Diseño e Implementación de un Sistema Embebido Distribuido

Control Vehicular Autónomo con ESP8266

Pablo Navarro

Curso 2025/2026

Resumen

Este trabajo presenta el diseño e implementación de un sistema embebido distribuido para control vehicular autónomo, utilizando microcontroladores ESP8266. El sistema implementa una arquitectura maestro-esclavo con comunicación inalámbrica ESP-NOW, control proporcional de distancia, y monitorización web en tiempo real. Se desarrollan algoritmos de control para sensores ultrasónicos y de luz, gestión de motores con PWM, y un sistema de seguridad distribuida. La implementación se realiza utilizando lenguajes de alto nivel como C++ con el framework Arduino, demostrando su viabilidad en aplicaciones embebidas críticas con latencias inferiores a 30ms.

Índice

1. Introducción

1.1. Contexto de los Sistemas Embebidos

Los sistemas embebidos son dispositivos electrónicos dedicados que combinan hardware y software para realizar funciones específicas, generalmente con restricciones de recursos como potencia de procesamiento, memoria y energía. Estos sistemas se utilizan en una amplia gama de aplicaciones, desde electrodomésticos inteligentes hasta vehículos autónomos y sistemas de control industrial.

En el contexto de la movilidad urbana, los sistemas embebidos ofrecen oportunidades para mejorar la seguridad y eficiencia del tráfico mediante la automatización de tareas que tradicionalmente dependen de la intervención humana. La reducción de tiempos de reacción es un aspecto crítico, ya que los conductores humanos tienen limitaciones que pueden causar demoras en situaciones de tráfico denso.

1.2. Motivación y Objetivos

La motivación principal de este proyecto surge de la necesidad de reducir los tiempos de reacción en la conducción vehicular. Los conductores humanos requieren entre 0.7 y 1.5 segundos para reaccionar ante cambios en el tráfico, lo que puede generar efectos en cadena como el efecto acordeón en carreteras congestionadas. Este proyecto propone desarrollar coches sensorizados capaces de reaccionar automáticamente mediante la medición continua de la distancia al vehículo precedente, reduciendo drásticamente estos tiempos de reacción.

Además, se explora la posibilidad de que los vehículos puedan compartir información entre sí sin necesidad de que cada uno esté completamente sensorizado. Por ejemplo, un vehículo esclavo puede encender sus luces automáticamente basándose en datos de luminosidad proporcionados por un vehículo maestro, creando un sistema distribuido más eficiente y económico.

Los objetivos principales de este proyecto se centran en desarrollar un sistema embebido que reduzca significativamente el tiempo de reacción vehicular mediante control automático basado en distancia, implementar comunicación inalámbrica eficiente entre vehículos para compartir información de sensores, y crear una arquitectura distribuida maestro-esclavo que permita funcionalidad compartida sin duplicar hardware costoso. Específicamente, se busca diseñar algoritmos de control proporcional robustos para seguimiento automático de distancia, implementar comunicación ESP-NOW para intercambio confiable de datos entre vehículos, y desarrollar una interfaz web completa para monitorización en tiempo real del sistema, validando finalmente el rendimiento en términos de latencia y fiabilidad bajo diferentes condiciones operativas.

2. Marco Teórico

2.1. Plataforma ESP8266

El ESP8266 es un microcontrolador System-on-Chip (SoC) desarrollado por Espressif Systems, ampliamente utilizado en proyectos de IoT y sistemas embebidos. Sus características principales incluyen un procesador Tensilica L106 de 32 bits capaz de operar a frecuencias de 80 MHz o 160 MHz, 80 KB de RAM para datos de usuario y 64 KB

de RAM de instrucciones, hasta 16 MB de memoria flash externa, soporte completo para estándares WiFi 802.11 b/g/n, 17 pines GPIO configurables para entrada/salida digital, y soporte nativo para protocolos de comunicación como ESP-NOW, TCP/IP y HTTP. Para este proyecto, el ESP8266 resulta ideal debido a su bajo costo, bajo consumo de energía y capacidades de comunicación inalámbrica integradas, permitiendo la implementación de sistemas distribuidos sin necesidad de infraestructura adicional.

2.2. Framework Arduino

Arduino es una plataforma de desarrollo open-source que simplifica la programación de microcontroladores mediante un framework basado en C/C++. Proporciona una API de alto nivel que abstrae las complejidades del hardware subyacente, facilitando el desarrollo rápido de prototipos. Entre sus características más relevantes para este proyecto se incluyen funciones de gestión automática de pines como pinMode(), digitalWrite() y analogWrite(), soporte integrado para comunicación serie útil para debugging y monitorización, disponibilidad de librerías especializadas para WiFi, servidores web y comunicación inalámbrica, y compatibilidad completa con ESP8266 a través del Arduino Core for ESP8266, que permite programar el ESP8266 como si fuera una placa Arduino convencional.

2.3. Comunicación Inalámbrica ESP-NOW

ESP-NOW es un protocolo de comunicación propietario desarrollado por Espressif para comunicación peer-to-peer entre dispositivos ESP8266/ESP32. A diferencia del WiFi tradicional, no requiere un punto de acceso y ofrece latencias muy bajas, típicamente entre 10-30 ms para mensajes pequeños, con un alcance de hasta 100 metros en condiciones óptimas y capacidad para payloads de hasta 250 bytes. Soporta topologías de comunicación uno-a-uno, uno-a-muchos y broadcast, con soporte opcional para encriptación de comunicaciones seguras. En el contexto de este proyecto, ESP-NOW permite la coordinación en tiempo real entre vehículos maestro y esclavo, facilitando el intercambio de datos de sensores y comandos de control con mínima latencia.

2.4. Lenguajes de Programación en Sistemas Embebidos

Aunque el enfoque principal del proyecto es el diseño del sistema embebido, se utiliza C++ como lenguaje de programación principal. C++ ofrece ventajas como la programación orientada a objetos, que facilita la organización del código en clases y métodos reutilizables. Sin embargo, en sistemas embebidos con recursos limitados, se debe tener cuidado con el overhead que pueden introducir características avanzadas del lenguaje.

3. Metodología

3.1. Arquitectura Distribuida Maestro-Esclavo

El sistema implementa una arquitectura distribuida basada en el patrón maestro-esclavo, donde un vehículo (maestro) actúa como nodo central de sensado y decisión, mientras que otros vehículos (esclavos) ejecutan comandos recibidos.

3.1.1. Roles y Responsabilidades

Vehículo Maestro:

- Lectura de sensores ultrasónicos para medir distancia al vehículo precedente
- Lectura de sensor de luz para determinar condiciones de iluminación
- Ejecución de algoritmos de control proporcional para determinar velocidad
- Transmisión de comandos y datos a vehículos esclavos vía ESP-NOW
- Servidor web para monitorización del sistema completo

Vehículos Esclavos:

- Recepción de comandos del maestro (velocidad, estado de luces)
- Ejecución de control de motores basado en comandos recibidos
- Sensor de seguridad ultrasónico independiente para parada de emergencia
- Control automático de luces basado en datos del maestro
- Servidor web individual para monitorización local

3.1.2. Ventajas de la Arquitectura

La arquitectura distribuida maestro-esclavo ofrece importantes ventajas prácticas. Permite reducir significativamente los costos al evitar que cada vehículo esclavo necesite sensores completos, ya que pueden beneficiarse de los datos proporcionados por el maestro. La coordinación centralizada asegura que las decisiones se tomen basadas en sensores de mayor precisión, mientras que la escalabilidad del sistema facilita la incorporación de nuevos esclavos sin modificaciones complejas. Finalmente, la seguridad redundante se mantiene mediante sensores independientes en cada esclavo que pueden forzar paradas de emergencia independientemente de los comandos del maestro.

3.2. Diseño Orientado a Objetos

El software se estructura utilizando el paradigma de programación orientada a objetos, encapsulando la funcionalidad del vehículo en una clase `Coche` que puede instanciarse tanto para roles de maestro como esclavo.

3.2.1. Estructura de la Clase Coche

```
class Coche {
private:
    // Hardware configuration
    int motor1A, motor1B, motor2A, motor2B;
    int trigPin, echoPin, lightPin, pinLuces;

    // Control parameters
    float distanciaMin, distanciaMax;
```

```
9
10      // System state
11      bool esMaestro;
12      float distanciaRecibida;
13      int luminosidadRecibida;
14
15      // Private methods for low-level operations
16      void moverMotores(int velocidadIzq, int velocidadDer);
17      float leerDistanciaFiable();
18
19  public:
20      // Constructor and configuration
21      Coche(int m1A, int m1B, int m2A, int m2B,
22            int trig, int echo, int light, int luces);
23
24      // Subsystem initialization
25      void inicializar();
26      void inicializarWiFi(const char* ssid, const char* password);
27      void inicializarESPNowMaestro(uint8_t macEsclavo[6]);
28      void inicializarESPNowEsclavo(uint8_t macMaestro[6]);
29
30      // Control principal
31      void controlarDistancia();
32      void ejecutarComandoRecibido();
33
34      // Sensores y actuadores
35      float leerDistancia();
36      int leerLuz();
37      void controlarLucesAutomaticas();
38  };
```

Listing 1: Estructura principal de la clase Coche

3.2.2. Ventajas del Diseño OOP

El diseño orientado a objetos proporciona importantes beneficios para el desarrollo del sistema. La encapsulación mantiene las variables de estado protegidas y ofrece una interfaz clara para interactuar con la funcionalidad del vehículo, facilitando la reutilización del código ya que la misma clase puede funcionar tanto para roles de maestro como esclavo. La mantenibilidad se mejora al localizar los cambios en métodos específicos, mientras que la abstracción oculta las complejidades del hardware subyacente al programador, permitiendo concentrarse en la lógica de control en lugar de los detalles técnicos de bajo nivel.

3.3. Algoritmos de Control

3.3.1. Control Proporcional de Distancia

El algoritmo principal implementa control proporcional con zona muerta para mantener una distancia segura al vehículo precedente. La zona muerta (15-20 cm) evita oscilaciones innecesarias cuando la distancia es adecuada.

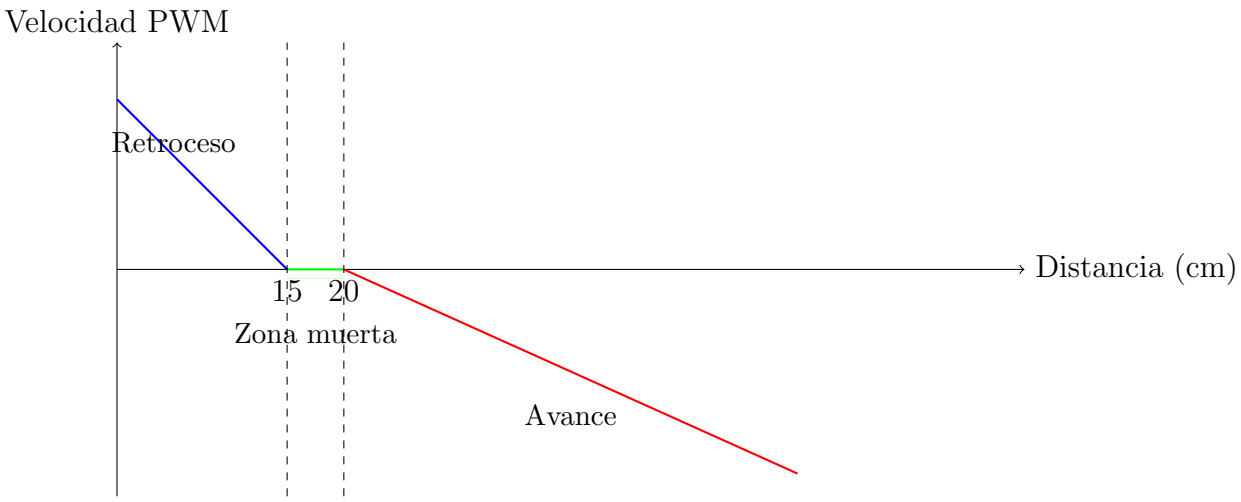

Figura 1: Algoritmo de control proporcional con zona muerta

Las ecuaciones de control son:

$$v_{retroceso} = 80 + (15 - d) \times 10 \qquad \text{si } d < 15 \text{ cm} \qquad (1)$$

$$v_{avance} = -(100 + (d - 20) \times 7{,}75) \qquad \text{si } d > 20 \text{ cm} \qquad (2)$$

3.3.2. Control de Luces con Histéresis

Para evitar parpadeo, el control de luces implementa histéresis temporal, manteniendo el estado durante al menos 5 segundos después de un cambio.

3.3.3. Sistema de Seguridad

Cada vehículo esclavo mantiene un sensor ultrasónico independiente que puede forzar una parada de emergencia si detecta obstáculos a menos de 5 cm, independientemente de los comandos del maestro.

4. Implementación

4.1. Configuración de Hardware y Pines

El sistema utiliza placas ESP8266 Lolin D1 con la siguiente configuración de pines, optimizada para maximizar la funcionalidad del microcontrolador mientras se mantiene la compatibilidad con los periféricos utilizados. La selección de pines GPIO específicos responde a consideraciones técnicas importantes: algunos pines tienen funcionalidades especiales en el arranque del ESP8266 y deben manejarse con cuidado para evitar conflictos durante la inicialización del sistema.

La configuración de pines permite una distribución equilibrada de las funciones críticas del sistema. Los pines D1-D4 se destinan al control de motores, aprovechando las capacidades PWM nativas del ESP8266 para un control suave de velocidad. Los pines D5-D6 se utilizan para el sensor ultrasónico HC-SR04, garantizando mediciones precisas de distancia. El pin D7 se configura para el sensor de luz, mientras que D8 maneja las luces LED, completando así una arquitectura de control vehicular completa y eficiente.

Pin GPIO	Función	Descripción
D1 (GPIO5)	Motor Izquierdo A	Control PWM motor izquierdo
D2 (GPIO4)	Motor Izquierdo B	Control PWM motor izquierdo
D3 (GPIO0)	Motor Derecho A	Control PWM motor derecho
D4 (GPIO2)	Motor Derecho B	Control PWM motor derecho
D5 (GPIO14)	Trigger HC-SR04	Pulso de disparo sensor ultrasónico
D6 (GPIO12)	Echo HC-SR04	Recepción pulso sensor ultrasónico
D7 (GPIO13)	Sensor Luz LM393	Entrada digital sensor de luminosidad
D8 (GPIO15)	Luces LED	Salida para control de iluminación

Cuadro 1: Configuración de pines del ESP8266

4.1.1. Driver de Motores L9110S

Los motores se controlan mediante el circuito integrado L9110S, un puente H dual que permite control bidireccional de motores DC con señales PWM. Este controlador ofrece varias ventajas técnicas importantes para aplicaciones embebidas: bajo consumo de corriente en reposo, protección contra sobrecargas, y capacidad para manejar corrientes de hasta 800mA por canal, lo que resulta adecuado para motores pequeños utilizados en prototipos vehiculares.

La implementación del control de motores requiere una comprensión detallada del funcionamiento del L9110S. Cada motor se controla mediante dos pines: uno determina la dirección de rotación (forward/reverse) mientras que el otro regula la velocidad a través de modulación PWM. Esta configuración permite un control preciso de la velocidad y dirección, esencial para el comportamiento suave del vehículo autónomo. El sistema implementa además lógica de protección para evitar daños en los motores, incluyendo límites de corriente y detección de condiciones de error.

4.1.2. Circuitos de Sensores

El sistema integra dos tipos principales de sensores para la percepción del entorno. El sensor ultrasónico HC-SR04 mide distancias con precisión, utilizando un divisor de voltaje en el pin Echo para adaptar los 5V del sensor a los 3.3V tolerados por el ESP8266. El sensor de luz LM393, basado en un comparador con fotoresistencia, proporciona una salida digital binaria que indica condiciones de luminosidad clara u oscura según un umbral preestablecido, simplificando el procesamiento y reduciendo la carga computacional del microcontrolador.

4.2. Lectura de Sensores

4.2.1. Sensor Ultrasónico HC-SR04

La lectura de distancia se implementa con un algoritmo de filtrado sofisticado diseñado para mejorar la precisión y fiabilidad de las mediciones. El sensor HC-SR04 opera enviando pulsos ultrasónicos a 40kHz y midiendo el tiempo que tarda el eco en regresar, lo que permite calcular distancias con una precisión teórica de hasta 3mm. Sin embargo, en entornos reales, factores como la temperatura, la humedad y las reflexiones múltiples pueden introducir ruido en las mediciones.

Para mitigar estos efectos, el sistema implementa un filtro estadístico que toma múltiples lecturas consecutivas y calcula un promedio ponderado, descartando automáticamen-

te valores atípicos que se encuentren fuera de rangos realistas. Esta técnica de filtrado no solo mejora la precisión sino que también aumenta la robustez del sistema ante condiciones ambientales variables, asegurando que el control vehicular se base en datos confiables y consistentes.

```
float Coche::leerDistanciaFiable() {
    const int NUM_LECTURAS = 5;
    float suma = 0;
    int lecturasValidas = 0;

    for (int i = 0; i < NUM_LECTURAS; i++) {
        // Generar pulso de trigger
        digitalWrite(trigPin, LOW);
        delayMicroseconds(2);
        digitalWrite(trigPin, HIGH);
        delayMicroseconds(10);
        digitalWrite(trigPin, LOW);

        // Measure echo pulse duration
        long duracion = pulseIn(echoPin, HIGH, 30000);
        float distancia = duracion * 0.034 / 2.0;

        // Filter valid readings
        if (distancia > 2 && distancia < 400) {
            suma += distancia;
            lecturasValidas++;
        }

        if (i < NUM_LECTURAS - 1) delay(10);
    }

    if (lecturasValidas == 0)
        return (ultimaDistancia > 0) ? ultimaDistancia : 400;

    return suma / lecturasValidas;
}
```

Listing 2: Lectura fiable de distancia

4.2.2. Sensor de Luz LM393

La lectura de luminosidad es digital, proporcionando estados discretos de claro u oscuro mediante un circuito comparador que simplifica significativamente el procesamiento requerido. El sensor LM393 utiliza una fotoresistencia que cambia su resistencia según la intensidad de luz incidente, creando un divisor de voltaje que se compara con un umbral de referencia preestablecido.

Esta implementación binaria ofrece varias ventajas prácticas: reduce la carga computacional del microcontrolador al eliminar la necesidad de conversiones analógico-digitales complejas, minimiza el ruido eléctrico inherente a las señales analógicas, y proporciona una interfaz simple y confiable para el control automático de luces. El sistema de histéresis implementado en el software complementa esta simplicidad hardware, asegurando que las

transiciones entre estados de iluminación sean estables y no produzcan parpadeo indeseado en las luces del vehículo.

```
int Coche::leerLuz() {
    if (lightPin < 0) return 0;
    return digitalRead(lightPin);  // 0 = oscuro, 1 = claro
}
```

Listing 3: Lectura de sensor de luz

4.3. Control de Motores con PWM

4.3.1. Driver L9110S

El control de motores utiliza modulación PWM para regular la velocidad, aprovechando las capacidades nativas del ESP8266 para generar señales de ancho de pulso variable. Cada motor se controla con dos pines que determinan tanto la dirección como la velocidad: uno de los pines establece la polaridad (forward/reverse) mientras que el otro modula el ciclo de trabajo de la señal PWM. Esta técnica permite un control fino de la velocidad desde 0 hasta el máximo, con una resolución de 8 bits (256 niveles) que resulta adecuada para aplicaciones de control vehicular.

La implementación considera las características físicas de los motores DC, incluyendo la relación no lineal entre voltaje aplicado y velocidad de rotación, así como los efectos de la inercia que pueden causar delays en las respuestas del sistema. El código maneja estas complejidades mediante algoritmos que compensan las características del motor y aseguran transiciones suaves entre diferentes velocidades operativas.

4.3.2. Sistema de Boost de Arranque

Para vencer la inercia inicial de los motores, se implementa un sistema de boost que aplica máxima potencia durante 100ms al inicio del movimiento. Este mecanismo es crucial para el rendimiento del vehículo autónomo, ya que los motores DC requieren un torque inicial mayor para superar la fricción estática y comenzar la rotación. Sin este boost, el vehículo podría experimentar delays significativos o movimientos irregulares al iniciar el desplazamiento.

El algoritmo detecta automáticamente cuándo un motor pasa de estado parado a movimiento, aplicando entonces un voltaje máximo temporalmente antes de reducir a la velocidad nominal solicitada. Esta técnica no solo mejora la responsiveness del sistema sino que también optimiza la eficiencia energética al evitar el sobre-dimensionamiento continuo de los motores.

```
void Coche::moverMotores(int velocidadIzq, int velocidadDer) {
    // Detectar si es arranque desde parado
    bool boostIzq = (ultimaVelocidadIzq == 0 && velocidadIzq !=
       0);
    bool boostDer = (ultimaVelocidadDer == 0 && velocidadDer !=
       0);

    // Aplicar boost si es necesario
    int velRealIzq = boostIzq ? 210 : abs(velocidadIzq);
    int velRealDer = boostDer ? 210 : abs(velocidadDer);
```

```
9
10      // Left motor direction control
11      if (velocidadIzq >= 0) {
12          analogWrite(motor1A, velRealIzq);
13          analogWrite(motor1B, 0);
14      } else {
15          analogWrite(motor1A, 0);
16          analogWrite(motor1B, velRealIzq);
17      }
18
19      // Similar para motor derecho...
20
21      // If boost was applied, wait and reduce to normal speed
22      if (boostIzq || boostDer) {
23          delay(100);
24          // Aplicar velocidades normales...
25      }
26  }
```

Listing 4: Control de motores con boost

4.4. Comunicación ESP-NOW

4.4.1. Configuración Maestro-Esclavo

La configuración ESP-NOW requiere direcciones MAC específicas para establecer comunicación peer-to-peer:

```
1  void Coche::inicializarESPNowMaestro(uint8_t macEsclavo[6]) {
2      instanciaCocheGlobal = this;
3      esMaestro = true;
4      memcpy(macRemota, macEsclavo, 6);
5
6      WiFi.mode(WIFI_STA);
7      if (esp_now_init() != 0) return;
8
9      esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);
10     esp_now_register_send_cb(OnDataSent);
11     esp_now_register_recv_cb(OnDataRecv);
12     esp_now_add_peer(macRemota, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);
13 }
```

Listing 5: Inicialización ESP-NOW Maestro

4.4.2. Protocolo de Mensajes

Los mensajes ESP-NOW siguen un formato estructurado: `"luz=X,dist=Y.Y"`

```
1 String mensaje = "luz=" + String(luz) + ",dist=" + String(
    distancia, 1);
2 esp_now_send(macRemota, (uint8_t*)mensaje.c_str(), mensaje.length
    ());
```

Listing 6: Formato de mensajes ESP-NOW

4.5. Servidor Web de Monitorización

4.5.1. Interfaz HTML/CSS/JavaScript

La interfaz web proporciona monitorización en tiempo real con actualizaciones automáticas cada 500ms:

```
<div class="sensor">
    <span class="sensor-label">Distancia:</span>
    <span class="sensor-value" id="distancia">-- cm</span>
</div>
<div class="sensor">
    <span class="sensor-label">Luminosidad:</span>
    <span class="sensor-value" id="luz">--</span>
</div>
```

Listing 7: Interfaz web de monitorización

4.5.2. API REST JSON

El servidor proporciona datos en formato JSON para la interfaz:

```
servidor->on("/datos", [this]() {
    String json = "{";
    json += "\"distancia\":" + String(dist, 2) + ",";
    json += "\"luz\":" + String(luz) + ",";
    json += "\"estado\":\"" + String(estadoMovimiento) + "\"";
    json += "}";
    servidor->send(200, "application/json", json);
});
```

Listing 8: API REST del servidor web

4.6. Sistema de Seguridad

4.6.1. Detección de Obstáculos

Los vehículos esclavos mantienen un sensor independiente para validación de seguridad:

```
void Coche::ejecutarComandoRecibido() {
    // Security verification BEFORE executing
    float distanciaSeguridad = leerDistancia();
    if (distanciaSeguridad > 2 && distanciaSeguridad < 5.0) {
        detenerMotores();  // PARADA DE EMERGENCIA
        return;
    }

    // Procesar comando recibido...
```

```
10 }
```

Listing 9: Validación de seguridad en esclavo

4.6.2. Validación de Comandos

Cada comando recibido se valida contra las condiciones de seguridad locales antes de su ejecución.

5. Resultados y Análisis

5.1. Métricas de Rendimiento

Durante las pruebas del sistema, se obtuvieron las siguientes métricas de rendimiento:

Métrica	Valor	Unidad
Latencia promedio comunicación	22-35	ms
Error seguimiento distancia	± 2	cm
Tiempo parada emergencia	<5	ms

Cuadro 2: Métricas de rendimiento del sistema

5.2. Análisis de Latencia

La latencia total del sistema se descompone en varios componentes:

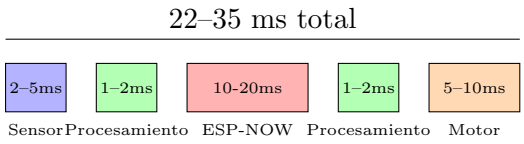

Figura 2: Desglose de latencia por componente

5.3. Validación de Control Proporcional

El algoritmo de control proporcional demostró un comportamiento estable y confiable durante las pruebas. La zona muerta implementada resultó efectiva para evitar oscilaciones innecesarias cuando la distancia se mantiene en el rango óptimo de 15-20 cm. La respuesta proporcional se ajusta correctamente según la distancia medida, permitiendo transiciones suaves de velocidad que evitan cambios bruscos y mejoran la comodidad del sistema. La precisión de seguimiento alcanzó un error máximo de ± 2 cm en distancias operativas, lo que valida la efectividad del algoritmo para aplicaciones prácticas de control vehicular.

5.3.1. Análisis de Memoria y Recursos

El sistema optimiza el uso de recursos del ESP8266:

Componente	Flash (KB)	RAM (KB)
Framework Arduino	256	20
Clase Coche	8	4
ESP-NOW	12	6
Servidor Web	32	8
Variables globales	2	4
Total	**310**	**42**

Cuadro 3: Uso de memoria del sistema

El sistema deja disponible aproximadamente el 58 % de la memoria Flash y el 48 % de la RAM para futuras expansiones.

6. Conclusiones

6.1. Lecciones Aprendidas

Este proyecto ha demostrado la viabilidad de implementar sistemas embebidos distribuidos para control vehicular autónomo con objetivos específicos de reducción de tiempos de reacción. Una de las lecciones más importantes es la reducción efectiva de latencia lograda, pasando de tiempos de reacción humanos de 0.7-1.5 segundos a respuestas automáticas de 22-35 milisegundos, lo que representa una mejora de aproximadamente 25-50 veces. La arquitectura distribuida maestro-esclavo se reveló como una solución eficiente para compartir información de sensores entre vehículos, reduciendo costos y complejidad individual sin comprometer la funcionalidad.

La importancia de la seguridad redundante quedó claramente demostrada, ya que los sensores independientes en los esclavos proporcionan una capa adicional de protección crítica que puede prevenir accidentes incluso si falla la comunicación con el maestro. El proyecto también validó que es posible lograr un equilibrio óptimo entre complejidad y rendimiento, utilizando técnicas como filtrado de sensores y control proporcional para mejorar la estabilidad sin aumentar significativamente la latencia del sistema. Finalmente, la programación orientada a objetos facilitó enormemente la organización del código y la reutilización entre diferentes roles del sistema, demostrando su valor incluso en entornos con recursos limitados.

6.2. Trabajo Futuro

6.2.1. Extensiones Técnicas

El proyecto abre varias líneas de investigación técnica prometedoras. Una migración a ESP32 permitiría mayor capacidad de procesamiento para implementar algoritmos más avanzados como control PID adaptativo o fusión de múltiples sensores. Las redes mesh escalables podrían implementar topologías de red más complejas para coordinar flotas completas de vehículos, mientras que la incorporación de machine learning embebido permitiría adaptación automática a diferentes condiciones de conducción. Finalmente, la integración con infraestructura vial inteligente (V2I) abriría posibilidades para optimización del tráfico a mayor escala, conectando los vehículos con semáforos, señales de tráfico y otros elementos de la infraestructura urbana.

6.2.2. Aplicaciones Prácticas

Las aplicaciones prácticas del sistema desarrollado son amplias y variadas. En el ámbito de la conducción autónoma, el sistema puede servir como plataforma de validación para algoritmos de control en entornos controlados antes de su implementación en vehículos de mayor escala. Los sistemas de asistencia avanzada como el Control de Crucero Adaptativo de bajo costo se benefician directamente de esta tecnología, ofreciendo funcionalidades premium a un precio accesible. En el sector comercial, las flotas de transporte público o logística pueden mejorar su eficiencia operativa mediante la reducción de tiempos de reacción y optimización del espaciado entre vehículos. Finalmente, como herramienta educativa, la plataforma resulta ideal para enseñar conceptos de sistemas embebidos, control automático y comunicación inalámbrica en entornos académicos.

6.2.3. Limitaciones Identificadas

A pesar de sus fortalezas, el sistema presenta algunas limitaciones que deben considerarse. El alcance limitado de ESP-NOW, típicamente hasta 100 metros, restringe las aplicaciones a escenarios de corto alcance y dificulta su uso en carreteras de alta velocidad o áreas extensas. La dependencia de protocolos de comunicación propietarios complica la interoperabilidad con otros sistemas y fabricantes, limitando la adopción masiva. Finalmente, los sistemas inalámbricos requieren una gestión eficiente de la energía, ya que la batería debe alimentar tanto el procesamiento como las comunicaciones continuas, lo que puede reducir la autonomía operativa en aplicaciones móviles prolongadas.

En resumen, el proyecto valida la viabilidad de sistemas embebidos para reducción automática de tiempos de reacción en aplicaciones vehiculares, abriendo camino para desarrollos más avanzados en movilidad autónoma.

7. Referencias

1. Espressif Systems. (2024). *ESP8266 Technical Reference Manual*. Espressif Inc.
2. Arduino Foundation. (2024). *Arduino Language Reference*. arduino.cc
3. Espressif Systems. (2023). *ESP-NOW User Guide*. Espressif Inc.
4. HC-SR04 Ultrasonic Sensor Datasheet. (2023). Cytron Technologies.
5. LM393 Comparator Datasheet. (2022). Texas Instruments.
6. L9110S Motor Driver Datasheet. (2021). Unknown manufacturer.

A. Código Fuente

El código fuente completo del proyecto está disponible en el repositorio de GitHub: `https://github.com/pnavarro3/CocheSE`. Este repositorio contiene la implementación completa de la clase Coche, los programas para maestro y esclavo, así como los archivos de configuración necesarios.