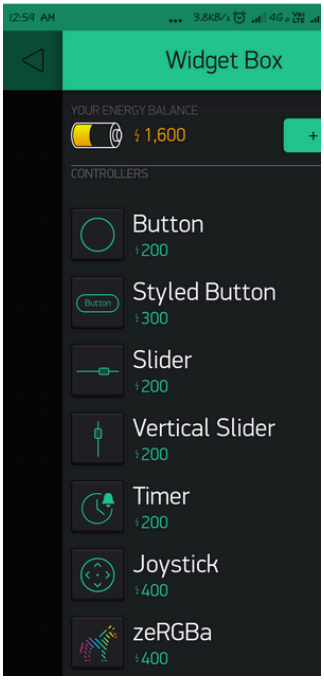
i. 

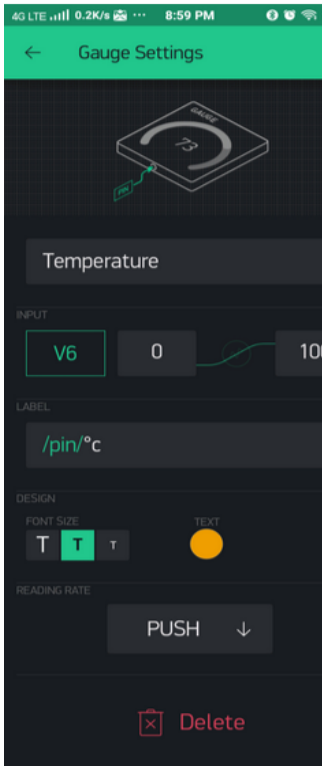
2. Once installation is complete, open the app
3. Sign up with a gmail id and then create a new project.
4. Click on create option and fill the details like project name, device used and connection type.
5. Auth token will be generated which will be available in the settings option of this project and on your **Gmail Account.**

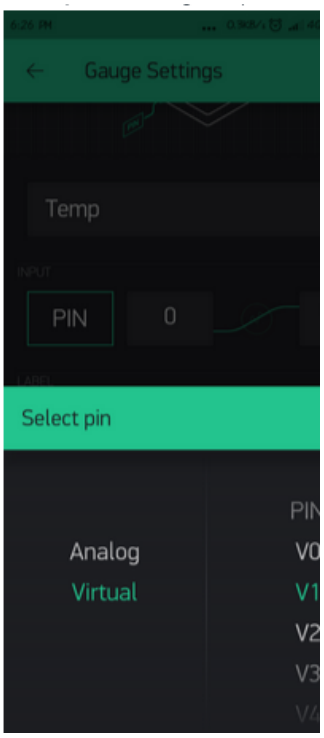




6. Once your project is created, insert different types of widget into it. For this experiment, add 2 Gauges and 2 Graphs.
7. Tap on the Widget and select the respective Virtual pins for temperature and humidity data(V6 and V8 for temperature and V5 and V7 for humidity).







8. Now provide a name to the widget by editing the settings. Also, select the pin for the ‘OUTPUT’ and give names to On/Off labels.
9. Ensure that the Reading rate is set as '1' second for all Widgets. And add gauges and graphs for both Humidity and Temperature
10. After all the process with this settings your Dashboard will appear as show below:

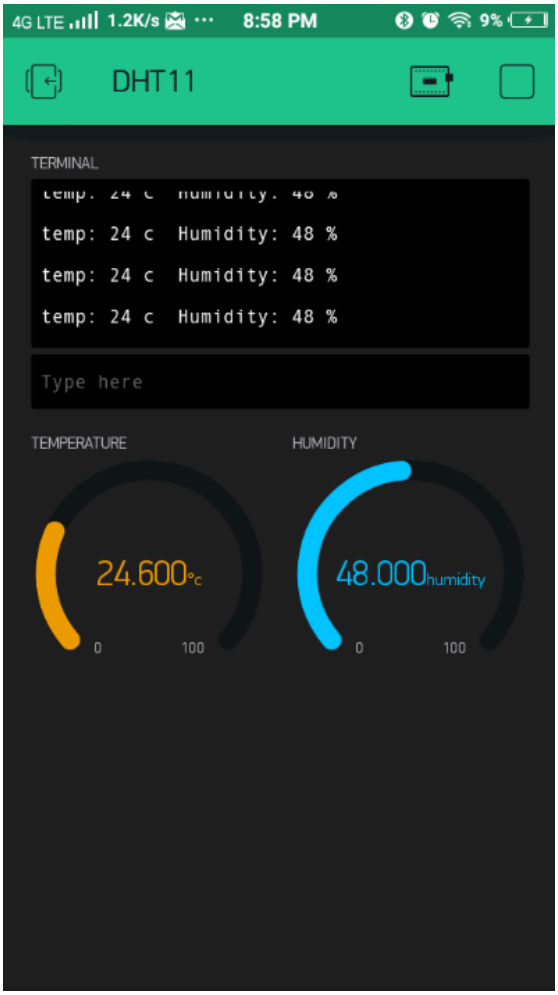
11. Open your Blynk registered Gmail account and open the mail from Blynk app which gives you AUTH token for your project "**DHT11"**; copy that auth code to Auth type in the code box.
12. Insert your Hotspot ID password into your SSID and Password on your Arduino Sketch.
13. Download the following libraries:
    - SPI.h
    - ESP8266WiFi.h
    - BlynkSimpleEsp8266.h
    - SimpleTimer.h
    - DHT.h

Program:

```
#define BLYNK_PRINT Serial // Comment this out to disable prints and save space
#include <SPI.h>;
#include <ESP8266WiFi.h>;
#include <BlynkSimpleEsp8266.h>;
#include <SimpleTimer.h>;
#include <DHT.h>;

// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = "YOUR_BLYNK_AUTH_KEY";
```

```
// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "WIFI_SSID";
char pass[] = "WIFI_PASSWORD";

#define DHTPIN 2 // What digital pin we're connected to

// Uncomment whatever type you're using!
#define DHTTYPE DHT11 // DHT 11
//#define DHTTYPE DHT22 // DHT 22, AM2302, AM2321
//#define DHTTYPE DHT21 // DHT 21, AM2301

DHT dht(DHTPIN, DHTTYPE);
SimpleTimer timer;

// This function sends Arduino's up time every second to Virtual Pin (5, 6, 7
&amp;amp;amp;amp;amp;amp; 8).
// In the app, Widget's reading frequency should be set to PUSH. This means
// that you define how often to send data to Blynk App.
void sendSensor()
{
 float h = dht.readHumidity();
 float t = dht.readTemperature(); // or dht.readTemperature(true) for Fahrenheit

 if (isnan(h) || isnan(t)) {
 Serial.println("Failed to read from DHT sensor!");
 return;
 }
 // You can send any value at any time.
 // Please don't send more that 10 values per second.
 Blynk.virtualWrite(V5, h); // Humidity for gauge
 Blynk.virtualWrite(V6, t); // Temperature for gauge
 Blynk.virtualWrite(V7, h); // Humidity for graph
 Blynk.virtualWrite(V8, t); // Temperature for graph
}
void setup()
{
 Serial.begin(115200); // See the connection status in Serial Monitor
 Blynk.begin(auth, ssid, pass);
 dht.begin();
 // Setup a function to be called every second
 timer.setInterval(1000L, sendSensor);
}
void loop()
{
 Blynk.run(); // Initiates Blynk
 timer.run(); // Initiates SimpleTimer
}
```

Output:



Result:

Thus an IoT Based temperature monitoring system using Arduino Uno and Blynk App has been implemented successfully.

Experiment 9: Create an IoT scenario using Cooja Simulator

Aim: To create an IoT scenario using the Cooja simulator and simulate a network of sensor nodes.

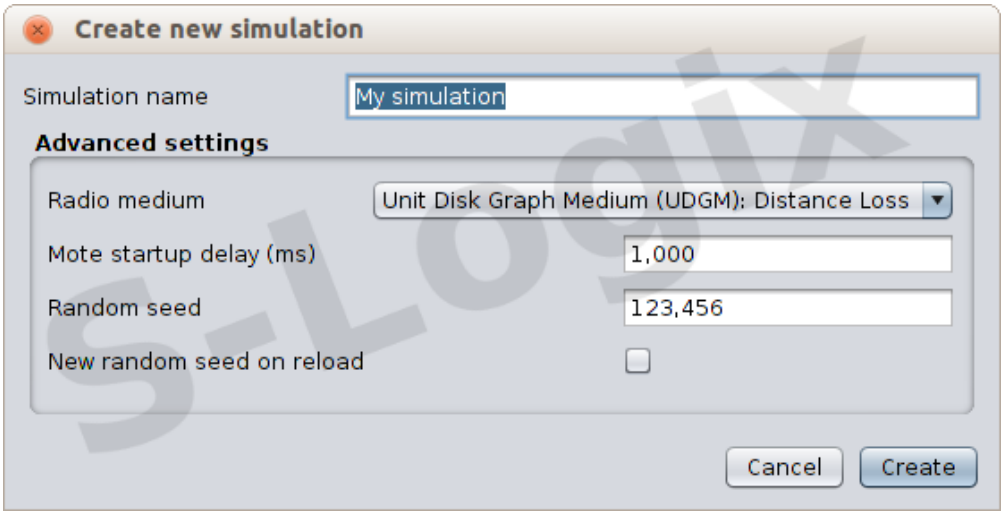
Requirements:

- Contiki OS installed
- Cooja simulator
- Basic understanding of IoT concepts

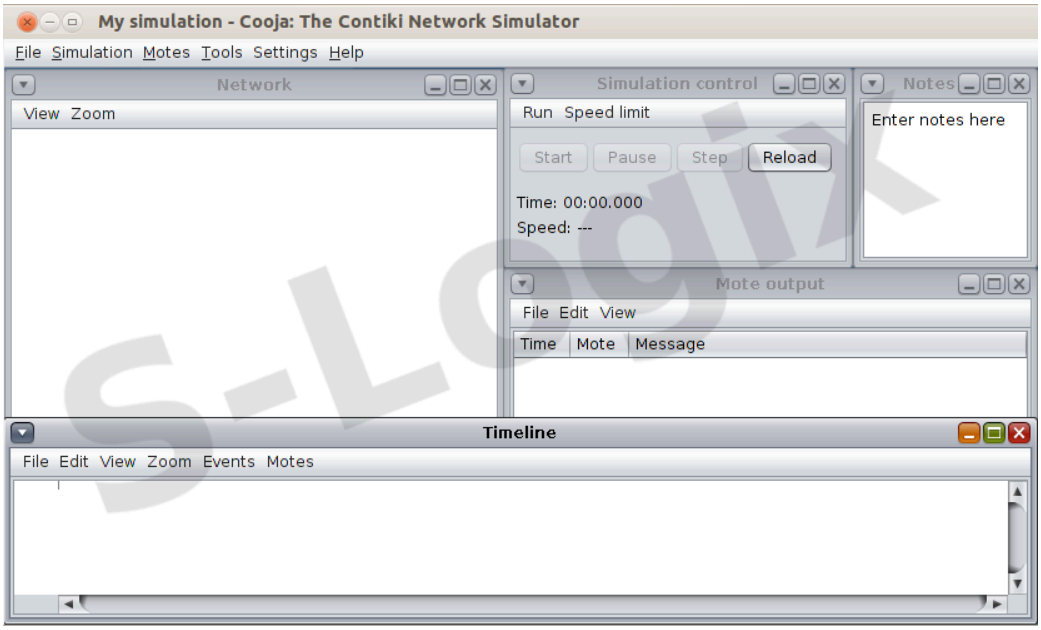
Procedure:

1. Open Cooja Simulator:
    - Launch Contiki OS.
    - Run the terminal and enter the command to open Cooja:
      cd contiki/tools/cooja
      ant run

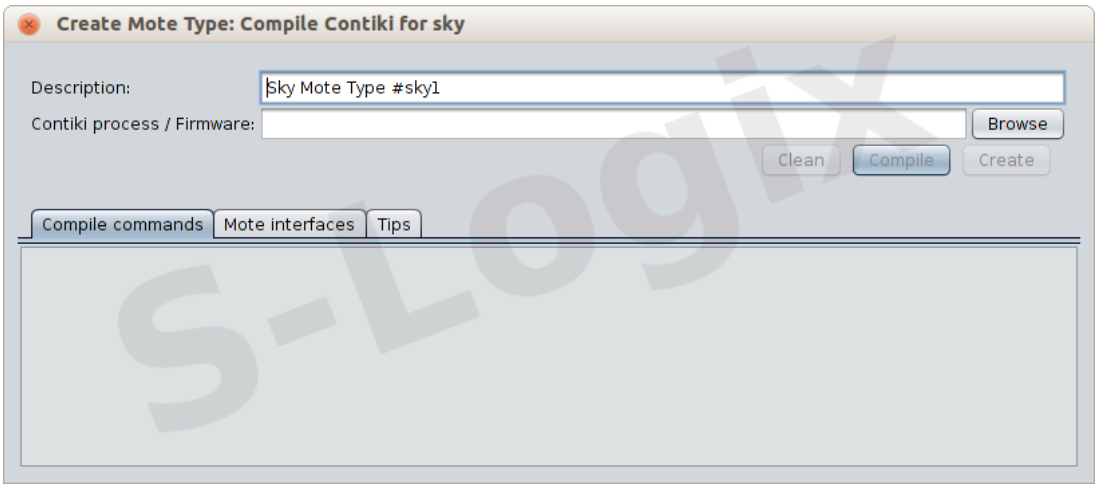
2. Create a New Simulation:
    - In the Cooja window, go to File > New Simulation.
    - Enter a name for your simulation and click Create.

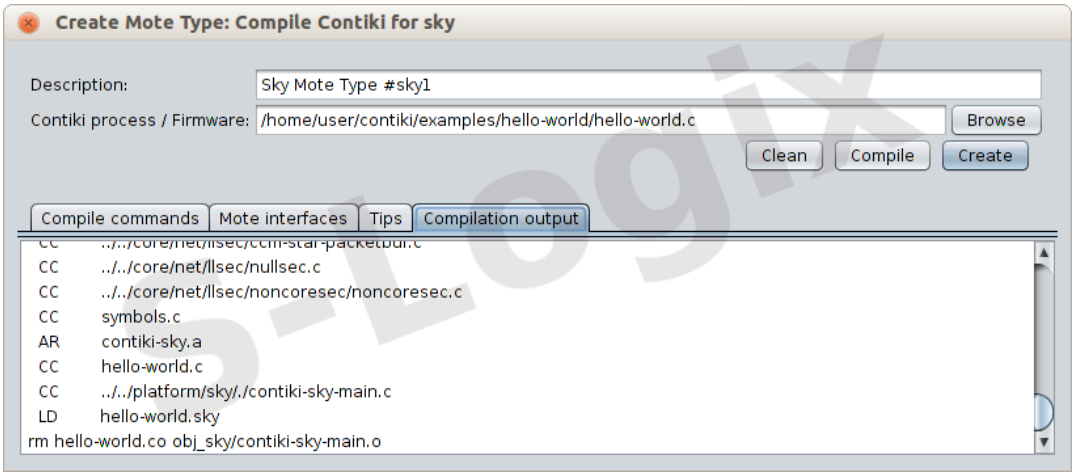


    - A new simulation gets created

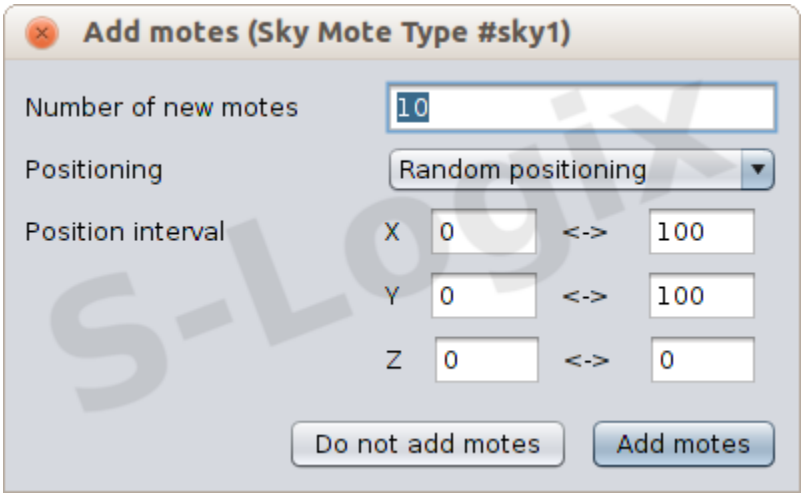


3. Add Nodes:
    - Right-click in the Simulation window and select Add motes > Create New Mote Type > Sky Mote.
    - Compile the firmware using the default or custom Contiki firmware.
    - Select any file for simulation, then select clean and compile. Select create option
    - Click Add Mote and place it on the grid.
    - Repeat the steps to add multiple nodes.

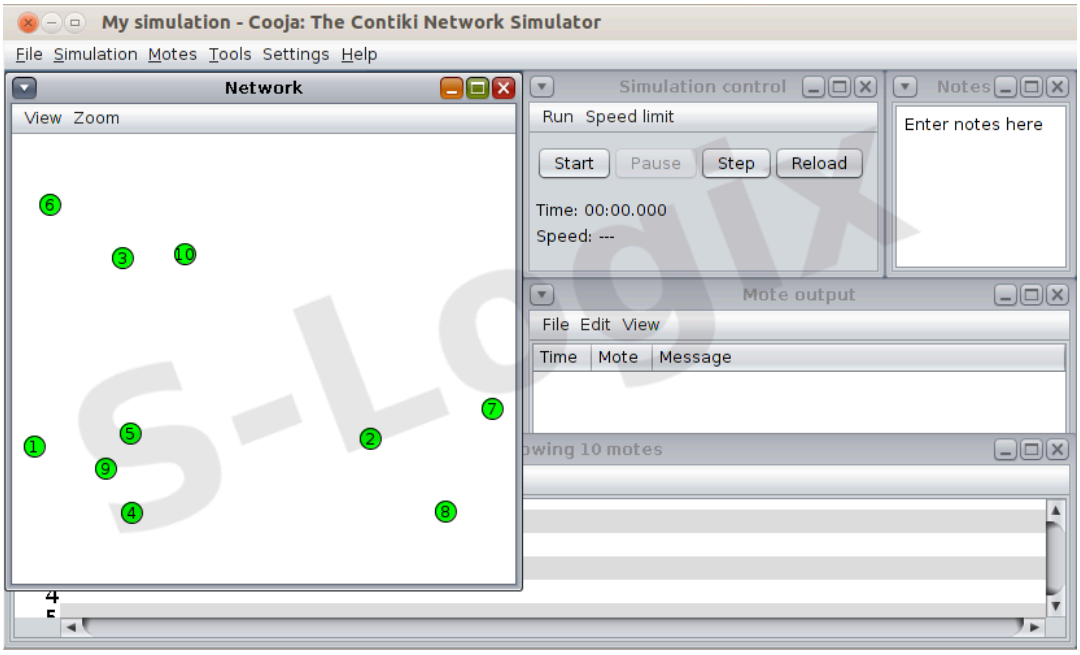




4. Configure Node Settings:
    - Set node attributes like radio range, position, and ID.



5. Connect Nodes:
    - Ensure nodes are within communication range to form a network.
6. Run Simulation:
    - Click Start to run the simulation.
    - Observe node interactions and network topology.



7. Visualize Network:
    - Use Network > Show Radio Environment to see connections and data transfer.

Result:

Thus the IoT scenario has been successfully created and simulated using Cooja.

Experiment 10: Send Data Between an IoT Client and Server Using Cooja Simulator

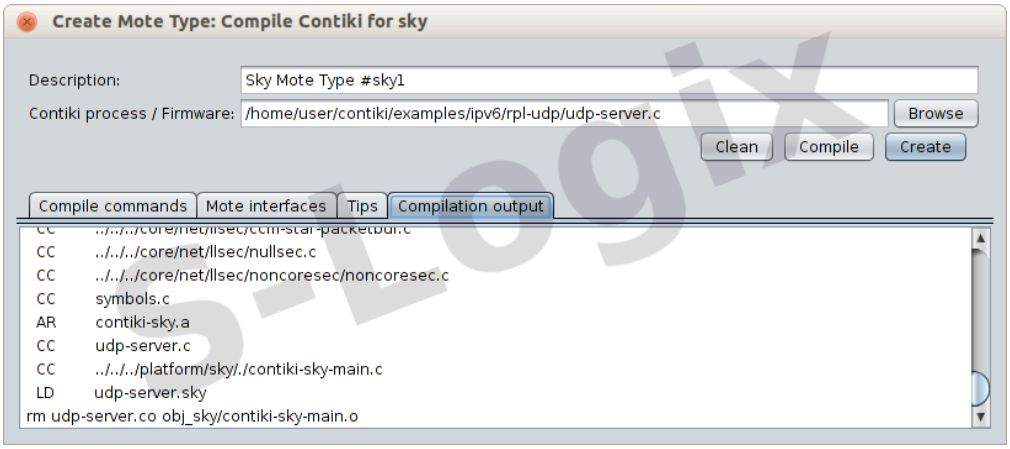
Aim: To establish communication between an IoT client and server using Cooja simulator.

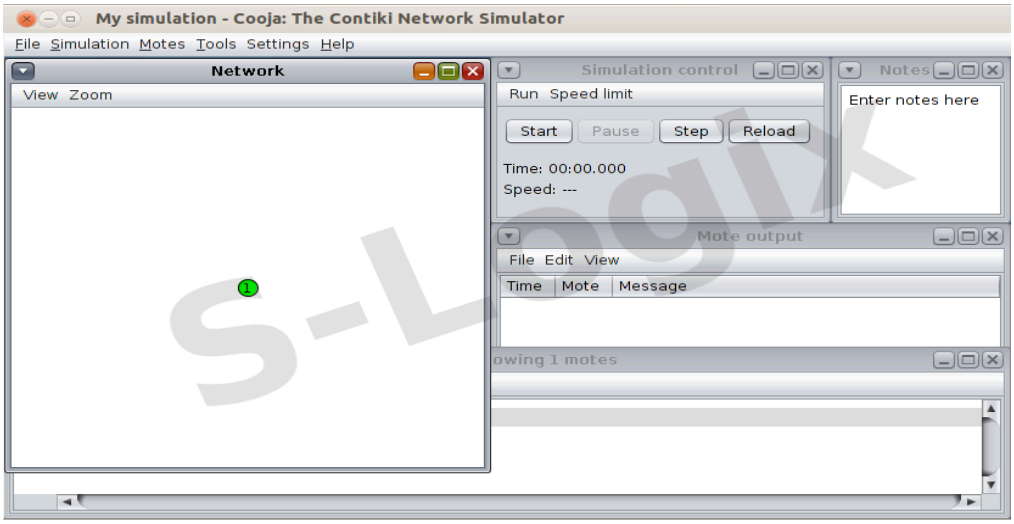
Requirements:

- Contiki OS installed
- Cooja simulator
- Basic knowledge of client-server communication

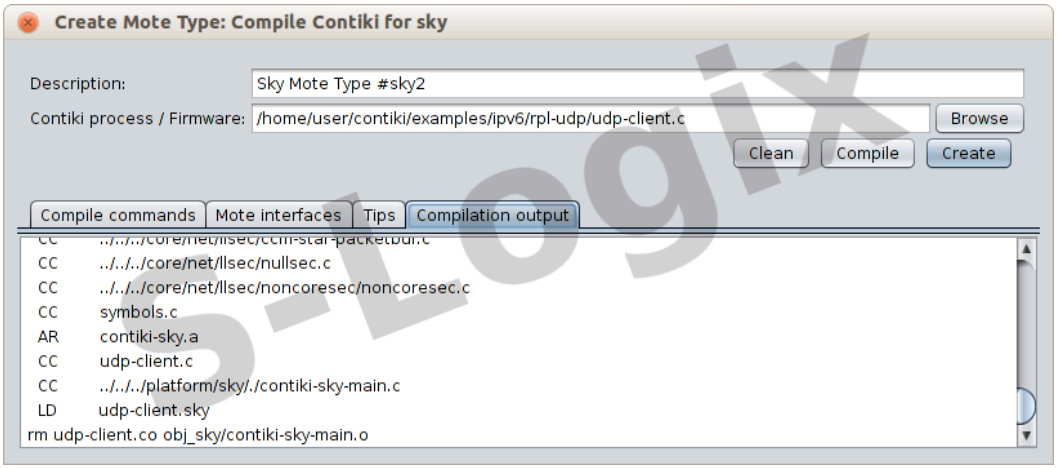
Procedure:

1. Open Cooja:
    - Launch Contiki OS and start Cooja as explained in Experiment 1.
2. Create a New Simulation:
    - Go to File > New Simulation and create a new simulation.
3. Add Server Node:
    - Add a Sky Mote and load a simple UDP server firmware.
    - Compile and add the node to the simulation.



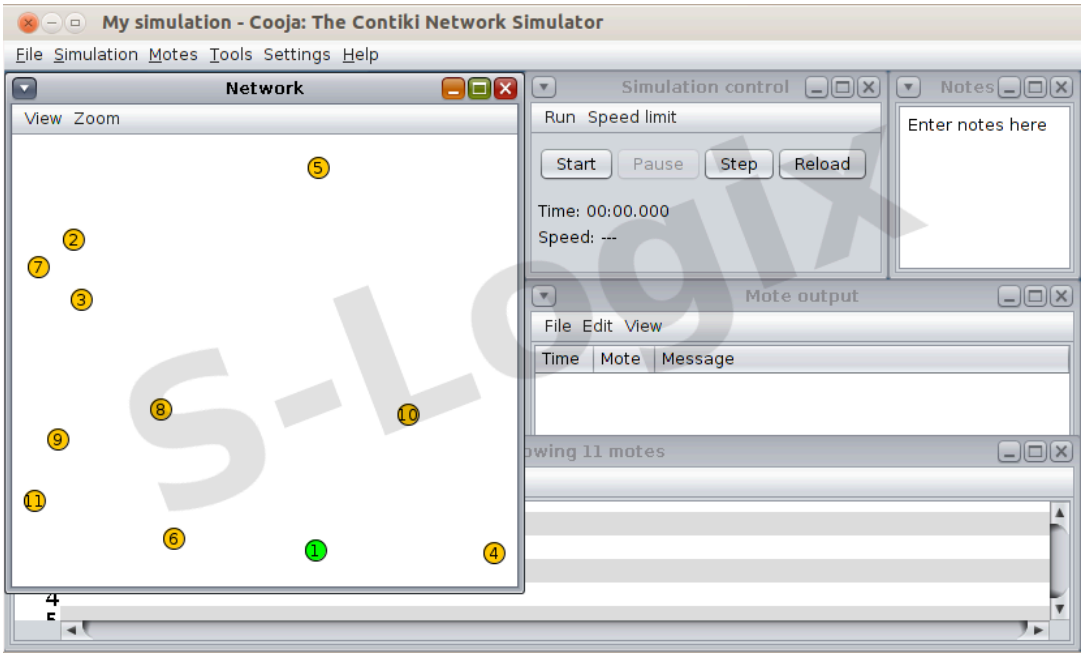


4. Add Client Node:
    - Add another Sky Mote and load a UDP client firmware.

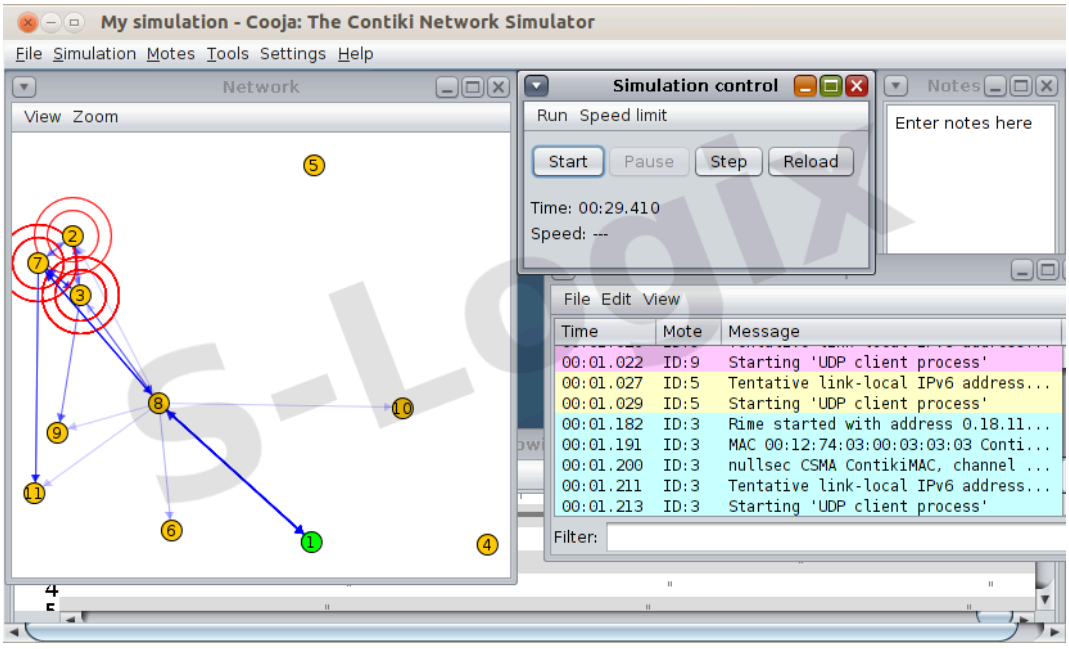


5. Assign IP Addresses:
    - Configure IP addresses for both client and server nodes.
6. Set Up Communication:
    - Ensure nodes are within range.

- Program the client to send data packets to the server.



7. Run Simulation:
   - Start the simulation and observe packet transmission.



8. Check Logs:
   - Use Mote output to verify data exchange.

Result:

Thus data communication has been successfully established between an IoT client and server using Cooja simulator.