



UNIVERSIDAD DE GRANADA

Trabajo de Fin de Grado: Domotización de Maqueta

Miguel López Martínez

Grado en Ingeniería Electrónica Industrial

18 de mayo de 2025

Índice

1. Introducción	4
2. Instalación de Home Assistant en Raspberry Pi	4
2.1. Requisitos Previos	4
2.2. Descarga e Instalación de Home Assistant OS	4
2.3. Primer Arranque y Acceso Inicial	5
2.4. Configuración Inicial del Sistema	5
2.5. Ventajas del Home Assistant OS	6
2.6. Resumen	6
3. Configuración de Red y Acceso Remoto en Home Assistant	6
3.1. Asignación de IP Fija en la Red Local	7
3.2. Acceso Remoto Seguro con DuckDNS	8
3.3. Apertura de Puertos en el Router	9
3.4. Consideraciones sobre la IP Pública y DIGI Plus	10
3.5. Resumen	10
4. Configuración del Broker MQTT Mosquitto en Home Assistant	10
4.1. Instalación del Complemento Mosquitto	10
4.2. Creación de Usuario MQTT	11
4.3. Configuración del Complemento	11
4.4. Puertos Configurados	12
4.5. Integración con Home Assistant	12
4.6. Notas Adicionales	12
5. Modularización de la Configuración YAML y Automatización con MQTT	12
5.1. Nueva Estructura de <code>configuration.yaml</code>	12
5.2. Automatización: API Meteorológica para Control Lumínico	13
5.3. Estructura del Archivo <code>mqtt.yaml</code>	13
5.3.1. Luces RGB	13
5.3.2. Sensores y Actuadores	14
5.4. Justificación Técnica	14
5.4.1. Código de Configuración MQTT	14
5.5. Conclusión	16
6. Configuración Código ESP-32	16
6.1. Inclusión de Librerías	16
6.2. Credenciales WiFi	17
6.3. Configuración del Cliente MQTT	17
6.4. Configuración de LEDs	17
6.5. Pines de Entrada y Sensores	17
6.6. Pines del Motor (Puente H)	18
6.7. Tópicos MQTT Definidos	18
6.8. Variables Globales	18
6.9. Enumeración <code>EstadoCaja</code>	19
6.10. Prototipos de Funciones	19



6.11. Tópicos MQTT Utilizados	19
6.12. Función <code>setup()</code>	20
6.13. Función <code>configurarPines()</code>	21
6.14. Función <code>conectarWiFi()</code>	22
6.15. Función <code>configurarMQTT()</code>	22
6.16. Función <code>configurarMCPWM()</code>	24
6.17. Función <code>loop()</code>	26
6.18. Función <code>medirLux()</code>	27
6.19. Función <code>detectarMovimiento()</code>	28
6.20. Control de Caja Motorizada: <code>controlarCaja()</code> , <code>moverCaja()</code> y <code>detenerMotorCaja()</code>	29
6.21. Función <code>callback()</code>	32
6.22. Función <code>procesarMensajeJSON()</code>	34
6.23. Función <code>actualizarLed()</code>	35
6.24. Función <code>actualizarLed9()</code>	36
6.25. Función <code>apagarTodasLasLuces()</code>	37
6.26. Función <code>reconnect()</code>	37
6.27. Código de la ESP32 (Firmware en C++)	39
7. Interfaz de Usuario en Home Assistant (Lovelace)	48
7.1. Configuración de Tarjetas de Entidades	48
7.1.1. Luces RGB (LEDs)	48
7.1.2. Control de la Caja Electrónica	49
7.1.3. Velocidad del Segundo Motor	49
7.1.4. Sensores Binarios	49
7.1.5. Sensores Analógicos	49
7.1.6. Automatizaciones	49
7.2. Resumen de Diseño	50
8. Hardware	50
8.1. Raspberry Pi 3	50
8.2. ESP32	51
8.3. Módulo puente H L298N	51
8.4. Sensor de luz TEMT6000	52
8.5. Sensor magnético KY-025	53
8.6. Elección del motor DC con piñón-cremallera	53
8.7. LEDs NeoPixel	54
9. Presupuesto	55
10. Bibliografía Técnica de Componentes	56
11. Bibliografía de Funcionalidades de Home Assistant	57

1. Introducción

En la actualidad, la automatización del entorno doméstico, también conocida como domótica, está adquiriendo un protagonismo creciente debido al auge de las tecnologías IoT (Internet of Things) y la necesidad de sistemas eficientes, seguros y sostenibles. La posibilidad de controlar y monitorizar dispositivos del hogar en tiempo real, desde cualquier lugar, representa un avance significativo tanto en comodidad como en ahorro energético.

Este Trabajo de Fin de Grado tiene como objetivo el diseño e implementación de un sistema domótico funcional mediante la domotización de una maqueta, utilizando como plataforma central **Home Assistant**, un software de código abierto ampliamente adoptado en el ámbito de la automatización residencial. Para ello, se emplea una **Raspberry Pi 3** como unidad principal de procesamiento, por su equilibrio entre rendimiento, coste y eficiencia energética.

A lo largo del proyecto se detallan todas las fases del desarrollo, desde la instalación y configuración del sistema hasta la integración de sensores, actuadores y el diseño de una interfaz de usuario accesible. Asimismo, se analizan los beneficios y limitaciones del sistema implementado, valorando su viabilidad en aplicaciones reales.

Con este trabajo se busca no solo aplicar los conocimientos adquiridos durante la titulación en *Ingeniería Electrónica Industrial*, sino también explorar soluciones tecnológicas que puedan ser trasladables al ámbito profesional, educativo o doméstico.

2. Instalación de Home Assistant en Raspberry Pi

Para la implementación del centro de control domótico, se utilizó una **Raspberry Pi 3** como dispositivo principal por su bajo consumo, buen rendimiento y compatibilidad con Home Assistant. La instalación se realizó siguiendo las instrucciones oficiales del proyecto en <https://www.home-assistant.io/installation/raspberrypi>.

2.1. Requisitos Previos

Antes de iniciar el proceso, se necesitaban los siguientes elementos:

- Raspberry Pi 3
- Fuente de alimentación oficial
- Tarjeta microSD (32 GB, clase 10)
- Conexión a Internet (preferiblemente por cable Ethernet)
- Ordenador con lector de tarjetas SD
- Software Raspberry Pi Imager

2.2. Descarga e Instalación de Home Assistant OS

Home Assistant se ejecuta sobre su propio sistema operativo llamado **Home Assistant OS**, especialmente optimizado para dispositivos como la Raspberry Pi.

1. Se accedió a la web oficial: <https://www.home-assistant.io/installation/raspberrypi>

2. Se descargó la imagen correspondiente a la Raspberry Pi utilizada, en formato `.img`.
3. Con el programa `Raspberry Pi Imager` se grabó la imagen en la tarjeta microSD.

2.3. Primer Arranque y Acceso Inicial

1. Se insertó la tarjeta microSD en la Raspberry Pi.
2. Se conectó la Raspberry Pi a la red local mediante cable Ethernet.
3. Se encendió el dispositivo y se esperaron aproximadamente 20 minutos para que se completara la instalación inicial.
4. Desde un navegador web, se accedió a Home Assistant a través de la dirección:

`http://homeassistant.local:8123`

En caso de fallo con la dirección local, también se podía usar la IP local asignada al dispositivo (por ejemplo, `http://192.168.1.140:8123`).

2.4. Configuración Inicial del Sistema

Una vez iniciado Home Assistant:

- Se creó un usuario administrador.
- Se estableció la ubicación geográfica del dispositivo.
- Se configuraron los sistemas de detección automática de dispositivos y servicios.

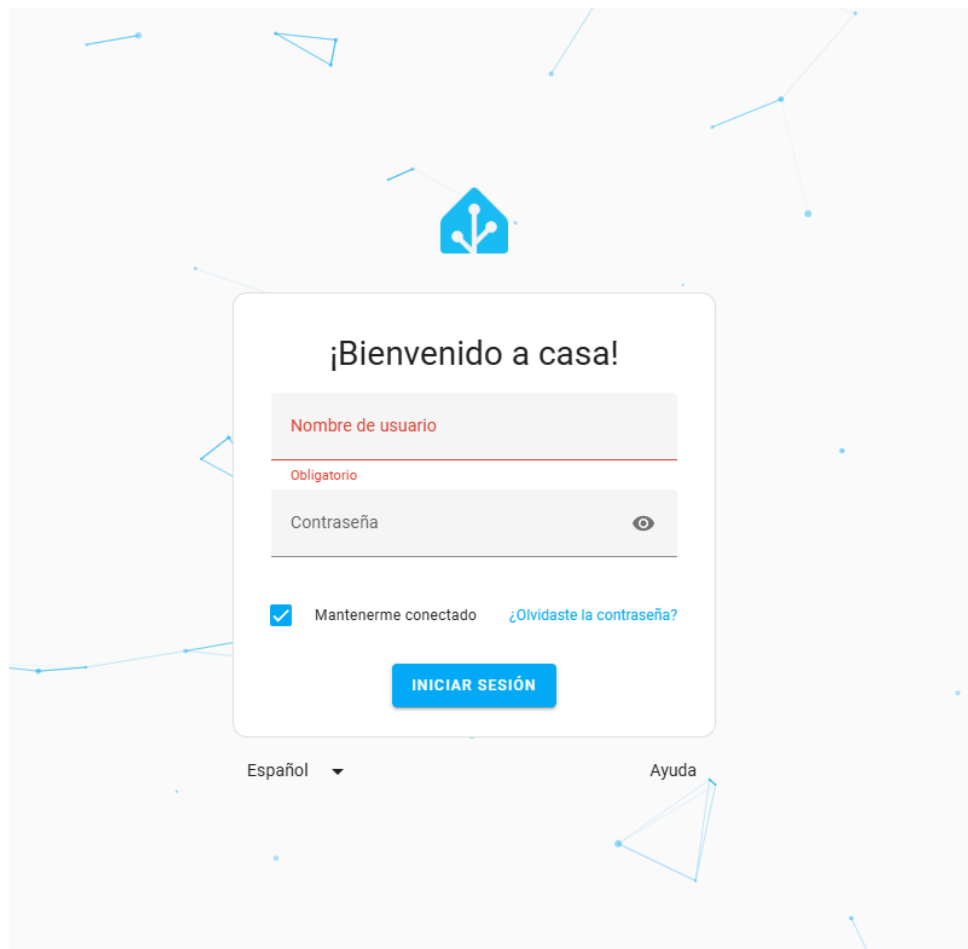


Figura 1: Pantalla de configuración inicial de Home Assistant tras el primer arranque.

2.5. Ventajas del Home Assistant OS

Se eligió instalar la versión completa de Home Assistant OS (en lugar de Home Assistant Core sobre Raspbian), ya que incluye:

- Sistema operativo ligero y optimizado
- Actualizaciones automáticas y seguras
- Acceso a **Supervisor**, que permite gestionar complementos (Add-ons)
- Mayor compatibilidad con integraciones como MQTT, DuckDNS, entre otros

2.6. Resumen

La instalación de Home Assistant en la Raspberry Pi permitió transformar este pequeño ordenador en un potente centro de control domótico. Su fiabilidad, consumo reducido y versatilidad lo hacen ideal para proyectos de automatización del hogar.

3. Configuración de Red y Acceso Remoto en Home Assistant

Una correcta configuración de red es esencial para garantizar el acceso seguro, estable y remoto al sistema domótico basado en Home Assistant. Esta sección detalla cómo se

estableció una IP fija local, se configuró un dominio dinámico con DuckDNS, se abrieron puertos en el router y se resolvieron limitaciones con la IP pública mediante el proveedor de servicios DIGI.

3.1. Asignación de IP Fija en la Red Local

Para evitar problemas de conectividad con dispositivos y servicios dependientes de una dirección IP constante, se configuró Home Assistant con una IP estática:

- **Dirección IP:** 192.168.1.140
- **Máscara de red:** 255.255.255.0
- **Puerta de enlace:** 192.168.1.1
- **DNS primario:** 100.90.1.1
- **DNS secundario:** 100.100.1.1

Configura las interfaces de red

ENU1U1**WLAN0**

IPv4

☐ Automático

☒ Estática

☐ Deshabilitado

Dirección IP

192.168.1.140

Máscara de red

255.255.255.0

+ AÑADIR DIRECCIÓN

Dirección de la puerta de enlace

192.168.1.1

Servidor DNS

100.90.1.1

Servidor DNS

100.100.1.1

+ AÑADIR SERVIDOR DNS

IPv6

RESTABLECER CONFIGURACIÓN

GUARDAR

Figura 2: Configuración de IP estática en Home Assistant.

3.2. Acceso Remoto Seguro con DuckDNS

Para permitir el acceso externo a Home Assistant, se utilizó el complemento oficial **DuckDNS**, que permite vincular una IP dinámica a un subdominio personalizado. El procedimiento fue el siguiente:

1. Registro en <https://duckdns.org> con una cuenta de Google.
2. Creación del subdominio: `tfg-miguel-lopez.duckdns.org`.
3. Instalación del complemento DuckDNS desde **Ajustes** → **Complementos** → **DuckDNS**.
4. Configuración del archivo del complemento:


```

1  lets_encrypt:
2    accept_terms: true
3    algo: secp384r1
4    certfile: fullchain.pem
5    keyfile: privkey.pem
6    token: *****
7  domains:
8    - tfg-miguel-lopez.duckdns.org

```

5. Adición de certificados SSL en configuration.yaml:

```

1  http:
2    ssl_certificate: /ssl/fullchain.pem
3    ssl_key: /ssl/privkey.pem

```

Código 1: Archivo mqtt.yaml con configuración completa del sistema

Figura 3: Configuración del DNS dinámico DuckDNS en el router Zyxel

3.3. Apertura de Puertos en el Router

Para habilitar el acceso remoto, se abrió el puerto 8123 (HTTP) desde el NAT del router. Aunque se probaron configuraciones con HTTPS, el protocolo HTTP ofreció mayor estabilidad en este caso práctico.

- **Puerto externo:** 8123
- **Puerto interno:** 8123 (por defecto en Home Assistant)
- **Protocolo:** TCP

#	Estado	Nombre del servicio	IP de origen	Interfaz WAN	Dirección IP del servidor	Puerto inicial	Puerto final	Puerto de inicio de traducción	Puerto final de traducción	Protocolo	Modificar
1	🟡	duckdns	Predeterminado		192.168.1.140	8123	8123	8123	8123	TCP	✎ 🗑
2	🟡	puerto_443	Predeterminado		192.168.1.140	443	443	443	443	TCP	✎ 🗑
3	🟡	puerto-80	Predeterminado		192.168.1.140	80	80	80	80	TCP	✎ 🗑

Figura 4: Redirección de puertos configurada en el router

3.4. Consideraciones sobre la IP Pública y DIGI Plus

Una de las principales dificultades fue la ausencia de una IP pública dedicada. Las compañías de bajo coste, como DIGI, suelen asignar IPs compartidas (CG-NAT), lo que impide la apertura de puertos de forma directa. Esta limitación se resolvió contratando el servicio **DIGI Plus**, que proporciona una IP pública real, indispensable para:

- Abrir puertos para acceso remoto
- Hacer funcionar DuckDNS correctamente
- Evitar restricciones de conectividad en el entorno de red

Sin una IP pública, múltiples usuarios de una misma zona comparten la misma IP externa, lo que impide la llegada directa de conexiones entrantes desde Internet a Home Assistant.

3.5. Resumen

La configuración de red incluyó los siguientes pasos clave:

1. Asignación de IP fija local para estabilidad en la red.
2. Configuración de acceso remoto mediante DuckDNS y certificados SSL.
3. Apertura de puertos en el router para redirigir tráfico externo a Home Assistant.
4. Contratación de IP pública dedicada con DIGI Plus para garantizar conectividad total.

Esta infraestructura de red permite acceder de forma remota, segura y confiable al sistema domótico desde cualquier parte del mundo.

4. Configuración del Broker MQTT Mosquitto en Home Assistant

Para habilitar la comunicación entre los dispositivos del sistema domótico mediante el protocolo MQTT, se utilizó el complemento oficial **Mosquitto Broker** en Home Assistant. A continuación, se detalla el procedimiento de instalación y configuración realizado.

4.1. Instalación del Complemento Mosquitto

La instalación se llevó a cabo desde la interfaz de Home Assistant siguiendo los siguientes pasos:

1. Navegar a Configuración → Complementos → Tienda de complementos.
2. Buscar el complemento *Mosquitto broker* y hacer clic en él.
3. Presionar el botón INSTALAR.

4.2. Creación de Usuario MQTT

Para permitir la autenticación de clientes, se creó un usuario MQTT desde Home Assistant:

- **Nombre de usuario:** miguelmqtt
- **Contraseña:** miguelmqtt

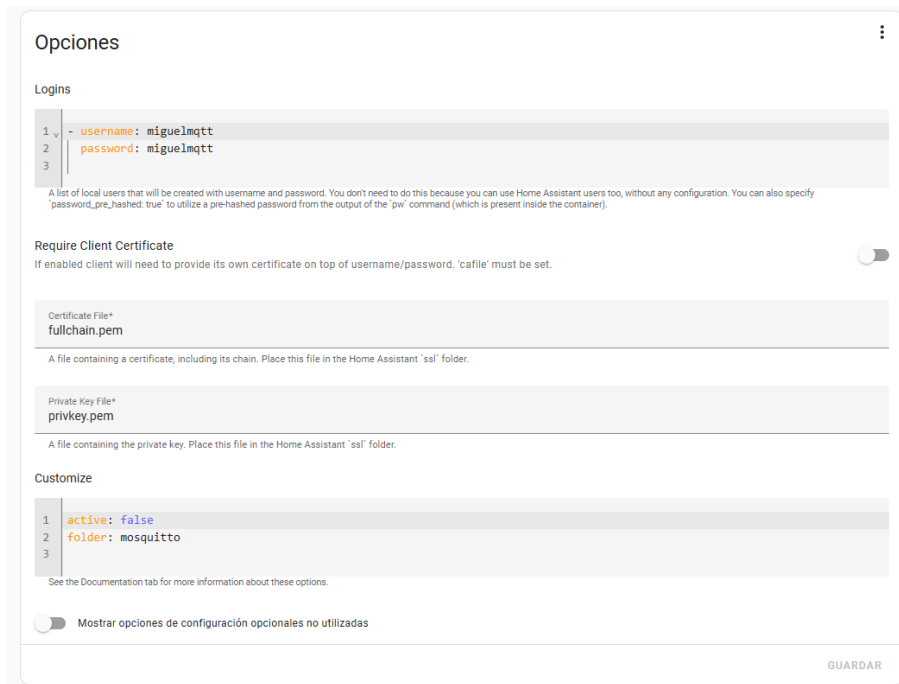
Este usuario fue creado a través del menú **Configuración** → **Personas** → **Usuarios** y no directamente desde la configuración del complemento.

4.3. Configuración del Complemento

La configuración YAML utilizada en el complemento fue la siguiente:

```
1 logins:
2   - username: miguelmqtt
3     password: miguelmqtt
4 customize:
5   active: false
6   folder: mosquitto
7 certfile: fullchain.pem
8 keyfile: privkey.pem
9 require_certificate: false
```

Cabe destacar que esta configuración también puede realizarse mediante la interfaz gráfica de Home Assistant, como se muestra en la siguiente imagen:



The image shows the 'Opciones' (Options) configuration page for the MQTT integration in Home Assistant. The page is divided into several sections:

- Logins:** A list of local users. The first entry is expanded, showing 'username: miguelmqtt' and 'password: miguelmqtt'.
- Require Client Certificate:** A toggle switch is currently turned off. Below it, a note states: 'If enabled client will need to provide its own certificate on top of username/password. 'cafile' must be set.'
- Certificate File*:** A text input field containing 'fullchain.pem'. Below it, a note states: 'A file containing a certificate, including its chain. Place this file in the Home Assistant 'ssl' folder.'
- Private Key File*:** A text input field containing 'privkey.pem'. Below it, a note states: 'A file containing the private key. Place this file in the Home Assistant 'ssl' folder.'
- Customize:** A list of options. The first entry is expanded, showing 'active: false' and 'folder: mosquitto'.

At the bottom, there is a toggle switch labeled 'Mostrar opciones de configuración opcionales no utilizadas' (Show optional configuration options not used) and a 'GUARDAR' (Save) button.

Figura 5: Configuración del complemento MQTT a través de la interfaz gráfica.

4.4. Puertos Configurados

Se configuraron los siguientes puertos en el complemento, los cuales fueron expuestos en el host para permitir distintas formas de conexión:

- **1883/tcp** - MQTT estándar sin cifrado
- **1884/tcp** - MQTT sobre WebSocket sin cifrado
- **8883/tcp** - MQTT cifrado con SSL
- **8884/tcp** - MQTT sobre WebSocket cifrado con SSL

4.5. Integración con Home Assistant

Una vez iniciado el complemento, se detectó automáticamente la integración MQTT mediante:

- Configuración → Dispositivos y servicios → Integraciones

Se habilitó la detección automática y se completó la integración con un solo clic, permitiendo que Home Assistant pudiera tanto publicar como suscribirse a tópicos MQTT.

4.6. Notas Adicionales

- No se permite el acceso anónimo, por lo que es obligatorio el uso de credenciales.
- No se activaron las listas de control de acceso (ACL), aunque es posible configurarlas posteriormente si se desea restringir el acceso por usuario o tópico.

5. Modularización de la Configuración YAML y Automatización con MQTT

Durante el desarrollo del sistema domótico, se decidió realizar una reestructuración completa del archivo `configuration.yaml` para mejorar la organización, legibilidad y mantenimiento del código. En lugar de mantener toda la configuración centralizada, se optó por dividirla en módulos temáticos mediante directivas `!include`. Esta práctica no solo facilita la depuración de errores, sino que también reduce la probabilidad de fallos de compilación ante configuraciones extensas o desordenadas.

5.1. Nueva Estructura de `configuration.yaml`

```
1 # Carga por defecto de las integraciones. No eliminar.
2 default_config:
3
4 # Temas personalizados
5 frontend:
6   themes: !include_dir_merge_named themes
7
8 # Inclusión de módulos externos
9 automation: !include automations.yaml
```

```
10 script: !include scripts.yaml
11 scene: !include scenes.yaml
12 mqtt: !include mqtt.yaml
```

5.2. Automatización: API Meteorológica para Control Lumínico

Se integró una automatización basada en la entidad `sun.sun` de Home Assistant, que utiliza datos astronómicos (amanecer, anochecer) para determinar si es de día o de noche. Esta automatización publica dicha información mediante MQTT cada 5 minutos, permitiendo a un ESP controlar las luces de la maqueta.

```
1
2
3 - alias: "Enviar estado de día o noche por MQTT cada 5 minutos"
4   trigger:
5     - platform: state
6       entity_id: sun.sun
7     - platform: time_pattern
8       minutes: "/5"
9   action:
10    - service: mqtt.publish
11      data:
12        topic: "casa/luz/dia_noche"
13        payload: >
14          {% if is_state('sun.sun', 'above_horizon') %}
15            "es de dia"
16          {% elif is_state('sun.sun', 'below_horizon') %}
17            "es de noche"
18          {% else %}
19            "desconocido"
20          {% endif %}
21        retain: true
```

5.3. Estructura del Archivo mqtt.yaml

En el archivo `mqtt.yaml` se integraron múltiples dispositivos usando distintos tópicos. Inicialmente se valoró usar un único tópico para controlar todas las luces, pero se descartó por la complejidad técnica y por la escasa escalabilidad. Dividir los tópicos por LED facilita el control individual y evita conflictos cuando se manejan múltiples dispositivos simultáneamente.

5.3.1. Luces RGB

`home/led_X/set` es el tópico para cada uno de los nueve LEDs controlados.

- Control por RGB
- Brillo ajustable
- Soporte para múltiples modos de color

5.3.2. Sensores y Actuadores

- **Sensor de luz ambiente:** mide lux y originalmente servía para controlar luces (reemplazado por la API solar).
- **Sensor ultrasónico:** implementado de forma experimental para detectar proximidad; fue una alternativa al sensor de movimiento.
- **Sensor de movimiento:** usado finalmente para activar o desactivar luces por presencia.
- **Motores:** se implementaron dos controles de velocidad PWM (de 0 a 100) para regular:
 1. La velocidad de un motor principal de la maqueta
 2. Un motor encargado de abrir/cerrar una caja que aloja la electrónica
- **Interruptor (switch):** controla si la caja debe abrirse o cerrarse.

5.4. Justificación Técnica

Se eligió un diseño modular y tópico por dispositivo por las siguientes razones:

- El ESP32 puede manejar múltiples tópicos sin dificultad.
- Un único tópico para múltiples dispositivos complicaría la lógica de control y no aportaría ventajas claras.
- La modularización mejora la mantenibilidad y la claridad del proyecto.
- Dividir la configuración en múltiples archivos evita errores frecuentes en proyectos complejos con muchos nodos y automatizaciones.

5.4.1. Código de Configuración MQTT

A continuación, se presenta la configuración completa del archivo `mqtt.yaml`, donde se integran luces RGB, sensores, actuadores, y controles PWM mediante el protocolo MQTT:

```
1 light:
2   - name: "LED 1"
3     state_topic: "home/led_1/set"
4     command_topic: "home/led_1/set"
5     qos: 0
6     schema: json
7     brightness: true
8     supported_color_modes:
9       - "rgb"
10  - name: "LED 2"
11    state_topic: "home/led_2/set"
12    command_topic: "home/led_2/set"
13    qos: 0
```

```
15     schema: json
16     brightness: true
17     supported_color_modes:
18       - "rgb"
19     ...
20   - name: "LED 9"
21     state_topic: "home/led_9/set"
22     command_topic: "home/led_9/set"
23     qos: 0
24     schema: json
25     brightness: true
26     supported_color_modes:
27       - "rgb"
28
29 sensor:
30   - name: "Luz Ambiente"
31     state_topic: "casa/luz/lux"
32     unit_of_measurement: "lx"
33     value_template: "{{ value | round(2) }}"
34     device_class: "illuminance"
35
36   - name: "Distancia Ultrasonidos"
37     state_topic: "home/ultrasonic/distance"
38     unit_of_measurement: "cm"
39     value_template: "{{ value.split(' ')[0] }}"
40
41 binary_sensor:
42   - name: "Cercanía Ultrasonidos"
43     state_topic: "home/ultrasonic/proximity"
44     payload_on: "true"
45     payload_off: "false"
46     device_class: presence
47
48   - name: "Nivel de Luz Ambiente"
49     state_topic: "home/ambient/light"
50     payload_on: "true"
51     payload_off: "false"
52     device_class: light
53
54   - name: "Sensor de Movimiento"
55     state_topic: "home/motion_sensor"
56     device_class: motion
57     payload_on: "1"
58     payload_off: "0"
59
60 number:
61   - name: "Velocidad caja"
62     state_topic: "home/motor/speed"
63     command_topic: "home/motor/speed/set"
64     min: 0
65     max: 100
```

```
66     step: 1
67     unit_of_measurement: "%"
68     mode: slider
69
70 - name: "Velocidad Motor"
71   state_topic: "home/motor2/speed"
72   command_topic: "home/motor2/speed/set"
73   min: 0
74   max: 100
75   step: 1
76   unit_of_measurement: "%"
77   mode: slider
78
79 switch:
80 - name: "Caja - Abrir"
81   command_topic: "home/caja/estado/set"
82   payload_on: "abrir"
83   payload_off: "cerrar"
84
85 - name: "Luz nocturna"
86   command_topic: "home/luz/nocturna"
87   payload_on: "encender"
88   payload_off: "apagar"
89
```

5.5. Conclusión

Gracias a esta arquitectura modular y el uso eficiente del protocolo MQTT, el sistema permite una comunicación fluida y escalable entre Home Assistant y los microcontroladores. Además, la incorporación de automatizaciones basadas en condiciones ambientales (día/noche, presencia, luz ambiente) aporta inteligencia al sistema, optimizando el consumo energético y mejorando la experiencia de uso.

6. Configuración Código ESP-32

A continuación se detallan las librerías incluidas, así como las variables y constantes utilizadas en la configuración del entorno WiFi, MQTT, sensores, actuadores y tópicos de comunicación.

6.1. Inclusión de Librerías

- <WiFi.h>: Permite la conexión del ESP32 a redes WiFi.
- <PubSubClient.h>: Cliente MQTT para gestionar la comunicación con un broker.
- <Adafruit_NeoPixel.h>: Manejo de tiras LED RGB tipo NeoPixel.
- <ArduinoJson.h>: Interpretación de mensajes JSON, ideal para estructurar datos complejos.

- `<driver/mcpwm.h>`: Librería nativa del ESP32 para generar señales PWM avanzadas (control de motores).

6.2. Credenciales WiFi

- `const char* ssid = "Wifi Homer";`
Nombre de la red inalámbrica a la que se conecta el ESP32.
- `const char* password = ".^na101064*";`
Contraseña para la red WiFi.

6.3. Configuración del Cliente MQTT

- `mqtt_server`: IP local del servidor MQTT (broker).
- `mqtt_username`, `mqtt_password`: Credenciales de autenticación para el cliente MQTT.
- `WiFiClient espClient`: Cliente base de red.
- `PubSubClient client(espClient)`: Cliente MQTT usando `espClient` como transporte TCP.

6.4. Configuración de LEDs

- `#define PIN 2`: Pin digital del ESP32 donde se conecta la tira de LEDs.
- `NUMPIXELS = 9`: Número total de LEDs.
- `Adafruit_NeoPixel pixels(...)`: Objeto controlador de la tira RGB.
- `bool controlLed9Activo`: Determina si el LED 9 debe cambiar su color automáticamente según el estado día/noche.
- `String estadoDiaNoche = ".es de noche"`: Estado simulado del entorno, útil para pruebas.

6.5. Pines de Entrada y Sensores

- `SENSOR_PIN = 35`: Pin analógico conectado al sensor PIR de movimiento.
- `sensorPin = 34`: Pin conectado al sensor de luz (LDR o fotodiodo).
- `trigPin = 25`, `echoPin = 33`: Pines del sensor ultrasónico para medir distancia.
- `sensor1 = 12`, `sensor2 = 13`: Pines auxiliares para sensores adicionales (por ejemplo, posición de una caja).

6.6. Pines del Motor (Puente H)

- Motor 1:
 - PWM: `gpioPWMA = 18`
 - Dirección: `gpioDir1 = 5, gpioDir2 = 17`
- Motor 2:
 - PWM: `gpioPWM1A = 26`
 - Dirección: `gpioDir3 = 27, gpioDir4 = 14`

6.7. Tópicos MQTT Definidos

- `ledControlTopic[]`: Lista de tópicos para controlar individualmente los 9 LEDs RGB.
- `diaNocheTopic`: Tópico que indica si es de día o de noche.
- `luxTopic`: Nivel de luz ambiental medido.
- `motionTopic`: Estado del sensor de movimiento PIR.
- `distanceTopic`: Distancia medida por el sensor ultrasónico.
- `topic_motor, topic_motor2`: Tópicos para controlar dos motores independientes.
- `topic_estado_caja`: Estado de apertura o cierre de una caja (ejemplo de actuador).
- `topic_luz_nocturna`: Control para una luz nocturna automática.

6.8. Variables Globales

- `valorSensor, voltaje, lux`: Variables usadas para el cálculo de luz ambiental.
- `duration, distance`: Tiempo y distancia medida con el sensor ultrasónico.
- `abrirCaja`: Bandera para determinar si una caja debe abrirse.
- `velocidadCaja = 100`: Valor de velocidad aplicado a los motores al abrir la caja.
- `ultimoMovimiento`: Marca de tiempo del último movimiento detectado.
- `tiempoInactividad = 18000`: Umbral de tiempo (3 minutos) sin movimiento antes de apagar las luces automáticamente.
- `lucesApagadasPorInactividad`: Bandera para saber si las luces fueron apagadas por falta de actividad.

6.9. Enumeración EstadoCaja

Se define una enumeración para representar distintos estados posibles de una caja motorizada:

- ESTADO_INVALIDO: Estado por defecto o desconocido.
- CAJA_CERRADA: Estado cuando la caja está completamente cerrada.
- CAJA_ABIERTA: Estado cuando la caja está completamente abierta.
- EN_TRANSICION: Indica que la caja se encuentra en movimiento.

6.10. Prototipos de Funciones

- `callback(char*, byte*, unsigned int)`: Función que se ejecuta al recibir un mensaje MQTT.
- `procesarMensajeJSON(char*, StaticJsonDocument<256>&)`: Procesa mensajes JSON para ejecutar acciones, como actualizar LEDs o activar motores.

6.11. Tópicos MQTT Utilizados

- `ledControlTopic[i]`: Arreglo de tópicos (`home/led_X/set`) para controlar individualmente los LEDs 1 a 9.
- `diaNocheTopic = casa/luz/dia_noche`: Publica o recibe el estado ambiental (día o noche).
- `luxTopic = casa/luz/lux`: Tópico para publicar los niveles de iluminación (lux) detectados por el sensor de luz.
- `motionTopic = "home/motion_sensor"`: Publica el estado del sensor de movimiento PIR (1 si detecta movimiento, 0 en reposo).
- `topic_motor = "home/motor/speed/set"`: Tópico para el control del primer motor (dirección y velocidad).
- `topic_motor2 = "home/motor2/speed/set"`: Tópico para el segundo motor, usado generalmente en sistemas de doble eje o cajas automatizadas.
- `topic_estado_caja = "home/caja/estado/set"`: Define el estado deseado de una caja controlada por motores (abrir/cerrar).
- `topic_luz_nocturna = "home/luz/nocturna"`: Permite encender o apagar una luz nocturna automática según condiciones ambientales.

6.12. Función `setup()`

La función `setup()` se ejecuta una única vez al encender o reiniciar el ESP32. En ella se configuran todos los componentes clave del sistema: comunicación serial, LEDs, pines de entrada/salida, conexión WiFi, comunicación MQTT y control de motores mediante MCPWM.

```
1 void setup() {  
2   Serial.begin(115200);  
3   pixels.begin();  
4   configurarPines();  
5   conectarWiFi();  
6   configurarMQTT();  
7   configurarMCPWM();  
8   Serial.println(" Configuración completada");  
9   ultimoMovimiento = millis();  
10 }
```

Descripción línea por línea:

- `Serial.begin(115200);`
Inicializa la comunicación serial a 115200 baudios para monitoreo y depuración desde el ordenador.
- `pixels.begin();`
Inicializa la tira de LEDs NeoPixel para que puedan ser controlados desde el código. Es obligatorio antes de usar cualquier función relacionada con los LEDs.
- `configurarPines();`
Configura los pines GPIO utilizados por sensores, motores y otros dispositivos como entradas o salidas, según corresponda.
- `conectarWiFi();`
Llama a la función que establece la conexión a la red WiFi usando las credenciales definidas previamente.
- `configurarMQTT();`
Inicializa el cliente MQTT, establece el servidor broker, define la función de callback y se suscribe a los tópicos necesarios.
- `configurarMCPWM();`
Configura el módulo de control PWM del ESP32 para habilitar el manejo de motores a través de señales de modulación por ancho de pulso.
- `Serial.println(" Configuración completada");`
Imprime un mensaje indicando que todas las configuraciones iniciales fueron completadas exitosamente.
- `ultimoMovimiento = millis();`
Inicializa la variable que registra la última detección de movimiento, útil para aplicar lógica de inactividad. En este caso para apagar luces tras un periodo de inactividad.

Resumen: La función `setup()` establece todos los parámetros y módulos necesarios para que el sistema funcione correctamente. Sin esta inicialización, los sensores, actuadores y la comunicación con otros dispositivos no estarían listos para operar durante el ciclo principal (`loop()`).

6.13. Función `configurarPines()`

Esta función configura los pines GPIO del ESP32 que están conectados a sensores y motores. Es esencial para asegurar que cada pin actúe correctamente como entrada o salida y que los motores comiencen en estado apagado.

```
1 void configurarPines() {  
2   pinMode(SENSOR_PIN, INPUT);  
3   pinMode(gpioDir1, OUTPUT);  
4   pinMode(gpioDir2, OUTPUT);  
5   pinMode(gpioDir3, OUTPUT);  
6   pinMode(gpioDir4, OUTPUT);  
7   pinMode(sensor1, INPUT);  
8   pinMode(sensor2, INPUT);  
9   digitalWrite(gpioDir1, LOW);  
10  digitalWrite(gpioDir2, LOW);  
11  digitalWrite(gpioDir3, LOW);  
12  digitalWrite(gpioDir4, LOW);  
13 }
```

Descripción línea por línea:

- `pinMode(SENSOR_PIN, INPUT);`
Configura el pin conectado al sensor de movimiento (PIR) como entrada digital.
- `pinMode(gpioDir1, OUTPUT);` y `pinMode(gpioDir2, OUTPUT);`
Establecen los pines de dirección del primer motor como salidas. Estos controlan el sentido de giro.
- `pinMode(gpioDir3, OUTPUT);` y `pinMode(gpioDir4, OUTPUT);`
Configuran los pines de dirección del segundo motor también como salidas.
- `pinMode(sensor1, INPUT);` y `pinMode(sensor2, INPUT);`
Definen los pines conectados a sensores digitales adicionales (por ejemplo, para detectar posición o límites) como entradas.
- `digitalWrite(..., LOW);`
Se aseguran de que todos los pines de dirección de los motores estén en estado bajo al iniciar el sistema. Esto evita activaciones no deseadas de los motores en el arranque.

Resumen: La función `configurarPines()` establece el comportamiento de cada pin usado en el proyecto y garantiza que el sistema arranque en un estado seguro. Es una buena práctica inicializar explícitamente los pines para prevenir errores eléctricos o comportamientos impredecibles.

6.14. Función conectarWiFi()

Esta función tiene como propósito establecer la conexión del ESP32 a una red WiFi utilizando el nombre de red (SSID) y la contraseña previamente definidos.

```
1 void conectarWiFi() {  
2     Serial.println("Iniciando conexión WiFi...");  
3     WiFi.begin(ssid, password);  
4     while (WiFi.status() != WL_CONNECTED) {  
5         delay(500);  
6         Serial.print(".");  
7     }  
8     Serial.println("\n Conectado a WiFi");  
9 }
```

Descripción línea por línea:

- `Serial.println("Iniciando conexión WiFi...");`
Muestra un mensaje en el monitor serial indicando que ha comenzado el intento de conexión a la red WiFi.
- `WiFi.begin(ssid, password);`
Inicia el proceso de conexión a la red inalámbrica usando las credenciales almacenadas en las variables `ssid` y `password`.
- `while (WiFi.status() != WL_CONNECTED)`
Se mantiene en un bucle de espera hasta que el estado de la conexión indique que se ha establecido exitosamente.
- `delay(500);`
Introduce una pausa de 500 milisegundos entre cada intento de verificación del estado de conexión para evitar sobrecarga.
- `Serial.print(".");`
Imprime un punto en el monitor serial en cada iteración del bucle, funcionando como una barra de progreso visual.
- `Serial.println("\n Conectado a WiFi");`
Una vez que la conexión ha sido establecida, imprime un mensaje de confirmación en el monitor serial.

Resumen: Esta función es crucial para habilitar la comunicación del dispositivo con el servidor MQTT y otros servicios en red. Su diseño incluye retroalimentación visual por consola para facilitar la depuración y verificar si el ESP32 logró conectarse correctamente al WiFi.

6.15. Función configurarMQTT()

La función `configurarMQTT()` establece los parámetros de conexión del cliente MQTT, define la función de recepción de mensajes (callback) y suscribe el dispositivo a los tópicos necesarios para su operación. Esta función es clave para que el sistema IoT pueda interactuar con un broker MQTT como Mosquitto o plataformas como Home Assistant.

```
1 void configurarMQTT() {
2     client.setServer(mqtt_server, 1883);
3     client.setCallback(callback);
4     reconnect();
5
6     for (int i = 0; i < 9; i++) {
7         client.subscribe(ledControlTopic[i]);
8     }
9
10    client.subscribe(diaNocheTopic);
11    client.subscribe(topic_motor);
12    client.subscribe(topic_motor2);
13    client.subscribe(topic_estado_caja);
14    client.subscribe(topic_luz_nocturna);
15 }
```

Descripción línea por línea:

- `client.setServer(mqtt_server, 1883);`
Define la dirección IP del broker MQTT y el puerto de conexión estándar (1883). Esto permite al ESP32 saber a dónde enviar y desde dónde recibir mensajes.
- `client.setCallback(callback);`
Establece la función `callback()` como la encargada de gestionar los mensajes entrantes desde cualquier tópico al que el cliente esté suscrito.
- `reconnect();`
Intenta conectar el cliente al broker MQTT. Si la conexión no existe o se ha perdido, esta función la restablece.
- `for (int i = 0; i < 9; i++) { ... }`
Bucle que recorre los 9 tópicos correspondientes al control individual de cada LED RGB. El cliente se suscribe a cada uno para recibir órdenes de encendido/apagado o cambio de color.
- `client.subscribe(diaNocheTopic);`
Tópico que comunica si es de día o de noche, utilizado por el sistema para controlar el LED 9 o activar la luz nocturna.
- `client.subscribe(topic_motor);`
Tópico destinado a controlar el primer motor (por ejemplo, velocidad y dirección de giro).
- `client.subscribe(topic_motor2);`
Tópico para el segundo motor, útil para sistemas con doble eje o mecanismos sincronizados.
- `client.subscribe(topic_estado_caja);`
Permite recibir órdenes para abrir o cerrar una caja o compartimento controlado por motor.

- `client.subscribe(topic_luz_nocturna);`
Tópico que activa o desactiva la luz nocturna de manera automática según condiciones predefinidas.

Resumen: La función `configurarMQTT()` asegura que el dispositivo esté correctamente conectado al servidor MQTT y preparado para recibir comandos desde diversos tópicos. Esto permite una integración fluida con sistemas de control domótico y aplicaciones de automatización.

6.16. Función `configurarMCPWM()`

Esta función configura el módulo **MCPWM** (Motor Control Pulse Width Modulation) del ESP32 para controlar dos motores mediante señales PWM, especificando los parámetros necesarios para el funcionamiento de cada uno.

```
1 void configurarMCPWM() {
2     Serial.println("Configurando MCPWM...");
3
4     // Configuración del primer motor
5     mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, gpioPWM0A);
6     mcpwm_config_t pwm_config;
7     pwm_config.frequency = 10000;
8     pwm_config.cmpr_a = 0;
9     pwm_config.cmpr_b = 0;
10    pwm_config.counter_mode = MCPWM_UP_COUNTER;
11    pwm_config.duty_mode = MCPWM_DUTY_MODE_0;
12    mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);
13
14    // Configuración del segundo motor correctamente
15    mcpwm_gpio_init(MCPWM_UNIT_1, MCPWM0A, gpioPWM1A);
16    mcpwm_config_t pwm_config2;
17    pwm_config2.frequency = 10000;
18    pwm_config2.cmpr_a = 0;
19    pwm_config2.cmpr_b = 0;
20    pwm_config2.counter_mode = MCPWM_UP_COUNTER;
21    pwm_config2.duty_mode = MCPWM_DUTY_MODE_0;
22    mcpwm_init(MCPWM_UNIT_1, MCPWM_TIMER_1, &pwm_config2);
23 }
```

Descripción línea por línea:

- `Serial.println("Configurando MCPWM...");`
Imprime un mensaje por consola para indicar el inicio del proceso de configuración PWM.
- `mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, gpioPWM0A);`
Asocia el pin GPIO del primer motor a una señal PWM generada por la unidad 0 del MCPWM.

- `mcpwm_config_t pwm_config;`
Declara e inicializa la estructura de configuración para el primer motor.
- `pwm_config.frequency = 10000;`
Establece la frecuencia del PWM en 10 kHz, adecuada para aplicaciones de control de velocidad en motores.
- `pwm_config.cmpr_a = 0;` y `pwm_config.cmpr_b = 0;`
Define el ciclo de trabajo inicial como 0 %, es decir, el motor comenzará apagado.
- `pwm_config.counter_mode = MCPWM_UP_COUNTER;`
Configura el contador en modo ascendente (cuenta desde 0 hasta el valor máximo y luego reinicia).
- `pwm_config.duty_mode = MCPWM_DUTY_MODE_0;`
Define el modo de trabajo de la señal PWM (modo básico donde la señal es activa durante la parte alta del ciclo).
- `mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);`
Aplica la configuración al temporizador 0 de la unidad 0 del MCPWM.
- `mcpwm_gpio_init(MCPWM_UNIT_1, MCPWM0A, gpioPWM1A);`
Asocia el pin del segundo motor a una señal PWM generada por la unidad 1 del MCPWM.
- `mcpwm_config_t pwm_config2;`
Se define una nueva estructura de configuración para el segundo motor, separada del primero.
- `pwm_config2.frequency = 10000;`
También usa una frecuencia de 10 kHz para el segundo motor.
- `pwm_config2.cmpr_a = 0;` y `pwm_config2.cmpr_b = 0;`
Ambos ciclos de trabajo inician en 0 %.
- `pwm_config2.counter_mode = MCPWM_UP_COUNTER;`
Modo de conteo ascendente.
- `pwm_config2.duty_mode = MCPWM_DUTY_MODE_0;`
Modo de señal PWM básico.
- `mcpwm_init(MCPWM_UNIT_1, MCPWM_TIMER_1, &pwm_config2);`
Aplica la configuración a la unidad 1 y temporizador 1 del MCPWM, asegurando que ambos motores se controlen de forma independiente.

Resumen: La función `configurarMCPWM()` inicializa los módulos MCPWM del ESP32 para el control independiente de dos motores. Cada motor recibe su propia señal PWM, con su propio pin y temporizador, permitiendo un control preciso de velocidad y dirección.

6.17. Función loop()

La función `loop()` es el núcleo del sistema embebido. Se ejecuta de forma continua y gestiona tanto la conectividad MQTT como las tareas periódicas relacionadas con sensores, actuadores y lógica de automatización.

```
1 void loop() {
2   if (!client.connected()) reconnect();
3   client.loop();
4   controlarCaja();
5
6   static unsigned long lastTime = 0;
7   if (millis() - lastTime >= 5000) {
8     lastTime = millis();
9     medirLux();
10    detectarMovimiento();
11
12    if (millis() - ultimoMovimiento >= tiempoInactividad && !lucesApagadasPorInactividad)
13      apagarTodasLasLuces();
14    lucesApagadasPorInactividad = true;
15  }
16
17  if (controlLed9Activo) actualizarLed9();
18 }
19 }
```

Descripción línea por línea:

- `if (!client.connected()) reconnect();`
Verifica si el cliente MQTT está conectado. Si no lo está, intenta reconectarse llamando a la función `reconnect()`.
- `client.loop();`
Llama al método interno del cliente MQTT, que mantiene la conexión activa y gestiona los mensajes entrantes y salientes.
- `controlarCaja();`
Llama a una función que gestiona el comportamiento de una caja motorizada, probablemente con base en mensajes recibidos o sensores de estado.
- `static unsigned long lastTime = 0;`
Variable que conserva su valor entre iteraciones del bucle. Se usa para medir intervalos de tiempo sin reiniciar el contador global `millis()`.
- `if (millis() - lastTime >= 5000)`
Cada 5 segundos, se ejecuta el siguiente bloque de instrucciones, lo que permite espaciar lecturas de sensores y ahorrar recursos.
- `lastTime = millis();`
Actualiza el marcador de tiempo para la próxima ejecución dentro de 5 segundos.

- `medirLux()`;
Mide la cantidad de luz ambiental y la publica en el tópico correspondiente.
- `detectarMovimiento()`;
Lee el sensor PIR y actualiza el estado de movimiento en el sistema.
- `if (millis() - ultimoMovimiento >= tiempoInactividad ...)`
Si ha pasado más tiempo del definido como inactividad (`tiempoInactividad`) desde el último movimiento detectado, y las luces aún no han sido apagadas automáticamente, se ejecuta la lógica de ahorro de energía.
- `apagarTodasLasLuces()`;
Apaga todos los LEDs o sistemas de iluminación conectados.
- `lucesApagadasPorInactividad = true;`
Marca que las luces fueron apagadas por inactividad, para evitar ejecutar esta acción repetidamente.
- `if (controlLed9Activo) actualizarLed9();`
Si está activado el modo automático del LED 9 (controlado por condiciones como día/noche), se actualiza su estado visual.

Resumen: La función `loop()` gestiona la operación continua del sistema: mantiene la conexión MQTT, ejecuta tareas cada cierto tiempo, procesa lecturas de sensores, y aplica lógica de automatización (como apagar luces por inactividad). Es el corazón reactivo y periódico del sistema embebido.

6.18. Función `medirLux()`

La función `medirLux()` permite medir la intensidad de luz ambiental utilizando un sensor analógico de luz (como un fototransistor o una fotorresistencia), convirtiendo el valor leído a una estimación en lux y enviándola al servidor MQTT. Esta información puede ser utilizada por otros nodos del sistema para ajustar iluminación, activar alertas o tomar decisiones contextuales.

```
1 void medirLux() {  
2     valorSensor = analogRead(sensorPin);  
3     voltaje = (valorSensor / 4095.0) * 3.3;  
4     lux = (600.0 * voltaje) / 2.3;  
5     char luxMessage[8];  
6     dtostrf(lux, 1, 2, luxMessage);  
7     client.publish(luxTopic, luxMessage);  
8     Serial.print("Lux: ");  
9     Serial.println(luxMessage);  
10 }
```

Descripción línea por línea:

- `valorSensor = analogRead(sensorPin);`
Lee el valor analógico del sensor de luz. El valor leído varía entre 0 y 4095 en el ESP32, que tiene una resolución ADC de 12 bits.

- `voltaje = (valorSensor / 4095.0) * 3.3;`
Convierte el valor analógico leído a voltaje, considerando un rango de 0 a 3.3V.
- `lux = (600.0 * voltaje) / 2.3;`
Convierte el voltaje a una estimación de lux. Este factor depende del tipo específico de sensor utilizado y su curva de respuesta. En este caso, se aplica una fórmula empírica basada en una calibración estimada.
- `char luxMessage[8];`
Se declara un arreglo de caracteres para almacenar la representación en texto del valor de lux que será enviado por MQTT.
- `dtostrf(lux, 1, 2, luxMessage);`
Convierte el valor numérico de tipo `float` a una cadena de caracteres con un decimal de al menos un dígito y dos cifras decimales.
- `client.publish(luxTopic, luxMessage);`
Publica el valor de lux en formato texto al tópico MQTT correspondiente, permitiendo que otros dispositivos lo consulten.
- `Serial.print("Lux: "); Serial.println(luxMessage);`
Imprime por consola el valor de lux calculado, útil para monitoreo y depuración durante el desarrollo.

6.19. Función `detectarMovimiento()`

Esta función tiene por objeto decodificar el valor del sensor de movimiento y, en caso de detección, reiniciar el contador de inactividad que apaga automáticamente las luces tras un período sin movimiento. Además, publica el estado del sensor al broker MQTT y brinda retroalimentación por consola.

```
1 void detectarMovimiento() {
2     int sensorValue = digitalRead(SENSOR_PIN);
3
4     // Si se detecta movimiento, actualiza el tiempo
5     if (sensorValue == HIGH) {
6         ultimoMovimiento = millis();
7         lucesApagadasPorInactividad = false; // Marcar que las luces pueden volver a encenderse
8     }
9
10    // Publicar al broker MQTT como lo hacías
11    char motionState[2];
12    sprintf(motionState, "%d", sensorValue);
13    client.publish(motionTopic, motionState);
14
15    Serial.print("Movimiento: ");
16    Serial.println(sensorValue ? "Detectado" : "No detectado");
17 }
```

Descripción línea por línea:

- `int sensorValue = digitalRead(SENSOR_PIN);`
Lee el estado del pin digital al que está conectado el sensor PIR. Devuelve HIGH si hay movimiento y LOW si no lo hay.
- `if (sensorValue == HIGH)`
Si se detecta movimiento:
 - `ultimoMovimiento = millis();`
Reinicia el contador de inactividad almacenando el momento actual, lo que evita que las luces se apaguen.
 - `lucesApagadasPorInactividad = false;`
Marca que el sistema puede volver a encender luces, ya que se ha restablecido la actividad.
- `char motionState[2];`
Arreglo de caracteres que contendrá el estado del sensor en formato de texto.
- `sprintf(motionState, "%d", sensorValue);`
Convierte el valor leído (0 o 1) a texto para su publicación por MQTT.
- `client.publish(motionTopic, motionState);`
Publica el estado actual del sensor PIR en el tópico MQTT `home/motion_sensor`.
- `Serial.print("Movimiento: "); Serial.println(...);`
Imprime por consola si se ha detectado o no movimiento, útil para monitoreo en tiempo real.

Resumen: `detectarMovimiento()` permite al sistema IoT interpretar el estado de un sensor de movimiento, reiniciar la lógica de apagado por inactividad cuando sea necesario, y comunicar su estado a través del protocolo MQTT. Esta funcionalidad es esencial para automatizar la iluminación o sistemas de seguridad basados en presencia para que vuelvan a encenderse si se detecta presencia.

6.20. Control de Caja Motorizada: `controlarCaja()`, `moverCaja()` y `detenerMotorCaja()`

Este conjunto de funciones controla el comportamiento de una caja motorizada con base en dos sensores digitales que indican su posición. El sistema es capaz de determinar si la caja está completamente cerrada, abierta, en movimiento, o si los sensores se encuentran en un estado inválido.

La señal `abrirCaja`, que es enviada desde Home Assistant, determina si el sistema debe proceder a abrir o cerrar la caja. Según esta señal y el estado actual detectado, se activa o detiene el motor.

Lógica de estados:

- **CAJA_CERRADA:** ambos sensores activos (HIGH).
- **CAJA_ABIERTA:** solo el sensor 2 activo.

- **EN_TRANSICION:** ambos sensores inactivos (LOW), la caja se está moviendo.
- **ESTADO_INVALIDO:** cualquier combinación no contemplada.

```

1 void controlarCaja() {
2     int s1 = digitalRead(sensor1);
3     int s2 = digitalRead(sensor2);
4
5     EstadoCaja estadoActual;
6
7     if (s1 == HIGH && s2 == HIGH) {
8         estadoActual = CAJA_CERRADA;
9     } else if (s1 == LOW && s2 == HIGH) {
10        estadoActual = CAJA_ABIERTA;
11    } else if (s1 == LOW && s2 == LOW) {
12        estadoActual = EN_TRANSICION;
13    } else {
14        estadoActual = ESTADO_INVALIDO;
15    }

```

Interpretación de sensores:

- Se leen los sensores digitales.
- Se asigna el estado correspondiente según las combinaciones posibles.

```

1 if (estadoActual != ultimoEstado) {
2     switch (estadoActual) {
3         case CAJA_CERRADA:
4             Serial.println(" Caja cerrada");
5             if (abrirCaja) {
6                 Serial.println(" Abriendo...");
7             }
8             break;
9         case CAJA_ABIERTA:
10            Serial.println(" Caja abierta");
11            if (!abrirCaja) {
12                Serial.println(" Cerrando...");
13            }
14            break;
15        case EN_TRANSICION:
16            Serial.println(" Caja en transición...");
17            if (abrirCaja) {
18                Serial.println(" Abriendo...");
19            } else {
20                Serial.println(" Cerrando...");
21            }
22            break;
23        case ESTADO_INVALIDO:
24            Serial.println(" Estado de sensores inválido.");

```



```
25     break;
26 }
27 ultimoEstado = estadoActual;
28 }
```

Impresión informativa:

- Solo se imprime si el estado ha cambiado.
- Informa si se ejecutará una acción de apertura o cierre en base a la señal `abrirCaja`.

```
1  if (estadoActual == CAJA_CERRADA) {
2      if (abrirCaja) {
3          moverCaja(velocidadCaja);
4      } else {
5          detenerMotorCaja();
6      }
7  } else if (estadoActual == CAJA_ABIERTA) {
8      if (!abrirCaja) {
9          moverCaja(-velocidadCaja);
10     } else {
11         detenerMotorCaja();
12     }
13 } else if (estadoActual == EN_TRANSICION) {
14     if (abrirCaja) {
15         moverCaja(velocidadCaja);
16     } else {
17         moverCaja(-velocidadCaja);
18     }
19 } else {
20     detenerMotorCaja();
21 }
22 }
```

Lógica de acción del motor:

- Si la caja está cerrada y debe abrirse, se activa el motor en dirección de apertura.
- Si está abierta y debe cerrarse, se activa en sentido contrario.
- Si está en movimiento (transición), el motor actúa conforme a la orden `abrirCaja`.
- Si el estado es inválido, se detiene el motor por seguridad.

Funciones auxiliares

```
1  void moverCaja(int velocidad) {
2      float duty = map(abs(velocidad), 1, 100, 60, 100);
3      digitalWrite(gpioDir1, velocidad > 0);
4      digitalWrite(gpioDir2, velocidad < 0);
5      mcpwm_set_duty(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_A, duty);
6  }
```



Función moverCaja():

- Calcula el ciclo de trabajo (duty cycle) en función de la velocidad solicitada.
- Ajusta los pines de dirección para establecer el sentido de giro.
- Aplica la señal PWM usando el módulo MCPWM del ESP32.

```
1 void detenerMotorCaja() {  
2     digitalWrite(gpioDir1, LOW);  
3     digitalWrite(gpioDir2, LOW);  
4     mcpwm_set_duty(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_A, 0);  
5 }
```

Función detenerMotorCaja():

- Pone ambos pines de dirección en bajo para detener el motor.
- Establece el duty cycle en 0 %, es decir, desactiva el PWM.

Resumen: El conjunto de funciones descritas permite un control preciso y seguro de una caja motorizada, adaptándose tanto a su estado físico como a las órdenes de control externo. Gracias a la lógica basada en sensores y estados, se asegura que el motor actúe solo cuando sea necesario, prolongando la vida útil del sistema y evitando movimientos incorrectos.

6.21. Función callback()

La función `callback()` es invocada automáticamente cada vez que el cliente MQTT recibe un mensaje en uno de los tópicos suscritos. Su objetivo es interpretar el mensaje, identificar el tópico, y ejecutar la acción correspondiente sobre luces, motores o la caja motorizada.

```
1 void callback(char* topic, byte* payload, unsigned int length) {  
2     String message;  
3     for (int i = 0; i < length; i++) {  
4         message += (char)payload[i];  
5     }  
6  
7     Serial.print(" MQTT [");  
8     Serial.print(topic);  
9     Serial.print("]: ");  
10    Serial.println(message);
```

Lectura del mensaje:

- Se convierte el arreglo de bytes `payload` en un `String` legible (`message`).
- Se imprime el contenido y el tópico desde el que fue recibido.


```
1  if (String(topic) == topic_luz_nocturna) {
2      if (message == "encender") {
3          controlLed9Activo = true;
4          actualizarLed9();
5          Serial.println(" Luz nocturna activada (azul o rojo según hora)");
6      } else if (message == "apagar") {
7          controlLed9Activo = false;
8          pixels.setPixelColor(8, pixels.Color(0, 0, 0));
9          pixels.show();
10         Serial.println(" Luz nocturna desactivada");
11     }
12     return;
13 }
```

Control de luz nocturna:

- Si el mensaje es `"encender"`, se activa el modo automático del LED 9.
- Si el mensaje es `"apagar"`, el LED 9 se apaga manualmente.

```
1  if (String(topic) == topic_motor2) {
2      procesarMotor(message, gpioDir3, gpioDir4, MCPWM_UNIT_1, MCPWM_TIMER_1, MCPWM_OPR
3      return;
4  }
```

Control del segundo motor:

- El mensaje se pasa a la función `procesarMotor()`, que controla el motor conectado al conjunto de pines de la caja o un eje secundario.

```
1  if (String(topic) == topic_motor) {
2      velocidadCaja = constrain(message.toInt(), 0, 100);
3      Serial.print(" Velocidad caja: ");
4      Serial.println(velocidadCaja);
5
6      if (velocidadCaja == 0) {
7          Serial.println(" Velocidad 0 - Deteniendo motor");
8          detenerMotorCaja();
9      }
10     return;
11 }
```

Ajuste de velocidad del motor principal (caja):

- Se actualiza la variable `velocidadCaja` según el mensaje recibido.

- Si se recibe velocidad 0, el motor se detiene inmediatamente como medida de seguridad.

```

1  if (String(topic) == topic_estado_caja) {
2      abrirCaja = (message == "abrir");
3      Serial.print(" Acción recibida: ");
4      Serial.println(abrirCaja ? "ABRIR" : "CERRAR");
5      return;
6  }

```

Control de estado de la caja:

- Cambia el valor de `abrirCaja` según el mensaje recibido (`"abrir"` o `"cerrar"`).
- Esto afectará la lógica de movimiento evaluada en la función `controlarCaja()`.

6.22. Función `procesarMensajeJSON()`

La función `procesarMensajeJSON()` interpreta un mensaje JSON recibido vía MQTT, extrae los valores RGB para controlar un LED específico y, además, reinicia el temporizador de inactividad cuando se detecta actividad manual desde una plataforma como Home Assistant.

```

1
2  void procesarMensajeJSON(char* topic, StaticJsonDocument<256>& doc) {
3      String state = doc["state"];
4
5      int r = doc["color"]["r"];
6      int g = doc["color"]["g"];
7      int b = doc["color"]["b"];
8
9      if (r < 0 || r > 255 || g < 0 || g > 255 || b < 0 || b > 255) return;
10
11     actualizarLed(topic, r, g, b);
12
13     // Reset por actividad manual desde Home Assistant
14     ultimoMovimiento = millis();
15     lucesApagadasPorInactividad = false;
16
17     Serial.println(" Actividad manual detectada: reiniciando temporizador");
18 }

```

Descripción línea por línea:

- `String state = doc["state"];`
Extrae el campo "state" del mensaje JSON. Aunque no se utiliza aquí directamente, puede servir para detectar comandos como `"ON"` o `"OFF"`.

- `int r, g, b = doc[çolor"][r/g/b];`
Obtiene los valores de color rojo, verde y azul del objeto `çolor` dentro del mensaje JSON.
- `if (...)`
Verifica que los valores RGB estén dentro del rango permitido (0 a 255). Si no lo están, se descarta el mensaje para evitar errores.
- `actualizarLed(...);`
Llama a la función que aplica el color especificado al LED correspondiente al tópico MQTT.
- `ultimoMovimiento = millis();`
Reinicia el temporizador de actividad. Si este no se renicia si no se detecta movimiento y se manipulan las luces desde mi interfaz web estas no se apagarían automáticamente tras el periodo de inactividad.
- `lucesApagadasPorInactividad = false;`
Indica que las luces pueden volver a encenderse, ya que se ha registrado una acción reciente del usuario.
- `Serial.println(...);`
Muestra un mensaje en el monitor serial indicando que se ha detectado actividad manual.

Resumen: `procesarMensajeJSON()` no solo aplica un cambio visual al sistema (control de LED), sino que también reinicia el control de inactividad, integrando de forma inteligente la intervención humana dentro de un sistema automatizado. Es especialmente útil en entornos domóticos donde las acciones manuales deben priorizarse frente a rutinas automáticas.

6.23. Función `actualizarLed()`

La función `actualizarLed()` se encarga de aplicar un color RGB a uno de los LEDs de la tira NeoPixel, en función del tópico MQTT recibido. Esta función es llamada después de interpretar un mensaje JSON que contiene valores de color.

```
1 void actualizarLed(const char* topic, int r, int g, int b) {
2     for (int i = 0; i < 9; i++) {
3         if (String(topic) == ledControlTopic[i]) {
4             pixels.setPixelColor(i, pixels.Color(r, g, b));
5             pixels.show();
6             return;
7         }
8     }
9 }
```

Descripción línea por línea:

- `for (int i = 0; i < 9; i++)`
Se recorre el arreglo de tópicos `ledControlTopic[]` que contiene los nombres de los tópicos MQTT asociados a cada uno de los 9 LEDs.
- `if (String(topic) == ledControlTopic[i])`
Compara el tópico recibido como argumento con cada tópico del arreglo. Si hay coincidencia, se identifica cuál LED debe ser modificado.
- `pixels.setPixelColor(i, pixels.Color(r, g, b));`
Establece el color del LED en la posición `i` utilizando los valores RGB recibidos.
- `pixels.show();`
Aplica todos los cambios realizados en los LEDs. Es obligatorio para que las modificaciones sean visibles físicamente.
- `return;`
Termina la función tras encontrar y actualizar el LED correspondiente. Esto evita que se siga iterando innecesariamente.

Resumen: La función `actualizarLed()` vincula un mensaje MQTT con un LED específico, aplicando un color RGB personalizado. Es un componente clave en sistemas de iluminación inteligente donde cada LED puede ser controlado de forma individual desde plataformas como Home Assistant.

6.24. Función `actualizarLed9()`

La función `actualizarLed9()` actualiza el color del LED número 9 de la tira `NeoPixel` en función del estado ambiental actual (día o noche), reflejado en la variable `estadoDiaNoche`. Se utiliza como parte de una lógica de iluminación nocturna automática.

```
1 void actualizarLed9() {  
2   if (estadoDiaNoche == "es de noche") {  
3     pixels.setPixelColor(8, pixels.Color(0, 0, 255));  
4   } else {  
5     pixels.setPixelColor(8, pixels.Color(255, 0, 0));  
6   }  
7   pixels.show();  
8 }
```

Descripción línea por línea:

- `if (estadoDiaNoche == "es de noche")`
Evalúa el valor de la variable `estadoDiaNoche`, que debe ser actualizada previamente por un sensor o mensaje MQTT. Si es de noche, se procede a iluminar el LED en azul.
- `pixels.setPixelColor(8, pixels.Color(0, 0, 255));`
Establece el color azul en el LED número 9 (índice 8 en programación). Representa la activación de una luz nocturna.

- `else { pixels.setPixelColor(8, pixels.Color(255, 0, 0)); }`
Si no es de noche (es de día), el LED se configura en rojo, lo cual sirve como indicativo visual del cambio de estado.
- `pixels.show();`
Aplica el cambio de color al LED. Es obligatorio para que el nuevo color se visualice.

Resumen: La función `actualizarLed9()` permite representar de forma visual el estado del entorno (día/noche) usando colores simbólicos en un LED dedicado. Su activación automática puede estar controlada por sensores de luz o por señales externas enviadas desde una plataforma de automatización del hogar como en este caso se esta usando una API de meteorología.

6.25. Función `apagarTodasLasLuces()`

La función `apagarTodasLasLuces()` apaga los 9 LEDs RGB del sistema estableciendo su color en negro (`RGB(0, 0, 0)`). Es utilizada cuando se detecta inactividad prolongada, como una forma de ahorro energético o automatización.

```
1 void apagarTodasLasLuces() {  
2     for (int i = 0; i < 9; i++) {  
3         actualizarLed(ledControlTopic[i], 0, 0, 0); // Apaga cada LED  
4     }  
5     Serial.println(" Todas las luces apagadas por inactividad");  
6 }
```

Descripción línea por línea:

- `for (int i = 0; i < 9; i++)`
Itera sobre los 9 LEDs definidos en el sistema.
- `actualizarLed(ledControlTopic[i], 0, 0, 0);`
Llama a la función `actualizarLed()` para cada LED, pasándole el tópico correspondiente y el color negro (`RGB 0,0,0`), lo que equivale a apagarlo.
- `Serial.println(...);`
Imprime un mensaje en el monitor serial indicando que todas las luces han sido apagadas debido a inactividad, lo cual es útil para verificación en tiempo real.

Resumen: `apagarTodasLasLuces()` es una función de soporte fundamental en la gestión de eficiencia energética del sistema. Permite apagar automáticamente la iluminación cuando no se detecta presencia durante un tiempo prolongado, reforzando la lógica de automatización inteligente.

6.26. Función `reconnect()`

Esta función asegura que el cliente MQTT permanezca conectado. Si la conexión se pierde, intenta reconectarse al servidor de manera indefinida hasta lograrlo. Es fundamental para mantener la comunicación en sistemas IoT confiables.

```
1 void reconnect() {
2     while (!client.connected()) {
3         Serial.print("Conectando al servidor MQTT...");
4         if (client.connect("ESP32_Client", mqtt_username, mqtt_password)) {
5             Serial.println(" Conectado");
6         } else {
7             Serial.print(" Fallo, rc=");
8             Serial.print(client.state());
9             Serial.println(" Intentando de nuevo en 5 segundos");
10            delay(5000);
11        }
12    }
13 }
```

Descripción línea por línea:

- `while (!client.connected()) {`
Se entra en un bucle que se mantiene activo mientras el cliente MQTT no esté conectado al servidor.
- `Serial.print("Conectando al servidor MQTT...");`
Imprime un mensaje informando que se está intentando establecer la conexión.
- `if (client.connect("ESP32_Client", mqtt_username, mqtt_password)) {`
Intenta conectar al broker MQTT usando un identificador de cliente y las credenciales definidas. Si tiene éxito, entra al bloque correspondiente.
- `Serial.println(" Conectado");`
Muestra en el monitor serial un mensaje de éxito una vez que se ha conectado correctamente.
- `else {`
Si la conexión falla, se ejecutan las instrucciones dentro de este bloque.
- `Serial.print(" Fallo, rc=");`
Indica que ocurrió un error de conexión.
- `Serial.print(client.state());`
Muestra el código de error correspondiente al estado del cliente MQTT, lo cual ayuda en la depuración del problema.
- `Serial.println("Intentando de nuevo en 5 segundos");`
Informa al usuario que el sistema intentará conectarse nuevamente luego de una pausa.
- `delay(5000);`
Introduce una espera de 5 segundos antes de volver a intentar la conexión, evitando ciclos de reconexión demasiado rápidos que puedan saturar la red o el broker.

Resumen: La función `reconnect()` mantiene la conexión MQTT activa. Si el ESP32 pierde la conexión, esta función garantiza que se reestablezca automáticamente, asegurando la continuidad del sistema IoT y la recepción de comandos desde el broker.

6.27. Código de la ESP32 (Firmware en C++)

A continuación, se presenta el código completo implementado en la placa ESP32, que permite la comunicación con Home Assistant mediante MQTT, el control de nueve LEDs RGB, la lectura de sensores y el accionamiento de motores a través de modulación por ancho de pulso (PWM).

```
1  #include <WiFi.h>
2  #include <PubSubClient.h>
3  #include <Adafruit_NeoPixel.h>
4  #include <ArduinoJson.h>
5  #include <driver/mcpwm.h>
6
7  // ----- WiFi -----
8  const char* ssid = "Wifi Homer";
9  const char* password = "Ana101064*";
10
11 // ----- MQTT -----
12 const char* mqtt_server = "192.168.1.140";
13 const char* mqtt_username = "miguelmqtt";
14 const char* mqtt_password = "miguelmqtt";
15
16 WiFiClient espClient;
17 PubSubClient client(espClient);
18
19 // ----- LEDs -----
20 #define PIN 2
21 #define NUMPIXELS 9
22 Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
23 bool controlled9Activo = false;
24 String estadoDiaNoche = "es de noche";
25
26 // ----- Pines sensores -----
27 #define SENSOR_PIN 35 // PIR
28 int sensorPin = 34; // Sensor de luz
29 const int trigPin = 25;
30 const int echoPin = 33;
31 const int sensor1 = 12;
32 const int sensor2 = 13;
33
34 // ----- Motor Puente H -----
35 const int gpioPwm0A = 18;
36 const int gpioDir1 = 5;
37 const int gpioDir2 = 17;
38 const int gpioPwm1A = 26;
39 const int gpioDir3 = 27;
40 const int gpioDir4 = 14;
41
42 // ----- MQTT Topics -----
```



```
43  const char* ledControlTopic[] = {
44      "home/led_1/set", "home/led_2/set", "home/led_3/set",
45      "home/led_4/set", "home/led_5/set", "home/led_6/set",
46      "home/led_7/set", "home/led_8/set", "home/led_9/set"
47  };
48  const char* diaNocheTopic = "casa/luz/dia_noche";
49  const char* luxTopic = "casa/luz/lux";
50  const char* motionTopic = "home/motion_sensor";
51  const char* distanceTopic = "home/ultrasonic/distance";
52  const char* topic_motor = "home/motor/speed/set";
53  const char* topic_motor2 = "home/motor2/speed/set";
54  const char* topic_estado_caja = "home/caja/estado/set";
55  const char* topic_luz_nocturna = "home/luz/nocturna";
56
57
58  // ----- Variables -----
59  int valorSensor = 0;
60  float voltaje = 0.0;
61  float lux = 0.0;
62  long duration;
63  float distance;
64  bool abrirCaja = true;
65  int velocidadCaja = 100;
66
67  unsigned long ultimoMovimiento = 0;
68  const unsigned long tiempoInactividad = 18000; // 3 minutos en ms
69  bool lucesApagadasPorInactividad = false;
70
71
72
73  enum EstadoCaja {
74      ESTADO_INVALIDO,
75      CAJA_CERRADA,
76      CAJA_ABIERTA,
77      EN_TRANSICION
78  };
79
80  EstadoCaja ultimoEstado = ESTADO_INVALIDO;
81
82  // ----- Prototipos -----
83  void callback(char*, byte*, unsigned int);
84  void procesarMensajeJSON(char*, StaticJsonDocument<256>&);
85
86  // ----- Setup -----
87  void setup() {
88      Serial.begin(115200);
89      pixels.begin();
90      configurarPines();
```




```
91   conectarWiFi();
92   configurarMQTT();
93   configurarMCPWM();
94   Serial.println(" Configuración completada");
95   ultimoMovimiento = millis();
96
97 }
98
99 void configurarPines() {
100   pinMode(SENSOR_PIN, INPUT);
101   pinMode(gpioDir1, OUTPUT);
102   pinMode(gpioDir2, OUTPUT);
103   pinMode(gpioDir3, OUTPUT);
104   pinMode(gpioDir4, OUTPUT);
105   pinMode(sensor1, INPUT);
106   pinMode(sensor2, INPUT);
107   digitalWrite(gpioDir1, LOW);
108   digitalWrite(gpioDir2, LOW);
109   digitalWrite(gpioDir3, LOW);
110   digitalWrite(gpioDir4, LOW);
111 }
112
113 void conectarWiFi() {
114   Serial.println("Iniciando conexión WiFi...");
115   WiFi.begin(ssid, password);
116   while (WiFi.status() != WL_CONNECTED) {
117     delay(500);
118     Serial.print(".");
119   }
120   Serial.println("\n Conectado a WiFi");
121 }
122
123 void configurarMQTT() {
124   client.setServer(mqtt_server, 1883);
125   client.setCallback(callback);
126   reconnect();
127
128   for (int i = 0; i < 9; i++) {
129     client.subscribe(ledControlTopic[i]);
130   }
131
132   client.subscribe(diaNocheTopic);
133   client.subscribe(topic_motor);
134   client.subscribe(topic_motor2);
135   client.subscribe(topic_estado_caja);
136   client.subscribe(topic_luz_nocturna);
137
138 }
```

139

```
140 void configurarMCPWM() {
141     mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, gpioPWMOA);
142     mcpwm_config_t pwm_config;
143     pwm_config.frequency = 10000;
144     pwm_config.cmpr_a = 0;
145     pwm_config.cmpr_b = 0;
146     pwm_config.counter_mode = MCPWM_UP_COUNTER;
147     pwm_config.duty_mode = MCPWM_DUTY_MODE_0;
148     mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);
149
150     mcpwm_gpio_init(MCPWM_UNIT_1, MCPWM0A, gpioPWM1A);
151     mcpwm_config_t pwm_config2;
152     pwm_config2.frequency = 10000;
153     pwm_config2.cmpr_a = 0;
154     pwm_config2.cmpr_b = 0;
155     pwm_config2.counter_mode = MCPWM_UP_COUNTER;
156     pwm_config2.duty_mode = MCPWM_DUTY_MODE_0;
157     mcpwm_init(MCPWM_UNIT_1, MCPWM_TIMER_1, &pwm_config2);
158 }
```

159

```
160 // ----- Loop -----
```

161

```
162 void loop() {
163     if (!client.connected()) reconnect();
164     client.loop();
165     controlarCaja();
166
167     static unsigned long lastTime = 0;
168     if (millis() - lastTime >= 5000) {
169         lastTime = millis();
170         medirLux();
171         medirDistancia();
172         detectarMovimiento();
173
174         if (millis() - ultimoMovimiento >= tiempoInactividad && !lucesApagadasPorInactividad) {
175             apagarTodasLasLuces();
176             lucesApagadasPorInactividad = true;
177         }
178
179         if (controlLed9Activo) actualizarLed9();
180     }
181 }
```

181

182

```
183 // ----- Lógica de sensores -----
```

184

```
185 void medirLux() {
186     valorSensor = analogRead(sensorPin);
187     voltaje = (valorSensor / 4095.0) * 3.3;
```



```
187     lux = (600.0 * voltaje) / 2.3;
188     char luxMsg[8];
189     dtostrf(lux, 1, 2, luxMsg);
190     client.publish(luxTopic, luxMsg);
191     Serial.print("Lux: ");
192     Serial.println(luxMsg);
193 }
194
195
196 void detectarMovimiento() {
197     int sensorValue = digitalRead(SENSOR_PIN);
198
199     // Si se detecta movimiento, actualiza el tiempo
200     if (sensorValue == HIGH) {
201         ultimoMovimiento = millis();
202         lucesApagadasPorInactividad = false; // Marcar que las luces pueden volver a encender
203     }
204
205     // Publicar al broker MQTT como lo hacías
206     char motionState[2];
207     sprintf(motionState, "%d", sensorValue);
208     client.publish(motionTopic, motionState);
209
210     Serial.print("Movimiento: ");
211     Serial.println(sensorValue ? "Detectado" : "No detectado");
212 }
213
214
215 // ----- Caja lógica -----
216 void controlarCaja() {
217     int s1 = digitalRead(sensor1);
218     int s2 = digitalRead(sensor2);
219
220     EstadoCaja estadoActual;
221
222     if (s1 == HIGH && s2 == HIGH) {
223         estadoActual = CAJA_CERRADA;
224     } else if (s1 == LOW && s2 == HIGH) {
225         estadoActual = CAJA_ABIERTA;
226     } else if (s1 == LOW && s2 == LOW) {
227         estadoActual = EN_TRANSICION;
228     } else {
229         estadoActual = ESTADO_INVALIDO;
230     }
231
232     // Solo imprimir si hay cambio de estado
233     if (estadoActual != ultimoEstado) {
234         switch (estadoActual) {
```



```
235     case CAJA_CERRADA:
236         Serial.println(" Caja cerrada");
237         if (abrirCaja) {
238             Serial.println(" Abriendo...");
239         }
240         break;
241     case CAJA_ABIERTA:
242         Serial.println(" Caja abierta");
243         if (!abrirCaja) {
244             Serial.println(" Cerrando...");
245         }
246         break;
247     case EN_TRANSICION:
248         Serial.println(" Caja en transición...");
249         if (abrirCaja) {
250             Serial.println(" Abriendo...");
251         } else {
252             Serial.println(" Cerrando...");
253         }
254         break;
255     case ESTADO_INVALIDO:
256         Serial.println(" Estado de sensores inválido.");
257         break;
258 }
259 ultimoEstado = estadoActual;
260 }
261
262 // Control del motor, esto siempre se ejecuta
263 if (estadoActual == CAJA_CERRADA) {
264     if (abrirCaja) {
265         moverCaja(velocidadCaja);
266     } else {
267         detenerMotorCaja();
268     }
269 } else if (estadoActual == CAJA_ABIERTA) {
270     if (!abrirCaja) {
271         moverCaja(-velocidadCaja);
272     } else {
273         detenerMotorCaja();
274     }
275 } else if (estadoActual == EN_TRANSICION) {
276     // Aquí ahora sí actúa el motor según abrirCaja
277     if (abrirCaja) {
278         moverCaja(velocidadCaja);
279     } else {
280         moverCaja(-velocidadCaja);
281     }
282 } else {
```

```
283     detenerMotorCaja();
284 }
285 }
286
287
288 void moverCaja(int velocidad) {
289     float duty = map(abs(velocidad), 1, 100, 60, 100);
290     digitalWrite(gpioDir1, velocidad > 0);
291     digitalWrite(gpioDir2, velocidad < 0);
292     mcpwm_set_duty(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_A, duty);
293 }
294
295 void detenerMotorCaja() {
296     digitalWrite(gpioDir1, LOW);
297     digitalWrite(gpioDir2, LOW);
298     mcpwm_set_duty(MCPWM_UNIT_0, MCPWM_TIMER_0, MCPWM_OPR_A, 0);
299 }
300
301 // ----- MQTT Callback -----
302 void callback(char* topic, byte* payload, unsigned int length) {
303     String message;
304     for (int i = 0; i < length; i++) {
305         message += (char)payload[i];
306     }
307
308     Serial.print(" MQTT [");
309     Serial.print(topic);
310     Serial.print("]: ");
311     Serial.println(message);
312
313     if (String(topic) == topic_luz_nocturna) {
314         if (message == "encender") {
315             controlledLed9Activo = true;
316             actualizarLed9();
317             Serial.println(" Luz nocturna activada (azul o rojo según hora)");
318         } else if (message == "apagar") {
319             controlledLed9Activo = false;
320             pixels.setPixelColor(8, pixels.Color(0, 0, 0)); // Apaga LED 9
321             pixels.show();
322             Serial.println(" Luz nocturna desactivada");
323         }
324         return;
325     }
326
327
328     if (String(topic) == topic_motor2) {
329         procesarMotor(message, gpioDir3, gpioDir4, MCPWM_UNIT_1, MCPWM_TIMER_1, MCPWM_OPR_A);
330         return;
331     }
332 }
```

```
331     }
332
333     if (String(topic) == topic_motor) {
334         velocidadCaja = constrain(message.toInt(), 0, 100);
335         Serial.print(" Velocidad caja: ");
336         Serial.println(velocidadCaja);
337
338         if (velocidadCaja == 0) {
339             Serial.println(" Velocidad 0 - Deteniendo motor");
340             detenerMotorCaja();
341         }
342
343
344
345         return;
346     }
347
348     if (String(topic) == topic_estado_caja) {
349         abrirCaja = (message == "abrir");
350         Serial.print(" Acción recibida: ");
351         Serial.println(abrirCaja ? "ABRIR" : "CERRAR");
352         return;
353     }
354
355     StaticJsonDocument<256> doc;
356     DeserializationError error = deserializeJson(doc, message);
357     if (error) {
358         Serial.print(" Error JSON: ");
359         Serial.println(error.c_str());
360         return;
361     }
362
363     procesarMensajeJSON(topic, doc);
364 }
365
366 void procesarMotor(String message, int pinDir1, int pinDir2, mcpwm_unit_t unit, mcpwm_
367     int val = message.toInt();
368     if (val == 0) {
369         digitalWrite(pinDir1, LOW);
370         digitalWrite(pinDir2, LOW);
371         mcpwm_set_duty(unit, timer, opr, 0);
372         return;
373     }
374
375     float duty = map(abs(val), 1, 100, 60, 100);
376     digitalWrite(pinDir1, val > 0);
377     digitalWrite(pinDir2, val < 0);
378     mcpwm_set_duty(unit, timer, opr, duty);
```



```
379 }
380
381 // ----- JSON MQTT -----
382 void procesarMensajeJSON(char* topic, StaticJsonDocument<256>& doc) {
383     String state = doc["state"];
384
385     int r = doc["color"]["r"];
386     int g = doc["color"]["g"];
387     int b = doc["color"]["b"];
388
389     if (r < 0 || r > 255 || g < 0 || g > 255 || b < 0 || b > 255) return;
390
391     actualizarLed(topic, r, g, b);
392
393     // Reset por actividad manual desde Home Assistant
394     ultimoMovimiento = millis();
395     lucesApagadasPorInactividad = false;
396
397     Serial.println(" Actividad manual detectada: reiniciando temporizador");
398 }
399
400
401 void actualizarLed(const char* topic, int r, int g, int b) {
402     for (int i = 0; i < 9; i++) {
403         if (String(topic) == ledControlTopic[i]) {
404             pixels.setPixelColor(i, pixels.Color(r, g, b));
405             pixels.show();
406             return;
407         }
408     }
409 }
410
411
412 void actualizarLed9() {
413     if (estadoDiaNoche == "es de noche") {
414         pixels.setPixelColor(8, pixels.Color(0, 0, 255));
415     } else {
416         pixels.setPixelColor(8, pixels.Color(255, 0, 0));
417     }
418     pixels.show();
419 }
420
421 void apagarTodasLasLuces() {
422     for (int i = 0; i < 9; i++) {
423         actualizarLed(ledControlTopic[i], 0, 0, 0); // Apaga cada LED
424     }
425     Serial.println(" Todas las luces apagadas por inactividad");
426 }
```

```
427
428
429 void reconnect() {
430     while (!client.connected()) {
431         Serial.print("Conectando al servidor MQTT...");
432         if (client.connect("ESP32_Client", mqtt_username, mqtt_password)) {
433             Serial.println(" Conectado");
434             configurarMQTT(); // Re-suscribe tópicos
435         } else {
436             Serial.print(" Error: ");
437             Serial.print(client.state());
438             Serial.println(" Reintentando...");
439             delay(5000);
440         }
441     }
442 }
443
```

7. Interfaz de Usuario en Home Assistant (Lovelace)

Para la visualización y control del sistema domótico se utilizó la interfaz gráfica **Lovelace**, incluida por defecto en Home Assistant. En una etapa inicial se exploraron opciones adicionales desde el repositorio *HACS* (Home Assistant Community Store) para mejorar el frontend. Sin embargo, se optó por conservar la interfaz predeterminada de Home Assistant, debido a su diseño minimalista, intuitivo y suficientemente funcional para las necesidades del proyecto.

Una de las grandes ventajas de Lovelace es que detecta automáticamente las nuevas entidades definidas en los archivos de configuración YAML, siempre que el sistema se reinicie correctamente. Esto simplifica el proceso de incorporación de dispositivos y automatizaciones a la interfaz.

7.1. Configuración de Tarjetas de Entidades

Para estructurar la información y facilitar el control, se emplearon tarjetas del tipo **entities**, donde se agrupan dispositivos con funciones similares.

7.1.1. Luces RGB (LEDs)

```
1 type: entities
2 entities:
3   - light.led_1
4   - light.led_2
5   - light.led_3
6   - light.led_4
7   - light.led_5
8   - light.led_6
9   - light.led_7
```



```
10 - light.led_8
11 - light.led_9
```

7.1.2. Control de la Caja Electrónica

```
1 type: entities
2 entities:
3   - switch.caja_abrir
4   - number.velocidad_caja
```

7.1.3. Velocidad del Segundo Motor

```
1
2 type: entities
3 entities:
4   - number.velocidad_motor
```

7.1.4. Sensores Binarios

Sensor de Movimiento:

```
1 type: entities
2 entities:
3   - binary_sensor.sensor_de_movimiento
```

Nivel de Luz Ambiente:

```
1 type: entities
2 entities:
3   - binary_sensor.nivel_de_luz_ambiente
```

7.1.5. Sensores Analógicos

Sensor de Luz Ambiente:

```
1 type: entities
2 entities:
3   - sensor.luz_ambiente
```

7.1.6. Automatizaciones

Estado Día/Noche vía MQTT:

```
1 type: entities
2 entities:
3   - automation.enviar_estado_de_dia_o_noche_por_mqtt_cada_5_minutos
```

7.2. Resumen de Diseño

La configuración se centró en una interfaz simple, funcional y clara, priorizando la agrupación lógica de entidades. Se evitó el uso de tarjetas personalizadas innecesarias para mantener la estabilidad y simplicidad del sistema. Esto también asegura que el sistema sea fácilmente comprensible por otros usuarios o técnicos que necesiten intervenir en el futuro.

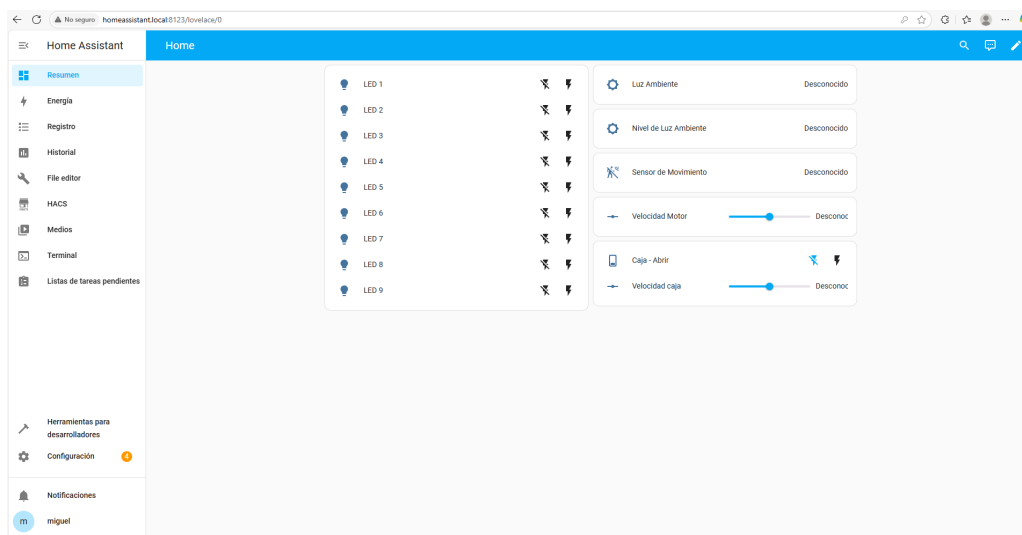


Figura 6: Captura de pantalla de la interfaz Lovelace configurada

8. Hardware

8.1. Raspberry Pi 3

Para este proyecto de domótica, se ha elegido la **Raspberry Pi 3** como unidad central de procesamiento. Esta decisión responde a varios criterios clave:

- **Costo reducido:** Comparada con otros miniordenadores o microcontroladores más avanzados, la Raspberry Pi 3 ofrece un equilibrio ideal entre funcionalidad y precio, situándose por debajo de los 60 euros incluyendo fuente de alimentación y carcasa.
- **Suficiencia de recursos:** Con su procesador quad-core ARM Cortex-A53 y 1 GB de RAM, es más que capaz de ejecutar servicios de domótica como Home Assistant, además de manejar múltiples sensores y actuadores en paralelo sin problemas de rendimiento.
- **Compatibilidad y comunidad:** La Raspberry Pi 3 cuenta con un amplio soporte de software y una comunidad activa. Esto facilita la instalación de sistemas operativos optimizados (como Raspberry Pi OS o distribuciones específicas para domótica como Home Assistant OS), así como la resolución de problemas.
- **Conectividad integrada:** Dispone de conectividad Wi-Fi y Bluetooth incorporadas, eliminando la necesidad de módulos externos y simplificando la comunicación con sensores, nodos IoT y dispositivos móviles.

En resumen, la Raspberry Pi 3 se presenta como una solución económica, accesible y suficientemente potente para cubrir los requerimientos del presente sistema de automatización del hogar.

8.2. ESP32

El microcontrolador **ESP32** ha sido elegido como componente clave para tareas distribuidas dentro del sistema domótico, especialmente como nodo periférico encargado de la lectura de sensores y el control de dispositivos remotos. Esta decisión se basa en sus múltiples ventajas:

- **Capacidad de cómputo destacable:** El ESP32 cuenta con un procesador dual-core a 240 MHz y 520 KB de SRAM, lo cual le permite ejecutar múltiples tareas en paralelo, desde la adquisición de datos hasta su procesamiento y transmisión. Esto lo convierte en una plataforma potente para microcontrolador, muy superior a alternativas como el ESP8266.
- **Conectividad a Internet:** Gracias a su módulo Wi-Fi integrado, el ESP32 puede conectarse directamente a una red local o a Internet, permitiendo la comunicación constante con la Raspberry Pi o servidores en la nube. Esta característica lo hace ideal para aplicaciones de Internet de las Cosas (IoT), donde la comunicación remota es esencial.
- **Soporte para protocolos IoT:** El ESP32 es compatible con protocolos como MQTT, ampliamente utilizado en domótica por su eficiencia y estructura basada en *publicación/suscripción*. Esto le permite suscribirse a múltiples tópicos y reaccionar en tiempo real ante mensajes recibidos, facilitando una arquitectura descentralizada y reactiva.
- **Bajo coste y consumo:** Por un precio inferior a 4 euros y con modos de ahorro energético avanzados, el ESP32 permite desplegar múltiples nodos inalámbricos sin comprometer ni el presupuesto ni la eficiencia energética del sistema.
- **Flexibilidad en conexiones:** Dispone de numerosos pines para entradas y salidas digitales, analógicas, PWM, y buses de comunicación como I2C, SPI o UART, lo que permite conectarle fácilmente sensores, actuadores y otros periféricos.

En conjunto, el ESP32 proporciona una solución robusta, económica y flexible para la implementación de nodos IoT dentro de un sistema domótico distribuido.

8.3. Módulo puente H L298N

En el diseño del sistema se ha seleccionado el módulo **puente H L298N** para el control de motores de corriente continua. Aunque existen alternativas más simples y económicas como el **L293D**, el **L298N** representa una solución más robusta y adecuada para aplicaciones que demandan una mayor potencia, incluso por encima de la que se prevé utilizar en este proyecto.

- **Control eficiente mediante PWM:** El L298N permite el control tanto de la velocidad como del sentido de giro de los motores mediante señales PWM, lo que ofrece un control fino y eficiente con una implementación sencilla desde microcontroladores como el ESP32 o la Raspberry Pi.

- **Versatilidad y flexibilidad:** A diferencia de un controlador de motor paso a paso —opción inicialmente considerada por su capacidad para controlar con precisión el número de vueltas y la posición— el uso del L298N con motores DC proporciona mayor flexibilidad, menor coste y menor complejidad en el control.
- **Robustez:** El L298N está diseñado para manejar cargas de mayor corriente, lo que garantiza mayor durabilidad y menor riesgo de fallos ante picos de consumo. Esto lo hace adecuado tanto para motores pequeños como para otros más potentes si se decidiera escalar el sistema en el futuro.
- **Alternativa de control de posición:** Aunque los motores DC no permiten un control de posición directo como los paso a paso, este inconveniente se compensa mediante el uso de dos sensores magnéticos que permiten determinar la posición de forma efectiva, fiable y económica.

Por todo lo anterior, el módulo L298N ofrece un equilibrio entre coste, potencia y control, convirtiéndose en la opción más adecuada para el sistema motorizado del proyecto.

8.4. Sensor de luz TEMT6000

En una primera etapa del diseño del sistema domótico, se incorporó el **sensor de luz TEMT6000** como una solución sencilla y económica para automatizar el encendido y apagado de las luces en función de la luminosidad ambiental.

- **Motivación inicial:** El objetivo era que el sistema fuese capaz de detectar condiciones de baja iluminación (por ejemplo, al anochecer) y activar las luces de forma autónoma, sin necesidad de intervención del usuario ni programación horaria fija.
- **Funcionamiento:** El TEMT6000 es un fototransistor que genera una señal analógica proporcional a la luz incidente, lo que permite determinar con precisión los niveles de luminosidad del entorno.
- **Limitación en el entorno de pruebas:** Aunque el planteamiento inicial era prometedor, se encontró una limitación significativa: la maqueta del sistema está situada permanentemente en una habitación interior, sin exposición directa a la luz natural. Esto redujo drásticamente la utilidad del sensor, que no reflejaba de forma realista las condiciones lumínicas exteriores.
- **Solución alternativa:** Para superar esta limitación, se optó por sustituir parcialmente la función del TEMT6000 mediante el uso de una **API meteorológica**, capaz de consultar información sobre la hora de salida y puesta del sol en tiempo real, y adaptar el comportamiento del sistema de iluminación en función de estos datos.

Aunque el TEMT6000 no se utilizó de forma activa en la versión final, su incorporación inicial demostró una aproximación válida y su uso podría retomarse en instalaciones reales con acceso a luz exterior.

8.5. Sensor magnético KY-025

Para conocer el estado de apertura o cierre de una caja dentro del sistema domótico, se ha optado por emplear el **sensor magnético KY-025** en lugar de otros mecanismos más tradicionales como botones de contacto o finales de carrera.

- **Necesidad del sistema:** Era fundamental incorporar un mecanismo que permitiera detectar si la caja estaba abierta o cerrada, con el fin de activar determinadas funciones (como alarmas, registro de eventos o desactivación del sistema) en función de su estado.
- **Solución inicial:** En un primer planteamiento, se consideró el uso de pulsadores o interruptores mecánicos que cambiaran de estado al contacto con la tapa de la caja. Sin embargo, este enfoque presentaba problemas de fiabilidad y desgaste por fricción, así como una mayor complejidad en el montaje físico.
- **Ventajas del KY-025:** El sensor magnético KY-025, que funciona en conjunto con un imán, permite detectar sin contacto físico la presencia o ausencia del imán. Esto simplifica notablemente la instalación, elimina desgaste mecánico y mejora la fiabilidad del sistema a largo plazo.
- **Funcionamiento en el proyecto:** Al colocar un imán en la tapa de la caja y el sensor KY-025 en la base, se puede determinar si la caja está cerrada (el sensor detecta el campo magnético) o abierta (el imán se aleja). Este sistema no requiere presión, alineación precisa ni mantenimiento, y es más tolerante a pequeños desplazamientos o vibraciones.
- **Relación con otros elementos:** Además, este tipo de sensor se ha integrado con éxito en el sistema de detección de posición del motor, ya que permite definir estados lógicos (posición 1 o posición 2) sin necesidad de un motor paso a paso, lo que reduce costes y complejidad.

Gracias a estas ventajas, el KY-025 se ha consolidado como una solución eficaz, económica y robusta para la detección de estados físicos en el proyecto.

8.6. Elección del motor DC con piñón-cremallera

Para el mecanismo de apertura y cierre de una tapa dentro del sistema, se ha optado por un conjunto sencillo formado por un **motor DC acoplado a un sistema de piñón y cremallera**, ambos fabricados en plástico y adquiridos como un pack económico. Esta elección responde a criterios de simplicidad, funcionalidad y facilidad de integración.

- **Objetivo funcional:** El sistema únicamente necesita realizar un movimiento de apertura y cierre, sin necesidad de un control preciso del ángulo, velocidad variable compleja o esfuerzos mecánicos elevados. Por tanto, se descartaron soluciones más complejas por ser innecesarias.
- **Alternativas evaluadas:**
 - *Servomotores:* Aunque permiten un control preciso del ángulo, requieren un mayor control electrónico y suelen ser más costosos.

- *Motores paso a paso*: Fueron considerados inicialmente por su capacidad de controlar con precisión el número de pasos y, por tanto, la posición. Sin embargo, resultan innecesarios para una tarea binaria como abrir o cerrar, y requieren una electrónica de control más compleja.
- *Sistemas con bisagras automatizadas o actuadores lineales*: También se valoraron, pero se descartaron por su coste, volumen y la necesidad de piezas adicionales.

■ **Ventajas del sistema piñón-cremallera:**

- Su mecánica simple permite convertir el movimiento rotativo del motor en uno lineal con facilidad.
- No requiere el uso de piezas impresas en 3D, lo cual es una gran ventaja práctica. La impresión 3D de elementos pequeños puede presentar problemas de precisión, ajuste o resistencia mecánica, lo cual se evita completamente con esta solución comercial.
- El montaje es directo y funcional, reduciendo el tiempo de desarrollo y minimizando los puntos de fallo mecánico.

- **Complemento con sensores:** El sistema se integra perfectamente con los sensores magnéticos KY-025, que permiten detectar la posición final de la tapa (abierta o cerrada), sin necesidad de retroalimentación por parte del motor.

Esta solución representa un equilibrio ideal entre simplicidad mecánica, fiabilidad y coste, cumpliendo con creces los requerimientos funcionales del proyecto.

8.7. LEDs NeoPixel

Para este proyecto se ha optado por utilizar LEDs del tipo *NeoPixel* (basados en el chip WS2812), los cuales ofrecen ventajas significativas en comparación con LEDs tradicionales o matrices de control más complejas.

Uno de los principales motivos para su elección es que cada LED es direccionable individualmente y todos pueden ser controlados en cadena a través de un único pin digital del microcontrolador. Esto simplifica considerablemente el cableado, reduciendo el número de pines necesarios y dejando libres otros recursos del microcontrolador para otras tareas.

Además, los NeoPixel permiten definir el color y el brillo de cada LED mediante simples estructuras de datos y bucles iterativos, lo que facilita la implementación del código de control. Su compatibilidad con bibliotecas como `Adafruit_NeoPixel` en plataformas como ESP32 hace que su uso sea accesible y estable.

Otro aspecto clave es su escalabilidad: no existe una limitación práctica estricta sobre la cantidad de LEDs que se pueden utilizar, más allá del consumo eléctrico y la memoria del microcontrolador. Esto significa que el sistema puede crecer fácilmente en número de LEDs sin requerir rediseño del hardware o de la arquitectura de control.

Por estas razones —facilidad de conexión, control eficiente mediante código, y escalabilidad— los NeoPixel han resultado una opción ideal para el sistema de iluminación RGB implementado en la maqueta domótica del proyecto.

9. Presupuesto

A continuación se detalla el presupuesto estimado para el desarrollo del Trabajo Fin de Grado (TFG), incluyendo los componentes electrónicos necesarios junto con su precio y enlace de compra:

Componente	Precio (€)	Enlace
Raspberry Pi 3 + fuente + caja	54.72	https://es.aliexpress.com/item/1005007655721767.html
ESP32	3.79	https://es.aliexpress.com/item/1005006347872239.html
Tarjeta SD 32GB Clase 10	7.94	https://www.pccomponentes.com/
Puente H L298N	2.96	https://es.aliexpress.com/item/1005008756464145.html
Detector de luz TEMENT6000	0.52	https://es.aliexpress.com/item/1005001565357867.html
Imanes de neodimio	1.65	https://es.aliexpress.com/item/1005008955502063.html
2x Detector imán KY-025	1.02	https://es.aliexpress.com/item/1005008685008619.html
Fuente alimentación 5V	3.49	https://es.aliexpress.com/item/1005007805634378.html
Motor DC piñón-cremallera	3.29	https://es.aliexpress.com/item/1005007676181805.html
Matriz NeoPixel 8x8 (64 LEDs)	47.85	https://ultra-lab.net/producto/neopixel-neomatrix-8x8-64-led-rgb/ (0.75€/LED)
Total	126.33	

Cuadro 1: Presupuesto detallado del TFG

10. Bibliografía Técnica de Componentes

Componente	Ficha Técnica / Fuente
ESP32-WROOM-32	https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf
Raspberry Pi 3 Model B+	https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf
LEDs NeoPixel WS2812B	https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf
Sensor Magnético KY-025	https://datasheet4u.com/datasheet/Joy-IT/KY-025-1402036
Sensor de Luz TEMT6000	https://www.vishay.com/docs/81579/temt6000.pdf
Puente H L298N	https://www.st.com/resource/en/datasheet/l298.pdf

Cuadro 2: Referencias técnicas utilizadas en el diseño del sistema

11. Bibliografía de Funcionalidades de Home Assistant

Funcionalidad / Integración	Referencia oficial / documentación
Home Assistant (plataforma principal)	https://www.home-assistant.io/docs/
Configuración del archivo <code>configuration.yaml</code>	https://www.home-assistant.io/docs/configuration/basic/
Integración MQTT	https://www.home-assistant.io/integrations/mqtt/
Broker MQTT Mosquitto (Add-on)	https://www.home-assistant.io/addons/mosquitto/
Automatizaciones	https://www.home-assistant.io/docs/automation/
Scripts	https://www.home-assistant.io/docs/scripts/
Sensores personalizados MQTT	https://www.home-assistant.io/integrations/sensor.mqtt/
Actuadores y switches MQTT	https://www.home-assistant.io/integrations/switch.mqtt/
Iluminación RGB con MQTT	https://www.home-assistant.io/integrations/light.mqtt/
Configuración de dashboards (Lovelace UI)	https://www.home-assistant.io/lovelace/
File Editor (Add-on para editar archivos YAML)	https://github.com/home-assistant/addons/tree/master/file_editor
Terminal y SSH Add-on	https://www.home-assistant.io/addons/ssh/

Cuadro 3: Referencias funcionales utilizadas en el desarrollo con Home Assistant