

Desenvolvimento de Sistemas

**Projeto
Sistema Autônomo de Monitoramento de Nível de
Reservatório (Alerta Visual).**

Vitória Nunes Pereira

Sumário

1.	Objetivo	3
2.	Visão Geral	3
3.	Componentes e Hardware	3
4.	Constantes e Limiares	4
5.	Funções Principais	4
6.	Lógica do Loop Principal (loop())	5
7.	Considerações de Otimização	6
8.	Hardware e Montagem:	6
9.	Código Completo	7

1. Objetivo

O objetivo primário deste sistema é desenvolver e implementar um monitor de nível autônomo utilizando a plataforma Arduino e um sensor ultrassônico HC-SR04, com a finalidade de:

- **Medir** continuamente o nível de preenchimento (ocupação) de um reservatório, calibrado para uma distância máxima de 45.0 cm.
- **Fornecer feedback visual** em tempo real sobre o estado do nível através de um sistema de alerta tricolor (LEDs Verde, Amarelo e Vermelho), baseado em limiares de ocupação predefinidos (Bom: >70%, Atenção: 30% a 70%, Crítico: < 30%).
- **Garantir a operação não-bloqueante** do alerta de estado Crítico (LED Vermelho piscante) para manter a precisão e a frequência das medições.

2. Visão Geral

O sistema mede a distância da superfície do material até o sensor, calcula a porcentagem de ocupação e aciona um dos três LEDs para indicar o status do nível (Bom, Atenção ou Crítico) com base em limiares predefinidos.

3. Componentes e Hardware

COMPONENTE	FUNÇÃO	PINO ARDUINO	VARIÁVEL	TIPO
Sensor HC-SR04	Pino Trigger (Saída)	D9	TRIG_PIN	Saída
Sensor HC-SR04	Pino Echo (Entrada)	D10	ECHO_PIN	Entrada
LED Verde	Nível BOM (Cheio)	D5	LED_VERDE	Saída
LED Amarelo	Nível ATENÇÃO (Médio)	D4	LED_AMARELO	Saída
LED Vermelho	Nível CRÍTICO (Vazio)	D3	LED_VERMELHO	Saída

4. Constantes e Limiares

Constante	Valor	Unidade	Descrição
DISTANCIA_MAXIMA_CM	45.0	cm	Distância calibrada quando o reservatório está vazio (ponto de referência \$0\%\$ de ocupação).
LIMIAR_CRITICO	30	%	Limiar inferior. Abaixo deste valor, o estado é CRÍTICO (LED Vermelho).
LIMIAR_ATENCAO	70	%	Limiar superior para ATENÇÃO. Entre 30\% e 70\%, o estado é ATENÇÃO (LED Amarelo).
Intervalo Pisca	200	ms	Frequência de chaveamento do LED Vermelho no estado CRÍTICO.

5. Funções Principais

1.1 getDistance()

Esta função é responsável por interagir com o sensor ultrassônico e calcular a distância.

- Ação: Envia um pulso de 10µs no pino TRIG_PIN e mede a duração do pulso de retorno (duração) no pino ECHO_PIN usando pulseIn().
- Cálculo: Utiliza a fórmula da velocidade do som (aproximadamente 340 m/s ou 29.1µs/cm):

$$\text{Distância (cm)} = \frac{\text{Duração}(\mu s) \times 0.034}{2}$$

- Retorno: Distância lida em centímetros (float).

1.2 CalculatePercentage(float dist_lida, float dist_maxima)

Converte a distância lida em uma porcentagem de ocupação do reservatório.

- Cálculo do Nível Preenchido (nivel_cm): A distância preenchida pelo material é a diferença entre a distância total (dist_maxima) e a distância lida (dist_lida).

$$\text{Nível (cm)} = \text{DISTANCIA_MAXIMA_CM} - \text{distancia_cm}$$

- Cálculo da Porcentagem:

$$\text{Porcentagem(\%)} = \frac{\text{Nível (cm)}}{\text{DISTANCIA_MAXIMA_CM}} \times 100$$

- Tratamento de Erros: Garante que o nível em cm não seja negativo e limita a porcentagem a \$100\%\$.
- Retorno: Porcentagem de ocupação (int) entre \$0\\$ e \$100\\$

1.3 setLeds(int nível)

Aplica a lógica de controle dos LEDs com base na porcentagem de ocupação.

NÍVEL DE OCUPAÇÃO	ESTADO	AÇÃO DO LED
NÍVEL < 30%	CRÍTICO	LED Vermelho (Controlado para piscar no loop())
30% < NÍVEL < 70%	ATENÇÃO	LED Amarelo Fixo (HIGH)
NÍVEL > 70%	BOM/CHEIO	LED Verde Fixo (HIGH)

Nota: Esta função *desliga* os LEDs não utilizados para o estado atual, mas não controla o pisca-pisca do LED Vermelho.

6. Lógica do Loop Principal (loop())

O fluxo do programa é projetado para ser **não bloqueante** (exceto por um delay (500) no estado "Não Crítico", que pode ser otimizado).

1. **Medição:** Chama getDistance() para obter a leitura.
2. **Cálculo:** Chama calculatePercentage() para converter a leitura em % de ocupação.

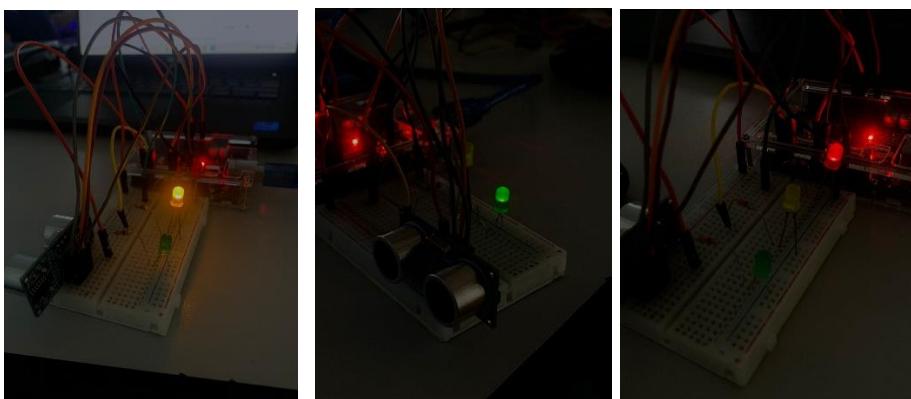
3. **LEDs Fixos:** Chama setLeds() para acender o LED Verde ou Amarelo, ou preparar para o estado Crítico.
4. **Debug:** Imprime o status atual no Monitor Serial.
5. **Controle Não Bloqueante do LED Vermelho:**

- No estado **CRÍTICO** (Nível < LIMIAR_CRÍTICO), a função millis() é usada para alternar o estado do LED Vermelho a cada 200 ms (intervaloPisca), garantindo que o sistema continue respondendo e realizando novas leituras.
- Caso contrário, o LED Vermelho é desligado e um delay(500) é aplicado.

7. Considerações de Otimização

Delay no loop(): O delay(500) na seção else do controle de pisca-pisca introduz um bloqueio de 500 ms a cada ciclo quando o sistema não está no estado Crítico. Para sistemas com requisitos de resposta mais rápidos, este delay deve ser removido ou substituído por uma técnica de temporização não bloqueante baseada em millis(), garantindo que a medição e o feedback ocorram em um intervalo regular.

8. Hardware e Montagem:



9. Código Completo

```
// Pinos do Sensor Ultrassônico HC-SR04
const int TRIG_PIN = 9;    // Pino Trigger (Saída)
const int ECHO_PIN = 10;   // Pino Echo (Entrada)

// Pinos dos LEDs Indicadores
const int LED_VERDE = 5;  // Nível BOM (Cheio)
const int LED_AMARELO = 4; // Nível ATENÇÃO (Médio)
const int LED_VERMELHO = 3; // Nível CRÍTICO (Vazio)

const float DISTANCIA_MAXIMA_CM = 45.0;

// LIMIARES DE ALERTA (em porcentagem de ocupação)
const int LIMIAR_CRITICO = 30; // Abaixo de 30% é CRÍTICO (LED Vermelho)
const int LIMIAR_ATENCAO = 70; // Entre 30% e 70% é ATENÇÃO (LED Amarelo)
// Acima de 70% é CHEIO (LED Verde)
long duracao;      // Armazena a duração do pulso de eco
float distancia_cm; // Distância lida em centímetros
int nivel_porcentagem; // Nível de ocupação (0 a 100%)

// Variável de controle de tempo para o LED piscante (não bloqueia o código)
unsigned long tempoAnterior = 0;
const long intervaloPisca = 200; // Pisca a cada 200 milissegundos

float getDistance() {
    // Limpa o pino TRIG
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);

    // Dispara o pulso de 10us
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    // Lê o tempo de duração do pulso de eco (ida e volta)
    duracao = pulseIn(ECHO_PIN, HIGH);

    // Calcula a distância em cm (velocidade do som é aprox. 340m/s ou 29.1 us/cm)
    // Formula: Distância = (Duração em us * 0.034) / 2
    distancia_cm = duracao * 0.034 / 2;

    return distancia_cm;
}
```

```

int calculatePercentage(float dist_lida, float dist_maxima) {
    // 'Nível' é a distância preenchida pelo material (Distância Máxima
    - Distância Lida)
    float nivel_cm = dist_maxima - dist_lida;

    // Evita leituras negativas (erro do sensor)
    if (nivel_cm < 0) nivel_cm = 0;

    // Calcula a Porcentagem de Ocupação
    int percentagem = (nivel_cm / dist_maxima) * 100;

    // Limita a 100%
    if (percentagem > 100) percentagem = 100;

    return percentagem;
}

void setLeds(int nivel) {
    // Desliga os LEDs (exceto se o Vermelho precisar piscar)
    digitalWrite(LED_VERDE, LOW);
    digitalWrite(LED_AMARELO, LOW);

    if (nivel < LIMIAR_CRITICO) {
        // ESTADO CRÍTICO: O LED VERMELHO será controlado para piscar no
        loop principal.
        digitalWrite(LED_AMARELO, LOW); // Garante que Amarelo está
        desligado
        digitalWrite(LED_VERDE, LOW); // Garante que Verde está desligado

    } else if (nivel < LIMIAR_ATENCAO) {
        // ESTADO ATENÇÃO: Amarelo Fixo
        digitalWrite(LED_VERMELHO, LOW);
        digitalWrite(LED_AMARELO, HIGH);

    } else {
        // ESTADO CHEIO/BOM: Verde Fixo
        digitalWrite(LED_VERMELHO, LOW);
        digitalWrite(LED_VERDE, HIGH);
    }
}

void setup() {
    // Configuração da comunicação serial para debug (Monitor Serial)
    Serial.begin(9600);
    Serial.println("--- Sistema Autônomo de Monitoramento de Nível ---");
    Serial.print("Distância Máxima Calibrada (Vazio): ");
    Serial.print(DISTANCIA_MAXIMA_CM);
    Serial.println(" cm");
}

```

```

// Configuração dos Pins
pinMode(TRIG_PIN, OUTPUT);
pinMode(ECHO_PIN, INPUT);
pinMode(LED_VERDE, OUTPUT);
pinMode(LED_AMARELO, OUTPUT);
pinMode(LED_VERMELHO, OUTPUT);

// Desliga todos os LEDs na inicialização
digitalWrite(LED_VERDE, LOW);
digitalWrite(LED_AMARELO, LOW);
digitalWrite(LED_VERMELHO, LOW);
}

void loop() {
    // 1. REALIZA A MEDIÇÃO
    distancia_cm = getDistance();

    // 2. CALCULA O NÍVEL EM PORCENTAGEM
    nivel_porcentagem = calculatePercentage(distancia_cm,
DISTANCIA_MAXIMA_CM);

    // 3. APLICA A LÓGICA DO LED
    setLeds(nivel_porcentagem);

    // 4. SAÍDA DE DEBUG NO MONITOR SERIAL
    Serial.print("Distância Lida: ");
    Serial.print(distancia_cm);
    Serial.print(" cm | Nível de Ocupação: ");
    Serial.print(nivel_porcentagem);
    Serial.println(" %");

    // 5. CONTROLE DO PISCA-PISCA DO LED VERMELHO (ESTADO CRÍTICO)
    // Usamos millis() para não bloquear outras funções
    if (nivel_porcentagem < LIMIAR_CRITICO) {
        unsigned long tempoAtual = millis();
        if (tempoAtual - tempoAnterior >= intervaloPisca) {
            tempoAnterior = tempoAtual;

            // Inverte o estado do LED Vermelho (liga/desliga)
            digitalWrite(LED_VERMELHO, !digitalRead(LED_VERMELHO));
        }
    } else {
        // Se não estiver em estado crítico, garante que o LED Vermelho
        está desligado
        digitalWrite(LED_VERMELHO, LOW);
        // Pequeno atraso para não sobrecarregar o loop (se o LED não
        estiver piscando)
        delay(500);
    }
}

```

}