# Código Fuente

## ❖ Objetivo principal

**Monitorear en tiempo real los niveles de luz y humedad de una planta utilizando sensores, enviar los datos al servidor en formato JSON para proporcionar al usuario información actualizada y brindar recomendaciones personalizadas a través de WhatsApp.**

## ❖ Secciones importantes

## Software del dispositivo

### *Inclusión de bibliotecas*

Codigogrupo.ino | gestos.h

```
1   #include <Wire.h>
2   #include <BH1750.h>
3   #include <WiFi.h>
4   #include <HTTPClient.h>
5   #include "M5Atom.h"
6   #include "gestos.h" // Archivo que contiene las definiciones de las caras
```

Estas bibliotecas son fundamentales para el funcionamiento del dispositivo, ya que permiten la comunicación I2C con el sensor de luz (*Wire.h* y *BH1750.h* ), la conexión WiFi del ESP32 ( *WiFi.h* ), el envío de datos al servidor mediante HTTP (*HTTPClient.h* ), y la configuración de ATOM Matrix junto con el control de sus expresiones faciales ( *M5Atom.h* y *gestos.h* ).

### *Configuración de conexión WiFi*

```
13  const char* ssid = "HONOR";        // Nombre de usuario (SSID de tu red)
14  const char* password = "12345678";     // Contraseña de la red
15  const char* serverName = "https://stingray-app-onxu7.ondigitalocean.app/update_plant_state";
```

Estos parámetros son claves para la conectividad del dispositivo, como el **SSID** y la **contraseña** de la red WiFi, necesarios para establecer la conexión inalámbrica. Además, especifica la dirección del servidor mediante la variable **serverName**, que contiene la URL donde se enviarán los datos de los sensores para su procesamiento y almacenamiento.

***Lectura de sensores y lógica***

```
39    float lux = lightMeter.readLightLevel();
40    int soilValue = analogRead(soilMoisturePin);
41
42    // Clasificación de humedad del suelo
43    String humidityLevel = (soilValue < 940) ? "Escaso" :
44                           (soilValue > 1500) ? "Excedente" : "Óptimo";
45
46    // Clasificación de luz
47    String lightLevel = (lux < 200) ? "Escaso" :
48                        (lux > 1200) ? "Excedente" : "Óptimo";
```

La función **lightMeter.readLightLevel()** obtiene el nivel de luz ambiental en lux, mientras que **analogRead**(soilMoisturePin) mide el valor analógico del sensor capacitivo de humedad del suelo. Luego, clasifica ambos párametros en "Escaso", "Óptimo" o "Excedente" según rangos establecidos, y guarda las categorías en variables de texto.

***Expresiones faciales***

```
50    if ((humidityLevel == "Escaso" && lightLevel == "Escaso") ||
51        (humidityLevel == "Escaso" && lightLevel == "Excedente") ||
52        (humidityLevel == "Excedente" && lightLevel == "Escaso") ||
53        (humidityLevel == "Excedente" && lightLevel == "Excedente")) {
54      show_face(FACE_SAD); // Cara triste
55    } else if ((humidityLevel == "Escaso" && lightLevel == "Óptimo") ||
56               (humidityLevel == "Excedente" && lightLevel == "Óptimo") ||
57               (humidityLevel == "Óptimo" && lightLevel == "Excedente") ||
58               (humidityLevel == "Óptimo" && lightLevel == "Escaso")) {
59      show_face(FACE_NEUTRONE); // Cara neutra
60    } else if (humidityLevel == "Óptimo" && lightLevel == "Óptimo") {
61      show_face(FACE_HAPPY); // Cara feliz
62    }
```

Las expresiones faciales de ATOM Matrix se ajustan automáticamente en función de los niveles de luz y humedad detectados. Si las condiciones son óptimas, muestra una cara feliz, mientras que, si los niveles son escasos o excedentes, se muestra una cara triste y si ambos parámetros difieren, se muestra una cara neutra, proporcionando una representación visual del estado de la planta.

***Envío de datos al servidor***

```
64  if (WiFi.status() == WL_CONNECTED) {
65    HTTPClient http;
66
67    // Crea el JSON con los datos de luz y humedad
68    String jsonPayload = "{";
69    jsonPayload += "\"lux\": " + String(lux) + ",";
70    jsonPayload += "\"soil_moisture\": " + String(soilMoisturePercentage) + ",";
71    jsonPayload += "\"light_level\": \"" + lightLevel + "\",";
72    jsonPayload += "\"humidity_level\": \"" + humidityLevel + "\"";
73    jsonPayload += "}";
74
75    http.begin(serverName); // Inicia la conexión con el servidor
76    http.addHeader("Content-Type", "application/json"); // Especifica JSON como el tipo de contenido
77    int httpResponseCode = http.POST(jsonPayload); // Realiza la solicitud POST con el JSON
78
79    // Verifica la respuesta del servidor
80    if (httpResponseCode > 0) {
81      String response = http.getString();
82      Serial.println("Respuesta del servidor: " + response);
83    } else {
84      Serial.println("Error en la solicitud POST: " + String(httpResponseCode));
85    }
86
```

Una vez que la conexión WiFi está establecida, se recopilan los datos (intensidad de luz, humedad del suelo, y niveles de luz y humedad) y se envía al servidor en formato JSON utilizando una solicitud HTTP POST. Este paso permite que el servidor reciba y procese los datos para futuras acciones.

***Visualización en el monitor serial***

```
Respuesta del servidor: {"message":"Plant state updated successfully!"}

Luz: 19.17 lx, Humedad: 100.00%
Respuesta del servidor: {"message":"Plant state updated successfully!"}

Luz: 19.17 lx, Humedad: 99.00%
```

Se visualiza en tiempo real los valores de los sensores, como la intensidad de luz y la humedad del suelo, facilitando la depuración del sistema y asegurando que los datos se estén obteniendo correctamente antes de ser enviados al servidor.

***Código completo***

```
#include <Wire.h>
#include <BH1750.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include "M5Atom.h"
#include "gestos.h" // Archivo que contiene las definiciones de las caras

BH1750 lightMeter;
const int soilMoisturePin = 33; // Pin analógico para el sensor capacitivo de humedad

const char* ssid = "OPPO A79 5G";        // Nombre de usuario (SSID de tu red)
const char* password = "Milena2708";     // Contraseña de la red
const char* serverName = "https://stingray-app-onxu7.ondigitalocean.app/update_plant_state";
// URL del servidor

void setup() {

  M5.begin(true, false, true);
  delay(100);
  M5.IMU.Init();
  Wire.begin(25, 21); // GPIO21 (SCL), GPIO25 (SDA)
  lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE);
  Serial.begin(115200);
  pinMode(soilMoisturePin, INPUT);

  WiFi.begin(ssid, password);
  Serial.print("Conectando a WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("\nConectado a WiFi!");
}

void loop() {
  // Lee la luz y la humedad
  float lux = lightMeter.readLightLevel();
  int soilValue = analogRead(soilMoisturePin);

  String humidityLevel = (soilValue < 940) ? "Escaso" :
                         (soilValue > 1500) ? "Excedente" : "Óptimo";

  String lightLevel = (lux < 200) ? "Escaso" :
                      (lux > 1200) ? "Excedente" : "Óptimo";
```

```
  // Cambia las expresiones faciales según los niveles
  if ((humidityLevel == "Escaso" && lightLevel == "Escaso") ||
      (humidityLevel == "Escaso" && lightLevel == "Excedente") ||
      (humidityLevel == "Excedente" && lightLevel == "Escaso") ||
      (humidityLevel == "Excedente" && lightLevel == "Excedente")) {
    show_face(FACE_SAD); // Cara triste
  } else if ((humidityLevel == "Escaso" && lightLevel == "Óptimo") ||
             (humidityLevel == "Excedente" && lightLevel == "Óptimo") ||
             (humidityLevel == "Óptimo" && lightLevel == "Excedente") ||
             (humidityLevel == "Óptimo" && lightLevel == "Escaso")) {
    show_face(FACE_NEUTRONE); // Cara neutra
  } else if (humidityLevel == "Óptimo" && lightLevel == "Óptimo") {
    show_face(FACE_HAPPY); // Cara feliz
  }

  // Enviar datos al servidor si el WiFi está conectado
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    String jsonPayload = "{";
    jsonPayload += "\"lux\": " + String(lux) + ",";
    jsonPayload += "\"soil_moisture\": " + String(soilValue) + ",";
    jsonPayload += "\"light_level\": \"" + lightLevel + "\",";
    jsonPayload += "\"humidity_level\": \"" + humidityLevel + "\"";
    jsonPayload += "}";

    http.begin(serverName); // Inicia la conexión con el servidor
    http.addHeader("Content-Type", "application/json"); // Especifica JSON como el tipo de
contenido
    int httpResponseCode = http.POST(jsonPayload); // Realiza la solicitud POST con el JSON

    // Verifica la respuesta del servidor
    if (httpResponseCode > 0) {
      String response = http.getString();
      Serial.println("Respuesta del servidor: " + response);
    } else {
      Serial.println("Error en la solicitud POST: " + String(httpResponseCode));
    }

    http.end(); // Finaliza la conexión
  } else {
    Serial.println("Error: No conectado a WiFi.");
  }
  // Mostrar los valores en el monitor serial
  Serial.print("Luz: ");
  Serial.print(lux);
  Serial.print(" lx, Humedad: ");
  Serial.print(soilValue);
```

```
  Serial.println(" unidades analógicas");

  delay(10000); // Esperar 10 segundos antes de la siguiente lectura
}
```

**Código: "gestos.h"**

```
#include "Arduino.h"
#include "M5Atom.h"

int FACE_NEUTRONE[5][5] = {
  {0,0,0,0,0},
  {0,1,0,1,0},
  {0,0,0,0,0},
  {1,1,1,1,1},
  {0,0,0,0,0}
};

int FACE_HAPPY[5][5] = {
  {0,0,0,0,0},
  {0,1,0,1,0},
  {0,0,0,0,0},
  {1,0,0,0,1},
  {0,1,1,1,0}
};

int FACE_SAD[5][5] = {
  {0,0,0,0,0},
  {0,1,0,1,0},
  {0,0,0,0,0},
  {0,1,1,1,0},
  {1,0,0,0,1}
};

void show_face(int _face[5][5]){
  int pixelIndex = 0;
  for(int i = 0; i < 5; i++){
    for(int j = 0; j < 5; j++){
      pixelIndex = i * 5 + j; // Calcula el índice del píxel en una matriz 5x5
      if (_face[i][j] == 1) {
        M5.dis.drawpix(pixelIndex, 0xFFFFFF); // Blanco
      } else {
        M5.dis.drawpix(pixelIndex, 0x000000); // Apagado
      }
    }
  }
}
```

# Software del servidor

***Ruta para procesar***

```python
@app.route('/alert', methods=['POST'])
def alert():
    data = request.get_json()  # Extrae los datos enviados
    luz = data.get('luz')  # Nivel de luz
    humedad = data.get('humedad')  # Nivel de humedad

    if luz and humedad:
        alert_message = f"Alerta: EL NIVEL DE LUZ ES {luz} Y LA HUMEDAD ES {humedad}."
        gpt_alert_response = gpt_alert(alert_message)  # Procesa la alerta con GPT
        send_message(gpt_alert_response)  # Envía un mensaje usando Twilio
        return jsonify({"message": "Alert processed successfully!"}), 200
    else:
        return jsonify({"error": "Missing 'luz' or 'humedad' in the data"}), 400

```

Define una ruta que escucha solicitudes POST en /alert. Se utiliza para recibir datos relacionados con **alertas de luz y humedad**.

Extrae un JSON del cuerpo de la solicitud y obtiene los valores de luz y humedad.

1. **Verificación de datos:**
   Si ambos datos están presentes, genera un mensaje de alerta con los valores recibidos.
2. **Procesamiento de alerta:**
   El mensaje se pasa a la función gpt_alert para procesarlo, y luego se envía a través de send message (simulando un envío por Twilio).
3. **Respuesta:**
   - Si los datos son válidos, devuelve un mensaje indicando que la alerta se procesó exitosamente (código 200).
   - Si faltan datos, responde con un error (código 400).

***Ruta para recibir datos y responder con GPT***

```python
@app.route('/receive-data', methods=['POST'])
def receive_data():
    incoming_que = request.values.get('Body', '').lower()  # Mensaje recibido por WhatsApp
    from_user_number = request.values.get('From', '').lower()  # Número del usuario
    file_path = f"uploads/{from_user_number}.txt"
```

Este fragmento define una ruta en Flask para gestionar solicitudes POST en /receive-data, procesando mensajes entrantes de WhatsApp. Extrae el contenido del mensaje ( Body) y el número del remitente ( From), utilizando el número para identificar un archivo de historial asociado, almacenado en una carpeta de

```python
    # Carga mensajes históricos
    history_messages = []
    if os.path.exists(file_path):
        with open(file_path, 'r') as file:
            history_messages = file.readlines()

    # Agrega el mensaje actual al historial
    history_messages.insert(0, "user: " + incoming_que + '\n')
    history_messages = history_messages[-15:]  # Limita a los últimos 15 mensajes
    full_chat = ''.join(history_messages)

    # Respuesta generada con GPT
    answer = gpt_response(incoming_que, full_chat)
    history_messages.insert(0, "cayeplant: " + answer + '\n')

    # Guarda el historial actualizado
    with open(file_path, 'w') as file:
        file.writelines(history_messages)

    # Respuesta a Twilio
    bot_resp = MessagingResponse()
    msg = bot_resp.message()
    msg.body(answer)
    return str(bot_resp)
```

En esta parte se carga el historial de mensajes de un archivo asociado al usuario, agrega el mensaje recibido, lo limita a los últimos 15 mensajes y lo combina en un historial completo. Luego, genera una respuesta con gpt_response, actualiza el historial con esta respuesta y lo guarda en el archivo. Finalmente, envía la respuesta generada al usuario a través de Twilio.

***Método para procesar alertas con GPT***

```python
def gpt_alert(alert_message):
    # Simulación de llamada a la API de OpenAI (simplificado)
    response = f"ALERTA URGENTE: {alert_message}. Por favor, actúa rápido para proteger tus plantas."
    return response
```

Este método genera una alerta personalizada utilizando un mensaje dado, simulando una respuesta de la API de GPT, y devuelve un texto de advertencia urgente para proteger las plantas.

***Método para responder mensajes con GPT***

```python
def gpt_response(user_message, full_chat):
    # Simulación de respuesta usando historial
    response = f"Hola, amigo humano 🌿. Según el chat: '{full_chat}'. ¡Recuerda darme agua y luz! ☀️💧"
    return response
```

Simula una respuesta generada por GPT basada en un mensaje del usuario y el historial completo del chat, desarrollando una respuesta personalizada con un tono amigable.

***Método para enviar un mensaje (simplificado)***

```
61  def send_message(answer):
62      print(f"Mensaje enviado: {answer}")
```

Este método simula el envío de un mensaje mostrando la proporción del contenido.

***Método para leer archivos de texto***

```
65  def read_txt_file(filename):
66      try:
67          with open(filename, 'r') as file:
68              return file.read()
69      except Exception as e:
70          print(f"Error leyendo archivo: {e}")
71          return None
```

Lee el contenido de un archivo de texto y lo devuelve; si se produce un error durante la lectura, muestra el mensaje de error y retorna None.

***Inicia la aplicación***

```
73  # Inicia la aplicación
74  if _name_ == '_main_':
75      app.run(debug=True)
```

Inicia la ejecución de la aplicación Flask solo si el archivo es el programa principal, activando el modo de depuración para facilitar el desarrollo al mostrar errores detallados y permitidos.

***Visualización del servidor***