International Institute of Information Technology

Introduction to IoT

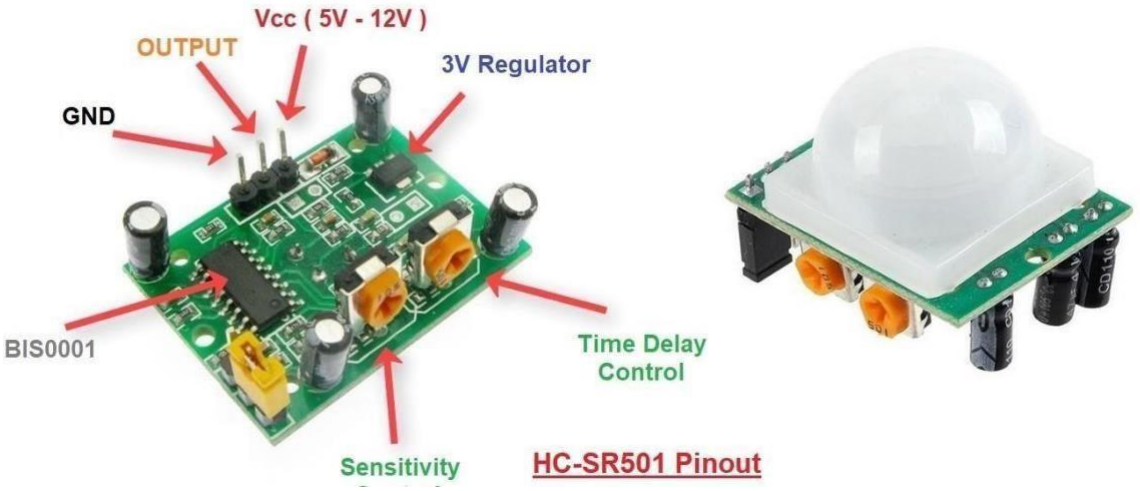
Spring2024 Lab

LAB-6

Overview:

In this lab session, we will learn to use a motion sensor to detect an object and flag off the detection information over WiFi to a server and use that information to activate a LED connected to our ESP32.

Motion Sensor:



The PIR (Passive Infrared) Motion Sensor Detector Module (HC-SR501) allows us to sense motion. It is almost always used to detect the motion of a human body within the sensor's range. It is often referred to as "PIR", "Pyroelectric", "Passive Infrared" or as the "IR Motion" sensor.

HC-SR501 is basically a motion detector sensor, which uses infrared waves for detection of an object. It is an automatic control device having large sensitivity and high reliability. It is used in auto-sensing control devices, where we need to perform motion detection.

The Specifications are given as:

- Wide Working Voltage Range: DC 4.5V - 20V
- Current Drain: <60µA
- Detection Angle: < 140°

- Detection Distance: 3 to 7m (can be adjusted)
- Blockade Time: 2.5s (Default)
- Work Temperature: -20°C to +80°C

Pinout:

- VCC -
  Input Voltage is +5V for typical applications. Can range from 4.5V to 12V.
- High/Low Output (D_{out}) -

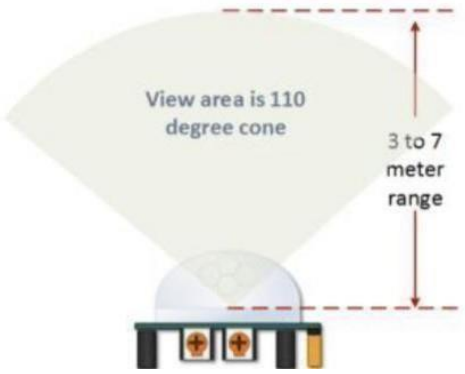
Digital Pulse is HIGH (33V) when triggered (Motion detected), LOW (0V) when idle (no motion is detected)

- Ground -
  Connected to ground of the circuit.

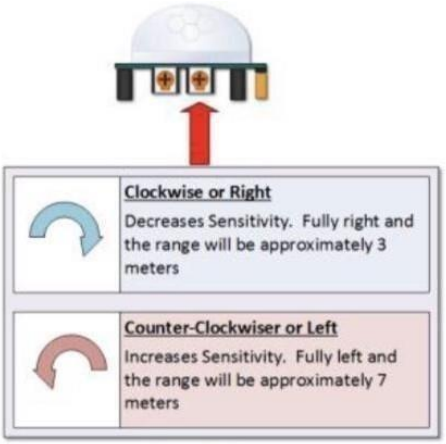
Components:

- Time Delay Control – Orange colored screw to regulate the delay time. • Sensitivity Control – Orange colored screw to regulate sensitivity
- Regulator- An IC (HT7133-1) that regulates voltage flow.
- BIS0001- a PIR controller that works on analog mixing digital techniques.

PIR sensors might need some time to warm-up and stabilize before they accurately detect motion. As they are designed to detect changes in infrared radiation, they hence need to adjust to the surrounding temperature before experimentation. Although the exact warm-up time can vary depending on the sensor model, environment etc., 30 seconds is a fair time to give to your sensor to calibrate.

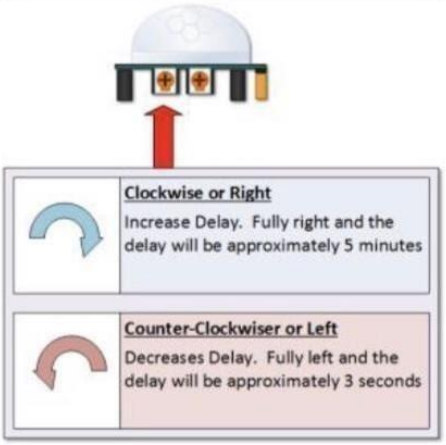


The device will detect motion inside a 110 degree cone with a range of 3 to 7 meters.



PIR Range (Sensitivity) Adjustment

As mentioned, the adjustable range is from approximately 3 to 7 meters.



Time Delay Adjustment

The time delay adjustment determines how long the output of the PIR sensor module will remain high after detection motion. The range is from about 3 seconds to five minutes.

The output of this device will go LOW (or OFF) for approximately 3 seconds AFTER this time delay completes. In other words, ALL motion detection is blocked during this three-second period.

For Example – Imagine that you're in the single trigger mode (see below) and your time delay is set to 5 seconds.

- The PIR will detect motion and set it high for 5 seconds.
- After 5 seconds, the PIR will set its output LOW for about 3 seconds.
- During the three seconds, the PIR will not detect motion and detected motion will once again set the output high and the output will remain on as dictated by the Time Delay Adjustment and trigger mode selection.

(Note: The default delay and sensitivity are sufficient to perform this experiment. Don't regulate the controller unnecessarily as it is very sensitive)

WiFi Module on ESP32:

ESP32 comes with a Wifi and bluetooth module embedded in it. This WiFi module can connect to your WLAN or mobile hotspot. This is to be used to start a local host on your system which can be further accessed by devices on your WLAN or mobile

hotspot. The latter aim of this manual is to make the web server accessible anywhere in the world over the internet.

EXPERIMENT 1A

Use a PIR motion sensor to detect motion and update the info on a webpage using the server approach: (Please refer to Experiment 0 given on the last page of the manual).

Required Hardware:

- Breadboard
- ESP32
- PIR Motion Sensor
- Jumper Wires

Circuit Connections:

- Connect the PIR sensor to input pin on ESP32
- Connect VCC to Vin and GND to ground on ESP32

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 localIP. Copy this localIP and paste into your browser to get the webpage.

f. User 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 <script> tag. All modern browsers have a builtin 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 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.

Expected Output:

You must show PIR sensor reading (digital) on a webpage. This value can be a boolean expression or just text for example “Motion Detected” or “No Motion Detected”. Pseudocode is just for your ease to understand what is to be done. Deviating from the pseudocode but still fulfilling the requirements won’t incur any penalty.

EXPERIMENT 1B

In this experiment we learn to operate the PIR motion sensor in the multiple trigger mode and increment a counter on the webpage when the motion is detected.

In Single Trigger Mode, the Output goes HIGH when motion is detected. It goes LOW for some time and again gives HIGH if the object remains in motion. So, the PIR sensor in Single Trigger Mode gives HIGH/LOW pulses if the object is in continuous motion.

In the multiple Trigger mode, the output goes HIGH when motion is detected and stays HIGH till the object stays in motion. If the object’s motion is halted, the PIR gives HIGH state for up to some specified delay.

To switch from the single trigger to multiple trigger mode in the HC-SR501 module, you need to locate the jumper labeled ‘L’ on the back of the module. This jumper is

used to select the trigger mode of the module. To switch to the multiple trigger mode, you need to remove the jumper from 'L' pins and place it on the H-pins. This will change the trigger mode from single to multiple. After changing the jumper, you may need to power cycle the module to ensure that the change takes effect.

The task at hand is to modify the code from the experiment above and operate the provided PIR sensor in the Multiple Trigger Mode. We aim to update a counter on the webpage according to the number of motions detected by the sensor. The pseudocode for updating information on the webpage remains the same.

EXPERIMENT 2

Light up an LED through a webpage using client approach. (Please refer to Experiment 0 given at the last page of this manual)

Required Hardware:

- Breadboard
- ESP32
- LED (of any color)
- Jumper Wires

Circuit Connections

- Connect LED with similar output pins in parallel fashion on ESP32

Required Libraries:

- WiFi.h (built-in) [Documentation for library]

Pseudocode:

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 localIP. Copy this localIP and paste into your browser to get the webpage.

f. server.begin() to start the server.

4. In void loop()

a. Make a client using WiFiClient client = server.available();

b. Use if(client){}, then make a blank string String currentLine = “”; then use while (client.connected()) {} and if (client.available()) {} along with appropriate serial monitor outputs.

c. Make a character char c = client.read(); and Serial.write(c);

d. Understand and follow the following code snippet to make an HTML page.

e. The client.print function sends data in HTML format to the local host and this HTML input is read by the browser to show the output on screen as if it’s an HTML document. Note that here you do not have an HTML document. This is just done by passing HTML codes to your local host using client.print.

```
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();
    // the content of the HTTP response follows the header:
    client.print("<h1>Click <a href=\"/H\">here</a> turn the LED on</h1><br>");
    client.print("<h1>Click <a href=\"/L\">here</a> turn the LED off<h1><br>");
    // 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
}
```

f. Look into the request from the client to light the LED.

g. Here /L creates a next page to the link. Suppose you get 10.62.35.34 as your localIP. The above snipper will make 10.62.35.34/L as a webpage that shows that current output is on LOW.

```
if (currentLine.endsWith("GET /L")) {
  digitalWrite(26, LOW);               // GET /L turns the LED off
}
```

Similarly use ("GET /H") for HIGH

h. Use client.stop(); to stop the connection from the existing client and print on the serial monitor.

Expected Output:
You must make links (or Buttons) for controlling LED on a webpage using HTML. Pseudocode is just for your ease to understand what is to be done. Deviating from pseudocode but still fulfilling the requirements won't incur any penalty.

BONUS EXPERIMENT

Turn your localIP 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. **https://ngrok.com/ for documentation.**

Steps:

1. Signup for ngrok.
2. Look into https://dashboard.ngrok.com/get-started/setup for setup.
3. Create a link from running in the command terminal
   ./ngrok http <your local ip>
   For example ./ngrok http 10.62.35.34
4. Send the link to your respective TA to get your setup verified.

If you face difficulties in using ngrok, you may use other applications like localtunnel etc. Please see their documentation.