



# Vehículo Autónomo **ESP32 y HiveMQ**

Este proyecto detalla la implementación de un vehículo autónomo controlado por un ESP32, comunicando con un broker MQTT público (HiveMQ) mediante MQTT sobre TLS/WSS, permitiendo control tanto desde aplicaciones IoT como desde dashboards web HTTPS.

## Descripción General del Proyecto



### Controlado por ESP32

El corazón del sistema, gestionando hardware y comunicaciones.



### Conectado a HiveMQ Broker Público

Utiliza un broker MQTT público con soporte WSS para comunicación segura desde navegadores web.



### Comunicación MQTT sobre TLS

Protocolo ligero y seguro para el intercambio de mensajes.

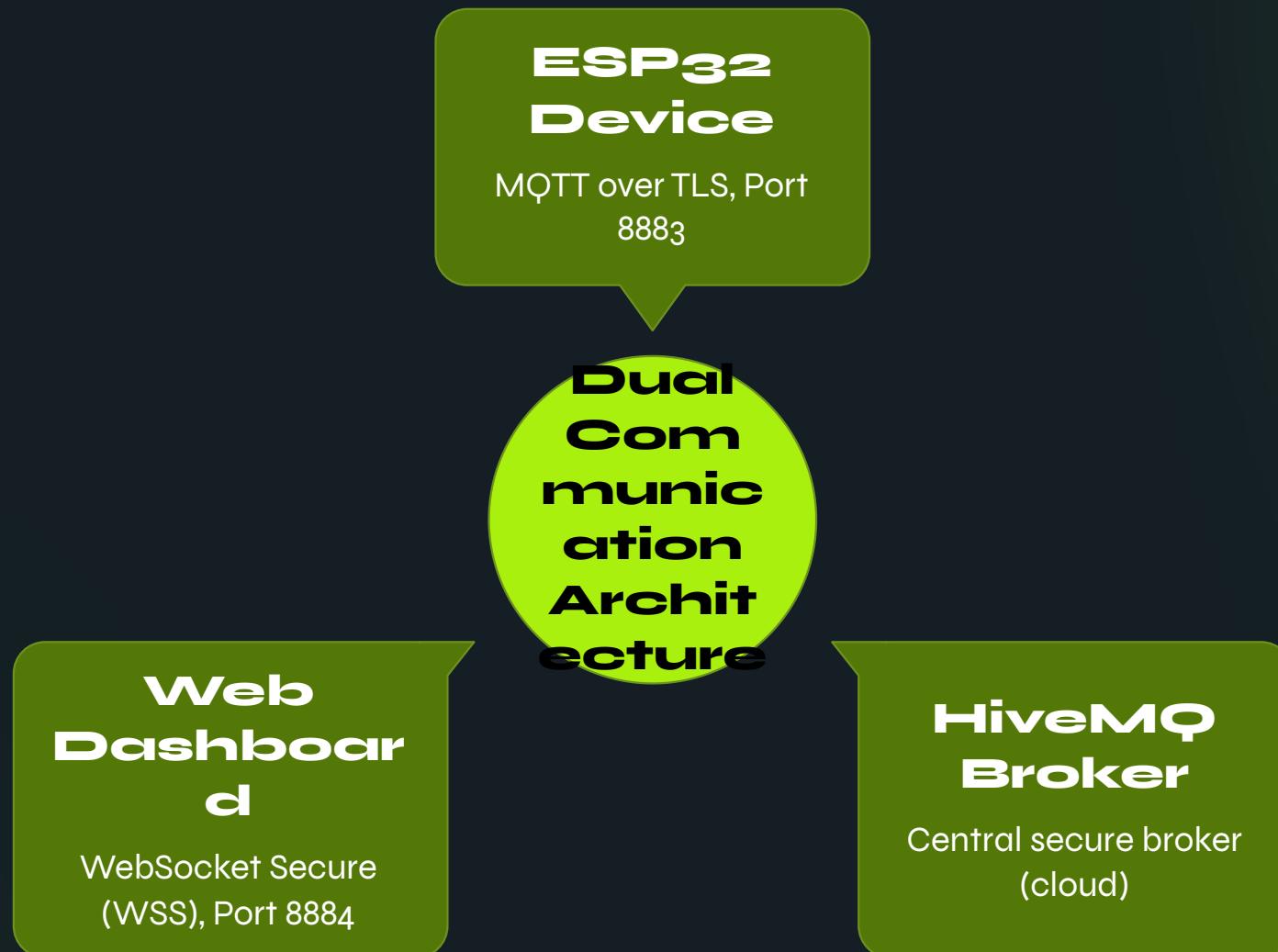


### Certificados X.509

Asegura la autenticación y cifrado de la comunicación.

# Arquitectura de Comunicación Dual

El sistema soporta ahora dos tipos de conexión, garantizando versatilidad y seguridad para diferentes clientes.



## **ESP32 ⇌ HiveMQ Broker (MQTT sobre TLS)**

- Puerto 8883 para conexiones TLS tradicionales
- Ideal para dispositivos IoT como el ESP32
- Autenticación mediante usuario/contraseña

## **Dashboard Web ⇌ HiveMQ Broker (WebSocket Seguro - WSS)**

- Puerto 8884 para conexiones WSS
- Compatible con navegadores HTTPS
- Permite dashboards web en tiempo real

# Ventajas y Consideraciones de HiveMQ Broker Público

El uso del broker público HiveMQ ofrece múltiples beneficios, especialmente para fases de desarrollo, aunque presenta algunas limitaciones importantes en seguridad para datos sensibles en producción.



## Configuración Rápida y Sencilla

Inicia tu proyecto sin complicaciones ni largas configuraciones.



## Soporte Nativo para WSS

Ideal para integrar dashboards web HTTPS, facilitando la visualización en tiempo real.



## Sin Costos de Infraestructura

Ahorra en hardware y mantenimiento al no necesitar servidores propios.



## Perfecto para Prototipos y Demos

Una solución ágil y eficiente para probar y presentar tus ideas.



## Cifrado TLS/WSS en Transporte

Garantiza la seguridad de tus comunicaciones durante el envío de datos.

## Consideraciones de Seguridad

### ✓ Tráfico Cifrado Extremo a Extremo

Todos los mensajes están protegidos de intercepciones y accesos no autorizados.

### ✓ Nadie Puede Interceptar Mensajes

La seguridad inherente de TLS/WSS asegura que tus datos permanezcan privados.

### ✗ Servidor Compartido por Miles de Usuarios

Al ser un servicio público, compartes la infraestructura con una gran cantidad de usuarios, lo que puede introducir limitaciones.

### ✗ Sin Control de Acceso Granular (ACLs)

No permite definir permisos detallados para usuarios o tópicos específicos, lo que es crucial para aplicaciones empresariales.

### ✗ No Apto para Datos Sensibles en Producción

Debido a la naturaleza compartida y la falta de control granular, no se recomienda para manejar información confidencial en entornos de producción.

**Ideal para:** Proyectos educativos, demos, prototipos y desarrollo.

Made with **GAMMA**

# Capacidades del ESP32 a Bordo



## Control de Motores

Gestión precisa de la movilidad del vehículo a través del L298N.



## Lectura de Distancia

Sensor HC-SR04 para la detección de obstáculos y navegación.



## IMU (MPU6050)

Acelerómetro y giroscopio para la orientación y estabilidad del vehículo.

El ESP32 actúa como el cerebro del vehículo, integrando diversos sensores y actuadores para su funcionamiento.

# API MQTT: Configuración de Conexión HiveMQ

## Configuración de Conexión HiveMQ

Para la conexión MQTT, utilizaremos el broker público de HiveMQ con la siguiente configuración:

- **Broker:** broker.hivemq.com
- **Puerto TLS (ESP32):** 8883
- **Puerto WSS (Dashboard Web):** 8884
- **Autenticación:** Usuario/contraseña (opcional para broker público)

Los tópicos de suscripción y publicación se mantienen como antes.

## Payloads de Movimiento

```
{ "dir": "forward", "speed": 50, "duration": 2 }
```

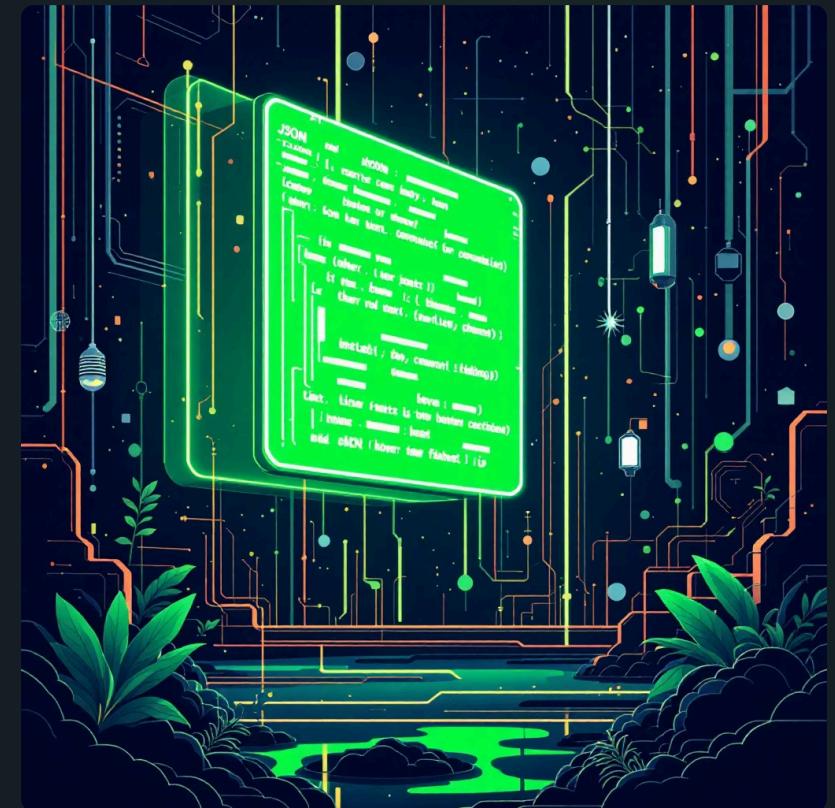
- dir: "forward", "backward", "left", "right", "stop".
- speed: 0-100% (convertido a PWM 0-255).
- duration: segundos (máximo 5s).

## Payload de Detención

```
{ "stop": true }
```

## Payload de Distancia Segura

```
{ "safe_stop_cm": 30 }
```



# API MQTT: Publicación de Telemetría

→ **sebas/car/status**

Estado actual del movimiento: { "moving": true, "dir": "forward", "speed": 120 }

→ **sebas/car/distance**

Lecturas del sensor de distancia: { "cm": 25.4, "obstacle": false }

→ **sebas/car imu**

Datos del acelerómetro y giroscopio: { "ax": 0.12, "ay": 9.80, ... }

→ **sebas/car/alert**

Notificaciones de eventos, como obstáculos: { "type": "obstacle", "cm": 12.5 }

→ **sebas/car/lwt**

Mensaje "Last Will and Testament" al conectar: { "status": "online" }

# Estructura Interna del Firmware

01

## **setup()**

- Inicialización de pines
- Configuración WiFi STA
- Configuración TLS + autenticación
- Conexión a HiveMQ Broker
- Suscripción a tópicos de comandos
- Publicación de estado inicial

02

## **loop()**

- Mantenimiento de conexión MQTT
- Control de movimiento con timeout
- Detección activa de obstáculos
- Envío periódico de telemetría

Una estructura clásica de Arduino para un control robusto y reactivo.

# Librerías Clave Utilizadas



## **<Arduino.h>**

Funciones básicas y configuración del entorno Arduino.



## **<WiFi.h> & <WiFiClientSecure.h>**

Conexión a redes Wi-Fi y establecimiento de conexiones seguras.



## **<PubSubClient.h>**

Implementación del cliente MQTT para la comunicación con el broker público de HiveMQ.



## **<Wire.h>**

Comunicación I2C para sensores como el MPU6050.



## **Drivers Internos**

Control directo de PWM y GPIO para motores y otros periféricos.

# Optimización de Memoria

## Uso Eficiente

El firmware actual demuestra un uso de memoria muy bajo, optimizado para las capacidades del ESP32.

"Sketch uses 1346 bytes (76%) of program storage space."

Este resultado, generado por el IDE durante la compilación, indica un código compacto y eficiente.



# Limitaciones Actuales

## Reconexión WiFi

Reintentos básicos sin backoff exponencial ni manejo avanzado.

## Parsing JSON Manual

Implementación frágil; se recomienda usar ArduinoJson para robustez.

## Control PID Inexistente

El giro de los motores carece de precisión debido a la ausencia de control PID.

## Movimiento Limitado

Duración máxima de movimiento de 5 segundos, definida por MAX\_MOVE\_MS.

## Validación de Payload

Falta de validación profunda, lo que puede llevar a errores con payloads mal formados.

# Próximos Pasos y Mejoras

1

## Reconexión Robusta

Implementar estrategias de reconexión WiFi avanzadas.

2

## Integrar ArduinoJson

Mejorar el parsing de comandos para mayor fiabilidad.

3

## Control PID

Añadir control PID para mayor precisión en el movimiento.

4

## Validación de Comandos

Implementar validación estricta de payloads entrantes.

# Alternativas para Producción

Para aquellos proyectos que necesiten migrar de un broker público a una solución más robusta y controlada, presentamos las siguientes alternativas:



## AWS IoT Core con WSS

- Máxima seguridad con certificados X.509
- Soporte nativo para WebSockets Seguros
- Escalabilidad empresarial
- Costos por uso



## Mosquitto en EC2 con WSS

- Control total sobre la configuración
- Certificados SSL propios
- ACLs personalizadas
- Costo fijo de servidor



## Broker Privado Dedicado

- Máximo control y personalización
- Configuración de seguridad específica
- Ideal para aplicaciones críticas
- Requiere más mantenimiento

La elección de la opción adecuada dependerá de las necesidades específicas de seguridad, escalabilidad, control y presupuesto de cada proyecto.