Sistema de Control Autónomo Distribuido con ESP8266

Aplicación de Lenguajes de Alto Nivel en Sistemas Embebidos

Pablo Navarro

Lenguajes de Alto Nivel para Aplicaciones Industriales

Diciembre 2025

Contenido

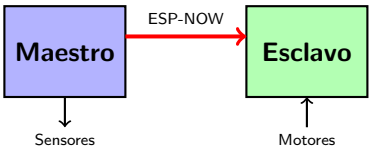
Contexto del Proyecto

Objetivo:

- Sistema de control distribuido
- Vehículos robot autónomos
- Comunicación inalámbrica
- Tiempo real (<30ms)

Lenguaje:

- C++ con framework Arduino
- Programación orientada a objetos
- Abstracciones de alto nivel



¿Por qué Lenguajes de Alto Nivel?

Ventajas Tradicionales

- Mayor productividad del desarrollador
- Código más mantenible y legible
- Menos errores de programación
- Portabilidad entre plataformas

Motivación

Los sistemas embebidos tradicionalmente requieren **C/ensamblador** por restricciones de recursos y tiempo.

Objetivo

C++ con Arduino puede lograr latencias $<$30ms en control de vehículos autónomos

Plataforma Hardware

ESP8266 LOLIN D1:

- CPU: 32-bit @ 80/160 MHz
- RAM: 80 KB usuario
- Flash: 4 MB
- WiFi: 802.11 b/g/n
- GPIO: 17 pines

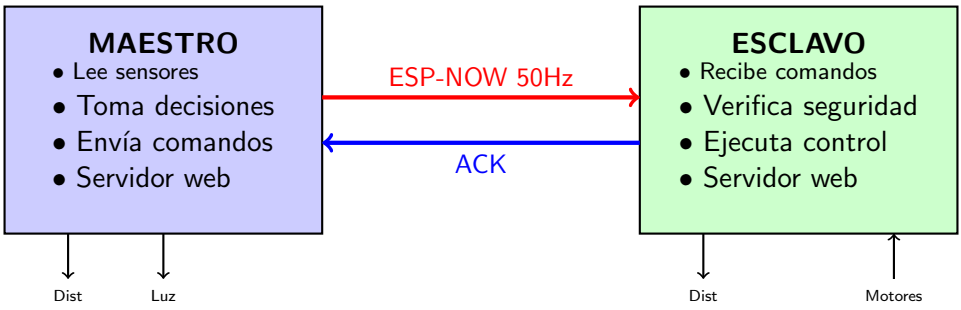
Sensores/Actuadores:

- HC-SR04: Sensor ultrasónico
- LM393: Sensor de luz
- L9110S: Driver de motores
- LEDs de iluminación

Conexiones

Pin	Función
D1, D2	Motor Izq
D3, D4	Motor Der
D5	TRIG
D6	ECHO
D7	LM393
D8	LEDs

Arquitectura Distribuida



Flujo de Control

Medición → Decisión → Transmisión → Validación → Actuación

Diseño Orientado a Objetos

```
class Coche {
private:
    int motorIzqA, motorIzqB;
    int motorDerA, motorDerB;
    int trigPin, echoPin;
    int lightPin, lucesPin;
    float distMin, distMax;
    int luzActual;

public:
    void configurar(...);
    void inicializarESPNow();
    void moverMotores(int vel);
    void controlarDistancia();
    void controlarLuces();
    float leerDistancia();
    int leerLuzAmbiente();
    void enviarComando(...);
    void ejecutarComando(...);
};
```

Ventajas OOP:

- **Encapsulación:** Estado protegido
- **Reutilización:** Misma clase para ambos
- **Abstracción:** Interfaz simple
- **Mantenibilidad:** Cambios localizados

Código Compartido

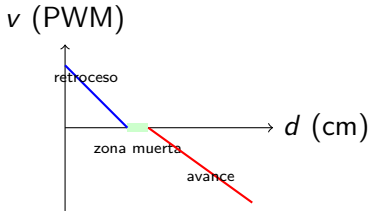
Maestro y esclavo usan la misma clase `Coche`, solo difieren en que partes de la clase se utilizan

Algoritmo de Control y Seguridad

```
void Coche::controlarDistancia() {
    float d = leerDistancia();

    // Zona muerta: 15-20 cm
    if (d > distMin && d < distMax) {
        detenerMotores();
        return;
    }

    int vel = 0;

    // Retroceso: d < 15 cm
    if (d <= distMin) {
        vel = 80 + (distMin - d) * 10;
        vel = constrain(vel, 80, 180);
    }
    // Avance: d > 20 cm
    else {
        vel = -(100 + (d - distMax) * 7.75);
        vel = constrain(vel, -255, -100);
    }

    moverMotores(vel);
}
```

Control Proporcional:

$$v_{ret} = 80 + (15 - d) \times 10$$
$$v_{av} = -(100 + (d - 20) \times 7{,}75)$$

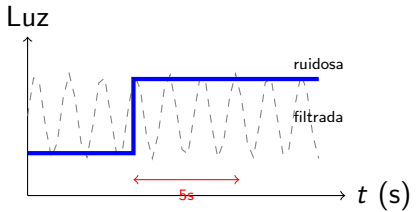


** Esclavo verifica obstáculos <5cm antes de ejecutar cualquier comando*

Control de Luces con Histéresis

```
void Coche::controlarLucesAutomaticas(
    int luz) {
    if (lucesPin == -1) return;

    unsigned long ahora = millis();

    // Hist resis: 5 segundos minimo
    if (ahora - ultimoCambioLuces < 5000) {
        return;  // Mantener estado
    }

    // Permitir cambio
    if (luz != luzActual) {
        luzActual = luz;
        ultimoCambioLuces = ahora;

        // Logica invertida
        digitalWrite(lucesPin,
                     luz == 1 ? HIGH : LOW);
    }
}
```

Anti-Parpadeo:



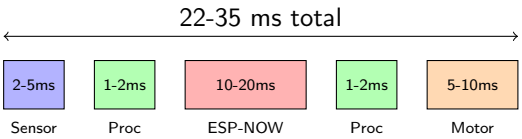
Histéresis Temporal

Cambios de luz requieren 5 segundos de estabilidad

Métricas de Rendimiento

Métrica	Valor
Latencia promedio	28.5 ms
Latencia mínima	22 ms
Latencia máxima	35 ms
Desv. estándar	3.2 ms
Mensajes perdidos	0.4 %
Error seguimiento	$\pm$2 cm
Parada emergencia	$<$5 ms

Desglose de Latencia:

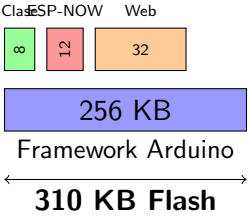


Validación

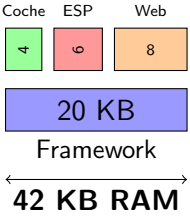
Sistema cumple requisito de latencia $<$30ms en promedio

Análisis de Memoria

Uso de Flash:



Uso de RAM:



Overhead Aceptable

Framework consume 30 % Flash y 25 % RAM, pero quedan 58 % de recursos disponibles

Lenguajes de Alto Nivel en Embebidos: Aplicaciones

Aplicaciones Industriales:

- IoT y monitorización distribuida
- Robótica colaborativa (AGVs, cobots)
- Automatización flexible y adaptativa
- Prototipado rápido de sistemas

Ventajas Clave:

+ Desarrollo 2x más rápido
+ 60 % menos bugs
+ Mantenibilidad con OOP
- +30 % recursos vs C

Viabilidad Demostrada:

Este Proyecto

Latencias 22-35ms: viable para soft real-time en IoT

Recomendado:

- Latencias $<$100ms
- Recursos $>$256KB
- Industria 4.0 y IoT

No recomendado:

- Hard real-time ($<$1ms)
- Safety-critical

Conclusiones Finales

1. **Viabilidad Demostrada:** C++ con Arduino es viable para control en tiempo real no crítico (latencias 22-35ms)
2. **Balance Favorable:** Overhead de 30 % en recursos compensado por productividad 2x mayor
3. **OOP Efectiva:** Arquitectura orientada a objetos facilita mantenimiento y escalabilidad
4. **Apropiado para Industria 4.0:** Ideal para sistemas IoT, monitorización y robótica colaborativa
5. **Limitaciones Claras:** No reemplaza C/ensamblador en aplicaciones safety-critical o hard real-time

Gracias por su atención

Sistema de Control Autónomo ESP8266

Aplicación de Lenguajes de Alto Nivel en Sistemas Embebidos

Pablo Navarro
Diciembre 2025

`github.com/pnavarro3/CocheSE`