UNIVERSITAT DE BARCELONA

Trabajo final de grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

IMPLEMENTACIÓN DE UN SISTEMA DE VIDEOVIGILANCIA CON ESP32-CAM

Autor: Eric Ferreira Paredes

Director:	Dr. Manuel López de Miguel
Realizado en:	Departamento de Matemáticas e Informática

Barcelona, 10 de junio de 2025

Abstract

This final degree project presents the design and implementation of a low-cost video surveillance system based on ESP32-CAM microcontrollers. The system enables real-time video streaming, image capture, and motion detection, using the MQTT protocol for efficient communication and local data handling to ensure user privacy. It is possible to manage several cameras simultaneously, and the system can send automatic notifications via Telegram when motion is detected. A desktop interface facilitates device configuration and monitoring, while a simulator was developed to emulate multiple ESP32-CAM units for testing. The solution demonstrates the feasibility of creating scalable, privacy-focused surveillance networks without relying on commercial cloud services.

Resumen

Este Trabajo de Fin de Grado describe el desarrollo de un sistema de videovigilancia distribuido empleando microcontroladores ESP32-CAM. El sistema permite capturar y transmitir imágenes en tiempo real mediante el protocolo MQTT, detectando movimientos y notificando automáticamente a los usuarios a través de Telegram. Todo el procesamiento se realiza de manera local, sin depender de servicios en la nube, lo que garantiza la privacidad del usuario. Se ha implementado una aplicación de escritorio para gestionar y visualizar múltiples cámaras simultáneamente, y se ha desarrollado un simulador que permite validar el sistema sin necesidad de dispositivos físicos. El proyecto demuestra que es posible construir soluciones de vigilancia económicas, escalables y respetuosas con la privacidad.

Resum

En aquest projecte final de grau s'ha dissenyat i implementat un sistema de videovigilància assequible basat en microcontroladors ESP32-CAM. La plataforma permet capturar imatges, detectar moviment i transmetre vídeo en temps real, garantint la privacitat dels usuaris mitjançant el processament local de dades i la comunicació eficient a través del protocol MQTT. El sistema suporta la gestió simultània de diverses càmeres i envia alertes automàtiques per Telegram quan es registra activitat. Per facilitar-ne la configuració i el seguiment, s'ha desenvolupat una interfície d'escriptori. També s'ha creat un simulador que replica el funcionament de múltiples unitats ESP32-CAM per a proves. Aquesta solució demostra la viabilitat de construir xarxes de vigilància escalables i respectuoses amb la privacitat, sense necessitat d'utilitzar serveis externs comercials.

Agradecimientos

A mi tutor Manel López por su orientación y paciencia a lo largo de todo el trabajo. Su experiencia, disponibilidad y consejos han sido fundamentales para el desarrollo del proyecto.

A mi familia, por su apoyo y cariño incondicional con el que me han acompañado en cada paso. Gracias por estar siempre ahí, animándome incluso en los momentos más difíciles, y por ser un pilar fundamental en esta etapa tan significativa de mi vida.

Índice

Índice de figuras

Índice de tablas

Glosario

- **MQTT (Message Queuing Telemetry Transport):** Protocolo de mensajería ligero basado en el modelo *publish/subscribe*, utilizado para la comunicación entre dispositivos en redes IoT.
- **IP (Internet Protocol):** Dirección numérica que identifica de forma única a un dispositivo dentro de una red para permitir su comunicación.
- **Broker:** Servidor intermediario en una red MQTT que gestiona y distribuye los mensajes entre los dispositivos conectados.
- **Topic:** Canal lógico en MQTT utilizado para clasificar y direccionar los mensajes entre publicadores y suscriptores.
- **IoT (Internet of Things):** Red de dispositivos físicos conectados a Internet que pueden recopilar, enviar y recibir datos para interactuar con el entorno y otros dispositivos.
- **API (Application Programming Interface):** Conjunto de funciones y reglas que permiten la interacción entre diferentes programas o servicios.
- **Endpoint:** Dirección específica (URL) dentro de una API a la que se pueden enviar solicitudes para realizar una acción.
- **JSON (JavaScript Object Notation):** Formato ligero de intercambio de datos estructurados, ampliamente utilizado en comunicación entre aplicaciones.
- **Access Point (AP)**: Modo de operación en el que un dispositivo crea su propia red Wi-Fi para que otros equipos se conecten directamente a él, facilitando la comunicación y el intercambio de datos sin necesidad de infraestructura adicional.
- **Thread (Hilo):** Unidad de ejecución independiente dentro de un programa, que permite realizar múltiples tareas de forma concurrente.

1. Introducción

1.1. Contexto

En la actualidad, los sistemas de transmisión de video integrados en dispositivos de bajo consumo han mostrado un notable crecimiento en el mercado de las nuevas tecnologías, siendo particularmente notable su uso en el desarrollo de aplicaciones de Internet de las Cosas (IoT). Estos sistemas permiten capturar y enviar secuencias de video o imágenes a través de redes inalámbricas, abriendo la puerta a soluciones portátiles, autónomas y de bajo coste para vigilancia, monitoreo remoto y automatización inteligente.

Este tipo de dispositivos combina un microcontrolador relativamente potente, conectado con otros dispositivos con capacidad Wi-Fi o Bluetooth y periféricos como cámaras, sensores y/o actuadores, detectores de presencia o acelerómetros. En particular, el uso de cámaras permite capturar imágenes y transmitirlas en tiempo real mediante protocolos como HTTP o RTSP. La arquitectura de este tipo de dispositivos los hace ideales para entornos donde se requiere transmisión visual sin la intervención de infraestructuras complejas, como en aplicaciones de seguridad doméstica, agricultura inteligente, robots móviles, control de acceso o detección de intrusos.

La necesidad de este tipo de sistemas surge de la demanda creciente de dispositivos autónomos capaces de interpretar y compartir información visual en tiempo real, especialmente en zonas remotas o sin acceso a redes cableadas. El uso de microcontroladores con soporte multimedia permite mantener el consumo energético bajo, lo que es clave para operaciones prolongadas con baterías o paneles solares.

En la actualidad, la videovigilancia y la seguridad se a convertido en un aspecto fundamental tanto en el ámbito doméstico como en el empresarial. La creciente preocupación de proteger hogares, locales y empresas ha llevado a un incremento considerable de la venta de dispositivos de videovigilancia que permiten monitorizar y registrar eventos en tiempo real. Estos dispositivos no solo contribuyen a la prevención de robos, vandalismo o incidentes, sino que también facilitan el monitoreo remoto, ofreciendo más tranquilidad y control a los usuarios.

Otro punto interesante en el que los sistemas de videovigilancia tienen un importante crecimiento es el envejecimiento poblacional y el aumento de personas en situación de dependencia. Estos dos factores han generado una creciente necesidad de soluciones tecnológicas que garanticen su seguridad, autonomía y calidad de vida. En este contexto, los sistemas de videovigilancia basados en dispositivos IoT con cámara integrada se presentan como una alternativa accesible, flexible, económica y eficiente.

Estos dispositivos permiten la monitorización remota en tiempo real mediante la captura y transmisión de imágenes o video a través de redes Wi-Fi. Su bajo costo, reducido tamaño y capacidad de operación autónoma los hacen especialmente adecuados para su instalación en entornos domésticos. Pueden integrarse fácilmente en plataformas de hogares inteligentes y sistemas de alerta, permitiendo a familiares

o cuidadores supervisar el estado de una persona sin necesidad de intervención continua ni presencia física.

Más allá de la simple vigilancia, este tipo de sistemas puede incorporar algoritmos de visión artificial o detección de eventos anómalos, como caídas, ausencia de movimiento prolongada o patrones de comportamiento inusuales, que disparen alertas automáticas hacia cuidadores, servicios médicos o familiares a través de notificaciones en el móvil o correo electrónico. Además, la combinación con otros sensores IoT (como detectores de movimiento, temperatura, humedad o presencia) permite crear entornos inteligentes proactivos, capaces de adaptarse a las necesidades del usuario y responder ante posibles riesgos, como escapes de gas, puertas abiertas o falta de actividad prolongada.

Sin embargo, muchos productos comerciales presentan altos costos y así como la dependencia de servidores externos para almacenar o procesar las imágenes o videos. Esa dependencia plantea preocupaciones en términos de privacidad, además de limitar la autonomía del usuario, obligándolos a confiar en terceros para la gestión de sus datos. En muchos casos, las empresas proveedoras de estos servicios establecen planes de suscripción que pueden volverse caros con el tiempo, especialmente para los usuarios que necesitan almacenar grandes volúmenes de información o acceder a funciones avanzadas como el análisis de video con inteligencia artificial.

Para abordar esta problemática, se ha desarrollado un sistema de seguridad basado en ESP32-CAM, una solución de Internet de las Cosas (IoT) que permite capturar, procesar y transmitir imágenes sin necesidad de depender de plataformas en la nube y a un bajo costo.

1.2. Objetivos y motivación

El objetivo principal de este trabajo es el siguiente:

Diseñar un sistema de monitorización remota de viviendas, locales y negocios que permita la captura, transmisión y grabación de imágenes en tiempo real utilizando el microcontrolador ESP32-CAM. Además, integrar en el sistema la detección de movimiento, configuración personalizada y notificaciones automáticas. Todo ello garantizando un enfoque eficiente, seguro y respetuoso con la privacidad del usuario.

Este objetivo principal se desglosa en los siguientes objetivos específicos:

- **Desarrollar un sistema de transmisión en video en tiempo real utilizando el protocolo de comunicación MQTT** (*Message Queuing Telemetry Transport*)[22] y la red Wi-Fi, permitiendo la visualización remota de la cámara.
- **Implementar la grabación de video y captura de imágenes**, con opciones manuales como automáticas para facilitar el almacenamiento de eventos importantes.

- **Añadir un sistema de detección de movimiento**, que active automáticamente la grabación de imágenes cuando se detecte actividad en la escena.
- **Permitir la configuración personalizada de la cámara**, ofreciendo ajustes de resolución, brillo y otros parámetros clave.
- **Integrar un sistema de notificaciones en tiempo real utilizando un bot de Telegram**[1], para alertar al usuario cuando se detecte movimiento o se produzca un evento relevante.
- **Garantizar la seguridad y privacidad de los datos**, evitando los servidores en la nube y asegurando que toda la información se gestione de forma local.
- **Realizar pruebas y validaciones del software desarrollado**, evaluando su estabilidad, seguridad y eficiencia en diferentes condiciones de uso.

1.3. Estructura del documento

Este documento se estructura en siete capítulos principales:

- **Capítulo 1 – Introducción:** Se presenta el contexto del proyecto, la motivación y los objetivos específicos que guían el desarrollo del sistema.
- **Capítulo 2 – Estado del arte:** Se analizan las tecnologías actuales en el ámbito de la videovigilancia, las ventajas del ESP32-CAM y el uso del protocolo MQTT, así como una revisión de alternativas comerciales y arquitecturas IoT.
- **Capítulo 3 – Ingeniería de concepto:** Se define la arquitectura general del sistema, los módulos que lo componen y el flujo de comunicación entre los distintos elementos.
- **Capítulo 4 – Ingeniería de detalle:** Se profundiza en la implementación de cada uno de los componentes del sistema, incluyendo el software del ESP32-CAM, las aplicaciones de escritorio, el servicio de notificaciones y el simulador.
- **Capítulo 5 – Resultados:** Se analiza el cumplimiento de los objetivos planteados y se describen los resultados obtenidos durante la validación del sistema.
- **Capítulo 6 – Cronograma y costes:** Se presenta una estimación del tiempo invertido en cada fase del desarrollo y los recursos económicos necesarios.
- **Capítulo 7 – Conclusiones y trabajo futuro:** Se exponen las conclusiones alcanzadas y se plantean posibles líneas de mejora y ampliación del sistema.

2. Estado del arte

2.1. Sistemas de seguridad actuales

2.1.1. Evolución de la videovigilancia

En los últimos años, el uso de sistemas de videovigilancia ha experimentado un crecimiento notable en diversos entornos, desde hogares particulares hasta oficinas, industrias y establecimientos comerciales. Este aumento ha sido impulsado tanto por la creciente preocupación por la seguridad como por la disponibilidad de tecnologías más accesibles y avanzadas como el IoT (*Internet of Things*)[2]. Según un informe realizado por "Grand View Research"[3], se estima que el mercado global de la videovigilancia alcanzará los 147.660 millones de dólares en 2030, con una tasa de crecimiento anual compuesta (CAGR) del 12.1 % entre 2025 y 2030. Para entender cómo se ha llegado a este punto, es importante considerar la evolución que han tenido estos sistemas a lo largo del tiempo.

Las primeras configuraciones estaban basadas en cámaras analógicas, que son dispositivos que capturan y retransmiten video no digital. Estos tipos de cámaras están basadas en circuitos cerrados de televisión CCTV y utilizan cables coaxiales para transmitir el video utilizando señales eléctricas. Además, estas soluciones de seguridad estaban limitadas a la vigilancia pasiva, es decir, sistemas que grababan continuamente sin análisis inteligente, la cual cosa los videos solo eran revisados tras producirse un incidente. Esto no solo requería gran almacenamiento, sino también supervisión humana constante, lo que generaba costos elevados y baja eficiencia.

Con la transformación digital y el desarrollo de nuevas tecnologías, las cámaras analógicas han sido progresivamente reemplazadas por soluciones digitales más eficientes, como las cámaras IP. Estas cámaras hacen uso del protocolo IP (*Internet Protocol*)[4] y permiten el monitoreo en tiempo real desde cualquier dispositivo conectado a la red o a Internet. Asimismo, dan la posibilidad de utilizar análisis de video inteligente basado en inteligencia artificial para detectar y clasificar eventos o movimientos sospechosos.

2.1.2. Comparativa de sistemas analógicos y digitales

En la Tabla 1 se muestra una comparativa de las cámaras analógicas y digitales según un conjunto de características:

Características	**Cámaras IP**	**Cámaras Analógicas**
Resolución	Alta (hasta 4K y más)	Limitada (generalmente 480p o 720p)
Conexión	IP (Ethernet o wifi)	Cables (Coaxiales y DVRs)
Acceso remoto	Si (cualquier dispositivo)	No (acceso local generalmente)
Almacenamiento	En la nube o en servidores	Discos locales (DVRs)
Análisis inteligente	Si (Detección de movimiento, reconocimiento facial, etc.)	No (solo grabación)
Coste	Más alto inicialmente	Más económico inicialmente

Tabla 1: Comparación de características entre cámaras IP y cámaras analógicas

Como se puede observar, las cámaras IP ofrecen ventajas significativas en cuanto a calidad, conectividad y funcionalidades avanzadas. Estas características han impulsado su evolución y adopción en distintos ámbitos.

A continuación, se presentan algunas de las principales tendencias que están marcando el uso y desarrollo de las cámaras IP en la actualidad:

- Vigilancia inteligente basada en IA con reconocimiento facial, detección de movimiento avanzada y análisis de comportamiento.
- Sistemas distribuidos en la nube, donde las grabaciones y notificaciones se gestionan remotamente.
- Automatización del hogar (domótica) que integra la seguridad con otros elementos como cerraduras electrónicas, asistentes de voz, alarmas, etc.
- Sistemas inalámbricos, más flexibles y fáciles de instalar que los cableados.

Dado las ventajas significativas de las cámaras IP, este proyecto ha adoptado el uso de este tipo de dispositivos, ya que su capacidad para integrar inteligencia artificial y la automatización de procesos permite implementar un sistema de vigilancia más eficiente, flexible y accesible. Esta elección no solo responde a criterios tecnológicos, sino también a la necesidad de adaptarse a las tendencias actuales.

2.2. Tecnologías de videovigilancia basadas en IoT

2.2.1. Principales aplicaciones de IoT en videovigilancia

Una de las tecnologías que ha impulsado significativamente la evolución de los sistemas de videovigilancia es el "Internet de las Cosas" (IoT). El IoT añade y facilita la conectividad y automatización, haciendo que estos sistemas puedan reaccionar de forma inteligente y sin interacción humana directa ante distintos escenarios o situaciones.

Las aplicaciones del IoT en videovigilancia van más allá de la simple captura y transmisión de vídeo. Este enfoque permite crear ecosistemas inteligentes que mejoran la eficiencia, reducen el consumo energético y aumentan la capacidad de respuesta haciendo uso de sensores, microcontroladores y redes de comunicación. Algunas de las aplicaciones actuales más relevantes son:

- **Sincronización de cámaras con sensores PIR (*Passive Infrared*):** en muchos sistemas IoT, las cámaras IP se combinan con sensores de movimiento PIR. Cuando el sensor detecta una fuente de calor en movimiento (por ejemplo, una persona), activa automáticamente la grabación de la cámara y puede enviar una notificación al móvil del usuario. Esto reduce el almacenamiento innecesario y mejora la capacidad de respuesta ante intrusiones.
- **Control de accesos con autenticación multi-factor:** el IoT permite integrar cámaras IP con cerraduras inteligentes, lectores de tarjetas RFID (*Radio Frequency Identification*) y sensores biométricos. Por ejemplo, al detectar un rostro autorizado mediante reconocimiento facial, la cámara puede activar una cerradura electrónica conectada a la red IoT mediante protocolos como MQTT[22] o Zigbee[31]. Si el rostro no es reconocido, se puede activar una alerta y negar el acceso automáticamente
- **Integración con sensores ambientales en zonas sensibles:** en laboratorios o salas de servidores, los sensores de temperatura, humedad o calidad del aire pueden estar conectados al sistema de videovigilancia. Si las condiciones exceden los umbrales normales, se activa la grabación o se toma una captura de video para documentar lo que está ocurriendo en el entorno físico.

Estas aplicaciones demuestran cómo el IoT puede convertir los sistemas de videovigilancia en plataformas escalables y personalizables según las necesidades del entorno y del usuario.

2.2.2. Elementos clave para un sistema IoT de videovigilancia

La arquitectura de un sistema IoT de videovigilancia se basa en la integración de múltiples componentes tecnológicos que trabajan de manera conjunta para proporcionar soluciones de seguridad avanzadas. Estos sistemas combinan dispositivos de captura (como cámaras), sensores, redes de comunicación y plataformas de control

centralizado para crear un entorno interconectado que ofrece vigilancia en tiempo real, análisis inteligente y respuesta autónoma a eventos.

A continuación, en la Figura 1, se describe la estructura general de un sistema IoT de videovigilancia:

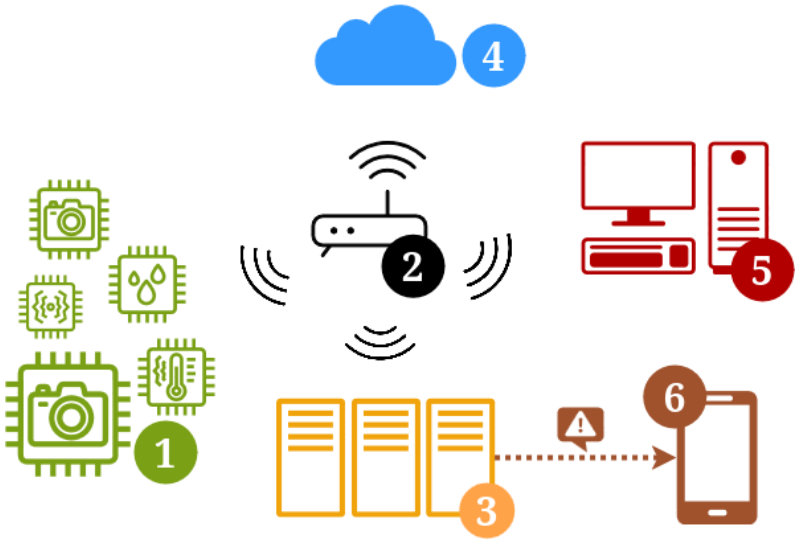

Figura 1: Arquitectura de un sistema de videovigilancia basado en IoT

1. **Dispositivos de captura:** El componente principal en cualquier sistema de videovigilancia es la cámara. En un sistema IoT, estas cámaras suelen ser cámaras IP, como el ESP32-CAM, que permiten la transmisión de video en tiempo real a través de redes inalámbricas o cableadas. En este grupo, también se incluyen los sensores de movimiento (sensores PIR) y otros tipos de sensores ambientales (de temperatura, humedad, etc.). Estos dispositivos pueden tener la capacidad activar las cámaras automáticamente, además de otras funciones.

2. **Red de comunicación:** El sistema IoT depende de una red de comunicación eficiente que permita la transmisión de los datos entre dispositivos de captura y el centro de control o servidores. Los dispositivos pueden comunicarse a través de tecnologías como Wi-Fi, Ethernet, Zigbee o LoRa. La red de comunicación debe ser segura, confiable y capaz de manejar grandes volúmenes de datos en tiempo real, como el video transmitido por las cámaras. El protocolo de comunicación más utilizado en sistemas IoT es MQTT (*Message Queuing Telemetry Transport*), que permite la comunicación eficiente y ligera entre los dispositivos, garantizando que los mensajes sean entregados de forma confiable incluso en redes de baja capacidad.

3. **Plataforma de procesamiento y análisis:** Una vez que los datos (como el video en tiempo real o las lecturas de sensores) son transmitidos a través de la red, es necesario un servidor de procesamiento o plataforma en la nube para analizarlos. Este servidor puede estar ubicado localmente (en el caso de soluciones no basadas en la nube) o en servidores remotos, en plataformas como AWS (*Amazon Web Services*)[5], Google Cloud[6] o Microsoft Azure[7].

La plataforma de procesamiento realiza análisis en tiempo real utilizando algoritmos de inteligencia artificial (IA) y aprendizaje automático para detectar patrones, reconocer objetos o personas, y clasificar eventos relevantes.

4. **Almacenamiento de datos:** El almacenamiento de los datos de video y sensores puede realizarse en dos lugares principales: en la nube o localmente en servidores. El almacenamiento en la nube ofrece flexibilidad, escalabilidad y la posibilidad de acceder a los datos desde cualquier lugar a través de una conexión a Internet. Se pueden utilizar plataformas de almacenamiento en la nube como Amazon S3[8] o Google Drive[9] para guardar videos de largo plazo, realizando copias de seguridad automáticas y mejorando la accesibilidad. Por otro lado, también se puede hacer uso del almacenamiento local, utilizando servidores propios para guardar los datos. Estos sistemas pueden ofrecer mayor control sobre los datos y privacidad, aunque dependen de la capacidad de almacenamiento físico y pueden ser menos escalables que las soluciones basadas en la nube.

5. **Centro de control y monitoreo:** El centro de control es la plataforma donde los usuarios pueden visualizar las cámaras en tiempo real, recibir notificaciones de eventos y gestionar todo el sistema de seguridad. Además de la visualización en tiempo real, los sistemas de monitoreo suelen incluir herramientas de análisis y generación de informes, lo que permite a los usuarios revisar incidentes pasados, gestionar la configuración de cámaras y sensores, o tomar decisiones basadas en los datos recopilados.

6. **Acciones y respuestas automáticas:** Una de las características más avanzadas de los sistemas IoT de videovigilancia es la capacidad de realizar acciones automáticas en respuesta a eventos específicos. Por ejemplo, al detectar un intruso mediante una cámara con análisis inteligente, el sistema puede activar una alarma, enviar notificaciones a los usuarios, encender luces o incluso bloquear el acceso a ciertas áreas. Estas respuestas incrementan la seguridad, ya que las acciones pueden ejecutarse más rápido que en un sistema manual.

2.3. Microcontrolador ESP32-CAM para videovigilancia

Para el desarrollo de este proyecto, se ha seleccionado el microcontrolador ESP32-CAM debido a su bajo coste, alto rendimiento y capacidad para integrar conectividad Wi-Fi y Bluetooth, junto con una cámara. En las siguientes secciones se detallarán las capacidades técnicas del ESP32-CAM y justificaciones de esta elección.

2.3.1. Capacidades técnicas del ESP32-CAM

El ESP32-CAM[10] es un microcontrolador de bajo coste y alto rendimiento que combina conectividad Wi-Fi y Bluetooth con una cámara integrada. Además, como se puede ver en la Figura 2, se trata de un microcontrolador pequeño, lo que lo convierte en una opción ideal para soluciones de videovigilancia compactas y económicas. Este módulo está basado en el chip ESP32-S[11], desarrollado por Espressif Systems[12], y ofrece un rendimiento suficiente para tareas como la captura de imágenes, transmisión de vídeo en tiempo real y procesamiento básico en el propio dispositivo.

Figura 2: Módulo ESP32-CAM con la cámara OV2640 integrada

Las principales características son las siguientes:

- **Procesador:** ESP32 con CPU de doble núcleo (Xtensa LX6[13]), capaz de trabajar a 240 MHz, ofreciendo suficiente potencia de procesamiento para tareas de transmisión de vídeo y compresión de imágenes.
- **Memoria:** 520 KB de SRAM interna y hasta 4 MB de memoria externa PSRAM, que permiten almacenar imágenes temporales y mejorar el rendimiento en aplicaciones de transmisión en tiempo real.
- **Almacenamiento:** Soporte para tarjetas MicroSD (mediante interfaz SPI), útil para guardar capturas o registros localmente.
- **Conectividad:** Wi-Fi 802.11 b/g/n integrada y Bluetooth 4.2, permitiendo una comunicación inalámbrica eficiente para la transmisión de vídeo en red o el control remoto del dispositivo.

- **Cámara:** Sensor OV2640 integrado, capaz de capturar imágenes en resoluciones de hasta 1600x1200 (UXGA), con diferentes formatos de salida (JPEG, BMP, etc.), y configurable según las necesidades del usuario.
- **GPIOs:** Aunque el módulo tiene un número limitado de pines accesibles debido al tamaño reducido, permite conectar sensores adicionales (como sensores PIR de movimiento), LEDs de estado o incluso un flash LED integrado para visión nocturna básica.
- **Consumo energético:** El ESP32-CAM ofrece varios modos de operación que permiten optimizar el consumo energético según las necesidades del sistema. Entre estos modos se incluyen: *Active mode*, en el que el procesador y los periféricos funcionan a plena capacidad; *Modem-sleep*, donde la CPU sigue activa, pero el módem está desactivado; *Light-sleep*, en el que la CPU se detiene, pero los periféricos pueden mantenerse activos; y *Deep-sleep*, el modo de menor consumo, donde solo se conserva el RTC (reloj en tiempo real) y se pueden utilizar despertadores externos o temporizados. Estas características permiten al ESP32-CAM integrarse eficientemente en sistemas autónomos alimentados por batería o energía solar.

Estas capacidades hacen del ESP32-CAM una opción especialmente atractiva para proyectos de videovigilancia, sistemas de monitoreo remoto y aplicaciones IoT en entornos donde el costo, el tamaño y la conectividad son factores críticos. Su versatilidad permite implementaciones tanto locales (con grabación en MicroSD) como remotas (streaming por Wi-Fi), adaptándose a distintos escenarios de uso.

Además, este microcontrolador ha sido utilizado previamente en otros proyectos en los que he participado junto a mi director de TFG. Gracias a esta experiencia previa, he adquirido un conocimiento sólido sobre su funcionamiento y sus capacidades. Por esta razón, y siguiendo la recomendación de mi tutor, se ha optado por incorporarlo también en el desarrollo de este nuevo sistema, con el objetivo de mantener una coherencia técnica con los trabajos anteriores y aprovechar al máximo los recursos ya aprendidos.

2.3.2. Ventajas frente alternativas comerciales

Como se puede ver en la Figura 3 y Figura 4 existen otras alternativas en el mercado orientadas a la videovigilancia, como las cámaras IP de marcas comerciales reconocidas como Xiaomi[14], TP-Link[15], Ezviz[16], etc. Estas cámaras ofrecen soluciones "*plug & play*" con interfaces de usuario amigables y funcionalidades integradas. Sin embargo, el uso del ESP32-CAM presenta una serie de ventajas significativas, especialmente en contextos donde se valora la flexibilidad, el bajo costo y el control total sobre el sistema.

Figura 3: Cámara IP Xiaomi Domo MIJIA 360º Smart Home PTZ

Figura 4: Cámara IP TP-Link Tapo TC 1080p Wi-Fi

A continuación se muestran las ventajas principales del uso del microcontrolador ESP32-CAM frente a otras alternativas comerciales:

- **Coste reducido:** Una de las principales ventajas es el coste. Mientras que las cámaras comerciales pueden superar fácilmente los 30 o 40 euros por unidad, un módulo ESP32-CAM tiene un coste mucho menor, situándose en torno a los 6 y 10 euros. Esto permite implementar sistemas de videovigilancia distribuidos a bajo coste, especialmente útil en proyectos de bajo presupuesto o instalaciones de gran escala.
- **Control total y software libre:** Las soluciones comerciales suelen estar limitadas al ecosistema de la marca, dependiendo de aplicaciones móviles y servidores en la nube. En cambio, con este microcontrolador el usuario puede desarrollar e implementar su propio *software*, controlando totalmente cómo se capturan, procesan, almacenan y transmiten las imágenes.
- **Flexibilidad:** El ESP32-CAM permite una amplia personalización, desde el ajuste de parámetros de la cámara hasta la incorporación de sensores externos (movimiento, temperatura, humedad, etc.). Esta capacidad de adaptación es limitada o inexistente en cámaras comerciales cerradas.

- **Privacidad:** Este microcontrolador permite implementar soluciones completamente locales, sin necesidad de enviar datos a servidores externos o depender de servicios en la nube. Esto supone una ventaja importante en términos de privacidad, ya que el usuario tiene control total sobre dónde se almacenan las imágenes y quién puede acceder a ellas. En lo contrario, muchas cámaras comerciales requieren conectarse a plataformas propietarias que gestionan los datos de forma remota, lo que puede suponer un riesgo si se produce una brecha de seguridad o si se desconocen las políticas de tratamiento de datos.
- **Compatibilidad con IoT:** El ESP32-CAM se integra fácilmente con plataformas IoT como Home Assistant[17], Node-RED[18], MQTT o servidores personalizados, lo que facilita su uso en proyectos domóticos avanzados. En cambio, muchas cámaras comerciales están diseñadas para funcionar únicamente con sus propias apps y servicios.

2.3.3. Comparativa entre microcontroladores

Existen otros microcontroladores en el mercado que, al igual que el ESP32-CAM, ofrecen capacidades útiles para implementar soluciones de videovigilancia o proyectos IoT. Muchos de ellos comparten características clave como conectividad inalámbrica, bajo consumo energético y facilidad de programación, lo que los hace también viables en aplicaciones similares. Sin embargo, al comparar distintas opciones, es importante tener en cuenta factores como el rendimiento, la integración de la cámara, la comunidad de desarrollo o la compatibilidad con otros dispositivos.

A continuación, en la Tabla 2, se presenta una comparativa entre el ESP32-CAM y otros microcontroladores populares utilizados en proyectos de videovigilancia o transmisión de imagen:

Dispositivo	**Cámara Integrada**	**Conectividad**	**Coste**	**Características**
ESP32-CAM	Sí	Wi-Fi y Bluetooth	6-10 €	Bajo consumo pero requiere programación
Rasberry Pi Zero W + Cámara	No	Wi-Fi y Bluetooth	20-30 €	Mayor procesamiento y consumo
Arduino + Cámara	No	Requiere módulos	10-15 €	Requiere varios módulos
ESP32 + Cámara	No	Wi-Fi y Bluetooth	5-6 €	Depende de cámara externa

Tabla 2: Comparación de características entre cámaras IP y cámaras analógicas

Como se puede ver, el ESP32-CAM ofrece una solución equilibrada en cuanto a coste, facilidad de uso e integración, especialmente en proyectos donde se prioriza la eficiencia y la simplicidad. Otras opciones ofrecen mayores capacidades, pero a costa de mayor complejidad técnica, consumo energético o precio.

2.3.4. Consideraciones prácticas y limitaciones

A pesar de las numerosas ventajas que ofrece el ESP32-CAM como solución de videovigilancia, su implementación conlleva una serie de aspectos prácticos y limitaciones técnicas que deben tenerse en cuenta antes de adoptarlo en un proyecto real.

Uno de los factores más relevantes es la calidad de imagen. La cámara que incorpora el ESP32-CAM (modelo OV2640) proporciona una resolución aceptable para tareas básicas, pero queda lejos de la calidad que ofrecen muchas cámaras comerciales, especialmente en condiciones de poca iluminación o cuando se requieren imágenes con alto nivel de detalle. Esto puede representar una desventaja en escenarios donde la nitidez o la fiabilidad visual son críticas.

Otro aspecto a considerar es su capacidad de procesamiento. Si bien el módulo es eficiente para tareas simples de captura y transmisión de imágenes, no está diseñado para realizar procesamiento de vídeo avanzado. Funciones como el reconocimiento facial, la detección de movimiento precisa o el análisis de vídeo suelen requerir dispositivos con mayor potencia computacional, como una Raspberry Pi[19]. Por lo que, si se quiere hacer uso de estas tareas más exigentes, será necesario apoyarse en servidores externos o integrar el sistema con otros dispositivos capaces de asumir la carga de procesamiento.

La conectividad Wi-Fi integrada es uno de sus puntos fuertes, pero también puede representar una limitación en entornos con señal inestable o zonas amplias donde la cobertura es irregular. En estos casos, es posible que se necesite el uso de repetidores o redes alternativas, aumentando el costo.

Finalmente, desde el punto de vista de la configuración y el mantenimiento, aunque el ESP32-CAM es accesible para desarrolladores con cierta experiencia, su uso puede resultar desafiante para usuarios sin conocimientos técnicos. La necesidad de programar el dispositivo y configurar correctamente la red puede ser un obstáculo en proyectos que buscan una solución completamente "*plug & play*".

En resumen, el ESP32-CAM es una opción muy válida para sistemas de videovigilancia de bajo presupuesto en los cuales se busca hacer énfasis en la privacidad, pero presenta limitaciones en cuanto a calidad, procesamiento y facilidad de uso. Evaluar estas cuestiones es esencial para determinar si es la solución adecuada para un caso de uso específico.

2.4. Comunicación mediante el protocolo MQTT

Para la transmisión de datos en el sistema de videovigilancia propuesto, se ha decidido utilizar el protocolo MQTT debido a su comunicación ligera y confiable entre dispositivos, específicamente con el ESP32-CAM. A continuación, se explica el funcionamiento básico de MQTT, sus componentes principales y la justificación de su uso.

2.4.1. Funcionamiento básico del protocolo MQTT

El protocolo MQTT (*Message Queuing Telemetry Transport*)[22] es un protocolo de mensajería ligero, diseñado para entornos con recursos limitados y redes poco fiables. Se utiliza ampliamente en aplicaciones del Internet de las Cosas (IoT) debido a su bajo consumo de ancho de banda y su capacidad para gestionar gran cantidad de dispositivos distribuidos.

MQTT se basa en un modelo de comunicación del tipo publicador-suscriptor, lo que permite una arquitectura desacoplada entre emisores y receptores de mensajes. Esto significa que los dispositivos no necesitan conocer la identidad ni la ubicación de los demás dispositivos, lo que facilita la escalabilidad y la flexibilidad en la comunicación.

Como se puede ver en la Figura 5 el protocolo MQTT se basa en cuatro componentes principales:

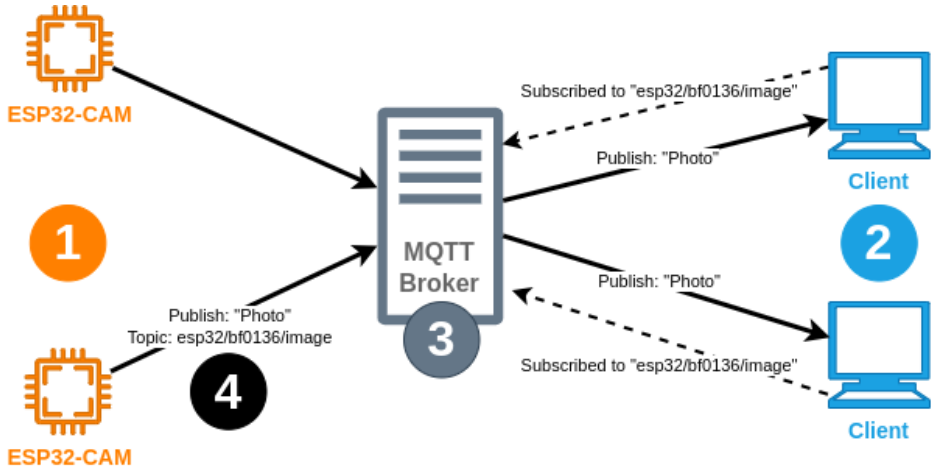

Figura 5: Diagrama de componentes del protocolo MQTT

1. **Publicadores (*publishers*)**: Son los encargados de enviar mensajes. Estos generan datos o eventos y los transmiten a través del protocolo MQTT, sin necesidad de saber quién los va a recibir. Por ejemplo, se pueden publicar valores de un sensor, mensajes de estado, imágenes o cualquier tipo de información. Estos mensajes se publican en un *topic* concreto.

2. **Suscriptores (*subscribers*):** Son los dispositivos o aplicaciones que se suscriben a uno o varios *topics* con el objetivo de recibir los mensajes que se publiquen en ellos. Los suscriptores no necesitan conocer el origen de los datos; simplemente reciben los mensajes que coinciden con los temas a los que están suscritos. Esto permite que múltiples suscriptores reciban el mismo mensaje de forma simultánea.

3. ***Broker*:** El *broker* es el núcleo del sistema MQTT. Se trata de un servidor que recibe todos los mensajes de los publicadores y los reenvía a los suscriptores que estén interesados. Es responsable de gestionar las conexiones, distribuir los mensajes de forma eficiente y garantizar, en función del nivel de calidad de servicio (QoS), que los mensajes se entregan correctamente. Sin el *broker*, la comunicación entre publicadores y suscriptores no sería posible.

4. **Tópicos (*topics*):** Son identificadores jerárquicos que actúan como canales de comunicación. Cada mensaje que se publica en MQTT va asociado a un *topic*, y los suscriptores indican a qué *topics* desean acceder. Por ejemplo, uno de ellos podría ser "esp32/image", y un cliente suscrito al *topic* recibiría todos los mensajes que se publiquen en este. Los *topics* permiten una organización estructurada y lógica del flujo de mensajes dentro del sistema.

2.4.2. Seguridad en sistemas MQTT

El protocolo MQTT no fue creado originalmente con un enfoque centrado en la seguridad. Esto significa que, para su aplicación en entornos donde la confidencialidad y la integridad de los datos son críticas, es necesario implementar mecanismos adicionales que refuercen su protección frente a posibles amenazas.

TLS (*Transport Layer Security*)[20] es el estándar actual para el cifrado de comunicaciones en red, y se puede utilizar junto al protocolo MQTT para garantizar la confidencialidad e integridad de los mensajes transmitidos entre los clientes y el *broker*. Al establecer una conexión TLS, los datos viajan cifrados, de modo que incluso si la comunicación fuera interceptada, el contenido resultaría ilegible para un atacante. Esta característica es especialmente relevante cuando se utiliza MQTT sobre redes públicas o poco seguras, ya que protege la información frente a ataques de *sniffing*[21] o manipulación de paquetes.

TLS (Transport Layer Security) es el estándar actual para el cifrado de comunicaciones en red, y se puede utilizar junto al protocolo MQTT para garantizar la confidencialidad e integridad de los mensajes transmitidos entre los clientes y el *broker*. Al establecer una conexión TLS, los datos viajan cifrados, de modo que incluso si la comunicación fuera interceptada, el contenido resultaría ilegible para un atacante. Esta característica es especialmente relevante cuando se utiliza MQTT sobre redes públicas o poco seguras, ya que protege la información frente a ataques de *sniffing* o la manipulación de paquetes.

Junto al cifrado, otro pilar básico de la seguridad en MQTT es la autenticación de los clientes. La forma más habitual de implementar este control es mediante el

uso de usuario y contraseña. Dichas credenciales las debe proporcionar el usuario al establecer la conexión con el *broker*. Esta autenticación básica permite verificar la identidad del cliente antes de permitirle publicar o suscribirse a cualquier tópico, y es un primer filtro importante para evitar accesos no autorizados. Aunque se trata de un mecanismo sencillo, cuando se combina con una conexión cifrada mediante TLS, proporciona una protección adecuada para muchos escenarios prácticos, especialmente en aplicaciones domésticas o de pequeña escala.

2.4.3. Comparativa de MQTT con otros protocoles comunes

Aparte del protocolo MQTT, existen otros protocolos comunes que también son utilizados para la comunicación entre dispositivos en entornos distribuidos y de IoT, como HTTP[23] y CoAP[24]. Cada uno de estos protocolos tiene sus propias características y ventajas dependiendo del tipo de aplicación, los requisitos de comunicación y las limitaciones del sistema. A continuación, se compara MQTT con estos otros protocolos, destacando sus diferencias principales y las situaciones en las que cada uno puede ser más adecuado.

MQTT vs. HTTP: HTTP (*Hypertext Transfer Protocol*) es uno de los protocolos más utilizados en la comunicación en redes, especialmente en aplicaciones web. Sin embargo, para aplicaciones de IoT, como un sistema de videovigilancia, HTTP presenta algunas limitaciones. HTTP sigue un modelo de comunicación cliente-servidor, en el cual el cliente debe hacer solicitudes constantes al servidor para obtener datos. Esto puede generar una sobrecarga significativa en el tráfico de la red y aumentar la latencia, lo que es especialmente problemático en sistemas que requieren actualizaciones en tiempo real. Por otro lado, MQTT usa un modelo publicador-suscriptor que mantiene una conexión persistente entre el cliente y el *broker*, lo que permite una comunicación más eficiente, con menos sobrecarga y un menor uso de ancho de banda en comparación con HTTP, que necesita abrir y cerrar conexiones repetidamente.

MQTT vs. CoAP: CoAP (*Constrained Application Protocol*) es otro protocolo diseñado específicamente para aplicaciones de IoT y redes con recursos limitados. A diferencia de HTTP, CoAP utiliza UDP en lugar de TCP, lo que lo hace más ligero y rápido en cuanto a latencia. CoAP también implementa un modelo de comunicación similar al de HTTP, pero de manera más eficiente, con soporte para las operaciones de *request/response*. Aunque CoAP es eficiente en términos de ancho de banda y latencia, su principal desventaja frente a MQTT radica en el manejo de la comunicación. CoAP es más adecuado para comunicaciones de corto alcance y situaciones donde los dispositivos tienen una comunicación ocasional. Sin embargo, MQTT funciona mucho mejor en entornos donde se requiere comunicación constante y en tiempo real entre dispositivos, como es el caso de un sistema de videovigilancia.

2.5. Sistema propuesto

El sistema de videovigilancia propuesto en este proyecto se basa en el uso del microcontrolador ESP32-CAM como unidad principal de captura y transmisión de imágenes. Esta propuesta tiene como objetivo ofrecer una alternativa económica, flexible y respetuosa con la privacidad del usuario frente a los sistemas comerciales existentes.

El núcleo del sistema está compuesto por uno o varios módulos ESP32-CAM configurados para capturar vídeo o imágenes en tiempo real y transmitirlos mediante el protocolo MQTT a través de una red Wi-Fi local. La elección de este protocolo se debe a su bajo consumo de ancho de banda y su eficiencia para entornos con recursos limitados, lo que lo hace especialmente adecuado para dispositivos IoT.

El sistema incluye las siguientes funcionalidades clave:

- **Configuración remota simplificada:** Cada ESP32-CAM se configura mediante una aplicación de escritorio que utiliza el punto de acceso del dispositivo, permitiendo establecer los parámetros de red Wi-Fi y el *broker* MQTT sin necesidad de modificar el código fuente. Esta herramienta facilita la incorporación de nuevos nodos al sistema de forma simple y sin requerir conocimientos avanzados.

- **Visualización centralizada en tiempo real:** Desde otra aplicación de escritorio es posible visualizar simultáneamente hasta 12 cámaras activas. La interfaz permite seleccionar y monitorizar en vivo las transmisiones, ofreciendo una solución eficiente para la vigilancia de múltiples espacios desde un único punto de control.

- **Control de parámetros de cámara:** La aplicación principal permite modificar dinámicamente ajustes del sensor de cada ESP32-CAM, como la resolución, el brillo, la calidad de imagen o el contraste, adaptando el rendimiento de cada dispositivo a las condiciones específicas del entorno.

- **Captura y grabación manual:** El usuario puede capturar imágenes instantáneas o grabar vídeo de forma manual directamente desde la aplicación, permitiendo documentar situaciones específicas según las preferencias.

- **Detección de movimiento con notificación automática:** A través de un aplicativo de terminal externo, se puede levantar un servicio de monitoreo que analiza el vídeo de las cámaras activas en busca de movimiento. Al detectarse un evento, el sistema genera automáticamente una alerta, graba el momento del suceso y lo envía mediante Telegram a los clientes o chats previamente configurados para cada cámara en la aplicación de escritorio.

- **Privacidad y control local de datos:** Todo el sistema está diseñado para operar en red local, sin necesidad de servicios en la nube. Todo el sistema se gestiona desde el entorno del usuario, asegurando la privacidad de los datos capturados.

- **Seguridad y control de acceso:** Además, el sistema permite habilitar cifrado de comunicaciones mediante TLS en el protocolo MQTT, asegurando la confidencialidad e integridad de los datos transmitidos entre dispositivos. También se contempla la autenticación mediante usuario y contraseña, lo que evita accesos no autorizados a las cámaras o al *broker*.

3. Ingeniería de concepto

En esta sección se definen y estructuran los componentes principales del sistema, estableciendo la arquitectura general y describiendo las funciones de cada módulo, así como las interacciones y comunicaciones entre ellos.

3.1. Arquitectura del sistema

Para crear el sistema propuesto, se ha dividido el proyecto de *software* en diversos submódulos que implementan sus funciones correspondientes, lo que permite una gestión más eficiente y escalable del desarrollo. Esta modularización asegura que cada componente pueda ser desarrollado, probado y mejorado de manera independiente, reduciendo la complejidad y facilitando la incorporación de nuevas funcionalidades en el futuro.

En las siguientes subsecciones se describen los módulos principales que conforman la arquitectura y sus responsabilidades.

3.1.1. Programa inicializador de la cámara

En este submódulo se implementa el programa que permite configurar las cámaras con los parámetros necesarios para usar MQTT y la red Wi-Fi deseada. En la Figura 6 se puede ver la interfaz gráfica de dicho programa.

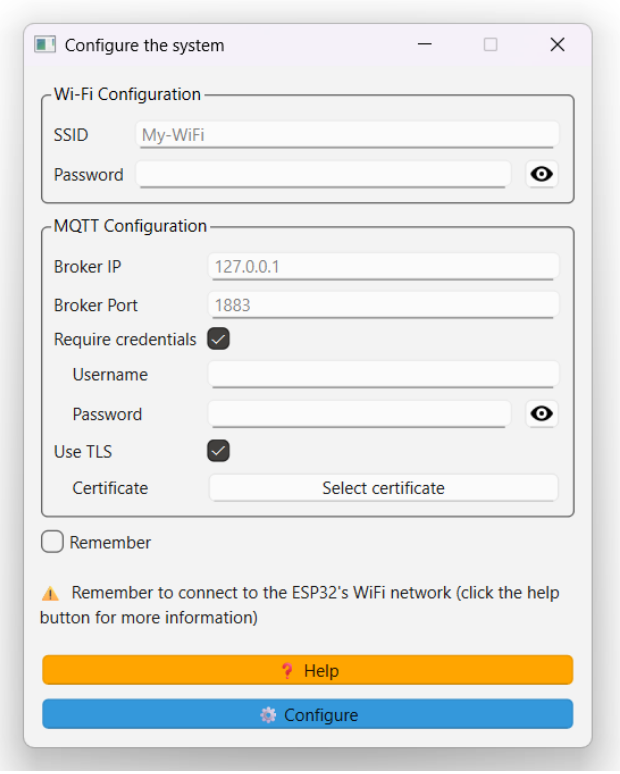

Figura 6: Interfaz gráfica del programa configurador del ESP32-CAM

3.1.2. Programa principal

El programa principal es el encargado de gestionar y visualizar las diferentes cámaras conectadas en tiempo real. Además, permite configurar los diferentes parámetros de los sensores de las cámaras, como pueden ser el brillo, la resolución, la ganancia, etc. En la Figura 7 se puede ver la pantalla principal de la aplicación utilizando 12 cámaras simuladas.

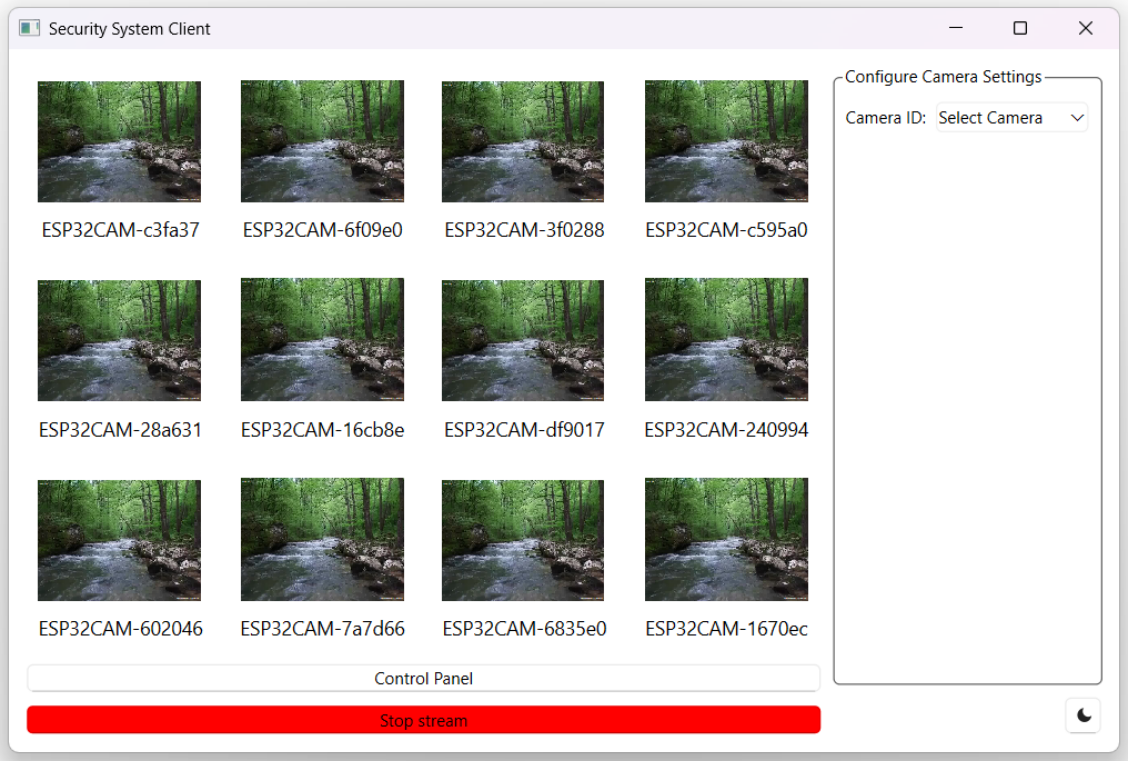

Figura 7: Interfaz gráfica del programa principal con 12 cámaras simuladas

Asimismo, el programa permite capturar imágenes o grabar el contenido visualizado por las cámaras, además de configurar el servicio de notificaciones en caso de que esté activo (ver Figura 8).

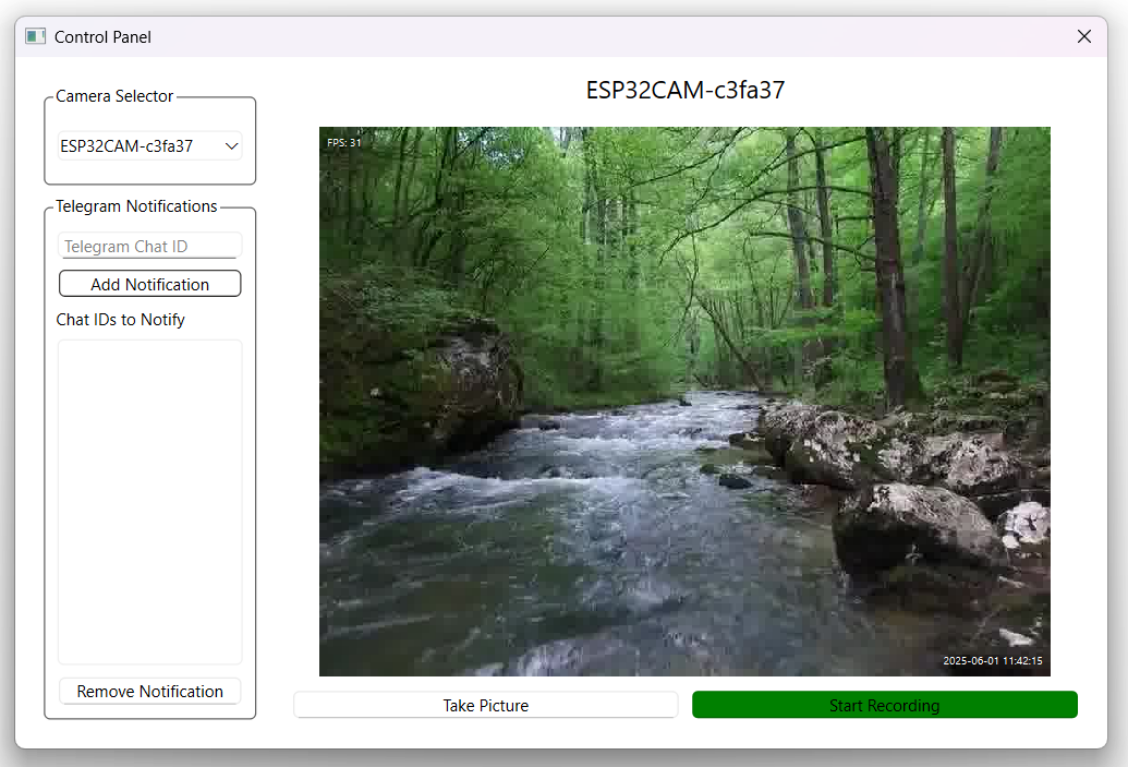

Figura 8: Panel de captura del programa principal

3.1.3. Servicio de notificaciones

El módulo de servicio de notificaciones contiene toda la lógica que se encarga de detectar movimientos en las cámaras y notificar a los chats de Telegram configurados desde el programa principal. Este módulo no contiene interfaz gráfica, ya que su objetivo es trabajar como un servicio en segundo plano del sistema.

Además, este módulo está pensado para que funcione de forma totalmente complementaria al programa principal. Por lo que ofrece flexibilidad para seguir desarrollando soluciones con algoritmos más complejos y avanzados a futuro.

3.1.4. Programa del ESP32-CAM

El programa del ESP32-CAM es el código encargado de gestionar la lógica que debe seguir el microcontrolador para su correcto funcionamiento dentro del sistema.

Este código se carga inicialmente en la memoria del ESP32-CAM e incluye la inicialización de la cámara, la conexión a la red Wi-Fi especificada y el establecimiento de la comunicación con el *broker* MQTT para el envío y recepción de mensajes.

3.1.5. Broker MQTT

Para poder interactuar utilizando el protocolo MQTT, se ha creado un módulo de Docker[25] que permite desplegar un contenedor con un *broker* MQTT con las configuraciones y credenciales deseadas.

3.2. Flujo de comunicación

Para que el sistema desarrollado sea robusto y fiable, es fundamental que el flujo de comunicación entre los diferentes componentes esté bien estructurado. Por ello, en esta sección se describen los pasos involucrados en la configuración y funcionamiento del sistema, desde la conexión inicial hasta la recepción de notificaciones de eventos relevantes.

3.2.1. Configuración inicial de la cámara

La configuración inicial de la cámara ESP32-CAM es el primer paso fundamental en la configuración del sistema propuesto.

Cuando el dispositivo ESP32-CAM se enciende por primera vez, actúa como un punto de acceso Wi-Fi temporal (*Access Point*) y despliega un servidor HTTP. El AP y el servidor HTTP permiten a los usuarios conectarse al ESP32-CAM con el fin de especificar al dispositivo los parámetros necesarios para su funcionamiento (ver Figura 9).

A través de la aplicación de configuración, el usuario puede ingresar los parámetros de red necesarios, como el nombre de la red Wi-Fi (SSID), el *broker* MQTT a utilizar, etc. Estos datos se transmiten al ESP32-CAM que, seguidamente, desconecta su AP temporal y se conecta a la red Wi-Fi proporcionada.

Este paso es crucial para que el sistema se integre correctamente a la infraestructura de la red local, permitiendo la transmisión de imágenes y datos a través del protocolo MQTT.

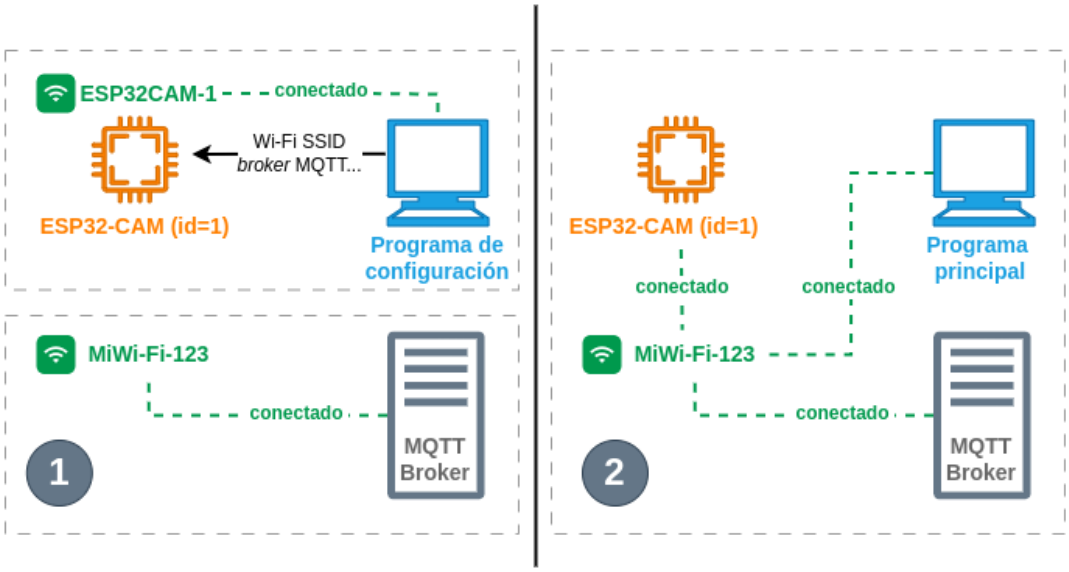

Figura 9: Comunicación para la configuración inicial de la cámara

3.2.2. Conexión inicial mediante MQTT

Una vez que las ESP32-CAM han sido configuradas, el programa principal necesita identificar cuántas cámaras están activas y correctamente conectadas al *broker* MQTT, con el fin de poder recibir sus imágenes y establecer comunicación individual con cada una de ellas.

Para lograrlo, se ha implementado un flujo de conexión inicial que permite detectar automáticamente las cámaras disponibles (ver Figura 10). Este proceso comienza con el envío de un mensaje de tipo "*ping*" por parte del programa principal a un *topic* específico al que están suscritas todas las cámaras.

Al recibir este mensaje, cada cámara activa responde publicando un mensaje "*pong*" en otro *topic* designado para este fin. Esta respuesta incluye información esencial como su identificador único (ID) y los parámetros de configuración actuales.

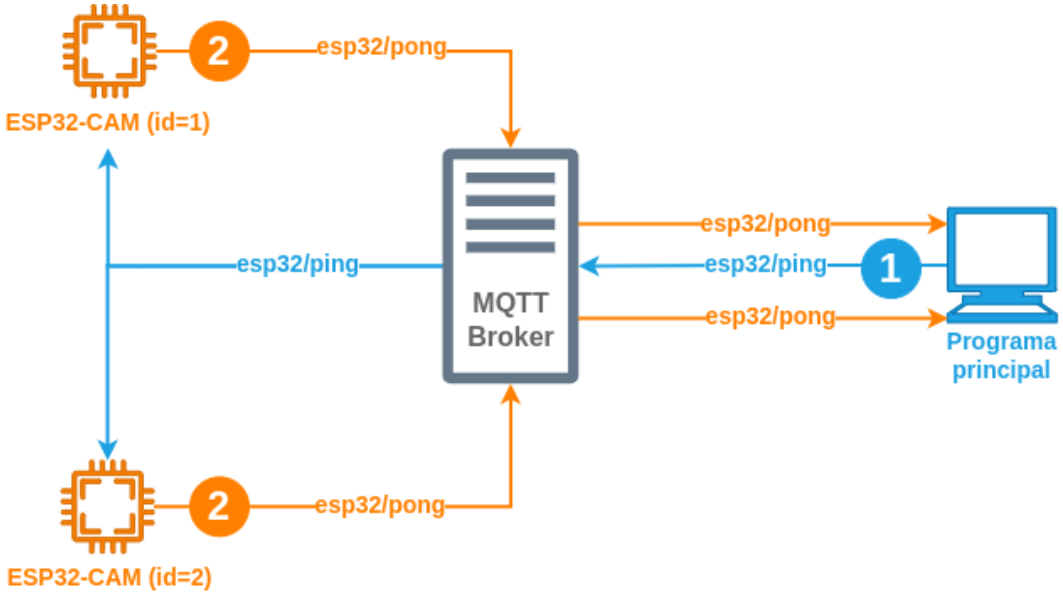

Figura 10: Comunicación para la detección de cámaras conectadas

Gracias a este intercambio de mensajes, el programa principal puede construir una lista precisa y actualizada de las cámaras que se encuentran en línea y listas para operar, permitiendo así una gestión dinámica y flexible.

Además de identificar las cámaras activas, el programa principal necesita identificar si el servicio de notificaciones mediante Telegram está disponible. Para ello, se sigue un patrón similar al anterior: el programa principal envía un mensaje de tipo "*ping*" a un *topic* específico al que está suscrito el servicio de notificaciones. Si este servicio está en ejecución, responde con un mensaje "*pong*" que confirma su disponibilidad (ver Figura 11).

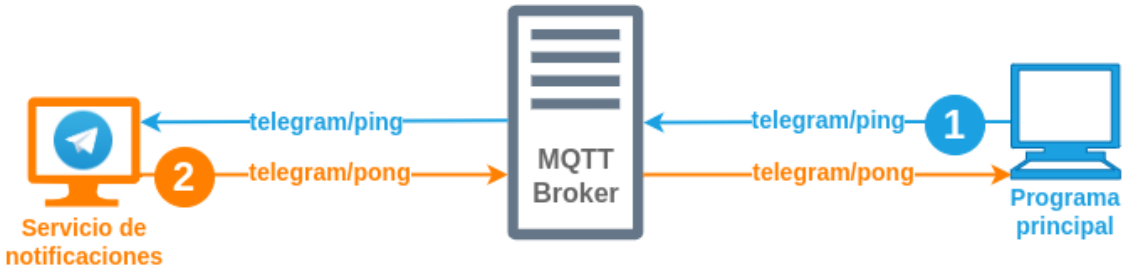

Figura 11: Comunicación para la detección del servicio de notificaciones

Este mecanismo unificado de detección permite iniciar el sistema de forma ordenada y centralizada, asegurando que el programa principal solo muestre e interactúe con los componentes correspondientes a las cámaras y servicios de notificación que realmente están disponibles en ese momento.

3.2.3. Transmisión de imágenes

Una vez que el programa principal ha identificado las cámaras activas, se suscribe a un *topic* específico para cada una de ellas, a través del cual podrá recibir sus imágenes (ver Figura 12). Cada cámara utiliza su propio *topic* para enviar las capturas, lo que permite una gestión individualizada del flujo de datos.

Las cámaras transmiten imágenes de forma continua, enviándolas tan rápido como su capacidad de procesamiento y la conexión de red lo permitan. De esta manera, el sistema asegura una vigilancia en tiempo real, adaptándose al rendimiento de cada dispositivo sin necesidad de sincronización externa.

Finalmente, las imágenes que se reciben son procesadas y mostradas por el programa.

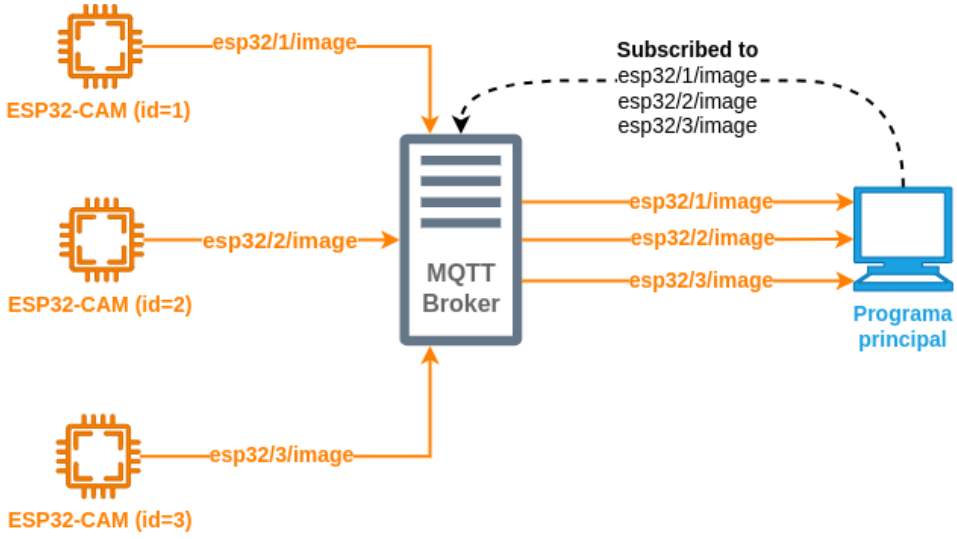

Figura 12: Comunicación para el envío de imágenes

3.2.4. Configuración del sensor

Cuando el programa principal está en ejecución, permite la configuración de los sensores de las cámaras conectadas al sistema en tiempo real. Para llevar a cabo esta configuración, el programa envía los parámetros de configuración específicos al *topic* correspondiente de cada cámara. Estos parámetros pueden incluir ajustes como la ganancia, resolución del sensor, etc.

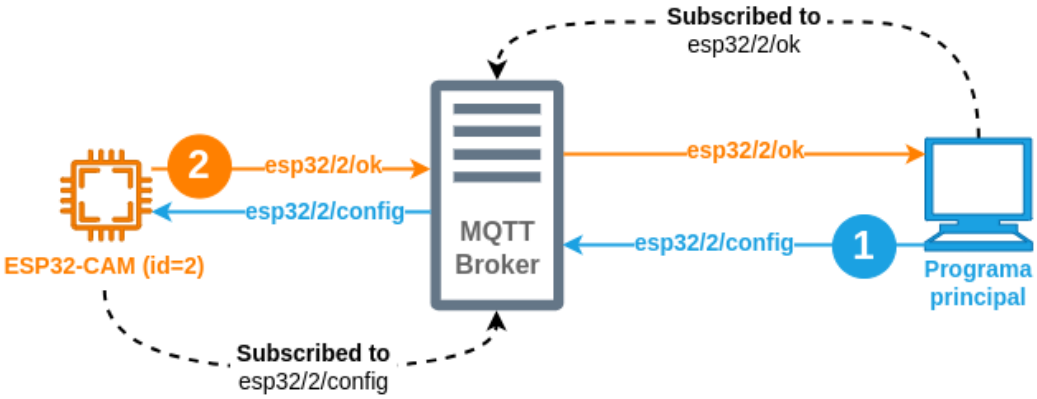

Figura 13: Comunicación para la configuración del sensor

Una vez que la configuración se envía, el programa espera recibir una respuesta de la ESP32-CAM usando un *topic* correspondiente (ver Figura 13). Esta respuesta es un mensaje que confirma que la cámara ha recibido y procesado correctamente la configuración, indicando que el sensor ha sido ajustado según los parámetros enviados.

3.2.5. Configuración de notificaciones

El flujo de comunicación para la configuración de las notificaciones sigue un proceso similar al de la configuración de los sensores de la cámara, utilizando el protocolo MQTT para establecer la comunicación.

En primer lugar, el programa principal envía a un *topic* el ID de la cámara que se desea monitorizar junto con el chat ID de Telegram correspondiente.

Una vez que el servicio de notificaciones recibe esta información, responde confirmando que la configuración ha sido realizada correctamente (ver Figura 14). A partir de ese momento, el sistema podrá enviar notificaciones automáticamente al chat de Telegram especificado, alertando sobre movimientos detectados en las cámaras.

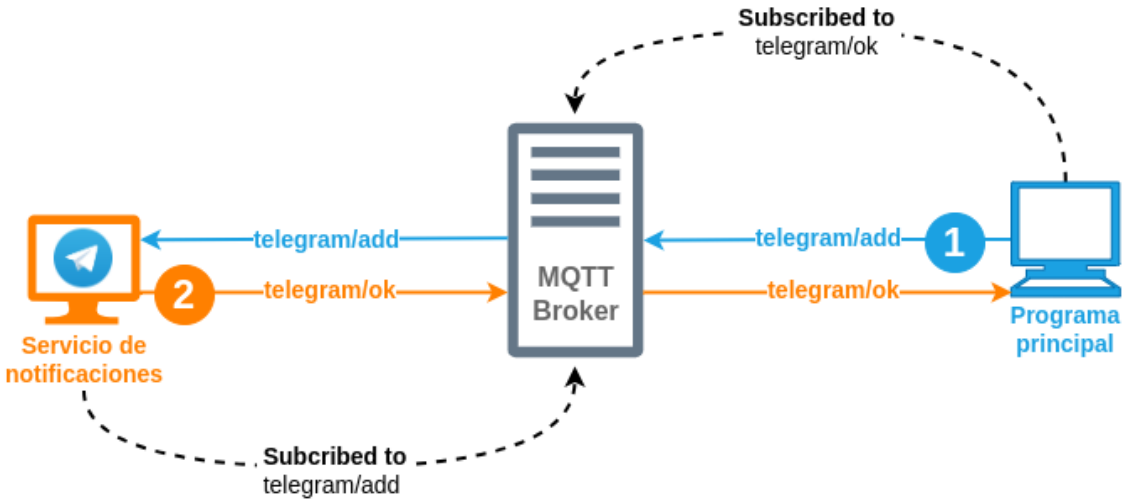

Figura 14: Comunicación para añadir las notificaciones de telegram

Para poder eliminar las notificaciones configuradas, se sigue el mismo flujo de comunicación, pero utilizando un *topic* diferente para mandar la información necesaria.

4. Ingeniería de detalle

4.1. Programa del ESP32-CAM

El código desarrollado para el módulo ESP32-CAM tiene como objetivo implementar un sistema que pueda integrarse con el programa principal, permitiendo la configuración del microcontrolador sin necesidad de modificar continuamente el código fuente. Para lograrlo, se ha diseñado un mecanismo basado en parámetros dinámicos que se reciben de forma remota. Durante la primera fase, el microcontrolador despliega un *Access Point*[32] que permite la configuración a través de un servidor HTTP. Una vez establecida la conectividad, la configuración en tiempo real se gestiona mediante mensajes enviados a través del protocolo MQTT[22]. Este enfoque permite adaptar el comportamiento del sistema de forma flexible y remota, sin requerir la reprogramación manual del ESP32-CAM.

4.1.1. Estados del sistema

El comportamiento que debe seguir el microcontrolador se ha modelado mediante un diagrama de flujo (*flow chart*) que representa las diferentes fases por las que pasa el sistema desde su arranque hasta su funcionamiento esperado.

Al encenderse, el microcontrolador se inicializa y despliega un AP con el nombre de `ESP32CAM-<ID>` junto a un servidor HTTP que espera una solicitud al *endpoint* `/configure` con los parámetros necesarios: Wi-Fi SSID[33] con su contraseña, IPv4[4] del *broker* MQTT y sus credenciales junto a un certificado TLS[34] en caso de que sea necesario. Una vez que el microcontrolador recibe los parámetros, desconecta el AP y se intenta conectar a la red Wi-Fi; si consigue conectarse, procede a conectarse al *broker* MQTT.

Tras conectarse a la red Wi-Fi y al *broker* MQTT, envía imágenes constantemente al *topic* `/esp32/ID/image`. En caso de que en cualquier momento reciba una configuración por el *topic* `/esp32/ID/config`, deja de enviar imágenes y configura el sensor con los parámetros recibidos. Después de configurar el sensor, se siguen enviando imágenes.

Si durante el proceso de configuración o en el envío de imágenes el dispositivo pierde la conexión con el *broker* MQTT, se inicia automáticamente un intento de reconexión que se repetirá hasta restablecer la comunicación. Del mismo modo, si se pierde la conexión con la red Wi-Fi, el sistema intentará reconectarse de forma continua hasta lograr una conexión estable.

Para poder entender mejor el funcionamiento, en la Figura 15 se muestra de forma visual la lógica descrita anteriormente:

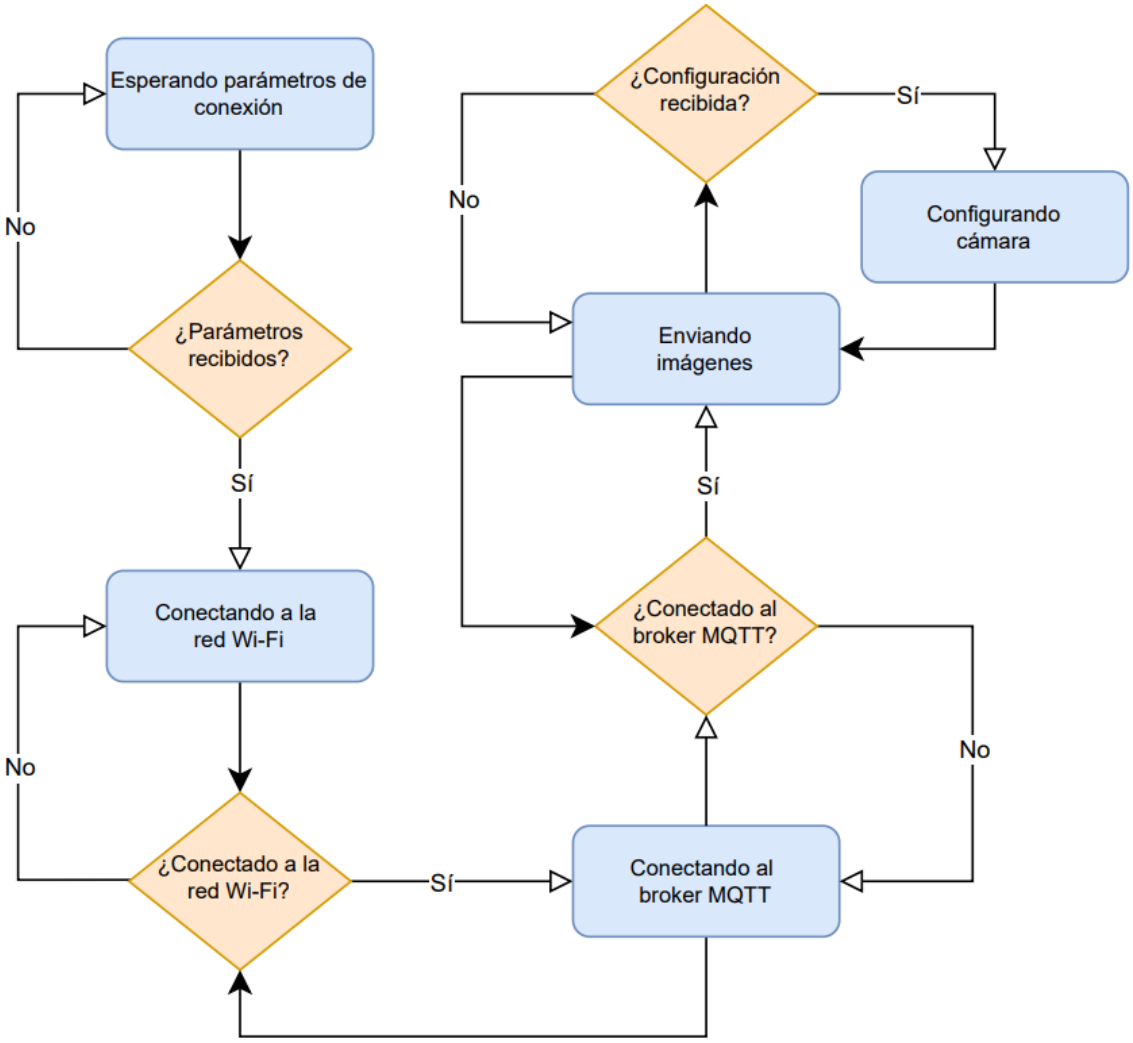

Figura 15: Diagrama de flujo del código del microcontrolador ESP32-CAM

4.1.2. Entorno de desarrollo y herramientas

Para el desarrollo del programa del microcontrolador ESP32-CAM, se ha utilizado el programa de código abierto Arduino IDE en la versión 2.3.6[35], ya que permite escribir, compilar, cargar y depurar código a microcontroladores de una forma sencilla y con buen soporte.

Para que el entorno de desarrollo funcione junto al ESP32-CAM, se ha añadido el Board Manager oficial de Espressif Systems de la versión 3.0.7[36], que incluye compatibilidad con los modelos más comunes, incluyendo la AI Thinker ESP32-CAM utilizada en este proyecto.

Aparte de las librerías ya incluidas por defecto con Arduino, se instalaron librerías externas para poder facilitar diferentes funciones dentro del programa. A continuación, se muestran las librerías externas utilizadas:

- **PubSubClient 2.8.0[26]:** Permite la comunicación mediante el protocolo MQTT.
- **ArduinoJson 7.4.1[27]:** Se utiliza para codificar y descodificar mensajes JSON[37].
- **ESPAsyncWebSrv 1.2.9[28]:** Proporciona un servidor web asíncrono que permite gestionar peticiones HTTP desde el ESP32.
- **AsyncTCP 1.1.4[29]:** Libreria requerida para el funcionamiento del servidor asíncrono.
- **ESPAsyncTCP 1.2.4[30]:** Requerimiento para poder utilizar el servidor asíncrono en placas ESP32.

Todas estas librerías han sido instaladas y gestionadas mediante el gestor de librerías del propio Arduino IDE 2[35], ya que ayuda a su correcta integración y compatibilidad.

4.1.3. Estructura del código

El código de este programa se estructura en torno a tres archivos principales:

- `esp32-cam-client.ino` que contiene el código principal del programa, incluyendo la inicialización de la cámara, la conexión Wi-Fi y la transmisión de imágenes.
- `camera_pins.h` que define la asignación de pines entre el microcontrolador y el módulo de cámara OV2640.
- `partitions.csv` que establece el esquema de partición de la memoria flash del dispositivo, organizando el almacenamiento de firmware y otros datos esenciales para su funcionamiento.

4.1.4. Captura y transmisión de imágenes

Una de las partes más importantes de este programa es la captura y transmisión de imágenes, ya que debe realizarse de forma rápida y eficiente para garantizar una visualización prácticamente en tiempo real. Esto requiere escoger un formato de imagen adecuado para poder comprimir significativamente las imágenes sin perder demasiada calidad.

En este caso se ha escogido el formato JPEG[38], ya que ofrece un tamaño de imagen más reducido comparado con otros formatos disponibles. Además, se adecúa mejor al sistema, ya que al tener menor tamaño, permite que la transmisión de imágenes sea más rápida, lo cual resulta fundamental si el sistema debe mostrar imágenes en tiempo real.

Para justificar esta elección, en la Tabla 3 se muestra una comparativa de tamaños entre distintos formatos de imagen soportados por el ESP32-CAM usando la resolución 400x296 píxeles. Esta comparación permite visualizar claramente las diferencias en tamaño de imagen, mostrando las ventajas del uso de JPEG en un entorno donde el rendimiento y la velocidad son prioritarios.

Formato	**Tamaño**	**Compresión**
RGB565	236800 Bytes	No
RGB888	355200 Bytes	No
GRAYSCALE	118400 Bytes	No
JPEG	$\approx$ 6500 Bytes	Sí

Tabla 3: Comparativa de formatos de imagen disponibles para el ESP32-CAM

4.2. Programa inicializador

Como se ha visto en el apartado 4.1.1 cuando se enciende el microcontrolador, este despliega su AP y un servidor HTTP que permite configurar el dispositivo. Por lo tanto, el objetivo principal de este programa es mostrar una interfaz que permita al usuario proporcionar los parámetros necesarios y enviárselos al ESP32-CAM de una forma sencilla.

4.2.1. Lenguaje y librerías

Para la implementación de este programa se ha utilizado el lenguaje de programación Python 3.13.3[39], ya que es un lenguaje de alto nivel que facilita el desarrollo de aplicaciones gracias a su gran repertorio de bibliotecas.

Además, para la interfaz gráfica se ha utilizado Qt6[40], ya que es un *framework* muy completo, multiplataforma y que permite diseñar en detalle interfaces gráficas. Para integrarlo con Python, se ha utilizado el *binding* PyQt6.

A continuación se muestra un listado de las bibliotecas externas de Python utilizadas para este programa:

- `PyQt6 6.9.0`[41]: Permite crear interfaces gráficas usando Qt6 y Python.
- `Requests 2.32.3`[42]: Permite realizar solicitudes HTTP de forma simple.

Para programar todo el programa, se ha utilizado el editor de código Visual Studio Code[46] de Microsoft[47], ya que es muy versátil y contiene muchas extensiones útiles para el desarrollo.

4.2.2. Estructura del código

- `assets/`: Carpeta que contiene las imágenes utilizadas.
- `Main.py`: Es el archivo que permite ejecutar el programa con todos sus elementos.
- `ui/`: Carpeta que contiene todos los archivos de elementos visuales.
  - `LoadingPopup.py`: Clase que permite mostrar un *popup* de carga cuando el programa lo necesite.
  - `MainView.py`: Clase de la pantalla principal del programa.
  - `MQTTGroup.py`: Clase que crea el formulario que permite introducir los valores de relacionados con MQTT.
  - `WifiGroup.py`: Clase que crea el formulario que permite introducir los valores relacionados con el Wi-Fi.
- `utils/`: Carpeta que contiene los archivos con utilidades para el programa.

- `ConfigureThread.py`: Clase que permite crear un hilo para enviar la configuración por HTTP.
- `api_client.py`: Contiene la función para mandar la configuración por HTTP.
- `config.py`: Contiene las funciones para guardar o recuperar el formulario enviado en caso de que se quiera reutilizar.

4.2.3. Funcionamiento

En este apartado se detalla cómo es el flujo de ejecución del programa y su funcionamiento.

Cuando se ejecuta el programa, se muestra la ventana principal (ver Figura 16) donde se puede introducir el SSID y la contraseña de la red Wi-Fi donde se encuentra el *broker* MQTT. Además, también permite introducir la IPv4 y el puerto del *broker* dentro de la red Wi-Fi junto a unas credenciales en caso de que sean necesarias. Esta ventana también da la opción de seleccionar un certificado TLS correspondiente al bróker MQTT en caso de que el bróker esté configurado para que funcione con cifrado TLS.

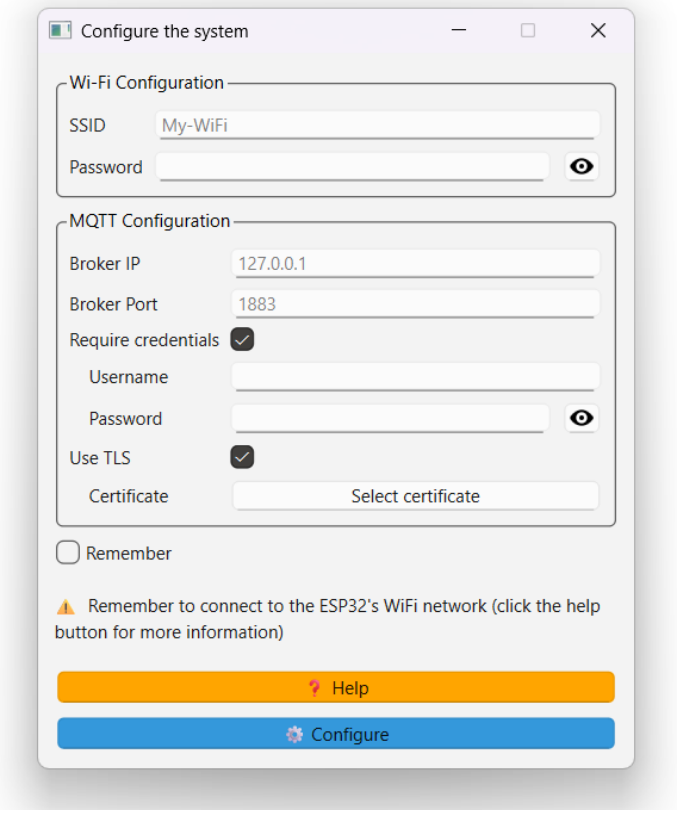

Figura 16: Ventana principal del programa inicializador del ESP32-CAM

Como se puede ver en la advertencia de la ventana principal (Figura 16), para configurar una cámara específica, es necesario conectarse a su red Wi-Fi. Para

entender mejor esto, se ha creado un botón que muestra un *popup* de ayuda (ver Figura 17).

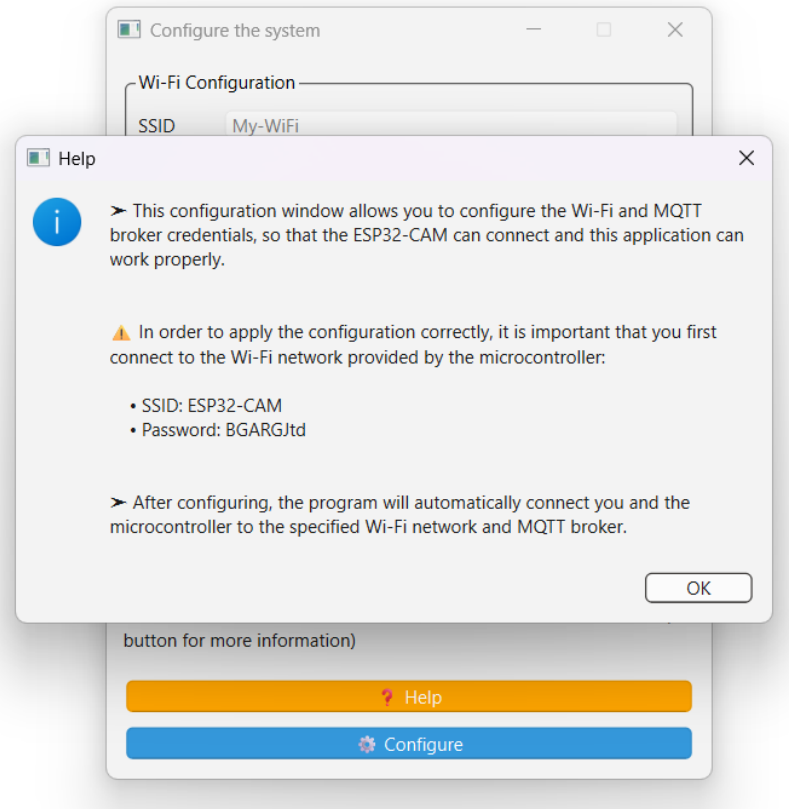

Figura 17: Ventana de ayuda del programa inicializador del ESP32-CAM

A continuación, en la Figura 18 se muestra cuál sería el flujo correcto de la inicialización de la cámara. Primeramente, rellenando los parámetros requeridos y confirmándolos, pulsando el botón de `Configure`, todo ello conectado a la red Wi-Fi del ESP32-CAM.

Cabe recalcar que si en algún momento hay algún error o los datos introducidos por el usuario no son correctos, se muestran mensajes de error (ver Figura 19).

Si se completa este proceso con éxito, la cámara se conectará automáticamente a la red Wi-Fi y al *broker* MQTT, como ya se ha indicado en secciones anteriores.

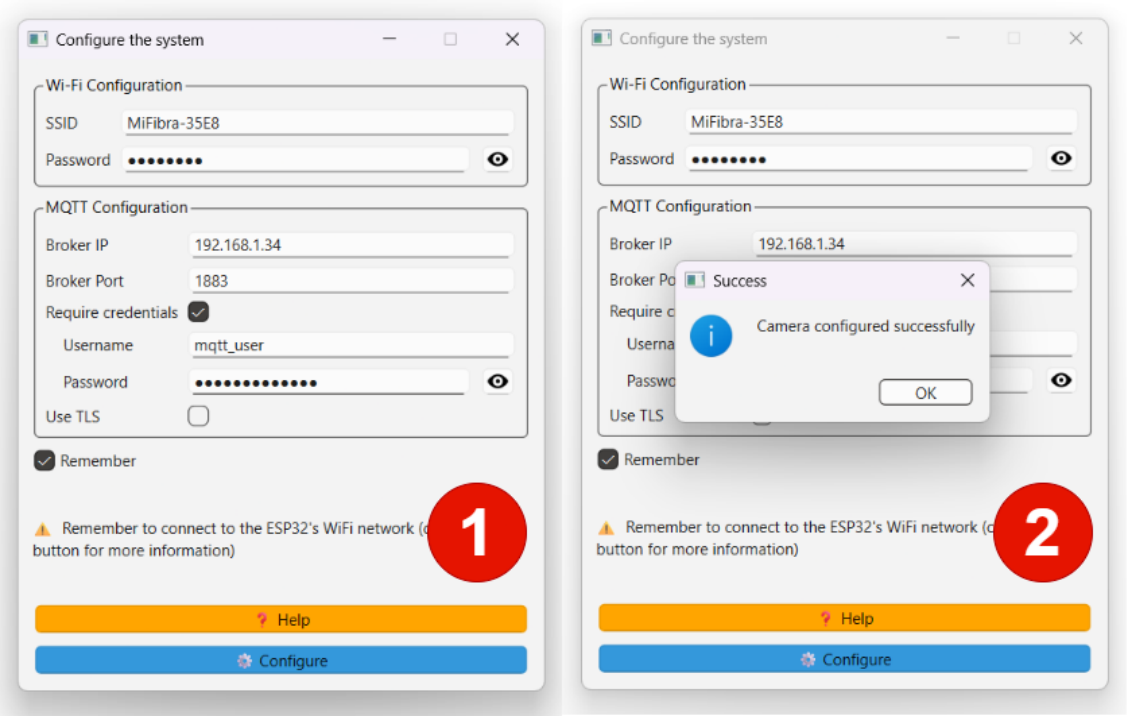

Figura 18: Proceso de configuración del programa inicializador del ESP32-CAM

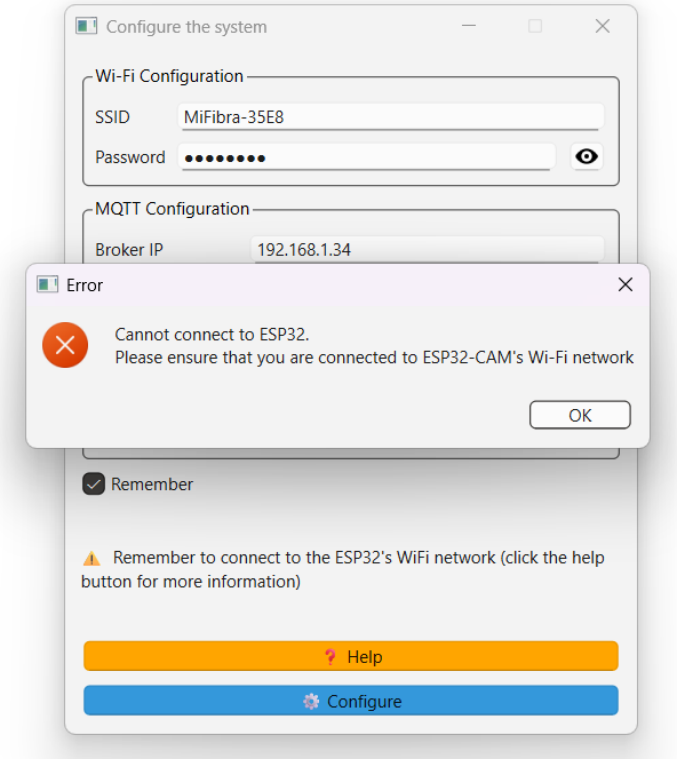

Figura 19: Ventana de error del programa inicializador del ESP32-CAM

4.2.4. Envío de parámetros

Una de las partes críticas de este programa es el envío de los parámetros a la cámara. Para enviar dichos parámetros se utiliza la librería `requests`[42] para enviar un JSON usando el método POST al siguiente *endpoint*:

`http://192.168.4.1/configure`

Cabe recalcar que este *endpoint* solo está disponible si el usuario se conecta previamente a la red Wi-Fi del microcontrolador.

El programa envía al microcontrolador un mensaje JSON con la siguiente estructura:

```
{
    "wifi_ssid": "nombre_red",
    "wifi_password": "contraseña_red",
    "mqtt_broker_address": "192.168.1.100",
    "mqtt_port": 1883,
    "mqtt_user": "usuario",
    "mqtt_pass": "contraseña",
    "tls_certificate": "certificado"
}
```

Una vez enviado este JSON mediante HTTP, el microcontrolador procesa los datos y realiza los pasos detallados en la sección 4.1.1.

4.3. Programa principal

Una vez desarrollado el programa encargado de inicializar las cámaras, fue necesario implementar una segunda aplicación que permitiera visualizar en tiempo real las cámaras activas y configurar los sensores según los requisitos específicos del entorno. Por lo que se ha desarrollado un programa principal que integra tanto la visualización en vivo como las herramientas de configuración de sensores, ofreciendo una interfaz centralizada para el control y supervisión del sistema completo. Además, permite configurar el servicio complementario de notificaciones de Telegram si está en uso.

4.3.1. Lenguaje y librerías

En el desarrollo del programa principal se ha utilizado Python 3.13.3[39] para seguir la misma línea que en el programa inicializador de ESP32-CAM. También se ha hecho uso del *framework* Qt6[40] para la interfaz gráfica junto a las siguientes bibliotecas externas de Python:

- `PyQt6 6.9.0`[41]: Permite crear interfaces gráficas usando Qt6 y Python.
- `Paho MQTT 2.1.0`[43]: Permite conectarse utilizando el protocolo de comunicación MQTT.
- `OpenCV 4.11.0`[44]: Permite realizar procesamiento de imágenes y tareas de visión artificial.
- `Numpy 2.2.5`[45]: Biblioteca requerida por `OpenCV` para su funcionamiento. Permite realizar operaciones matemáticas complejas (en este caso tratar con imágenes).

Además, también se ha utilizado el editor de código Visual Studio Code[46] para el desarrollo.

4.3.2. Estructura del código

A continuación, se muestra la organización del código utilizada siguiendo la arquitectura definida por el patrón Modelo-Vista-Controlador (MVC):

- `Main.py`: Archivo principal que permite ejecutar el programa con todos sus componentes.
- `config/`: Carpeta que guarda archivos JSON con los valores del formulario enviado para conectarse con MQTT.
- `Controller/`: Carpeta que contiene el controlador del programa.
    - `Controller.py`: Clase que gestiona la lógica entre el modelo y la vista del programa.

- `Model/`: Carpeta que contiene los archivos relacionados con el modelo del programa.
  - `ESP32Cam.py`: Clase que representa el estado actual de una ESP32CAM con sus configuraciones y últimas imágenes enviadas.
  - `MQTTClient.py`: Clase cliente MQTT que hereda de `MQTTSubject` y que permite obtener los mensajes enviados por el *broker* y reenviarlos a las clases que implementen la clase `MQTTObserver`.
  - `MQTTObserver.py`: Clase abstracta[48] que permite obtener los datos que le envía la clase `MQTTSubject` usando el método abstracto `update()`.
  - `MQTTSubject.py`: Clase abstracta que permite notificar y enviar datos a las clases `MQTTObserver`.
  - `Resolutions.py`: Contiene clases *enum*[49] relacionadas con las resoluciones soportadas por ESP32CAM.
  - `VideoRecorder.py`: Clase que permite crear videos a partir de las imágenes que se obtienen del microcontrolador.
- `Utils/`: Carpeta que contiene los archivos de utilidades.
  - `config.py`: Contiene las funciones para guardar o recuperar el formulario enviado en caso de que se quiera reutilizar.
  - `dialogs.py`: Contiene funciones para mostrar *popups* de errores o información en cualquier punto del programa.
  - `files.py`: Contiene funciones para guardar archivos en el sistema.
  - `validator.py`: Contiene funciones para validar las entradas del usuario en los formularios.
- `View/`: Carpeta que contiene los archivos relacionados con la vista del programa.
  - `CameraGrid.py`: Clase encargada de mostrar la cuadrícula de las imágenes en tiempo real de las cámaras activas.
  - `CameraSettingsPanel.py`: Clase encargada de mostrar el panel lateral que permite configurar los sensores de la cámara.
  - `CameraViewer.py`: Clase encargada de mostrar el identificador de la cámara junto a un `FrameView`. Se utiliza como elemento dentro del `CameraGrid`.
  - `CaptureViewer.py`: Clase encargada de mostrar un `FrameView` correspondiente a una cámara en la pantalla de `ControlPanel`.
  - `ConfigView.py`: Clase correspondiente a la vista que permite conectarse al *broker* MQTT.
  - `ControlPanel.py`: Clase encargada de mostrar el panel lateral que permite configurar las notificaciones de cada cámara.

- `ControlView.py`: Clase correspondiente a la pantalla del panel de control.
- `FrameView.py`: Clase que permite mostrar imágenes y actualizarlas en tiempo real.
- `LoadingPopup.py`: Clase que muestra un *popup* de carga.
- `MainView.py`: Clase correspondiente a la pantalla principal del programa.
- `MQTTGroup.py`: Clase que muestra el formulario de la conexión con MQTT en la clase `ConfigView`.

4.3.3. Patrones de diseño

Durante el desarrollo del programa, se han utilizado varios patrones de diseño con el objetivo de mejorar la mantenibilidad, escalabilidad y organización del código. Concretamente, se han utilizado los patrones Modelo-Vista-Controlador (*Model-View-Controller*) [53] y Observador (*Observer*)[54].

Modelo-Vista-Controlador: El Modelo-Vista-Controlador define la estructura principal de la aplicación, permitiendo una clara separación de responsabilidades entre los distintos componentes. Los componentes principales son los siguientes:

- **Modelo:** Encapsula la lógica de negocio y el acceso a los datos. Es el responsable de gestionar el estado de la aplicación y notificar los cambios relevantes.
- **Vista:** Se encarga de la presentación de la información al usuario. Escucha los cambios del modelo y actualiza la interfaz cuando es necesario.
- **Controlador:** Actúa como intermediario entre la vista y el modelo. Por ejemplo, recibe las entradas del usuario, las procesa, y realiza las acciones necesarias sobre el modelo, actualizando posteriormente la vista.

A continuación, en la Figura 20 se muestra una interpretación gráfica de estos elementos:

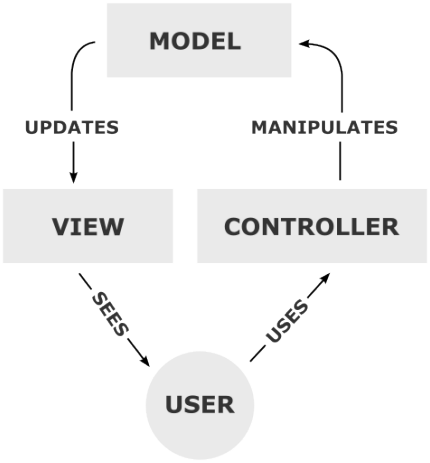

Figura 20: Diagrama del patrón de diseño Modelo-Vista-Controllador

Por lo tanto, este patrón se ha utilizado para estructurar de forma clara los diferentes archivos y clases que se han utilizado en el programa. En las secciones 4.3.2 y 4.3.4 se muestran en detalle los elementos finalmente utilizados usando este patrón.

Observador: El patrón Observador permite establecer una relación de dependencia entre múltiples objetos, de modo que, cuando uno de ellos cambia su estado, notifica automáticamente a todos los objetos dependientes para que se actualicen con la nueva información.

Este patrón está compuesto por varios elementos clave que permiten su funcionamiento:

- **Sujeto (*Subject*):** Es el objeto principal que contiene una lista de observadores y se encarga de notificarlos cuando sea necesario (normalmente con una función `notify()`).
- **Observador (*Observer*):** Es cualquier objeto que quiere recibir avisos del sujeto. Debe de tener un método (normalmente llamado `update()`) para actualizarse cuando hay cambios.

Para entender mejor su funcionamiento, en la Figura 21 se muestra un diagrama que contiene los elementos del patrón de diseño Observador.

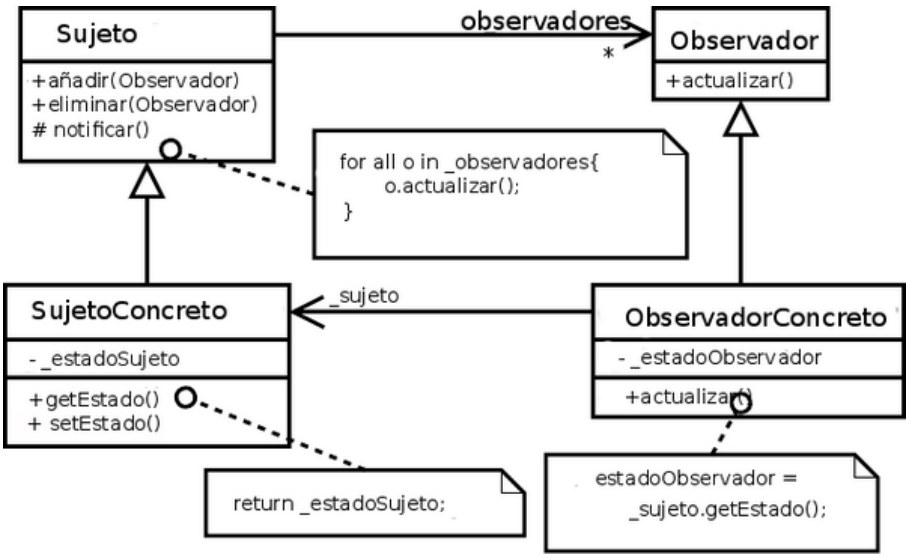

Figura 21: Diagrama del patrón de diseño Observador [54]

En esta aplicación se ha utilizado este patrón, ya que los objetos de la clase `ESP32Cam` necesitan actualizar constantemente el estado de su cámara física correspondiente (configuraciones e imágenes recibidas). Además, la interfaz gráfica (vista) debe mostrar errores o actualizarse con los datos que se reciben mediante MQTT cuando sea necesario.

Por lo tanto, para que el código quede limpio y organizado, la clase `MQTTClient` implementa `Subject` y las clases que implementan `Observer` son las clases `ESP32Cam` para las imágenes y `Controller` para actualizar la vista (ver Figura 22 de la siguiente sección).

4.3.4. Diagrama de clases

Con el fin de representar gráficamente la estructura del programa y ver de forma clara las relaciones entre los distintos componentes, se ha creado un diagrama de clases usando UML(*Unified Modeling Language*)[55]. En la Figura 22 se puede ver el diagrama de clases del modelo y el controlador; en la Figura 23 se presenta el diagrama correspondiente a la vista y el controlador.

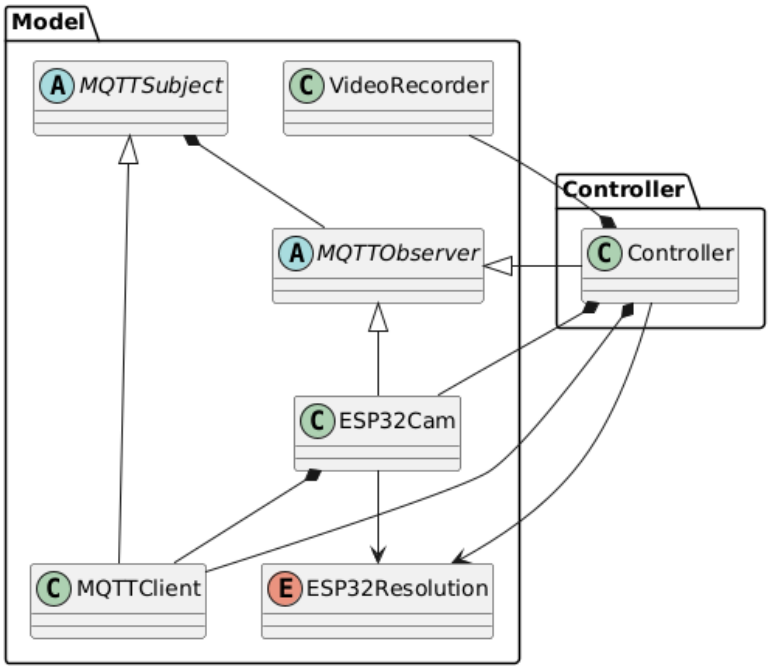

Figura 22: Diagrama de clases del modelo y el controlador

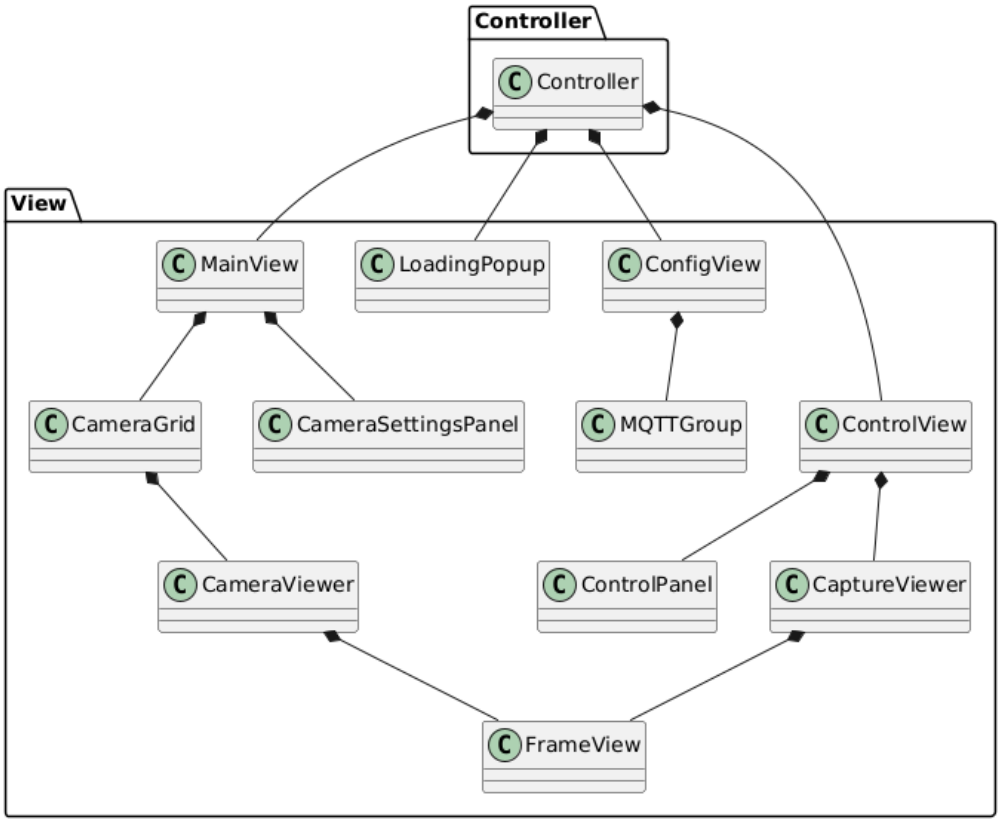

Figura 23: Diagrama de clases de la vista y el controlador

4.3.5. Funcionamiento del programa

A continuación se detalla cómo es el funcionamiento y flujo del programa principal interactuando con sus distintas ventanas.

Al ejecutar el programa, aparece una primera ventana de configuración (ver Figura 24), parecida al programa inicializador, que permite establecer los valores necesarios para que el programa se conecte al *broker* MQTT. Los parámetros que se definen son los mismos a los que se configuran desde el programa inicializador de ESP32-CAM: IP y puerto del *broker*, credenciales en caso de que sean necesarias y certificado TLS si se hace uso de cifrado TLS.

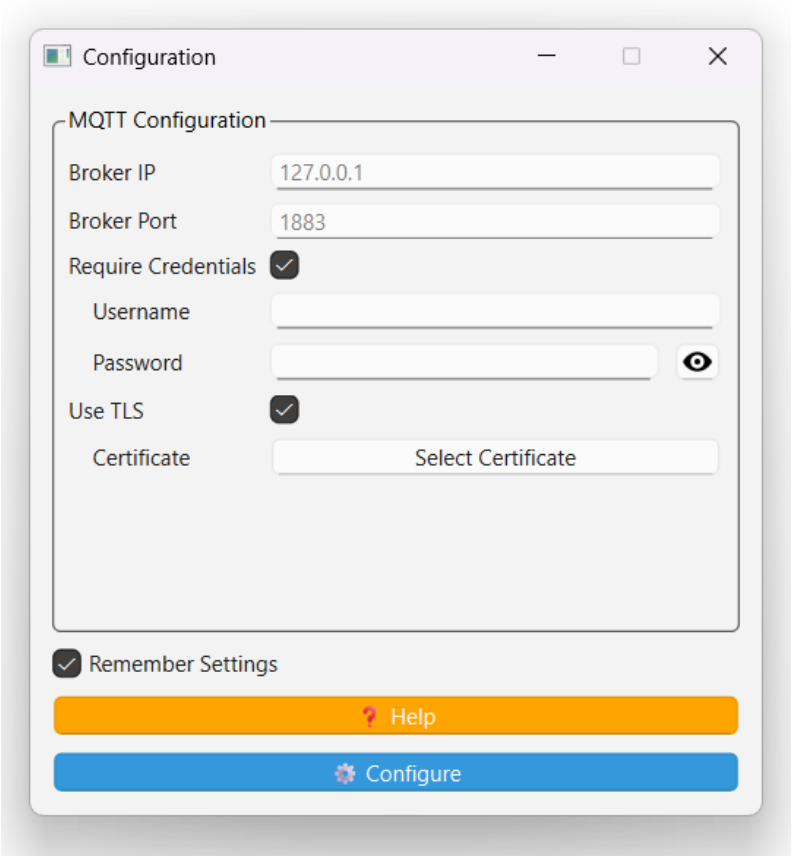

Figura 24: Ventana de configuración del programa principal

Esta ventana contiene también un botón de ayuda que explica en qué consiste este panel (ver Figura 25).

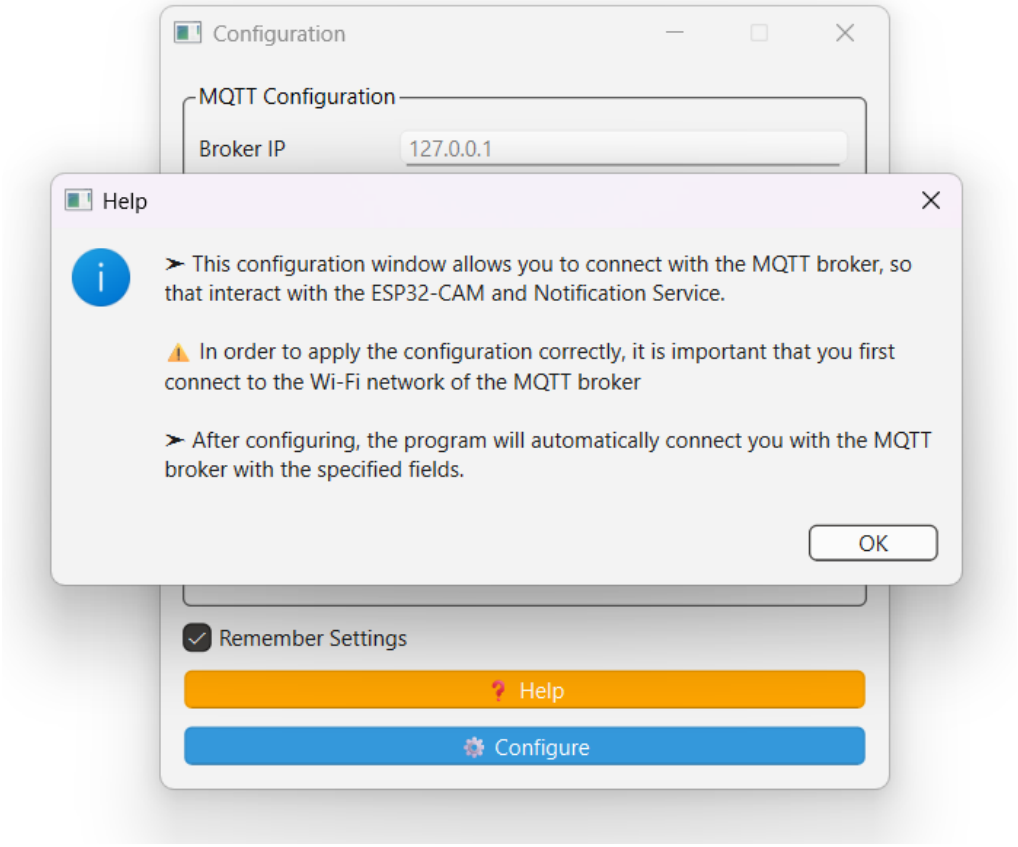

Figura 25: Ventana de ayuda de la configuración del programa principal

Por lo tanto, si se rellenan los campos correctamente estando conectados a la red Wi-Fi del *broker*, el programa se conecta automáticamente, muestra una ventana de confirmación (ver Figura 26) y seguidamente empieza a buscar cámaras conectadas y en funcionamiento. En caso de que haya algún problema con los parámetros, con la conexión o que no se encuentren cámaras disponibles, el programa muestra los errores con una ventana emergente (ver Figura 27).

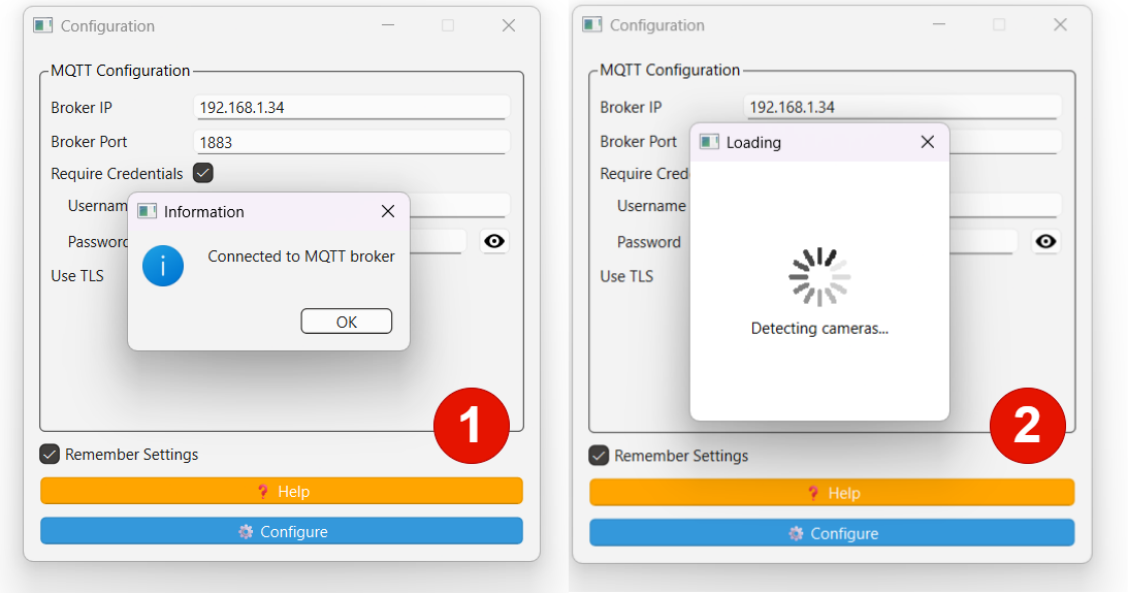

Figura 26: Proceso de la configuración del programa principal

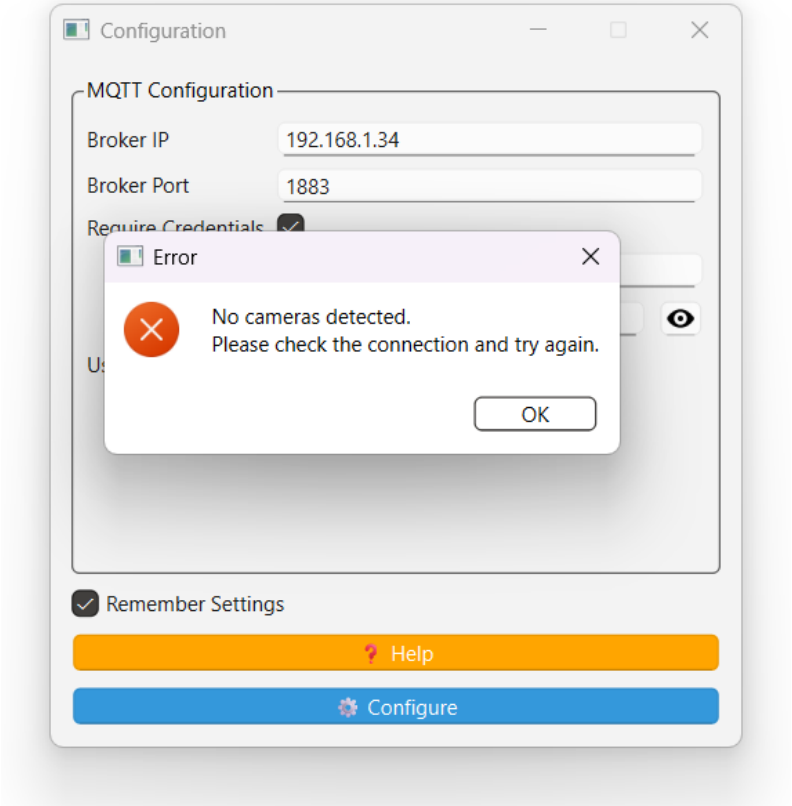

Figura 27: Ventana de error del programa principal

Una vez el programa se conecta correctamente y detecta las cámaras disponibles, muestra un *grid* con la visualización en tiempo real de dichas cámaras, junto a un panel lateral que permite configurar el sensor de cada una de ellas (ver Figura 28).

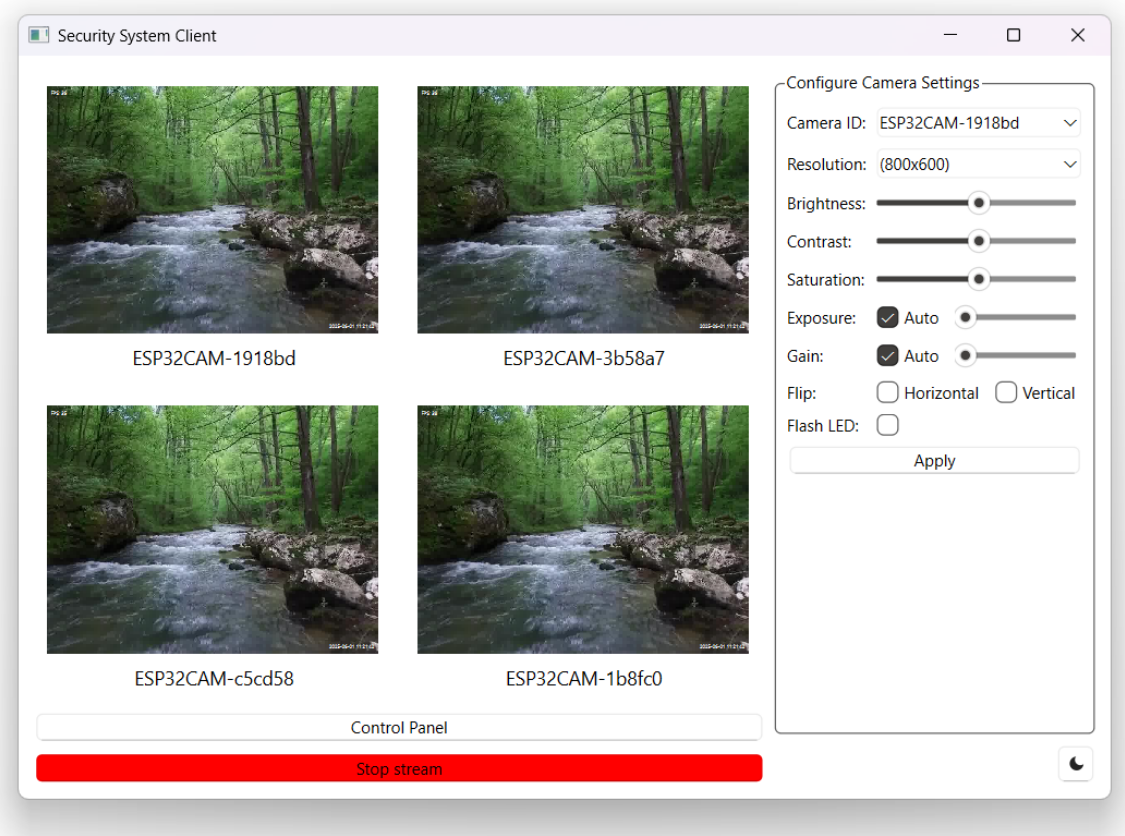

Figura 28: Ventana principal con cuatro cámaras simuladas del programa principal

El panel lateral de configuración permite cambiar parámetros del sensor como el brillo, el contraste, la saturación, la exposición y la ganancia. Además, permite voltear horizontalmente o verticalmente la imagen, así como encender o apagar el LED del microcontrolador.

Esta ventana también contiene un botón para poder parar o encender la transmisión en vivo, así como un botón para mostrar el *Control Panel* (ver Figura 29).

El *Control Panel* permite grabar o tomar imágenes de la cámara que se está visualizando; además, permite configurar el servicio de notificaciones cuando este está activo, añadiendo o borrando los diferentes chats (ver Figura 30). El funcionamiento de este servicio se detalla en la sección 4.4.

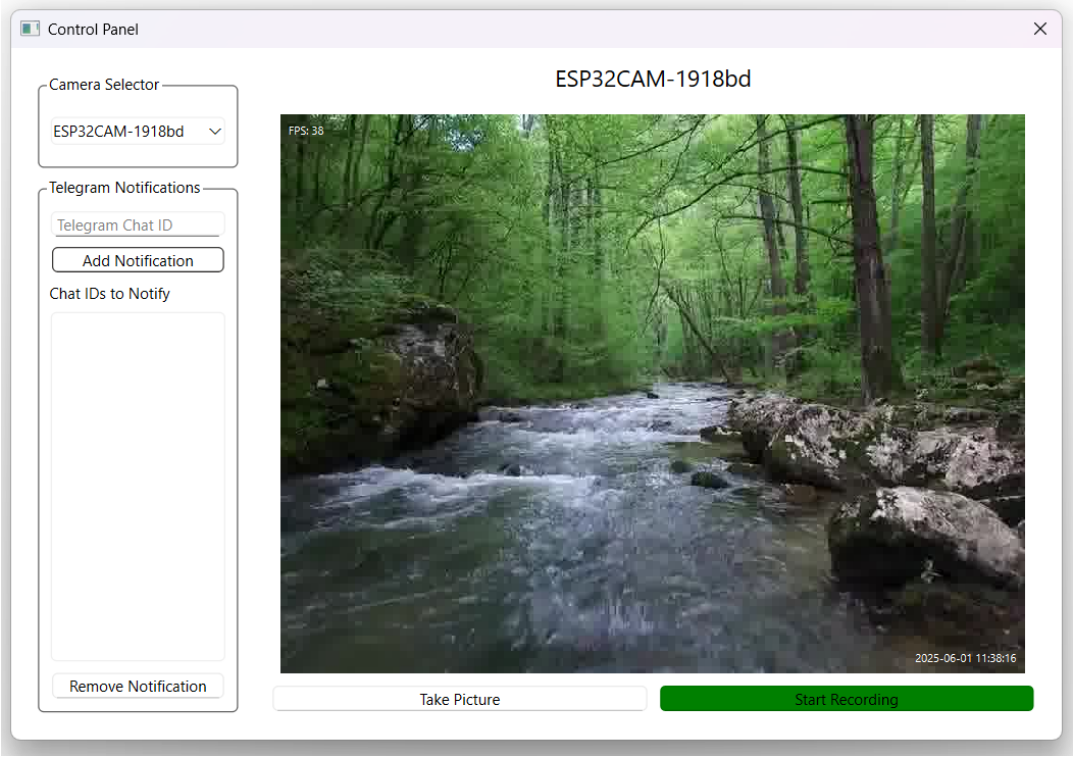

Figura 29: Ventana de control del programa principal

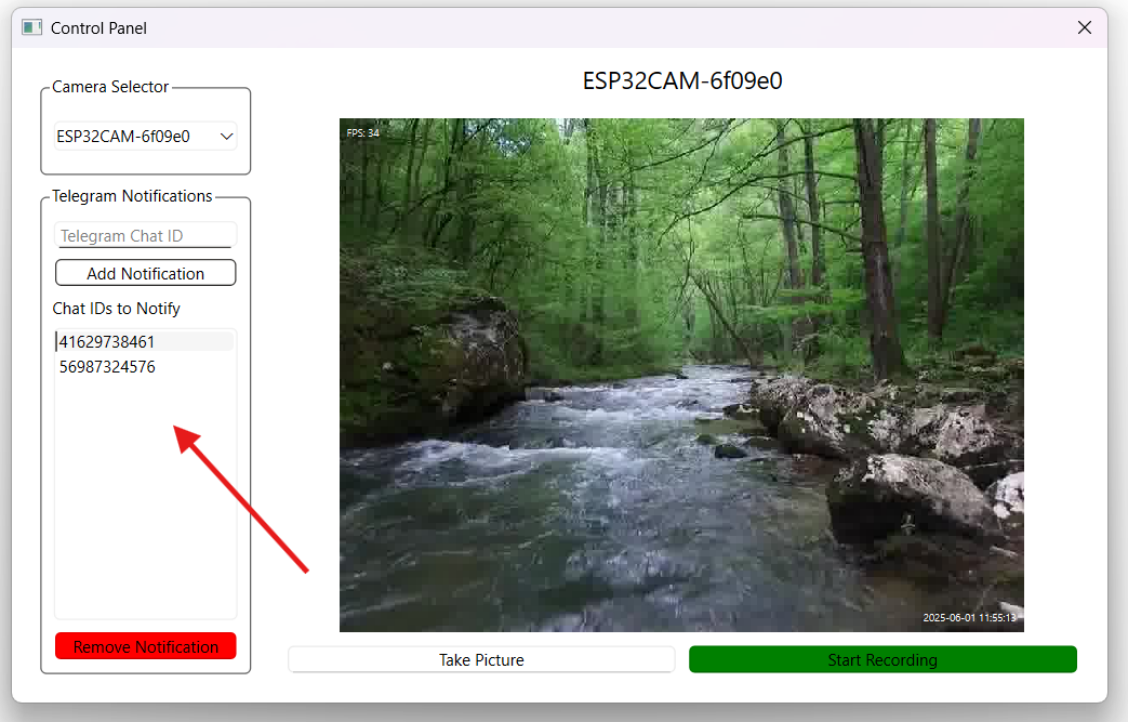

Figura 30: Ventana de control con la configuración de notificaciones

4.3.6. Grabación y captura de imágenes

Una de las partes críticas de la aplicación es el desarrollo de la funcionalidad de grabación y captura de imágenes, que permite almacenar de forma continua o puntual la información visual obtenida desde la cámara, observada desde la vista de control.

Para ello, se ha implementado un sistema de grabación basado en la captura periódica de fotogramas, que se gestiona mediante un temporizador que controla los cuadros por segundo (FPS). El uso de este temporizador es fundamental para garantizar la sincronización del vídeo, ya que la llegada de imágenes desde las cámaras se produce de forma asincrónica y no controlada. En consecuencia, podrían producirse pérdidas de fotogramas en determinados intervalos de tiempo, y no tener disponibles 30 imágenes en todos los segundos. Por lo tanto, para generar el vídeo, se obtiene la última imagen disponible en intervalos fijos de tiempo, asegurando que el vídeo resultante tenga una frecuencia de 30 FPS sin saltos ni inconsistencias.

Para crear el vídeo se ha utilizado la librería `opencv-python`[44] usando el módulo `VideoWriter` que permite generar un vídeo a partir de imágenes que se van añadiendo constantemente a su *writer*.

Una vez se pulsa el botón de empezar la grabación del vídeo, el programa pide un directorio para guardar el vídeo que se va a crear (ver Figura 31).

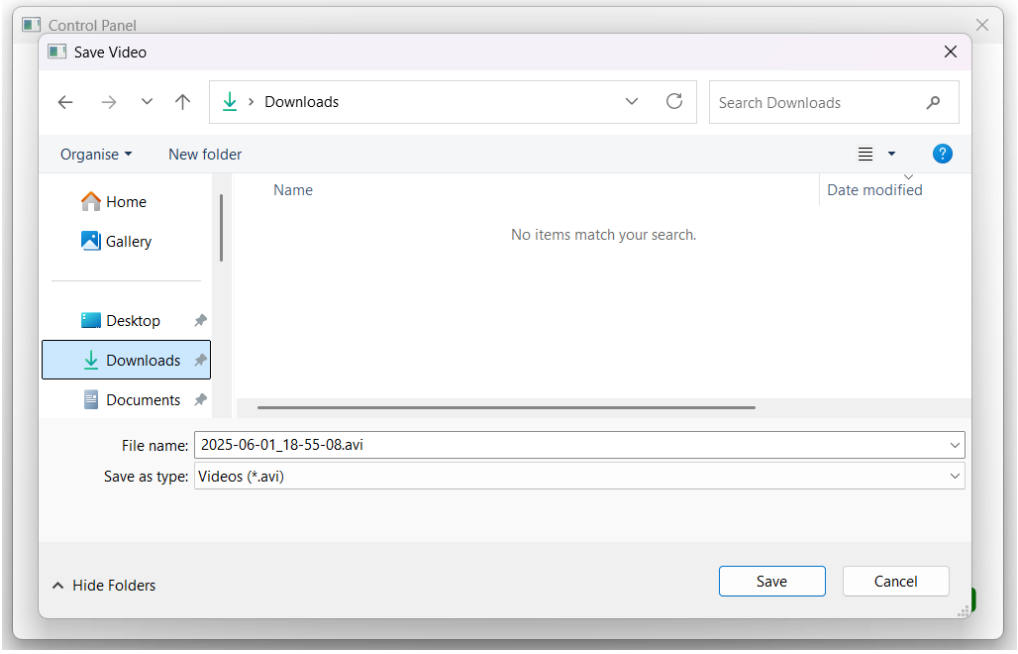

Figura 31: Ventana de selección de directorio para guardar el vídeo

Respecto a la captura de imagen, se obtiene la última imagen mostrada en el momento que se pulsa el botón de *Take Picture* y se pide al usuario un directorio para guardarla usando `opencv-python`[44], como se hace con el vídeo.

4.4. Servicio de notificaciones

Este programa o servicio es un complemento al programa principal. Este permite añadir notificaciones (con grabación) a diferentes chats de Telegram cuando se detecta movimiento en las cámaras configuradas. El algoritmo que detecta movimiento es bastante sencillo, pero da paso a que en un futuro se puedan añadir algoritmos más avanzados que permitan incluso el reconocimiento facial o de personas.

4.4.1. Lenguaje y librerías

Para el desarrollo de este módulo también se ha hecho uso de Python 3.13.3[39], ya que permite crear con facilidad pequeños scripts para terminal. Como este módulo no tiene interfaz gráfica, se ha hecho uso de la librería `argparse` que permite analizar y definir los parámetros pasados por la línea de comandos de la terminal.

A continuación se muestra un listado con las librerías externas de Python utilizadas para este servicio:

- `Paho MQTT 2.1.0`[43]: Permite conectarse utilizando el protocolo de comunicación MQTT.
- `OpenCV 4.11.0`[44]: Permite realizar procesamiento de imágenes y tareas de visión artificial. En este caso se ha utilizado para la grabación de imágenes.
- `Numpy 2.2.5`[45]: Biblioteca requerida por `OpenCV` para su funcionamiento. Permite realizar operaciones matemáticas complejas (como caso tratar con imágenes).
- `Requests 2.32.3`[42]: Permite realizar solicitudes HTTP de forma simple. En este caso se ha utilizado para interactuar con la API de Telegram.

Además, para este servicio, también se ha utilizado el patrón de diseño Observador detallado en la sección 4.3.3 para ir guardando las imágenes que se reciben de forma asíncrona en las distintas cámaras configuradas.

4.4.2. Funcionamiento del servicio

El servicio se activa a partir de la recepción de un chat ID y un ID de cámara, que permiten al sistema identificar a qué conversación de Telegram deben enviarse las notificaciones correspondientes. Una vez recibidos estos datos, el programa comprueba si la cámara está registrada en el sistema.

En caso de que no lo esté, la registra, crea un hilo de ejecución que se encarga de analizar continuamente las imágenes recibidas desde dicha cámara y añade el chat ID al listado de destinatarios asociados a ella.

Si la cámara ya está registrada, simplemente se añade el nuevo chat ID a su lista de chats asociados.

El hilo que procesa las imágenes compara continuamente las capturas recibidas para detectar variaciones significativas que puedan indicar movimiento. Si se detecta dicho movimiento, el sistema genera un video que incluye los cinco segundos anteriores y los cinco segundos posteriores al evento, y lo envía automáticamente por Telegram a todos los chats ID vinculados con esa cámara.

En la Figura 32 se muestra un diagrama de flujo que presenta de forma gráfica la lógica que sigue el servicio.

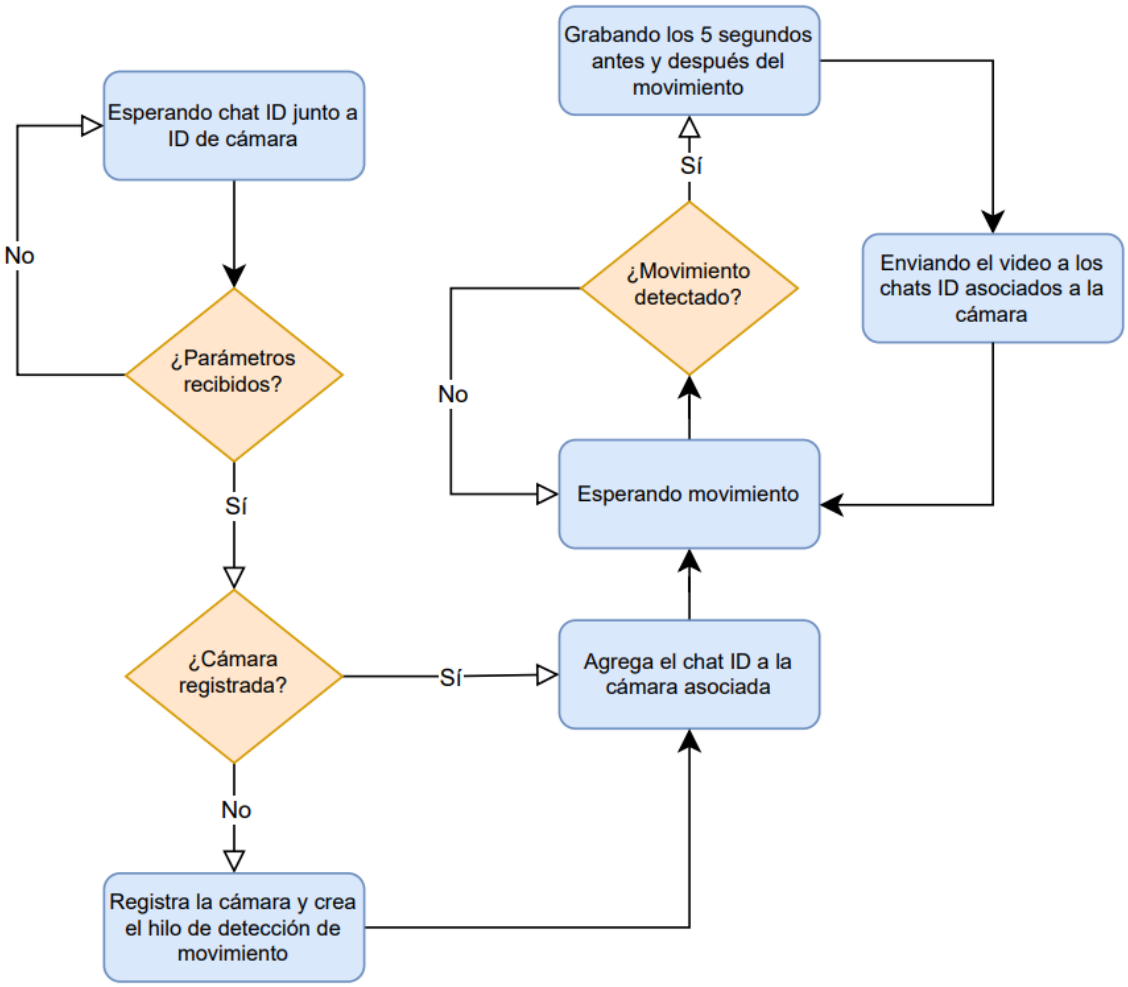

Figura 32: Diagrama de flujo del código del servicio de notificaciones

Para poder hacer funcionar este servicio, es necesario tener creado y configurado un bot de Telegram y hacer uso de su token correspondiente. Para la creación del bot se puede utilizar, por ejemplo, el bot llamado *BotFather*[50].

Por lo tanto, para ejecutar este servicio se tiene que hacer uso de estos argumentos de línea de comandos definidos con `argparse`:

- `--ip` (`-i`): Dirección IP del *broker* MQTT. Este argumento es obligatorio, ya que el servicio necesita conectarse a dicho *broker* para recibir los mensajes

con las imágenes.

- `--port` (`-p`): Puerto del *broker* MQTT. También es obligatorio y se especifica como un valor entero.
- `--token` (`-t`): Token del bot de Telegram. Este parámetro es necesario para que el servicio pueda autenticarse y enviar mensajes a través de la API de Telegram.
- `--username` (`-U`): Nombre de usuario para autenticarse en el *broker* MQTT. Este argumento es opcional, ya que depende de si el *broker* requiere credenciales.
- `--password` (`-P`): Contraseña correspondiente al nombre de usuario anterior. También es un argumento opcional.
- `--crt` (`-c`): Ruta al archivo de certificado TLS, en caso de que se utilice una conexión segura. Este parámetro es opcional.

Un ejemplo básico de ejecución desde terminal sin certificado TLS ni credenciales podría ser el siguiente:

```
python3 telegram_notifier.py -i 192.168.1.10 -p 1883 -t 412734174
```

4.4.3. Detección de movimiento

El proceso de detección de movimiento empieza seleccionando un conjunto de los 5 fotogramas más recientes de la cámara almacenados en memoria. Estas imágenes se convierten a escala de grises para eliminar la influencia del color en la detección. A continuación, se calcula la diferencia absoluta entre cada par de imágenes consecutivas, lo que permite resaltar las regiones donde ha habido cambios.

Las diferencias obtenidas se someten a un proceso de *binary thresholding* (ver Figura 33), que transforma las imágenes en representaciones de blanco y negro utilizando un umbral. Posteriormente, se aplica un algoritmo de detección de contornos (ver Figura 34) sobre estas imágenes binarias para identificar regiones donde hay cambio. Si se detecta algún contorno durante las comparaciones, se interpreta como que hay movimiento en la escena.

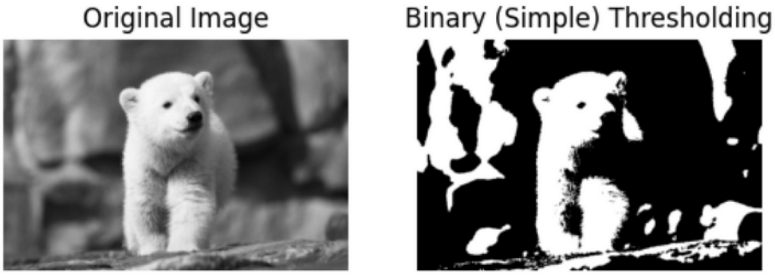

Figura 33: Ejemplo de binary thresholding en imágenes [51]

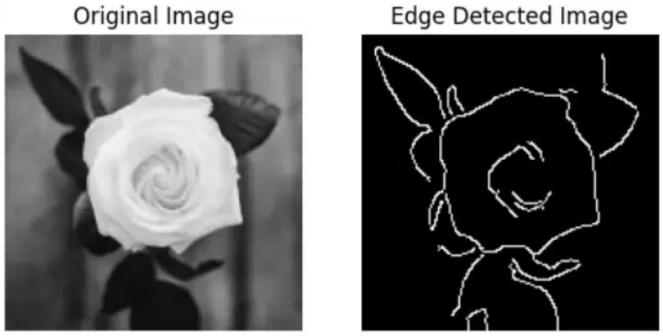

Figura 34: Ejemplo de detección de contornos en imágenes [52]

4.4.4. Envío de notificaciones de Telegram

El sistema envía notificaciones a través de la plataforma Telegram utilizando la API oficial de bots. Para ello, se establece una conexión HTTP con el *endpoint* correspondiente al método de envío de vídeo, utilizando un token del bot creado. El *endpoint* utilizado es el siguiente:

`https://api.telegram.org/bot<token>/sendVideo`

Cuando se detecta movimiento, el sistema prepara el mensaje que será enviado, junto con el archivo de vídeo generado de la misma forma que se hace en el programa principal (sección 4.3.6). Este contenido se envía mediante una petición POST, que incluye tanto los datos de texto (mensaje y chat ID destinatario) como el archivo multimedia.

En la Figura 35 se puede ver el mensaje que recibe el destinatario por Telegram cuando se detecta movimiento en una cámara. Los chats destinatarios se configuran desde la ventana de control del programa principal (ver Figura 30).

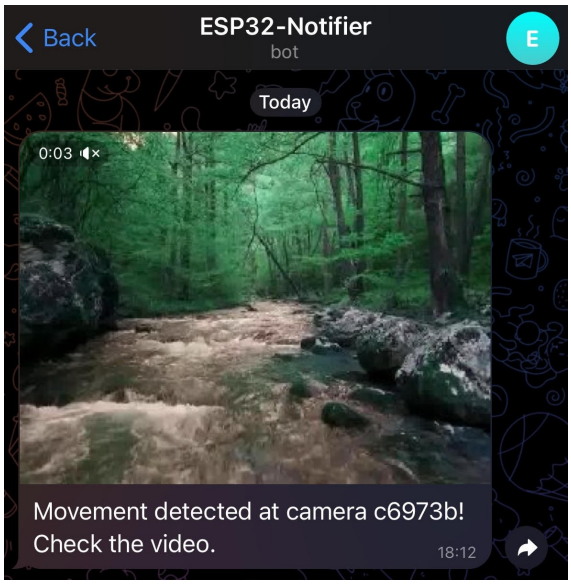

Figura 35: Recepción de notificación de movimiento de Telegram

4.5. Broker MQTT

Para la gestión de la comunicación entre las cámaras y el servidor utilizando el protocolo MQTT, es necesario un *broker*, es decir, un intermediario encargado de recibir y distribuir los mensajes entre los distintos clientes conectados.

En este proyecto se ha utilizado *Eclipse Mosquitto*[56], un *broker* MQTT ligero, de código abierto compatible con el estándar MQTT. Este *broker* ofrece soporte tanto para conexiones simples como para autenticación mediante credenciales y comunicación cifrada con TLS, lo que lo convierte en una solución versátil, adaptable a distintos niveles de seguridad y complejidad.

El servicio de MQTT *Mosquitto* se ha configurado y desplegado mediante Docker Compose[57], lo que permite definir las distintas variantes de configuración de forma modular.

4.5.1. Configuración simple

Para probar su forma más básica, el *broker* MQTT se ejecuta sin autenticación ni cifrado. Esta configuración es insegura, ya que permite a cualquier usuario interactuar con el *broker*.

El servicio se define con su archivo de configuración `mosquitto.conf` exponiendo el puerto 1883 y permitiendo conexiones anónimas:

```
listener 1883 0.0.0.0
allow_anonymous true
```

4.5.2. Configuración con credenciales

Con el objetivo de reforzar la seguridad del sistema, se puede habilitar un mecanismo de autenticación basado en usuario y contraseña. Esta medida garantiza que únicamente los clientes autorizados puedan establecer conexión con el *broker*, evitando accesos no deseados. Para activar esta funcionalidad, es necesario incluir un archivo de contraseñas y modificar la configuración del *broker* para que exija autenticación durante el proceso de conexión.

El archivo de configuración correspondiente se muestra a continuación:

```
listener 1883 0.0.0.0
allow_anonymous false
password_file /mosquitto/config/passwd
```

4.5.3. Configuración con TLS

Para garantizar la confidencialidad e integridad de los mensajes intercambiados entre los dispositivos y el servidor, se puede habilitar la comunicación cifrada mediante el protocolo TLS (Transport Layer Security). Esta capa de seguridad permite

proteger los datos transmitidos frente a posibles interceptaciones o manipulaciones por parte de terceros.

La configuración con TLS requiere proporcionar al *broker* los certificados necesarios para establecer conexiones seguras. En concreto, se deben incluir el certificado del servidor, su clave privada y el certificado de la autoridad certificadora (CA) utilizada. Estos archivos se incorporan al contenedor Docker mediante el `Dockerfile` y se referencian en el archivo de configuración `mosquitto.conf`.

Además, el *broker* debe estar configurado para escuchar en el puerto 8883, reservado para conexiones MQTT seguras. Una vez habilitada esta funcionalidad, solo los clientes que dispongan de los certificados adecuados podrán conectarse de forma cifrada, lo que refuerza significativamente la seguridad del sistema.

A continuación se muestra la configuración correspondiente:

```
listener 8883 0.0.0.0
allow_anonymous false
password_file /mosquitto/config/passwd
certfile /mosquitto/config/certs/mosquitto.crt
keyfile /mosquitto/config/certs/mosquitto.key
cafile /mosquitto/config/certs/mosquitto.crt
require_certificate false
```

4.6. Simulador de ESP32-CAM

Durante el desarrollo de este proyecto se ha utilizado un simulador de ESP32-CAM para evitar estar constantemente configurando las diferentes cámaras, así como para poder simular el uso paralelo de muchas de ellas. Esta herramienta ha resultado fundamental tanto para las fases de prueba como para la validación del sistema en condiciones controladas.

El simulador emula el comportamiento de una cámara real basada en ESP32-CAM, incluyendo el envío periódico de imágenes a través del protocolo MQTT utilizando la misma lógica vista en el apartado 4.1.1.

Cabe destacar que este simulador no permite configuraciones del sensor como sí ocurre con las cámaras reales, ya que su funcionamiento está basado en el envío continuo de fotogramas extraídos de un vídeo de ejemplo previamente cargado. Es decir, en lugar de capturar imágenes en tiempo real, el simulador reproduce un conjunto de imágenes, simulando así el comportamiento de una cámara que transmite vídeo en directo.

4.6.1. Lenguaje y librerías

Para desarrollar este simulador se ha hecho uso de Python 3.13.3[39] y la misma librería externa de MQTT que se ha utilizado en los otros módulos:

- `Paho MQTT 2.1.0`[43]: Permite conectarse utilizando el protocolo de comunicación MQTT.

Como este módulo no tiene interfaz gráfica, es decir, se ejecuta en línea de comandos, se ha hecho uso de la librería `argparse` que permite gestionar y definir los parámetros pasados por la terminal.

4.6.2. Funcionamiento del simulador

Para poder ejecutar el programa, es necesario proporcionar los parámetros descritos a continuación a través de la línea de comandos:

- `--ip` (`-i`): Dirección IP del *broker* MQTT al que se enviarán los fotogramas simulados. Este parámetro es obligatorio.
- `--port` (`-p`): Puerto del *broker* MQTT. También es obligatorio e indica el canal de comunicación a utilizar.
- `--number` (`-n`): Número de dispositivos ESP32-CAM a simular simultáneamente. Es un argumento obligatorio que permite generar hasta 12 ESP32-CAM simuladas.
- `--username` (`-U`): Nombre de usuario para la autenticación con el *broker*, en caso de que esté habilitada. Es opcional.

- `--password` (`-P`): Contraseña correspondiente al usuario anterior. También es opcional.
- `--crt` (`-c`): Ruta al certificado TLS para establecer una conexión segura con el *broker*, en caso de ser necesario. Este argumento es opcional.

Un ejemplo simple de ejecución sin credenciales ni certificado TLS sería el siguiente:

```
python3 simulator.py -i 192.168.1.10 -p 1883 -n 5
```

Cuando se indica un número de dispositivos con el parámetro `-n 5`, el programa simula de forma concurrente cinco instancias de cámaras ESP32-CAM. Cada una de estas instancias se comporta como un dispositivo independiente, con un identificador único y un *topic* propio de envío de imágenes dentro del *broker* MQTT.

Para lograr esta concurrencia, el simulador utiliza múltiples hilos de ejecución (*threads*), donde cada hilo se encarga de gestionar el envío periódico de fotogramas correspondientes. De esta manera, se consigue una simulación realista del comportamiento de múltiples cámaras funcionando en paralelo, permitiendo comprobar la estabilidad del sistema y la correcta gestión de datos en condiciones donde hay muchas cámaras simultáneas.

5. Resultados

En esta sección se realiza un repaso de los objetivos planteados al inicio del proyecto, con el fin de determinar si se han cumplido tras el proceso de desarrollo e implementación.

El primer objetivo consistía en desarrollar un sistema de transmisión de vídeo en tiempo real utilizando el protocolo MQTT y la red Wi-Fi, permitiendo la visualización remota de la cámara. Este objetivo se ha cumplido plenamente mediante el desarrollo de un firmware para el microcontrolador ESP32-CAM, que permite la captura de imágenes periódicas y su publicación a través de MQTT. Las imágenes pueden visualizarse desde la aplicación de escritorio, lo que permite un seguimiento remoto en tiempo real del entorno monitorizado.

En segundo lugar, se planteaba la implementación de funcionalidades de grabación de vídeo y captura de imágenes, tanto de forma manual como automática. Este objetivo también se ha alcanzado satisfactoriamente. Desde la aplicación de escritorio es posible activar manualmente la grabación o captura de imágenes, y también se ha implementado el servicio de notificaciones que activa la grabación automáticamente ante ciertos eventos, como la detección de movimiento. Los vídeos se almacenan localmente y pueden ser reproducidos posteriormente para su revisión.

El siguiente objetivo estaba enfocado en la detección automática de movimiento. Para ello, se ha desarrollado un algoritmo que compara los últimos fotogramas recibidos y detecta variaciones significativas que indican actividad en la escena. Esta funcionalidad ha sido integrada satisfactoriamente en el servicio de notificaciones.

Otro de los objetivos definidos era permitir la configuración personalizada de la cámara, ofreciendo ajustes como resolución, brillo y otros parámetros. El programa principal está preparado para enviar configuraciones a través de MQTT al microcontrolador ESP32-CAM. Se ha desarrollado un flujo de configuración que permite enviar parámetros personalizados a cada dispositivo durante su ejecución, cumpliendo con este objetivo.

También se ha integrado con éxito un sistema de notificaciones en tiempo real mediante un bot de Telegram. Cuando se detecta movimiento, el sistema de notificaciones genera un vídeo y lo envía automáticamente a todos los usuarios suscritos. Esta funcionalidad ha sido verificada mediante pruebas que confirman la recepción de alertas en tiempo real.

En lo que respecta a la seguridad y privacidad, uno de los objetivos clave era evitar el uso de servidores en la nube y gestionar todos los datos de forma local. Este principio se ha cumplido, ya que tanto el procesamiento como el envío de imágenes se realizan en la red local. Además, se ha configurado un *broker* MQTT, con soporte para autenticación y cifrado TLS, para garantizar la confidencialidad e integridad de las comunicaciones.

Por último, se estableció como objetivo la validación del software desarrollado mediante pruebas en distintos entornos. Para ello, se ha creado un simulador de ESP32-CAM que permite emular múltiples dispositivos enviando imágenes a través

de MQTT de forma concurrente. Gracias a esta herramienta, ha sido posible realizar pruebas de carga y comprobar el comportamiento del sistema ante múltiples flujos de datos. Además, se han llevado a cabo pruebas funcionales con cámaras físicas, confirmando la estabilidad y eficiencia del sistema en condiciones reales.

En conjunto, se concluye que el sistema desarrollado cumple con los objetivos establecidos, y que el sistema desarrollado ofrece una base sólida para su posible evolución hacia una solución aún más completa.

6. Cronograma y costes

6.1. Cronograma

El desarrollo del proyecto se ha llevado a cabo a lo largo de 24 semanas, siguiendo una planificación distribuida en distintas fases. A continuación, en la Figura 36, se detalla el diagrama de Gantt dividido por las distintas etapas y su duración aproximada:

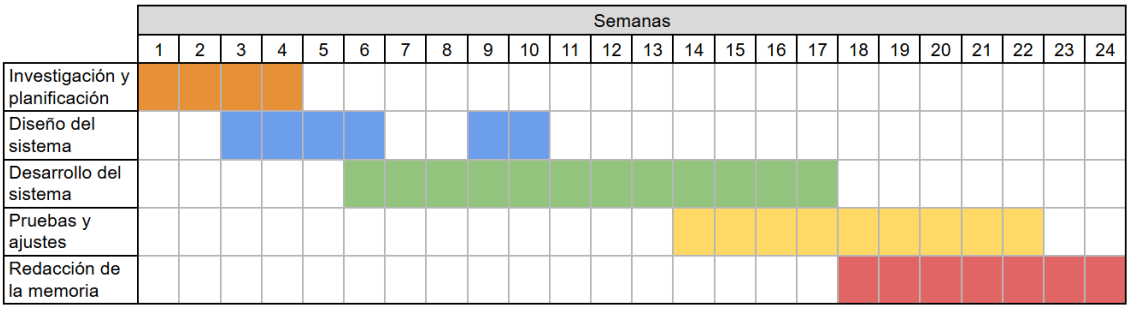

Figura 36: Diagrama de Gantt sobre la planificación del proyecto

Una vez definido el cronograma, a continuación se describen de forma resumida las principales etapas, destacando los objetivos y tareas realizadas:

1. **Investigación y planificación:** En esta etapa se llevó a cabo el análisis del estado del arte, la definición de los objetivos del proyecto y la selección de las tecnologías a utilizar.

2. **Diseño del sistema:** Se diseñó la estructura del sistema, definiendo los distintos componentes (cámaras, servidor, interfaz) y el flujo de datos entre ellos.

3. **Desarrollo del sistema:** En esta fase se implementaron todos los módulos del sistema, incluyendo el firmware del ESP32-CAM, la aplicación de escritorio, el sistema de notificaciones y el simulador de cámaras.

4. **Pruebas y ajustes:** Se realizaron pruebas de las aplicaciones, para verificar el correcto funcionamiento de cada componente. También se validó el sistema en conjunto, realizando ajustes para solucionar errores en el código y *bugs*.

5. **Redacción de la memoria:** En esta última etapa se redactó la memoria del trabajo describiendo todo su desarrollo.

6.2. Costes

Este proyecto ha sido desarrollado con una inversión económica mínima en materiales, complementada por una significativa dedicación en tiempo de trabajo. A nivel de hardware, únicamente ha sido necesaria la adquisición de dos módulos ESP32-CAM, mientras que todas las herramientas software utilizadas son de código abierto y gratuitas. El desarrollo del sistema ha supuesto una inversión estimada de 250 horas de trabajo, lo cual representa el principal coste del proyecto.

En la Tabla 4 se detallan los costes estimados de este proyecto, suponiendo que las horas de trabajo se pagan a 10 €/h:

Recurso	**Coste estimado**
2x Módulos ESP32-CAM	13,50 €
Software y librerías (libre)	0,00 €
Hardware (PC personal reutilizado)	0,00 €
Tiempo de desarrollo (250 h × 10 €/h)	2500,00 €
Total estimado	**2513,50 €**

Tabla 4: Coste total estimado del proyecto

Esta estimación refleja el valor real del proyecto si se considerara como un desarrollo técnico profesional. La mayor parte del coste corresponde al tiempo de dedicación, lo que pone en valor la complejidad técnica y el esfuerzo requerido para implementar un sistema completo, funcional y escalable desde cero.

7. Conclusiones y trabajo futuro

7.1. Conclusiones

Este Trabajo de Fin de Grado ha demostrado la viabilidad de desarrollar un sistema de videovigilancia distribuido utilizando microcontroladores ESP32-CAM y tecnologías de código abierto, con un enfoque centrado en la eficiencia, escalabilidad y respeto por la privacidad del usuario.

A lo largo del proyecto se han cumplido los objetivos establecidos inicialmente. Se ha implementado un sistema capaz de recibir imágenes desde múltiples cámaras, procesarlas localmente para detectar movimiento, generar grabaciones relevantes y enviar notificaciones automáticas mediante Telegram. La arquitectura diseñada permite escalar el número de dispositivos gracias al uso del protocolo MQTT y a un tratamiento asíncrono de los datos mediante hilos.

Además, se ha desarrollado un simulador de cámaras que ha permitido realizar pruebas exhaustivas sin necesidad de múltiples dispositivos físicos, lo que ha facilitado la validación del sistema en entornos de desarrollo. También se ha configurado un *broker* MQTT con soporte para autenticación y cifrado TLS, aumentando la seguridad de las comunicaciones.

En resumen, se ha logrado diseñar e implementar una solución funcional, económica y adaptable que puede ser utilizada como base para sistemas de videovigilancia personalizados, especialmente en entornos domésticos o educativos donde se priorice el control local de los datos y la facilidad de despliegue.

7.2. Trabajo a futuro

Aunque el sistema desarrollado cumple con los objetivos establecidos y ha demostrado ser funcional y escalable, existen múltiples líneas de trabajo futuro que podrían mejorar sus capacidades.

Uno de los módulos con mayor potencial de mejora es el de notificaciones y la detección de eventos. Actualmente, el sistema utiliza una técnica basada en diferencias entre fotogramas consecutivos para detectar movimiento, lo cual es efectivo, pero susceptible a falsos positivos causados por cambios de iluminación drásticos. En este sentido, podría integrarse un sistema de análisis más avanzado basado en algoritmos de visión por computador o modelos de inteligencia artificial, capaces de distinguir entre distintos tipos de eventos (personas, animales, objetos) e incluso realizar seguimiento de movimiento o reconocimiento de patrones.

También sería interesante desarrollar un módulo de almacenamiento local que permita registrar los eventos detectados en una base de datos. Esto permitiría al usuario consultar el historial de grabaciones y obtenerlas en caso de que sea necesario.

Referencias

[1] Telegram – a new era of messaging. (s. f.). *Telegram*. Recuperado de https://telegram.org/

[2] IBM. (2024, 2 de mayo). *Internet de las cosas*. International Business Machines. Recuperado de https://www.ibm.com/es-es/topics/internet-of-things

[3] Grand View Research: *Video surveillance market size to reach $147.66Bn by 2030*. (s. f.). Recuperado de https://www.grandviewresearch.com/press-release/global-video-surveillance-market

[4] Colaboradores de Wikipedia. (2025b, mayo 21). *Dirección IP*. Wikipedia, la Enciclopedia Libre. Recuperado de https://es.wikipedia.org/wiki/Direcci%C3%B3n_IP

[5] Amazon Web Services, Inc. (s. f.). *AWS — Cloud Computing - Servicios de informática en la nube*. Recuperado de https://aws.amazon.com/es/

[6] Google Cloud. (s. f.). *Servicios de cloud computing — Google Cloud*. Recuperado de https://cloud.google.com/?hl=es

[7] Microsoft Azure. (s. f.). *Servicios en la nube — Microsoft Azure*. Recuperado de https://azure.microsoft.com/es-es

[8] Amazon Web Services, Inc. (s. f.). *AWS — Almacenamiento de datos seguro en la nube (S3)*. Recuperado de https://aws.amazon.com/es/s3/

[9] Google Workspace. (s. f.). *Google Drive: comparte archivos online con almacenamiento seguro en la nube — Google Workspace*. Recuperado de https://workspace.google.com/intl/es/products/drive/

[10] Pascual, C. (2022, 19 de enero). *ESP32 CAM introducción y primeros pasos*. Programarfacil Arduino y Home Assistant. Recuperado de https://programarfacil.com/esp32/esp32-cam/

[11] Espressif Systems. (s. f.). *ESP Modules — Espressif Systems*. Recuperado de https://www.espressif.com/en/products/modules

[12] Espressif Systems. (s. f.). *Wireless SoCs, Software, Cloud and AIoT Solutions — Espressif Systems*. Recuperado de https://www.espressif.com/en

[13] Tensilica, Inc. (2004). *Xtensa LX Microprocessor Overview Handbook*. Recuperado de https://loboris.eu/ESP32/Xtensa_lx%20Overview%20handbook.pdf

[14] Xiaomi España. (s. f.). *Xiaomi® España — Comprar Móviles en Tienda Online Oficial*.
Recuperado de https://www.mi.com/es/

[15] TP-Link. (s. f.). *TP-Link España - Equipos de red Wi-Fi para el hogar y para empresas*.
Recuperado de https://www.tp-link.com/es/

[16] EZVIZ. (s. f.). *EZVIZ España - Soluciones de seguridad inteligente para el hogar*.
Recuperado de https://www.ezviz.com/es

[17] Home Assistant. (s. f.). *Home Assistant*. Home Assistant.
Recuperado de https://www.home-assistant.io/

[18] Node-RED. (s. f.). *Node-RED: Programación de bajo código para aplicaciones orientadas a eventos*.
Recuperado de https://nodered.org/

[19] Raspberry Pi. (s. f.). *Raspberry Pi: Ordenadores de placa única de bajo costo*.
Recuperado de https://www.raspberrypi.com/

[20] Wikipedia contributors. (2025, 15 mayo). *Transport layer security*. Wikipedia.
Recuperado de https://en.wikipedia.org/wiki/Transport_Layer_Security

[21] Wikipedia contributors. (2024, 9 octubre). *Sniffing attack*. Wikipedia.
Recuperado de https://en.wikipedia.org/wiki/Sniffing_attack

[22] MQTT - The Standard for IoT Messaging. (s. f.). Recuperado de https://mqtt.org/

[23] HTTP — MDN. (2025, 27 marzo). *MDN Web Docs*.
Recuperado de https://developer.mozilla.org/es/docs/Web/HTTP

[24] Colaboradores de Wikipedia. (2025, 8 enero). *COAP*. Wikipedia, la Enciclopedia Libre.
Recuperado de https://es.wikipedia.org/wiki/CoAP

[25] «Manuals». (2025, 29 abril). *Docker Documentation*. Recuperado de https://docs.docker.com/manuals/

[26] Knolleary. (s. f.). *GitHub - knolleary/pubsubclient: A client library for the Arduino Ethernet Shield that provides support for MQTT*. GitHub. Recuperado de https://github.com/knolleary/pubsubclient

[27] BenoitBlanchon. (s. f.). *Documentation*. ArduinoJson. Recuperado de https://arduinojson.org/v7/

[28] Dvarrel. (s. f.). *GitHub - dvarrel/ESPAsyncWebSrv: Async Web Server for ESP8266 and ESP32*. GitHub. Recuperado de https://github.com/dvarrel/ESPAsyncWebSrv

[29] Dvarrel. (s. f.-a). *GitHub - dvarrel/AsyncTCP: Async TCP Library for ESP32*. GitHub. Recuperado de https://github.com/dvarrel/AsyncTCP

[30] Dvarrel. (s. f.-b). *GitHub - dvarrel/ESPAsyncTCP: Async TCP Library for ESP8266*. GitHub. Recuperado de https://github.com/dvarrel/ESPAsyncTCP

[31] Colaboradores de Wikipedia. (2024, 19 septiembre). *Zigbee*. Wikipedia, la Enciclopedia Libre. Recuperado de https://es.wikipedia.org/wiki/Zigbee

[32] ¿Qué es un punto de acceso? (2021, 22 septiembre). *Cisco*. Recuperado de https://www.cisco.com/c/es_mx/solutions/small-business/resource-center/networking/what-is-access-point.html

[33] ¿Qué es un SSID de Wi-Fi? (2025, 28 mayo). *Kaspersky*. Recuperado de https://latam.kaspersky.com/resource-center/definitions/what-is-an-ssid

[34] ¿Qué es un certificado SSL? - Explicación del certificado SSL/TLS - AWS. (s. f.). *Amazon Web Services, Inc.* Recuperado de https://aws.amazon.com/es/what-is/ssl-certificate/

[35] Arduino. (s. f.). *Getting started with the Arduino IDE 2.0*. Recuperado de https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/

[36] Espressif. (s. f.). *GitHub - espressif/arduino-esp32: Arduino core for the ESP32*. GitHub. Recuperado de https://github.com/espressif/arduino-esp32

[37] Colaboradores de Wikipedia. (2025b, mayo 14). *JSON*. Wikipedia, la Enciclopedia Libre. Recuperado de https://es.wikipedia.org/wiki/JSON

[38] Colaboradores de Wikipedia. (2025b, abril 25). *Joint Photographic Experts Group*. Wikipedia, la Enciclopedia Libre. Recuperado de https://es.wikipedia.org/wiki/Joint_Photographic_Experts_Group

[39] Python Release Python 3.13.3. (s. f.). *Python.org*. Recuperado de https://www.python.org/downloads/release/python-3133/

[40] QT 6.9. (s. f.). Recuperado de https://doc.qt.io/qt-6/

[41] PyQT6. (2025, 8 abril). *PyPI*. Recuperado de https://pypi.org/project/PyQt6/

[42] Requests. (2024, 29 mayo). *PyPI*. Recuperado de https://pypi.org/project/requests/

[43] Paho-mqtt. (2024, 29 abril). *PyPI*. Recuperado de https://pypi.org/project/paho-mqtt/

[44] Opencv-python. (2025, 16 enero). *PyPI*. Recuperado de https://pypi.org/project/opencv-python/

[45] Numpy. (2025, 17 mayo). *PyPI*. Recuperado de https://pypi.org/project/numpy/

[46] Visual Studio Code - Code editing. Redefined. (2021, 3 noviembre). Recuperado de https://code.visualstudio.com/

[47] Microsoft. (s. f.). Microsoft: IA, nube, productividad, informática, juegos y aplicaciones. Recuperado de https://www.microsoft.com/es-es

[48] iKenshu. (2024, 20 agosto). Qué es una clase abstracta en la programación orientada a objetos. Recuperado de https://platzi.com/blog/clases-abstractas/

[49] Enum — Support for enumerations. (s. f.). *Python Documentation*. Recuperado de https://docs.python.org/es/3/library/enum.html

[50] BotFather. (s. f.). *Telegram*. Recuperado de https://telegram.me/BotFather

[51] Mavuzer, D. C. (2023, 5 octubre). *Mastering thresholding Techniques - DevOps.dev*. Medium. Recuperado de https://blog.devops.dev/mastering-thresholding-techniques-96706ce36d07

[52] GeeksforGeeks. (2024, 22 julio). *What is Edge Detection in Image Processing?* GeeksforGeeks. Recuperado de https://www.geeksforgeeks.org/what-is-edge-detection-in-image-processing/

[53] Colaboradores de Wikipedia. (2025b, marzo 7). Modelo–Vista–Controlador. *Wikipedia, la Enciclopedia Libre*. Recuperado de https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

[54] Colaboradores de Wikipedia. (2025c, marzo 12). Observer (patrón de diseño). *Wikipedia, la Enciclopedia Libre*. Recuperado de https://es.wikipedia.org/wiki/Observer_(patr%C3%B3n_de_dise%C3%B1o)

[55] What is UML — Unified Modeling Language. (s. f.). Recuperado de https://www.uml.org/what-is-uml.htm

[56] Eclipse Mosquitto. (2018, 8 enero). *Eclipse Mosquitto*. Recuperado de https://mosquitto.org/

[57] «Docker compose». (2025, 28 abril). *Docker Documentation*. Recuperado de https://docs.docker.com/compose/