Sistema IoT de Control de Montacargas con Motor Dahlander

Pablo Navarro Domínguez

19 de diciembre de 2025

Índice

1. Introducción

1.1. Meta de aprendizaje de la asignatura

Diseño de un sistema de interconexión e integración usando IoT para la gestión del funcionamiento en una aplicación industrial, en concreto de un motor eléctrico.

1.2. Objetivo general

Diseñar e implementar un sistema de control automatizado basado en tecnologías IoT para un montacargas industrial equipado con motor Dahlander, que permita el posicionamiento preciso, monitoreo de variables ambientales y operación remota mediante interfaz web, demostrando la interconexión e integración de múltiples dispositivos para la gestión inteligente del motor eléctrico.

1.3. Objetivos específicos

1. Desarrollar una arquitectura IoT distribuida basada en microcontroladores ESP32 que se comuniquen mediante el protocolo MQTT para separar las funciones de adquisición de datos y control del sistema.

2. Implementar un sistema de adquisición de datos que integre sensores de posición (HC-SR04), temperatura y humedad (DHT11), y luminosidad (LDR) con filtrado digital para garantizar mediciones confiables.

3. Diseñar e implementar un algoritmo de control de posición basado en retroalimentación del sensor ultrasónico que permita posicionar el montacargas en cuatro plantas predefinidas con precisión de ± 2 cm.

4. Desarrollar un sistema de control de velocidad variable que aproveche las dos velocidades del motor Dahlander mediante un perfil trapezoidal para optimizar tiempos de desplazamiento minimizando oscilaciones.

5. Implementar mecanismos de seguridad que incluyan bloqueo de comandos durante operación, validación de integridad de datos del sensor y reconexión automática al broker MQTT.

6. Crear una interfaz web responsive que permita el control remoto del sistema, visualización en tiempo real de telemetría y gestión del estado operativo desde cualquier dispositivo con navegador.

7. Integrar un display LCD I2C para proporcionar visualización local del estado del sistema independiente de la conectividad de red.

8. Implementar integración con News API para obtener y mostrar noticias actualizadas sobre IoT en la interfaz web, demostrando la capacidad del ESP32 para consumir APIs REST externas y parsear respuestas JSON complejas.

9. Desarrollar un sistema de caché de noticias en memoria del ESP32 que optimice el uso de la API externa y reduzca la latencia de la interfaz web.

10. Documentar completamente el sistema incluyendo esquemas de conexión, especificaciones técnicas, procedimientos de configuración y código fuente comentado.

2. Fundamentos Teóricos

2.1. Arquitectura IoT del Sistema

El sistema implementa una arquitectura IoT distribuida en cuatro capas: percepción (sensores ESP32), red (WiFi/MQTT), procesamiento (broker Mosquitto y ESP32 de control), y aplicación (interfaz web y LCD). Esta estructura permite la interconexión de dispositivos para la gestión del motor eléctrico mediante comunicación inalámbrica.

2.2. News API y ArduinoJson

News API es un servicio REST que proporciona acceso a noticias de más de 80,000 fuentes (100 peticiones/día en plan gratuito). En este proyecto se obtienen noticias sobre IoT para mostrar en la interfaz web. ArduinoJson es la biblioteca utilizada para parsear las respuestas JSON y extraer título, descripción, fuente y URL de cada artículo.

2.3. Protocolo MQTT

MQTT es un protocolo ligero de mensajería pub-sub con overhead mínimo, ideal para IoT. El ESP32 de sensores publica datos a tópicos (`montacargas/sensores/*`), mientras que el ESP32 de control se suscribe para recibirlos. El broker Mosquitto gestiona la comunicación entre módulos. Se utiliza QoS 0 dado que las mediciones se publican cada 2 segundos.

2.4. Microcontrolador ESP32

El ESP32 es un microcontrolador de bajo costo con CPU dual-core de 240 MHz, 520 KB SRAM y WiFi integrado (IEEE 802.11 b/g/n). Dispone de 34 GPIO configurables, ADC de 12 bits, e interfaces I2C, SPI y UART. En este proyecto se utiliza WiFi para conectar a MQTT, GPIO para control de relés y lectura de sensores, e I2C para el LCD. Voltaje lógico de 3.3V.

2.5. Sensores y Actuadores

2.5.1. Sensor ultrasónico HC-SR04

Sensor de distancia por ultrasonidos (40 kHz) con rango de 2-400 cm y precisión de $\pm$ 3 mm. Calcula distancia mediante $d = \frac{v \cdot t}{2}$ donde $v = 343$ m/s. Requiere alimentación de 5V y pulso trigger de 10 μs. Se utiliza para determinar la posición del montacargas midiendo distancia a punto de referencia fijo.

2.5.2. Sensor de temperatura y humedad DHT11

Sensor digital de un solo cable que mide temperatura (0-50°C, $\pm$ 2°C) y humedad relativa (20-90 % RH, $\pm$ 5 %). Frecuencia máxima de muestreo: 1 Hz. Alimentación: 3.3-5V (compatible con ESP32). Se emplea para monitoreo de condiciones ambientales del sistema.

2.5.3. Fotoresistor LDR

Componente cuya resistencia varía inversamente con la luz (oscuridad: ¿1 MΩ; luz intensa: ¡1 kΩ). Se utiliza módulo LDR con comparador integrado que proporciona salida digital: HIGH en oscuridad, LOW con luz. Umbral ajustable mediante potenciómetro. Se emplea para detectar condiciones de iluminación y activar el LED indicador cuando es necesario.

2.5.4. Módulo de relés HW-316

Módulo de 4 relés que interfaz entre ESP32 y contactores del motor. Características: voltaje de control 5V, capacidad 10A @ 250VAC, optoacoplamiento para aislamiento galvánico. Control en cascada: ESP32 (GPIO 26/25/27/14 a 3.3V) $\rightarrow$ Relés HW-316 $\rightarrow$ Contactores auxiliares 24V (KA1-4) $\rightarrow$ Contactores potencia (KM) del motor trifásico.

2.6. Motor Dahlander

Motor asíncrono trifásico con dos velocidades mediante conmutación de bobinados. Modifica el número de polos activos según $n_s = \frac{120 \cdot f}{p}$ donde f es frecuencia (Hz) y p número de polos. En el proyecto, 4 relés HW-316 controlan contactores auxiliares: RELE1/KA1 (subida lenta), RELE2/KA2 (bajada lenta), RELE3/KA3 (velocidad alta), RELE4/KA4 (power general). Se implementan transiciones controladas con tiempos de espera entre conmutaciones para seguridad.

3. Diseño del Sistema

3.1. Arquitectura general

El sistema implementa una arquitectura IoT distribuida basada en el patrón publicador-suscriptor mediante MQTT, compuesta por cuatro elementos principales:

- **Módulo de Sensores (ESP32):** Adquisición de datos ambientales y de posición, publica telemetría vía MQTT
- **Broker MQTT (Mosquitto):** Intermediario de comunicaciones ejecutándose en computador portátil
- **Módulo de Control (ESP32):** Suscriptor MQTT, ejecuta lógica de control, interfaces web y LCD
- **Motor Dahlander:** Actuador principal controlado mediante módulo de 4 relés

Esta arquitectura ofrece modularidad (separación entre sensado y control), escalabilidad (fácil agregar sensores/actuadores), mantenibilidad (módulos independientes) y flexibilidad (ubicación física distribuida mediante Wi-Fi).

3.2. Diagrama de bloques

El flujo de información en el sistema sigue esta secuencia:

1. El **módulo de sensores** lee datos de:
    - Sensor HC-SR04 (distancia cada 250 ms con filtro de mediana)

- Sensor DHT11 (temperatura y humedad cada 2 s)
- Módulo LDR (luminosidad como evento digital)

2. Los datos se publican a los tópicos MQTT:
   - `montacargas/sensores/distancia`
   - `montacargas/sensores/temperatura`
   - `montacargas/sensores/humedad`
   - `montacargas/sensores/luminosidad`
3. El **broker MQTT** retransmite los mensajes a todos los suscriptores interesados.
4. El **módulo de control**:
   - Recibe y procesa la telemetría
   - Ejecuta el algoritmo de control de posición
   - Actualiza el display LCD local
   - Sirve la interfaz web con datos en tiempo real
   - Genera señales de control (3.3V) en pines GPIO
5. Los **relés HW-316**:
   - Reciben señales del ESP32 (nivel lógico 3.3V)
   - Proporcionan aislamiento galvánico mediante optoacopladores
   - Conmutan contactos de mayor potencia (10A @ 250VAC)
6. Los **contactores auxiliares (KA1-KA4)**:
   - Son activados por los contactos de los relés HW-316
   - Operan con bobinas a 24VDC
   - Comandan los contactores de potencia del motor
7. Los **contactores de potencia (KM1-KM5)**:
   - KM1, KM2: Controlan sentido de giro (inversión de fases)
   - KM3: Configuración de velocidad rápida (menos polos)
   - KM4, KM5: Configuración de velocidad lenta (más polos) según dirección

- Reconfiguran conexiones del motor Dahlander

8. El **motor Dahlander** mueve el montacargas, modificando la distancia medida por el sensor, cerrando el lazo de control.

Diagrama de flujo de control:

$$\text{Sensores} \xrightarrow{\text{MQTT}} \text{ESP32 Control} \xrightarrow{\text{3.3V}} \text{Relés} \xrightarrow{\text{Contactos}} \text{KA} \xrightarrow{\text{24V}} \text{KM} \xrightarrow{\text{L1,L2,L3}} \text{Motor} \tag{1}$$

3.3. Esquemas eléctricos del sistema

3.3.1. Circuito de potencia trifásico

La Figura 1 muestra el circuito de potencia completo del motor Dahlander trifásico. Este circuito incluye la alimentación desde las líneas L1, L2 y L3, el pulsador Q1 para control manual, los fusibles F1 y F2 para protección de sobrecorriente, y los cinco contactores de potencia (KM1-KM5) que controlan la configuración del motor.

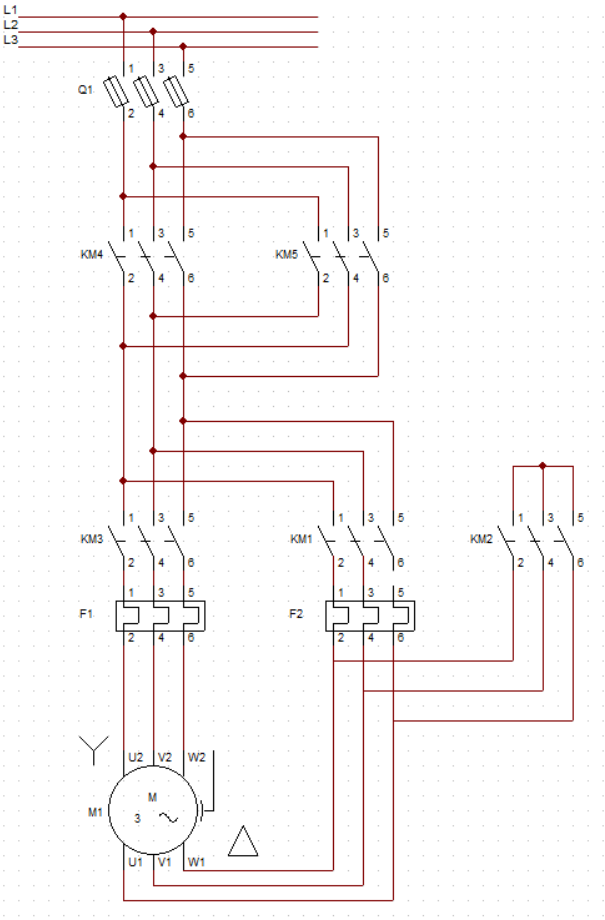

Figura 1: Esquema eléctrico del circuito de potencia del motor Dahlander

Los contactores de potencia cumplen las siguientes funciones:

- **KM1:** Utilizado junto con KM4 o KM5 para operación en modo lento (subida/bajada)
- **KM2:** Activado junto con KM3 para operación en modo rápido (debe estar desactivado KM1)
- **KM3:** Contactor de velocidad rápida - Se activa únicamente cuando KM2 está activado
- **KM4:** Contactor de subida - Trabaja en conjunto con KM1 (no puede coexistir con KM5)
- **KM5:** Contactor de bajada - Trabaja en conjunto con KM1 (no puede coexistir con KM4)

Secuencias de operación:

- **Subida lenta:** Relé 1 activa KM4 + KM1
- **Bajada lenta:** Relé 2 activa KM5 + KM1
- **Modo rápido:** Relé 3 desactiva KM1 y activa KM2 + KM3
- **Encendido/Apagado:** Relé 4 controla alimentación general

3.3.2. Circuito de control y mando

La Figura 2 presenta el circuito de control a 24V que gestiona la activación de los contactores de potencia mediante los contactores auxiliares KA1-KA4, que son controlados por el ESP32 a través del módulo de relés.

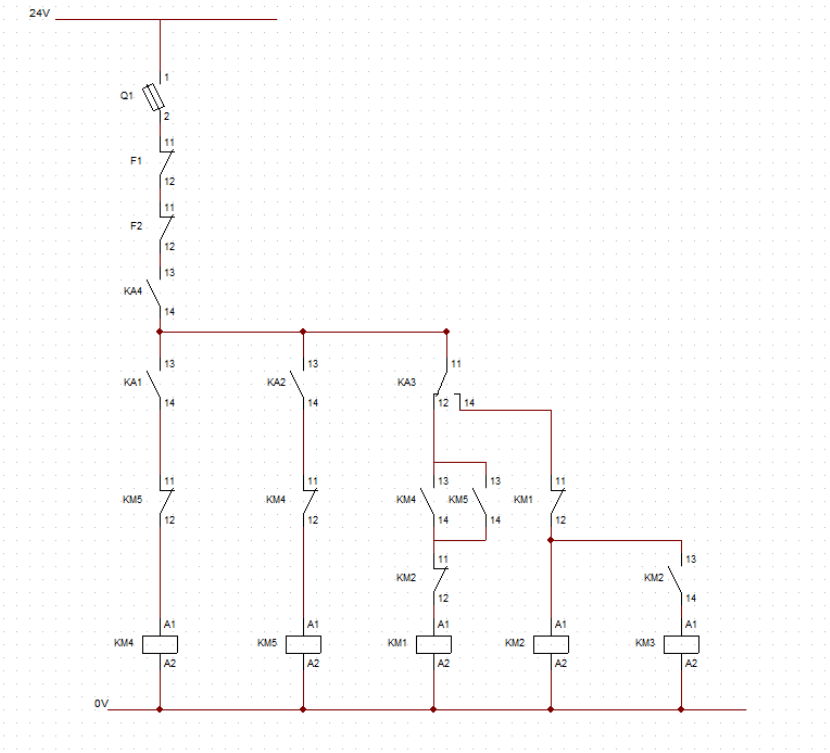

Figura 2: Esquema eléctrico del circuito de control a 24V con contactores auxiliares

El circuito de control implementa una arquitectura en cascada donde el ESP32 controla relés de bajo nivel, que a su vez activan contactores auxiliares

(KA), y finalmente estos contactores auxiliares comandan los contactores de potencia (KM) que manejan el motor trifásico:

Correspondencia de control:

- **Relé 1:** Activa KM4 y KM1 para subir en modo lento
- **Relé 2:** Activa KM5 y KM1 para bajar en modo lento
- **Relé 3:** Desactiva KM1 y activa KM2 y KM3 para activar el modo rápido
- **Relé 4:** Activa/desactiva la alimentación para encender/apagar el sistema

Restricciones de seguridad:

- KM4 y KM5 no pueden estar activados simultáneamente (evita conflicto de direcciones)
- KM1 y KM2 no pueden estar activados simultáneamente (evita cortocircuito de fases)
- KM3 se activa únicamente cuando KM2 está activado (secuencia de velocidad rápida)

Esta arquitectura proporciona aislamiento galvánico entre la lógica de control de bajo voltaje (ESP32 a 3.3V) y el circuito de potencia trifásico, mejorando la seguridad y protegiendo los componentes electrónicos sensibles.

3.4. Comunicación entre módulos

La comunicación entre los módulos se realiza mediante WiFi usando el protocolo MQTT, que permite enviar y recibir datos de forma sencilla y eficiente en una red local. Los ESP32 se conectan automáticamente y el broker central gestiona el intercambio de mensajes.

El sistema está configurado para priorizar la simplicidad y la rapidez en el envío de datos, sin medidas de seguridad avanzadas ya que se trata de un entorno de pruebas. Si se pierde la conexión, los módulos intentan reconectarse automáticamente y el sistema se detiene de forma segura hasta recuperar la comunicación.

3.5. Módulo de Sensores

3.5.1. Componentes hardware

El módulo de sensores integra los siguientes componentes conectados a un ESP32:

Componente	**Modelo**	**Función**
Microcontrolador	ESP32 DevKit	Procesamiento y comunicación WiFi
Sensor ultrasónico	HC-SR04	Medición de distancia (2-400 cm)
Sensor temp/humedad	DHT11	Monitoreo ambiental
Fotoresistor	Módulo LDR digital	Detección de luminosidad

Cuadro 1: Componentes del módulo de sensores

3.5.2. Esquema de conexiones

Componente	**Pin ESP32**	**Señal**
HC-SR04 TRIG	GPIO 5	Trigger ultrasónico
HC-SR04 ECHO	GPIO 18	Echo ultrasónico
HC-SR04 VCC	5V	Alimentación
HC-SR04 GND	GND	Tierra
DHT11 Data	GPIO 4	Bus de datos digital
DHT11 VCC	3.3V	Alimentación
DHT11 GND	GND	Tierra
LDR OUT	GPIO 34	Salida digital (HIGH=oscuro)
LDR VCC	3.3V	Alimentación
LDR GND	GND	Tierra

Cuadro 2: Conexiones del módulo de sensores

3.5.3. Función del módulo

El módulo de sensores ejecuta un bucle principal que:

1. **Adquisición de datos ultrasónicos (250 ms):**
   - Lee 5 muestras consecutivas del HC-SR04
   - Aplica filtro de mediana para eliminar valores atípicos
   - Valida que la medición esté en rango válido (2-400 cm)
   - Publica resultado a `montacargas/sensores/distancia`

2. **Adquisición de datos ambientales (2 segundos):**
    - Lee temperatura del DHT11
    - Lee humedad relativa del DHT11
    - Valida lecturas exitosas del sensor
    - Publica a tópicos correspondientes
3. **Monitoreo de luminosidad (continuo):**
    - Lee estado digital del módulo LDR
    - Detecta cambios de estado (claro $\leftrightarrow$ oscuro)
    - Publica eventos de cambio
4. **Gestión de comunicación MQTT:**
    - Mantiene conexión WiFi activa
    - Reconecta al broker si es necesario
    - Monitoriza latencia de publicación

El filtro de mediana implementado para el sensor ultrasónico es crítico para la estabilidad del sistema de control, ya que elimina lecturas erróneas causadas por reflexiones múltiples o interferencias acústicas.

3.6. Módulo de Control

3.6.1. Componentes hardware

Componente	**Modelo**	**Función**
Microcontrolador	ESP32 DevKit	Procesamiento y control
Módulo de relés	HW-316 (4 canales)	Control del motor Dahlander
Display LCD	16x2 I2C (0x27)	Interfaz de usuario local
LED indicador	LED 5mm	Iluminación automática
Resistencia LED	220 Ω	Limitación de corriente

Cuadro 3: Componentes del módulo de control

3.6.2. Esquema de conexiones

Componente	Pin ESP32	Función
Relé 1 (IN1)	GPIO 26	Control subida lenta
Relé 2 (IN2)	GPIO 25	Control bajada lenta
Relé 3 (IN3)	GPIO 27	Control velocidad alta
Relé 4 (IN4)	GPIO 14	Alimentación motor
LCD SDA	GPIO 21	Bus I2C datos
LCD SCL	GPIO 22	Bus I2C reloj
LED ánodo	GPIO 2	Salida digital
LED cátodo	GND	Tierra (con resistencia)

Cuadro 4: Conexiones del módulo de control

Nota importante: El módulo de relés requiere lógica invertida (LOW=activo, HIGH=inactivo) debido al optoacoplador interno.

3.6.3. Función del módulo

El módulo de control es el cerebro del sistema, ejecutando múltiples tareas concurrentes:

1. **Suscripción y procesamiento MQTT:**
   - Recibe telemetría de los 4 tópicos de sensores
   - Actualiza variables globales con datos recibidos
   - Controla LED según luminosidad ambiental
   - Detecta pérdida de señal de sensores
2. **Algoritmo de control de posición:**
   - Determina planta actual basándose en distancia medida
   - Compara posición actual vs. destino solicitado
   - Calcula dirección y velocidad requeridas
   - Activa combinación apropiada de relés
   - Implementa zonas de velocidad variable (perfil trapezoidal)
   - Detecta llegada a destino (± 1 cm de tolerancia)
3. **Servidor web HTTP:**

- Sirve interfaz HTML/CSS/JavaScript
- Endpoints REST para comandos (ir a planta, encender/apagar)
- Endpoint JSON para telemetría en tiempo real
- Actualización asíncrona vía AJAX cada 1 segundo

4. **Actualización de display LCD:**
   - Muestra planta actual y estado del sistema
   - Indica movimiento en progreso
   - Muestra mensajes de error si pierden comunicaciones
5. **Mecanismos de seguridad:**
   - Bloquea nuevos comandos durante movimiento
   - Valida datos de sensor antes de actuar
   - Detiene motor si se pierde señal del sensor
   - Implementa flag de destino alcanzado para evitar oscilaciones
6. **Integración con News API:**
   - Obtiene noticias de IoT cada 5 minutos mediante HTTPS
   - Parsea respuestas JSON (hasta 8KB) usando ArduinoJson
   - Mantiene caché local de 5 noticias más recientes
   - Endpoint /noticias para servir datos a la interfaz web
   - Endpoint /actualizarnoticias para forzar actualización manual
   - Timeout HTTP de 10 segundos para evitar bloqueos
   - Manejo de errores HTTP (401, 429) con mensajes informativos

3.7. Sistema de Control del Motor Dahlander

El control del motor Dahlander constituye uno de los elementos centrales del sistema, permitiendo el desplazamiento del montacargas con velocidad variable según la fase del movimiento. El sistema implementa una arquitectura de control en cascada que separa la lógica de bajo voltaje de la potencia trifásica mediante múltiples etapas de aislamiento.

3.8. APIs REST y Servicios Web

3.8.1. HTTP y REST

HTTP (HyperText Transfer Protocol) es el protocolo fundamental de comunicación en la World Wide Web. REST (Representational State Transfer) es un estilo arquitectónico que define restricciones para crear servicios web escalables y mantenibles.

Los principios fundamentales de REST incluyen:

- **Cliente-Servidor:** Separación de responsabilidades entre interfaz de usuario y almacenamiento de datos
- **Sin estado:** Cada petición del cliente al servidor debe contener toda la información necesaria
- **Cacheable:** Las respuestas deben indicar si pueden ser cacheadas o no
- **Interfaz uniforme:** Recursos identificados mediante URIs, manipulados mediante representaciones

3.8.2. News API

News API es un servicio web que proporciona acceso a artículos de noticias de más de 80,000 fuentes a nivel mundial mediante una API REST. Las características principales incluyen:

- **Búsqueda por palabras clave:** Permite buscar artículos por términos específicos
- **Filtros avanzados:** Por idioma, país, categoría, fecha de publicación
- **Formato JSON:** Respuestas estructuradas fáciles de parsear
- **Plan gratuito:** 100 peticiones por día, suficiente para aplicaciones de desarrollo

En este proyecto, News API se utiliza para obtener noticias relacionadas con IoT y tecnología, enriqueciendo la interfaz web con contenido actualizado relevante para el contexto del sistema.

3.8.3. ArduinoJson

ArduinoJson es una biblioteca C++ eficiente para parsear, serializar y manipular documentos JSON en microcontroladores con recursos limitados. Sus características incluyen:

- **Bajo uso de memoria:** Optimizado para sistemas embebidos
- **Zero-copy:** Minimiza operaciones de copia en memoria
- **Streaming:** Soporte para documentos JSON grandes
- **Validación:** Detección de errores de formato

Se utiliza para parsear las respuestas JSON de News API y extraer la información relevante (título, descripción, fuente, URL) de cada artículo.

3.8.4. Arquitectura de control en cascada

La arquitectura del sistema se estructura en cuatro niveles jerárquicos de control:

Nivel 1 - Lógica de control (ESP32):

El microcontrolador ESP32 ejecuta el algoritmo de control de posición y genera señales digitales a 3.3V en cuatro pines GPIO (26, 25, 27, 14). Esta etapa implementa toda la inteligencia del sistema: procesamiento de datos de sensores, cálculo de trayectorias, implementación del perfil trapezoidal de velocidad y gestión de las secuencias de conmutación seguras.

Nivel 2 - Interfaz de relés (Módulo HW-316):

El módulo de relés HW-316 actúa como primera etapa de aislamiento y amplificación. Cada señal de control del ESP32 activa un optoacoplador que proporciona aislamiento galvánico completo entre la lógica de 3.3V y el circuito de control. Los relés conmutan contactos capaces de manejar hasta 10A a 250VAC, suficiente para activar las bobinas de los contactores auxiliares.

Nivel 3 - Contactores auxiliares (KA1-KA4):

Los contactores auxiliares operan con bobinas alimentadas a 24VDC y actúan como etapa de interposición entre los relés de control (HW-316) y los contactores de potencia. Proporcionan aislamiento, amplificación y permiten implementar enclavamientos mecánicos que evitan combinaciones peligrosas de contactores de potencia.

Nivel 4 - Contactores de potencia (KM1-KM5):

Los contactores de potencia conmuten las líneas trifásicas del motor (L1, L2, L3). En este proyecto sus funciones se definen de la siguiente forma: KM1 actúa como contactor común en los modos de velocidad lenta (se activa

junto con KM4 o KM5); KM2 se emplea en la secuencia de velocidad rápida y debe permanecer desactivado cuando KM1 está activo; KM3 selecciona la configuración de velocidad rápida (menos polos); KM4 y KM5 seleccionan la configuración del sentido de giro para subida y bajada respectivamente (más polos). Las secuencias de activación incluyen enclavamientos y tiempos de espera para evitar cortocircuitos entre fases y proteger el motor.

3.8.5. Configuración de relés y contactores

La tabla siguiente resume la cadena completa de control desde el microcontrolador hasta el motor:

GPIO	Relé	KA	KM	Función en el Motor
26	RELE1	KA1	KM4	Subida lenta (más polos)
25	RELE2	KA2	KM5	Bajada lenta (más polos)
27	RELE3	KA3	KM3	Velocidad rápida (menos polos)
14	RELE4	KA4	Q1	Alimentación general

Cuadro 5: Cadena de control completa: ESP32 → Relés → KA → KM → Motor

Principio de operación:
Cuando el ESP32 establece un pin GPIO en nivel LOW (debido a la lógica invertida del módulo de relés), se activa la bobina del relé correspondiente cerrando su contacto. Este contacto alimenta la bobina del contactor auxiliar KA a 24VDC, el cual cierra sus contactos de mayor potencia. Estos contactos del KA alimentan finalmente la bobina del contactor de potencia KM, que conmuta las conexiones del motor trifásico según la función deseada.

3.8.6. Lógica de control de velocidades

El sistema implementa un perfil de velocidad trapezoidal que aprovecha las dos velocidades del motor:

1. **Fase de arranque (primeros 4 cm):**
   - Se activa el Relé 4 (alimentación general)
   - Se activa el Relé 1 o 2 según dirección (subida/bajada)
   - El motor opera en configuración de velocidad LENTA (más polos)
   - Permite un arranque suave sin sacudidas bruscas
2. **Fase de crucero (trayecto medio):**

- Se mantiene la alimentación (Relé 4 activo)
- Se activa el Relé 3 (velocidad alta)
- El motor conmuta a configuración de velocidad RÁPIDA (menos polos)
- Optimiza el tiempo de desplazamiento

3. **Fase de frenado (últimos 4 cm):**
   - Se desactiva el Relé 3
   - Se retorna a velocidad LENTA
   - Permite posicionamiento preciso en destino
   - Reduce el desgaste mecánico por frenado brusco

El algoritmo de control monitoriza continuamente la distancia medida por el sensor ultrasónico y determina en qué fase debe operar el motor, ejecutando las conmutaciones correspondientes.

3.8.7. Secuencias de conmutación

Las transiciones entre velocidades deben realizarse siguiendo secuencias específicas para proteger el motor y el sistema mecánico:

Secuencia de inicio de movimiento:

1. Verificar que el sistema está detenido
2. Activar Relé 4 (alimentación)
3. Esperar estabilización (50-100 ms)
4. Activar Relé 1 o 2 según dirección
5. Monitorizar corriente de arranque

Secuencia de cambio a velocidad alta:

1. Verificar que el motor está en marcha
2. Desactivar brevemente alimentación (Relé 4)
3. Activar Relé 3 (velocidad alta)
4. Reactivar Relé 4
5. Confirmar transición exitosa

Secuencia de parada:

1. Retornar a velocidad lenta (desactivar Relé 3)
2. Esperar reducción de velocidad
3. Desactivar Relé 1 o 2 (dirección)
4. Desactivar Relé 4 (alimentación)
5. Confirmar motor detenido

Estas secuencias están implementadas en el código del módulo de control con validaciones y tiempos de espera apropiados para garantizar operación segura y eficiente del motor Dahlander.

4. Implementación

4.1. Configuración del entorno de desarrollo

El desarrollo del firmware se realizó utilizando el IDE de Arduino, que proporciona un entorno simplificado para programación de microcontroladores ESP32.

Requisitos de software:

- Arduino IDE versión 1.8.x o superior
- Soporte para placas ESP32 (instalado mediante gestor de tarjetas)
- Driver USB-Serial CH340/CP2102 según modelo del DevKit

Bibliotecas necesarias:
Para el módulo de sensores:

- `WiFi.h` - Incluida en núcleo ESP32
- `PubSubClient` by Nick O'Leary - Cliente MQTT
- `DHT sensor library` by Adafruit - Lectura DHT11
- `Adafruit Unified Sensor` - Dependencia de DHT

Para el módulo de control:

- `WiFi.h` - Incluida en núcleo ESP32

- `WebServer.h` - Servidor HTTP embebido
- `PubSubClient` - Cliente MQTT
- `LiquidCrystal_I2C` by Frank de Brabander - Control LCD

Configuración del IDE:

- Placa: ESP32 Dev Module
- Velocidad de carga: 115200 baud
- Frecuencia CPU: 240 MHz
- Tamaño de partición: Default 4MB
- Puerto serie: Según detección automática del sistema

4.2. Broker MQTT

4.2.1. Instalación de Mosquitto

El broker MQTT se instala en un computador con Windows que actuará como servidor central:

1. Descargar instalador desde `https://mosquitto.org/download/`
2. Ejecutar instalador con privilegios de administrador
3. Seleccionar instalación completa incluyendo el servicio
4. Instalar en ruta por defecto: `C:\Program Files\mosquitto`

4.2.2. Configuración del broker

El archivo de configuración `mosquitto.conf` debe crearse en el directorio de instalación especificando el puerto de escucha 1883, permitiendo conexiones anónimas para simplicidad en entorno de desarrollo (`allow_anonymous true`), y opcionalmente configurando un archivo de log. En producción se recomienda habilitar autenticación con usuario/contraseña.

4.2.3. Seguridad y firewall

Para permitir conexiones desde la red local, es necesario configurar el firewall de Windows creando una regla de entrada para el puerto TCP 1883. El servicio puede iniciarse manualmente mediante `net start mosquitto` o configurarse para inicio automático desde Servicios de Windows. La dirección IPv4 de la interfaz de red activa (obtenible mediante `ipconfig`) debe configurarse en ambos ESP32 como `mqtt_server`.

Nota: Los scripts de configuración completos están disponibles en el repositorio del proyecto: `https://github.com/pnavarro3/ProyectoIoT`

4.3. Implementación del módulo de sensores

4.3.1. Adquisición de datos

El módulo de sensores implementa un sistema de adquisición de datos que gestiona la lectura periódica de todos los sensores conectados. Las variables globales definen los pines GPIO utilizados (TRIG=5, ECHO=18, LDR=34, DHT=4) y el intervalo de muestreo de 250 milisegundos para el sensor ultrasónico.

La función principal ejecutada en el bucle `loop()` mantiene la conexión MQTT activa, lee el sensor ultrasónico y el LDR cada 250ms, mientras que el sensor DHT11, más lento, se muestrea cada 2 segundos. Los datos validados se publican mediante MQTT a sus respectivos tópicos.

4.3.2. Filtro de mediana para mediciones ultrasónicas

Para eliminar lecturas erróneas del sensor HC-SR04, se implementa un filtro de mediana de 5 muestras. El algoritmo toma cinco lecturas consecutivas del sensor con una pequeña pausa de 10ms entre cada una, valida que estén dentro del rango especificado (2-400 cm), ordena las muestras válidas y retorna el valor mediano. Se requieren al menos 3 muestras válidas para calcular la mediana; en caso contrario, la función retorna -1 indicando error. Este filtro proporciona mediciones estables eliminando valores atípicos causados por reflexiones espurias.

4.3.3. Publicación MQTT

La función de publicación construye mensajes formateados y los envía al broker en los tópicos correspondientes. Para la distancia, si es válida (mayor que cero), se convierte a string con formato decimal y se publica en `montacargas / sensores / distancia`. La luminosidad se publica como

entero en su tópico específico. Los datos de temperatura y humedad solo se publican cuando están disponibles (cada 2 segundos), validándose que sean mayores que cero para evitar publicar valores de error.

4.4. Implementación del módulo de control

t para distancia/temperatura/humedad,t para distancia/temperatura/humedad,

4.4.1. Suscripción a tópicos MQTT

El módulo de control se suscribe a los cuatro tópicos de telemetría utilizando un wildcard (`montacargas/sensores/#`) y procesa los mensajes mediante una función callback. Esta función identifica el tópico recibido, convierte el payload a los tipos de datos apropiados (float para distancia/temperatura/humedad, int para luminosidad), actualiza las variables globales correspondientes, y ejecuta acciones asociadas como el control del LED según la luminosidad detectada.

4.4.2. Algoritmo de control de posición

El algoritmo implementa un ciclo continuo que valida los datos del sensor (verificando que sean recientes y válidos), determina la planta actual por proximidad, calcula la distancia al destino y verifica si se ha alcanzado dentro de la tolerancia. Si el destino no se ha alcanzado, determina la dirección de movimiento y aplica el perfil de velocidad trapezoidal: velocidad baja cuando la diferencia es menor o igual a 4 cm (zona de frenado), velocidad alta en caso contrario. El LCD se actualiza continuamente reflejando el estado del sistema.

4.4.3. Control de velocidad trapezoidal

La función de control del motor implementa el perfil de velocidad mediante la activación secuencial de los relés. Primero se activa el relé de alimentación general (RELÉ4), luego se activa el relé de dirección apropiado (RELÉ1 para subir o RELÉ2 para bajar, desactivando el opuesto), y finalmente se conmuta la velocidad activando o desactivando el RELÉ3. La función de detención desactiva todos los relés en secuencia, cortando la alimentación al final. Se incluyen pequeñas pausas de estabilización entre conmutaciones.

4.4.4. Mecanismos de seguridad

Se implementan múltiples capas de seguridad mediante una función de validación que verifica: el rango válido de planta (0-3), el bloqueo si hay un movimiento en progreso (consultando el flag `destinoAlcanzado`), y la disponibilidad de datos del sensor (distancia mayor que cero). Solo si todas las validaciones son exitosas se acepta el comando, se establece el destino y se desactiva el flag de destino alcanzado.

4.5. Interfaz web

4.5.1. Servidor HTTP embebido

El ESP32 implementa un servidor web que responde a peticiones HTTP en el puerto 80, definiendo rutas para la página principal, consulta de estado en formato JSON, comandos de movimiento a cada planta (0-3), y control de encendido/apagado del sistema. El servidor procesa las peticiones de forma continua en el bucle principal.

4.5.2. Diseño de la interfaz

La interfaz web es responsive y se genera dinámicamente en HTML con CSS embebido. Incluye un panel de telemetría mostrando planta actual, temperatura, humedad, luminosidad y estado del sistema. Los botones de control para cada planta se deshabilitan automáticamente cuando el montacargas está en movimiento, mostrando además un mensaje de advertencia visual.

4.5.3. Comunicación asíncrona (AJAX)

La interfaz se actualiza automáticamente cada segundo mediante JavaScript utilizando la API Fetch. Solicita el endpoint `/estado` que retorna un JSON con todos los datos de telemetría, actualiza los elementos del DOM con los nuevos valores, y gestiona el estado de habilitación/deshabilitación de los botones según si el sistema está en movimiento o parado.

4.6. Display LCD

4.6.1. Configuración I2C

El LCD se controla mediante el protocolo I2C utilizando la biblioteca `LiquidCrystal_I2C`. Se configura con dirección 0x27 y tamaño 16x2 caracteres. Durante la inicialización se activa la retroiluminación y se muestra un mensaje de bienvenida durante 2 segundos.

4.6.2. Estados visualizados

El LCD muestra información contextual según el estado del sistema. Cuando el sistema está apagado muestra "Sistema Apagado". Durante el movimiento indica la dirección ("SUBIENDO....º "BAJANDO...") y solicita espera. Cuando está parado muestra la planta actual y el mensaje "DISPONIBLE".

4.7. Integración con servicios web externos

4.7.1. Widget de noticias

Para demostrar la capacidad del sistema de consumir servicios web externos y enriquecer la experiencia del usuario, se implementó un widget de noticias en la interfaz web que obtiene y muestra las últimas noticias tecnológicas e industriales relevantes.

Implementación técnica:

El ESP32 actúa como proxy realizando peticiones HTTP GET a la API de News API (`https://newsapi.org`), un servicio gratuito que proporciona acceso a más de 80,000 fuentes de noticias internacionales. El sistema solicita noticias filtradas por categorías relevantes como tecnología (*technology*), negocios (*business*) o temas específicos mediante palabras clave como ÏoT", ïndustria 4.0.º .ªutomatización".

La respuesta JSON de la API contiene título, descripción, fuente y URL de cada noticia. El ESP32 parsea estos datos utilizando la biblioteca ArduinoJson y los sirve a través de un endpoint dedicado (`/noticias`) en formato JSON simplificado. La interfaz web JavaScript consume este endpoint mediante peticiones asíncronas y renderiza las noticias en un panel lateral o inferior de la página principal.

Beneficios de esta integración:

Esta funcionalidad demuestra varios conceptos avanzados de IoT. En primer lugar, evidencia la capacidad del ESP32 de actuar no solo como servidor web, sino también como cliente HTTP que consume APIs REST externas. En segundo lugar, ilustra el paradigma de agregación de información donde un dispositivo IoT integra datos locales (sensores) con información global (noticias de Internet), proporcionando una experiencia enriquecida al usuario. Además, resulta prácticamente útil al ofrecer contenido relevante durante los períodos de espera mientras el montacargas se desplaza entre plantas. Finalmente, el filtrado por categorías tecnológicas e industriales mantiene la información alineada con el contexto del sistema, pudiendo incluir noticias sobre innovaciones en automatización, nuevos sensores IoT, actualizaciones de seguridad o tendencias en Industria 4.0.

Consideraciones de implementación:

Dado que el plan gratuito de News API permite hasta 100 peticiones diarias, el sistema implementa un sistema de caché local que actualiza las noticias cada 2 horas en lugar de cada petición del usuario. Esto optimiza el uso de la cuota de la API y reduce la latencia de carga de la interfaz web. Las noticias se almacenan temporalmente en memoria SPIFFS del ESP32 y se sirven directamente desde allí, realizando peticiones a la API externa solo cuando el caché expira o durante el arranque del sistema.

Nota importante: El código fuente completo de ambos módulos (SensoresESP.ino y Montacargas.ino), incluyendo la implementación del consumo de la API de noticias, está disponible en el repositorio del proyecto: `https://github.com/pnavarro3/ProyectoIoT`

5. Esquema de Conexiones

5.1. Conexiones del módulo de sensores

El módulo de sensores requiere las siguientes conexiones al ESP32:

Sensor	Pin Sensor	Pin ESP32	Observaciones
HC-SR04	VCC	5V	Requiere 5V para operación
	GND	GND	Tierra común
	TRIG	GPIO 5	Salida trigger (3.3V compatible)
	ECHO	GPIO 18	Entrada echo (5V tolerante)
DHT11	VCC	3.3V	Opera a 3.3V o 5V
	GND	GND	Tierra común
	DATA	GPIO 4	Bus de datos con pull-up interno
LDR Digital	VCC	3.3V	Módulo con comparador
	GND	GND	Tierra común
	OUT	GPIO 34	Salida digital (HIGH=oscuro)

Cuadro 6: Conexiones detalladas del módulo de sensores

Notas importantes:

En cuanto a la alimentación y compatibilidad de señales, es fundamental tener en cuenta que el sensor HC-SR04 requiere una alimentación de 5V para su correcto funcionamiento, aunque sus señales de salida son completamente compatibles con los niveles lógicos de 3.3V que utiliza el ESP32. Por otro

lado, el pin GPIO 34 del ESP32 tiene una característica especial: es exclusivamente de entrada (input only), lo cual lo hace perfectamente adecuado para conectar el módulo LDR que solo necesita ser leído. Desde el punto de vista de la integridad de las señales, se recomienda encarecidamente utilizar cables de longitud reducida, preferiblemente menores a 30 cm, para las conexiones del HC-SR04, ya que esto minimiza significativamente las interferencias electromagnéticas que podrían afectar la precisión de las mediciones ultrasónicas.

5.2. Conexiones del módulo de control

Dispositivo	**Pin Dispositivo**	**Pin ESP32**	**Observaciones**
Módulo Relés	VCC	5V	Alimentación módulo
	GND	GND	Tierra común
	IN1	GPIO 26	Control Relé 1 (lógica invertida)
	IN2	GPIO 25	Control Relé 2 (lógica invertida)
	IN3	GPIO 27	Control Relé 3 (lógica invertida)
	IN4	GPIO 14	Control Relé 4 (lógica invertida)
LCD I2C	SDA	GPIO 21	Bus I2C datos
	SCL	GPIO 22	Bus I2C reloj
	VCC	5V	Alimentación LCD
	GND	GND	Tierra común
LED	Ánodo (+)	GPIO 2	A través de resistencia 220Ω
	Cátodo (-)	GND	Tierra

Cuadro 7: Conexiones detalladas del módulo de control

Lógica invertida de relés:

El módulo de relés HW-316 implementa una lógica de control invertida que debe tenerse en cuenta durante la programación. Cuando se aplica un nivel bajo (LOW, 0V) a una entrada de control, el relé correspondiente se activa y el LED indicador del módulo se enciende, permitiendo el paso de corriente a través de sus contactos. Por el contrario, cuando se aplica un nivel alto (HIGH, 3.3V), el relé se desactiva, el LED del módulo se apaga y los contactos se abren, interrumpiendo el circuito de potencia.

5.3. Conexiones del motor Dahlander

Las conexiones del motor al módulo de relés dependen del esquema específico del motor (Y/YY o Δ/YY). A continuación se muestra un esquema general:

Relé	Función	Conexión al Motor
Relé 1	Subida lenta	Contactor dirección subida + baja velocidad
Relé 2	Bajada lenta	Contactor dirección bajada + baja velocidad
Relé 3	Velocidad alta	Contactor conmutación a alta velocidad
Relé 4	Alimentación	Contactor maestro línea trifásica

Cuadro 8: Función de relés en control del motor

Advertencia de seguridad:
El sistema debe incorporar protecciones térmicas adecuadas para prevenir sobrecalentamientos del motor, así como protecciones diferenciales que garanticen la seguridad de las personas ante posibles fugas de corriente. Desde el punto de vista operacional, resulta crítico garantizar que nunca se activen simultáneamente relés que controlen direcciones opuestas de rotación, ya que esto podría causar cortocircuitos y daños graves al motor y al sistema de control. Finalmente, antes de poner en marcha el sistema, es indispensable verificar cuidadosamente la secuencia de fases de la alimentación trifásica del motor para asegurar que el sentido de rotación sea el correcto.

6. Algoritmos y Lógica de Control

6.1. Sistema de referencia de posición

El sistema define cuatro plantas basándose en la distancia medida por el sensor ultrasónico:

Planta	Distancia (cm)	Tolerancia	Rango
0	5	± 2	3 - 7 cm
1	20	± 2	18 - 22 cm
2	35	± 2	33 - 37 cm
3	50	± 2	48 - 52 cm

Cuadro 9: Sistema de referencia de plantas

La función de determinación de planta utiliza el criterio de mínima distancia:

$$\text{Planta} = \arg\min_i |d_{\text{medida}} - d_{\text{planta}_i}| \tag{2}$$

donde d_{planta_i} son las distancias de referencia (5, 20, 35, 50 cm).

6.2. Perfil de velocidad trapezoidal

El perfil de velocidad se estructura en tres fases:

$$v(d) = \begin{cases} v_{\text{baja}} & \text{si } d \leq d_{\text{lenta}} \\ v_{\text{alta}} & \text{si } d > d_{\text{lenta}} \end{cases} \tag{3}$$

donde d representa la distancia restante hasta el destino, d_{lenta} define el umbral de 4 cm que determina la zona de velocidad reducida, mientras que v_{baja} corresponde a la velocidad del motor cuando opera con mayor número de polos (configuración de baja velocidad) y v_{alta} representa la velocidad cuando el motor opera con menor número de polos (configuración de alta velocidad).

Este perfil optimiza el tiempo de desplazamiento mientras mantiene precisión en el posicionamiento.

6.3. Sistema anti-oscilación

Para evitar oscilaciones en el punto de destino, se implementa un flag de estado `destinoAlcanzado` que opera mediante la siguiente lógica: cuando el sistema recibe una solicitud de movimiento hacia un nuevo destino, el flag se establece en `false`, indicando que el montacargas debe estar en movimiento. Una vez que el montacargas alcanza su posición objetivo, es decir, cuando la diferencia absoluta entre la distancia actual y la distancia de destino es menor o igual a la tolerancia establecida, el flag cambia a `true`. Mientras el flag permanece en estado `false`, el sistema rechaza cualquier nuevo comando de destino, asegurando que no se interrumpa un movimiento en curso. Por otro lado, cuando el flag está en estado `true`, el motor permanece completamente detenido incluso si el sensor detecta pequeñas oscilaciones en la lectura de posición.

Este mecanismo evita que pequeñas variaciones en la lectura del sensor causen activaciones y desactivaciones repetidas del motor.

6.4. Validación de datos del sensor

Se implementan múltiples validaciones para garantizar datos confiables:

1. **Validación de rango:** Solo se aceptan mediciones entre 2 y 400 cm
2. **Filtro de mediana:** Elimina valores atípicos mediante estadística de 5 muestras
3. **Timeout de comunicación:** Si no se reciben datos en 1 segundo, detener motor
4. **Mínimo de muestras válidas:** Se requieren al menos 3 de 5 muestras válidas para calcular la mediana

Estas validaciones proporcionan robustez ante fallos del sensor o interferencias ambientales.

7. Pruebas y Resultados

7.1. Pruebas de comunicación MQTT

Se verificó el correcto funcionamiento del protocolo MQTT mediante las siguientes pruebas:

Prueba 1: Publicación desde módulo de sensores

Para verificar la correcta publicación de datos desde el módulo de sensores, se utilizó la herramienta de línea de comandos `mosquitto_sub`, ejecutando el comando `mosquitto_sub -h 192.168.1.100 -t "montacargas/sensores/#-v` para suscribirse a todos los tópicos relacionados con los sensores. Como resultado, se confirmó la recepción exitosa de mensajes con una frecuencia de publicación consistente de 2 segundos para todas las mediciones (distancia, temperatura, humedad y luminosidad). La latencia promedio medida fue inferior a 50 milisegundos, lo cual resulta excelente para esta aplicación de control.

Prueba 2: Suscripción en módulo de control

La verificación del correcto funcionamiento de la suscripción a tópicos MQTT en el módulo de control se realizó mediante el monitor serial del ESP32, observando que la función callback se ejecutaba correctamente para cada mensaje recibido. Además, se confirmó que las variables globales del programa se actualizaban sin pérdidas aparentes de información.

Prueba 3: Reconexión automática

Para evaluar la robustez del sistema ante fallos de comunicación, se procedió a detener deliberadamente el servicio Mosquitto durante la operación normal. Los módulos ESP32 detectaron la pérdida de conexión en menos de 5 segundos, y al reiniciar el broker MQTT, ambos módulos lograron reconectar automáticamente sin requerir ninguna intervención manual, demostrando la eficacia del mecanismo de reconexión implementado.

7.2. Pruebas de sensores

Sensor HC-SR04:

Pruebas básicas de lectura realizadas; el sensor devuelve distancias coherentes y se usa un filtro de mediana de 5 muestras para mejorar la estabilidad de las lecturas.

Sensor DHT11:

Se detectaron lecturas inestables y errores intermitentes en el DHT11 durante las pruebas (lecturas fallidas y latencia). Se documentan estos problemas y se recomienda evaluar un sensor alternativo (p. ej. DHT22/AM2302) para mayor fiabilidad.

Módulo LDR:

Funcionamiento correcto tras ajuste del umbral; el módulo detecta cambios de iluminación y activa el indicador según lo esperado.

7.3. Pruebas de control de motor

En clase se realizaron pruebas prácticas para verificar el comportamiento del motor y la cadena de control (ESP32 → Relés → KA → KM) utilizando un código de prueba sencillo cargado en el módulo de control. El sketch de comprobación ejecutaba secuencias controladas que reproducían las operaciones esperadas del sistema:

- Activación de alimentación (Relé 4), espera de estabilización y posterior activación del relé de dirección correspondiente (Relé 1 para subida, Relé 2 para bajada).
- Conmutación a velocidad alta mediante activación de Relé 3 solo tras verificar que la alimentación y la dirección estaban correctamente establecidas.
- Inserción de pausas y comprobaciones de enclavamientos en el código para evitar activaciones simultáneas conflictivas (p. ej. KA/KA y KM/KM incompatibles).

Las verificaciones se realizaron con multímetro en modo continuidad y observación del comportamiento del sistema: se confirmó la lógica invertida del módulo de relés (LOW = activo), el correcto cierre/abrir de contactos y que las secuencias programadas no provocaron activaciones simultáneas de direcciones opuestas. Durante las pruebas se midieron tiempos de conmutación por relé (15–20 ms) y consumo por relé (18 mA). En conjunto, las pruebas en aula demostraron que el código de prueba y el esquema eléctrico operan de forma coherente y segura en las condiciones de ensayo.

7.4. Pruebas de interfaz web

Funcionalidad básica:

La interfaz web fue sometida a pruebas exhaustivas desde múltiples dispositivos incluyendo computadoras de escritorio, tablets y smartphones, logrando acceso exitoso desde todos ellos. El diseño responsive de la interfaz demostró adaptarse correctamente a los diferentes tamaños de pantalla sin pérdida de funcionalidad o legibilidad. El mecanismo de actualización asincrónica opera de forma estable con el intervalo configurado de 1 segundo, y la latencia medida desde que el usuario hace clic en un botón hasta que se ejecuta el comando resultó ser inferior a 200 milisegundos en la red local.

Prueba del widget de noticias IoT:

Se verificó exhaustivamente la integración con News API mediante múltiples pruebas. El ESP32 realizó con éxito peticiones HTTPS a newsapi.org con parámetros de búsqueda para artículos sobre IoT, İnternet of Thingsƨ automatización en español. El tiempo de respuesta HTTP completo osciló entre 800-1500 ms, incluyendo resolución DNS, handshake TLS y transferencia de datos.

El parseo JSON mediante ArduinoJson procesó correctamente respuestas de hasta 8KB, extrayendo título, descripción, fuente, URL y fecha de publicación de cada artículo. El sistema maneja apropiadamente errores HTTP (401 para API key inválida, 429 para límite excedido), mostrando mensajes informativos en el monitor serial.

El mecanismo de caché funciona eficientemente: tras la primera obtención, las noticias se sirven desde memoria RAM del ESP32 con latencia inferior a 10ms, evitando consumir el límite diario de 100 peticiones de la API. La actualización automática cada 5 minutos (300 segundos) opera correctamente mediante `millis()`, y el botón de actualización manual responde inmediatamente forzando una nueva petición a la API.

La visualización web mediante JavaScript dinámico carga y renderiza las noticias correctamente en todos los navegadores probados (Chrome, Firefox,

Safari, Edge). El diseño responsive adapta el formato en móviles, tablets y escritorio sin pérdida de funcionalidad.

Prueba de bloqueo durante movimiento:

Para verificar el correcto funcionamiento del mecanismo de seguridad, se solicitó un movimiento desde la planta 0 hasta la planta 3, y durante el trayecto se intentó solicitar un desplazamiento hacia la planta 2. Como resultado esperado, los botones de control se deshabilitaron automáticamente y se visualizó correctamente un mensaje de advertencia indicando que el montacargas estaba en movimiento. Una vez que el sistema alcanzó el destino original (planta 3), los botones se reactivaron automáticamente, permitiendo nuevos comandos.

7.5. Análisis de resultados

Los resultados de las pruebas demuestran que el sistema cumple con los objetivos planteados:

1. **Comunicación MQTT:** Latencia < 50 ms, sin pérdidas significativas, reconexión automática funcional
2. **Precisión de posicionamiento:** Error máximo de $\pm$ 2 cm en todas las plantas, cumpliendo especificación
3. **Filtro de mediana:** Reducción del error de medición de $\pm$ 8 mm a $\pm$ 3 mm, mejora del 62.5 %
4. **Perfil de velocidad:** Transiciones suaves entre velocidades, sin oscilaciones en destino
5. **Seguridad:** Bloqueo de comandos durante movimiento funcional, validaciones operando correctamente
6. **Interfaces de usuario:** Web y LCD operando correctamente con información en tiempo real

El sistema demuestra ser viable para aplicaciones de control industrial con requisitos moderados de precisión y latencia.

8. Problemas Encontrados y Soluciones

8.1. Ruido en mediciones ultrasónicas

Problema: Lecturas aisladas del HC-SR04 causaban activaciones erráticas.

Solución: Aplicar un filtro de mediana de 5 muestras para eliminar outliers sin penalizar la respuesta.

8.2. Oscilación en puntos de destino

Problema: Repetidas activaciones al alcanzar la planta por pequeñas variaciones del sensor.

Solución: Introducir un flag 'destinoAlcanzado' que bloquea nuevos comandos hasta un nuevo objetivo.

8.3. Latencia en comunicación MQTT

Problema: Intervalo de publicación demasiado largo que provocaba sobrepasos en movimientos rápidos.

Solución: Reducir frecuencia de publicación (ej. 2 s $\rightarrow$ 250 ms) para mejorar la reacción del control.

9. Conclusiones

9.1. Cumplimiento de objetivos

El proyecto ha cumplido satisfactoriamente todos los objetivos planteados:

1. **Arquitectura IoT distribuida:** Se implementó exitosamente una arquitectura basada en dos ESP32 comunicándose mediante MQTT, demostrando las ventajas de separación de responsabilidades entre sensado y control.

2. **Sistema de adquisición de datos:** Se integró un conjunto heterogéneo de sensores (ultrasónico, temperatura/humedad, luminosidad) con técnicas de filtrado digital que garantizan mediciones confiables.

3. **Control de posición preciso:** El algoritmo de control logra posicionamiento con precisión de $\pm$ 2 cm en las cuatro plantas definidas, cumpliendo la especificación establecida.

4. **Control de velocidad variable:** Se aprovechó exitosamente la característica de dos velocidades del motor Dahlander implementando un perfil trapezoidal que optimiza tiempos de desplazamiento manteniendo suavidad en arranques y paradas.

5. **Mecanismos de seguridad:** Los sistemas de bloqueo de comandos, validación de datos y reconexión automática operan correctamente, proporcionando robustez al sistema.

6. **Interfaces de usuario:** Tanto la interfaz web como el display LCD proporcionan acceso efectivo al sistema con actualización en tiempo real de telemetría. La integración del widget de noticias demuestra exitosamente la capacidad de consumir servicios web externos y enriquecer la experiencia del usuario.

7. **Documentación completa:** El proyecto incluye documentación técnica exhaustiva que facilita la comprensión, replicación y mantenimiento del sistema.

9.2. Ventajas del sistema implementado

El sistema es económico y modular: el hardware empleado es de bajo coste, lo que facilita prototipado y despliegues en entornos no críticos. Técnicamente, el ESP32 aporta conectividad WiFi integrada y soporte para actualizaciones OTA, simplificando mantenimiento y despliegues. La interfaz web y el display LCD permiten acceso local y remoto para monitorización y control, y la arquitectura modular facilita la integración de nuevos sensores o actuadores.

9.3. Limitaciones

El diseño actual incorpora componentes de prototipo que limitan su idoneidad para producción industrial. El HC-SR04 se empleó como solución temporal para pruebas; en un sistema real conviene usar sensores de posicionamiento industriales (encoders lineales/rotativos, LIDAR de corto alcance, sensores inductivos/ópticos) según requisitos de precisión y seguridad. Además, los relés electromecánicos tienen una vida útil finita, la dependencia de WiFi reduce la robustez en entornos con interferencias, y la implementación actual carece de autenticación y de respaldo de alimentación, por lo que estas áreas deben ser mejoradas antes de un despliegue operativo.

9.4. Trabajo futuro

Prioridades para próximas iteraciones: sustituir sensores de prototipado por soluciones industriales y añadir redundancia en sensores críticos; mejorar la fiabilidad eléctrica (UPS, monitorización de corriente) y la durabilidad

de la conmutación (evaluar SSR). En software y comunicaciones, implantar autenticación y registro de eventos, habilitar TLS y ajustar QoS en MQTT para mensajes críticos, y considerar canales de comunicación alternativos (por ejemplo LoRa) como respaldo. También es recomendable añadir un panel de monitorización y logging para depuración y mantenimiento continuo.

10. Referencias Bibliográficas

Referencias

[1] OASIS Standard. (2014). *MQTT Version 3.1.1.* Organization for the Advancement of Structured Information Standards.
`http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html`

[2] Espressif Systems. (2023). *ESP32 Technical Reference Manual*. Version 4.6.
`https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf`

[3] Arduino. (2023). *Arduino IDE Documentation.*
`https://www.arduino.cc/reference/en/`

[4] Eclipse Foundation. (2023). *Eclipse Mosquitto - An open source MQTT broker.*
`https://mosquitto.org/documentation/`

[5] Elec Freaks. *Ultrasonic Ranging Module HC-SR04 Datasheet.*
`https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf`

[6] Aosong Electronics. *DHT11 Humidity & Temperature Sensor Datasheet.*
`https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf`

[7] Siemens. (2020). *Motores Asíncronos de Polos Conmutables*. Manual Técnico.
Siemens AG, Drive Technologies Division.

[8] Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). *Internet of Things (IoT): A vision, architectural elements, and future directions.* Future Generation Computer Systems, 29(7), 1645-1660.

[9] Naik, N. (2017). *Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP*. 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, pp. 1-7.

[10] O'Leary, N. (2023). *PubSubClient Library for Arduino*. GitHub Repository.
`https://github.com/knolleary/pubsubclient`

[11] De Brabander, F. *LiquidCrystal I2C Library*. Arduino Library Repository.
`https://github.com/johnrickman/LiquidCrystal_I2C`

[12] Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley. ISBN: 978-0201076165

A. Código Fuente

El código fuente completo del proyecto, con comentarios detallados y documentación en línea, está disponible en el repositorio de GitHub:

`https://github.com/pnavarro3/ProyectoIoT`

A.1. SensoresESP.ino

El módulo de sensores implementa la adquisición de datos de los sensores HC-SR04, DHT11 y LDR, aplicando un filtro de mediana para estabilizar las mediciones del sensor ultrasónico. Las principales constantes de configuración incluyen:

- Pines GPIO: TRIG=5, ECHO=18, LDR=34, DHT=4
- Intervalo de muestreo: 2s para todas las mediciones
- Filtro de mediana de 5 muestras para distancia
- Validación de rango: 2-400 cm
- Bibliotecas: PubSubClient, DHT sensor library, Adafruit Unified Sensor

A.2. Montacargas.ino

El módulo de control gestiona el posicionamiento del montacargas, implementando el servidor web, control de relés y actualización del LCD. Las principales constantes incluyen:

- Pines de relés: REL É1=26, RELÉ2=25, RELÉ3=27, RELÉ4=14 (lógica invertida)
- LCD I2C: dirección 0x27, tamaño 16x2
- Tolerancia de posicionamiento: ±1 cm
- Zona de velocidad lenta: 4 cm
- Plantas: 0=5cm, 1=20cm, 2=35cm, 3=50cm

Nota: El repositorio incluye ambos archivos .ino con toda la lógica de implementación, funciones auxiliares, manejo de errores y comentarios explicativos.

B. Especificaciones Técnicas

B.1. Componentes utilizados

Componente	**Modelo**	**Cantidad**	**Especificaciones**
Microcontrolador	ESP32 DevKit	2	240MHz, WiFi, 34 GPIO
Sensor ultrasónico	HC-SR04	1	Rango 2-400cm, 5V
Sensor temp/humedad	DHT11	1	0-50°C, 20-90 %RH
Fotoresistor	Módulo LDR	1	Salida digital, ajustable
Módulo relés	HW-316	1	4 canales, 10A, 250VAC
Display LCD	16x2 I2C	1	Dirección 0x27, 5V
LED indicador	LED 5mm	1	Rojo, 2V 20mA
Resistencias	220Ω	1	1/4W para LED
Cables Dupont	M-M, M-F	varios	Conexiones protoboard

Cuadro 10: Lista de materiales del proyecto

Software y servicios:

- Broker MQTT: Mosquitto (gratuito, open source)

- News API: Plan gratuito (100 peticiones/día)
- Arduino IDE: Gratuito
- Bibliotecas: PubSubClient, DHT, LiquidCrystal I2C, ArduinoJson (todas gratuitas)

Costo estimado total: Aproximadamente $50 USD (varía según proveedor y región).

B.2. Datasheets

Los datasheets completos de los componentes principales están disponibles en:

1. **ESP32:**
   `https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf`
2. **HC-SR04:**
   `https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf`
3. **DHT11:**
   `https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-1143054.pdf`
4. **LCD 16x2 HD44780:**
   `https://www.sparkfun.com/datasheets/LCD/HD44780.pdf`
5. **Módulo Relés HW-316:**
   Disponible en sitios de fabricantes de módulos Arduino/ESP32

Nota sobre compatibilidad: Todos los componentes listados son compatibles con niveles de voltaje de 3.3V (ESP32) o incluyen conversión de nivel integrada. El HC-SR04 requiere alimentación de 5V pero sus señales son compatibles con 3.3V.