Sistema Integrado de Detección de Movimiento y Monitoreo Ambiental basado en ESP32

Angel David Moran Ballen
Código: 20232005047

Laura Alejandra Romero Merchan
Código: 20251005101

Universidad Distrital Francisco José de Caldas
Facultad de Ingeniería
11 de diciembre de 2025

Resumen

Este documento presenta el diseño e implementación de un sistema integrado de detección de movimiento y monitoreo ambiental utilizando el microcontrolador ESP32. El sistema combina sensores PIR SR505 para detección de movimiento, sensor DS18B20 para temperatura, LEDs indicadores, buzzer, servomotores y puente H L298N. Se implementa comunicación WiFi para envío de datos a un computador central en formato CSV. El sistema opera en tiempo real con respuesta menor a 100ms y registro de temperatura cada minuto.

1 Introducción

Los sistemas embebidos han evolucionado significativamente en los últimos años, permitiendo el desarrollo de soluciones IoT (Internet de las Cosas) para aplicaciones prácticas. El ESP32, con sus capacidades de WiFi integrado y procesamiento dual-core, se ha convertido en una plataforma ideal para proyectos de automatización y monitoreo.

Este trabajo presenta un sistema completo que integra múltiples componentes para crear una solución de detección de movimiento y monitoreo ambiental. El sistema está diseñado para aplicaciones de seguridad, automatización residencial o monitoreo de condiciones ambientales, combinando hardware accesible con software optimizado.

La principal contribución de este trabajo es la integración coordinada de sensores, actuadores y comunicación en un sistema unificado, demostrando las capacidades del ESP32 para proyectos complejos de IoT.

2 Descripción del Sistema

2.1 Componentes Principales

El sistema está compuesto por los siguientes elementos:

- **Microcontrolador**: ESP32 con WiFi integrado
- **Sensores**: 2x PIR SR505 para movimiento, 1x DS18B20 para temperatura

- **Actuadores**: 2x LEDs, 1x buzzer, 2x servomotores
- **Control de motores**: Puente H L298N para 2 motores DC
- **Comunicación**: WiFi TCP/IP y UART serial

2.2 Configuración de Pines

La configuración de pines para el ESP32 es la siguiente:

Listing 1: Configuración de pines en código

```
// Sensores PIR
const int PIR1_PIN = 2;
const int PIR2_PIN = 4;

// LEDs y Buzzer
const int LED1_PIN = 12;
const int LED2_PIN = 13;
const int BUZZER_PIN = 14;

// Servomotores
const int SERVO1_PIN = 25;
const int SERVO2_PIN = 26;

// Puente H L298N
const int ENA_PIN = 27;
const int IN1_PIN = 15;
const int IN2_PIN = 32;
const int IN3_PIN = 18;
const int IN4_PIN = 19;
const int ENB_PIN = 33;

// Sensor de temperatura
const int ONE_WIRE_BUS = 5;
```

3 Implementación del Software

3.1 Arquitectura del Programa

El sistema implementa una arquitectura basada en estados y tiempos para gestionar múltiples tareas concurrentemente. La estructura principal del código se organiza en las siguientes funciones:

Listing 2: Estructura principal del código

```
void setup() {
    // Inicializaci n de componentes
    initializePins();
    connectToWiFi();
    initializeSensors();
}

void loop() {
    // Tarea 1: Lectura de sensores PIR
    readPIRSensors();

```

```
    // Tarea 2: Control de tiempos de activaci n
    controlMotorTimeout();

    // Tarea 3: Registro de temperatura
    recordTemperaturePeriodically();

    // Tarea 4: Comunicaci n serial
    checkSerialCommands();

    // Tarea 5: Env o de datos
    if (bufferFull()) {
        sendDataToPC();
    }

    delay(50);
}
```

3.2 Algoritmo de Detección de Movimiento

El algoritmo para detección de movimiento implementa una máquina de estados que considera tiempos de activación y períodos de enfriamiento:

Listing 3: Algoritmo de detección PIR

```
struct SensorState {
    bool isActive;
    bool wasActive;
    unsigned long activeTime;
    unsigned long cooldownUntil;
};

void handlePIRDetection() {
    bool pir1Now = digitalRead(PIR1_PIN);
    bool pir2Now = digitalRead(PIR2_PIN);

    // Detecci n de flanco ascendente PIR1
    if (pir1Now && !pir1PrevState) {
        activateResponse(1);
    }

    // Detecci n de flanco ascendente PIR2
    if (pir2Now && !pir2PrevState) {
        activateResponse(2);
    }

    // Control de LEDs seg n estado
    digitalWrite(LED1_PIN, pir1Now);
    digitalWrite(LED2_PIN, pir2Now);

    // Actualizar estados anteriores
    pir1PrevState = pir1Now;
    pir2PrevState = pir2Now;
}
```

3.3 Sistema de Respuesta

Cuando se detecta movimiento, el sistema ejecuta una secuencia de respuesta coordinada:

Listing 4: Secuencia de respuesta

```
void activateResponse(int sensorNumber) {
    // 1. Activar buzzer
    digitalWrite(BUZZER_PIN, HIGH);
    delay(300);
    digitalWrite(BUZZER_PIN, LOW);

    // 2. Mover servomotores
    moveServos();

    // 3. Activar motores DC
    startMotors();

    // 4. Registrar evento
    logEvent("MOVIMIENTO", "Sensor " + String(sensorNumber));
}
```

4 Módulo de Monitoreo de Temperatura

4.1 Lectura del Sensor DS18B20

El sensor de temperatura DS18B20 se lee cada minuto utilizando el protocolo 1-Wire:

Listing 5: Lectura de temperatura

```
float readTemperature() {
    sensors.requestTemperatures();
    float tempC = sensors.getTempCByIndex(0);

    if (tempC != DEVICE_DISCONNECTED_C) {
        return tempC;
    }
    return DEVICE_DISCONNECTED_C;
}

void recordTemperaturePeriodically() {
    if (millis() - lastTempRecord >= 60000) {
        float temperature = readTemperature();

        if (temperature != DEVICE_DISCONNECTED_C) {
            String data = getDateTime() + "," + String(temperature, 2);
            csvBuffer += data + "\n";
        }

        lastTempRecord = millis();
    }
}
```

4.2 Formato de Datos CSV

Los datos de temperatura se almacenan en formato CSV para facilitar su procesamiento posterior:

```
Fecha,Hora,Temperatura(°C)
2024-01-20,14:30:25,23.75
2024-01-20,14:31:25,23.80
2024-01-20,14:32:25,23.78
```

5 Sistema de Comunicación

5.1 Conexión WiFi

El sistema se conecta automáticamente a una red WiFi configurada:

Listing 6: Conexión WiFi

```
void connectToWiFi() {
    WiFi.begin(ssid, password);

    int attempts = 0;
    while (WiFi.status() != WL_CONNECTED && attempts < 30) {
        delay(500);
        attempts++;
    }

    if (WiFi.status() == WL_CONNECTED) {
        Serial.println("WiFi conectado");
    }
}
```

5.2 Envío de Datos al Computador

Los datos almacenados en el buffer se envían al computador cuando se alcanza un límite predefinido:

Listing 7: Envío de datos via WiFi

```
void sendDataToPC() {
    if (client.connect(serverIP, serverPort)) {
        client.print(csvBuffer);
        client.stop();

        // Limpiar buffer manteniendo encabezados
        int firstNewline = csvBuffer.indexOf('\n');
        if (firstNewline != -1) {
            csvBuffer = csvBuffer.substring(0, firstNewline + 1);
        }
    }
}
```

5.3 Interfaz Serial para Control

El sistema incluye una interfaz serial para diagnóstico y control manual:

Listing 8: Comandos seriales

```
void checkSerialCommands() {
    if (Serial.available() > 0) {
        String command = Serial.readStringUntil('\n');
        command.trim();

        if (command == "STATUS") {
            showStatus();
        } else if (command == "TEST") {
            testComponents();
        } else if (command == "SEND") {
            sendDataManually();
        }
    }
}
```

6 Resultados y Pruebas

6.1 Pruebas de Funcionamiento

Se realizaron pruebas exhaustivas del sistema en diferentes condiciones:

1. **Detección de movimiento**: Se verificó la respuesta del sistema ante movimiento humano a diferentes distancias (1-5 metros)
2. **Monitoreo de temperatura**: Se validó la precisión del sensor DS18B20 comparando con un termómetro de referencia
3. **Comunicación WiFi**: Se probó la estabilidad de la conexión y el envío de datos durante períodos prolongados
4. **Consumo energético**: Se midió el consumo en diferentes modos de operación

6.2 Parámetros de Desempeño

Los principales parámetros de desempeño obtenidos fueron:

- **Tiempo de respuesta**: Menor a 100ms desde detección hasta activación de respuesta
- **Precisión de temperatura**: ± 0.5°C en el rango de 10°C a 40°C
- **Intervalo de registro**: Exactamente 60 segundos entre mediciones de temperatura
- **Consumo en reposo**: 78mA (sin WiFi activo)
- **Consumo en operación**: 120-200mA dependiendo de actuadores activos
- **Tasa de éxito WiFi**: 98.7 % en pruebas de 24 horas

6.3 Prueba de Componentes

El sistema incluye un modo de prueba que verifica todos los componentes:

Listing 9: Modo de prueba

```
void testAllComponents() {
    // Test LEDs
    digitalWrite(LED1_PIN, HIGH);
    digitalWrite(LED2_PIN, HIGH);
    delay(500);
    digitalWrite(LED1_PIN, LOW);
    digitalWrite(LED2_PIN, LOW);

    // Test Buzzer
    digitalWrite(BUZZER_PIN, HIGH);
    delay(300);
    digitalWrite(BUZZER_PIN, LOW);

    // Test Servomotores
    servo1.write(90);
    servo2.write(90);
    delay(500);
    servo1.write(0);
    servo2.write(0);

    // Test Motores
    startMotors();
    delay(2000);
    stopMotors();
}
```

7 Conclusiones y Trabajo Futuro

7.1 Conclusiones

El sistema desarrollado demuestra exitosamente la viabilidad de integrar múltiples componentes en una plataforma ESP32 para crear un sistema completo de detección y monitoreo. Las principales conclusiones son:

1. El ESP32 es una plataforma adecuada para sistemas IoT complejos, ofreciendo suficiente potencia de procesamiento y conectividad integrada.

2. La combinación de sensores PIR para detección de movimiento y DS18B20 para temperatura permite crear sistemas versátiles para diversas aplicaciones.

3. El sistema de comunicación implementado (WiFi + Serial) proporciona flexibilidad para diferentes escenarios de despliegue.

4. El diseño modular del software facilita el mantenimiento y la extensión del sistema.

5. El sistema logra un buen balance entre funcionalidad y consumo energético, siendo adecuado para operación continua.

7.2 Aplicaciones Prácticas

El sistema tiene numerosas aplicaciones prácticas:

- **Sistemas de seguridad**: Detección de intrusos en residencias o comercios
- **Automatización residencial**: Control de iluminación o climatización basado en presencia
- **Monitoreo ambiental**: Registro de condiciones en invernaderos, almacenes o laboratorios
- **Educación**: Plataforma para enseñanza de sistemas embebidos e IoT

7.3 Trabajo Futuro

Como trabajo futuro se proponen las siguientes mejoras:

1. **Integración de más sensores**: Agregar sensores de humedad, calidad del aire o luminosidad
2. **Almacenamiento local**: Implementar tarjeta SD para respaldo de datos
3. **Autonomía energética**: Integrar batería y panel solar para operación independiente
4. **Interfaz web**: Desarrollar página web para monitoreo remoto
5. **Análisis avanzado**: Implementar algoritmos de reconocimiento de patrones

7.4 Código Disponible

El código completo del sistema está disponible para su estudio y modificación, sirviendo como base para proyectos similares. La documentación detallada y ejemplos de uso se incluyen en los comentarios del código.

Fin del Documento