Sistema de Monitoreo Ambiental con ESP32, OLED, Sensores Avanzados y Servidor Flask en Termux

Cesar Mateo Gonzalez Quiroga

***Abstract*—Este documento describe el diseño e implementación de un sistema de monitoreo ambiental embebido, versátil y de bajo consumo, basado en el microcontrolador ESP32. El sistema incorpora un sensor ambiental avanzado (BME280/BME680) para mediciones precisas de temperatura, humedad y presión (y calidad del aire en el BME680), junto con un sensor de luz ambiental BH1750. Los datos recopilados se visualizan en tiempo real en una pantalla OLED (SH1106) y se transmiten vía WiFi a un servidor Flask alojado en un dispositivo Android utilizando Termux. Para optimizar el consumo de energía y prolongar la vida útil del dispositivo, se implementa el modo de "Deep Sleep" del ESP32, permitiendo períodos de inactividad programada. Además, el sistema incluye sincronización horaria NTP para timestamping preciso y robustas capacidades de reconexión WiFi automática. Este proyecto demuestra una solución IoT completa, desde la adquisición de datos y su visualización local, hasta la comunicación con un backend móvil y la gestión eficiente de energía, sentando las bases para aplicaciones industriales y de monitoreo a largo plazo.**

***Index Terms*—ESP32, IoT, monitoreo ambiental, Flask, Termux, Deep Sleep, BME280, BME680, BH1750, OLED, NTP, eficiencia energética.**

I. Objetivo

Diseñar y construir un sistema embebido basado en ESP32 que mida temperatura, humedad, presión (y calidad del aire si se usa BME680) y luz ambiental, muestre los datos en una pantalla OLED y los envíe a un servidor Flask corriendo en un dispositivo Android mediante Termux. El sistema incluye sincronización horaria NTP, reconexión WiFi automática y una gestión de energía eficiente mediante el modo Deep Sleep para prolongar su autonomía y vida útil. La selección de sensores más sofisticados busca elevar la precisión y fiabilidad, acercando el sistema a estándares de aplicaciones industriales.

II. Componentes Utilizados

Los principales componentes de hardware y software utilizados para la construcción de este sistema de monitoreo ambiental incluyen:

- **ESP32 Dev Module:** Microcontrolador principal con conectividad WiFi y Bluetooth Low Energy (BLE), equipado con un procesador de doble núcleo, ideal para aplicaciones IoT.
- **Sensor BME280/BME680:** Sensores de Bosch Sensortec para la medición precisa de temperatura, humedad y presión barométrica. El BME680 extiende estas capacidades con la medición de Compuestos Orgánicos Volátiles (VOCs) para inferir la calidad del aire. Ambos se comunican vía I2C.
- **Sensor BH1750:** Sensor de luz digital que proporciona mediciones de iluminancia en lux, también utilizando la interfaz I2C.
- **Pantalla OLED SH1106 (128x64):** Una pantalla monocromática compacta de alta resolución y bajo consumo de energía, ideal para la visualización local de datos, con interfaz I2C.
- **Smartphone con Termux:** Un dispositivo Android que opera como un servidor Flask local, encargado de recibir, procesar y almacenar los datos ambientales transmitidos desde el ESP32.
- **Librerías Arduino:** Colección de bibliotecas de software esenciales para la programación del ESP32, incluyendo: 'WiFi' para conectividad de red, 'HTTPClient' para realizar peticiones HTTP, 'Wire' para comunicación I2C, 'Adafruit GFX' y 'Adafruit SH110X' para el control de la pantalla OLED, 'Adafruit BME280/BME680' para la interfaz con los sensores ambientales, 'BH1750' para el sensor de luz, y la librería 'Time' para la sincronización NTP.
- **Librerías Python (Termux):** Para el servidor Flask, se emplean las librerías 'Flask' para el framework web y 'requests' para posibles futuras interacciones bidireccionales.

III. Diseño del Sistema

El sistema se compone de un microcontrolador ESP32 actuando como el nodo principal de sensado y comunicación. Se conecta a una red WiFi para acceder a internet (sincronización NTP) y para enviar datos a un servidor local. Los datos ambientales son capturados por sensores de alta precisión (BME280/BME680 y BH1750) a través del bus I2C. Una pantalla OLED permite la visualización en tiempo real de estas métricas, junto con la hora actual. La estrategia de eficiencia energética se basa en el ciclo de "Deep Sleep" del ESP32, donde el microcontrolador entra en un estado de bajo consumo entre mediciones, despertándose solo para realizar las tareas programadas (lectura, envío, visualización) antes de volver a dormir.

A. Diagrama de Conexiones

Se omite el diagrama específico en este informe para brevedad, pero las conexiones clave son:

Fig. 1. Sensor de temperatura y humedad (referencial; idealmente una imagen del BME280/BME680).

Fig. 2. Ejemplo de un sensor de luz BH1750.

- **ESP32 (GPIO 21 - SDA, GPIO 22 - SCL):** Conectados a los pines SDA y SCL del módulo OLED, el sensor BME280/BME680 y el sensor BH1750.
- **Alimentación:** Los módulos (OLED, BME280/BME680, BH1750) son alimentados por los pines 3.3V o 5V (según requiera cada módulo) y GND del ESP32.
- **Conexión USB:** Para programación y alimentación inicial del ESP32.

- **Nota sobre el Deep Sleep:** Para despertar el ESP32 del Deep Sleep, se utiliza generalmente un temporizador interno o un pin de "wake-up" (ej. RTC$_GPIOs$).

IV. Implementación del Código

A. Código del ESP32 (Arduino IDE)

Este código gestiona la lectura de los sensores BME280/BME680 y BH1750, la conexión WiFi, la sincronización NTP, la visualización en la pantalla OLED y el envío de datos al servidor Flask. Se incorpora la funcionalidad de Deep Sleep para la gestión de energía.

```
#include <Arduino.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>
#include <Adafruit_BME280.h> // Para BME280. Si usas BME680, ser a Adafruit_BME680.h
#include <Adafruit_Sensor.h> // Necesario para Adafruit BME280/BME680
#include <BH1750.h>          // Para sensor de luz BH1750
```

Fig. 3. Ejemplo de una pantalla OLED SH1106 de 128x64 píxeles.

```

// --- Definiciones para la pantalla OLED ---
#define I2C_ADDRESS 0x3C   // Direcci n I2C t pica para OLED (0x3C o 0x3D)
#define SCREEN_WIDTH 128   // Ancho de la pantalla OLED en p xeles
#define SCREEN_HEIGHT 64   // Alto de la pantalla OLED en p xeles
#define OLED_RESET -1      // No usar pin de reset (para I2C)

// Objeto de la pantalla OLED ( GLOBAL !)
Adafruit_SH1106G display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// --- Logo de Wi-Fi (Ejemplo 16x16 pixels - REEMPLAZAR CON TU PROPIO BITMAP GENERADO) ---
const unsigned char PROGMEM wifi_logo_bits[] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
#define WIFI_LOGO_WIDTH 16
#define WIFI_LOGO_HEIGHT 16

// --- Credenciales de la red WiFi ---
const char* ssid = "FAMILIA-GONZALEZ";
const char* pass = "1002391401";

// --- Definiciones para la conexi n a Termux ---
const char* serverIp = "192.168.1.6"; // IP de tu tel fono con Termux
const int serverPort = 5000;

// --- Configuraci n NTP para la hora en tiempo real ---
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = -5 * 3600; // GMT-5 para Colombia
const int daylightOffset_sec = 0;

// --- Periodo de Deep Sleep (en microsegundos) ---
// 5 minutos = 300 segundos = 300 * 1000 * 1000 = 300,000,000 microsegundos
// Para pruebas, puedes usar un valor menor, ej. 30 segundos = 30 * 1000 * 1000
const uint64_t uS_TO_S_FACTOR = 1000000;  /* Conversion factor for micro seconds to seconds */
const uint64_t TIME_TO_SLEEP = 5 * 60 * uS_TO_S_FACTOR; /* Time ESP32 will go to sleep (in seconds) */

// --- Objetos de sensores ---
// Usa uno u otro, no ambos a la vez
Adafruit_BME280 bme; // Objeto para BME280 (Si usas BME680, cambia a Adafruit_BME680)
```

```
BH1750 bh1750; // Objeto para sensor de luz BH1750

void setup() {
  Serial.begin(115200);
  Serial.println("
      ------------------------------------");
  Serial.println("Iniciando ESP32...");
  Serial.println("
      ------------------------------------");

  // --- Inicializaci n de la pantalla OLED ---
  Wire.begin(); // Inicializa el bus I2C
  if (!display.begin(I2C_ADDRESS, true)) {
    Serial.println(F("Error al iniciar la pantalla
        OLED. Verifica conexiones y direcci n I2C."
        ));
    while (true);
  }
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SH110X_WHITE);
  display.setCursor(0, 0);
  display.println("Iniciando...");
  display.println("Sensores & WiFi");
  display.display();

  // --- Inicializaci n de sensores I2C ---
  // BME280/BME680
  if (!bme.begin(0x76)) { // La direcci n I2C
      com n es 0x76 o 0x77
    Serial.println("Error al encontrar BME280/BME680
        . Verifica conexiones o direcci n I2C.");
    display.clearDisplay(); display.setCursor(0,0);
        display.println("Error BME!"); display.
        display();
    // Considerar un while(true) o intentar de nuevo
  } else {
    Serial.println("BME280/BME680 encontrado.");
    bme.setSamplingForFixedMode(); // Configura el
        muestreo para lectura simple
  }

  // BH1750
  if (!bh1750.begin(BH1750::CONTINUOUS_HIGH_RES_MODE
      )) { // Modo de alta resoluci n
    Serial.println("Error al encontrar BH1750.
        Verifica conexiones o direcci n I2C.");
    display.clearDisplay(); display.setCursor(0,0);
        display.println("Error BH1750!"); display.
        display();
    // Considerar un while(true) o intentar de nuevo
  } else {
    Serial.println("BH1750 encontrado.");
  }


  // --- Conexi n WiFi ---
  Serial.print("Intentando conectar a la red WiFi: "
      );
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  int intentos = 0;
  while (WiFi.status() != WL_CONNECTED && intentos <
       60) {
    delay(500);
    Serial.print(".");
    display.print("."); display.display(); //
        Mostrar puntos en la OLED
    intentos++;
  }
  Serial.println("");

  display.clearDisplay();
  display.setCursor(0, 0);


  if (WiFi.status() == WL_CONNECTED) {
    Serial.println(" WiFi  conectado exitosamente!")
        ;
    Serial.print("Direcci n IP: ");
    Serial.println(WiFi.localIP());

    display.println("WiFi Conectado!");
    display.print("IP: ");
    display.println(WiFi.localIP());
    display.drawBitmap(SCREEN_WIDTH -
        WIFI_LOGO_WIDTH - 2, 0, wifi_logo_bits,
        WIFI_LOGO_WIDTH, WIFI_LOGO_HEIGHT,
        SH110X_WHITE);
    display.display();
    delay(2000);

    // --- Configurar NTP una vez conectado a WiFi
        ---
    configTime(gmtOffset_sec, daylightOffset_sec,
        ntpServer);
    Serial.println("Hora NTP configurada.");
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("Sincronizando hora...");
    display.display();
    delay(2000);
  } else {
    Serial.println(" Error  al conectar a la red
        WiFi!");
    Serial.println("Verifica el SSID y la
        contrase a.");
    display.println("Error WiFi!");
    display.println("Verifica cred.");
    display.display();
    // Si no hay WiFi, no podemos enviar datos ni
        obtener hora.
    // Podr amos ir a Deep Sleep o intentar
        indefinidamente. Aqu  se ir  a Deep Sleep
        tras un tiempo.
  }
}

void loop() {
  // --- Lectura de sensores y env o de datos ---
  float t = bme.readTemperature();
  float h = bme.readHumidity();
  float p = bme.readPressure() / 100.0F; // Presi n
      en hPa (hectopascales)
  float lux = bh1750.readLightLevel(); // Lectura
      del sensor de luz

  // Si usas BME680, tambi n podr as leer la
      calidad del aire:
  // float gas_resistance = bme.readGas();

  // --- Verificar si la lectura es v lida ---
  if (isnan(t) || isnan(h) || isnan(p) || isnan(lux)
      ) {
    Serial.println(" Error  al leer de un sensor!");
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("Error Sensor!");
    display.println("Reintentando...");
    display.display();
    delay(2000); // Peque a pausa antes de intentar
         de nuevo o ir a dormir
    // Si hay error en la lectura, se puede optar
        por no enviar datos
    // y entrar directamente en Deep Sleep para
        reintentar en el pr ximo ciclo
    esp_deep_sleep_start();
  }
```

```
  // --- Mostrar datos en el Monitor Serial ---
  Serial.print("Temperatura: "); Serial.print(t);
      Serial.println(" *C"); % Corregido: quitar (
      char)247
  Serial.print("Humedad: "); Serial.print(h); Serial
      .println(" %");
  Serial.print("Presion: "); Serial.print(p); Serial
      .println(" hPa");
  Serial.print("Luminosidad: "); Serial.print(lux);
      Serial.println(" lux");
  // if (bme.chipID == BME680_CHIP_ID) Serial.print
      ("Calidad Aire (VOC): "); Serial.println(
      gas_resistance);

  // --- Enviar datos a Termux ---
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    String serverPath = "http://" + String(serverIp)
        + ":" + String(serverPort) + "/dht_data";
        // Mantener /dht_data por compatibilidad
    Serial.print("Enviando datos a: "); Serial.
        println(serverPath);

    http.begin(serverPath);
    http.addHeader("Content-Type", "application/json
        ");

    String jsonPayload = "{";
    jsonPayload += "\"temperatura\":" + String(t) +
        ",";
    jsonPayload += "\"humedad\":" + String(h) + ",";
    jsonPayload += "\"presion\":" + String(p) + ",";
    jsonPayload += "\"lux\":" + String(lux);
    // if (bme.chipID == BME680_CHIP_ID) jsonPayload
        += ",\"calidad_aire\":" + String(
        gas_resistance);
    jsonPayload += "}";

    Serial.print("Payload JSON: "); Serial.println(
        jsonPayload);

    int httpResponseCode = http.POST(jsonPayload);

    if (httpResponseCode > 0) {
      Serial.printf("C digo de Respuesta HTTP: %d\n
          ", httpResponseCode);
      String responsePayload = http.getString();
      Serial.println("Respuesta del servidor Termux:
          ");
      Serial.println(responsePayload);
    } else {
      Serial.printf("Error en la petici n HTTP: %s\
          n", http.errorToString(httpResponseCode).
          c_str());
    }
    http.end();
  } else {
    Serial.println("WiFi desconectado. No se
        pudieron enviar los datos.");
    display.clearDisplay();
    display.setCursor(0, 0);
    display.println("WiFi OFF!");
    display.println("Sin envio.");
    display.display();
  }

  // --- Mostrar datos y hora en la pantalla OLED
      ---
  display.clearDisplay();
  display.setCursor(0, 0);

  // Mostrar el logo de WiFi si est  conectado
  if (WiFi.status() == WL_CONNECTED) {
    display.drawBitmap(SCREEN_WIDTH -
        WIFI_LOGO_WIDTH - 2, 0, wifi_logo_bits,
        WIFI_LOGO_WIDTH, WIFI_LOGO_HEIGHT,
        SH110X_WHITE);
  }

  // Mostrar hora
  struct tm timeinfo;
  if (getLocalTime(&timeinfo)) {
    display.setTextSize(1);
    display.setCursor(0, 0);
    char timeBuffer[9];
    strftime(timeBuffer, sizeof(timeBuffer), "%H:%M
        :%S", &timeinfo);
    display.print(timeBuffer);
  } else {
    display.setTextSize(1);
    display.setCursor(0, 0);
    display.println("Sinc. hora...");
  }

  // Mostrar Temperatura (m s grande)
  display.setTextSize(2);
  display.setCursor(0, 20);
  display.print(t, 1);
  display.print("*"); display.println("C"); %
      Corregido: quitar (char)247

  // Mostrar Humedad
  display.setCursor(0, 45);
  display.print(h, 1);
  display.println(" %RH");

  // Mostrar Presi n y Lux (m s peque os o en
      otra l nea si hay espacio)
  display.setTextSize(1);
  display.setCursor(70, 0); // Posici n para
      presi n
  display.print(p, 0); display.println("hPa");
  display.setCursor(70, 10); // Posici n para lux
  display.print(lux, 0); display.println("lux");

  display.display();

  // --- Configurar Deep Sleep ---
  Serial.printf("Entrando en Deep Sleep por %llu
      segundos...\n", TIME_TO_SLEEP / uS_TO_S_FACTOR
      );
  display.clearDisplay();
  display.setCursor(0,0);
  display.println("Zzz...");
  display.println("Ahorro energia");
  display.display();
  delay(1000); // Peque a pausa para asegurar que
      el mensaje se muestre

  esp_deep_sleep_enable_timer_wakeup(TIME_TO_SLEEP);
       // Configura el temporizador para despertar
  esp_deep_sleep_start(); // Inicia el Deep Sleep
}
```

Listing 1. Código del Firmware del ESP32 con BME280/BME680, BH1750, OLED y Deep Sleep.

B. Código del Servidor Flask (Python en Termux)

El servidor Flask se encargará de recibir los datos en formato JSON enviados por el ESP32, procesarlos y almacenarlos o mostrarlos. Se ha modificado para esperar los nuevos campos de datos (presión y lux, y calidad del aire si aplica).

```
# dht_server.py
from flask import Flask, request, jsonify
import datetime
import os
```

```
5
6  app = Flask(__name__)
7
8  # Aseg rate de que el directorio de logs exista si
       lo quieres en un lugar espec fico
9  log_dir = "/data/data/com.termux/files/home/logs" #
       Ruta t pica de Termux
10 if not os.path.exists(log_dir):
11     os.makedirs(log_dir)
12
13 log_file_path = os.path.join(log_dir, "ambiental_log
       .txt")
14
15 @app.route('/dht_data', methods=['POST'])
16 def receive_dht_data():
17     if request.is_json:
18         data = request.get_json()
19         temperatura = data.get('temperatura')
20         humedad = data.get('humedad')
21         presion = data.get('presion')
22         lux = data.get('lux')
23         # Si usas BME680, espera este campo tambi n
24         # calidad_aire = data.get('calidad_aire')
25
26         timestamp = datetime.datetime.now().strftime
               ("%Y-%m-%d %H:%M:%S")
27
28         log_entry = f"[{timestamp}] Temp: {
               temperatura}C, Hum: {humedad}%, Pres: {
               presion}hPa, Lux: {lux}lux"
29         # if calidad_aire is not None:
30         #     log_entry += f", AQI: {calidad_aire}"
31         log_entry += "\n"
32
33         print(log_entry.strip()) # Imprimir en
               consola de Termux
34
35         with open(log_file_path, "a") as f:
36             f.write(log_entry)
37
38         return jsonify({"status": "success", "
               message": "Datos recibidos y registrados
               "}), 200
39     else:
40         print(f"[{datetime.datetime.now().strftime
               ('%Y-%m-%d %H:%M:%S')}] Error: Petici n
                no es JSON")
41         return jsonify({"status": "error", "message"
               : "Content-Type must be application/json
               "}), 400
42
43 @app.route('/')
44 def index():
45     return "Servidor de Monitoreo Ambiental. Env a
           POST a /dht_data"
46
47 if __name__ == '__main__':
48     # Para que sea accesible desde la red local
49     app.run(host='0.0.0.0', port=5000, debug=True)
```

Listing 2. Código del Servidor Flask en Termux.

V. Mejoras Implementadas y Justificación

A. Integración de Sensores Avanzados (BME280/BME680 y BH1750)

La sustitución del sensor DHT11 por el **BME280 (o BME680)** y la adición del **BH1750** eleva significativamente la capacidad y precisión del sistema.

- **BME280/BME680:** Estos sensores de Bosch Sensortec son reconocidos por su alta precisión y estabilidad en la medición de temperatura ($\pm 0.5°C$), humedad relativa ($\pm 3\%$) y presión barométrica (± 1.0 hPa). El BME680 además integra un sensor de gas para medir Compuestos Orgánicos Volátiles (VOCs), lo que permite inferir la calidad del aire. Su interfaz I2C facilita la integración con múltiples dispositivos en un mismo bus.
- **BH1750:** Este sensor de luz ambiental, también con interfaz I2C, proporciona mediciones de iluminancia en lux de alta resolución (hasta 1 lux). Es crucial para aplicaciones donde el nivel de luz es un factor relevante, como en la agricultura de precisión o la gestión energética de edificios.

Estas mejoras hacen el sistema apto para entornos donde la fiabilidad y exactitud de los datos son críticas, como en la monitorización de ambientes industriales o agrícolas.

B. Optimización de Energía: Modo Deep Sleep (Cryo Sueño)

La implementación del modo **Deep Sleep** en el ESP32 es una estrategia fundamental para reducir drásticamente el consumo de energía y, por ende, prolongar la vida útil del dispositivo, especialmente en aplicaciones alimentadas por batería.

- **Funcionamiento:** Durante el "Deep Sleep", el ESP32 apaga la mayoría de sus componentes, incluyendo la CPU y la radio WiFi, manteniendo solo la circuitería del RTC (Real-Time Clock) activa. Después de un período predefinido (e.g., 5 minutos), el RTC activa el microcontrolador, que se reinicia y ejecuta el código desde 'setup()', realiza sus tareas (lectura, envío, visualización) y vuelve a entrar en Deep Sleep.
- **Beneficios:**
  - **Reducción del Consumo:** El consumo de corriente se reduce de decenas de mA (miliamperios) en modo activo a solo unos pocos μA (microamperios) en Deep Sleep.
  - **Mayor Vida Útil:** Al reducir el tiempo de operación continua y el calentamiento, se disminuye el desgaste de los componentes.
  - **Autonomía Prolongada:** Permite que el dispositivo funcione por meses o incluso años con una sola batería, ideal para implementaciones remotas sin acceso constante a una fuente de energía.

Esta función es vital para proyectos de IoT a largo plazo y de bajo mantenimiento.

C. Sincronización Horaria NTP

La sincronización horaria mediante el protocolo NTP (Network Time Protocol) garantiza que todos los registros de datos tengan un timestamp preciso y uniforme. Esto es crucial para el análisis de datos, la correlación de eventos y la generación de gráficos históricos.

D. Reconexión WiFi Automática

La capacidad de reconexión automática asegura la robustez del sistema frente a interrupciones temporales de la red WiFi. El ESP32 monitorea continuamente el estado de la conexión

y, si se pierde, intenta reconectarse de forma autónoma, minimizando la pérdida de datos y la necesidad de intervención manual.

VI. Consideraciones para la Implementación Industrial

Para llevar este sistema a un punto de aplicación industrial, se deben considerar los siguientes aspectos:

- **Robustez del Hardware:** Utilizar módulos ESP32 de grado industrial o placas diseñadas para entornos exigentes (ej., con protección ESD, filtros de ruido, mayor rango de temperatura de operación).
- **Sensores Calibrados y de Mayor Precisión/Durabilidad:** Aunque los BME y BH1750 son excelentes, en ciertos casos industriales (ej., cámaras limpias, procesos químicos) podrían requerirse sensores con certificaciones específicas o mayor resistencia a agentes corrosivos.
- **Alimentación y Batería:** Diseñar una etapa de alimentación robusta, con protección contra sobretensiones y gestión eficiente de baterías recargables (ej., LiPo con circuito de carga y protección).
- **Comunicación y Redundancia:** En lugar de solo WiFi a un Termux local, considerar protocolos más robustos para IoT industrial como MQTT (con un broker centralizado y seguro), LoRaWAN para largas distancias, o incluso 4G/5G para ubicaciones remotas. Implementar mecanismos de reintento y colas de mensajes.
- **Plataforma de Datos (Cloud/Local):** Un servidor Flask en Termux es ideal para un prototipo. Para uso industrial, se necesitaría una base de datos más robusta (ej., InfluxDB para series temporales, PostgreSQL) y una plataforma de visualización (ej., Grafana, un dashboard web personalizado) alojada en un servidor dedicado o en la nube (AWS IoT, Azure IoT, Google Cloud IoT).
- **Seguridad:** Implementar cifrado (HTTPS/TLS) para la comunicación, autenticación de dispositivos (certificados), y protección contra accesos no autorizados. La ciberseguridad es crítica en entornos industriales.
- **Caja y Protección (IP Rating):** La caja de proyecto de 5.7x8.4x3.6 cm es un buen inicio. Para un entorno industrial, se necesitaría una caja con un grado de protección IP (Ingress Protection) adecuado contra polvo y agua (ej., IP65, IP67), y posiblemente resistencia a vibraciones o químicos.
- **Actualizaciones de Firmware (OTA):** Implementar un mecanismo de "Over-The-Air" (OTA) para actualizar el firmware del ESP32 de forma remota, facilitando el mantenimiento y la implementación de nuevas características sin acceso físico.

VII. Conclusiones y Trabajo Futuro

El sistema de monitoreo ambiental desarrollado demuestra la viabilidad de crear soluciones IoT completas y eficientes usando el ESP32. La integración de sensores avanzados, la visualización local en OLED y la comunicación con un backend móvil (Termux) establecen una base sólida. La incorporación del Deep Sleep es un paso crucial hacia dispositivos de larga autonomía.

Como trabajo futuro, se propone:

- Explorar opciones de comunicación MQTT para una integración con plataformas IoT industriales.
- Desarrollar una interfaz de usuario más interactiva en el servidor Flask (o una nueva plataforma web) para la visualización histórica y el análisis de los datos.
- Investigar el uso de baterías y la gestión de la energía a nivel de hardware para optimizar aún más el consumo.
- Considerar la implementación de algoritmos de Machine Learning en el dispositivo (Edge AI) para análisis de datos in situ o detección de anomalías.
- Realizar pruebas exhaustivas de ciberseguridad en todos los componentes del sistema, desde el firmware del ESP32 hasta el servidor Flask en Termux, aplicando técnicas de escaneo y análisis de vulnerabilidades para identificar y mitigar posibles riesgos.

References