Introduction to IoT
Lab 4
Spring 2025
Occupancy Counter

Overview

In this experiemnt, you will work on two sensors together: the ultrasonic and IR sensors. You will count the total number of people present inside the room at a given moment and show this count value on your local web server using the Wi-Fi functionality of the ESP32. Moving on, we will convert this local IP address to a global link so that anyone across the globe can access it.

IR Sensor

IR sensors typically consist of an infrared LED and a photodiode. The LED emits infrared radiation, and the photodiode measures the reflected radiation. By measuring the reflected radiation, the sensor can determine the distance of an object. IR sensors are widely used for obstacle detection and distance measurement applications because of their low cost, ease of use, and reliability. They are typically used in applications where the object to be detected is within a certain range, typically a few centimeters to a few meters.

Key Specifications

- **Object Detection:** Infrared obstacle avoidance sensor can detect objects at a range of up to 30 cm, although the range can vary depending on the specific sensor model.
- **Operating Voltage:** The typical operating voltage range for infrared obstacle avoidance sensor is 3-5V DC.
- **Current Consumption:** The current consumption of infrared obstacle avoidance sensor is generally low, typically ranging from 20-50 mA.
- **Output Signal:** Infrared obstacle avoidance sensors typically provide a digital output signal, which can be used to indicate the presence or absence of an obstacle. Some sensors may also provide an analog output signal, which can be used to determine the distance to the obstacle.
- **Operating Temperature:** Infrared obstacle avoidance sensors can typically operate over a wide temperature range, from -20 to 60 degrees Celsius.
- **Detection Angle:** Infrared obstacle avoidance sensors have a narrow detection angle, typically around 35-45 degrees.
- **Sensitivity:** The sensitivity of infrared obstacle avoidance sensors can be adjusted using a potentiometer, allowing them to be tuned for optimal performance in different environments.
- **Interference:** Infrared obstacle avoidance sensors can be affected by interference from other sources of infrared radiation, such as sunlight or fluorescent lighting. Some sensors may include features to help reduce interference, such as a shield to block out unwanted signals.

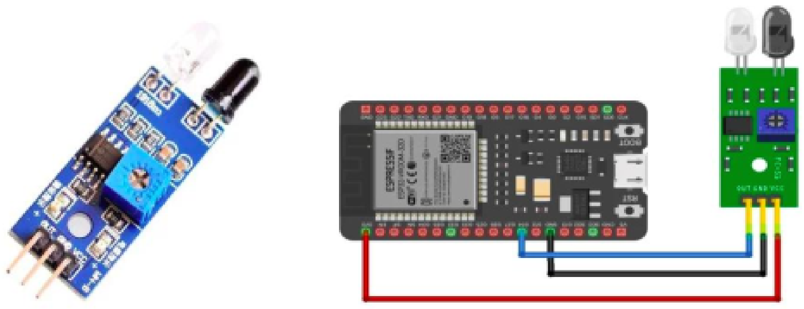


Figure 1: IR sensor with ESP32 wiring

Ultrasonic Sensor

Refer to Lab 3

Experiment 1:- Connecting the IR Sensor to the ESP32 for Obstacle Detection

Required Sensors

- **IR Sensor**
- **ESP32 Microcontroller**
- **Breadboard and Jumper Wires**

Connections

- Connect the Vcc pin of the IR sensor to a 3.3V or 5V power source on the ESP32.
- Connect the GND pin of the IR sensor to the ground (GND) on the ESP32.
- Connect the OUT pin of the IR sensor to a digital input pin on the ESP32 (e.g., GPIO pin).

Pseudo Code

```
1. Define the pin for the IR sensor.
2. Initialize serial communication.
3. Set the IR sensor pin as INPUT.
4. Repeat indefinitely:
   a. Read the digital signal from the IR sensor using digitalRead().
   b. If the signal is HIGH:
        - Print "Object detected" to the serial monitor.
   c. Else:
        - Print "No object detected" to the serial monitor.
   d. Wait for 5 seconds.
```

Explanation

The pseudocode outlines a program for interfacing an IR sensor with an ESP32 microcontroller. It begins by defining the GPIO pin connected to the sensor's OUT pin. The program initializes serial communication for monitoring and configures the specified pin as INPUT to read the digital signal from the IR sensor. Within a continuous loop, it reads the sensor's digital output, prints the result to the serial monitor, and introduces a delay of at least 1 second to stabilize readings. The loop repeats indefinitely, enabling continuous monitoring of the IR sensor's output for applications such as proximity detection.

Expected Output

- When an object is detected by the IR sensor, the serial monitor will display: `"Object detected"`.
- When no object is detected by the IR sensor, the serial monitor will display: `"No object detected"`.
- The output will refresh every 5 seconds as specified in the pseudocode, allowing real-time feedback on the sensor's status.

Experiment 2:- Turn on an LED when an obstacle is detected.

Required Sensors

Before making the connections, ensure you have the following sensors and components:

- **IR Sensor**
- **ESP32 Microcontroller**
- **LED**
- **Resistor (220 ohms)**
- **Breadboard and Jumper Wires**

Connections

- **Connect the IR Sensor:**
  - Connect the VCC pin of the IR sensor to the 3.3V output on the ESP32.
  - Connect the GND pin of the IR sensor to the GND (ground) pin on the ESP32.
  - Connect the OUT (signal) pin of the IR sensor to a GPIO pin on the ESP32.
- **Connect the LED:**
  - Connect the anode (longer lead) of the LED to a current-limiting resistor (220 ohms).
  - Connect the other end of the resistor to pin 13 on the ESP32.
  - Connect the cathode (shorter lead) of the LED to the GND (ground) pin on the ESP32.

In Short-

IR Sensor:

- VCC → 3.3V on ESP32
- GND → GND on ESP32
- OUT → GPIO pin on ESP32

LED:

- Anode (longer lead) → Resistor → Pin 13 on ESP32
- Cathode (shorter lead) → GND on ESP32

Pseudo Code

```
1. Define the GPIO pin for the IR sensor.
2. Define the GPIO pin for the LED.
3. Initialize serial communication.
4. Configure the IR sensor pin as INPUT.
5. Configure the LED pin as OUTPUT.
6. Repeat indefinitely:
   a. Read the digital signal from the IR sensor using digitalRead().
   b. If the signal is HIGH (object detected):
         - Turn on the LED.
         - Wait for half a second.
         - Turn off the LED.
         - Wait for half a second.
   c. Else:
         - Print "No object detected" to the serial monitor.
   d. Wait for 1 second before the next iteration.
```

Expected Output

- When an obstacle is detected by the IR sensor (the sensor's OUT pin goes HIGH), the LED connected to pin 13 will turn ON.
- When no obstacle is detected (the sensor's OUT pin goes LOW), the LED will remain OFF.
- The system will continuously monitor for obstacles, and the LED will respond in real time based on the sensor's readings.

Experiment 3: Write a Timer Function to Check if the Object is Continuously Detected for 10 Seconds, and if so, Activate the LED

Components Required

- **IR Sensor**
- **LED**
- **Resistor (220 ohms)**
- **ESP32 or any microcontroller with GPIO pins**
- **Breadboard and jumper wires**

Pseudo Code

```
1. Define the GPIO pin for the IR sensor.
2. Define the GPIO pin for the LED.
3. Initialize a variable (lastDetectionTime) to store the time of the last detection.
4. Initialize a constant (detectionInterval) for the desired detection interval (e.g., 1
5. Initialize serial communication.
6. Configure the IR sensor pin as INPUT.
7. Configure the LED pin as OUTPUT.
8. Repeat indefinitely:
   a. Read the digital signal from the IR sensor using digitalRead().
   b. Print the sensor value to the serial monitor.
   c. If the signal is HIGH (object detected):
         - Check if the object has been detected continuously for 10 seconds:
              - If true:
                   - Turn on the LED.
                   - Print "Object detected for 10 seconds!" to the serial monitor.
              - Else:
                   - Update lastDetectionTime to the current time.
   d. Else:
         - Turn off the LED.
         - Reset lastDetectionTime to the current time.
   e. Wait for a short delay for stability.
```

Explanation

The pseudocode outlines the logic for an object detection system using an IR sensor and an LED. It initializes variables to store the time of the last detection and sets a constant for the desired detection interval, such as 10 seconds. In a continuous loop, the code reads the digital signal from the IR sensor, prints the sensor value to the serial monitor, and checks if an object is detected. If an object is continuously detected for 10 seconds, it turns on the LED and prints an alert. If no object is detected, it turns off the LED and resets the timer. The system then adds a delay to ensure stability before repeating the process. This design allows for the monitoring of continuous object detection and provides visual feedback through the LED. Adjustments can be made based on specific hardware and project needs.

Expected Output

- When the IR sensor detects an object continuously for 10 seconds:
  - The LED turns ON.
  - The serial monitor displays "Object detected for 10 seconds!".
- If no object is detected, the LED stays OFF, and the serial monitor prints "No object detected".

Experiment 4A: Designing a Room Occupancy Counter with Single Door Access

Make a system that can count the number of people present inside a room with a single door (both entry and exit take place through it only) and show this count on your local host.

Required Hardware

- Breadboard
- ESP32 Dev Module
- HC-SR04 sensor
- IR sensor
- Jumper Cables

Diagram Showing Tentative Arrangement of Sensors

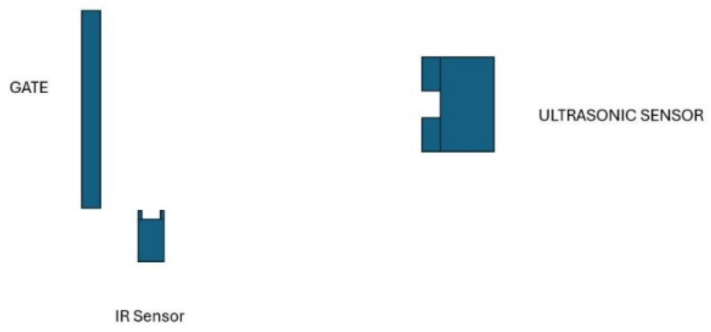


Figure 2: Tentative arrangement of sensors.

Working Principle

- An IR sensor is placed just at the entrance of the gate, and an ultrasonic sensor is placed in front of the gate facing towards it.
- Whenever a person moves in or out, the IR sensor will detect it, and whether to increase or decrease the count is based on the direction of motion, which can be determined by the ultrasonic sensor.

Procedure

- Connect both the sensors to the ESP32.
- Place the sensors in the proper position as shown in the diagram to emulate the given scenario.
- Write the code with proper logic to display the count on the web server.

Pseudocode

For Setting Up Local Host

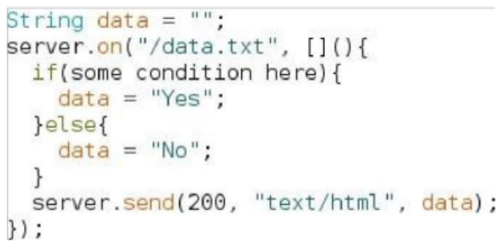
Required Libraries

- `WiFi.h` (built-in) `[Documentation for library]`
- `WebServer.h` in WiFiWebServer on Arduino library manager.

Pseudo-Code and Steps

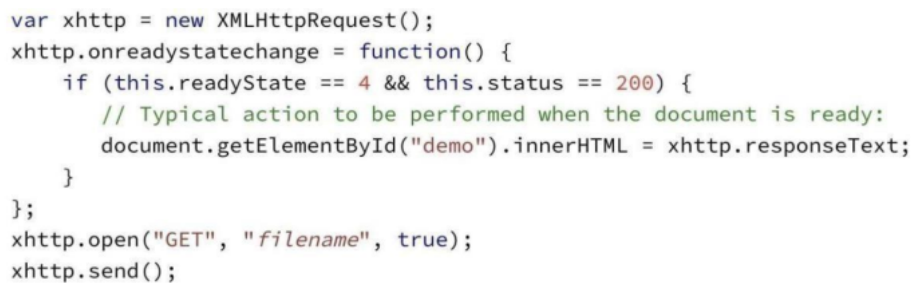
1. Include library and define `ssid` and `password` for your WLAN or hotspot as `const char*`.
2. Define a server as `WebServer server(80);`.
3. In `void setup()`:
   (a) Use `Serial.begin()`.
   (b) Define pin connections for LED using `pinMode()`.
   (c) Print the name of `ssid` connecting to and update the connecting status on the serial monitor using appropriate words.
   (d) Use `WiFi.begin(ssid, password);` to begin the connection and `WiFi.status() != WL_CONNECTED;` [Read Documentation. for meanings of different flags] to check connection status and `WiFi.localIP();` to get the IP to be connected, print all the statuses on the serial monitor.
   (e) Print `WiFi.localIP();` on serial monitor to get a local IP. Copy this local IP and paste into your browser to get the webpage.
   (f) Use `server.on()` to start your server. Pass necessary string arguments. Suppose you want to send a string `str_new` through the server then include:
      - `server.send(200, "text/html", data);`

      Here `200` means gateway, `404` would mean not found, `"text/html"` means the type of data, and the last argument will be the string `str_new` in this case.

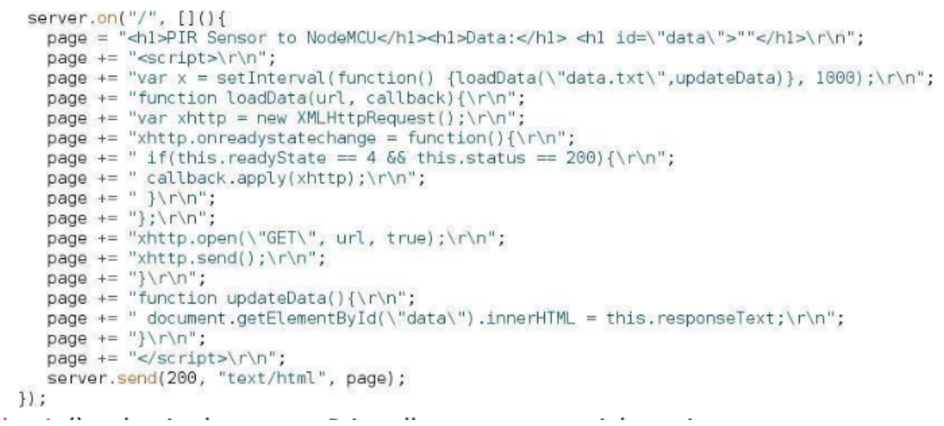


```
String data = "";
server.on("/data.txt", [](){
  if(some condition here){
    data = "Yes";
  }else{
    data = "No";
  }
  server.send(200, "text/html", data);
});
```

   (g) Use this same method to pass HTML/JS inputs to the server. We have to make a string that contains HTML code for the webpage. JavaScript can be inserted in HTML using the `<script>` tag. All modern browsers have a built-in `XMLHttpRequest` object to request data from a server. This is used to update the webpage without reloading it. It can be used to directly receive and send data to the server in the background.



```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
       // Typical action to be performed when the document is ready:
       document.getElementById("demo").innerHTML = xhttp.responseText;
    }
};
xhttp.open("GET", "filename", true);
xhttp.send();
```



```
server.on("/", [](){
  page = "<h1>PIR Sensor to NodeMCU</h1><h1>Data:</h1> <h1 id=\"data\">""</h1>\r\n";
  page += "<script>\r\n";
  page += "var x = setInterval(function() {loadData(\"data.txt\",updateData)}, 1000);\r\n";
  page += "function loadData(url, callback){\r\n";
  page += "var xhttp = new XMLHttpRequest();\r\n";
  page += "xhttp.onreadystatechange = function(){\r\n";
  page += " if(this.readyState == 4 && this.status == 200){\r\n";
  page += " callback.apply(xhttp);\r\n";
  page += " }\r\n";
  page += "};\r\n";
  page += "xhttp.open(\"GET\", url, true);\r\n";
  page += "xhttp.send();\r\n";
  page += "}\r\n";
  page += "function updateData(){\r\n";
  page += " document.getElementById(\"data\").innerHTML = this.responseText;\r\n";
  page += "}\r\n";
  page += "</script>\r\n";
  server.send(200, "text/html", page);
});
```

(h) server.begin() to begin the server. Print all statuses on serial monitor.

4. In void loop() update motion sensor readings and include server.handleClient();. Print all the statuses on the serial monitor.

For Working of the Circuit

```
1. Initialize Variables:
   - Set count = 0
   - Set previous distance = 0

2. Main Loop:
   - Enter an infinite loop:

3. Read Sensors:
   - Read the status of the IR sensor (ir_status)
   - Read the distance from the ultrasonic sensor (ultrasonic distance)

4. Motion Detection:
   - If ir_status is HIGH (indicating motion detected):
     - Check the change in distance from the ultrasonic sensor:
       - If ultrasonic distance is greater than previous distance: Increment count.
       - If ultrasonic distance is less than previous distance: Decrement count.

5. Update Previous Distance:
   - Set previous distance = ultrasonic distance

6. Display or Transmit Count:
   - Display the current count value on the web server (process to set up is described i

7. Delay:
   - Introduce a delay (e.g., delay(100) or set accordingly) to prevent rapid counting.

8. Repeat Loop:
   - Go back to the beginning of the loop to continue monitoring.
```

Expected Output

You must count the number of persons inside the room correctly and show that on the local host.

Experiment 4B: Making Local IP Accessible Globally Using Third-Party Applications

Turn your local IP or local host into a global link using 3rd-party applications.

Ngrok provides a real-time web UI where you can introspect all HTTP traffic running over your tunnels. Replay any request against your tunnel with one click. Visit `https://ngrok.com/` for documentation.

Steps

1. Sign up for ngrok.
2. Look into `https://dashboard.ngrok.com/get-started/setup` for setup.
3. Create a link by running the following command in the terminal:

   ```
       ./ngrok http <your-local-ip>
   ```

   For example:

   ```
       ./ngrok http 10.62.35.34
   ```

4. Embed your link into a QR code for the TA to easily access your web server.

Expected Output

After running the ngrok command, a public URL (e.g., https://1234abcd.ngrok.io) will be generated. This URL will allow global access to your local web server. You can embed this URL into a QR code for easy access. The ngrok dashboard will provide real-time traffic monitoring, and you can replay requests directly from the dashboard.