Análise Detalhada do Código do EcoSniff
Autor: Ryan Morais

Este código é escrito para uma placa ESP32 (um microcontrolador) que utiliza diversos sensores para medir parâmetros ambientais, como temperatura, umidade e concentrações de gases (CO2, Metano, Amônia e Sulfeto de Hidrogênio). Os dados coletados são enviados para o aplicativo Blynk através de uma conexão Wi-Fi, permitindo o monitoramento remoto.

Vamos explicar passo a passo cada trecho do código.

---

1. Definições e Inclusão de Bibliotecas

```
// Definições do Blynk - substitua com suas informações do Blynk

#define BLYNK_TEMPLATE_ID "Seu_BLYNK_TEMPLATE_ID"

#define BLYNK_TEMPLATE_NAME "EcoSniff"

#define BLYNK_AUTH_TOKEN "Seu_BLYNK_AUTH_TOKEN"


// Definição para permitir o uso do Serial para depuração no Blynk

#define BLYNK_PRINT Serial


// Inclusão das bibliotecas necessárias

#include <WiFi.h>                // Biblioteca para conexão Wi-Fi no ESP32

#include <WiFiClient.h>          // Biblioteca para cliente Wi-Fi

#include <BlynkSimpleEsp32.h>    // Biblioteca Blynk para ESP32

#include "DHT.h"                 // Biblioteca para o sensor DHT11

#include "MQUnifiedsensor.h"     // Biblioteca unificada para sensores MQ
```

Explicação:

- **Definições do Blynk:**

    - `#define BLYNK_TEMPLATE_ID`, `BLYNK_TEMPLATE_NAME` e `BLYNK_AUTH_TOKEN`: São constantes usadas para configurar a conexão com o aplicativo Blynk. Você deve substituí-las pelas suas informações específicas fornecidas pelo Blynk.
- **Depuração Serial com Blynk:**
    - `#define BLYNK_PRINT Serial`: Permite que as mensagens de depuração do Blynk sejam enviadas para o monitor serial, o que é útil para acompanhar o funcionamento do código durante o desenvolvimento.
- **Inclusão de Bibliotecas:**
    - `WiFi.h` e `WiFiClient.h`: Bibliotecas padrão do ESP32 para conectar-se a redes Wi-Fi e criar clientes Wi-Fi.
    - `BlynkSimpleEsp32.h`: Biblioteca que facilita a integração do ESP32 com o Blynk.
    - `DHT.h`: Biblioteca para comunicação com sensores de temperatura e umidade DHT11.
    - `MQUnifiedsensor.h`: Biblioteca que unifica a comunicação com sensores de gases da série MQ, facilitando a leitura e calibração.

---

2. Configuração do Sensor DHT11

```
// Configuração do sensor DHT11

#define DHTPIN 32              // Pino digital conectado ao sensor DHT11

#define DHTTYPE DHT11          // Definição do tipo de sensor DHT utilizado


// Inicialização do objeto DHT

DHT dht(DHTPIN, DHTTYPE);
```

Explicação:

- **Definição do Pino e Tipo do Sensor:**
    - `DHTPIN`: Define o pino digital do ESP32 que está conectado ao sensor DHT11 (neste caso, o pino 32).
    - `DHTTYPE`: Especifica o tipo de sensor DHT utilizado, que pode ser DHT11 ou DHT22. Aqui, estamos usando o DHT11.
- **Inicialização do Objeto DHT:**

- `DHT dht(DHTPIN, DHTTYPE);`: Cria um objeto `dht` da classe `DHT`, que será usado para interagir com o sensor DHT11.

---

3. Configurações Gerais para os Sensores MQ

```
// Configurações gerais para os sensores MQ

#define Board        ("ESP-32")          // Nome da placa

#define Voltage_Resolution  3.3          // Resolução de tensão do ADC (3.3V para ESP32)

#define ADC_Bit_Resolution  12           // Resolução em bits do ADC (12 bits para ESP32)

#define RatioMQ135CleanAir  3.6          // Razão RS/R0 em ar limpo para o sensor MQ-135

#define RatioMQ4CleanAir    4.4          // Razão RS/R0 em ar limpo para o sensor MQ-4

#define RatioMQ137CleanAir  27.5         // Razão RS/R0 em ar limpo para o sensor MQ-137

#define RatioMQ136CleanAir  6.5          // Razão RS/R0 em ar limpo para o sensor MQ-136
```

Explicação:

- **Configurações da Placa e ADC:**
  - `Board`: Nome da placa usada (ESP32).
  - `Voltage_Resolution`: A tensão máxima que o conversor analógico-digital (ADC) da placa pode ler. O ESP32 opera em 3.3V.
  - `ADC_Bit_Resolution`: Resolução em bits do ADC do ESP32. Com 12 bits, o ADC pode representar valores de 0 a 4095.
- **Razões RS/R0 em Ar Limpo:**
  - `RatioMQ135CleanAir`, `RatioMQ4CleanAir`, etc.: São constantes que representam a razão entre a resistência do sensor (RS) e a resistência de calibração (R0) quando o sensor está em ar limpo. Essas razões são específicas para cada tipo de sensor e são usadas durante a calibração.

---

4. Definição dos Pinos Analógicos para os Sensores MQ

```
// Definição dos pinos analógicos onde os sensores MQ estão conectados
#define MQ135PIN 36                 // Pino analógico para o sensor MQ-135
#define MQ4PIN   39                 // Pino analógico para o sensor MQ-4
#define MQ137PIN 34                 // Pino analógico para o sensor MQ-137
#define MQ136PIN 33                 // Pino analógico para o sensor MQ-136
```

Explicação:

- **Cada sensor MQ está conectado a um pino analógico específico do ESP32:**
  - `MQ135PIN`**:** Pino 36 para o sensor MQ-135.
  - `MQ4PIN`**:** Pino 39 para o sensor MQ-4.
  - `MQ137PIN`**:** Pino 34 para o sensor MQ-137.
  - `MQ136PIN`**:** Pino 33 para o sensor MQ-136.

---

5. Criação dos Objetos para os Sensores MQ

```
// Criação dos objetos para cada sensor MQ
MQUnifiedsensor MQ135(Board, Voltage_Resolution, ADC_Bit_Resolution, MQ135PIN, "MQ-135");
MQUnifiedsensor MQ4(Board, Voltage_Resolution, ADC_Bit_Resolution, MQ4PIN, "MQ-4");
MQUnifiedsensor MQ137(Board, Voltage_Resolution, ADC_Bit_Resolution, MQ137PIN, "MQ-137");
MQUnifiedsensor MQ136(Board, Voltage_Resolution, ADC_Bit_Resolution, MQ136PIN, "MQ-136");
```

Explicação:

- **Aqui, criamos objetos para cada sensor MQ usando a classe** `MQUnifiedsensor`**:**
  - **Cada objeto é inicializado com os parâmetros:**
    - `Board`**:** Nome da placa (ESP32).
    - `Voltage_Resolution`**:** Resolução de tensão do ADC (3.3V).
    - `ADC_Bit_Resolution`**:** Resolução em bits do ADC (12 bits).
    - `Pino Analógico`**:** O pino ao qual o sensor está conectado.

- `"Nome do Sensor"`: Uma string com o nome do sensor.

---

6. Variáveis para Disponibilidade dos Sensores

```
// Variáveis booleanas para indicar a disponibilidade dos sensores após calibração
bool sensorMQ135Disponivel = true;
bool sensorMQ4Disponivel = true;
bool sensorMQ137Disponivel = true;
bool sensorMQ136Disponivel = true;
```

Explicação:

- **São variáveis do tipo booleano (verdadeiro ou falso) que indicam se cada sensor está disponível para uso após o processo de calibração.**
  - **Inicialmente, todos estão definidos como `true` (disponível).**

---

7. Credenciais da Rede Wi-Fi

```
// Credenciais da rede Wi-Fi - substitua pelos dados da sua rede
char ssid[] = "Seu_SSID";              // Nome da rede Wi-Fi (SSID)
char pass[] = "Sua_Senha";             // Senha da rede Wi-Fi
```

Explicação:

- `ssid[]`: Uma string que contém o nome da sua rede Wi-Fi.
- `pass[]`: Uma string que contém a senha da sua rede Wi-Fi.
- **Importante:** Substitua `"Seu_SSID"` e `"Sua_Senha"` pelas informações da sua rede para que o ESP32 possa se conectar à internet.

---

8. Função setup()

```
void setup() {
  // Inicialização da comunicação serial para depuração
```

```
  Serial.begin(115200);


  // Inicialização do sensor DHT11
  dht.begin();


  // Configuração dos sensores MQ


  // Configuração do MQ135 para medir CO2
  MQ135.setRegressionMethod(1);          // Define o método de regressão (1 = potência)
  MQ135.setA(110.47); MQ135.setB(-2.862); // Define os coeficientes da equação para CO2
  MQ135.init();                          // Inicializa o sensor MQ135


  // Configuração do MQ4 para medir Metano (CH4)
  MQ4.setRegressionMethod(1);            // Define o método de regressão
  MQ4.setA(1000.0); MQ4.setB(-2.186);    // Define os coeficientes para CH4
  MQ4.init();                            // Inicializa o sensor MQ4


  // Configuração do MQ137 para medir Amônia (NH3)
  MQ137.setRegressionMethod(1);          // Define o método de regressão
  MQ137.setA(102.2); MQ137.setB(-2.473);  // Define os coeficientes para NH3
  MQ137.init();                          // Inicializa o sensor MQ137


  // Configuração do MQ136 para medir Sulfeto de Hidrogênio (H2S)
  MQ136.setRegressionMethod(1);          // Define o método de regressão
  MQ136.setA(44.947); MQ136.setB(-3.445); // Define os coeficientes para H2S
```

```
  MQ136.init();                         // Inicializa o sensor MQ136

  // Início da calibração dos sensores MQ
  Serial.println("Calibrando sensores... Por favor, aguarde.");

  // Calibração do MQ135
  // ... (código para calibrar o MQ135)

  // Calibração do MQ4
  // ... (código para calibrar o MQ4)

  // Calibração do MQ137
  // ... (código para calibrar o MQ137)

  // Calibração do MQ136
  // ... (código para calibrar o MQ136)

  Serial.println("Calibração concluída!");

  // Conexão à rede Wi-Fi
  WiFi.begin(ssid, pass); // Inicia a conexão Wi-Fi usando o SSID e a senha fornecidos
  // Aguarda até que a conexão seja estabelecida
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print("."); // Imprime um ponto a cada meio segundo enquanto não está
conectado
  }
```

```
  Serial.println("\nConectado ao Wi-Fi");


  // Configuração e conexão ao Blynk
  Blynk.config(BLYNK_AUTH_TOKEN); // Configura o Blynk com o token de autenticação
  Blynk.connect();               // Conecta ao servidor Blynk
}
```

Explicação:

A função `setup()` é executada uma vez quando o ESP32 é ligado ou reiniciado. Nela, realizamos as configurações iniciais necessárias para o funcionamento do dispositivo.

- **Inicialização da Comunicação Serial:**
    - `Serial.begin(115200);`**:** Inicia a comunicação serial com uma taxa de 115200 bauds, permitindo que o ESP32 envie mensagens para o computador (útil para depuração).
- **Inicialização do Sensor DHT11:**
    - `dht.begin();`**:** Prepara o sensor DHT11 para começar a coletar dados de temperatura e umidade.
- **Configuração dos Sensores MQ:**
    - **Para cada sensor MQ, definimos:**
        - **Método de Regressão:** `setRegressionMethod(1);` indica que usaremos uma equação de potência para converter as leituras em concentrações de gás.
        - **Coeficientes da Equação (A e B):** Esses valores são específicos para cada gás e são usados na equação de calibração do sensor.
        - **Inicialização do Sensor:** `init();` Prepara o sensor para operação.
- **Calibração dos Sensores MQ:**
    - **A calibração é essencial para garantir que os sensores forneçam leituras precisas.**
    - **Para cada sensor:**
        - **Tentamos calibrar até 3 vezes. Se a calibração falhar após 3 tentativas, o sensor é marcado como indisponível.**
        - **Durante a calibração:**
            - **Realizamos 10 leituras do sensor.**

        - **Calculamos o valor médio de R0 (resistência de calibração).**
        - **Verificamos se o valor de R0 é válido (não infinito e diferente de zero).**
- **Conexão à Rede Wi-Fi:**
    - `WiFi.begin(ssid, pass);`**:** Inicia a tentativa de conexão à rede Wi-Fi com as credenciais fornecidas.
    - **O laço** `while (WiFi.status() != WL_CONNECTED)` faz com que o código espere até que a conexão seja estabelecida.
    - `Serial.print(".");`**:** Enquanto espera, imprime pontos no monitor serial para indicar que está tentando conectar.
- **Configuração e Conexão ao Blynk:**
    - `Blynk.config(BLYNK_AUTH_TOKEN);`**:** Configura o Blynk com o token de autenticação do seu projeto.
    - `Blynk.connect();`**:** Estabelece a conexão com o servidor do Blynk.

---

9. Função loop()

```
void loop() {
  Blynk.run(); // Mantém a comunicação com o Blynk ativa

  // Leitura dos dados do sensor DHT11 (temperatura e umidade)
  float temperatura = dht.readTemperature(); // Lê a temperatura em graus Celsius
  float umidade = dht.readHumidity();        // Lê a umidade relativa em porcentagem

  // Verifica se as leituras são válidas
  if (isnan(temperatura) || isnan(umidade)) {
    Serial.println("Falha ao ler do sensor DHT11!");
  } else {
    // Imprime os valores lidos no monitor serial
    Serial.print("Temperatura: ");
    Serial.print(temperatura);
```

```
  Serial.print(" °C | Umidade: ");
  Serial.print(umidade);
  Serial.println(" %");

  // Envia os dados para o aplicativo Blynk
  Blynk.virtualWrite(V1, temperatura); // Envia a temperatura para o Virtual Pin V1
  Blynk.virtualWrite(V2, umidade);     // Envia a umidade para o Virtual Pin V2
}

// Leitura e envio dos dados do sensor MQ135 (CO2)
if(sensorMQ135Disponivel){
  MQ135.update();                    // Atualiza a leitura do sensor
  float ppmMQ135 = MQ135.readSensor();  // Lê a concentração de CO2 em ppm
  Serial.print("CO2 (MQ135): ");
  Serial.print(ppmMQ135);
  Serial.println(" ppm");
  Blynk.virtualWrite(V0, ppmMQ135);     // Envia o valor para o Virtual Pin V0
} else {
  Serial.println("MQ135 indisponível.");
}

// Leitura e envio dos dados do sensor MQ4 (Metano - CH4)
if(sensorMQ4Disponivel){
  MQ4.update();
  float ppmMQ4 = MQ4.readSensor();
  Serial.print("CH4 (MQ4): ");
```

```
  Serial.print(ppmMQ4);
  Serial.println(" ppm");
  Blynk.virtualWrite(V3, ppmMQ4);
} else {
  Serial.println("MQ4 indisponível.");
}

// Leitura e envio dos dados do sensor MQ137 (Amônia - NH3)
if(sensorMQ137Disponivel){
  MQ137.update();
  float ppmMQ137 = MQ137.readSensor();
  Serial.print("NH3 (MQ137): ");
  Serial.print(ppmMQ137);
  Serial.println(" ppm");
  Blynk.virtualWrite(V4, ppmMQ137);
} else {
  Serial.println("MQ137 indisponível.");
}

// Leitura e envio dos dados do sensor MQ136 (Sulfeto de Hidrogênio - H2S)
if(sensorMQ136Disponivel){
  MQ136.update();
  float ppmMQ136 = MQ136.readSensor();
  Serial.print("H2S (MQ136): ");
  Serial.print(ppmMQ136);
  Serial.println(" ppm");
```

```
    Blynk.virtualWrite(V5, ppmMQ136);
  } else {
    Serial.println("MQ136 indisponível.");
  }

  // Aguarda 2 segundos antes de realizar a próxima leitura
  delay(2000);
}
```

Explicação:

A função `loop()` é executada repetidamente, mantendo o programa em execução contínua. Nela, realizamos leituras dos sensores e enviamos os dados para o Blynk.

- **Mantém a Comunicação com o Blynk:**
    - `Blynk.run();`**:** Necessário para manter a conexão com o Blynk ativa e processar os dados enviados/recebidos.
- **Leitura do Sensor DHT11:**
    - `float temperatura = dht.readTemperature();`**:** Lê a temperatura atual.
    - `float umidade = dht.readHumidity();`**:** Lê a umidade relativa atual.
    - **Verificamos se as leituras são números válidos (não são `NaN` - Not a Number).**
    - **Se as leituras forem válidas:**
        - **Imprimimos os valores no monitor serial.**
        - **Enviamos os dados para o Blynk usando `Blynk.virtualWrite(V1, temperatura);` e `Blynk.virtualWrite(V2, umidade);`, onde `V1` e `V2` são os pinos virtuais no aplicativo Blynk.**
- **Leitura dos Sensores MQ:**
    - **Para cada sensor MQ disponível:**
        - **Atualizamos a leitura do sensor com `MQX.update();`.**
        - **Lemos a concentração do gás em ppm com `float ppmMQX = MQX.readSensor();`.**
        - **Imprimimos o valor no monitor serial.**

        - **Enviamos o valor para o Blynk usando `Blynk.virtualWrite(VX, ppmMQX);`, onde `VX` é o pino virtual correspondente no Blynk.**
- **Pausa entre as Leituras:**
    - **`delay(2000);`: Aguarda 2000 milissegundos (2 segundos) antes de iniciar o próximo ciclo do `loop()`. Isso evita que o código execute muito rapidamente e sobrecarregue a CPU ou a rede.**

---

10. Observações Finais

- **Pinos Virtuais no Blynk:**
    - No aplicativo Blynk, você pode configurar widgets que correspondem aos pinos virtuais definidos no código (V0, V1, V2, etc.). Dessa forma, os dados enviados pelo ESP32 serão exibidos no aplicativo.
- **Calibração dos Sensores MQ:**
    - A calibração é crucial para garantir que as leituras dos sensores sejam precisas. Ela ajusta os sensores para que eles saibam qual é a leitura em condições conhecidas (ar limpo, neste caso).
- **Depuração:**
    - As mensagens impressas no monitor serial são úteis para verificar se o código está funcionando corretamente e para identificar possíveis problemas.
- **Segurança:**
    - Certifique-se de não compartilhar suas credenciais de Wi-Fi ou tokens do Blynk publicamente. No código fornecido, elas estão como placeholders (**`"Seu_SSID"`, `"Sua_Senha"`, `"Seu_BLYNK_AUTH_TOKEN"`**).
- **Bibliotecas e Dependências:**
    - As bibliotecas usadas devem estar instaladas no ambiente de desenvolvimento (por exemplo, Arduino IDE) para que o código compile corretamente.
    - **As bibliotecas `DHT.h` e `MQUnifiedsensor.h`** podem ser instaladas via Gerenciador de Bibliotecas do Arduino IDE.

Resumo

Este código permite que um ESP32 leia dados de vários sensores (temperatura, umidade e gases específicos) e envie essas informações para um aplicativo Blynk via Wi-Fi. Ele inclui procedimentos para calibrar os sensores de gases, lidar com possíveis erros de leitura e manter uma conexão estável com a internet e o servidor do Blynk. Com este sistema, é

possível monitorar em tempo real as condições ambientais de um local, o que pode ser útil em aplicações como monitoramento de qualidade do ar, detecção de gases nocivos, entre outros.
