



UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y
DISEÑO INDUSTRIAL

Grado en Ingeniería Electrónica y Automática Industrial

TRABAJO FIN DE GRADO

**DISEÑO DE SISTEMA DOMÓTICO DE
RIEGO CONTROLADO POR
COMUNICACIONES MÓVILES (4G)**

Carlos Osuna Cabezudo

Tutor: Alberto Brunete González

Departamento: Ingeniería Eléctrica, Electrónica, Automática y Física
Aplicada

Madrid, Septiembre, 2025



UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y
DISEÑO INDUSTRIAL

Grado en Ingeniería Electrónica y Automática Industrial

TRABAJO FIN DE GRADO

**DISEÑO DE SISTEMA DOMÓTICO DE
RIEGO CONTROLADO POR
COMUNICACIONES MÓVILES (4G)**

Carlos Osuna Cabezudo

Firma Tutor

Copyright ©2025. Carlos Osuna Cabezudo

Esta obra está licenciada bajo la licencia Creative Commons Atribución-NoComercial-SinDerivadas 3.0 Unported (CC BY-NC-ND 3.0). Para ver una copia de esta licencia, visite

<http://creativecommons.org/licenses/by-nc-nd/3.0/deed.es> o envíe una carta a Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, EE.UU.

Todas las opiniones aquí expresadas son del autor, y no reflejan necesariamente las opiniones de la Universidad Politécnica de Madrid.

Título: Diseño de sistema domótico de riego controlado por comunicaciones móviles (4G)

Autor: Carlos Osuna Cabezudo

Tutor: Alberto Brunete González

Cotutor: nombre del cotutor

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día de de ... en, en la Escuela Técnica Superior de Ingeniería y Diseño Industrial de la Universidad Politécnica de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO

PRESIDENTE

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor, Alberto Brunete González, por su guía, paciencia y apoyo constante durante el desarrollo de este trabajo fin de grado. Sus orientaciones han sido clave para afrontar con seguridad cada etapa del proyecto.

Asimismo, agradezco a mi familia por su confianza y respaldo incondicional, y a mis amigos, tanto por acompañarme en este proceso como por prestarme material para la construcción de la maqueta, sin cuya ayuda no hubiera sido posible culminar este trabajo.

Finalmente, quiero dar las gracias a la Universidad Politécnica de Madrid y a la Escuela Técnica Superior de Ingeniería y Diseño Industrial por brindarme la oportunidad de formarme y por facilitar los recursos necesarios para el desarrollo de este proyecto. Igualmente, a todos los compañeros que me han acompañado durante estos cuatro años por su cercanía y disposición para ayudarnos mutuamente.

Muchas gracias a todos,

Carlos Osuna Cabezudo

Resumen

Este proyecto se resume en el desarrollo de un sistema domótico para el control de un invernadero inteligente. Para ello, se combina un microcontrolador STM32F411E-Discovery con sensores de temperatura, humedad y luminosidad (SHT85 y VEML7700), junto a un módulo WiFi **ESP8266** para la comunicación inalámbrica.

Los datos se transmiten mediante el protocolo MQTT a una Raspberry Pi que actúa como servidor, donde se procesan y se muestran en la plataforma Home Assistant. De forma paralela, el microcontrolador envía los datos directamente a ThingSpeak a través del módulo ESP8266, permitiendo su almacenamiento en la nube.

El sistema implementa control remoto de actuadores tanto en modo manual como automático, garantizando independencia entre la toma de medidas y el manejo de los actuadores. La automatización en modo automático se realiza mediante reglas basadas en umbrales definidos por el usuario, lo que permite mantener unas condiciones ambientales estables sin intervención manual.

Las principales aportaciones de este trabajo son:

- El diseño de una arquitectura híbrida, con doble envío de datos (local y nube), que asegura redundancia y flexibilidad.
- La implementación de máquinas de estados robustas para la comunicación WiFi y MQTT, evitando bloqueos a largo plazo y garantizando fiabilidad en la transmisión.
- La integración de la supervisión local mediante Home Assistant, que habilita control en tiempo real.
- La posibilidad de control de actuadores en dos modos operativos: manual, con órdenes directas del usuario, y automático, con reglas por umbrales configurables.
- Un diseño modular y escalable, que facilita la incorporación de nuevos sensores o actuadores en el futuro.

El proyecto demuestra la viabilidad de un sistema de monitorización y control de bajo coste, basado en tecnologías IoT abiertas y fácilmente adaptables a distintos escenarios domóticos.

Palabras clave: Invernadero domótico, STM32, ESP8266, MQTT, Home Assistant, IoT.

Abstract

This project summarizes the development of a home automation system for the control of an intelligent greenhouse. For this purpose, an STM32F411E-Discovery microcontroller is combined with temperature, humidity, and luminosity sensors (SHT85 and VEML7700), along with an ESP8266 WiFi module for wireless communication.

The data are transmitted via the MQTT protocol to a Raspberry Pi acting as a server, where they are processed and displayed on the Home Assistant platform. In parallel, the microcontroller sends the data directly to ThingSpeak through the ESP8266 module, allowing cloud storage.

The system implements remote control of actuators in both manual and automatic modes, ensuring independence between data acquisition and actuator management. Automatic operation is carried out through rules based on user-defined thresholds, allowing stable environmental conditions without manual intervention.

The main contributions of this work are:

- The design of a hybrid architecture, with dual data transmission (local and cloud), ensuring redundancy and flexibility.
- The implementation of robust state machines for WiFi and MQTT communication, preventing long-term lockups and ensuring transmission reliability.
- The integration of local supervision through Home Assistant, enabling real-time control.
- The possibility of actuator control in two operating modes: manual, with direct user commands, and automatic, with configurable threshold rules.
- A modular and scalable design, allowing the easy incorporation of additional sensors or actuators in the future.

The project demonstrates the feasibility of a low-cost monitoring and control system, based on open IoT technologies that are easily adaptable to different home automation scenarios.

Keywords: Smart greenhouse, STM32, ESP8266, MQTT, Home Assistant, IoT.

Índice general

Agradecimientos	vii
Resumen	ix
Abstract	xi
Índice	xvi
1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Objetivos	1
1.3. Alcance	2
1.4. Materiales utilizados	3
1.5. Estructura del documento	4
2. Marco Teórico	7
2.1. Microcontroladores y sistemas embebidos	7
2.2. Comunicaciones inalámbricas	8
2.2.1. Wi-Fi y módulos ESP8266	8
2.3. Protocolos de comunicación	9
2.3.1. I ² C	9
2.3.2. UART	9
2.3.3. MQTT	9
2.4. Plataformas y herramientas software	10
2.4.1. STM32CubeIDE	10
2.4.2. ThingSpeak	10
2.4.3. Home Assistant	10
2.4.4. Eagle	10
2.5. Sensores empleados	11
2.5.1. Sensor SHT85	11
2.5.2. Sensor VEML7700	11
2.6. Actuadores en sistemas de riego	12
2.6.1. Ventilador	12
2.6.2. Bomba de agua	13
2.6.3. Foco LED	13

3. Estado del arte	15
3.1. Domótica	15
3.2. Aplicaciones en la agricultura	16
3.3. Aplicaciones IoT	17
3.4. Sensores	18
3.5. Protocolos de comunicación	18
4. Requisitos y diseño del sistema	21
4.1. Arquitectura del sistema	21
4.1.1. Esquema general del sistema	21
4.1.2. Finalidad del esquema	21
4.1.3. Descripción de la arquitectura	22
4.1.4. Relación entre nodos	23
4.2. Resumen general del sistema	23
4.3. Descripción de los topics MQTT	23
5. Sensores y actuadores	25
5.1. Sensores	25
5.1.1. Sensor de temperatura y humedad SHT85	25
5.1.2. Sensor de luminosidad VEML7700	27
5.1.3. Asignación de pines	28
5.2. Actuadores	29
5.2.1. Hardware	29
5.2.2. Control de actuadores	29
5.2.3. Interfaz de potencia: relés y seguridad	31
5.3. Periféricos	31
5.3.1. Display LCD	31
6. Comunicaciones en STM32	33
6.1. ThingSpeak: almacenamiento en la nube	33
6.1.1. Protocolo de comunicación	33
6.1.2. Secuencia de comandos AT	33
6.1.3. Máquina de estados para ThingSpeak	34
6.1.4. Visualización en la nube	35
6.2. Broker MQTT mediante ESP8266	35
6.2.1. Protocolo de comunicación	35
6.2.2. Publicación de mensajes en formato JSON	37
6.2.3. Máquina de estados para MQTT broker	37
6.2.4. Ventajas del formato unificado JSON	38
6.2.5. Visualización en Home Assistant	38
7. Raspberry Pi	39
7.1. Configuración de la Raspberry Pi	39
7.1.1. Instalación del sistema operativo Raspberry Pi OS Lite	39
7.1.2. Instalación del broker MQTT	40
7.1.3. Instalación de Docker	40
7.1.4. Acceso remoto mediante SSH	42
7.1.5. Instalación de Home Assistant mediante Docker Compose	42
7.1.6. Asignación de IP fija en la Raspberry Pi	43

7.2.	Integración de MQTT con Home Assistant	43
7.2.1.	Instalación y configuración del broker MQTT en Raspberry Pi	44
7.2.2.	Conexión de Home Assistant al broker MQTT	45
7.3.	Código <code>configuration.yaml</code>	46
7.3.1.	Sensores MQTT (<code>sensor</code>)	47
7.3.2.	Actuadores y control (<code>switch</code> , <code>select</code>)	48
7.3.3.	Parámetros ajustables (<code>number</code>)	49
7.3.4.	Sensores lógicos (<code>binary_sensor</code>)	50
7.3.5.	Notas de validación y reinicio	51
7.4.	Código <code>automations.yaml</code>	51
7.4.1.	Publicación de estado de modo automático	52
7.4.2.	Reglas de control por umbrales	52
7.4.3.	Eventos y condiciones auxiliares	55
7.5.	Código de control de actuadores con <i>script</i> Python	55
7.5.1.	Descripción del servicio y dependencias	56
7.5.2.	Estructura del código (<code>actuadores_mqtt.py</code>)	56
7.5.3.	Código completo del <i>script</i> Python	57
7.5.4.	Servicio de arranque en <i>systemd</i>	59
7.6.	Códigos de interfaz en Home Assistant	60
7.6.1.	Gráficas y datos actuales	60
7.6.2.	Activación de actuadores	61
7.6.3.	Modo automático	62
7.6.4.	Cámara del invernadero	63
7.6.5.	Pronóstico meteorológico	64
8.	Videovigilancia	67
8.1.	Instalación y configuración de cámara USB en Raspberry Pi	67
8.1.1.	Instalación del software	67
8.1.2.	Configuración de <code>motion</code>	67
8.1.3.	Ejecución del servicio	68
8.1.4.	Acceso al streaming desde la red local	68
8.2.	Integración de la cámara en Home Assistant	68
8.2.1.	Añadir la cámara como dispositivo MJPEG	70
8.2.2.	Visualización en el panel principal	70
8.3.	Inicio automático de la retransmisión de la cámara	72
8.3.1.	Creación del servicio <code>motion</code>	72
8.3.2.	Habilitación del servicio en el arranque	72
8.3.3.	Modificación del archivo de configuración	73
8.3.4.	Resultado	73
9.	Cableado del sistema	75
9.1.	Alimentación del sistema	75
9.2.	Criterios de diseño	75
9.2.1.	Cargadores Xiaomi	75
9.2.2.	Unificación de masas	76
9.2.3.	Raspberry Pi y actuadores	76
9.3.	Relés de control	76
9.4.	Montaje del sistema	77

9.5. Planos eléctricos en Eagle	81
10.Pruebas y resultados	87
10.1. Pruebas	87
10.2. Resultados	88
10.3. Dificultades	89
10.4. Discusión	91
10.5. Discusión	91
11.Gestión del proyecto	93
11.1. Ciclo de vida	93
11.2. Planificación	94
11.2.1. Planificación inicial	94
11.2.2. Planificación final	94
11.3. Presupuesto	95
11.3.1. Personal	95
11.3.2. Material	95
11.3.3. Resumen de costes	95
12.Conclusiones	97
12.1. Conclusión	97
12.2. Desarrollos futuros	97
A. Vídeo demostrativo del proyecto	99
B. Repositorio de código en GitHub	101
C. Anexo: Tabla de pines	103
C.1. Introducción	103
C.1.1. Asignación de pines y funciones	103
C.1.2. Configuración de pines en STM32CubeIDE	103
Bibliografía	105

Índice de figuras

3.1. Ejemplo de sistema domótico con integración de dispositivos conectados [23].	16
4.1. Arquitectura del sistema domótico de invernadero.	21
5.1. Sensor de temperatura y humedad SHT85	25
5.2. Diagrama de flujo de lectura, validación y reinicio del sensor SHT85.	26
5.3. Sensor de luminosidad VEML7700	27
5.4. Diagrama de flujo de lectura, validación y reinicio del sensor VEML7700.	28
5.5. Ventilador para regulación térmica y circulación de aire.	29
5.6. Foco LED controlado por relé para iluminación suplementaria.	30
5.7. Bomba de agua para riego automatizado del invernadero.	30
5.8. Módulo Keyestudio RPI 4-Relay Shield (KS0212) conectado a la Raspberry Pi 4B como interfaz de potencia.	31
5.9. Visualización de las magnitudes medidas a través del Display LCD.	32
6.1. Diagrama de estados del envío a ThingSpeak.	35
6.2. Gráficas en ThingSpeak de temperatura, humedad y luminosidad.	36
6.3. Indicadores y alarmas configurados en ThingSpeak.	36
6.4. Diagrama de estados del envío de datos por MQTT.	38
7.1. Monitorización de recursos en la Raspberry Pi utilizando bpytop.	39
7.2. Estado del servicio Mosquitto activo en Raspberry Pi.	40
7.3. Arquitectura de servicios desplegada en la Raspberry Pi (Broker MQTT fuera de Docker).	41
7.4. Esquema de la arquitectura: acceso remoto por SSH.	42
7.5. Configuración de direcciones IP dinámicas (DHCP) en el router.	44
7.6. Configuración de la integración MQTT en Home Assistant.	46
7.7. Visualización de las gráficas desde la interfaz web de Home Assistant.	61
7.8. Visualización de las gráficas desde la aplicación móvil de Home Assistant.	61
7.9. Visualización de los datos medidos en tiempo real.	61
7.10. Visualización de los botones para el control de los actuadores.	65
7.11. Visualización del bloque de configuración del modo automático.	65
7.12. Cámara del invernadero en tiempo real.	65
7.13. Pronóstico meteorológico integrado.	65
8.1. Webcam HP 320 FHD conectada a la Raspberry Pi para capturar vídeo en directo.	69
8.2. Visualización web de la cámara HP 320 FHD.	69

8.3.	Formulario de configuración de cámara MJPEG en Home Assistant. . .	70
8.4.	Configuración de tarjeta Lovelace con la cámara MJPEG.	71
8.5.	Vista final del panel con la cámara integrada.	71
8.6.	Definición del servicio <code>motion.service</code>	72
8.7.	Servicio <code>motion</code> ejecutándose correctamente al arranque del sistema. .	73
9.1.	Cargadores Xiaomi empleados en el sistema (arriba: modelo MDY-09- EW, abajo: modelo MDY-11-EZ).	76
9.2.	Módulo Keyestudio RPI 4-Relay Shield (KS0212) unido a Raspberry Pi 4B.	77
9.3.	Proceso de corte inicial de la madera con sierra de calar.	78
9.4.	Proceso de corte circular en la madera con la sierra de calar.	78
9.5.	Detalle del corte circular en la madera con la sierra de calar.	78
9.6.	Colocación inicial del ventilador en el panel lateral.	79
9.7.	Vista lateral del panel con el ventilador ya fijado.	79
9.8.	Estructura con ventiladores montados y bandeja metálica instalada. .	80
9.9.	Vista trasera del sistema con el cableado organizado y protegido. . . .	80
9.10.	Panel frontal con la fuente de alimentación, pantalla LCD, poten- ciómetro, sensores y ventiladores ya instalados.	84
9.11.	Proceso de soldadura de los cables del sistema.	84
9.12.	Resultado final del montaje con el cableado organizado en regletas. .	85
9.13.	Diagrama de potencia con foco (230VAC), ventilador (12VDC) y bomba (5VDC).	85
10.1.	Gráfica de temperatura en ThingSpeak con pérdidas de datos.	89
11.1.	Diagrama de flujo del ciclo de vida del proyecto.	94
11.2.	Diagrama de Gantt del proyecto (Febrero–Septiembre).	95
C.1.	Configuración de pines en STM32CubeIDE	104

Índice de tablas

4.1. Resumen funcional y arquitectónico del sistema	24
4.2. Resumen de <code>state_topic</code> y <code>command_topic</code> empleados en el sistema.	24
9.1. Distribución de alimentación en la regleta de cuatro tomas.	75
11.1. Coste de personal estimado para el proyecto.	95
11.2. Presupuesto de hardware del proyecto.	96
11.3. Coste total del proyecto.	96
C.1. Asignación de pines del STM32F411 para sensores, comunicación y periféricos.	103

Capítulo 1

Introducción

1.1. Motivación del proyecto

El proyecto surge de la necesidad de implementar soluciones domóticas aplicadas al sector agrícola, concretamente al control de invernaderos. La optimización del consumo de agua y energía es un reto fundamental en un contexto de cambio climático y escasez de recursos.

El desarrollo de un sistema basado en microcontroladores y comunicaciones inalámbricas permite no solo automatizar el riego, la ventilación y la iluminación, sino también monitorizar en tiempo real variables críticas como la temperatura, la humedad y la luminosidad.

La motivación principal radica en diseñar una solución económica, escalable y robusta que pueda ser aplicada tanto en entornos de investigación como en explotaciones agrícolas de pequeña y mediana escala, contribuyendo a mejorar la eficiencia y sostenibilidad de los cultivos bajo invernadero.

1.2. Objetivos

El objetivo principal del presente Trabajo Fin de Grado es diseñar e implementar un sistema domótico de riego inteligente, con capacidad de comunicación remota y supervisión local, que permita la monitorización en tiempo real de variables ambientales y el control de actuadores desde una interfaz accesible en dispositivos móviles. El sistema se fundamenta en un microcontrolador STM32F411E-Discovery y se apoya en tecnologías IoT para la adquisición, transmisión y visualización de datos.

Durante el desarrollo del proyecto, se han definido los siguientes objetivos específicos:

- Implementar la lectura digital de sensores ambientales (SHT85 para temperatura y humedad, y VEML7700 para luminosidad) mediante comunicación I²C.
- Transmitir periódicamente los datos recogidos a la plataforma en la nube ThingSpeak para su almacenamiento y representación gráfica.
- Enviar simultáneamente los datos en formato JSON a un broker MQTT local (instalado en la Raspberry Pi) para su integración con Home Assistant.

- Permitir el control remoto de actuadores (foco 230 VAC, bomba de agua 5 VDC y ventiladores 12 VDC) desde Home Assistant, gestionados por la Raspberry Pi mediante un script en Python y un módulo de relés.
- Desarrollar una máquina de estados robusta que gestione las comunicaciones con ThingSpeak y el broker MQTT.
- Implementar sistemas de detección de errores y reconexión automática del módulo WiFi ESP8266, incluyendo control por tiempo de espera (timeout) y reinicios forzados periódicos.
- Integrar una webcam conectada a la Raspberry Pi para la supervisión visual en tiempo real del invernadero.
- Incorporar una pantalla LCD de 16x2 para mostrar en tiempo real los valores medidos por los sensores.

Estos objetivos se han ido adaptando progresivamente según los retos técnicos surgidos, manteniendo como prioridad la funcionalidad, la fiabilidad y la autonomía del sistema en un entorno real de uso agrícola.

1.3. Alcance

El presente Trabajo Fin de Grado abarca el diseño e implementación de un sistema de riego inteligente basado en tecnologías IoT y comunicación remota. El proyecto integra tanto la adquisición de datos ambientales como el control de actuadores, asegurando una gestión eficiente y autónoma del invernadero. Concretamente, el sistema desarrollado incluye:

1. Un sistema de control central basado en el microcontrolador STM32F411E-Discovery.
2. Comunicación inalámbrica mediante el módulo WiFi ESP8266 a través de UART.
3. Lectura periódica de sensores de temperatura, humedad y luminosidad mediante el protocolo I²C.
4. Visualización local de datos en un display LCD de 16x2 caracteres.
5. Transmisión de datos al broker MQTT instalado en una Raspberry Pi para su integración con Home Assistant.
6. Envío simultáneo de información a la plataforma ThingSpeak mediante peticiones HTTP GET para almacenamiento y análisis en la nube.
7. Configuración de la Raspberry Pi como gateway local y servidor domótico con Home Assistant.
8. Supervisión y control desde la aplicación móvil de Home Assistant, con acceso tanto a las lecturas de sensores como al manejo de actuadores.
9. Activación de actuadores (foco de 230 VAC, bomba de agua de 5 VDC y ventiladores de 12 VDC) mediante relés gestionados desde la Raspberry Pi a través de un script en Python.

10. Incorporación de una webcam conectada a la Raspberry Pi para la supervisión visual en tiempo real del estado del invernadero.

Este alcance refleja el desarrollo de un sistema funcional, escalable y robusto, que combina la monitorización ambiental con el control remoto de actuadores, integrando herramientas de visualización locales y en la nube.

1.4. Materiales utilizados

En el desarrollo del sistema se han empleado distintos componentes de hardware y software, que se citan a continuación:

Hardware

- **STM32F411E-Discovery** [33].
- **Módulo WiFi ESP8266** [6].
- **Sensor de temperatura y humedad SHT85** [30].
- **Sensor de luminosidad VEML7700** [34].
- **Pantalla LCD 16x2** [10].
- **Potenciómetro**, utilizado para el ajuste del contraste de la pantalla LCD [1].
- **Raspberry Pi 4B 2Gb** [26].
- **Tarjeta microSD SanDisk 32GB** [29].
- **Carcasa oficial para Raspberry Pi 4** [24].
- **Fuente de alimentación oficial USB-C para Raspberry Pi 4** [25].
- **Módulo de relés Keyestudio 4 canales**, para control de cargas [20].
- **Fuente de alimentación conmutada 60W**, salida 5V/4A y 12V/3A [35].
- **Ventiladores 12V** [2].
- **Foco LED 10W** [12].
- **Bomba de agua 5V** [28].
- **Webcam HP 320 FHD** [18].
- **Cables Dupont 28AWG (M/M, M/F, F/F)**, empleados únicamente en la fase de pruebas [15].

Software

- **STM32CubeIDE** [32].
- **ThingSpeak (MATLAB IoT)** [22].
- **Home Assistant** [16].
- **Mosquitto MQTT Broker** [11].
- **WinSCP**, para la transferencia de archivos entre el PC y la Raspberry Pi [36].
- **Eagle**, para el diseño de los planos eléctricos [5].

1.5. Estructura del documento

A continuación, y para facilitar la lectura del documento, se detalla el contenido de cada capítulo:

- En el **Capítulo 1** se presenta la introducción al trabajo, incluyendo motivación, objetivos, alcance, materiales utilizados y la estructura del documento.
- En el **Capítulo 2** se desarrolla el marco teórico, repasando conceptos de micro-controladores, comunicaciones inalámbricas, protocolos, plataformas software, sensores y actuadores.
- En el **Capítulo 3** se expone el estado del arte, con aplicaciones de la domótica, la agricultura, el IoT, así como una revisión de sensores y protocolos de comunicación.
- En el **Capítulo 4** se definen los requisitos y el diseño del sistema, mostrando la arquitectura, la relación entre nodos y la descripción de los **topics** MQTT.
- En el **Capítulo 5** se describen los sensores y actuadores empleados, detallando hardware, control, interfaz de potencia y periféricos como el display LCD.
- En el **Capítulo 6** se abordan las comunicaciones en el STM32, tanto con ThingSpeak como con el broker MQTT mediante el ESP8266, incluyendo máquinas de estados y publicación de datos en formato JSON.
- En el **Capítulo 7** se detalla la configuración de la Raspberry Pi, la integración con Home Assistant, el código en YAML, los scripts en Python y la visualización de datos.
- En el **Capítulo 8** se desarrolla la parte de videovigilancia, desde la instalación de la cámara USB y el software **motion**, hasta su integración en Home Assistant y el arranque automático.
- En el **Capítulo 9** se explican los aspectos de cableado del sistema, alimentación, criterios de diseño, relés de control, montaje y planos eléctricos.
- En el **Capítulo 10** se recogen las pruebas y resultados obtenidos, así como las dificultades encontradas y la discusión correspondiente.
- En el **Capítulo 11** se describe la gestión del proyecto, con su ciclo de vida, planificación, estimación de costes y resumen de presupuesto.

- En el **Capítulo 12** se presentan las conclusiones y los posibles desarrollos futuros.
- Finalmente, se incluyen los **Anexos**, que abarcan el vídeo demostrativo del proyecto, el repositorio de código en GitHub y la tabla de pines.

Capítulo 2

Marco Teórico

El presente capítulo reúne los fundamentos necesarios para comprender el desarrollo del sistema de invernadero domótico. No se pretende realizar una revisión exhaustiva de toda la bibliografía existente, sino proporcionar una base conceptual que facilite la lectura y justifique las decisiones técnicas adoptadas a lo largo del trabajo.

La exposición se estructura en varias secciones: en primer lugar se introducen los microcontroladores y sistemas embebidos, destacando el papel del STM32F411 como núcleo del sistema. A continuación se abordan las comunicaciones inalámbricas, centrándose en la tecnología Wi-Fi y en el módulo ESP8266 como interfaz de conexión. Seguidamente se describen los protocolos de comunicación empleados (I²C, UART y MQTT), esenciales para la transmisión de datos entre los distintos elementos.

El capítulo continúa con las plataformas y herramientas software, entre las que se encuentran STM32CubeIDE para el desarrollo embebido, ThingSpeak como plataforma en la nube, Home Assistant como interfaz de usuario y Eagle para el diseño de los esquemas eléctricos. Posteriormente se presentan los sensores empleados (SHT85 y VEML7700), detallando su principio de funcionamiento y la lógica de control automático asociada. Finalmente, se analizan los actuadores (ventilador, bomba de agua y foco LED), responsables de mantener las condiciones del invernadero dentro de los parámetros definidos por el usuario.

De este modo, el marco teórico establece un hilo conductor que conecta los aspectos conceptuales con su aplicación práctica en el sistema desarrollado.

2.1. Microcontroladores y sistemas embebidos

Un *microcontrolador* es un circuito integrado que reúne en un solo chip un procesador, memoria y periféricos de entrada/salida. A diferencia de un ordenador personal, cuyo propósito es ejecutar aplicaciones de propósito general, los microcontroladores están diseñados para tareas específicas, normalmente en tiempo real, dentro de un sistema electrónico.

Estos dispositivos se emplean de forma masiva en la industria, la automoción y el Internet de las Cosas (IoT), ya que permiten integrar control, sensado y comunicación en un único componente de bajo coste y reducido consumo energético.

Los *sistemas embebidos* se definen como aquellos que integran hardware y software dedicados para realizar una función concreta dentro de un dispositivo mayor (por ejemplo, el control de un invernadero inteligente o la gestión de un sensor en una red

IoT). En este tipo de sistemas la eficiencia y la fiabilidad son más importantes que la capacidad de cálculo general.

En el contexto de este proyecto se ha utilizado la placa **STM32F411E-Discovery**, basada en el microcontrolador *STM32F411* de STMicroelectronics. Este microcontrolador pertenece a la familia ARM Cortex-M4, ofreciendo un buen equilibrio entre prestaciones (velocidad de procesamiento de hasta 100 MHz, periféricos de comunicación como I2C, UART o SPI) y bajo consumo, lo que lo convierte en una plataforma adecuada para aplicaciones de monitorización y control en entornos de IoT.

2.2. Comunicaciones inalámbricas

En el ámbito del Internet de las Cosas (IoT), las comunicaciones inalámbricas juegan un papel esencial, ya que permiten transmitir datos entre dispositivos sin necesidad de cableado. Gracias a ellas, sensores, actuadores y sistemas de control pueden integrarse en redes locales o en la nube, facilitando la monitorización remota y la automatización de procesos. Existen múltiples tecnologías de comunicación inalámbrica aplicadas al IoT, como Bluetooth Low Energy, Zigbee, LoRaWAN o Wi-Fi, cada una con sus ventajas en cuanto a alcance, consumo y velocidad.

En este proyecto se ha optado por el uso de **Wi-Fi**, debido a que ofrece una alta velocidad de transmisión de datos y una infraestructura ampliamente disponible, lo que simplifica la integración del sistema con plataformas en la nube y con el servidor local instalado en la Raspberry Pi.

2.2.1. Wi-Fi y módulos ESP8266

El estándar *Wi-Fi* permite la interconexión de dispositivos a través de redes inalámbricas de área local (WLAN), operando normalmente en las bandas de 2.4 GHz y 5 GHz. Sus principales ventajas son la velocidad de transmisión y la compatibilidad con la mayoría de routers domésticos e industriales. Sin embargo, comparado con otros protocolos orientados al IoT, presenta un mayor consumo energético, lo que puede ser una limitación en dispositivos alimentados por batería.

Para dotar al microcontrolador STM32F411 de conectividad Wi-Fi, se ha empleado el **módulo ESP8266**, un *SoC* (System on Chip) de bajo coste y bajo consumo. Este tipo de circuitos integrados agrupan en un único chip tanto el procesador como la memoria, los periféricos y, en este caso, la radio Wi-Fi junto con la pila de red TCP/IP. De esta forma, el ESP8266 puede funcionar de manera autónoma o como complemento de otro microcontrolador.

En este proyecto, el ESP8266 se comunica con el STM32 mediante interfaz UART a través de comandos AT, actuando como puente entre el microcontrolador y la red Wi-Fi. Cumple la función de enviar los datos recogidos por los sensores hacia el *broker* MQTT ubicado en la Raspberry Pi, así como transmitir la información a la nube (ThingSpeak). Además, el módulo permite la recepción de mensajes desde la red, posibilitando el control remoto de los actuadores. Su versatilidad y el amplio soporte en la comunidad lo convierten en una opción idónea para aplicaciones IoT de bajo coste y rápida implementación.

2.3. Protocolos de comunicación

Los protocolos de comunicación definen las reglas mediante las cuales los dispositivos electrónicos intercambian información. En el contexto de los sistemas embebidos e IoT, la elección del protocolo adecuado es fundamental, ya que condiciona aspectos como la velocidad de transmisión, la fiabilidad, el consumo energético y la complejidad de la implementación.

En este proyecto se utilizan tres protocolos principales: **I²C**, para la comunicación con sensores; **UART**, para la interacción entre el microcontrolador STM32 y el módulo Wi-Fi ESP8266; y **MQTT**, para el envío y recepción de datos a través de la red. Cada uno de ellos cumple un papel específico y se complementan entre sí dentro de la arquitectura del sistema.

2.3.1. I²C

El protocolo *Inter-Integrated Circuit* (I²C) es un estándar de comunicación serie diseñado para conectar múltiples dispositivos mediante solo dos líneas: una de datos (SDA) y una de reloj (SCL). Se basa en una arquitectura maestro-esclavo, donde un único maestro (en este caso, el STM32F411) controla la comunicación y los sensores actúan como esclavos.

La principal ventaja de I²C es la simplicidad en el cableado y la posibilidad de conectar varios dispositivos en el mismo bus, cada uno con una dirección única. En este proyecto, se ha utilizado para interconectar el sensor de temperatura y humedad SHT85 y el sensor de luminosidad VEML7700, garantizando lecturas periódicas con baja complejidad de hardware.

2.3.2. UART

La *Universal Asynchronous Receiver-Transmitter* (UART) es una interfaz de comunicación serie asíncrona ampliamente utilizada en sistemas embebidos. A diferencia de I²C, no requiere de una línea de reloj, ya que la sincronización se realiza mediante una configuración previa de parámetros como la velocidad en baudios, el número de bits de datos y la paridad.

En este proyecto, la interfaz UART conecta el microcontrolador STM32F411 con el módulo Wi-Fi ESP8266. A través de este protocolo se envían comandos AT y se reciben respuestas, lo que permite configurar el módulo, gestionar la conexión inalámbrica y transmitir datos hacia la red.

2.3.3. MQTT

El protocolo *Message Queuing Telemetry Transport* (MQTT) es un estándar ligero de mensajería orientado a la publicación/suscripción. Diseñado específicamente para entornos IoT, permite intercambiar datos entre dispositivos de forma eficiente incluso en redes con ancho de banda limitado o conexiones poco fiables.

En este modelo, los dispositivos publican mensajes en *topics*, y otros dispositivos suscritos a esos mismos *topics* reciben la información. La comunicación se gestiona a través de un intermediario denominado *broker*, que en este proyecto se implementa en la Raspberry Pi mediante Eclipse Mosquitto.

El uso de MQTT facilita tanto la transmisión de datos de sensores hacia la nube y hacia Home Assistant, como el envío de órdenes de control a los actuadores. Gracias a su bajo consumo de recursos y a su amplia adopción en el ámbito IoT, se convierte en una pieza clave en la arquitectura del sistema.

2.4. Plataformas y herramientas software

El desarrollo de un sistema IoT no se limita únicamente al hardware, sino que requiere de un conjunto de plataformas y herramientas software que permitan programar el microcontrolador, almacenar y procesar los datos, así como visualizarlos de forma clara para el usuario final. En este proyecto se han empleado diversas soluciones, tanto locales como en la nube, que en conjunto conforman la arquitectura completa del sistema.

2.4.1. STM32CubeIDE

STM32CubeIDE es el entorno de desarrollo integrado (IDE) proporcionado por STMicroelectronics para la programación de microcontroladores STM32. Combina en una misma herramienta el editor de código, el compilador, el depurador y un configurador gráfico de periféricos. Esto facilita la generación automática de código de inicialización y reduce la complejidad en la puesta en marcha de proyectos embebidos. En este trabajo se ha utilizado para implementar la lógica de adquisición de datos de los sensores, la gestión de las comunicaciones (UART e I²C) y la integración con el módulo ESP8266 para transmitir la información al resto de plataformas.

2.4.2. ThingSpeak

ThingSpeak es una plataforma en la nube orientada al IoT que permite el almacenamiento y análisis de datos en tiempo real. Está desarrollada y mantenida por MathWorks, lo que le permite integrarse con MATLAB para el procesamiento avanzado de la información. En este proyecto se emplea como sistema de respaldo y análisis adicional, enviando los valores de temperatura, humedad y luminosidad a través de Internet para su registro y posterior representación.

2.4.3. Home Assistant

Home Assistant es una plataforma de automatización del hogar de código abierto que permite centralizar la gestión de dispositivos IoT. Su arquitectura modular y su compatibilidad con múltiples protocolos la hacen especialmente útil para integrar sensores, actuadores y servicios en un único panel de control accesible desde dispositivos móviles o navegadores web. En este trabajo, Home Assistant se ejecuta en la Raspberry Pi y permite visualizar los datos recibidos por MQTT, así como enviar órdenes a los actuadores en modo manual o en función de umbrales predefinidos en el modo automático.

2.4.4. Eagle

Autodesk Eagle es un software de diseño asistido por ordenador (CAD) especializado en esquemas eléctricos y diseño de placas de circuito impreso (PCB). Permite

crear diagramas esquemáticos, asignar componentes a bibliotecas y generar el diseño físico de la PCB con sus correspondientes capas de cobre, máscaras y serigrafías.

En este proyecto se ha utilizado Eagle para elaborar los planos eléctricos del invernadero, definiendo las conexiones entre el STM32, los sensores (SHT85 y VEML7700), el módulo ESP8266, la pantalla LCD y los actuadores. Sin embargo, no se ha llegado a fabricar ninguna PCB, ya que el cableado se ha realizado de manera directa desde el núcleo principal (la placa STM32F411E-Discovery) hacia el resto de elementos del sistema. De este modo, la propia placa de desarrollo actúa como centro de conexiones, asumiendo el papel que tendría una PCB dedicada.

El uso de Eagle en este contexto ha permitido documentar gráficamente el diseño y facilitar la comprensión de la arquitectura hardware, aunque no se haya producido físicamente una placa personalizada.

Los planos se pueden visualizar en la sección Planos eléctricos en Eagle.

2.5. Sensores empleados

Los sensores son los encargados de transformar magnitudes físicas en señales digitales que puedan ser interpretadas por el microcontrolador. En un sistema de monitorización como el planteado en este proyecto, resultan fundamentales para medir las variables ambientales que condicionan el estado del invernadero, como la temperatura, la humedad y la luminosidad. A continuación se describen los sensores seleccionados.

2.5.1. Sensor SHT85

El SHT85, desarrollado por Sensirion, es un sensor digital de alta precisión que combina en un solo encapsulado la medición de humedad relativa y temperatura. Se comunica mediante el protocolo I²C, lo que facilita su integración en sistemas embebidos con un número reducido de pines. Entre sus características destacan una precisión típica de $\pm 1,5\%$ en humedad relativa y $\pm 0,1^\circ\text{C}$ en temperatura.

En este proyecto, el SHT85 permite registrar de manera continua las condiciones de temperatura y humedad del invernadero. Sus datos no solo se emplean para la monitorización, sino también para la toma de decisiones en el **modo automático** de control. A través de Home Assistant, el usuario puede establecer los umbrales deseados para estas variables:

- Si la temperatura medida supera el umbral máximo definido por el usuario, se activa automáticamente el ventilador con el fin de reducir el exceso de calor.
- Si la humedad relativa cae por debajo del umbral mínimo configurado, se activa la bomba de agua para regar y compensar la falta de humedad.

De esta forma, el SHT85 actúa como el sensor principal para mantener estables las condiciones climáticas del invernadero, garantizando que los actuadores respondan de manera adecuada a los parámetros establecidos por el cliente.

2.5.2. Sensor VEML7700

El VEML7700, fabricado por Vishay Semiconductors, es un sensor de luminosidad ambiental que ofrece una salida digital proporcional a la cantidad de luz incidente.

Se comunica mediante el protocolo I²C, lo que permite integrarlo en el mismo bus que el SHT85 y simplificar el cableado del sistema. Entre sus características se encuentran una alta sensibilidad y un amplio rango dinámico, capaces de detectar desde niveles muy bajos de iluminancia hasta condiciones de alta luminosidad, emulando la respuesta del ojo humano a la luz visible.

En este proyecto, el VEML7700 se utiliza para monitorizar de forma continua la cantidad de luz natural disponible en el invernadero. Al igual que en el caso del SHT85, sus datos se emplean en el **modo automático** de control gestionado desde Home Assistant. El usuario establece un umbral mínimo de luminosidad que representa el nivel adecuado de iluminación para las plantas.

- Si el valor de luminosidad medido es inferior al umbral mínimo configurado, el sistema considera que hay baja luz ambiental y se activa el foco LED.

De esta forma, el VEML7700 asegura que el nivel de iluminación dentro del invernadero se mantenga siempre en los valores deseados, complementando la luz natural con iluminación artificial cuando sea necesario.

2.6. Actuadores en sistemas de riego

Además de los sensores, los actuadores desempeñan un papel fundamental en el invernadero, ya que permiten modificar las condiciones internas en función de las medidas registradas. Mientras los sensores aportan información del entorno, los actuadores son los responsables de ejecutar acciones concretas para mantener las variables dentro de los rangos deseados por el usuario.

En este proyecto los actuadores no se controlan directamente desde el microcontrolador STM32, sino desde la **Raspberry Pi**, mediante un script en Python que emplea las librerías *Paho-MQTT* y *RPi.GPIO*. De esta forma, los actuadores se gestionan a través de mensajes MQTT enviados desde Home Assistant, lo que permite mayor independencia entre la parte de sensado (STM32) y la parte de actuación (Raspberry Pi).

Se han empleado tres actuadores principales: un ventilador para regular la temperatura, una bomba de agua para el riego y un foco LED para complementar la iluminación.

2.6.1. Ventilador

El ventilador empleado es el modelo **Aigo AR12PRO** de 120 mm, habitualmente utilizado en ordenadores personales para la refrigeración de CPU o cajas de PC. Este tipo de ventiladores están diseñados para proporcionar un flujo de aire constante y silencioso, lo que los hace adecuados también para aplicaciones de ventilación en pequeños entornos cerrados.

Funciona a 12 VDC y dispone de conexión de 4 pines (alimentación, masa, señal de control PWM y señal de monitorización de velocidad), además de incorporar iluminación RGB. En este proyecto se aprovecha su capacidad de ventilación para regular la temperatura en el invernadero.

En el modo automático, el ventilador se activa cuando la temperatura registrada por el sensor SHT85 supera el umbral máximo definido por el usuario en Home Assistant. La Raspberry Pi recibe la orden por MQTT y ejecuta un script en Python

que conmuta el relé encargado de alimentar el ventilador con los 12 V necesarios para su funcionamiento.

De esta manera se garantiza un control seguro, aislando el sistema de control de bajo voltaje (3,3 V de la Raspberry Pi) de la etapa de potencia de 12 V requerida por el ventilador.

2.6.2. Bomba de agua

La bomba empleada es un modelo sumergible de 5V, compacto y diseñado para aplicaciones en acuarios y sistemas de riego de pequeña escala. En el modo automático, la bomba se activa cuando la humedad relativa medida por el SHT85 desciende por debajo del umbral mínimo configurado en Home Assistant. El comando MQTT llega al script en Python, que gestiona el relé encargado de alimentar la bomba.

2.6.3. Foco LED

El foco LED seleccionado es un proyector de 10W con luz blanca neutra, suficiente para complementar la iluminación natural dentro del invernadero en condiciones de baja luminosidad. En el modo automático, se activa cuando el sensor VEML7700 detecta que la iluminación está por debajo del umbral mínimo establecido en Home Assistant. La orden se transmite mediante MQTT y el script en la Raspberry Pi conmuta el relé correspondiente, encendiendo el foco LED de manera inmediata.

Gracias a este enfoque, los actuadores quedan gestionados de manera robusta por la Raspberry Pi, mientras que el microcontrolador STM32 se centra exclusivamente en la captura de datos de los sensores.

Capítulo 3

Estado del arte

En este capítulo se analizan las tecnologías y soluciones existentes relacionadas con los sistemas domóticos aplicados a invernaderos inteligentes. El objetivo es situar el proyecto dentro del marco tecnológico actual, describiendo los avances en domótica, las aplicaciones IoT más relevantes, los sensores de precisión empleados y los protocolos de comunicación que hacen posible la integración de los distintos dispositivos.

3.1. Domótica

La domótica se encuentra cada vez más presente en la vida cotidiana, aunque su adopción no siempre resulta sencilla, especialmente para las personas de mayor edad. La transición hacia el uso de nuevas tecnologías suele implicar una curva de aprendizaje que dificulta su integración en determinados grupos de población. Sin embargo, la automatización de tareas en el hogar representa una ayuda fundamental para mejorar la calidad de vida, la autonomía y la seguridad de este colectivo.

Varios países europeos están apostando por implementar viviendas adaptadas con soluciones domóticas orientadas al cuidado de personas mayores. Estas tecnologías permiten reducir la dependencia de familiares o cuidadores, al ofrecer herramientas que facilitan la autosuficiencia. Por ejemplo, el acceso remoto a servicios médicos mediante teleconsulta evita desplazamientos innecesarios, lo que supone una ventaja tanto para el paciente como para quienes deben asistirlo.

En el caso particular de la agricultura, la domótica aplicada a invernaderos permite controlar de forma remota variables ambientales como la temperatura, la humedad o la iluminación, mejorando el rendimiento de los cultivos y reduciendo el consumo de recursos [14].

De este modo, las tecnologías domóticas ya no se limitan al confort del hogar, sino que se convierten en herramientas clave para sectores productivos.

En sus primeras aplicaciones, la domótica se apoyaba en sistemas de control básicos con temporizadores y relés. Estos métodos eran poco flexibles, ya que no se adaptaban a las condiciones reales del entorno. Con el tiempo, la aparición de microcontroladores programables como *Arduino* y *Raspberry Pi* facilitó la creación de prototipos de bajo coste capaces de interactuar con sensores y actuar sobre diferentes dispositivos eléctricos. En la actualidad, gracias a plataformas de supervisión como Home Assistant, los sistemas domóticos permiten integrar múltiples dispositivos en

una única interfaz accesible desde el ordenador o el teléfono móvil, ofreciendo un control completo y centralizado.

Un ejemplo de aplicación de la domótica se muestra en la Figura 3.1, donde se aprecia cómo distintos dispositivos del hogar se interconectan a través de una red común. De manera similar, en un invernadero domótico los sensores de temperatura, humedad y luminosidad se comunican con un microcontrolador, que a su vez envía la información a una plataforma de gestión capaz de accionar ventiladores, bombas de riego o focos de iluminación de forma automática.



Figura 3.1: Ejemplo de sistema domótico con integración de dispositivos conectados [23].

La evolución de la domótica ha permitido además el control remoto mediante dispositivos móviles, posibilitando el acceso desde cualquier lugar con conexión a internet. En el caso de los invernaderos, esto se traduce en la capacidad de activar el riego o la ventilación sin necesidad de estar físicamente presente, optimizando así la gestión de los recursos y aumentando la eficiencia del sistema.

3.2. Aplicaciones en la agricultura

En el ámbito agrícola, la incorporación de tecnologías domóticas ha cobrado especial relevancia debido a la necesidad de optimizar recursos y facilitar el trabajo en entornos que demandan un alto esfuerzo físico y una elevada cantidad de tiempo. La gestión automatizada de variables como la temperatura, la humedad y la luminosidad permite a los agricultores mantener unas condiciones óptimas de cultivo sin necesidad de supervisión constante.

Entre los actuadores más empleados en este sector destacan los sistemas de riego automatizados mediante bombas de agua, capaces de adaptarse a las necesidades hídricas en función de los datos recogidos por sensores de humedad del suelo. Asimismo, el control de ventiladores contribuye a regular la circulación de aire en invernaderos, evitando la acumulación de humedad excesiva y reduciendo el riesgo de plagas o enfermedades. Otro actuador de gran interés es la iluminación artificial mediante focos LED, que asegura el desarrollo de los cultivos en escenarios de baja radiación solar o durante periodos prolongados de oscuridad.

La integración de estos dispositivos con plataformas de gestión centralizadas permite a los agricultores monitorizar y controlar sus producciones de forma remota, incluso a través de aplicaciones móviles. De este modo, se facilita la toma de decisiones en tiempo real, se optimiza el uso de agua y energía, y se incrementa la productividad, respondiendo a la creciente necesidad de sostenibilidad en la agricultura moderna.

3.3. Aplicaciones IoT

El Internet de las Cosas (IoT) ha transformado la forma en la que se supervisan y controlan dispositivos distribuidos. En sus inicios, las aplicaciones IoT en entornos domésticos o agrícolas se limitaban a sistemas propietarios y cerrados, que requerían servidores específicos y tenían altos costes de implementación. Estas soluciones eran poco accesibles para usuarios sin conocimientos avanzados y no permitían una integración sencilla entre distintos fabricantes [4].

Con la expansión del software libre y el abaratamiento de plataformas como Raspberry Pi, surgieron soluciones de código abierto capaces de centralizar el control de sensores y actuadores en una única interfaz. Este cambio marcó un punto de inflexión, ya que permitió a investigadores, empresas y usuarios particulares construir sistemas escalables y económicos [9].

Actualmente, una de las plataformas más extendidas es Home Assistant, un software de código abierto que destaca por:

- Su versatilidad, al permitir integrar dispositivos de múltiples fabricantes en una misma interfaz.
- Su compatibilidad con protocolos estándar como MQTT, Zigbee o Z-Wave.
- La amplia comunidad de desarrolladores que aporta integraciones con multitud de dispositivos distintos.
- Su interfaz gráfica personalizable y accesible desde cualquier dispositivo.

Junto a Home Assistant existen alternativas consolidadas como:

- **OpenHAB**: plataforma también de código abierto, con una arquitectura modular basada en Java.
- **Domoticz**: solución ligera que se ejecuta en hardware de bajo coste, muy utilizada en proyectos de automatización sencilla.
- **Node-RED**: herramienta de programación visual orientada a flujos, ideal para integrar dispositivos IoT y servicios en la nube sin necesidad de programar en detalle.

De acuerdo con estudios recientes [14], estas plataformas de gestión IoT se han convertido en un pilar fundamental para aplicaciones de agricultura inteligente, ya que permiten la centralización de datos, la automatización de procesos y la supervisión remota. En este contexto, Home Assistant se ha consolidado como la opción más utilizada en proyectos académicos y domésticos por su facilidad de instalación y su compatibilidad con distintos protocolos.

3.4. Sensores

El uso de sensores en agricultura ha evolucionado de manera notable en las últimas décadas. En un inicio, el control de variables ambientales en invernaderos se realizaba de forma manual, mediante termómetros analógicos, higrómetros o medidores de luz de baja precisión. Estos sistemas eran poco fiables y requerían la presencia constante del operario, lo que limitaba la capacidad de automatización [13].

Con la introducción de la electrónica aplicada a la agricultura, comenzaron a emplearse sensores básicos de bajo coste, como termistores o fotodiodos, que permitían una monitorización continua, aunque con limitaciones importantes en cuanto a precisión, estabilidad y rango de operación [31]. Estos dispositivos facilitaron la automatización inicial de sistemas de riego o ventilación, pero presentaban problemas de calibración y envejecimiento prematuro en entornos de alta humedad.

En la actualidad, se dispone de sensores digitales de alta precisión específicamente diseñados para condiciones adversas, que integran calibración interna, compensación térmica y protocolos de comunicación estandarizados como I²C. Entre ellos destacan:

- **SHT85** de Sensirion: sensor de temperatura y humedad con exactitud típica de $\pm 1,5\%$ en HR y $\pm 0,1^{\circ}\text{C}$ en temperatura, protegido frente a polvo y salpicaduras gracias a una membrana con certificación IP67 [30]. Este dispositivo garantiza estabilidad a largo plazo y reduce la necesidad de calibración periódica.
- **VEML7700**: sensor de luminosidad de alta resolución, capaz de medir desde niveles de iluminación muy bajos hasta plena exposición solar, con linealidad elevada y bajo nivel de ruido, lo que permite una monitorización fiable de la radiación solar incidente [34].

El empleo de sensores modernos presenta múltiples beneficios frente a las generaciones anteriores:

- Mayor **precisión y repetibilidad**, gracias a la compensación interna y a los algoritmos de calibración de fábrica.
- **Durabilidad y resistencia**, al integrar membranas protectoras y encapsulados diseñados para ambientes hostiles como los invernaderos.
- Compatibilidad con sistemas embebidos a través de buses de comunicación, lo que simplifica su integración en microcontroladores como STM32 o ESP8266.

Atendiendo a estudios recientes [3, 21], la combinación de sensores digitales de alta precisión con sistemas IoT ha permitido incrementar la eficiencia en el uso de agua hasta en un 30 % en invernaderos. Estos avances consolidan la importancia de emplear sensores robustos y precisos como base para cualquier sistema domótico agrícola.

3.5. Protocolos de comunicación

La comunicación entre dispositivos constituye la base de cualquier sistema IoT, ya que permite la transmisión de datos entre sensores, microcontroladores y plataformas de supervisión. En el ámbito agrícola, la elección del protocolo es determinante

porque influye en el consumo energético, la latencia, la fiabilidad y la escalabilidad del sistema.

Entre los protocolos más empleados destacan:

- **MQTT (Message Queuing Telemetry Transport)**: diseñado para comunicaciones ligeras en entornos con recursos limitados. Utiliza un modelo *publish/subscribe*, lo que facilita la integración entre sensores y plataformas como Home Assistant, Node-RED o ThingSpeak. Su bajo consumo de ancho de banda lo convierte en un referente para muchos proyectos [19].
- **CoAP (Constrained Application Protocol)**: protocolo de comunicación diseñado para dispositivos con recursos limitados en términos de memoria, capacidad de procesamiento y consumo energético. Está basado en el modelo arquitectónico REST, lo que significa se accede a los recursos a través de los métodos estándar de HTTP (GET, POST, PUT y DELETE). A diferencia de HTTP, utiliza UDP como protocolo de transporte, lo que reduce la sobrecarga y permite tiempos de respuesta más cortos. Asimismo, CoAP soporta mecanismos de confirmación de mensajes y multicasting, lo que lo hace especialmente adecuado para aplicaciones en redes de sensores inalámbricos y sistemas IoT de bajo consumo [8].
- **AMQP (Advanced Message Queuing Protocol)**: orientado a aplicaciones empresariales, garantiza entrega fiable de mensajes con confirmaciones y colas persistentes. Aunque más pesado que MQTT, se emplea en sistemas donde la fiabilidad prima sobre el bajo consumo [7].
- **HTTP/HTTPS**: tradicionalmente usado en servicios web y plataformas en la nube. En IoT se emplea aún para integraciones con APIs (como ThingSpeak), aunque su sobrecarga lo hace menos eficiente que alternativas como MQTT o CoAP.
- **Protocolos locales (I²C, UART, SPI)**: se utilizan en la comunicación interna entre microcontroladores y sensores. Por ejemplo, I²C conecta el STM32 con el SHT85 y el VEML7700, mientras que UART se emplea para enlazar el STM32 con el módulo ESP8266 encargado de la conectividad WiFi.

De acuerdo con distintas revisiones recientes [40, 39], la tendencia actual se orienta al uso de protocolos ligeros como MQTT o CoAP. Estos protocolos permiten diseñar sistemas IoT agrícolas con bajo consumo energético, alta flexibilidad y compatibilidad con el resto de dispositivos.

Capítulo 4

Requisitos y diseño del sistema

4.1. Arquitectura del sistema

En esta sección se representa la estructura y comunicación entre los diferentes componentes del sistema domótico para invernadero, destacando los nodos principales, los protocolos de comunicación y las conexiones funcionales.

4.1.1. Esquema general del sistema

La Figura 4.1 muestra el diagrama de flujo general del sistema domótico desarrollado. En él se representan los principales bloques funcionales, las comunicaciones entre dispositivos y la integración con servicios en la nube y plataformas locales.

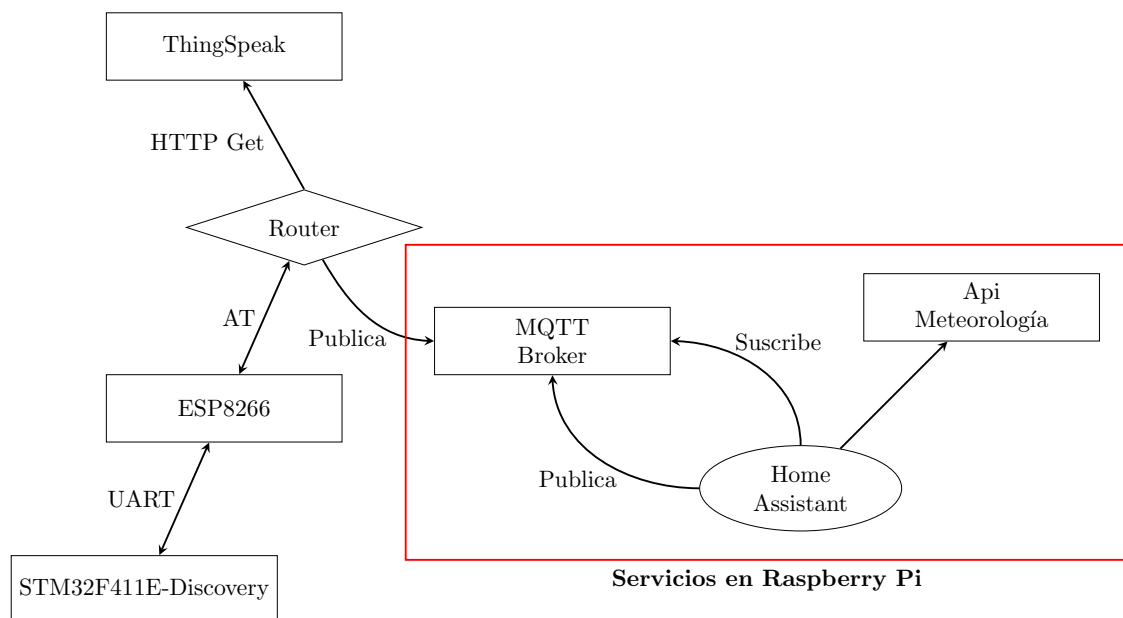


Figura 4.1: Arquitectura del sistema domótico de invernadero.

4.1.2. Finalidad del esquema

El diagrama de flujo es esencial para entender la estructura e integración del sistema domótico desarrollado. Permite identificar los nodos principales, los métodos

de comunicación y el sentido del flujo de datos. Asimismo, destaca la interacción entre los componentes hardware (como el microcontrolador STM32, sensores y actuadores) y los módulos software (como Home Assistant y el broker MQTT), facilitando la comprensión global del sistema.

4.1.3. Descripción de la arquitectura

El anterior esquema representa el flujo de datos y la comunicación entre los distintos módulos que componen el sistema domótico desarrollado para el invernadero. A continuación, se detallan los elementos y conexiones más relevantes:

- **STM32F411E-Discovery:** Microcontrolador principal encargado de la adquisición de datos procedentes de sensores ambientales (temperatura, humedad y luminosidad). Además, gestiona la comunicación UART con el módulo ESP8266. También se encarga de controlar los actuadores del sistema (focos, ventiladores, bomba de agua) según las órdenes recibidas desde Home Assistant.
- **ESP8266:** Módulo WiFi que actúa como puente entre el microcontrolador y el router, utilizando comandos AT a través de UART. Es el encargado de gestionar las conexiones TCP/IP necesarias para la transmisión de datos. Mediante comandos AT, se envían mensajes HTTP hacia ThingSpeak y paquetes MQTT hacia el broker alojado en la Raspberry Pi. La transmisión se sincroniza de forma que cada minuto se envían los datos a ThingSpeak y, con un desfase de 30 segundos, se envían también por MQTT a la Raspberry Pi. Esto evita colisiones en la comunicación y garantiza una distribución ordenada de la información.
- **Router:** Proporciona conectividad a Internet al módulo ESP8266 y a la Raspberry Pi, permitiendo la comunicación en red local y con servicios externos.
- **MQTT Broker:** Servidor MQTT implementado en la Raspberry Pi. Recibe los datos publicados por el ESP8266 y los distribuye a los distintos suscriptores, como Home Assistant.
- **Home Assistant:** Plataforma de automatización que actúa como suscriptor de los datos del sistema. Permite visualizar en tiempo real las variables del invernadero desde el teléfono móvil y enviar comandos al microcontrolador para activar/desactivar actuadores.
- **ThingSpeak:** Plataforma en la nube utilizada para almacenar y visualizar los datos del sistema a través del protocolo HTTP. Recibe los valores directamente desde el ESP8266 mediante peticiones HTTP GET.
- **API de Meteorología:** Servicio externo que proporciona datos climáticos de referencia, accesible desde Home Assistant para mejorar la toma de decisiones del sistema. Especialmente útil para conocer la probabilidad de lluvia y así evitar un riego innecesario.
- **Protocolo MQTT:** Utilizado tanto para publicar como para suscribirse a mensajes entre nodos del sistema, especialmente entre el ESP8266, la Raspberry Pi y Home Assistant.
- **Protocolo UART:** Comunicación serie entre el STM32 y el ESP8266, clave para la transmisión de datos desde el microcontrolador hacia la red WiFi.

4.1.4. Relación entre nodos

En el diagrama de flujo representado en la Figura 4.1, se emplean distintos tipos de flechas para indicar la dirección de la comunicación entre los nodos del sistema:

- **Flechas dirigidas en un solo sentido:** Indican una comunicación unidireccional, en la que un nodo transmite información a otro sin recibir respuesta directa. Por ejemplo, el módulo ESP8266 realiza peticiones HTTP hacia ThingSpeak para subir datos, pero ThingSpeak no responde con comandos ni datos procesables por el sistema.
- **Flechas bidireccionales:** Representan comunicaciones en ambos sentidos, donde los dos nodos pueden intercambiar datos entre sí. Un ejemplo claro es la conexión UART entre la STM32 y el ESP8266. Por un lado, la STM32 envía al ESP8266 los valores medidos por los sensores (como temperatura, humedad y luminosidad) para que este los publique en el broker MQTT o los transfiera a ThingSpeak. Por otro lado, el ESP8266 también transmite información a la STM32: recibe órdenes desde Home Assistant (ejecutándose en la Raspberry Pi) a través del protocolo MQTT, y las reenvía a la placa mediante UART para activar o desactivar los actuadores.
- **Publicar y suscribirse (MQTT):** El protocolo MQTT se basa en un modelo *publish/subscribe* en el que:
 - **Publicar (publish)** significa enviar un mensaje a un *topic* concreto del broker. Por ejemplo, el ESP8266 publica un mensaje JSON con los valores de temperatura, humedad y luminosidad al topic **Invernadero/sensores**: Ejemplo de mensaje publicado en el topic:

```
{
  "temp": 17.54,
  "hum": 62.12,
  "lux": 80.57
}
```

- **Suscribirse (subscribe)** implica registrarse en un *topic* para recibir automáticamente los mensajes publicados en él. Por ejemplo, Home Assistant se suscribe al topic **Invernadero/estado** para mostrar los valores actuales en su interfaz.

Este modelo permite desacoplar los nodos, ya que un publicador no necesita saber quién está recibiendo sus mensajes, y un suscriptor puede recibir datos de múltiples fuentes sin establecer conexiones directas con ellas.

4.2. Resumen general del sistema

El resumen general del sistema se resume en la tabla 4.1.

4.3. Descripción de los topics MQTT

Elemento	Descripción
Microcontrolador	STM32F411E-Discovery. Lectura de sensores, control de relés, envío de datos por UART al módulo ESP8266.
Sensores	■ SHT85: Temperatura y humedad vía I2C1. ■ VEML7700: Luminosidad en lux vía I2C2.
Pantalla LCD	Display 16x2 conectado en modo 4 bits por GPIO. Muestra en tiempo real los valores medidos.
Comunicación WiFi	Módulo ESP8266 controlado por comandos AT vía UART6. Publica datos en ThingSpeak y en el broker MQTT.
Protocolo de envío	MQTT (JSON por topic <code>Invernadero/estado</code>) y HTTP (ThingSpeak).
Gateway y servidor	Raspberry Pi con Mosquitto como broker MQTT y Home Assistant como servidor domótico.
Interfaz de usuario	App móvil/web de Home Assistant para visualizar datos y controlar el riego.
Control de actuadores	Gestión de ventilador, bomba y foco mediante topics <code>Invernadero/actuadores/{dispositivo}/control</code> .
Servicios adicionales	■ ThingSpeak: almacenamiento en la nube y gráficas. ■ API meteorológica: integración en la aplicación de Home Assistant.

Tabla 4.1: Resumen funcional y arquitectónico del sistema

State / Command Topic	Contenido
<code>Invernadero/sensores</code>	Medidas JSON (T/H/L)
<code>Invernadero/actuadores/ventilador/estado</code>	Estado ventilador
<code>Invernadero/actuadores/ventilador/control</code>	Control ventilador
<code>Invernadero/actuadores/bomba/estado</code>	Estado bomba
<code>Invernadero/actuadores/bomba/control</code>	Control bomba
<code>Invernadero/actuadores/foco/estado</code>	Estado foco
<code>Invernadero/actuadores/foco/control</code>	Control foco
<code>Invernadero/modo/automatico/estado</code>	Estado automático
<code>Invernadero/modo/automatico/control</code>	Control automático
<code>Invernadero/umbrales/temp/min/set</code>	Ajuste Tmin
<code>Invernadero/umbrales/temp/min/state</code>	Valor Tmin
<code>Invernadero/umbrales/temp/max/set</code>	Ajuste Tmax
<code>Invernadero/umbrales/temp/max/state</code>	Valor Tmax
<code>Invernadero/umbrales/humedad/min/set</code>	Ajuste HRmin
<code>Invernadero/umbrales/humedad/min/state</code>	Valor HRmin
<code>Invernadero/umbrales/humedad/max/set</code>	Ajuste HRmax
<code>Invernadero/umbrales/humedad/max/state</code>	Valor HRmax
<code>Invernadero/umbrales/luz/min/set</code>	Ajuste Lmin
<code>Invernadero/umbrales/luz/min/state</code>	Valor Lmin
<code>Invernadero/umbrales/luz/max/set</code>	Ajuste Lmax
<code>Invernadero/umbrales/luz/max/state</code>	Valor Lmax
<code>Invernadero/activacion/ventilador</code>	Activación ventilador
<code>Invernadero/activacion/bomba</code>	Activación bomba
<code>Invernadero/activacion/foco</code>	Activación foco

Tabla 4.2: Resumen de `state_topic` y `command_topic` empleados en el sistema.

Capítulo 5

Sensores y actuadores

5.1. Sensores

5.1.1. Sensor de temperatura y humedad SHT85

Descripción general

El sensor **SHT85** de Sensirion es el encargado de medir la *temperatura* y la *humedad relativa* del ambiente. Este sensor digital se comunica con el microcontrolador STM32F411 mediante el bus I2C1 y opera en modo periódico con alta repetibilidad, proporcionando una medición por segundo de forma autónoma (véase Figura 5.1).

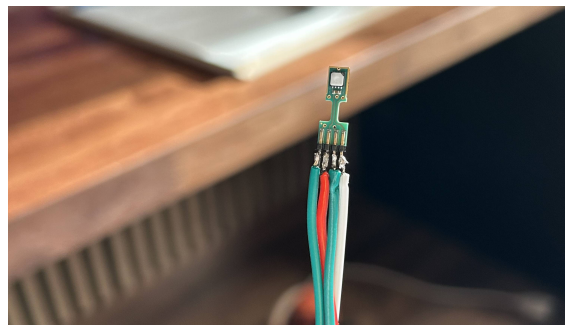


Figura 5.1: Sensor de temperatura y humedad SHT85

Inicialización y lectura de datos

La función `SHT85_Init()` configura el sensor para operar en modo periódico a 1 medición por segundo, con alta precisión. Para ello, se envía el comando `0x2130` a través del bus I2C1. Las mediciones se leen mediante `ReadSHT85_Periodic()`, que transmite el comando `0xE000` (Fetch Data) y recibe 6 bytes con los datos crudos.

Conversión de datos

Los valores de temperatura y humedad se convierten a unidades físicas mediante las fórmulas indicadas por el fabricante en las hojas de características:

$$\text{Temperatura } (^{\circ}\text{C}) = -45 + 175 \cdot \frac{\text{temp_raw}}{65535,0} \quad (5.1)$$

$$\text{Humedad relativa (\%RH)} = 100 \cdot \frac{\text{hum_raw}}{65535,0} \quad (5.2)$$

Gestión de errores y reinicio automático

Para aumentar la robustez del sistema, se implementa la función `SHT85_ErrorReset()`, que detecta mediciones fuera de rango (como temperaturas inferiores a -44°C o humedades menores al 1%). Si se detectan 10 errores consecutivos, se ejecuta un *soft reset* mediante `ResetSHT85()`, y se vuelve a activar el modo periódico.

Cálculo de promedios

Con el objetivo de reducir el ruido de las mediciones y obtener datos más estables, se implementan las funciones `calculatorAverageTemperature()` y `calculatorAverageHumidity()`, que calculan promedios cada 60 lecturas (60 segundos). Estos valores medios son los que posteriormente se envían por MQTT o a la nube.

Diagrama de funcionamiento

El proceso de lectura y gestión del sensor SHT85 se organiza mediante una máquina de estados que permite validar los datos obtenidos y garantizar la fiabilidad del sistema ante posibles errores de comunicación. En caso de detectar mediciones anómalas de forma consecutiva, se ejecuta un reinicio automático del sensor para restablecer su funcionamiento. La lógica de este proceso se representa en la Figura 5.2.

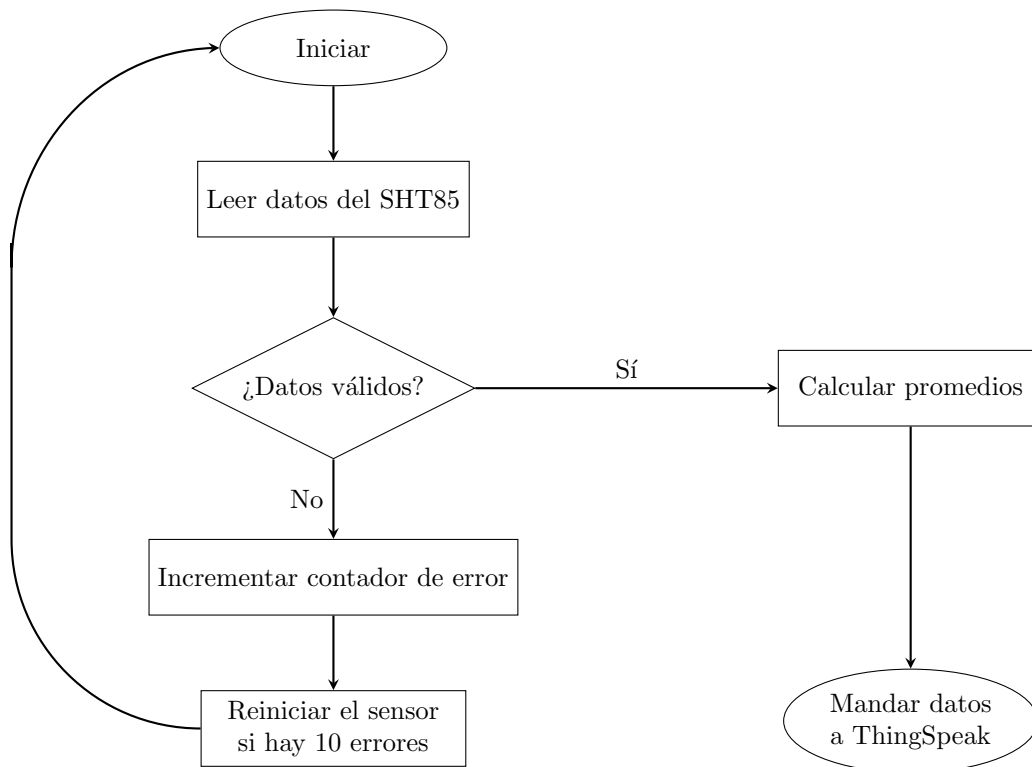


Figura 5.2: Diagrama de flujo de lectura, validación y reinicio del sensor SHT85.

Asignación de pines

- **SHT85**: conectado al bus I2C1 del STM32.
- **Pin SCL**: PB6
- **Pin SDA**: PB7

5.1.2. Sensor de luminosidad VEML7700

Descripción general

El sensor **VEML7700** de Vishay es un sensor digital de luminosidad que permite medir el nivel de luz ambiental en lux. Se comunica con el microcontrolador STM32 mediante el bus I2C2. Gracias a su alta resolución y sensibilidad, es ideal para sistemas de control de iluminación y análisis ambiental (véase Figura 5.3).



Figura 5.3: Sensor de luminosidad VEML7700

Inicialización y lectura de datos

El sensor se inicializa mediante la función `VEML7700_Init()`, que escribe en su registro de configuración `0x00` para activar el sensor con ganancia `x1` y tiempo de integración de `100 ms`. La lectura se realiza desde el registro `0x04`, que devuelve dos bytes con el valor crudo de luminosidad.

Conversión de datos

Los datos leídos se convierten a unidades de lux aplicando el factor de conversión indicado por el fabricante:

$$\text{Lux} = \text{lux_raw} \cdot 0,0036 \quad (5.3)$$

Cálculo de promedio

Para obtener mediciones más estables, se implementa la función `calculateAverageLuminosity()`, que calcula un promedio cada 40 lecturas. Este valor promedio es el que se transmite por MQTT o se envía a la nube.

Frecuencia de lectura y envío

El sistema realiza una lectura del VEML7700 cada segundo. Cada 60 segundos, se calcula el promedio y se integra en un mensaje JSON junto con los datos de temperatura y humedad, que es enviado por MQTT o ThingSpeak.

Diagrama de funcionamiento

El flujo de operación del sensor de luminosidad VEML7700 sigue una lógica similar al sensor de temperatura y humedad. Se valida cada lectura recibida y, en caso de detectar errores consecutivos, se reinicia automáticamente el sensor para asegurar lecturas fiables. La Figura 5.4 muestra el diagrama de funcionamiento implementado para este proceso.

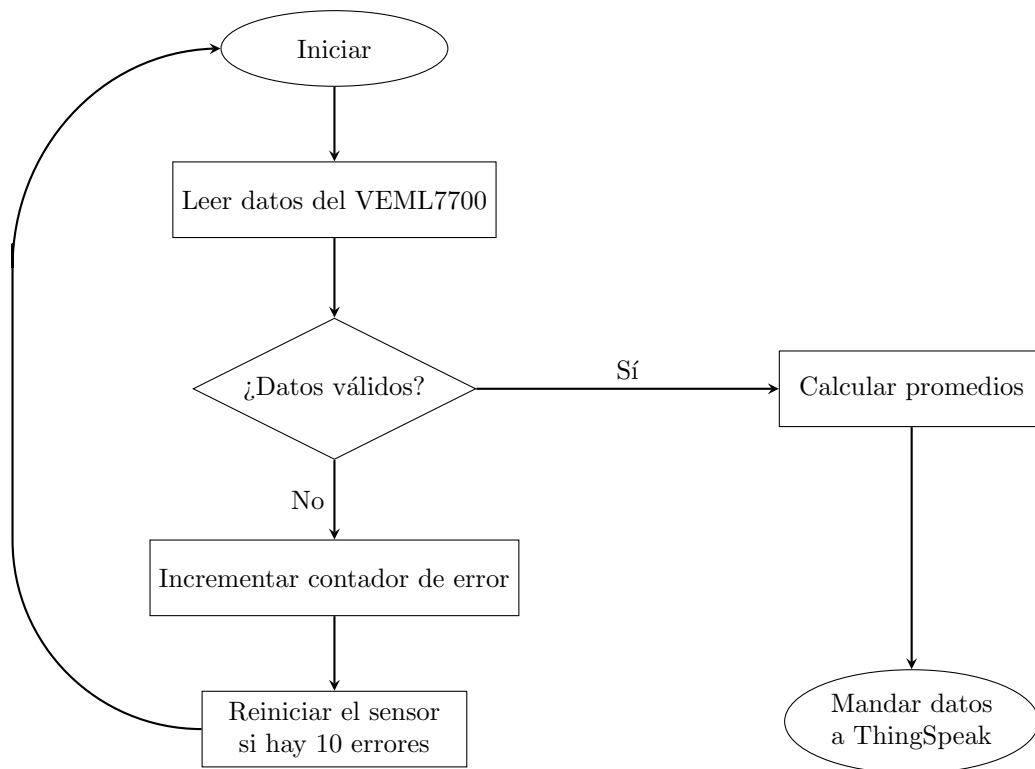


Figura 5.4: Diagrama de flujo de lectura, validación y reinicio del sensor VEML7700.

5.1.3. Asignación de pines

- **VEML7700:** conectado al bus I2C2 del STM32.
- **Pin SCL:** PB10
- **Pin SDA:** PB3

5.2. Actuadores

5.2.1. Hardware

Se han seleccionado tres actuadores principales, cuyo funcionamiento estará controlado por el STM32 a través de relés:

1. **Ventiladores AR12 PRO:** Funcionan a 12 VDC, mejoran la circulación de aire y evitan la acumulación de humedad, contribuyendo a regular la temperatura interna (véase Figura 5.5).
2. **Foco 10 W:** Funciona a 230 VAC y aporta luz artificial para compensar la falta de luz natural, con control inteligente para alcanzar el umbral recomendado de exposición solar diaria (10–12 h), como se muestra en la Figura 5.6.
3. **Bomba de agua:** Funciona a 5 VDC y automatiza el riego a partir de los datos sensados, con posibilidad de añadir sensores de nivel para supervisar el depósito (véase Figura 5.7).



Figura 5.5: Ventilador para regulación térmica y circulación de aire.

5.2.2. Control de actuadores

Los actuadores del sistema (ventiladores, bomba de agua y foco de iluminación) se controlan directamente desde la **Raspberry Pi**, la cual recibe las órdenes a través de **Home Assistant**. Esta arquitectura permite **aislar físicamente** la etapa de potencia de la de sensorización, de modo que el microcontrolador STM32 se dedica exclusivamente a la adquisición y procesamiento de datos, mientras que la Raspberry asume las tareas de control y supervisión.

Además, la Raspberry Pi ofrece una mayor robustez y estabilidad que la STM32 para la gestión de los actuadores, ya que cuenta con un sistema operativo completo, soporte nativo de red y capacidad de ejecutar servicios en segundo plano. De esta manera se garantiza un control más fiable a largo plazo y se facilita la integración con Home Assistant.

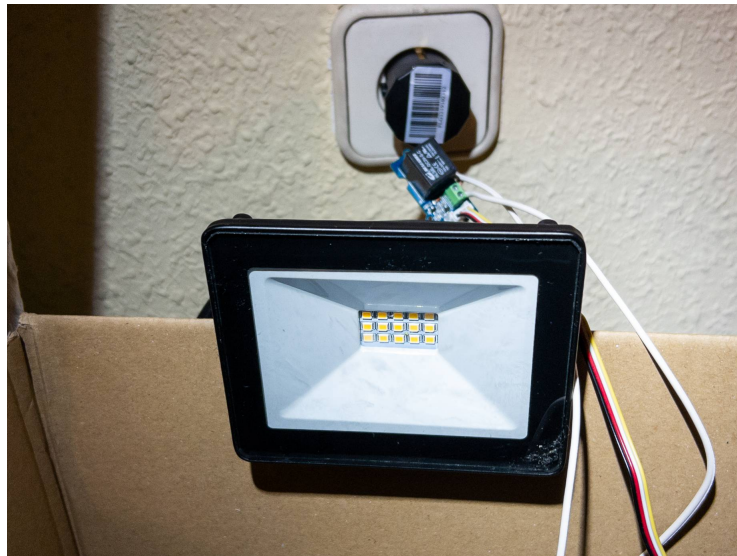


Figura 5.6: Foco LED controlado por relé para iluminación suplementaria.



Figura 5.7: Bomba de agua para riego automatizado del invernadero.

Para más detalles, véase la Sección 7.5.

5.2.3. Interfaz de potencia: relés y seguridad

Los actuadores del invernadero (ventiladores, bomba de agua y sistema de iluminación) trabajan algunos a tensiones de **230 V AC**, por lo que no es posible controlarlos de manera directa desde el microcontrolador STM32 ni desde la Raspberry Pi. Para ello se utiliza un módulo de relés con optoacopladores (véase la Figura 5.8)., que actúa como interfaz de potencia y proporciona aislamiento galvánico entre la electrónica de control y las cargas de alta tensión.

Gracias a este diseño, la lógica de control se mantiene en un entorno seguro a bajo voltaje, mientras que la conmutación de los dispositivos de red eléctrica se realiza de forma fiable y protegida. Los relés soportan tensiones de hasta 250 V AC o 30 V DC, lo que permite accionar de manera segura los distintos actuadores del sistema.

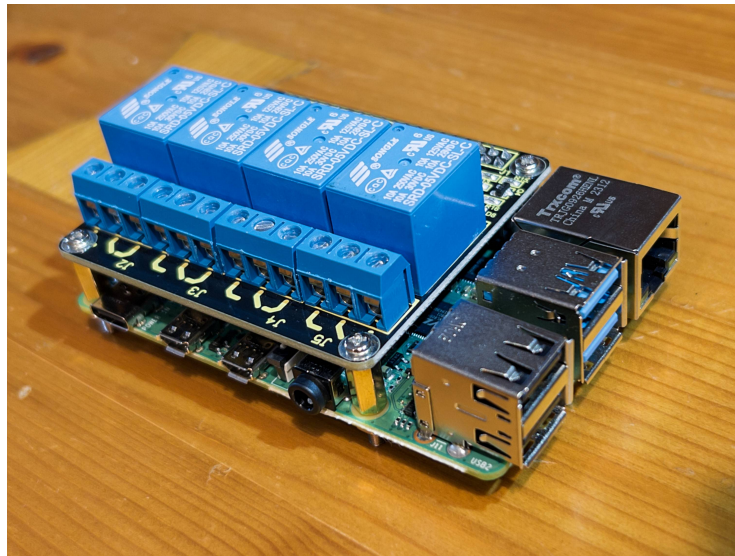


Figura 5.8: Módulo Keyestudio RPI 4-Relay Shield (KS0212) conectado a la Raspberry Pi 4B como interfaz de potencia.

5.3. Periféricos

5.3.1. Display LCD

El sistema incorpora un display LCD alfanumérico de 16x2 caracteres con retro-iluminación LED blanca, conectado directamente al microcontrolador STM32F411. Este elemento permite visualizar localmente los valores actuales de temperatura, humedad y luminosidad, sin necesidad de conexión a red o interfaz externa.

Conexión e interfaz

El display opera en modo de 4 bits y se conecta al STM32 a través de pines GPIO del puerto C. Se utilizan las siguientes señales:

- RS (PC0): selección de registro (comando o datos).

- E (PC1): señal de habilitación.
- D4-D7 (PC2-PC5): líneas de datos.

Inicialización y control

La inicialización del display se realiza mediante una secuencia estándar para pantallas HD44780, configurando:

- Modo 4 bits de comunicación.
- 2 líneas de texto con matriz de caracteres 5x7.
- Cursor oculto y desplazamiento automático.

La escritura se realiza carácter a carácter con la función `LCD.Print()`, que utiliza internamente `LCD.SendData()`. El sistema también permite posicionar el cursor mediante `LCD.SetCursor(row, col)` y limpiar la pantalla con `LCD.Clear()`.

Contenido mostrado

En el bucle principal del programa, se actualiza periódicamente el contenido del display para mostrar las variables medidas por los sensores. La disposición seleccionada es la siguiente: la línea 1 se reserva para temperatura y humedad, mientras que la línea 2 muestra la luminosidad en lux, como se aprecia en la Figura 5.9.

Temperatura: 16.29°C | Humedad: 51 %RH
Luminosidad: 0.126 lux

Esta información proporciona una referencia visual en tiempo real al usuario, sin necesidad de depender de entornos remotos.



Figura 5.9: Visualización de las magnitudes medidas a través del Display LCD.

Capítulo 6

Comunicaciones en STM32

6.1. ThingSpeak: almacenamiento en la nube

ThingSpeak es una plataforma IoT que permite el almacenamiento y representación gráfica de datos en la nube. En este proyecto se utiliza para registrar las variables ambientales medidas en el invernadero, ofreciendo un histórico consultable desde cualquier navegador web.

6.1.1. Protocolo de comunicación

La comunicación se realiza mediante el módulo WiFi ESP8266 actuando como cliente TCP. El STM32 envía solicitudes HTTP de tipo GET al servidor `api.thingspeak.com`, tal y como se muestra en la siguiente petición:

```
GET /update?api_key=APIKEY&field1=16.29&field2=51.61&field3=81.37 HTTP/1.1
Host: api.thingspeak.com
Connection: close
```

Código 6.1: Petición HTTP GET enviada desde el STM32 al servidor ThingSpeak

donde:

- APIKEY es la clave de escritura del canal en ThingSpeak.
- field1, field2, field3 corresponden a temperatura, humedad y luminosidad.

6.1.2. Secuencia de comandos AT

La conexión se controla mediante comandos AT enviados al ESP8266:

1. AT+CIPSTART para iniciar la conexión TCP.
2. AT+CIPSEND con la longitud del mensaje HTTP.
3. Envío del mensaje HTTP.

La comunicación se maneja mediante interrupciones UART y verificación de respuestas con timeout para garantizar robustez.

6.1.3. Máquina de estados para ThingSpeak

El envío de datos a ThingSpeak se implementa mediante una **máquina de estados no bloqueante** definida en `wifi_thingspeak.c` y `wifi_thingspeak.h`.

```
void processThingSpeakStateMachine() {
    if (thingSpeakState != ts_prev_state) {
        ts_prev_state = thingSpeakState;
        ts_enter_tick = HAL_GetTick();
    }
    if (thingSpeakState != 0u && (HAL_GetTick() - ts_enter_tick) >
        ANTIBLOCK_STATE_MACHINE) {
        alertaBloqueo = true;
        return;
    }

    switch (thingSpeakState) {
        case 1: {
            closeConnection();
            thingSpeakState = 2;
            break;
        }
        case 2: {
            Uart_flush();
            Uart_sendstring("AT+CIPSTART=\"TCP\", \"api.thingspeak.com\", 80\r\n");
            if (!Wait_for("OK\r\n") && !Wait_for("CONNECT\r\n") &&
                !Wait_for("CLOSED\r\n")) {
                closeConnection();
                thingSpeakState = 0;
                break;
            }
            thingSpeakState = 3;
            break;
        }
        case 3: {
            Uart_flush();
            char http_request[250];
            sprintf(http_request, "GET /update?api_key=%s&field1=%.2f&
                field2=%.2f&field3=%.2f HTTP/1.1\r\nHost: api.thingspeak.com\r\n
                Connection: close\r\n\r\n", wifiApiKey,
                wifiTemperature, wifiHumidity, wifiLuminosity);
            Uart_sendstring("AT+CIPSEND=");
            Uart_printbase(strlen(http_request), 10);
            Uart_sendstring("\r\n");
            while (!Wait_for(">"));
            thingSpeakState = 4;
            break;
        }
        case 4: {
            Uart_flush();
            char http_request[250];
            sprintf(http_request, "GET /update?api_key=%s&field1=%.2f&
                field2=%.2f&field3=%.2f HTTP/1.1\r\nHost: api.thingspeak.com\r\n
                Connection: close\r\n\r\n", wifiApiKey,
                wifiTemperature, wifiHumidity, wifiLuminosity);
            Uart_sendstring(http_request);
            if (Wait_for("SEND OK\r\n")) {
                thingSpeakState = 5;
            } else {
                thingSpeakState = 0;
            }
            break;
        }
        case 5: {
            LCD_PrintCenteredMessage("Datos enviados", "ThingSpeak");
            closeConnection();
            thingSpeakState = 0;
            break;
        }
        default: {
```

```
        closeConnection();  
        thingSpeakState = 0;  
        break;  
    }  
}
```

Código 6.2: Máquina de estados para la comunicación con ThingSpeak en `wifi_thingspeak.c`

La arquitectura y el flujo de la máquina de estados se representan de forma esquemática en el diagrama de la Figura 6.1, donde se observan las transiciones principales desde la inicialización hasta la confirmación final del envío.

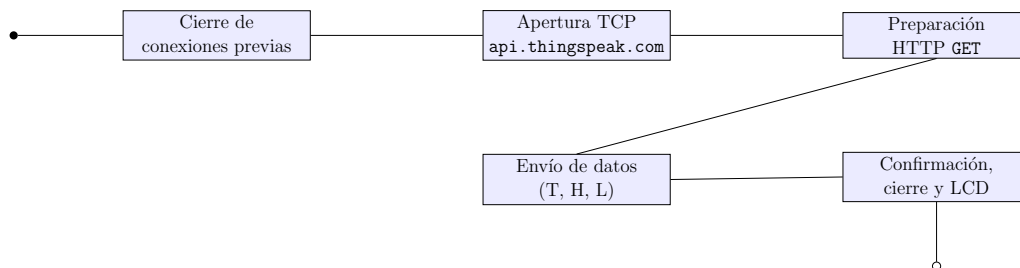


Figura 6.1: Diagrama de estados del envío a ThingSpeak.

6.1.4. Visualización en la nube

Una vez recibidos, ThingSpeak representa automáticamente los datos en gráficas dinámicas consultables vía web. La Figura 6.2 muestra un ejemplo de las gráficas generadas, mientras que la Figura 6.3 refleja indicadores y alarmas configurados para los actuadores.

El enlace para acceder a estos registros es <https://thingspeak.mathworks.com/channels/2859744>.

6.2. Broker MQTT mediante ESP8266

Además del envío a ThingSpeak, el sistema permite publicar los datos ambientales del invernadero al broker MQTT instalado en la Raspberry Pi. La comunicación se realiza desde el STM32 hacia el módulo ESP8266 a través del puerto UART, utilizando comandos AT y paquetes formateados según el protocolo MQTT.

6.2.1. Protocolo de comunicación

El microcontrolador STM32 envía comandos al ESP8266 mediante UART2. El ESP8266 actúa como cliente TCP conectado al broker MQTT, ubicado en la Raspberry Pi con IP 192.168.1.221 y puerto 1883.

La gestión de los eventos de comunicación se realiza con interrupciones (`HAL_UART_Receive_IT()`) y una **máquina de estados no bloqueante** que asegura la robustez de la conexión y evita bloqueos prolongados en el flujo principal.

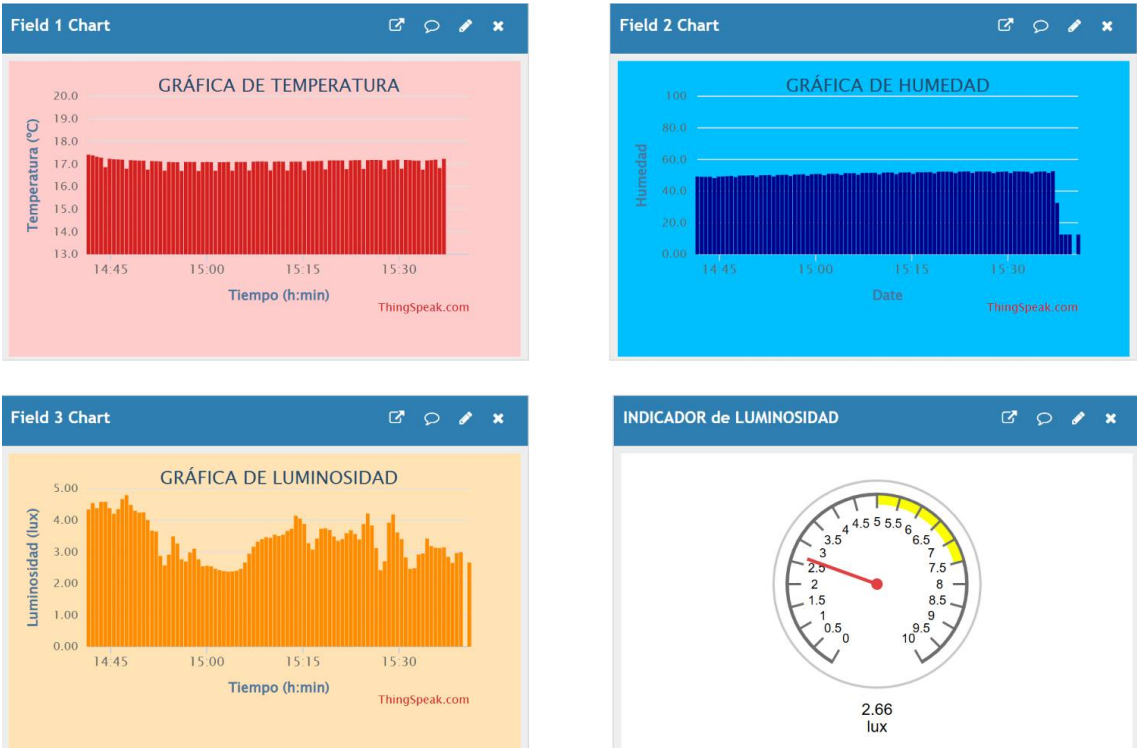


Figura 6.2: Gráficas en ThingSpeak de temperatura, humedad y luminosidad.

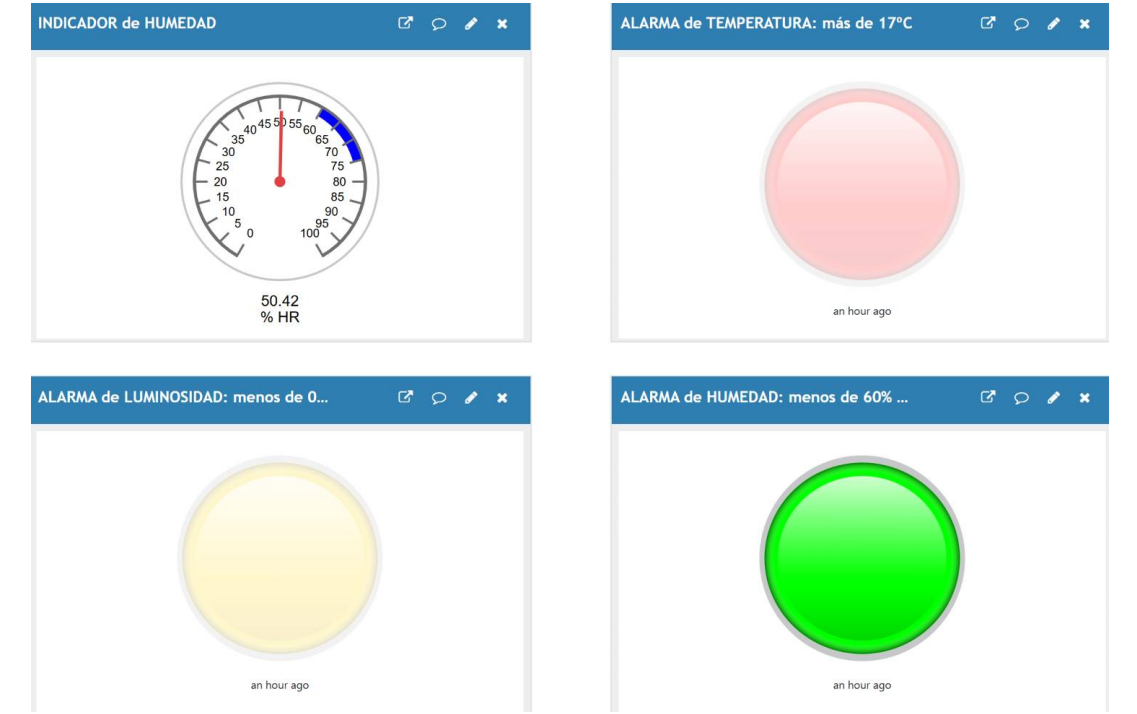


Figura 6.3: Indicadores y alarmas configurados en ThingSpeak.

6.2.2. Publicación de mensajes en formato JSON

El STM32 construye un mensaje unificado en formato JSON que agrupa temperatura, humedad y luminosidad. Este mensaje se publica bajo el topic `Invernadero/sensores`, definido en `mqtt_raspberry.h`.

Ejemplo de mensaje publicado:

```
{"temp":16.29,"hum":51.62,"lux":81.37}
```

En el código en C, la cadena JSON se genera con `sprintf()`:

```
char json[100];  
sprintf(json, "{\"temp\":%.2f,\"hum\":%.2f,\"lux\":%.2f}", t, h, l);
```

Código 6.3: Generación de la cadena JSON en `mqtt_raspberry.c`

6.2.3. Máquina de estados para MQTT broker

El envío de datos al broker MQTT en la Raspberry Pi se implementa mediante una **máquina de estados no bloqueante**, definida en los ficheros `mqtt_raspberry.c` y `mqtt_raspberry.h`. Este mecanismo permite gestionar la conexión, confirmación y publicación sin bloquear el flujo principal del programa, a la vez que incorpora comprobaciones de error y recuperación automática ante fallos de comunicación.

```
void processRpiStateMachine(void) {  
    static uint32_t stateTimeout = 0;  
    if (rpiState != rpi_prev_state) {  
        rpi_prev_state = rpiState;  
        rpi_enter_tick = HAL_GetTick();  
    }  
    if (rpiState != 0u && (HAL_GetTick() - rpi_enter_tick) >  
        ANTIBLOCK_STATE_MACHINE) {  
        alertaBloqueo = true;  
        return;  
    }  
  
    switch (rpiState) {  
        case 1: {  
            closeConnection();  
            rpiState = 2;  
            break;  
        }  
        case 2: {  
            Uart_flush();  
            char cmd[100];  
            sprintf(cmd, "AT+CIPSTART=\"%TCP\", \"%s\", %d\r\n", MQTT_BROKER_IP,  
                MQTT_BROKER_PORT);  
            Uart_sendstring(cmd);  
            if (!Wait_for("OK\r\n") && !Wait_for("CLOSED\r\n")) {  
                closeConnection();  
                rpiState = 0;  
                break;  
            }  
            rpiState = 3;  
            break;  
        }  
        case 3: {  
            Uart_flush();  
            mqtt_send_connect_packet();  
            rpiState = 4;  
            stateTimeout = HAL_GetTick();  
            break;  
        }  
    }  
}
```

```

case 4: {
    if (mqtt_wait_connack()) {
        rpiState = 5;
        break;
    } else if (HAL_GetTick() - stateTimeout > 5000) {
        closeConnection();
        rpiState = 0;
        break;
    }
    break;
}
case 5: {
    Uart_flush();
    mqtt_publish_unified_json(MQTT_TOPIC_JSON);
    rpiState = 6;
    stateTimeout = HAL_GetTick();
    break;
}
case 6: {
    closeConnection();
    rpiState = 0;
    LCD_PrintCenteredMessage("Datos enviados", "HomeAssistant");
    break;
}
default:{
    closeConnection();
    rpiState = 0;
    break;
}
}
}

```

Código 6.4: Máquina de estados para la comunicación con broker MQTT en `mqtt_raspberry.c`

La arquitectura y el flujo de la máquina de estados para la publicación MQTT se muestran en la Figura 6.4, donde se aprecian las transiciones desde el cierre de conexiones previas y la apertura TCP con envío del **CONNECT**, hasta la recepción de **CONNACK**, la publicación del mensaje JSON y el cierre con notificación en LCD.

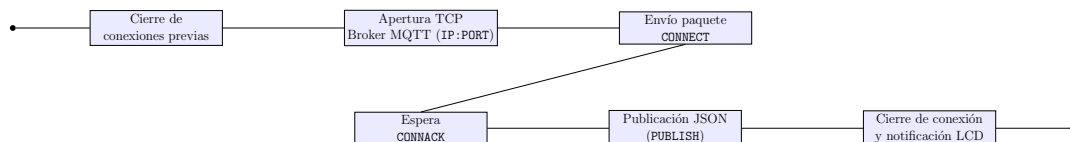


Figura 6.4: Diagrama de estados del envío de datos por MQTT.

6.2.4. Ventajas del formato unificado JSON

El uso de un único mensaje JSON que agrupa temperatura, humedad y luminosidad aporta varias ventajas. Por un lado, se reduce el tráfico en la red MQTT al transmitirse todos los valores en una sola publicación. Además, la integración en Home Assistant se simplifica al extraer todas las variables desde un único mensaje, lo que a su vez facilita el registro histórico y la visualización en plataformas compatibles.

6.2.5. Visualización en Home Assistant

La información publicada en el broker MQTT se integra en la interfaz de Home Assistant, donde se representa de forma gráfica junto con el resto de variables del invernadero. La Figura 7.7 muestra un ejemplo de esta visualización, con la evolución temporal de la temperatura en la aplicación web.

Capítulo 7

Raspberry Pi

7.1. Configuración de la Raspberry Pi

7.1.1. Instalación del sistema operativo Raspberry Pi OS Lite

Se comenzó utilizando una versión de Raspbian con entorno gráfico, pero se detectó que consumía demasiados recursos, afectando al rendimiento general del sistema. Para optimizar el funcionamiento, se decidió instalar **Raspberry Pi OS Lite**, una versión más ligera y sin entorno gráfico. Esta configuración permite trabajar cómodamente mediante conexión SSH desde el ordenador de sobremesa, mientras la Raspberry Pi permanece conectada por cable Ethernet en otra ubicación de la casa, junto al router. Además, se instaló la herramienta `bpytop` para monitorizar el uso de recursos del sistema de forma eficiente. En la Figura 7.1 se muestra una captura donde se aprecia el estado de la memoria y uso de disco.

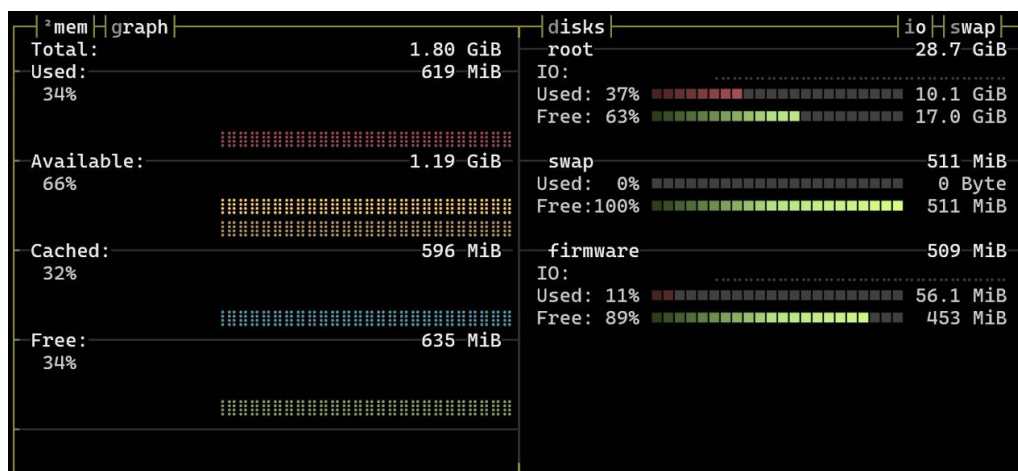


Figura 7.1: Monitorización de recursos en la Raspberry Pi utilizando `bpytop`.

Como se observa en la Figura 7.1, el sistema presenta un uso de memoria RAM de aproximadamente 619 MiB (34 % del total), manteniendo además unos 596 MiB en memoria cacheada. En cuanto al apartado de IO, se muestra la actividad de lectura y escritura en disco¹, la cual se mantiene en niveles bajos. De este modo, se

¹IO hace referencia a las operaciones de entrada/salida (*Input/Output*) realizadas en los dispositivos de almacenamiento, indicando la cantidad de datos leídos o escritos en un periodo de tiempo.

garantiza suficiente margen para la ejecución continua de los procesos del invernadero domótico.

7.1.2. Instalación del broker MQTT

Se procedió a instalar el broker MQTT Mosquitto directamente sobre el sistema operativo de la Raspberry Pi, sin utilizar contenedores Docker. Para ello, se actualizó el sistema e instalaron los paquetes necesarios mediante el siguiente comando:

```
sudo apt update
sudo apt install -y mosquitto mosquitto-clients
```

Código 7.1: Instalación de docker-compose en la Raspberry Pi

Una vez instalado, se habilitó el servicio para que se inicie automáticamente al arrancar el sistema:

```
sudo systemctl enable mosquitto
sudo systemctl start mosquitto
```

Código 7.2: Habilitación y arranque del servicio Mosquitto

Para comprobar su correcto funcionamiento, se puede utilizar el siguiente comando:

```
sudo systemctl status mosquitto
```

Código 7.3: Comprobación del estado del servicio Mosquitto

Este comando muestra el estado del servicio, permitiendo verificar si está activo y en ejecución. En la Figura 7.2 se muestra un ejemplo de salida cuando el broker está funcionando correctamente.

```
osuna@osuna:~$ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; preset: enabled)
   Active: active (running) since Sat 2025-05-03 09:59:09 CEST; 1 week 1 day ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
   Process: 566 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exited, status=0/SUCCESS)
   Process: 570 ExecStartPre=/bin/chown mosquitto /var/log/mosquitto (code=exited, status=0/SUCCESS)
   Process: 573 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exited, status=0/SUCCESS)
   Process: 574 ExecStartPre=/bin/chown mosquitto /run/mosquitto (code=exited, status=0/SUCCESS)
  Main PID: 575 (mosquitto)
    Tasks: 1 (limit: 1586)
       CPU: 66ms
   CGroup: /system.slice/mosquitto.service
           └─575 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

May 03 09:59:08 osuna systemd[1]: Starting mosquitto.service - Mosquitto MQTT Broker...
May 03 09:59:09 osuna mosquitto[575]: 1746259148: Loading config file /etc/mosquitto/conf.d/local.conf
May 03 09:59:09 osuna systemd[1]: Started mosquitto.service - Mosquitto MQTT Broker.
```

Figura 7.2: Estado del servicio Mosquitto activo en Raspberry Pi.

7.1.3. Instalación de Docker

Docker se instaló en la Raspberry Pi utilizando el *convenience script* oficial proporcionado en <https://get.docker.com/>. Este método permite una instalación rápida y automatizada de la última versión estable de Docker mediante los siguientes comandos:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Este método es apropiado para entornos de desarrollo como el presente proyecto, aunque no se recomienda en entornos de producción críticos debido a la falta de control detallado sobre la instalación.

Tras completar la instalación de Docker, se añadió el usuario **osuna** al grupo **docker** para poder ejecutar comandos de Docker sin necesidad de usar **sudo**, mediante el siguiente comando:

```
sudo usermod -aG docker osuna
```

Código 7.4: Instalación de Docker mediante el *convenience script*

Posteriormente, fue necesario cerrar sesión y volver a iniciarla para que los cambios surtieran efecto. Además, se instaló **docker-compose** para gestionar contenedores de manera más sencilla y estructurada mediante archivos **.yaml**. La instalación se realizó con el siguiente comando:

```
sudo apt install docker-compose
```

Código 7.5: Instalación de **docker-compose**

De esta forma, es posible levantar múltiples servicios de forma coordinada utilizando un único archivo de configuración.

La elección de utilizar Docker junto con Docker Compose responde a la necesidad de disponer de un entorno de servicios ligero, modular y fácilmente gestionable. Al encapsular cada servicio en su propio contenedor, se evita la instalación directa en el sistema operativo y se mejora la portabilidad del proyecto a otros sistemas en el futuro. Este enfoque, además, aporta una mayor flexibilidad y modularidad, ya que permite añadir, modificar o sustituir componentes de forma independiente sin afectar al resto de la arquitectura. La Figura 7.3 muestra esta estructura, con los servicios desplegados en la Raspberry Pi y el broker MQTT ejecutándose de forma independiente.

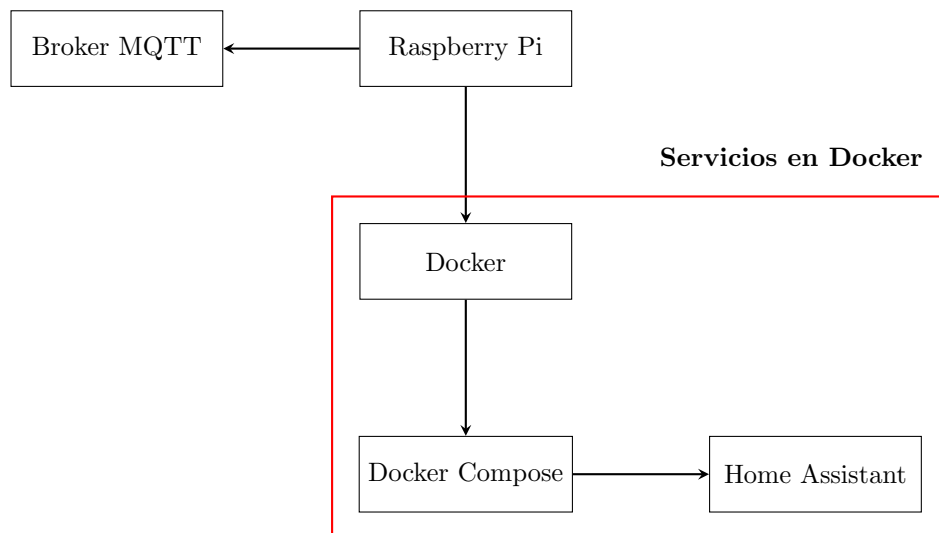


Figura 7.3: Arquitectura de servicios desplegada en la Raspberry Pi (Broker MQTT fuera de Docker).

La guía oficial utilizada para instalar Docker sobre Raspberry Pi OS Lite (basado en Debian) está disponible en:

<https://docs.docker.com/engine/install/debian/>

7.1.4. Acceso remoto mediante SSH

Para facilitar el desarrollo del software, se ha optado por utilizar **SSH** (Secure Shell), un protocolo de red criptográfico que permite establecer conexiones seguras entre dos sistemas mediante una arquitectura cliente-servidor. Esta elección permite ubicar la Raspberry Pi en cualquier lugar de la vivienda —en este caso, junto al router para aprovechar la conexión por cable Ethernet— y acceder a ella de forma remota desde el ordenador de sobremesa situado en otra habitación. Esta configuración mejora la flexibilidad y la comodidad durante el desarrollo y la administración del sistema.

A continuación, se representa mediante un diagrama la arquitectura de red y los servicios desplegados en la Raspberry Pi. En primer lugar, se muestra la conexión remota mediante **SSH** desde el ordenador de sobremesa, y posteriormente, la estructura interna explicada previamente. La Figura 7.4 ilustra esta configuración, destacando que el broker MQTT se ejecuta fuera del entorno Docker.

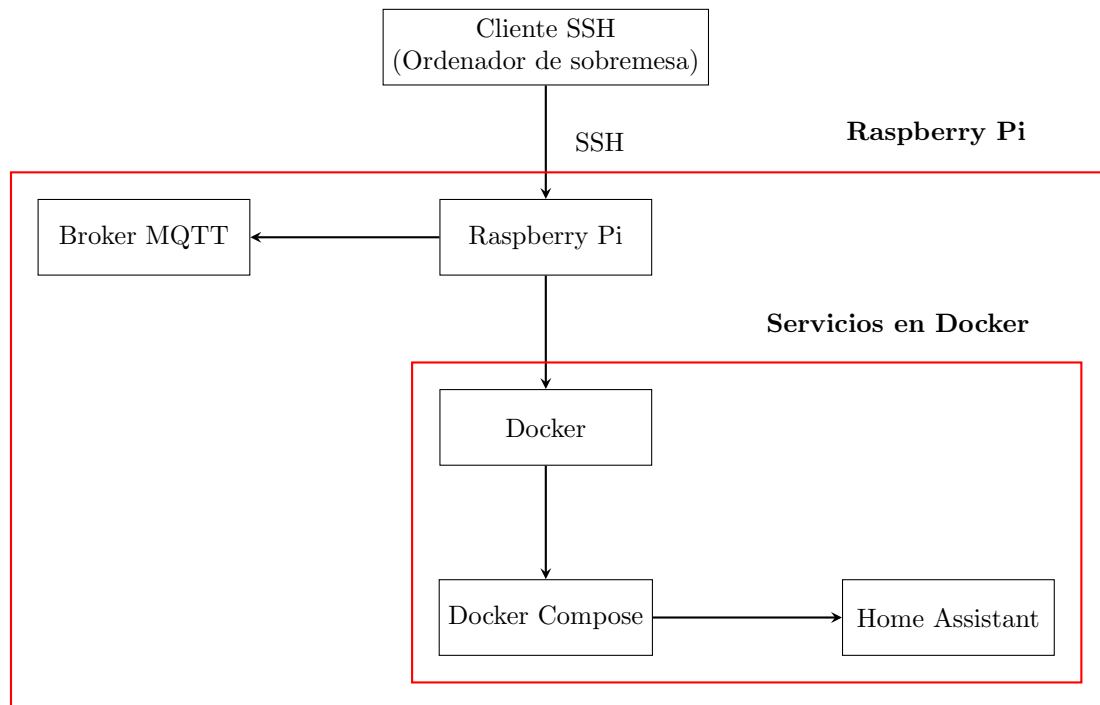


Figura 7.4: Esquema de la arquitectura: acceso remoto por SSH.

7.1.5. Instalación de Home Assistant mediante Docker Compose

Home Assistant se instaló dentro de un contenedor de Docker utilizando `docker-compose`, lo que facilita su despliegue, actualización y mantenimiento. Para ello, se preparó un archivo `docker-compose.yml` específico en la Raspberry Pi, en el que se definió el servicio con la imagen oficial de Home Assistant para arquitecturas ARM.

El archivo `docker-compose.yml` utilizado fue el siguiente:

```
services:
  homeassistant:
    container_name: homeassistant
```

```
image: "ghcr.io/home-assistant/home-assistant:stable"
volumes:
  - ./config:/config
  - /etc/localtime:/etc/localtime:ro
  - /run/dbus:/run/dbus:ro
restart: unless-stopped
privileged: true
network_mode: host
```

Código 7.6: Configuración del servicio Home Assistant en `docker-compose.yml`

A continuación, se ejecutaron los siguientes comandos para lanzar el servicio:

```
sudo apt install docker-compose
sudo docker-compose up -d
```

Código 7.7: Ejecución del servicio con `docker-compose`

Gracias al uso de `network_mode: host`, Home Assistant puede acceder directamente a los dispositivos y puertos de la red local, lo que facilita la integración con otros elementos del sistema como el broker MQTT instalado directamente en la Raspberry Pi.

El servicio quedó configurado para iniciarse automáticamente tras un reinicio, y puede ser gestionado fácilmente con comandos como `docker-compose stop`, `start`, o `logs`. Actualmente, el sistema funciona de forma estable y sin incidencias, con Home Assistant accesible desde el navegador web en la dirección IP de la Raspberry Pi por el puerto 8123.

La guía oficial utilizada para realizar esta instalación sobre Raspberry Pi OS Lite (basado en Debian) está disponible en [17]:

<https://www.home-assistant.io/installation/linux>

7.1.6. Asignación de IP fija en la Raspberry Pi

Se ha modificado la configuración de red de la Raspberry Pi para pasar de una dirección IP asignada dinámicamente (mediante DHCP) a una dirección IP estática y fija. Inicialmente, el sistema obtenía su IP de forma automática dentro del rango configurado en el router, lo que podía provocar cambios de dirección tras cada reinicio. Para evitarlo, se utilizó la herramienta `NetworkManager` mediante el comando `nmcli`, especificando manualmente las direcciones IP deseadas: `192.168.1.220` para la conexión por cable Ethernet y `192.168.1.221` para la interfaz inalámbrica WLAN, además de la puerta de enlace (`192.168.1.1`) y el servidor DNS. De este modo, se asegura que la Raspberry Pi mantenga siempre las mismas direcciones IP en la red local, facilitando así el acceso remoto constante mediante SSH.

7.2. Integración de MQTT con Home Assistant

En este apartado se describe el proceso de integración del protocolo MQTT entre la Raspberry Pi y la plataforma Home Assistant. Se abordan tanto la instalación y configuración del broker en la Raspberry Pi como la conexión de Home Assistant al mismo, incluyendo la creación de usuarios y la integración del protocolo desde la interfaz web.

Red Local

Red Local 192.168.1.1

Máscara de subred 255.255.255.0

DHCP ACTIVADO

Dirección IP inicio rango 192.168.1.33

Dirección IP fin rango 192.168.1.99

Configurador de servidores DNS (se recomienda no modificar)

Servidor DNS1 80.58.61.254 Servidor DNS2 80.58.61.250

Aplicar Cambios

Figura 7.5: Configuración de direcciones IP dinámicas (DHCP) en el router.

7.2.1. Instalación y configuración del broker MQTT en Raspberry Pi

En este proyecto se utiliza el protocolo MQTT como medio principal de comunicación entre el microcontrolador STM32 y el servidor domótico Home Assistant, ejecutado sobre una Raspberry Pi. Para ello, se ha implementado un broker MQTT local basado en el software **Mosquitto**, una de las soluciones más populares y ligeras para este propósito.

Contexto: Home Assistant Container vs Home Assistant OS

Home Assistant puede ejecutarse de distintas formas. En este proyecto se ha optado por utilizar **Home Assistant Container**, desplegado mediante **Docker Compose** sobre un sistema operativo Raspberry Pi OS Lite (basado en Debian). Esta versión ofrece flexibilidad y compatibilidad con sistemas ya existentes, pero a diferencia de **Home Assistant OS** o **Supervised**, no incluye una tienda de complementos (Add-on Store) ni el supervisor. Por tanto, cualquier funcionalidad adicional, como el broker MQTT, debe instalarse y configurarse manualmente mediante la consola.

Si bien esto requiere más configuración, también proporciona una gran versatilidad y libertad para instalar cualquier aplicación que se necesite. De haberse optado por **Home Assistant OS**, el sistema habría estado más limitado al ecosistema cerrado que impone ese entorno.

Configuración del broker

Una vez instalado, se configuró Mosquitto para operar de forma segura. Por defecto, el servicio permite conexiones anónimas, lo cual no es recomendable en un entorno de red doméstica. Para deshabilitar esta opción y establecer autenticación, se creó un archivo de configuración adicional:

```
sudo nano /etc/mosquitto/conf.d/local.conf
```

Código 7.8: Edición del archivo de configuración de Mosquitto

Con el siguiente contenido:

```
listener 1883
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Código 7.9: Contenido de `local.conf` para deshabilitar acceso anónimo

Esto indica al broker que escuche en el puerto 1883, rechace conexiones anónimas y utilice un archivo de contraseñas para validar usuarios.

Creación de usuarios MQTT

Para permitir el acceso controlado al broker, se creó un usuario denominado **osuna** mediante el siguiente comando:

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd osuna
```

Código 7.10: Creación de usuario **osuna** en Mosquitto

Este comando genera el archivo de contraseñas y añade un usuario llamado **osuna**. El sistema solicitará la contraseña deseada, que será encriptada y almacenada de forma segura. A partir de este momento, cualquier cliente MQTT deberá autenticarse con esas credenciales para poder conectarse.

7.2.2. Conexión de Home Assistant al broker MQTT

Ejecución de Home Assistant en contenedor Docker

En el presente proyecto, Home Assistant se ha desplegado como un contenedor Docker en la Raspberry Pi, utilizando la imagen oficial `ghcr.io/home-assistant/home-assistant:stable`. Esta forma de instalación permite aislar la aplicación del sistema operativo anfitrión y facilita su mantenimiento, actualización y portabilidad.

El contenedor se gestiona desde el sistema anfitrión mediante Docker CLI. Puede verificarse su estado con el comando:

```
docker ps
```

Código 7.11: Verificación del contenedor de Home Assistant en ejecución

y se observa un contenedor en ejecución con nombre **homeassistant**.

Este enfoque implica que Home Assistant se ejecuta en un entorno controlado, sin acceso directo a servicios del sistema como Mosquitto, a menos que se configuren explícitamente mediante red o volúmenes. No obstante, gracias al uso de puertos estándar, se puede comunicar perfectamente con servicios externos al contenedor, como el broker MQTT instalado en el sistema base.

Integración desde la interfaz web

La integración con Home Assistant se llevó a cabo desde su interfaz gráfica, una vez que el servicio Mosquitto quedó operativo en la Raspberry Pi. Para ello se recurrió a la sección **Configuración > Dispositivos y servicios**, donde se añadió el componente MQTT.

En este proyecto se optó por Home Assistant Core ejecutado en un contenedor Docker, lo que implicaba la ausencia de un broker embebido —a diferencia de Home Assistant OS— y obligaba a introducir manualmente los parámetros de conexión. Por esta razón se configuró la Raspberry Pi como servidor MQTT, empleando la dirección IP 192.168.1.220 cuando se utilizaba la conexión por Ethernet y la dirección 192.168.1.221 en el caso de la interfaz inalámbrica. En ambos casos se recurrió al puerto estándar 1883, correspondiente a conexiones sin TLS, y se definieron credenciales específicas mediante un usuario `osuna` y su contraseña asociada, generada previamente con `mosquitto_passwd`.

Una vez comprobada la conexión, Home Assistant pasó a escuchar de forma automática el topic reservado `homeassistant/#`. Gracias a ello, los dispositivos que enviaban mensajes bajo el estándar de *MQTT Discovery* —un sistema de autoconfiguración que permite anunciar sus características mediante mensajes JSON en topics específicos— fueron identificados sin intervención manual, lo que simplificó la incorporación de nuevos equipos y favoreció el uso directo de sus datos en la definición de automatizaciones.

En la Figura 7.6 se muestra este proceso de integración en Home Assistant.

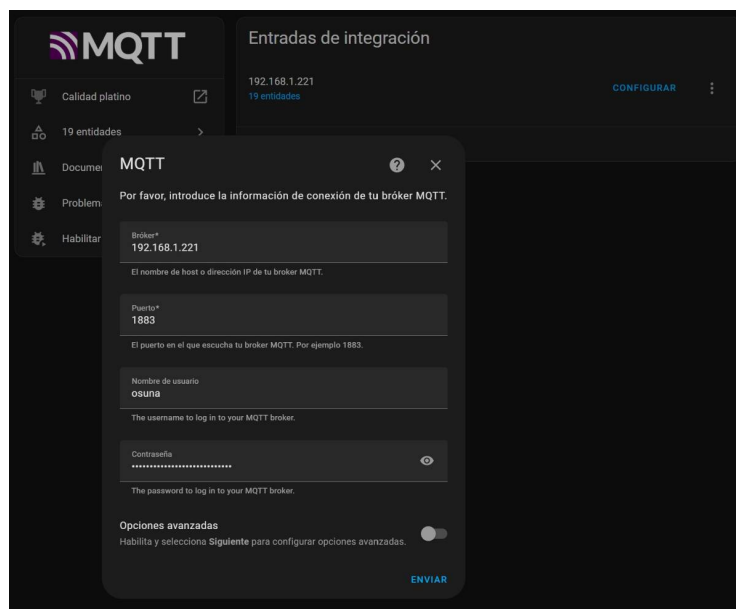


Figura 7.6: Configuración de la integración MQTT en Home Assistant.

7.3. Código `configuration.yaml`

El archivo `configuration.yaml` constituye el núcleo de la configuración en Home Assistant. En este proyecto se empleó para definir los sensores, actuadores y parámetros de control que intercambian información mediante MQTT con el microcontrolador STM32 y la Raspberry Pi. De esta manera, Home Assistant pudo interpretar los datos recibidos de temperatura, humedad y luminosidad, además de ofrecer al usuario opciones de control tanto manual como automático sobre los actuadores del invernadero.

Con el fin de mantener una estructura clara, el archivo se organizó en bloques

diferenciados según el tipo de entidad declarada: **sensor**, **switch**, **select**, **number** y **binary_sensor**. Cada sección se presenta acompañada del fragmento de código YAML correspondiente y de una explicación de su función dentro del sistema.

Es importante destacar que, antes de definir las entidades MQTT, el archivo ya incluía una serie de directivas que aparecen por defecto en cualquier instalación de Home Assistant y que no guardan relación con la creación del broker:

```
# Loads default set of integrations. Do not remove.
default_config:

# Load frontend themes from the themes folder
frontend:
  themes: !include_dir_merge_named themes

automation: !include automations.yaml
script: !include scripts.yaml
scene: !include scenes.yaml
```

Código 7.12: Directivas por defecto en `configuration.yaml`

La instrucción **default_config** carga las integraciones básicas de la plataforma, necesarias para habilitar la interfaz gráfica y servicios fundamentales.

Asimismo, se encuentra la sección **frontend**, que permite cargar temas personalizados desde el directorio **themes**, y las referencias externas a otros ficheros de configuración —**automations.yaml**, **scripts.yaml** y **scenes.yaml**— que Home Assistant genera por defecto para mantener la configuración modular y organizada.

7.3.1. Sensores MQTT (sensor)

Una vez aclarada la configuración base de Home Assistant, se añadieron manualmente las entidades correspondientes a los sensores que envían información desde el STM32 a través de MQTT.

```
mqtt:
  sensor:
    - name: "Temperatura Invernadero"
      unique_id: sensor_temperatura_invernadero
      state_topic: "Invernadero/sensores"
      unit_of_measurement: "C"
      value_template: "{{ value_json.temp }}"
    - name: "Humedad Invernadero"
      unique_id: sensor_humedad_invernadero
      state_topic: "Invernadero/sensores"
      unit_of_measurement: "%"
      value_template: "{{ value_json.hum }}"
    - name: "Luminosidad Invernadero"
      unique_id: sensor_luminosidad_invernadero
      state_topic: "Invernadero/sensores"
      unit_of_measurement: "lux"
      value_template: "{{ value_json.lux }}"
```

Código 7.13: Definición de sensores MQTT (**sensor**) en `configuration.yaml`

En este bloque se declararon tres sensores MQTT: temperatura, humedad y luminosidad. Todos comparten el mismo **state_topic** (**Invernadero/sensores**), ya que los datos se envían en un único mensaje JSON desde el microcontrolador. El parámetro **value_template** permitió extraer cada valor específico del objeto JSON recibido. La opción **unit_of_measurement** definió las unidades que se muestran en la interfaz, mientras que **unique_id** garantizó la identificación inequívoca de cada entidad dentro de Home Assistant.

7.3.2. Actuadores y control (switch, select)

Además de los sensores, en el archivo `configuration.yaml` se declararon los actuadores y los mecanismos de control asociados. Para ello se emplearon entidades de tipo `switch` y `select`, que permiten gestionar el ventilador, la bomba de agua, el foco y el modo automático del sistema. Asimismo, se reservaron *topics* para el control RGB de los ventiladores, con el fin de ofrecer una interfaz completa en Home Assistant.

```
switch:
  - name: "Ventilador Invernadero"
    unique_id: switch_ventilador_invernadero
    state_topic: "Invernadero/actuadores/ventilador/estado"
    command_topic: "Invernadero/actuadores/ventilador/control"
    payload_on: "ON"
    payload_off: "OFF"
    retain: true
    optimistic: true

  - name: "Bomba Invernadero"
    unique_id: switch_bomba_invernadero
    state_topic: "Invernadero/actuadores/bomba/estado"
    command_topic: "Invernadero/actuadores/bomba/control"
    payload_on: "ON"
    payload_off: "OFF"
    retain: true
    optimistic: true

  - name: "Foco Invernadero"
    unique_id: switch_foco_invernadero
    state_topic: "Invernadero/actuadores/foco/estado"
    command_topic: "Invernadero/actuadores/foco/control"
    payload_on: "ON"
    payload_off: "OFF"
    retain: true
    optimistic: true

  - name: "Modo Automatico"
    unique_id: switch_modos_automatico
    state_topic: "Invernadero/modo/automatico/estado"
    command_topic: "Invernadero/modo/automatico/control"
    payload_on: "ON"
    payload_off: "OFF"
    retain: true
    optimistic: true

  - name: "RGB Power"
    unique_id: rgb_power
    state_topic: "Invernadero/actuadores/rgb/estado"
    command_topic: "Invernadero/actuadores/rgb/control"
    payload_on: "ON"
    payload_off: "OFF"
    retain: true
    optimistic: true

select:
  - name: "RGB Mode"
    unique_id: rgb_mode
    state_topic: "Invernadero/actuadores/rgb/mode/state"
    command_topic: "Invernadero/actuadores/rgb/mode/set"
    options: ["off", "rojo", "verde", "azul", "arcoiris"]
    optimistic: true
    retain: true
```

Código 7.14: Definición de actuadores y control (switch, select) en `configuration.yaml`

En este bloque, cada `switch` permitió activar o desactivar un actuador concreto (ventilador, bomba, foco) o el propio modo automático del sistema. Por su parte,

el `select` definió un conjunto de opciones para controlar los efectos RGB de los ventiladores.

No obstante, durante la implementación se comprobó que no era posible accionar correctamente la iluminación RGB: los pines GPIO de la Raspberry Pi proporcionan una señal de 3,3 V, insuficiente para controlar un módulo que requiere niveles lógicos de 5 V. Para solventar esta limitación sería necesario incorporar un circuito de adaptación de nivel (por ejemplo, un *level shifter* con transistores MOSFET o un controlador dedicado a 5 V). Esta mejora se plantea como una posible extensión futura del proyecto, ya que permitiría aprovechar plenamente las funcionalidades RGB previstas en la configuración.

7.3.3. Parámetros ajustables (number)

Además de los sensores y actuadores, se configuraron entidades de tipo `number` para definir umbrales de temperatura, humedad y luminosidad directamente desde la interfaz de Home Assistant. Estos valores son transmitidos al microcontrolador mediante MQTT y empleados en el modo automático para decidir la activación de los actuadores.

```
number:
  - name: "Umbral Temp Min"
    unique_id: umbral_temp_min
    command_topic: "Invernadero/umbrales/temp/min/set"
    state_topic: "Invernadero/umbrales/temp/min/state"
    min: 10
    max: 50
    step: 0.5
    unit_of_measurement: "C"
    optimistic: true
    retain: true

  - name: "Umbral Temp Max"
    unique_id: umbral_temp_max
    command_topic: "Invernadero/umbrales/temp/max/set"
    state_topic: "Invernadero/umbrales/temp/max/state"
    min: 10
    max: 50
    step: 0.5
    unit_of_measurement: "C"
    optimistic: true
    retain: true

  - name: "Umbral Humedad Min"
    unique_id: umbral_humedad_min
    command_topic: "Invernadero/umbrales/humedad/min/set"
    state_topic: "Invernadero/umbrales/humedad/min/state"
    min: 0
    max: 100
    step: 1
    unit_of_measurement: "%"
    optimistic: true
    retain: true

  - name: "Umbral Humedad Max"
    unique_id: umbral_humedad_max
    command_topic: "Invernadero/umbrales/humedad/max/set"
    state_topic: "Invernadero/umbrales/humedad/max/state"
    min: 0
    max: 100
    step: 1
    unit_of_measurement: "%"
    optimistic: true
    retain: true
```

```
- name: "Umbral Luminosidad Min"
  unique_id: umbral_luminosidad_min
  command_topic: "Invernadero/umbrales/luz/min/set"
  state_topic: "Invernadero/umbrales/luz/min/state"
  min: 0
  max: 60
  step: 0.5
  unit_of_measurement: "lx"
  optimistic: true
  retain: true

- name: "Umbral Luminosidad Max"
  unique_id: umbral_luminosidad_max
  command_topic: "Invernadero/umbrales/luz/max/set"
  state_topic: "Invernadero/umbrales/luz/max/state"
  min: 0
  max: 250
  step: 1
  unit_of_measurement: "lx"
  optimistic: true
  retain: true
```

Código 7.15: Definición de parámetros ajustables (number) en configuration.yaml

En este bloque se establecieron seis parámetros ajustables: mínimo y máximo de temperatura, humedad y luminosidad. Cada entidad incluye un rango de valores (min, max), un incremento de ajuste (step) y la unidad de medida correspondiente. Al modificarse desde la interfaz, Home Assistant publica el nuevo valor en el command_topic y lo mantiene sincronizado en el state_topic, asegurando que el STM32 disponga siempre de los umbrales actualizados para la gestión automática del invernadero.

7.3.4. Sensores lógicos (binary_sensor)

Finalmente, se añadieron entidades de tipo binary_sensor, orientadas a representar estados lógicos del sistema en forma de variables booleanas (activo/inactivo). Estas entidades resultan especialmente útiles para crear automatizaciones en Home Assistant, ya que permiten definir condiciones basadas en el estado de los actuadores o del propio modo de funcionamiento.

```
binary_sensor:
- name: "Activacion Ventilador"
  unique_id: bin_act_ventilador
  state_topic: "Invernadero/activacion/ventilador"
  payload_on: "ON"
  payload_off: "OFF"
  device_class: power

- name: "Activacion Bomba"
  unique_id: bin_act_bomba
  state_topic: "Invernadero/activacion/bomba"
  payload_on: "ON"
  payload_off: "OFF"
  device_class: power

- name: "Activacion Foco"
  unique_id: bin_act_foco
  state_topic: "Invernadero/activacion/foco"
  payload_on: "ON"
  payload_off: "OFF"
  device_class: light

- name: "Modo Automatico Activo"
```

```
unique_id: modo_automatico_activo
state_topic: "Invernadero/modo/automatico/estado"
payload_on: "ON"
payload_off: "OFF"
device_class: running
```

Código 7.16: Definición de sensores lógicos (`binary_sensor`) en `configuration.yaml`

En este bloque se definieron cuatro sensores lógicos: activación de ventilador, bomba y foco, además del estado del modo automático. Cada entidad escucha un `state_topic` específico y cambia entre `ON` y `OFF` según el mensaje recibido. El parámetro `device_class` permitió mostrar en la interfaz un icono acorde con el tipo de dispositivo (energía para ventilador y bomba, luz para foco, estado de ejecución para el modo automático). De esta manera, Home Assistant pudo reflejar en tiempo real si cada actuador estaba activo o inactivo, facilitando tanto la supervisión como la creación de automatizaciones dependientes de estos estados.

7.3.5. Notas de validación y reinicio

Cada vez que se introdujeron cambios en el archivo `configuration.yaml`, fue necesario validar la sintaxis desde la propia interfaz de Home Assistant. Esta comprobación permitió detectar posibles errores de indentación o incompatibilidades en la estructura YAML antes de aplicar la nueva configuración.

Una vez confirmada la validez del archivo, se procedió a reiniciar Home Assistant para que los cambios entrasen en funcionamiento. Este reinicio resultó imprescindible tanto en la incorporación inicial de los bloques `sensor`, `switch`, `select`, `number` y `binary_sensor`, como en las posteriores modificaciones de los `topics` y umbrales empleados por el sistema.

De esta forma se garantizó que las nuevas entidades estuvieran disponibles en la interfaz y listas para ser utilizadas en automatizaciones o en el control manual de los actuadores.

7.4. Código `automations.yaml`

El archivo `automations.yaml` recoge la lógica de automatización implementada en Home Assistant. En este proyecto se utilizó para definir las reglas que gobiernan el funcionamiento automático del sistema, basadas en los valores de los sensores y en los umbrales configurados por el usuario.

De este modo fue posible, por ejemplo, activar el ventilador cuando la temperatura superaba un límite, encender la bomba de agua en condiciones de baja humedad o accionar el foco cuando la luminosidad descendía por debajo del umbral mínimo. También se incluyó una automatización dedicada a la publicación periódica del estado del modo automático, lo que permitió mantener sincronizado al microcontrolador con la plataforma.

A continuación se presentan los bloques de automatización desarrollados, organizados según su función: publicación de estado, control por umbrales y condiciones auxiliares.

7.4.1. Publicación de estado de modo automático

En esta sección se definen las automatizaciones destinadas a inicializar y mantener sincronizado el estado del modo automático entre Home Assistant y el broker MQTT en la Raspberry Pi. Se busca garantizar que al arrancar la plataforma no queden estados indeterminados y que cualquier cambio realizado en el control se refleje en la variable de estado retenida.

```
# Forzar Modo Automatico = OFF al iniciar / reconectar
- alias: "Modo automatico: forzar OFF al arrancar"
  id: modo_automatico_off_arranque
  mode: single
  trigger:
    - platform: homeassistant
      event: start
    - platform: mqtt
      topic: homeassistant/status
      payload: "online"
  action:
    - delay: "00:00:02"
    - service: mqtt.publish
      data:
        topic: Invernadero/modo/automatico/control
        payload: "OFF"
        retain: true

# Reflejar control -> estado (unica fuente del estado retenido)
- alias: "Modo automatico - reflejar control en estado"
  id: modo_auto_echo_control
  mode: restart
  trigger:
    - platform: mqtt
      topic: Invernadero/modo/automatico/control
  action:
    - service: mqtt.publish
      data:
        topic: Invernadero/modo/automatico/estado
        payload: "{{ trigger.payload }}"
        retain: true
```

Código 7.17: Automatizaciones de publicación de estado del modo automático en `automations.yaml`

En este bloque se implementaron dos automatizaciones clave: la primera fuerza el valor **OFF** del modo automático cada vez que Home Assistant arranca o se reconecta, mientras que la segunda refleja cualquier cambio en el control dentro del estado retenido. De este modo, se asegura la coherencia entre la interfaz de usuario y el registro de los `topics` dentro del broker.

7.4.2. Reglas de control por umbrales

Estas automatizaciones permiten activar o desactivar los actuadores en función de los valores de temperatura, humedad y luminosidad medidos en el invernadero. El objetivo es que, cuando el sistema se encuentra en modo automático, los dispositivos se enciendan o apaguen de acuerdo con los umbrales definidos por el usuario, garantizando así unas condiciones ambientales óptimas.

```
# --- VENTILADOR (T) ---
# ON si T > Tmax
- alias: "AUTO Ventilador ON por Temperatura"
  id: auto_vent_on_t
  mode: restart
  trigger:
    - platform: state
```

```
    entity_id:
      - sensor.temperatura_invernadero
      - number.umbral_temp_max
      - switch.modo_automatico
  condition:
    - condition: state
      entity_id: switch.modo_automatico
      state: "on"
    - condition: template
      value_template: >
        {{ states('sensor.temperatura_invernadero')|float(0) >
          states('number.umbral_temp_max')|float(0) }}
  action:
    - service: mqtt.publish
      data: {topic: Invernadero/actuadores/ventilador/control, payload: "ON",
        retain: true}

# OFF al estar 2 min por debajo de Tmax
- alias: "AUTO Ventilador OFF por Temperatura"
  id: auto_vent_off_t
  mode: restart
  trigger:
    - platform: template
      value_template: >
        {{ states('sensor.temperatura_invernadero')|float(0) <
          states('number.umbral_temp_max')|float(0) }}
      for: "00:00:00"
  condition:
    - condition: state
      entity_id: switch.modo_automatico
      state: "on"
  action:
    - service: mqtt.publish
      data: {topic: Invernadero/actuadores/ventilador/control, payload: "OFF",
        retain: true}

# --- BOMBA (H) ---
# ON si H < Hmin
- alias: "AUTO Bomba ON por Humedad Baja"
  id: auto_bomba_on_h
  mode: restart
  trigger:
    - platform: state
      entity_id:
        - sensor.humedad_invernadero
        - number.umbral_humedad_min
        - switch.modo_automatico
  condition:
    - condition: state
      entity_id: switch.modo_automatico
      state: "on"
    - condition: template
      value_template: >
        {{ states('sensor.humedad_invernadero')|float(0) <
          states('number.umbral_humedad_min')|float(0) }}
  action:
    - service: mqtt.publish
      data: {topic: Invernadero/actuadores/bomba/control, payload: "ON",
        retain: true}

# OFF al estar 2 min >= Hmin
- alias: "AUTO Bomba OFF por Humedad en rango"
  id: auto_bomba_off_h
  mode: restart
  trigger:
    - platform: template
      value_template: >
        {{ states('sensor.humedad_invernadero')|float(0) >=
          states('number.umbral_humedad_min')|float(0) }}
      for: "00:00:00"
```

```
condition:
  - condition: state
    entity_id: switch.modo_automatico
    state: "on"
action:
  - service: mqtt.publish
    data: {topic: Invernadero/actuadores/bomba/control, payload: "OFF",
      retain: true}

# --- FOCO (LUX) ---
# ON si L < Lmin
- alias: "AUTO Foco ON por Luminosidad Baja"
  id: auto_foco_on_1
  mode: restart
  trigger:
    - platform: state
      entity_id:
        - sensor.luminosidad_invernadero
        - number.umbral_luminosidad_min
        - switch.modo_automatico
  condition:
    - condition: state
      entity_id: switch.modo_automatico
      state: "on"
    - condition: template
      value_template: >
        {{ states('sensor.luminosidad_invernadero')|float(0) <
          states('number.umbral_luminosidad_min')|float(0) }}
  action:
    - service: mqtt.publish
      data: {topic: Invernadero/actuadores/foco/control, payload: "ON",
        retain: true}

# OFF al estar 2 min > Lmin
- alias: "AUTO Foco OFF por Luminosidad en rango"
  id: auto_foco_off_1
  mode: restart
  trigger:
    - platform: template
      value_template: >
        {{ states('sensor.luminosidad_invernadero')|float(0) >=
          states('number.umbral_luminosidad_min')|float(0) }}
      for: "00:00:00"
  condition:
    - condition: state
      entity_id: switch.modo_automatico
      state: "on"
  action:
    - service: mqtt.publish
      data: {topic: Invernadero/actuadores/foco/control, payload: "OFF",
        retain: true}
```

Código 7.18: Automatizaciones de control por umbrales en `automations.yaml`

En la práctica, el comportamiento se resume en las siguientes reglas:

- **Ventilador:** se activa cuando la temperatura supera el umbral máximo definido y se apaga cuando desciende por debajo de este valor.
- **Bomba de agua:** se activa cuando la humedad cae por debajo del umbral mínimo configurado y se desactiva cuando alcanza de nuevo valores iguales o superiores.
- **Foco:** se enciende cuando la luminosidad está por debajo del mínimo establecido y se apaga al recuperarse los niveles adecuados.

En este proyecto, el retardo (`for`) se ha configurado con valor 0, de modo que la conmutación es instantánea. No obstante, este parámetro puede emplearse como

mecanismo de histéresis o *antirrebote*, evitando cambios demasiado frecuentes en caso de fluctuaciones rápidas de las variables medidas.

7.4.3. Eventos y condiciones auxiliares

Además de las reglas principales de control por umbrales, se añadieron automatizaciones auxiliares que permiten mantener la coherencia en el sistema. En particular, se diseñó una rutina para guardar los valores de los umbrales ajustados por el usuario, así como el modo de funcionamiento del módulo RGB. De esta forma, cualquier cambio en los parámetros se refleja automáticamente en los `topics` de estado, asegurando su persistencia y disponibilidad para el resto de entidades.

```
# Para guardar los valores de umbrales
- alias: "Echo SET->STATE (umbrales y RGB mode)"
  id: echo_set_state_umbrales_rgb
  mode: parallel
  trigger:
    - platform: mqtt
      topic: Invernadero/umbrales/+/+/set
    - platform: mqtt
      topic: Invernadero/actuadores/rgb/mode/set
  action:
    - variables:
        st: "{{ trigger.topic | replace('/set','/state') }}"
    - service: mqtt.publish
      data:
        topic: "{{ st }}"
        payload: "{{ trigger.payload }}"
        retain: true
```

Código 7.19: Automatizaciones auxiliares de sincronización en `automations.yaml`

En este bloque se observa que, al recibir un mensaje en un `topic` de tipo `/set`, se publica automáticamente el mismo valor en el correspondiente `/state`. Gracias a ello, los umbrales configurados y el modo RGB quedan retenidos en el broker MQTT, evitando pérdidas de información y facilitando la sincronización entre Home Assistant y los dispositivos conectados.

7.5. Código de control de actuadores con *script* Python

Además de la configuración en Home Assistant mediante ficheros YAML, se desarrollaron *scripts* en Python para gestionar directamente los actuadores conectados a la Raspberry Pi. Estos programas permitieron establecer la comunicación con el broker MQTT, suscribirse a los `topics` definidos y ejecutar acciones sobre los dispositivos físicos (ventilador, bomba, foco e iluminación RGB).

El uso de Python ofreció mayor flexibilidad para integrar bibliotecas externas y controlar los pines GPIO de la Raspberry Pi, aunque se identificó una limitación importante: las salidas de 3,3 V de la placa no resultaron suficientes para accionar correctamente el control RGB, que requería niveles de 5 V. Por este motivo, el código referente al RGB se decidió suprimir.

En esta sección se muestra la estructura del código empleado, las dependencias necesarias, el manejo de los GPIO y la lógica de suscripción y publicación en MQTT que permitió coordinar los actuadores con la plataforma domótica.

7.5.1. Descripción del servicio y dependencias

Para ejecutar el control de actuadores mediante Python en la Raspberry Pi, se definió un entorno virtual con las librerías necesarias y un servicio de `systemd` que permitiera su ejecución automática al inicio del sistema. Las principales dependencias incluyeron `paho-mqtt` para la comunicación con el broker MQTT y `RPi.GPIO` para la gestión de los pines de entrada/salida digitales.

```
# Crear entorno virtual
python3 -m venv .venv
source .venv/bin/activate

# Instalar librerías necesarias
pip install paho-mqtt RPi.GPIO
```

Código 7.20: Instalación de dependencias en el entorno virtual de Python

Asimismo, se configuró un servicio de `systemd` para garantizar que el script de control se ejecutase en segundo plano de forma continua y se reiniciara en caso de fallo o reinicio de la Raspberry Pi. Este servicio se creó en la dirección:

```
sudo nano /etc/systemd/system/actuadores.service}
```

Código 7.21: Creación del servicio `systemd` para el script de actuadores

El contenido de este archivo se muestra en el Código 7.22, donde se define la ejecución automática del script `actuadores_mqtt.py`.

```
[Unit]
Description=Control Actuadores Invernadero
After=network.target

[Service]
ExecStart=/home/osuna/rpi_relay_4/.venv/bin/python
           /home/osuna/rpi_relay_4/actuadores_mqtt.py
WorkingDirectory=/home/osuna/rpi_relay_4
Restart=always
User=osuna

[Install]
WantedBy=multi-user.target
```

Código 7.22: Servicio `systemd` para ejecutar el script de control en la Raspberry Pi

De este modo, el servicio permitió integrar el control de actuadores en el arranque del sistema operativo, evitando la necesidad de lanzar manualmente el script tras cada reinicio de la Raspberry Pi.

7.5.2. Estructura del código (`actuadores_mqtt.py`)

El control de los actuadores se centralizó en un único *script* en Python, `actuadores_mqtt.py`, encargado de gestionar la comunicación con el broker MQTT y de reflejar las órdenes de Home Assistant sobre los dispositivos físicos. Su estructura puede dividirse en varios bloques diferenciados:

- En primer lugar, se definen los **parámetros de conexión** al broker MQTT, el comportamiento de la lógica de activación (`ACTIVE_HIGH`) y el diccionario de dispositivos, donde se especifica para cada actuador el pin GPIO, el `topic` de control y el `topic` de estado.

- A continuación, se realiza la **inicialización de los pines GPIO**, configurando cada salida en el estado inicial correspondiente y almacenando un registro del estado actual de cada dispositivo.
- Posteriormente, se incluyen **funciones auxiliares** como el ajuste de salidas GPIO, la normalización de mensajes recibidos en distintos formatos (ON/OFF, 1/0, true/false, encender/apagar) y la publicación de estados en el broker.
- El bloque central define las **funciones de comunicación MQTT**: `on_connect`, donde se establecen las suscripciones e inicializa el estado de los actuadores; y `on_message`, que procesa cada mensaje entrante y aplica el cambio correspondiente al actuador asociado.
- Finalmente, la función `main()` instancia el cliente MQTT, configura los manejadores de eventos, define el *Last Will* para notificar desconexiones como `OFFLINE`, y mantiene en ejecución continua el bucle de escucha de mensajes.

La implementación completa del código, junto con su explicación detallada, se presenta en la siguiente subsección (7.5.3).

7.5.3. Código completo del *script* Python

En esta subsección se presenta la implementación íntegra del script de control de actuadores, `actuadores_mqtt.py`, donde se integran todas las funciones descritas previamente: inicialización de pines GPIO, gestión de suscripciones y publicaciones en MQTT, y lógica de control de los dispositivos del invernadero.

```
#!/usr/bin/env python3
import time
import logging
import RPi.GPIO as GPIO
import paho.mqtt.client as mqtt

# ===== AJUSTES =====
MQTT_HOST = "192.168.1.221"
MQTT_PORT = 1883
MQTT_USER = None
MQTT_PASS = None

ACTIVE_HIGH = True # KS0212: HIGH = rele ON

DEVICES = {
    "ventilador": {
        "pin": 4, # BCM
        "control": "Invernadero/actuadores/ventilador/control",
        "state": "Invernadero/actuadores/ventilador/estado",
    },
    "bomba": {
        "pin": 22, # BCM
        "control": "Invernadero/actuadores/bomba/control",
        "state": "Invernadero/actuadores/bomba/estado",
    },
    "foco": {
        "pin": 6, # BCM
        "control": "Invernadero/actuadores/foco/control",
        "state": "Invernadero/actuadores/foco/estado",
    },
}

# =====

logging.basicConfig(level=logging.INFO, format="%(asctime)s %(message)s")
```

```
GPIO.setmode(GPIO.BCM)
for d in DEVICES.values():
    GPIO.setup(d["pin"], GPIO.OUT, initial=GPIO.LOW if ACTIVE_HIGH else GPIO.HIGH)

current = {name: None for name in DEVICES.keys()}

def gpio_set(pin: int, on: bool):
    GPIO.output(pin, GPIO.HIGH if (on == ACTIVE_HIGH) else GPIO.LOW)

def norm(payload: bytes) -> str:
    s = payload.decode(errors="ignore").strip().lower()
    if s in ("on", "1", "true", "encender", "enable", "activo"): return "ON"
    if s in ("off", "0", "false", "apagar", "disable", "inactivo"): return "OFF"
    return "UNKNOWN"

def publish_state(c, topic: str, state: str):
    c.publish(topic, state, qos=1, retain=True)

def device_by_topic(topic: str):
    for name, d in DEVICES.items():
        if topic == d["control"]:
            return name
    return None

def apply_state(c, name: str, new_state: str):
    if new_state not in ("ON", "OFF"):
        return
    if current.get(name) != new_state:
        logging.info(f"{name}: {current.get(name)} -> {new_state}")
        gpio_set(DEVICES[name]["pin"], new_state == "ON")
        current[name] = new_state
        publish_state(c, DEVICES[name]["state"], new_state)
    else:
        logging.info(f"{name}: ignorado (sin cambio) -> {new_state}")

def on_connect(c, u, f, rc):
    logging.info(f"MQTT conectado (rc={rc})")
    for name, d in DEVICES.items():
        c.subscribe(d["control"], qos=1)
        publish_state(c, d["state"], "ONLINE")
        gpio_set(d["pin"], False)
        current[name] = "OFF"
        publish_state(c, d["state"], "OFF")

def on_message(c, u, msg):
    name = device_by_topic(msg.topic)
    if not name:
        logging.warning(f"Topic no manejado: {msg.topic}")
        return
    new_state = norm(msg.payload)
    if new_state == "UNKNOWN":
        logging.warning(f"{name}: payload no valido: {msg.payload!r}")
        return
    apply_state(c, name, new_state)

def main():
    client = mqtt.Client(client_id="rpi-actuadores")
    client.on_connect = on_connect
    client.on_message = on_message
    if MQTT_USER:
        client.username_pw_set(MQTT_USER, MQTT_PASS)
    client.will_set("Invernadero/actuadores/status", "OFFLINE", retain=True)

    client.connect(MQTT_HOST, MQTT_PORT, keepalive=30)
    client.loop_start()
    try:
        publish_state(client, "Invernadero/actuadores/status", "ONLINE")
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
```

```
pass
finally:
    for name, d in DEVICES.items():
        gpio_set(d["pin"], False)
        current[name] = "OFF"
        publish_state(client, d["state"], "OFF")
    publish_state(client, "Invernadero/actuadores/status", "OFFLINE")
    GPIO.cleanup()
    client.loop_stop()
    client.disconnect()

if __name__ == "__main__":
    main()
```

Código 7.23: Script de control de actuadores vía MQTT en Raspberry Pi (`actuadores_mqtt.py`).

El Código 7.23 implementa el servicio de control de actuadores en la Raspberry Pi. Primero se definen los parámetros de conexión y los pines de cada dispositivo, dejando todo inicializado en estado seguro. La función `gpio_set` activa o desactiva un pin, mientras que `norm` traduce distintos formatos de entrada (ON/OFF, 1/0, encender/apagar) a un valor estándar. `publish_state` envía el estado al broker, y `apply_state` aplica cambios solo cuando son necesarios. En `on_connect` se gestionan las suscripciones y el estado inicial, y en `on_message` se interpretan las órdenes recibidas. Por último, `main()` arranca el cliente MQTT, establece el *Last Will* (OFFLINE) y mantiene el bucle de ejecución hasta que el servicio se detiene, apagando los actuadores de forma segura.

7.5.4. Servicio de arranque en *systemd*

Para garantizar la ejecución automática del script `actuadores_mqtt.py` en cada arranque de la Raspberry Pi, se creó una unidad de servicio en *systemd*. La edición del archivo se realizó con el siguiente comando:

```
sudo nano /etc/systemd/system/actuadores.service
```

Código 7.24: Creación del servicio *systemd* para `actuadores_mqtt.py`

El contenido final del fichero `actuadores.service` es el siguiente:

```
[Unit]
Description=Control de actuadores invernadero (MQTT)
After=network-online.target mosquitto.service
Wants=network-online.target

[Service]
Type=simple
User=osuna
Group=osuna
WorkingDirectory=/home/osuna/rpi_relay_4
Environment=PYTHONUNBUFFERED=1
# espera un poco para evitar carreras al arrancar red/mosquitto
ExecStartPre=/bin/sleep 2
ExecStart=/usr/bin/python3 -u /home/osuna/rpi_relay_4/actuadores_mqtt.py
Restart=on-failure
RestartSec=3
KillMode=control-group
TimeoutStopSec=5

[Install]
WantedBy=multi-user.target
```

Código 7.25: Unidad de servicio *systemd* para el control de actuadores.

En este servicio se establece la descripción y dependencias para que el script se inicie tras la red y el broker `mosquitto`, se configura la ejecución bajo el usuario `osuna` en el directorio del archivo, y se define el comando de arranque del script junto con parámetros de reinicio automático en caso de fallo. Finalmente, la sección `Install` asegura que el servicio se cargue de manera persistente al inicio del sistema.

7.6. Códigos de interfaz en Home Assistant

En esta sección se recopilan los fragmentos de código `YAML` empleados para la construcción de la interfaz del invernadero en Home Assistant. Cada bloque define las tarjetas gráficas que se muestran en el panel principal, organizadas en subsecciones temáticas: control manual de actuadores, configuración del modo automático, visualización de sensores, cámara en tiempo real y pronóstico meteorológico.

El objetivo es documentar de forma clara y reproducible la estructura de la interfaz, de modo que cualquier usuario pueda replicar o modificar el panel a partir de los ejemplos incluidos.

7.6.1. Gráficas y datos actuales

Este bloque del panel reúne, por un lado, la evolución temporal de las variables medidas (histórico de 72 horas) y, por otro, una tarjeta compacta con los valores instantáneos de los sensores del invernadero.

```
type: grid
cards:
  - type: heading
    heading: Graficas Invernadero
    heading_style: title
    icon: mdi:air-conditioner
  - type: history-graph
    title: Grafica de Sensores
    entities:
      - entity: sensor.temperatura_invernadero
      - entity: sensor.humedad_invernadero
      - entity: sensor.luminosidad_invernadero
    hours_to_show: 72
    refresh_interval: 60
```

Código 7.26: Tarjeta de gráficas históricas de sensores en Home Assistant

```
type: grid
cards:
  - type: heading
    heading_style: title
    heading: Datos actuales
    icon: mdi:air-conditioner
  - type: vertical-stack
    cards:
      - type: entities
        title: Sensores
        entities:
          - sensor.temperatura_invernadero
          - sensor.humedad_invernadero
          - sensor.luminosidad_invernadero
```

Código 7.27: Tarjeta de datos actuales (lecturas instantáneas)

La primera tarjeta (`history-graph`) muestra temperatura, humedad y luminosidad con un horizonte de 72 h y actualización cada 60 s, pues es cuando se mandan

nuevas mediciones. La segunda tarjeta (**entities**) presenta en una lista compacta los valores actuales de esos mismos sensores para consulta rápida en el panel. La Figura 7.7 y la Figura 7.8 ilustran visualmente este bloque, tanto en la interfaz web como en la aplicación móvil de Home Assistant.

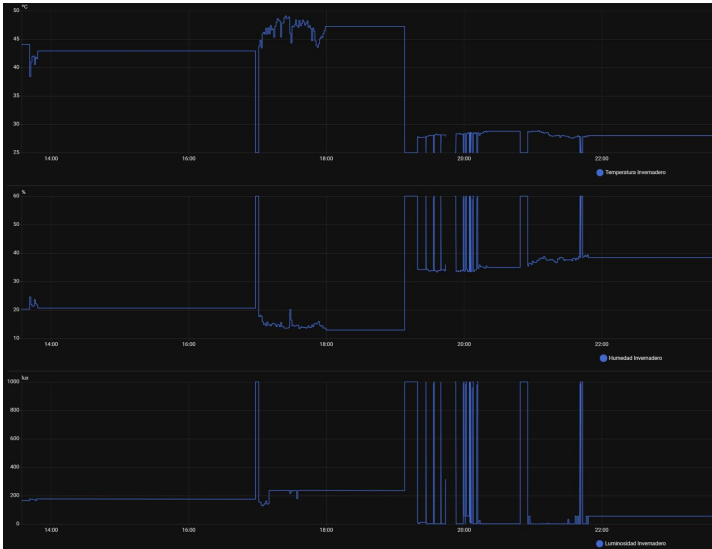


Figura 7.7: Visualización de las gráficas desde la interfaz web de Home Assistant.

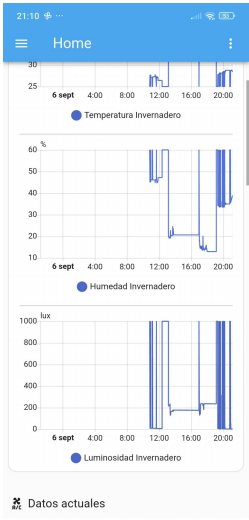


Figura 7.8: Visualización de las gráficas desde la aplicación móvil de Home Assistant.

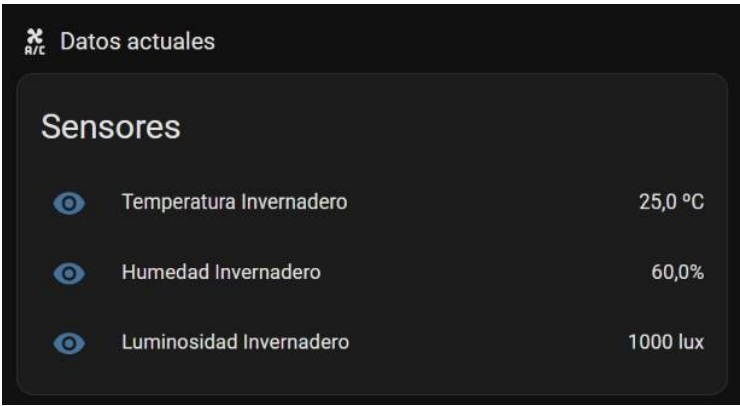


Figura 7.9: Visualización de los datos medidos en tiempo real.

7.6.2. Activación de actuadores

Este bloque del panel permite el control manual de los actuadores del invernadero. Se incluyen dos botones globales para encender o apagar todos los dispositivos simultáneamente, junto con botones individuales que muestran y controlan el estado del ventilador, la bomba y el foco, de manera individual.

```
type: grid
cards:
```

```
- type: heading
heading: ACTUADORES
heading_style: title
icon: mdi:air-conditioner
- type: vertical-stack
cards:
  - type: horizontal-stack
    cards:
      - type: button
        name: Encender todo
        icon: mdi:lightning-bolt-outline
        tap_action:
          action: call-service
          service: switch.turn_on
          target:
            entity_id:
              - switch.ventilador_invernadero
              - switch.bomba_invernadero
              - switch.foco_invernadero
      - type: button
        name: Apagar todo
        icon: mdi:power
        tap_action:
          action: call-service
          service: switch.turn_off
          target:
            entity_id:
              - switch.ventilador_invernadero
              - switch.bomba_invernadero
              - switch.foco_invernadero
    - type: button
      entity: switch.ventilador_invernadero
      name: Ventilador
      show_state: true
    - type: button
      entity: switch.bomba_invernadero
      name: Bomba
      show_state: true
    - type: button
      entity: switch.foco_invernadero
      name: Foco
      show_state: true
```

Código 7.28: Tarjeta de control global de actuadores en Home Assistant

La primera tarjeta define el encabezado de la sección y los botones para encender o apagar simultáneamente todos los actuadores mediante el servicio `switch.turn_on` o `switch.turn_off`. La segunda tarjeta incluye tres botones individuales que permiten gestionar de manera independiente cada dispositivo, mostrando además su estado actual en tiempo real. La Figura 7.10 ilustra este bloque en la aplicación.

7.6.3. Modo automático

Este bloque del panel habilita el funcionamiento automático del sistema, en el que los actuadores se controlan en función de umbrales configurables. Se incluye un botón principal para activar o desactivar el modo automático y, cuando este está en marcha, se muestran las tarjetas con los valores límite de temperatura, humedad y luminosidad.

```
type: grid
cards:
  - type: heading
    heading: MODO AUTOMATICO
    heading_style: title
    icon: mdi:refresh-auto
```

```
- type: vertical-stack
  cards:
    - type: button
      entity: switch.modo_automatgico
      name: Modo Automatico
      icon: mdi:refresh-auto
      show_state: true
      tap_action:
        action: toggle
      hold_action:
        action: more-info
    - type: conditional
      conditions:
        - entity: binary_sensor.modo_automatgico_activo
          state: "on"
      card:
        type: entities
        title: Umbrales de Activacion Automatica
        entities:
          - entity: number.umbral_temp_max
            name: Umbral Temp Max
          - entity: number.umbral_humedad_min
            name: Umbral Humedad Min
          - entity: number.umbral_luminosidad_min
            name: Umbral Luminosidad Min
```

Código 7.29: Tarjeta de configuración del modo automático en Home Assistant

La primera tarjeta define el encabezado y el botón de control del modo automático. La segunda tarjeta, visible únicamente cuando el sistema está activo, permite configurar los umbrales que gobiernan la activación de los actuadores en función de las condiciones ambientales. La Figura 7.11 muestra este bloque dentro de la aplicación.

7.6.4. Cámara del invernadero

Este bloque permite la visualización en directo del invernadero mediante una cámara IP integrada en el panel de Home Assistant. La tarjeta incluye un encabezado específico y una vista en vivo en formato panorámico, con la dirección IP configurada en la red local.

```
type: grid
cards:
  - type: heading
    heading: Camara Invernadero
    heading_style: title
    icon: mdi:webcam
    badges:
      - type: entity
        entity: camera.camara_invernadero
    tap_action:
      action: url
      url_path: ""
  - camera_view: live
    type: picture-glance
    title: Camara Invernadero
    entities: []
    camera_image: camera.camara_invernadero
    image: http://192.168.1.221:8081/
    image_entity: person.osuna
    aspect_ratio: "16:9"
    entity: camera.camara_invernadero
    grid_options:
      columns: full
column_span: 2
```

Código 7.30: Tarjeta de cámara en Home Assistant

La primera tarjeta define el encabezado y la entidad asociada a la cámara, mientras que la segunda muestra la transmisión en directo en formato **picture-glance**, con un aspecto 16:9. De esta manera, el usuario puede supervisar en tiempo real el estado del invernadero desde la interfaz web o la aplicación móvil. La Figura 7.12 ilustra este bloque en la aplicación.

Véase el Capítulo 8 para conocer más detalles sobre el sistema de videovigilancia.

7.6.5. Pronóstico meteorológico

Este bloque del panel integra la información meteorológica de la ubicación del invernadero. Se presentan dos tarjetas: la primera muestra la evolución horaria de la temperatura y el estado del tiempo para las próximas ocho horas, mientras que la segunda ofrece el pronóstico diario para los cinco días siguientes.

```
type: grid
cards:
  - type: vertical-stack
    cards:
      - show_current: true
        show_forecast: true
        type: weather-forecast
        entity: weather.forecast_casa
        name: Pronostico por horas
        forecast_type: hourly
        forecast_slots: 8
      - type: weather-forecast
        entity: weather.forecast_casa
        name: Proximos dias
        forecast_type: daily
        forecast_slots: 5
```

Código 7.31: Tarjeta de pronóstico meteorológico en Home Assistant

La primera tarjeta permite consultar el estado actual y el pronóstico por horas, útil para prever cambios inmediatos de temperatura, humedad o precipitaciones. La segunda tarjeta amplía la vista al medio plazo, mostrando de forma compacta la previsión de varios días. La Figura 7.13 muestra este bloque en la aplicación.



Figura 7.10: Visualización de los botones para el control de los actuadores.



Figura 7.11: Visualización del bloque de configuración del modo automático.



Figura 7.12: Cámara del invernadero en tiempo real.



Figura 7.13: Pronóstico meteorológico integrado.

Capítulo 8

Videovigilancia

8.1. Instalación y configuración de cámara USB en Raspberry Pi

Para la transmisión de vídeo en tiempo real desde una Raspberry Pi sin entorno gráfico, se ha utilizado una cámara USB junto con el software `motion`. El objetivo era visualizar el vídeo en directo a través de Home Assistant.

8.1.1. Instalación del software

Se utilizó el sistema operativo Raspberry Pi OS Lite, por lo que se procedió a instalar `motion` desde los repositorios oficiales:

```
sudo apt update
sudo apt install motion -y
```

Código 8.1: Instalación del software `motion` en Raspberry Pi OS Lite

8.1.2. Configuración de `motion`

El archivo de configuración principal se encuentra en `/etc/motion/motion.conf`. Se realizaron las siguientes modificaciones clave:

- Activar el modo demonio para que se ejecute en segundo plano:

```
daemon on
```

- Configurar la máxima resolución y frecuencia que permite la webcam:

```
width 1920
height 1080
framerate 30
```

- Configurar la máxima calidad visual disponible:

```
movie_quality 100
```

- Deshabilitar la grabación de vídeo para no llenar la microSD:

```
movie_output off
```

- Activar el servidor de streaming y permitir acceso remoto:

```
stream_port 8081
stream_localhost off
```

Añadir texto superpuesto en el vídeo:

```
text_left CAMERA INVERNADERO OSUNA
text_right %Y-%m-%d\n%T-%q
```

8.1.3. Ejecución del servicio

Para aplicar los cambios por primera vez, se debe ejecutar `motion` como superusuario utilizando el archivo `/etc/motion/motion.conf`:

```
sudo motion -c /etc/motion/motion.conf
```

Código 8.2: Ejecución inicial del servicio `motion`

Este comando debe usarse únicamente si `motion` no se está ejecutando ya. En caso de que el proceso esté activo, primero debe detenerse con:

```
sudo pkill motion
sudo motion -c /etc/motion/motion.conf
```

Código 8.3: Reinicio del servicio `motion` tras detener procesos activos

8.1.4. Acceso al streaming desde la red local

La webcam utilizada es el modelo **HP 320 FHD**, conectada por USB a la Raspberry Pi. Este dispositivo ofrece resolución Full HD (1920x1080) y una frecuencia de refresco de 30 fps, lo que permite una retransmisión fluida del vídeo, como se muestra en la Figura 8.1.

El software `motion` se encarga de capturar y servir el flujo de vídeo en directo. Por defecto, este servicio utiliza el puerto 8081 para transmitir el stream en formato MJPEG. Esta configuración se define en el archivo de configuración:

```
stream_port 8081
```

Una vez iniciado el servicio, el vídeo puede visualizarse desde cualquier navegador conectado a la misma red local, sin necesidad de autenticación ni software adicional. Para ello, basta con introducir en el navegador la dirección IP de la Raspberry Pi seguida del puerto configurado:

```
http://192.168.1.220:8081/
```

La Figura 8.2 muestra un ejemplo de la retransmisión en directo del vídeo capturado por la cámara, accedido desde un navegador web dentro de la red local.

8.2. Integración de la cámara en Home Assistant

Una vez que la cámara USB HP 320 FHD ha sido configurada correctamente en la Raspberry Pi mediante el software `motion`, el siguiente paso consiste en integrarla visualmente en la interfaz de Home Assistant.



Figura 8.1: Webcam HP 320 FHD conectada a la Raspberry Pi para capturar vídeo en directo.

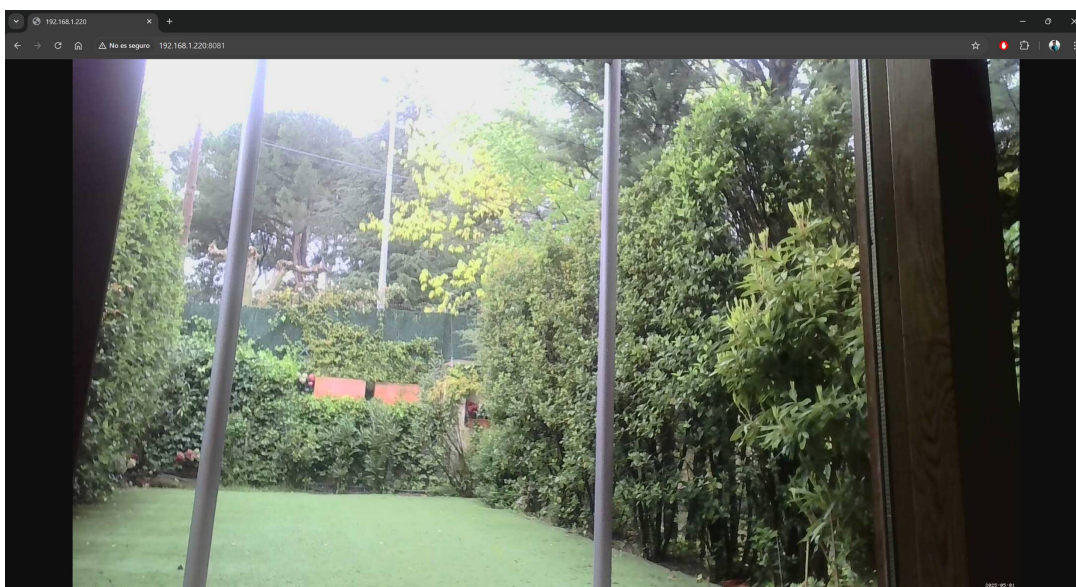


Figura 8.2: Visualización web de la cámara HP 320 FHD.

8.2.1. Añadir la cámara como dispositivo MJPEG

Dado que las versiones actuales de Home Assistant no permiten configurar cámaras genéricas vía YAML, se debe utilizar la interfaz gráfica para añadir la cámara.

Desde la vista principal de Home Assistant, se accede a **Configuración** → **Dispositivos y servicios** y se selecciona **Añadir integración**. A continuación, se busca y selecciona la opción **MJPEG IP Camera**. La Figura 8.3 muestra el formulario de configuración que debe completarse con la URL del streaming.

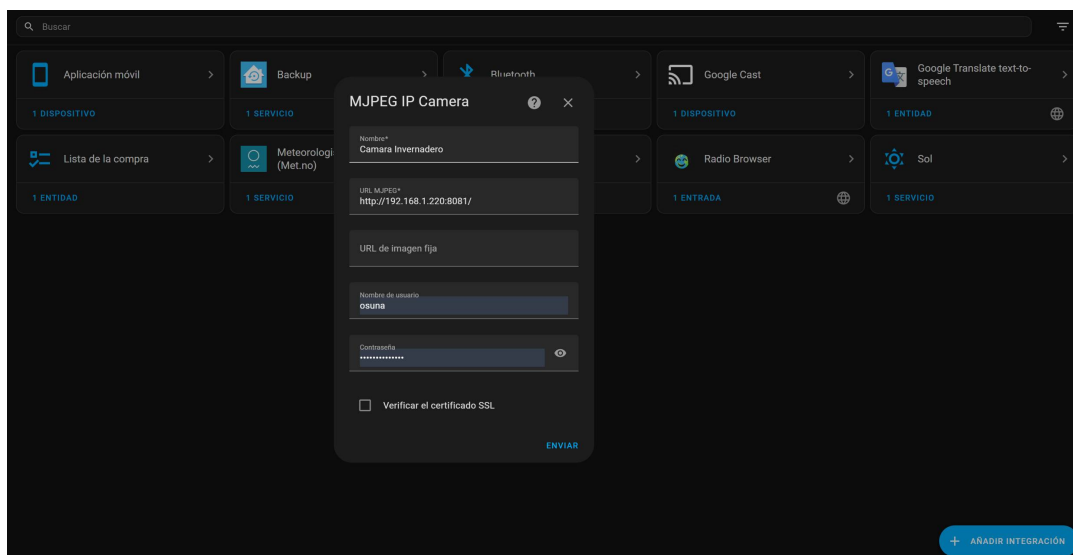
The image shows a screenshot of the Home Assistant web interface. A modal dialog titled 'MJPEG IP Camera' is open in the center. The dialog has a search bar at the top with the text 'Buscar'. Below the search bar, there are several input fields: 'Nombre*' with the value 'Camara Invernadero', 'URL MJPEG*' with the value 'http://192.168.1.220:8081/', 'URL de imagen fija', 'Nombre de usuario' with the value 'osuna', and 'Contraseña' with masked characters. There is a checkbox for 'Verificar el certificado SSL' which is unchecked. At the bottom right of the dialog is a blue button labeled 'ENVIAR'. In the background, the Home Assistant dashboard is visible, showing various integrations like 'Aplicación móvil', 'Backup', 'Bluetooth', 'Google Cast', 'Google Translate text-to-speech', 'Lista de la compra', 'Meteorología (Met.no)', 'Radio Browser', and 'Sol'. A blue button at the bottom right of the dashboard says '+ AÑADIR INTEGRACIÓN'.

Figura 8.3: Formulario de configuración de cámara MJPEG en Home Assistant.

En el formulario, se introducen los siguientes datos:

- **Nombre:** Cámara Invernadero
- **URL MJPEG:** http://192.168.1.220:8081
- **Autenticación:** Nombre de usuario y contraseña utilizados para entrar en Home Assistant
- **Certificado SSL:** Desactivado

8.2.2. Visualización en el panel principal

Una vez añadida la integración, se procede a mostrar la cámara en el panel principal de Home Assistant. Para ello, se edita la vista y se añade una nueva tarjeta del tipo **Vistazo de imagen**, donde se selecciona como fuente la entidad de cámara recién creada, como se aprecia en la Figura 8.4.

Finalmente, tras guardar los cambios, la cámara queda integrada en el tablero principal de Home Assistant. La imagen en directo se actualiza constantemente a través del stream MJPEG proporcionado por el servidor **motion**, como se muestra en la Figura 8.5.

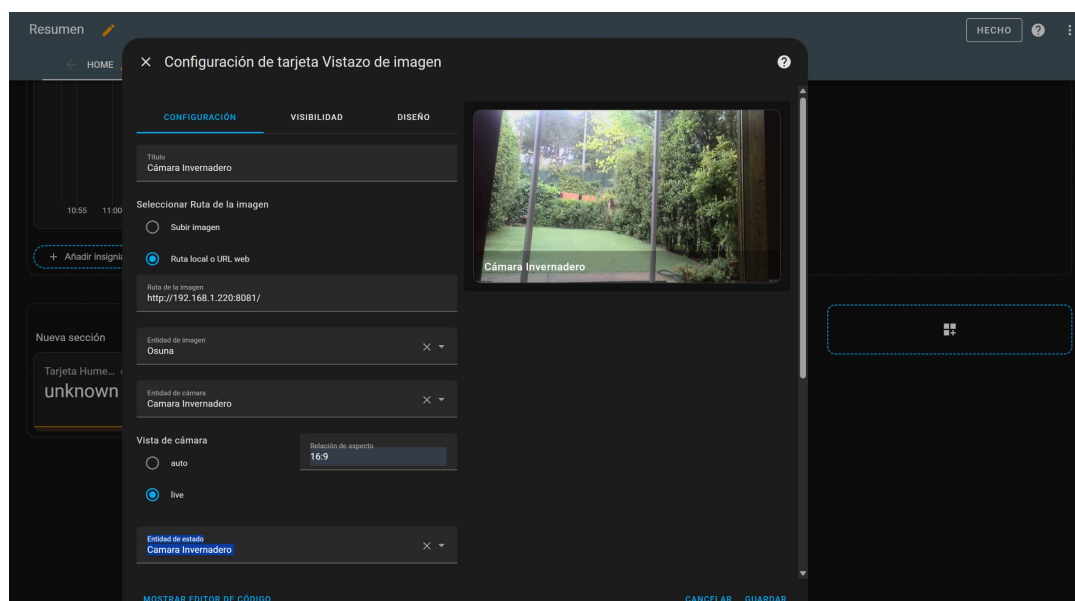


Figura 8.4: Configuración de tarjeta Lovelace con la cámara MJPEG.

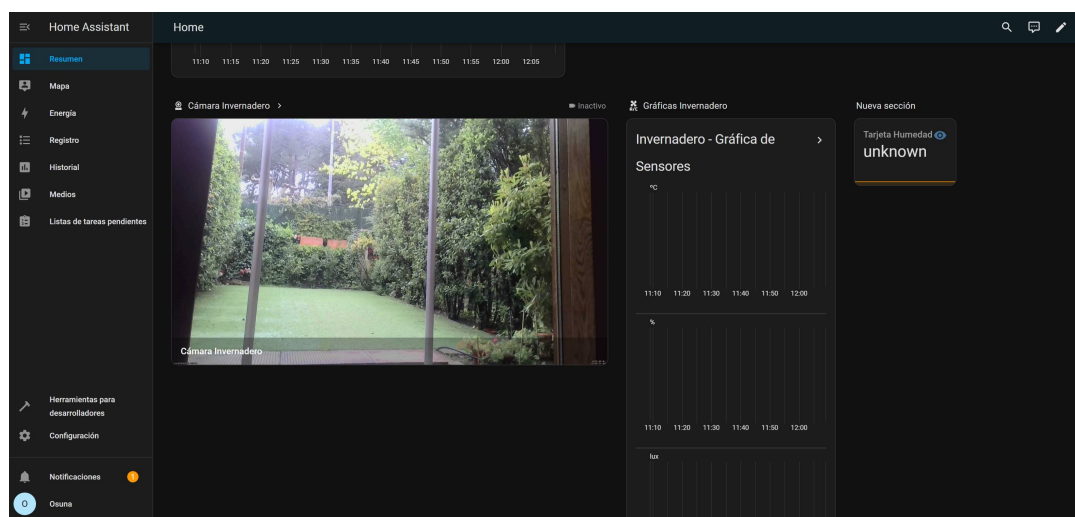


Figura 8.5: Vista final del panel con la cámara integrada.

8.3. Inicio automático de la retransmisión de la cámara al arrancar la Raspberry Pi

Para garantizar que el servicio de retransmisión de vídeo mediante webcam se active automáticamente al iniciar la Raspberry Pi, se ha configurado un servicio personalizado gestionado por `systemd`. Este enfoque asegura robustez, control del proceso y reinicio automático en caso de fallos.

8.3.1. Creación del servicio `motion`

Se ha creado un archivo de servicio en la ruta `/etc/systemd/system/motion.service`, el cual define cómo debe iniciarse y mantenerse activo el servicio de retransmisión de vídeo. En la Figura 8.6 se muestra el contenido del fichero, incluyendo parámetros como el tipo de ejecución, el usuario asignado y la política de reinicio automático ante errores.

```
[Unit]
Description=Motion Daemon
After=network.target

[Service]
ExecStart=/usr/bin/motion -c /etc/motion/motion.conf
ExecReload=/bin/kill -HUP $MAINPID
Restart=always
User=root
Group=root

[Install]
WantedBy=multi-user.target
```

Figura 8.6: Definición del servicio `motion.service`.

- `ExecStart` especifica el comando de inicio de `motion` con el archivo de configuración correspondiente.
- `Restart=always` garantiza que el servicio se reinicie si se detiene inesperadamente.
- `User=root` y `Group=root` se utilizan debido a que el usuario principal (`osuna`) no dispone de privilegios suficientes para acceder a la webcam y a recursos del sistema.

8.3.2. Habilitación del servicio en el arranque

Una vez definido el servicio, se ejecutan los siguientes comandos para habilitarlo y cargarlo en el sistema:

```
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable motion.service
sudo systemctl start motion.service
```

Código 8.4: Comandos para habilitar y arrancar el servicio `motion` en el sistema

La Figura 8.7 muestra cómo verificar el estado del servicio utilizando `systemctl status`, confirmando que el proceso se encuentra activo y operativo tras el arranque de la Raspberry Pi.

```
osuna@osuna:/etc/systemd/system $ sudo nano /etc/systemd/system/motion.service
osuna@osuna:/etc/systemd/system $ sudo systemctl daemon-reexec
osuna@osuna:/etc/systemd/system $ sudo systemctl daemon-reload
osuna@osuna:/etc/systemd/system $ sudo systemctl enable motion.service
osuna@osuna:/etc/systemd/system $ sudo systemctl start motion.service
osuna@osuna:/etc/systemd/system $ systemctl status motion.service
● motion.service - Motion Daemon
   Loaded: loaded (/etc/systemd/system/motion.service; enabled; preset: enabled)
   Active: active (running) since Fri 2025-05-02 11:04:00 CEST; 7ms ago
     Main PID: 1479 (motion)
       Tasks: 1 (limit: 1586)
        CPU: 0
      CGroup: /system.slice/motion.service
              └─1479 /usr/bin/motion -c /etc/motion/motion.conf

May 02 11:04:00 osuna systemd[1]: Started motion.service - Motion Daemon.
```

Figura 8.7: Servicio motion ejecutándose correctamente al arranque del sistema.

8.3.3. Modificación del archivo de configuración

Se ha identificado un conflicto al utilizar el modo `daemon on` por defecto en el archivo `/etc/motion/motion.conf`. Este modo permite que `motion` se convierta automáticamente en un proceso en segundo plano, lo que interfiere con la gestión del proceso por parte de `systemd`. Como consecuencia, el servicio se cerraba prematuramente y entraba en bucle de reinicios, provocando errores de tipo *start-limit-hit*.

Para evitar este comportamiento, se ha modificado la siguiente línea del archivo de configuración:

```
daemon on -> daemon off
```

Esto permite a `systemd` mantener el control del proceso principal de `motion`, evitando duplicaciones y bloqueos, y asegurando una inicialización estable del servicio al arranque del sistema.

8.3.4. Resultado

Con esta configuración, la retransmisión de la webcam queda operativa automáticamente cada vez que se arranca la Raspberry Pi, accesible desde el navegador a través del puerto 8081, cumpliendo con los requisitos de disponibilidad continua para el sistema de monitorización del invernadero.

Capítulo 9

Cableado del sistema

9.1. Alimentación del sistema

El sistema se alimenta a través de una regleta de cuatro tomas, en la que se conectan distintos cargadores y una fuente conmutada dedicada. La distribución de alimentación se resume en la Tabla 9.1.

9.2. Criterios de diseño

En este apartado se describen las decisiones tomadas respecto a la elección de fuentes de alimentación, la gestión de masas compartidas o aisladas y la forma de alimentar tanto la Raspberry Pi como los actuadores. Asimismo, se detalla el módulo de relés empleado para garantizar el aislamiento galvánico entre la electrónica de control y las cargas de potencia.

9.2.1. Cargadores Xiaomi

En la Figura 9.1 se muestran los dos cargadores Xiaomi empleados en el sistema. El **cargador Xiaomi grande** (modelo **MDY-11-EZ** [38]) se destina al módulo **ESP8266**, ya que este dispositivo presenta picos de consumo elevados (300–500 mA) durante las transmisiones Wi-Fi. Para garantizar un funcionamiento estable es necesario disponer de un amplio margen de corriente, y este adaptador proporciona hasta **3 A**, más que suficiente para cubrir las demandas del módulo.

De forma complementaria, se emplea un **cargador Xiaomi más compacto** (modelo **MDY-09-EW** [37]), con una capacidad de hasta **2 A**, para alimentar la placa **STM32F411E-Discovery**. Dado que el consumo típico de esta placa se sitúa en torno a 100–150 mA, la potencia suministrada por este adaptador resulta sobrada, asegurando un funcionamiento fiable con amplio margen de seguridad.

Enchufe	Dispositivo conectado	Notas
1	Cargador Xiaomi grande → ESP8266	Masa compartida
2	Cargador Xiaomi pequeño → STM32F411E-Discovery	
3	Cargador oficial Raspberry Pi → Raspberry Pi 4	Masa aislada 1
4	Fuente conmutada → Actuadores	Masa aislada 2

Tabla 9.1: Distribución de alimentación en la regleta de cuatro tomas.



Figura 9.1: Cargadores Xiaomi empleados en el sistema (arriba: modelo MDY-09-EW, abajo: modelo MDY-11-EZ).

9.2.2. Unificación de masas

Para garantizar la correcta comunicación UART entre el microcontrolador STM32 y el módulo ESP8266, se ha realizado la unión de sus **referencias de masa**. De este modo ambos dispositivos comparten la misma referencia eléctrica, evitando problemas de transmisión por diferencias de potencial.

9.2.3. Raspberry Pi y actuadores

La **Raspberry Pi 4** se alimenta mediante su cargador oficial [27], operando de forma autónoma. Esta gestiona los actuadores a través de un **script en Python** que publica y se suscribe a los tópicos MQTT. La Raspberry Pi no comparte masa con el STM32 ni con el ESP8266, por lo que se considera **aislada** del resto de la instrumentación.

Los **actuadores** (ventilador, bomba de agua y foco) se alimentan mediante una **f fuente conmutada independiente** [35], lo que permite manejar cargas de potencia sin comprometer la electrónica de control, manteniendo también el aislamiento eléctrico.

9.3. Relés de control

El control de los actuadores se realiza mediante un **Keyestudio RPI 4-Relay Shield (KS0212)** [20], conectado directamente a la Raspberry Pi. Este módulo integra relés SONGLE con optoacopladores, lo que asegura la separación galvánica

entre la etapa de control digital y los contactos de potencia. Gracias a este aislamiento, es posible conmutar cargas de hasta 250 VAC o 30 VDC sin que estas tensiones afecten a la electrónica de la Raspberry Pi.

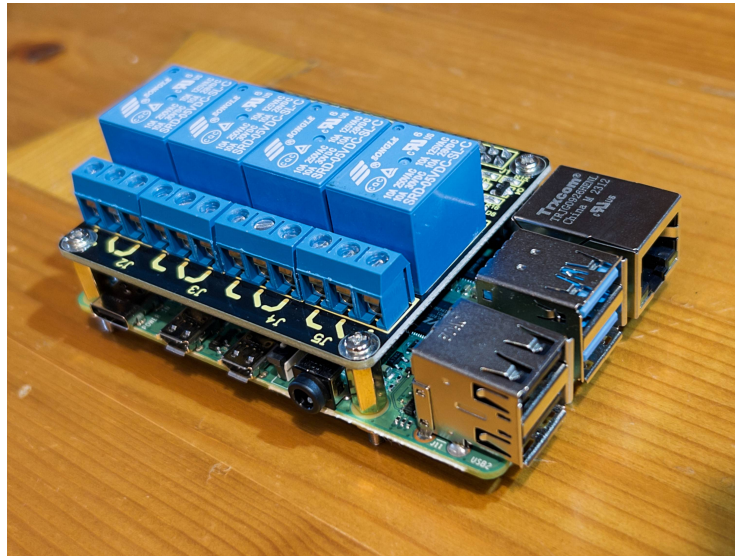


Figura 9.2: Módulo Keyestudio RPI 4-Relay Shield (KS0212) unido a Raspberry Pi 4B.

9.4. Montaje del sistema

El montaje físico del sistema se inició con la preparación de la estructura de madera que serviría como soporte para todos los componentes electrónicos. La base se reutilizó a partir de una balda de estantería doméstica, aportando robustez y sostenibilidad al proyecto. En esta fase inicial se practicaron los cortes en los tableros con ayuda de una sierra de calar (Figura 9.3), generando los orificios circulares necesarios para la entrada y salida de aire de los ventiladores. El proceso de realización de los orificios circulares destinados a la ventilación se muestra en la Figura 9.4 y la Figura 9.5.

Con la estructura ya preparada, se procedió al montaje de los paneles de madera con su base y a la instalación de los ventiladores en sus ubicaciones correspondientes. Además, se comenzó fijando la placa STM32 para ir avanzando en el montaje de forma ordenada desde la parte posterior hacia la frontal (Figura 9.6 y Figura 9.7).

La siguiente etapa consistió en la fijación de la estructura de soporte metálica que simula la bandeja del invernadero, así como la distribución de la Raspberry Pi con su módulo de relés y el módulo WiFi ESP8266 sobre la base de madera (Figura 9.8).

Posteriormente se instalaron los distintos elementos en el panel frontal, como la pantalla LCD, el potenciómetro, los sensores y la fuente conmutada. Una vez montada la estructura, se procedió a organizar el cableado en la parte posterior, identificándolo mediante etiquetas y agrupándolo dentro de un tubo corrugado para garantizar tanto la seguridad como el orden (Figura 9.9).

A continuación, se instaló la fuente de alimentación conmutada sobre la pared más alargada, fijándola firmemente mediante un par de tornillos. Asimismo, se realizó un orificio con taladro para permitir el paso ordenado del mazo de cables desde la parte trasera hacia el frontal del sistema, facilitando así el conexionado de todos los elementos electrónicos (Figura 9.10).



Figura 9.3: Proceso de corte inicial de la madera con sierra de calar.

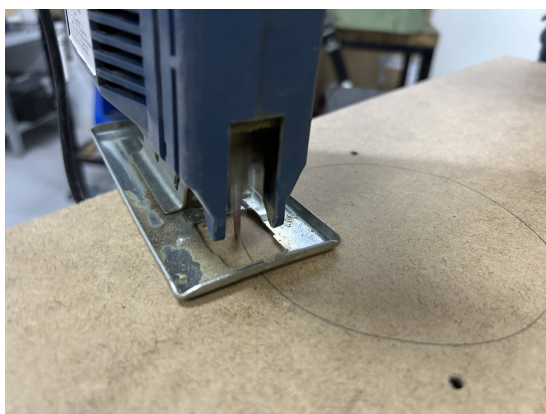


Figura 9.4: Proceso de corte circular en la madera con la sierra de calar.

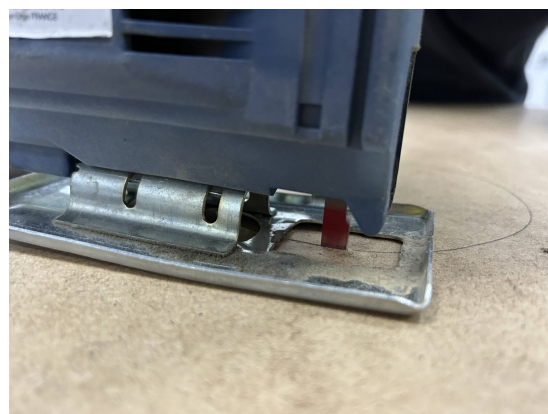


Figura 9.5: Detalle del corte circular en la madera con la sierra de calar.



Figura 9.6: Colocación inicial del ventilador en el panel lateral.

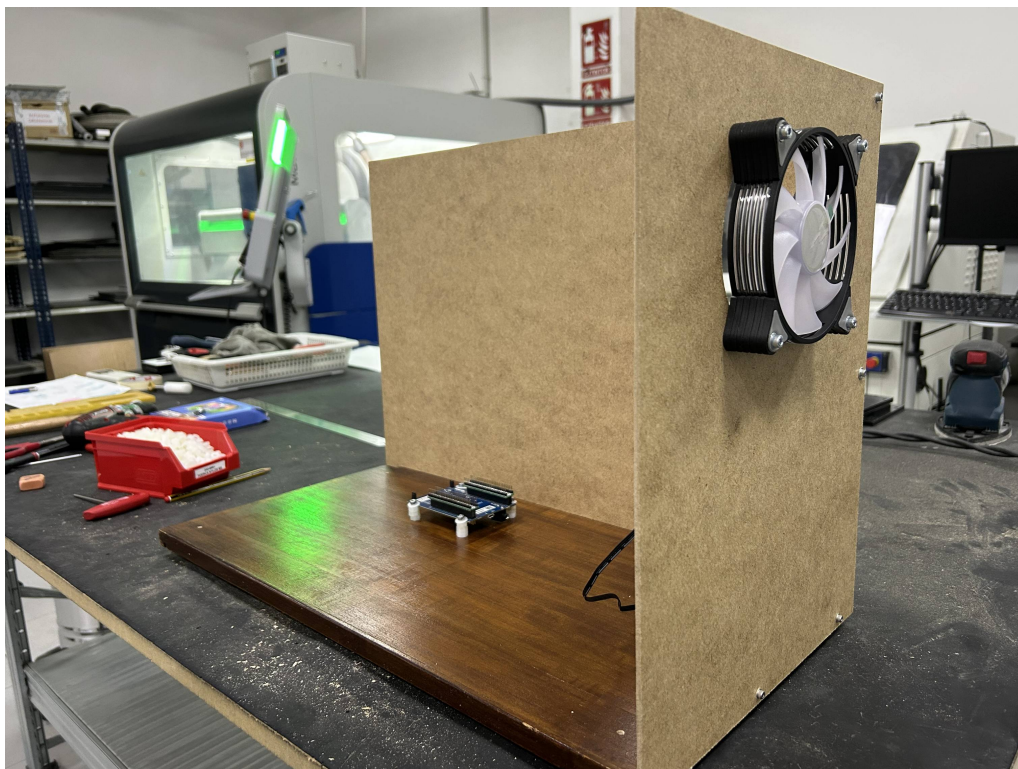


Figura 9.7: Vista lateral del panel con el ventilador ya fijado.



Figura 9.8: Estructura con ventiladores montados y bandeja metálica instalada.

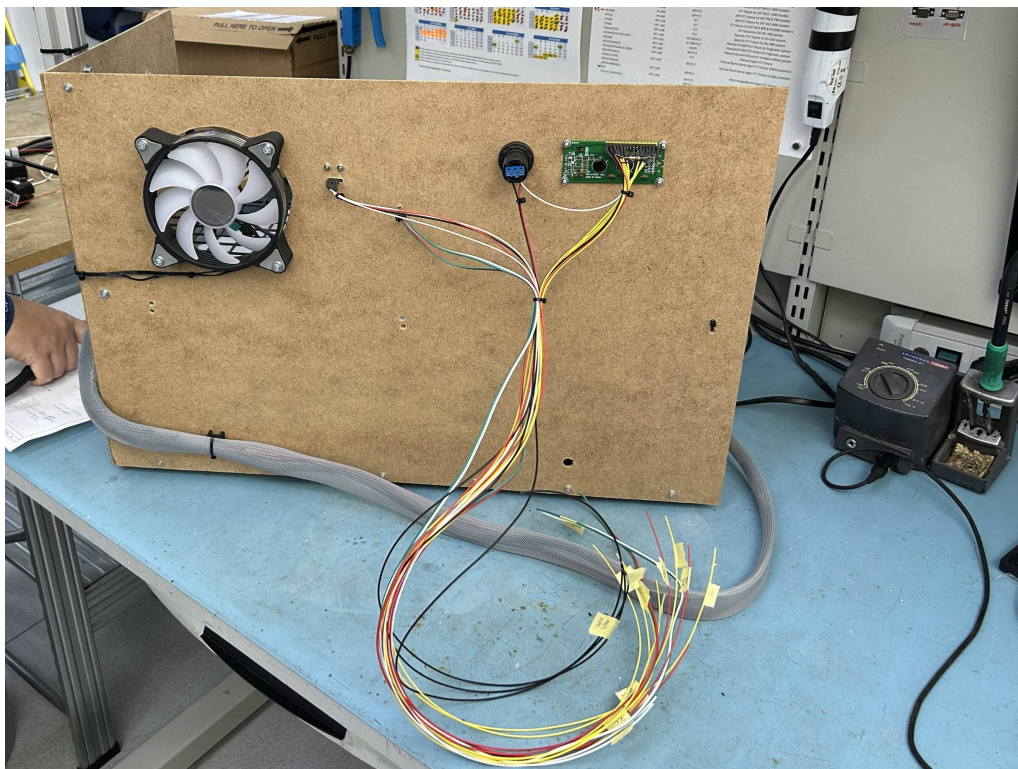


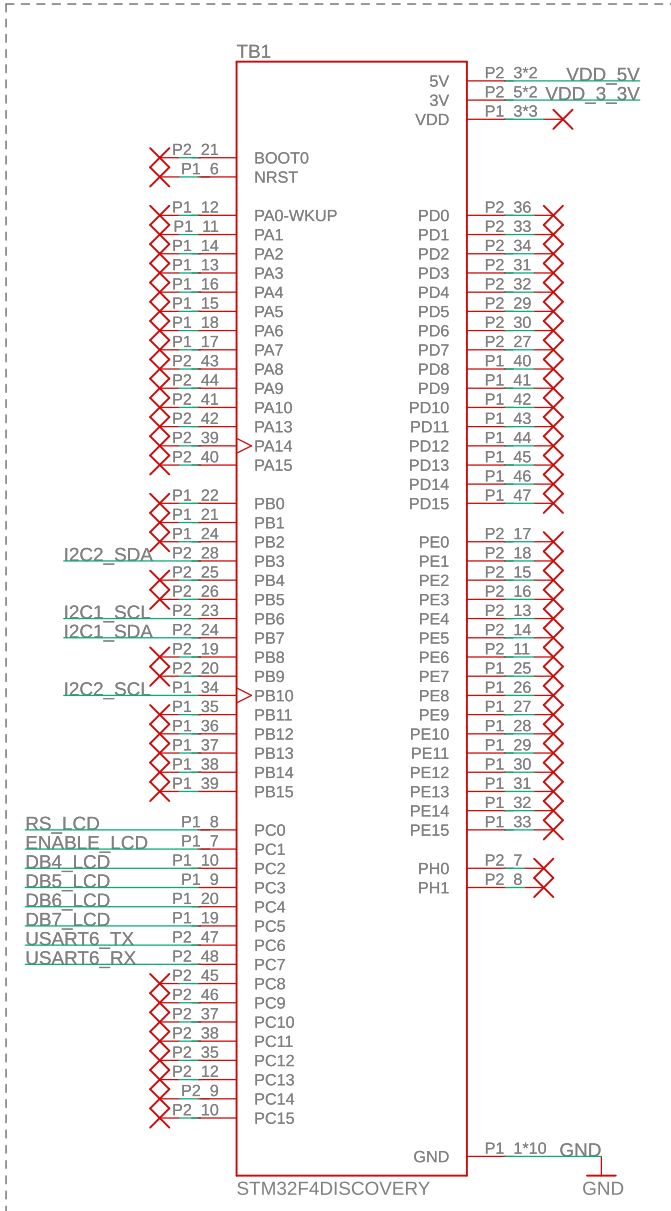
Figura 9.9: Vista trasera del sistema con el cableado organizado y protegido.

Finalmente, se realizaron las soldaduras y conexiones necesarias para unir todos los elementos: desde la fuente de alimentación hasta el STM32, el ESP8266 y la Raspberry Pi, asegurando la correcta comunicación entre sensores, actuadores y sistema de control (Figura 9.11). Para las conexiones en la placa STM32 se emplearon regletas de pines, lo que permite sustituir la placa de forma sencilla en caso de avería, evitando tener que soldar nuevamente los cables (Figura 9.12).

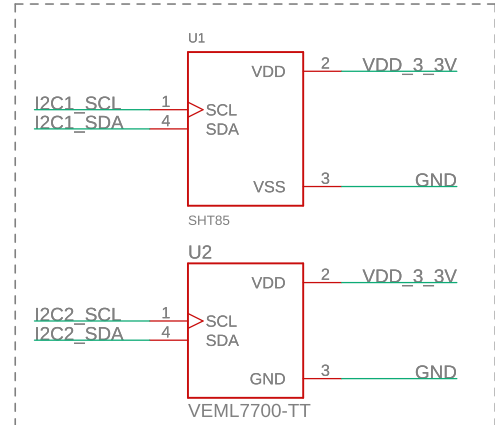
9.5. Planos eléctricos en Eagle

En esta sección se presentan los esquemas eléctricos diseñados en *Eagle*, junto con diagramas simplificados que muestran de forma clara la conexión de los sensores, actuadores y las líneas de alimentación del sistema.

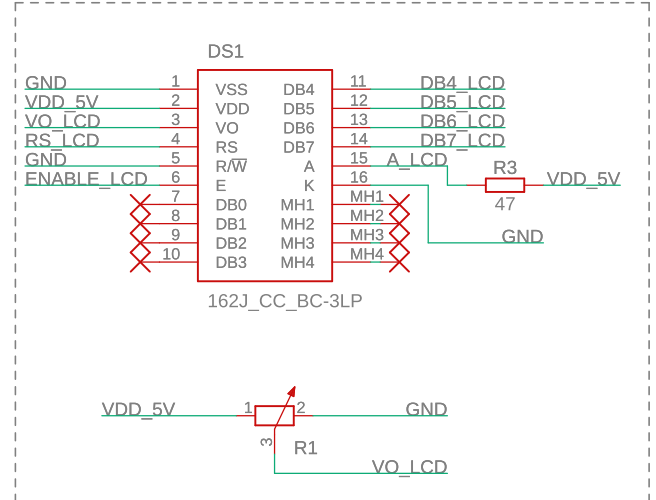
STM32F411



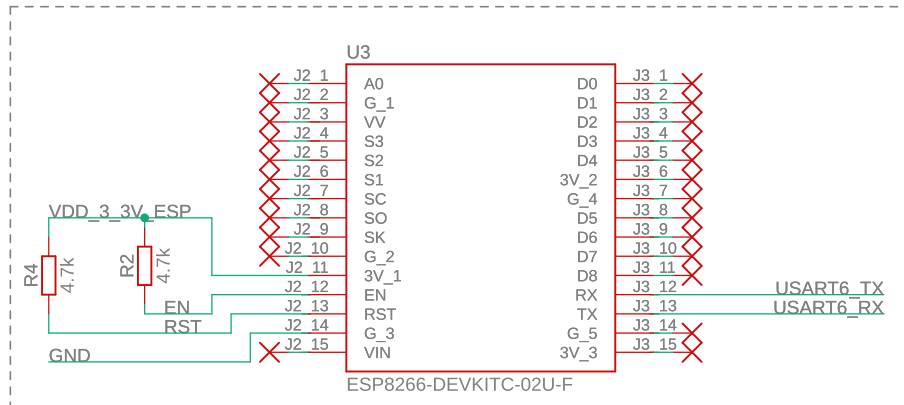
SENSORES



LCD_PANTALLA



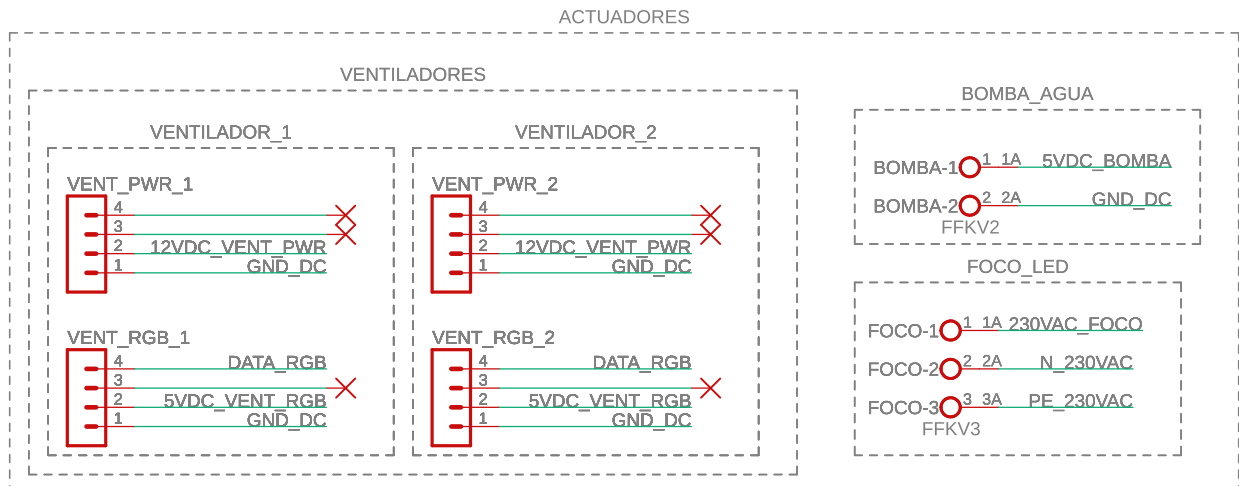
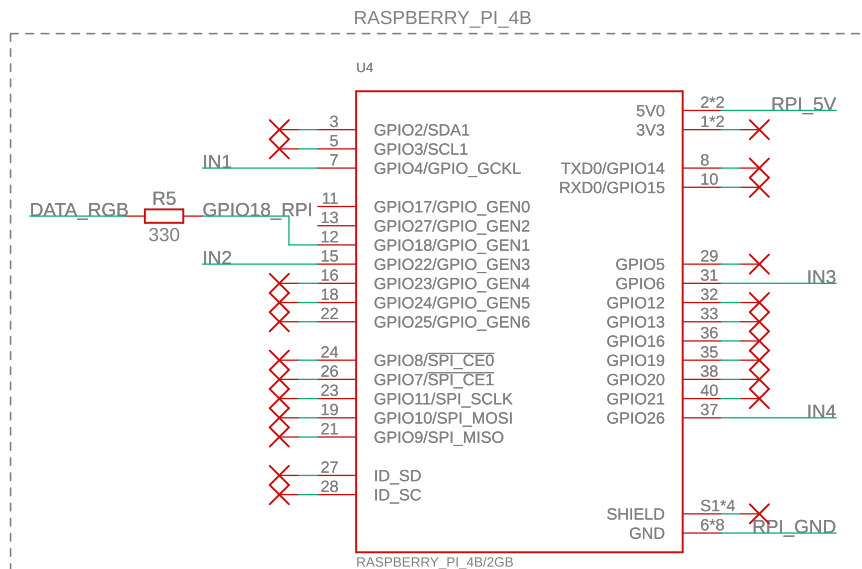
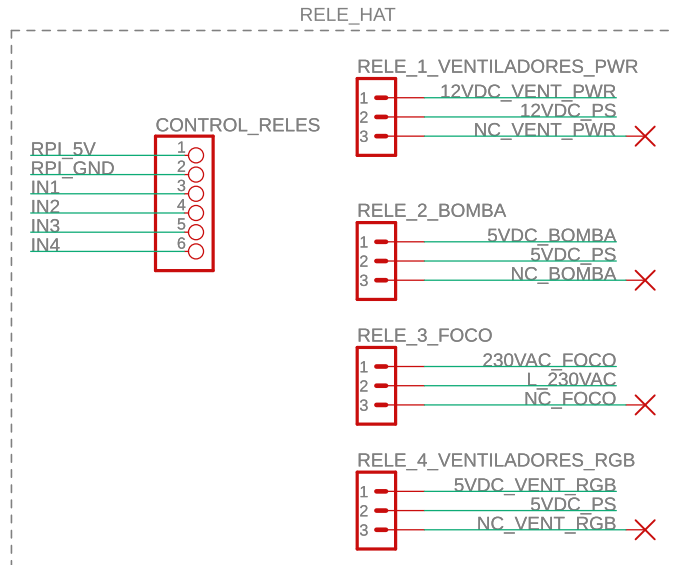
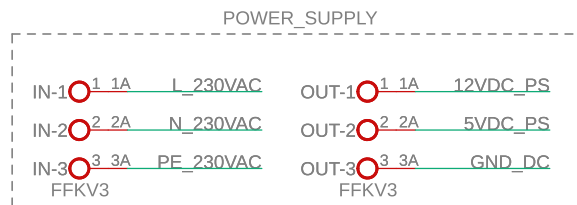
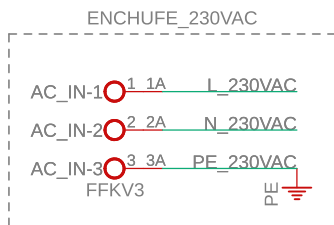
ESP8266



Date: 07/09/2025 11:08

TITLE: TFG_Invernadero

Sheet: 1/2



Date: 07/09/2025 11:08

TITLE: TFG_Invernadero

Sheet: 2/2

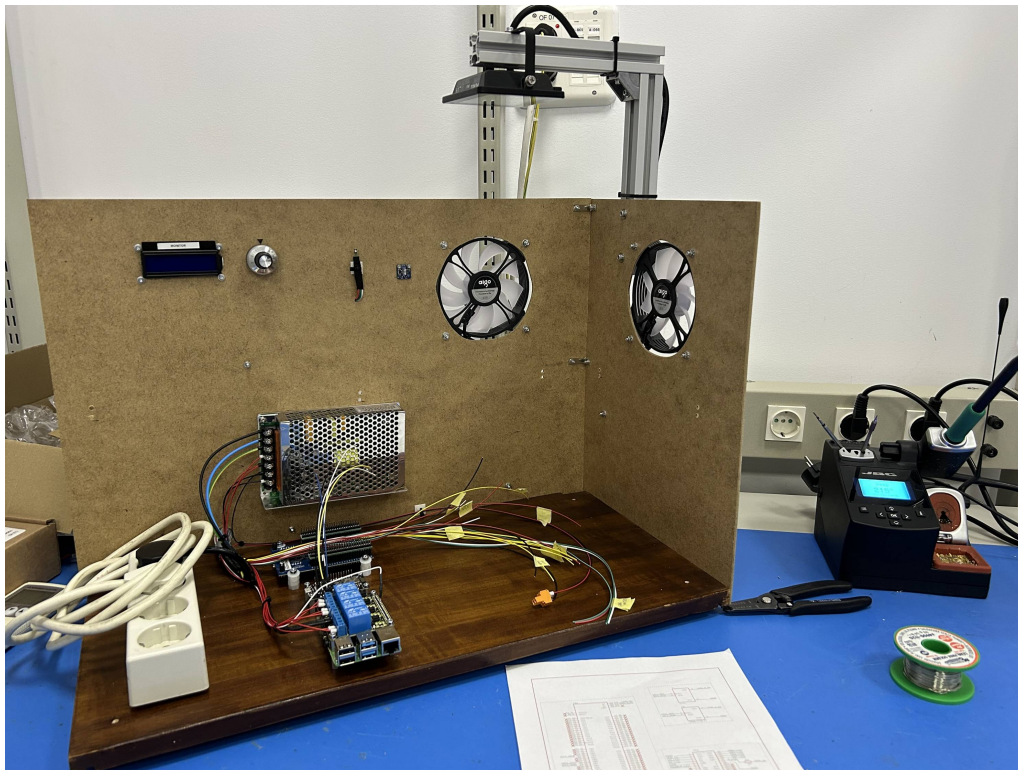


Figura 9.10: Panel frontal con la fuente de alimentación, pantalla LCD, potenciómetro, sensores y ventiladores ya instalados.



Figura 9.11: Proceso de soldadura de los cables del sistema.

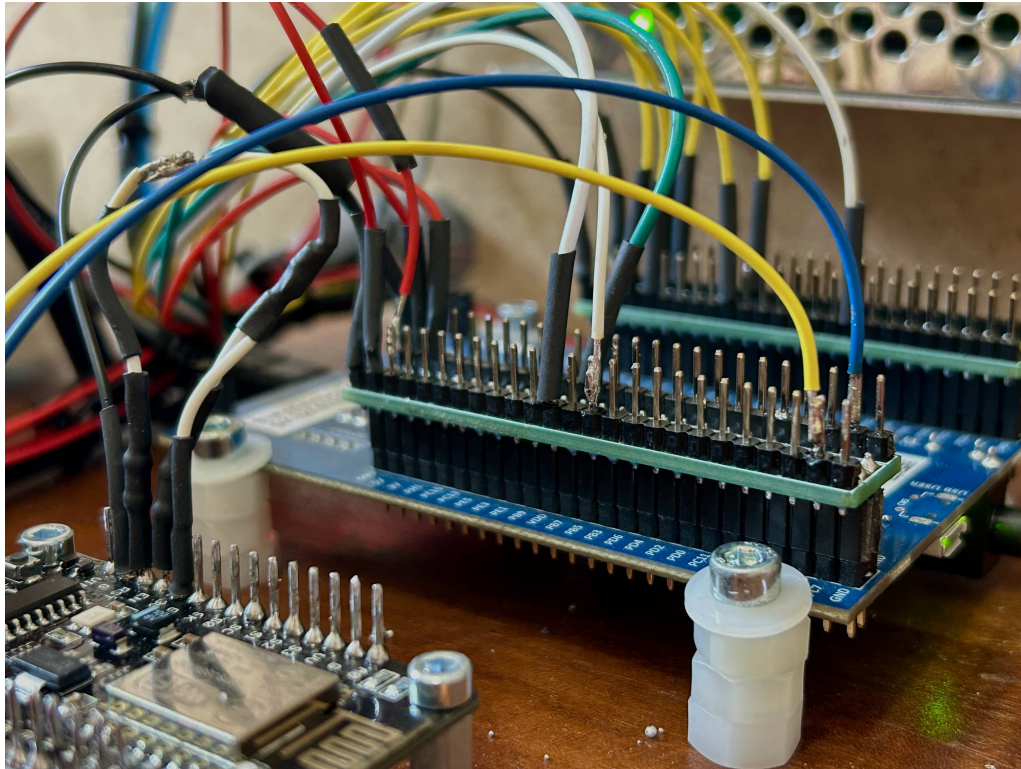


Figura 9.12: Resultado final del montaje con el cableado organizado en regletas.

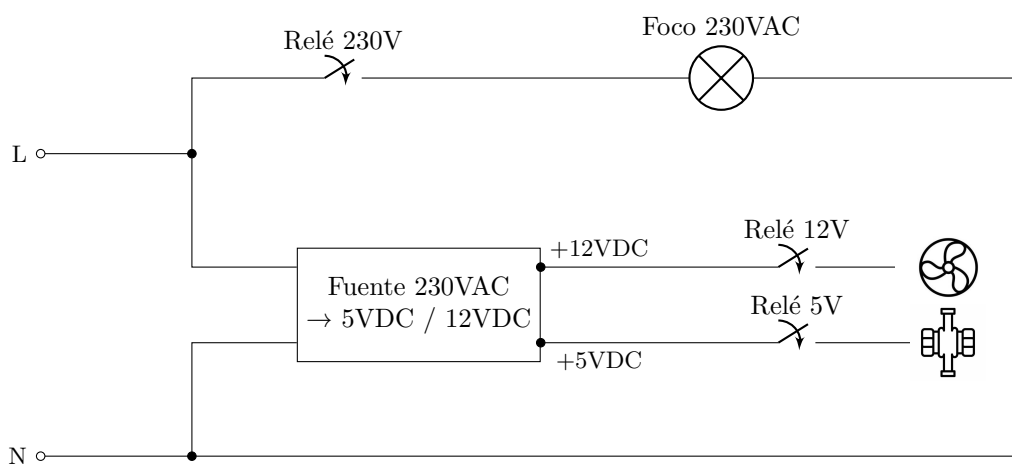


Figura 9.13: Diagrama de potencia con foco (230VAC), ventilador (12VDC) y bomba (5VDC).

Capítulo 10

Pruebas y resultados

En este capítulo se documenta el proceso de validación del sistema desarrollado, desde las pruebas iniciales con el microcontrolador STM32 hasta la integración final con la Raspberry Pi y la plataforma Home Assistant. Se detallan los problemas encontrados, las soluciones implementadas y los resultados obtenidos tras la puesta en marcha.

10.1. Pruebas

El trabajo comenzó exclusivamente con la placa STM32F411E-Discovery [33], donde se programaron los primeros drivers de los sensores SHT85 [30] y VEML7700 [34], así como el control del display LCD de 16×2 caracteres [10]. La comunicación con los sensores se realizó mediante buses I2C independientes, configurando I2C1 para el SHT85 y I2C2 para el VEML7700. El módulo ESP8266 [6] se conectó al microcontrolador mediante USART6, implementando un flujo de comandos AT para establecer la conexión WiFi y enviar datos a la nube a través de ThingSpeak.

Las primeras pruebas de envío fueron especialmente complicadas: en múltiples ocasiones el sistema quedaba bloqueado durante la publicación, sin enviar confirmación al STM32. En un principio no se sabía si la causa era el límite de frecuencia de actualización impuesto por ThingSpeak (que rechaza envíos muy seguidos), errores en el firmware del ESP8266 o problemas de ruido en la comunicación UART. Para depurar la situación se implementó una máquina de estados, de modo que en cada envío quedaba registrada la fase de la transmisión en el que se había quedado bloqueado.

A continuación, se llevaron a cabo pruebas específicas para cada bloque del sistema:

- **Pruebas de sensores:** se verificó la correcta adquisición de datos de temperatura, humedad y luminosidad mediante mediciones comparativas con instrumentos de referencia, comprobando la estabilidad de las lecturas en el tiempo.
- **Pruebas de comunicación I2C:** se evaluó la fiabilidad del bus frente a posibles errores de transmisión, introduciendo lecturas consecutivas a diferentes velocidades de muestreo.
- **Pruebas de comunicación UART con el ESP8266:** se ensayaron distintos comandos AT para garantizar que el módulo respondiera de forma coherente en todas las fases de la conexión WiFi y del envío a la nube.

- **Pruebas de temporización:** se realizaron varios experimentos ajustando los periodos de muestreo y publicación de datos para identificar el intervalo mínimo de envío sin provocar rechazos en ThingSpeak.
- **Pruebas de visualización en LCD:** se testearon los mensajes mostrados en el display de 16×2, verificando que la información de sensores y estados del sistema se presentaba correctamente.
- **Pruebas de integración con Raspberry Pi:** tras la inclusión de la Raspberry Pi 4, se ensayó la publicación en el broker MQTT y la correcta suscripción de Home Assistant a los *topics*.
- **Pruebas de actuadores:** se comprobó el accionamiento de los relés desde Home Assistant, verificando la respuesta inmediata de la bomba de agua, los ventiladores y el foco.
- **Pruebas de sistema operativo en la Raspberry Pi:** inicialmente se utilizó Raspberry Pi OS con entorno gráfico, lo que permitía trabajar con una interfaz completa. Sin embargo, el rendimiento no era el óptimo, especialmente al ejecutar simultáneamente Home Assistant y el broker MQTT. Tras varias pruebas se decidió migrar a la versión Raspberry Pi OS Lite (sin interfaz), lo que mejoró notablemente el rendimiento y permitió trabajar de forma más ágil mediante conexiones SSH desde un ordenador.

Posteriormente, tras consolidar la comunicación con la nube, se planteó junto al tutor la posibilidad de añadir una Raspberry Pi 4 [26] como servidor local. En ella se instaló Home Assistant junto con un broker MQTT, permitiendo centralizar tanto la supervisión como el control de actuadores en un panel web y en la aplicación móvil. Esta decisión transformó el proyecto en un sistema híbrido: por un lado el STM32 continúa sensando y enviando datos; por otro, la Raspberry Pi actúa como servidor de automatización, integrando los datos en gráficas y gestionando los relés de los actuadores.

10.2. Resultados

Los resultados muestran que el sistema es capaz de:

- Leer temperatura y humedad con el SHT85 (precisión ± 0.1 °C y ± 1.5 %RH).
- Medir la luminosidad con el VEML7700 y publicarla periódicamente.
- Mostrar las variables en el LCD de 16×2 caracteres, muy útil para que el usuario pueda visualizar las variables en tiempo real.
- Publicar datos en ThingSpeak y, en paralelo, enviar mensajes en formato JSON por MQTT hacia el broker de la Raspberry Pi.
- Visualizar en Home Assistant la evolución temporal de las variables y el estado de los actuadores, así como activar/desactivar manualmente bomba, foco y ventiladores a través de un módulo de 4 relés.

Los problemas iniciales con bloqueos de la ESP8266 se redujeron tras implementar los *timeouts*, un sistema de reconexión periódica y el registro de las respuestas recibidas por UART para detectar fallos como `DNS failure` o `WIFI DISCONNECT`.

10.3. Dificultades

Durante el desarrollo del proyecto se presentaron múltiples dificultades técnicas que requirieron un proceso de depuración detallado y prolongado. A continuación se describen los principales problemas encontrados y las soluciones implementadas en cada caso.

Error en ThingSpeak

En las primeras pruebas de envío de datos a ThingSpeak se detectó que, al intentar realizar publicaciones cada 15 o 20 segundos, los mensajes llegaban de forma intermitente: uno sí y otro no. Este fallo generó gran confusión y bloqueó el avance durante varias semanas, puesto que los ajustes iniciales parecían correctos. Tras un proceso de prueba y error, se descubrió que la causa estaba en la configuración del reloj del microcontrolador. Al aumentar la frecuencia del `SYSCLK` de 50 MHz a 80 MHz en el apartado *Clock Configuration* del proyecto, las publicaciones comenzaron a recibirse correctamente en ThingSpeak cada 20 segundos, resolviendo definitivamente el problema.

En la Figura 10.1 se muestra el problema detectado en ThingSpeak, donde se aprecian interrupciones en la curva de temperatura debido a la pérdida de publicaciones.

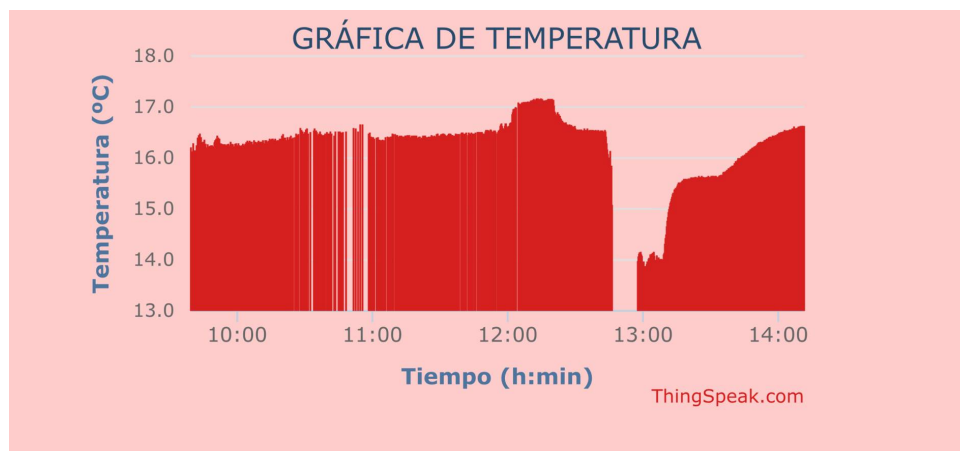


Figura 10.1: Gráfica de temperatura en ThingSpeak con pérdidas de datos.

Falta de respuesta del ESP8266

Otro de los principales inconvenientes surgió con la comunicación UART entre el STM32 y el módulo ESP8266. En muchas ocasiones no se recibía ninguna respuesta a los comandos enviados, lo que dificultaba enormemente la depuración y generaba gran pérdida de tiempo. La solución consistió en reorganizar el código del proyecto desde cero, implementando una estructura más limpia y clara. De esta forma, se pudo garantizar que los comandos AT enviados al ESP8266 devolvieran las respuestas esperadas, permitiendo continuar con el desarrollo.

Comunicación MQTT a bajo nivel

Finalmente, uno de los retos más complejos fue la construcción manual de los paquetes MQTT para enviarlos a la Raspberry Pi. Este enfoque requirió programar a muy bajo nivel, investigando en profundidad la estructura de los frames MQTT y comprobando cada byte transmitido. Aunque este proceso fue largo y propenso a errores, aportó un aprendizaje significativo sobre el protocolo y permitió alcanzar una implementación funcional. Con la experiencia adquirida, se considera que en un futuro proyecto sería más eficiente apoyarse en librerías de terceros ya verificadas, dedicando el esfuerzo principal a otros apartados del sistema. No obstante, haber completado con éxito una implementación tan cercana al hardware es un aspecto que refuerza la solidez técnica del trabajo desarrollado.

Selección de la instalación de Home Assistant

Otra de las dificultades encontradas estuvo relacionada con la elección del tipo de instalación más adecuada de Home Assistant en la Raspberry Pi. En un primer momento se valoró la opción de utilizar Home Assistant Core, lo que hubiera otorgado un elevado nivel de personalización y control. Sin embargo, esta alternativa requería programar en Python gran parte de las integraciones, aumentando considerablemente la complejidad y el tiempo de desarrollo.

Por otro lado, también se barajó la posibilidad de instalar Home Assistant OS, que ofrece una configuración mucho más sencilla y rápida. No obstante, esta opción presentaba una limitación importante: el sistema quedaba restringido a las integraciones disponibles en el propio Home Assistant, dificultando la instalación de servicios externos en la Raspberry Pi.

A esto se sumó que el rendimiento ofrecido por Raspberry Pi OS con interfaz gráfica no era el esperado. En ese momento surgieron dudas importantes sobre el camino a seguir: por un lado, migrar a Raspberry Pi OS Lite (sin interfaz), que aportaba un mejor rendimiento aunque a costa de perder la comodidad de la interfaz gráfica; y por otro, apostar por Home Assistant Core, una alternativa más flexible pero que exigía un mayor esfuerzo de programación y mantenimiento.

Finalmente, se optó por un enfoque intermedio: instalar Home Assistant en contenedores Docker sobre Raspberry Pi OS Lite (sin interfaz) para maximizar el rendimiento del sistema. Esta solución permitió, además, alcanzar un equilibrio entre flexibilidad y facilidad de mantenimiento, ya que posibilita la ejecución de Home Assistant junto con otros servicios en la misma Raspberry Pi, sin las restricciones de Home Assistant OS y sin la sobrecarga de tener que desarrollar todo en Python como con Home Assistant Core.

Montaje de la maqueta

Otra dificultad destacada fue la construcción física de la maqueta, un proceso que se extendió durante varias semanas debido a la cantidad de pasos intermedios implicados. En primer lugar, fue necesario diseñar la localización de cada componente dentro del sistema, planificando cuidadosamente la disposición de sensores, actuadores, placas electrónicas y la propia Raspberry Pi. Para ello resultó fundamental contar con los planos previamente realizados en Eagle, que sirvieron de guía para el cableado y las conexiones internas.

Posteriormente se procedió al ensamblaje de todos los elementos en paneles de madera, que actuaron como soporte estructural de la maqueta. Este trabajo requirió cortes y ajustes precisos, además de la fijación de cada módulo en su lugar correspondiente.

Uno de los aspectos más laboriosos fue el cableado: se tuvieron que soldar manualmente múltiples conexiones, asegurando tanto la continuidad eléctrica como un acabado limpio y robusto. Este proceso, aunque repetitivo y exigente para los que no están acostumbrados, permitió consolidar una maqueta estable y funcional, reflejando físicamente todo el diseño electrónico y de comunicaciones planteado en la fase inicial del proyecto.

Cambio de configuración de red

Aunque no constituye una dificultad técnica en el desarrollo, sí ha generado cierta inquietud de cara a la exposición final. Durante las primeras fases del proyecto se trabajó con una conexión por Ethernet, utilizando la dirección IP 192.168.1.220. Sin embargo, al avanzar hacia el montaje definitivo de la maqueta y buscar mayor flexibilidad para situar el invernadero en el jardín, fue necesario migrar a una red inalámbrica WiFi, empleando la dirección IP 192.168.1.221.

Este cambio implicó modificar numerosos archivos de configuración en el sistema, tanto en el STM32 como en los ficheros de integración de Home Assistant dentro de la Raspberry Pi. Aunque el procedimiento resultó exitoso, persiste la preocupación de que en el día de la defensa la red disponible asigne una IP distinta. En tal caso, sería necesario realizar nuevamente la adaptación de las direcciones IP en todos los puntos críticos del proyecto para garantizar que la maqueta funcione correctamente en la demostración en directo. No obstante, este inconveniente puede evitarse asignando directamente a la Raspberry Pi una IP fija, por ejemplo la 192.168.1.221. De esta manera, se asegura que la maqueta funcione siempre con la misma configuración independientemente de la red disponible.

10.4. Discusión

Los resultados evidencian que el mayor esfuerzo se concentró en la parte de comunicación entre el STM32 y el módulo ESP8266. El comportamiento errático del envío a ThingSpeak obligó a desarrollar un control robusto mediante máquina de estados y mecanismos de recuperación. La integración de la Raspberry Pi y Home Assistant supuso un valor añadido muy significativo: no solo permitió evitar la dependencia de la nube, sino que abrió la posibilidad de control local en tiempo real y la futura ampliación con bases de datos como InfluxDB y sistemas de visualización como Grafana.

En conclusión, el sistema probado cumple los objetivos planteados, demostrando que es posible sensor, publicar y controlar actuadores en un invernadero domótico mediante una arquitectura mixta basada en STM32 y Raspberry Pi.

10.5. Discusión

Los resultados obtenidos ponen de manifiesto que el mayor esfuerzo del proyecto se concentró en la comunicación entre el STM32 y el módulo ESP8266. El comporta-

miento irregular del envío de datos a ThingSpeak obligó a diseñar una máquina de estados más robusta y a implementar mecanismos de recuperación frente a bloqueos, garantizando así la continuidad del sistema en funcionamiento prolongado.

La incorporación de la Raspberry Pi y Home Assistant supuso un valor añadido decisivo: permitió reducir la dependencia de servicios en la nube y habilitó un control local en tiempo real, además de abrir la puerta a futuras ampliaciones con bases de datos como InfluxDB y sistemas de visualización avanzada como Grafana. Esta integración enriqueció notablemente la versatilidad y escalabilidad del sistema.

En conclusión, el prototipo desarrollado cumple satisfactoriamente con los objetivos planteados. Ha quedado demostrado que es posible sensar variables ambientales, transmitir datos y controlar actuadores en un invernadero domótico mediante una arquitectura híbrida basada en STM32 y Raspberry Pi.

Capítulo 11

Gestión del proyecto

En este capítulo se describe la organización del proyecto desde el punto de vista de la gestión: ciclo de vida, planificación temporal y análisis económico. El objetivo es mostrar cómo se estructuró el trabajo para alcanzar los objetivos definidos en los plazos establecidos y con los recursos disponibles.

11.1. Ciclo de vida

El desarrollo del sistema siguió un ciclo de vida clásico en ingeniería, dividido en las siguientes fases:

- **Definición:** Identificación de la necesidad (control local y remoto de un invernadero) y establecimiento de requisitos funcionales.
- **Análisis:** Estudio de posibles arquitecturas hardware/software, incluyendo alternativas de comunicación y plataformas de integración.
- **Diseño:** Selección de microcontrolador (STM32F411E-Discovery), módulo WiFi (ESP8266) y Raspberry Pi como nodo central con Home Assistant, así como definición de protocolos (UART, I2C, MQTT).
- **Implementación:** Desarrollo de drivers en C para sensores (SHT85, VEML7700), integración con el ESP8266 y configuración de Home Assistant en la Raspberry Pi.
- **Pruebas:** Validación de la comunicación WiFi, publicación MQTT y control de actuadores mediante relés. Se implementaron mecanismos de robustez (máquina de estados, timeouts).
- **Maqueta:** Construcción física del prototipo en estructura de madera, montaje de ventiladores, relés, sensores y cableado, asegurando la integración mecánica y eléctrica de todos los componentes.
- **Validación:** Verificación de que el sistema cumplía con los objetivos de monitorización y control definidos en la introducción del proyecto.
- **Documentación:** Redacción de la memoria, anexos técnicos y diagramas (véase Figura 11.1).

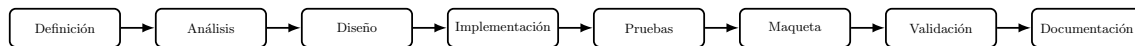


Figura 11.1: Diagrama de flujo del ciclo de vida del proyecto.

11.2. Planificación

La planificación del proyecto se organizó siguiendo una secuencia de fases, cada una dependiente de la finalización de la anterior. Esto permitió estructurar el trabajo de forma lineal y controlada, evitando solapamientos innecesarios. La única excepción fue la fase de documentación, que se mantuvo activa de forma transversal durante todo el desarrollo.

11.2.1. Planificación inicial

En un primer planteamiento, se establecieron hitos principales que incluían el diseño de la arquitectura, el desarrollo de drivers, la integración con el módulo WiFi, la instalación de Home Assistant y la validación global del sistema. La estimación temporal resultaba optimista y asumía que los problemas técnicos serían mínimos, lo que posteriormente no se cumplió.

11.2.2. Planificación final

La planificación final se ajustó a la realidad tras el desarrollo. El calendario quedó estructurado de la siguiente manera:

- **Definición** (1 semana, febrero): establecimiento de objetivos y requisitos iniciales.
- **Análisis** (2 semanas, febrero-marzo): estudio de alternativas hardware y software, selección de arquitecturas y protocolos.
- **Diseño** (1 semana, marzo): definición detallada de la arquitectura final y selección de componentes.
- **Implementación** (15 semanas, marzo-junio): desarrollo del firmware en C para STM32, integración de sensores (SHT85 y VEML7700), comunicación con el ESP8266 y configuración del entorno en Raspberry Pi con Home Assistant.
- **Pruebas** (5 semanas, julio): verificación de comunicación, publicación MQTT y control de actuadores mediante relés, con especial atención a los mecanismos de robustez frente a errores.
- **Maqueta** (3 semanas, agosto): construcción física de la estructura en madera, montaje de sensores, relés, ventiladores y cableado.
- **Validación** (2 semanas, agosto-septiembre): comprobación de que el sistema cumplía los objetivos de monitorización y control planteados inicialmente.
- **Documentación** (de febrero a septiembre): redacción progresiva de la memoria y anexos técnicos, manteniéndose en paralelo al resto de fases hasta la entrega final.

En resumen, el proyecto se extendió desde febrero hasta septiembre. La mayor parte del tiempo se concentró en la fase de implementación, que absorbió casi cuatro meses de trabajo continuo, seguida de pruebas y maqueta. La validación y la redacción final de la memoria cerraron el ciclo. La documentación, aunque distribuida en el tiempo, permitió mantener un registro constante de avances y cambios realizados.

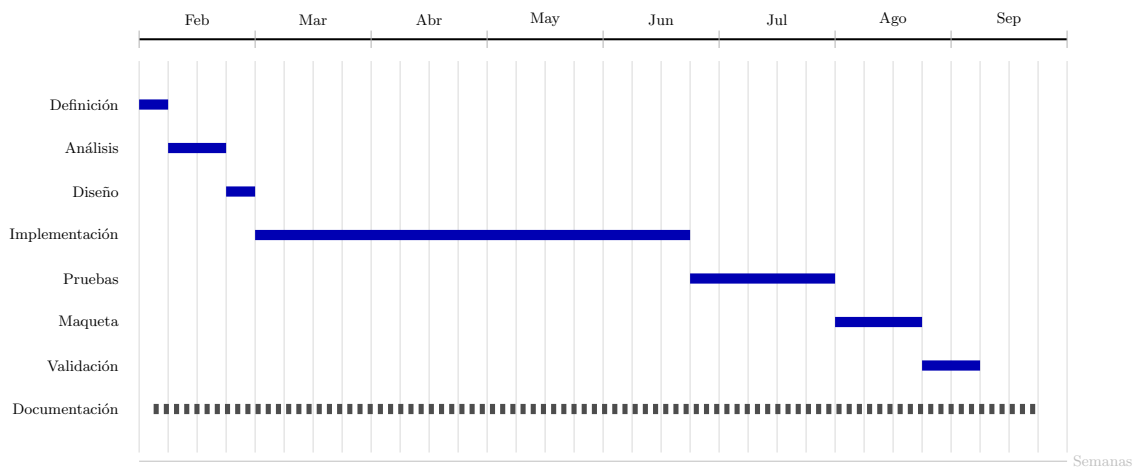


Figura 11.2: Diagrama de Gantt del proyecto (Febrero–Septiembre).

11.3. Presupuesto

El presupuesto incluye los costes de personal (dedicación estimada) y de material (componentes de hardware empleados). A continuación, se presenta el desglose.

11.3.1. Personal

Se estima una dedicación de unas 430 horas de trabajo para el desarrollo completo del proyecto. Considerando un coste académico de referencia de 15 €/h, el coste total de personal asciende a seis mil cuatrocientos cincuenta euros.

Ocupación	Horas	Precio (€/hora)	Importe (€)
Alumno / Ingeniero en formación	430	15	6.450
Total	430		6.450

Tabla 11.1: Coste de personal estimado para el proyecto.

11.3.2. Material

Los componentes principales empleados se recogen en la Tabla 11.2, incluyendo microcontrolador, sensores, pantalla LCD, Raspberry Pi, relés, entre otros.

11.3.3. Resumen de costes

Finalmente, en la Tabla 11.3, teniendo en cuenta los costes de material y de personal, se muestra el coste total del desarrollo del proyecto, que resulta en seis mil setecientos veintiséis euros con veintiún céntimos.

Componente	Precio (€)
STM32F411E-Discovery	15,11
Módulo WiFi ESP8266 NodeMCU	7,64
Sensor SHT85 (T/H)	32,42
Sensor VEML7700 (luminosidad)	1,31
Pantalla LCD 16x2 Displaytech	14,62
Potenciómetro (contraste LCD)	1,20
Raspberry Pi 4 (2 GB)	59,99
Tarjeta microSD SanDisk 32 GB	5,70
Carcasa oficial Raspberry Pi 4	9,01
Fuente oficial Raspberry Pi 4 (USB-C 5V 3A)	6,88
Fuente conmutada 60W (5V/12V)	25,79
Módulo de relés Keyestudio (4 canales)	10,99
Ventiladores DC 5V	10,00
Foco LED 10W	11,63
Bomba de agua 5V	11,99
Webcam HP 320 FHD	30,00
Cables Dupont (kit 120 uds., pruebas)	6,95
Bombas de agua	14,98
Total	276,21

Tabla 11.2: Presupuesto de hardware del proyecto.

Tipo de coste	Importe (€)
Material	276,21
Personal	6.450,00
Total	6.726,21

Tabla 11.3: Coste total del proyecto.

Capítulo 12

Conclusiones

Se presentan a continuación las conclusiones extraídas tras la realización del proyecto, así como las posibles líneas de desarrollo futuro.

12.1. Conclusión

Una vez finalizado el proyecto, se ha demostrado que es posible integrar un microcontrolador STM32 con sensores ambientales, un módulo WiFi ESP8266 y una Raspberry Pi actuando como servidor local para monitorizar y controlar un invernadero domótico. El sistema desarrollado cumple los objetivos planteados: medir en tiempo real variables de temperatura, humedad y luminosidad, transmitirlos mediante MQTT y ThingSpeak, y accionar actuadores como ventiladores, bomba de agua y foco LED en función de dichas condiciones.

Además, la arquitectura híbrida con Home Assistant permite tanto el control manual como la automatización local, ofreciendo una solución robusta y flexible sin depender exclusivamente de servicios en la nube.

12.2. Desarrollos futuros

Como desarrollos futuros, se plantean varias líneas de mejora e investigación:

- Integrar un sistema de inteligencia artificial que procese la previsión meteorológica de los próximos días, permitiendo ajustar de manera autónoma la cantidad de riego y la velocidad de ventilación en función de las condiciones climáticas esperadas.
- Realizar un análisis más exhaustivo de los datos registrados, por ejemplo mediante herramientas como MATLAB, que permitan filtrar y procesar la información para extraer conclusiones de mayor valor añadido.
- Implementar un sistema de notificaciones inteligentes que resuma al usuario el estado del invernadero y le avise únicamente en caso de anomalías o situaciones críticas, evitando que deba interpretar continuamente gráficas o registros históricos.
- Ampliar la integración con bases de datos como InfluxDB y plataformas de visualización como Grafana, para disponer de un sistema avanzado de análisis y monitorización a largo plazo.

Apéndice A

Vídeo demostrativo del proyecto

Se ha realizado un vídeo explicativo del funcionamiento del sistema, disponible en el siguiente enlace:

<https://youtu.be/Rs87myozX1s?si=rto6ceitn8ntuMzJ>

Tiene una duración de 7 minutos y 45 segundos, en los que se presenta de forma clara el resultado de la maqueta final. A lo largo del vídeo se explican también los protocolos y flujos de comunicación empleados en el proyecto, mostrando cómo se integran los distintos elementos del sistema. Finalmente, se ofrece una demostración visual de la aplicación Home Assistant en el teléfono móvil, desde la cual es posible supervisar las variables y controlar los actuadores en tiempo real.

Apéndice B

Repositorio de código en GitHub

El código en lenguaje C desarrollado en STM32CubeIDE durante todo el proyecto se encuentra disponible en el siguiente repositorio:

<https://github.com/0suuuunaa/TFG-Invernadero>

En dicho repositorio se incluyen los drivers de los sensores, las rutinas de comunicación mediante el módulo ESP8266, la implementación de la máquina de estados para el envío de datos por MQTT y a ThingSpeak, así como los ficheros de integración con la Raspberry Pi y Home Assistant.

Apéndice C

Anexo: Tabla de pines

C.1. Introducción

En este proyecto, se define la asignación de pines del microcontrolador STM32 para integrar diferentes periféricos y módulos. La configuración de pines se detalla a continuación en la Tabla C.1, destacando la funcionalidad de cada uno.

C.1.1. Asignación de pines y funciones

La Tabla C.1 resume la configuración de pines definida en STM32CubeIDE para este proyecto.

Periférico	Función	Pin STM32	Descripción
I2C1 – SHT85	I2C1_SCL	PB8	Señal de reloj
	I2C1_SDA	PB9	Línea de datos
I2C2 – VEML7700	I2C2_SCL	PB10	Señal de reloj
	I2C2_SDA	PB11	Línea de datos
USART6 – ESP8266	USART6_TX	PC6	Transmisión hacia ESP8266
	USART6_RX	PC7	Recepción desde ESP8266
LCD 16x2	D0	PC0	RS LCD
	D1	PC1	ENABLE LCD
	D2	PC2	Datos LCD
	D3	PC3	Datos LCD
	D4	PC4	Datos LCD
	D5	PC5	Datos LCD

Tabla C.1: Asignación de pines del STM32F411 para sensores, comunicación y periféricos.

C.1.2. Configuración de pines en STM32CubeIDE

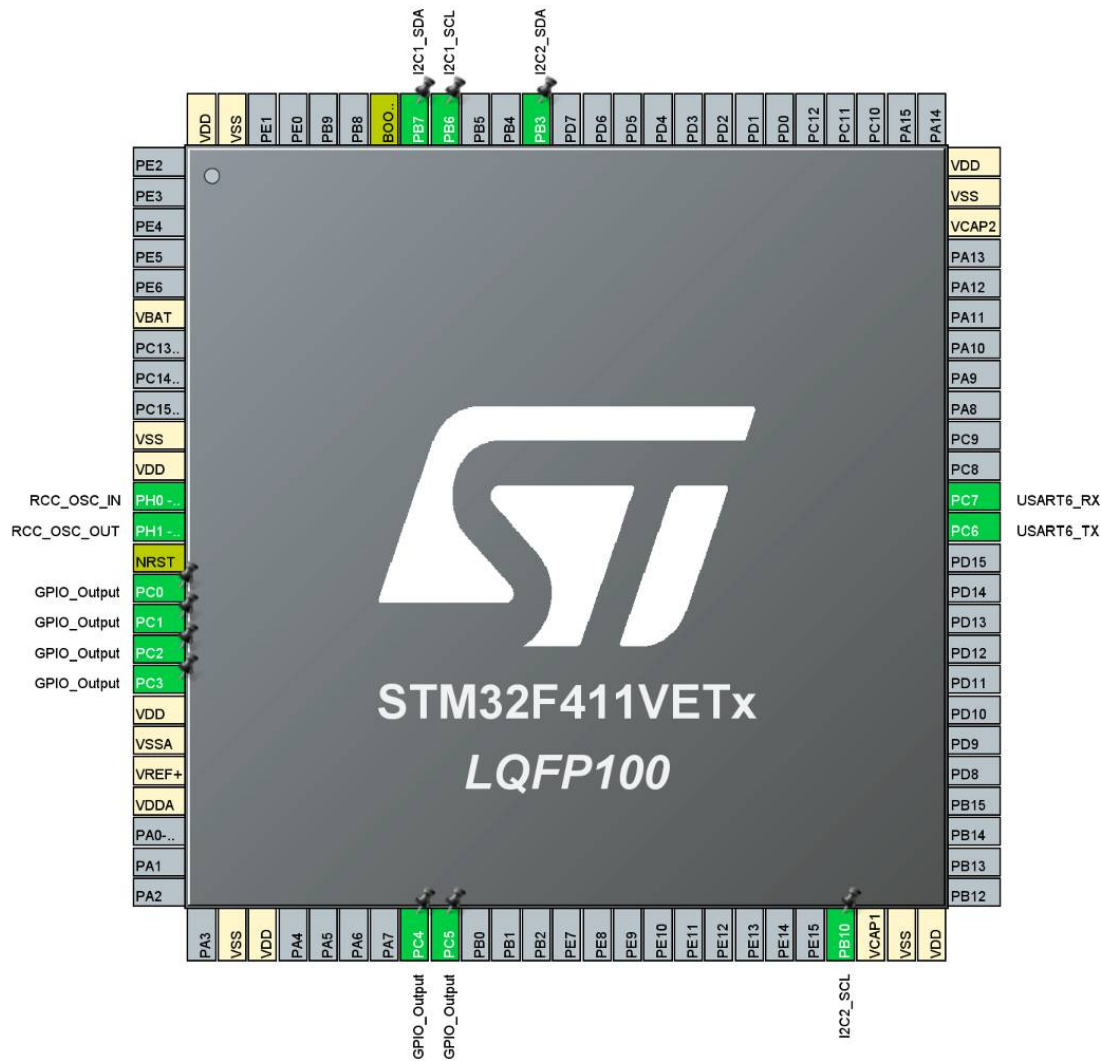


Figura C.1: Configuración de pines en STM32CubeIDE

Bibliografía

- [1] AliExpress. Aliexpress. potenciómetro de ajuste para lcd. url: <https://es.aliexpress.com/item/1005001519467295.html> (visitada el 17/02/2025).
- [2] AliExpress. Aliexpress. ventilador de refrigeración dc 5v para proyectos electrónicos. url: <https://es.aliexpress.com/item/1005005904586187.html> (visitada el 17/02/2025).
- [3] N. Arunkumar et al. Smart greenhouse using iot and cloud computing. *Computers and Electronics in Agriculture*, 181:45–60, 2021.
- [4] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.
- [5] Autodesk. Autodesk. eagle pcb design software. url: <https://www.autodesk.com/products/eagle> (visitada el 01/08/2025).
- [6] AZDelivery. Azdelivery. esp8266 wi-fi soc module nodemcu lolin v3. url: <https://www.amazon.es/dp/B06Y1ZPNMS> (visitada el 17/02/2025).
- [7] Andrew Banks and Rahul Gupta. Mqtt version 5.0, 2019. OASIS Standard.
- [8] Carsten Bormann, Angelo P Castellani, and Zach Shelby. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012.
- [9] Li Da Xu, Wu He, and Shancang Li. The internet of things: Cloud-based and sensor networks integration. *Future Generation Computer Systems*, 29(7):1645–1660, 2014.
- [10] Displaytech. Displaytech. 162j cc bc-3lp - 16x2 lcd display module. url: <https://www.digikey.es/es/products/detail/displaytech/162j-cc-bc-3lp/10494906> (visitada el 17/02/2025).
- [11] Eclipse Foundation. Eclipse foundation. eclipse mosquitto - mqtt broker. url: <https://mosquitto.org> (visitada el 01/08/2025).
- [12] Electrónica Embajadores. Electrónica embajadores. foco led 10w blanco neutro (leda6001nw-b). url: <https://www.electronicaembajadores.com/es/Productos/Detalle/IIF7030/iluminacion-y-senalizacion-general/focos-y-proyectores/foco-led-10-w-blanco-neutro-leda6001nw-b/> (visitada el 17/02/2025).
- [13] F. García. *Manual de domótica e inmótica*. Ra-Ma Editorial, Madrid, 2010.

- [14] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.
- [15] HeyNana. Heynana. kit de 120 cables dupont 3 en 1 (m/m, m/f, f/f) 28awg multicolor. url: <https://www.amazon.es/dp/B0965R9J21> (utilizados durante la fase de pruebas, no en la presentación final del proyecto. visitada el 27/02/2025).
- [16] Home Assistant Community. Home assistant community. home assistant. url: <https://www.home-assistant.io> (visitada el 01/08/2025).
- [17] Home Assistant Project. Home assistant installation on linux. <https://www.home-assistant.io/installation/linux>. visitada el 20/03/2025 2025.
- [18] HP Inc. Hp inc. hp 320 fhd webcam (53x26aa). url: <https://www.hp.com/es-es/shop/product.aspx?id=53X26AA&opt=ABB&sel=ACC> (visitada el 25/02/2025).
- [19] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE)*, pages 791–798. IEEE, 2008.
- [20] Keyestudio. Keyestudio. módulo de relés 4 canales para raspberry pi. url: <https://amzn.eu/d/bdYmKIj> (visitada el 25/08/2025).
- [21] X. Liu, J. Chen, F. Gao, and Y. Zhang. Internet of things for greenhouse environmental monitoring: A review. *Computers and Electronics in Agriculture*, 170:105–112, 2020.
- [22] MathWorks. Mathworks. thingspeak - matlab iot analytics. url: <https://thingspeak.com> (visitada el 01/08/2025).
- [23] Pixabay. Casa inteligente - casa, tecnología, 2018. Imagen con licencia libre de uso, obtenida de Pixabay.
- [24] Raspberry Pi Foundation. Raspberry pi foundation. carcasa oficial para raspberry pi 4 - negra/gris. url: <https://www.pccomponentes.com/raspberry-carcasa-oficial-para-raspberry-pi-4-negra-gris> (visitada el 12/03/2025).
- [25] Raspberry Pi Foundation. Raspberry pi foundation. fuente de alimentación usb-c para raspberry pi 4 (5v 3a 15w) - negra. url: <https://www.pccomponentes.com/raspberry-fuente-de-alimentacion-usb-c-para-raspberry-pi-4-5v-3a-15w-negra> (visitada el 01/08/2025).
- [26] Raspberry Pi Foundation. Raspberry pi foundation. raspberry pi 4 modelo b - 2gb. url: <https://www.pccomponentes.com/raspberry-pi-4-modelo-b-2gb> (visitada el 12/03/2025).
- [27] Raspberry Pi Foundation. Raspberry pi. fuente de alimentación oficial usb-c 5v 3a para raspberry pi 4 (15w). url: <https://www.pccomponentes.com/raspberry-fuente-de-alimentacion-usb-c-para-raspberry-pi-4-5v-3a-15w-negra?srsId=AfmB0orynTnVXNH8ndbmH60z0-XVeB2fxDfB9NusA2rGhfBTZUCDbme0> (visitada el 07/09/2025).

- [28] RUNCCI YUN. Runcci yun. bomba de agua sumergible sin escobillas 5v para acuarios y fuentes. url: <https://www.amazon.es/dp/B082PM8L6X> (visitada el 17/02/2025).
- [29] SanDisk. Sandisk. ultra microsdhc 32gb uhs-i clase 10 con adaptador sd. url: <https://www.pccomponentes.com/sandisk-ultra-microsdhc-32gb-uhs-i-clase-10-adaptador-sd> (visitada el 12/03/2025).
- [30] Sensirion AG. Sensirion ag. sht85 humidity and temperature sensor. url: <https://www.digikey.es/es/products/detail/sensirion-ag/SHT85/9666378> (visitada el 11/12/2024).
- [31] R. Singh et al. Sensor-based smart agriculture: A review of trends, applications and challenges. *IEEE Sensors Journal*, 18(20):8470–8487, 2018.
- [32] STMicroelectronics. Stmicroelectronics. stm32cubeide. url: <https://www.st.com/en/development-tools/stm32cubeide.html> (visitada el 01/08/2025).
- [33] STMicroelectronics. Stmicroelectronics. stm32f411e-disco discovery kit for stm32f411 line. url: <https://www.digikey.es/es/products/detail/stmicroelectronics/STM32F411E-DISCO/5131480> (visitada el 11/12/2024).
- [34] Vishay Semiconductors. Vishay semiconductors. veml7700 ambient light sensor. url: <https://www.digikey.es/es/products/detail/vishay-semiconductor-opto-division/VEML7700-TT/6210690> (visitada el 11/12/2024).
- [35] Walfront. Walfront. fuente de alimentación conmutada 60w, salida 5v/4a y 12v/3a. url: <https://www.amazon.es/dp/B09BDF686X> (visitada el 25/08/2025).
- [36] WinSCP Project. Winscp project. winscp - free sftp, scp and ftp client for windows. url: <https://winscp.net> (visitada el 01/08/2025).
- [37] Xiaomi Inc. Xiaomi. cargador usb modelo mdy-09-ew (2a). url: <https://rounded.com/xiaomi-travel-charger-2000mah-white-mdy-09-ew.html> (visitada el 07/09/2025).
- [38] Xiaomi Inc. Xiaomi. cargador usb modelo mdy-11-ez (33w). url: <https://mistore-eg.com/en/xiaomi-charger-mdy-11-ez-33w-white> (visitada el 07/09/2025).
- [39] Runhe Yu, Fei Zhang, and Huansheng Ning. A survey on security issues in services and communication of internet of things. *IEEE Internet of Things Journal*, 7(4):2641–2654, 2020.
- [40] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.