Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Electricidad y Computación

Sistema IoT de vigilancia a través de la plataforma Telegram

Proyecto Integrador

Previo la obtención del Título de:

Ingeniero en Telecomunicaciones

Presentado por:

Paul Bryan Jiménez Villegas

Luis Alfredo Chávez Lalán

Guayaquil - Ecuador

Año: 2023

Dedicatoria

Este proyecto es dedicado en primer lugar a Dios, guía constante en mi camino académico. A mi amado padre y su esposa junto a mi madre cuyos sacrificios y amor incondicional han sido mi inspiración. A mi querida esposa por su apoyo inquebrantable. A mis abuelos que han sido pilares en mi vida. A mis hermanos, cuyas risas y complicidad han enriquecido mi vida. A mis respetados profesores gracias por su orientación y enseñanzas transformadoras. Este logro es el resultado del apoyo y amor de cada uno de ustedes.

Paul Jiménez

Dedicatoria

El presente proyecto lo dedico principalmente a Dios, por darme la fuerza necesaria para culminar esta meta. A mi querida madre, cuyo sacrificio, apoyo incondicional y amor infinito han sido mi mayor fortaleza. A mi abuelita, mi hermano y mi tía Rosa quienes me han apoyado y dado fuerzas en este camino. Agradezco a mi padre, a pesar de ya no estar, su legado de determinación y valores perdura en cada paso que doy. Este logro es también suyo. A mis profesores, por su orientación y enseñanzas a lo largo de este camino académico. ¡Gracias a todos por ser mi motivación y sostén inquebrantable!"

Luis Chávez

Agradecimientos

Mis agradecimientos a nuestros guías, Juan Carlos Avilés y María Antonieta Álvarez quienes de la mejor manera han sabido ayudarnos en la culminación de este proyecto. Así mismo a mi futuro colega de la carrera Luis Chávez quien desde un inicio fue un pilar y un amigo alentador para culminar este proyecto. Finalmente agradezco a mis amigos y familiares por sus buenos deseos y apoyo desde el inicio, sin ellos nada de esto habría sido posible.

Paúl Jiménez

Agradecimientos

Mis más sinceros agradecimientos a nuestros profesores, Juan Carlos Avilés y María Antonieta Álvarez por su paciencia y apoyo a lo largo del desarrollo de este proyecto. También quiero expresar mi agradecimiento a mi compañero de la carrera, Paúl Jiménez, por darme su amistad y apoyo en el transcurso de la carrera. Finalmente, agradezco a mis amigos y familiares quienes estuvieron ahí para ayudarme con sus conocimientos, por los consejos y la confianza que brindaron en mí. Sin ellos, nada de esto habría sucedido.

Luis Chávez

Declaración Expresa

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; Paul Bryan Jiménez Villegas y Luis Alfredo Chávez Lalán y damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Paul Bryan Jiménez Villegas

Luis Alfredo Chávez Lalán

Evaluadores

Firmado electrónicamente por:
MARIA ANTONIETA ALVAREZ VILLANUEVA

PhD. María Álvarez Villanueva

Profesor de Materia

Firmado electrónicamente por:
JUAN CARLOS AVILES CASTILLO

PhD. Juan Avilés Castillo

Tutor de proyecto

Resumen

La investigación actual se enfoca en el diseño y la ejecución de un sistema de seguridad para hogares utilizando la plataforma de mensajería Telegram. El objetivo de este proyecto es llevar a cabo la implementación de un sistema IoT de vigilancia de bajo costo y fácil instalación para residencias debido a la situación de peligro que está pasando el país. El sistema cuenta con dos módulos Esp32Cam, los cuales están en la configuración Maestro- Esclavo y utilizando el protocolo ESP-NOW para poder recibir y enviar datos de un módulo aotro, como seria la información de sensores de movimiento y magnético. Además, se integra una alarma sonora al sistema de vigilancia. En el presente trabajo se han integrado solo 2 cámaras con posibilidad de escalabilidad de más cámaras y sensores. Los datos de los módulos e información de los sensores son procesados en el módulo Maestro como mensajes de alerta al usuario, estos datos son enviados a través de la API de Telegram para hacer llegar las notificaciones y fotos al usuario. Cuando se detecte un movimiento fuera o dentro de la residencia, o que alguna de las dos puertas se abra, el sistema enviará una notificaciónde alerta al usuario a su cuenta de Telegram; así mismo el usuario tendrá la decisión de recibir una foto del entorno o interior de su residencia y si existe alguna persona intentando entrar a la residencia, se podrá, mediante comandos, activar una alarma sonora que ayudara a persuadir a la persona no deseada o alertar vecinos.

Palabras Clave: Seguridad, protocolo ESP-NOW, configuración Maestro-Esclavo, sensores, cámaras.

Abstract

This degree work focuses on the development and implementation of a residential security system using the Telegram messaging platform. The purpose of this project is to implement a low-cost, easy-to-install IoT surveillance system for residences because of the perilous conditions prevailing in the country. The system has two Esp32Cam modules, which are in the Master-Slave configuration and using the ESP-NOW protocol to be able to receive and send data from one module to another, such as information from the motion and magnetic sensors. In addition, an audible alarm is integrated into the surveillance system. In this part, only 2 cameras have been integrated with the possibility of scalability of more cameras and sensors. The data from the modules and information from the sensors are processed in the Master module as alert messages to the user. This data is sent through the Telegram API to send notifications and photos to the user. When a movement is detected outside or inside the residence, or one of the two doors is opened, the system will send an alert notification to the user to their Telegram account, likewise the user will have the decision to receive a photo of the environment or inside your residence and if there is a person hovering or wanting to enter the residence, you can use commands to activate an audible alarm that will help persuade the unwanted person.

Keywords: *Security, ESP-NOW protocol, Master-Slave configuration, sensors, cameras.*

Índice General

Abreviaturas

IoT	Internet of Things
Wi-Fi	Wireless Fidelity
ACK	Acknowledgment
API	Application Programming Interface
TCP	Transmission Control Protocol
IP	Internet Protocol
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
SMTP	Simple Mail Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SSL	Secure Sockets Layer
MAC	Media Access Control
TLS	Transport Layer Security
TX	Transmission
RX	Reception
GND	Ground
MAC	Media Access Control
IDE	Integrated Development Environment
PIR	Passive Infrared
GPIO	General Purpose Input/Output
Li-Ion	Lithium Ion
AC	Alternating Current
DC	Direct Current
USB	Universal Serial Bus
IO0	Input/Output zero
COM	Communication
TTL	Transistor-Transistor Logic

POST	Power On Self Test
CA	Certificate Authority
PSK	Pre-Shared Key
ID	Identifier
UXGA	Ultra Extended Graphics Array

Simbología

GHz	Gigahercios
Mbps	Megabits por Segundo
V	Voltio
m	Metro
mAh	Miliamperios-hora
ms	Milisegundo
dBm	Decibelio-Milivatio
mA	Miliamperio
h	Hora

Índice de Figuras

Índice de Tablas

Capítulo 1

1. Introducción

En la situación actual que está atravesando el país, la seguridad se ha vuelto un tema de preocupación para las familias ecuatorianas. El avance tecnológico ha permitido la implementación de sistemas innovadores que superan cada día los sistemas de seguridad convencionales. Entre estas soluciones, el Internet de las Cosas (IoT) ha surgido como una poderosa herramienta para mejorar la seguridad residencial. Sin embargo, la seguridad residencial tradicional ha dependido en gran medida de sistemas de alarma y vigilancia convencionales. Aunque estos métodos siguen siendo efectivos y también usados actualmente, el avance tecnológico permite la generación de nuevas ideas para afrontar el tema de la seguridad.

En temas de seguridad residencial en un entorno IoT, la acción eficiente de accesos y la supervisión precisa son primordiales para garantizar la seguridad y el cuidado de los residentes. Estos sistemas tradicionales de seguridad residencial han estado basados en métodos manuales o semiautomatizados, como las cerraduras y alarmas básicas. No obstante, estas soluciones presentan vulnerabilidades y limitaciones significativas: son propensas a fallas en la detección temprana de intrusiones, carecen de capacidad para un monitoreo detallado y no permiten una respuesta inmediata en situaciones de emergencia. La vigilancia y el control de accesos de un hogar dependen en gran medida de métodos tradicionales, lo que resulta un proceso lento con las respuestas ante posibles amenazas, y además aumentan la probabilidad de una intrusión.

La utilización de los sistemas actuales, como el propuesto basado en tecnologías de IoT, facilita la supervisión y control de la seguridad residencial.

El Internet de las Cosas (IoT) está compuesto por una serie de dispositivos que pueden comunicarse entre sí y con el usuario a través de Internet. Esta interconexión brinda la oportunidad de monitorear y controlar una variedad de elementos en el hogar, desde sistemas de alarma, cámaras de seguridad, cerraduras electrónicas y sensores de movimiento.

1.1 Descripción del Problema

Hoy en día, existe una preocupante tendencia al alza en la tasa de robos a viviendas cuando los residentes se encuentran ausentes. Esta problemática plantea serias amenazas a la tranquilidad y a la integridad de las personas, así como a sus bienes.

Los hogares desocupados se han convertido en objetivos de actos delictivos, lo que genera inseguridad y angustia en la población. La falta de soluciones efectivas y accesibles para abordar esta cuestión hace surgir la necesidad de crear nuevas maneras de proteger los hogares y sus habitantes.

Según Tecnicom empresa experta en sistemas de seguridad en España, los robos en residencias no solo resultan en pérdidas materiales significativas, sino que también provocan un profundo impacto psicológico en las víctimas, generando un ambiente de desconfianza y temor en las comunidades. [1] Por lo tanto, es crucial encontrar soluciones innovadoras y efectivas que disuadan los actos delictivos en los hogares y restablezcan la sensación de seguridad en el entorno residencial.

En un análisis de las estadísticas nacionales de robos a domicilio que proporciona la Fiscalía General del Estado, en el período de enero a noviembre en los años 2019, 2020 y 2021. En 2019, se reportaron un total de 10,107 casos, una cifra que disminuyó notablemente en 2020, descendiendo a 6,643. Sin embargo, en 2021, se registró un aumento en esta cifra, alcanzando los 7,449 casos. [2] A continuación, las cifras de este análisis se muestran en la figura 1.1.

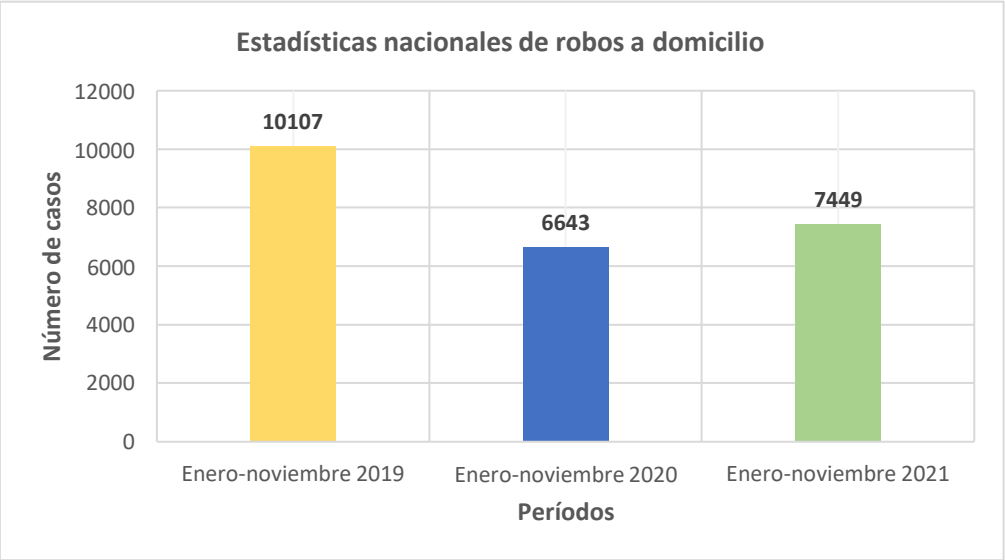

Figura 1.1 *Serie Enero - noviembre 2019, 2020 y 2021* [2]

Nota. Estos datos reflejan una dinámica cambiante en la seguridad de las viviendas a nivel nacional durante estos años.

Por otra parte, la falta de integración de sistemas de IoT para la seguridad residencial impide ofrecer una experiencia de seguridad verdaderamente eficiente y adaptada a las necesidades modernas de protección del hogar.

Este proyecto de investigación pretende solucionar una parte de la seguridad del hogar, ya que actualmente existe una alta tasa de robos a domicilios cuando no hay nadie en casa; esto se logrará mediante la vigilancia por imágenes usando la aplicación Telegram e incorporando un botón de pánico activado mediante mensaje de texto. Adicionalmente, se utilizará una alarma para disuadir el acto delictivo y en consecuencia reducir las probabilidades de robo en el hogar. Este proyecto está enfocado para el sector residencial.

Por las razones antes citadas, este proyecto tiene como objetivo que el usuario pueda ser notificado si alguna persona no identificada se acerca a la residencia, mediante el uso de un sistema IoT de detección de movimiento, alarma y envío de imágenes a través de la plataforma Telegram. Con esto se espera que el usuario pueda tener un control más accesible a la vigilancia de su residencia, ya que involucra nuevas tecnologías como el uso de Telegram siendo esta una aplicación de seguimiento para la observación y monitoreo de la residencia.

El enfoque integral propuesto e implementado busca ayudar al control de la seguridad y brindar una mayor facilidad de supervisión de las viviendas.

1.2 Justificación del Problema

Hoy, en el mercado existen muchos sistemas IoT que son ocupados en el área de la seguridad; sin embargo, son costosos y no se pueden vincular con otras plataformas para su respectiva observación. Estas deficiencias, que incluyen costos elevados, complejidad en la instalación, falta de expansión personalizada, tiempos de respuesta limitados y una sensación de falta de control, han creado la necesidad de una solución más eficiente, además de requerir contratos de servicio a largo plazo. [3]

Esto limita su accesibilidad y hace que muchas personas no puedan beneficiarse de una seguridad efectiva en sus hogares. Por ejemplo, los sistemas de cámaras de vigilancia tradicionales a menudo requieren una inversión inicial sustancial y un alto consumo de datos, junto con costos adicionales de instalación y mantenimiento. [4] Además, pueden ser complejos de configurar y operar, lo que puede llevar a que los usuarios no aprovechen completamente sus capacidades de vigilancia.

Por otro lado, la implementación de un sistema IoT, para este proyecto, reduce la dependencia de costosos sistemas de seguridad. Un dispositivo que utiliza tecnología IoT y se comunica a través de la plataforma de mensajería Telegram es una solución más asequible y eficiente. También es más accesible, ya que es fácil de instalar y utilizar, lo cual es ideal para varios usuarios. Además, muchos sistemas de seguridad tradicionales tienen limitaciones en cuanto a su capacidad de personalización y expansión. [5] Estos sistemas suelen ser rígidos en su diseño y no se adaptan fácilmente a las necesidades específicas de cada hogar. Por ejemplo, la mayoría de los sistemas de alarma tienen características predefinidas que no pueden modificarse fácilmente.

Ante esta situación, se propone la implementación de un dispositivo innovador y programable que, por medio de la detección de movimiento, botón de pánico virtual, sensor magnético de apertura de puerta y envío de imágenes a través de la plataforma de mensajería

Telegram. Mediante la utilización de dos módulos ESP32-CAM, el usuario pueda vigilar y controlar su domicilio de una manera dinámica y accesible. Al emplear un control de vigilancia basado en IoT, se reduce la necesidad de costosos y espaciosos sistemas de seguridad que actualmente ofrece el mercado. Cabe indicar que este proyecto está pensado para poder expandirse a más cámaras utilizando más módulos ESP32-CAM que permitan reforzar la supervisión domiciliaria como, por ejemplo, en la parte posterior de la residencia.

1.3 Objetivos

1.1.1 Objetivo general

Implementar y desarrollar un sistema de seguridad utilizando el microprocesador ESP32-CAM y sensores con asistencia remota para el monitoreo telemétrico de una residencia.

1.1.2 Objetivos específicos

1. Estudiar el uso de un procesador ESP32-CAM en el control remoto de un sistema de seguridad usando la plataforma Telegram.
2. Integrar el módulo ESP32-CAM y sensores en un sistema de seguridad residencial, que emplee la plataforma de mensajería Telegram para el envío de notificaciones en tiempo real ante detecciones de anomalías, aplicando el protocolo ESP-NOW para la incorporación de más sensores asegurando un monitoreo efectivo de la residencia.
3. Evaluar el rendimiento del sistema IoT observando la activación de los sensores y la rapidez en la notificación, para el logro de una respuesta rápida desde la detección de cualquier intruso en la residencia.

1.4 Marco teórico

1.4.1 El Internet de las cosas (IoT)

Actualmente, la aplicación de Internet de las cosas (IoT) en la seguridad residencial ha generado un campo de estudio y desarrollo con gran potencial.. El Internet de las Cosas se define como una nueva arquitectura que enlaza dispositivos físicos mediante la conexión a Internet, facilitando la captura y transmisión de datos de manera instantánea. [6] Este paradigma

proporciona un terreno fértil para el diseño de sistemas avanzados para contribuir al tema de seguridad y el bienestar en los entornos residenciales.

La función principal del Internet de las Cosas (IoT) radica en permitir la adquisición de datos, el manejo y la automatización de procedimientos con el propósito de mejorar la eficacia y la capacidad de decisión en diversas aplicaciones. [7] En el ámbito particular de esteproyecto, se aspira aplicar estos principios a la seguridad residencial mediante la integraciónde dispositivos y sistemas conectados, aprovechando la plataforma Telegram para ofrecer alos usuarios un control intuitivo y acceso remoto a los datos generados por el sistema IoT. **Ventajas del IoT:** [7]

- Automatización y eficiencia: El IoT permite automatizar procesos con el fin de mejorar la eficiencia y la capacidad de toma de decisiones en diversas aplicaciones. Por ejemplo, en la manufactura, la automatización basada en IoT puede aumentar la productividad y reducir los errores.
- Captura de datos en tiempo real: Los dispositivos IoT tienen la capacidad de adquirir datos al instante, proporcionando información importante para la toma de decisiones.
- Ahorro de energía y recursos: En aplicaciones de IoT para hogares inteligentes, iluminación eficiente, sistemas de calefacción, sistemas de refrigeración, y gestión de energía que ayuda a un consumo menor de energía y reducir las tarifas energéticas.
- Mayor comodidad y calidad de vida: Los dispositivos de Internet de las cosas (IoT) en el hogar contribuyen a una mejoría en la calidad de vida. Por ejemplo, termostatos inteligentes pueden ajustar la temperatura de forma automática y sistemas de seguridad pueden proporcionar tranquilidad.
- Nuevas oportunidades de negocio: El IoT crea nuevas oportunidades de negocio al habilitar servicios y soluciones innovadoras. Empresas de todos los tamaños pueden encontrar formas de monetizar los datos recopilados a través de dispositivos IoT.

Desventajas del IoT: [7]

- Seguridad y privacidad: La interconexión de dispositivos IoT puede crear vulnerabilidades de seguridad. La ausencia de medidas de seguridad apropiadas puede llevar a la vulnerabilidad de información confidencial o incluso en la toma de control de dispositivos.
- Interoperabilidad: Los dispositivos IoT a menudo provienen de diferentes fabricantes y pueden no ser compatibles entre sí. Esto puede dificultar la creación de sistemas cohesivos y funcionales.
- Costos iniciales y mantenimiento: La implementación de soluciones IoT puede requerir una inversión inicial significativa en hardware y software. Además, el mantenimiento y actualización de dispositivos y sistemas pueden ser costosos a lo largo del tiempo.
- Sobrecarga de datos: El IoT genera grandes cantidades de datos. Gestionar, almacenar y analizar estos datos puede ser desafiante y costoso. Además, puede haber preocupaciones sobre la recopilación de datos excesivos y su impacto en la privacidad.
- Dependencia de la conectividad: El IoT depende de la conectividad a Internet. Si la red falla o es inestable, los dispositivos IoT pueden volverse inoperables.
- Cuestiones éticas y legales: El IoT plantea preguntas éticas y legales relacionadas con la propiedad y el uso de datos confidenciales.

1.4.2 Módulo ESP32-CAM

La Placa ESP32-CAM es una parte esencial en la realización de un sistema IoT de seguridad. El ESP32-CAM incorpora un microcontrolador de doble núcleo ESP32, brindando un rendimiento robusto y capacidad de procesamiento versátil para una variedad de aplicaciones. Además, está equipado con una cámara OV2640 de 2 megapíxeles que brinda imágenes y videos de buena calidad. Por otra parte, cuenta con capacidades de conectividad Wi-Fi, que lo hace ideal para aplicaciones que involucran la transmisión inalámbrica de datos, como la vigilancia remota, la detección de movimiento y otros proyectos relacionados con la captura de imágenes y videos. [8] Para este proyecto, la integración del ESP32-CAM (ver

figura 1.2) posibilita la vigilancia remota y la detección visual de eventos relevantes para la seguridad residencial.

Figura 1.2 *Imagen de la ESP32-CAM* [9]

Nota. Este módulo está diseñado con el objetivo de alcanzar un óptimo desempeño en potencia y frecuencia de radio, lo que lo hace versátil, resistente y confiable en diversas aplicaciones y situaciones. [8]

Se utiliza ampliamente en el campo de IoT, destacando características que lo hacen especialmente adecuado para este entorno, tales como: [10]

- Se activa de forma periódica al detectar una condición específica.
- Su ciclo de trabajo es bajo para reducir el menor consumo energético del chip.
- La capacidad de ajuste en la salida del amplificador de potencia permite un equilibrio adecuado entre la velocidad de datos y el rango de distancia en la comunicación.

Este dispositivo tiene unas características importantes lo que hace ideal para el sistema IoT debido a su conectividad Wi-Fi. Entre sus funcionalidades compatibles con Wi-Fi se incluye: [10]

- 802.11 b/g/n
- 802.11 n (2.4GHz), mayor de 150 Mbps
- Monitoreo automático de los Beacon
- Cuatro interfaces virtuales Wi-Fi
- Desfragmentación
- Bloqueo inmediato de ACK

1.4.3 Protocolo de comunicación UART

El Receptor-Transmisor Asíncrono Universal (UART) desempeña un papel crucial en la implementación de este proyecto de seguridad residencial mediante la plataforma Telegram. El protocolo UART ofrece una transmisión de datos máxima de 5 Mbps. Su característica opera asíncronamente facilitando la interacción con los dispositivos, ya que no requiere la transmisión de un reloj entre ellos. Es esencial destacar que el reloj interno de cada dispositivo UART debe operar a un múltiplo de la velocidad en baudios, asegurando que cada bit se muestre N veces. Por ende, si se habla de la comunicación entre el dispositivo controlador (ESP32-CAM) y los dispositivos descendentes, como los sensores de movimiento y magnetismo, se emplean únicamente dos hilos, simplificando la implementación. [11]

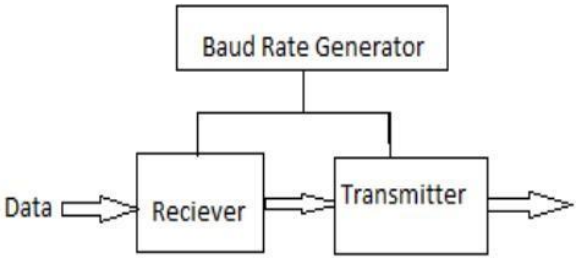

Figura 1.3 *Módulo UART* [12]

Nota. En el diagrama nos muestra el módulo de comunicación serie UART en la cual se divide en tres submódulos: el generador de velocidad en baudios, el módulo receptor y el módulo transmisor.

Características del protocolo UART: [12]

- **Asincronía:** La comunicación UART es asincrónica, esto significa que no requiere un reloj compartido entre los dispositivos conectados. En cambio, utiliza bits de inicio y parada para delimitar cada byte de datos.
- **Baud Rate:** La velocidad de transmisión de datos, medida en baudios, es crucial en la configuración de UART. Todos los dispositivos deben operar a la misma velocidad para una comunicación exitosa.

- **Bits de Datos y Bits de Parada:** La configuración de UART incluye la cantidad de bits de datos por byte y los bits de parada. Comúnmente, Se emplean 8 bits de información junto con 1 o 2 bits de parada.
- **Full-Duplex o Half-Duplex:** UART puede operar en modos full-duplex o half-duplex siendo de transmisión y recepción simultaneas y alternadas respectivamente.

1.4.4 Protocolo de comunicación ESP-NOW

ESP-NOW, desarrollado por Espressif Systems, Es un protocolo de comunicación sin cables que facilita la interacción entre diversos dispositivos sin depender de una red particular. Se adapta a los modos de conexión Wi-Fi de los procesadores, tales como ACCESS POINT, STATION o ACCESS POINT + STATION, ofreciendo una versatilidad extensa en la utilización de las tarjetas.. Este protocolo permite el intercambio rápido de pequeños paquetes de datos a través de las bandas de frecuencia de 2,4GHz. Su funcionamiento implica el emparejamiento inicial de dispositivos, pero una vez establecido, la conexión es automática. Además, ESP-NOW opera como un protocolo de comunicación Wi-Fi sin conexión a internet, encapsulando los datos de la aplicación en un marco de acción específicodel proveedor y transmitiéndolos de un dispositivo Wi-Fi a otro sin necesidad de establecer una conexión previa. [13]

ESP-NOW representa un protocolo de comunicación sin cables creado por Espressif Systems que opera en un formato maestro/esclavo, permitiendo la comunicación entre dispositivos. Cada dispositivo puede conectarse hasta con 20 pares, con restricciones en el número de claves de comunicación según el tipo de conexión. Las conexiones entre pares son significativamente más rápidas que las conexiones Wi-Fi convencionales, facilitando un consumo eléctrico reducido. Este rendimiento eficiente es especialmente beneficioso para dispositivos alimentados por baterías, como sensores o mandos inalámbricos, permitiendo activar, recopilar datos o enviar órdenes en cuestión de milisegundos y luego volver al reposo. [14]

Figura 1.4 *Comunicación ESP-NOW en dos direcciones* [13]

Nota. En la imagen nos muestra dos módulos ESP32-CAM utilizando protocolo de comunicación ESP-NOW en dos direcciones en la cual pueden actuar como remitentes y destinatarios de los mensajes, lo cual la hace más flexible.

Características del ESP-NOW: [13]

- ESP-NOW es compatible con la comunicación unicast, ya sea cifrada o sin cifrar.
- Se permite la combinación de clientes con cifrado y sin cifrado en una red única..
- Permite la transmisión de hasta 250 bytes de carga útil en cada comunicación.
- Ofrece la capacidad de configurar callbacks que informan a la aplicación sobre la correcta realización de la transmisión.
- Sistemas que integran ESP-NOW pueden superar los 200 metros en entornos de campo abierto.

1.4.5 Plataforma Telegram

La elección de Telegram como plataforma de mensajería ofrece un canal seguro y cifrado para la comunicación, posibilitando que los usuarios reciban alertas y observen imágenes en tiempo real. La integración con esta plataforma proporciona una solución eficiente para la supervisión remota, facilitando la comunicación y el envío instantáneo de imágenes. [15]

Características de la plataforma: [16]

- **Privacidad y Seguridad:** Telegram permite mantener las conversaciones seguras y privadas, ofreciendo chats secretos con cifrado de extremo a extremo.

- **Datos en la Nube:** Los mensajes y archivos multimedia se almacenan en la nube, lo que facilita la sincronización en múltiples dispositivos.
- **Grupos y Canales:** Se pueden crear grupos y canales con un gran número de miembros, ya sean públicos o privados, lo que facilita la comunicación con amigos o una audiencia más amplia.
- **Bots y API:** Telegram brinda la posibilidad de crear bots y aplicaciones personalizadas que interactúen con la plataforma, permitiendo una amplia gama de usos y servicios.
- **Interfaz Amigable:** Telegram ofrece una interfaz intuitiva en diversas plataformas, lo que la hace fácil de usar y accesible para todos.
- **Multiplataforma:** Se puede utilizar Telegram en varios dispositivos, como celulares, computadoras de escritorio y navegadores web.
- **Mensajería Rápida:** Telegram se destaca por su velocidad y eficiencia en la entrega de mensajes, lo que garantiza una comunicación fluida.

Capítulo 2

2. Diseño del sistema IoT para el monitoreo de la residencia

Al abordar la implementación de este sistema IoT, se describen los protocolos utilizados para la conexión Wi-Fi y la transmisión de datos. El diseño general del sistema se expone a través de diagramas metodológicos y de flujo, detallando el funcionamiento y proceso de implementación. La sección de implementación proporciona información sobre materiales, configuraciones de hardware y software, códigos de programación y conexiones. Además, se destaca la escalabilidad del sistema, sugiriendo la capacidad de agregar más dispositivos.

2.1 Comunicación del sistema IoT

En esta sección se detalla los protocolos de comunicación utilizados para la conexión entre el módulo ESP32-CAM y la API de Telegram, garantizando la transferencia segura de datos mediante una red Wi-Fi. Además, se describe los protocolos que se aplica en este sistema.

2.1.1 Protocolo TCP/IP

TCP/IP que permite una transmisión de datos segura en una red, asegurando que cada mensaje llegue de manera íntegra a su destino. Con el fin de lograr este objetivo, se fragmenta la información en unidades más pequeñas denominadas paquetes, los cuales son posteriormente reensamblados al llegar a su destino. [17]

Por ejemplo, En la figura 2.1 se detalla el proceso seguido para enviar una foto por medio del ESP32-CAM en que el protocolo TCP/IP divide los datos en paquetes a través de un proceso de cuatro capas. Estos datos cruzan varias capas en un orden específico durante el envío y, posteriormente, siguen la ruta inversa al ser recopilados en su destino. Para establecer la comunicación mediante una red Wi-Fi entre el módulo ESP32-CAM y la plataforma Telegram, se utiliza el protocolo destino. Este proceso funciona porque esta estandarizado lo que facilita una manera más eficiente para gestionar la transmisión de datos en una red, garantizando una comunicación ordenada y confiable.

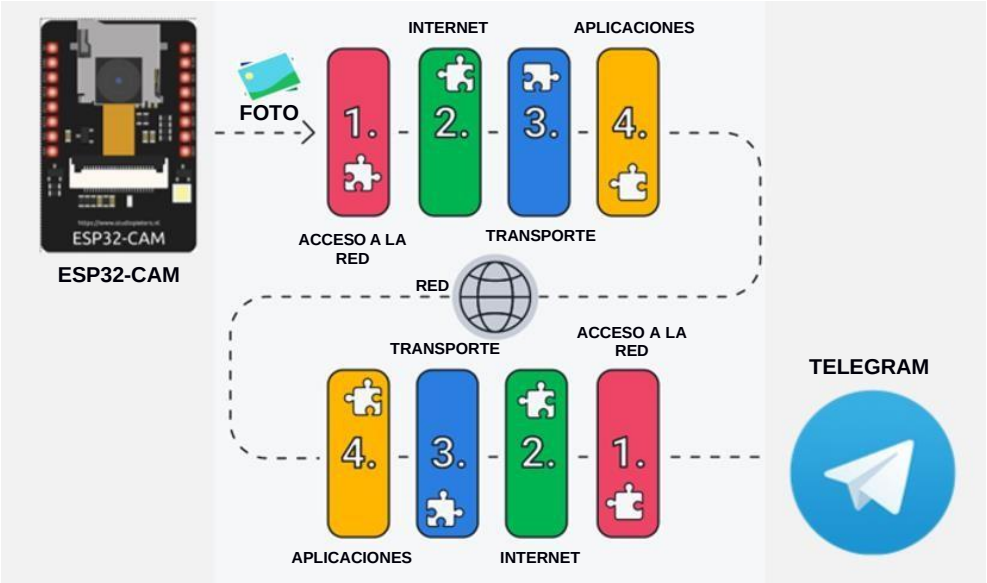

Figura 2.1 *Diagrama del modelo TCP/IP* [18]

Nota: Este diagrama representa el modelo TCP/IP y cómo fragmenta la información en paquetes, enviándola a través de cuatro capas diferentes.

Estas cuatro capas son identificadas como: (1) Acceso a la red, que supervisa la infraestructura física facilitando la comunicación entre computadoras a través de Internet; (2) Internet, responsable de gestionar el flujo y enrutamiento del tráfico para garantizar el envío rápido y preciso de datos; (3) Transporte, que establece una conexión de datos confiable entre dos dispositivos de comunicación; y (4) Aplicaciones, que posibilita al usuario acceder a la red mediante aplicaciones como Telegram. Estos forman un grupo de protocolos, cuando un usuario envía información, este modelo TCP/IP guía la información a través de estas capas en un orden específico, y cuando se reciben la información, sigue el orden inverso. [18]

El modelo TCP/IP incluye diversos protocolos de Internet que establecen cómo se gestionan y transmiten los datos. Lo más comunes son HTTP, FTP y SMTP, siendo estos con mayor frecuencia en combinación con el modelo TCP/IP.

2.1.2 Protocolo HTTPS

Una vez establecida la conexión con el modelo TCP/IP, el módulo ESP32-CAM utiliza un protocolo de aplicación llamado HTTPS conocida como protocolo de transferencia de hipertexto seguro que añade una capa de seguridad mediante la encriptación de

comunicación SSL/TLS. [19] Este protocolo nos ayuda a cifrar la comunicación y asegurar la transmisión de datos de manera segura entre el ESP32-CAM y los servidores de Telegram.

Esto es muy importante cuando los usuarios transmiten datos confidenciales, como, por ejemplo, al tomar fotos comprometidas dentro de la casa. Con el protocolo HTTPS, se impide que los sitios web divulguen su información de manera que sea fácilmente visible para cualquier persona que esté tratando de capturar datos ajenos en la red. [20] Con la plataforma Telegram, suele corregir cualquier vulnerabilidad detectada mediante sus actualizaciones regulares para garantizar la seguridad de sus usuarios.

2.1.3 Protocolo UART

Para la interconexión de los sensores y los módulos ESP32-CAM, estableciendo comunicación en serie por medio de la interfaz UART. La conexión física se establece conectando los pines de transmisión (TX) de un módulo al pin de recepción (RX) del otro y viceversa. [21] La Figura 2.2 presenta un esquema de conexión UART entre dos dispositivos.

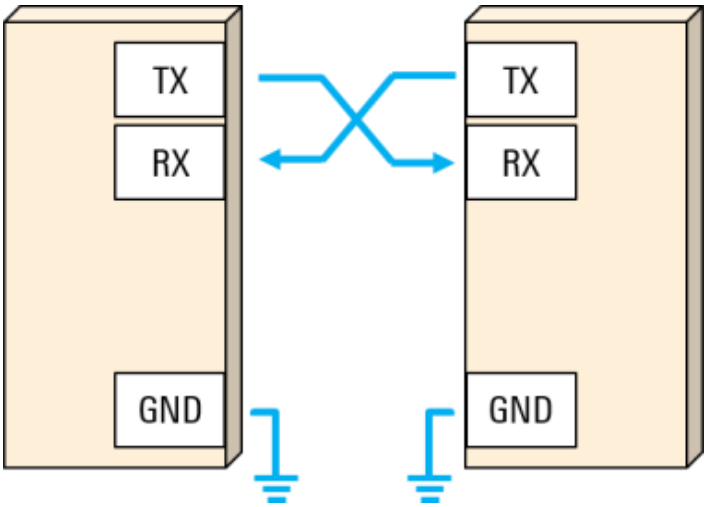

Figura 2.2 *Esquema de conexión UART* [21]

Nota: En el diagrama de conexión, el pin TX nos permite transmitir datos, mientras que, que el pin RX nos permite recibir datos.

En el código de cada módulo, se realiza la configuración de la comunicación UART. El módulo maestro utiliza Serial.write() para enviar datos, mientras que el módulo esclavo

utiliza Serial.read() para recibir estos datos. Además, se define una estructura de mensaje para organizar la información intercambiada, como el tipo de evento detectado. [22]

2.1.4 Protocolo ESP-NOW

La configuración de las ESP32-CAM se ha realizado tanto por Wi-Fi como por ESP-NOW, además la comunicación es de manera bidireccional, en este caso es necesario obtener la dirección MAC de cada dispositivo subiendo el código a cada ESP32CAM (ver Apéndice C). Cada uno de los 2 dispositivos cuenta con sensores de movimiento y sensores magnéticos de puertas. El dispositivo principal tiene un sistema de alarma del tipo buzzer. Con respecto a la transferencia de información, el dispositivo secundario al detectar movimiento por alguno de los sensores envía el estado de estos al dispositivo principal por medio de ESP-NOW. El dispositivo principal es el encargado de transferir la mayoría de información por Wi-Fi hacia la API de Telegram. A cada uno de los dispositivos se les puede solicitar una captura de imagen. Se probó en su momento la transferencia de la imagen del dispositivo primario al secundario mediante ESP-NOW, pero se presentaron ciertas limitaciones ya que el protocolo al usar radiofrecuencia tiene una capacidad máxima de transferencia de 250 bytes por mensaje, el cual es debido a que la imagen generada tiene un peso mayor. Se hizo necesario fragmentar en diversos paquetes ya que debido a los otros procesos que ocurrían en cada sensor (como la recolección de datos de los sensores y la conexión con Wi-Fi del primer dispositivo), se generaba una pérdida de paquetes en el dispositivo principal por lo que la imagen del dispositivo secundario no se podía obtener. Lo que se optó fue implementar Wi-Fi en el segundo dispositivo únicamente para la subida de la imagen secundaria. Esta limitación se puede mejorar a futuro mediante la implementación de un algoritmo más robusto en el dispositivo principal que permita controlar la cantidad de paquetes recibidos y faltantes para poder solicitar de nuevo los paquetes al dispositivo secundario.

2.2 Diseño general del sistema IoT

En esta sección se describe los diagramas de la metodología, de conexiones y de flujo del sistema IoT. Además, se indica el funcionamiento y el proceso para la implementación del proyecto.

2.2.1 Diagrama Metodológico

En la ejecución de este proyecto se procede a dividir en diversas etapas durante su desarrollo. En la Figura 2.3 se describe el esquema empleado para llevar a cabo cada fase, desde la concepción inicial del sistema hasta su implementación definitiva..

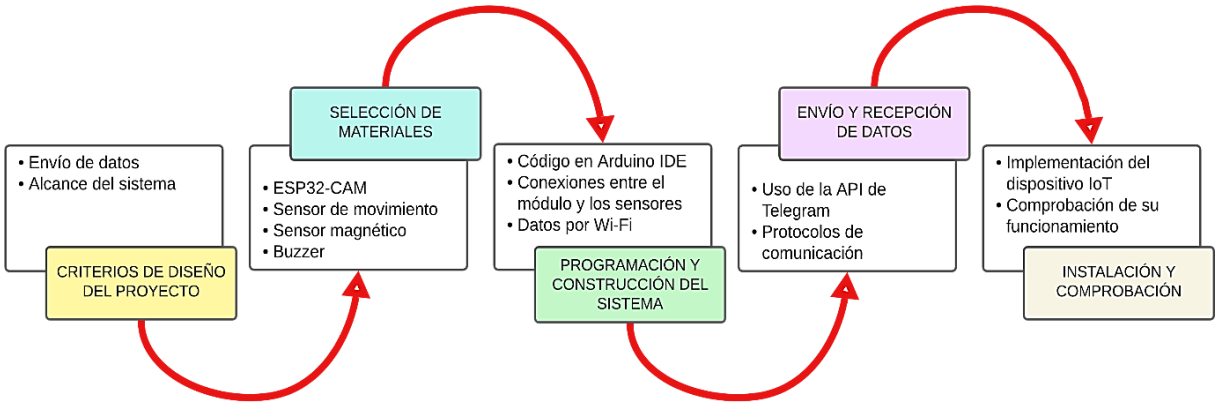

Figura 2.3 *Diagrama de metodología* [23]

Nota. El diagrama muestra la metodología empleada para la implementación del sistema IoT.

Se inicia la evaluación de los criterios destinados al diseño de nuestro sistema, considerando los requisitos principales. Se garantiza que el sistema sea capaz de transmitir y recibir datos bidireccionalmente. Después, se elige los materiales, siendo el módulo ESP32-CAM como la pieza fundamental de nuestro sistema y los diferentes sensores como el de movimiento y magnético, para así realizar la programación y conexión del sistema utilizando el entorno de desarrollo integrado de Arduino (Arduino IDE) para posteriormente verificar la conectividad.

Adicionalmente, se continua al siguiente bloque de envío y recepción de datos, donde se puso a prueba el prototipo y se verifica la conexión con la API de Telegram evidenciando el funcionamiento de los protocolos el cual permite que se establezca de forma segura la

comunicación con el usuario. Finalmente se realiza la instalación del dispositivo IoT en una residencia y se comprueba el correcto funcionamiento del sistema.

2.2.2 Diagrama de conexiones

Se realiza el esquema de conexiones correspondiente al prototipo del sistema IoT donde se detalla la conexión de cada elemento con el microcontrolador ESP32-CAM. Se observa el dispositivo Maestro que constan de los sensores, buzzer y la batería conectada al módulo ESP32-CAM, Tal como se puede apreciar en la Figura 2.4, y a su vez en la Figura 2.5, se visualiza el dispositivo Esclavo que incluye los sensores, y la batería conectada al módulo ESP32- CAM. Ambas partes están conectadas a la misma red Wi-Fi donde el dispositivo Esclavo se encarga enviar datos al dispositivo maestro, que a su vez se encarga de transmitir la información al usuario.

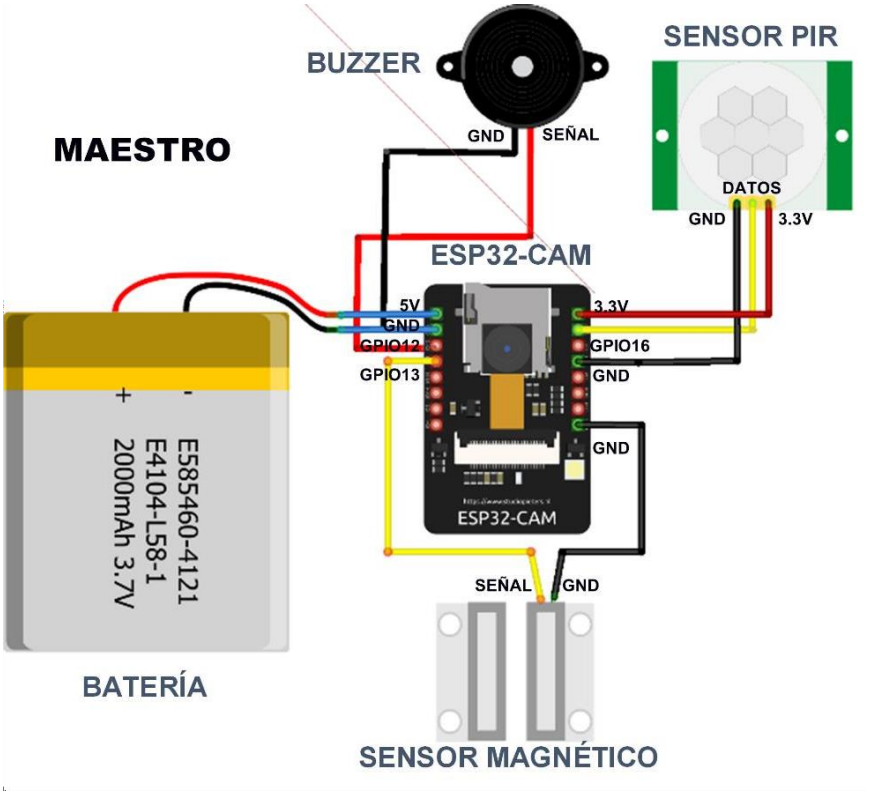

Figura 2.4 *Esquema de conexiones del dispositivo Maestro*

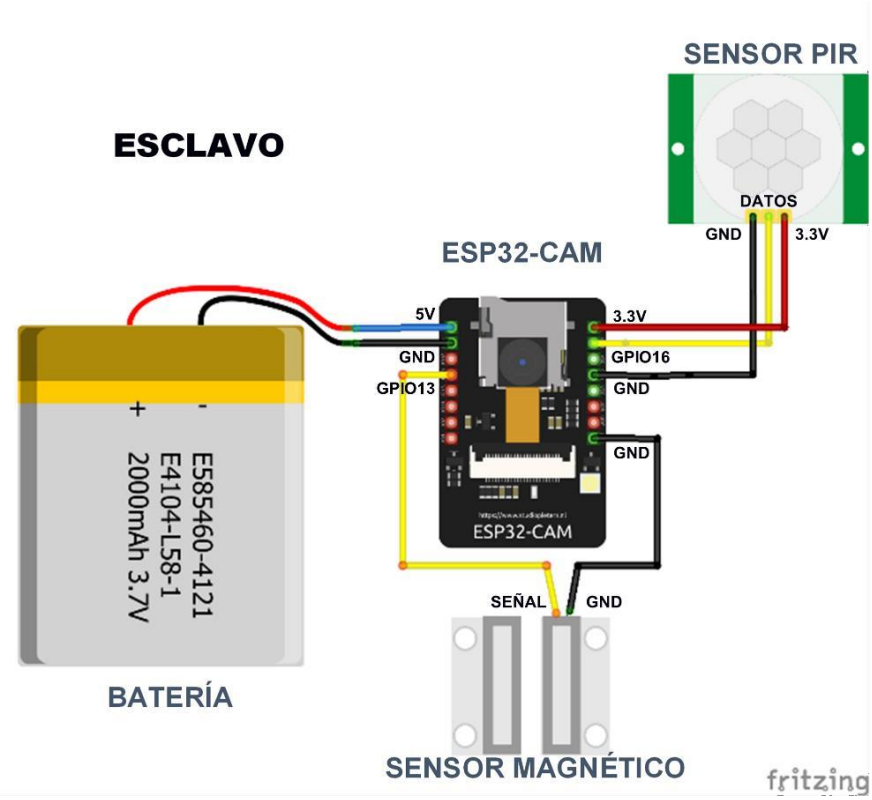

Figura 2.5 *Esquema de conexiones del dispositivo Esclavo*

Nota. Los diagramas muestran los dispositivos Maestro y Esclavo del sistema, detallando las conexiones en cada uno de ellos.

En la figura 2.4 se detalla la conexión del dispositivo Maestro, este es responsable de establecer la conexión con la API de Telegram, gestionando el envío y recepción de todos los datos hacia y desde la aplicación de Telegram. La Figura 2.5 describe la conexión del dispositivo Esclavo, el cual se encarga de recopilar los datos provenientes del sensor magnético, de movimiento y de la segunda cámara, para luego enviar estos datos al dispositivo Maestro y a su vez este se encarga de toda la comunicación con la aplicación de Telegram.

Tanto en el dispositivo Maestro como en el Esclavo, ambos módulos deben estar alimentados por corriente directa de 5 voltios, en este caso se utiliza una batería recargable para cada módulo. La batería va conectada en los pines 5V y GND tal como se muestra en la figura 2.4 y 2.5. Para los sensores de movimiento en cada caso se alimenta de 3,3 voltios el cual va conectado a los pines 3.3V y GND y para los datos se conecta al pin GPIO16,

aparte para los sensores de movimiento se conectan a los pines GPIO13 y GND. Finalmente, el buzzer, que se encargara de dar un aviso sonoro en la residencia, se conecta a los pines GPIO12 y GND.

2.2.3 Diagrama de flujo del Sistema IoT

Se procede a realizar el esquema del sistema el cual es importante para poder entender el funcionamiento técnico de este proyecto.

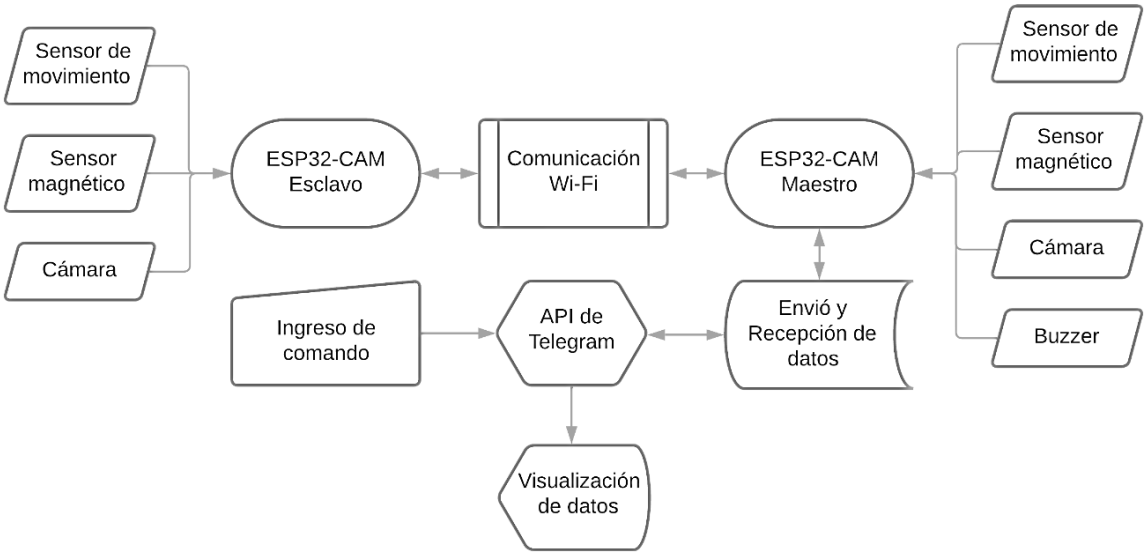

Figura 2.6 *Diagrama de flujo del sistema IoT*

Nota. El diagrama muestra el proceso de ejecución del sistema, detallando los pasos en cada uno de ellos.

La figura 2.6 muestra el proceso de ejecución y funcionamiento del sistema implementado, la información que reciben los módulos Maestro y Esclavo viene de los sensores y cámaras; ambos módulos se conectan entre si utilizando un canal de comunicación Wi-Fi de banda de 2.4 GHz. El dispositivo Maestro es el que se encarga de mantener comunicación con la API de Telegram. Como se había explicado anteriormente Se emplea el protocolo HTTPS para garantizar una mayor seguridad en la transmisión y recepción de datos. Luego de establecer conexión con la API de Telegram, se procede a enviar o recibir datos según los comandos del usuario o la activación de sensores en los módulos.

2.3 Implementación del sistema IoT

En esta sección de detalla las listas de los materiales a utilizar y además, se incluyen las configuraciones tanto a nivel de hardware como de software necesarias para la implementación del sistema., así como códigos deprogramación y conexiones.

Lista de los materiales

Se detalla los materiales utilizados en la implementación del sistema IoT. En la tabla 2.1, se da una descripción de cada dispositivo que se utiliza, incluyendo el modelo especifico correspondiente.

Dispositivos	Descripción	Modelo
Microcontrolador	Un microcontrolador que recibe información de los sensores, la procesa, la transmite a través de Internet.	ESP32-CAM
Sensor de movimiento	Un sensor que detecta movimiento al percibir cambios en el entorno. Tiene un rango de alcance de 3 a 7 metros.	HC-SR501
Sensor magnético	Un sensor que detecta cambios en un campo magnético para activar una respuesta específica.	MC-38
Buzzer	Un componente electroacústico que convierte señales eléctricas en sonido.	SFM-27
Batería Li-Ion	Una batería de 5V y 5000mAh, empleada como fuente de alimentación para el sistema.	N/A
Adaptador AC/DC	Un adaptador para facilitar la carga eficiente de la batería y la alimentación del sistema, con una fuente de alimentación de 5V.	N/A

Tabla 2.1 *Tabla de materiales utilizados* [10]

Nota. En la tabla se detalla los materiales empleados en el proyecto.

2.3.1 Configuración de Hardware y Software

Para ejecutar la implementación de este sistema IoT se necesita tener instalado el IDE de Arduino para poder subir el código de programación a la placa ESP32- CAM; esto se logra utilizando un módulo USB a TTL. La elección de utilizar un módulo USB a TTL se fundamenta en la importancia de garantizar una comunicación serial segura entre la computadora y la placa ESP32-CAM. Este módulo actúa como un convertidor de interfaz, facilitando la transferencia de datos y la programación de la placa a través de una conexión USB estándar, lo que resulta esencial para el proceso de desarrollo y actualización del firmware en el dispositivo IoT. (Ver Apéndice A y B).

Conexión del ESP32-CAM

Para subir el código a la placa se requiere emplear un módulo USB a TTL y realizar la conexión de cables según lo ilustrado en la Figura 2.7. Además, se realiza un puente en la placa entre los pines GND (tierra) e IO0 (entrada/salida cero) para habilitar el modo de carga de código. Este puente entre GND e IO0 se lleva a cabo con el propósito de poner la placa ESP32-CAM en un estado específico conocido como "modo de carga" o "modo de programación". Al conectar GND e IO0, se indica a la placa que debe prepararse para recibir el nuevo código, permitiendo así la programación exitosa a través del módulo USB a TTL.

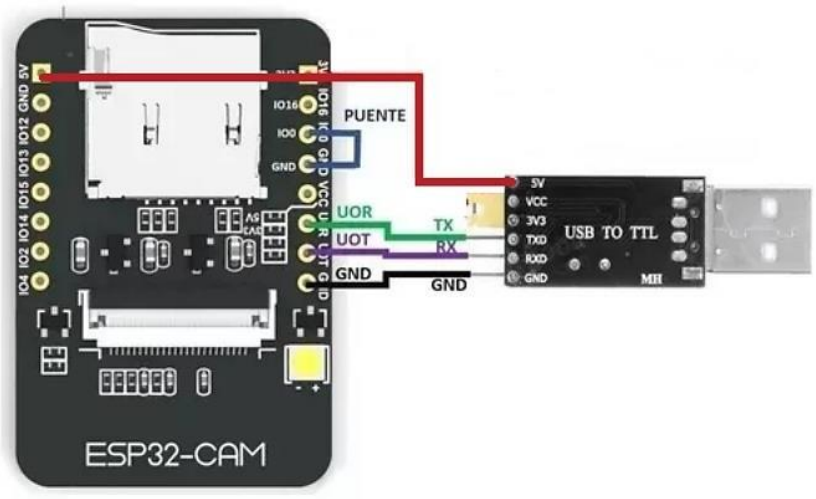

Figura 2.7 *Diagrama de conexión del ESP32-CAM al módulo USB a TTL* [24]

Nota. El diagrama muestra la conexión del ESP32-CAM al módulo USB a TTL para subir el código necesario de las cámaras.

Programación del ESP32CAM

Al momento de subir el código a la placa se debe tener en cuenta ciertos pasos que se indican a continuación:

Conectar el USB a TTL a un puerto USB de la computadora.

Verificar en el administrador de dispositivos el número de puerto COM asignado a la placa.

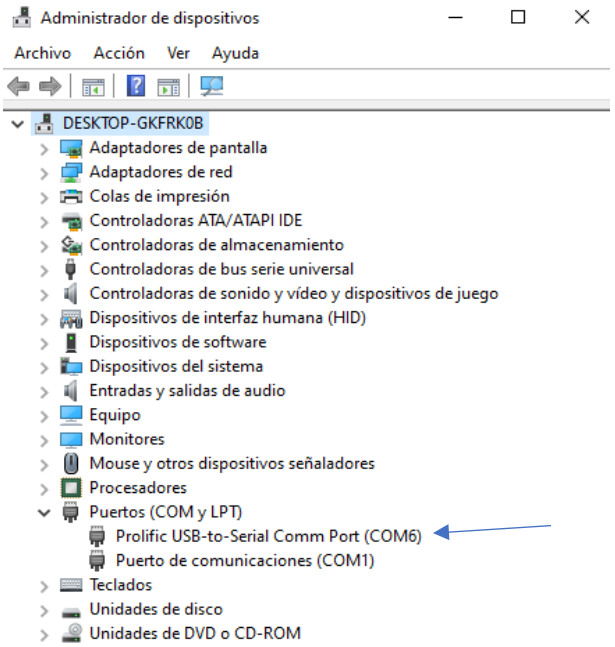

Figura 2.8 *Verificación del puerto COM.*

En el IDE de Arduino se elige el Puerto COM correspondiente como vimos en el paso anterior.

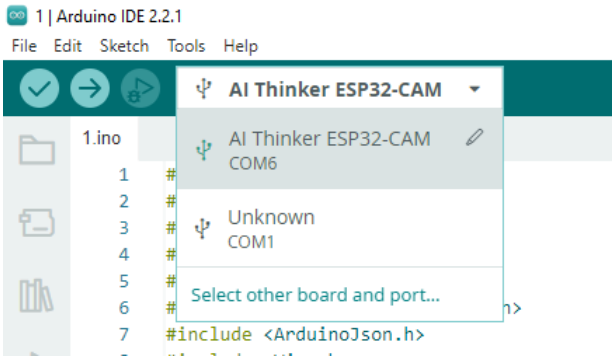

Figura 2.9 *Selección del puerto COM en el IDE de Arduino*

A continuación, se hace clic en compilar y subir.

Para finalizar se quita el puente entre el pin GND y IO0 para salir del modo de programación y se presiona en la placa el botón RESET.

Funcionamiento de la cámara

La cámara del ESP32-CAM es el dispositivo que se encarga de la captura de las imágenes de este sistema IoT. Su tarea principal consiste en adquirir y transmitir imágenes cuando los sensores se activan o cuando el usuario lo solicita. Para poder realizar esa tarea el sistema debe estar conectado a una red Wi-Fi además del uso adecuado de librerías para el correcto funcionamiento. La elección del modelo especifico de la cámara y la configuración de los pines viene por defecto para cada tipo de cámara, como se ilustra en la figura 2.10, y que influyen en el rendimiento y la integración eficiente de la cámara en el sistema.

```
26  //CAMERA_MODEL_AI_THINKER
27  #define PWDN_GPIO_NUM     32
28  #define RESET_GPIO_NUM    -1
29  #define XCLK_GPIO_NUM      0
30  #define SIOD_GPIO_NUM     26
31  #define SIOC_GPIO_NUM     27
32
33  #define Y9_GPIO_NUM       35
34  #define Y8_GPIO_NUM       34
35  #define Y7_GPIO_NUM       39
36  #define Y6_GPIO_NUM       36
37  #define Y5_GPIO_NUM       21
38  #define Y4_GPIO_NUM       19
39  #define Y3_GPIO_NUM       18
40  #define Y2_GPIO_NUM        5
41  #define VSYNC_GPIO_NUM    25
42  #define HREF_GPIO_NUM     23
43  #define PCLK_GPIO_NUM     22
```

Figura 2.10 *Selección de los pines GPIO para el uso de la cámara*

Conexión a la red Wi-Fi

Para conectar el ESP32-CAM a Wi-Fi es necesario tener varias consideraciones para poder ejecutar correctamente el proyecto. En primer lugar, se debe integrar en el código las librerías WiFi.h y WiFiClientSecure.h, reconocidas por su frecuente utilización en proyectos

de esta naturaleza. Estas librerías desempeñan un papel fundamental al facilitar las operaciones de gestión y establecimiento de conexiones Wi-Fi de manera segura. Además, Es fundamental especificar el nombre y la contraseña correspondientes a la red Wi-Fi a la cual la placa se conectará. Se destaca la importancia de optar por una red Wi-Fi que opere en la frecuencia de 2.4 GHz, dado que la placa ESP32-CAM es compatible únicamente con esta frecuencia. Este procesose detalla en la figura 2.11 para proporcionar una guía visual de la configuración pertinente en el contexto del proyecto.

```
314   // Connect to Wi-Fi
315   // WiFi.mode(WIFI_STA);
316   WiFi.mode(WIFI_AP_STA);
317   // WiFi.mode(WIFI_STA);
318   int32_t channel = getWiFiChannel(ssid);
319   esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE);
320
321   Serial.println();
322
323   // Iniciar WiFi
324   initWifi();
```

Figura 2.11 *Código fuente para la conexión Wi-Fi*

Procedimiento para el envío de fotos

El proceso de enviar imágenes desde un módulo ESP32-CAM a la aplicación Telegram implica la captura de imágenes, luego de la compresión y codificación de las imágenes, y su posterior envío a través de la red. La conexión a Internet se establece a través del protocolo Wi-Fi, utilizando el estándar 802.11 b/g/n, que posibilita la conectividad en las bandas de frecuencia de 2.4 GHz, con la configuración de las credenciales de la red Wi-Fi implementado en el código.

Para la comunicación con la plataforma de Telegram, se utiliza la API de Telegram mediante solicitudes HTTP. El módulo construye una solicitud POST multipart/form-data que incluye la imagen capturada. La biblioteca WifiClientSecure podría ser empleada para facilitar el manejo de las solicitudes HTTP. Esta biblioteca implementa soporte para conexiones seguras mediante TLS/SSL. Existen tres formas de establecer una conexión segura utilizando la clase WiFiClientSecure: utilizando un certificado de autoridad de certificación (CA) raíz,

utilizando un certificado de CA raíz más un certificado y una clave de cliente, y utilizando una clave precompartida (PSK). [25]

La API de Telegram procesa la solicitud y almacena la imagen en sus servidores. Además, la comunicación con la API de Telegram se realiza generalmente a través del protocolo HTTPS. Finalmente, el ESP32-CAM recibe confirmaciones de la API de Telegram sobre el éxito del envío para generar notificaciones en la aplicación. Este proceso implica la interacción de varios protocolos, incluyendo Wi-Fi para la conectividad, HTTP/HTTPS para la comunicación con la API de Telegram, y JPEG para la codificación de imágenes, así asegurando el envío eficiente y seguro de la información.

Funcionamiento de los sensores

La configuración de los sensores implica la definición de los pines GPIO que serán empleados en el código. En particular, para el sensor de movimiento, el sensor magnético y el buzzer, se ha asignado el pin 13, 15 y 12, respectivamente. Esta asignación específica de pines se detalla en la figura 2.12.

Cada sensor cumple con funciones distintivas en el sistema. El sensor de movimiento, por ejemplo, se basa en tecnologías como infrarrojos o ultrasonidos para detectar cambios en el entorno cercano, generando señales eléctricas que son interpretadas por el microcontrolador. El sensor magnético opera mediante la detección de cambios en el campo magnético, proporcionando información sobre la apertura o cierre de puertas o ventanas. En cuanto al buzzer, su función radica en la emisión de señales acústicas en respuesta a determinadas condiciones, contribuyendo así a las alertas auditivas del sistema.

```
99    // Pines
100   #define FLASH_LED_PIN 4
101   #define PIR_PIN 13
102   #define MAGNETIC_PIN 15
103   #define BUZZER_PIN 12
```

Figura 2.12 *Selección de los pines GPIO para el uso de los sensores*

2.3.2 Creación Bot de Telegram

En esta subsección se describe la obtención de un token de la API de Telegram y un ID para el servicio de mensajería. En primer lugar, se debe tener instalado en el dispositivo celular la aplicación de mensajería Telegram; luego se busca la palabra botfather o en el navegador se ingresa la dirección t.me/botfather ilustrado en la figura 2.13.

Figura 2.13 *Pasos para crear un Bot*

Se da clic en la palabra INICIAR para empezar a crear el Bot y se sigue una serie de pasos ilustrado en la figura 2.14.

Figura 2.14 *Pasos para crear un Bot*

Al haber seguido todos los pasos correctamente, el Bot nos entregará un token de acceso a la API de Telegram.

También es necesario obtener un Chat-ID para asegurar de que no se reciban mensajes que no provienen de nuestra cuenta de Telegram; para obtener este ID se busca en Telegram IdBot o en el navegador se ingresa a la siguiente dirección t.me/myidbot ilustrado en la figura 2.15.

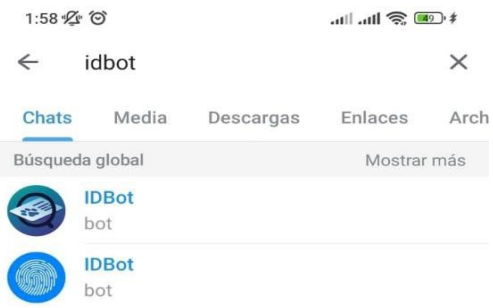

Figura 2.15 *Pasos para obtener un ID*

Se da clic en la palabra INICIAR para obtener un ID y se sigue una serie de pasos ilustrado en la figura 2.16.

Figura 2.16 *Pasos para obtener un ID*

Al finalizar, ya se han obtenido el token de la API de Telegram y el Chat-ID para poder utilizarlos en el código fuente de este proyecto.

Al tener el Token de la API y el Chat-ID del Telegram se procede a ingresar esos datos en la programación del sistema; esto es necesario ya que con esto es posible la comunicación entre la aplicación y el ESP32-CAM. También es importante añadir las librerías necesarias para que funcione el código; en este caso se utiliza la librería UniversalTelegramBot.h el cual permite que la placa se conecte al servidor de Telegram, En el código, es necesario declarar las variables para el token y el Chat-ID, tal como se evidencia en la Figura 2.17.

```
14  // Se declara String para el chatid al igual que el token.
15  String chatId = "1685390701";
16
17  // Se busca botfather para obtener el token de acceso al bot.
18  String BOTtoken = "6985309105:AAEyn73ZOMeHw7QGSpzt4mkMhhWle4VvdzQ";
```

Figura 2.17 *Código fuente para la validación de la API de Telegram*

2.4 Escalabilidad del sistema

Considerando la necesidad de escalabilidad en un futuro y de una fácil expansión del sistema, se sugiere agregar más dispositivos al sistema sin cambiar la estructura fundamental del código. Por ejemplo, agregando otros módulos ESP32-CAM, sensores de movimiento y sensores magnéticos con los cuales se pueda operar de manera independiente y de fácil expansión sin alterar la estructura central del código. Este sistema se complementaría con una gestión centralizada donde un servidor central o una aplicación en la nube que coordina la comunicación entre dispositivos, simplificando la recopilación de datos y permitiendo una gestión eficiente de la detección de movimiento. Adicionalmente, establecer una base de datos para el almacenamiento garantiza la retención de información de los dispositivos, como eventos de detección de movimiento e imágenes capturadas, aprovechando la flexibilidad que brindan los servicios en la nube. Por otra parte, la capacidad de realizar actualizaciones remotas de firmware añade un elemento clave para mantener el sistema eficiente y actualizado sin la necesidad de acceso físico en cada dispositivo, contribuyendo así a la escalabilidad.

Por último, con el fin de mejorar la eficacia del sistema, se sugiere la incorporación de técnicas de aprendizaje automático (machine learning)., donde el entrenamiento de modelos con conjuntos de datos de imágenes permitirá al sistema aprender y adaptarse a diversas condiciones de iluminación y escenarios, mejorando la precisión en la detección de movimiento y reduciendo falsos positivos. Esta combinación de estrategias proporciona un enfoque completo y escalable para un sistema de seguridad residencial inteligente basado en IoT.

Capítulo 3

3. Resultados y análisis

En este capítulo, se exponen los resultados derivados de evaluar el rendimiento del prototipo a través de un análisis minucioso que incluye aspectos como el tiempo de retardo en el procesamiento para el envío de notificaciones, la duración de la batería, costos, entre otros. Estos resultados proporcionan una visión clara y detallada del funcionamiento del sistema, permitiendo una evaluación integral de su eficacia y eficiencia.

3.1 Pruebas y validación

En esta fase del proyecto, se elabora la interfaz entre el sistema de seguridad residencial basado en IoT y la plataforma de mensajería Telegram. Como se indicó, la selección estratégica de Telegram como canal de comunicación proporciona una base sólida para la transmisión eficiente de alertas y notificaciones en tiempo real, respaldada por su robusto protocolo HTTPS para garantizar un nivel de seguridad elevado.

Los pasos iniciales incluyen la configuración y creación de un Bot de Telegram, estableciendo así la conexión fundamental entre el sistema y la plataforma de mensajería. Posteriormente, se lleva a cabo la implementación detallada de comandos específicos y se aborda el manejo efectivo de mensajes, asegurando una integración fluida y funcional entre el sistema IoT y Telegram.

3.1.1 Prototipo del proyecto

Se exhibe el prototipo correspondiente a la fase inicial del proyecto. En la Figura 3.1, se aprecia la disposición del prototipo en una maqueta junto con un protoboard de dimensiones reducidas. La placa ESP32-CAM se encuentra instalada en la parte lateral de la maqueta como se puede observar en la figura 3.2, junto con los sensores debidamente conectados y con la implementación final del código cargado en la placa (Apéndice A). A continuación, se muestra la figura 3.1, que ilustra el estado del prototipo en esta etapa del desarrollo.

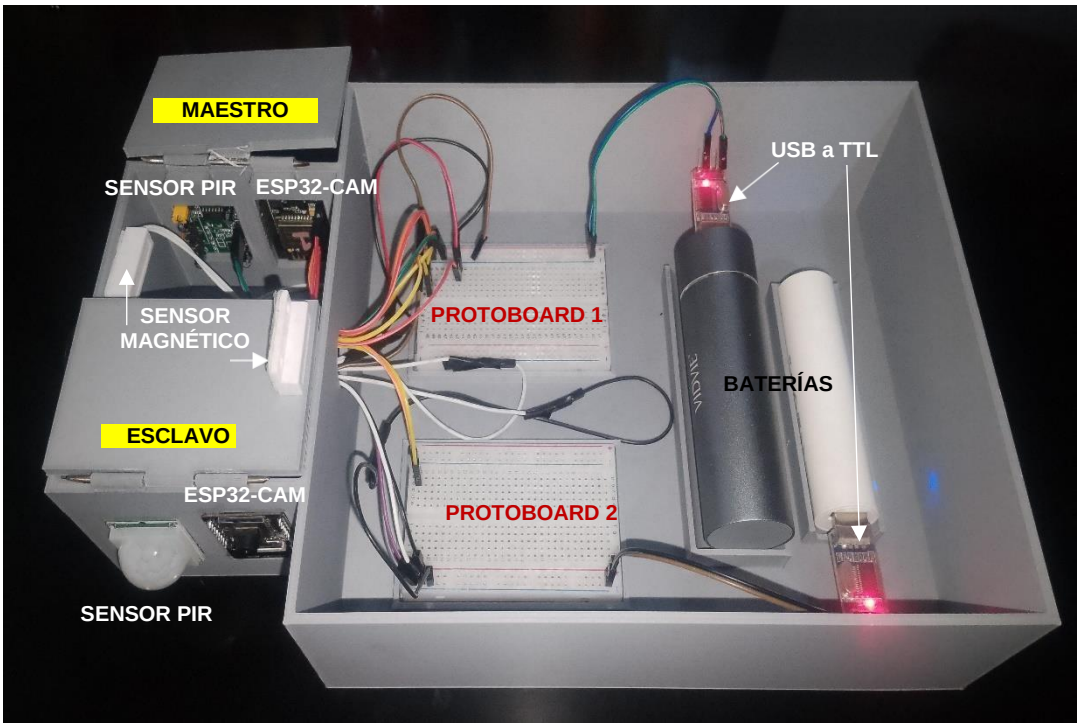

Figura 3.1 *Implementación del sistema IoT (Vista Frontal)*

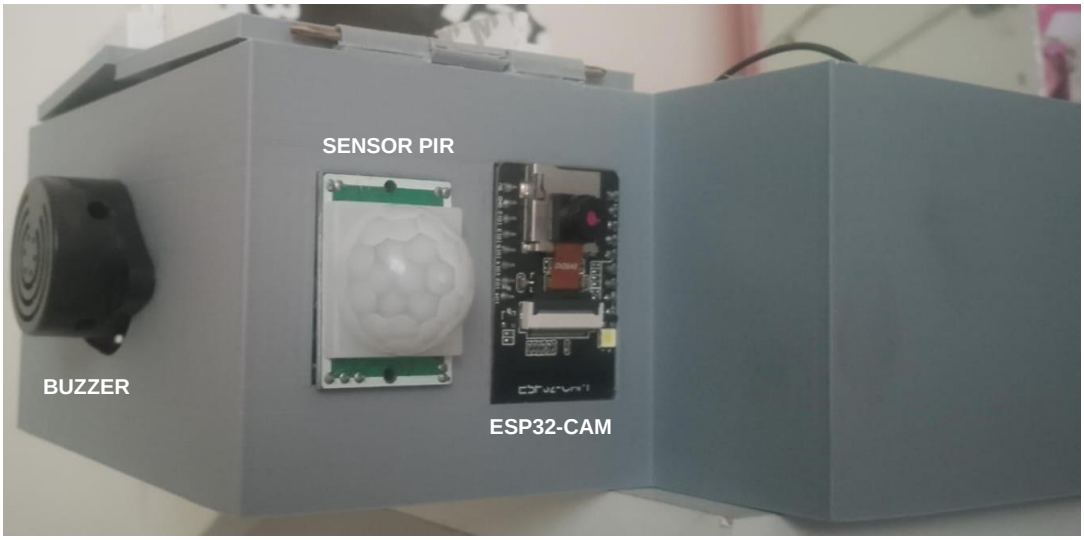

Figura 3.2 *Implementación del sistema IoT (Vista Lateral)*

Nota: Las Figuras 3.1 y 3.2 exhiben la implementación del sistema IoT, detallando las conexiones entre los sensores y la alarma correspondientes a cada módulo.

Además de las especificaciones técnicas, es necesario por parte del usuario conocer cómo comunicarse con el sistema; esto se logra a través de comandos que se envían por mensaje de texto directamente en el Bot de Telegram como se muestra en la tabla 3.1.

ítem	Comando	Descripción del comando
1	/cam1	Toma una foto de la Cámara exterior
2	/cam2	Toma una foto de la Cámara interior
3	/flash1	Activa/Desactiva el flash del módulo exterior
4	/flash2	Activa/Desactiva el flash del módulo interior
5	/alarmaon	Activa la alarma sonora
6	/alarmaoff	Desactiva la alarma sonora
7	/alarmstatus	Muestra el estado de la alarma
8	/off	Apaga el sistema de notificaciones
9	/on	Enciende el sistema de notificaciones
10	/start	Da inicio al Bot o muestra un menú de comandos

Tabla 3.1 *Tabla de comandos para el control del sistema*

Nota. En la tabla se detallan los comandos del sistema IoT con sus debidas descripciones, los cuales se ejecutan a través de la mensajería de Telegram.

3.1.2 Retardo en procesamiento y envío de notificaciones

Durante la fase experimental de este proyecto de seguridad IoT con Telegram, se realizaron 1000 pruebas para evaluar el impacto del retardo en el procesamiento y envío de datos. Cada prueba consistió en simular condiciones diversas, incluyendo situaciones de tráfico de red variable y distancia en la que se encuentra el Wi-Fi tomando como dato el nivel de potencia del dispositivo Wi-Fi. Durante cada toma de prueba, se registraron los tiempos de respuesta del sistema en relación con el envío de datos a través de la plataforma Telegram. La variabilidad en los resultados permitió identificar patrones y determinar la robustez del sistema. Estos datos experimentales ofrecen una visión detallada

del rendimiento del sistema en términos de velocidad y eficiencia en la transmisión de alertas, información crucial para optimizar el diseño y la implementación del proyecto en entornos del mundo real. En la Figura 3.3 se presenta el gráfico de tiempo y la señal aproximada que el sistema toma desde la solicitud del usuario o activación de un sensor hasta la notificación en el Telegram, como es de esperarse a medida que nosotros nos alejemos del Wi-Fi la señal se hace más pobre el cual hará que tarde la comunicación con la API de Telegram haciendo un mayor retardo.

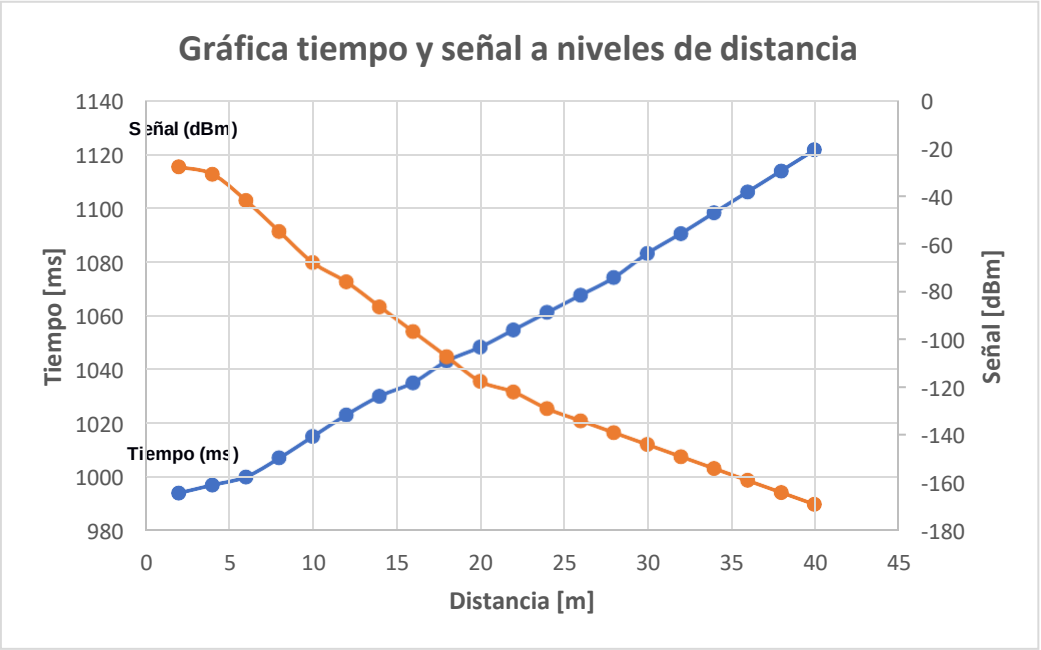

Figura 3.3 *Gráfica de las pruebas obtenidas en el delay*

Nota. En la gráfica se muestra el tiempo de retardo desde la solicitud del usuario hasta la notificación en Telegram a diferentes distancias medidas desde el dispositivo Wi-Fi.

Para interactuar con el sistema de vigilancia se procede a utilizar la aplicación Telegram, usando comandos como los mostrados en la tabla 3.1 los usuarios cuentan con la flexibilidad de navegar con el propósito de visualizar las cámaras, activar la alarma o desactivar las notificaciones según sus necesidades y preferencias. En la figura 3.4 en la parte C se muestra un ejemplo utilizando el comando /cam1 y /cam2 donde podemos ver las notificaciones con las fotos adjuntas en cada mensaje de la cámara 1 que va en la parte delantera de la residencia y cámara 2 que va adentro de la residencia, además en la parte A y B respectivamente también se puede observar las notificaciones de detección de

movimiento dentro y fuera de la casa y las notificaciones de cuando se abren las puertas de la residencia.

Figura 3.4 *Notificaciones en la aplicación de Telegram*

Nota. En la figura se muestran los mensajes recibidos de acuerdo con los comandos enviados por el usuario.

En la figura 3.5 se puede observar la interacción entre los sensores de movimiento y magnéticos. Si se abre una puerta del domicilio (representado como la parte A de la maqueta), se activa el sensor magnético enviando así una notificación al Telegram del usuario como se ve en la figura 3.4 informando de tal evento y que hay movimiento en ese lado del domicilio; lo mismo sucede con apertura de puertas en otro lado del domicilio representado por el lado B de la maqueta.

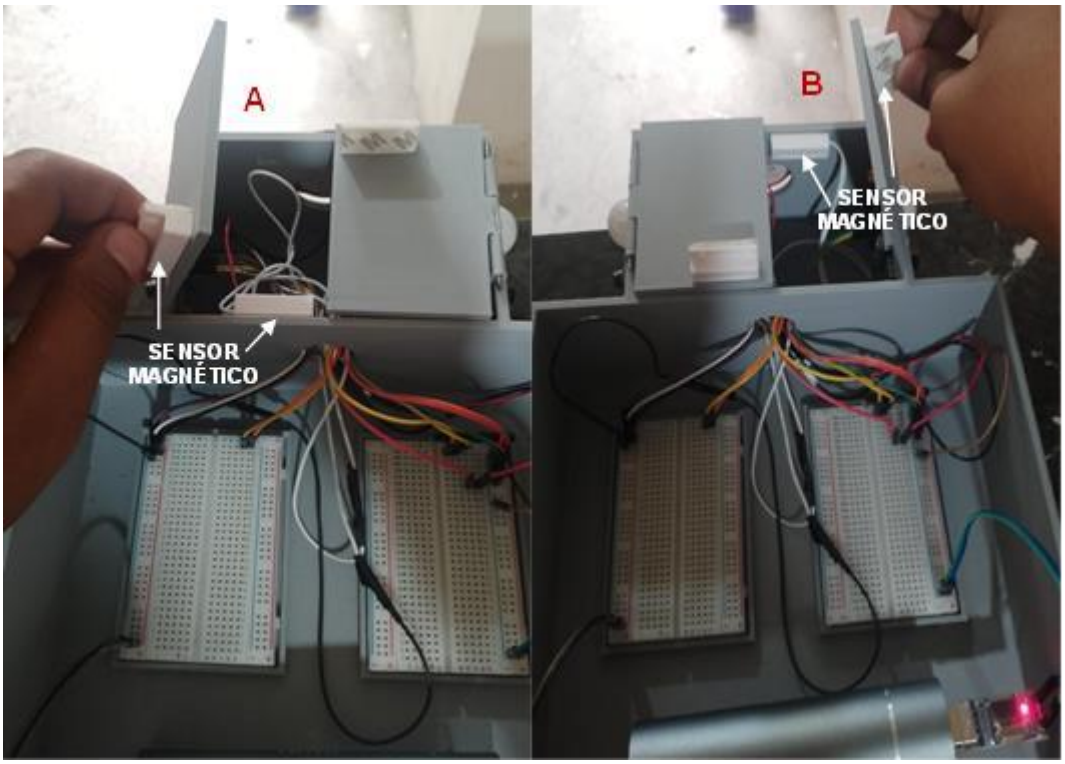

Figura 3.5 *Activando los sensores de movimiento y de puertas*

Nota. En la figura se muestran la interacción entre los sensores de movimiento y magnéticos del sistema.

Para poder conseguir los resultados de retardo utilizando el IDE de Arduino, hemos utilizado un código para obtener el tiempo de retardo en cada proceso de captura de foto. La función principal utilizada se llama millis(). Como se puede observar en la figura 3.6, esta función devuelve el número de milisegundos desde que el programa inició o desde que se reinició el dispositivo; en este caso toma el tiempo desde que se envía el comando para la captura de foto hasta que la foto es recibida en el dispositivo. Cabe recalcar que existen muchos factores para el retardo de las notificaciones siendo el principal la saturación de la API de Telegram.

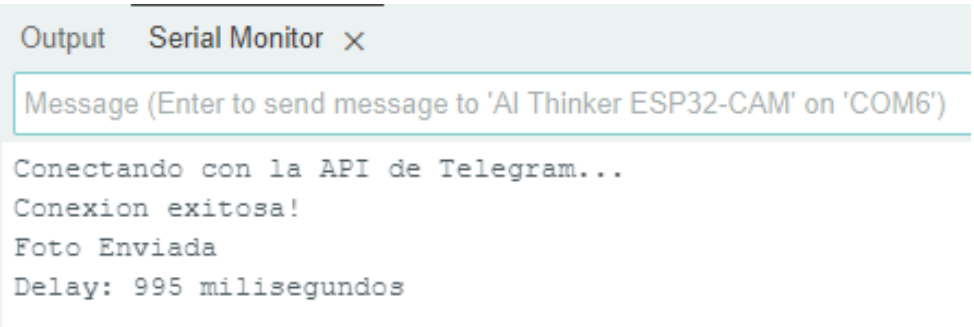

Figura 3.6 *Prueba de retardo de envío de foto a Telegram con resolución UXGA*

Nota. En la figura se muestran el tiempo que tarda el envío de la foto a la aplicación Telegram utilizando el serial monitor del IDE de Arduino.

3.1.3 Autonomía del sistema

El consumo de energía de un módulo ESP32-CAM puede variar según varios factores, como la configuración específica, la frecuencia de operación, la activación de determinadas funciones y la carga de trabajo. En condiciones típicas de uso, el ESP32-CAM consume alrededor de 180 mA durante la transmisión de datos o la captura de imágenes. Para determinar cuánto tiempo de autonomía tendría este sistema con una batería de 3000 mAh y que consume de 180 mA, se utiliza en la ecuación (3.1):

$$Timepo\ de\ duración\ (en\ horas) = \frac{Capacidad\ de\ la\ batería\ (mAh)}{Consumo\ del\ dispositivo\ (mA)} \quad \textbf{(3.1)}$$

En este caso:

$$Tiempo\ de\ duración = \frac{3000mAh}{180mA} = 16.67\ horas$$

Esto significa que, en teoría y bajo las condiciones específicas mencionadas, la batería de 3000 mAh alimentaría el ESP32-CAM durante aproximadamente 16.67 horas. En la parte práctica, realizando pruebas de autonomía durante 5 días, se ha logrado determinar las horas promedio en las cuales nuestro sistema operaría si la residencia careciera de suministro eléctrico, tal como se ilustra en la figura 3.7.

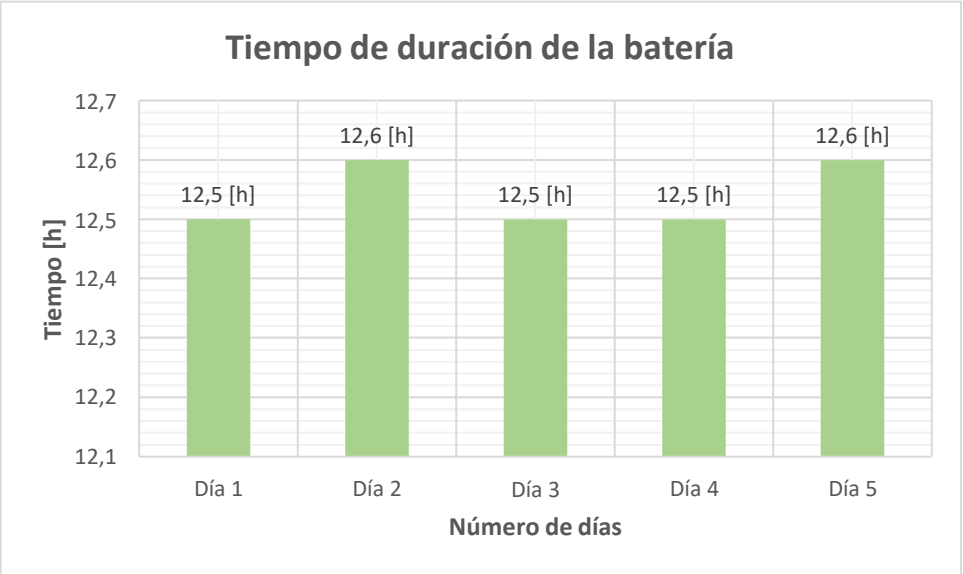

Figura 3.7 *Gráfica de duración de la batería*

Nota. En la gráfica se detallan las pruebas realizadas durante los 5 días, especificando la duración de la batería ante cortes de energía eléctrica**.**

El análisis de autonomía de la batería revela una discrepancia entre la duración teórica y la observada en la práctica. Teóricamente, se estimó que la batería de 3000 mAh alimentaría el sistema durante aproximadamente 16 horas, basándose en el consumo previsto del ESP32-CAM y otros componentes. Sin embargo, en la práctica, la duración real se reduce a 12 horas. Este descenso puede atribuirse a factores como pérdidas de eficiencia, variaciones en las condiciones de uso y la presencia de otros posibles consumidores de energía. Este análisis subraya la importancia de considerar factores prácticos y condiciones del mundo real al planificar la autonomía de la batería, ofreciendo una visión más precisa para la gestión eficaz de la energía en el sistema de seguridad IoT.

3.2 Análisis de Costos

Con el fin de obtener comparaciones entre resultados, empleamos los procedimientos que se presentan en la tabla 3.2, la cual detalla los costos.

Cantidad	Nombre	Precio	Total
2	ESP32-CAM	$ 22,00	$ 44,00
1	Buzzer	$ 1,00	$ 1,00
2	Sensores PIR	$ 2,25	$ 4,50
2	Sensores magnéticos	$ 2,50	$ 5,00
2	Protoboard	$ 2,50	$ 5,00
2	Regulador de voltaje	$ 3,50	$ 7,00
2	Baterías recargables	$ 10,00	$ 20,00
	Total		$ 86,50

Tabla 3.2 *Tabla de costo de los elementos del sistema*

Nota. La tabla proporciona detalles sobre los costos de los elementos empleados en el desarrollo del sistema IoT.

En el análisis de costos de este proyecto de seguridad IoT, se han considerado diversos elementos financieros que abarcan la adquisición de hardware y sensores. Se ha evaluado el costo de los componentes electrónicos esenciales, como la placa ESP32-CAM ylos sensores utilizados. Es crucial considerar tanto los costos iniciales como los recurrentes para Obtener una comprensión integral de la inversión requerida en la implementación y mantenimiento del sistema de seguridad IoT. Realizando una breve comparativa entre el costo de implementación de este sistema IoT y los sistemas tradicionales disponibles en el mercado, se puede afirmar que constituye una opción económicamente favorable para establecer un sistema de seguridad en el hogar.

Capítulo 4

4. Conclusiones y recomendaciones

Se presentan las conclusiones y recomendaciones definitivas del proyecto, abordando las interpretaciones generales de los datos recolectados durante las pruebas.

4.1 Conclusiones

- Al utilizar el uso del procesador ESP32-CAM en el control remoto de un sistema de seguridad a través de Telegram, se observó que el uso de esta tecnología para este proyecto es eficaz y accesible desde cualquier ubicación.
- Los resultados obtenidos al evaluar el rendimiento del sistema IoT, tomando datos como resoluciones y tiempo de activación eficiente de los sensores, junto con una notificación instantánea a través de Telegram demuestra un sistema que garantiza una respuesta inmediata ante cualquier evento de seguridad.
- La evaluación de la autonomía de la batería revela una diferencia notable entre la estimación teórica de 16 horas y la duración efectiva de 12 horas promedio ya que hay muchas condiciones que se deben tomar en cuanta como la variabilidad en la potencia de la señal de red, la posible saturación de los servidores de Telegram y los consumos de potencia del sistema distintos a los calculados teóricamente.
- La utilización del protocolo TCP/IP estableció una comunicación segura entre el módulo ESP32-CAM y la plataforma Telegram a través de la red Wi-Fi, donde la fragmentación de datos en paquetes ha asegurado la integridad de los mensajes, destacando su papel crucial en la fiabilidad y estabilidad de la transmisión de datos.

4.2 Recomendaciones

- Adoptar un enfoque modular en el diseño del sistema, permitiendo la fácil incorporación de nuevos dispositivos, como otros módulos ESP32-CAM, sensores de movimiento y sensores magnéticos, sin alterar la estructura central del código. Esto facilitará futuras expansiones y garantizará la escalabilidad del sistema.
- Establecer una base de datos destinada al almacenamiento de información, como eventos de detección de movimiento e imágenes capturadas, con el propósito de ofrecer una solución resistente y duradera.

- La utilización de servicios en la nube permitirá flexibilidad en el acceso y respaldo de datos, asegurando su integridad a largo plazo.
- Realizar actualizaciones remotas de firmware del módulo ESP32-CAM para mantener el sistema eficiente sin necesidad de acceso físico a cada dispositivo. Esta característica contribuirá significativamente a la gestión continua y mejoras del sistema, garantizando su funcionalidad a largo plazo.
- Incorporar técnicas de Machine Learning al sistema para una detección automática de patrones y reducción de falsos positivos.

Referencias

[1] «Los 6 daños psicológicos y emocionales más comunes tras un robo, según Tecnicom,» *Comunicae,* 22 Mayo 2018.

[2] «Fiscalía General del Estado,» 08 Diciembre 2021. [En línea]. Available: https://www.fiscalia.gob.ec/estadisticas-de-robos/. [Último acceso: 18 Octubre 2023].

[3] B. C. J. Mera, «DISEÑO DE UN PROTOTIPO INTELIGENTE DE SEGURIDAD CON TECNOLOGÍA OPEN SOURCE PARA VIVIENDAS EN EL SECTOR DE LA COOPERATIVA SERGIO TORAL,» Guayaquil-Ecuador, 2022.

[4] C. Angulo, «DESARROLLO DE UN SISTEMA DE CONTROL DE ÁNGULOS DE CÁMARAS IP MEDIANTE IOT PARA EL MONITOREO EN TIEMPO REAL DE UNA VIVIENDA,» Latacunga-Ecuador, 2023.

[5] V. Sanchez, «¿Son efectivas las cámaras de video vigilancia para reducirlos delitos?,» *URVIO,* 2018.

[6] R. H. Weber, «Internet of Things - New Security and Privacy Challenges,» *Computer Law & Security,* vol. 26, nº 1, pp. 23-30, 2010.

[7] P. Lucena, «Ventajas y desventajas del internet de las cosas (IOT),» U. Cesuma, s.f.. [En línea]. Available: https://www.cesuma.mx/blog/ventajas-y-desventajas-del-internet-de-las-cosas-iot.html. [Último acceso: 14 Noviembre 2023].

[8] «Hardware - Nicla family,» s.f.. [En línea]. Available: https://www.arduino.cc/pro/hardware-nicla-family/. [Último acceso: 14 Noviembre 2023].

[9] NOVATRONIC. [En línea]. Available: https://novatronicec.com/index.php/product/esp32cam-wifi-bluetooth-incluye-camara-ov2640/.

[10] L. Llamas, «Pinout y detalles del hardware del ESP32,» 18 Agosto 2023. [En línea]. Available: https://www.luisllamas.es/esp32-detalles-hardware-pinout/.

[11] Z. Peterson, «Altium,» 26 Septiembre 2021. [En línea]. Available: https://resources.altium.com/es/p/i2c-vs-spi-vs-uart-how-layout-these-common-buses.

[12] R. Laddha, «A Review on Serial Communication by UART,» *Advanced Research in Computer Science and Software Engineering,* vol. 3, nº 1, pp. 366-369, 2013.

[13] D. Carrasco, «Electrosoftcloud,» 14 Mayo 2021. [En línea]. Available: https://www.electrosoftcloud.com/esp-now-conecta-dos-o-mas-esp32-esp8266/.

[14] D. No, «Conexión ESP-NOW,» 20 Noviembre 2020. [En línea]. Available: https://www.esploradores.com/practica-6-conexion-esp-now/.

[15] Telegram, «Preguntas de Telegram,» s.f.. [En línea]. Available: https://telegram.org/faq/es#p-que-es-telegram-que-puedo-hacer-aqui.

[16] G. Global, «Cómo usar Telegram - ¿Qué es Telegram?,» s.f.. [En línea]. Available: https://edu.gcfglobal.org/es/curso-de-telegram/que-es-telegram/1/#.

[17] A. Robledano, «TCP/IP,» OpenWebinars, 18 Junio 2019. [En línea]. Available: https://openwebinars.net/blog/que-es-tcpip/.

[18] D. Bodnar, «¿Cómo funciona el modelo TCP/IP?,» AVG, 4 Junio 2021. [En línea]. Available: https://www.avg.com/es/signal/what-is-tcp-ip.

[19] SSL, «¿Qué es HTTPS?,» 12 Octubre 2021. [En línea]. Available: https://www.ssl.com/es/preguntas-frecuentes/que-es-https/.

[20] Cloudflare, «¿Por qué es importante HTTPS?,» s.f.. [En línea]. Available: https://www.cloudflare.com/es-es/learning/ssl/what-is-https/#:~:text=El%20protocolo%20de%20transferencia%20de,de%20las%20transferencias%20de%20datos..

[21] R. Schwarz, «Entendiendo el UART,» R&S®Essentials, s.f.. [En línea]. Available: https://www.rohde-schwarz.com/lat/productos/prueba-y-medicion/essentials-test-equipment/digital-oscilloscopes/entendiendo-el-uart_254524.html.

[22] Xukyo, «Comunicarse con Arduino,» AranaCorp, 23 Septiembre 2019. [En línea]. Available: https://www.aranacorp.com/es/comunicarse-con-arduino/.

[23] H. U. J. Espinoza, «Sistema de riego automatizado para el cultivo de arroz en Santa Lucía,» Guayaquil-Ecuador, 2023.

[24] F. Mahedero, «DESARROLLO DE UNA APLICACIÓN IoT PARA EL ENVÍO DE IMÁGENES MEDIANTE EL PROTOCOLO MQTT,» Valencia-España, 2020.

[25] Github, 24 Octubre 2023. [En línea]. Available: https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFiClientSecure.

Apéndices

Apéndice A. *Código del dispositivo Maestro*

```
// Bibliotecas
#include <UniversalTelegramBot.h>

#include <esp_wifi.h>

#include <ArduinoJson.h>

#include "soc/soc.h"

#include <WiFi.h>

#include <WiFiClientSecure.h>

#include "soc/rtc_cntl_reg.h"

#include <esp_camera.h>

#include <Arduino.h>

#include <esp_now.h>

uint8_t broadcastAddress[] = {0xC8, 0xF0, 0x9E, 0xEA, 0xD4, 0x88};

int flashValue;
int camValue;
int buzzerValue;
int magneticValue;
int pirValue;

int incomingMagneticValue;
int incomingPirValue;
int incomingIsPhotoValue;
size_t incomingPhotoSizeValue;
uint8_t *incomingPhotoDataValue = nullptr;

String successValue;

typedef struct struct_message_action
{
  bool flash = false;
  bool cam = false;
  bool buzzer = false;
} ActionMessage;

typedef struct struct_message_sensor
{
  int magnetic;
  int pir;
```

```
  int imageSize;
} SensorMessage;

// Creamos un struct_message para las lecturas de los sensores
ActionMessage deviceActionsValue;

// Creamos un struct_message para recibir las lecturas del equipo 2
SensorMessage incomingReadingsValue;

esp_now_peer_info_t peerInfoValue;

// Credenciales WiFi
const char *ssidValue = "SpeedNet-Jimenez";
const char *passwordValue = "marv3l123";

constexpr char WIFI_SSID[] = "SpeedNet-Jimenez";
constexpr char WIFI_PASS[] = "marv3l123";

// Telegram bot token
String BOTtokenValue = "6985309105:AAEyn73ZOMeHw7QGSpzt4mkMhhWle4VvdzQ";
String CHAT_IDValue = "1685390701";

// Otras variables
int botRequestDelayValue = 1000;
int botLongRequestDelayValue = 10000;

unsigned long lastTimeBotRanValue;
unsigned long lastTimeBotSendValue;

// Variables globales
uint8_t *imageDataBufferValue = nullptr;
int imageSizeValue = 0;
int receivedDataSizeValue = 0;
int packetSizeValue = 240;

// Pines
#define FLASH_LED_PIN 4
#define PIR_PIN 13
#define MAGNETIC_PIN 15
#define BUZZER_PIN 12

// Estados
bool buzzerStatusValue = false;
bool buzzer2StatusValue = false;
bool alertStatusValue = true;
bool sendPhotoValue = false;
bool sendPhoto2Value = false;
bool flashStateValue = LOW;
bool flash2StateValue = LOW;
bool alertMovStatus1Value = true;
bool alertMovStatus2Value = true;
```

```
bool magneticStatusSend1Value = false;
bool magneticStatusSend2Value = false;

// Objects
WiFiClientSecure clientTCPValue;
UniversalTelegramBot botValue(BOTtokenValue, clientTCPValue);


// Mensajes
const String buzzerOnMsgValue = "Alarma ON";
const String buzzerOffMsgValue = "Alarma OFF";
const String pirONMsgValue = "Se detecta movimiento AFUERA de la casa";
const String pir2ONMsgValue = "Se detecta movimiento ADENTRO de la casa";
const String magneticONMsgValue = "Puerta DELANTERA abierta";
const String magneticOFFMsgValue = "Puerta DELANTERA cerrada";
const String magnetic2ONMsgValue = "Puerta TRASERA abierta";
const String magnetic2OFFMsgValue = "Puerta TRASERA cerrada";
const String alertONMsgValue = "Notificaciones ON";
const String alertOffMsgValue = "Notificaciones OFF";
const String errorMsgValue = "Comando Incorrecto";
const String alertOnPirMsg1Value = "Notificaciones PIR1 ON";
const String alertOffPirMsg1Value = "Notificaciones PIR1 OFF";

const String alertOnPirMsg2Value = "Notificaciones PIR2 ON";
const String alertOffPirMsg2Value = "Notificaciones PIR2 OFF";

// Sirve para obtener el canal del wifi para espnow
int32_t getWiFiChannelValue(const char *ssid)
{
  if (int32_t n = WiFi.scanNetworks())
  {
    for (uint8_t i = 0; i < n; i++)
    {
      if (!strcmp(ssid, WiFi.SSID(i).c_str()))
      {
        return WiFi.channel(i);
      }
    }
  }
  return 0;
}

// Callback cuando la data es enviada
void OnDataSentValue(const uint8_t *mac_addr, esp_now_send_status_t status)
{
  Serial.print("\r\nEstado del ultimo paquete enviado\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Envio exitoso" : "Envio fallido");
  if (status == 0)
  {
    successValue = "Envio exitoso :)";
  }
  else
```

```
  {
    successValue = "Envio fallido :(";
  }
}

// Callback cuando la data es recibida
void OnDataRecvValue(const uint8_t *mac, const uint8_t *incomingData, int len)
{
  Serial.println("Obteniendo datos...");
  Serial.print("Tamaño de incomingData: ");
  Serial.println(sizeof(incomingData));
  Serial.print("Tamaño de incomingReadings: ");
  Serial.println(sizeof(incomingReadingsValue));
  Serial.print("Tamaño de len: ");
  Serial.println(len);
  // Identificar si los datos son de los sensores o de la imagen
  if (len == sizeof(incomingReadingsValue))
  {
    //  Datos de los sensores
    Serial.println("SENSOR DATA");
    processSensorDataValue(incomingData, len);
  }
  else
  {
    // Paquete debe ser máximo hasta 240 bytes
    Serial.println("IMAGEN DATA");
    // Datos de la imagen
    processImageDataValue(incomingData, len);
  }
}
// Función para procesar los datos de los sensores recibidos
void processSensorDataValue(const uint8_t *incomingData, int len)
{
  Serial.println("Procesando data del sensor");

  // Copiamos los datos recibidos a la estructura
  memcpy(&incomingReadingsValue, incomingData, sizeof(incomingReadingsValue));
  Serial.print("Bytes recibidos: ");
  Serial.println(len);
  incomingMagneticValue = incomingReadingsValue.magnetic;
  incomingPirValue = incomingReadingsValue.pir;
  imageSizeValue = incomingReadingsValue.imageSize;

  if (imageSizeValue > 0)
  {
    // Asignar memoria para imageDataBuffer
    Serial.println("Reasignando memoria");
    imageDataBufferValue = new uint8_t[imageSizeValue];
    if (imageDataBufferValue == nullptr)
    {
      Serial.println("Error al asignar memoria para imageDataBuffer");
      // Manejar el error adecuadamente
```

```
      return;
    }
  }
}

// Función para procesar los datos de la imagen recibidos
void processImageDataValue(const uint8_t *data, int len)
{
  Serial.println("IncomingPhotoData");
  for (int i = 0; i < 20; i++)
  {
    Serial.print(data[i], HEX);
    Serial.print(" ");
  }
  Serial.println();

  Serial.print("receivedDataSize: ");
  Serial.println(receivedDataSizeValue);
  Serial.print("imageSize: ");
  Serial.println(imageSizeValue);

  // Asegúrate de que los datos recibidos quepan en el búfer
  if (receivedDataSizeValue + len <= imageSizeValue)
  {
    Serial.println();
    Serial.println("PAQUETE RECIBIDO");
    Serial.println();

    // Copia los datos recibidos al búfer de la imagen
    memcpy(imageDataBufferValue + receivedDataSizeValue, data, len);
    receivedDataSizeValue += len;

    // Verifica si se ha recibido la imagen completa
    if (receivedDataSizeValue == imageSizeValue)
    {
      Serial.println();
      Serial.println();
      Serial.println("----------Imagen recibida completamente.----------");
      Serial.println();
      Serial.println();
      Serial.println();

      sendPhoto2Value = true;

      // Reinicializa las variables para la próxima imagen
      receivedDataSizeValue = 0;
    }
  }
  else
  {
    Serial.println("Error: Los datos recibidos exceden el tamaño esperado.");
    // Puedes implementar manejo de errores aquí, por ejemplo, reiniciar la recepción de la imagen
```

```
    receivedDataSizeValue = 0;
  }
}

void setup()
{
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);

  // Iniciamos el monitor serial e imprimimos una línea vacía
  Serial.begin(115200);
  Serial.println();

  // Seteamos entradas y salidas
  pinMode(FLASH_LED_PIN, OUTPUT);
  pinMode(PIR_PIN, INPUT);
  pinMode(MAGNETIC_PIN, INPUT);
  pinMode(BUZZER_PIN, OUTPUT);

  // Seteamos flash de acuerdo a estado inicial
  digitalWrite(FLASH_LED_PIN, flashStateValue);

  // Un pequeño delay para sincronizar todo
  delay(30);

  // Configuramos e iniciamos la cámara
  configInitCamera();

  // Connect to Wi-Fi
  // WiFi.mode(WIFI_STA);
  WiFi.mode(WIFI_AP_STA);
  // WiFi.mode(WIFI_STA);
  int32_t channel = getWiFiChannelValue(ssidValue);
  esp_wifi_set_channel(channel, WIFI_SECOND_CHAN_NONE);

  Serial.println();

  // Iniciar WiFi
  initWifi();

  // Iniciar ESP-NOW
  initESPNOW();

  Serial.println("Setup completo");
}

void loop()
{
  // Si se detecta foto cam1
  if (sendPhotoValue)
  {
    Serial.println("Preparando foto");
    sendPhotoTelegram();
```

```
    sendPhotoValue = false;
  }

  // Si se detecta foto cam1
  if (sendPhoto2Value)
  {
    Serial.println("Preparando foto 2");
    sendPhoto2Telegram();
  }

  // Si alarma esta activadada
  if (buzzerStatusValue)
  {
    digitalWrite(BUZZER_PIN, HIGH);
    delay(250);
    digitalWrite(BUZZER_PIN, LOW);
    delay(250);
    Serial.println("Alarma activada");
  }
}

// Enviando datos a telegram
if (millis() > lastTimeBotSendValue + botRequestDelayValue)
{
    Serial.println("Enviando datos a telegram");
    // Cuando detecte movimiento en PIR del equipo 1 envía una alerta
    int pirValueLocal = digitalRead(PIR_PIN);
    if (pirValueLocal == HIGH && alertMovStatus1Value)
    {
        sendMessage(CHAT_IDValue, pirONMsgValue);
    }

    if (incomingPirValue == HIGH && alertMovStatus2Value)
    {
        sendMessage(CHAT_IDValue, pir2ONMsgValue);
        incomingPirValue = incomingReadingsValue.pir = LOW;
    }
}

int magneticValueLocal = digitalRead(MAGNETIC_PIN);
if (magneticValueLocal == HIGH && !magneticStatusSend1Value)
{
    sendMessage(CHAT_IDValue, magneticONMsgValue);
    magneticStatusSend1Value = !magneticStatusSend1Value;
}
else if (magneticValueLocal == false && magneticStatusSend1Value)
{
    sendMessage(CHAT_IDValue, magneticOFFMsgValue);
    magneticStatusSend1Value = !magneticStatusSend1Value;
}

if (incomingMagneticValue == HIGH && !magneticStatusSend2Value)
```

```
{
    sendMessage(CHAT_IDValue, magnetic2ONMsgValue);
    incomingMagneticValue = incomingReadingsValue.magnetic = LOW;
    magneticStatusSend2Value = !magneticStatusSend2Value;
}
else if (incomingMagneticValue == false && magneticStatusSend2Value)
{
    sendMessage(CHAT_IDValue, magnetic2OFFMsgValue);
    magneticStatusSend2Value = !magneticStatusSend2Value;
}

// Recibiendo datos de telegram
if (millis() > lastTimeBotRanValue + botRequestDelayValue)
{
    Serial.println("Recibiendo datos de telegram");
    int numNewMessages = botValue.getUpdates(botValue.last_message_received + 1);
    while (numNewMessages)
    {
        Serial.println("Respuesta obtenida!");
        handleNewMessages(numNewMessages);
        numNewMessages = botValue.getUpdates(botValue.last_message_received + 1);
    }
    lastTimeBotRanValue = millis();
}

updateDisplay();
delay(2000);
}

void sendMessage(String id, String message)
{
    if (alertStatusValue)
    {
        botValue.sendMessage(id, message, "");
        lastTimeBotSendValue = millis();
    }
}

void sendToAnotherDev()
{
    Serial.println("Enviando mensaje a otro dispositivo");
    // Enviamos el mensaje a través del protocolo ESP-NOW
    esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *)&deviceActionsValue,
sizeof(deviceActionsValue));

    if (result == ESP_OK)
    {
        Serial.println("Envío del mensaje exitoso");
    }
    else
    {
        Serial.println("Envío del mensaje fallido");
```

```
    }
}
```

Apéndice B. *Código del dispositivo Esclavo*

```
// Bibliotecas
#include <UniversalTelegramBot.h>
#include <esp_wifi.h>
#include <ArduinoJson.h>
#include "soc/soc.h"
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include "soc/rtc_cntl_reg.h"
#include <esp_camera.h>
#include <Arduino.h>
#include <esp_now.h>

uint8_t espNowBroadcastAddress[] = {0xB8, 0xD6, 0x1A, 0x67, 0xD8, 0x58};

int flashState;
int cameraState;
int buzzerState;
int magneticSensor;
int pirSensor;

uint8_t *imageDataBuffer;   // Puntero al búfer de imagen
int imageSize = 0;          // Tamaño total del búfer de imagen
int packetSize = 240;       // Tamaño máximo de cada paquete

String transmissionResult;

typedef struct struct_message_action
{
  bool flash = false;
  bool camera = false;
  bool buzzer = false;

} MessageActionStruct;

MessageActionStruct deviceActions;

// Token y ID del bot de Telegram
String BOTtoken = "6985309105:AAEyn73ZOMeHw7QGSpzt4mkMhhWle4VvdzQ";
String CHAT_ID = "1685390701";

int botRequestDelay = 1000;
unsigned long lastTimeBotRanValue;

// Definición de pines
```

```
#define FLASH_LED_PIN 4
#define PIR_PIN 13
#define MAGNETIC_PIN 15
#define BUZZER_PIN 12

// Estados
bool buzzerStatus = false;
bool flashState = LOW;

WiFiClientSecure clientTCP;
UniversalTelegramBot botValue(BOTtoken, clientTCP);

#define fileDatainMessage 240.0
#define UARTWAITHANDSHACK 1000

// Función para obtener el canal de WiFi
int32_t getWiFiChannel(const char *ssid)
{
  if (int32_t n = WiFi.scanNetworks())
  {
    for (uint8_t i = 0; i < n; i++)
    {
      if (!strcmp(ssid, WiFi.SSID(i).c_str()))
      {
        return WiFi.channel(i);
      }
    }
  }
  return 0;
}

void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
{
  Serial.print("\r\nEstado del último paquete enviado\t");
  Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Envío exitoso" : "Envío fallido");
  if (status == 0)
  {
    transmissionResult = "Envío exitoso :)";
  }
  else
  {
    transmissionResult = "Envío fallido :(";
  }
}

void OnDataRecv(const uint8_t *mac, const uint8_t *incomingData, int len)
{
  Serial.println("Recibiendo datos...");
  memcpy(&deviceActions, incomingData, sizeof(deviceActions));
  Serial.print("Bytes recibidos: ");
  Serial.println(len);
  buzzerState = deviceActions.buzzer;
```

```
  cameraState = deviceActions.camera;
  flashState = deviceActions.flash;
}

void setup()
{
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);

  delay(30);

  configInitCamera();
  initWifi();
  initESPNOW();

  Serial.println("Configuración completa");
}

void loop()
{
  pirSensor = digitalRead(PIR_PIN);
  magneticSensor = digitalRead(MAGNETIC_PIN);

  if (pirSensor == HIGH || magneticSensor == HIGH)
  {
    Serial.println("Enviando datos de sensores");
    sendSensorData(0);
  }

  if (cameraeStado != flasheStado)
  {
    flasheStado = cameraeStado;
    digitalWrite(FLASH_LED_PIN, flasheStado);
  }

  if (buzzereeStado != buzzereStado)
  {
    buzzereStado = buzzerStado;
    digitalWrite(BUZZER_PIN, buzzerStatus);
  }

  updateDisplay();
  delay(500);
}

if (cameraState == HIGH)
{
  cameraState = LOW;
  deviceActions.camera = cameraState;
  // Serial.println("Inicia proceso de foto");
  // sendImage();

  sendPhotoTelegram();
```

```
}
if (buzzer != buzzereStado)
{
  buzzereStado = buzzer;
  digitalWrite(BUZZER_PIN, buzzerStatus);
}

updateDisplay();
delay(500);
}

  if (psramFound())
  {
    cameraConfig.frame_size = FRAMESIZE_UXGA;
    cameraConfig.jpeg_quality = 10;
    // cameraConfig.jpeg_quality = 5;
    cameraConfig.fb_count = 1;
  }
  else
  {
    cameraConfig.frame_size = FRAMESIZE_SVGA;
    cameraConfig.jpeg_quality = 12;
    // cameraConfig.jpeg_quality = 5;
    cameraConfig.fb_count = 1;
  }


  esp_err_t cameraInitError = esp_camera_init(&cameraConfig);
  if (cameraInitError != ESP_OK)
  {
    Serial.printf("Error al inicializar la cámara 0x%x", cameraInitError);
    delay(1000);
    ESP.restart();
  }
}

void initWifi()
{

  // WiFi.mode(WIFI_MODE_STA);

  // int32_t channel = getWiFiChannel(WIFI_SSID);

  WiFi.mode(WIFI_MODE_APSTA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);

  Serial.printf("Conectando a %s .", WIFI_SSID);

  // Agrega certificado de api Telegram
  clientTCP.setCACert(TELEGRAM_CERTIFICATE_ROOT);
  while (WiFi.status() != WL_CONNECTED)
  {
```

```
    Serial.print(".");
    delay(200);
  }
  Serial.println(" ok");

  IPAddress ip = WiFi.localIP();

  Serial.printf("SSID: %s\n", WIFI_SSID);
  Serial.printf("Channel: %u\n", WiFi.channel());
  Serial.printf("IP: %u.%u.%u.%u\n", ip & 0xff, (ip >> 8) & 0xff, (ip >> 16) & 0xff, ip >> 24);
}

void initESPNOW()
{
  if (esp_now_init() != ESP_OK)
  {
    Serial.println("Error de inicialización del protocolo ESP-NOW");
    while (1)
      ;
  }

  esp_now_register_send_cb(OnDataSent);

  memcpy(peerInfo.peer_addr, espNowBroadcastAddress, 6);
  // peerInfo.ifidx = ESP_IF_WIFI_STA;
  peerInfo.encrypt = false;

  if (esp_now_add_peer(&peerInfo) != ESP_OK)
  {
    Serial.println("No se pudo agregar el par");
    while (1)
      ;
  }
  esp_now_register_recv_cb(OnDataRecv);
}

void updateDisplay()
{
  Serial.println("ACCIONES ENTRANTES");
  Serial.print("Flash: ");
  Serial.println(deviceActions.flash);
  Serial.print("Cámara: ");
  Serial.println(deviceActions.camera);
  Serial.print("Buzzer: ");
  Serial.println(deviceActions.buzzer);
  Serial.println();
}

void sendSensorData(int imageSize)
{
  Serial.println("Enviando sensorData...");
  // Envía magneticSensor y pirSensor como un solo paquete
```

```
  struct SensorData
  {
    int magnetic;
    int pir;
    int imageSize;
  };
  SensorData sensorData;
  sensorData.magnetic = magneticSensor;
  sensorData.pir = pirSensor;
  sensorData.imageSize = imageSize;

  esp_err_t result = esp_now_send(espNowBroadcastAddress, (uint8_t *)&sensorData,
sizeof(sensorData));

  if (result == ESP_OK)
  {
    Serial.println("Envío del mensaje exitoso");
  }
  else
  {
    Serial.println("Envío del mensaje fallido");
    Serial.print("Error al enviar SensorData. Código de error: ");
    Serial.println(result);
  }
}

if (cameraState == HIGH)
{
  cameraState = LOW;
  deviceActions.camera = cameraState;
  // Serial.println("Inicia proceso de foto");
  // sendImage();

  sendPhotoTelegram();
}
// Cambio de estado en buzzer
if (buzzer != buzzerStatus)
{
  buzzerStatus = buzzer;
  digitalWrite(BUZZER_PIN, buzzerStatus);
}

updateDisplay();
delay(500);
}


// Función para enviar imagen
void enviarImagen()
{
  Serial.println("Capturando foto...");
```

```
  // Descarta la primera imagen debido a la mala calidad
  camera_fb_t *fb = NULL;
  // Captura una nueva foto
  fb = NULL;
  fb = esp_camera_fb_get();
  if (!fb)
  {
    Serial.println("Error al capturar imagen de la cámara");
    delay(1000);
    ESP.restart();
    return;
  }

  Serial.println("FOTO CAPTURADA");

  // Obtiene el puntero a los datos de la imagen y el tamaño de la imagen
  uint8_t *datosImagen = fb->buf;
  size_t tamanoImagen = fb->len;

  Serial.println("DatosFotoEntrante");
  for (int i = 0; i < 20; i++)
  {
    Serial.print(datosImagen[i], HEX);
    Serial.print(" ");
  }
  Serial.println();
  enviarDatosSensor(tamanoImagen);
  enviarDatosImagen(datosImagen, tamanoImagen); // Envía el búfer de imagen

  esp_camera_fb_return(fb);

  Serial.println("FOTO ENVIADA");
}

// Función para enviar un paquete de datos
void enviarDatosImagen(const uint8_t *datos, int longitud)
{

  Serial.println("Enviando imagen...");
  int datosRestantes = longitud;
  int offset = 0;

  while (datosRestantes > 0)
  {
    int longitudPaquete = min(tamanoPaquete, datosRestantes);

    Serial.println("DatosFotoEntrante");
    for (int i = 0; i < 20; i++)
    {
      Serial.print((datos + offset)[i], HEX);
      Serial.print(" ");
    }
```

```
    esp_err_t resultado = esp_now_send(direccionBroadcast, datos + offset, longitudPaquete);

    if (resultado == ESP_OK)
    {
      Serial.println("Envío del mensaje exitoso");
      Serial.print("Offset: ");
      Serial.println(offset);

      offset += longitudPaquete;
      datosRestantes -= longitudPaquete;
    }
    else
    {
      Serial.println("Envío del mensaje fallido");
    }

    delay(100);
  }

  datosImagen = nullptr;
  tamanoImagen = 0;
  offset = 0;
  datosRestantes = 0;
}

String enviarFotoTelegrama()
{
  const char *miDominio = "api.telegram.org";
  String obtenerTodo = "";
  String obtenerCuerpo = "";

    Serial.println("Error al capturar imagen de la cámara");
    delay(1000);
    ESP.restart();
    return "Error al capturar imagen de la cámara";
  }

  Serial.println("Conectando a " + String(miDominio));

  if (clientTCP.connect(miDominio, 443))
  {
    Serial.println("Conexión exitosa");

    String cabecera = "--AlertESP32\r\nContent-Disposition: form-data; name=\"chat_id\"; \r\n\r\n" +
ID_CHAT + "\r\n--AlertESP32\r\nContent-Disposition: form-data; name=\"photo\";
filename=\"esp32-cam.jpg\"\r\nContent-Type: image/jpeg\r\n\r\n";
    String cola = "\r\n--AlertESP32--\r\n";

    size_t longitudImagen = fb->len;
    size_t longitudExtra = cabecera.length() + cola.length();
    size_t longitudTotal = longitudImagen + longitudExtra;
```

```
clientTCP.println("POST /bot" + TOKEN_BOT + "/sendPhoto HTTP/1.1");
clientTCP.println("Host: " + String(miDominio));
clientTCP.println("Content-Length: " + String(longitudTotal));
clientTCP.println("Content-Type: multipart/form-data; boundary=AlertESP32");
clientTCP.println();
clientTCP.print(cabecera);

uint8_t *fbBuf = fb->buf;
size_t fbLen = fb->len;
for (size_t n = 0; n < fbLen; n = n + 1024)
{
  if (n + 1024 < fbLen)
  {
    clientTCP.write(fbBuf, 1024);
    fbBuf += 1024;
  }
  else if (fbLen % 1024 > 0)
  {
    size_t resto = fbLen % 1024;
    clientTCP.write(fbBuf, resto);
  }
}

clientTCP.print(cola);

esp_camera_fb_return(fb);

int tiempoEspera = 10000; // tiempo de espera 10 segundos
long inicioTemporizador = millis();
boolean estado = false;

while ((inicioTemporizador + tiempoEspera) > millis())
{
  Serial.print(".");
  delay(100);
  while (clientTCP.available())
  {
    char c = clientTCP.read();
    if (estado == true)
      obtenerCuerpo += String(c);
    if (c == '\n')
    {
      if (obtenerTodo.length() == 0)
        estado = true;
      obtenerTodo = "";
    }
    else if (c != '\r')
      obtenerTodo += String(c);
    inicioTemporizador = millis();
  }
  if (obtenerCuerpo.length() > 0)
```

```
        break;
    }
    clientTCP.stop();
    Serial.println(obtenerCuerpo);
  }
  else
  {
    obtenerCuerpo = "Fallo al conectar con api.telegram.org.";
    Serial.println("Fallo al conectar con api.telegram.org.");
  }
  return obtenerCuerpo;
}
```

Apéndice C. ***Código para obtener la dirección MAC del dispositivo Maestro y Esclavo***

```
#include "WiFi.h"

void configuracionInicial() {
  Serial.begin(115200);
  WiFi.mode(WIFI_MODE_STA);
  Serial.println(WiFi.macAddress());
}

void buclePrincipal() {

}
```